IMPERIAL

BEng Individual Project

Imperial College London

Department of Computing

Tracking the Buzz: Honey Bee Computer Vision Tracking

Author:
Daniel George Wilcox

Supervisor:
Nuri Cingillioglu, PhD

Second Marker:
Nick Powell, PhD

June 13, 2025

Abstract

Understanding honeybee waggle dance communication requires detailed behavioural analysis, but traditional manual tracking methods are labour-intensive for researchers. This project presents ApiCV, a novel computer-vision-assisted annotation system that addresses the unique challenges of markerless honeybee tracking in dense hive environments. Unlike existing solutions that require extensive pre-training or domain-specific annotations, ApiCV employs an innovative ensemble approach inspired by Monte Carlo localization and Kalman filtering, using point clouds rather than individual tracking points, to establish robust consensus-based predictions.

Evaluation on annotated footage from entomologists at the University of Edinburgh shows ApiCV achieving mean euclidean distances of 3.1-5.8 pixels from ground truth annotations, compared to 4.4-10.1 pixels for unmodified contemporary point tracking. By dramatically reducing annotation workload while maintaining accuracy suitable for behavioural research, ApiCV enables scalable analysis of honeybee communication patterns and follower behaviours.

Acknowledgements

Thank you to Dr Nuri Cingillioglu for guiding me through this project, and Dr Ben Glocker for his insight and advice in computer vision. Thank you to Dr Anna Hadjitofi for meeting with me and providing the valuable data to make this project possible. Thank you James, Euan, Pedro, and Sophia for my time at Imperial.

Contents

Chapter 1

Introduction

Honeybees, upon finding a source of food, will return to their hive and often communicate its location with other bees in the form of the "waggle dance": an intricate performance where the finder will dance in a figure-eight while waggling its tail back and forth [1]. Entomologists from the University of Edinburgh, namely Anna Hadjitolfi, the primary beneficiary of this project, have been tracking these bees hoping to answer questions such as:

- *"When and where in the hive do these bees dance?"*
- *"What are the followers doing before, during, and after watching the dance?"*
- *"How is this information, if at all, propagated through the hive?"*

Questions which have so far required difficult, expensive, and time-consuming manual tracking and annotation methods. As a result, entomologists have begun turning to modern computer vision methods to automate both the tracking process and the generation of reliable annotations.

While computer vision has advanced substantially in recent years, and has the potential to make the entomologists' research much easier and quicker, automated tracking and annotation of *honeybees* presents distinct challenges. Some of these are:

- Hives are often crowded and busy, with bees climbing over and under each other and crawling into combs, leading to very dense environments with high occlusion. See figure 1.1.
- Bees look very similar: It is tough for a human to distinguish the difference between two bees, let alone hundreds. For a computer, it is extremely difficult to determine the visual distinction (known as the "pixel personalities") between them.
- Footage is difficult and expensive to obtain. It can cost researchers thousands of pounds to purchase and set up appropriate filming equipment. Although this is also a problem for manual tracking, the lack of footage (particularly *annotated* footage) makes it difficult to train computer vision models. Creating high-quality annotations manually is labour-intensive and requires entomology expertise.
- Even with a sufficient amount of footage, not all hives look the same. An *apis florea* hive, for example, looks very different from an *apis mellifera* hive. This means that even if a model is trained with a large amount of annotated footage, it is very difficult to transfer this knowledge directly to another hive, requiring new annotations for each domain.
- Beyond just the visual difference between species of bee, training data can be sensitive to lighting conditions, scale, framerate, etc. Coupled with the difficulty in obtaining footage, this causes further difficulty in the transferability of existing point-tracking solutions.

Existing solutions have succeeded, notably Bozek et al.'s 2021 paper *Markerless tracking of an entire honey bee colony* [2] and Rozenbaum et al.'s *Machine learning-based bee recognition and tracking for advancing insect behaviour research* [3]. These papers present promising results that entomologists can use to perform automated tracking in place of manual annotations.

Bozek et al.'s paper from 2017 [4] demonstrates a high accuracy for tracking multiple bees at once, but used Amazon Mechanical Turk, a crowdsourcing marketplace, to obtain hundreds

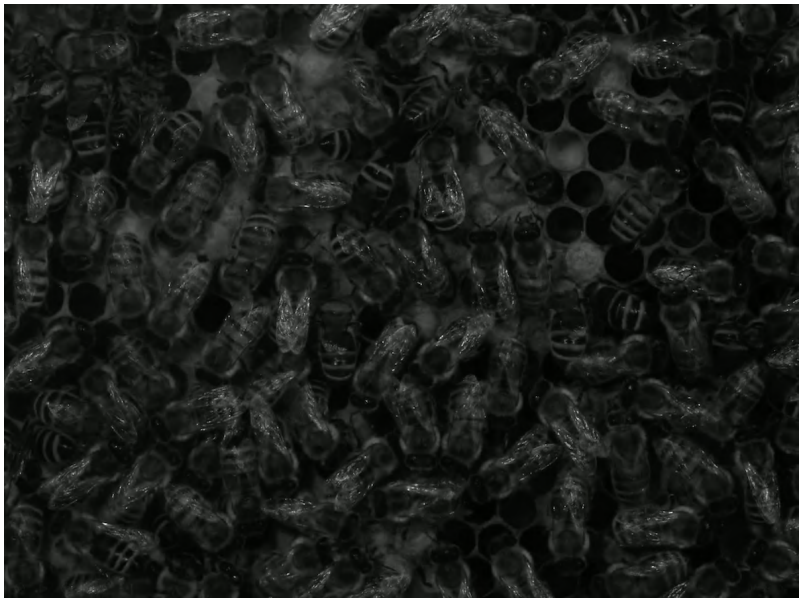


Figure 1.1: Example Image of Dense Honeybee Hive

of thousands of annotations. Kongsilp et al.'s paper [5] presented an effective method to track individual bees using Kalman Filtering, but a lengthy annotation process was still required. Additionally, all of these networks had been individually trained to work on their own hives, and it proved difficult to run the same algorithm on new data without extensive re-annotation. Finally, not all the code for these papers is readily available, and the code that *is* available proved tedious and difficult to configure. This makes many of these solutions impractical to be used by entomologists, whose speciality lies in the behaviour of bees rather than the implementation of machine learning software or the creation of large-scale annotation datasets.

These papers, alongside entomologists at the University of Edinburgh, express a clear demand for versatile bee-tracking software that can both track bees and efficiently generate the annotations needed for reliable analysis. Therefore, the goal of this project is to produce such a software, satisfying the below requirements:

1. **Portable/Transferable**: Able to work in many different domains of bees *without* requiring extensive manual annotation that would be prohibitive for smaller research teams.
2. **Efficient Annotation Generation**: Capable of producing high-quality annotations, dramatically reducing the manual annotation burden.
3. **Functional in Dense Environments**: Effective tracking in dense and sometimes heavily occluded environments.
4. **Accurate**: Able to differentiate and correctly match bees across frames, potentially through semi-supervised tracking methods.
5. **Usable**: Can be used effectively by entomologists to assist in their research, without requiring knowledge of machine learning architectures or excessive time creating annotation datasets.

This project presents ApiCV, a novel approach to object tracking and annotation by considering *clouds* of points instead of individuals selections, and establishing a consensus. ApiCV separates itself from contemporary computer vision and point tracking solutions by using statistical uncertainty minimisation techniques inspired by the field of robotics.

With semi-supervised validations and robust predictions, the amount of work required to produce annotations with ApiCV is greatly reduced compared to annotating entirely manually. This enables entomologists to produce reliable annotations across varying domains while providing accurate tracking capabilities.

The rest of this report is structured as follows.

- **Background**: Containing relevant background knowledge, a literature review, and terminology used throughout the report.

- **ApiCV**: A broad overview of the downstream application, its purpose, and its use.
- **Tracking Algorithm**: The method(s) by which the tracking and annotation portion of the algorithm is implemented.
- **Evaluation**: Quantitative and qualitative evaluations demonstrating the improvements across different experiments.
- **Conclusion**: Concluding remarks on the strengths and limitations of the project, as well as future work that could be done.
- **References**: Bibliography and other sources accessed during background research and project development.
- **Declarations and Ethical Considerations**

Chapter 2

Background

2.1 Honeybees and Project Beneficiaries

Entomology Entomology is the scientific study of insects, their biology, ecology, and relationship to the environment. Melittology is a branch of entomology specifically concerning the study of bees.

Project Beneficiaries The data upon which this project focuses comes from entomology researchers at the University of Edinburgh, particularly Anna Hadjitofi. These researchers have provided a mix of annotated and unannotated footage of various honeybee hives they have collected. They may be referred to throughout the paper as the "entomologists", "project beneficiaries," or "researchers."

Some of the data they have provided is not publicly available, but has been approved to be shared in chunks for the purposes of this project and report.

Waggle Dance and Followers The honey bee waggle dance is a form of communication where, after finding a source of food, a honeybee will perform a figure-eight dance while waggling its thorax. The waggle direction indicates the direction of food relative to the sun, and the total duration indicates how far away the food can be found. This behaviour was first decoded by Karl von Frisch in 1967 in *The Dance Language and Orientation of Bees*.

The motivation behind tracking and annotating honeybees is to assess the behaviour of the *followers*. The followers are those bees that are watching the dancer during its performance. Followers may then take different actions such as leaving to find the source of food or even propagating the information throughout the hive. These are the questions entomologists wish to answer.

Thorax and Abdomen The thorax and abdomen are two commonly annotated locations on a honeybee in the ground truth data. It is useful to know the difference between them when reviewing the evaluation sections in 5 to understand the positioning of the annotations. Figure 2.3a displays some of the ground truth annotations provided by the entomologists, where the pink arrow points to what is labelled as the abdomen (the back / rear of the bee) and the green arrow points to what is labelled as the thorax (the head of the bee).

Hives / Domains / Datasets The data used for experimentation and evaluation comes from the entomologists from the University of Edinburgh, and consists of multiple different hives (or "domains" as they are referred to throughout the report).

The *annotated* datasets are:

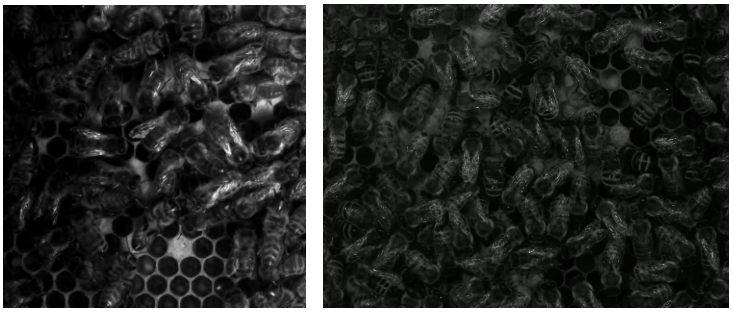
- **ROUND_DANCE**: A 25fps video of a bee performing the "round dance" (a different type of dance from the waggle dance). Annotations have been made on the dancer's abdomen and thorax. Displayed in figure 2.1a.
- **RECORDING-2023**: A 30fps video of a few follower bees, named n32, n33, and n34, watching another bee perform its dance. Annotations have been made on the thoraxes of these bees. Displayed in figure 2.1b.

The annotations for these videos are considered to be the ground truth data against which evaluation was performed.

The *unannotated* resources are:

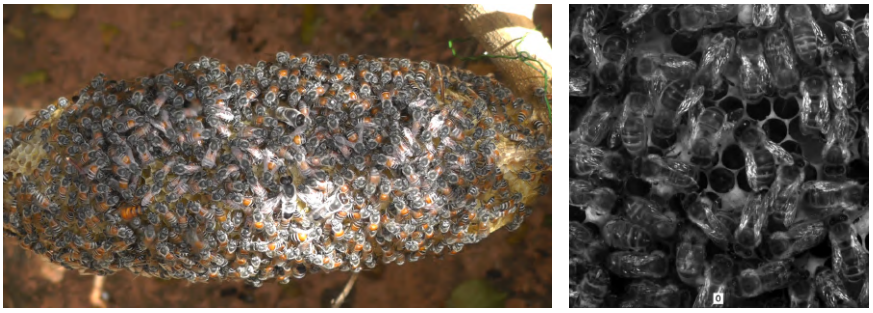
- **DANCE_15**: A 15fps video of a bee performing the waggle dance, of which one of the followers was tracked. Displayed in figure 2.2b.
- **OUTSIDE_FLOREA**: A 15fps full RGB video of an *apis florea* hive, of which a random bee was selected for tracking. Displayed in figure 2.2a.

These are videos that do not have ground truth data against which to assess. Validations (detailed further in 4.5) were produced for situations evaluations that didn't require ground truth data. Precise validation values used during evaluation can be found in appendix figures A.5, A.7, A.9, and A.11.



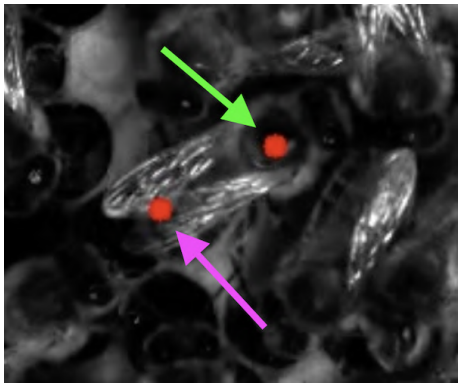
(a) ROUND_DANCE (b) RECORDING-2023

Figure 2.1: Sample Frames of the Annotated Footage

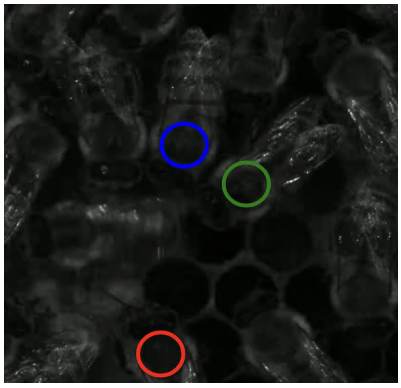


(a) OUTSIDE_FLOREA (b) DANCE_15

Figure 2.2: Sample Frames of the Unannotated Footage



(a) Abdomen (Pink Arrow) and Thorax (Green Arrow)



(b) Example of Multi-Object Tracking in RECORDING-2023

2.2 Machine Learning and Computer Vision

Machine Learning Machine Learning is a subset of artificial intelligence that enables computers to learn and make predictions or decisions from data without being explicitly programmed for each specific task. It involves developing algorithms that can identify patterns in data and use these patterns to make informed predictions on new, unseen data.

Computer Vision Computer Vision is a field of artificial intelligence that enables computers to interpret and understand visual information from the world, including images and videos. It involves developing algorithms and techniques to extract meaningful information from visual data, such as object detection, recognition, and tracking.

Deep Learning and Convolutional Neural Networks Deep Learning is a type of machine learning that uses multiple layers in its neural network to learn abstract feature representations from raw data. Convolutional Neural Networks (CNNs) are a type of deep learning network designed to process grid-like data (such as images), and are particularly effective for computer vision tasks due to their ability to detect spatial patterns and features at multiple scales.

Optical Flow Analysis Optical Flow Analysis is a computer vision technique that estimates the *motion* of objects between consecutive frames in a video sequence. It calculates the apparent motion of brightness intensity patterns, creating motion fields that can track how pixels move over time.

The fundamental principle is captured by the optical flow constraint equation:

$$I_x u + I_y v + I_t = 0 \tag{2.1}$$

where

- I_x and I_y are spatial brightness intensity gradients, representing (x, y) pixel intensities
- I_t is the temporal brightness intensity gradient, representing intensity changes over time
- (u, v) represents the optical flow *velocity* vector at each pixel

For the purposes of this project, it is sufficient to understand that optical flow analysis is the basis upon which object tracking algorithms are built.

Transfer Learning Transfer Learning allows pre-trained models to be adapted for new tasks with minimal additional training. This could be particularly valuable in domains like bee tracking where annotated data is scarce, as models trained on general visual tasks can be fine-tuned for specific applications.

2.3 Object Tracking

Object Tracking Object Tracking is the process of locating and following objects of interest through a sequence of video frames. It typically involves two main phases: detection (identifying objects in the current frame) and association (linking detections across frames to maintain consistent identities).

Multi-Object Tracking Multi-Object Tracking (MOT) extends single-object tracking to handle multiple targets simultaneously, requiring solutions for data association problems when objects appear, disappear, or occlude each other. While this project typically focuses on tracking a single subject (bee) at a time, all clouds are entirely independent of each other so multiple bees can be tracked at once.

An example can be seen in figure 2.3b, tracking bees n32, n33, and n34 at the same time in RECORDING-2023. This is detailed during the evaluations in section 5.1.

Point Tracking Point tracking is a type of object tracking that focuses on following specific feature points or "query points" rather than entire object boundaries. Instead of tracking bounding boxes, point tracking maintains the trajectories of individual pixel locations or sparse sets of points that represent salient features of the target object. This approach is useful when objects are small, computational efficiency is critical, or only specific parts of an object need to be monitored.

Kalman Filtering Kalman Filtering is a recursive algorithm that estimates the state of a dynamic system from noisy observations. In tracking applications, it predicts object positions based on motion models and updates these predictions when new measurements are available.

For the purposes of this project, the filter has been simplified to:

- A **Prediction Phase**: $x_k = F_k x_{k-1} + w_k$
- An **Observation Phase**: $z_k = H_k x_k$

where

$$\begin{aligned} x_k &= \text{true state at time } k \\ F_k &= \text{prediction matrix} \\ w_k &= \text{process noise} \\ H_k &= \text{observation matrix} \\ z_k &= \text{observed value of true state} \end{aligned}$$

Generally speaking, a prediction is made at time k by processing previous state x_{k-1} with the assumption of some noise w_k. An observed value z_k is then made by processing the prediction.

Markerless and Markered Tracking Markerless tracking relies solely on natural visual features, which is preferable for biological studies to avoid disturbing natural behaviour but presents greater technical challenges. Markered tracking uses artificial markers (like QR codes or coloured tags) attached to objects for easier identification.

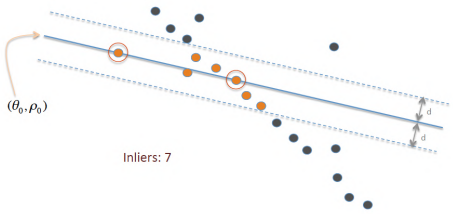
2.4 Point Clouds and Statistics

Point Clouds Point Clouds are collections of data points in space represented as (x, y) coordinates in 2D. In computer vision, point clouds can represent object shapes, feature locations, or tracking points distributed across a region of interest.

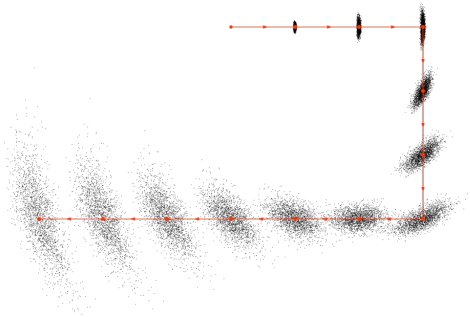
Ensemble Methods Ensemble Methods in the context of point tracking involve using multiple points to make collective decisions, similar to ensemble learning in machine learning where multiple models vote on predictions. This project uses consensus among cloud points to improve tracking robustness compared to single-point methods.

Consensus Consensus in point cloud processing refers to the agreement among multiple data points or measurements to determine a reliable estimate or decision. In the context of point tracking and ensemble methods, consensus involves aggregating information from multiple points in the cloud to identify patterns, filter out outliers, or make robust predictions. This approach leverages the collective "vote" of numerous points to achieve greater accuracy and reliability than single-point methods.

RANSAC RANSAC (Random Sample Consensus) is an iterative algorithm for robust parameter estimation in the presence of outliers. It randomly samples **subsets** of data points to fit models and identifies the subset with the most inliers (points that fit the model within a tolerance). Figure 2.3c demonstrates how parameters θ_0 and ρ_0, the parameters for a linear function, are estimated by assessing inliers within some threshold d.



(c) Example of RANSAC Used for Linear Fitting [6]



(d) Example of Monte Carlo Localization [7]

Clustering and Outlier Detection Clustering is the process of assigning objects in a set into *groups* whose attributes are most similar to each other. Clustering is frequently used for spatial data, but various modifications exist depending on the end goal. An example can be seen in figure 2.3.

Algorithms like DBSCAN (Density-Based Spatial Clustering of Applications with Noise) group points based on density, automatically identifying outliers as points that do not belong to any cluster. This is scale-invariant when parameters are adjusted relative to the point cloud radius r.

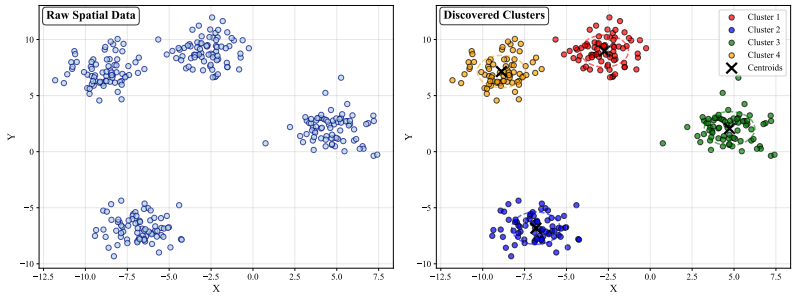


Figure 2.3: 2D Clustering Example

Monte Carlo Localization A technique used by robots to localize their position in space using a point cloud filter. Given a map of the environment, the algorithm estimates the position and orientation of a robot as it moves and senses the environment.

Monte Carlo Localization is not used directly in this project, but inspires other aspects such as the statistical approach to tracking. Figure 2.3d demonstrates how the uncertainty of an object's position increases over time as it moves through its environment.

Variance and Covariance Variance and covariance are statistical measures that quantify the spread and relationship between random variables.

Variance, denoted as $\text{var}(X)$ or σ^2, measures how much a random variable X deviates from its expected value μ. It is defined as:

$$\text{var}(X) = E[(X - \mu)^2] = E[X^2] - (E[X])^2$$

A higher variance indicates that the values are more spread out from the mean, while a lower variance indicates that values cluster more closely around the mean.

Covariance, denoted as $\text{cov}(X, Y)$, measures the degree to which two random variables X and Y change together. It is defined as:

$$\text{cov}(X, Y) = E[(X - \mu_X)(Y - \mu_Y)] = E[XY] - E[X]E[Y]$$

A positive covariance indicates that the variables tend to increase together, a negative covariance indicates that one variable tends to decrease as the other increases, and a covariance of zero indicates no linear relationship between the variables.

Gaussian Distributions A Gaussian distribution is a type of continuous probability distribution that accepts a mean μ and variance σ^2 to distribute values.

A 2D Gaussian distribution (or more generally, a multivariate Gaussian distribution) is a Gaussian distribution across two dimensions, which can measure the correlation between different random variables X and Y. 2D Gaussians accept a 2D mean μ and a 2×2 covariance matrix Σ where

$$\Sigma = \begin{bmatrix} \text{var}(X) & \text{cov}(X,Y) \\ \text{cov}(X,Y) & \text{var}(Y) \end{bmatrix}$$

Examples of 1D and 2D Gaussian distributions can be seen in figure 2.4.

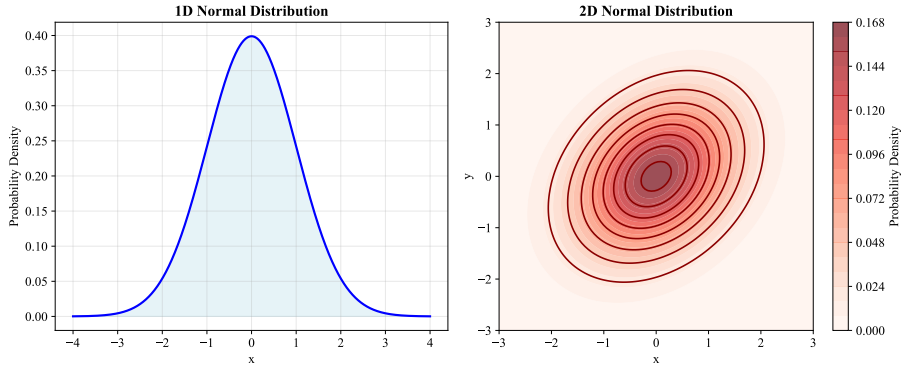


Figure 2.4: Normal Distributions Example

2.5 Literature Review

Initial research was performed to understand the current state of the art of point tracking, both regarding honeybees and more generally.

Honeybee Tracking: Problem Context and Requirements

Modern research on honeybees has focused on the *followers* of the waggle dance, the honeybees who watch the dancer and decode its meaning [1]. Understanding follower behaviour has proved particularly challenging due to the complex dynamics of the hive environment, where multiple bees may follow a single dancer, and potentially propagate information throughout the hive.

Recent studies such as Anna Hadjitofi and Barbara Webb's *Dynamic antennal positioning allows honeybee followers to decode the dance* [1] and Smith et al.'s *Long-term tracking and quantification of individual behavior in bumble bee colonies* [8] emphasize tracking specific anatomical features of followers, such as thorax position, antennae base, and eye orientation, to better understand how they interpret and respond to the dance information. Numerous parties, particularly the entomologists from the University of Edinburgh, have expressed interest in a semi-automated tracking process, such that entomologists do not have to spend as much time dealing with raw video footage, and can instead focus their efforts on studying the bees behaviour.

Tracking Methodologies and Approaches

Markered versus Markerless Tracking

To perform *markered* tracking with bees (defined in section 2.3), researchers must physically attach markers to individual bees, which is both time-consuming and potentially disruptive to natural bee behavior. Therefore, markerless tracking approaches have been implemented to overcome these limitations.

A notable example is Bozek et al.'s (2021) [4] colony-wide tracking system, which attempted to track bees without markers by using deep learning and ellipsoid-based annotations. However, their system faced significant challenges, including difficult setup procedures that hindered adoption by entomologists, limited trajectory tracking duration of only 5 minutes, and compatibility issues with modern deep learning frameworks.

While their approach demonstrated the potential of markerless tracking through detailed visual feature learning, the technical implementation barriers and short tracking duration highlighted the ongoing challenges in developing practical, scalable solutions for automated bee tracking without markers.

Identification and Frame-to-Frame Tracking

After appropriate identification of individual bees (which can be done by iteration over available frames), the more difficult requirement of linking and tracking across frames is presented. Various approaches have been developed to handle the occlusion problems inherent in dense hive environments.

Some solutions have attempted to solve occlusion by labelling bees as "half-bees" or "full-bees" [4], where others have simply set a cut-off of 80%, identifying a bee if at least that much of its body is visible [5]. Bozek et al's 2021 paper *Markerless tracking of an entire honey bee colony* [4] introduced their own loss function when training a Convolutional Neural Network, while Kongsilp et al. [5] used Kalman Filtering and the Hungarian Algorithm to focus on a single bee. Both papers incorporated the current trajectory of the bee to predict its next position.

In terms of annotation strategies, Kongsilp et al. [5] decided to use polygons to represent the entire bee when labelling (appendix figure A.1). The use of polygons as opposed to single points to represent bees is unique across contemporary methods.

Hybrid and Alternative Approaches

Beyond the traditional markered and markerless methods, researchers have explored innovative hybrid approaches. Smith et al. [8] introduced a novel method using both tag-based *and* pose-estimation techniques (implemented with deep learning) to perform long-term (48h) tracking of multiple bumblebee species, bridging the gap between the two primary methodologies. Rodriguez et al. similarly used a mix of both marked and unmarked bees in *Automated Video Monitoring of Unmarked and Marked Honey Bees at the Hive Entrance* [9].

For less densely-occupied environments, Rozenbaum et al. used YOLO (You Only Look Once), a fast CNN algorithm for image recognition, in *"Machine learning-based bee recognition and tracking for advancing insect behavior research"* [3]. An important method introduced in this paper is their automatic image processing system, which allowed Rozenbaum et al. to produce 4000 annotated images very quickly and easily. It is worth noting, however, that their model was trained on a single bee in a maze, rather than a large collection of bees naturally in their hive.

Contemporary Technologies and Solutions

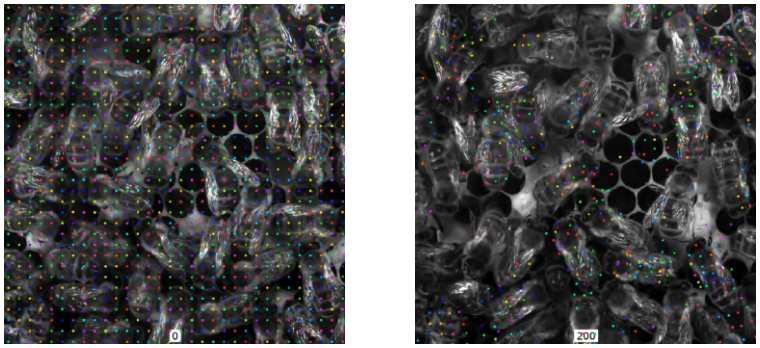
DeepLabCut for Pose Estimation

DeepLabCut is an open source toolbox for markerless pose estimation based on transfer learning with deep neural networks (Mathis et al., [10], https://www.mackenziemathislab.org/deeplabcut). DeepLabCut represents an advancement in markerless pose estimation, particularly relevant for tracking specific anatomical points on bees. The framework excels at precise point detection for tracking thorax position, antennae base, and directional indicators, which are crucial reference points in current research.

The entomologists, however, have noted challenges when applying DeepLabCut to bee tracking in *hives* due to the noise and occlusion of the hive environments. The primary limitations include substantial annotation requirements and difficulty handling setup variations across different hive environments. These challenges are pronounced in crowded scenarios and variable lighting conditions, common in hive observations, where current automatic methods have proven insufficient for reliable tracking.

TAPIR: Advanced Point Tracking

TAPIR (Tracking Any Point with per-frame Initialization and temporal Refinement) is an advanced point-tracking model developed by Google DeepMind and the University of Oxford, designed to accurately track any desired point on a physical surface within video sequences [11] https://deepmind-tapir.github.io/. The model operates in two main stages: a matching stage that



(a) Frame 0 (b) Frame 200

Figure 2.5: Rainbow Visualization on *Apis Mellifera*

identifies candidate point matches across frames, and a refinement stage that enhances trajectory and feature accuracy through local correlations.

The system's self-supervised learning approach eliminates the need for extensive manual annotations, a significant advantage over traditional tracking methods that require hundreds of thousands of labelled examples.

Upon testing TAPIR with bees, minimal annotation was required. The Rainbow Visualization prefab (which initialises a grid of points across the entire video) demonstrated impressive tracking capabilities, but many points lost their subjects and became attached to other bees. Examples of the Rainbow Visualization prefab for a zoomed in video of the *apis mellifera* dancing are seen in figure 2.5 and appendix figure A.3. Accuracy improved upon increasing the total number of points scattered.

For implementation in bee tracking scenarios, TAPIR offers several key advantages. Its sparse tracking mode provides quick processing times for initial analysis, and requires no initial annotation meaning it can be used for more robust downstream applications.

Current Limitations and Implementation Challenges

Data Collection and Annotation Bottlenecks

Automated honeybee tracking faces significant challenges in data collection and annotation. Ground truth acquisition is difficult due to dense, occluded hive environments. While large-scale annotation efforts exist (Smith et al.'s 375,000 labelled bees [8]), they remain costly, time-consuming, and create bottlenecks for resource-limited researchers.

Environmental Variability and Domain Transfer

Environmental variability—lighting, camera quality, hive type—severely limits cross-domain generalizability. Dense object tracking under these conditions proves challenging, with models like TAPIR frequently failing due to occlusion and false negatives.

Testing Bozek et al.'s solution [2] on the footage provided by the entomologists demonstrates poor transferability. After resolving version control issues, their pre-trained model produced incoherent results (appendix figure A.2): honeycombs were labelled as bees and arrows were randomly positioned. Re-training would require extensive annotation, highlighting the need for transferable, minimally-annotated tracking methods.

2.6 Project Terminology

Various terminology is used throughout this report to describe aspects of the software. The most common and important of these are:

Query Point A query point is the specific point upon the subject (bee) that the *user* wishes to track. A point cloud is formed around the query point. The query points points are the centres of the circles in figure 2.7a, and labelled q in figure 2.6.

Cloud Point Cloud points are the points generated around the query point q. These are the multicoloured points in figure 2.7b, and labelled c_i in figure 2.6.

Point Cloud Movement Point cloud movement is used to describe the movement of cloud points after some time δt. These points are moved by some optical flow analysis or point tracking software. This can be seen in the translation and rotation of the coloured points in figure 2.6, where initial cloud points are labelled c_i and clouds points that have been moved are labelled c_i'.

Cloud Generation/Reconstruction Cloud generation is the process of generating cloud points to be moved and analysed. Figure 2.7a demonstrates how users may select a query point, and figure 2.7b displays the cloud that is generated.

Cloud reconstruction is the process of rebuilding a cloud after it has become deformed due to point cloud movement. Cloud reconstruction techniques are discussed in section 4.4.

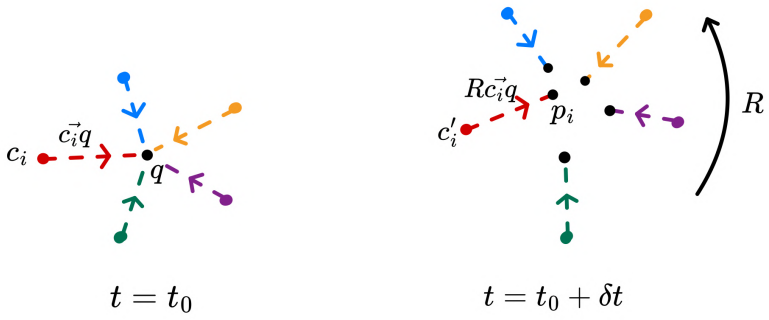
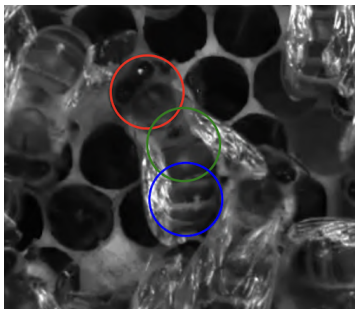
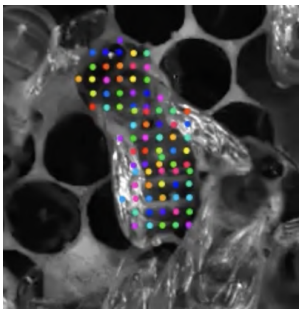


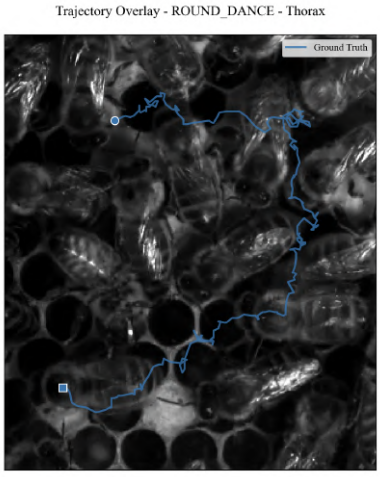
Figure 2.6: Terminology Diagram



(a) Query Point Selection Circles



(b) Cloud Points



(c) Example of Tracks for Ground Truth Annotations

Figure 2.7: Point and Tracks Diagrams

Inliers and Outliers When clustering points in 2D, some points are considered to be *inliers*, which are points that fit well within identified clusters and follow the expected data patterns. In contrast, *outliers* are points that deviate significantly from cluster structures, typically by being isolated.

Tracks Tracks is a term used to describe the 2D movement of a point over time. It can be used interchangeably with "annotations," as they both represent movement over time. An example can be seen in figure 2.7c.

Displacement Vector The displacement vector of a cloud point c_i is the initial vector from a cloud point to its associated query point q before point cloud movement. This is established during cloud generation or reconstruction. These vectors are labelled $\vec{c_iq}$ in figure 2.6.

The *original* displacement vectors (a term frequently used in 4.4 are the displacement vectors calculated during cloud generation, and are not overwritten after cloud reconstruction.

Predictor A predictor is the combination of a moved cloud point c_i' and its rotated displacement vector $R\vec{c_iq}$. Specifically, it is labelled p_i in figure 2.6 and can be expressed as

$$p_i = c_i' + R\vec{c_iq}$$

.

Query Point Prediction After determining all of the individual predictors p_i, a Query Point Prediction (or Predicted Query Point) p is the final prediction of the query point for a cloud. The details of how this is calculated is outlined in section 4.3.

Deformity Deformity is the primary metric used to determine the reliability of a point cloud, as well as the most likely rotation of the cloud after some time δt. It is defined as the determinant of the covariance matrix of the *predictors* of a cloud.

For a 2D cloud of predictors X, deformity is calculated as:

$$\begin{aligned}\text{Deformity}(X) = D(X) &= \det(\Sigma) \\ &= \det\left(\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T\right)\end{aligned}$$

From an intuitive perspective, it quantifies the "agreement" amongst points. Should all points in a distribution be in the same position, the deformity would be 0 (as there is no variance amongst them).

Validations Validations are times during processing in which the user may be requested to manually annotate the subject's position. The user will have been given the predicted query point p, and asked to provide a validated query point v.

Smoothing / Smoothed Tracks Smoothing, detailed in full in section 4.6, is used to generate less jittery tracks and avoid "jumps" between validated and predicted points.

Segments The tracking process is split apart into segments, details of which are discussed in section 4.2. Briefly, segments are 1 second chunks of the video being processed, of which the tracking pipeline iterates through.

For example, for a 50fps video, 1 segment could be frames 0 - 49 or 150 - 199.

Notation Summary

- q = the original query point selected by a user
- $C_k = \{c_1, ..., c_n\}$: The cloud points at the end of segment k
- $P_k = \{p_1, ..., p_n\}$: The predictors of the point cloud, where each $p_i = c_i + R\vec{c_iq}$
- p: The predicted query point as mentioned in "Query Point Prediction"
- v: The query point validated by the user as mentioned in "Validations"

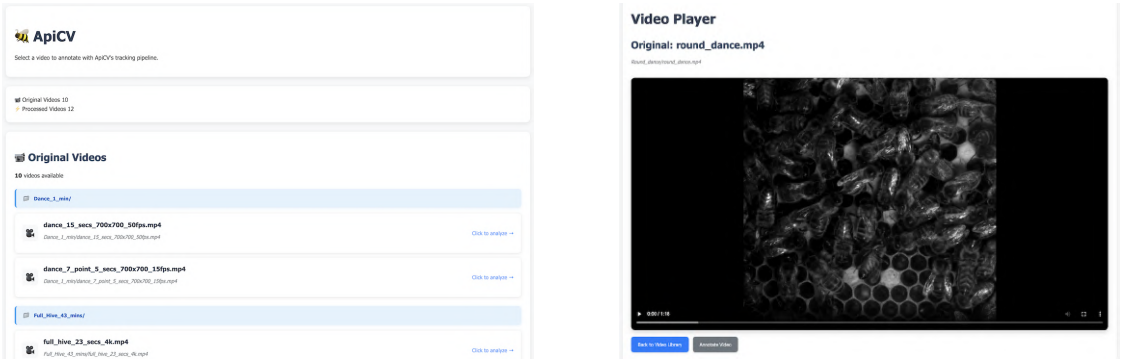
Chapter 3

ApiCV

3.1 Computer-Vision-Assisted Annotation Software

To address the problems introduced in section 1, this project presents ApiCV, a computer-vision-assisted annotation software that allows users to select and track a bee throughout a provided video. The software integrates human expertise with automated computer vision algorithms to achieve robust tracking performance.

The process begins by first selecting the video the user would like to analyse, seen in figure 3.1a. Afterwards, the user may select "Annotate Video", seen in figure 3.1b, to move to the "frame analysis" page beginning with figure 3.2a



(a) Video Selection (b) Annotate Video Selection

Figure 3.1: ApiCV Opening Pages

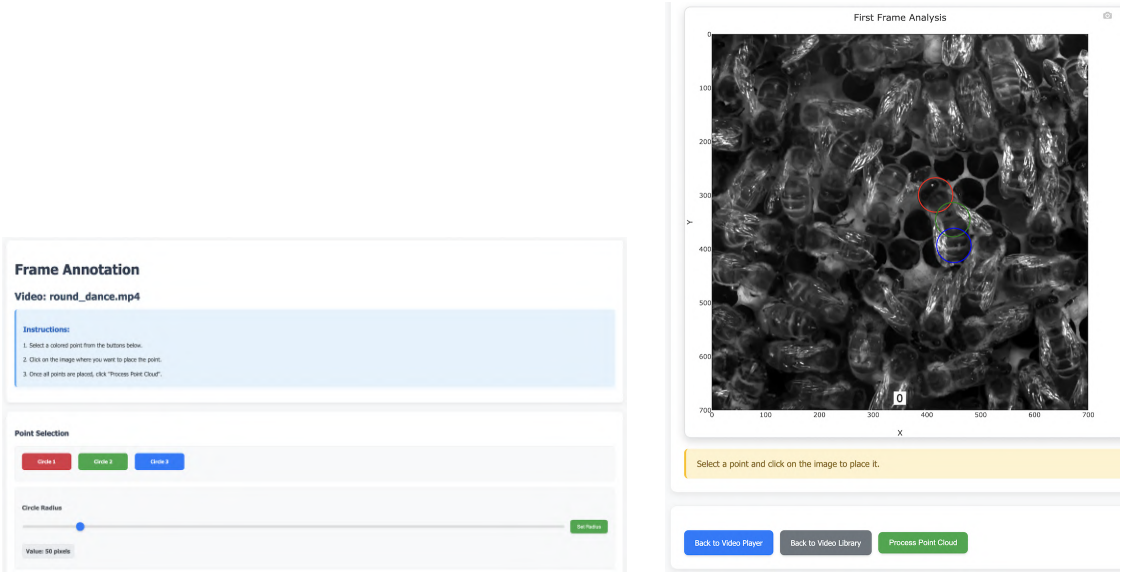
The user is presented with the first frame of the video they have selected, as can bee seen in figure 3.2b. They can select points to track and adjust their positions by either clicking on the image itself or importing points if they have been saved in advance. This allows for both manual initial annotation and the reuse of previously established tracking points.

The points the user wishes to track are referred to as "query points", as defined in 2.6. They are initially displayed on the screen as circles, representing the centre of the query point q and the radius of the desired point cloud r. The visual representation provides immediate feedback on the selected tracking region, allowing users to assess whether the initial selection captures the target bee's features.

The user is also able to perform parameter tuning and component selection, as can be seen in figures 3.6 and 3.7, detailed later in section 3.1. After parameter tuning and point selection, the user may begin processing by clicking "Process Point Cloud". This will start the internal tracking process detailed in Section 4.

Feedback and Validation

Tracking is done in segments (as mentioned in section 2.6 and further detailed in section 4.2), that may or may not request human validation, depending on the program's confidence.



(a) Frame Analysis Opening Section

(b) Query Point Placement Graph

Figure 3.2: Point Selection and Query Point Placement

If validation is required due to high levels of deformity or outliers, the user will be prompted to *validate* the prediction, as can be seen in figure 3.3. The user may choose which point to adjust and modify their position in real-time.

Validation may be required for a variety of reasons, such as bees moving at high speeds, bees performing complex behavioural patterns such as waggle dances, or temporary occlusion by other bees. The validation system serves as a recovery method to assist with long-term tracking, and is shown in evaluation section 5.2 to be essential for ApiCV's success.

Feedback is provided to the user in the form of logs, timeline frames, annotations, and past validations. These can be seen in figures 3.4 and 3.5. Logs and timeline frames keep the user updated on the current state of processing, past validations may be saved for later (should the user wish to try different parameters), and annotations are the end product of ApiCV. Alongside these forms of feedback, the graph is updated at the end of every segment to inform the user of the current tracking progress.

After validation / segment processing, tracking is repeated from the new starting point and continues until the requested video duration is complete. This iterative approach is outlined in section 4.1 and ensures that corrections propagate forward through subsequent tracking segments.

Parameter Tuning

Alongside immediate processing, the user is introduced to a range of parameters, as can be seen in figure 3.6, which may need to be tuned to fit the video in question. These parameters control key aspects of the tracking algorithm and may be adjusted based on video characteristics such as resolution, bee size, movement patterns, and environmental conditions.

- **Video Processing Duration**: How many seconds (from the beginning of the video) to process
- **DBSCAN (Clustering) Epsilon** ϵ: Clustering helps determine confidence in the query point predictions by predicting outliers. ϵ is used to describe how dense a cluster should be. More in section 4.3.
- **Deformity Delta** δ: Used to calculate deformity ratio. Because deformity has no upper bound, a manual input is required (but the default of 0.01 is typically sufficient). More in section 4.5.
- **Smoothing Alpha** α: Smoothing is used to smooth tracks if there are any poor predictions. More in section 4.6.

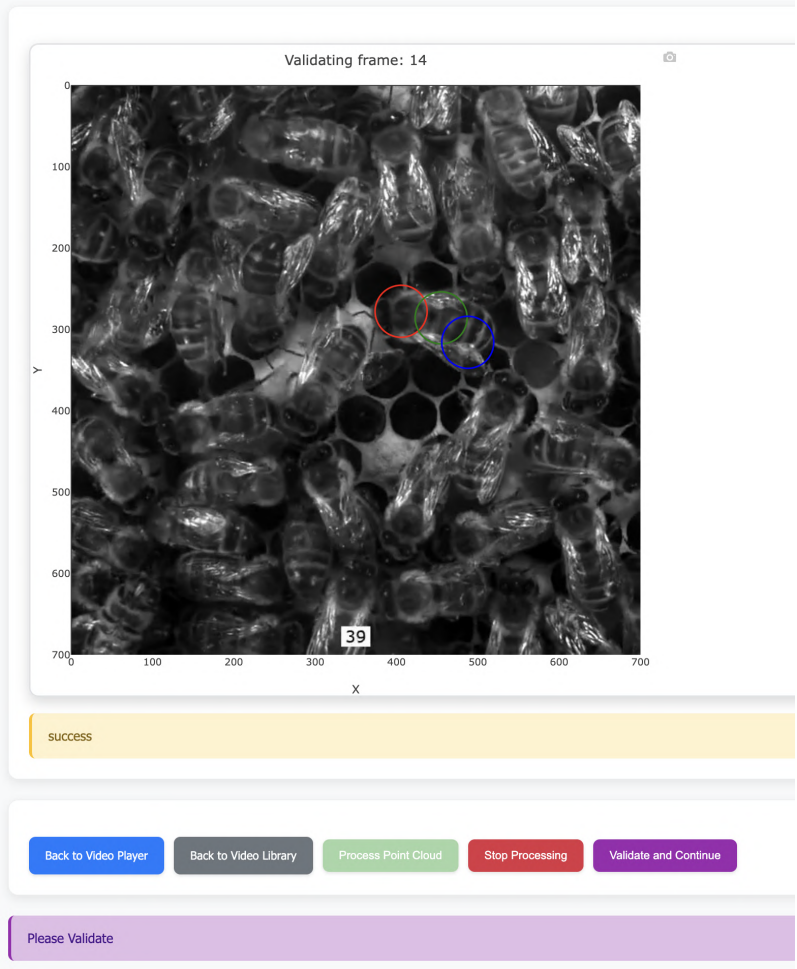


Figure 3.3: Validation Request

ϵ and δ are both multiplied by either the radius r or r^4 where necessary, to maintain scale proportionality (details discussed in sections 4.3 and 4.5). Despite this, further manual tuning may be required for optimal performance.

(**Note**: The system assumes constant scale throughout processing, meaning it assumes the size of an individual bee does not change much throughout the video.)

Software Architecture

The software architecture employs a client-server model to separate user interface concerns from computational processing. Figure 3.8 displays a diagram of user interactions and server communications.

Frontend The frontend software is written with EJS (Embedded JavaScript, a variation of HTML for dynamic web pages) and JavaScript, and serves pages with Express as a locally hosted server. This provides a portable web-based interface and easy setup for entomologists.

Backend The backend software runs on a Flask server, providing the frontend with designated endpoints for processing requests. The Flask framework provides the necessary infrastructure for executing computer vision algorithms and managing tracking state across video segments.

Communication Communication between the front- and back-ends is done via SocketIO, which was necessary to enable bidirectional communication during processing and validation phases. This real-time communication protocol ensures that user interactions and algorithm outputs are synchronized throughout the annotation workflow, and provides immediate feedback during operation.

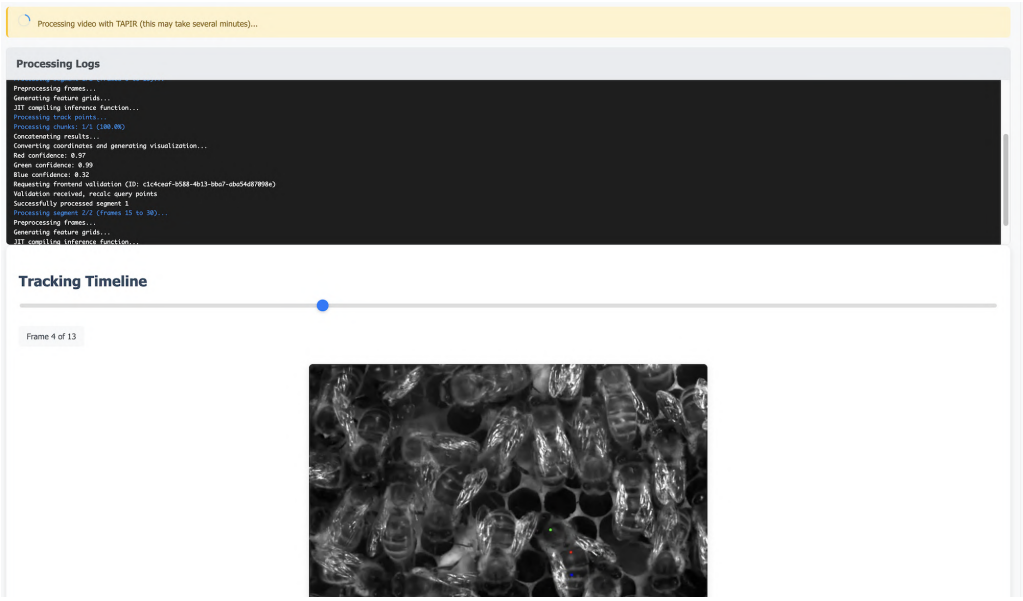


Figure 3.4: Logs and Timeline Feedback

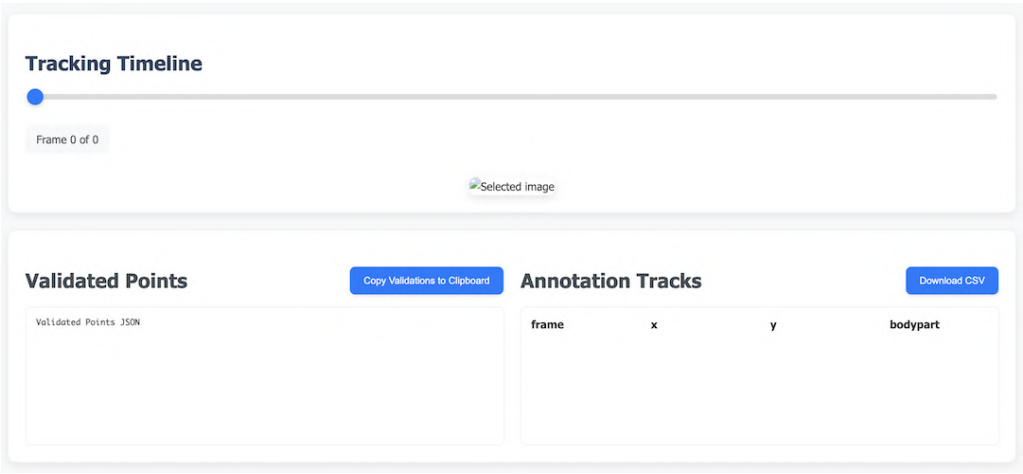


Figure 3.5: Annotations and Past Validations

Since the project requires handling potentially very large (2GB+) videos, care was taken to write processed segments to the disk and clearing memory where possible after use.

Summary

ApiCV provides a comprehensive computer-vision-assisted annotation framework that combines automated tracking algorithms with human oversight to achieve robust bee tracking in video footage. The software features an intuitive web-based interface that guides users through video selection, query point placement, parameter configuration, and interactive validation. Key components include real-time feedback mechanisms, segmented processing with confidence-based validation requests, and configurable parameters for clustering, deformity detection, and track smoothing.

The success of ApiCV's tracking performance relies on its underlying algorithmic and statistical approach, which addresses the inherent challenges of tracking subjects in complex, dynamic environments through ensemble-based methods.

Figure 3.6: Parameter Configuration

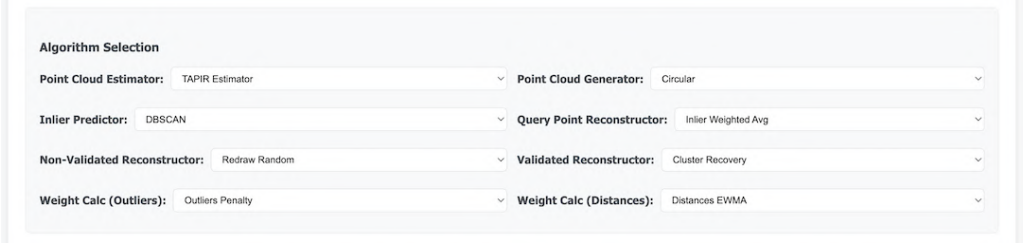


Figure 3.7: Component Selection

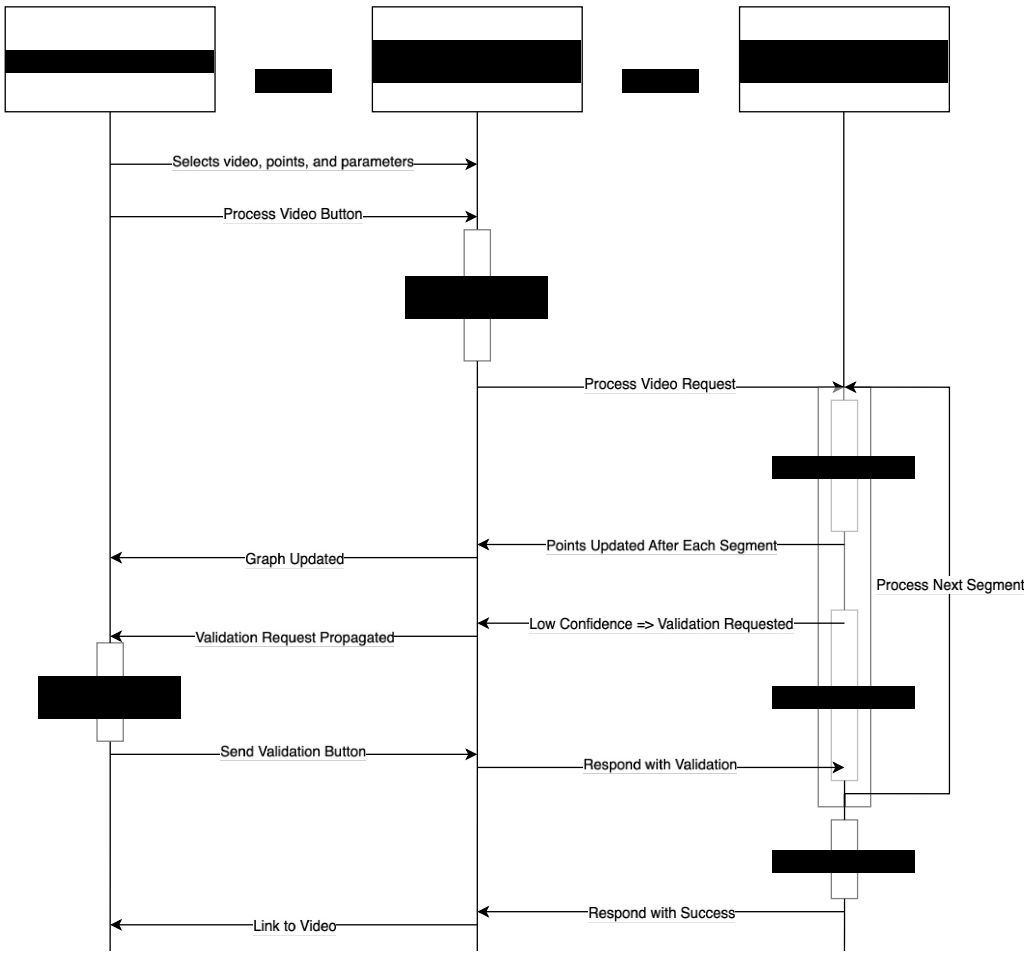


Figure 3.8: Simplified Diagram of Software Architecture and Communication

Chapter 4

Tracking Algorithm

ApiCV presents a novel approach to point tracking by analysing and processing the movement of an entire cloud of points, establishing a consensus, and improving robustness by guiding the cloud's movement. This "ensemble" approach is inspired by a mix of Monte Carlo Localization (defined in 2.4), commonly used in robotics applications.

This approach was prompted by frequent observations of noise, outliers, and occlusion when applying point tracking algorithms to honeybee hives. While individual points exhibited poor tracking of their subjects, groups of points tended to move in the same general direction.

Experiments were performed to determine the best methods of processing a point cloud. The result is a semi-supervised pipeline that can predict the movement of a selected query point with minimal human validation. The implementation of this consensus-driven approach relies on adapting traditional Kalman Filtering principles to operate on point-cloud data, making modifications to both the prediction and update phases of the standard algorithm.

4.1 Kalman Filtering on Point Clouds

Kalman Filtering (definition in 2.3) consists of:

1. A **Prediction Phase**: Producing estimates of the state variables (in this case the location of the query point q as a function of the point cloud) and

2. An **Observation Phase**: Update the weights of the estimation using the next measurement.

The key innovation lies in treating each point within the cloud as an individual predictor (as discussed in section 2.6) that contributes to the overall state estimation.

Prediction Phase Fig 4.1 demonstrates how each cloud point c_i can be considered a *predictor*: c_i represents a cloud point at the beginning of a segment, q represents the known / validated query point at initial time t_0, and $\vec{c_iq}$ represents the displacement vector between c_i and q. After time δt, the cloud point has been moved to position c_i', and the subject may have rotated by some matrix R, creating prediction p_i such that

$$c_i' + R\vec{c_iq} = p_i$$

These predictors are then combined (techniques of which are discussed in section 4.3) to establish the *query point prediction*.

Update Phase Within a single cloud, each cloud point has an associated *weight* w_i, initially uniform across the cloud, demonstrating their reliability as predictors. Some points may have landed in better positions than others, and are therefore allocated a higher weight for future predictions.

The weight update of a predictor is based on

1. whether or not the predictor is an outlier and

2. its euclidean distance to the validated/accepted query point position

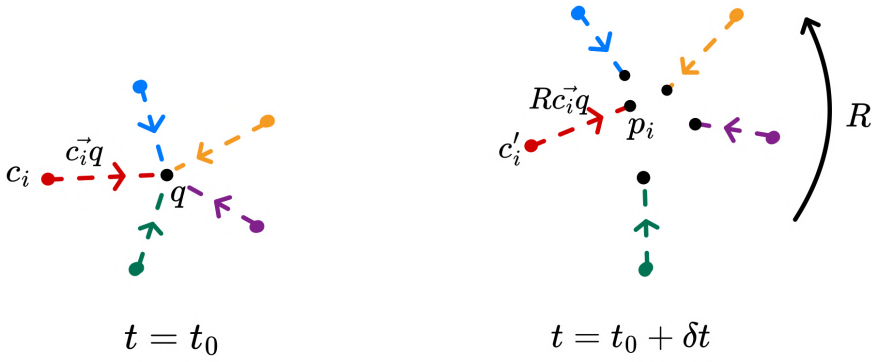


Figure 4.1: Diagram of predictors.

During instances of low confidence, the user is asked for validation and the weight w_i for a predictor p_i is updated (detailed in section 4.5). During instances of high confidence, the consensus of the points is accepted, distance-based weight updates are not performed, and the program moves on.

How accurate a predictor is depends on a number of factors, further explained in section 4.5. By updating predictors according to their accuracy, future predictions end up being closer to their validated / ground truth positions. The result is an *increase in prediction quality* over time.

Tracking Pipeline

The predict/update phases of the Kalman Filter is implemented in a number of components, each one having required different experimentation and tuning to get right.

In ApiCV, the pipeline is implemented in a `VideoProcessor()` as follows:[1]

1. Point clouds are generated surrounding the query point. Further details on cloud generation are discussed in section 4.4.

2. Points are then processed in "segments" (detailed in section 4.2) to ease computing resources and re-structure the clouds to improve accuracy over time.

3. For each segment, cloud movement is approximated by the optical flow tracker of choice (detailed in section 4.2):

   ```
   # Optical flow processing
   slice_result = estimator.process_video_slice(frames, width, height, resized_points)

   # Get final positions
   final_positions = slice_result.get_final_positions(point_clouds)
   ```

4. Cloud processing begins by determining the most likely rotation of the cloud, the total deformity, and each cloud point's prediction of where it believes the query point is. Details on this step can be found in section 4.3.

   ```
   # Calculate rotation, deformity, and predictions
   rotation, deformity, predictions = calculate_rotation_deformity(old_cloud, final_pos)
   ```

5. Next steps include determining the inliers of the cloud after movement, and predicting the location of the query point according to the prediction component (detailed in section 4.3).

   Additionally, the first weight updates are performed, based on which points are considered to be inliers and outliers (detailed in section 4.5).

[1] Verbose code, such as passing many arguments into class constructors or functions, were replaced with $\cdots$ to ease readability.

```
# Predict inliers
cloud_inliers = inlier_predictor.predict_inliers(old_cloud, predictions)

# Predict query point
query_prediction = query_point_predictor.reconstruct_query_point(
    old_cloud, predictions, cloud_inliers
)

# Update weights by outliers
weights = calculate_outlier_weights(old_cloud, cloud_inliers)
```

6. The point cloud is then reconstructed according to the selected cloud reconstructor component, which includes rebuilding outliers if necessary (detailed in section 4.4).

```
# Reconstruct point cloud
predicted_cloud = reconstruct_point_cloud(
    old_cloud, final_pos, cloud_inliers, rotation, query_prediction, weights
)
```

7. Validation may be requested from the user during instances of low confidence.

   If validation is requested, this will re-write the current query point value and update cloud point weights based on their proximity to the validated position. Otherwise, the current prediction and weights are accepted and the pipeline continues (detailed in section 4.5).

```
# Validate and update weights by distances
query_points, point_clouds = validate_and_update_weights(
    point_clouds, predicted_clouds, inliers, deformities, ...
)
```

8. Finally, the results are combined to generate tracks which are layered on top of the original video (detailed in section 4.6).

```
# Obtain segment tracks (interpolated, raw, and smoothed)
interpolated_tracks, raw_predictions, smoothed_tracks = generate_segment_tracks(...)

# Generate video segments and annotations
video_segment = slice_result.get_video()
```

Through this iterative prediction and update process, the system continuously improves its tracking accuracy by learning from the reliability of individual cloud points.

The success of this Kalman filtering approach depends on approximating the movement of individual points within the cloud, addressed in the following section 4.2.

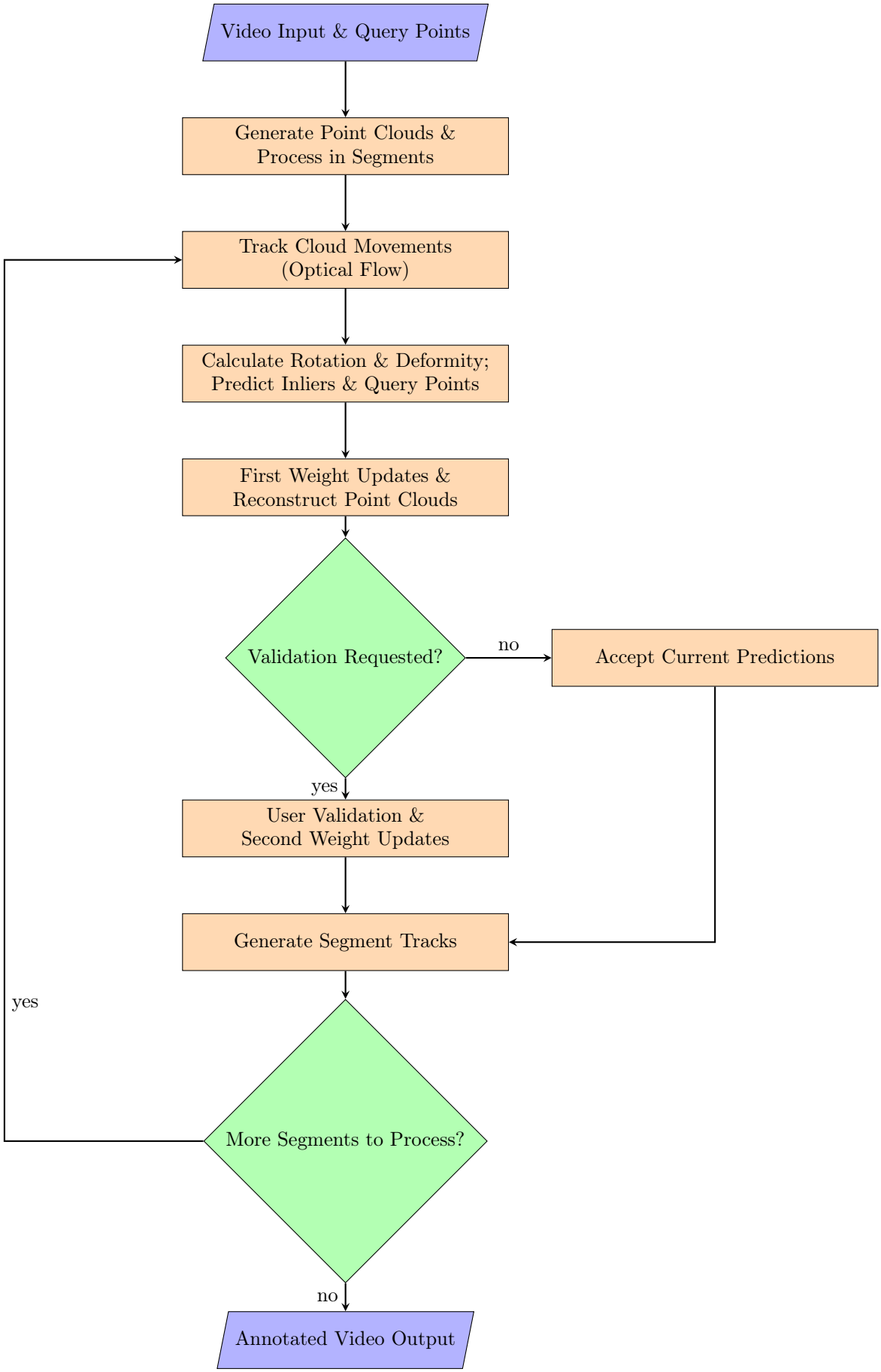


Figure 4.2: Diagram Representing Tracking Pipeline

4.2 Point Cloud Movement

Cloud-based tracking in ApiCV requires understanding the overall cloud movement (see 2.6) before predicting individual query point trajectories. That is, in order to predict the movement of a query point $q \to p$, ApiCV must first understand the movement of all the cloud points $c_i \to c_i'$.

A key aspect of the program is its ability to work without prior training. This is due to the wide range of domains users may need to work with. The goal, therefore, is to find an out-of-the-box method that can track individual points reasonably well, so that it may be analysed and processed by the rest of the pipeline.

Cloud Movement Function

The underlying point tracking method needs to be a function

$$g(x, y, f_0, f) = (x', y')$$

that accepts an arbitrary point (x, y) at frame f_0, and returns the approximate location (x', y') of that point at frame f. This function need not be perfect, and may be subject to some noise w, but should be good enough for approximations. This will be referred to as the "cloud movement function."

The cloud points generated at the beginning of the processing phase should be moved by the cloud movement function, providing a list of the positions $(x', y')[]$ for each cloud point throughout the frames $\in [f_0, f]$ in the segment.

This project uses TAPIR, mentioned in literature review section 2.5, which is a "Tracking Any Point" library by Google DeepMind [11], and serves as a baseline for the rest of the tracking pipeline.

The system's modular architecture facilitates the integration of alternative point-tracking methodologies as they become available. During ApiCV's development cycle, for example, DeepMind released TAPNext [12] (`https://tap-next.github.io/`), a successor to TAPIR, demonstrating the rapid evolution of this field.

ApiCV's design emphasises compatibility with any point-tracking library, so long as it fits the requirements of the cloud movement function g. This ensures ApiCV is adaptable to future technological advances and maintains relevance as underlying tracking methodologies continue to evolve.

Processing in Segments

The tracking is done in *segments*, as mentioned in section 2.6. Breaking apart processing into several smaller chunks provides a number of benefits:

Updates Segments provide structured, logical points at which the weights of the cloud points can be updated

Reconstruction/Realignment As points drift away and create noise, the cloud can be reconstructed and points can be re-positioned in better locations. While this could be done on a frame-by-frame basis, working with small segments avoids excessive computation.

Processing and Memory TAPIR, and any optical flow program for that matter, requires substantial computation power. More importantly, the longer these programs run the more memory they require. This is due to the nature of working with video, typically with high framerates, and the need to compare points against previous frames.

By processing in segments, ApiCV writes previous segments and information to persistent memory, and frees up space for further computation.

4.3 Query Point Prediction

Query point prediction is the fundamental goal of any tracking software: *given an initial position* (x, y) *at time* t_0*, where has that point moved to after time* δt*?* This is the same goal, albeit aiming for higher accuracy, as the cloud movement function g.

After determining the movement of a *point cloud* according to the cloud movement function g, ApiCV's approach to answering this question uses ensemble (2.4) and consensus-based (2.4) methods. That means taking a top-down view of the cloud, its starting $(x, y)[]$ and ending positions $(x', y')[]$, the cloud points' current weights, and determining what the most likely position p of the query point is. This happens in the *Prediction* phase of the Kalman Filter architecture outlined in section 4.1.

Query point prediction, alongside cloud reconstruction (further discussed in section 4.4), was one of the more experimental parts of the project. Many different methods were attempted, the full evaluation of which is discussed in section 5.3.

Outliers and Inliers

An influential part of the query point prediction involves determining which cloud point are / have become *outliers*, and which are *inliers*. Initial experimentation with untrained point tracking software proved that cloud points could experience drift for a number of different reasons such as attaching to the background, attaching to another bee, or just poor positioning.

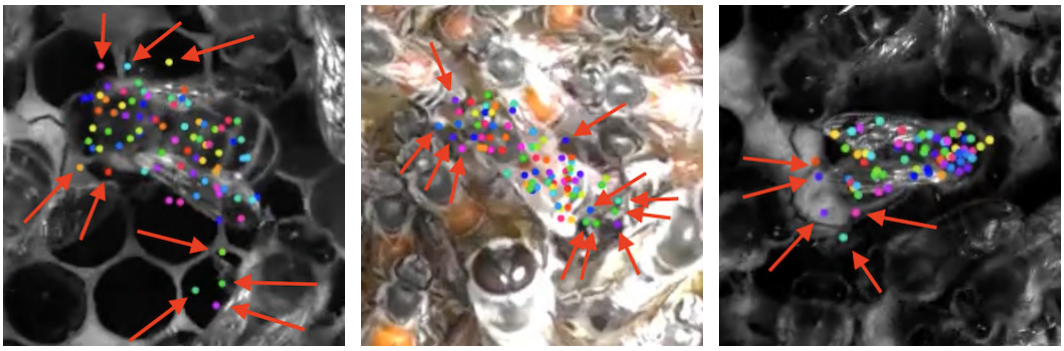


Figure 4.3: Point clouds pre-reconstruction after 1 second of movement. Arrows point to cloud points that should be considered outliers

It was clear that some form of outlier/inlier detection was necessary to determine which points were "reliable," and which points should be discarded during prediction.

Clustering Clustering, as defined in background section 4.3, is the process of categorising points into groups, and can also be used to determine inliers and outliers of a dataset.

The initial attempt at clustering involved implementing RANSAC (as mentioned in 2.4), which is an iterative method to determine parameters of a some model from data that contains outliers (which should be accorded no influence). RANSAC proved to be a strong candidate but experienced performance issues due to the number of iterations required to effectively cluster and determine outliers.

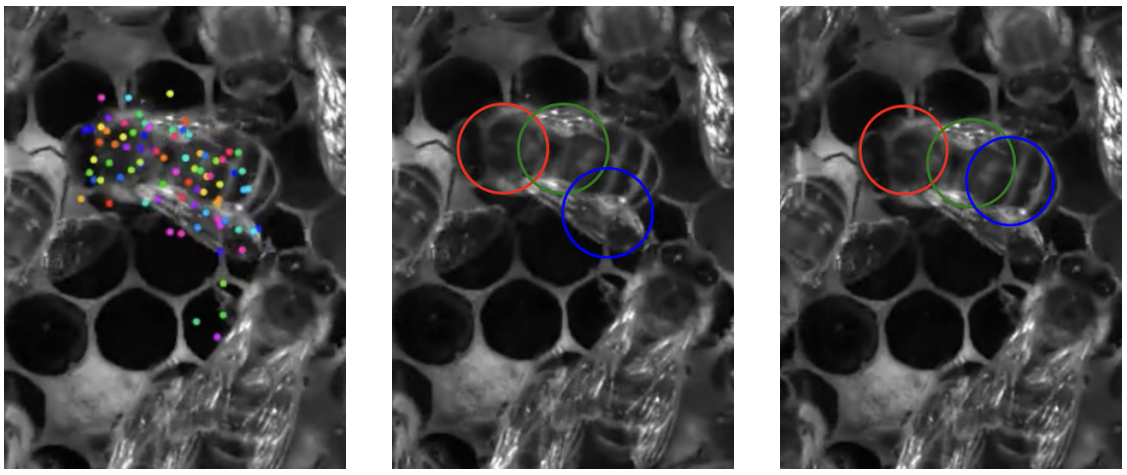
The most suitable solution for outlier detection ended up being Sci-Kit Learn's DBSCAN, which accepts a parameter epsilon ϵ representing the maximum distance between two points for them to be considered "neighbours." The value used in ApiCV is:

$$\epsilon_{DBSCAN} = r * \epsilon_{\text{user}}$$

That is, the radius r multiplied by the coefficient passed in by the user, ϵ_{user}, as discussed in 3.1. This ensures clustering is proportionate to the size of a point cloud.

Clustering with DBSCAN proved to be an efficient and effective method of determining outliers/inliers of a particular cloud. Figures 4.4 and 4.5 demonstrate improvements to the prediction accuracy by implementing clustering and excluding outliers from predictions. The mean distances

from figure 4.5 were 7.22px *without* clustering, but 5.58px *with* clustering, resulting in a 22.7% improvement in prediction value.



(a) DANCE_15 Underlying Point Cloud After 1 Second

(b) DANCE_15 Query Point Predictions with **No Clustering**

(c) DANCE_15 Query Point Predictions with **Clustering** ($\epsilon = 1$)

Figure 4.4: Qualitative Comparison of Query Point Predictions With and Without Clustering.

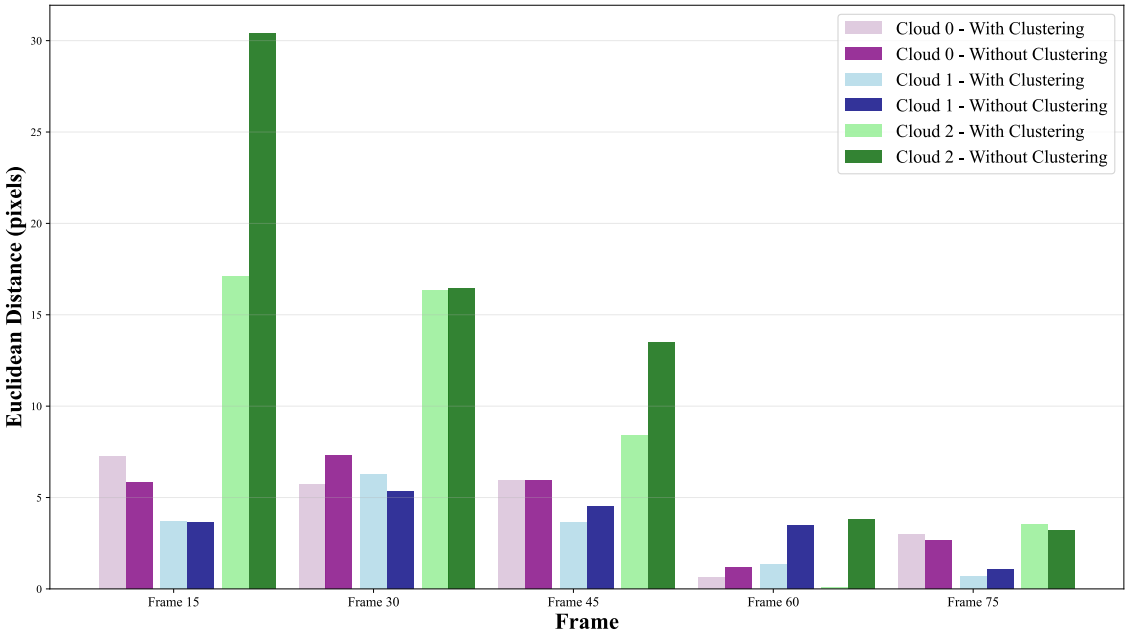


Figure 4.5: Euclidean Distance Comparisons With and Without Clustering For First 5 Seconds of DANCE_15. Exact validation values can be found in A.7

Implementation and Rotation-Invariance

An important consideration when clustering was achieving rotation-invariance. As demonstrated in figure 4.1, the cloud may have rotated by the end of the segment. Simply adding the original displacement vectors to the the new clouds may result in inconsistent predictions. An example can be seen in figure 4.6. Note how, without consideration of rotation, the displacement vectors after time $t + \delta t$ are identical to those at time t_0, resulting in particularly inconsistent predictions p_i for the query point. When rotation is taken into consideration, predictions p_i are closer together, albeit with some noise, resulting in a greater consensus.

In the initial RANSAC implementation, rotation was simply treated as another parameter for

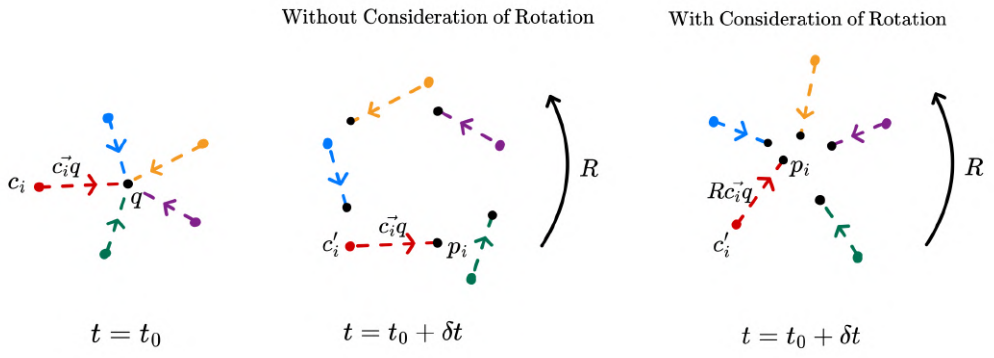


Figure 4.6: Diagram of Predictions With and Without Consideration of Rotation.

estimation, alongside the query point prediction, outliers, and inliers. When shifting to clustering, an iterative approach (inspired by rotation-invariant depth-measurement histograms used in robotics) was implemented with satisfactory results.

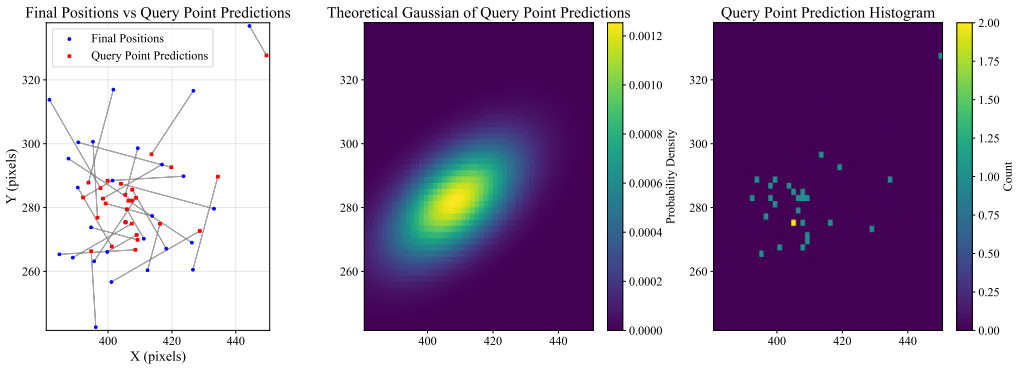


Figure 4.7: Distribution For 290° of Rotation in DANCE_15 (Frame 49, Cloud 0). Deformity = 16,001.17

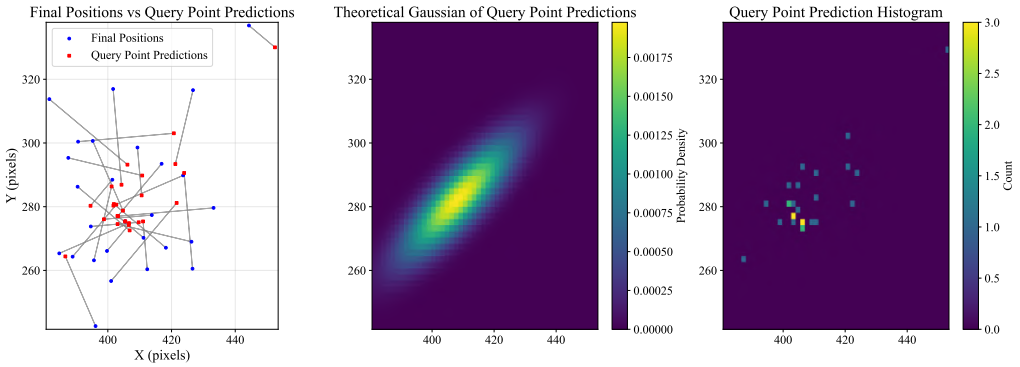


Figure 4.8: Distribution For 310° of Rotation in DANCE_15 (Frame 49, Cloud 0). Deformity = 6,541.45

After rotation, predictions are tested using deformity D (detailed in 2.6 of the terminology section). Rotations that generated predictions with the lowest cloud deformity are accepted and clustering is applied to determine outliers/inliers.

Figures 4.7 and 4.8 demonstrate how rotation affects the deformity of a prediction. 310° of rotation results in a lower deformity value than 290°, and would therefore be accepted as the most

likely rotation of a cloud.

Point Prediction

After clustering, the *point prediction* process begins. Various methods were attempted, including a **Simple Average**, a **Weighted Average**, an **Incremental Neural Network**, and the **Weighted Average of Inliers**.

These predictions correspond to the "prediction phase" of the Kalman Filter architecture, accepting the displacement vectors, cloud point positions, inliers, and rotation to determine the most likely position for the query point.

Incremental Neural Network The initial approach was to use an Incremental Neural Network (INN) that would improve in its prediction accuracy as the evaluation progresses. The results were unfortunately insufficient, especially when compared to the algorithmic approaches, but could be improved with more hyper-parameter tuning.

Statistical Approach: Weighted Average of Inliers The best performing prediction method was to consider only the inliers of the final point cloud (as discussed in the clustering section), and calculate the *weighted* average of these inliers, based on their previously assessed weights. Figures 4.9 to 4.12 demonstrate the improvement in deformity of a cloud when weights and inliers are considered. A full evaluation is detailed in section 5.3.

The weighted average of all inliers proved to not only have the lowest deformity metric, but also the shortest average euclidean distance from the *predicted* query point to the *validated* query point. It was therefore selected as the default **Query Point Predictor** component.

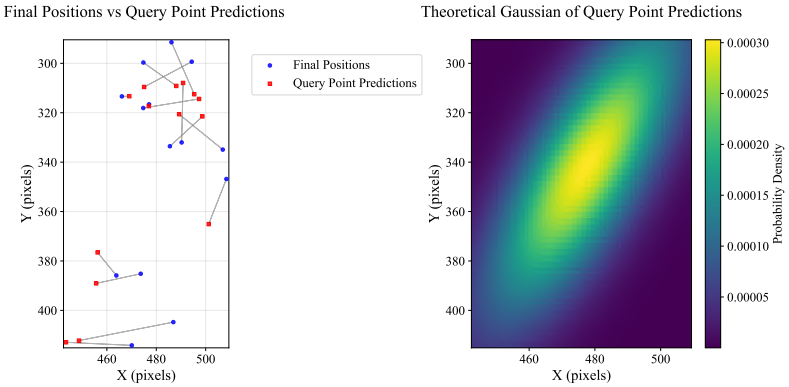


Figure 4.9: Average of All Predictors - DANCE_15 (Segment 1, Cloud 2).
Deformity = 276,045.32

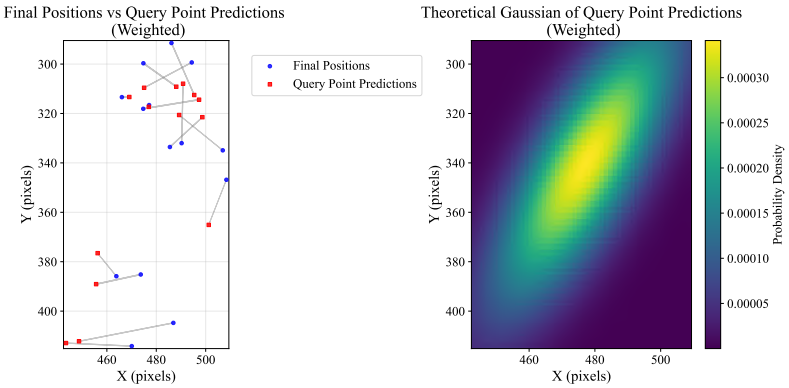


Figure 4.10: Weighted Average of All Predictions - DANCE_15 (Segment 1, Cloud 2).
Deformity = 217,456.30

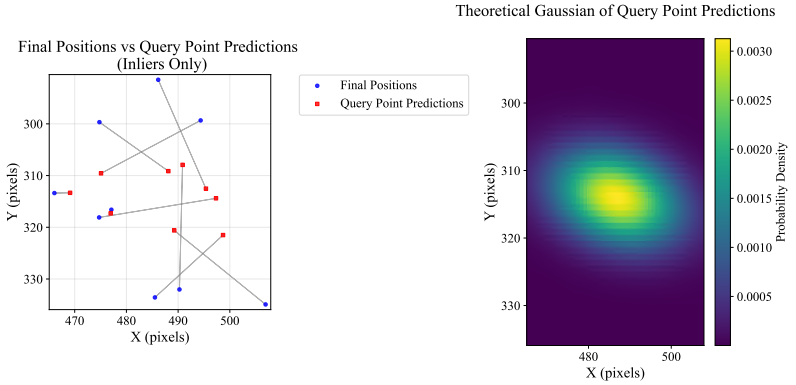


Figure 4.11: Average of All Inliers - DANCE_15 (Segment 1, Cloud 2).
Deformity = 2571.94

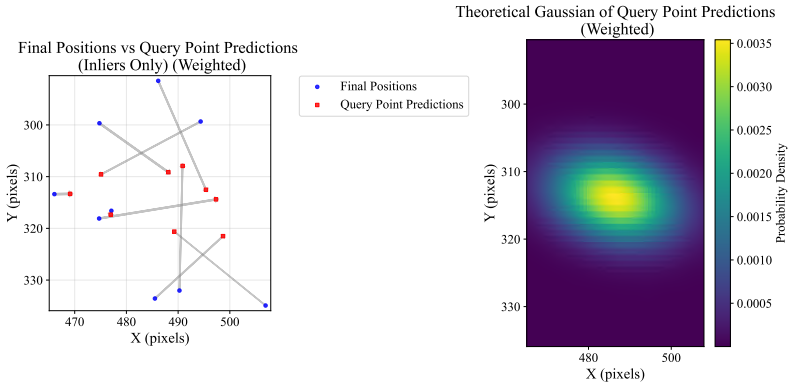


Figure 4.12: Weighted Average of All Inliers - DANCE_15 (Segment 1, Cloud 2).
Deformity = 2014.27

Implementation After rotation, inlier, and weight considerations, the Weighted Average of Inliers prediction is straightforward:

$$\tilde{w}_i = \frac{w_i}{\sum_{j \in \text{inliers}} w_j} \quad \text{for } i \in \text{inliers}$$
$$p = \sum_{i \in \text{inliers}} \tilde{w}_i * p_i$$

where

$$w_i = \text{weight for point } i$$
$$p_i = \text{point } i\text{'s prediction}$$
$$n = \text{number of cloud points}$$
$$\text{inliers} = \text{indices for inlier predictions}$$

After query point prediction is finished, the point cloud needs to be reconstructed to continue processing the next segment.

4.4 Cloud Reconstruction

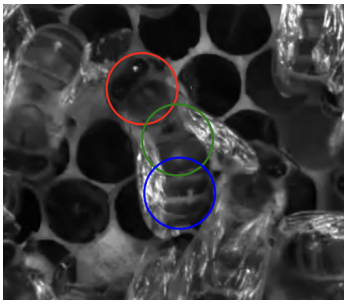
Besides query point prediction, cloud reconstruction proved to be one of the most important aspects of the implicit reliability of predictions.

Cloud reconstruction is the process of generating / rebuilding the point cloud after a segment. It is one of the primary achievements of this project since leaving clouds untouched throughout tracking would be no different than running TAPIR or some other optical flow analysis without any modifications.

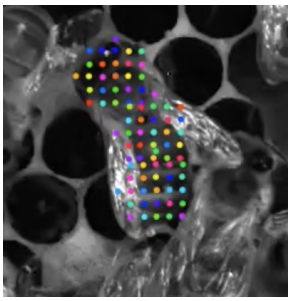
Cloud reconstruction was also very experimental in nature. The evaluation metric used was deformity D, explained in section 2.6. The ideal behaviour is that well-constructed point clouds would maintain relatively the same structure at the end of a segment that it had at the beginning, resulting in a low deformity metric.

Initial Cloud Generation

In all experiments, clouds are initially generated in the same way: as a circle with centre (x, y) and radius r, the initial query point and radius proivded by the user.



(a) Initial Query Points and Radius Selections



(b) Clouds Generated with Selections

Figure 4.13: Cloud generation from initial query point and radius selection

The radius r is an important parameter: it not only dictates the size of the point cloud to be generated, but is the only parameter that the tracking algorithm uses to assess the size of a subject/bee. The radius is additionally when calculating deformity and clustering as discussed in the outliers/inliers section 4.3.

Each cloud is generated as a circular grid as follows:

```
cloud_points = []

# Generate grid according to a provided perimeter density
grid_step = (2 * radius) / sqrt(n_points_per_perimeter * 3)
x_min, x_max = center_x - radius, center_x + radius
y_min, y_max = center_y - radius, center_y + radius
x_coords = np.arange(x_min, x_max, grid_step)
y_coords = np.arange(y_min, y_max, grid_step)

# Add all points within the radius
for x in x_coords:
    for y in y_coords:
        distance = sqrt((x - center_x)**2 + (y - center_y)**2)
        if distance <= radius:
            cloud_points.append((x, y))
```

Weights are initially set to be uniform, since no prior information is known about the reliability of their positions.

$$\forall i \in [1, n]. \left(w_i = \frac{1}{\sum_{j=1}^{n} w_j} \right)$$

where n = number of cloud points.

Displacement vectors (defined in background section 2.6) are initialised, and the point cloud is created.

```
    # Initialise displacement vectors
    vectors_cp_to_qp = np.array([query_point - point for point in cloud_points])

    return PointCloud(
        query_point,
        cloud_points,
        radius,
        vectors_cp_to_qp,
        rotation=0.0,
        weights=weights
    )
```

Cloud Reconstruction

After a segment, the cloud will have shifted / deformed from its original shape. Reconstruction is the process of at least partially recreating the shape of the original point cloud. Like query point prediction, various methods were attempted, an evaluation of which is detailed in section 5.3.

The final method produced (and now the default option in ApiCV) was labelled **Reconstruct Outliers with Cluster Recovery**, and works as follows:

First assess the euclidean distance between the *predicted* query point, and the *validated / accepted* query point. If this is greater than the radius r of the cloud, the prediction has strayed too far off track and the cloud should be reconstructed entirely. Otherwise, the prediction is close enough that only the *inliers* need to be reconstructed. Figure 4.14 expresses this.

$$d_{\text{val}} = \|v - p\|_2$$

$$\forall c_i \in C. c_i = \begin{cases} \text{reconstruct\_outliers}(c_i) & \text{if } d_{\text{val}} < r \\ \text{reconstruct\_with\_centre\_rotation}(c_i) & \text{otherwise} \end{cases}$$

where

$$C = \text{the cloud points to be reconstructed}$$
$$v = \text{validated query point}$$
$$p = \text{predicted query point}$$
$$r = \text{point cloud radius}$$

Figure 4.14: Reconstruct Outliers with Cluster Recovery

Reconstructing outliers is implemented by selecting a random original displacement vector (original displacement vector is defined in 2.6) to replace the outlier with, and retaining all the rest.[2]

[2]Randomly selected displacement vectors do not need to be rotated. The original displacement vectors are generated in a uniform 2D grid, so random selection would result in the same expected value with and without rotation.

$$c_i = \begin{cases} c_i & \text{if } i \in \text{inliers} \\ v + \text{random\_choice}(\mathbf{V}_{\text{orig}}) & \text{if } i \notin \text{inliers} \end{cases}$$

where

$$\mathbf{V}_{\text{orig}} = \text{the original displacement vectors, determined during cloud initialisation}$$

Figure 4.15: reconstruct_outliers(c_i)

By replacing the outlier point, it is effectively given a "second chance" to become more reliable if it can settle down into a better position on the subject.

Full reconstruction involves re-creating the initial cloud grid, but with the rotation calculated in section 4.3.

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$
$$c_i = R \cdot \mathbf{v}_{\text{orig}_i} + v$$

where

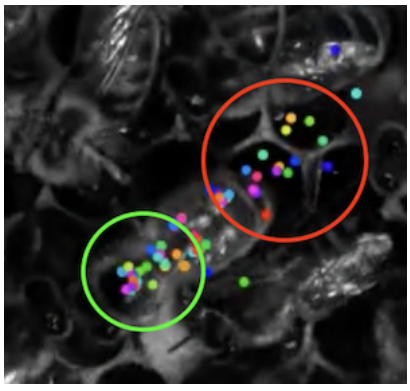
θ = rotation angle determined in 4.3

$\mathbf{v}_{\text{orig}_i}$ = the original displacement vector of point c_i, determined during cloud initialisation

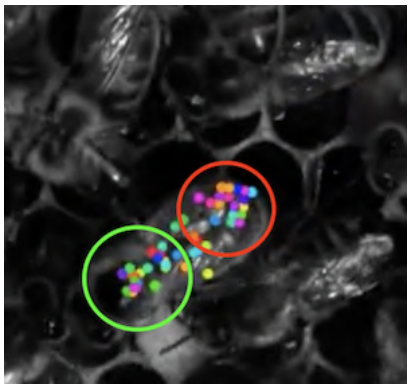
Figure 4.16: reconstruct_with_centre_rotation

In summary, if the validation is too far away from the prediction, perform a full reset. Otherwise, replace all outliers with a random point and continue.

Figures 4.17 and 4.18 display the cloud points of a bee before and after reconstruction according to the final reconstruction algorithm. In figure 4.17, the green circle demonstrates the "Reconstruct Outliers" part of the algorithm, whereas the red circle demonstrates the "Reconstruct with Centre Rotation" part. Figure 4.19 displays more clearly which points belong to which cloud in the example 4.18.

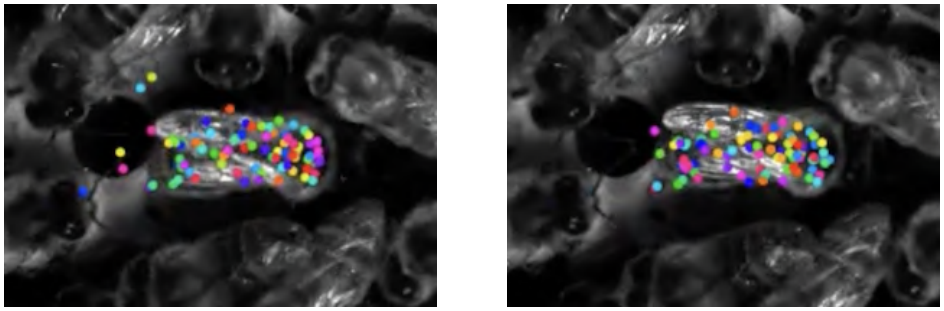


(a) Cloud Points Pre-Reconstruction



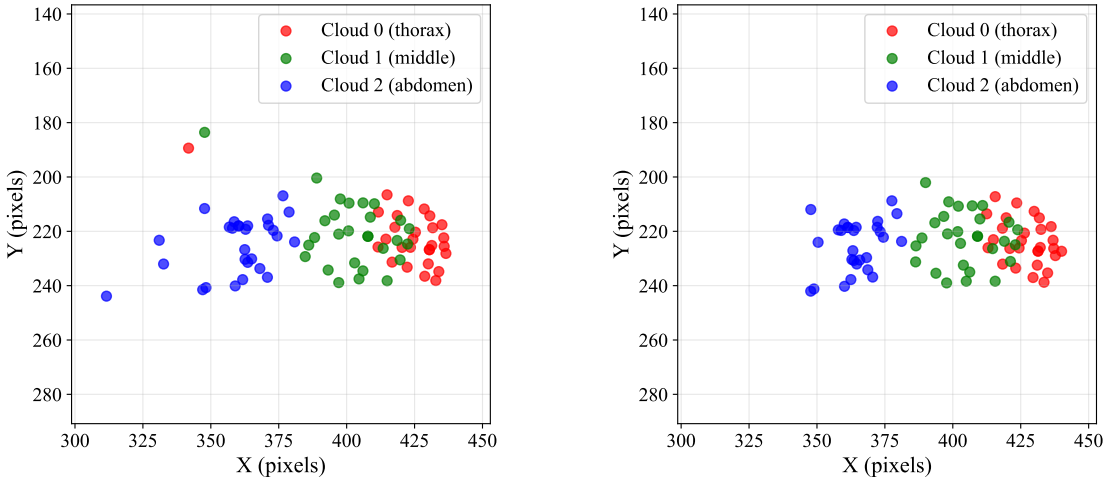
(b) Cloud Points Post-Reconstruction

Figure 4.17: Cloud Reconstruction in Segment 9 of ROUND_DANCE



(a) Cloud Points Pre-Reconstruction (b) Cloud Points Post-Reconstruction

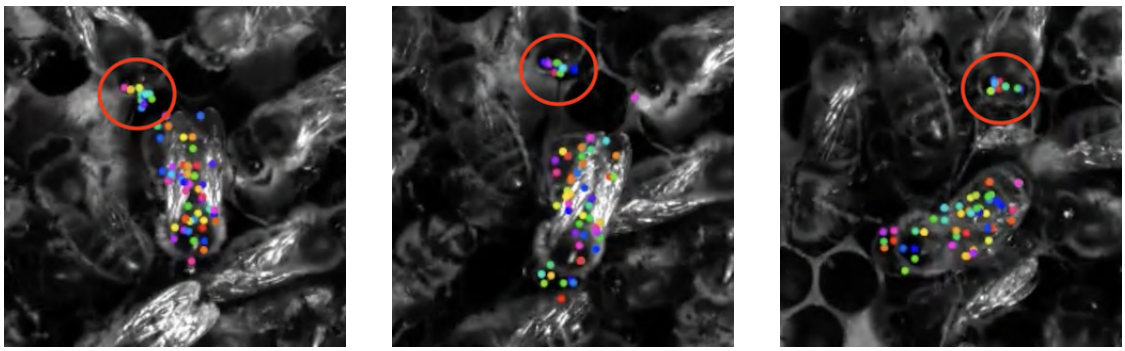
Figure 4.18: Cloud Reconstruction in Segment 1 of ROUND_DANCE.



(a) ROUND_DANCE Pre-Reconstruction Cloud Points

(b) ROUND_DANCE Post-Reconstruction Cloud Points

Figure 4.19: Cloud Reconstruction in Segment 1 of ROUND_DANCE, as can be seen in figure 4.18.



(a) Beginning Segment (b) Middle Segment (c) Ending Segment

Figure 4.20: Example of a Mistaken Cluster

Mistaken Clustering The motivation behind a full cloud reset was due to a common phenomenon referred to as "mistaken clustering." Mistaken clustering is when the algorithm believes it has found a good cluster and therefore *doesn't* reconstruct the inlier points, but the cluster is not actually attached to the bee. This often happens if a cluster has attached itself to the background or to another bee.

An example of this can be seen in figure 4.20, where the cluster has attached itself to a different bee, creating a feedback loop by identifying the wrong points as inliers. The mistaken cluster is circled in red. When mistaken clustering occurs, deformity increases and a validation request is triggered. By assessing the distance between the prediction p and the validation v, mistaken clusters can be mitigated early on.

Recalculating Displacement Vectors The final aspect of cloud reconstruction is recalculating the displacement vectors. Specifically, that is updating $\vec{c_iq}$ to $\vec{c_i'q}$ after a new q has been accepted, as demonstrated in figure 4.21.

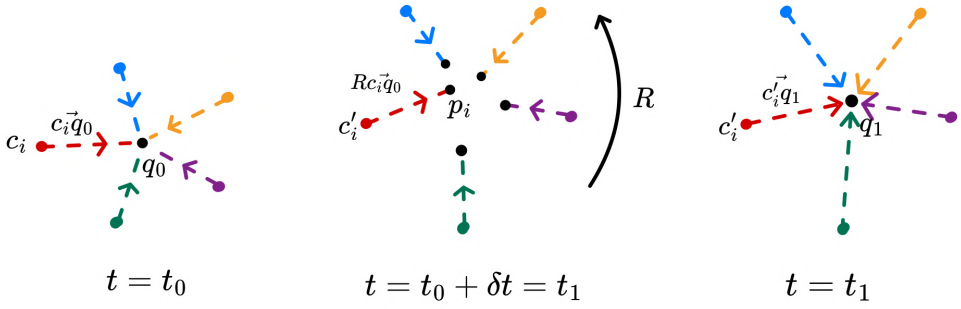


Figure 4.21: Visual Demonstration of How Displacement Vectors c_iq are redrawn

This is necessary to maintain accuracy in subsequent segments, as individual cloud points will have drifted during point cloud movement.

Resulting Behaviour

The resulting behaviour of the "Reconstruct Outliers + Cluster Recovery" method was that well-positioned points ended up "settling in" to parts of the bee that tended to be reliable. By increasing the weights of these well-positioned points, overall reliability of these predictions is maintained in subsequent frames.

4.5 Reliability of Points and User Validation

After query point predictions, the *Observation* phase of the Kalman Filter architecture outlined in section 4.1 begins.

One of the benefits of using a cloud of points instead of a single one is adding *weights* to each predictor based on its performance. Weights are initialized uniformly and updated based on clustering and distance to the validation point (if validation is requested).

First Observation: Outlier-Based Weight Updates

Outlier-based weight updates happens after query point prediction in section 4.3 but before cloud reconstruction in section 4.4. This is due to the fact that validation may not always be required, at which point the pipeline can accept its prediction and continue to the next segment.

After performing clustering (as discussed in 4.3), outlier points that are reconstructed are necessarily given lower weights. This is the first "observation" of the Kalman Filter, and is not only needed because the outlier points were poorly positioned prior to reconstruction, but also due to the uncertainty of their new position.

Outlier weight updates are made according to the following equation to penalize outliers:

$$\tilde{w}_i^{(t+1)} = \begin{cases} w_i^{(t)} \cdot \lambda & \text{if } i \text{ is an inlier} \\ w_i^{(t)} & \text{otherwise} \end{cases}$$
$$w_i^{(t+1)} = \frac{\tilde{w}_i^{(t+1)}}{\sum_{j=1}^{N} \tilde{w}_j^{(t+1)}}$$

where

$$w_i^{(t)} = \text{previous weight for point } i$$
$$w_i^{(t+1)} = \text{updated weight for point } i$$
$$\lambda = \text{weight penalty. Default is } \lambda = 0.5$$

User Validation and Cloud Confidence

The primary purpose of ApiCV was to provide an annotation tool that doesn't require large amounts of prior training or annotation. While ensemble and consensus-based approaches work well when the subject is making regular, steady movement, problems frequently occur during high movement speeds and rotations. Additionally, in particularly challenging domains such as the *apis florea* hive (the OUTSIDE_FLOREA video, displayed in background section 2.1), large amounts of noise and occlusion can cause severe problems for consistent tracking. As a result, a degree of human validation is needed to keep annotations on-track in these cases.

Requesting occasional annotation still maintains the initial goal of *minimising* the amount of human input required to make annotations, as validation is requested at most once every segment (1 second by default). This means the same amount of work is required to annotate 1 minute of a 10fps video as is required to annotate 1 minute of a 50fps video.

Validation need not always be required, however. Cloud confidence C is used to determine whether or not a user needs to provide validation, and is based on the inlier ratio of the cloud and the deformity confidence:

$$\text{Inlier Confidence:} \quad \rho = \frac{|\text{inliers}|}{|\text{point cloud}|}$$
$$\text{Deformity Confidence:} \quad \phi = 1 - \min\left(\frac{D}{r^4\delta}, 1\right)$$
$$\text{Total Confidence:} \quad C = \frac{\rho + \phi}{2}$$

Deformity D is the variance of the "most likely rotation" of the cloud (as discussed in section 4.3), which is the determinant of the covariance matrix of the query point predictions.

It is necessary to divide the deformity D by r^4 due to how the deformity scales with the radius, which is effectively a scaling factor between distributions. Say a distribution X' is formed by scaling another distribution X by scaling factor a (e.g. changing the radius from 20 to 32). It then follows that, by the definition of deformity:

$$\begin{aligned} \text{Deformity}(X') &= \text{Deformity}(aX) \\ &= \det(\Sigma) \\ &= \det \begin{bmatrix} \text{Var}(aX_1) & \text{Cov}(aX_1, aX_2) \\ \text{Cov}(aX_1, aX_2) & \text{Var}(aX_2) \end{bmatrix} \\ &= \text{Var}(aX_1)\text{Var}(aX_2) - \text{Cov}(aX_1, aX_2)^2 \\ &\propto a^4\text{Var}(X_1)\text{Var}(X_2) - a^4\text{Cov}(X_1, X_2)^2 \\ &\propto a^4\text{Deformity}(X) \end{aligned}$$

Deformity is therefore divided by r^4 to achieve more consistent results for calculating confidence.

There is, however, a degree of uncertainty considering how the shape of a cloud can change. To account for this, deformity is additionally divided by the coefficient δ, the "Deformity Delta", passed in by the user as discussed in section 3.1. This provides the user with control over how confident they wish ApiCV to be. Higher δ values will result in higher confidence, and therefore be less likely to trigger validations. Lower values will do the opposite.

The two forms of confidence, inlier and deformity, typically go hand-in-hand, but there are exceptions to these cases in which it is useful to take the average of the two.

The trigger to request validation is based on the minimum of all confidence values. If any one is lower than the threshold, they will request validation from the user. This is implemented as follows: [1]

```
def validate_and_update_weights(...):
    # Calculate confidence for each point cloud
    confidences = [
        cloud_confidence(inliers, deformity, deformity_delta, radius, ...)
        for cloud, inliers, deformity in zip(predicted_point_clouds, inliers, deformities)
    ]

    # Trigger validation if any confidence is below threshold
    if min(confidences) <= self.confidence_threshold:
        # Get user validation and reconstruct point clouds
        final_queries, final_clouds = self._handle_user_validation(
            predicted_point_clouds, predicted_queries
        )
    else:
        # Accept predictions as-is
        final_queries = predicted_queries
        final_clouds = predicted_point_clouds
```

Second Observation: Distance-Based Weight Updates

If validation is requested, the user provides the *new* query points through the frontend ApiCV interface, functioning as the second "observation" of the Kalman Filter. This will:

1. Replace whatever the previous predicted query point p was with the new, *validated* query point (used to update the displacement vectors as discussed in section 4.4).

2. Update the predictor weights w_i according to the distance between their prediction p_i and the validated query point v.

The distance-based weight updates are done according to an **exponentially-weighted moving average**[3] such that older observations are given less weight:

$$\tilde{w}_i^{(t+1)} = w_i^{(t)} \cdot (1 - \sigma) + \exp\left(-\frac{d_i}{r}\right) \cdot \sigma$$

$$w_i^{(t+1)} = \frac{\tilde{w}_i^{(t+1)}}{\sum_{j=1}^{N} \tilde{w}_j^{(t+1)}}$$

where

[1] Verbose code, such as passing many arguments into functions or unnecessary details, were removed or replaced with $\cdots$ to ease readability.

[3] In the incremental neural network method for query point prediction, these weights are ignored as the weights are implicitly changed inside the network.

$$
\begin{aligned}
w_i^t &= \text{previous weight for point } i \\
w_i^{(t+1)} &= \text{updated weight for point } i \\
d_i &= \text{distance from point } i\text{'s prediction to the true query point} \\
\sigma &= \text{error sigma} \\
r &= \text{point cloud radius} \\
N &= \text{total number of points}
\end{aligned}
$$

Continuing the Tracking Pipeline

After weight updates and validations, the Observation phase of the Kalman Filter is finished and results are used in the next segment's Prediction phase.

Outside of the tracking pipeline, ApiCV still needs to turn predictions and validations into concrete annotations for the user.

4.6 Producing Annotations

While the user may provide ground-truth annotations every validation request, it is ApiCV's job to produce reliable annotations for the *intermediate* frames.

For example, in a 50fps video, the user may provide ground truth data for only 1/50 frames. Predicting the remaining 49 is the goal of ApiCV.

Smoothing Methods

The program produces intra-segment tracks (annotations) using the underlying point cloud associated with a query point (the same way it produces predictions at the end of a segment). On their own, however, these may end up being jittery.

Furthermore, the user-validated query point v will often be different from the predicted point p. If ApiCV purely used its own predicted values, there would be a "jump" at the end of each validated segment from p to v. This means a balance must be struck between the human-validated positions and the predicted tracks.

Interpolation-Mean Balance

This balance is achieved by mixing:

- a linear interpolation of the beginning and final query points of a segment
- the weighted average of inliers in each frame, using the same algorithm to predict the query point at the *end* of a segment (as discussed in section 4.3)

The balance is not equal throughout the segment. At the beginning and end of the segment, the weight is 100% towards the interpolated points. This ensures that the first and last frames will have precisely correct annotations and avoid sudden "jumps."

For intermediate tracks, e.g. halfway through a segment, more weight is given to the predicted points. If a subject is moving and *rotating*, for example, interpolation will not accurately reflect the curve of its movement. Therefore, the ratio of how much weight to give interpolations vs. predictions changes throughout the segment.

The equation used to allocate interpolation weight is defined as:

$$y = (\frac{-1}{f-1}x + 1)^{\alpha} \qquad (\text{for } x \in [0, \tfrac{f-1}{2}])$$
$$y = (\frac{1}{f-1}x)^{\alpha} \qquad (\text{for } x \in (\tfrac{f-1}{2}, f-1])$$
$$(4.1)$$

where

$$y = \text{the interpolation weight } \in (0, 1]$$
$$(1-y) = \text{the prediction weight } \in [0, 1)$$
$$f = \text{video framerate}$$
$$\alpha = \text{smoothing factor } \alpha$$

Figure 4.22 demonstrates the weight given to *interpolation* throughout a segment in a 30fps video (such as RECORDING-2023), while figure 4.23 demonstrates the interpolation weight throughout a segment in a 15fps video (such as DANCE_15). Note how interpolation is granted 100% weight at the beginning and end of each segment, but is lower throughout the segment to give more weight to predictions.

The user may adjust how much interpolation and prediction they would like using the Smoothing Alpha α parameter (presented in section 3.1) as follows:

- $\alpha = 1$ will result in a roughly 50-50 split between raw mean tracks and interpolation.
- $\alpha > 1$ will give more weight to the raw mean than to the interpolation.
- $\alpha < 1$ will give more weight to interpolation than the raw mean.
- $\alpha = 0$ will give 100% weight to interpolation throughout the entire segment.

(**Note**: This is not the final output. All tracks and annotations (weighted inlier mean, interpolations, and smoothed tracks) are be available in the CSV annotations file should the user wish to change the smoothing alpha later.)

Smoothing is primarily meant to remove any "jumps" between predictions and validations, so it is recommended to keep $\alpha > 1$ (the default being $\alpha = 3$). Too much weight given to interpolation can often lead to less accurate tracks. Figure 4.24 highlights some of the jumps that can occur, and how smoothing eases them.

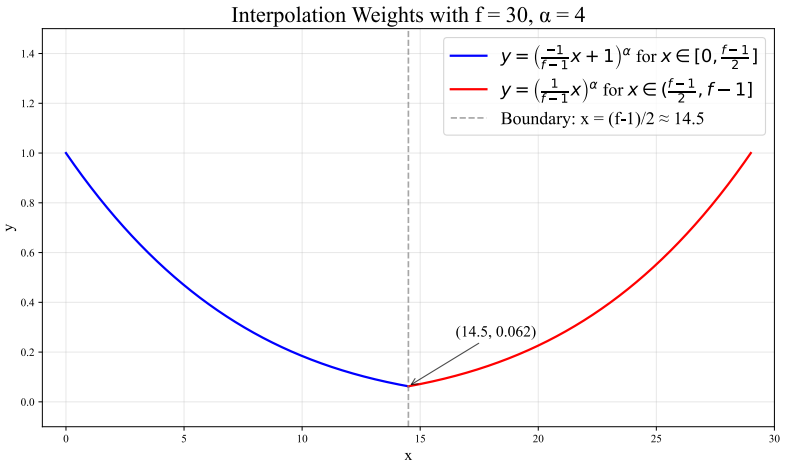


Figure 4.22: Interpolation Weights for $f = 30$ and $\alpha = 4$

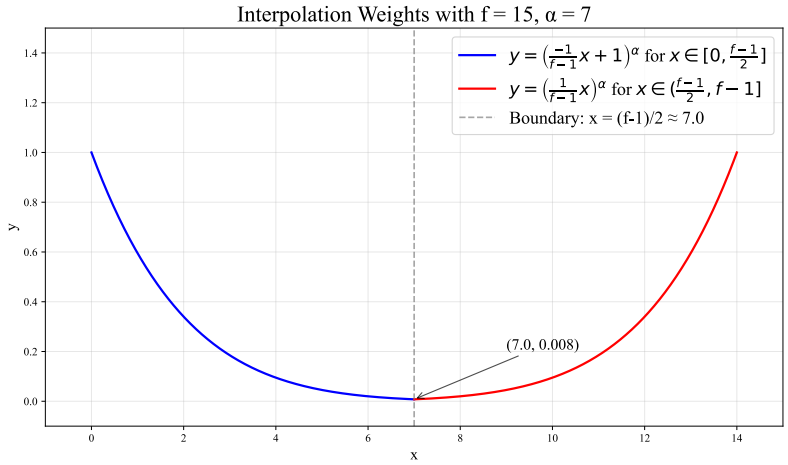


Figure 4.23: Interpolation Weights for $f = 15$ and $\alpha = 7$

Figure 4.24: Example of Smoothed and Unsmoothed Tracks in OUTSIDE_FLOREA. Jumps Circled in Red

Chapter 5

Evaluation

Evaluation was done both while experimenting with different tracking components, and as an overall comparison against the ground truth data.

Final **Intra-Segment** and **Persistency** evaluations were done after determining the best tracking components. These ended up being, as discussed in 4.3 and 4.4, the **Weighted Average of Inliers** for query point prediction and the **Redraw Outliers + Recovery** cloud reconstruction method.

The experimental results that justify these choices of query point prediction and cloud reconstruction components are displayed fully in section 5.3.

Annotations As explained in 2.1, entomologists from the University of Edinburgh provided some ground truth annotations they had made by hand, alongside the corresponding annotated videos. These videos are named ROUND_DANCE and RECORDING-2023.

For videos with no annotations, (DANCE_15 and OUTSIDE_FLOREA) validations were produced to assess persistency, query point prediction, and cloud reconstruction (the full validations of which can be found in appendix figures A.5, A.7, A.9, and A.11.

Parameters used in each experiment are also available in appendix figures A.6, A.8, A.10, and A.12.

5.1 Intra-Segment Evaluation

A primary goal of the project is to minimise the amount of manual work required when annotating a video, while also providing relatively reliable / stable results as discussed in the introduction 1.

When the user *does* provide annotations, they are taken as the ground truth. As discussed in the annotations section (4.6), the real benefit is providing accurate annotations for the frames that *aren't* validated by the user (that is, every other frame in the segment).

Evaluation Metric: Euclidean Distance to Ground Truth

To assess ApiCV's ability to generate reliable annotations, the frame-by-frame euclidean distances from produced annotations to the ground truth data were measured.

These are assessed with *full validation*, meaning validation is requested from the user 100% of the time, so the only point of comparison between methods is how close the predictions are *within* the segments. This can be seen in the graphs with the dashed grey lines, representing the points at which validation was requested. The drops to 0 pixels on these dashed lines come from validations.

Figures 5.1, 5.2, 5.4, 5.5, and 5.6 all display two lines:

- One in red, for annotations produced by an unadjusted point tracking (in this case TAPIR, the approximate cloud movement function discussed in 4.2)
- One in blue, for annotations produced by ApiCV using its default components (specifically those discussed in implementation sections 4.2, 4.3, 4.4, and 4.5)

For each frame f (along the x-axis), the distance (along the y-axis) is the euclidean distance from the respective tracking method's prediction p_f to the ground truth annotations v_f provided by the entomologists at the University of Edinburgh.

Additionally, overlays are provided for visualisation of the annotations, which can be seen in figures 5.3 and 5.7.

Results - ROUND_DANCE

Figures 5.1 and 5.2, and annotation overlays 5.3 demonstrate ApiCV's improvement over standard unannotated optical flow methods such as TAPIR with no adjustments. Table 5.1 displays quantitative improvements in mean and median distances from ApiCV annotations to the ground truth annotations, demonstrating a 28-29% improvement in mean distance.

The abdomen was particularly difficult to track using both tracking methods. In frame 100, a follower bee moved very closely to the tracking subject's abdomen creating a partial occlusion, which confused both TAPIR and ApiCV. Were it not for validation, the points would not have made it back onto the desired abdomen position.

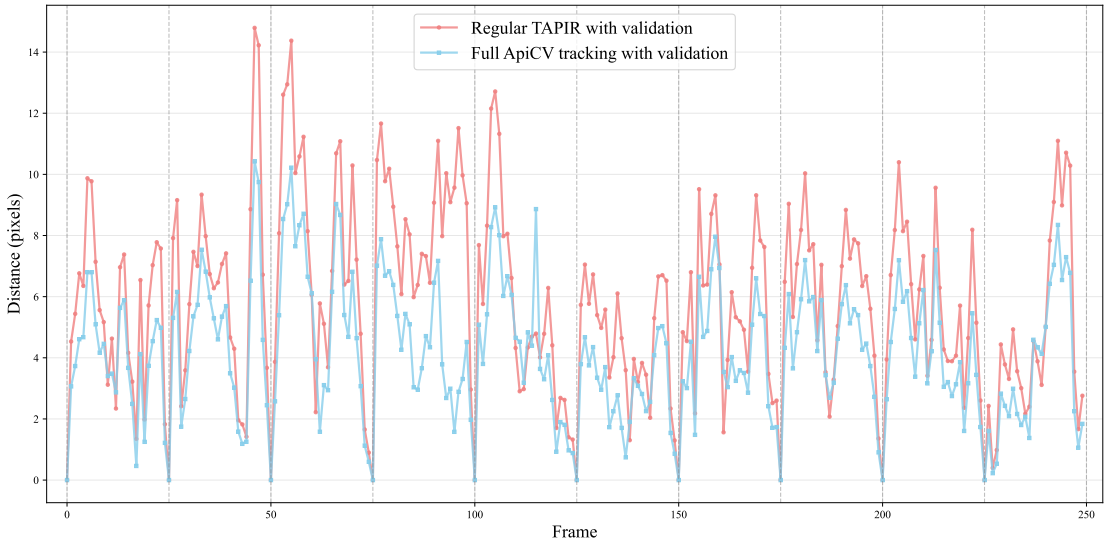


Figure 5.1: Distance Comparison for ROUND_DANCE Thorax with Smoothing

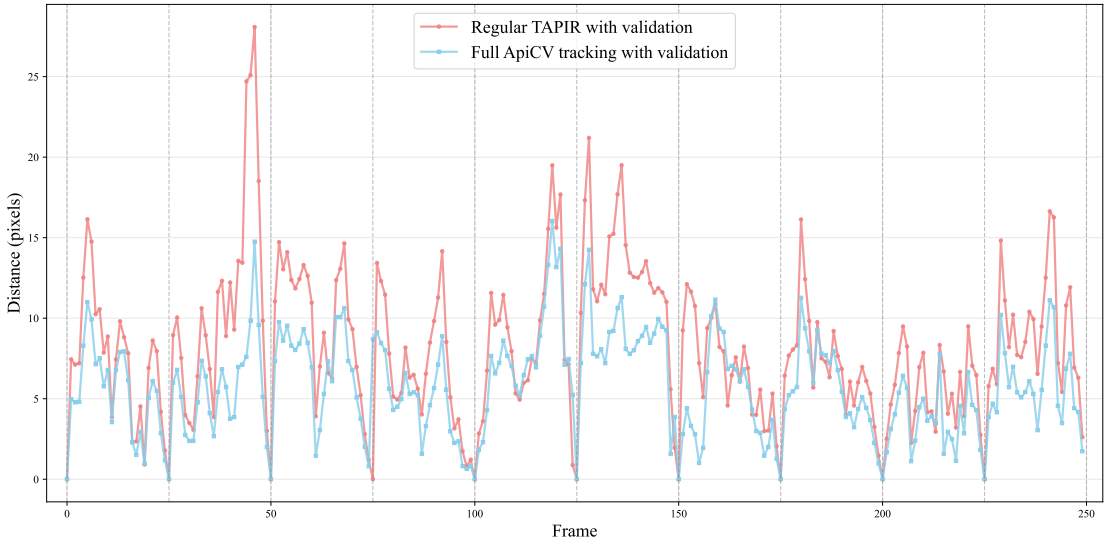
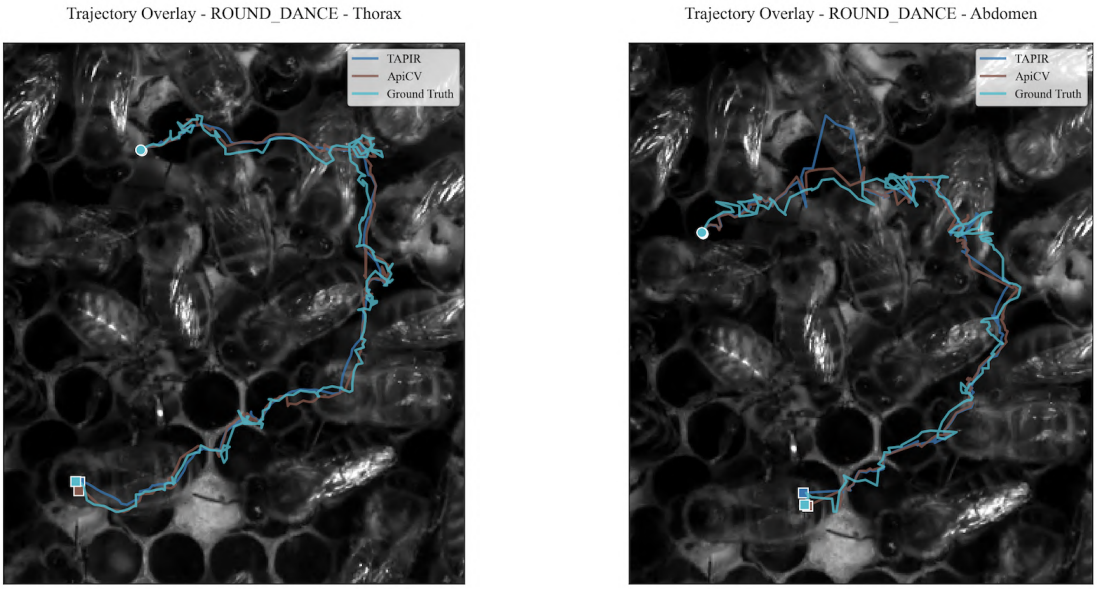


Figure 5.2: Distance Comparison for ROUND_DANCE Abdomen with Smoothing



(a) Thorax Overlay

(b) Abdomen Overlay

Figure 5.3: Trajectory Overlays of Annotations in ROUND_DANCE (with validations)

Subject	Tracking Method	Mean	% Change in Mean	Median	σ
Thorax	TAPIR	5.8329	-28.4%	5.7727	3.0919
	ApiCV	**4.1752**		**4.1486**	**2.2252**
Abdomen	TAPIR	8.2582	-29.9%	7.6965	4.7013
	ApiCV	**5.7914**		**5.6643**	**3.1232**

Table 5.1: ROUND_DANCE Ground Truth Distance Comparisons (all units in pixels)

Results - RECORDING-2023

The follower bees in RECORDING-2023 do not move much throughout their annotations. n32 and n33 make slow movements to and from the dancer they are watching, which can be seen in the increasingly worse results for regular TAPIR distance comparisons in figures 5.4 and 5.5. This is due to TAPIR not sufficiently following the bees when they do move.

Subject	Tracking Method	Mean	% Change in Mean	Median	σ
n32	TAPIR	6.6519	-53.1%	4.9672	5.2031
	ApiCV	**3.1225**		**2.9936**	**2.9588**
n33	TAPIR	10.0688	-63.9%	9.0023	7.0525
	ApiCV	**3.6366**		**3.3394**	**2.1691**
n34	TAPIR	4.3858	-20.3%	3.6089	2.8398
	ApiCV	**3.4941**		**3.2724**	**1.9644**

Table 5.2: RECORDING-2023 Ground Truth Distance Comparisons (all units in pixels)

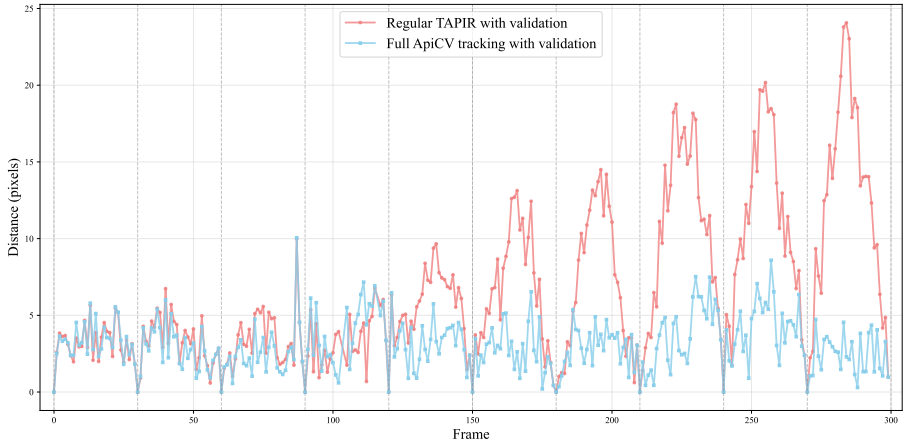


Figure 5.4: Distance Comparison for RECORDING-2023 n32 with Smoothing

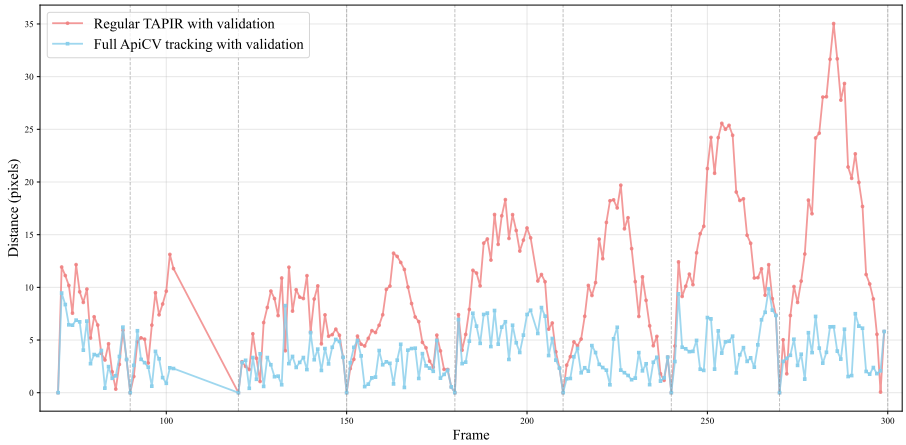


Figure 5.5: Distance Comparison for RECORDING-2023 n33 with Smoothing

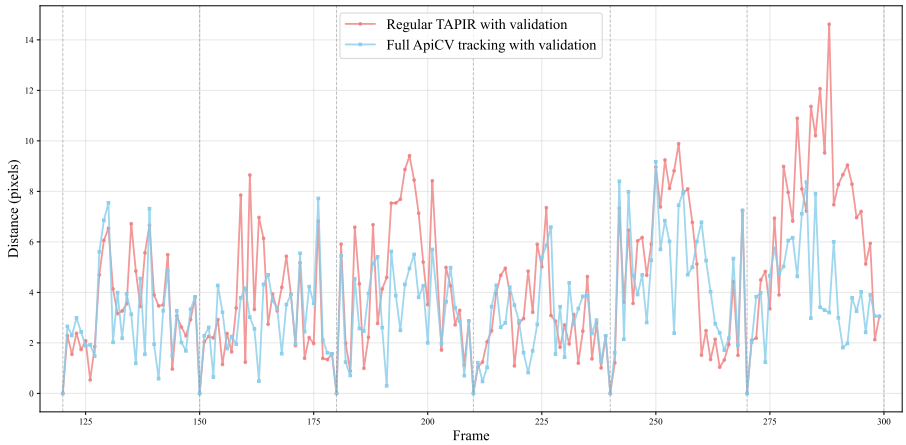
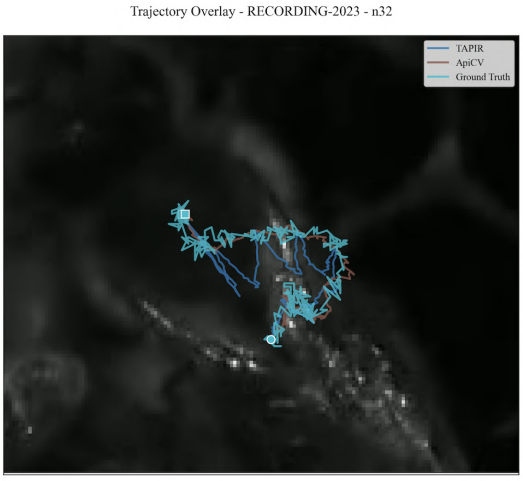
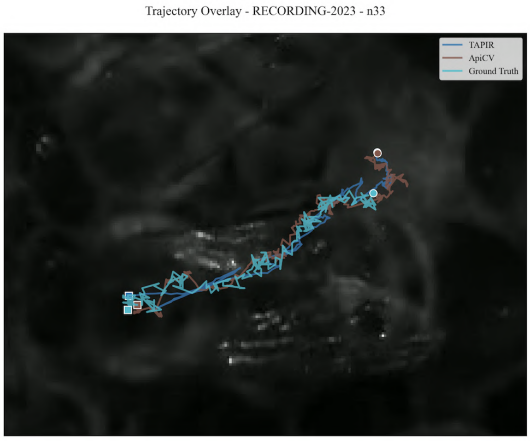


Figure 5.6: Distance Comparison for RECORDING-2023 n34 with Smoothing (Outlier Removed)

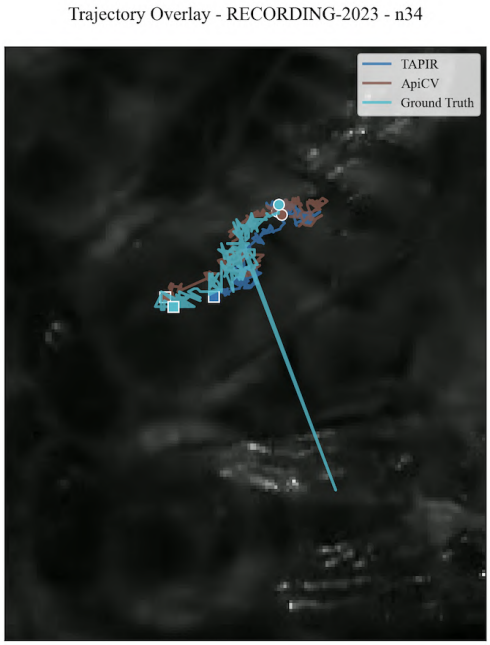
Note: An outlier in the ground truth annotation was removed from the n34 distance comparison graph (figure 5.6) to ease readability. The outlier can be qualitatively seen in the tracks overlay in the tracks overlay in figure 5.7c.



(a) n32 Trajectory Overlay



(b) n33 Trajectory Overlay



(c) n34 Trajectory Overlay

Figure 5.7: Trajectory Overlays of Annotations in RECORDING-2023 (with validations)

ApiCV, on the other hand, was much more reliable in its ability to follow the subjects' movements, demonstrated by much more stable distance comparisons. This is characterised by much lower variances across distances in table 5.2.

Intra-Segment Evaluation Conclusion

The intra-segment evaluation demonstrates that ApiCV consistently outperforms unadjusted point-tracking methods like TAPIR across all tested subjects and datasets. With mean euclidean distances reduced by 28-64% compared to TAPIR alone, ApiCV's prediction and cloud reconstruction approaches produce reliable tracks and annotations compared to the ground truth data.

Furthermore, the required human annotation of once every 1 second of video minimises the labour required to produce such annotations, especially compared to full manual tracking.

5.2 Persistency

While intra-segment evaluation demonstrates the improved accuracy ApiCV has over contemporary point-tracking methods, it is measured by enforcing validations for every 1 second of video footage.

Persistency is a measure of how long a tracking method can go *without* requesting any validation from the user.

Evaluation Metric: Euclidean Distance Between Prediction and Validation at the End of Processing

To assess ApiCV's ability to persist on top of its subjects, the euclidean distance from the final predicted location to the ground truth annotation is measured.

These are assessed with *no validations*, such that each tracking method must retain its position on the bee entirely without assistance.

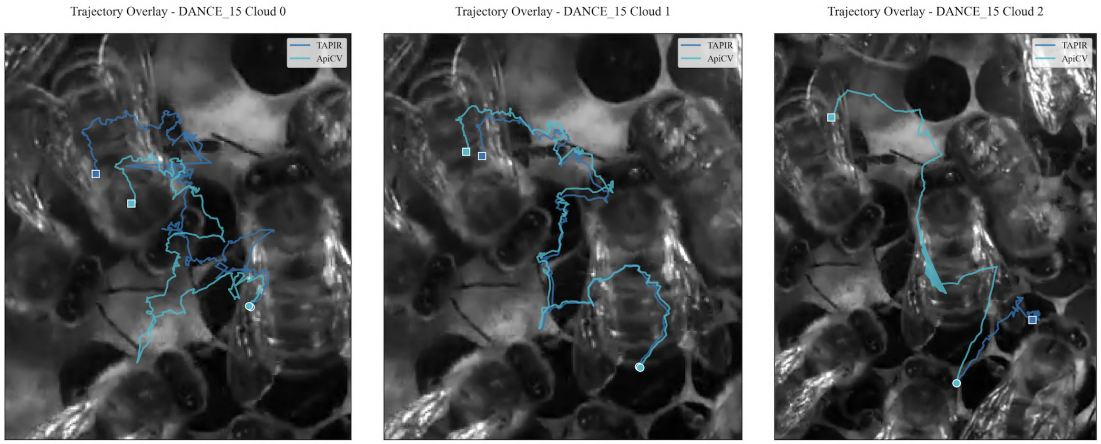
One of the videos used to assess persistency was DANCE_15 which, as mentioned in 5, did not have any ground truth annotations. Validations for the final frames were produced instead.

Results - DANCE_15

DANCE_15 was assessed for 10 seconds, focusing on a follower bee that performs multiple rotations and walks backwards. Point 5.8a was initially placed on the bee's thorax, just below its eyes. Point 5.8b was originally placed in the middle of the bee, and 5.8c on the abdomen. ApiCV outperformed TAPIR in terms of persistency in all 3 annotations.

TAPIR was still able to track the thorax (5.8a) and middle (5.8b) of the subject relatively closely. The thorax point, however, drifted closer to the subject's left wing, while the middle drifted closer to its right edge. TAPIR completely lost track of the abdomen early on in 5.8c, which is the point in the video at which the subject makes a roughly 90° turn.

ApiCV, on the other hand, not only persisted on the subject throughout the whole 10 second slice, but was also substantially closer to the final validation positions, as can be seen in table 5.3. ApiCV only drifted slightly from the subject's thorax and middle, and drifted from the abdomen onto the subject's wing.



(a) Cloud 0 (thorax) Persistency (b) Cloud 1 (middle) Persistency (c) Cloud 2 (abdomen) Persistency

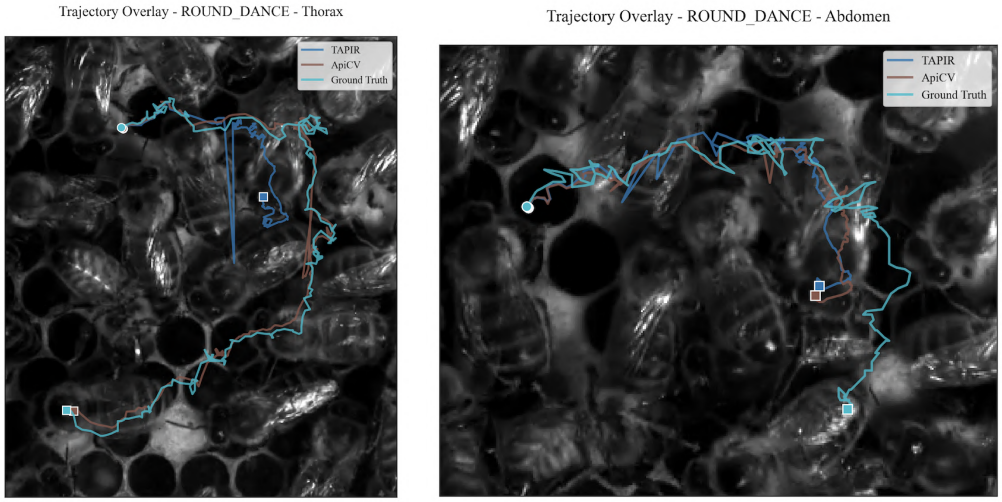
Figure 5.8: Persistencies for DANCE_15 after 10 Seconds. Initial positions are circles, and final positions are squares.

Results - ROUND_DANCE

As mentioned in the intra-segment evaluation of ROUND_DANCE, another bee causes partial occlusion over the subject's abdomen at around frame 100. Without validation, both TAPIR and ApiCV mistook the new bee for the subject, as can be seen in figure 5.9b and the distance values in table 5.3. This is an example of ApiCV's mistaken clustering (as discussed in 4.4).

Alongside just the abdomen, TAPIR lost track of the subject's thorax at about the same point, as displayed in 5.9a, and attached itself to one of the combs in the background. ApiCV, on the other hand, maintained persistency on the subject's thorax throughout the entire video slice. It had difficulty around the same frame of the video as TAPIR, but managed to maintain stability by excluding outlier points.

These results emphasize that while ApiCV improves upon standard point-tracking techniques in regard to persistency, validations are still important during moments of high uncertainty and occlusion.



(a) Thorax Persistency

(b) Abdomen Persistency

Figure 5.9: Persistency Tracks for ROUND_DANCE - no validation performed. Initial positions are circles, and final positions are squares.

Video	Subject	Tracking Method	Distance (pixels)
DANCE_15	Cloud 0	TAPIR	50.6
		ApiCV	**20.6**
	Cloud 1	TAPIR	14.03
		ApiCV	**3.9**
	Cloud 2	TAPIR	268.5
		ApiCV	**26.4**
ROUND_DANCE	Thorax	TAPIR	196.9
		ApiCV	**8.0**
	Abdomen	TAPIR	**170.2**
		ApiCV	177.3

Table 5.3: Euclidean Distances From Final Location To Correct Location

Persistency Evaluation Conclusion

Persistency evaluation demonstrates that ApiCV outperforms TAPIR in maintaining accurate tracking over extended periods *without* validation. Across both DANCE_15 and ROUND_DANCE datasets, ApiCV achieved significantly lower Euclidean distances from the correct, validated final positions. However, the abdomen tracking failure in ROUND_DANCE (177.3 pixels from the validated location) illustrates that even ApiCV is vulnerable to mistaken clustering when left completely unvalidated.

These results confirm that while ApiCV improves upon tracking persistency in honeybee hives, periodic validation remains essential for maintaining long-term accuracy.

5.3 Query Point Prediction and Cloud Reconstruction Evaluation

As discussed in 4.3, multiple methods were attempted to determine the best method of predicting a query point based on the consensus of the cloud points.

Evaluation Metric: Euclidean Distance Between Predicted and Validated Query Points

Predictions were assessed by measuring the euclidean distance between a predicted query point and the validated query point from the user. This method of evaluation is similar to the one used for intra-segment evaluation (euclidean distance from predictions to "correct" locations), but performed at the end of each segment to determine overall prediction quality.

While query point predictor techniques (4.3) and cloud reconstruction techniques (4.4) were developed separately, their evaluation ended up being highly coupled. This is due to how predictions work with fully reconstructed clouds vs. only partially reconstructed clouds. As a result, they were assessed together.

Methods Attempted

Query Point Prediction Techniques Assessed:

- **TAPIR**: Regular optical flow analysis with no modifications.
- **Average of All Predictors**: The query point is taken as the average of all predictors, regardless of weights or inliers/outliers.
- **Weighted Average of Inliers**: The query point is taken as the weighted average of all points determined to be inliers, as detailed in 4.3.

Cloud Reconstruction Techniques Assessed:

- **Full Reconstruction**: The point cloud is completely reconstructed as a circle with rotation, as if the cloud were being initialised for the first time as in 4.4.
- **Redraw Outliers + Recovery**: The final outlier redrawing and cluster recovery technique detailed in 4.4.

Query Point Evaluation Conclusion

In most instances, configuring ApiCV with the **Weighted Average of Inliers** for query point prediction, alongside the **Redraw Outliers + Recovery** method of cloud reconstruction, resulted in the strongest predictions.

It is worth noting, however, the variability of different techniques. The specific domain/video and placement of the query point was influential in which method performed best in each case. In ROUND_DANCE and OUTSIDE_FLOREA, for example, cloud points frequently fell off the subject or were occluded and became outliers. This meant points had to constantly be redrawn back onto the bee, creating higher uncertainty.

Additionally, not all methods of point-cloud-based tracking were an improvement over unadjusted tracking. The combination of *Average of All Predictors* with the *Redraw Outliers + Cluster Recovery* techniques typically generated poor results, often even worse than TAPIR on its own. This is likely due to how important weights and inliers become when only some points are redrawn, as opposed to all points, causing a simple average to yield suboptimal performance.

Overall, however, the final ApiCV configuration resulted in much closer predictions to the validation points that unadjusted TAPIR point tracking.

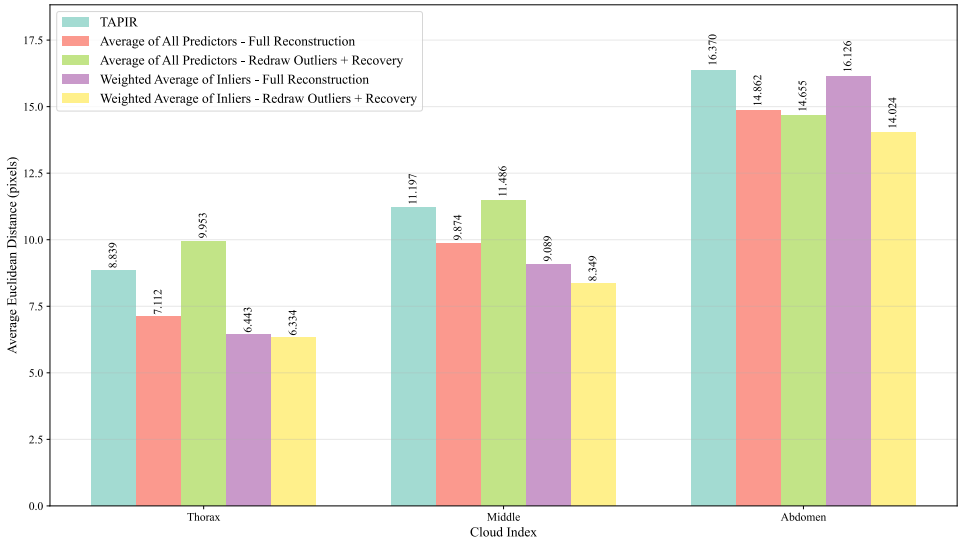


Figure 5.10: ROUND_DANCE Average Distances from Predictions to Validations

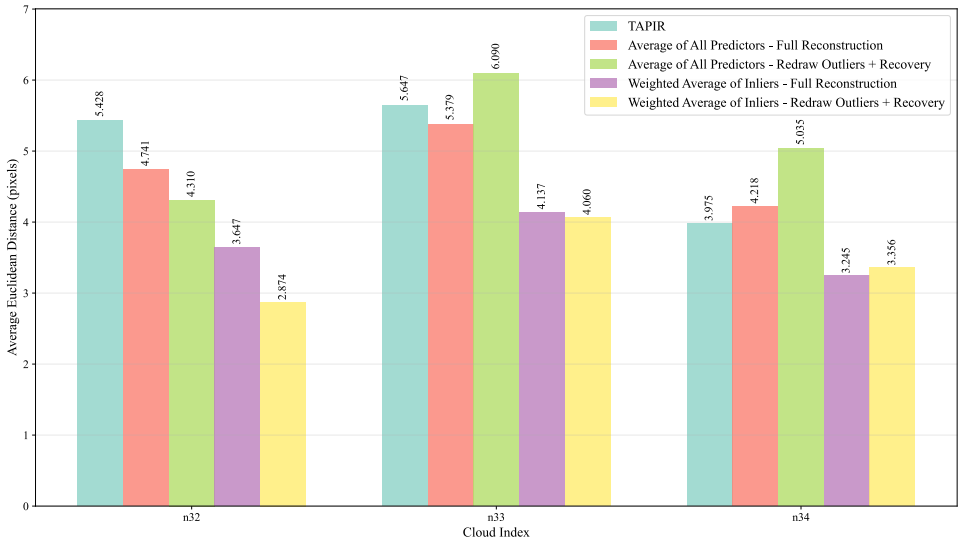


Figure 5.11: RECORDING_2023 Average Distances from Predictions to Validations

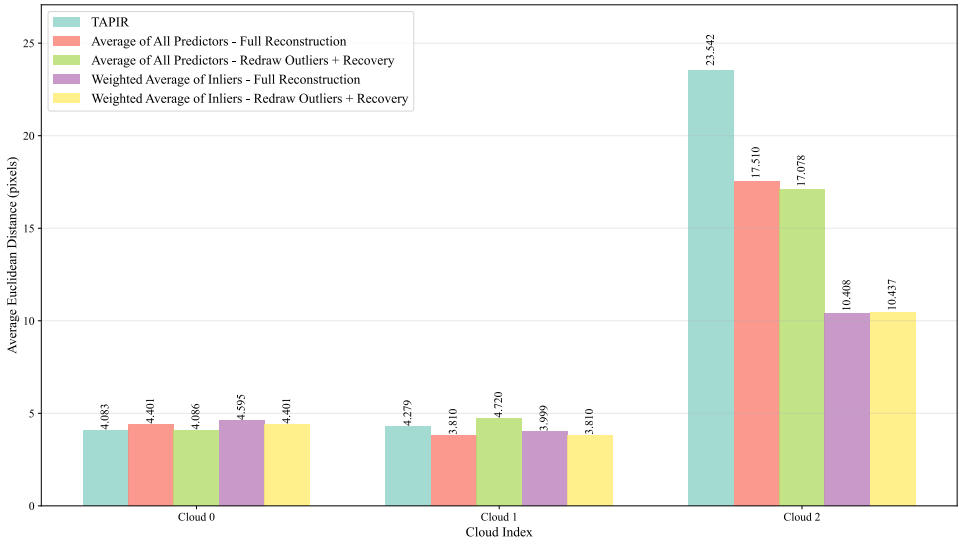


Figure 5.12: DANCE_15 Average Distances from Predictions to Validations

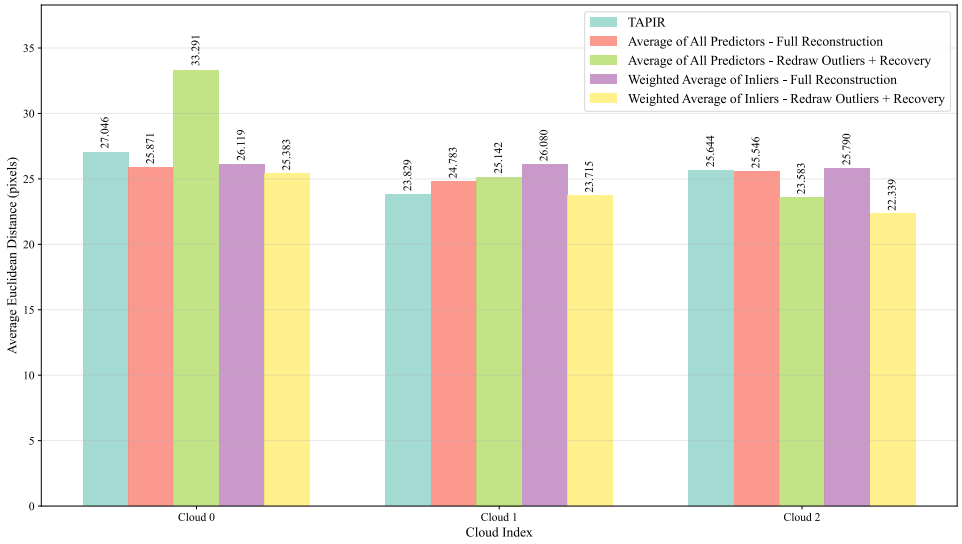


Figure 5.13: OUTSIDE_FLOREA Average Distances from Predictions to Validations

Chapter 6

Conclusion

ApiCV implements a novel new approach to point tracking and automated annotation by taking the consensus of a cloud of points, assisted by human validations, to generate robust and reliable trajectories. Notably, ApiCV requires no prior training or annotations on the dataset, distinguishing it from similar works that typically rely on trained neural networks requiring large collections of pre-existing annotations.

Evaluation presents substantial improvements over contemporary point-tracking techniques, both with and without validations, when used on honeybees in their natural hives. The absence of prior training requirements provides wide domain applicability, allowing ApiCV to be used across various domains to produce similar results with only small parameter tuning required for new datasets.

ApiCV demonstrated reliable annotation capabilities through both intra-segment and query point prediction evaluations when compared against ground truth data. The consistently small distances between predicted and validated / ground truth positions indicate that ApiCV produces stable and accurate results. Notably, the system achieved this performance while requiring human validation only once per second of video footage, meaning ApiCV handles the majority of the annotation workload independently and delivers on its primary goal of producing reliable annotations faster and more easily than manual methods.

Persistency evaluation highlighted ApiCV's significant improvement over contemporary tracking methods when requesting *no* annotation. Users can adjust the frequency of validations based on their precision requirements by modifying the Deformity Delta δ (3.1). With these optimised settings, ApiCV can operate for extended periods of time before requiring human validation, allowing users to avoid spending long periods of time producing their own annotations manually.

The transferability of ApiCV's approach extends beyond honeybee tracking to other entomological applications. The entomologists who provided the honeybee data have expressed interest in applying ApiCV to ant tracking footage, which is possible as the system is designed with domain transferability in mind.

These evaluations collectively validate ApiCV as a feasible solution for reducing manual annotation overhead, while maintaining high-quality tracking performance across a range of domains and conditions.

6.1 Limitations

Despite notable improvements in tracking honeybees in their natural environment, ApiCV has several limitations.

Necessity for Validation While ApiCV provides reliable and confident annotations during periods of simple movement, it remains dependent on user validation. Honeybees often make rapid movements that are difficult to track, resulting in frequent validation requests.

Failure Cases Unfortunately, while ApiCV performs well when tracking *follower* bees, it performs poorly when placed on a bee that is performing a waggle dance. The bee effectively "shakes off" the cloud points, which, when combined with movement, are unable to make their way back onto the bee.

Validation does not fix this issue. Even if the user places the query point back onto the bee correctly, the intra-segment predictions are very poor. ApiCV is unable to match the speed and complexity of the waggle dance movement, and functions much better on followers.

Parameter Tuning Parameters such as component selection, smoothing alpha α, DBSCAN epsilon ϵ, deformity delta δ, and optimal radius r for a given video all significantly influence performance. No dynamic or automated method exists to assist users in determining appropriate values. Users must test different parameters manually, which is time-consuming given the computational requirements of optical flow analysis.

User Interface The user interface requires substantial improvement. While manual annotation work is greatly reduced by a factor of 1/(video framerate), real-time tracking requires users to monitor validation requests, which are followed by lengthy computation times.

6.2 Future Work

Several features would substantially improve ApiCV and its tracking algorithm given additional development time.

Inter-Cloud Alignment A common occurrence involves one cloud drifting or losing track of the object while other clouds continue tracking successfully. This is typically reflected in the clouds' deformity and confidence metrics.

Currently, each cloud functions independently. However, considering inter-cloud relationships during query point prediction could yield recovery and confidence improvements. Grouping clouds to form a "skeleton" structure could allow confident clouds to readjust the positions of neighbouring clouds.

Adaptive Smoothing While the smoothing function helps prevent rapid jumps in annotations, it can still produce poor results during lateral movement. Adaptive smoothing that dynamically weights interpolation versus predictions based on movement stability could generate more reliable annotations.

Progressive Neural Network One difficulty in implementing the Incremental Neural Network for query point predictions was catastrophic forgetting: the network would over-fit to new predictions and struggle to predict anything beyond recently observed patterns.

Progressive Neural Networks [13] address this by creating new columns for each task and treating them independently. This approach could resolve the problems encountered with the INN by considering each prediction as separate but related.

Transfer Learning and Fine-Tuning Beyond implementing different neural network architectures for query point prediction, transfer learning and checkpoint fine-tuning could create more robust and adaptable tracking models. Pre-trained networks using available annotations and user validations could provide strong foundational features requiring minimal additional training to adapt to new videos.

Replacement for TAPIR As mentioned in Section 4.2, TAPIR is one point-tracking library providing a foundation for ApiCV. TAPNext [12], the successor to TAPIR, could serve as a more reliable cloud movement function g, improving overall performance.

Practical Deployment ApiCV is configured to run on two local servers. While this makes it relatively portable for users to set up on their own machines, it presents a bottleneck for researchers who prefer a more contained application. Proper practical deployment would further improve ApiCV's usability, making it more feasible as an alternative to contemporary products like DeepLabCut.

These improvements would enable ApiCV to improve tracking accuracy, and cement it as a reliable tool for point tracking and annotation production amongst researchers.

Chapter 7

Declarations

Use of Generative AI I acknowledge the use of Claude Sonnet 4 (Anthropic, https://claude.ai) to assist in generating graphs with matplotlib and debugging. I confirm that no content generated by AI has been presented as my own work.

Ethical Considerations Ethical considerations concerning this project include:

- The use of private *apis florea* video footage obtained by entomologists from the University of Edinburgh, of which approval has been granted to share images of for this report.
- The use of ApiCV to track/annotate unwilling participants. ApiCV should only be used to process and track subjects of which the user has been granted permission.
- Claiming ApiCV-produced annotated videos as intellectual property, without having intellectual property rights to the original video.

While the risk of working with animals, specifically honeybees, is not directly related to ApiCV (which is a downstream application), there is a risk in regards to users obtaining such footage. Users should not put bees (or any other subject) at risk to obtain footage for use in ApiCV.

Sustainability ApiCV has been run entirely locally, specifically on a 2022 MacBook Pro, M2 chip, requiring no use of energy-intensive high performance computing power. Initial research and background study required some use of Google Colab high performance GPUs, but never for long term computations. Computation power has further been minimised (as discussed in section 4.2) to ensure sustainability and energy efficiency.

Availability of Data and Materials Source code for ApiCV is available at `https://github.com/DWilcox02/BeeTrack`.

Most video material has been either turned into individual frames or tracks over time (such as figure 2.7c).

Supplementary material such as source videos, annotated videos, and annotation CSVs are available by following this Google Drive link (view access for anyone with the link): `https://drive.google.com/drive/folders/1YmMyPMDw5UVbK7XNhRjZPC-PplN_CnSq?usp=sharing`

Some data used for the development of this project is not publicly available (such as the OUTSIDE_FLOREA video), and has therefore not been shared.

References

[1] A. Hadjitofi and B. Webb, "Dynamic antennal positioning allows honeybee followers to decode the dance," en, *Current Biology*, vol. 34, no. 8, 1772–1779.e4, Apr. 2024.

[2] K. Bozek, L. Hebert, Y. Portugal, A. S. Mikheyev, and G. J. Stephens, "Author correction: Markerless tracking of an entire honey bee colony," *Nature Communications*, vol. 12, no. 1, p. 3121, 2021. DOI: 10.1038/s41467-021-23297-4.

[3] E. Rozenbaum, T. Shrot, H. Daltrophe, Y. Kunya, and S. Shafir, "Machine learning-based bee recognition and tracking for advancing insect behavior research," *Artificial Intelligence Review*, vol. 57, no. 9, p. 245, 2024. DOI: 10.1007/s10462-024-10879-z.

[4] K. Bozek, L. Hebert, A. S. Mikheyev, and G. J. Stephens, *Towards dense object tracking in a 2d honeybee hive*, 2017. arXiv: 1712.08324 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1712.08324.

[5] P. Kongsilp, U. Taetragool, and O. Duangphakdee, "Individual honey bee tracking in a beehive environment using deep learning and kalman filter," *Scientific Reports*, vol. 14, no. 1, p. 1061, 2024. DOI: 10.1038/s41598-023-44718-y.

[6] Xavi.borras, *RANSAC Inliers and Outliers*, Online; accessed 10 June 2025, 2014. [Online]. Available: %7Bhttps://commons.wikimedia.org/wiki/File:RANSAC_Inliers_and_Outliers.png%7D.

[7] Dllu, *Particle2dmotion*, Online; accessed 10 June 2025, 2013. [Online]. Available: %7Bhttps://commons.wikimedia.org/wiki/File:Particle2dmotion.svg#Licensing%7D.

[8] M. A.-Y. Smith, A. Easton-Calabria, T. Zhang, *et al.*, "Long-term tracking and quantification of individual behavior in bumble bee colonies," en, *Artificial Life and Robotics*, vol. 27, no. 2, pp. 401–406, May 2022.

[9] I. F. Rodriguez, J. Chan, M. Alvarez Rios, *et al.*, "Automated video monitoring of unmarked and marked honey bees at the hive entrance," *Frontiers in Computer Science*, vol. 3, p. 769 338, 2022. DOI: 10.3389/fcomp.2021.769338.

[10] A. Mathis, P. Mamidanna, K. M. Cury, *et al.*, "DeepLabCut: Markerless pose estimation of user-defined body parts with deep learning," en, *Nature Neuroscience*, vol. 21, no. 9, pp. 1281–1289, Sep. 2018.

[11] C. Doersch, Y. Yang, M. Vecerik, *et al.*, *Tapir: Tracking any point with per-frame initialization and temporal refinement*, 2023. arXiv: 2306.08637 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2306.08637.

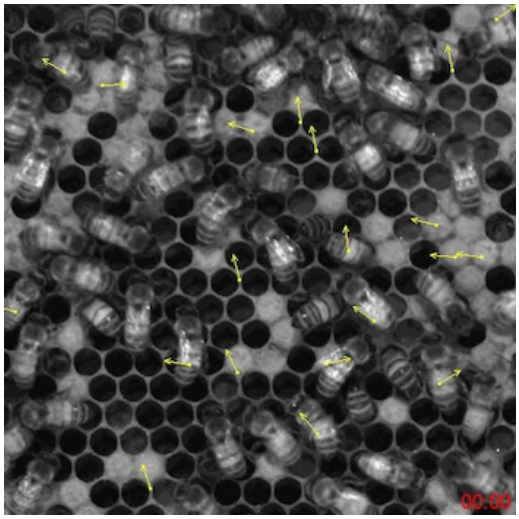
[12] A. Zholus, C. Doersch, Y. Yang, *et al.*, "Tapnext: Tracking any point (tap) as next token prediction," 2025. arXiv: 2504.05579 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2504.05579.

[13] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, *et al.*, "Progressive neural networks," 2022. arXiv: 1606.04671 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1606.04671.

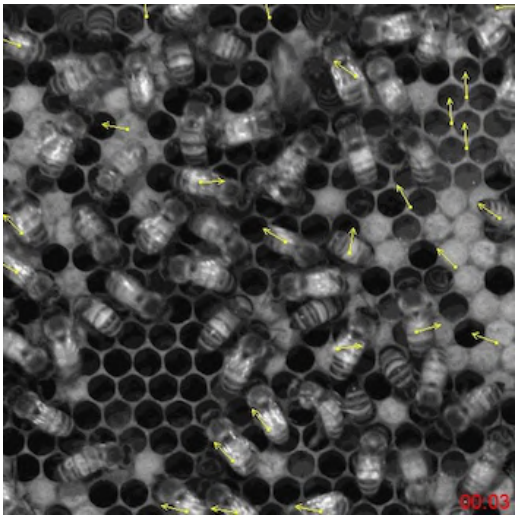
Appendix A

Appendix

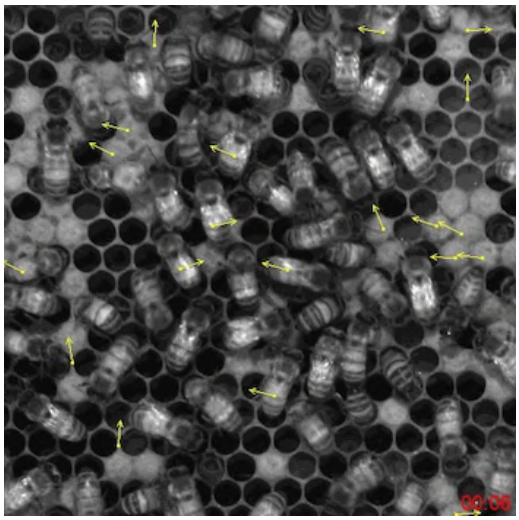
Figure A.1: Example of Kongsilp et al.'s polygon labelling [5]



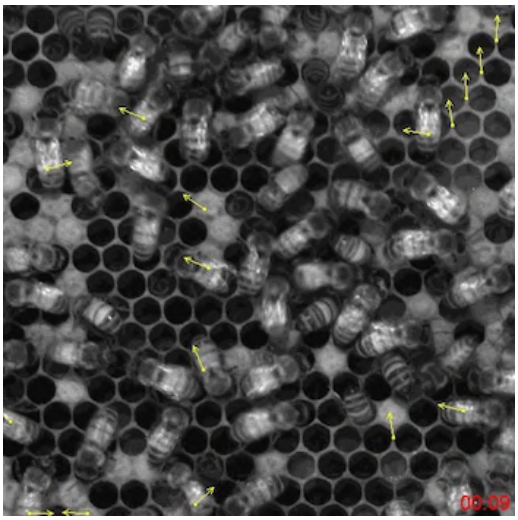
(a) 0 seconds / frame 1



(b) 3 seconds / frame 31

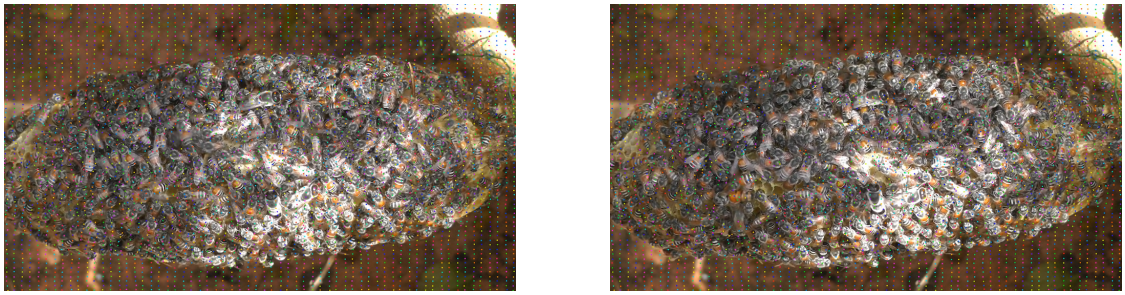


(c) 6 seconds / frame 61



(d) 9 seconds / frame 91

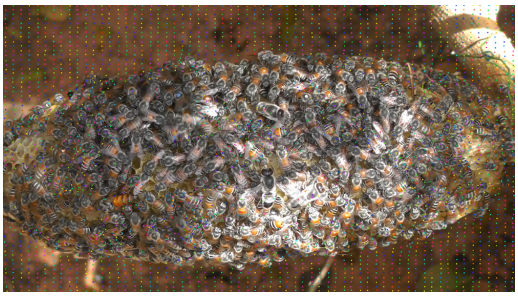
Figure A.2: Attempted immediated use of Bozek et al.'s markerless tracking method



(a) 0 seconds

(b) 1 second

(c) 2 seconds



(d) 3 seconds

Figure A.3: Rainbow Visualization on *apis florea* over 4 seconds.

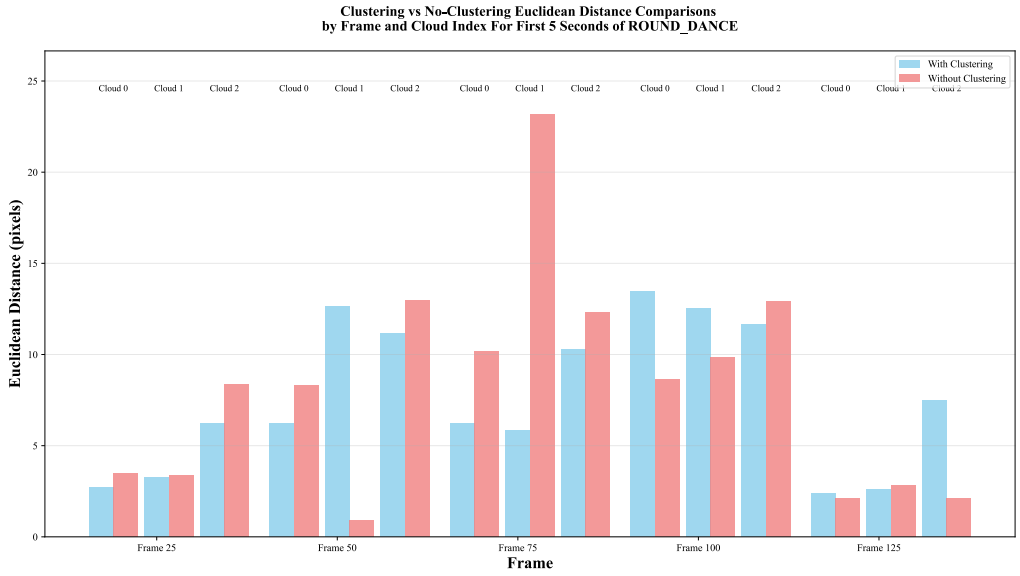


Figure A.4: Euclidean distance from **predicted query points** to **validated query points** for ROUND_DANCE video. The mean distances were 8.1px *without* clustering, but 7.65px *with* clustering.

```
Frame 0: [
    {"x":416.1,"y":299.1,"color":"red","radius":32},
    {"x":448.1,"y":345.1,"color":"green","radius":32},
    {"x":450.1,"y":393.1,"color":"blue","radius":32}
],
Frame 15: [
    {"x":402,"y":284.7,"color":"red","radius":32},
    {"x":449,"y":286.7,"color":"green","radius":32},
    {"x":488,"y":299.7,"color":"blue","radius":32}
],
Frame 30: [
    {"x":352,"y":317.7,"color":"red","radius":32},
    {"x":397,"y":304.7,"color":"green","radius":32},
    {"x":439,"y":296.7,"color":"blue","radius":32}
],
Frame 45: [
    {"x":343,"y":307.7,"color":"red","radius":32},
    {"x":375,"y":268.7,"color":"green","radius":32},
    {"x":408,"y":245.7,"color":"blue","radius":32}
],
Frame 60: [
    {"x":359.1,"y":275.2,"color":"red","radius":32},
    {"x":391.9,"y":228.6,"color":"green","radius":32},
    {"x":419.7,"y":202.6,"color":"blue","radius":32}
],
Frame 75: [
    {"x":380.1,"y":244.2,"color":"red","radius":32},
    {"x":407.5,"y":197,"color":"green","radius":32},
    {"x":435.9,"y":168.3,"color":"blue","radius":32}
]
```


Figure A.5: Validations for DANCE_15 evaluation at 15fps. Validation points were identical in every evaluation test


```
{
    "alpha": 3.0,
    "epsilon": 0.75,
    "delta": 0.01,
    "radius": 20.0
}
```


Figure A.6: Parameters Used for DANCE_15 Evaluation

```
Frame 0: [
    {"x":301,"y":227,"color":"red","radius":8},
    {"x":283.7,"y":237.7,"color":"green","radius":8},
    {"x":251,"y":260,"color":"blue","radius":8}
],
Frame 25: [
    {"x":343,"y":209,"color":"red","radius":8},
    {"x":319.7,"y":220.2,"color":"green","radius":8},
    {"x":287,"y":234,"color":"blue","radius":8}
],
Frame 50: [
    {"x":433,"y":227,"color":"red","radius":8},
    {"x":395.1,"y":224.1,"color":"green","radius":8},
    {"x":362,"y":230,"color":"blue","radius":8}
],
Frame 75: [
    {"x":454,"y":219,"color":"red","radius":8},
    {"x":427,"y":221,"color":"green","radius":8},
    {"x":391,"y":222,"color":"blue","radius":8}
],
Frame 100: [
    {"x":456,"y":293,"color":"red","radius":8},
    {"x":442,"y":265,"color":"green","radius":8},
    {"x":418,"y":236,"color":"blue","radius":8}
],
Frame 125: [
    {"x":459,"y":326,"color":"red","radius":8},
    {"x":453,"y":290,"color":"green","radius":8},
    {"x":427,"y":265,"color":"blue","radius":8}
],
Frame 150: [
    {"x":454,"y":358,"color":"red","radius":8},
    {"x":463,"y":330,"color":"green","radius":8},
    {"x":459,"y":293,"color":"blue","radius":8}
],
Frame 175: [
    {"x":402,"y":397,"color":"red","radius":8},
    {"x":432,"y":368,"color":"green","radius":8},
    {"x":453,"y":344,"color":"blue","radius":8}
],
Frame 200: [
    {"x":372,"y":413,"color":"red","radius":8},
    {"x":401,"y":401,"color":"green","radius":8},
    {"x":435,"y":376,"color":"blue","radius":8}
],
Frame 225: [
    {"x":325,"y":458,"color":"red","radius":8},
    {"x":347,"y":437,"color":"green","radius":8},
    {"x":377,"y":413,"color":"blue","radius":8}
],
Frame 250: [
    {"x":256,"y":457,"color":"red","radius":8},
    {"x":295,"y":456,"color":"green","radius":8},
    {"x":322,"y":448,"color":"blue","radius":8}
]
```


Figure A.7: Validations for ROUND_DANCE evaluation at 25fps. Validation points were identical in every evaluation test

```
{
    "alpha": 4.0,
    "epsilon": 1.0,
    "delta": 0.01,
    "radius": 8.0
}
```


Figure A.8: Parameters Used for ROUND_DANCE Evaluation

```
Frame 0: [
    {"x":800,"y":1039,"color":"red","radius":24},
    {"x":890.7,"y":849.9,"color":"green","radius":24},
    {"x":824.9,"y":806.7,"color":"blue","radius":24}
],
Frame 30: [
    {"x":806,"y":1028,"color":"red","radius":24},
    {"x":894.9,"y":854.2,"color":"green","radius":24},
    {"x":829,"y":804.5,"color":"blue","radius":24}
],
Frame 60: [
    {"x":806,"y":1029,"color":"red","radius":24},
    {"x":894.9,"y":865,"color":"green","radius":24},
    {"x":835.2,"y":806.7,"color":"blue","radius":24}
],
Frame 90: [
    {"x":813,"y":1034,"color":"red","radius":24},
    {"x":880,"y":872,"color":"green","radius":24},
    {"x":835.2,"y":806.7,"color":"blue","radius":24}
],
Frame 120: [
    {"x":816,"y":1023,"color":"red","radius":24},
    {"x":871,"y":870,"color":"green","radius":24},
    {"x":825,"y":804,"color":"blue","radius":24}
],
Frame 150: [
    {"x":820,"y":1011,"color":"red","radius":24},
    {"x":861,"y":879,"color":"green","radius":24},
    {"x":815,"y":811,"color":"blue","radius":24}
],
Frame 180: [
    {"x":808,"y":1006,"color":"red","radius":24},
    {"x":851,"y":892,"color":"green","radius":24},
    {"x":808,"y":822,"color":"blue","radius":24}
],
Frame 210: [
    {"x":795,"y":1006,"color":"red","radius":24},
    {"x":841,"y":895,"color":"green","radius":24},
    {"x":812,"y":819,"color":"blue","radius":24}
],
Frame 240: [
    {"x":783,"y":1012,"color":"red","radius":24},
    {"x":816,"y":907,"color":"green","radius":24},
    {"x":808,"y":825,"color":"blue","radius":24}
],
Frame 270: [
    {"x":774,"y":1003,"color":"red","radius":24},
    {"x":797,"y":916,"color":"green","radius":24},
    {"x":792,"y":833,"color":"blue","radius":24}
],
Frame 300: [
    {"x":773,"y":1001,"color":"red","radius":24},
    {"x":786,"y":910,"color":"green","radius":24},
    {"x":787,"y":837,"color":"blue","radius":24}
]
```


Figure A.9: Validations for RECORDING-2023 evaluation at 30fps. Validation points were identical in every evaluation test

```
    {
        "alpha": 5.0,
        "epsilon": 1.0,
        "delta": 0.01,
        "radius": 24.0
    }
```


Figure A.10: Parameters Used for ROUND_DANCE Evaluation


```
Frame 0: [
    {"x":1017.8,"y":638.1,"color":"red","radius":20},
    {"x":1061,"y":675.9,"color":"green","radius":20},
    {"x":1036.3,"y":658.9,"color":"blue","radius":20}
],
Frame 15: [
    {"x":1003.9,"y":644.8,"color":"red","radius":20},
    {"x":1045,"y":683.4,"color":"green","radius":20},
    {"x":1025.8,"y":663.3,"color":"blue","radius":20}
],
Frame 30: [
    {"x":957.3,"y":610.9,"color":"red","radius":20},
    {"x":1001.1,"y":641.7,"color":"green","radius":20},
    {"x":979.2,"y":627.8,"color":"blue","radius":20}
],
Frame 45: [
    {"x":913.4,"y":560,"color":"red","radius":20},
    {"x":949,"y":593.9,"color":"green","radius":20},
    {"x":932.6,"y":580,"color":"blue","radius":20}
],
Frame 60: [
    {"x":899.7,"y":541.4,"color":"red","radius":20},
    {"x":921.6,"y":586.2,"color":"green","radius":20},
    {"x":910.6,"y":560,"color":"blue","radius":20}
],
Frame 75: [
    {"x":864,"y":513.7,"color":"red","radius":20},
    {"x":896.9,"y":563,"color":"green","radius":20},
    {"x":880.5,"y":536.8,"color":"blue","radius":20}
]
```


Figure A.11: Validations for OUTSIDE_FLOREA evaluation at 15fps. Validation points were identical in every evaluation test


```
    {
        "alpha": 1.0,
        "epsilon": 1.0,
        "delta": 0.05,
        "radius": 20.0
    }
```


Figure A.12: Parameters Used for OUTSIDE_FLOREA Evaluation