University of Zurich[UZH]

**Department of Informatics**


Otto-Stern-Weg 3
8093 Zürich,
Switzerland

ETHZ - Institute for
Molecular Systems Biology

**Daniel Boschung**
Co-Project Leader
BeeLivingSensor

Binzmühlestrasse 14
CH-8050 Zürich-Oerlikon
Switzerland

UZH - Informatics and
Sustainability Group

**Prof. Dr. Lorenz Hilty**
Head of Group
**Dr. Clemens Mader**
Research Associate
Co-Project Leader

Gabriela Eugenia López Magaña
gabrielaeugenia.lopezmagana@uzh.ch
Matrikel-Nr.: 17-744-830

Pascal Engeli
pascal.engeli@uzh.ch
Matrikel-Nr.: 14-671-457


February 12, 2021

**Master's Project Report**

# Noninvasive bee tracking in videos. Deep learning algorithms and Cloud Platform design specifications.

## Abstract


*BeeLivingSensor* is a project with the goal of using 30'000 to 50'000 honey bees from a colony as living sensors for measuring the hive's health and environmental changes in a non-invasive way. The main goal of our project was to identify optimal algorithms for object-recognition and tracking of bees arriving and leaving the bee hive entrance.

This paper is divided into three main parts. In the introduction we explain the state of current research and the bee hive systems in our study. In the first section we explain the object detection performed by using YOLOv4. One of goals of this project was to find the best fitting models for the different hives, by exploring the space of training possibilities, we explain how we trained and validated the models using the data provided by the ETHZ-IMSB and in the recommendations we suggest strategies for the future training of models. The second part focuses on bee tracking since it's heavily dependent on the bee detection in video sequences. We explain the major decisions in developing the algorithm and validate it with different video settings. In the conclusion we make a reference to the contributions we did for the ongoing BeeLivingSensor Cloud Platform Project.


## Introduction

In recent years, Precision Beekeeping has developed into an active field of research where data is extracted from the beekeeping process and enriched with external data for making more informed decisions, monitor the state of the colony and be able to automatically pace some processes [24].

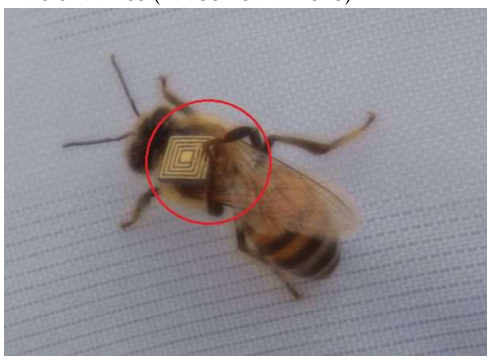
Some research has been conducted to improve the detection of honey bees, mainly under laboratory conditions that restrains the natural movement and behaviour of bees in the hive's entry, like exemplified in the figures of table 1, where bees are set under unnatural light conditions [2], guided trough narrow channels for counting with light barriers [10] or tagging them with RFID chips [8]. Parallel, there is great effort applied to the problems of multi-object tracking in videos in academia, research and the industry [23] [4], [22]. The main task of object detection is the location of objects by "bounding" them in rectangular boxes, and the classification of the detected bounded object in the classes that were set for the learning process of the algorithm. The goal of our project was to generate algorithms and pipelines that were able to identify the bees in video recordings as they approach their real-life bee hive entrances and as they leave them, to get information that should allow the identification of bee traffic. This Masters in Informatics project was developed in cooperation with the ETHZ Institute for Molecular Systems

Biology (IMSB) inside the frame of the long-term project *BeeLivingSensor* for data integration, bee watching and citizen-science. We took over the work from a previous team of the ETH DataLab where they used TensorFlow models.

(a) Honey bees in the camera field of view (Babic et. al.; 2016).

(b) Visual monitoring device in front of the hive entrance (Tausch et. al 2020)

(c) Honey bees going trough light barrier counting tunnels (Hudson; 2019).



(d) Example of bees trying to remove the RFID tag (De Souza; 2018).

Table 1: Intrusive observation devices.

Some challenges that are present on the problem of multi-object detection and tracking in video are frequent occlusions as bees walk on top of another, bee “bunching”, strong resemblance between objects and low resolution images, appearance and disappearance of bees and other objects, measurement differences and noise due to different video sources.

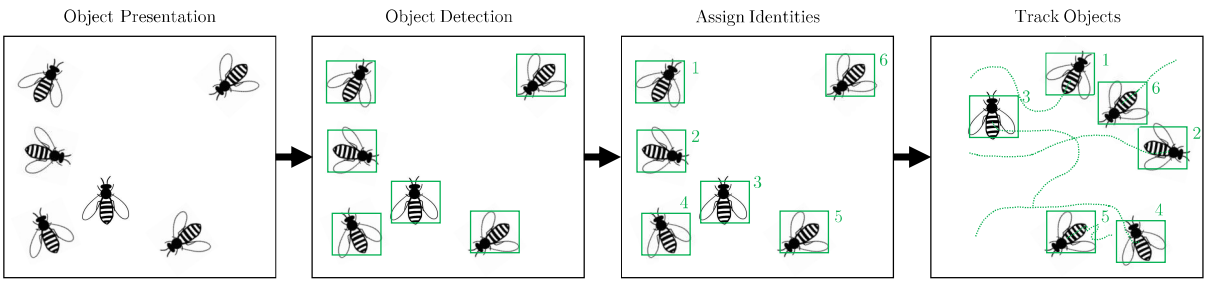
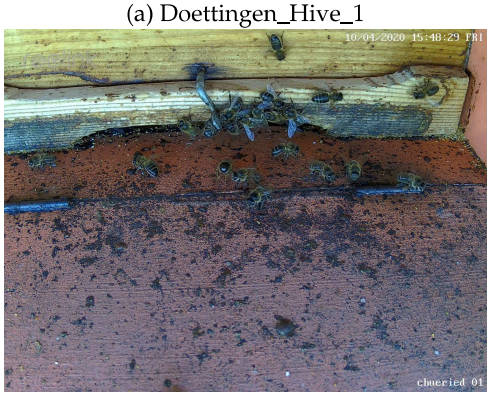



Figure 1: Flow of steps based on (Khan et. al.; 2019).

**Analyzing diverse hive systems** For the object detection, the ETHZ-IMSB provided us with video footage of 11 different bee hives. A very common type of hive entry is rectangular, where the material and colors vary, like seen in table 2; in table 3 we had two examples of hive entries with circular shape and built in wood; in table 4 we have hive entries built in more raw, "un-worked", materials like a cavity in a tree and a whole made in a hive made of hemp fiber. As part of our experiments was to test the viability of our algorithms in low resolution video taken from a usual mobile phone (table 5).

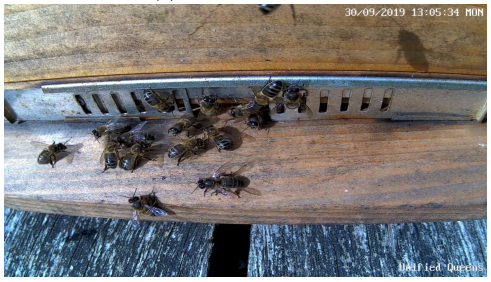

(a) Doettingen_Hive_1


(b) Echolinde




(c) Erlen_Hive_04

(d) Erlen_Hive_11




(e) UnitedQueens

Table 2: The hives under study. "Common" hive types.


(a) Chueried_01

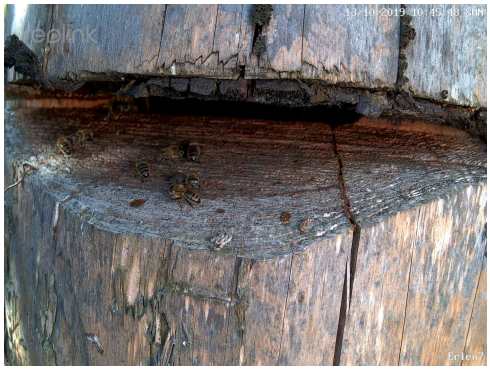
(b) Froh14

Table 3: The hives under study. Round entry hive types.

### Data Annotation

For our project, 3'000 frames were annotated. While training the NN, YOLO requires two sets of images, called the training and the testing set, we distributed 80% and 20% of the training frames. Additionally,

(a) Chueried_Hempbox



(b) Froh_23_TreeCavity

Table 4: The hives under study. Structured-background hive types.

(a) ClemensHiveRed

(b) ClemensHiveYellow

Table 5: The hives under study. Low resolution mobile telephone video.

we set apart an amount of images that was at least the size of the testing set for cross-validation of the trained models (table 8). YOLO works with a special file format of one text file per image that includes the coordinates, that are normalized in the range [0,1] and a numeric representation of the label; there is additionally a labelmap which maps the numeric IDs to strings for the object class. The ground truth data includes the image name, the classes of the annotated objects and the true bounding box coordinates of each of the objects in that image.

# Object Detection

## Object Detection Models

Object Detection are technologies that aim to detect instances of objects of a certain class in digital images and videos. Deep convolutional networks (DCNNs) have been developing fast in the last years and they are a preferred choice over previous machine learning hand-crafted methods. One of the reason for the good performance of CNNs is that they learn more representative features [23].

While testing some TensorFlows models, we also tried YOLOv3. We trained a general model that included all the hives. The results were somehow encouraging because we got 72% recall, 90%precision, after using a IoU threshold of 0.7 in Froh14, which was positive compared to the average 0.77 IoU reported by the previous team. For UnitedQueens we got R:0.93, P:0.99, compared to 0.799; for Chueried_01 we got similar values than from the ETH team 0.63, however for Froh23_TreeCavity we got 0.1 instead of 0.673.

In April 2020 YOLOv4 was released with significant improvements in terms of speed and performance, like observed in Figure 2 [6]. After training one general model and comparing the improvements we moved all project efforts into YOLOv4.

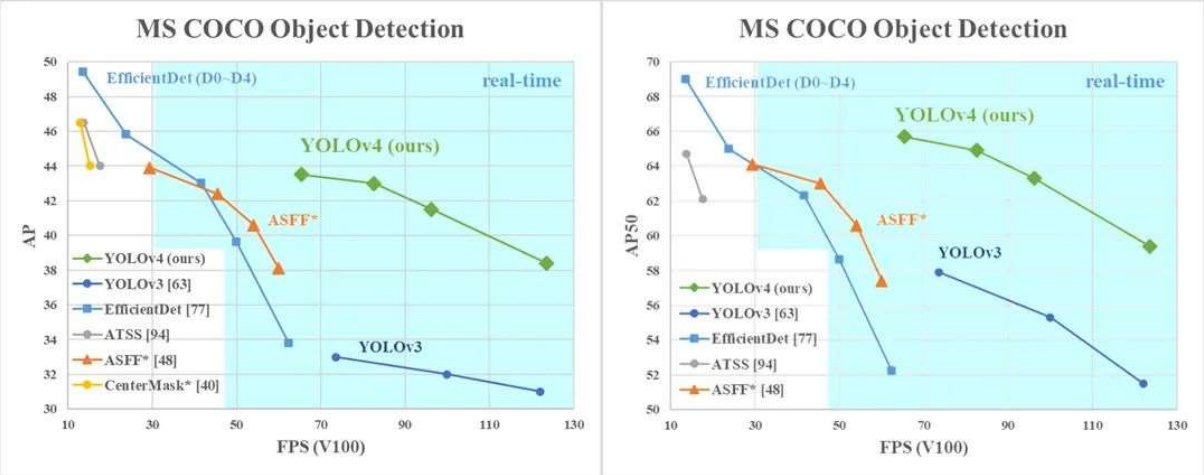



Figure 2: Comparison of YOLOv4 and other state-of-the-art object detectors (Bochkovskiy et. al.; 2020).

## Model Training and Validation

The first goal of the project was to find the best fitting model for each hive, by **exploring the space of training possibilities** aiming to find viable strategies. Each of the following subsections explains our motivation to test the given training strategy, the results we obtained and our conclusions about the given strategy.

In multi-object detection the models predict the occurrence and position of objects in the given image or video frame. For an understandable evaluation of our models we use precision and recall to evaluate the quality of the model's detection. *Recall* or *true positive rate* measures how good the model detects all the truly existing objects $\left(\frac{TP}{TP+FN}\right)$. *Precision* or *positive prediction value* shows how much of the detected objects are actually correct $\left(\frac{TP}{TP+FP}\right)$. In our project, for the validation phase, we set a threshold of 0.70 IoU for a bounding box to be counted as a TP.

## Training

For the training of our object detection models we worked in Google Colab. We worked mainly with the Darknet framework, YOLOv4, OpenCV, Nvidia cuDNN and sqlite3. The YOLO models are trained with starting weights (darknet53.conv.74.weights) that wer originally pre-trained for feature extraction in the ImageNet dataset and for detection on the COCO dataset, only the weights for the convolutional

layers are included, excluding the weights for the final fully connected layer which outputs the class probabilities for classification. YOLOv4 demands two sets of images, one for training and one test set that is used during the training to adjust the weights, this sets are displayed in Table 6. When we started navigating the dimensions for training we kept using all beehives in the test-set because we wanted to review the generalization of the models. Except otherwise explained the models are trained using a test-set that includes all bee hives.

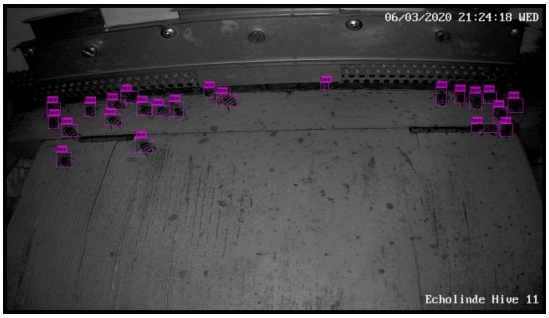
| | Training | | | Test | | |
|---|---|---|---|---|---|---|
| Beehive | Nr. Frames | Nr.Bees | Avg. Bees/Frame | Nr. Frames | Nr.Bees | Avg. Bees/Frame |
| Chueried_Hempbox | 160 | 3128 | 20 | 40 | 763 | 19 |
| Chueried_Hive01 | 80 | 531 | 7 | 20 | 134 | 7 |
| ClemensRed | 80 | 1266 | 16 | 20 | 321 | 16 |
| ClemensYellow | 80 | 1942 | 24 | 20 | 429 | 21 |
| Doettingen_Hive1 | 160 | 5021 | 31 | 40 | 1359 | 34 |
| Echolinde | 80 | 3563 | 45 | 20 | 826 | 41 |
| Echolinde_Night | 80 | 1793 | 22 | 20 | 408 | 20 |
| Erlen_Hive04_diagonalview | 160 | 2326 | 15 | 40 | 583 | 15 |
| Erlen_Hive04_frontview | 160 | 2268 | 14 | 40 | 569 | 14 |
| Erlen_Hive04_smartphone | 80 | 1068 | 13 | 20 | 306 | 15 |
| Erlen_Hive11 | 80 | 5990 | 75 | 20 | 1629 | 81 |
| Erlen_Hive11_Night | 80 | 3066 | 38 | 20 | 905 | 45 |
| Froh14 | 160 | 1591 | 10 | 40 | 481 | 12 |
| Froh23_TreeCavity | 160 | 616 | 4 | 40 | 156 | 4 |
| UnitedQueens | 160 | 1638 | 10 | 40 | 406 | 10 |

Table 6: Image sets for model training

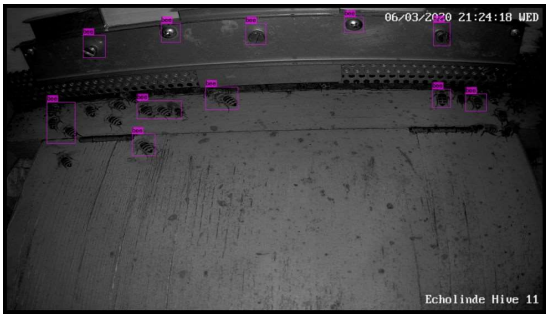
## Validation set

In order to find strategies for training in the Azure platform by prioritizing the different possible dimensions, it is important to qualitatively compare the trained models. In the field of object detection the COCO and Pascal VOC are datasets that support the research for object detection by providing a standard benchmark for models to be compared against each other for performance review, like it is observed in Figure 2 where YOLOv4 is compared to other object detectors using the COCO dataset [9].

In Table 7 we observe the large difference in performance that different models have on the same video frame. We see this behaviour on different models and hives despite models having achieved a high mAP on the same test data while training.




(a) Echolinde_Night Frame with Echolinde_Night Model




(b) Echolinde_Night Frame with Froh23_TreeCavity Model

Table 7: Different models applied to same video frame from Echolinde_Night.

For our project we developed our own dataset for bench marking. To have a broad and accurate idea of the performance of each model and the weight of the dimension we chose. Furthermore, we make

a cross-validation from all models using the validation set in order evaluate how our models might generalize.

In Table 8 we explain the number of frames and bees that we used to validate each bee hive. Additionally, in table 9 we augmented the set with color casting on green, red and gray-scale. To find the best models, we applied cross-validation, meaning that each of our trained models was validated against all other hives. For the most of our validations, except otherwise explained, we will use all bee hives except the low resolution videos (ClemensRed and ClemensYellow) and Chueried_Hempbox because of the highly textured and shaded background.

| Bee hive | Recording | Nr. Frames | Nr.Bees | Avg. Bees/Frame |
|---|---|---|---|---|
| Chueried_Hempbox | topfront | 40 | 774 | 19 |
| Chueried_Hive01 | topfront | 20 | 132 | 7 |
| ClemensRed | topfront | 20 | 336 | 17 |
| ClemensYellow | topfront | 20 | 486 | 24 |
| Doettingen_Hive1 | topfront | 40 | 1301 | 33 |
| Echolinde | topfront | 20 | 921 | 46 |
| Echolinde_Night | topfront night | 20 | 432 | 22 |
| Erlen_Hive04_diagonalview | diagonal | 40 | 607 | 15 |
| Erlen_Hive04_frontview | topfront | 40 | 570 | 14 |
| Erlen_Hive04_smartphone | diagonal smartphone | 20 | 302 | 15 |
| Erlen_Hive11 | topfront | 20 | 1632 | 82 |
| Erlen_Hive11_Night | topfront night | 20 | 788 | 39 |
| Froh14 | topfront | 40 | 419 | 10 |
| Froh23_TreeCavity | topfront | 40 | 158 | 4 |
| UnitedQueens | topfront | 40 | 408 | 10 |

Table 8: Validation images

| Bee Hive | Recording | Nr. Frames | Nr.Bees | Avg. Bees/Frame |
|---|---|---|---|---|
| Chueried_Hive01_green80 | topfront | 20 | 132 | 7 |
| Chueried_Hive01_red70 | topfront | 20 | 132 | 7 |
| Erlen_Hive11_grayscale | topfront | 20 | 1632 | 82 |
| Erlen_Hive11_red70 | topfront | 20 | 1632 | 82 |

Table 9: Additional validation images

## Validation outcomes

### Activation Functions

Activation functions define when a neuron will be activated based on it's input, they aim to introduce non-linearity, that is required to solve complexity. In 2019 Misra proposed Mish, a new activation function with improved performance [14]. In 2020 Darknet-Yolov4 included this activation function [6]. However, the Python library OpenCV, which we used for analyzing video and bulk management of frames, only included the support for Yolov4 on April 29, 2020 [3]. We tested the Leaky against the Mish activation function and in Table 15 we show the performance differences of the *relevant_hives* (excluding the low resolution hives and the Chueried_Hempbox due to its highly textured and shaded background), the recall increased by 9.8% and the precision by 5%.

### Quantity of training images

At the start of the project we placed a lot of effort in labeling large amounts of images. We then added this as a dimension to be researched. For bee hives Echolinde and Froh_14 we tested the impact of the

number of training images on performance. We generated configuration files with 10, 20, 40, 80 and 160 frames. The best models for both beehives were the ones with 80 images. For Froh14 this model is the overall top model with recall of 0.916 and precision of 0.909. For Echolinde, the 80-model was the 2nd top model (R:0.925, P:0.974) with a difference to 1st of -0.013 in recall and a gain of 0.012 in precision. After reaching this results we decided to train all our models with 80 images.

### Mobile phone video - low and high resolution

In the low resolution videos we achieved a maximum recall of 55% for ClemensYellow and 76% for ClemensRed, with precision of 79% and 76%. With high resolution Erlen_Hive04_smartphone we reached an F2-score of 93.6%. We recomend the use of HD images and video.

### Frame manipulation

- **Background subtraction.** Kale used the [11] the Gaussian mixture model (GMM) background subtraction method achieving improvements in the results for object recognition to a certain degree, but acknowledged that the method had difficulties identifying bees when they move too close or when they touch because then the model only finds one single contour, like observed in the larger bounding box in 3.

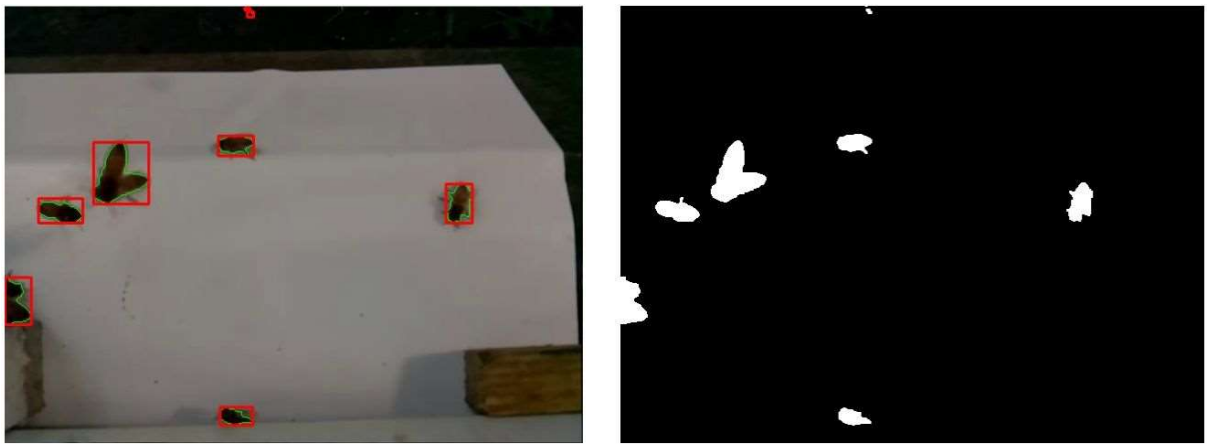


Figure 3: From left to right: Background subtracted from image with bees. Contours surrounding foreground and their bounding boxes.(Khan et. al.; 2019).

  We did some experiments with background subtraction with different parameters and models, like transforming the image to gray-scale before processing, applying Gaussian blur. Our attempts showed that, because of the nature of our images where the background is very structured, the shadows change constantly, and because, unlike in the academic studies, there is not a strong contrast between the background and the object to be detected. We show some examples of our test in table 10.
- **Color manipulation** We tried, like mentioned by Shiji et al. [20] in our experiment color casting (or jittering) did not cause any improvement in performance. Models trained with this images got worst rankings than the models trained on unchanged images for the same bee hive.

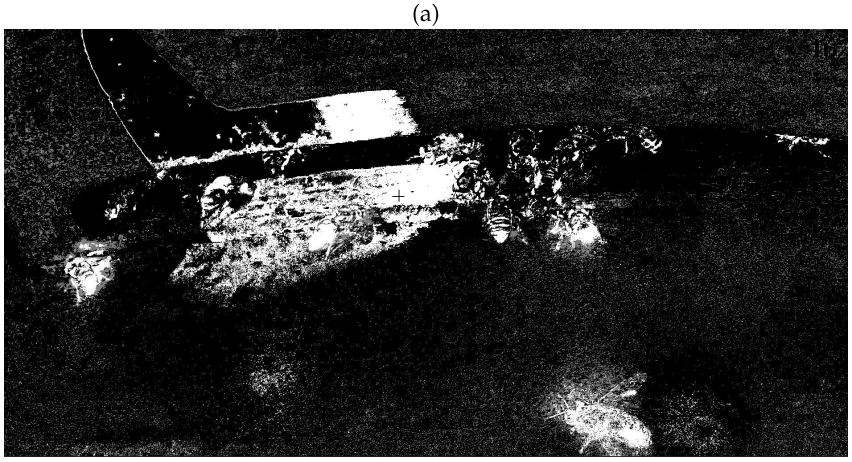
### Camera Angles

In Erlen_Hive04 we tested different camera angles. From the three different angles, shown in Table 11, the angle that reached the best performance was the top frontal view with F2-Score of 0.982, followed by the diagonal angle with 0.9605.

### General model

Because of the convenience of having one general model to apply to any given hive, our first step was to train a model (*staged_2/all_HD_hives_40*) with images from all relevant beehives, we excluded the low

(a)



(b)

Table 10: Different background subtraction methods.

(a) Erlen_Hive_04 top front view

(b) Erlen_Hive_04 diagonal view

(c) Erlen_Hive_04 mobile diagonal view

Table 11: Different camera angles for Erlen Hive 04.

resolution hives (ClemensRed and ClemensYellow) and the augmented data from color manipulation. In average, the top model has 4% more recall. Only for Erlen_Hive04_smartphone it was the top model with 0.012 difference in recall from 2nd best.

### Night models

We observed that for both beehives, Echolinde and Erlen_Hive_11, with videos with night setting, the models that were specifically trained for them performed top. The second best model was the general model (*staged_2/all_HD_hives_40*) that also included images of the night settings in it's training. The models of the related beehives that were trained on day light performed only 9th and 10th place with 0.29 and 0.39 less recall as the top model. We recommend, if only one model per hive will be trained then the night images should be included.

### Object Detection Recommendations

After our investigations into which factors are important for the improvement of performance in the training of the models we recommend to use Mish as activation function, unless a new improved function is released.

# Bee Tracking

In the second part of this report we utilize the object detection models in order to track objects over time. The goal of the bee tracking is maintaining a counting system to determine the traffic i.e. how many bees flew in and out of a hive. At the end of this section we will conduct an experiment using this counting system to analyze the traffic in a video of a bee hive.

The design of a multi-object tracking (MOT) system in a video sequence consists of a loop of two backward dependent parts: first detecting multiple objects in frame $f$ and second associating these objects in frame $f$ to those in frame $f-1$ [12]. The development of the MOT is based on the principles observed in [19]. The detection of multiple objects comes from the YOLO model previously discussed. The quality of a MOT system relies ultimately on the accuracy and recall of the object detection models [12] [5].

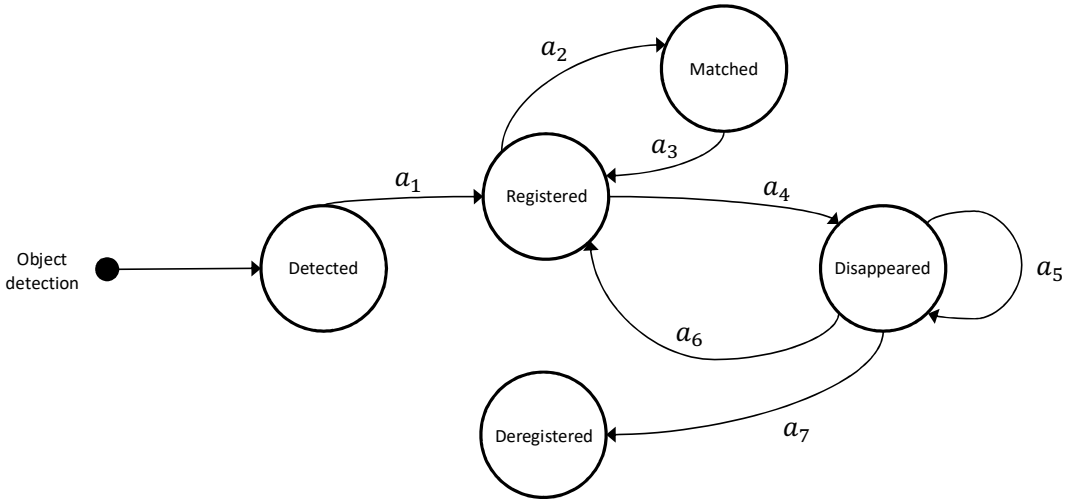



Figure 4: Five life stages of an object during tracking based on [22]

Our MOT system works in a decoupled way where the detection of a video (sequence of frames) are first stored in a database and then in a second step retrieved to analyze the traffic flow of the bees in the respective video. This process is twofold because of two reasons. First, the object detection requires computational power. For example detecting all objects within a single frame takes up on average roughly 1.63 seconds using Google Colab. If we work with a video of length 60 seconds and 100 frames per second, this results in 6000 frames and a processing time of about 2 hours and 40 minutes. In order to distribute the computational power on to different devices, the goal is to divide expensive tasks into stand alone parts for cloud platform outsourcing. Second, to develop the algorithm for tracking and counting the bees, the process of retrieving coordinates from a stable and instantly accessible source is much more efficient than detecting those coordinates every time from a video sequence.

The MOT system proposed in this report works in reference to [22] with four life stages and is extended by a fifth life stage. Those five life stages are illustrated in figure 4. The five life stages are according to [22] the management of detection, registration, matching, deregistration and handling the disappeared objects. Each of these life stages is connected via a path denoted with $a_{i \in \{1,2,3,4,5,6,7\}}$. In order to analyze the traffic flow of bees in a sequence of frames, we retrieve frame by frame from a video which is subject to be analyzed. For every frame all five life stages will be executed to keep track of changes between the frames except the first frame. The following paragraphs refer to each of the life stages. The goal is to cover all possible cases to demonstrate the functionalities of our MOT system.

The first life stage of our MOT system retrieves every detection for a frame $f$ in a vector of coordinates from the database where the $b$-th entry has the format $coordinates_{f,b} = [xmin_{f,b}, xmax_{f,b}, ymin_{f,b}, ymax_{f,b}]$.

### Registration

The detection vector gets sent via path $a_1$ to the registry. Suppose it is the first frame of the video and the registry is empty. For the first frame in the video, only one of the five life stages will be performed. If there is at least one bee in the first frame of the video located, the bee(s) will be registered as objects, each with four elements of information; It's coordinates, centroid and a newly created unique ID and a counter. The coordinates are given by the input from the object detection based on the frame number. The centroid consists of two coordinates of the center point of the object, which is computed according to equation 1 from the coordinates of the object. The objective in storing the centroid is to obtain a single position of a bee with an $x$ and $y$ coordinate. This single position is stored for every frame and can therefore displayed as the track of a bee after processing multiple frames. Every object gets a unique ID in order to access its status in the registry. The counter is responsible to keep track of the object's disappearance status. As soon as all the objects in the first frame are registered, all the registered objects with coordinates, IDs and history of centroids are returned. The history of centroids is for the first frame simply the actual position of a bee.

$$centroid = (x_{min} + \frac{x_{max} - x_{min}}{2}, y_{min} + \frac{y_{max} - y_{min}}{2}) \tag{1}$$

### Matching

For every subsequent frame where the registry is already filled with objects of previous frames, the update process heavily depends on the matching process to determine whether the objects have moved or disappeared. The extracted coordinates for the frame $f > 1$ again gets sent via path $a_1$ to the registry and checked if objects are already registered. Suppose this is the second frame $f = 2$ and the registry already contains objects from the first frame $f = 1$. Both the already registered objects and incoming coordinates get sent via path $a_2$ to the matching process where the coordinates of frame $f - 1$ (already registered) are compared with the new coordinates from frame $f$. In order to match the objects $b_f$ with $b_{f-1}$ we apply both a pairwise comparison of the minimal euclidean distance and the maximal overlap of their bounding boxes. Even though, mainly matching based on maximal overlap is done in the literature [4], we extended the matching process by adding the measurement of minimal euclidean distance to it. The main reason to not only rely on the maximum overlap measure is because we work with fast moving objects compared to walking people or slowly driving cars. Especially in scenarios where the bees were recorded with low frame per second ratios, the bounding boxes have no overlap at all because of the movement speed. In order to overcome this problem, we introduce another procedure by comparing the objects based on their distance.

$$D = ||\boldsymbol{b}_f - \boldsymbol{b}_{f-1}||_2 \tag{2}$$

The first procedure takes into account the pairwise distance between every object in $b_f$ and $b_{f-1}$ with $\boldsymbol{b_i} \in \mathbb{R}^4 : i \in \{1, ..., n\}$. Therefore, we compute the euclidean distance matrix in reference to [22] and [1] in equation 2. After obtaining the distance matrix $D$ we iterate over the columns which represent the objects of $b_f$ and sort it after the smallest distance for every index of the registered objects of $b_{f-1}$. Meaning we fix the index of $b_f$ and look for the index of the closest object in $b_{f-1}$. In an optimal case where every object of $b_{f-1}$ was found in proximity in $b_f$, then the distance matrix $D$ would have the smallest euclidean distances in the main diagonal. Because of the movement, the object detection records the detected objects in a different order.

A specific example of this ordering process is shown in figure 5 with the comparison of two frames. On the left hand side is a heat map, showing on the $y$-axis all registered objects ($b_{f-1}$) of the previous frame and on the $x$-axis all incoming objects ($b_f$) of the current frame. This image is a colored visualization of the distance matrix $D$. It is important to note that $D$ is not a square matrix. This means if $D \in \mathbb{R}^{m*n}$ and $m > n$, then there are more objects registered than found on the current frame, which results in a dimiensionality mismatch. In the case of our example in figure 5 the distance matrix has the form $D \in \mathbb{R}^{23*22}$. This means one object disappeared between $f-1$ and $f$ and we can only reassign 22 of 23 objects in the current frame. On the other hand, if $D \in \mathbb{R}^{m*n}$ and $m < n$, additional objects appeared, which were not registered before. First, it is important to keep in mind that when looking at the $x$-axis of figure 5 that the numbering does not depend on the ID of the registered objects on the $y$-axis. The

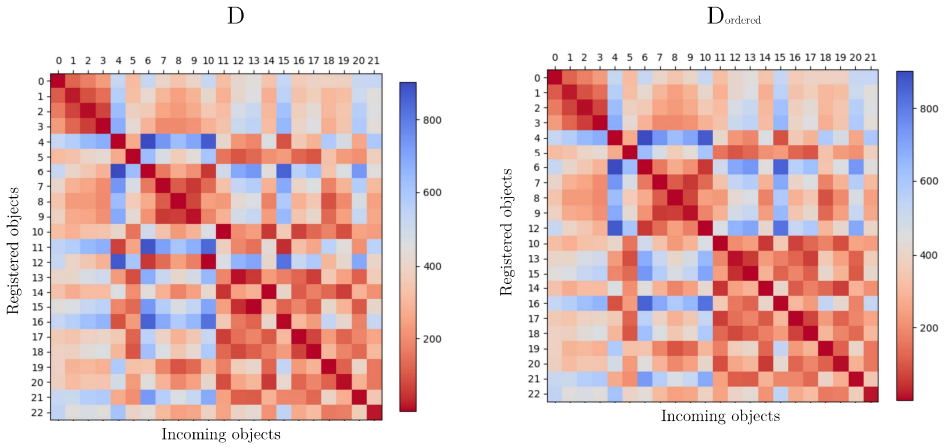



Figure 5: Left: distance matrix $D$ with distanced colored from close (red) to far (blue). Right: ordered distance matrix $D_{ordered}$ also colored from close (red) to far (blue)

numbering is simply a temporary ID given to incoming objects and disregarded after matching the registered objects. These temporary IDs reflect the order by which the object detection recorded the objects in the processed frame. This means that new objects can appear anywhere on the $x$-axis of the distance matrix $D$. In order to identify the newly appeared objects if $D \in \mathbb{R}^{m*n}$ and $m < n$, we apply the linear sum assignment (equation 3), also known as the Hungarian or Munkres algorithm as used in [4] and formally explained in [7]:

$$\begin{aligned} \min \sum_{i=1}^{m} \sum_{j=1}^{n} d_{i,j} * a_{i,j}, \\ s.t. \sum_{i=1}^{m} a_{i,j} = 1, 0 \leq \sum_{j=1}^{n} a_{i,j} \leq 1, \\ a_{i,j} \in \{0,1\} \end{aligned} \tag{3}$$

where $D \in \mathbb{R}^{m*n}$ is the distance matrix, $d_{i,j}$ is the distance between registered object $i$ and the incoming object $j$. We obtain a resulting binary matrix $A \in \mathbb{R}^{m*n}$, where $a_{i,j} = 1$ if and only if the $i$-th registered object is assigned to the $j$-th incoming object, if there was no match, then $a_{i,j} = 0$. By obtaining the optimal assignment, we check for the temporary IDs which are not contained in the optimal assignment since there has not been a registered object for the newly appeared objects. When identified these new objects, we first register those objects on path $a_3$ and start over on path $a_2$ to calculate the new distance matrix $D$, which now has the form $D \in \mathbb{R}^{n*n}$.

By looking at the colors of the left plot of figure 5 we note that we are not in an optimal scenario as described above, where the closest distances, colored in red, are on the main diagonal. Especially between objects 10 and 15 it is notable that the object detection recorded the objects in a different order such that we end up in a color mismatch. Overall, dense regions can be observed by spotting red clusters on the heat map. These are locations where multiple objects are located in close proximity. Movement in those regions might lead to a a different order recording of the found objects during the object detection.

The two aforementioned problems, dimensionality and color mismatch, lead to the already motivated necessity of reordering the distance matrix $D$ by reassigning the incoming objects to the already registered objects. This is done as already described above by assigning the incoming objects to the registered objects by the smallest euclidean distance such that the main diagonal of $D_{ordered}$ has the minimal values of the distances in the row and columns. In detail, the dimensionality and color mismatch can be solved by iteratively assigning incoming objects to the closest registered objects and eliminating one of the registered objects where none of the incoming objects can be assigned to. In this specific example it can be observed that no incoming object had a close distance to object 11 and therefore object 11 would be marked as disappeared in the current frame.

The second procedure employs a pairwise comparison of the overlapping bounding boxes of each object in $b_f$ and $b_{f-1}$ and tries to assign incoming objects where the overlap is maximal with an already

registered object. This concept was already applied in [4] using the intersection-over-union (IOU) score shown in 4. Similarly as in the first procedure with reordering the distance matrix $D$ such that the minimal distance lies on the main diagonal, we reorder the overlap matrix $O$ such that the maximal IOU scores lie on the main diagonal of $O$. Creating the overlap matrix $O$ also requires the pairwise comparison of incoming objects and already registered objects. The overlap matrix $O$ also suffers from dimensionality and color mismatch problems. We apply a similar ordering algorithm to iteratively assign incoming objects to the registered objects such that their IOU scores are maximal and the resulting $O_{ordered}$ is a square matrix. The implementation in the code was lead by a suggested answer from [21].

$$IOU(a,b) = \frac{Area(a) \bigcap Area(b)}{Area(a) \bigcup Area(b)} \tag{4}$$

By applying both procedures, the distance and overlap matching of incoming objects and registered objects, it is possible to obtain a diverging assignment of those objects. To overcome differing matches, we merge both orderings and favor the ordering resulting from the IOU matching process due to the usage in literature as in [4] and [5]. However, the IOU matching procedure requires an overlap of the objects in the frames $f$ and $f-1$. As soon as the objects move between the frames in a higher speed, which is plausible with bees, their bounding boxes do not overlap anymore which results in IOU scores to be 0. The overlap matrix $O$ is sparser compared to the distance matrix $D$. Whenever this is the case, we favor the association of the distance matching procedure.

As a last step before definitely assigning the incoming objects of the current frame to the registered objects, a comparison with a threshold is done. Without a threshold, suppose we only have one object in frame $f-1$ and only one object in frame $f$, which are not the same and are located on the opposite site in each frame. By simply applying the matching procedure, they would end up with a IOU score in the overlap matrix $O$ of 0 and a large distance in the distance matrix $D$. However, because these are the only objects which can be compared between $f-1$ and $f$, it would be the best fit even though, they are not the same object. By introducing a threshold for the maximal distance and minimal IOU score, we can limit the matching procedure to only assigning a match in close proximity. On the one hand, the IOU score of the match has to be equal or above $\sigma_{IOU}$. Or on the other hand, the distance of the match has to be below or equal to $\sigma_d$. The distance threshold $\sigma_d$ should behave inverse proportional to the frame rate. This is because with lower frame rates the movement of the objects between $f-1$ and $f$ is larger and therefore the threshold for distance is allowed to be larger. The IOU threshold $\sigma_{iou}$ is proportional to the frame rate because with smaller movement the overlap should be higher for matching objects. At least one of these criteria has to be met in order to assign an incoming object to a registered object. The absence of especially the distance threshold $\sigma_d$ would allow matches of objects which are far away from each other and physically not possible to be the same object. Both $\sigma_d$ and $\sigma_{IOU}$ can be set manually and depend on the frame rate of the video to adjust to the distance the objects can travel between frame $f-1$ and $f$. If one of these criteria is met, the matches are sent back on path $a_3$ visualized in figure 4 in order to update the registry with the newly obtained locations of the registered objects and the IDs of the disappeared objects in the current frame. However, if the proposed match between registered and incoming object does not meet the requirements in form of the thresholds, the registered object would be reported as disappeared.

### Missing Objects

Missing objects originate from two different sources, either an object has left the frame or a false negative detection has been made during the object detection [5]. Because of these two sources, handling missing objects has a vital impact on the tracking task at hand. Bees can disappear in the video either by entering the hive or by flying out of the hive and out of the perception field of the camera. Whenever a bee is not visible anymore because it disappeared either in the hive or flew out, it could directly be deregistered. However, the tracking algorithm can not distinguish between a missing bee because it disappeared or because of a false negative detection. Therefore, every missing object has to be handled as a false negative detection and further looked for in the next $t$ frames.

A missing object results in a non square matrix during the matching process in both the distance matrix $D$ and the overlap matrix $O$ if for both $D \in \mathbb{R}^{m*n}$ and $m > n$ and $O \in \mathbb{R}^{m*n}$ and $m > n$. Since we only can assign visible objects in the current frame $f$, the resulting matrices $D_{ordered}$ and $O_{ordered}$ will have a square shape such that $D_{ordered} \in \mathbb{R}^{n*n}$ and $O_{ordered} \in \mathbb{R}^{n*n}$. The objects $m-n$ are from $f-1$ where no incoming object of $f$ can be matched with are sent via path $a_3$ back to the registry after the

matching process is finished. Every object that could not be matched in the current frame get sent via path $a_4$ to a disappearance handler. Within this handler every object will be checked for the number of frames this object could not be matched with an incoming object of the current frame $f$. This threshold $\sigma_{disappeared}$ is also dependent on the frame rate of the video and represents the memory of the tracking algorithm to keep looking for a missing object. If we work with a high frame rate, the movements of the objects are smaller and therefore more subsequent detections can be missed out in order to find the same object in close proximity again. If $\sigma_{disappeared} = 0$ the MOT has no memory of missing objects and would directly send the object via path $a_7$ to the deregistration process. However, if $\sigma_{disappeared} = t$ with $t > 0$ the object will be kept registered in the registry without updating the current location for the next $t$ frames on path $a_6$. The objective with $t > 0$, which is represented as path $a_5$, is to overcome false negative detections and reassign the object during the matching process as soon as the object is detected again. For the case if $\sigma_{disappeared} = t$ and $t >> 0$ disappeared objects are kept alive for a longer time and an ID switch becomes possible with another object in the exact same location as the last known location of the disappeared object. Whenever an object has been able to be matched again and the current location is updated in the registry, the counter $\sigma_{disappeared}$ gets reset.

If the number of frames where the object could not be matched with any incoming objects such that $\sigma_{disappeared} > t$, the object is sent via $a_7$ to the deregistration process. During this process all attributes of the object are deleted and we can further investigate whether the object entered the or left the hive for the sake of counting the bees.

### Counting

The initial goal of the bee tracking is to track bees over time to establish a counting system in order to be aware of how many bees enter and leave the hive. With the five life stages displayed in figure 4 we maintain a registry of how many bees are active in the current frame and how many bees have been deregistred due to disappearance in a number of subsequent frames. In order to tell how many bees actually entered or left the hive, only the deregistered objects and their last known location are informative.

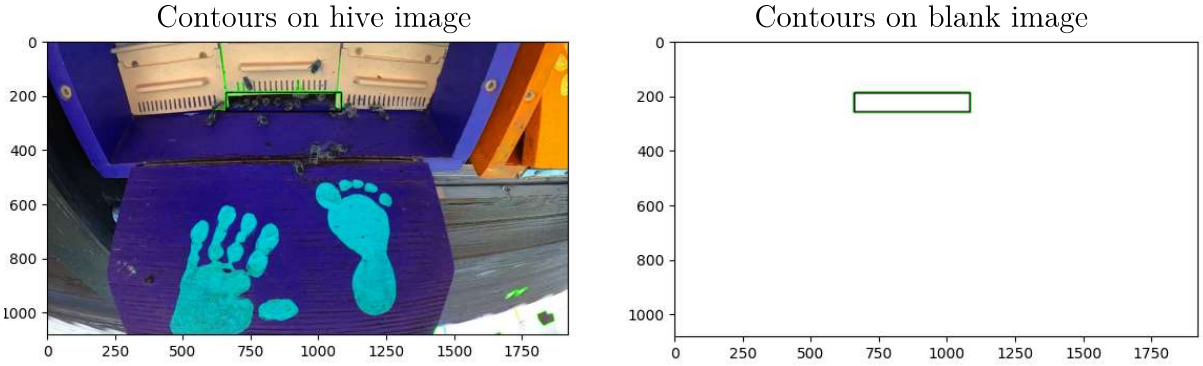



Figure 6: Left: detected contours on an image extracted from a hive video and manually drawn rectangle around the hive entry. Right: detected contours on a blank white image with a rectangle with the same coordinates as drawn on the extracted image on the left

Over the span of the project, we have analyzed a number of different bee hives and most of those bee hives have different entries. Some are round and some are rectangular. Their location in the video are heavily dependent on how the camera was installed in front of the hive. This is why we established the definition of the entry as a manual process before being able to track and count the bees in this particular hive. Detecting the entry of a bee hive could be an own detection problem by itself, on which we did not focus on this project. Therefore, we first extract an arbitrary image from the video of a given hive with its height and width. Relative to those measures we draw either a circle or a rectangle, depending on the actual entry of the hive in the image, manually on top of the extracted image and record the coordinates of the drawn geometric form. With the help of the Python implementation of OpenCV [16], it is feasible to detect contours as continuous points along a specific color within an image. However, after searching for contours in the extracted image from a bee hive, not only the drawn geometric form as the entry was

found but also other contours in the image. This problem can be observed on the left side in figure 6 where many different contours are detected around the entry and on the lower right part of the image. This is why we settled for an alternative contour detection with less noise in it. By creating a blank white image with the same height and width as the selected image from the video and the geometric form with the same relative coordinates, the contour detection only finds the drawn entry of the hive. This method results in a clean detected contour in figure 6 on the right side.

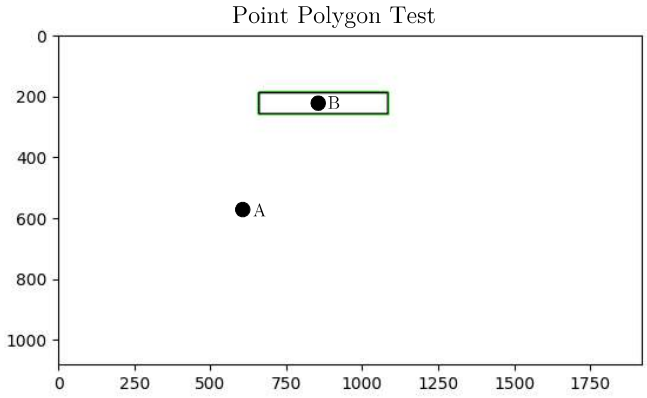



Figure 7: Detected contours on a blank white image with a point polygon test of points A and B

The extrapolation of the detected contours of a blank white image on the frames of the video from the Doettingen hive results in a convex hull as described in [15]. The goal of working with a convex hull is to check with the OpenCV function of the point polygon test, whether a point lies within a contour or outside of it [17]. The applied principle of the point polygon test can be seen in figure 7, where two points are plotted. The coordinates of point A lie outside of the contour and point B inside of it. By applying the point polygon test on both points with respect to the contour we obtain $-1$ for point A for being outside of the contour and $1$ for point B for being inside of the contour. If a point directly lies on the contour the obtained value would be $0$. For every frame and every object detected, we check for the object being inside or outside the contour which is equivalent of being inside the hive entrance or outside of the hive entrance. Similarly as the past positions of the objects, we also record whether these positions were inside or outside of the contour. As soon as an object gets deregistered because it could not be found after a period of $t$ frames, the history of positions gets analyzed with respect to the contour. We defined two policies of whether a pattern assembles to a bee entering the hive or leaving the hive. These policies incorporate a threshold $\sigma_f$ (equation 5) that is dependent on the frame rate of the video. Because a higher frame rate captures objects with smaller distances, the threshold $\sigma_f$ is allowed to be higher than with smaller frame rates. For a bee to be counted as entering the hive the very first position of the bee has to be outside of the contour (point polygon test returning $-1$), there has to be at least $\sigma_f - 1$ detections of the bee outside of the contour and the last $\sigma_f$ detections have to be inside the contour (point polygon test returning $1$). Similarly, for a bee to be counted as exiting the hive, the first position has to be within the contour with at least $\sigma_f - 1$ detections inside the contour and the last $\sigma_f$ detections outside the contour.

$$\sigma_f = \lfloor \frac{FPS}{25} \rfloor \tag{5}$$

### Visual Representation

During the development of the matching process the visual representation was key to identify the successful assignment between frames and the same bees. However, the frame by frame processing and storing the visual representation within a resulting video is computationally expensive. The resulting video consists of the input video and its detections and is therefore an exact copy of the video with additional annotations to it. Even though the development code with the visualized output can not be applied because of the aforementioned reasons for a streamlined and scaleable counting system, it may be very insightful to study the behaviour of bees around the hive. Therefore, we created two versions;

the visualized counting and a more efficient variant of headless counting, which only keeps track of the traffic of bees.

Both systems consist of a loop over all frames and the MOT which proceeds after the five life stages in figure 4. Before starting to loop over an input video, the video has to be analyzed by the object detection and all its results have to be stored in a database. Then, as described in the counting paragraph, an arbitrary image has to be extracted and the entry to the bee hive manually drawn. As soon as these three inputs (video, detections, coordinates of entry) are provided, both counting systems extrapolate the coordinates of the entry to a blank white image and detect the contours of the bee hive entrance. Furthermore, the frame rate of the video to be analyzed will be extracted from the video and the actual tracker initialized relative to the frame rate of the video. The MOT requires four inputs: the distance threshold $\sigma_d$, the IOU threshold $\sigma_{iou}$, the memory threshold of missing detection $\sigma_{disappeared}$ and the maximal number of the trace, which only is required for visualization purposes. The visualized counting system reads in the input video and the maximal number of frames from the database, whereas the headless counting system only requires the maximum frames. Within a loop, the systems start with the first frame, extract all coordinates detected by the object detection from the database. All coordinates from the first frame complete the life stages in figure 4. As a result the tracker returns again the coordinates but this time with an ID and all past locations as centroids. Next, all current coordinates are checked whether the points are located within the contour of the drawn entrance or not. Every object and its position relative to the contour based on the point polygon test are stored in a separate dictionary. This dictionary gets extended with the position relative to the contour for every frame. As soon as the object is not registered anymore, those positions get analyzed by the policy from the counting paragraph and the counting parameters get updated if the policies hold. At the end, the visualized counting system annotates the frame with the current location, the ID and the tracks recorded for every object. Additionally, the counters for the number of bees flown in or out, the last recorded activity, the frame rate and the frame currently processed get displayed on the bottom left of the frame. As soon as all frames of the video are processed, the visualized counting system stores the output as a video file. Both counting systems print out the resulting counts of the whole traffic analyzed at the end of the loop. An example of the output in video format can be seen in figure 8.


Figure 8: Extraction of a frame during the visualized counting system loop with the contour around the entrance, the bees with their current location and past location as trace and the information on the bottom left

### Frame Rate Experiment

As repeatedly mentioned in the sections above, the parameters for the headless and visualized counting system strongly depend on the frame rate of the input video. A higher frame rate in the video captures more frames per second. The cameras utilized to record test videos were able to film with 25, 50, 100 and 200 FPS. If we consider a video with 200 frames per second, the counting system has twice as many frames to analyze the traffic of bees compared to a video with 100 FPS. The more frames per second a video provides, the smaller are the distances of moving objects within that video. This actively demonstrates that in general objects travel shorter distances between the frames and false negative detections have a lower chance to miss out reassignment. This section aims to analyze how well the counting systems can cope with the different frame rates of videos and to give a recommendation of which frame rate works best for our counting systems.

Therefore, we process the same video with the four frame rates and compare their output. Even though, we are provided with video sequences for each of the FPS settings, we decided to work with a single video recorded with 200 FPS. In order to establish a valid experimental foundation for comparison purposes, we scale down the same 200 FPS video to 100, 50 and 25 FPS by skipping frames. This is achieved by only looking at every second frame for 100 FPS, every fourth frame for 50 FPS and every eighth frame for 25 FPS. For each of these FPS settings we need to initiate the MOT with the parameters for the distance threshold $\sigma_d$, the IOU threshold $\sigma_{iou}$, the memory threshold $\sigma_{disappeared}$ and the trace length. Both distance and IOU threshold can be set by example of a fast moving bee. Because such a bee would state the edge case for our counting system to still make a correct assignment and therefore represent the maximal distance and IOU score for the matching process. We located a exemplary bee which flies out of the hive in the middle of the aspect field and extracted the last two coordinates of the respective frames for each of the four FPS settings. The numbers we obtained by analysing a flying bee can be seen in table 12. The coordinates of the bounding box for the frame $f$ are all the same for each frame rate setting because it is the last detection of the bee flying out. The only coordinates that change are the ones from frame $f-1$, which is with decreasing frame rate further away from the last detection. This can be observed by looking at the IOU scores and the euclidean distances for each frame rate setting. Even with 200 FPS the last two coordinates only overlap to $52.03\%$. The bounding boxes of the last two detections with 25 FPS do not overlap at all and result therefore in a IOU score of $0\%$.

| Frame Rate | Coordinates $f-1$ | Coordinates $f$ | IOU Score | Euclidean Distance |
|---|---|---|---|---|
| 200 FPS | (803, 458, 843, 529) | (800, 473, 840, 558) | 0.5203 | 32.9242 |
| 100 FPS | (807, 441, 840, 512) | (800, 473, 840, 558) | 0.2598 | 56.9034 |
| 50 FPS | (808, 410, 853, 475) | (800, 473, 840, 558) | 0.0102 | 105.3138 |
| 25 FPS | (814, 345, 854, 400) | (800, 473, 840, 558) | 0 | 204.3037 |

Table 12: Threshold indication with bounding box coordinates in the form of ($x_{min}$,$y_{min}$,$x_{max}$,$y_{max}$) of two frames with their respective IOU scores and euclidean distances for each analyzed frame rate setting

In order to use these observed values for an edge case of a fast moving object, we added a small margin to cover other even more extreme cases. Therefore, we reduced the values for the IOU threshold $\sigma_{iou}$ from the observed IOU value of the last two frames of the flying bee. The only exception is the IOU threshold for the 25 FPS setting, where the observed IOU score was 0. If the IOU threshold would be adjusted to 0, every match would be possible and random assignments of missing objects can happen. Thus, we set the IOU threshold for 25 FPS to 0.0025 and let the distance be up to 250. The selected thresholds can be seen in table 13. The goal of different thresholds for each frame rate setting is to overcome the missing frames and the resulting distances the objects are apart from $f-1$ to $f$.

| Frame Rate | IOU threshold $\sigma_{iou}$ | Distance threshold $\sigma_d$ | Memory $\sigma_{disappeared}$ in frames |
|---|---|---|---|
| 200 FPS | 0.5 | 50 | 20 |
| 100 FPS | 0.2 | 100 | 15 |
| 50 FPS | 0.005 | 150 | 10 |
| 25 FPS | 0.0025 | 250 | 5 |

Table 13: Actual thresholds used for different frame rate setting for the counting systems

For the purpose of comparing the traffic outcome of the different frame rate settings, we utilized

our headless counting system and adjusted it to produce a more informative output. Therefore, not only the bees which flew in or out were counted but also their tracks were stored. We then plotted for better visibility the tracks of those bees on the blank white image, which was used to detect the bee hive entrance. First we produced for the each of the frame rate settings a visualized version of all incoming bees with their tracks in green and all outgoing bees with their tracks in red. The different outputs can be observed in figure 9. The main difference between the frame rate settings appears to be the length of the tracks especially of the incoming bees. When comparing the incoming count of the different frame rates, there were 90 bees counted in for 200 FPS and only around 55 for the remaining frame rates. Even though, the policy for incoming bees for lower frame rates are less restrictive, such that for example for 100 frames per second, only 4 detections have to be within the contour of the hive entrance. However, the outgoing bees seem to be counted in a more comparable way since this policy allows objects with the first detection within the small contour and depending on equation 5 the number of detections outside of the contour. It can be observed that the policy for an incoming bee is harder to fulfill because the entrance is a narrow region with many bees in it. This may result in random switches if objects are not visible due to false negative detections or other bees crawling over each other. Additionally, it can be seen that the lower the frame rate gets, the less tracks are recorded in the middle to lower part of the figures in 9. This is due to the movement speed of the bees in this area of the bee hive. The faster the bees move, the harder it is to reassign them during the matching process with lower frame rates. However, around the hive the speed of the bees are lower and therefore more tracks are recorded for all of the frame rate settings.

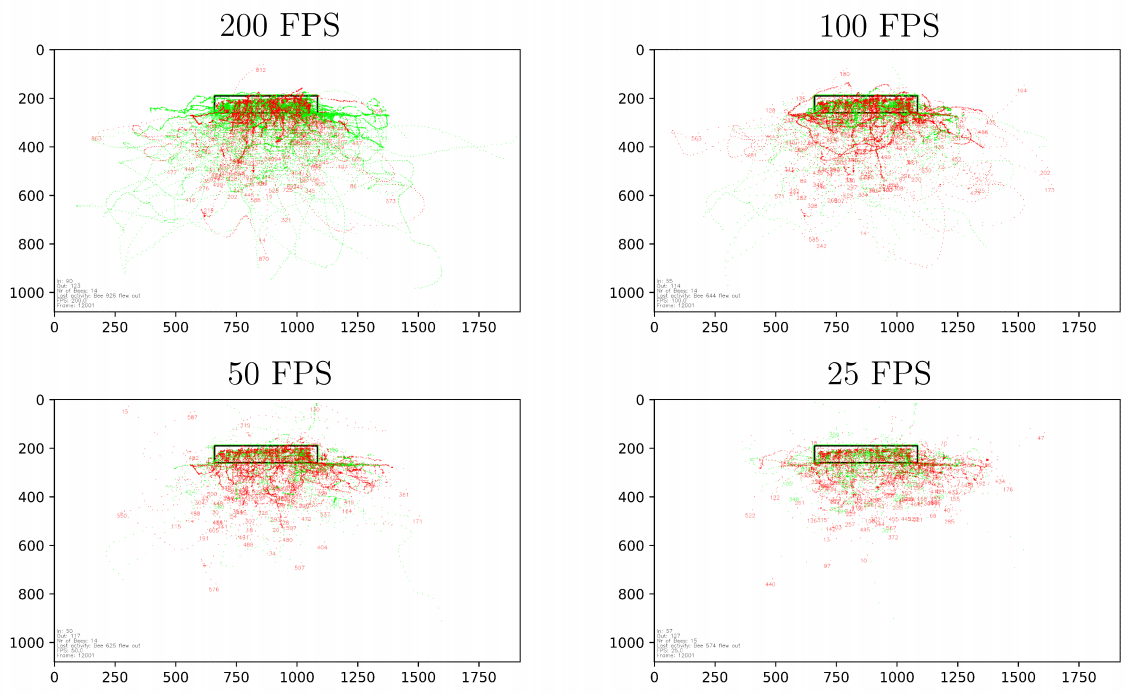



Figure 9: Tracks of bees flying in (green) and bees flying out (red) with the four analyzed FPS settings

When comparing the different tracks in figure 9 one can observe that every frame rate setting captures unique tracks of the bees. Even though, every frame rate setting works with the exact same detections, lower FPS rates are still able to draw different tracks of incoming and outgoing bees, which the 200 FPS setting is not able to capture. When not only focusing on incoming and outgoing bees but also all other bees which were not counted, we obtain a much more comprehensive overview of the traffic around the hive. This experiment can be seen in figure 10. The tracks which do not belong to an object that was finally counted in or out are colored in blue and make up the majority of tracks. It may be

that these tracks belong to bees which only visit the hive and do not enter the hive. However, the more plausible hypothesis for the blue tracks is that these bees were lost by the counting system on the way to the hive or out of the hive and thus registered as new objects. The fact that a bee has been lost does not imply that it was not counted. It may be that a bee was counted as leaving the hive correctly and on the way out of the hive due to a number of false negative detections deregistered. The deregistration results from multiple failed attempts to reassign the same object after false negative detections. It may be either too far away from the last known detection and can not be matched with the current detection due to the thresholds $\sigma_{iou}$ and $\sigma_d$. Or on the other hand, if the object suffered from false negative detections for more frames than the memory $\sigma_{disappeared}$ allowed it to be. This is why it is vital to set these thresholds accordingly with an aim to optimize the matching process after false negative detections.

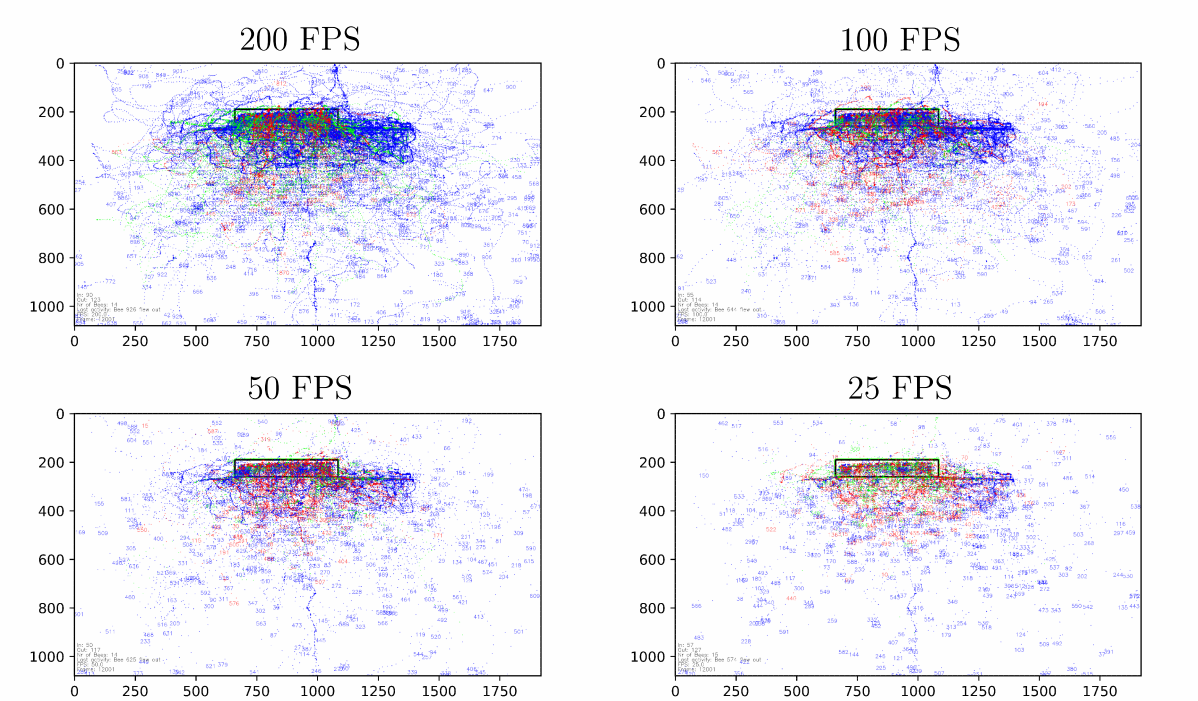



Figure 10: Tracks of bees flying in (green) and bees flying out (red) and bees not counted in or out (blue) with the four analyzed FPS settings

Overall, the images in figure 9 and 10 are more colorful with higher frame rates than lower FPS settings. This is due to the fact that the counting system draws for every frame and every bee a dot and therefore 200 FPS visualizes for each object 8 times more points than for 25 FPS. This also results in more accurately monitored and therefore longer tracks of the bees. Both, the headless and visualized counting systems were streamlined to process 200 FPS videos and to track bees for one specific hive. There might be more potential with further optimization of the thresholds and improved object detection models. However, whenever it is possible to record a bee hive with high FPS settings, it is advisable to do so in order to closely track the activities of the objects and therefore study the accurate behavior of bees.

## BeeLivingSensor Cloud Platform Project

BeeLivingSensor is a long term project of the ETHZ-IMSB, we helped recruiting the following team that would work in the project. For that we also defined the user and system functionality basic requirements as well as a concept of MVP. Based on our experiences with our work with object detection and training we provided knowledge-transfer and worked together with the team to define process flows for the main functionality of the platform like adding new beehives, training new models and validating them. We provided additionally the trained models as well as our source code.

## Future work

There is a huge exploration space of the dimensions that can be changed to improve the object detection. However we believe that with our project we show that with a small number of images and training in one or two directions it is possible to reach high performance for the resulting models for the most common beehives. Research on augmentation methods like Cropping, Flipping, WGAN, Rotation [20], could be useful to tackle beehives with more complex backgrounds.

The average 4% difference in recall that the general model showed against the top models of each hive opens the question if adding more data for training a general model might surpass this difference. An important consideration would be to constantly review the efforts against the gain in the performance of the models, because like we observed in the staged training, when mixing beehives that are not similar, some models actually reduced it's performance.

A promising area for future work would be to test FastYOLO, that can reach up to 155 frames per second while still achieving double the mAP of other real-time detectors [18]. This additionally to the use of IoT RaspberryPi Cameras [2] and the easy implementation of IoT devices in the Azure Platform could make a viable display of modular sensors that would reduce the traffic by avoiding the continuous upload of heavy video raw data, except for random samples for validation.

The automated detection of the entrance of a bee hive could also facilitate the addition of new hives in a more automated fashion. As described in our report, we overcame this problem by manually drawing the entrance on top of a frame and then extrapolating the coordinates to a blank white image. This procedure is not transferable to another video with different angles and certainly not to a different bee hive. Another approach would be to provide the user two types of entrances (round, square) and advise the installation of the camera to comply by annotating the visual aspect field of the camera with the target location of the bee hive entrance.

The future work regarding the inner working of tracking mainly focuses on the matching process. There exist at least two known methods to overcome false negative detections by predicting the motion and future location of the object. On the one hand, there is the application of a Kalman Filter as used in [13]. On the other hand, a trend is observable by applying a Long Short-Term Memory (LSTM) network in order to analyze the tracks of the bees and predict the target position as it is applied in [22]. When considering more advanced methods of tracking it has always be kept in mind that the majority of research is done by experimenting with tracking people or cars. Some techniques however, for example a specific feature extraction technique can be used in order to further distinguish between different objects, can only be partially applied to tracking bees, because bees do not differ in appearance as much as people or cars. It is likely that in the context of bees, future work has to rely mainly on spacial features instead of visual features.

An important aspect to implement in future projects is testing the tracking of a specific video against a benchmark in order to optimize the thresholds of the MOT. This can be done by manually labeling the whole video length with bounding boxes and IDs. Then, the proposed assignments for every frame and every object can be compared to the ground truth and further analyzed for threshold optimization.

## Conclusion

Our contribution to the *BeeLivingSensor* in the field of Object Detection was to prepare the leg work of testing different paths, generating the top ranking models that reach a recall higher of 90% for all relevant hives, like shown in Figure 11. This models were imported in the Azure platform and creating a standard cross-validation process. In the scope of our project we demonstrated that before investing efforts in label thousands of frames and processing all data for augmentation or training general models, it is worth exploring the effectiveness of the approaches.

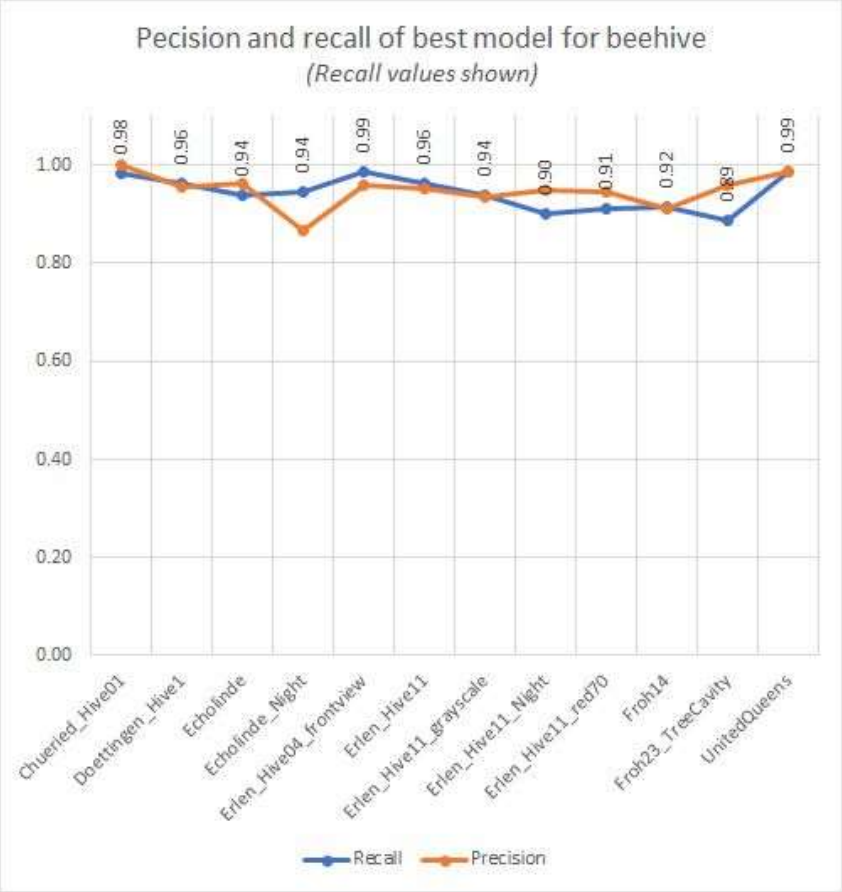



Figure 11: Recall and precision of top models for relevant beehives

Especially when recording bees in different settings with varying camera settings, aspect angles, aspect ratios, camera location, bee hive background, light conditions and many more. These variations in the data foundation have to be compensated by the amount of data to get the object detection system to generalize well over all these varying conditions. In order to enable reasonable MOT results in all these different scenarios, there have to be object detection models in place, which deliver consistent and reliable results. Because our results, we are inconclusive on the question if a general model that includes many more training images might reach a better performance than the models that were top, we mention this in Future work.

To summarize the second part of this report, we extended the model of [22] to five life stages and led the foundation for a counting system for bees almost from scratch. For every frame in an input video these five life stages are performed. First the bee hive entrance gets detected with a trick on a blank white image. Then the coordinates of the input video get extracted from a database and registered. To assign the coordinates of a subsequent frame, always the last frame $f-1$ and the current frame $f$ are analyzed and the coordinates get matched with already registered objects within a matching process. Whenever an object can not be matched, it will be noted as disappeared. The counting systems work with a memory to overcome false negative detections in order to reassign an object even though the coordinates are not present in the current frame $f$. If the memory is exhausted, the registered object gets deregistered from the registry. After the deregistration process, the past locations of these objects gets

analyzed and counted towards incoming or outgoing bees. To conclude, we developed two counting systems that employ the same counting methodology but differ in the produced output. The visualized counting system outputs a video for the comprehension of the results and was mainly used during the development phase. In order to work towards a more efficient counting process running on a cloud platform, we also adjusted the visualized counting system and cropped out the computationally expensive visualization part to only produce a summary of the traffic for the input video and the coordinates saved in the database beforehand. The counting systems deliver reasonable results for videos recorded with 200 frames per second. However, further development on the object detection part directly influences the quality of the object tracking. Additionally, also the object tracking and counting is subject to improvement by considering more advanced methodologies for example in the matching process. However, the systems provide a novelty in tracking bees within a streamlined pipeline.

## Advisors:

**Research Associate. Informatics and Sustainability Research UZH**
Dr. Clemens Mader
**Project Leader. Institute of Molecular Systems Biology ETHZ**
Mr. Daniel Boschung
**Professor. Informatics and Sustainability Research UZH**
Dr. Lorenz Hilty

**Signatures:**

Gabriela Eugenia López Magaña

Dr. Clemens Mader

Pascal Engeli

Prof Dr. Lorenz Hilty

# References

[1] S. Albanie. Euclidean distance matrix trick. *Retrieved from Visual Geometry Group, University of Oxford*, 2019.

[2] Z. Babic, R. Pilipovic, V. Risojevic, and G. Mirjanic. Pollen bearing honey bee detection in hive entrance video recorded by remote embedded system for pollination monitoring. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, III-7:51–57, June 2016.

[3] L. Batanina. opencv. support yolov4 17185, 2020.

[4] E. Bochinski, V. Eiselein, and T. Sikora. High-speed tracking-by-detection without using image information. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE, 2017.

[5] E. Bochinski, T. Senst, and T. Sikora. Extending iou based multi-object tracking by visual information. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE, 2018.

[6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[7] H. Cui, J. Zhang, C. Cui, and Q. Chen. Solving large-scale assignment problems by kuhn-munkres algorithm hong cui1, jingjing zhang2, b, chunfeng cui3, c, qinyu chen4, d. 2016.

[8] P. De Souza, P. Marendy, K. Barbosa, S. Budi, P. Hirsch, N. Nikolic, T. Gunthorpe, G. Pessin, and A. Davie. Low-cost electronic tagging system for bee monitoring. *Sensors*, 18:2124, 07 2018.

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[10] T. Hudson. Easy bee counter, 2019.

[11] D. J. Kale, R. Tashakkori, and R. M. Parry. Automated beehive surveillance using computer vision. In *SoutheastCon 2015*, pages 1–3, 2015.

[12] G. Khan, Z. Tariq, and M. U. G. Khan. Multi-person tracking based on faster r-cnn and deep appearance features. In *Visual Object Tracking in the Deep Neural Networks Era*. IntechOpen, 2019.

[13] X. Li, K. Wang, W. Wang, and Y. Li. A multiple object tracking method using kalman filter. In *The 2010 IEEE international conference on information and automation*, pages 1862–1866. IEEE, 2010.

[14] D. Misra. Mish: A self regularized non-monotonic activation function, 2020.

[15] OpenCV. Contour features. `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contour_features/py_contour_features.html`, 2021. Version 4.5.1-dev.

[16] OpenCV. Contours: Getting started. `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_begin/py_contours_begin.html#contours-getting-started`, 2021. Version 4.5.1-dev.

[17] OpenCV. Contours: More functions. `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_more_functions/py_contours_more_functions.html`, 2021. Version 4.5.1-dev.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection, 2016.

[19] A. Rosebrock. Opencv people counter. `https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/`, 2018.

[20] J. Shijie, W. Ping, J. Peiyi, and H. Siping. Research on data augmentation for image classification based on convolution neural networks. In *2017 Chinese Automation Congress (CAC)*, pages 4165–4170, 2017.

[21] Stack Overflow. Calculating percentage of bounding box overlap, for image detector evaluation. `https://stackoverflow.com/questions/25349178/calculating-percentage-of-bounding-box-overlap-for-image-detector-evaluation`, 2019.

[22] J. Xiang, G. Zhang, J. Hou, N. Sang, and R. Huang. Multiple target tracking by learning feature representation and distance metric jointly. *arXiv preprint arXiv:1802.03252*, 2018.

[23] Y. Xiao, Z. Tian, J. Yu, Y. Zhang, S. Liu, S. Du, and X. Lan. A review of object detection based on deep learning. *Multimedia Tools and Applications*, 79(33):23729–23791, Sept. 2020.

[24] A. Zacepins, E. Stalidzans, and J. Meitalovs. Application of information technologies in precision apiculture. In *Proceedings of the 13th International Conference on Precision Agriculture (ICPA 2012)*, 2012.

# Apendix

| Trained Model | Precision | Recall | F2 - score | F1 - score |
|---|---|---|---|---|
| staged_2/all_HD_hives_40 | 0.94 | 0.81 | 0.83 | 0.87 |
| staged_2/BeeWatch_Chueried_01_Erlen_Hive_11_Froh_23_UnitedQueens | 0.95 | 0.76 | 0.79 | 0.85 |
| staged_2/Chueried_Hive01_UnitedQueens | 0.86 | 0.36 | 0.40 | 0.50 |
| staged_2/Doettingen_Echolinde_Erlen_UnitedQueens | 0.94 | 0.72 | 0.76 | 0.81 |
| staged_2/Doettingen_Echolinde_Erlen_UnitedQueens_Froh14 | 0.96 | 0.68 | 0.72 | 0.79 |
| staged_2/Erlen_Hive04_Hive11_UnitedQueens | 0.95 | 0.67 | 0.71 | 0.79 |
| staged_2/Erlen_Hive11_UnitedQueens | 0.91 | 0.54 | 0.59 | 0.68 |
| staged_2/Froh14_20_Chueried_Hive01 | 0.90 | 0.55 | 0.59 | 0.68 |
| individual_training/Chueried_Hive01 | 0.94 | 0.22 | 0.27 | 0.36 |
| individual_training/Chueried_Hive01_red70 | 0.83 | 0.39 | 0.43 | 0.53 |
| individual_training/Erlen_Hive11 | 0.93 | 0.62 | 0.66 | 0.74 |
| individual_training/Froh14 | 0.84 | 0.38 | 0.42 | 0.52 |
| individual_training/Froh23_TreeCavity | 0.88 | 0.45 | 0.50 | 0.60 |
| on_qty_of_train_images/Echolinde_10 | 0.82 | 0.51 | 0.55 | 0.63 |
| on_qty_of_train_images/Echolinde_20 | 0.75 | 0.64 | 0.66 | 0.69 |
| on_qty_of_train_images/Echolinde_40 | 0.75 | 0.60 | 0.62 | 0.67 |
| on_qty_of_train_images/Echolinde_80 | 0.90 | 0.60 | 0.64 | 0.72 |
| on_qty_of_train_images/Froh14_10 | 0.90 | 0.28 | 0.33 | 0.43 |
| on_qty_of_train_images/Froh14_160 | 0.86 | 0.49 | 0.54 | 0.63 |
| on_qty_of_train_images/Froh14_20 | 0.83 | 0.48 | 0.53 | 0.61 |
| on_qty_of_train_images/Froh14_40 | 0.79 | 0.48 | 0.52 | 0.60 |
| on_qty_of_train_images/Froh14_80 | 0.86 | 0.49 | 0.54 | 0.63 |
| single_hive/Chueried_Hempbox | 0.93 | 0.01 | 0.01 | 0.01 |
| single_hive/Chueried_Hive01 | 0.88 | 0.25 | 0.29 | 0.39 |
| single_hive/ClemensYellow | 0.72 | 0.19 | 0.22 | 0.30 |
| single_hive/Doettingen_Hive1 | 0.81 | 0.66 | 0.69 | 0.73 |
| single_hive/Echolinde | 0.90 | 0.60 | 0.64 | 0.72 |
| single_hive/Echolinde_Night | 0.94 | 0.33 | 0.38 | 0.49 |
| single_hive/Erlen_Hive04_diagonalview | 0.94 | 0.45 | 0.51 | 0.61 |
| single_hive/Erlen_Hive04_frontview | 0.96 | 0.43 | 0.48 | 0.59 |
| single_hive/Erlen_Hive04_smartphone | 0.94 | 0.36 | 0.41 | 0.52 |
| single_hive/Erlen_Hive11 | 0.91 | 0.59 | 0.64 | 0.72 |
| single_hive/Erlen_Hive11_grayscale | 0.91 | 0.76 | 0.78 | 0.83 |
| single_hive/Erlen_Hive11_Night | 0.92 | 0.46 | 0.51 | 0.61 |
| single_hive/Froh14 | 0.86 | 0.49 | 0.54 | 0.63 |
| single_hive/Froh23_TreeCavity | 0.89 | 0.40 | 0.45 | 0.55 |
| single_hive/UnitedQueens | 0.96 | 0.30 | 0.34 | 0.45 |

Table 14: General Model Performance

Table 15: Leaky vs Mish Activation Function Results

| Beehive name Activation function | Trained Model | TP | FP | FN | Precision | Recall | F2 - score | F1 - score |
|---|---|---|---|---|---|---|---|---|
| **Chueried_Hempbox** | | | | | | | | |
| Mish | single_hive/Chueried_Hempbox | 85 | 6 | 689 | 0.93 | 0.11 | 0.13 | 0.00 |
| Leaky | on_qty_of_train_images/Echolinde_20 | 101 | 2663 | 575 | 0.04 | 0.15 | 0.09 | 0.00 |
| **Chueried_Hive01** | | | | | | | | |
| Mish | single_hive/all_hives_trainP | 130 | 0 | 2 | 1.00 | 0.98 | 0.99 | 0.00 |
| Leaky | individual_training/Chueried_Hive01_red70 | 88 | 4 | 44 | 0.96 | 0.67 | 0.71 | 0.00 |
| **Chueried_Hive01_green80** | | | | | | | | |
| Mish | single_hive/all_hives_trainP | 120 | 1 | 12 | 0.99 | 0.91 | 0.92 | 0.00 |
| Leaky | individual_training/Chueried_Hive01_red70 | 91 | 5 | 41 | 0.95 | 0.69 | 0.73 | 0.00 |
| **Chueried_Hive01_red70** | | | | | | | | |
| Mish | single_hive/all_hives_trainP | 120 | 0 | 12 | 1.00 | 0.91 | 0.93 | 0.00 |
| Leaky | individual_training/Chueried_Hive01_red70 | 93 | 14 | 39 | 0.87 | 0.70 | 0.73 | 0.00 |
| **ClemensRed** | | | | | | | | |
| Mish | single_hive/ClemensYellow | 185 | 49 | 151 | 0.79 | 0.55 | 0.59 | 0.00 |
| Leaky | single_hive/ClemensYellow | 201 | 51 | 135 | 0.80 | 0.60 | 0.63 | 0.00 |
| **ClemensYellow** | | | | | | | | |
| Mish | single_hive/ClemensYellow | 370 | 114 | 116 | 0.76 | 0.76 | 0.76 | 0.00 |
| Leaky | single_hive/ClemensYellow | 338 | 107 | 148 | 0.76 | 0.70 | 0.71 | 0.00 |
| **Doettingen_Hive1** | | | | | | | | |
| Mish | single_hive/Doettingen_Hive1 | 1253 | 57 | 48 | 0.96 | 0.96 | 0.96 | 0.00 |
| Leaky | on_qty_of_train_images/Doettingen_Hive1_160 | 1238 | 272 | 63 | 0.82 | 0.95 | 0.92 | 0.00 |
| **Echolinde** | | | | | | | | |
| Mish | single_hive/Echolinde | 852 | 22 | 69 | 0.97 | 0.93 | 0.93 | 0.00 |
| Leaky | on_qty_of_train_images/Echolinde_20 | 837 | 64 | 84 | 0.93 | 0.91 | 0.91 | 0.00 |
| **Echolinde_Night** | | | | | | | | |
| Mish | single_hive/Echolinde_Night | 408 | 63 | 24 | 0.87 | 0.94 | 0.93 | 0.00 |
| Leaky | single_hive/Erlen_Hive11_Night | 366 | 29 | 66 | 0.93 | 0.85 | 0.86 | 0.00 |
| **Erlen_Hive04_diagonalview** | | | | | | | | |
| Mish | single_hive/Erlen_Hive04_diagonalview | 563 | 18 | 44 | 0.97 | 0.93 | 0.94 | 0.00 |
| Leaky | single_hive/Erlen_Hive04_diagonalview | 522 | 46 | 85 | 0.92 | 0.86 | 0.87 | 0.00 |
| **Erlen_Hive04_frontview** | | | | | | | | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Mish | single_hive/Erlen_Hive04_frontview | 561 | 14 | 9 | 0.98 | 0.98 | 0.98 | 0.00 |
| Leaky | single_hive/Erlen_Hive04_frontview | 543 | 10 | 27 | 0.98 | 0.95 | 0.96 | 0.00 |
| **Erlen_Hive04_smartphone** | | | | | | | | |
| Mish | single_hive/Erlen_Hive04_diagonalview | 279 | 26 | 23 | 0.91 | 0.92 | 0.92 | 0.00 |
| Leaky | single_hive/Erlen_Hive04_diagonalview | 255 | 29 | 47 | 0.90 | 0.84 | 0.85 | 0.00 |
| **Erlen_Hive11** | | | | | | | | |
| Mish | single_hive/all_hives_trainP | 1571 | 79 | 61 | 0.95 | 0.96 | 0.96 | 0.00 |
| Leaky | individual_training/Erlen_Hive11 | 1523 | 129 | 109 | 0.92 | 0.93 | 0.93 | 0.00 |
| **Erlen_Hive11_Night** | | | | | | | | |
| Mish | single_hive/Erlen_Hive11_Night | 710 | 38 | 78 | 0.95 | 0.90 | 0.91 | 0.00 |
| Leaky | single_hive/Erlen_Hive11_Night | 717 | 55 | 71 | 0.93 | 0.91 | 0.91 | 0.00 |
| **Erlen_Hive11_grayscale** | | | | | | | | |
| Mish | single_hive/Erlen_Hive11_grayscale | 1532 | 107 | 100 | 0.93 | 0.94 | 0.94 | 0.00 |
| Leaky | single_hive/Erlen_Hive11_grayscale | 1471 | 73 | 161 | 0.95 | 0.90 | 0.91 | 0.00 |
| **Erlen_Hive11_red70** | | | | | | | | |
| Mish | single_hive/Erlen_Hive11_grayscale | 1487 | 85 | 145 | 0.95 | 0.91 | 0.92 | 0.00 |
| Leaky | single_hive/Erlen_Hive11_grayscale | 1354 | 51 | 278 | 0.96 | 0.83 | 0.85 | 0.00 |
| **Froh14** | | | | | | | | |
| Mish | single_hive/Froh14 | 384 | 38 | 35 | 0.91 | 0.92 | 0.92 | 0.00 |
| Leaky | on_qty_of_train_images/Froh14_20 | 285 | 19 | 134 | 0.94 | 0.68 | 0.72 | 0.00 |
| **Froh23_TreeCavity** | | | | | | | | |
| Mish | single_hive/all_hives_trainP | 140 | 6 | 18 | 0.96 | 0.89 | 0.90 | 0.00 |
| Leaky | single_hive/Froh23_TreeCavity | 121 | 102 | 34 | 0.54 | 0.78 | 0.72 | 0.00 |
| **UnitedQueens** | | | | | | | | |
| Mish | single_hive/all_hives_trainP | 403 | 5 | 5 | 0.99 | 0.99 | 0.99 | 0.00 |
| Leaky | staged_2/Erlen_Hive11_UnitedQueens | 385 | 4 | 23 | 0.99 | 0.94 | 0.95 | 0.00 |