



Department of Computer Science

# Video processing for improved tracking of ants in colony emigration experiments

Seth Alexander Zea

---

A dissertation submitted to the University of Bristol in accordance with the requirements of  
the degree of Master of Science in the Faculty of Engineering

---

September 2017 | CSMSC-17



0000042659

# **Declaration:**

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Seth Alexander Zea, September 2017

# Executive Summary

This thesis reviews the tracking software BrisTracks and different tracking algorithms to test their effectiveness with videos of *Temnothorax albipennis*' migration from old nest to new nest. As a result of this survey, it was concluded that tracking hundreds of featureless blobs is a difficult task which will result in tracks being lost or ids of blobs switched when these come in contact. Therefore, this project proposes a different approach to tracking *Temnothorax albipennis*: detect when an error has occurred, reset the tracker and create tracklets, re-join these tracklets after the tracking has taken place, use different parameters such as prediction of movement and behaviour to increase the accuracy. To get to this conclusion, the following was investigated:

- An in-depth study of *Temnothorax albipennis* to understand their behaviours.
- A thorough study of OpenCV and C++ to understand BrisTracks. Various Computer Vision techniques to begin thinking of possible solutions.
- Python (with its subsequent data analysis libraries such as: Pandas, SciPy and Matplotlib) for data analysis, visualization and to implement various prototypes. MATLAB to test a fruit-fly tracker.
- Trigonometry which aids in the creation of a tandem-run Python function to classify a leader from a follower.

## Main Contributions:

- A report on the performance of various OpenCV tracking algorithms.
- A Python class which successfully classifies a leader ant from a follower from files of tracks.
- A semi-automated tracker (written in Python using the OpenCV library) which allows for the gathering of accurate data to test various implementations regarding behaviour.
- A prototype of a failure detector in tracking which resets when detection has occurred; thus producing tracklets instead of tracks.
- A prototype of a function to re-join these tracklets based on: prediction and behaviour.

**Results and Evaluation:** The results gives us confidence that one can extract behaviours from files of ant tracks and use these as a parameter to improve the accuracy of tracking *Temnothorax albipennis*. The evaluation shows that the work completed is a “fresh” start to tracking featureless blobs: use the fact that these blobs display behaviours instead of just tracking their features.

# *Acknowledgements*

I would like to express my gratitude for the support and advice from Dr. Oliver Ray: for inspiring people to work at their best. I would like also to thank Dr. Neill Campbell: for his endless support, great advice, inspiration and friendship. I would like to thank the members of the Bristol Ant Lab: for their feedback, for the time they spent listening to us.

I would also like to thank the many lecturers at the Bristol University Department of Computer Science.

I would also like to thank my friends here in the UK for helping, supporting and providing me with advice through the difficult times in the last months. These have been some of the hardest times: thanks for providing a shoulder to lean on to.

Lastly, I would like to thank my parents: for being an endless source of inspiration and support. My brother: for teaching me everything I know.

# Contents

<b>Executive Summary</b>	<b>i</b>
<b>Acknowledgements</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Aims and Objectives . . . . .	1
1.2 Areas of Study . . . . .	1
1.3 Added Value . . . . .	2
1.4 The Outline of the Thesis . . . . .	2
<b>2 Project Background</b>	<b>4</b>
2.1 What is teaching? . . . . .	5
2.2 Why is this research important? . . . . .	6
2.2.1 How has this been acknowledged? . . . . .	7
2.3 The Beginnings of the Project . . . . .	8
2.3.1 A quick glance at <i>Temnothorax albipennis</i> . . . . .	8
2.3.2 <i>Temnothorax albipennis</i> and its behaviours . . . . .	9
2.4 Video and Tracking in Computer Vision . . . . .	10
2.4.1 How does a computer track objects? . . . . .	11
2.5 The brief history of BrisTracks . . . . .	12
2.5.1 The birth of BrisTracks . . . . .	12
<b>3 Preliminary Investigations</b>	<b>14</b>
3.1 BrisTracks . . . . .	14
3.1.1 Social-Carry . . . . .	16
3.1.2 Tandem-Run . . . . .	16
3.2 OpenCV Investigations . . . . .	17
3.2.1 C++ and Python . . . . .	18
3.2.2 Detection . . . . .	19
3.2.3 Tracking . . . . .	19
3.2.4 BOOSTING and MIL . . . . .	20
3.2.5 MEDIANFLOW . . . . .	21
3.2.6 KCF . . . . .	21
3.2.7 Inside the Nest . . . . .	22
3.2.8 Other Trackers . . . . .	24
3.2.9 Kalman Filters and the KuhnMunkres Algorithm . . . . .	25
3.2.10 Conclusion of Preliminary Investigations . . . . .	28
<b>4 Tools for Gathering and Analyzing Data</b>	<b>30</b>
4.1 How to Tell if It is a Tandem-Run? . . . . .	30
4.1.1 Using Past Frame Information and Tracks . . . . .	30
4.1.2 Conclusion of the Tandem-Run Experiment . . . . .	34

4.2	Semi-Automated Multi-Tracking . . . . .	34
4.2.1	Multi-Object Tracking for Python . . . . .	35
4.2.2	Conclusion to the Multi-Object Tracker in Python . . . . .	36
4.3	Automating the Tracker, Detecting Errors and Behavioural Tracklets . . . . .	36
4.3.1	Detecting Errors . . . . .	37
4.4	Tracklets . . . . .	39
4.4.1	Predicting movement . . . . .	40
4.4.2	Cost Function . . . . .	41
<b>5</b>	<b>Results</b>	<b>43</b>
5.1	Tandem-Run Class . . . . .	43
5.1.1	Results for 2 ants files . . . . .	44
5.1.2	Results for 3 ants file . . . . .	44
5.1.3	Observations of the Experiment . . . . .	44
5.1.4	Conclusion to the Tandem-Run Experiment . . . . .	44
5.2	Rejoining Tracks . . . . .	45
5.2.1	Results for Automatic Restart . . . . .	45
5.2.2	Results for Euclidean Join (No Prediction) . . . . .	45
5.2.3	Results for Euclidean Join (With Prediction and Cost Function) . . . . .	46
5.2.4	Observations of Rejoining Tracks . . . . .	46
5.3	Rejoin tracklets with Tandem-Run . . . . .	46
5.3.1	Results . . . . .	47
5.4	Conclusion to Results . . . . .	47
<b>6</b>	<b>Evaluation</b>	<b>49</b>
6.1	Completion of Objectives . . . . .	49
6.1.1	Tandem-Run Classification . . . . .	49
6.1.2	Automating the Restart of Trackers . . . . .	49
6.1.3	Behavioral Tracker . . . . .	50
6.2	Evaluating the Methodology . . . . .	51
6.3	The Project's position in Research . . . . .	52
<b>7</b>	<b>Conclusion and Future Work</b>	<b>54</b>
7.1	Future and Work . . . . .	54
7.1.1	Video Quality . . . . .	54
7.1.2	OpenCV Newer Version . . . . .	54
7.1.3	Social-Carry . . . . .	55
7.1.4	Cost Function . . . . .	55
7.1.5	Nest Tracking . . . . .	55
7.2	Conclusion . . . . .	56
<b>A</b>	<b>Code Samples</b>	<b>58</b>

**Bibliography**

**62**

# 1. *Introduction*

This chapter will present the project and outline the structure of the thesis.

## 1.1 Aims and Objectives

The aim of this project is to use past frame information from files of ant species *Temnothorax albipennis*' tracks to help classify the ants' social behaviour: Tandem-runs. Once the social behaviour is correctly classified; this, in turn, will be used to help improve the accuracy of ant tracking softwares' (such as BrisTracks) when they fail or when they begin to lose tracks. Current programs can accurately track ants over long periods of time, others, such as BrisTracks, can even disambiguate when two ants are in close proximity. However, these programs often fail due to occlusion, clustering or when these ants have exactly the same size and shape. The result of these failures is that the data provided by the trackers is often inaccurate and unusable for lab experiments.

These aims will be achieved by fulfilling the subsequent objectives:

1. To develop and experiment with a novel way of classifying the social behaviour: tandem-run.
2. To develop a tool which will aid with the gathering of data (such as tracks of two ants while they are tandem-running) by exploring the multiple classes and functions in the latest version of OpenCV.
3. To explore, test and develop a prototype for error detection in automated trackers.
4. To demonstrate and propose a different way of tracking social insects which will be based on using their behaviours as part of a cost function for a future insect tracker.

## 1.2 Areas of Study

The following areas will be researched by this project:

1. Computer vision as a discipline on its own.



2. Current methods in the latest version of OpenCV for object detection and tracking. Alongside this, the use of C++, Python, Object Oriented Programming and industry standard practices for developing open-source code.
3. Data analysis methodologies such as: wrangling, cleaning and visualizing.
4. Biology as an overarching subject to better understand the behaviours of *Temnothorax albipennis*.
5. Various tracking softwares, including BrisTracks, where they fail and how these can be used to their full advantage.

### 1.3 Added Value

The project will propose a novel technique in the field of computer vision and biology to try to improve the accuracy of gathering data from *Temnothorax albipennis* nest migration. One of the immediate added values gained from this project is that the results acquired from the various experiments will have benefits for the Bristol Ant Lab and potentially other developers/biologists who wish to expand the project to suit their needs.

For the lab, the biologist will benefit since there will be a novel way of classifying ants engaged in tandem-runs and there will be a semi-automated piece of software to gather data. With this semi-automated software, the user can observe ants, track them and use the results to analyze the data and, hopefully, to test hypotheses.

For the computer vision tracking community, there will be a proposed method of tracking (not only of *Temnothorax albipennis* but any other social animal which displays marked behaviours) which involves using their marked behaviours as a parameter for building a cost function suited to analyze the animal's activities.

### 1.4 The Outline of the Thesis

**Chapter 2: Project Background** will touch upon one of the main motivations for conducting these experiments and will try to engage the reader with deeper questions such as the meaning of teaching and learning in the animal kingdom. It will also describe the beginnings of the ant tracker BrisTracks to better understand the necessities of these computer programs in laboratory settings such as the Bristol Ant Lab.

**Chapter 3: Preliminary Investigations** presents the reader with an investigation of BrisTracks and where it fails at gathering accurate data for laboratory analysis. After that, an investigation of OpenCV tracking algorithms will be discussed and how these perform under pre-recorded videos of *Temnothorax albipennis* nests and migrations. This chapter also introduces a MATLAB implementation of a fruit-fly tracker and will highlight the best features that can be extracted from said program. Lastly, this chapter finishes with a conclusion and justification for exploiting the behaviours of social insects as a means to improve tracking.

**Chapter 4: Tools for Tracking and Analyzing Data** describes the implementation of the tandem-run Python class which determines if an ant is a leader or a follower. This chapter also outlines the implementation of the semi-automated tracking software which aids in gathering tandem-run tracks to evaluate the tandem-run Python class. This chapter also introduces a way in which an automated tracking software can detect errors and thus reset upon detection to help with accuracy by creating “tracklets” instead of continuing tracking wrong objects.

**Chapter 5: Results and Evaluation** will display the data collected and the ways in which it was analyzed and experimented with in order to find percentages of accuracy. By running various types of experiments, this section will conclude with a new, improved way of tracking social-insects. The ideas and hypotheses proposed by the results of this chapter will, then, be evaluated with various staff from the Lab.

**Chapter 6: Conclusions:** summarizes the project’s main contributions and outlines some of the work that can be carried into the future.

## 2. *Project Background*

In the animal kingdom, interactions between members of the same species are some of the most crucial in the animal's life: behaviours such as parental care, defense, gathering resources, exploring and mating. However, the complexity of these behaviours, which generally involve multiple members, makes the quantitative study of these interactions difficult for scientists [1]. There is a need for scientists/researchers of social behaviours to benefit from the advances in automated measurement of social interactions. One of these benefits could be the detection of patterns that may prove to be too subtle for biologists to observe.

Social interactions happen over different temporal and spatial scales, and vary in complexity [1]. Definitions of behaviours also vary; one notable definition is by Sokolowski in which she argues that one individual of the same species influences the behaviour of another individual [2]. This definition holds true when compared with the extensive literature in the study of the social behaviours, and interactions, of the ant species *Temnothorax albipennis* conducted by Professor Nigel Franks and his team at the Bristol Ant Lab.

Automating the analysis of social behaviours is, currently, the task of the Bristol Ant Lab's sub branch, the Computational Ant Lab. There are many different approaches in which the Computational Ant Lab is helping with the analysis of these social behaviours: this project focuses on video collection, tracking and classifying behaviours. Automating the analysis of social behaviours has a number of requirements: recording the behaviour, tracking the position of the individuals and their interactions, recognizing these individuals and their interactions across time and space, and quantifying these interactions and their patterns [1].

The project's main topic of interest is teaching and learning in the animal kingdom. It is from this main topic where the inspiration to carry out the research on how to improve the Bristol Ant Lab's tracking software comes from. To better illustrate the underlying debate in the subject of teaching, a brief explanation of what teaching is, according to the literature, will be given. A brief summary on why doing this kind of project is necessary, specifically for the researchers in the field, will follow.

## 2.1 What is teaching?

Much debate on whether teaching is solely a human endeavor that can only be done by humans is still underway. One possible definition of teaching (or, perhaps, what a teacher represents) is explained by Caro et al, who argues that “An individual actor A can be said to teach if it modifies its behavior only in the presence of a naive observer, B, at some cost or at least without obtaining an immediate benefit for itself” [3]. This definition already sets a 4 point standard on what a teacher should be; an individual is a teacher if: “1) it modifies its behaviour in the presence of a naive observer, 2) at some initial cost to itself, 3) in order to set a example, 4) so that the other individual can learn more quickly” [4]. One early champion for humans and teaching is introduced in the famous 1978 paper *Does the chimpanzee have a theory of mind?* by Premack et al. In this paper, the general consensus is that evaluation “presupposes an image or mental representation of a standard” [5]. Under this consensus, “teaching still looks to be unique to humans” [4].

However, because of the work by Richardson et al in the 2007 paper *Teaching with Evaluation in Ants*, substantial evidence has been provided to argue that teaching is not solely enshrined to humans [6]. Ants, they argue, display the best, if not the only, model for teaching and learning in the animal kingdom apart from humans. Their “well-controlled” [7] experiments along with arduous and time consuming experiments of the ant species *Temnothorax albipennis* not only tries to disprove the general consensus of teaching and evaluation, they also improve Caro et al’s definition of teaching by adding an extra criterion: 5) “Teaching also involves bidirectional feedback between teacher and pupil” [4]. As it turns out, *Temnothorax albipennis*, when it migrates from an old nest to a new nest, exhibit a peculiar social behaviour called tandem-running (which is “a form of recruitment in which a single well-informed worker guides a naive nestmate to a goal”) [6]. In tandem-running *Temnothorax albipennis*’s leaders successfully meet all of the 4 steps required to become a teacher as challenged by Caro et al. The follower reinforces the fifth point where teaching involves bidirectional feedback between teacher and pupil. Thus leader becomes teacher and follower becomes pupil.

Despite the fantastic discoveries about teaching in *Temnothorax albipennis* while tandem-running, one question still lingers: is this “teaching”, as viewed from a human perspective, or is this a mere transfer of information? Leadbeater et al, at the end of the article *Social Learning: Ants and the Meaning of Teaching* argue that it “would appear rather useful to reserve distinct terms for distinct types of information exchange between animals” [7].

The debate over the definition of teaching in animals and how it could differ from humans is an everlasting argument which a great deal of time and many disciplines. This project is not concerned with trying to find the definition of teaching; rather it tries to provide a tool for biologists to analyze the behaviours of *Temnothorax albipennis*. Richardson et al's experiments of *Temnothorax albipennis*, for example, have provided the scientific community with the beautiful picture that sophisticated outcomes can be reached through simple rules, where these rules are the bio-directional feedback between leader and follower ant while they emigrate. Trying to achieve conclusions such as Richardson's by providing a tool to gather the subtle information that these fascinating insects may provide, is the desired goal.

## 2.2 Why is this research important?

Despite the wonderful discoveries from *Temnothorax albipennis* and their simple rules, much research on *Temnothorax albipennis* is still taking place in the Bristol Ant Lab under various talented researchers in animal behaviour and biology. However, these experiments usually take place in the Summer as *Temnothorax albipennis* cannot be left in the labs for longer periods of time as they will start to get used to the lab settings. Once the ants are accustomed to living under lab conditions, by having a constant supply of food and shelter, the experiments could become compromised as *Temnothorax albipennis* will not exhibit its natural behaviours.

The biggest drawback that biologists face at the Bristol Ant Lab in particular (but also biologists in many other disciplines which study social behaviours in general) is that the majority of these experiments which involve multiple animals coming in close proximity to display their social behaviours, are non-automated and time consuming. Whether biologists are studying bees, termites, flies, larvae, worms, or mice; these animals often interact with their conspecific. Biologists frequently have to observe a multitude of species for long periods of time, often having to keep track of a large number of individuals, in order to gather data about their interactions and their behaviors. For example, at the Bristol Ant Lab, where biologists are working with populations of ants as large as 200+ individuals, gathering data takes a very long time. This is a big problem, as biologists should be spending more time analyzing data rather than gathering it.

Pre-recording these experiments is a great attempt at distributing the work of gathering and analyzing data, especially as *Temnothorax albipennis*'s migrations take hours to finish. The worst case, however, is if something goes wrong during the experiments and the fault is not caught until it is analyzed in these pre-recorded videos; biologists face the possibility

of wasting time by having to drive to the Devon coast in the UK, to gather a new set of ants to bring to the lab to resume the experiment. It hasn't happened, but it is still a possibility that could be avoided by having some form of automated system to guard against most possible scenarios. Lastly, as time is of the essence, automating part of the research will buy invaluable time to either conduct more experiments, to test a new hypothesis or to analyze trivial data.

### 2.2.1 How has this been acknowledged?

The Computational Ant Lab is a great example of an attempt to bridge the gap between the “analysis, simulation and automation of scientific experiments relating to collective decision making in ant colonies” [8]. The goal of the Computational Ant Lab is to merge the use of computers (and/or computerized techniques) to the experiments of ants in order to further the studies of these. By doing so, the Bristol Ant Lab benefits fully in that more data is provided to them; thus allowing them to conduct more experiments in the small windows of time available and hence enabling them to test their hypotheses more quickly with potentially more data being gathered.

One of the outcomes from the Computational Ant Lab in 2016 was a promising new piece of software for tracking multiple objects (ants) in pre-recorded videos called BrisTracks (an extension of the Swistrack framework based on the OpenCV library). According to its author, BrisTracks outperforms other tracking software in that not only is it excellent at tracking multiple objects, it is also open-source, which makes it essential for developers to manipulate its code to better suit specific job needs [9]. BrisTracks is the first step at automating one portion (the video portion as there are other projects involving robots and simulations) of the whole Computational Ant Lab's vision of automating ant experiments. It is a great first step, but it is unsuitable for use in experiments as it currently fails to track multiple ants in close proximity.

As it stands, BrisTracks tracks ants in close proximity on a single-frame resection basis [9], meaning that the software takes one frame at a time and decides whether there are ants touching one another and splits them according to the algorithm provided. This works fine when there are two ants of the same size in the frame, but as will be shown in the section “The shortcomings of BrisTracks”, there are multiple cases where this fails.

## 2.3 The Beginnings of the Project

This section will delve deeper into the study of *Temnothorax albipennis* as a field of study. It will demonstrate why there's a need for the software, the opportunities and limitations at hand.

### 2.3.1 A quick glance at *Temnothorax albipennis*

The project's main topic of interest is teaching and learning in the animal kingdom, and the project's intention is to increase the accuracy of the ant tracking software BrisTracks to help simplify research conducted by the scientists at the Bristol Ant Lab. Though teaching in the animal kingdom is the main driving force behind the project, talking about ants cannot be done without a few words on complexity.

One of the earliest, and definitely most enjoyable, examples on complexity spawning from simple rules in computer science and mathematics is the famous Conway's "Game of Life". The basic idea of the game is to observe how organisms change over time when the player (or in this case, the programmer) applies Conway's "generic laws" for births, deaths and survivals [10]. The rules for the "Game of Life" are simple: there's an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead, or "populated" or "unpopulated" [11]. Each cell will interact with its neighbours and therefore abide by the rules established at the beginning of the game. The game is known as a zero-player game, meaning that its evolution is based on its initial state and no other input is required.

Conway's "Game of Life" is just a small example of complex behaviour happening from a determined set of initial rules. Yet another more relevant example of the interest of people for complexity based on simple rules, is from Bonabeu et al's 1997 paper *Self-organization in social insects*. Just like Conway's "Game of Life", Bonabeu's paper argues that "interactions among simple individuals can produce highly structured collective behaviours" in animals, specially in social insects [12]. These two examples bring onto the table an old, yet deep interest, for the study of complexity based on simplicity.

Yet the applications that understanding complexity could have in today's main challenges are based on studies such as the one conducted by the Bristol Ant Lab. Take for example Hauert et al's *Ant-based swarming with positionless micro air vehicles for communication relay*. In this paper, Hauert et al utilize the discovery of the fact that an army of ants is capable of laying and "maintaining pheromone paths leading from their nest to food sources in nature" to deploy communication pathways between multiple users in disaster

zones [13]. The paper states that the main “inspiration” to set up and deploy these pathways comes from watching ants [13]. Understanding how their “simple” behaviours lead to complex organization does not only lead to applications and materializing products for today’s problems, it could also lead to a deeper understanding about us humans. Take, for example, Christensen et al’s idea that *Temnothorax albipennis* complexity is similar to urban living. “Prediction for social systems is a major challenge” [14] they argue, but the study and understating of these ants could lead to better ways of organizing and maintaining social living in urban communities.

This research focuses on the ants’ behaviours while they are migrating, since understanding these behaviours is the main current interest in the realm of teaching/learning for the biologists at the lab. Extensive non-automated research on *Temnothorax albipennis* and its behaviours has been carried out through the years. The next subsection will give a brief survey about the *Temnothorax albipennis*’s behaviour when they are migrating.

### 2.3.2 *Temnothorax albipennis* and its behaviours

Doran et al in *Economic investment by ant colonies in searches for better homes* states that the number of *Temnothorax albipennis*’s scouts in a given time has a direct correlation with the quality of their nest [15]. As such, *Temnothorax albipennis* is “pushed” to seek for better homes if the quality of their nest is impaired or if they sense there is some threat to the greater well-being of the colony. Once a scout ant finds a suitable nest for the colony to “migrate” to, the scout ant will return to the old nest to bring another “naive” ant to the new nest via a peculiar social behaviour identified in the literature as tandem-running. Tandem-running happens when a naive follower ant follows a leader scout within touching distance to the new nest. The follower ant needs to reinforce its presence to the leader ant within a time frame or else, the leader will assume that the tandem-run has been broken and will return to the old nest to find another recruit. It is this bidirectional feedback, alongside the fact that the leader ant modifies its behaviour in the presence of a follower, which not only becomes relevant and important to the biologist, but it will also become a key feature in the following sections.



FIGURE 2.1: An illustration of a tandem-run by a different species of ant. Notice the antennae touching the leader ant [16].

As more and more ants occupy the new nest, this reaches a “quorum”, and thus ants who already know the way to and from the nests will perform another social behaviour



commonly referred to in the literature as social-carry. Social-carry, as the name implies, is physically carrying another ant to speed up the process of migration [15]. Thus, the migration process is completed when all of the ants from the old nest are moved to the new nest via these two simple yet interesting social behaviours.



FIGURE 2.2: A picture of a leader ant performing a social-carry. [17].

## 2.4 Video and Tracking in Computer Vision

Vision is something humans excel at. According to Davies in *Computer and Machine Vision*, “the majority of the visual and mental processing tasks that the eye-brain system can perform in a flash have no chance of being performed by present-day man-made systems” [18]. A video is simply a lot of pictures, or frames, displayed one after the other at a continuous fast rate. A typical camera recorder will take about 25 to 30 frames per second, thus creating the “illusion” of movement as one frame is displayed after the other in a timely manner.

For the recording of the videos in the lab settings, special care in optimizing the video quality to make the automated analysis and to improve the quality of that analysis has therefore been taken into consideration. For example, an “arena” has been created by the Bristol Ant Lab where *Temnothorax albipennis* currently resides. There is uniform and sufficient lighting in order to have high contrast in colour and intensity between the ants and the background, there are no reflections being emitted from the surrounding walls (which is very helpful as the tracking software cannot discriminate between a reflection and a real ant). However, one extra optimization which the next set of recordings should take into consideration, is to position the camera in such a way that the whole arena is recorded and not just a section of it; this could eliminate the problem of not having continuously visible ants and thus completely eradicate the fact that some ant tracks are lost due to them exiting the frame.

### 2.4.1 How does a computer track objects?

Computers, on the other hand, are champions at doing the repetitive work that humans find either boring, tiring or not worthwhile. Before the automation of measurement and quantification of social interactions, scientists had to manually track and record these interactions; a task that would take a very long time. Computers do this much much faster, leaving the scientist enough time to analyze the data instead of gathering it. By allowing the scientist to analyze the collected data, there are positive potential outcomes: such as recognizing subtle patterns and reducing human bias.

After the video has been collected, the position of the object or particle (in this case, of the ant) can be tracked in every frame of the video. Most tracking softwares estimate the (x,y) coordinate which belongs to the particle's centroid or geometric center [1]; a centroid is the average position of all the points in the shape. Normally, a tracking software creates a geometric figure around the particle (BrisTracks creates a rectangle) and then finds the centroid to output its location in the current frame. Some tracking softwares can also return information about the object being tracked: such as the direction of the head or the location of its body parts [1]: currently, BrisTracks does not do that and it is something that could be improves in the future.

In order for the tracking to be as error free as possible, great care needs to be taken in order to differentiate the pixels belonging to the ant (foreground) from the arena (background). As mentioned above, the ant's arena plays a major role in ensuring that these videos are made as clean as possible. Most tracking softwares will use a technique called "background subtraction" which estimates the appearance of the arena when there are no particles (ants) present and then subtracts it from the current frame to then threshold the difference [19]. This is the main reason why keeping the arena with constant lighting, free from shadows and reflections, is necessary.



FIGURE 2.3: An image of the arena to be used for background subtraction. Notice the lack of edges or walls. This could be further improved by recording new videos of the complete arena.

## 2.5 The brief history of BrisTracks

BrisTracks, as one could potentially guess from its name, is a tracking software specifically catered for the Bristol Ant Lab and its sub-branch the Computational Ant Lab. Since its initial project conception by Dr. Oliver Ray, the main goal of the project has been to steer away from the proprietary software AnTracks. As preliminary work carried out by Alex Blaza shows, AnTracks proved difficult to work with since its proprietary nature wouldn't allow future developers to properly change the code to cater for specific needs [9] such as working with *Temnothorax albipennis*. For example, research carried out by the Computational Ant Lab showed that AnTracks was not able to disambiguate between interacting ants; since most of the work carried out by the Bristol Ant Lab has to do with ants coming in close proximity (tandem-runs, for example), AnTracks had to be either modified or discarded. Modifying the software proved to be difficult, according to Blaza, since AnTracks is a propriety software and the original developer does not allow for further development. As a result, the Computational Ant Lab decided to push the project in the direction of open-source software and discard AnTracks altogether.

On further research, Blaza found that the open-source Swistrack was the best candidate to be steered in the direction sought by the Computational Ant Lab. According to Correll et al, the creators of Swistrack, the tracking software was conceived in order to be “platform-independent, easy-to-use, and robust” which is specifically tailored to “research in swarm robotics and behavioral biology” [20]. Not only did Swistrack prove to be a better candidate in terms of being open-source, it also outperformed AnTracks in speed [9] of tracking. However, problems associated with AnTracks were still present in Swistrack. For example, both softwares are still unable to deal with ant collisions in an accurate manner, most specifically when *Temnothorax albipennis* engages in tandem-running and social-carry. As a result, Swistrack had to be modified in order to start disambiguating the social behaviours displayed by *Temnothorax albipennis*.

### 2.5.1 The birth of BrisTracks

According to the creators of Swistrack, the tracking software is easy to use and is specifically catered for tracking in lab environments. The software is written in C++, which is a general-purpose programming language that has object-oriented and generic programming features, while also providing facilities for low-level memory manipulation [21]. It uses OpenCV which stands for open-source computer vision library and it is an open-source computer vision and machine learning software library specifically built to provide a common infrastructure for computer vision applications and to accelerate the use of

machine perception [22]. These two combined create a comprehensive library that drives the majority of the Swistrack components. Swistrack is easily extensible through the use components (plug-ins) which work in the same way one would lay bricks or Lego pieces; that is, each component needs to be laid on top of the other, in an ordered manner, for the software to properly work [23]. It has a user friendly interface which allows non-computer savvy people to quickly get their favourite videos loaded into the software to start using the tracking features.

In order to serve the Computational Ant Lab and the Bristol Ant Lab, Swistrack has been heavily “refactored” (a software development technique which is the process of restructuring existing computer code), and some of its components have been taken away since they do not serve any purpose for the task of tracking these particular ants. Many other user interface methods have also been added, such as a useful Superzoom function which allows the user to get very close to the blobs in question.

### 3. *Preliminary Investigations*

Because the intention of BrisTracks is to accurately track *Temnothorax albipennis* by disambiguating the social behaviours of tandem-running and social-carry, this section will show how BrisTracks attempts to disambiguate these and where it falls behind in doing so. Then various OpenCV algorithms are tested with the pre-recorded videos of migrating ants to find the best candidate for building a semi-automated tool for gathering data. After that, other techniques in detection and tracking are tested. Finally, a conclusion that tracking based on features alone is not enough when trying to track hundreds of objects which, one way or another, have similar features and are indistinguishable.

#### 3.1 BrisTracks

As it was explained earlier, BrisTracks uses a series of components which, if laid in an orderly manner, will output the desired results. The first component in the chain is called “Input from AVI file”; this is where the user selects which video it wants to use. The second component in the chain is called “Conversion to Color”; here, every input image (in this case each frame from the video) is converted to RGB (Red, Green and Blue) for further processing. Then comes the component “Background Subtraction”; this is essential as it subtracts the background image from the input image. What is really good about this technique (specially for these types of lab experiments) is that if the camera is fixed (which ours is) only the particles added to the scene remain after the background had been subtracted. Last of the basic components is “Threshold with Independent Threshold Values (color)”; this component separates the pixels that belongs to tracked particles, then it converts the input image into a binary black or white having white for objects and black as background. As one can see from Figure 3.1, this “pipeline” process creates white blobs for the particles to be tracked. White blobs will have a determined size, area and circumference depending on the object filtered.

Once the input image has been converted to a binary image, BrisTracks uses a component called antDetection which allows the user to input a specific parameter in order to identify which is a tandem-run and which is a social-carry by selecting how “compact” each blob should be. This component takes the area of the blob and divides it by its circumference (since a circle is the most compact figure possible); the result is a number varying in compactness. To illustrate this idea, let us take the examples of a tandem-run and a social-carry: when an ant is following another ant on the rear, one can see that

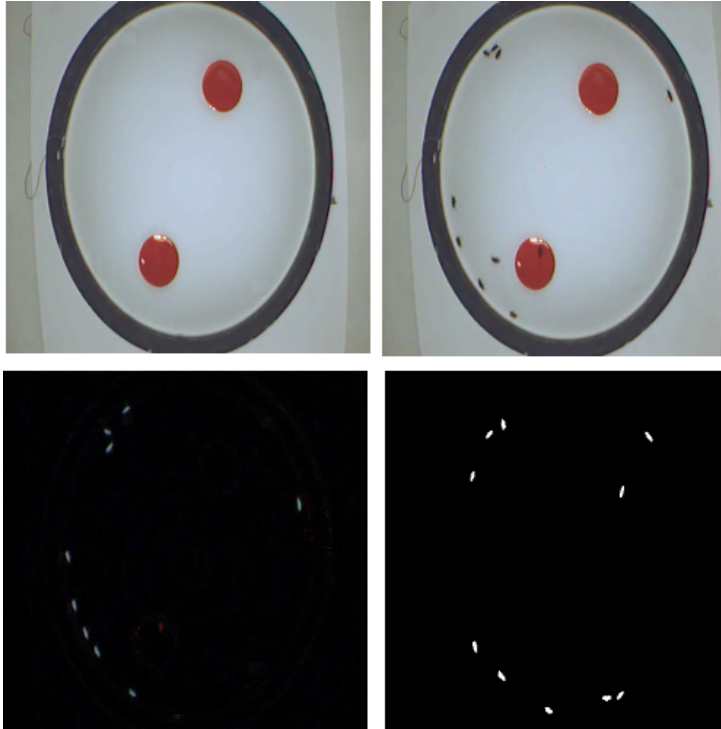


FIGURE 3.1: From upper left corner to bottom right corner: 1) the background image to be subtracted. 2) the frame where the background subtraction takes place. 3) the resulting image after the background subtraction. 4) after Threshold with Independent Threshold Values; notice what is left behind is a simple binary image of white and black.

the “compactness” is less than, say, two ants in absolute close proximity because one is carrying the other Figure 3.2. What is more compact: a line of three students or three students hugging? The same principle applies in this method: the more compact pixels there are, the closer it is to being a social-carry; the same applies inversely. Nevertheless, one can quickly see that although this is a very nice way to disambiguate between two social behaviours, there are still cases where this method does not apply as a whole.

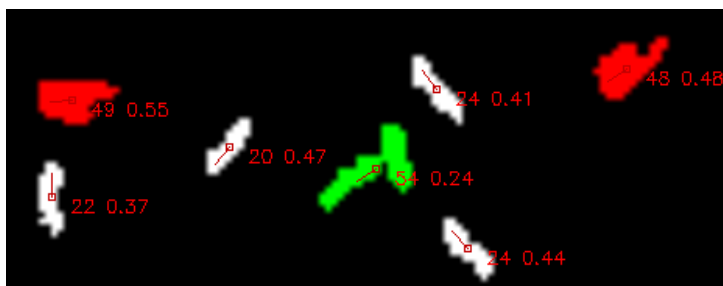


FIGURE 3.2: White blobs are single ants, Red blobs are social-carry and Green blobs are tandem-runs

Also, another problem found whilst testing BrisTracks is that the user needs to know the exact number of blobs he/she wishes to track. This can quickly become problematic as ants tend to come in and out of, not only the video frame, but the nest which is fully covered. Tracks, in BrisTracks, are not created or destroyed after the program has started; therefore, at time when the tracks leave the scene, these just linger at the edge of the screen and are sometimes picked up by a different blob. This creates a problem as the same id for that particular track is assigned to a different individual.

### 3.1.1 Social-Carry

For BrisTracks to be accurate and be fully functional in a lab setting, there needs to be a way for it to count, track and display how many social-carries happen during a full fledged migration. Upon further exploration of one of the videos, it became obvious that, when presented with “dubious” scenarios, BrisTracks did not perform accordingly. One of these “scenarios” has a direct correlation with ant size; there have been many cases in the preliminary studies that point to a social-carry performed by two very small ants; thus rendering the compactness of a single ant and ignoring the disambiguation. Figure 3.3 and Figure 3.4 show how two small ants performing a social-carry would flicker into a properly tagged social-carry and then as a single ant. This problem arises because this blob’s compactness sits right at the threshold. The flicker happens as the ant carrying the other ant turns; thus increasing the compactness as the angle of the facing ant changes.

Another investigation showed that BrisTracks wrongly tagged a big working ant as a social-carry since the compactness of this single ant was beyond the threshold. Again, because the threshold is manually set before the experiment takes place: the computer will treat anything that surpasses that threshold as a social-carry.

One last downfall encountered during the preliminary studies was also a common problem that many biologists encounter trying to automate this kind of experiment: the problem with swapping ids whenever multiple individuals come into contact [24], [25], [26], [27]. This is a big problem, and probably the most challenging one, as it is well known that *Temnothorax albipennis* is in constant proximity with its conspecifics.

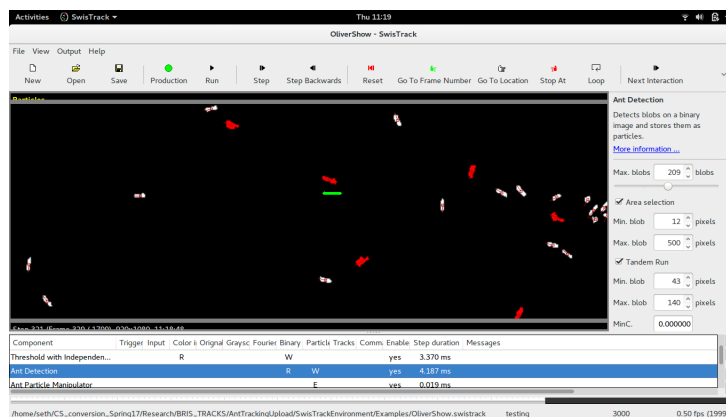


FIGURE 3.3: Social-carry properly tracked. Notice the ant(s) above the green line.

### 3.1.2 Tandem-Run

The preliminary studies found that BrisTracks treats every single ant collision as a tandem-run. As one can see from Figure 3.5, there is a collision of two social-carries wrongly

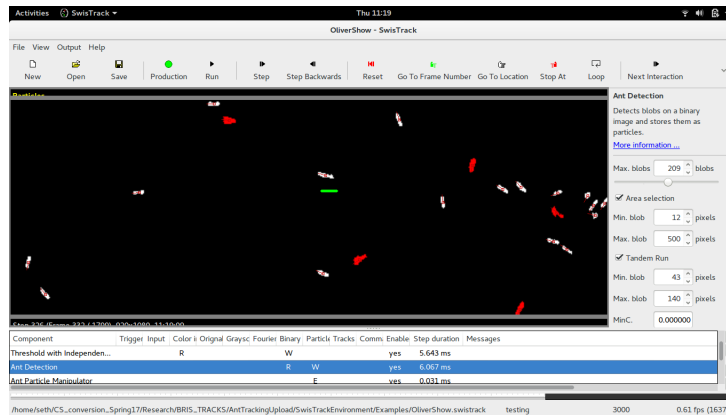


FIGURE 3.4: Social-Carry lost track. Notice the ant(s) above the green line.

tagged. From the literature surveyed so far, there hasn't been a single indication that social-carries engage in tandem-runs; this wrongful tag will therefore lead to unreliable data to the scientists.

Another important discovery performed in this preliminary study has to do with what Richardson et al described in their paper as “giving-up-time” [6]. Giving-up-time is the time a leader ant is willing to wait for the follower ant to touch the leader's rear end with its antennae. If the follower ant takes a long time to make contact (one leader ant was willing to wait 1 minute in a tandem-run that normally takes 2-3 minutes [6]), then the leader ant will return to the nest. Upon analyzing some of the videos of the migration, it was clear to see the ant leader waiting for the follower ant to proceed with touching the leader with its antennae. This discovery will have to be discussed with the biologists at the Bristol Ant Lab as they are the ones who will decide whether implementing a feature to analyze this behaviour is necessary.

Tandem-runs are, perhaps, the most important behaviour to be analyzed as they demonstrate the idea of teaching discussed in Chapter 2. Accurately identifying, tagging and tracking these behaviours is imperative. There are various ways that this can be achieved (the previous version of BrisTracks demonstrated one solution by tagging all collisions as tandem-runs). To properly implement a new version of the tandem-run in BrisTracks, further discussion with the biologists at the Bristol Ant Lab will have to take place.

## 3.2 OpenCV Investigations

As it was stated above, all of the difficult procedures which SwiTracks (and subsequently BrisTracks) performs when it comes down to object detection and tracking is carried out by the heavily used and well-documented computer vision framework OpenCV. The latest version of SwiTracks is number 4; this version relies on the OpenCV version 2.0 and its



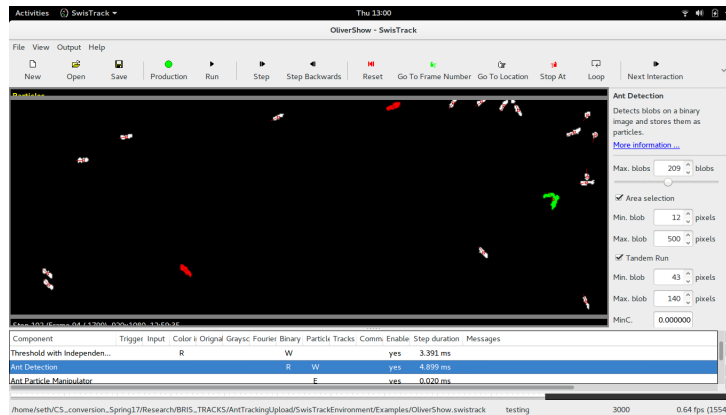


FIGURE 3.5: Two social-carries colliding. The output is a tandem-run

following updates 2.1 and so on; but does not include OpenCV 3.0. This is important to note as there have been many changes in OpenCV starting with version 3.0 which should be appropriately investigated in order to take full advantage and exploit the potential of the features which OpenCV has to offer. Some of these features include ways in which the developers have incorporated background subtraction alongside their object detection class: thus making sure that these two features are standard and well used throughout; it also means that they run parallel with one another and are kept as robust as possible. Yet, the most interesting and perhaps, most promising new additions of OpenCV 3.0 are the introduction to new tracking algorithms such as the Convolutional Neural Network (CNN) and the well tested Kernelized Correlation Filters (KCF) which has proved to be quick and efficient.

As a result, this section will delve deeper into ways in which OpenCV was explored in order to investigate how BrisTracks (or perhaps any other Ant Tracker) classification accuracy and tracking could be further improved by incorporating the various ways of extracting and analyzing the data provided by the trackers.

### 3.2.1 C++ and Python

Swistracks' alongside the features introduced by BrisTracks and its classification classes' are all written in C++. Some time was devoted to learning and understanding how these classes worked together. However, since one of the objectives was to investigate OpenCV 3.0, and yet, another objective was to keep the code as clean and as user friendly as possible (for some of this code is eventually going to be used by scientists), Python seemed like a suitable candidate to perform the following experiments using OpenCV 3.0. Further study into Python showed that it is well suited for scientific purposes as it tries to emulate as close as possible what MATLAB does. Python, as opposed to MATLAB, is open source. There are many packages and libraries which Python possesses, some of which proved to

be essential when we had to analyze data, wrangle the data, clean the data and use the data for our own benefit. Some of these packages include: NumPy, Pandas and Matplotlib [28], [29], [30].

### 3.2.2 Detection

One of the first limitations found whilst conducting the preliminary studies on BrisTracks was that the user had to know, beforehand, how many ants/blobs were going to be tracked; an accurate number of blobs needed to be known before any tracking would take place. This alone presents a problem to the user since there are many variables that can change whilst conducting a tracking experiment. One of the most common things that could happen is that ants go inside the new nest and stay there for an undefined period of time. Because the new nest needs to be covered by keeping the nest dark and obstructing light to help with humidity, there is no way an overhead camera could potentially pick up any ants that reside in the new nest. As BrisTracks can neither create nor destroy new tracks, any ant track that walks inside the new nest just hangs there to be later picked up by anything that comes across its path. This already creates a slight problem as the current software cannot distinguish when a new blob enters the field and when it leaves; thus rendering wrong tracking data as the experiment is conducted.

To tackle this problem, various methods of blob detection had to be carefully studied under the OpenCV version 3.2.0. The most common and widely used blob detection is the class `cv2.SimpleBlobDetector_Params()`. As one can see from Figure A.1, there are various parameters which the user needs to input in order to best detect the desired blobs. Note that perhaps the three most important parameters which needed to be tested over various videos of ants were: area, circularity and inertia. These three parameters proved to be crucial as the videos which were provided to evaluate this experiment had ant sizes of an area of 10 pixels. The inertia was trivial to figure out since these ants are elongated and have a low circularity; meaning, they resemble an ellipse than a circle.

### 3.2.3 Tracking

As Satya Mallick points out in the article *Object Tracking using OpenCV (C++ / Python)*, tracking “is faster than detection, can help when detection fails and it preserves identity when detection does not” [31]. As some of the investigation proved, tracking more than two blobs by using detection alone starts to become problematic since detection works on a frame by frame basis, meaning that identities are not preserved and are thus subject to being swapped or lost (in scenarios when blobs are occluding one another). Hence,

OpenCV provides various tracking algorithms which make tracking blobs a lot more accessible and scalable as some of these algorithms take advantage of mathematical formulas or past-frame information to deduce either the trajectory of the blob or the location in which that particular blob might be frames later.

In the next section, the project will take a closer look at some of these algorithms which OpenCV conveniently provides in order to see whether any of these could perhaps help in improving the accuracy of tracking blobs in BrisTracks for later refactoring. For the sake of brevity and because most of the inner workings of the algorithms are, perhaps, beyond the scope of this project, a superficial glance at how the tracking algorithms provided by OpenCV 3.0 worked against the videos provided by the Bristol Ant Lab will be discussed.

### 3.2.4 BOOSTING and MIL

As Mallick points out, the BOOSTING algorithm is “based on an online version of Adaboost the algorithm that the HAAR cascade based face detector uses internally” [31]. It is his suggestion to avoid this algorithm altogether as it is “a decade old” and does not perform well considering that there are other, better, faster, tracking algorithms in OpenCV which do a better job. When tested against a small clip of two ants tandem-running, it performed poorly, it was very slow and the tracker quickly lost the tracks when the two ants came in close proximity. Using this algorithm in a clip with 50+ ants and/or inside the nest didn’t yield the desired results either.

MIL, which stands for Multiple Instance Learning, performed considerably better with the same tandem-running video clip described earlier. The MIL algorithm is similar to the BOOSTING algorithm, the main difference being that “instead of considering only the current location of the object as a positive example, (MIL) looks in a small neighborhood around the current location to generate several potential positive examples” [31]. Now this is perhaps very useful when the tracker is trying to track, say, a person crossing the computer screen from left to right; but when trying to track multiple objects which come into contact every so often, this starts to present a big problem as the bounding box of the MIL algorithm will believe that the object which is currently being tracked has moved, see Figure 3.6. The bounding boxes of just two ants jumps back and forth as these two ants engage in their known bio-directional feedback. Another problem presented by this algorithm is scalability: when tested against a video of 10+ ants, this algorithm performed poorly, since it clogged down the computer and it was almost impossible to gather any useful information after 1 minute of tracking these ants. As the documentation shows, the MIL algorithm will try to search its neighborhood for a positive “centered” image;

this, scaled up to hundreds of bounding boxes, will eventually slow down performance, as it was demonstrated with just 10+ ants. In a nutshell, the MIL algorithm works best for when trying to track objects which have unique features and when these objects forgo periods of occlusion.

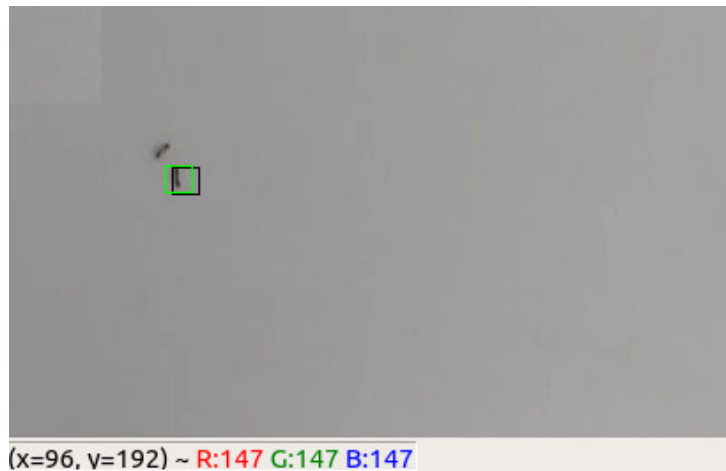


FIGURE 3.6: MIL bounding box which has wrongfully switched to a neighboring ant. Perhaps because both are of same size and shape.

### 3.2.5 MEDIANFLOW

MEDIANFLOW is an algorithm which performs excellently when the motion of the tracked object is small and predictable. The MEDIANFLOW algorithm worked like a charm with video clips of single ants scouting for new nests and it also works very well in a tandem-run when it is tracking the follower ant. Tracking the leader, however, threw up various errors; it is speculated that since the leader needs to stop every so often to wait for the follower ant, and because these stops are random and unpredictable, the MEDIANFLOW bounding box fails: especially when the follower ant touches the leader ant after a period of absence as the leader's bounding box is switched to the follower ant Figure 3.7. In most cases when observing these tandem-runs, the follower ant almost never stops: this is the reason why MEDIANFLOW works better for the follower than for the leader. This algorithm also didn't work well either with video clips of full migrations as most ants (single or tandem-runs) engage in constant collision; thus switching the bounding box and losing ids soon after. This algorithm does not scale as nicely as some of the other ones; specially when tested against a video clip of 5+ ants.

### 3.2.6 KCF

Of all of the OpenCV algorithms which were tested with the video clips provided, KCF is the one which performed the best. As the documentation explains, KCF builds on the work of previous trackers such as MIL and BOOSTING; however, the main difference

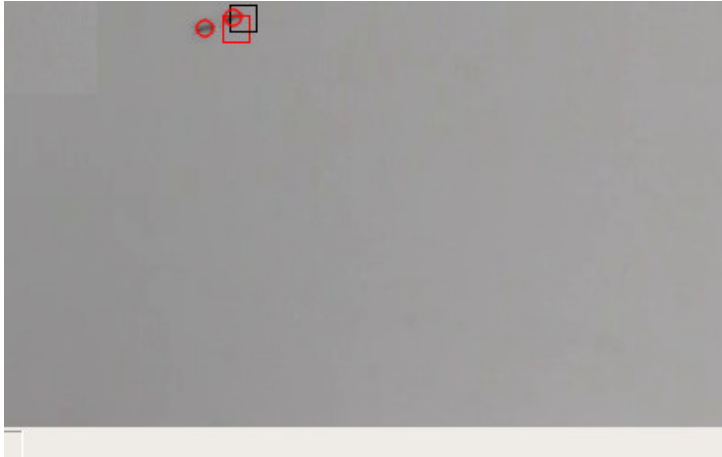


FIGURE 3.7: MEDIANFLOW bounding box left behind by the leader ant because of the stop and go nature of the tandem-run.

is that it “utilizes that fact that the multiple positive samples used in the MIL tracker have large overlapping regions. This overlapping data leads to some nice mathematical properties that is exploited by this tracker to make tracking faster and more accurate at the same time” [31]. The KCF is not only popular among the OpenCV community; it is a tracker which has seen a recent upsurge in popularity from many celebrated institutions and research communities. KCF seems to be the tracker which others try to compare against their own [32], [33] when testing whether any of their trackers can outperform KCF.

Unlike the other trackers when tested against our video clips, KCF performed remarkably quickly with videos of 30+ ants; thus suggesting a good scalability. Another good feature of this tracker is that, when performed on solo ants, the bounding box of KCF was never left behind; however, as with the other trackers, it would perform poorly when the ants touched each other, or more specifically, when they overlapped each other. Yet, out of all of the OpenCV trackers, KCF is the only one which would lost tracks the least. On clips of two tandem-running ants of 30 to 60 seconds duration, KCF would lose the track on average 2-3 times.

### 3.2.7 Inside the Nest

All of the mentioned OpenCV tracking algorithms were, therefore, tested against high quality videos of ants in close proximity which were residing inside the nest. The idea behind this experiment was to take full advantage of features which only high quality close up videos of ants could provide in order to test which of these algorithms and their respective bounding boxes would lose the tracks the least. The result for most of the algorithms was poor, as the nature of having ants touching and/or occluding one

another was the determining factoring for the loss of the bounding boxes in highly dense populations of ants.

One of the biggest problems which this experiment faced when trying to track ants inside the nest was that the bounding box was surrounding the whole ant. This presented itself as problematic as these OpenCV bounding boxes do not rotate; therefore, when ants changed position and/or rotated, the tracker would quickly lose its targeted area of tracking or would be susceptible to another “stealing” the bounding box Figure 3.8. This way of tracking high quality video of ants inside the nest would, therefore, not work if one wanted to use these OpenCV algorithms for later experiments inside the nest.



FIGURE 3.8: Bounding boxes covering the whole ant and partially lost tracks.

Tracking inside the nest was a problem which was presented to by the Bristol Ant Lab to see if there could be ways in which the software could accurately track some of these individuals in highly concentrated areas. The application for tracking them inside the nest varies: from trying to figure out how long a scout ant would spend inside the nest to, just simply, counting how many ants are inside the nest. Therefore, a lot more time was devoted to finding ways in which one could track some of these individuals in highly populated areas.

Something to contrast between these videos of nests against the videos of the ant migration is how much higher the quality of the videos of the nests are. This is because the camera is positioned a lot closer to the ants, thus rendering more and better features of the ants to be exploited. This is an advantage for the OpenCV trackers, as they need to have distinct features inside their bounding boxes. Because of this, one can take advantage of some of the distinctive features of the ant: the head, the antennae, the thorax or the gaster (the tail of the ant).

For the next experiment, one used the bounding boxes to surround and track the gaster of the ant. This proved to be effective in contrast to tracking the whole body due to the fact that the gaster is of a circular shape Figure 3.9. A circle inside a box, when rotated, will preserve its shape, and thus would almost never leave the region of interest (ROI). For



this experiment, the KCF tracking algorithm was used because it would be light and fast to track and wouldn't clog down memory as these videos tend to be big. It was a great pleasure to see that one was able to effectively track of and maintaining the ids of ants inside the nest for longer periods of time than even the videos of tandem-runs and full migrations. This allowed us to conclude that, when presented with specific shapes and/or features, and when these features are circular, and when the quality of the video is high, the KCF algorithm works best and is suitable for implementation in later experiments. Perhaps these results can influence the next set of experiments on ant nests to use KCF for their trackings.

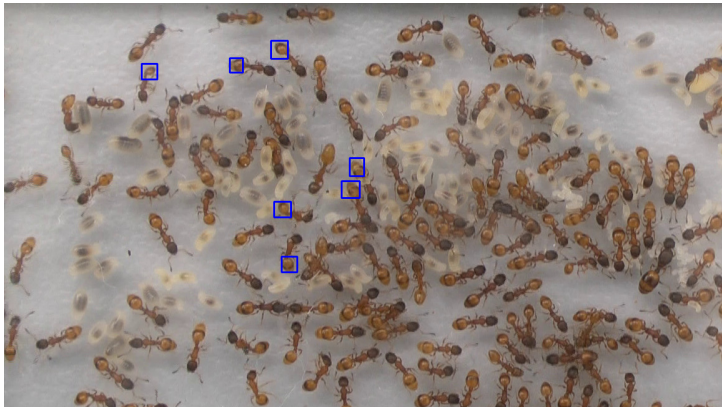


FIGURE 3.9: KCF tracker inside the nest. Notice the density of blobs to keep tracking. By tracking the gaster alone, one is able to preserve the tracker for longer periods of time. No ids are switched.

### 3.2.8 Other Trackers

OpenCV 3.0 offers two more tracking algorithms which need to be mentioned in order to demonstrate the breath of this investigation. The first one is a tracker called TLD which stands for Tracking, Learning and Detection. The TLD tracker “follows the object from frame to frame. The detector localizes all appearances that have been observed so far and corrects the tracker if necessary” [34]. This tracker was useless for our experiments since it takes too long to start learning what it is that it needs to track. Another big problem with this tracker and the migration experiment is that all blobs have the same features, thus confusing the tracker and rendering wrong results. With the provided videos, TLD was never really able to track a single ant.

The second tracking algorithm is the promising GOTURN tracking algorithm which is based on Convolutional Neural Networks (CNN). Very little can be said about this tracking algorithm since the OpenCV 3.2 version which this experiment uses has a bug which crashes the program as a whole. Perhaps, in later experiments, one could use this tracker to find out whether it is a useful tracker in the migration experiment or the nest experiment.

### 3.2.9 Kalman Filters and the KuhnMunkres Algorithm

For this investigation, one stepped away from experimenting with the OpenCV tracking algorithms to test a video tutorial which one stumbled upon whilst researching ways in which people have tackled the question of tracking bugs in pre-recorded videos. The YouTube user by the name of Student Dave, kindly provides a four part tutorial on how to detect, and track bugs using MATLAB. For this, one followed his tutorial and borrowed the majority of his MATLAB code to test whether his claims of accurately tracking bugs would offer an optimal performance with the videos of ant migration. Some very interesting conclusions were gathered from this investigation.

MATLAB is not an image processing software like OpenCV; rather, it is a high-performance software for technical computing “whose basic data element is an array that does not require dimensioning” [35]. Because MATLAB is not an image processing software, basic steps in video processing need to be carried out by secondary software. An example of this is sectioning the frames from the provided videos. In OpenCV, this step is automatic since one only needs to input the fact that one is analyzing a video as opposed to an image, and OpenCV would take care of sectioning the video into its component frames. However, MATLAB does not have a function which does this; and as a result one needed to use other methods of extracting the frames and saving them in files as individual frames for later analysis. This is already a long winded process which takes some time to do, however, for the sake of the experiment, these steps needed to be taken into consideration. To separate the video clips into component frames, one followed a tutorial on how to extract frames from videos and wrote a Python script which became very useful for this experiment. Once the frames are saved in a file as individual JPEG files, then the first step takes place: filtering.

Student Dave, in this tutorial, introduces a very interesting way of extracting blobs from images. As opposed to BrisTracks and OpenCV methods in which background subtraction is employed to highlight the differences in surrounding frames, this tutorial uses a method called “extracting the laplacian of the Gaussian of individual blobs” to highlight areas in which the pixel density is the highest and thus the area of which the user is interested in analyzing. Once one applies this method alongside the specific parameters for which we are interested in retrieving: such as the area of the blob for example, then we are presented with a Gaussian structure such as the one in Figure 3.10.

Then a laplacian is applied to the Gaussian to smooth the frame and to get rid of any noise which would otherwise give wrong output, and the result is a filtered image of just peaks and valleys which can later be tagged and tracked accordingly. An example of this is in



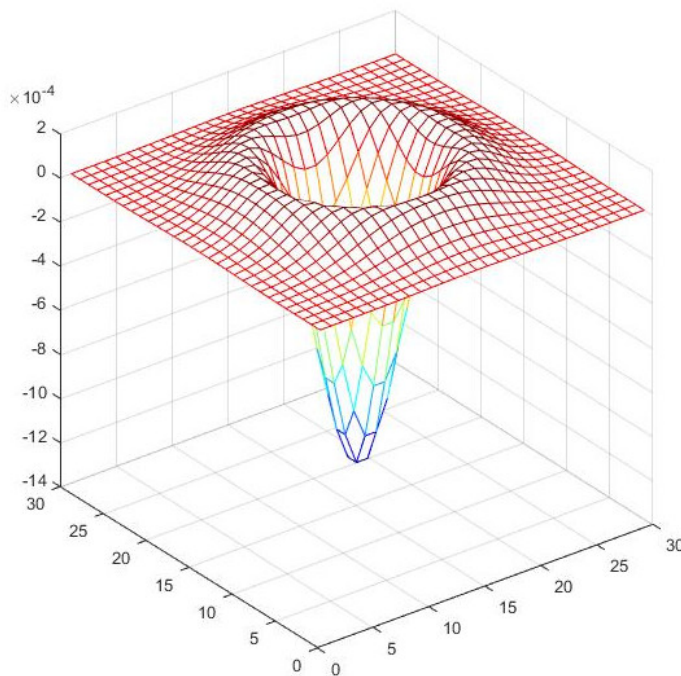


FIGURE 3.10: Gaussian of the blob in question.

Figure 3.11. This proved to be an interesting and novel method of extracting blobs from frames because, unlike background subtraction, with this method we can disambiguate between two touching ants based on their combined peaks and valleys. In Figure 3.11, for example, it is the red colour, or the valleys, which we are interested in tracking. If this was subtracted from the background, this image would look like a single long blob with no distinct features. Once we've successfully filtered out the valleys, then we apply a detection method to extract the position of each valley in an X and Y coordinate. These positions are stored in an array-like structure, for each frame, along the whole video. No ids are given since it is impossible, at this point, to know which one is which based on this method.

Once the detections are saved in the data structure, then the Kalman-Filter is applied to these points to best extract the estimates of the given blobs per frame. What the Kalman-Filter does is try to predict where the blob will be in the next frame based on the trajectory and velocity of a given blob taken from the previous detections on the previous frames. At this point we have two coordinates, one with detections and one with estimates, both of which are lacking any identification. Next, the pairwise detection and estimates points will be inserted into a Hungarian Algorithm to best match them based on a cost measure. Because of the cost measure, the detections can be best matched with the estimates and thus these points will have their proper ids. Having a Hungarian Algorithm producing some kind of cost measure allows for better tracking when, for example, some

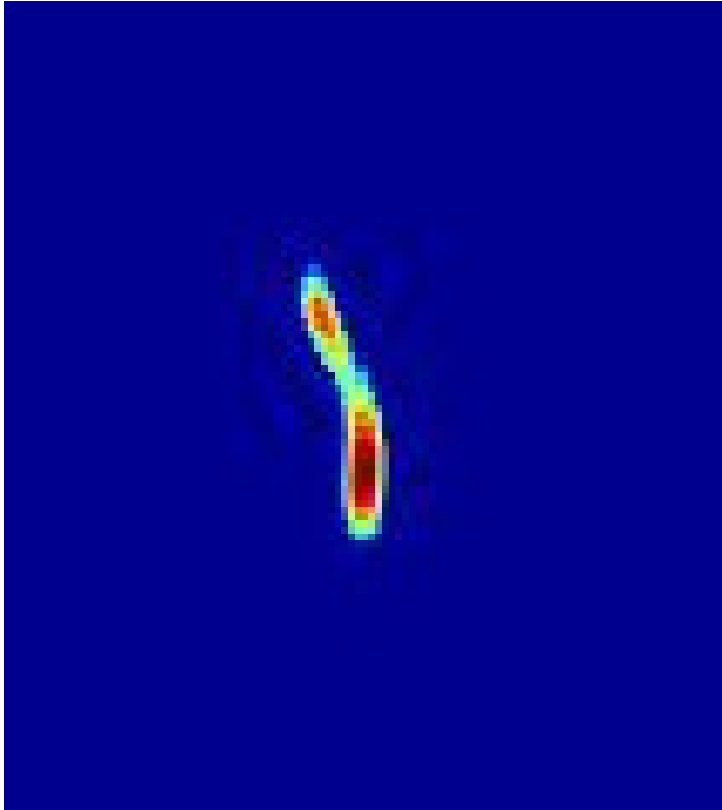


FIGURE 3.11: Laplacian applied to the Gaussian to smooth the image and extract its peaks and valleys. This is an image of two ants in close proximity. Notice the distinctive red of each ant, thus suggesting a way in which we can extract individual ants despite the fact that both are touching.

blobs undergo occlusion. It also allows us to give ids to the blobs as the frames come in from the loop sequence.

Now, this method, tested against a video of fruit flies which is kindly provided by the tutorial, works best when the motion of the blobs are smooth and steady. This does not work well in the videos of either ant migration or tandem-runs. It seems that the default setting for the prediction is set to assume that the detected and predicted blobs will have some kind smooth trajectory which will not be interrupted along the video. The ants display some very interesting behaviours with various forms of acceleration and trajectory which makes it difficult for any “out of the box” algorithm to predict and track: their movements are not smooth and they are very hard to guess.

One useful piece of information gained from this experiment is that, by using Student Dave’s approach to extract blobs, one was able to successfully detect ants’ heads and gasters from highly populated videos such as the video from the nest experiment. In Figure 3.12 one can see that the heads and the gasters of the ants are properly detected and crosshairs are set dead in the centre of such. Not much time was spent using this method of detection for the nest experiment, but these findings thus suggest that one can use this method to better detect specific features of the ant when the videos are of high quality and when the ants are in close proximity.



FIGURE 3.12: Image showing detected heads and gaster from Student Dave's approach to blob detection.

Yet another useful piece of information gained from this investigation into the use of Kalman-Filtering and Hungarian Algorithm is the use of a cost function to better assign blobs in video sequences. Because this method of tracking uses a cost function based on the predicted and detected coordinates, and because these assume some kind of information which is gained from the past frames, our subsequent experiments will use the fact that these ants display certain behaviours which could be exploited in order to create a new kind of cost function. This could allow ids to be assigned for some of these ants based on predicted movement and, perhaps, behaviour.

### 3.2.10 Conclusion of Preliminary Investigations

Any tracker and/or tracking algorithm will have its advantages and disadvantages when trying to accurately track blobs in video sequences. Some trackers will be best suited for objects with certain shapes while others will perform better with blobs that have smooth trajectories and predictable states. No tracker seems to be “best” to track out of the box accurately and comprehensively. Certain tweaks and fixes are needed to be implemented in any tracker for best results. From these preliminary investigations, we’ve concluded that each and every single approach to tracking blobs can offer an insight into implementing a new way of tracking the desired ants. BrisTracks performs expertly when disambiguating two ants in close proximity; it can keep its ids and track them accordingly. However, it performs poorly when trying to distinguish between ants tandem-running and Social-Carrying. It also cannot dynamically create or destroy tracks; thus causing it to display wrong data when ants enter or leave the arena. We’ve also discovered that the OpenCV tracking algorithm KCF works best when the videos are of high-quality and when the desired blob to be tracked displays unique features which can be extracted for tracking. An example of this is ants in the nest which display features such as the gaster or thorax: these features can be encapsulated in the bounding boxes and successfully tracked for long periods of time. We’ve also discovered novel ways of filtering frames and extracting desired blobs; Student Dave’s approach to applying the laplacian of the Gaussian allows

us to extract clear points in the frames which can be saved in a data structure for further analysis.

The following experiments will try to investigate ways in which there could be some amendments to BrisTracks (such as creating and destroying tracks based on blob detection) and using the best properties of Student Dave's approach to tracking to see if a new way of tracking *Temnothorax albipennis* based on their behaviours and predicted movement is possible.

## 4. *Tools for Gathering and Analyzing Data*

This chapter will detail the many ways in which this project borrowed some of the best practices from the preliminary investigations in order to build and implement ways in which one could gather more accurate data, analyze this data and perhaps answer questions regarding ant behaviours. All of the implementations from this chapter are separate from BrisTracks owing to the fact that these are prototypes of experiments that will, hopefully, make it into BrisTracks in the near future.

### 4.1 How to Tell if It is a Tandem-Run?

One of the overall questions which the Bristol Ant Lab would like to answer from the videos of ant migrations is: how many tandem-runs happen at a given time during the experiment? Another is: has a tandem-run started in the middle of the arena? Yet another is: how many reverse tandem-runs take place during an ant migration?

BrisTracks tries to answer these questions by analyzing each frame and making sure that blobs which are interacting with another have a certain area size. Although this works well for preserving identities, it fails when trying to answer whether a tandem-run behaviour is taking place. This project's proposed idea for analyzing tracks and answering behavioural questions will look at the past information retrieved from the tracks to then analyze these tracks in order to better disambiguate a tandem-run from a Normal Ant and from a social-carry.

#### 4.1.1 Using Past Frame Information and Tracks

tandem-running is characterized by having a follower ant bumping into the rear of the leader ant every so often to let the leader ant know that the follower ant is still engaged in the behaviour. The leader ant, as it was shown previously, waits (or stops moving), for the follower; meanwhile the follower ant needs to scan the area before it lets the leader ant know it is still following. This behaviour is very marked among *Temnothorax albipennis* while it migrates from an old nest to a new nest. Because this behaviour is so marked among ants engaged in migrating, and because so far we are only concerned with tracking

*Temnothorax albipennis* while it migrates from nest to nest, one of the proposed objectives was to take a look at the tracks as a whole to analyze the ants' paths on from the tracks provided by the tracker. Assuming that the tracks are accurate and that they come from two ants (one leader and one follower), then one will be able to extract this information in order to find clues as to whether these tracks and displaying a behaviour such as a tandem-run.

The hypothesis for this experiment is that, given a video (or tracks) of two ants engaged in a tandem-run, and given the fact that follower ant will always be following (or pointing towards) the leader ant, one will be able to use the tracks of these two ants to create vectors and angles which will provide this experiment with the necessary data to determine which one is a leader ant and which is a follower ant. These vectors will help to produce an angle which will be extracted every Nth frame because the vectors change in time as they move through the file of tracks. After that, these angles will be stored in an array to be later analyzed via a histogram. The result should be of a follower ant, when pointing towards the leader ant, to have an angle of direction that is always closer to zero; the leader ant shouldn't have this as it is constantly moving away from the follower and thus not pointing at the follower ant.

For this first experiment of the tandem-run behaviour, the tracks came from a video clip of two ants engaged in a tandem-run which lasts for approximately 10 seconds. BrisTracks was used to extract the tracks from the video clip and it was ascertained that no tracks were either lost or swapped. Two tracks, with X and Y coordinates for every frame, are written out into a file and marked with their proper ids: see Figure 4.1. The numbers of each frame are also written out into the file for referencing purposes.

Once the coordinates are safely stored in a file, there needs to be way to extract this data to be stored in arrays inside the Python program's memory. To achieve this, it was mentioned earlier that this project was going to explore Python's external libraries such as NumPy. NumPy makes handling, cleaning and wrangling data very straight forward. For example, a data structure in NumPy would look something like: `NumPyArray = np.zeros(shape=(sizeOfData,2))`. This NumPy function initializes an array of the sizeOfData and initializes everything inside the array to 0's; the 2 in the function helps to tell the program that the array is an array of points for X and Y (a 3 would imply a 3-D point for X,Y,Z, and so on).

Since these points are stored in memory, it is necessary to create vectors out of these points. Notice in Figure 4.2 that the end point for every vector finishes at 15 frames. It was decided at the beginning of this experiment to use this arbitrary number of 15 frames due to the fact that any number lower than 15 would result in a small vector change

1	FRAME	ID	X	Y (0 is Leader, 1 is Follower)
2	1	0	638	666
3	1	1	627	677
4	2	0	638	666
5	2	1	627	677
6	3	0	638	666
7	3	1	627	676
8	4	0	638	666
9	4	1	627	677
10	5	0	639	666
11	5	1	627	676
12	6	0	638	666
13	6	1	627	677
14	7	0	640	666
15	7	1	627	676
16	8	0	641	666
17	8	1	627	676
18	9	0	641	666
19	9	1	627	676
20	10	0	641	665
21	10	1	627	675
22	11	0	642	665
23	11	1	628	674
24	12	0	644	664
25	12	1	629	674
26	13	0	645	665
27	13	1	628	673
28	14	0	644	663
29	14	1	629	672
30	15	0	645	663
31	15	1	629	671
32	16	0	644	663
33	16	1	629	669
34	17	0	645	663
35	17	1	630	668

FIGURE 4.1: A file containing two ants and their respective position for every frame.

from previous frames, thus making it more difficult to extract an angle from these vectors with enough variance. Extracting vectors from every  $N+20$  frames was also conducted; however, for a data sample of this size, this was not possible because not enough angles were produced.

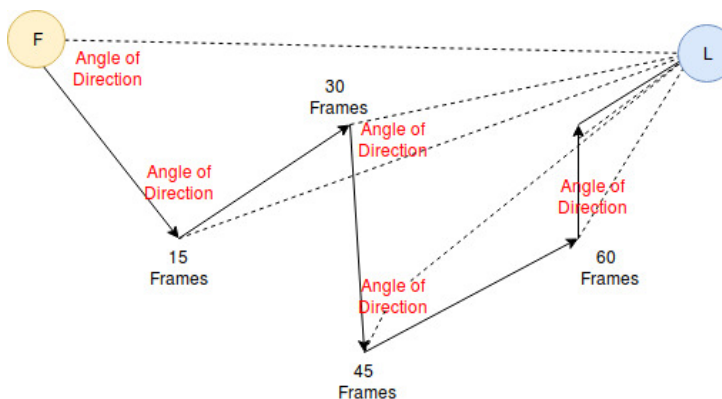


FIGURE 4.2: This graph depicts the path of a follower ant (F) as it searches its way towards a leader ant (L) in a span of 75 frames. The red caption “Angle of Direction” is a number taken from this angle in relation to where the leader is in order to analyze its behaviour.

In Figure 4.3 one can see how the inner angle of these vectors can be extracted. Let  $P1X, P1Y$  be the beginning of the vector and  $P2X, P2Y$  be end.  $P3X, P3Y$  is the point the ant should be pointing towards (in this case, follower pointing to leader and leader pointing to follower). The function returns an angle from -90 to +90 degrees, which will

be stored in an array and the program proceeds to do the same for the other ant: see Figure 4.3.

```

14
15 def ang(P1X,P1Y,P2X,P2Y,P3X,P3Y):
16     numerator = P2Y*(P1X-P3X) + P1Y*(P3X-P2X) + P3Y*(P2X-P1X)
17     denominator = (P2X-P1X)*(P1X-P3X) + (P2Y-P1Y)*(P1Y-P3Y)
18     print numerator
19     print denominator
20     if denominator != 0:
21         ratio = numerator/denominator
22         angleRad = math.atan(ratio);
23         angleDeg = (angleRad*180)/3.1416;|
24         return angleDeg
25     print angleDeg
26     if angleDeg<0:
27         angleDeg = 180+angleDeg
28     return angleDeg
29

```

FIGURE 4.3: A python function which determines the angle of direction of the ant. The basic idea for this function was taken from [36]

By the end of this program, there are two arrays of angles: from ant with id = 0 and ant with id = 1. For every array of angles, the program plots these numbers in a histogram and the results are as shown in Figures 4.4 and 4.5:

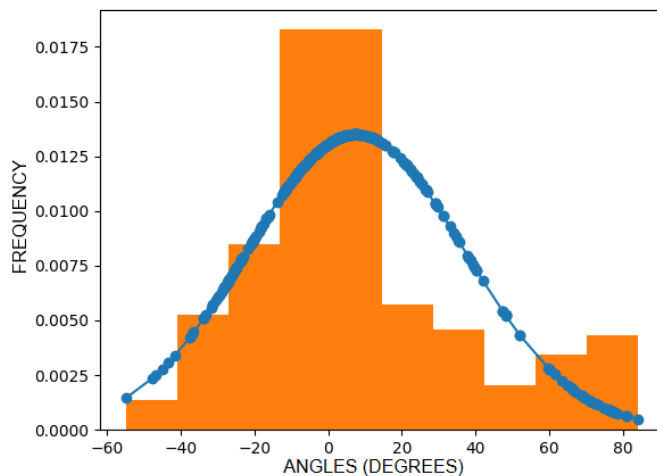


FIGURE 4.4: A histogram depicting the angle of direction from a follower ant. Notice the majority of the bins are around the zero mark; suggesting that it is pointing towards the leader ant.

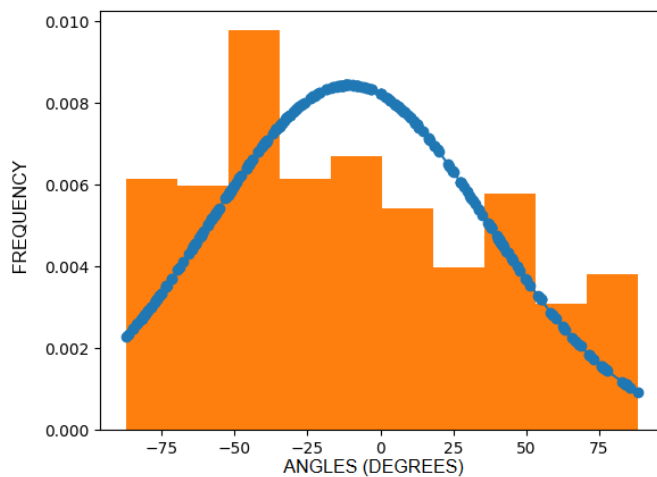


FIGURE 4.5: A histogram depicting a leader ant. Notices the bins are more scattered; suggesting that it is not pointing towards the follower ant but, rather, it is moving away.



### 4.1.2 Conclusion of the Tandem-Run Experiment

For this experiment, the coordinates of two ants were used to create vectors from which one was able to extract angles that are based on how these two ants interacted with one another. It was decided to use points from every 15 frames to allow the ants to display a long enough change in direction so that the vectors would be, perhaps, longer and better suited to display an angle. It was debated whether, by using angles of direction from two ants, one could determine whether one ant would have angles which suggested a direction towards another ant. These angles were plotted in a histogram to produce some form of data visualization in order to determine which histogram represents a leader ant and which one represents a follower ant. Figures 4.4 and 4.5 confirm our hypothesis. As a result, one safely concludes that one is able to extract tracks from files and run them against other tracks to tell whether that particular ant is engaged in a tandem-run.

This experiment, however, is of a single video clip of 10 seconds: the sampling data is not big enough. Therefore, in the next section, this project will show how one created a semi-automatic tracker which helps to gather more data to keep testing this hypothesis.

## 4.2 Semi-Automated Multi-Tracking

There needs to be a way to gather more data (tracks) of two ants while they are tandem-running to have a bigger data set for testing the behavioural hypothesis. For the previous experiment, one used an edited video clip of a tandem-run which displayed two ants for a duration of 10 seconds and to track their movements we used BrisTracks. For BrisTracks to track blobs, one needs to specify the number of tracks one wants to track: this poses a problem as BrisTracks automatically assigns tracks instead of letting the user click or choose which blob to track. It would have been possible to edit an N number of videos of two ants tandem-running; however, this would have required more time editing videos instead of learning how to properly use Computer Vision techniques, software developing practices and OpenCV skills. For that, this project opted to create a semi-automated tracking software for the sole purpose of gathering data to use in the experiments.

While conducting the preliminary investigations to test different OpenCV algorithms to track objects, one used the tutorials and code that is provided with the OpenCV download. OpenCV helps the user to hit the ground running by providing well documented code which helps to test their different algorithms. However, as one may very well assume, OpenCV provides the bare minimum to do such task as they assume that one is a confident and versatile computer programmer. Therefore, for the Semi-Automated Multi-Tracking

software to work, one needed to refactor a lot more of the provided code. OpenCV 3.0 comes with a class called Multi-Object Tracking which, as the name suggest, helps to create multiple instances of bounding boxes to be tracked in a single video. However, this class only works for C++ as there seems to be a bug for Python. This section will take a closer look at how one this bug was overcome and how a tool was created to extract tracks from the Bristol Ant Lab videos.

#### 4.2.1 Multi-Object Tracking for Python

When initializing the tracker in C++, one has the option to create a vector `<Rect2D>` object which allows multiple rectangles to be stored in a structure which is later passed into the initialization. Python, on the other hand, does not allow an object full of rectangles to pass into its initialization; instead one has to create a rectangle and pass it into the multi-tracker class: Figure A.3. This presents a problem because it assumes that the user needs to know how many rectangles will be created before it is even initialized.

To work around this problem, a while loop was created which “listens” to the users clicks. As the user clicks on more and more blobs, the click’s coordinates are used to create the bounding boxes. Since OpenCV creates bounding boxes from the top most left point of a rectangle, the click’s X coordinate is subtracted by half the length of the bounding box and the same is done for the Y coordinate. This ensures that the blob’s centre point stays in the middle and the bounding box is around the blob. After the bounding box is created, then it its passed into a Python list and stored. Once the user decides that it is time to begin the tracker, another loop starts which adds the boxes one by one into the OpenCV Python tracker program. This is a very different from the C++ way of adding all of the rectangles at once: Figure A.4. However, this way is a nice workaround to this problem which many users in the OpenCV forum have complained about.

Once the bounding boxes have been initialized and the tracker is successfully tracking the blobs and the video is moving along, then the next step is to have a way to reset the tracker for whenever these bounding boxes either return an error or when these boxes are swapped or when tracks are lost. As it was mentioned earlier, KCF performs quite well when tracking these ants in close proximity; however, over longer periods of time, some boxes start to drift away from their target and thus become susceptible to swapping to another ant. Unfortunately, any OpenCV tracker that comes “out of the box” does not allow for the user to manually update their location (another aspect the OpenCV than has been criticized). Instead, one needs to completely destroy the instance of the tracker and then create a new tracker with a fresh set of bounding boxes. This is also problematic

because, say the tracker has 10 boxes but only one has gone off, the user has to not only remember the id of every box and its location, but the user also needs to manually update all of them accordingly. At this point it would make a lot more sense to just edit videos and run them through other trackers.

Instead, to avoid the complexities of remembering ids and locations, a work around was to store the ids of the boxes, their sizes and their coordinates inside a Python dictionary which, in turn, would contain another dictionary for each and every frame. An example of this would look something like: `tracks = {frameNumber{(antId, size, location), ...} ...}`. This works magnificently because, by having these dictionaries in memory, one can now not only reset the tracker and create new boxes based on the current frame's coordinates, but one can also reset one bounding box based on its id and dump the rest in the tracker to avoid having to manually reset all of the boxes. With this method, one can also “rewind” the video to the frame where the tracker threw the error; thus allowing the user to manually correct the bounding boxes. By doing this, the user ensures that the tracker and its bounding boxes are effectively tracking everything with a degree of degree accuracy.

#### 4.2.2 Conclusion to the Multi-Object Tracker in Python

The end result is a tool which allows this project to: 1) click on multiple blobs in the video to initialize the bounding boxes around the blobs, 2) reset the tracker with the option to correct any number of bounding boxes, 3) rewind the video to the frame where the error has occurred, 4) create or destroy bounding boxes, 5) write these tracks to file with their respective frame numbers, ids and coordinates.

More time needs to be devoted to the creation of a user-friendly Graphical User Interface (GUI); however, for now and for the scope of this project, this tool is a great help for extracting meaningful and accurate data on the ants while they perform tandem-running, social-carry or anything else that one might be interested in tracking. With this data, this project can move on to test the previous hypothesis of successfully classifying a leader from a follower in a tandem-run.

### 4.3 Automating the Tracker, Detecting Errors and Behavioural Tracklets

One of the aims of this project is to introduce ways to help automate part of the research carried out by the Bristol Ant Lab in the migration of *Temnothorax albipennis* through

the recording and tracking of the movements of ants in videos. So far, this project has introduced several algorithms which have been tested with these videos, a semi-automated way of tracking these ants (though for a full migration this might prove to be too time consuming) and a function which can tell a leader from a follower in a tandem-run file of tracks.

This section is concerned with testing ways in which part of these last mentioned implementations can begin to be automated; that is, how can a tracker detect errors and how to tackle bigger problems such as lost tracks or swapped ids. Some ideas that will be presented here will include functions which will take into consideration the state of the ant (say, a follower ant in a tandem-run) to determine if it should keep that track after it was broken.

### 4.3.1 Detecting Errors

In the previous section a semi-automated piece of software was introduced which can use a function that rewinds to the frame where a bounding box has gone awry in order to adjust that bounding box. To successfully amend this bounding box, the user needed to stop the video, type in the frame to which it intended to go, choose the id of the ant from which it was going to amend the box and continue the tracker. This method of tracking is good enough if one is tracking a small number of ants, perhaps if one wants to gather accurate data for analysis of certain behaviours in a small sample of ants. However, this could easily become tedious as things will start to scale if one introduces hundreds of ants with hundreds of bounding boxes, all of which will start to miss their target at some point during the tracking experiment. This section will begin to look at a way in which one could use the blob detection class from OpenCV to aid the recognition of errors in tracking; the idea here being that the program should continue tracking despite the fact that the tracker has gone through a reset after detecting a problem. This already presents a different approach to BrisTracks: BrisTracks would continue to track even if these tracks were lost or if ants exited the video frame.

For the implementation of this feature, the use of the OpenCV function for blob detection to find the centre of each blob will be needed; then, depending on the size of the ants in question, the user specifies the size of the bounding boxes for the blobs. Once each blob has been detected by the `simpleBlobDetection()` OpenCV function, then exactly the same process from the semi-automated tracker takes place; that is, each point is taken to help create bounding boxes around it. Once the bounding boxes are set in place, then the tracker can begin as usual. The biggest difference from the semi-automated tracker is

that instead of just running the tracker on its own, this piece of software also runs a blob detection for every frame.

Since this tracker is running along with the blob detection function from OpenCV, in combination they can be used to help detect when things start to go wrong. How do we know when things are starting to go wrong with our tracker? After watching hours of videos for when the bounding box of each ant was starting to show errors, one started to notice that there was a common trend with these videos: most of the time, the bounding boxes would either drift away from the ant or another ant would pick up another ant's box. As a result of these errors, one ant could potentially end up with two boxes while another had none: see Figure 4.6.

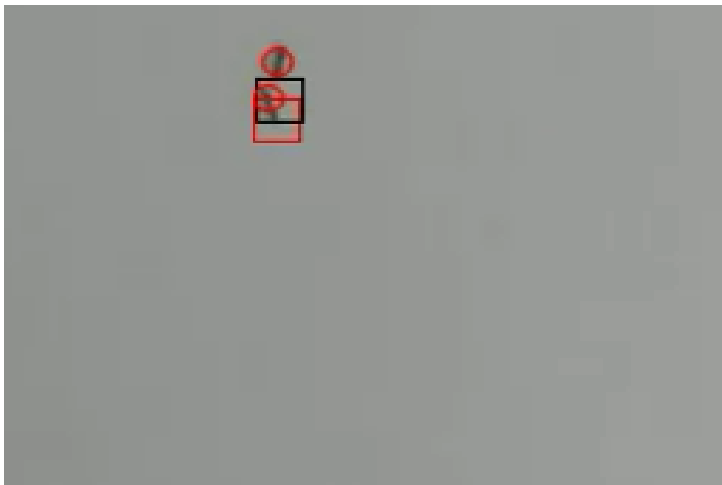


FIGURE 4.6: A blob has been detected (red circle) to be outside of the box (black and red boxes to help distinguish different ids); therefore the tracker is about to reset.

Once it was understood that these were the most common errors with the KCF trackers, then the project focused on trying to find a solution for how one could use the blob detection to detect and amend these errors. Therefore, for each frame, the position of the blob is known because of the blob detection as well as the position of the bounding box. If it is known that sometimes bounding boxes drift away from blobs, then an “if” statement is implemented in order to check whether each bounding box contains a blob inside of it. To do that, the following Stack Overflow tutorial is was studied to implement this feature: “Assuming the rectangle is represented by three points A,B,C, with AB and BC perpendicular, you only need to check the projections of the query point M on AB and BC:

$$0 \leq \text{dot}(AB, AM) \leq \text{dot}(AB, AB) \text{ \& \& } 0 \leq \text{dot}(BC, BM) \leq \text{dot}(BC, BC) \quad (4.1)$$

AB is vector AB, with coordinates (Bx-Ax,By-Ay), and  $\text{dot}(U,V)$  is the dot product of vectors U and V:  $U_x \cdot V_x + U_y \cdot V_y$ ” [37][38] and the Python implementation of this equation in Figure A.5.

That is to say: for every blob detection and returned centre point of said blob, the program makes sure that this point sits inside one of the bounding boxes; if it is not, then the box has drifted from that blob and a reset of the tracker takes place. Therefore, this helps to keep the automated tracker running smoothly and it also helps this project to have some degree of confidence that things will reset if a bounding box is not properly tracking.

Despite the fact that one can now run the tracker and have the bounding boxes reset when things go wrong, one of the problems that arises is that, sometimes, the ids of these ants are switched upon reset. The end result of this automated detection and tracker is a file with not tracks but “tracklets”. A tracklet, for the sake of simplicity, is a file with tracks that are separated when the reset takes place; the ids are switched and therefore each tracklet is unique and shouldn’t be considered part of the other until joined: see Figure 4.7.

## 4.4 Tracklets

Upon running this new function with several video clips of tandem runs, the result was that, indeed, the tracking would reset if it detected a blob outside the bounding box. However, a problem kept persisting: ids were being swapped when the tracker restarted. To address this issue, there needs to be a way to re-join these tracklets afterwards.

```

376 188 0 653 579
377 188 1 650 591
378 189 0 652 578
379 189 1 651 590
380 190 0 653 578
381 190 1 651 590
382 191 0 653 577
383 191 1 651 589
384 192 0 653 575
385 192 1 651 588
386 193 0 653 575
387 193 1 651 588
388 194 0 653 574
389 194 1 651 588
390 195 0 653 574
391 195 1 651 587
392 196 0 653 573
393 196 1 651 585
394 197 0 653 572
395 197 1 651 584
396 198 0 653 573
397 198 1 651 584
398 199 0 653 572
399 199 1 650 583
400 200 0 653 572
401 200 1 651 582
402 RESET
403 201 0 653 571
404 201 1 650 583
405 202 0 653 570
406 202 1 651 582
407 203 0 653 569
408 203 1 650 582
409 204 0 653 568

```

FIGURE 4.7: A file which contains the id of the ants and their respective tracks. Notice the introduction of the restart in between the tracks; thus demonstrating where the re-joining needs to take place.

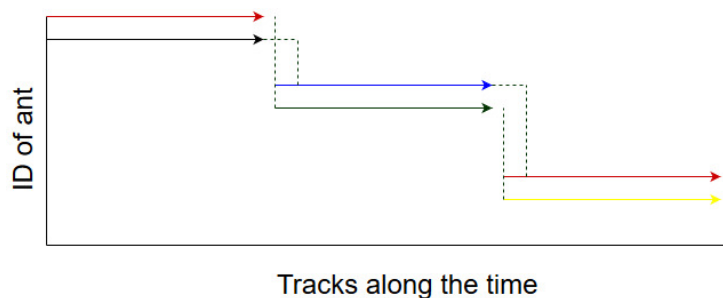


FIGURE 4.8: Figure showing tracklets being broken up by the restart function and possible ways joining them back together.

### 4.4.1 Predicting movement

One way to solve the problem of re-assigning the tracklets with their proper ids is to use a Euclidean (see the equation 4.2) distance function between two points: the point from before the reset and the point from after the reset. For example, in Figure 4.7 one would take the point in line 400 (653 572) and would try to find the minimal distance between the points in lines 403 and 404. Whichever point after the RESET yields the minimum distance, that should be the one which is going to be joined with the ids of line 400. Why, one may ask, should they be joined based on the minimum distance? The answer is that, from frame to frame ants should not move too much in distance; therefore anything that is closer to the point before the reset should be that same one blob.

$$d(p, q) = \sqrt{(q1 - p1)^2 + (q2 - p2)^2} \quad (4.2)$$

Python’s library SciPy, which is a “Python-based ecosystem of open-source software for mathematics, science, and engineering” [39] helps with the above equation by providing a function called `dst = distance.euclidean(point1, point2)`. Hence, the tandem-run python class will loop over a track file, it would find where the “RESET” keyword is and it would take the first id of the frame before the reset and use the ids of the frame after the reset. These distances are stored in a dictionary with the ids from which it was extracted and after all of the distances have been computed; then a `min()` function is deployed in order to pick the smallest number. This works well at finding a possible way to re-join the tracklets.

Yet, from the subsequent experiments on trying to re-join these tracklets, there were times when re-joining the tracklets based on their Euclidean distance alone proved to be inaccurate. This is because, once the detection and restart of the tracker had taken place, the ant in question would have moved in such a way that it was further away than the other ant. For example, one ant would have moved more than the other; the Euclidean distance function would try to join that particular point with the wrongly detected ant blob. Thus re-joining the tracks wrongly.

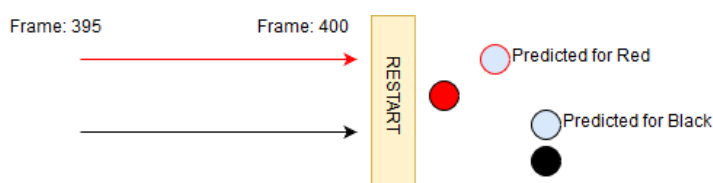


FIGURE 4.9: A chart depicting two ants moving in 5 frames, then a reset, then a prediction as to where they will be next and a detection. The detection is based on their respective colours for simplicity

To solve this new issue, another proposed method of re-joining these tracklets was introduced. Instead of basing the procedure on current frames; one could create a predictive point and then try to find the Euclidean distance between that predicted point and the detected point. For example, in Figure 4.9 there are two tracks that undergo a reset. If the previous method of rejoining the ants was deployed under this situation, then the black arrow would have been rejoined with the red detected circle because it would find it closer to that one as opposed to the black circle. The red arrow would have also joined with the red, thus creating a bigger problem. Instead, with the predictive method, detected points are used to obtain a distance. To create predicted points, one takes the frame where the reset happens and subtracts the points from this frame to the points of an arbitrary frame: in this case and for the subsequent experiments, 5<sup>1</sup> frames were chosen as an optimal number to go back to. Once the difference between these two frames is obtained, then this difference is added to the current frame from which the reset happened: the result is a new point which is the predicted point based on the movement of the ant. In Figure 4.9 one can see that the predicted point for the black arrow is now closer to the black detected circle than it is to the red detected circle. By using this method, one was able to successfully re-join the tracklets of the tracks that underwent reset; however more data and evaluations were necessary to build this project's last implementation.

#### 4.4.2 Cost Function

Whereas the previous implementation would join the tracks based on the predicted points and smallest Euclidean distance, this next and last implementation will take this idea further and use the predicted Euclidean distance to create a function which will return the sequence in which the tracks should be joined. The inspiration to create a cost function based on various parameters comes from observing how Student Dave's tutorial used the Hungarian Algorithm to best determine the assignments of the ids. In that tutorial, he uses the Kalman-Filter for prediction and the Hungarian Algorithm for assignment. This proposed function will use some prediction as well; however, since this last implementation is at the stage of ongoing work, the idea is that not only prediction will determine the cost, but behaviour as well and, perhaps later, velocity, average distance covered and so on.

The way to create this cost function is to first take the id of the first set of tracklets (say ids: 0 and 1) and find all possible combinations of joining 0 and 1 with the subsequent set of tracklets. For example, if  $R$  is the number of resets and  $T$  is the number of tracklets;

---

<sup>1</sup>Future experiments could perhaps go back even further to see if results are better or if the numbers change at all.

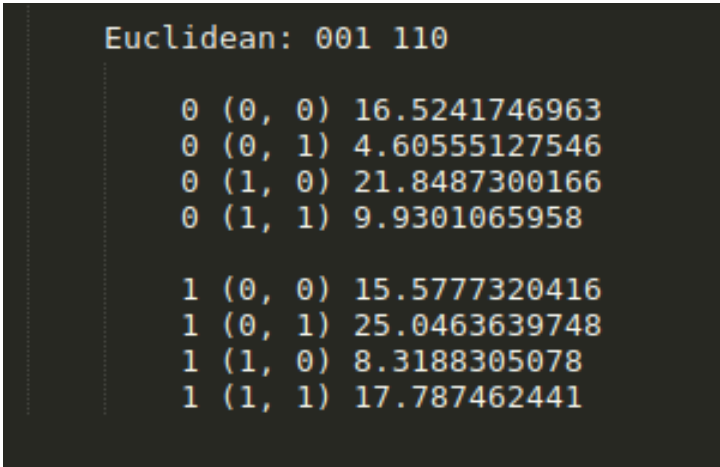


then:

$$N = T^R \quad (4.3)$$

$N$  would be the number of possible combinations for a particular file of tracklets. Thus, if a file contains 2 tracklets and 3 resets, then all possible combinations are 8; where 4 combinations would be for the first tracklet with  $id = 0$  and 4 for the other first tracklet with  $id = 1$ .

In the example given above, at every reset, the tracklet with  $id = 0$  will produce 2 possibilities with two distances, say:  $00 = 3.5677$  and  $01 = 22.90098$ . As one can see, tracklet 0 should be joined with tracklet 0 of because the distance is smaller than tracklet 1. This process continues for all other tracklets after the reset. At the very same time, these distances are being added to one another until something like in Figure 4.10 is returned. As one can see from Figure 4.10, there are 8 possible combinations (4 for  $id = 0$  and 4 for  $ids = 1$ ). For every initial  $id$ , the smallest number is picked as this is the smallest cost possible to join the tracklets based on their predicted distance. The result, in Figure 4.10, is that the tracklets should be joined in a sequence of 001 110. This is tested against the ground truth (which is displayed in Figure: 4.9) to make sure that things are working correctly.



Euclidean: 001 110		
0	(0, 0)	16.5241746963
0	(0, 1)	4.60555127546
0	(1, 0)	21.8487300166
0	(1, 1)	9.9301065958
1	(0, 0)	15.5777320416
1	(0, 1)	25.0463639748
1	(1, 0)	8.3188305078
1	(1, 1)	17.787462441

FIGURE 4.10: A screen-shot of returned sequences of a file of tracklets and the suggested approach to re-joining the tracklets

This idea and the evaluation will be presented in the next chapter with more data and a percentage of accuracy for  $N$  number of tandem-run videos. For now, the main point is that one can take the distance of each reset and tracklet and add them together to create a cost; if the cost is the minimum, one should be confident that that is the way tracklets should be joined.

## 5. *Results*

This chapter will detail the results of the implementations from the above mentioned programs. One important thing to point out is that the majority of the implementations described above did not happen before any evaluation, instead these implementations were created, they were tested and new ideas developed to make better amendments and/or to create new implementations. Therefore, last chapter tried to best describe the thought and work processes of how things were constructed one after the other. This chapter will be concerned with the data size and thoughts from people who observed the implementations in action.

### 5.1 Tandem-Run Class

For this experiment, the Semi-Automated tracking software was used to track 20 tandem-runs from one of 4 videos of a full migration which was provided by the Bristol Ant Lab in order to test this project's hypothesis. To do this, firstly the videos were watched for periods of five minutes until tandem-runs were identified; once the tandem-run was identified, the frame was written down on a piece of paper and the video was backtracked to the that specific frame. After the video was backtracked, the ants which were identified as tandem-running were clicked in order to begin the tracking process. To keep things consistent, it was made sure that the ant with  $id = 0$  was always going to be the leader and the ant with  $id = 1$  was going to be the follower. This was deliberate to make sure that everything was consistent in order to obtain ground truth to compare against our tandem-run Python class. The tandem-runs were tracked for periods of an average of 1.5 minutes as it was thought that this was plenty of time to obtain enough frames with their respective coordinates. If the tracker was beginning to lose any of the bounding boxes, the video was stopped and the bounding boxes were fixed (following the process as explained earlier) until everything was as smooth and accurate as possible. After it was decided that the tracking had been going for long enough, the tracker software was stopped and the tracks were written to file with unique names for each of the files.

Once the files of tracks were saved, the next stage was to run these files with the tandem-run Python class. These files would return two outputs: 1) histograms for every ant 2) the standard deviation of all the angles for every ant. Each returned standard deviation would also have the id of the ant in question; this was to help identify which would be the leader and which the follower, and it would also allow comparison with the ground truth.

Exactly the same process was carried out with three ant tracks: 2 ants engaged in tandem running and a third ant which was picked at random. The reason for this experiment was too see how close the standard deviation for the angle of direction of the third ant was to the follower. This experiment with three ants would help and guide this project to start thinking about scaling the process into future extensions of the project.

### 5.1.1 Results for 2 ants files

Of the 20 tandem runs with two ants, 20 out of 20 times was the tandem-run Python was class able to differentiate between a leader from a follower ant. Thus resulting in 100% percent accuracy of successfully distinguishing individuals in files of tracks.

### 5.1.2 Results for 3 ants file

Out of the 6 tandem runs with three ants (one, of course, was not engaged in tandem-running), 6 out of 6 times the tandem-run Python class was able to differentiate a leader from a follower ant. The third ant was taken into account, but since the the returned histograms showed that the follower was clearly classified, the next step was to make sure that the follower's standard deviation was the one which was in relation to the leader. All of this was done manually to make sure that the procedure was consistent and accurate.

### 5.1.3 Observations of the Experiment

For the experiment with files of 2 ants; an extra line of code was written to pick the smallest standard deviation and classify it as a follower ant (the opposite for the leader). This of course would not work for the experiment with three ants as sometimes the standard deviation of the third ant would be between the smallest and the largest number; therefore, for the experiment with three ants, extra care was taken to ensure that the returned results corresponded to each individual ant. There is plenty of room here for improvement and to begin scaling this design for bigger data sets.

### 5.1.4 Conclusion to the Tandem-Run Experiment

Since the tandem-run function works by having ants displaying their behaviour “in relation to the other”; one quickly realizes that any experiment with more than three ants would start to scale rapidly. For example, in the experiment with three ants, 6 standard deviations were returned: follower to leader, follower to third, leader to follower, leader

to third, third to follower, third to leader. The main point of this experiment was to test if the smallest standard deviation corresponded with follower to leader. The result was 100% accurate; thus suggesting that this is a great step forward in classifying the behaviour of tandem-run in these lab experiments by using past-frame information acquired from files of tracks. One way one can avoid this function from “exploding” in size is to avoid running the function against all ants. That is, it was thought that, perhaps, one can use distance as a filter to avoid obtaining standard deviations from all other ants. Perhaps can set an area of interest which is based on a distance: any ant that is within that distance will be compared “in relation to the other” to obtain the desired standard deviation.

## 5.2 Rejoining Tracks

This experiment takes four stages to evaluation and results: 1) using the automated restart function to detect and restart when bounding boxes go awry; therefore producing a file of tracklets, 2) joining the tracklets with the Euclidean distance before it was refactored to contain the predicted cost function, 3) joining these tracklets with the Euclidean Python class based on their lowest cost, 4) checking to see how accurate the tandem-run Python class works for the tracklets instead of the whole tracks: this is to find out if it is suitable to use the tandem-run function with the tracklets.

### 5.2.1 Results for Automatic Restart

Of 22 tandem-run tracks, the automatic restart was 48% accurate in labelling the correct id with their respective tracklets. This is no surprise as the restart will always begin a new tracker and the ids are lost upon deletion. New ids are created once it is restarted. The automatic restart function’s main purpose for this experiment is so that one can obtain tracklets for the purpose of re-joining them with the Python classes created earlier. The restart function’s other main purpose is so that this project can introduce a novel way of tracking; that is, a form of error detection and restart manoeuvre. The numbers presented here are to show that there will be an increase in accuracy via the next implementations.

### 5.2.2 Results for Euclidean Join (No Prediction)

Of the 22 tandem-run tracks, this function demonstrated a 52% accuracy in rejoining the tracklets. As was discussed earlier, using single frame resection alone would not be

sufficient to help with re-joining the tracklets. Once again, the number is not surprising since the main purpose of this experiment was to test the prediction and cost function. However, this number will be relevant for the next result.

### 5.2.3 Results for Euclidean Join (With Prediction and Cost Function)

Of the 22 tandem-run tracks, this function demonstrated a 81% accuracy in helping re-join the tracklets. There is a 30% increase in accuracy from the restart function and the Euclidean distance function. The predicted function, which uses past-frame information as it was stated in our main objectives, proves to be useful in helping to re-join lost tracklets.

### 5.2.4 Observations of Rejoining Tracks

Three methods of re-joining tracklets were used. Figure 5.1 shows that, using past-frame information to predict where the point will be after the tracking has undergone a reset, is better than using current frame information.

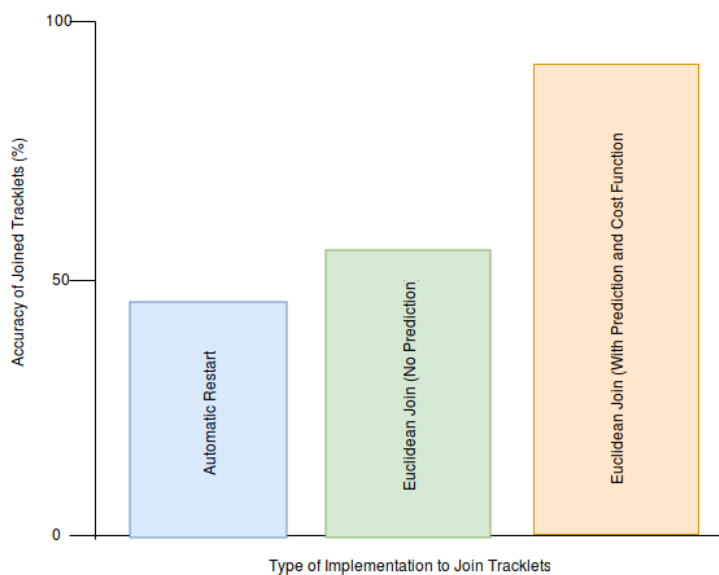


FIGURE 5.1: Chart with three different approaches to re-joining tracklets.

## 5.3 Rejoin tracklets with Tandem-Run

In this section, the use the behaviour of the tandem-run to test if by using behaviour alone one can achieve more accuracy than by using prediction alone. For this the same 22 files of tracks which were generated from the previous experiment were used. However,

as opposed to the section “Tandem-Run Class” which uses files of unbroken tracks, these 22 files have tracklets (or broken tracks) which yield a total of 48 tracklets combined.

### 5.3.1 Results

Of these 48 tracklets, 35 were confirmed accurate; thus resulting in a 73% chance of accuracy by trying to rejoin these tracklets with the tandem-run function alone.

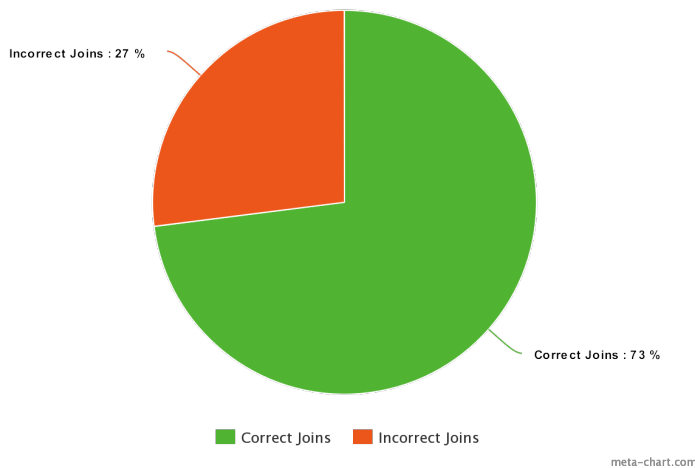


FIGURE 5.2: Pie chart depicting the accuracy of using the tandem-run function to join the tracklets. Sample is out of 48 tracklets from the the 22 files of tandem-run tracklets.

## 5.4 Conclusion to Results

The first impression that one gets from these sets of experiments is seeing how different in terms of accuracy the first experiment of 20 tracks with the tandem-run Python Class was to the second experiment of the tandem-run with 48 tracklets: Figure: 5.2. Why, one may ask, is the second experiment not as accurate as the first? The answer lies in the amount of data that is used when the tandem-run Python class is in used. For the first experiment, the files are of tracks, not tracklets. Each track is about 1.5 minutes long; thus allowing for enough angles to populate the arrays and to be passed into the histograms. The tracklets, on the other hand, are smaller sets of data. One file may contain more than one reset; every time the function detects a reset, it treats it as a new “track”; thus, when a file contains two resets, there are 3 sets of tracklets. The lack of data which goes into the tandem-function is the reason why, sometimes, the function cannot properly re-join the tracks accurately. This, as a result, lets us know with confidence that if this project wants to use the tandem-run function as a cost measure to be used when tracklets are present, then one needs to also make sure that the tracker does not reset too often. Having a file which contains multiple resets will yield countless tracklets with small time intervals in between them; thus making the tandem-run function unreliable.

The second impression from these experiments is that, when re-joining the tracklets, the prediction of points is a good starting point. By having enough confidence that predicting points will help with re-joining tracklets; one can safely conclude that the prediction function (or every return number from this function) is a good candidate for a future cost function tracker. It is for the future because, as of now, and even still, the use of the tandem-run alongside the prediction function is being tested in order to start prototyping an “ant tracker” which will take into account behaviour and other variables as parameters for a cost function. If one is confident that prediction yields an average of 80% accuracy when re-joining tracklets, then one can assign the prediction in the cost function to have, perhaps, more weight (or value) than the tandem-run function because prediction shows more confidence than the tandem-run.

Lastly, one of the files which displayed a wrong prediction was observed to have the right tandem-run joined tracklets. This is to say, the tracklets were incorrectly joined by predicting the points but correctly joined by using the tandem-run function. Although this is just one of the 22 files, it gives this project enough confidence to say that: by having a bidirectional relationship between prediction and behaviour, one can, perhaps, construct a better cost function model which would take this experiment from 81% accuracy to something a bit higher.

## 6. *Evaluation*

This chapter will start with an evaluation of the project’s objectives and the methodology of the project.

### 6.1 Completion of Objectives

This section discusses whether the four main objectives have been met by this project’s investigations, experiments and results. There were a series of evaluations throughout the period from which this project began. Members of the Bristol Ant Lab were kind enough to meet with some of us every week to see our progress and to give inputs, ideas and suggestions. Their feedback is invaluable since it is for their use that this project (and the many others which take place under the Computational Ant Lab) is directed. During these meetings, many of their evaluations proved to be crucial for the progress of some of the implementations mentioned earlier. As a result, the period of researching and implementing the code was also interwoven with many evaluations from members and staff.

#### 6.1.1 Tandem-Run Classification

This was one of the earliest problems which this project tackled and was set to improve. As mentioned in one of the objectives, the development and experimentation of classifying the tandem-run was primary. The histograms of the leader and follower ants were shown to members of the Bristol Ant Lab; their feedback was positive, with reviews such as “impressive” and “novel”. One member, of course, wanted to see more data since at the time of this evaluation, the histograms were of one file of tracks. Another member of the lab commented that these findings were “the icing on the cake” and that the project needed to be heading in the direction of automating and capturing accurate data. This feedback directed the project towards automating the restart function from the trackers to help with the automation and of the semi-automated tracker for the gathering of data.

#### 6.1.2 Automating the Restart of Trackers

This was a prototype of a “new” way of tracking. Previously, with BrisTracks, if a track had gone wrong, it would continue to track and output wrong data. This was not



good, especially at the time when this project was trying to gather more data to run the tandem-run experiments. Therefore, for a long period of time, the semi-automated tracker and the reset function were implemented and tested. The members of the Bristol Ant Lab liked the semi-automated way of tracking and commented that it was “useful” for gathering data. Of course, they didn’t find it “useful” for their since there is no Graphical User Interface; however I demonstrated the way in which I was able to track ants using this process alongside the KCF algorithm; at which point they were very excited to see that ants were being accurately tracked inside the nest. Thanks to this, there was a lot of talk of conducting and recording more migration experiments with better recording equipment and better lenses; the idea behind this is to capture as many features of the ant as possible. The experiment inside the nest proved to be a success since the bounding boxes were surrounding one feature of the ant, the gaster.

The members of the Lab do agree that, perhaps, it is a good idea to have a safety feature such as a reset function which would make sure that data is being accurately collected. A prototype for this was displayed and received with positive reviews. Of course, resetting the tracker needs to be done with better results. At present, this project’s reset function is “good enough” for the current experiments. However, as soon as more ants are added to the videos, things start to scale quickly. Scalability was an issue which was much debated in the following weeks. However, scaling this prototype is well beyond the scope of the project; the main objectives were to explore novel ways of improving tracking ants in lab experiments and to propose better automatizing methods.

### 6.1.3 Behavioral Tracker

This is a prototype which was suggested towards the end of the project. The reason being that this project needed to conduct experiments and to gather results in order to test different approaches to the problem of tracking social insects. It was demonstrated that we are capable of extracting a tandem-run from ant tracks, and that we can also re-join them based on behaviour and the predicted distance. There is, currently, much work being done to launch a prototype of this cost function which would combine both of these inputs. The idea of using these two features to join tracklets was met with mixed reviews: some members did find this approach quite “novel” and “interesting” since it tries to use behaviour as part of the component for better tracking. However, others quickly began to notice that, in terms of scalability, this proposed way of re-connecting tracks might not scale properly. Yet, I believe that this is an issue which could be solved with more time to develop ways in which one can help the automatic reset function from performing too many resets; this will reduce the amount of tracklets.

The Lab did think that, perhaps, it is a good idea to have tracklets instead of having a tracking software which does not check for errors. It was shown that trackers will fail at some point, and that it is best to play it safe by creating a “band aid” solution to a bigger problem. Re-joining tracklets properly could in fact enhance the accuracy of classified behaviours and tracked ants.

## 6.2 Evaluating the Methodology

This section discusses the project’s decisions, challenges and successes faced during the investigation and implementation phases of the project.

On a personal level, the decision to attempt to write a semi-automated tracker from scratch (inspired by various tutorials which were thoroughly studied either from the OpenCV website or reputable users) proved to be a success. A success because I was able to learn the basic process chain which a tracking software would require. I managed to satisfy personal challenges such as keeping things as simple as possible and uploading working scripts to GitHub every day I was working on an implementation. By following certain industry standards suggested in [40] [41], I was able to create an environment which helped me improve coding practices.

Building scripts from the ground up did come with their own unique challenges. Recurring bugs were a constant obstruction of the progress of this project as there were many days when it felt as if things weren’t advancing. It also meant that basic concepts, such as data structures and Python practices, needed to be understood as well as possible; thus resulting in days spent just practicing Python or studying very basic functions. Yet, all in all, it was all part of my personal objectives: learning Python, data analysis and implementing industry standards.

The decision to ask staff from the Department of Computer Science for help proved to be immensely beneficial as there were concepts (such as trigonometry) that I did not understand. By using trigonometry, I was able to build the tandem-run Python function which classifies a leader from a follower. The open door policy which the University adheres to was a great help. In hindsight, I would have approached staff even earlier to ask for help and to discuss the project’s direction.

The decision to propose a cost function which will take prediction and behaviours as parameters is, perhaps, the most inventive feature of this project as it is the culmination of the work researched in this project. Professor Neill Campbell qualifies it as “unique”

since it uses the unique features of *Temnothorax albipennis* migrations for the tracker's advantage.

The results section was almost all done manually; that is to say, the results needed to be manually compared with my own notes in order to make sure that things were consistent. This is yet another area which needs to be further improved since, as of now, many tests needed to be conducted manually. Time during the Summer is very short and I couldn't find time to start automating certain tests which I believe needed to be implemented. In hindsight, I would try to do less and adhere even stronger to more of the standards proposed in the books.

### 6.3 The Project's position in Research

This section discusses this research's position with the current ant trackers discussed in this thesis.

Perhaps, the most original implementation discussed in this thesis is the tandem-run Python class which classifies a leader ant from a follower based on their angle of direction and past frame information for files of tracks. Whereas other methods use frame by frame detection of shapes (BrisTracks uses the fact that two rectangle come together on the smallest side to classify it as a tandem-run) as their main classifier component, our approach focuses on the overall behaviours of the track: that is, we try to analyze how it "moves" or "behaves" throughout time. This method proved to be successful when presented with files of tandem-run ants and is a "step forward" in classifying ants based on their past behaviour.

Most trackers use features of their targets to try to keep a degree of accuracy. Our proposed method of tracking still follows this approach (we still need to track something as accurate as possible); however, because of the investigation carried out, we are confident that trackers will fail at some point in time. It is this failure that we try to exploit by detecting when it happens to create tracklets instead of tracks of wrong data. Our proposed method of tracking will continue to follow the standards of tracking; however, we try to add extra bits of information to increment the accuracy of rejoining the tracklets.

All of the algorithms, trackers and frameworks which I tested do not have a restart function which would destroy the tracks and restart the tracker with a fresh set of ids for every detected blob. The fact that these trackers lack a reset function implies that users and developers rely on the fact that their algorithms will work to some degree. However, for conducting important experiments such as the ones done by the Bristol Ant Lab, would

require some degree of confidence or, perhaps, a safety method which would check for failures. We introduced a prototype of this safety measure to introduce the fact that it is possible to check for failures by checking that every bounding box has a blob inside it.

Lastly, professor Neill Campbell said to me that “as far as I can tell, nobody has tried to rejoin tracklets by using behaviours”. This is because extracting behaviour is quite difficult specially when animals tend to modify their behaviours depending on infinite number factors. However, we are confident that using behaviour to re-join the tracklets in *Temnothorax albipennis* migration experiments might work because these experiments take into consideration one aspect of the ant’s life: the migration. It was explained in Chapter 2 that these ants will display very marked behaviours during the migration. It is these marked behaviours which our prototype of a cost function will latch on to produce results.

## 7. *Conclusion and Future Work*

This chapter will detail ways in which this project can be used to aid future projects. A detailed explanation of the incorporation of the project’s best features and successes will follow.

### 7.1 Future and Work

The bulk of the investigation and testing of this project weighs upon the fact that the results suggest that any future work can begin to incorporate the best features found by this project’s resolution into either BrisTracks or new social insect trackers. Here are some suggested ways to take this a step further.

#### 7.1.1 Video Quality

Videos of *Temnothorax albipennis* migrating from nest to nest need to be of better quality. It was demonstrated from the “inside the nest” experiment that these OpenCV algorithms work best when they can extract some kind of distinct feature from the object being tracked. To avoid losing tracks of ants which exit the frame, the arena needs to be smaller and the camera positioned at a height which can record the walls and all of the ants inside the arena. The nests need to be brought closer together<sup>1</sup> so that the camera can be positioned at lower height to help with the quality of the ant’s features. Lastly, a wide angle lens needs to be tested with these experiments to see whether it would distort the ants too much. By having a wide angle lens, the camera could be brought closer to the arena for even better resolution. Video quality is very important and the best equipment needs to be used for these experiments to ensure highest quality and better contrast of the ant’s features.

#### 7.1.2 OpenCV Newer Version

OpenCV contributors work really hard to release better and faster versions of the software. These versions need to be tested constantly with the migrations of *Temnothorax albipennis* in order to ensure that the best properties of OpenCV are being put to use. Currently,

---

<sup>1</sup>Nigel Franks suggests that 15 cm between nests is the minimum required distance for the ants to display their behaviours.

version 3.2.0 does not include the Convolutional Neural Network class because there is a bug. However, CNN could be an even better way to lose less tracks during the tracking experiments.

### 7.1.3 Social-Carry

**From inside the nest** is very difficult to tell if there's a social-carry because all of the ants are constantly touching each other and because, when background subtraction is applied, the nest looks like one big blob. However, last year's project demonstrated that, by having an area threshold for ants, one can use the fact that if it surpasses this threshold this can be classified as a social-carry. Although this presents a problem when single ants are bigger, one way to classify a social-carry when it leaves the nest is to use the OpenCV contour class. It seems, from various times when the videos were observed, that social-carried ants have a particular shape due to the fact of the carry. From an overhead camera, this shape resembles a sickle. One could use a shape finder to detect if there is a sickle-like object; if there is, then this can be tagged and classified as a social-carry. A size threshold could then be applied for even better accuracy.

**From the arena:** since this prototype uses bounding boxes, it was found out that when ants performed a social-carry from the middle of the arena, the two bounding boxes would have the same track coordinates after one ant picked up the other. Although there was no time to incorporate this feature into the classifier, one could use this information to deduce a social-carry. For example, upon analyzing the track file, if two tracks are exactly the same for a period of time, then this can be tagged as a social-carry.

### 7.1.4 Cost Function

Of course, the biggest proposal of this project is to create a "behavioural cost function" which would aid the accuracy of joining tracklets. More data and more tests need to be carried out to ensure the best possible weight for each of the features (i.e: should prediction be weighted more than behaviour because prediction is more accurate?). These are the questions that future projects can tackle as more tests are conducted and more data is extracted.

### 7.1.5 Nest Tracking

By studying better features for blob detection and tracking from OpenCV, one can begin to create a nest tracker. This project demonstrated that it is possible to use KCF and

bounding boxes to track the gaster of the ants. This was done in a semi-automated approach: the user had to create bounding boxes around the ant's gaster. However, better ways of detecting blobs, or even better, training a data set to identify gasters, could be used to dynamically create bounding boxes around the ants' gasters. Yet another way that this could be taken further is to use the example from Figure 3.12 for blob detection. The preliminary work with Student Dave's blob detection did show that it was possible (and highly likely) that the detection would place the cross-hairs dead in the centre of either the gaster or the head of the ant; thus suggesting high accuracy of detection in highly populated areas such as inside the nest.

## 7.2 Conclusion

This project investigated various ways to tackle the problem of accurately tracking interacting *Temnothorax albipennis* ants. Various proposals for the refactoring of BrisTracks and various other ant tracking software's were presented.

The project's background research surveyed the importance of conducting these experiments, the two interesting behaviours from *Temnothorax albipennis*, the difficulties of tracking these and why most trackers fail to disambiguate interacting featureless blobs. The preliminary research demonstrated that trackers often fail when trying to accurately track hundreds of featureless objects for intervals of hours. For that, a new way of tracking these was proposed: insert resets when detection of failure occurs and then try to re-join the tracklets at post-production based on certain parameters.

OpenCV was heavily studied in order to create prototypes of potential candidates for extending BrisTracks and/or other trackers. A tandem-run classification methodology was proposed and tested with results that give us confidence that one can extract the behaviour of these ants in order to aid the re-joining of the tracklets. A semi-automated Python class was created to help with the gathering of accurate data to test our hypothesis. A prototype of an automated tracker with a fail detection and reset was implemented in order to demonstrate that it is possible to detect failures which can be reset in order to create tracklets. Joining the tracklets at a post-production stage was researched and tested in order to demonstrate that this is possible. By using prediction based on past-frame information, we were able to increase the accuracy of the re-joined tracklets. Lastly, testing the tandem-run Python class with the tracklets showed that this is also another extra bit of information which can be used, alongside the prediction, to improve the accuracy of re-combining the tracklets.

Results proved that the tandem-run Python class works faultlessly when the tracks of specific tandem-runs are long enough to contain enough data. This function begins to fail when the tracks are small (for example in some tracklets). This prompted us to begin to work on better ways of resetting the tracker. Results also demonstrated that predicting the movement of ants is a good way to re-combine tracklets and that behaviour can be another great feature to incorporate in a future cost function.

**Project Successes:**

- Demonstrated that by using past-frame information one is capable of creating a classifier which can accurately detect a tandem-run's leader and follower ant. This is an improvement from classifying ants based on their current frame interaction.
- Created a tool which can be used to extract data from videos of ant migration; this was hugely important since it was needed to conduct several experiments.
- Demonstrated that tracking hundreds of featureless objects is a difficult task by testing several frameworks and algorithms. The field of computer vision has been working for decades on the problem of accurately tracking objects. This project, as a result, proposes a new way of tracking featureless objects: 1) trackers will fail, use a reset to create tracklets, 2) re-joining by building a cost function which can include a behavioural feature.



## A. *Code Samples*

```
11
12 # Setup SimpleBlobDetector parameters.
13 params = cv2.SimpleBlobDetector_Params()
14
15 # Change thresholds
16 params.minThreshold = 10
17 params.maxThreshold = 200
18
19
20 # Filter by Area.
21 params.filterByArea = True
22 params.minArea = 10
23
24 # Filter by Circularity
25 params.filterByCircularity = True
26 params.minCircularity = 0.1
27
28 # Filter by Convexity
29 params.filterByConvexity = True
30 params.minConvexity = 0.87
31
32 # Filter by Inertia
33 params.filterByInertia = True
34 params.minInertiaRatio = 0.1
35
36 # Create a detector with the parameters
37 detector = cv2.SimpleBlobDetector_create(params)
38
```

FIGURE A.1:  
Code to specify  
the correct area  
and circularity  
to detect. The  
parameters need  
to be specific  
to the target in  
question [42]

```
1 # create a folder to store extracted images
2 # From https://stackoverflow.com/questions/33311153/python-extracting-and-saving-video-frames
3 import os
4 folder = 'socialCarry_2'
5 os.mkdir(folder)
6
7
8 import cv2
9
10 vidcap = cv2.VideoCapture('/home/seth/Host_AntVideos/Examples/edited_video/00001b.mp4')
11 count = 0
12 while True:
13     success,image = vidcap.read()
14     if not success:
15         break
16     cv2.imwrite(os.path.join(folder,"frame{:d}.jpg".format(count)), image) # save frame as JPEG file
17     count += 1
18 print("{} Yout Images are extacted in this place {}".format(count,folder))
19
```

FIGURE A.2: A Python script to extract frames as image files from video clips.

FIGURE A.3: C++ code of the Multi-tracker initialization.

```

35
36 // set the tracking algorithm from parameter
37 if(argc>2)
38     trackingAlg = argv[2];
39
40 // create the tracker
41 MultiTracker trackers(trackingAlg);
42
43 // container of the tracked objects
44 vector<Rect2d> objects;
45
46 // set input video
47 std::string video = argv[1];
48 VideoCapture cap(video);
49
50 Mat frame;
51
52 // get bounding box
53 cap >> frame;
54 selectROI("tracker", frame, objects);
55
56 //quit when the tracked object(s) is not provided
57 if(objects.size()<1)
58     return 0;
59
60 // initialize the tracker
61 trackers.add(frame, objects);
62

```

FIGURE A.4: Python code of the Multi-tracker initialization.

```

173
174 # Initialize the video by selecting, manually, the ants to track
175 if roi == False:
176     boxess = list()
177     tracker = cv2.MultiTracker(algorithm)
178     r = cv2.selectROI(image)
179     bbox = (r[0] - 5, r[1] - 5, length, length)
180     boxess.append(bbox)
181     while True:
182         key2 = cv2.waitKey(1) or 0xff
183         if key2 == ord(' '):
184             r = cv2.selectROI(image)
185             bbox = (r[0] - 5, r[1] - 5, length, length)
186             boxess.append(bbox)
187         if key2 == ord('q'):
188             break
189     for bboxes in boxess:
190         tracker.add(image, bboxes)
191     ok, boxes = tracker.update(image)
192     roi = True
193

```

FIGURE A.5: Python implementation of the point outside the box formula.

```
# Check if the keyPoint is inside any of the boxes
if (0 <= np.dot(vectorize(A,B), vectorize(A,M)) <= np.dot(vectorize(A,B), vectorize(A,B)) and \
    (0 <= np.dot(vectorize(B,C), vectorize(B,M)) <= np.dot(vectorize(B,C), vectorize(B,C))):
    print True
```

# Bibliography

- [1] Robie Alice A., Seagraves Kelly M., S. E. Roian Egnor, and Kristin Branson. Machine vision methods for analyzing social interactions. *Journal of Experimental Biology*, 2017.
- [2] Sokolowski Maria B. Social interactions in 'simple' model systems. *Neuron*, 65(5): 780–794, 2010.
- [3] Caro T. M. and Hauser M. D. Is there teaching in nonhuman animals? *The Quarterly Review of Biology*, 67(2):151–174, 1992.
- [4] Franks Nigel R. Lessons from teaching in ants. URL <http://www.bris.ac.uk/engineering/media/engineering-mathematics/anm-meetings/amss/Franks.pdf>.
- [5] D. Premack and G. Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and Brain Sciences*, 4:515–526, 1978.
- [6] Thomas O. Richardson, Philippa A. Sleeman, John M. McNamara, Alasdair I. Houston, and Nigel R. Franks. Teaching with evaluation in ants. *Current Biology*, 17: 1520–1526, 2007.
- [7] Leadbeater Ellouise, Nigel E. Raine, and Lars Chittka. Social learning: Ants and the meaning of teaching. *Current Biology*, 6(9):323–325, 2006.
- [8] Computational Ant Lab. URL <https://sites.google.com/site/computationalantlab/>.
- [9] Blaza Alex. Improved computer vision for tracking ants. *University of Bristol*, 2016.
- [10] Gardner Martin. The fantastic combinations of John Conway's new solitaire game 'life'. *Scientific American*, 223:120–123, 1970.
- [11] Wikipedia, . URL [https://en.wikipedia.org/wiki/Conway's\\_Game\\_of\\_Life](https://en.wikipedia.org/wiki/Conway's_Game_of_Life).
- [12] Bonabeau Eric, Theraulaz Guy, Deneubourg Jean-Louls, Aron Serge, and Scott Camazine. Self-organization in social insects. *Trends in Ecology and Evolution*, 12(5): 188–193, 1997.
- [13] Hauert Sabine, Winkler Laurent, Zufferey Jean-Christophe, and Floreano Dario. Ant-based swarming with positionless micro air vehicles for communication relay. *Swarm Intelligence*, 2(2):167–188, 2008.

- [14] Christensen Kim, Dario Papavassiliou, Alexandre de Figueiredo, Nigel R. Franks, and Ana B. Sendova-Franks. Universality in ant behaviour. *Journal of The Royal Society Interface*, 12:20140985, 2015.
- [15] Doran Carolina, Tom Pearce, Aaron Connor, Thomas Schlegel, Elizabeth Franklin, Ana B. Sendova-Franks, and Nigel R. Franks. Economic investment by ant colonies in searches for better homes. *Biology Letters*, 9:20130685, 2013.
- [16] ResearchGate. URL [https://www.researchgate.net/publication/256434778\\_Impact\\_of\\_Stress\\_on\\_Nest\\_Relocation\\_in\\_Ants](https://www.researchgate.net/publication/256434778_Impact_of_Stress_on_Nest_Relocation_in_Ants).
- [17] ScienceNews. URL <https://www.sciencenews.org/article/swarm-savvy>.
- [18] Davies E. R. *Computer and Machine Vision: Theory, Algorithms, Practicalities*. Elsevier, 2012. ISBN 9780123869081.
- [19] Piccardi M. Background subtraction techniques: a review. *IEEE International Conference on Systems, Man and Cybernetics*, 4:3099–3104, 2004.
- [20] Correll Nikolaus, Sempo Gregory, Yuri Lopez de Meneses, Jos Halloy, Jean-Louis Deneubourg, and Alcherio Martinoli. Swistrack: A tracking tool for multi-unit robotic and biological systems. *Intelligent Robots and Systems, IEEE/RSJ International Conference*:21852191, 2006.
- [21] Wikipedia, . URL <https://en.wikipedia.org/wiki/C++>.
- [22] OpenCV. URL <http://opencv.org/about.html>.
- [23] Wikipedia, . URL <https://en.wikibooks.org/wiki/SwisTrack>.
- [24] Marcovecchio Diego, Delrieux Claudio, Werdin Jorge, and Stefanazzi Natalia. A lightweight approach for analyzing insect behavior on a mobile system. *Conference on Computer Graphics, Visualization and Computer Vision*, 2014.
- [25] Lenz Philip, Geiger Andreas, and Urtasun Raquel. Followme: Efficient online min-cost flow tracking with bounded memory and computation. *IEEE International Conference on Computer Vision*, pages 4364–4372, 2015.
- [26] Fiaschi Luca, Diego Ferran, Gregor Konstantin, Schiegg Martin, Koethe Ullrich, Zlatić Marta Hamprecht, and Fred A. Tracking indistinguishable translucent objects over time using weakly supervised structured learning. *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

- [27] Malik M. Khan, Tayyab W. Awan, Intaek Kim, and Youngsung Soh. Tracking occluded objects using kalman filter and color information. *International Journal of Computer Theory and Engineering*, 6(5):438–442, 2014.
- [28] The pandas community. URL <https://pandas.pydata.org/>.
- [29] NumPy developers, . URL <https://numpy.org/>.
- [30] Eric Firing Michael Droettboom John Hunter, Darren Dale and the Matplotlib development team. URL <https://matplotlib.org/>.
- [31] Satya Mallick, . URL <http://www.learnopencv.com/object-tracking-using-opencv-cpp-python/>.
- [32] Junkai Ma, Haibo Luo, Bin Hui, and Zheng Chang. Robust scale adaptive tracking by combining correlation filters with sequential monte carlo. *Sensors*, 2017.
- [33] Chao Ma, Xiaokang Yang, Chongyang Zhang, and Ming-Hsuan Yang. Long-term correlation tracking. *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.
- [34] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 6, 2010.
- [35] Mathworks. Matlab. URL <https://www.mathworks.com/products/matlab.html>.
- [36] Stack Overflow. Calculate angle from three points, . URL <https://stackoverflow.com/documentation/math/7653/geometry#t=20170829162402345757>.
- [37] Stack Exchange. URL <https://math.stackexchange.com/questions/190111/how-to-check-if-a-point-is-inside-a-rectangle>.
- [38] Stack Overflow, . URL <https://stackoverflow.com/questions/2752725/finding-whether-a-point-lies-inside-a-rectangle-or-not>.
- [39] SciPy developers, . URL <https://www.scipy.org/>.
- [40] Andrew Hunt and David Thomas. *The Pragmatic Programmer*. Addison Wesley, 1999. ISBN 0-201-61622-X.
- [41] Principles of agile programming. URL <http://agilemanifesto.org/principles.html>.
- [42] Satya Mallick. Blob detection using opencv (python, c++), . URL <https://www.learnopencv.com/blob-detection-using-opencv-python-c/>.