A BEGINNING-TO-END SYSTEM FOR EFFICIENTLY GATHERING TRACKING DATA ON MULTIPLE TARGETS

by

Lance Andrew Rice

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Computer Science

Charlotte

2016

ABSTRACT

LANCE ANDREW RICE. A beginning-to-end system for efficiently gathering tracking data on multiple targets. (Under the direction of DR. MIN C. SHIN)

Multi-object tracking in video has the potential for broad applicability, from analyzing division-of-labor within biological networks such as insects, to surveillance and traffic monitoring. Despite any potential the field of multiple object tracking holds, it often remains passed over in favor of manual annotation methods due to difficulties in reliably obtaining accurate results. Existing tools for gathering tracking data are either very time consuming, inaccessible due to non-intuitive interfaces, or simply produce too many tracking failures with no means of correcting them. In this thesis, we address the multi-object tracking task that is ill served by existing tracking tools. Specifically, we discuss intuitive means of parameter tuning, training various subroutines of the tracking algorithm, and correcting tracking failures. To this end, we developed a new desktop application, ABCTracker, which provides a sequence of efficient steps the direct the user from beginning (the video) to the end (accurate tracking information).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Due to the availability of high quality and inexpensive video cameras, as well as the convenience of high-powered computers, an increase in demand for automated object tracking has emerged in several domains. Accurate object tracking serves as a catalyst for researchers like biologists to understand collective decision making and division-of-labor within biological networks such as insects [13, 17, 19, 18] and enables various applications in areas like surveillance [4], proxemics [21], and traffic monitoring [1]. Manually measuring the location and interactions of objects in these fields is a time-consuming process and can be an inhibiting factor towards comprehensive analysis. Efforts have been made in recent years to meet such demands through advancements in automated tracking algorithms [24, 20] as well as software applications such as video annotation tools [22, 3, 25] and systems designed to track multiple objects given varying levels of user input [12, 2, 23]. We examine the current state of available video annotation and tracking software and discuss some of the limitations these tools have. A new end-to-end tracking system, ABC-Tracker, is presented which attempts to address the shortcomings of existing tools for multi-object tracking.

1.1 The Problem

During a users examination process, the objects within the video will exhibit behaviors which the observer will want to understand. For example, imagine a biologist monitoring the response within an ant colony to some stimulus, such as a new food source or foreign ants of another species, and is intrigued by an abrupt change in motion among the colony. This behavior shift will have multiple characteristics – the shift could trigger rapidly across the colony or gradually spread from the stimulus' location; portions of the colony could move toward or away from the stimulus at specific rates; or the colony members may aggregate around particular members in response – are a few examples, all of which can be quantitatively measured with frame-by-frame locations for object's of interest in the scene. The visual apparentness of such characteristics can vary, and the experiment will need to be repeated multiple times against control groups to generate statistically sound hypothesis models. The challenge for the biologist then becomes determining an appropriate means of collecting tracking information for multiple video recordings.

While it may be possible for the biologist to find some source code for an automated tracking algorithm, any novice to the field of multi-object tracking will soon discover that simple tracking algorithms fail to handle scenes with even minor levels object occlusion or similar appearances. More complex algorithms that better handle complex scenes have many parameters that must be tuned and/or learned models that must be trained, potentially for each video to be processed. An alternate approach for the biologist is to use one or more of the video annotation tools available.

Some video annotation tools allow the user to define each objects location in all or a subset of the video frames and apply interpolation for filling gaps. Videos containing either many objects, objects with erratic/fast motion, or both significantly impacts the time required to annotate the video. A more detailed overview of available video annotation tools is discussed in section 2.1.

The option any biologist would be most hopeful will work is to download and apply an object tracking system. Unfortunately, currently available multi-object tracking systems suffer from one or more of the following:

1. Was designed for a specific type of object and sometimes hold very particular assumptions on how the objects move or were recorded.

2. Must have each step of the tracking algorithm defined by the user and in even more situations requires parameter values of the tracking steps to be directly defined.

3. Difficult to install without sufficient technical background.

4. Interface design is unintuitive and lacks helpful features like contextually aware guidance or even in-app instructions.

5. Produces results containing tracking failures with either no means of correcting them, very specific assumptions in correction design (e.g. distinct markings for every object), or requires additional parameters to be defined.

The ability to correct tracking failures is rather important as failures hinder the reliability of information gathered on the objects. For example, ID switches (i.e.

when a track covering one object switches to a different object later in the video) will incorrectly represent the movement patterns, behavior, and interactions of two separate objects as all coming from a single target.

This problem illustration is not unique, and analogous ones could be drawn from different fields of investigation that could benefit from reliable tracking. Each of which would face similar difficulties with the tools available for obtaining them.

1.2 A Solution Sketch

Developing a general purpose system that can produce accurate tracking results efficiently with respect to user time and effort depends on how the system gathers information from the user to adjust the tracking algorithms, the tracking algorithms used to produce the tracking output and the mechanisms available for correcting any tracking failures. We believe a solution would need to facilitate the following when addressing the problem described above:

- Intuitive Means of Algorithm Tuning

  Of the several varieties of automated tracking algorithms, many are formulated as a pipeline consisting of a sequence of subroutines (e.g. pre-processing images, detection, low-level association, filtering, etc...). Each subroutine has a set of parameters that impact not only its performance but also the performance any subroutines that follow. The system should allow someone with no technical knowledge of the how the tracking algorithm works to tune all subroutines. In our design, we formulate parameter tuning and model training as a series of simple questions; the user does not need to define the arrangement of subroutines or the value of parameters directly (e.g. ForegroundThreshold = 0.9). Details on the proposed approach to parameter tuning are presented in section 3.2.

- Generalized Tracking Algorithm

  The selection of subroutines withing a tracking pipeline affects several things, including: how and what parameters are tuned by the user, the tracking accuracy of the algorithm, and the expected frequency of each tracking error type in the output. Knowing that one error type is more difficult to locate and correct than another should guide the tracking algorithm design towards reducing the overall user effort during correction. Details of the proposed tracking algorithm are discussed in section 3.2.

- Error Localization and Correction Features

  The ability to correct failures made by the tracking algorithm is paramount because error free results being produced by general purpose tracking algorithms is currently unrealistic. The system should allow the user to make nearly any correction possible and provide features for efficiently locating tracking failures. In our design, we provide two modes of correction:

  1. A sandbox mode which allows the user to visualize the current state of the corrected tracking results in several ways and provides operations for correcting every possible tracking failure.

  2. A guided correction mode which automatically finds potential errors and

presents solutions as multiple choice questions.

Details of the guided correction mode and the correction operations available are discussed in section 3.3.

CHAPTER 2: BACKGROUND

The history of automatically detecting and maintaining an object's location within a sequence of images is a surprisingly long one, spanning nearly three decades. During this time, the problem has become well studied among the computer vision community and accumulated a vast amount of literature. Here we provide only a very brief overview of the topic.

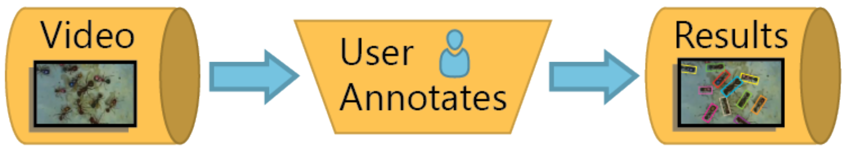
Initial efforts towards automated object tracking focused mainly on sequential, recursive algorithms which estimate the current location of a target using information from earlier frames. Particle filtering based tracking is one example and formulates the problem as a sequential Monte Carlo sampling from a proposal distribution. Samples from the proposal distribution, signifying the particles, are weighted based on a number factors (e.g. motion smoothness, appearance modeling) and collectively represent the estimated current state of the target object. Early particle filter approaches had moderate success in the case of single object tracking but are vulnerable to drifting failures in long video sequences or scenes with multiple interacting objects. A detailed survey of particle filter tracking methods is presented in [16].

The counterpart to sequential, recursive tracking algorithms is batch approaches. Batch algorithms, sometimes called global data association or tracking-by-detection based methods, express object tracking as an optimization problem over longer intervals of time. Global inference on association probabilities between detection responses in multiple frames is incrementally performed to assemble object trajectory fragments (tracklets). Batch approaches have the benefit of improved robustness against drifting, the possibility of recovering from tracking failures, and have shown to perform well in multi-target situations where there is no need for a real-time solution.

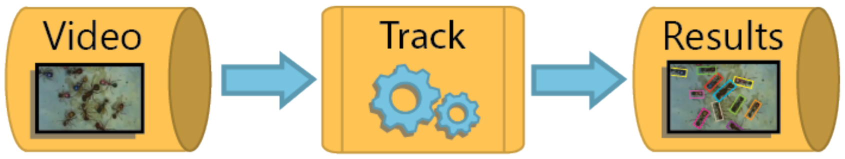
Compared to tracking a single target, multi-target tracking is significantly more complicated. Similar appearances between different objects, crowded scenes, and inter-object occlusions among a sometimes unknown and/or fluctuating number of objects are all additional difficulties in multi-object scenarios.

2.1 Video Annotation and Tracking Systems

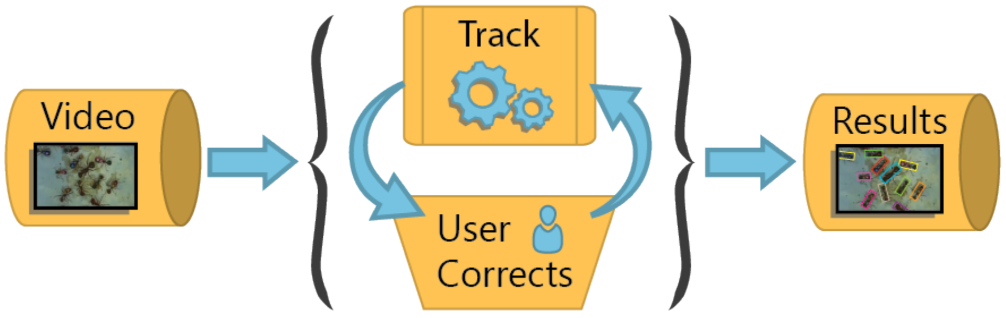
The Vatic system (Video Annotation Tool from Irvine California) [22] is related to this work in several ways. Vatic was originally developed to enable its users to annotate tracking locations for multiple objects in a video (as illustrated in figure 1a). It has an intuitive and simple interface design that allows creating, removing and updating object annotations at frame level accuracy. Once an object is initialized and had two or more locations defined in separate frames, linear interpolation is used by the system to fill in the gaps. This simple approach is much more efficient for the user than labeling frame-by-frame. Simple attributes for the defined objects can also be specified. Additionally, the Vatic system features crowd-sourcing the annotation work through Amazon Mechanical Turk as well as ways of comparing annotation results.



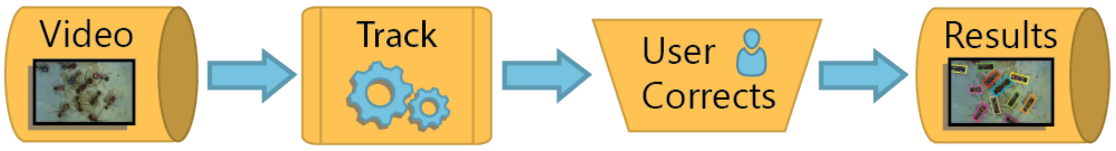
(a) Manual annotation



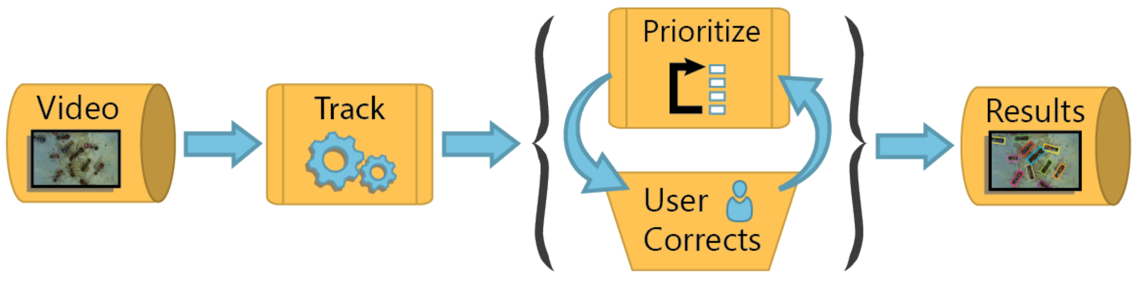
(b) Fully-automated



(c) Semi-automated



(d) Automated with post-tracking correction



(e) The proposed system

Figure 1: Abstract flow diagrams representing the different types of tools available for collecting tracking information. (a) Manual annotation systems [22, 25], (b) fully-automated tracking systems [3, 12], (c) semi-automated systems [23, 15]. (d) Automated Tracking Systems with post-tracking user correction [2, 14], and (e) the proposed automated tracking system.

The Vatic project allows for extensions to the system, one of which attempts to incorporate semi-automated tracking and active learning while interacting with the user. The tracking extension builds appearance models for each object using an SVM classifier with HOG features from user-annotated frames. Visual tracking is performed using the trained appearance models to interpolated missing information between user annotated frames. Active learning determines query frames to be annotated by the user by estimating expected label change on the predicted object trajectories [23]. A slight deterrent of Vatic comes from the fact that it was designed to be deployed as a web service and getting it to run on the desktop (or the web) is a bit involved without technical experience. Similar to the Vatic system is LabelMeVideo [25], which differs slightly by allowing for any polygon-shaped object annotation where Vatic only supports bounding boxes without orientation control.

A recent development called AntCounter [3] was designed to track leaf cutter ants in a scene for the primary purpose of counting the ants as they move in two opposing directions along a single trail. The system uses a simple tracking algorithm which matches nearest foreground blobs that are less than 10 pixels apart between frames. The system is sensitive to the size of the ants within the image, suggesting image scaling to get the tracker working appropriately, and states that it is designed specifically for tracking leaf cutter ants moving in two opposing directions.

Developers of SwissTrack [12] sought to design not only an interface for aiding in tuning parameters of tracking algorithms but also provide a flexible suite of tools for imaging, interfacing with the application, and defining new trackers. Through built-in and community contributed tracking subroutines, which they call components, users can either create or combine existing components in an attempt to fit their tracking needs. Components within the SwissTrack application require tuning, possibly on a per video basis for some components and must be performed by defining values for each parameter directly. In this work, we also modularize tracking subroutines for a more flexible and reusable means of defining trackers but seek to abstract direct parameter tuning as easy to understand questions.

Another tracking system that requires directly tuning parameters is CTrax [2]. CTrax is intended for automatically tracking multiple walking flies. In addition to tracking, the system provides a collection of Matlab toolboxes for social behavior detection and correcting tracking failures (the FixErrors GUI). After the user tunes the parameters for tracking and allows it to run, they then must define additional parameters in the FixErrors tool for assisting the user in locating tracking failures. Similar to this work, we provide a set of correction operations (e.g. add, remove, join) and assist in failure localization but do not require any parameters to be tuned during correction.

CHAPTER 3: SYSTEM DESIGN

The ABCTracker application is the result of several prototype iterations. Testing and feedback were provided by fellow lab members, biologists, and biology students at each stage of development. This incremental feedback was crucial for maintaining appropriate design heuristics towards:

1. Simplicity: of interface interactions and tuning algorithm parameters.
2. Efficiency: concerning user time spent interacting with the application.
3. Accuracy: the user should be able to reach their desired levels of accuracy with the features supplied by the application.

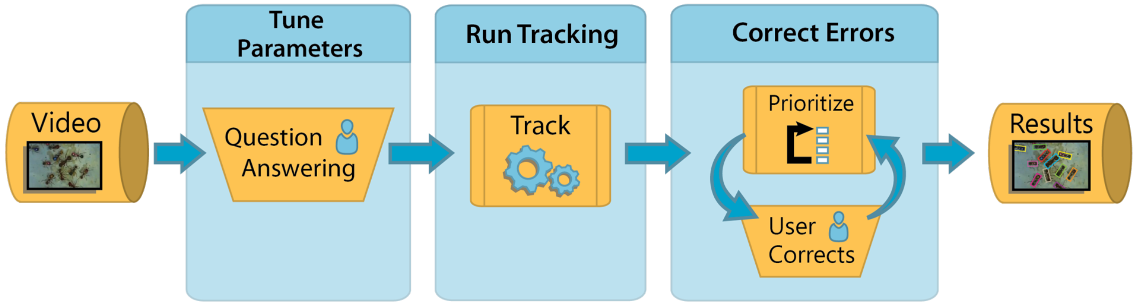


Figure 2: This abstract flow diagram outlines our approach for collecting tracking data on multiple targets within an input video. First, a small set of simple questions is asked of the user. These answers are used to tune many parameters of the tracking algorithm; this automatic tuning takes place during the initial step of the tracking phase. After applying the tuned tracker to the video, the user is directed to the error correction phase. To reduce the user's overall time interacting with the system we define two interaction phases with an extended period in between that allows the tracker to be automatically tuned and applied to the video.

In forging a solution to the problem presented in section 1.1 one must consider that the three fundamental functions of an end-to-end multi-object tracking system: tuning, tracking, and correcting are very interdependent from both the perspective of user effort as well as overall tracking performance (figure 3). Parameter tuning by the user must be as simple and efficient as possible but also gather a proper amount of information for the tracker to perform well, which in turn reduces user effort during correction. Additionally, the tracker should exploit the fact that particular tracking failures are more challenging and time-consuming to find and correct than other failure types. The extent of these interdependencies amplified the importance of incremental prototyping and guided a majority of the design decisions.

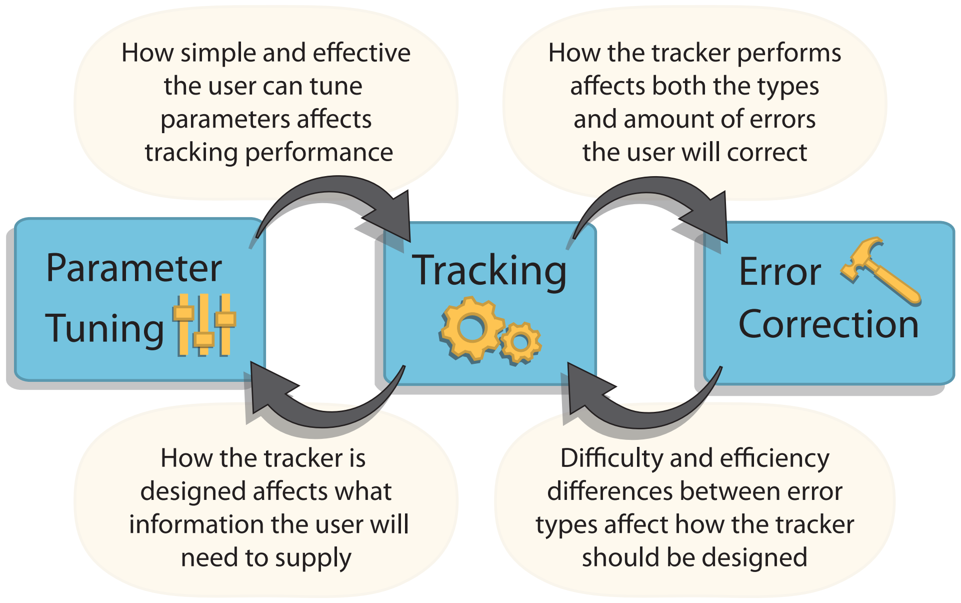

How simple and effective the user can tune parameters affects tracking performance
How the tracker performs affects both the types and amount of errors the user will correct
Parameter Tuning
Tracking
Error Correction
How the tracker is designed affects what information the user will need to supply
Difficulty and efficiency differences between error types affect how the tracker should be designed

Figure 3: An illustration of the three primary functions of the tracking system and the influences (arrows) they have on one another.

3.1 Overview of System Architecture

At the highest level of abstraction, the system is divided into two subsystems: the backend and the frontend. The backend is written primarily in Matlab and acts as a local server to the desktop application, receiving and processing frontend requests. The frontend, written in Java, serves as a client to the backend, handles presenting the desktop interface and sends processed user inputs as requests to the backend. Local TCP ports facilitate communication between the backend and frontend. Large portions of information (e.g. an entire set of tracking results) and data possibly needing to be accessed at a later date are stored as JSON files on the user's hard drive.

Application of the tracking algorithm to a video sequence is handled entirely by the backend. Because a majority of current tracking algorithms are composed of a series of tracking subroutines, we model trackers as a set of tracking "modules" that collectively represent the subroutines of a tracking algorithm. Modeling trackers in this way enable greater flexibility and encourages separation between properties of the object tracks and operations to perform on them. Three types of modules are defined:

1. Detection modules: compute object detections in video frames

2. Association modules: generate object trajectories

3. Miscellaneous modules: subroutines that do not pertain to either of the prior module types (e.g. pre-processing or filtering subroutines, a data structure(s)

building subroutine, ...)

Each module within a tracker has a corresponding set of parameters which are automatically tuned by the user input processing module (miscellaneous module type) that is present in every tracker's module set definition. As new tracking modules are implemented, they can easily be incorporated into existing tracking pipelines or form entirely new trackers with little or no effect on the backend or interface.

A tracking algorithm as we have defined above cannot perform its purpose without additional information, specifically what video to track, user inputs for tuning module parameters and storage location definitions for tracking progress made and the current state of the tracking results. Such information is defined in what we call a tracking "process". Throughout the remainder of the thesis we will refer to a "process" as a user named and initiated data structure that contains: what tracker will be used (i.e. module set), the video to track, the current state of progress made towards final results (e.g. initialized, awaiting tracking, awaiting correction, complete), and all user inputs to questions and corrections made through the application's interface.

Currently, all modules implemented in the systems can be effectively tuned by the user answering three simple questions. Section 3.2 concentrates on the application's primary tracking algorithm, the "default" tracker which is the most generalized tracking algorithm within the system and achieves the highest tracking performance given no additional information outside the user's input (i.e. no offline trained models). The final functional requirement of the system is correcting tracking failures which are discussed in section 3.3.

3.2 Tracking Algorithm

Most of the tracking parameters are automatically tuned based on the user's answer to one of the three questions asked during initialization of a tracking process, the "where are the objects in this frame" question. The user's answer to this produces individual foreground masks for every object within a set of at least two frames. We will frequently refer to these individual object foreground masks as object marks or user marks. Here we present details the "default" tracking algorithm's module set used in the application as well as how to automatically tune the modules parameters with the user's process initialization answers. Table 1 shows the modules within the "default" tracker and a brief summary of their purpose.

Table 1: List of tracking modules in the application's default tracker along with a brief statement of each modules purpose.

Module Name	Type	Summary
Input Processing	Misc.	Processes user input to questions to train object detector and tune most modules that follow.
HOG Detector	Det.	Detects objects in video frames
Detection Statistics	Misc.	Estimates detector performance with frame objects marked by the user.
Tunnels Builder	Misc.	Constructs occlusion foreground tunnels.
Still Obj. Tracking	Assoc.	Locates and tracks non-moving objects.
Tunnel Association	Assoc.	Uses detection responses and OST structure to build initial set of tracklets
Confidence Scoring	Misc.	Trains a detection confidence scoring function with low-level tracklets.
FP Filtering	Misc.	Remove tracklets likely to be False Positive.
MCMC Matching	Assoc.	Associates tracklets using forward and backward MCMC tracking convergence.

3.2.1 User Input Processing for Tuning Tracking Algorithm

Three questions are presented after the user creates a new tracking process. The first question allows the user to constrain the tracking algorithm to a sub-region of the video frames. Reducing the tracking area within the video allows for efficiency

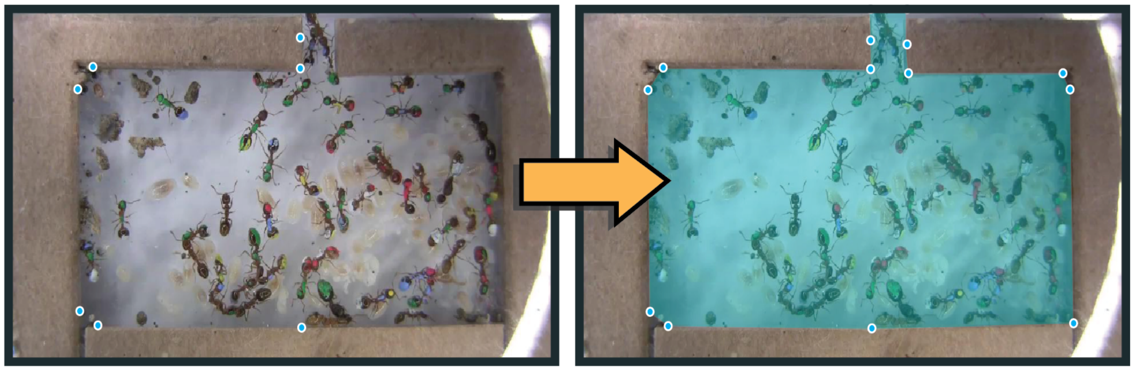


Figure 4: How the user defines the video region to be tracked. The left image shows a portion of the region perimeter defined and the right shows all perimeter points defined. The perimeter is filled for the user to double check it is correct.

improvements during detection and building occlusion foreground tunnels. The second question asks if the objects can enter and exit the scene and if so, where are the regions in which this can happen. Both of these questions can be answered very quickly (30 seconds apiece or less), and have a minor impact on the entire tracking process. The third question asks the user to mark every object in a select number of frames. This question is critical for tuning the tracking modules' parameters, tracking objects that do not move, and improves user efficiency during correction by ensuring objects are least covered in these select frames. The set of frames selected for user marking always contain the initial and ending frames of the video.

Additional video frames are evenly sampled for user marking until at least a total of four frames have been marked and at least 30 object marks have been defined. Most modules use information from the object marks in some way (size information primarily), but we defer minor uses to sections describing the remaining modules.

Given the set of user object marks, the initial input processing module creates a background image for foreground estimation, optimizes morphological operations for

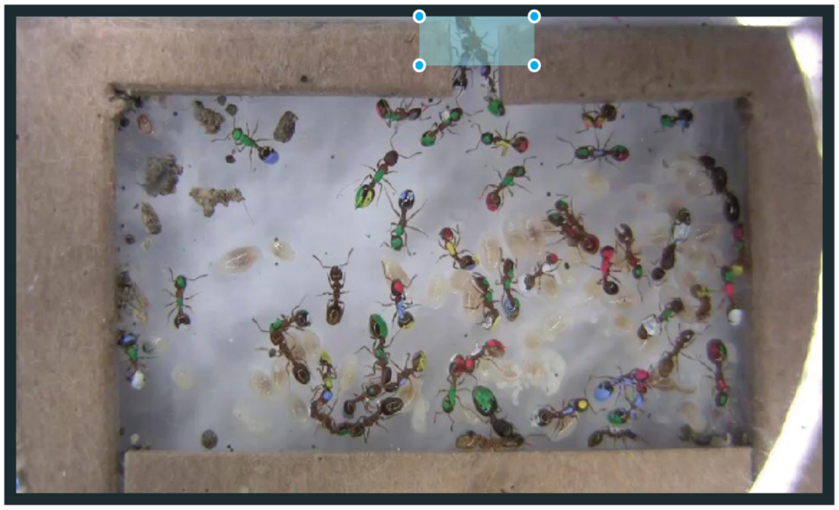


Figure 5: How the user defines object enter/exit regions. Clicking and dragging the mouse creates a new enter/exit region; multiple regions may be defined.

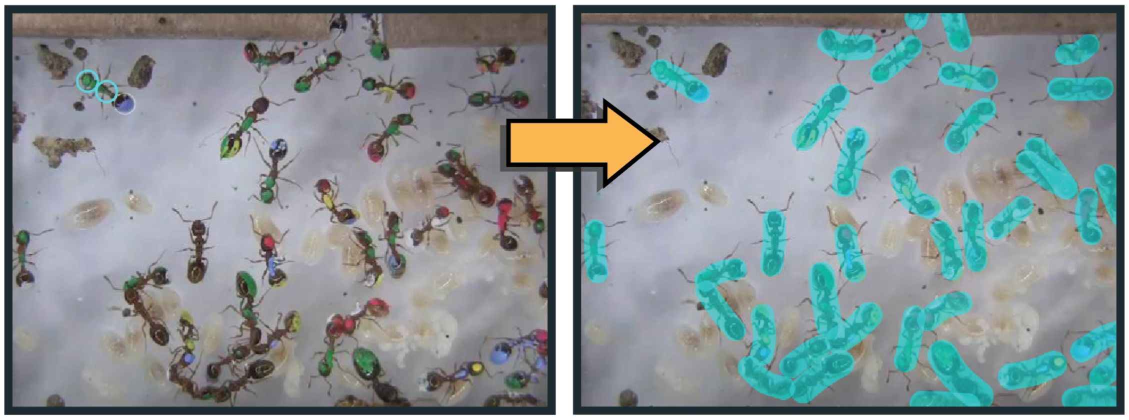


Figure 6: How the user marks the objects in a frame. Three clicks (showing the first two on the left) define the area occupied by the object. The right image shows the results of a user marking all objects in the frame. Note that overlapping marks remain separable.

refining the foreground estimation, and trains a new object detector. Foreground pixels are initially estimated using background subtraction where the background image is calculated using the set of frames with objects marked by the user. Since object foreground pixels are known in these select frames, using these frames only allows for determining an appropriate difference threshold as well as pixel locations

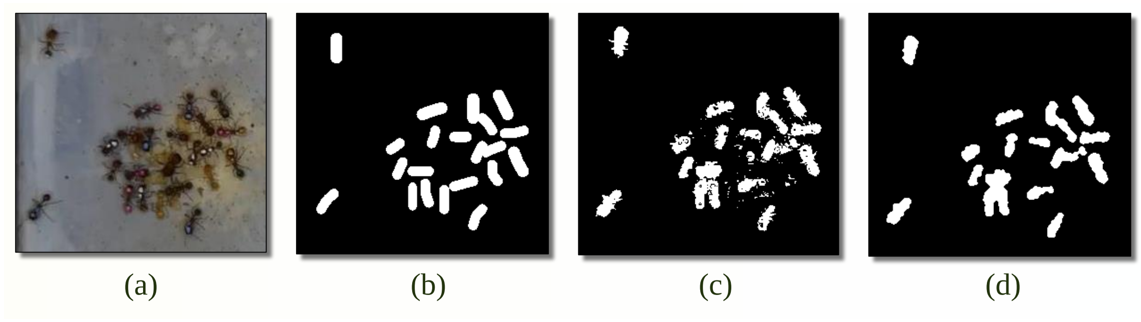


Figure 7: An example of (a) the input image, (b) union of user marks in the frame, (c) the raw foreground classification using background subtraction, and (d) the foreground results after morphological operations optimized by particle swarm have been applied.

likely to contain the non-moving object. Because initial foreground estimation can contain varying levels of noise and produce overly connected foreground blobs, we use particle swarm optimization of morphological operations for refinement. Parameters to be optimized are (ordered as how they are applied): an initial threshold value for the minimum area a foreground blobs must have, the number of times to perform the majority operation (sets a pixel to 1 if five or more pixels in its 3-by-3 neighborhood are 1s; otherwise sets the pixel to 0), the size of the structured element for performing morphological closing (dilation followed by erosion), and again the minimum area threshold on the foreground blobs resulting from the previous operations. The loss function of the particle swarm is formulated as a weighted average of the number of correct, over-segmented, under-segmented, and incorrect foreground pixels.

User markings are used to generate positive and negative examples for training a new object detector. Negative examples are created using background locations and patches centered between nearby user marks from the same frame. We use a linear SVM for classification with HOG features taken from the example patches.

The HOG features [5] use a block size of eight and nine orientation bins.

3.2.2 Object Detection

The detection module produces location, size and orientation information of detection responses in each frame. First, foreground estimation and refinement are applied to a given frame. Foreground blobs that meet the minimum and maximum area threshold (as defined by the user object marks during initial input processing) are given to the previously trained detector. Using the foreground blobs as region proposals reduces the computation time as well as the frequency of false positive detections. Positive responses from the detection module are then handed off to the feature extractor to have any features needed by following tracking modules extracted.

3.2.3 Occlusion Foreground Tunnels

The occlusion foreground tunnels building module is responsible for constructing foreground “tunnels” that span spatially and temporally over the video sequence. We follow the occlusion tunnel approach of [7] which aimed to improve the data association process between pairs of tracklets by reducing the set of possible associations. The foreground tunnels are represented as a directed acyclic graph (DAG) $F = (V, E)$ where $V = \{v_1, ..., v_p\}$ denotes the set of nodes (or vertices) and $E = \{e_1, ..., e_q\}$ denotes the set of edges. Given the set of foreground blobs at each frame, each foreground blob consisting of of pixels, p_i, at frame t_i are represented as a node, $v_i = \{p_i, t_i\}$, in F. Spatial and temporal proximity of the foreground blobs is considered for defining edges between nodes. Two nodes v_i, v_j are connected by an edge if $0 < t_j - t_i \leq n$ and $p_i \cap p_j \neq \emptyset$. We follow [7] and set the temporal threshold to be $n = 2$. Finally, transitive reduction is performed on F to remove any transitive edges (as $n > 1$).

The result of this module is a DAG where nodes represent foreground blobs and edges between nodes signify overlap among blobs between two consecutive frames. As previously stated, the intended purpose of occlusion foreground tunnels was for reducing the set of possible associations during tracklet association. In our proposed method we deviate from how [7] utilized the occlusion tunnels DAG and instead use it as part of an alternate form of tracklet building (association of detection responses) that requires less parameter tuning than previous methods. Details of how on this are provided in section 3.2.4.

3.2.4 Initial Tracklet Building

Two modules are responsible for constructing the initial set of tracklets. First, still object tracking followed association of frame detections. For tracking of still objects, we follow the same procedure as described in [6]. For each of the user marks (except ones within the final frame during the forward pass), we sequentially extract and compare SURF features between consecutive frames at the same location. If no motion is found between the two frames, then still tracking on the object continues. Still object tracking the target stops once the euclidean distance between the SURF features exceeds a threshold (empirically determined and predefined as 73). Tracking repeats the search during a backward pass in time for each user mark (excluding marks in the initial frame). To reduce the computational load, we bypass the tracklet building step (Section 2) by linking all the regions of the same non-moving ant into a tracklet. We compare the distance between tracks produced from still object tracking against frame detections and detections closer than some threshold (determined based on user marks) are removed.

After still object tracking has been performed, the remaining task for the building the initial set of tracklets is to establish associations between detection responses. A common approach to conservatively associating detections used in several works including [9, 6, 14] is to derive an association score on a small set of features (e.g. similarly between RGB histograms and/or size, expected displacement, ...). A dual threshold technique can be employed to ensure that only the most certain associations are made between pairs of detections. This approach requires developing relevant features, means of combining them into a single score, and determining suitable values for thresholds – all of which can changes significantly depending on type/density of the objects and recording conditions. Additionally, the associated tracks tend to be quite short, often failing to make connections in trivial situations. We develop an alternative approach which can accurately generate longer detection associations and does not require defining motion/appearance features, learning function parameters for scoring associations, or determining threshold values on the association scores.

The detection association module requires the occlusion foreground tunnels $F = (V, E)$ outlined in section 3.2.3 and the set of frame detection responses $\mathcal{D}$ produced by the detection module. First, detections $d_u \in \lceil$ gathered frame t_i are mapped to foreground blobs in $v_i \in V$ by determining if the centroid of the detection d_u

falls within the boundaries of blob v_i. For every $v_i \in V$ we calculate its indegree, $deg^-(v_i)$ (number of inbound edges), and outdegree, $deg^+(v_i)$ (number of outbound edges). The set of one-to-one nodes $\acute{V} = \{v_i \in V \mid deg^-(v_i) = 1 \; \& \; deg^+(v_i) = 1\}$ partitions the graph F into tunnel "lanes" (segments of foreground tunnel that do neither merge nor split). The number of blobs with a detection mapped to them vs. not is calculated for each lane (i.e. partition formed by $\acute{V}$), and lanes containing a majority of their blobs mapped are converted to tracklets. Blobs with no mapped detection but are part of a majority mapped lane are assigned interpolated values. Additionally, the module will check if any tracklets existed before the module was run (still object tracklets in this case). If tracklets do exist and contain a detection falling within any one of a lane's blobs, then that lane will not be converted to a tracklet. All detections not used to for building tracklets from the majority-mapped lanes are discarded.

The results from both the detection association module and the still object tracking module form the initial set of tracklets $\mathcal{T}^0$.

3.2.5 Detection Confidence Scoring and False Positive Removal

We train the learned classification model used by the detection module with a rather small number of examples (four frames worth of positive object examples in most cases), and the classification method was kept simple (linear SVM) in light of this. Although the number of false positive detections is reduced by using the refined foreground blobs as proposal regions, false detections still occur due to substantial inter-object occlusions and background debris. To further minimise the number of false positive tracks, we use the current set of tracklets to train a detection confidence scoring function for filtering tracklets likely to be false positive.

Detected locations from all tracklets present in every ηth [1] frame of the video are collected as positive training examples. From these frames, we also collect several sets of negative training examples. Each tracklet's detected location in the frame has a small amount of random noise added to the X and Y position as well as the angle. The amount of noise is randomly selected for both the X and Y positions and bounded between one-fifth and one-half the minimum dimension of the average user mark size. Noise added to the angle is bounded between 10 and 40. The direction of the noise is also random. We gather additional negative examples from tracklets near to one another. Specifically, tracklets closer than the average width of the user marks have an image patch that is centered and aligned along the line segment between the centroid of the two detections. The final set of negative examples is randomly sampled the from background regions of the video frame. We use HOG features with a block size roughly equal to one-fourth the minimum size dimension of the average user mark size. Because of the noise likely to be present in the training set due to false positive tracks, a Random Forest classifier is selected as the learning model and 300 decision trees are grown with the unbalanced dataset.

To filter false positives we first use the Random Forest classifier to assign a confidence score to every detection within a tracklet. A detection score is the average

[1]We use only a subset of the frames because of the increase in memory requirements with respect to video length and number of objects. Furthermore, consecutive frames produce nearly identical examples. We determine the number of frames that need to be skipped as what would roughly provide 5000 positive training examples (if possible) given the video length and the average number of user marks per annotated frame.

classification of the decision trees and ranges between 0 and 1. We calculate the mean detection confidence of each tracklet and ones with mean confidence less than 0.5 are removed. An exception is made for tracks that cover any user object marks.

3.2.6 Tracklet Association with MCMC Convergence

Given the set of tracklets formed by tunnel detection association and the still object tracking module, the goal of the current module is to make any appropriate connections remaining between the tracklets. Iterative matching based on convergence agreement from both forward and backward particle filtering is used to solve the data association problem on the incomplete tracklets. For both forward and backward tracking, we follow the MCMC particle filtering based approach of [8]. The method proposed in [8] improves upon [10] regarding tracking accuracy by incorporating global maximization of foreground for reducing the possibility of drafting failures and also regarding efficiency by using a variable proposal distribution among the targets (i.e. samples employed in Markov chain are not evenly distributed among the targets). They base their variable proposal distribution on estimated object motion.

Let $\mathcal{T}^0 = \{T_i\}$ be the set of tracklets which will be associated over a sequence of matching stages and $\mathcal{F} = \{F_t\}$ be the set of foreground pixels in each frame. Each stage of matching n uses the previous tracklet set $\mathcal{T}^{n-1}$ to produce the set of further associated tracklets $\mathcal{T}^n$. We define the Markov chain length (i.e. particles or iterations) to be 200 for all matching stages. During stage n of matching we first remove all foreground pixels within each $F_t \in \mathcal{F}$ which are covered by a tracklet $T_i \in$

$\mathcal{T}^{n-1}$. Doing so exploits the foreground maximization criteria of [8] and assists each matching stage in making more difficult tracklet associations than the one before. The pixels covered by tracklet $T_i \in \mathcal{T}^{n-1}$ in a frame t are calculated as the square region of size ρ centered at the (x,y) position of T_i in frame t. We set ρ to be the average maximum dimension of the user marks. Forward tracking is performed from the tail position of each tracklet ending before the last frame of the video. The joint state $\mathcal{X}$ contains the estimated position of all objects meeting this criterion. At each frame t during tracking we compute the distance $d(i, j)$ between the predicted object states within $\mathcal{X}$ for tracklet T_i and each tracklet T_j which begins in frame t. A tracklet T_i being tracked which has a distances $d(i, j) < \delta$ is said to have converged to tracklet T_j in the forward pass and MCMC tracking on T_i is terminated (i.e. particles will no longer consider T_i). In our experiments, we define the threshold δ to be the average minimum dimension of the user marks. We repeat the same procedure in the backward direction for tracklets who begin later than the initial frame of the video. After the forward and backward passes, the stage is complete and tracklets with unique and agreeing convergence in both directions are associated.

Matching could continue until to no further connections are made between the stages, although we found that ID switches are much more likely to occur during the final few stages if one uses such a stopping criteria. Additionally, each stage is computationally expensive, and the number of connections made during the final stages is few. Because ID switch errors are more costly to correct than fragment errors during user correction, we set the system to stop further matching stages when ten or fewer tracklet associations are established in a single stage. This policy produces an acceptable reduction in fragments without the risk of incurring numerous ID switch errors.

3.3 Error Correction

The design goals for the correction portion of the application were: total control – enable the user to make any correction they see necessary, and efficiency – promote efficiency in terms user time and effort. We provide two correction modes withing the application which roughly address each of the two design goals. The first mode, manual mode, displays all of the tracklets within the video, allowing the user to locate any errors and directly invoke correction operations to fix them. Tracking failures produced by the system fall into one of six categories (listed in table 2). These failure types represent the complete set of correction operations needed by the manual correction mode to satisfy the total control requirement. Section 3.3.1 discusses implementation details for the set of manual operations. The second of the two modes provided is Guided mode. Before the user begins the correction phase, the backend inspects the results produced by the tracker to locate potential errors. Specifically, possible fragment and ID switch errors are determined and captured as "reviews" to be presented to the user in prioritized order. Section 3.3.2 provides details on developing guided correction reviews.

Table 2: List of tracking failures the user will need to be able to correct and their descriptions.

Tracking Failure	Description
Fragment	Tracking fails to associate multiple tracks covering a single object. Tracks could be temporally separated or overlapping (leap frog).
ID Switch	Tracking makes an incorrect association causing one track to cover one object at some point in the video and a different object later.
Missed Object (FN)	An object is never covered by a track at any point in the video.
False Track (FP)	A track never covers an object in the video, or does but less appropriately than other track.
Off target	Some portion of a track does not accurately represent the state of the object it covers (location, orientation, or size)
Incomplete coverage	Either the beginning or ending portions of an object visible in the video is not covered by a track.

Each manual operation made or guided review answered through the application frontend is converted to an API request and delivered to the backend. Processing time varies for different requests and forcing the user to wait for each operation to be completed by the backend would be detrimental to overall time spent during correction. We include a backlog queue of correction requests invoked and design the backend to maintain the integrity of all backlogged requests as it processes the queue. After each correction request is applied, the backend informs the frontend of changes made to the current state of the tracking results and also provides the updated set of unanswered guided correction reviews.

3.3.1 Manual Correction Operations

In order to provide full flexibility for correcting tracking failures the system includes five manual correction operations (listed in Table 3) that enable correcting any possible error in the tracking results. Each of the manual correction operations has a corresponding API command available to the frontend.

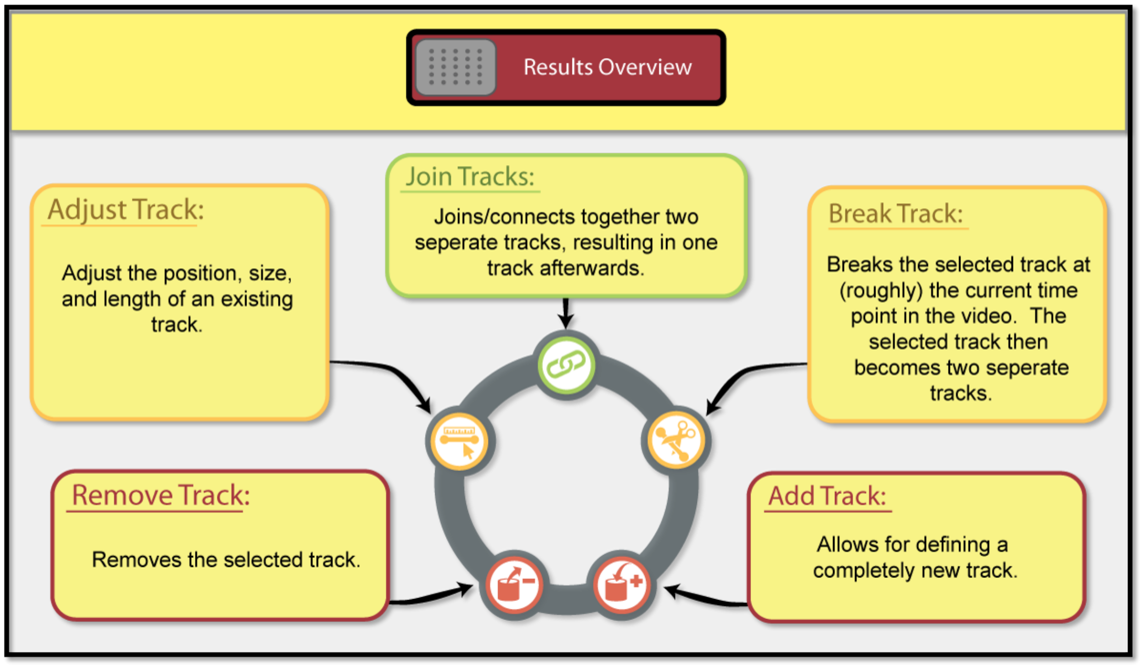

Results Overview
Join Tracks:
Joins/connects together two seperate tracks, resulting in one track afterwards.
Adjust Track:
Adjust the position, size, and length of an existing track.
Break Track:
Breaks the selected track at (roughly) the current time point in the video. The selected track then becomes two seperate tracks.
Remove Track:
Removes the selected track.
Add Track:
Allows for defining a completely new track.

Figure 8: Illustration of the manual operations available for error correction. This is one of several help dialogs provided to the user within the application.

Table 3: List of manual correction operations available and what tracking failures they are intended to address (see table 2).

Manual Operation	Description	Failures Addressed
Join	Connects, or merges if temporally overlaying, two separate tracks	Fragments
Break	Separates a single track into two tracks, roughly around the defined frame	ID Switches
Add	Create a new track	Missed Object (FN)
Remove	Deletes a track entirely	False Track (FP)
Adjust	Redefines any combination of location, orientation, size and duration properties of a track	Incomplete coverage & off target

The join command accepts two unique ID values of the tracklets involved in the join. First, a check is made to determine whether the two tracks temporally overlap one another. In the case of no temporal overlap, the two tracks are associated to form a single tracklet. Two situations can arise in the case of temporal overlap between the tracks. To determine the best way of merging the overlap between the tracks, we take advantage of the fact that the system keeps a record of every association made within a tracklet and stores this information in the tracklet's connections property. The connections between the two tracks are compared to determine if a leap-frog has occurred. Leap-frog connections result when three or more tracklets are correctly connected (technically) but skip an intermediate tracklet. Leap-frogs typically occur from suboptimal manual joins by the user or errors made by the system in an attempt to filter possible answers for guided fragment reviews. If a leap-frog has occurred, then the overlapping region of the tracks are merged by replacing interpolated gap portions with the location values of the intermediate tracklet that was leaped over. The remaining cases of joins involving tracklets with temporal overlap are handled by determining the longer of the two tracks and using its locations values as the merge result of the overlapping region. The result of the join operation is a single tracklet that is assigned the ID of the tracklet that starts first (i.e. closer to the initial frame of the video). All requests in the backlog involving a track ID of either of the two tracklets joined are updated to the appropriate ID resulting from the join. All guided reviews have their possible answers and priority values updated.

The break operation accept a tracklet ID and a frame number. Information related to connections made to form the tracklets are abstracted from the user and because of inaccuracies related to the frontend video player some inference must be made to determine what section of the tracklet should be broken. Again, we use the stored connection information of the tracklet. First, nearby connections are located by searching for connections made within the tracklet that are within 30 frames of the defined frame. If two or more connections are near, then the nearby connection starting soonest and the nearby connection ending latest are broken. The region between the two broken connections is discarded (along with any reviews associated).

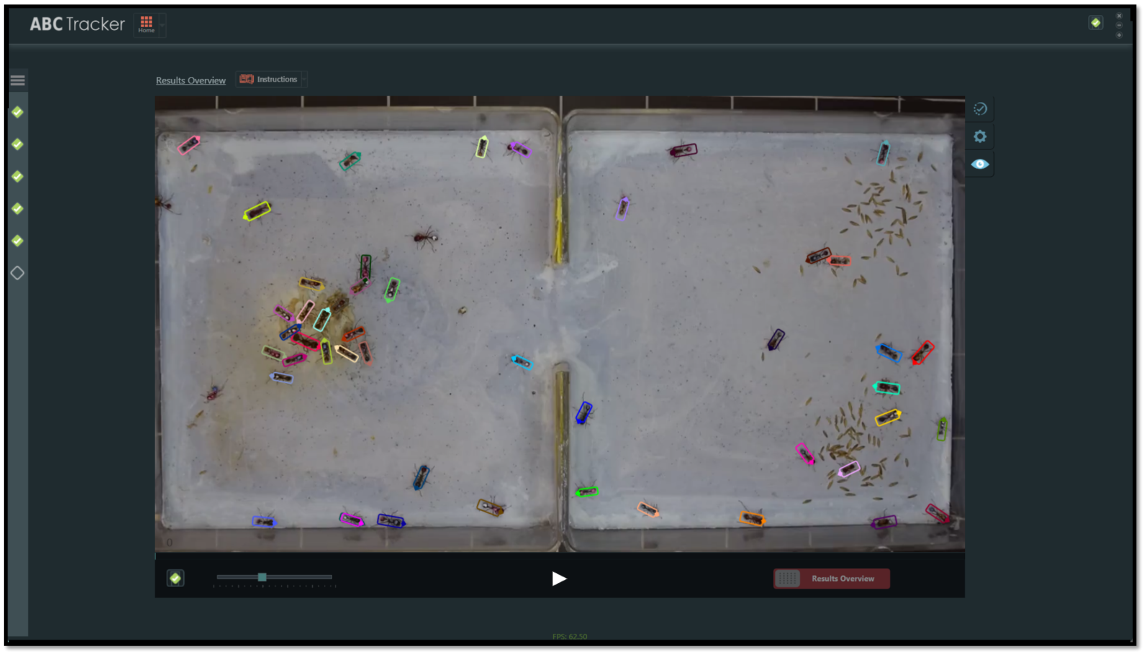


Figure 9: This shows the correction interface while performing manual corrections.

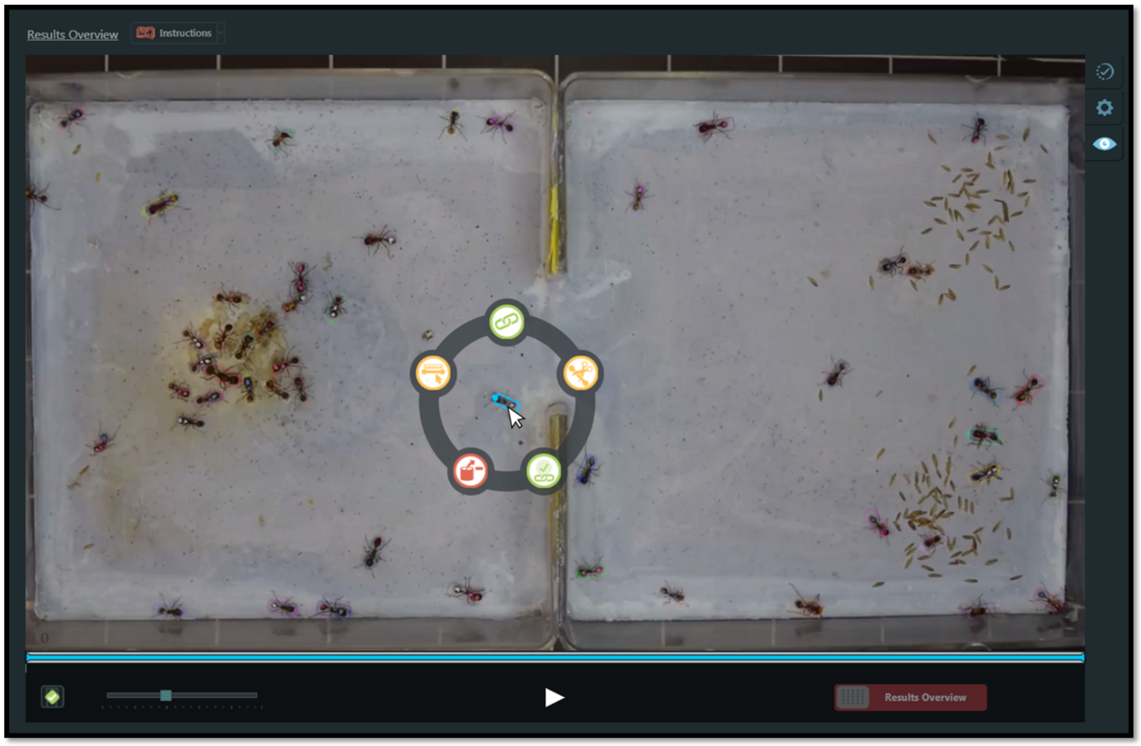


Figure 10: Demonstrates how the user can highlight individual tracks. Right clicking the track presents all operations that can be performed.

If there is only one nearby connection, then it is broken. In the remaining case, no nearby connections, we force a break on the tracklet that spans 30 frames centered about the defined frame. The result of the break operation is a single tracklet split into two tracks where the latter portion of the broken tracklet is assigned a new ID. Any guided reviews particular to the latter portion of the unbroken tracklet are transferred. Similar to join operation, the system must update backlogged operations to maintain their integrity, and all guided reviews must have their possible answers and priority values updated.

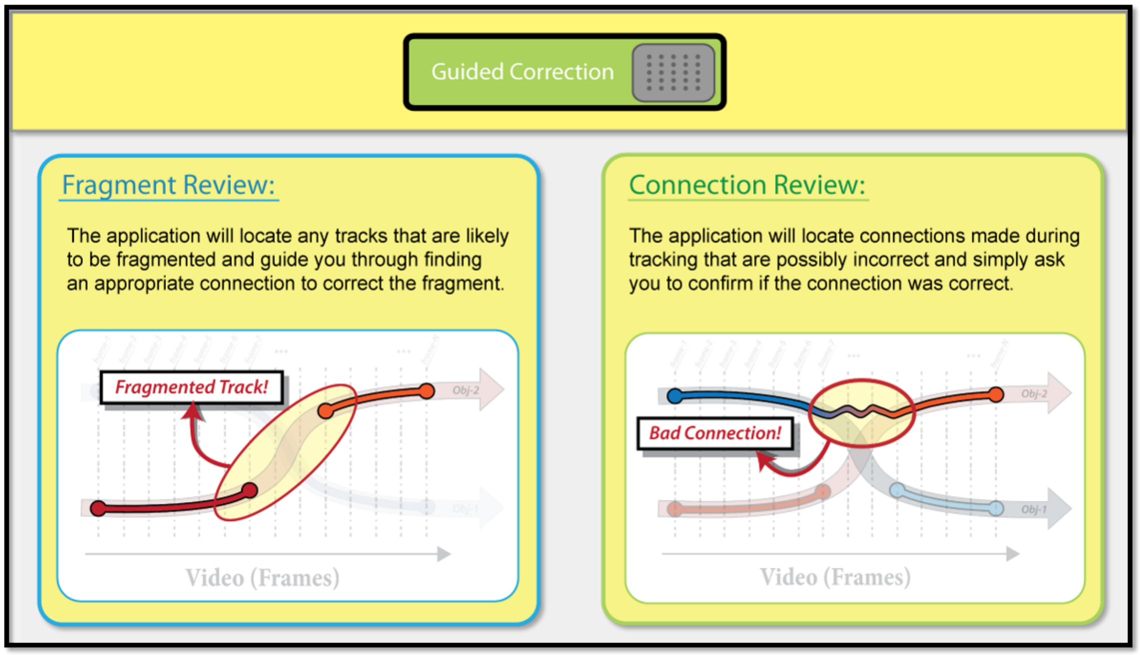
Adjustment operations accept a tracklet ID, a set of one or more frame numbers, and location/orientation values for each of the specified frames. An optional size parameter can also be provided. First, the frame number are checked to determine if they span beyond the current span of the defined tracklet. If so, the tracklet is expanded to ensure it covers the defined frames. Next, location/orientation provided are assigned to the tracklet. Any guided fragment review associated with the adjusted tracklets is updated to reflect any changes in the starting/ending frame of the track. The add operation is nearly identical to the adjustment operation except no ID is specified and results in an entirely new tracklet.

The final manual operation available is the remove command. It accepts only one input value, the ID of the track to remove. After removal of the track, any guided reviews pertaining to the removed track are flagged as invalid, and backlogged operations are updated to maintain their integrity by also flagging them as invalid if they involve the removed tracklet.

3.3.2 Tracklet Reviews for Guided User Correction

The primary purpose of guided correction reviews is to accelerate the rate of locating and correcting tracking failures for the user. We present two types of guided reviews for achieving such, which we will refer to as fragment reviews, and connection reviews.

Fragments are typically the most common type of error in many tracking algorithms. A very simple search heuristic, tracklets shorter than the length of the video, can detect errors concerning pairs of tracklets that should have been connected with %100 recall and to some level of precision depending on the frequency of objects entering and exiting the scene. To correct tracklet fragments, we need to know the proper association that the tracker failed to establish. After performing tracking,



The application will locate any tracks that are likely to be fragmented and guide you through finding an appropriate connection to correct the fragment.

The application will locate connections made during tracking that are possibly incorrect and simply ask you to confirm if the connection was correct.

Figure 11: Illustration of the two review types available for guided error correction. This is one of several help dialogs provided to the user within the application.

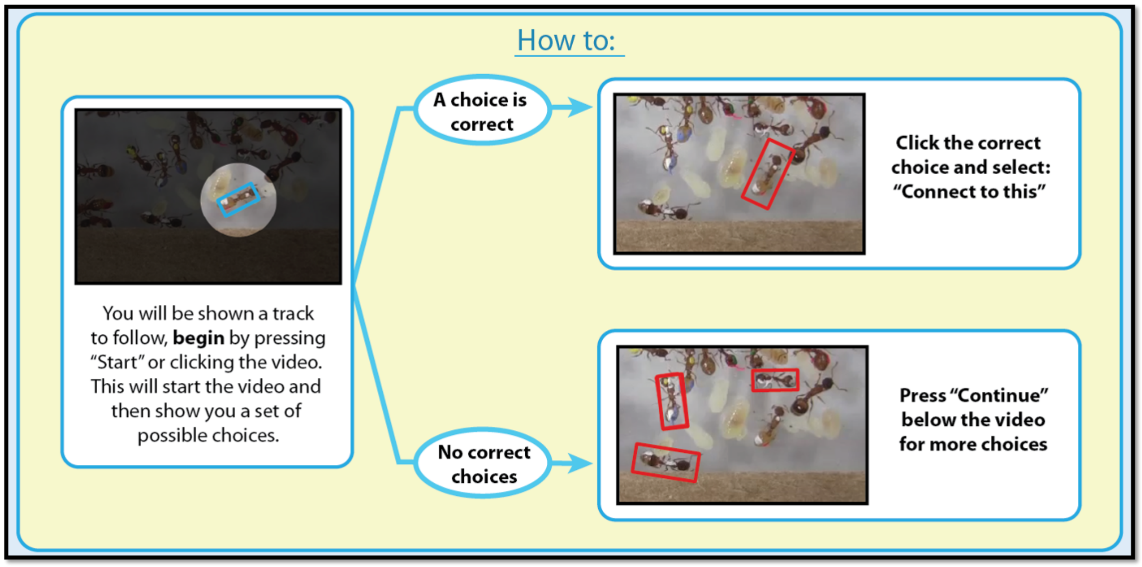
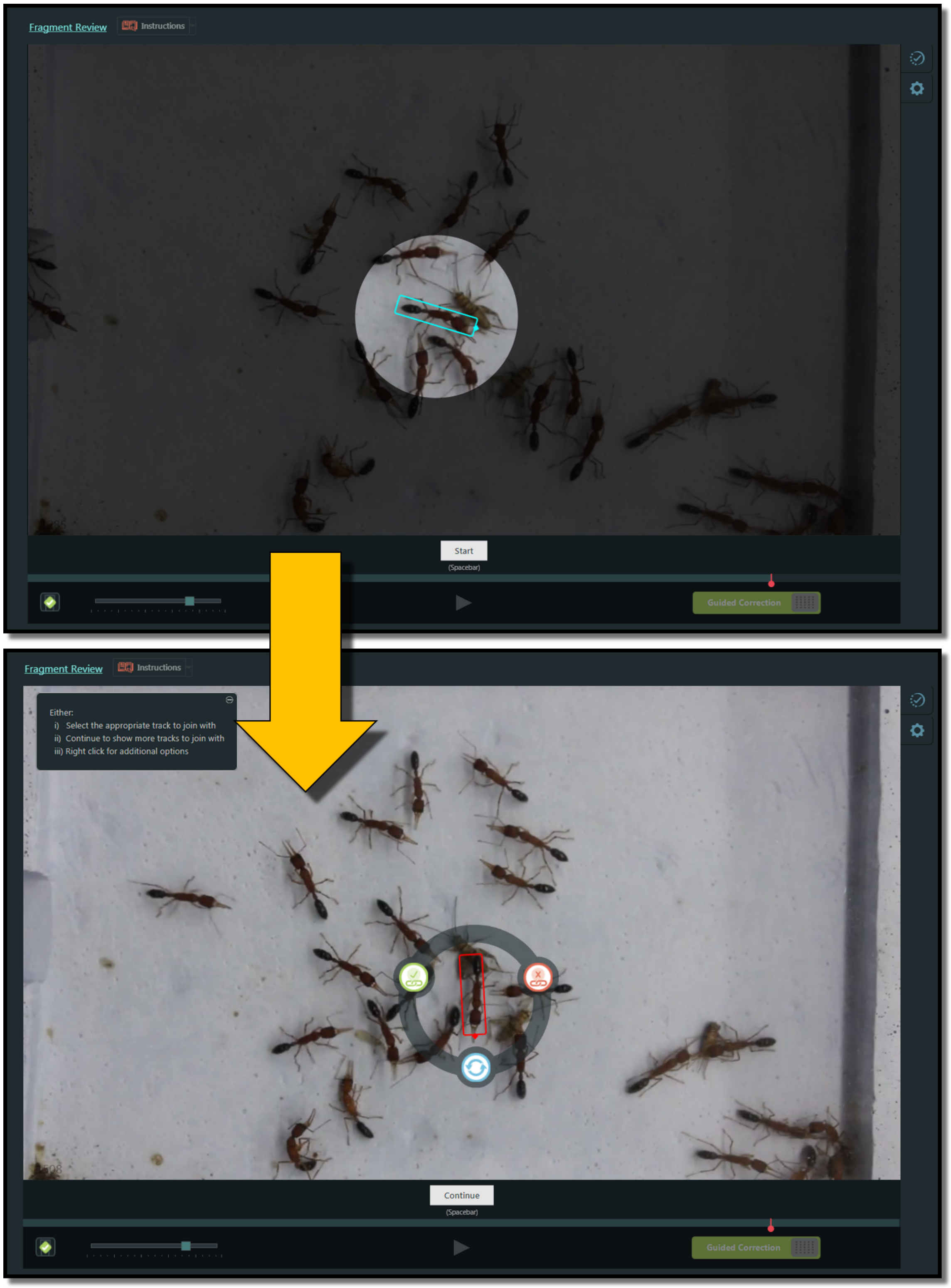

How to:
You will be shown a track to follow, **begin** by pressing "Start" or clicking the video. This will start the video and then show you a set of possible choices.
A choice is correct
Click the correct choice and select: "Connect to this"
No correct choices
Press "Continue" below the video for more choices

Figure 12: How the systems guides the user to correct fragmentation errors through fragment reviews. The tracklet likely to be fragmented is focused prior to the user initiating the review. Possible answers are then presented through video playback.

we take the set of resulting tracklets and determine which tracks do not span the entire duration of the video. A fragment review is created for each of these to be presented to the user. Because of how these reviews are displayed to the user for answering, the number of possible answers (i.e. tracklets which could be associated with the fragment) becomes a significant factor in how long each fragment review will take to answer. The higher the number of possible answers, the more frequently the application will need to halt and allow the user to answer. We reduce the number of possible answers by using information contained in the OsT structure. Similar to how Fasciano et al. uses OsT for filtering possible associations during tracklet matching, we do the same for filtering answers available to the user.

ID Switches (IDS) can be difficult to locate automatically as well as for the user without watching the video playback of the tracking results several times. Though



Either:
i) Select the appropriate track to join with
ii) Continue to show more tracks to join with
iii) Right click for additional options

Figure 13: Guided correction example for fragment review. The top image show the beginning state of the review and after starting the review a set of answers are presented as in the bottom image.

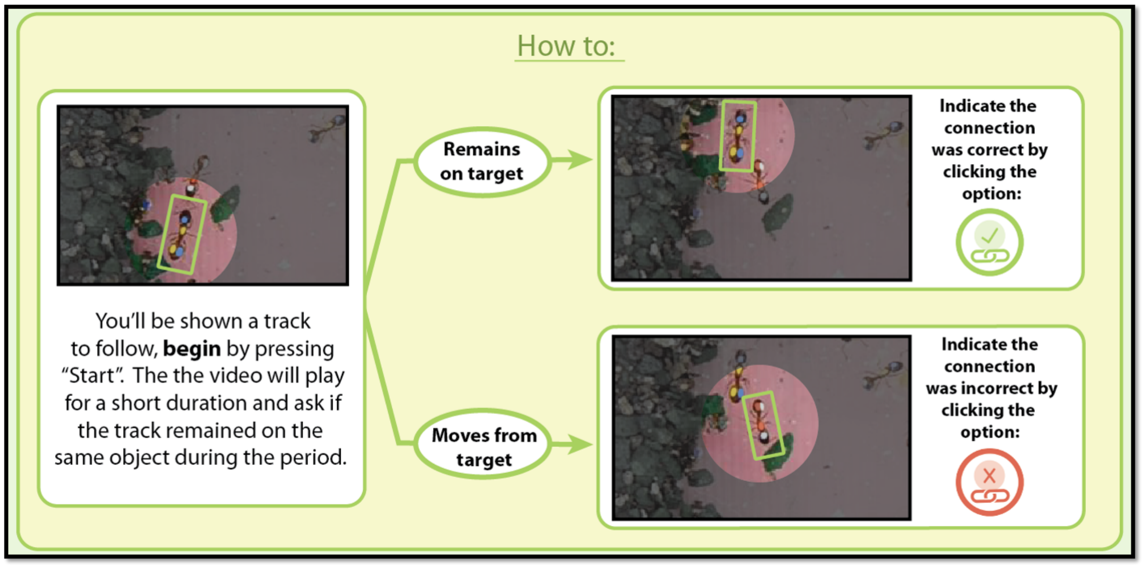

How to:
You'll be shown a track to follow, **begin** by pressing "Start". The the video will play for a short duration and ask if the track remained on the same object during the period.
Remains on target
Moves from target
Indicate the connection was correct by clicking the option:
Indicate the connection was incorrect by clicking the option:

Figure 14: How the systems guides the user to correct ID switch errors through connection reviews. The tracklet containing a connection likely to be IDS is focused prior to the user initiating the review. The video plays for a moment to show the connection made and then allows the user to answer if the connection was appropriate.

such errors can happen during any association module, primarily occur after detection association. Because the detection confidence scoring function was specifically trained to give low scores to patches located between tracklets, we use this as a means of detecting tracks that switch from one object to another (thus some detection patches should between the objects). For each connection, we use the detection confidence scoring function to calculate the mean confidence among frames within the connection.

Each connection made has a connection review created for it and the set of connection reviews are ranked based on their mean confidence score. Although connection reviews can be processed quickly by the user (less than four seconds apiece usually), it would be too time-consuming to review all connections made during tracking. We create a pool of connection reviews that will be examined by the user and increase its size as incorrect connections are pointed out. The initial size of the pool is set to be roughly 3% of the total number of connections made during matching and is populated based on the mean confidence score assigned to the connection reviews as previously discussed. For each connection review answered as incorrect, the pool is expanded to include an additional 1% of the total connections made. When the pool has been exhausted, the system suggests to the user to continue finding any remaining IDS through manual corrections.

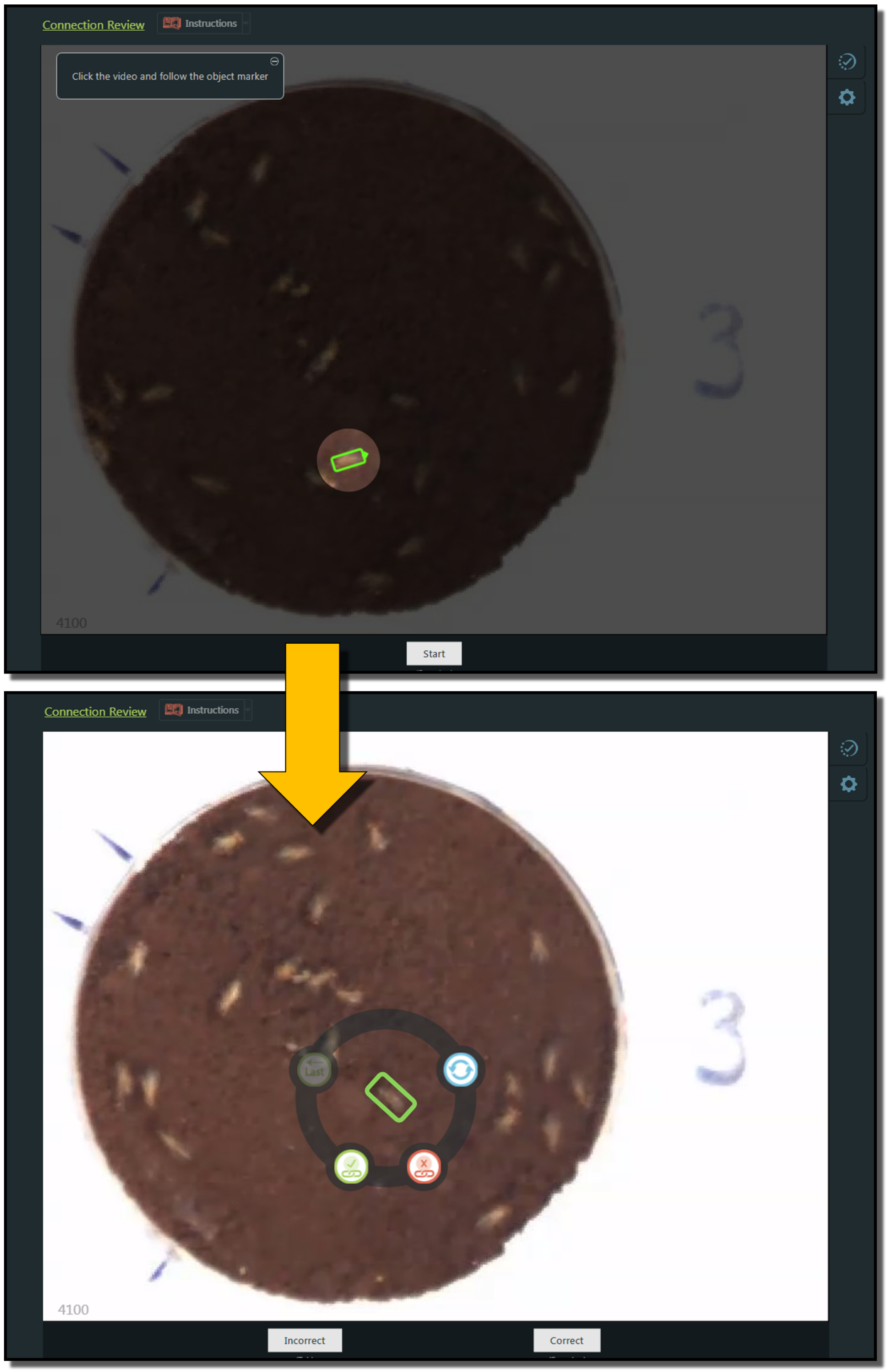


Figure 15: Guided correction example for connection review. The top image show the beginning state of the review. After starting the review the video plays to show the connection made and then allows to the user to answer (bottom image).

CHAPTER 4: EVALUATION

In this chapter we examine the performance of both specific parts of the ABC-Tracker system as well as how the application performs as an end-to-end system. The next few sections evaluate two critical areas of the system individually, specifically tracking performance and ordering of connection reviews for locating ID switch failures during guided user correction, on several challenging datasets. We then present the results of a user study conducted to determine user efficiency and effectiveness towards collecting accurate tracking data on a video.

4.1 Tracking Performance Evaluation

First, we assess the performance of the application's "default" tracker as described in section 3.2 and compare it with the work of [7]. We perform tracking evaluation with a total of nine 5,000 frame video recordings of biological objects. Seven recordings consist of 30 to 50 ants of which four videos have a small number of the objects entering and exiting the scene. Some of the ants within these videos have been painted to assist in identification. Two videos are of 22 unmarked Macrotermes Michaelseni termites confined to a circular dish captured at 15 frames per second. Ground truth was gathered for each video for assessment of tracking performance. We use the same evaluation metrics as used by [11, 9, 6]. Table 4 lists the metrics and their meanings.

Table 4: List of tracking evaluation metrics used for comparing performance of tracking algorithms.

Metric Name	Description
Fragment (Frag)	When a ground truth trajectory is interrupted by tracking results. (Failure to associate)
ID switch (IDS)	When an object track changes its matched ground truth identity. (Incorrect association)
Mostly Tracked (MT)	Ground truth trajectory tracked for more than 80% of video.
Partially Tracked (PT)	Ground truth trajectory tracked between 20% and 80% of the video.
Mostly Lost (ML)	Ground truth trajectory tracked for less than 20%.
Recall (Rec)	Percentage of ground truth locations covered by a track (sometimes called APTA).

The method compared against [7] follows a tracking-by-detection approach and utilizes the foreground occlusion sub-tunnels for filtering incorrect connections during iterative stages of tracklet association. A HybridBoost tracklet affinity scoring function [9] is trained offline for each stage of matching. Offline learning is also performed for training the object detector. Parameter values for all tracking subroutines in [7] and features used by the tracklet affinity scoring function (particularly motion features) are manually tuned. Additionally, location values for all objects within

Table 5: The average tracking performance comparison on four 5,000 frame videos from the C5/6 ant dataset between the method of [7] and the proposed method.

C5/6 Ant Colony Dataset

Method	Frag	IDS	MT	PT	ML
Fasciano [7]	44.8	6.8	**18.5**	15.3	7.8
Proposed	**15.8**	**4.0**	11.0	**25.0**	**5.5**

Table 6: The average tracking performance comparison on two 5,000 frame videos from the BR termite dataset between the method of [7] and the proposed method.

Termite Dataset

Method	Frag	IDS	MT	PT	ML
Fasciano [7]	32	12	19.5	2.5	0
Proposed	42.5	1	15.5	5.5	1

the initial and ending frames of each video tested are supplied for still object detection. Our proposed method only requires user answers to the questions asked during initialization of a tracking process as described in 3.2. Table 5 shows the average results of evaluation on the ant colony datasets and 6 shows the average results on the termite datasets.

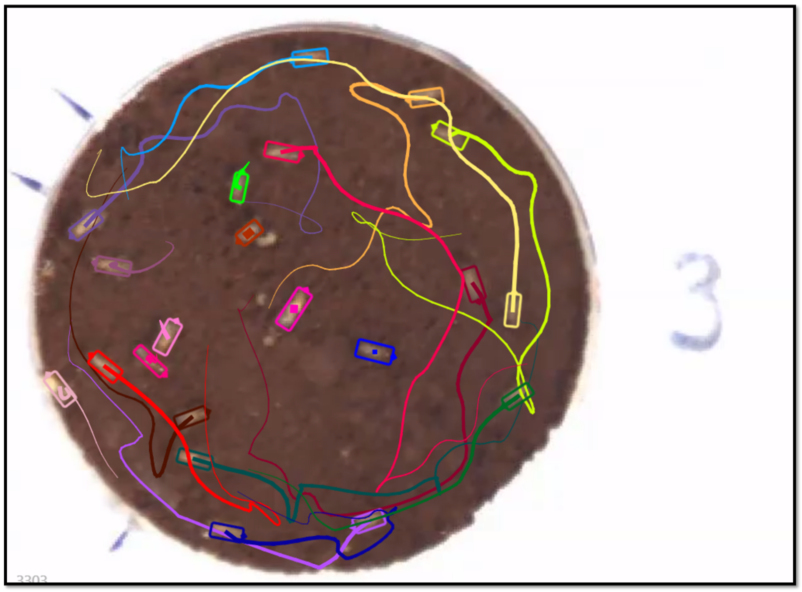


Figure 16: Example results on one of the termite datasets.

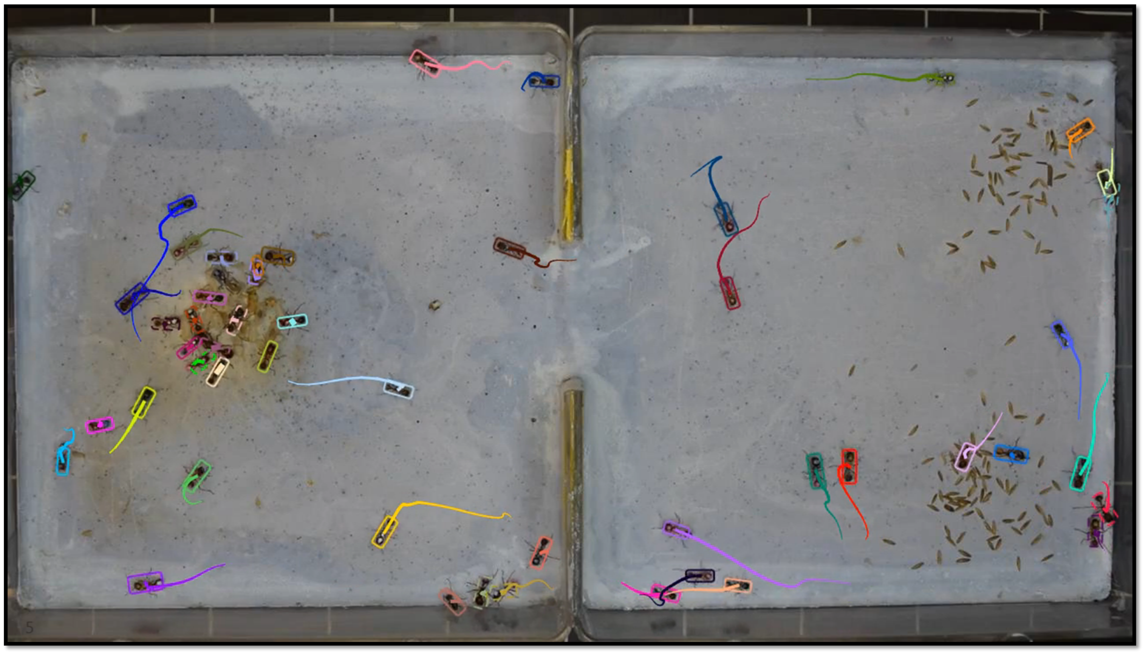


Figure 17: Example results on one of the ant colony datasets that do not allow enter/exit.

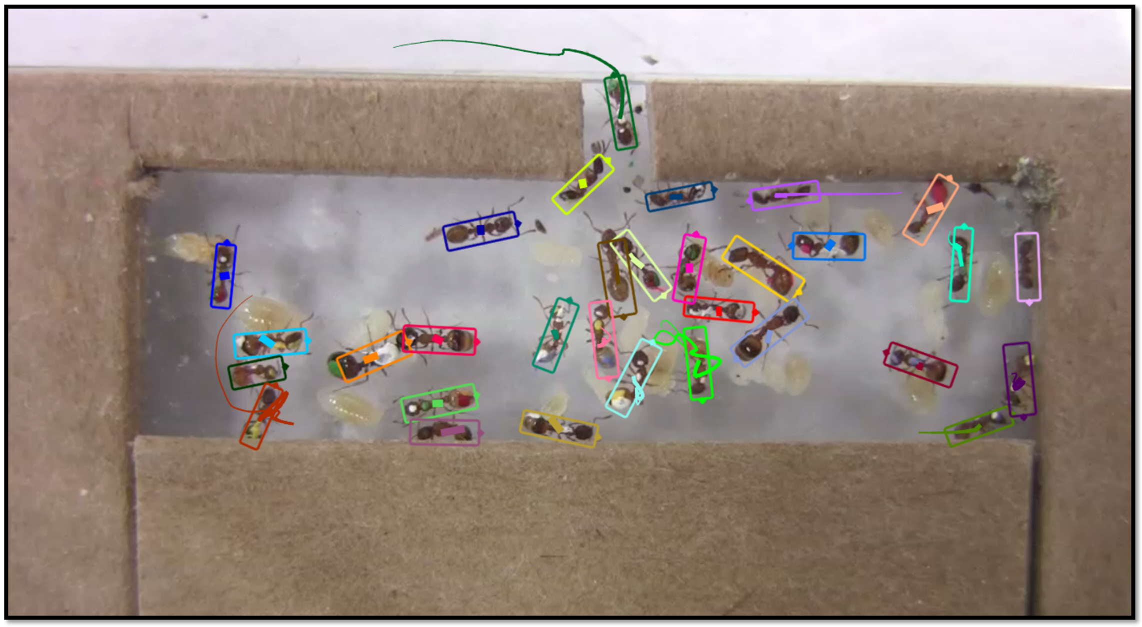
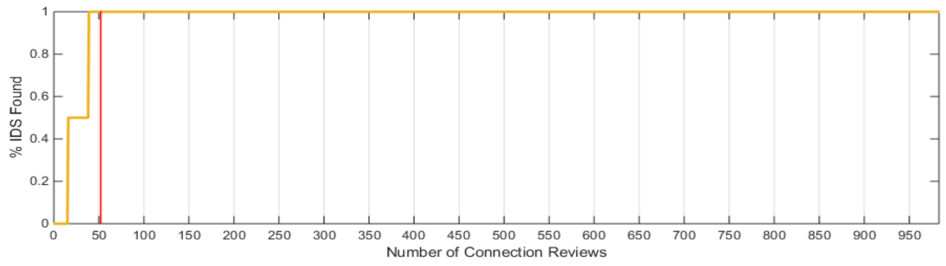


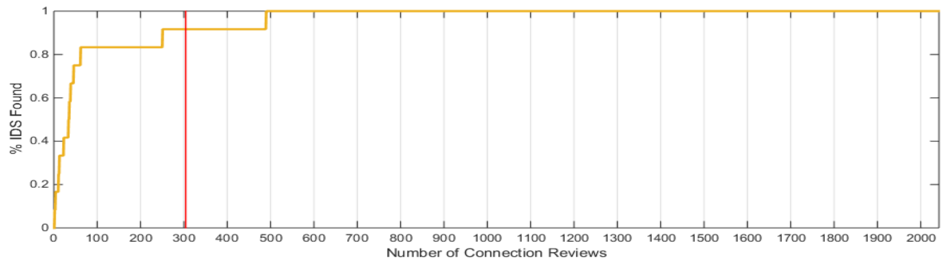
Figure 18: Example results on one of the ant colony datasets which allow enter/exit. Specifically, this is the C61 dataset from the C5/6 ant colonly dataset collection.

4.1.1 Evaluation of IDS Review Ordering

As previously discussed, ID switch errors are usually the most difficult tracking failures to locate and can cause significant representation mistakes in the tracking information gathered. Here we evaluate the system's ability to rank connection reviews (as described in section 3.3.2) so the user can quickly identify incorrect associations made during tracking. We use the same set of videos to evaluate the ranking performance as were used for evaluating tracking accuracy in section 4.1. Tracking is first performed for each video given user initialization question answers; then connection reviews are generated based on associations made by the tracker between pairs of tracklets. After ranking connection reviews as described in section 3.3.2, the ground truth is used to determine incorrect tracklet associations made by the system. Figures 19, 20 and 21 show the number connection reviews that would need to be reviewed to locate a particular percentage of the ID switches incurred. In each plot the vertical red line represents where the system would suggest that the user discontinues reviewing connections in guided correction mode.

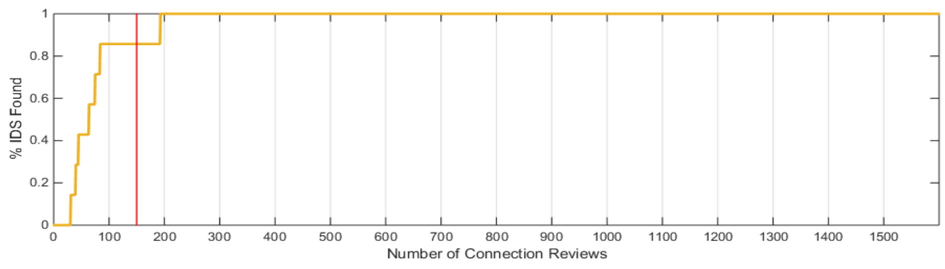


(a) Ant Colony Dataset - 1 (Treatment2162)

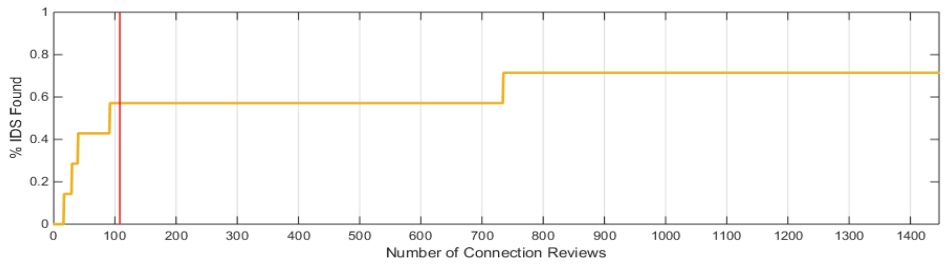


(b) Ant Colony Dataset - 2 (Baseline281)

Figure 19: Visualization of performance for ordering connection reviews. The vertical red line represents the number of reviews that the user will be shown given an initial pool size of 3% and 1% pool increase for each incorrect connect flagged. Note that we have not shown ordering results for the DSC-0003 ant dataset as no IDS occurred.

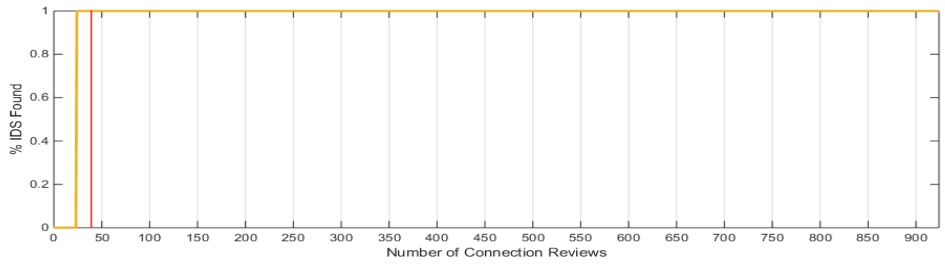


(a) Ant Colony Dataset - 3 (C51)

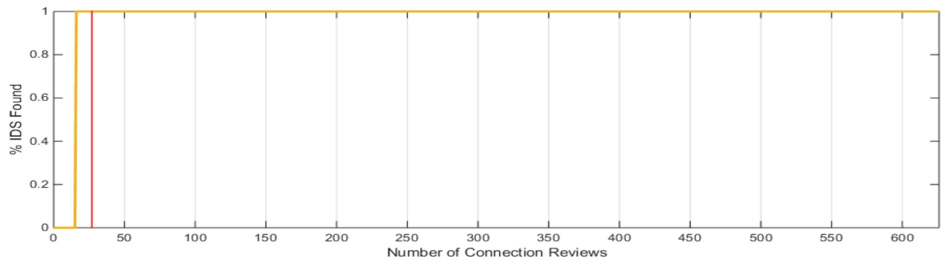


(b) Ant Colony Dataset - 4 (C52)

Figure 20: Visualization of performance for ordering connection reviews. The vertical red line represents the number of reviews that the user will be shown given an initial pool size of 3% and 1% pool increase for each incorrect connect flagged.



(a) Ant Colony Dataset - 5 (C61)



(b) Ant Colony Dataset - 6 (C62)

Figure 21: Visualization of performance for ordering connection reviews. The vertical red line represents the number of reviews that the user will be shown given an initial pool size of 3% and 1% pool increase for each incorrect connect flagged.

Table 7: The Results of the user study conducted with ant colony dataset C061 using (a) system design described by [14] and (b) the proposed system. Note that the the work of Nguyen et al. performed gap filling after user corrections but the proposed system does not. This explains the difference in recall values reported between the two approaches as gap filling has a significant impact on recall.

User Study Stage within Nguyen et al. [14]	Frg	IDS	Rec	User Time (min)
Tracking Results	43.0	15.0	0.41	n/a
Expert User Corrected	4.3	3.3	0.97	18.8

(a)

User Study Stage within Proposed System	Frg	IDS	Rec	User Time (min)
Tracking Results	12.0	1.0	0.66	10.95
Expert User Corrected	1.5	0.0	0.68	10.69

(b)

4.2 User Study

Here we evaluate the proposed system as a whole by means of a user study. To compare our system with the prior work of Nguyen et al. [14] the study was conducted on a 5000 frame video of 33 Temnothorax Rugatulus ants recorded at 30 frames per second. A sample image from the dataset is shown in Figure 18. Ground truth for this video was used to measure the accuracy of the tracking results and user corrections.

The study consisted of two participants who were asked to: create a new tracking process for the video, answer the initial set of questions as outlined in section 3.2.1, run tracking, and afterward perform corrections. We compare the results of the user study on our system against the work of Nguyen et al. [14] in table 7. Because the work of [14] used offline trained models and parameter values tuned by an expert specifically to the input video (i.e. participants of their study did not perform pa-

Table 8: Average interaction time breakdown for users of the proposed system.

Time in Manual Mode (min)	Time in Guided Mode (min)	Number of Connection Reviews	Time/ Connection Review (sec)	Number of Fragment Reviews	Time/ Fragment Review (sec)
3.54	7.16	39	3.55	11	26.46

rameter tuning or any other form of tracker modifications) there is no value to report for user time spent obtaining raw tracking output. Again, we point out that such an approach does not generalize as well as our system.

From table 7 we can see that the total time spent by the user for our system was roughly 22 minutes. This total time is only slightly more (about 4 minutes) than the time users spent performing corrections using the compared method and includes the period for tuning the tracking algorithm. Additionally, there were fewer errors overall produced by users of our system resulting in an 87.5% reduction in fragment errors over the prior work and all IDS errors corrected. Table 8 provides a detailed breakdown of time spent by users during the study. A majority of user time was spent within guided correction mode, and individual reviews took on average 3.55 seconds and 26.46 seconds for connection and fragment reviews respectively. These experimental results demonstrate that the proposed system can effectively allow users to tune tracking parameters, obtain raw tracking results and correction errors within the results.

CHAPTER 5: CONCLUSION

We have proposed a new end-to-end system for tracking multiple objects in a video sequence. We discuss the current state of software tools available for collecting tracking information and their limitations. The proposed system can tune parameters of the tracking algorithm given user answers to a set of simple questions. Tracking performance of algorithms used by the system showed on average improvement in ID switches and comparable performance regarding fragments when evaluated on several challenging datasets. As discussed, the reduction in ID switch frequency has a significant impact on user correction time. During the guided portion of user correction within the application, we show that roughly 90% of ID switches occurring in the tracking results can be automatically located and corrected in an efficient manner and only requires the user to review less than 7% of the connections made during tracking. Additionally, results of a user study show that users can correct 100% of the IDS failures and 88% of the fragment errors made by the tracking algorithm in approximately 10 minutes.

5.1 Future Work

Improved to the ABCTracker system are actively being made with feedback from its users. Luckily the system has many design features in place to accommodate new advances in automated tracking through its modular subroutine representation.

Specific directions for future work include improving the guided correction mode by incorporating new review types for correcting portions of tracks containing poor or uncertain localizations. Guided correction mode could be refining by leveraging further how tracking-by-detection methods operate for ordering guided reviews. Offline models could be trained for specific types of objects or recording conditions and based on user marks (i.e. image patches of the target objects) applied when appropriate. Additionally, foreground classification should be enhanced to handle complex scenarios such as heavy debris and recordings made with moving cameras.

REFERENCES

[1] S. Aslani and H. Mahdavi-Nasab. Optical flow based moving object detection and tracking for traffic surveillance. *International Journal of Electrical, Electronics, Communication, Energy Science and Engineering*, 7(9):789–793, 2013. 1

[2] K. Branson, A. A. Robie, J. Bender, P. Perona, and M. H. Dickinson. High-throughput ethomics in large groups of drosophila. *Nature methods*, 6(6):451–457, 2009. 1, 9, 11

[3] S. Bustamante and A. R. Amarillo-Suárez. Antcounter software: Counting leaf-cutting ants was never so precise, fast and easy. *Journal of Insect Behavior*, 29(3):262–272, 2016. 1, 9, 10

[4] M. Cristani, R. Raghavendra, A. Del Bue, and V. Murino. Human behavior analysis in video surveillance: A social signal processing perspective. *Neurocomputing*, 100:86–97, 2013. 1

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005. 21

[6] T. Fasciano, A. Dornhaus, and M. C. Shin. Ant tracking with occlusion tunnels. In *IEEE Winter Conference on Applications of Computer Vision*, pages 947–952. IEEE, 2014. 22, 23, 41

[7] T. Fasciano, A. Dornhausy, and M. C. Shin. Multiple insect tracking with occlusion sub-tunnels. In *2015 IEEE Winter Conference on Applications of Computer Vision*, pages 634–641. IEEE, 2015. 21, 22, 41, 42, 43

[8] M. Fletcher, A. Dornhaus, and M. C. Shin. Multiple ant tracking with global foreground maximization and variable target proposal distribution. In *Applications of Computer Vision (WACV), 2011 IEEE Workshop on*, pages 570–576, Jan 2011. 26, 27

[9] C. Huang, B. Wu, and R. Nevatia. Robust object tracking by hierarchical association of detection responses. In *European Conference on Computer Vision*, pages 788–801. Springer, 2008. 23, 41, 42

[10] Z. Khan, T. Balch, and F. Dellaert. Mcmc-based particle filtering for tracking a variable number of interacting targets. *IEEE transactions on pattern analysis and machine intelligence*, 27(11):1805–1819, 2005. 26

[11] Y. Li, C. Huang, and R. Nevatia. Learning to associate: Hybridboosted multi-target tracker for crowded scene. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2953–2960. IEEE, 2009. 41

[12] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli. Swistrack - a flexible open source tracking software for multi-agent systems. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4004–4010, Sept 2008. 1, 9, 10

[13] D. P. Mersch, A. Crespi, and L. Keller. Tracking individuals shows spatial fidelity is a key regulator of ant social organization. *Science*, 340(6136):1090–1093, 2013. 1

[14] H. Nguyen, T. Fasciano, D. Charbonneau, A. Dornhaus, and M. C. Shin. Data association based ant tracking with interactive error correction. In *IEEE Winter Conference on Applications of Computer Vision*, pages 941–946. IEEE, 2014. 9, 23, 48

[15] C. Poff, H. Nguyen, T. Kang, and M. C. Shin. Efficient tracking of ants in long video with gpu and interaction. In *Applications of Computer Vision (WACV), 2012 IEEE Workshop on*, pages 57–62, Jan 2012. 9

[16] G. M. Rao and C. Satyanarayana. Visual object target tracking using particle filter: a survey. *International Journal of Image, Graphics and Signal Processing*, 5(6):1250, 2013. 7

[17] N. Razin, J.-P. Eckmann, and O. Feinerman. Desert ants achieve reliable recruitment across noisy interactions. *Journal of The Royal Society Interface*, 10(82), 2013. 1

[18] J. Saragosti and D. J. Kronauer. Animal behavior: The truman show for ants. *Current Biology*, 23(13):R568 – R570, 2013. 1

[19] T. D. Seeley. *Honeybee democracy*. Princeton Univ. Press, 2010. 1

[20] A. W. M. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1442–1468, July 2014. 1

[21] D. Surie, B. Baydan, and H. Lindgren. Proxemics awareness in kitchen as-a-pal: Tracking objects and human in perspective. In *Intelligent Environments (IE), 2013 9th International Conference on*, pages 157–164, July 2013. 1

[22] C. Vondrick, D. Patterson, and D. Ramanan. Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, pages 1–21, 2012. 10.1007/s11263-012-0564-1. 1, 8, 9

[23] C. Vondrick and D. Ramanan. Video annotation and tracking with active learning. In *Advances in Neural Information Processing Systems*, pages 28–36, 2011. 1, 9, 10

[24] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13, 2006. 1

[25] J. Yuen, B. Russell, C. Liu, and A. Torralba. Labelme video: Building a video database with human annotations. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1451–1458. IEEE, 2009. 1, 9, 10