Efficient Computer Vision and Machine Learning Methods for Automating Large-Scale Analysis of Collective Animal Behavior

Doctoral thesis for obtaining the
academic degree
Doctor of Natural Sciences
(\textit{Dr. rer. nat.})

submitted by
Walter, Tristan Leonard

at the

\textit{Universität Konstanz}

Faculty of Sciences
Department of Biology

Konstanz, 2021
Date of the oral examination: 08.04.2022

1. Reviewer: Prof. Dr. Iain D. Couzin

2. Reviewer: Prof. Dr. Oliver Deussen
## Contents

**General introduction**  

1.1 Introduction .................................................. 8  
1.2 Results ....................................................... 11  
1.2.1 Tracking: Speed and Accuracy .......................... 15  
1.2.2 Visual Identification: Accuracy ......................... 17  
1.2.3 Visual Identification: Memory Consumption .......... 22  
1.2.4 Visual Identification: Processing Time ............... 24  
1.3 Discussion ..................................................... 25  
1.4 Materials & Methods ......................................... 28  
1.4.1 Segmentation ................................................. 29  
1.4.2 Tracking ..................................................... 30  
1.4.3 Automatic Visual Identification Based on Machine Learning 37  
1.5 Software and Licenses ......................................... 51

2. Identifying the spatiotemporal locations of anomalies in collective behavior using simplified representations of trajectory data  

2.1 Introduction .................................................. 53  
2.2 Related Work .................................................. 56  
2.3 Materials and methods ....................................... 58  
2.3.1 Video recordings and study site ......................... 58  
2.3.2 Analysis Setup (Hardware & Software) .................. 59  
2.3.3 Preparing the dataset ...................................... 59  
2.3.4 Normalization and cleaning .............................. 60  
2.3.5 Simple anomaly detection (Computer Vision) ........ 62  
2.3.6 Anomaly detection using sequence prediction (machine learning) 63  
2.3.7 Manual labels .............................................. 64  
2.3.8 Scoring ..................................................... 65  
2.3.9 Narrowing down the first responders .................. 68
2.4 Results ................................................................. 68
  2.4.1 Hyper-parameter analysis ........................................ 68
  2.4.2 Comparing the methods: Simple vs. machine learning based 71
  2.4.3 Inspecting detections and finding first responders .......... 73
2.5 Discussion ............................................................ 76
2.6 Future ................................................................. 78

3 A deep learning method for the automatic visual categorization of individuals in high-resolution videos 80
  3.1 Introduction .......................................................... 81
  3.2 Materials & methods ................................................ 83
    3.2.1 Recordings ....................................................... 84
    3.2.2 Pre-/Processing with TRex ...................................... 85
    3.2.3 Pre-/Processing with Detectron2 ............................ 86
    3.2.4 Data normalization .............................................. 87
    3.2.5 Updates to TRex: An adaptive matching algorithm ...... 87
    3.2.6 Updates to TRex: Categorization ......................... 90
    3.2.7 Analysis: Improving the consistency of caste assignments 93
    3.2.8 Analysis: Approximating tactile interactions ............ 94
  3.3 Results ............................................................... 96
    3.3.1 Tracking: Evaluating tracking speed and quality of results 97
    3.3.2 Categorization: Evaluating the quality of caste predictions 97
    3.3.3 Behavioral analysis .......................................... 100
  3.4 Discussion ............................................................ 105
    3.4.1 TRex updates ................................................... 105
    3.4.2 Behavior ......................................................... 107

Concluding remarks and perspectives 111

Appendix A Appendix - Chapter 1 115
  A.1 Installation requirements and Usage ............................ 115
    A.1.1 Workflow ......................................................... 116
    A.1.2 Output .......................................................... 117
  A.2 From video frame to blobs ...................................... 122
  A.3 Connected components algorithm ............................... 124
  A.4 Matching an object to an object in the next frame .......... 127
    A.4.1 Terminology ..................................................... 127
    A.4.2 Background ..................................................... 128
  A.5 Posture ............................................................... 141
    A.5.1 Connecting pixels to form an outline .................... 141
    A.5.2 Finding the tail .............................................. 144
Abstract

Research into biological systems has, as many other fields of research have, greatly profited from an ever-increasing range of computational tools. This is important, not just because automation can make life easier, but because it enables researchers to increasingly address questions that were previously impossible to answer. Around 50 years ago, in 1972, D. Radakov published his book on “Schooling in the Ecology of Fish”, detailing a vast number of interesting hypotheses and experiments to investigate the underlying factors of schooling - none of which he was able to evaluate quantitatively. Not for the lack of trying, but simply because the required technology did not yet exist. Back then, while already enjoying the benefits of the “motion picture film”, the paths and shapes of individuals had to be traced manually. This limited researchers to work with mere seconds of trajectory data, which is hardly enough to form any scientific conclusions about the rules governing each individual’s movements, or the sub-second structure of alarm responses. High-speed cameras, automated tracking and analysis, as well as a much more interconnected scientific community (especially because Radakov lived in the Soviet Union), are developments of the past few decades that significantly improved the prospects in, but not limited to, animal behavior research. In this thesis, I present multiple methods which are specifically designed to assist research in collective behavior, ecology, biomechanics, and hopefully others, that are powerful, yet pragmatic.

In the first chapter of my thesis, I present a tool, called TRex, that combines implementations of various methods, each addressing different aspects of typical video-based analyses of behavior, and highlight the importance of their synthesis into a single piece of software - as opposed to the prevalent fragmentation of tools in our field of research. The main focus of this chapter is a machine learning based method for the unsupervised visual identification of individuals, which I demonstrate to be superior to previously available implementations. Importantly, this chapter also builds the foundation for the later chapters, where I take these approaches further, addressing increasingly more difficult scenarios. In Chapter 2, where I focus on the exploration and handling of very large datasets using, as an example, field-data of the termite species Constrictotermes cyphergaster (Silvestri 1901). These eusocial insects are known to live in large colonies, making them an ideal benchmark for the application and, consequentially, improvement of my methods. Furthermore, on this basis, using a simplified representation of trajectory data, I present two methods for behavioral anomaly detection that are able to identify both the temporal and spatial coordinates of unusual events. Finally, in Chapter 3, I address the importance of inter-individual morphological differences, from both a methodological and a biological perspective. There, I present improvements for the matching algorithm I developed in Chapter 1, as well as a machine learning algorithm specifically designed to automatically identify different morphological “types” of individuals under minimal supervision. I end this chapter by using many of the previously described tools to further investigate the dataset I used
in Chapter 2, primarily focusing on the differences between two different types of termites (castes), “workers” and “soldiers”, in terms of their movement and an approximation of tactile interactions. This is, to our knowledge, the largest quantitative study of our study species (*C. cyphergaster*).

Together, the methods presented in this thesis form a versatile chain of efficient computer vision and machine learning tools to automate tracking, posture estimation, recording, dataset exploration and to classify morphological types of individuals. Despite the focus on termites in the last two chapters, all of these approaches are designed to work for a wide range of species and in different situations, with many possible applications in a wide range of research fields, including computational biology, neurobiology and ecology.


Historically, research in animal behavior has taken one of two forms: Experimental work in the laboratory, or observational field work. While we nowadays recognize the importance of both, conflicts about the proper way of conducting scientific research in Biology started in the 19th century and has continued since [111, 59]. The cause for this was the fear of biologists in the 19th century, that their descriptive observations from the “fields and woods” [111] would no longer be considered scientific enough - they found a seemingly better alternative in laboratory work (likely inspired by the successes of Physics at the time, [111]). Later, this heavy focus on Comparative Anatomy and dissection became the center of criticism - that zoologists sat in their laboratories, rather than going “out to see for themselves what animals did with all the organs portrayed in anatomy handbooks”, as Tinbergen put it in [212]. This conflict was likely a replicate, or potentially even the consequence, of an earlier conflict around the term “Comparative Psychology” in the late 1800’s. A paper from 1969, by Julian Jaynes [111], compares both histories and fittingly concludes on the words:

“The unknown tyranny of the past is the source of all unreason.”

It certainly was in a time of conflict, the early 1900’s, that the fundamental concepts of modern ethology were conceived. A push for more systematic field observations, by researchers like the Nobel laureates Lorenz [135] and Tinbergen [212], was formative for our own work in animal behavior today - all quantitative descriptions of animal behavior bear their mark.

Fortunately, both sides of the historical argument are nowadays increasingly losing their distinctiveness. Modern technology allows researchers to perform behavioral experiments in the wild, record and analyse the video footage on-site or back at the lab - dissecting it, if you will, into quantitative descriptions. Creating ethograms (thorough descriptions of the behavioral space of a species, which can have different forms i.e. long lists of “Eating: snout in contact with food”, or as in [214, 213]), as it has been proposed by early ethologists such as Tinbergen, has always been a large undertaking
and usually involved many years of meticulous observation. The emerging computerized production of
such ethograms, which is already partly possible [14, 87], is starting to change this. The dreams
of any ethologist - to merely point a camera at something for some time and get a breakdown of its
behavioral space - may soon become a reality.

Naturally, before such higher-level goals can be attempted, the ground work has to be laid. Au-
tomating the detection and tracking of individuals is essential to be able to provide the data necessary
for the, usually data-hungry, types of algorithms referred to above. A hunger that can barely ever be
satisfied by painstaking manual work, but is increasingly addressed by computer vision and machine
learning based approaches (e.g. [86, 184, 202, 160, 145, 201, 73, 79], etc.). This strive for automation
is, of course, not intended to replace human expertise entirely (although that cannot be ruled out).
Subject-specific knowledge about the model organism is still going to be an integral part of the plan-
ing and execution of experiments, and even more so for the interpretation of results. The goal is to
improve the life of researchers by letting computers take over the more tedious tasks.

Tracking is the best example of this: it is essential to many studies in animal behavior, yet, while
humans are able to successfully track the identities of a small number of individuals, more than four
at a time already constitute a problem [132]. The manual approach that has been employed histor-
ically, tracing video recordings by hand, is only applicable to very short videos and clearly demon-
strates what researchers were willing to go through for a small number of data points - as in [174],
where, by repeatedly projecting the film on a sheet of paper and marking the position of each indi-
nual, a few seconds of movement were “tracked”. Modern algorithms have already been used in
many research projects (see e.g. [159, 73, 28, 177] for tracking), and allow for much larger groups and
longer videos to be tracked (e.g. [173], also in Chapter 1). Other areas, such as the estimation of ani-
mal outlines and postures, have profited similarly. Where the body contours of *Trichoplax adhaerens*
previously had to be traced by hand (see [163] from 1989), it is now possible to keep track of shapes
and even individual limbs with little manual work (e.g. [86]).

The visual identification of individuals, which is covered in more detail in Chapter 1, is arguably
one of the best outcomes of these recent advances. While it is sufficient for the study of general
group-level effects (like general direction and center of mass), maintaining identities is essential for
the investigation of individual-level effects. Even with a perfect tracking algorithm, when multiple
individuals emerge from a refuge, and with no other source of identity information, the algorithm
cannot know who is whom, and mistakes can quickly accumulate. Visual identification, essentially
a more general form of face recognition, can provide the information necessary to correct potential
mistakes in videos of unmarked individuals it is finally, for the first time, possible to say that the pro-
duced trajectories are, indeed, associated with the correct individuals.

While the advantages of automation in many areas of data collection are undeniable, practical challenges accompany the advent of every novel technology. Researchers now not only have to be knowledgeable in their own field of research, but increasingly also in various areas of computer science - which includes, somewhat unsurprisingly, "technicalities": The recording hardware, the software and details about its segmentation algorithm (e.g. color based, background subtraction, machine learning based) have to be taken into account when designing experiments, along with the amount of space available on-disk and its write-speed. Sometimes, previously minor choices of equipment, like lights and curtains, are now crucial and small mistakes can be impactful. For example, videos with high temporal resolution have to be recorded strictly using direct-current LED lights, since the flicker of alternating current lights would halve the amount of video frames that are usable well-lit. Furthermore, many animals (such as most insects and birds), have a much higher temporal resolution (flicker fusion frequency) and thus alternating current lights would be perceived to flicker and thus would be non-naturalistic [192]. Temporally inconsistent lighting (like leaky window blinds), spatially inconsistent lighting (like shadows), and inadequate camera parameters (like aperture or focus) can negatively influence both the performance of visual identification algorithms, for example, and how long it takes to initialize them. Even the wall-adhesion of water around container edges has previously caused problems in some of our experiments with fish, to the point that a subset of frames became unusable for visual identification. This is further complicated by the differences between species and experiments - all experimental design choices also have to be validated against the temporal and spatial scales of the question: studying fast moving animals entails different hardware and software requirements than do studies on sedentary species.

The interaction between different softwares can be a major complication, too. Data formats may be different, and likely incompatible, between programs. Positions may be calculated with reference to the head/tail, or center position of the body. How will I load the XY positions into my visual field estimator? When the overhead for each research project is large, how are researchers going to focus on the actual study? A very real issue, and likely a direct consequence of this, is the development of a large number of very specialized solutions for niche problems. While this may be counter-intuitive at first glance, it is often simply faster to write your own software than to learn using existing ones - fully accepting that your own solution might be untested, or subpar in comparison. Thinking about my own justifications for making another tracking program, another visual field/posture estimator, another visualisation tool, and another tool for visual recognition, I set myself two primary goals:
(1) **Tackle the root cause by uniting all features under one banner.** This is what I argue in Chapter 1, where I combine many different methods in one user-friendly, open-source package (TRex) to streamline many of the tedious parts of behavioral analyses. Generality is an important addition to this point, since specialized approaches only further the fragmentation of tools.

(2) **Improving on what is already there.** Many available solutions are likely consequences of the issues mentioned above, and are thus often suboptimally implemented. Custom tracking algorithms may even be very accurate in terms of assignments, but are often also very slow and unoptimized, or use impractical amounts of RAM. This is not the fault of researchers (who did the admirable job of implementing the study and a software simultaneously) because, while it would often be relatively easy, fixing these optimization issues is usually not worth the additional effort for just one study.

To ensure the generality of the developed approaches in the following chapters, where necessary to test their general applicability to other species, data from different experiments and of various animals will be used. There are many established model species, which, due to their widespread use, should be included in these tests: fish are often used in collective behavior research \[208, 92\] and other fields \[67, 115, 54\], not just because many of them are social, display interesting behavioral features \[77, 174, 47\], (their embryos) are transparent \[67\] and easy to reproduce \[115\], but also because they can easily be observed in a laboratory setting (i.e. an aquarium). Rodents \[199\] and insects have been studied for similar reasons \[170\] (see \[61\] for an overview) - social insects are of particular interest to the study of collective behavior \[85, 84, 46\].

Following the rules stated above, the first chapter of this thesis focuses on the improvement and unification of some of the most fundamental tools used in animal behavior research today. These comprise an improved algorithm for recording and tracking (with closed-loop feedback), posture- and visual field estimation, and for the visual identification of individuals, controlled using an intuitive graphical user-interface (GUI). Because of the fundamental nature of these methods we used data from all of the different types of organisms listed above: large schools of fish are used for testing the speed and stability of the tracking algorithm; locusts, termites, and rodents have more complicated body shapes than most fish and so are useful to test the posture estimation - and together, they will allow us to test the versatility of our visual identification algorithm.

As an important extension of (2), optimizations do not merely benefit researchers in terms of time management, but can also allow them to ask entirely new questions. Significantly improving
the speed and memory profiles of a tracking program, as a pertinent example, allows us to investigate much larger groups of individuals at high spatial and temporal resolution. Since it is apparent that this, as a consequence, produces the problem of extremely large datasets, building robust routines to deal with them is essential. Routines are established through practice or, in this case, by working with organisms that live in large groups, thus imposing considerable computational challenges. Many animals live in groups, but not all of them are easy to acquire or record (e.g. due to size or cryptic behavior). Arguably, eusocial insects (e.g. bees, ants, and termites) are great for this. Termites are especially well-suited - for three major reasons:

(1) They usually appear in large numbers and are relatively small (3-4mm), which enables us to record their entire behavior with a single, unmoving camera - a much easier task than footage from a moving drone (see e.g. [86] and herdhover.com).

(2) Despite not being closely related to ants at all, termites do, similarly to ants [11, 126, 204], display interesting collective behavior and e.g. form trails (like Constrictotermes cyphergaster [155, 51]), or build tunnels (e.g. Coptotermes formosanus [97] or desert termites like Bau-caliotermes hainesi [218] - but not C. cyphergaster).

(3) Termite behavior remains relatively unexplored, compared to other eusocial insects mentioned above. Likely because they are difficult to keep alive under laboratory conditions - e.g. a high risk of fungus infections within their preferred, high humidity conditions [44, 68].

In the Brazilian wet seasons of 2019 and 2020, H. Hugo recorded videos of entire nests of freely behaving termites (Constrictotermes cyphergaster, Silvestri 1901) on a flat surface. Finally being able to track the foraging behavior of entire nests of termites (up to 3000 individuals visible simultaneously), and for multiple hours at a time, is an exciting development.

As part of the initial data exploration, Chapter 2 investigates behavioral outliers to determine whether anything “out of the ordinary”, or anomalies in collective behavior, can be automatically detected. I evaluate various deep neural networks and their use for anomaly detection in tracking data, along with a simpler/classical approach. The goal was to see if anomalies can be detected automatically in separate trials, and if using more computationally intense machine learning (or ML) algorithms is beneficial, as opposed to using simpler, more traditional image processing techniques. Both approaches operate on image-like data, composed of moving-average density and speed data of termites in two of the aforementioned videos, and score each part of the video with regards to its predictability (i.e. an “anomaly score”). These are produced by first tracking all individuals, and
then reducing their trajectory data into a simplified representation: sequences of multi-dimensional heatmaps. I demonstrate that it is possible to detect anomalies using this reduced form the trajectory data, and, to complete the circle, that it is possible to find the implicated individuals in the original tracking data.

Finally, in the last chapter, I assess the visual and behavioral differences between individuals more closely. An important aspect of termite physiology and behavior, which is not investigated in 2, is the multitude of developmental pathways in *C. cyphergaster*. Depending on environmental and genetic factors, individuals eventually differentiate into members of specific *castes* after leaving their larval stage \[152\]. Technically, there are more than two castes, but focussing on the two distinct end-points of the *apterous* line is reasonable here, since they are likely the only ones to leave the nest and thus to be encountered in our recordings. The two castes, “workers” and “soldiers”, take up different tasks in the colony, as is also apparent by their adaptive physiological differences \[152\]. As such, workers are built for “the collection and transformation of biomass” \[155\], while soldiers are very much equipped for defensive behavior (with a sort of “canon”, a frontal protrusion of their head from which they can secrete toxins \[45\]). These morphological, and presumably behavioral, differences make it even more important to quantitatively assess them in a group context. Since we have no knowledge of any algorithms for the large-scale assignment of categories, a new solution was required. In fact, to the best of our knowledge, chapters 2 and 3 are the first attempts to apply machine learning techniques to large-scale tracking data of termites. Previously, only few individuals were tracked/detected and usually with no involvement of ML (26 in \[188\], but no ML; 10 in \[109\], mainly as an exercise in path reconstruction). Other ML based studies appear to be somewhat unrelated and focus on the (visual) detection of local termite presence via their effects on the environment \[134, 3, 187\], and, even less related, on the analysis of mound material \[139\].

As the main focus of Chapter 3, I adapt the algorithm for visual identification from Chapter 1 to differentiate between *categories* of individuals instead of their identities. This is possible by leveraging the inherent visual differences between them, and the plasticity of machine learning algorithms - samples from multiple videos are labelled in a manual annotation process, culminating in a single network that works for all of them. This phase is aided greatly by transfer learning, a specially designed user interface and, importantly, the use of animations. Animations enable users to, instead of single frames, annotate hundreds or thousands of images with every click. Being able to determine castes for all individuals is a step forward for the analysis of termite behavior, allowing me to conclude the final chapter with a quantitative investigation into potential kinematic (and related) inter-caste differences - in a dataset of over 600 million individual detections of *C. cyphergaster*. 
TRex, a fast multi-animal tracking system with markerless identification, and 2D estimation of posture and visual fields

Automated visual tracking of animals is rapidly becoming an indispensable tool for the study of behavior. It offers a quantitative methodology by which organisms’ sensing and decision-making can be studied in a wide range of ecological contexts. Despite this, existing solutions tend to be challenging to deploy in practice, especially when considering long and/or high-resolution video-streams. Here, we present TRex, a fast and easy-to-use solution for tracking a large number of individuals simultaneously using background-subtraction with real-time (60Hz) tracking performance for up to approximately 256 individuals and estimates 2D visual-fields, outlines, and head/rear of bilateral animals, both in open and closed-loop contexts. Additionally, TRex offers highly-accurate, deep-learning-based visual identification of up to approximately 100 unmarked individuals, where it is between 2.5-46.7 times faster, and requires 2-10 times less memory, than comparable software (with relative performance increasing for more organisms/longer videos) and provides interactive data-exploration
within an intuitive, platform-independent graphical user-interface.

1.1 Introduction

Tracking multiple moving animals (and multiple objects, generally) is important in various fields of research such as behavioral studies, ecophysiology, biomechanics, and neuroscience ([56]). Many tracking algorithms have been proposed in recent years ([159], [73], [28], [177]), often limited to/only tested with a particular organism ([99], [24]) or type of organism (e.g. protists, [165]; fly larvae and worms, [181]). Relatively few have been tested with a range of organisms and scenarios ([173], [201], [183]). Furthermore, many existing tools only have a specialized set of features, struggle with very long or high-resolution (≥ 4K) videos, or simply take too long to yield results. Existing fast algorithms are often severely limited with respect to the number of individuals that can be tracked simultaneously; for example xyTracker ([177]) allows for real-time tracking at 40Hz while accurately maintaining identities, and thus is suitable for closed-loop experimentation (experiments where stimulus presentation can depend on the real-time behaviors of the individuals, e.g. [13], [26], [18]), but has a limit of being able to track only 5 individuals simultaneously. ToxTrac ([183]), a software comparable to xyTracker in its set of features, is limited to 20 individuals and relatively low frame-rates (≤25fps). Others, while implementing a wide range of features and offering high-performance tracking, are costly and thus limited in access ([158]). Perhaps with the exception of proprietary software, one major problem at present is the severe fragmentation of features across the various software solutions. For example, experimentalists must typically construct work-flows from many individual tools: One tool might be responsible for estimating the animal’s positions, another for estimating their posture, another one for reconstructing visual fields (which in turn probably also estimates animal posture, but does not export it in any way) and one for keeping identities – correcting results of other tools post-hoc. It can take a very long time to make them all work effectively together, adding what is often considerable overhead to behavioral studies.

TRex, the software released with this publication (available at trex.run under an Open-Source license), has been designed to address these problems, and thus to provide a powerful, fast and easy to use tool that will be of use in a wide range of behavioral studies. It allows users to track moving objects/animals, as long as there is a way to separate them from the background (e.g. static backgrounds, custom masks, as discussed below). In addition to the positions of individuals, our software provides other per-individual metrics such as body shape and, if applicable, head-/tail-position. This is achieved using a basic posture analysis, which works out of the box for most organisms, and, if re-
quired, can be easily adapted for others. Posture information, which includes the body center-line, can be useful for detecting e.g. courtship displays and other behaviors that might not otherwise be obvious from mere positional data. Additionally, with the visual sense often being one of the most important modalities to consider in behavioral research, we include the capability for users to obtain a computational reconstruction of the visual fields of all individuals (Strandburg-Peshkin et al. 203, Rosenthal et al. 186). This not only reveals which individuals are visible from an individual’s point-of-view, as well as the distance to them, but also which parts of others’ bodies are visible.

Included in the software package is a task-specific tool, TGrabs, that is employed to pre-process existing video files and which allows users to record directly from cameras capable of live-streaming to a computer (with extensible support from generic webcams to high-end machine vision cameras). It supports most of the above-mentioned tracking features (positions, posture, visual field) and provides access to results immediately while continuing to record/process. This not only saves time, since tracking results are available immediately after the trial, but makes closed-loop support possible for large groups of individuals (≤ 128 individuals). TRex and TGrabs are written in C++ but, as part of our closed-loop support, we are providing a Python-based general scripting interface which can be fully customized by the user without the need to recompile or relaunch. This interface allows for compatibility with external programs (e.g. for closed-loop stimulus-presentation) and other custom extensions.

The fast tracking described above employs information about the kinematics of each organism in order to try to maintain their identities. This is very fast and useful in many scenarios, e.g. where general assessments about group properties (group centroid, alignment of individuals, density, etc.) are to be made. However, when making conclusions about individuals instead, maintaining identities perfectly throughout the video is a critical requirement. Every tracking method inevitably makes mistakes, which, for small groups of two or three individuals or short videos, can be corrected manually – at the expense of spending much more time on analysis, which rapidly becomes prohibitive as the number of individuals to be tracked increases. To make matters worse, when multiple individuals stay out of view of the camera for too long (such as if individuals move out of frame, under a shelter, or occlude one another) there is no way to know who is whom once they re-emerge. With no baseline truth available (e.g. using physical tags as in [6], [154]; or marker-less methods as in [173], [184], [177]), these mistakes can not be corrected and accumulate over time, until eventually all identities are fully shuffled. To solve this problem (and without the need to mark, or add physical tags to individuals), TRex can, at the cost of spending more time on analysis (and thus not during live-tracking), automatically learn the identity of up to approximately 100 unmarked individuals based on their vi-
ual appearance. This machine-learning based approach, herein termed *visual identification*, provides an independent source of information on the identity of individuals, which is used to detect and correct potential tracking mistakes without the need for human supervision.

In this paper, we evaluate the most important functions of our software in terms of speed and reliability using a wide range of experimental systems, including termites, fruit flies, locusts and multiple species of schooling fish (although we stress that our software is not limited to such species).

Specifically regarding the visual identification of unmarked individuals in groups, *idtracker.ai* is currently state-of-the-art, yielding high-accuracy (>99% in most cases) in maintaining consistent identity assignments across entire videos ([184]). Similarly to *TRex*, this is achieved by training an artificial neural network to visually differentiate between individuals, and using identity predictions from this network to avoid/correct tracking mistakes. Both approaches work without human supervision, and are limited to approximately 100 individuals. Given that *idtracker.ai* is the only currently available tool with visual identification for such large groups of individuals, and also because of the quality of results, we will use it as a benchmark for our visual identification system. Results will be compared in terms of both accuracy and computation speed, showing *TRex*’ ability to achieve the same high level of accuracy but typically at far higher speeds, and with a much reduced memory requirement.

*TRex* is platform-independent and runs on all major operating systems (Linux, Windows, macOS) and offers complete batch processing support, allowing users to efficiently process entire sets of videos without requiring human intervention. All parameters can be accessed either through settings files, from within the graphical user interface (or *GUI*), or using the command-line. The user interface supports off-site access using a built-in web-server (although it is recommended to only use this from within a secure VPN environment). Available parameters are explained in the documentation directly as part of the GUI and on an external website (see below). Results can be exported to independent data-containers (*NPZ*, or *CSV* for plain-text type data) for further analyses in software of the user’s choosing. We will not go into detail regarding the many GUI functions since albeit being of great utility to the researcher, they are only the means to easily apply the features presented herein. Some examples will be given in the main text and appendix, but a comprehensive collection of all of them, as well as detailed documentation, is available in the up-to-date online-documentation which can be found at [trex.run/docs](http://trex.run/docs).
**Figure 1:** Videos are typically processed in four main stages, illustrated here each with a list of prominent features. Some of them are accessible from both TRex and TGrabs, while others are software specific (as shown at the very top). (a) The video is either recorded directly with our software (TGrabs), or converted from a pre-recorded video file. Live-tracking enables users to perform closed-loop experiments, for which a virtual testing environment is provided. (b) Videos can be tracked and parameters adjusted with visual feedback. Various exploration and data presentation features are provided and customized data streams can be exported for use in external software. (c) After successful tracking, automatic visual identification can, optionally, be used to refine results. An artificial neural network is trained to recognize individuals, helping to automatically correct potential tracking mistakes. In the last stage, many graphical tools are available to users of TRex, a selection of which is listed in (d).

**Figure 1—Video 1.** This video shows an overview of the typical chronology of operations when using our software. Starting with the raw video, segmentation using TGrabs (Figure 1a) is the first and only step that is not optional. Tracking (Figure 1b) and posture estimation (both also available for live-tracking in TGrabs) are usually performed in that order, but can be partly parallelized (e.g. performing posture estimation in parallel for all individuals). Visual identification (Figure 1c) is only available in TRex due to relatively long processing times. All clips from this composite video have been recorded directly in TRex. [https://youtu.be/g9EO17FZwM0](https://youtu.be/g9EO17FZwM0)

### 1.2 Results

Our software package consists of two task-specific tools, TGrabs and TRex, with different specializations. TGrabs is primarily designed to connect to cameras and to be very fast. It employs the same program code as TRex to achieve real-time online tracking, such as could be employed for closed-loop experiments (the user can launch TGrabs from the opening dialog of TRex). However, its focus on speed comes at the cost of not having access to the rich graphical user interface or more sophisticated (and thus slower) processing steps, such as deep-learning based identification, that TRex provides.

TRex focuses on the more time-consuming tasks, as well as visual data exploration, re-tracking existing results – but sometimes it simply functions as an easier-to-use graphical interface for tracking and adjusting parameters. Together they provide a wide range of capabilities to the user and are often
used in sequence as part of the same workflow. Typically, such a sequence can be summarized in four stages (see also Figure 2 for a flow diagram):

1. **Segmentation** in TGrabs. When recording a video or converting a previously recorded file (e.g. MP4, .AVI, etc.), it is segmented into background and foreground-objects (blobs), the latter typically being the entities to be tracked. Results are saved to a custom, non-proprietary video format (PV) (Figure 1a).

2. **Tracking** the video, either directly in TGrabs, or in TRex after pre-processing, with access to customizable visualizations and the ability to change tracking parameters on-the-fly. Here, we will describe two types of data available within TRex, 2D posture- and visual-field estimation, as well as real-time applications of such data (Figure 1b).

3. **Automatic identity correction** (Figure 1c), a way of utilizing the power of a trained neural network to perform visual identification of individuals, is available in TRex only. This step may not be necessary in many cases, but it is the only way to guarantee consistent identities throughout the video. It is also the most processing-heavy (and thus usually the most time-consuming) step, as well as the only one involving machine learning. All previously collected posture- and other tracking-related data are utilized in this step, placing it late in a typical workflow.

4. Data visualization is a critical component of any research project, especially for unfamiliar datasets, but manually crafting one for every new experiment can be very time-consuming. Thus, TRex offers a universal, highly customizable, way to make all collected data available for interactive exploration (Figure 1d) – allowing users to change many display options and recording video clips for external playback. Tracking parameters can be adjusted on the fly (many with visual feedback) – important e.g. when preparing a closed-loop feedback with a new species or setup.

Below we assess the performance of our software regarding three properties that are most important when using it (or in fact any tracking software) in practice: (i) The time it takes to perform tracking (ii) the time it takes to perform automatic identity correction and (iii) the peak memory consumption when correcting identities (since this is where memory consumption is maximal), as well as (iv) the accuracy of the produced trajectories after visual identification.
Figure 2: An overview of the interconnection between TRex, TGrabs and their data in- and output formats, with titles on the left corresponding to the stages in Figure 1. Starting at the top of the figure, video is either streamed to TGrabs from a file or directly from a compatible camera. At this stage, preprocessed data are saved to a .pv file which can be read by TRex later on. Thanks to its integration with parts of the TRex code, TGrabs can also perform online tracking for limited numbers of individuals, and save results to a .results file (that can be opened by TRex) along with individual tracking data saved to numpy data-containers (.npz) or standard CSV files, which can be used for analysis in third-party applications. If required, videos recorded directly using TGrabs can also be streamed to a .mp4 video file which can be viewed in commonly available video players like VLC.

While accuracy is an important metric and specific to identification tasks, time and memory are typically of considerable practical importance for all tasks. For example, tracking-speed may be the difference between only being able to run a few trials or producing more reliable results with a much larger number of trials. In addition, tracking speed can make a major difference as the number of individuals increases. Furthermore, memory constraints can be extremely prohibitive making tracking over long video sequences and/or for a large number of individuals extremely time-consuming, or impossible, for the user.

In all of our tests we used a relatively modest computer system, which could be described as a mid-range consumer or gaming PC:

- Intel Core i9-7900X CPU
- NVIDIA GeForce 1080 Ti
- 64GB RAM
Table 1: A list of the videos used in this paper as part of the evaluation of TRex, along with the species of animals in the videos and their common names, as well as other video-specific properties. Videos are given an incremental ID, to make references more efficient in the following text, which are sorted by the number of individuals in the video. Individual quantities are given accurately, except for the videos with more than 100 where the exact number may be slightly more or less. These videos have been analysed using TRex’ dynamic analysis mode that supports unknown quantities of animals.

<table>
<thead>
<tr>
<th>ID</th>
<th>species</th>
<th>common</th>
<th># ind.</th>
<th>fps (Hz)</th>
<th>duration</th>
<th>size (px²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Leucaspius delineatus</td>
<td>sunbleak</td>
<td>1024</td>
<td>40</td>
<td>8min20s</td>
<td>3866 × 4048</td>
</tr>
<tr>
<td>1</td>
<td>Leucaspius delineatus</td>
<td>sunbleak</td>
<td>512</td>
<td>50</td>
<td>6min40s</td>
<td>3866 × 4140</td>
</tr>
<tr>
<td>2</td>
<td>Leucaspius delineatus</td>
<td>sunbleak</td>
<td>512</td>
<td>60</td>
<td>5min59s</td>
<td>3866 × 4048</td>
</tr>
<tr>
<td>3</td>
<td>Leucaspius delineatus</td>
<td>sunbleak</td>
<td>256</td>
<td>50</td>
<td>6min40s</td>
<td>3866 × 4140</td>
</tr>
<tr>
<td>4</td>
<td>Leucaspius delineatus</td>
<td>sunbleak</td>
<td>256</td>
<td>60</td>
<td>5min59s</td>
<td>3866 × 4048</td>
</tr>
<tr>
<td>5</td>
<td>Leucaspius delineatus</td>
<td>sunbleak</td>
<td>128</td>
<td>60</td>
<td>6min</td>
<td>3866 × 4048</td>
</tr>
<tr>
<td>6</td>
<td>Leucaspius delineatus</td>
<td>sunbleak</td>
<td>128</td>
<td>60</td>
<td>5min59s</td>
<td>3866 × 4048</td>
</tr>
<tr>
<td>7</td>
<td>Danio rerio</td>
<td>zebrafish</td>
<td>100</td>
<td>32</td>
<td>1min</td>
<td>3584 × 3500</td>
</tr>
<tr>
<td>8</td>
<td>Drosophila melanogaster</td>
<td>fruit-fly</td>
<td>59</td>
<td>51</td>
<td>10min</td>
<td>2306 × 2306</td>
</tr>
<tr>
<td>9</td>
<td>Schistocerca gregaria</td>
<td>locust</td>
<td>15</td>
<td>25</td>
<td>1h0min</td>
<td>1880 × 1881</td>
</tr>
<tr>
<td>10</td>
<td>Constrictotermes cybergaster</td>
<td>termite</td>
<td>10</td>
<td>100</td>
<td>10min5s</td>
<td>1920 × 1080</td>
</tr>
<tr>
<td>11</td>
<td>Danio rerio</td>
<td>zebrafish</td>
<td>10</td>
<td>32</td>
<td>10min10s</td>
<td>3712 × 3712</td>
</tr>
<tr>
<td>12</td>
<td>Danio rerio</td>
<td>zebrafish</td>
<td>10</td>
<td>32</td>
<td>10min3s</td>
<td>3712 × 3712</td>
</tr>
<tr>
<td>13</td>
<td>Danio rerio</td>
<td>zebrafish</td>
<td>10</td>
<td>32</td>
<td>10min3s</td>
<td>3712 × 3712</td>
</tr>
<tr>
<td>14</td>
<td>Poecilia reticulata</td>
<td>guppy</td>
<td>8</td>
<td>30</td>
<td>3h15min22s</td>
<td>3008 × 3008</td>
</tr>
<tr>
<td>15</td>
<td>Poecilia reticulata</td>
<td>guppy</td>
<td>8</td>
<td>25</td>
<td>1h12min</td>
<td>3008 × 3008</td>
</tr>
<tr>
<td>16</td>
<td>Poecilia reticulata</td>
<td>guppy</td>
<td>8</td>
<td>35</td>
<td>3h18min13s</td>
<td>3008 × 3008</td>
</tr>
<tr>
<td>17</td>
<td>Poecilia reticulata</td>
<td>guppy</td>
<td>1</td>
<td>140</td>
<td>1h9min32s</td>
<td>1312 × 1312</td>
</tr>
</tbody>
</table>

Table 1—source data 1. Videos 7 and 8, as well as 13-11, are available as part of the original idtracker paper ([173]). Many of the videos are part of yet unpublished data: Guppy videos have been recorded by A. Albi, videos with sunbleak (Leucaspius delineatus) have been recorded by D. Bath. The termite video has been kindly provided by H. Hugo and the locust video by F. Oberhauser. Due to the size of some of these videos (>150GB per video), they have to be made available upon specific request. Raw versions of these videos (some trimmed), as well as full preprocessed versions, are available as part of the dataset published alongside this paper [221].

- NVMe PCIe x4 hard-drive
- Debian bullseye (debian.org)

As can be seen in the following sections (memory consumption, processing speeds, etc.) using a high-end system is not necessary to run TRex and, anecdotally, we did not observe noticeable improvements when using a solid state drive versus a normal hard drive. A video card (presently an NVIDIA card due to the requirements of TensorFlow) is recommended for tasks involving visual identification as such computations will take much longer without it – however, it is not required. We decided to
employ this system due to having a relatively cheap, compatible graphics card, as well as to ensure that we have an easy way to produce direct comparisons with idtracker.ai – which according to their website requires large amounts of RAM (32-128GB, idtrackerai online documentation) and a fast solid-state drive.

Table 1 shows the entire set of videos used in this paper, which have been obtained from multiple sources (credited under the table) and span a wide range of different organisms, demonstrating TRex’ ability to track anything as long as it moves occasionally. Videos involving a large number (>100) of individuals are all the same species of fish since these were the only organisms we had available in such quantities. However, this is not to say that only fish could be tracked efficiently in these quantities. We used the full dataset with up to 1024 individuals in one video (video 0) to evaluate raw tracking speed without visual identification and identity corrections (next sub-section). However, since such numbers of individuals exceed the capacity of the neural network used for automatic identity corrections (compare also [184] who used a similar network), we only used a subset of these videos (videos 7 through 16) to look specifically into the quality of our visual identification in terms of keeping identities and its memory consumption.

1.2.1 Tracking: Speed and Accuracy

In evaluating the Tracking portion of TRex, the main focus lies with processing speed, while accuracy in terms of keeping identities is of secondary importance. Tracking is required in all other parts of the software, making it an attractive target for extensive optimization. Especially with regards to closed-loop, and live-tracking situations, there may be no room even to lose a millisecond between frames and thus risk dropping frames. We therefore designed TRex to support the simultaneous tracking of many (≥256) individuals quickly and achieve reasonable accuracy for up to 100 individuals – which are the two suppositions we will investigate in the following.

Trials were run without posture/visual-field estimation enabled, where tracking generally, and consistently, reaches speeds faster than real-time (processing times of 1.5-40% of the video duration, 25-100Hz) even for a relatively large number of individuals (77-94.77% for up to 256 individuals, see A1). Videos with more individuals (>500) were still tracked within reasonable time of 235% to 358% of the video duration. As would be expected from these results, we found that combining tracking and recording in a single step generally leads to higher processing speeds. The only situation where this was not the case was a video with 1024 individuals, which suggests that live-tracking (in TGrabs) handles cases with many individuals slightly worse than offline tracking (in TRex). Otherwise, 5% to
35% shorter total processing times were measured (14.55% on average, see A4), compared to running TGrabs separately and then tracking in TRex. These percentage differences, in most cases, reflect the ratio between the video duration and the time it takes to track it, suggesting that most time is spent – by far – on the conversion of videos. This additional cost can be avoided in practice when using TGrabs to record videos, by directly writing to a custom format recognized by TRex, and/or using its live-tracking ability to export tracking data immediately after the recording is stopped.

We also investigated trials that were run with posture estimation enabled and we found that real-time speed could be achieved for videos with ≤ 128 individuals (see column "tracking" in A4). Tracking speed, when posture estimation is enabled, depends more strongly on the size of individuals in the image.

Generally, tracking software becomes slower as the number of individuals to be tracked increases, as a result of an exponentially growing number of combinations to consider during matching. TRex uses a novel tree-based algorithm by default (see Tracking), but circumvents problematic situations by falling back on using the Hungarian method (also known as the Kuhn–Munkres algorithm, [122]) when necessary. Comparing our mixed approach (see Tracking) to purely using the Hungarian method shows that, while both perform similarly for few individuals, the Hungarian method is easily outperformed by our algorithm for larger groups of individuals (as can be seen in A7). This might be due to custom optimizations regarding local cliques of individuals, whereby we ignore objects that are too far away, and also as a result of our optimized pre-sorting. The Hungarian method has the advantage of not leading to combinatorial explosions in some situations – and thus has a lower maximum complexity while proving to be less optimal in the average case. For further details, see the appendix: Matching an object to an object in the next frame.

In addition to speed, we also tested the accuracy of our tracking method, with regards to the consistency of identity assignments, comparing its results to the manually reviewed data (the methodology of which is described in the next section). In order to avoid counting follow-up errors as "new" errors, we divided each trajectory in the uncorrected data into "uninterrupted" segments of frames, instead of simply comparing whole trajectories. A segment is interrupted when an individual is lost (for any of the reasons given in Preparing Tracking-Data) and starts again when it is reassigned to another object later on. We term these (re-)assignments decisions here. Each segment of every individual can be uniquely assigned to a similar/identical segment in the baseline data and its identity. Following one trajectory in the uncorrected data, we can detect these wrong decisions by checking whether the baseline identity associated with one segment of that trajectory changes in the next. We found that roughly 80% of such decisions made by the tree-based matching were correct, even with relatively
high numbers of individuals (100). For trajectories where no manually reviewed data were available, we used automatically corrected trajectories as a base for our comparison – we evaluate the accuracy of these automatically corrected trajectories in the following section. Even though we did not investigate accuracy in situations with more than 100 individuals, we suspect similar results since the property with the strongest influence on tracking accuracy – individual density – is limited physically and most of the investigated species school tightly in either case.

Table 2–source data 1. This file contains all X and Y positions for each trial and each software combined into one very large table. This data is also available in different formats in [22].

Table 2–source data 2. Assignments between identities from multiple solutions, as calculated by a bipartite-graph matching algorithm. For each permutation of trials from TReX and idtracker.ai for the same video, the algorithm sought to match the trajectories of the same physical individuals in both trials with each other by finding the ones with the smallest mean euclidean distance per frame between them. Available at http://dx.doi.org/10.17617/3.4y, as T2_source_data.zip.

Table 2: Evaluating comparability of the automatic visual identification between idtracker.ai and TReX. Columns show various video properties, as well as the associated uniqueness score (see Box 1.1) and a similarity metric. Similarity (% similar individuals) is calculated based on comparing the positions for each identity exported by both tools, choosing the closest matches overall and counting the ones that are differently assigned per frame. An individual is classified as “wrong” in that frame, if the euclidean distance between the matched solutions from idtracker.ai and TReX exceeds 1% of the video width. The column “% similar individuals” shows percentage values, where a value of 99% would indicate that, on average, 1% of the individuals are assigned differently. To demonstrate how uniqueness corresponds to the quality of results, the last column shows the average uniqueness achieved across trials.

1.2.2 Visual Identification: Accuracy

Since the goal of using visual identification is to generate consistent identity assignments, we evaluated the accuracy of our method in this regard. As a benchmark, we compare it to manually reviewed datasets as well as results from idtracker.ai for the same set of videos (where possible). In order to validate trajectories exported by either software, we manually reviewed multiple videos with the help
from a tool within TRex that allows to view each crossing and correct possible mistakes in-place. Assignments were deemed incorrect, and subsequently corrected by the reviewer, if the centroid of a given individual was not contained within the object it was assigned to (e.g. the individual was not part of the correct object). Double assignments per object are impossible due to the nature of the tracking method. Individuals were also forcibly assigned to the correct objects in case they were visible but not detected by the tracking algorithm. After manual corrections had been applied, “clean” trajectories were exported – providing a per-frame baseline truth for the respective videos. A complete table of reviewed videos, and the percentage of reviewed frames per video, can be found in Table 3.

For longer videos (>1h) we relied entirely on a comparison between results from idtracker.ai and TRex. Their paper (184) suggests a very high accuracy of over 99.9% correctly identified individual images for most videos, which should suffice for most relevant applications and provide a good baseline truth. As long as both tools produce sufficiently similar trajectories, we therefore know they have found the correct solution.

<table>
<thead>
<tr>
<th>video</th>
<th># ind.</th>
<th>reviewed (%)</th>
<th>of that interpolated (%)</th>
<th>TRex</th>
<th>idtracker.ai</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>100</td>
<td>100.0</td>
<td>0.23</td>
<td>99.97 ± 0.013</td>
<td>98.95 ± 0.146</td>
</tr>
<tr>
<td>8</td>
<td>59</td>
<td>100.0</td>
<td>0.15</td>
<td>99.68 ± 0.533</td>
<td>99.94 ± 0.0</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>22.2</td>
<td>8.44</td>
<td>95.12 ± 6.077</td>
<td>N/A</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>100.0</td>
<td>1.21</td>
<td>99.7 ± 0.088</td>
<td>N/A</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>100.0</td>
<td>0.27</td>
<td>99.98 ± 0.0</td>
<td>99.96 ± 0.0</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>100.0</td>
<td>0.59</td>
<td>99.94 ± 0.006</td>
<td>99.63 ± 0.0</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>100.0</td>
<td>0.5</td>
<td>99.89 ± 0.009</td>
<td>99.34 ± 0.002</td>
</tr>
</tbody>
</table>

Table 3—source data 1. A table of positions for each individual of each manually approved and corrected trial.

Table 3: Results of the human validation for a subset of videos. Validation was performed by going through all problematic situations (e.g. individuals lost and correcting mistakes manually, creating a fully corrected dataset for the given videos. This dataset may still have missing frames for some individuals, if they could not be detected in certain frames (as indicated by “of that interpolated”). This was usually a very low percentage of all frames, except for video 9, where individuals tended to rest on top of each other – and were thus not tracked – for extended periods of time. This baseline dataset was compared to all other results obtained using the automatic visual identification by TRex (N = 5) and idtracker.ai (N = 3) to estimate correctness. We were not able to track videos 9 and 10 with idtracker.ai, which is why correctness values are not available.

A direct comparison between TRex and idtracker.ai was not possible for videos 9 and 10, where idtracker.ai frequently exceeded hardware memory-limits and caused the application to be terminated, or did not produce usable results within multiple days of run-time. However, we were able to successfully analyse these videos with TRex and evaluate its performance by comparing to manu-
ally reviewed trajectories (see below in Visual Identification: Accuracy). Due to the stochastic nature of machine learning, and thus the inherent possibility of obtaining different results in each run, as well as other potential factors influencing processing time and memory consumption, both TRex and idtracker.ai have been executed repeatedly (5x TRex, 3x idtracker.ai).

The trajectories exported by both idtracker.ai and TRex were very similar throughout (see Table 2). While occasional disagreements happened, similarity scores were higher than 98% in all and higher than 99% in most cases (i.e. less than 1% of individuals have been differently assigned in each frame on average). Most difficulties that did occur were, after manual review, attributable to situations where multiple individuals cross over excessively within a short time-span. In each case that has been manually reviewed, identities switched back to the correct individuals – even after temporary disagreement. We found that both solutions occasionally experienced these same problems, which often occur when individuals repeatedly come in and out of view in quick succession (e.g. overlapping with other individuals). Disagreements were expected for videos with many such situations due to the way both algorithms deal differently with them: idtracker.ai assigns identities only based on the network output. In many cases, individuals continue to partly overlap even while already being tracked, which results in visual artifacts and can lead to unstable predictions by the network and causing idtracker.ai’s approach to fail. Comparing results from both idtracker.ai and TRex to manually reviewed data (see Table 3) shows that both solutions consistently provide high accuracy results of above 99.5% for most videos, but that TRex is slightly improved in all cases while also having a better overall frame coverage per individual (99.65% versus idtracker.ai’s 97.93%, where 100% would mean that all individuals are tracked in every frame; not shown). This suggests that the splitting algorithm (see appendix, Algorithm for splitting touching individuals) is working to TRex’ advantage here.

Additionally, while TRex could successfully track individuals in all videos without tags, we were interested to see the effect of tags (in this case QR tags attached to locusts, see Figure 3a) on network training. In Figure 3 we visualise differences in network activation, depending on the visual features available for the network to learn from, which are different between species (or due to physically added tags, as mentioned above). The "hot" regions indicate larger between-class differences for that specific pixel (values are the result of activation in the last convolutional layer of the trained network, see figure legend). Differences are computed separately within each group and are not directly comparable between trials/species in value. However, the distribution of values – reflecting the network’s reactivity to specific parts of the image – is. Results show that the most apparent differences are found for the stationary parts of the body (not in absolute terms, but following normalization,
Locusts from video 9 with 15 tagged individuals (N: 5101, 7942, 9974) – the only video with physical tags. The network activates more strongly in regions close to the tag, as well as the bottom right corner.

Guppies from video 15 (N: 46378, 34733, 34745). Activations are less focussed and less consistent across individuals.

Flies from video 8 (N: 993, 1986, 993). Activations are not similar between individuals and show various “hotspots” across the entire body.

Termites from video 10 (N: 27097, 31135, 22746). Here, the connections between body-segments show strong activations – in contrast to very weak ones in other parts of the body.

Figure 3–source data 1. Code, as well as images/weights needed to produce this figure (see README).
Here, skews results. The maximum, however, is more reliable since it marks the memory that is necessary to run the system.

Table 4: idtracker.ai

<table>
<thead>
<tr>
<th>video</th>
<th>#ind.</th>
<th>length</th>
<th>max.consec.</th>
<th>TRex memory (GB)</th>
<th>idtracker.ai memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>10</td>
<td>10min</td>
<td>26.03s</td>
<td>$\phi 4.88 \pm 0.23$, max 6.31</td>
<td>$\phi 8.23 \pm 0.99$, max 28.85</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>10min</td>
<td>36.94s</td>
<td>$\phi 4.27 \pm 0.12$, max 4.79</td>
<td>$\phi 7.83 \pm 1.05$, max 29.43</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>10min</td>
<td>28.75s</td>
<td>$\phi 4.37 \pm 0.32$, max 5.49</td>
<td>$\phi 6.53 \pm 4.29$, max 29.32</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>1min</td>
<td>5.97s</td>
<td>$\phi 10.31 \pm 0.53$, max 13.45</td>
<td>$\phi 15.27 \pm 1.05$, max 24.39</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>72min</td>
<td>79.4s</td>
<td>$\phi 6.94 \pm 0.27$, max 10.71</td>
<td>N/A</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10min</td>
<td>1.91s</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>60min</td>
<td>7.64s</td>
<td>$\phi 13.81 \pm 0.53$, max 16.99</td>
<td>N/A</td>
</tr>
<tr>
<td>8</td>
<td>59</td>
<td>10min</td>
<td>102.35s</td>
<td>$\phi 12.4 \pm 0.56$, max 17.41</td>
<td>$\phi 35.3 \pm 0.92$, max 50.26</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>195min</td>
<td>145.77s</td>
<td>$\phi 12.44 \pm 0.8$, max 21.99</td>
<td>$\phi 35.08 \pm 4.08$, max 98.04</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>198min</td>
<td>322.57s</td>
<td>$\phi 16.15 \pm 1.6$, max 28.62</td>
<td>$\phi 49.24 \pm 8.21$, max 115.37</td>
</tr>
</tbody>
</table>

Table 4: Both TRex and idtracker.ai analysed the same set of videos, while continuously logging their memory consumption using an external tool. Rows have been sorted by video_length + #individuals, which seems to be a good predictor for the memory consumption of both solutions. idtracker.ai has mixed mean values, which, at low individual densities are similar to TRex’ results. Mean values can be misleading here, since more time spent in low-memory states skews results. The maximum, however, is more reliable since it marks the memory that is necessary to run the system. Here, idtracker.ai clocks in at significantly higher values (almost always more than double) than TRex.

as shown in Figure 8c), which makes sense seeing as this part (i) is the easiest to learn due to it being in exactly the same position every time, (ii) larger individuals stretch further into the corners of a cropped image, making the bottom right of each image a source of valuable information (especially in Figure 3a/Figure 3b) and (iii) details that often occur in the head-region (like distance between the eyes) which can also play a role here. "Hot" regions in the bottom right corner of the activation images (e.g. in Figure 3d) suggest that also pixels are reacted to which are explicitly not part of the individual itself but of other individuals – likely this corresponds to the network making use of size/shape differences between them.

As would be expected, distinct patterns can be recognized in the resulting activations after training as soon as physical tags are attached to individuals (as in Figure 3a). While other parts of the image are still heavily activated (probably to benefit from size/shape differences between individuals), tags are always at least a large part of where activations concentrate. The network seemingly makes use of the additional information provided by the experimenter, where that has occurred. This suggests that, while definitely not being necessary, adding tags probably does not worsen, and likely may even improve, training accuracy, for difficult cases allowing networks to exploit any source of inter-individual variation.
1.2.3 Visual Identification: Memory Consumption

In order to generate comparable results between both tested software solutions, the same external script has been used to measure shared, private and swap memory of idtracker.ai and TRex, respectively. There are a number of ways with which to determine the memory usage of a process. For automation purposes we decided to use a tool called syrupy, which can start and save information about a specified command automatically. We modified it slightly, so we could obtain more accurate measurements for Swap, Shared and Private separately, using ps_mem.

As expected, differences in memory consumption are especially prominent for long videos (4-7x lower maximum memory), and for videos with many individuals (2-3x lower). Since we already experienced significant problems tracking a long video (>3h) of only 8 individuals with idtracker.ai, we did not attempt to further study its behavior in long videos with many individuals. However, we would expect idtracker.ai’s memory usage to increase even more rapidly than is visible in Figure 4

![Figure 4: The maximum memory by TRex and idtracker.ai when tracking videos from a subset of all videos (the same videos as in Table 2). Results are plotted as a function of video length (min) multiplied by the number of individuals. We have to emphasize here that, for the videos in the upper length regions of multiple hours (16, 14), we had to set idtracker.ai to store segmentation information on disk – as opposed to in RAM. This uses less memory, but is also slower. For the video with flies we tried out both and also settled for on-disk, since otherwise the system ran out of memory. Even then, the curve still accelerates much faster for idtracker.ai, ultimately leading to problems with most computer systems. To minimize the impact that hardware compatibility has on research, we implemented switches limiting memory usage while always trying to maximize performance given the available data. TRex can be used on modern laptops and normal consumer hardware at slightly lower speeds, but without any fatal issues.

Figure 4—source data 1. Each data-point from Figure 4 as plotted, indexed by video and software used.](image-url)
Figure 5: Convergence behavior of the network training for three different normalization methods. This shows the maximum achievable validation accuracy after 100 epochs for 100 individuals (video 7), when sub-sampling the number of examples per individual. Tests were performed using a manually corrected training dataset to generate the images in three different ways, using the same, independent script (see Figure 8): Using no normalization (blue), using normalization based on image moments (green, similar to idtracker.ai), and using posture information (red, as in TRex). Higher numbers of samples per individual result in higher maximum accuracy overall, but – unlike the other methods – posture-normalized runs already reach an accuracy above the 90% mark for ≥ 75 samples. This property can help significantly in situations with more crossings, when longer global segments are harder to find.

Figure 5–source data 1. Raw data-points as plotted in Figure 5.

since it retains a lot of image data (segmentation/pixels) in memory and we already had to "allow" it to relay to hard-disk in our efforts to make it work for Videos 8, 14 and 16 (which slows down analysis). The maximum memory consumption across all trials was on average 5.01±2.54 times higher in idtracker.ai, ranging from 1.81 to 10.85 times the maximum memory consumption of TRex for the same video.

Overall memory consumption for TRex also contains posture data, which contributes a lot to RAM usage. Especially with longer videos, disabling posture can lower the hardware needs for running our software. If posture is to be retained, the user can still (more slightly) reduce memory requirements by changing the outline re-sampling scale (1 by default), which adjusts the outline resolution between sub- and super-pixel accuracy. While analysis will be faster – and memory consumption lower – when posture is disabled (only limited by the matching algorithm, see A.7), users of the visual identification might experience a decrease in training accuracy or speed (see Figure 5).
idtracker.ai

Table 5: Preprocessed log files (see also notebooks . zip in [221]) in a table format. The total processing time (s) of each trial is indexed by video and software used – TGrabs for conversion and TRex and idtracker . ai for visual identification. This data is also used in A4.

<table>
<thead>
<tr>
<th>video</th>
<th># ind.</th>
<th>length</th>
<th>sample</th>
<th>TGrabs (min)</th>
<th>TRex (min)</th>
<th>ours (min)</th>
<th>idtracker . ai (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>100</td>
<td>1min</td>
<td>1.61s</td>
<td>2.03 ± 0.02</td>
<td>74.62 ± 6.75</td>
<td>76.65</td>
<td>392.22 ± 119.43</td>
</tr>
<tr>
<td>8</td>
<td>59</td>
<td>10min</td>
<td>19.46s</td>
<td>9.28 ± 0.08</td>
<td>96.7 ± 4.45</td>
<td>105.98</td>
<td>4953.82 ± 115.92</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>60min</td>
<td>33.81s</td>
<td>13.17 ± 0.12</td>
<td>101.5 ± 1.85</td>
<td>114.67</td>
<td>N/A</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>10min</td>
<td>12.31s</td>
<td>8.8 ± 0.12</td>
<td>21.42 ± 2.45</td>
<td>30.22</td>
<td>127.43 ± 57.02</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>10min</td>
<td>10.0s</td>
<td>8.65 ± 0.07</td>
<td>23.37 ± 3.83</td>
<td>32.02</td>
<td>82.28 ± 3.83</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>10min</td>
<td>36.91s</td>
<td>8.65 ± 0.07</td>
<td>12.47 ± 1.27</td>
<td>21.12</td>
<td>79.42 ± 4.52</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10min</td>
<td>16.22s</td>
<td>4.43 ± 0.05</td>
<td>35.05 ± 1.45</td>
<td>39.48</td>
<td>N/A</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>195min</td>
<td>67.97s</td>
<td>109.97 ± 2.05</td>
<td>70.48 ± 3.67</td>
<td>180.45</td>
<td>707.0 ± 27.55</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>72min</td>
<td>79.36s</td>
<td>32.1 ± 0.42</td>
<td>30.77 ± 6.28</td>
<td>62.87</td>
<td>291.42 ± 16.83</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>198min</td>
<td>134.07s</td>
<td>133.1 ± 2.28</td>
<td>68.85 ± 13.12</td>
<td>201.95</td>
<td>1493.83 ± 27.75</td>
</tr>
</tbody>
</table>

1.2.4 Visual Identification: Processing Time

Automatically correcting the trajectories (to produce consistent identity assignments) means that additional time is spent on the training and application of a network, specifically for the video in question. Visual identification builds on some of the other methods described in this paper (tracking and posture estimation), naturally making it by far the most complex and time-consuming process in TRex – we thus evaluated how much time is spent on the entire sequence of all required processes. For each run of TRex and idtracker . ai, we saved precise timing information from start to finish. Since idtracker . ai reads videos directly and preprocesses them again each run, we used the same starting conditions with our software for a direct comparison:

A trial starts by converting/preprocessing a video in TGrabs and then immediately opening it in TRex, where automatic identity corrections were applied. TRex terminated automatically after satisfying a correctness criterion (high uniqueness value) according to equation (1.1). It then exported trajectories, as well as validation data (similar to idtracker . ai), concluding the trial. The sum of time...
spent within TGrabs and TRex gives the total amount of time for that trial. For the purpose of this test it would not have been fair to compare only TRex processing times to idtracker.ai, but it is important to emphasize that conversion could be skipped entirely by using TGrabs to record videos directly from a camera instead of opening an existing video file.

In Table 5 we can see that video length and processing times (in TRex) did not correlate directly. Indeed, a 1 minute video (video 8) took significantly longer than one that was 60 minutes long (video 15). The reason for this, initially counterintuitive, result is that the process of learning identities requires sufficiently long video sequences: longer samples have a higher likelihood of capturing more of the total possible intra-individual variance which helps the algorithm to more comprehensively represent each individual’s appearance. Longer videos naturally provide more material for the algorithm to choose from and, simply due to their length, have a higher probability of containing at least one higher-quality segment that allows higher uniqueness-regimes to be reached more quickly (see Guiding the Training Process and Stopping-criteria). Thus, it is important to use sufficiently long video sequences for visual identification, and longer sequences can lead to better results – both in terms of quality and processing time.

Compared to idtracker.ai, TRex (conversion + visual identification) shows both considerably lower computation times (2.57× to 46.74× faster for the same video), as well as lower variance in the timings (79% lower for the same video on average).

1.3 Discussion

We have designed TRex to be a versatile and fast program that can enable researchers to track animals (and other mobile objects) in a wide range of situations. It maintains identities of up to 100 un-tagged individuals and produces corrected tracks, along with posture estimation, visual-field reconstruction, and other features that enable the quantitative study of animal behavior. Even videos that can not be tracked by other solutions, such as videos with over 500 animals, can now be tracked within the same day of recording.

While all options are available from the command-line and a screen is not required, TRex offers a rich, yet straight-forward to use, interface to local as well as remote users. Accompanied by the integrated documentation for all parameters, each stating purpose, type and value ranges, as well as a comprehensive online documentation, new users are provided with all the information required for a quick adoption of our software. Especially to the benefit of new users, we evaluated the parameter space using videos of diverse species (fish, termites, locusts) and determined which parameters work
best in most use-cases to set their default values.

Figure 6: An overview of TRex’s main interface, which is part of the documentation at trex.run/docs. Interface elements are sorted into categories in the four corners of the screen (labelled here in black). The omni-box on the bottom left corner allows users to change parameters on-the-fly, helped by a live auto-completion and documentation for all settings. Only some of the many available features are displayed here. Generally, interface elements can be toggled on or off using the bottom-left display options or moved out of the way with the cursor. Users can customize the tinting of objects (e.g. sourcing it from their speed) to generate interesting effect and can be recorded for use in presentations. Additionally, all exportable metrics (such as border-distance, size, x/y, etc.) can also be shown as an animated graph for a number of selected objects. Keyboard shortcuts are available for select features such as loading, saving, and terminating the program. Remote access is supported and offers the same graphical user interface, e.g. in case the software is executed without an application window (for batch processing purposes).

The interface is structured into groups (see Figure 6), categorized by the typical use-case:

1. The main menu, containing options for loading/saving, options for the timeline and reanalysis of parts of the video
2. Timeline and current video playback information
3. Information about the selected individual
4. Display options and an interactive "omni-box" for viewing and changing parameters
5. General status information about TRex and the Python integration
The tracking accuracy of TRex is at the state-of-the-art while typically being 2.57× to 46.74× faster than comparable software and having lower hardware requirements – especially RAM. In addition to visual identification and tracking, it provides a rich assortment of additional data, including body posture, visual fields, and other kinematic as well as group-related information (such as derivatives of position, border and mean neighbor distance, group compactness, etc.); even in live-tracking and closed-loop situations.

Raw tracking speeds (without visual identification) still achieved roughly 80% accuracy per decision (as compared to >99% with visual identification). We have found that real-time performance can be achieved, even on relatively modest hardware, for all numbers of individuals ≤256 without posture estimation (≤128 with posture estimation). More than 256 individuals can be tracked as well, remarkably still delivering frame-rates at about 10-25 frames per second using the same settings.

Not only does the increased processing-speeds benefit researchers, but the contributions we provide to data exploration should not be underestimated as well – merely making data more easily accessible right out-of-the-box, such as visual fields and live-heatmaps, has the potential to reveal features of group- and individual behaviour which have not been visible before. TRex makes information on multiple timescales of events available simultaneously, and sometimes this is the only way to detect interesting properties (e.g. trail formation in termites).

Since the software is already actively used within the Max Planck Institute of Animal Behavior, reported issues have been taken into consideration during development. However, certain theoretical, as well as practically observed, limitations remain:

- Posture: While almost all shapes can be detected correctly (by adjusting parameters), some shapes – especially round shapes – are hard to interpret in terms of "tail" or "head". This means that only the other image alignment method (moments) can be used. However, it does introduce some limitations e.g. calculating visual fields is impossible.
- Tracking: Predictions, if the wrong direction is assumed, might go really far away from where the object is. Objects are then "lost" for a fixed amount of time (parameter). This can be "fixed" by shortening this time-period, though this leads to different problems when the software does not wait long enough for individuals to reappear.
- General: Barely visible individuals have to be tracked with the help of deep learning (e.g. using [32]) and a custom-made mask per video frame, prepared in an external program of the users choosing.
- Visual identification: All individuals have to be visible and separate at the same time, at least
once, for identification to work at all. Visual identification, e.g. with very high densities of individuals, can thus be very difficult. This is a hard restriction to any software since finding consecutive global segments is the underlying principle for the successful recognition of individuals.

We will continue updating the software, increasingly addressing the above issues (and likely others), as well as potentially adding new features. During development we noticed a couple of areas where improvements could be made, both theoretical and practical in nature. Specifically, incremental improvements in analysis speed could be made regarding visual identification by using the trained network more sporadically – e.g. it is not necessary to predict every image of very long consecutive segments, since, even with fewer samples, prediction values are likely to converge to a certain value early on. A likely more potent change would be an improved "uniqueness" algorithm, which, during the accumulation phase, is better at predicting which consecutive segment will improve training results the most. This could be done, for example, by taking into account the variation between images of the same individual. Other planned extensions include:

- (Feature): We want to have a more general interface available to users, so they can create their own plugins. Working with the data in live-mode, while applying their own filters. As well as specifically being able to write a plugin that can detect different species/annotate them in the video.
- (Crossing solver): Additional method optimized for splitting overlapping, solid-color objects. The current method, simply using a threshold, is effective for many species but often produces large holes when splitting objects consisting of largely the same color.

To obtain the most up-to-date version of TRex, please download it at trex.run or update your existing installation according to our instructions listed on trex.run/docs/install.html.

1.4 Materials & Methods

In the following sections we describe the methods implemented in TRex and TGrabs, as well as their most important features in a typical order of operations (see Figure 2 for a flow diagram), starting out with a raw video. We will then describe how trajectories are obtained and end with the most technically involved features.
1.4.1 Segmentation

When an image is first received from a camera (or a video file), the objects of interest potentially present in the frame must be found and cropped out. Several technologies are available to separate the foreground from the background (segmentation). Various machine learning algorithms are frequently used to great effect, even for the most complex environments (Hughey et al. 103, Robie et al. 182, Francisco et al. 70). These more advanced approaches are typically beneficial for the analysis of field-data or organisms that are very hard to see in video (e.g. very transparent or low contrast objects/animals in the scene). In these situations, where integrated methods might not suffice, it is possible to segment objects from the background using external, e.g. deep-learning based, tools (see next paragraph). However, for most laboratory experiments, simpler (and also much faster), classical image-processing methods yield satisfactory results. Thus, we provide as a generically-useful capability background-subtraction, which is the default method by which objects are segmented. This can be used immediately in experiments where the background is relatively static. Backgrounds are generated automatically by uniformly sampling images from the source video(s) – different modes are available (min/max, mode and mean) for the user to choose from. More advanced image-processing techniques like luminance equalization (which is useful when lighting varies between images), image undistortion, and brightness/contrast adjustments are available in TGrabs and can enhance segmentation results – but come at the cost of slightly increased processing time. Importantly, since many behavioral studies rely on ≥ 4K resolution videos, we heavily utilize the GPU (if available) to speed up most of the image-processing, allowing TReX to scale well with increasing image resolution.

TGrabs can generally find any object in the video stream, and subsequently pass it on to the tracking algorithm (next section), as long as either (i) the background is relatively static while the objects move at least occasionally, (ii) the objects/animals of interest have enough contrast to the background or (iii) the user provides an additional binary mask per frame which is used to separate the objects of interest from the background, the typical means of doing this being by deep-learning based segmentation (e.g. Caelles et al. 32). These masks are expected to be in a video-format themselves and correspond 1:1 in length and dimensions to the video that is to be analyzed. They are expected to be binary, marking individuals in white and background in black. Of course, these binary videos could be used on their own, but would not retain grey-scale information of the objects. There are a lot of possible applications where this could be useful; but generally, whenever individuals are really hard to detect visually and need to be recognized by a different software (e.g. a machine-learning-based segmentation like Maninis et al. 141). Individual frames can then be connected using our software as a
second step.

The detected objects are saved to a custom non-proprietary compressed file format (Preprocessed Video or PV, see appendix The PV file format), that stores only the most essential information from the original video stream: the objects and their pixel positions and values. This format is optimized for quick random index access by the tracking algorithm (see next section) and stores other meta-information (like frame timings) utilized during playback or analysis. When recording videos directly from a camera, they can also be streamed to an additional and independent MP4 container format (plus information establishing the mapping between PV and MP4 video frames).

1.4.2 Tracking

Once animals (or, more generally, termed "objects" henceforth) have been successfully segmented from the background, we can either use the live-tracking feature in TGrabs or open a pre-processed file in TRex, to generate the trajectories of these objects. This process uses information regarding an object’s movement (i.e. its kinematics) to follow it across frames, estimating future positions based on previous velocity and angular speed. It will be referred to as "tracking" in the following text, and is a required step in all workflows.

Note that this approach alone is very fast, but, as will be shown, is subject to error with respect to maintaining individual identities. If that is required, there is a further step, outlined in Automatic Visual Identification Based on Machine Learning below, which can be applied at the cost of processing speed. First, however, we will discuss the general basis of tracking, which is common to approaches that do, and do not, require identities to be maintained with high-fidelity. Tracking can occur for two distinct categories, which are handled slightly differently by our software:

1. there is a known number of objects
2. there is an unknown number of objects

The first case assumes that the number of tracked objects in a frame cannot exceed a certain expected number of objects (calculated automatically, or set by the user). This allows the algorithm to make stronger assumptions, for example regarding noise, where otherwise "valid" objects (conforming to size expectations) are ignored due to their positioning in the scene (e.g. too far away from previously lost individuals). In the second case, new objects may be generated until all viable objects in a frame are assigned. While being more susceptible to noise, this is useful for tracking a large number of objects, where counting objects may not be possible, or where there is a highly variable number of objects to be tracked.
For a given video, our algorithm processes every frame sequentially, extending existing trajectories (if possible) for each of the objects found in the current frame. Every object can only be assigned to one trajectory, but some objects may not be assigned to any trajectory (e.g. in case the number of objects exceeds the allowed number of individuals) and some trajectories might not be assigned to any object (e.g. while objects are out of view). To estimate object identities across frames we use an approach akin to the popular Kalman filter \cite{114} which makes predictions based on multiple noisy data streams (here, positional history and posture information). In the initial frame, objects are simply assigned from top-left to bottom-right. In all other frames, assignments are made based on probabilities (see appendix Matching an object to an object in the next frame) calculated for every combination of object and trajectory. These probabilities represent the degree to which the program believes that "it makes sense" to extend an existing trajectory with an object in the current frame, given its position and speed. Our tracking algorithm only considers assignments with probabilities larger than a certain threshold, generally constrained to a certain proximity around an object assigned in the previous frame.

Matching a set of objects in one frame with a set of objects in the next frame is representative of a typical assignment problem, which can be solved in polynomial time (e.g. using the Hungarian method Kuhn \cite{122}). However, we found that, in practice, the computational complexity of the Hungarian method can constrain analysis speed to such a degree that we decided to implement a custom algorithm, which we term tree-based matching, which has a better average-case performance (see evaluation), even while having a comparatively bad worst-case complexity. Our algorithm constructs a tree of all possible object/trajectory combinations in the frame and tries to find a compatible (such that no objects/trajectories are assigned twice) set of choices, maximizing the sum of probabilities amongst these choices (described in detail in the appendix Matching an object to an object in the next frame). Problematic are situations where a large number of objects are in close proximity of one another, since then the number of possible sets of choices grows exponentially. These situations are avoided by using a mixed approach: tree-based matching is used most of the time, but as soon as the combinatorical complexity of a certain situation becomes too great, our software falls back on using the Hungarian method. If videos are known to be problematic throughout (e.g. with >100 individuals consistently very close to each other), the user may choose to use an approximate method instead (described in the appendix section A.4), which simply iterates through all objects and assigns each to the trajectory for which it has the highest probability and subsequently does not consider whether another object has an even higher probability for that trajectory. While the approximate method scales better with an increasing number of individuals, it is "wrong" (seeing as it does not consider all pos-
sible combinations) – which is why it is not recommended unless strictly necessary. However, since it does not consider all combinations, making it more sensitive to parameter choice, it scales better for very large numbers of objects and produces results good enough for it to be useful in very large groups (see A2).

Situations where objects/individuals are touching, partly overlapping, or even completely overlapping, is an issue that all tracking solutions have to deal with in some way. The first problem is the detection of such an overlap/crossing, the second is its resolution. idtracker.ai, for example, deals only with the first problem: It trains a neural network to detect crossings and essentially ignores the involved individuals until the problem is resolved by movement of the individuals themselves. However, using such an image-based approach can never be fully independent of the species or even video (it has to be retrained for each specific experiment) while also being time-costly to use. In some cases the size of objects might indicate that they contain multiple overlapping objects, while other cases might not allow for such an easy distinction – e.g. when sexually dimorphic animals (or multiple species) are present at the same time. We propose a method, similar to xyTracker in that it uses the object’s movement history to detect overlaps. If there are fewer objects in a region than would be expected by looking at previous frames, an attempt is made to split the biggest ones in that area. The size of that area is estimated using the maximal speed objects are allowed to travel per frame (parameter, see documentation track_max_speed). This, of course, requires relatively good predictions or, alternatively, high frame-rates relative to the object’s movement speeds (which are likely necessary anyway to observe behavior at the appropriate time-scales).

By default, objects suspected to contain overlapping individuals are split by thresholding their background-difference image (see appendix section A.11), continuously increasing the threshold until the expected number (or more) similarly sized objects are found. Greyscale values and, more generally, the shading of three-dimensional objects and animals often produces a natural gradient (see for example Figure 8) making this process surprisingly effective for many of the species we tested with. Even when there is almost no visible gradient and thresholding produces holes inside objects, objects are still successfully separated with this approach. Missing pixels from inside the objects can even be regenerated afterwards. The algorithm fails, however, if the remaining objects are too small or are too different in size, in which case the overlapping objects will not be assigned to any trajectory until all involved objects are found again separately in a later frame.

After an object is assigned to a specific trajectory, two kinds of data (posture and visual-fields) are calculated and made available to the user, which will each be described in one of the following subsections. In the last subsection, we outline how these can be utilized in real-time tracking situations.
1.4.2.1 Posture Analysis

Groups of animals are often modeled as systems of simple particles (Inada & Kawachi 107, Cavagna et al. 35, Pérez-Escudero & de Polavieja 169), a reasonable simplification which helps to formalize/predict behavior. However, intricate behaviors, like courtship displays, can only be fully observed once the body shape and orientation are considered (e.g. using tools such as DeepPoseKit, Graving et al. 86, LEAP [166]/SLEAP [167], and DeepLabCut, Mathis et al. 143). TRex does not track individual body parts apart from the head and tail (where applicable), but even the included simple and fast 2D posture estimator already allows for deductions to be made about how an animal is positioned in space, bent and oriented – crucial e.g. when trying to estimate the position of eyes/antennae as part of an analysis, where this is required (e.g. Strandburg-Peshkin et al. 203, Rosenthal et al. 186). When detailed tracking of all extremities is required, TRex offers an option that allows it to interface with third-party software like DeepPoseKit (Graving et al. 86), SLEAP (Pereira et al. 167), or DeepLabCut (Mathis et al. 143). This option (output_image_per_tracklet), when set to true, exports cropped and (optionally) normalised videos per individual that can be imported directly into these tools – where they might perform better than the raw video. Normalisation, for example, can make it easier for machine-learning algorithms in these tools to learn where body-parts are likely to be (see Figure 5) and may even reduce the number of clicks required during annotation.

In TRex, the 2D posture of an animal consists of (i) an outline around the outer edge of a blob, (ii) a center-line (or midline for short) that curves with the body and (iii) positions on the outline that represent the front and rear of the animal (typically head and tail). Our only assumptions here are that the animal is bilateral with a mirror-axis through its center and that it has a beginning and an end, and that the camera-view is roughly perpendicular to this axis. This is true for most animals, but may not hold e.g. for jellyfish (with radial symmetry) or animals with different symmetries (e.g. radiolaria (protozoa) with spherical symmetry). Still, as long as the animal is not exactly circular from the perspective of the camera, the midline will follow its longest axis and a posture can be estimated successfully. The algorithm implemented in our software is run for every (cropped out) image of an individual and processes it as follows:

i. A tree-based approach follows edge pixels around an object in a clock-wise manner. Drawing the line around pixels, as implemented here, instead of through their centers, as done in comparable approaches, helps with very small objects (e.g. one single pixel would still be represented as a valid outline, instead of a single point).

ii. The pointiest end of the outline is assumed, by default, to be either the tail or the head (based
on curvature and area between the outline points in question). Assignment of head vs. tail can be set by the user, seeing as some animals might have "pointier" heads than tails (e.g. termite workers, one of the examples we employ). Posture data coming directly from an image can be very noisy, which is why the program offers options to simplify outline shapes using an Elliptical Fourier Transform (EFT, see Iwata et al. 108, Kuhl & Giardina 121) or smoothing via a simple weighted average across points of the curve (inspired by common subdivision techniques, see Warren & Weimer 223). The EFT allows for the user to set the desired level of approximation detail (via the number of elliptic fourier descriptors, EFDs) and thus make it "rounder" and less jittery. Using an EFT with just two descriptors is equivalent to fitting an ellipse to the animal’s shape (as, for example, xyTracker does), which is the simplest supported representation of an animal’s body.

iii. The reference-point chosen in (ii) marks the start for the midline-algorithm. It walks both left and right from this point, always trying to move approximately the same distance on the outline (with limited wiggle-room), while at the same time minimizing the distance from the left to the right point. This works well for most shapes and also automatically yields distances between a midline point and its corresponding two points on the outline, estimating thickness of this object’s body at this point.

Compared to the tracking itself, posture estimation is a time-consuming process and can be disabled. It is, however, required to estimate – and subsequently normalize – an animal’s orientation in space (e.g. required later in Automatic Visual Identification Based on Machine Learning), or to reconstruct their visual field as described in the following sub-section.

1.4.2.2 Reconstructing 2D Visual Fields

Visual input is an important modality for many species (e.g. fish Strandburg-Peshkin et al. 203, Bilotta & Saszik 21 and humans Colavita 41). Due to its importance in widely used model organisms like zebrafish (Danio rerio), we decided to include the capability to conduct a 2-dimensional reconstruction of each individual’s visual field as part of the software. The requirements for this are successful posture estimation and that individuals are viewed from above, as is usually the case in laboratory studies.

The algorithm makes use of the fact that outlines have already been calculated during posture estimation. Eye positions are estimated to be evenly distanced from the "snout" and will be spaced apart depending on the thickness of the body at that point (the distance is based on a ratio, relative to body-size, which can be adjusted by the user). Eye orientation is also adjustable, which influences the size of the stereoscopic part of the visual field. We then use ray-casting to intersect rays from each of
Figure 7: Visual field estimate of the individual in the center (zoomed in, the individuals are approximately 2-3cm long, video 15). Right (blue) and left (orange) fields of view intersect in the binocular region (pink). Most individuals can be seen directly by the focal individual (1, green), which has a wide field of view of $260^\circ$ per eye. Individual 3 on the top-left is not detected by the focal individual directly and not part of its first-order visual field. However, second-order intersections (visualized by grey lines here) are also saved and accessible through a separate layer in the exported data.

Figure 7–video 1. A clip from video 15, showing TRex’ visual-field estimation for Individual 1.

https://youtu.be/yEO_31pZIzU

the eyes with all other individuals as well as the focal individual itself (self-occlusion). Individuals not detected in the current frame are approximated using the last available posture. Data are organized as a multi-layered 1D-image of fixed size for each frame, with each image prepresenting angles from $-180^\circ$ to $180^\circ$ for the given frame. Simulating a limited field-of-view would thus be as simple as cropping parts of these images off the left and right sides. The different layers per pixel encode:

1. identity of the occluder
2. distance to the occluder
3. body-part that was hit (distance from the head on the outline in percent)

While the individuals viewed from above on a computer screen look 2-dimensional, one major disadvantage of any 2D approach is, of course, that it is merely a projection of the 3D scene. Any
visual field estimator has to assume that, from an individual’s perspective, other individuals act as an occluder in all instances (see Figure 7). This may only be partly true in the real world, depending on the experimental design, as other individuals may be able to move slightly below, or above, the focal individuals line-of-sight, revealing otherwise occluded conspecifics behind them. We therefore support multiple occlusion-layers, allowing second-order and Nth-order occlusions to be calculated for each individual.

1.4.2.3 Realtime Tracking Option for Closed-Loop Experiments

Live tracking is supported, as an option to the user, during the recording, or conversion, of a video in TGrabs. When closed-loop feedback is enabled, TGrabs focusses on maintaining stable recording frame-rates and may not track recorded frames if tracking takes too long. This is done to ensure that the recorded file can later be tracked again in full/with higher accuracy (thus no information is lost) if required, and to help the closed-loop feedback to stay synchronized with real-world events.

During development we worked with a mid-range gaming computer and Basler cameras at 90fps and 2048² px resolution, where drawbacks did not occur. Running the program on hardware with specifications below our recommendations (see Results), however, may affect frame-rates as described below.

TRes loads a prepared Python script, handing down an array of data per individual in every frame. Which data fields are being generated and sent to the script is selected by the script. Available fields are:

- Position
- Midline information
- Visual field

If the script (or any other part of the recording process) takes too long to execute in one frame, consecutive frames may be dropped until a stable frame-rate can be achieved. This scales well for all computer-systems, but results in fragmented tracking data, causing worse identity assignment, and reduces the number of frames and quality of data available for closed-loop feedback. However, since even untracked frames are saved to disk, these inaccuracies can be fixed in TRes later. Alternatively, if live-tracking is enabled but closed-loop feedback is disabled, the program maintains detected objects in memory and tracks them in an asynchronous thread (potentially introducing wait time after the recording stops). When the program terminates, the tracked individual’s data are exported – along with a results file that can be loaded by the tracker at a later time.
In order to make this interface easy to use for prototyping and to debug experiments, the script may be changed during its run-time and will be reloaded if necessary. Errors in the Python code lead to a temporary pause of the closed-loop part of the program (not the recording) until all errors have been fixed.

Additionally, thanks to Python being a fully-featured scripting language, it is also possible to call and send information to other programs during real-time tracking. Communication with other external programs may be necessary whenever easy-to-use Python interfaces are not available for e.g. hardware being used by the experimenter.

1.4.3 Automatic Visual Identification Based on Machine Learning

Tracking, when it is only based on individual’s positional history, can be very accurate under good circumstances and is currently the fastest way to analyse video recordings or to perform closed-loop experiments. However, such tracking methods simply do not have access to enough information to allow them to ensure identities are maintained for the duration of most entire trials – small mistakes can and will happen. There are cases, e.g. when studying polarity (only based on short trajectory segments), or other general group-level assessments, where this is acceptable and identities do not have to be maintained perfectly. However, consistent identities are required in many individual-level assessments, and with no baseline truth available to correct mistakes, errors start accumulating until eventually all identities are fully shuffled. Even a hypothetical, perfect tracking algorithm will not be able to yield correct results in all situations as multiple individuals might go out of view at the same time (e.g. hiding under cover or just occluded by other animals). There is no way to tell who is whom, once they re-emerge.

The only way to solve this problem is by providing an independent source of information from which to infer identity of individuals, which is of course a principle we make use of all the time in our everyday lives: Facial identification of con-specifics is something that is easy for most humans, to an extent where we sometimes recognize face-like features where there aren’t any. Our natural tendency to find patterns enables us to train experts on recognizing differences between animals, even when they belong to a completely different taxonomic order. Tracking individuals is a demanding task, especially with large numbers of moving animals (Liu et al. 132 shows humans to be effective for up to 4 objects). Human observers are able to solve simple memory recall tasks for 39 objects at only 92% correct (see Humphrey & Khan 106), where the presented objects do not even have to be identified individually (just classified as old/new) and contain more inherent variation than most con-specific
animals would. Even with this being true, human observers are still the most efficient solution in some cases (e.g. for long-lived animals in complex habitats). Enhancing visual inter-individual differences by attaching physical tags is an effective way to make the task easier and more straight-forward to automate. RFID tags are useful in many situations, but are also limited since individuals have to be in very close proximity to a sensor in order to be detected [22]. Attaching fiducial markers (such as QR codes) to animals allows for a very large number (thousands) of individuals to be uniquely identified at the same time (see Gernat et al. 79, Wild et al. 227, Mersch et al. 145, Crall et al. 48) – and over a much greater distance than RFID tags. Generating codes can also be automated, generating tags with optimal visual inter-marker distances [76], making it feasible to identify a large number of individuals with minimal tracking mistakes.

While physical tagging is often an effective method by which to identify individuals, it requires animals to be caught and manipulated, which can be difficult [145] and is subject to the physical limitations of the respective system. Tags have to be large enough so a program can recognize it in a video stream. Even worse, especially with increased relative tag-size, the animal’s behavior may be affected by the presence of the tag or during its application (Dennis et al. 57, Pankiw & Page 162, Sockman & Schwabl 197), and there might be no way for experimenters to necessarily know that it did (unless with considerable effort, see Switzer & Combes 206). In addition, for some animals, like fish and termites, attachment of tags that are effective for discriminating among a large number of individuals can be problematic, or impossible.

Recognizing such issues, [173] first proposed an algorithm termed idtracker, generalizing the process of pattern recognition for a range of different species. Training an expert program to tell individuals apart, by detecting slight differences in patterning on their bodies, allows the correction of identities without any human involvement. Even while being limited to about 15 individuals per group, this was a very promising approach. It became much improved upon only a few years later by the same group in their software idtracker.ai [184], implementing a paradigm shift from explicit, hard-coded, color-difference detection to using more general machine learning methods instead – increasing the supported group size by an order of magnitude.

We employ a method for visual identification in TRex that is similar to the one used in idtracker.ai, where a neural network is trained to visually recognize individuals and is used to correct tracking mistakes automatically, without human intervention – the network layout (see Figure 1c) is almost the same as well (differing only by the addition of a pre-processing layer and using 2D- instead of 1D-dropout layers). However, in TRex, processing speed and chances of success are improved (the former being greatly improved) by (i) minimizing the variance landscape of the problem and (ii) exploring
the landscape to our best ability, optimally covering all poses and lighting-conditions an individual can be in, as well as (iii) shortening the training duration by significantly altering the training process – e.g. choosing new samples more adaptively and using different stopping-criteria (accuracy, as well as speed, are part of the later evaluation).

While Tracking already tries to (within each trajectory) consistently follow the same individual, there is no way to ensure/check the validity of this process without providing independent identity information. Generating this source of information, based on the visual appearance of individuals, is what the algorithm for visual identification, described in the following subsections, aims to achieve. Re-stated simply, the goal of using automatic visual identification is to obtain reliable predictions of the identities of all (or most) objects in each frame. Assuming these predictions are of sufficient quality, they can be used to detect and correct potential mistakes made during Tracking by looking for identity switches within trajectories. Ensuring that predicted identities within trajectories are consistent, by proxy, also ensures that each trajectory is consistently associated with a single, real individual. In the following, before describing the four stages of that algorithm, we will point out key aspects of how tracking/image data are processed and how we addressed the points (i)-(iii) above and especially highlight the features that ultimately improved performance compared to other solutions.

### 1.4.3.1 Preparing Tracking-Data

Visual identification starts out only with the trajectories that the Tracking provides. Tracking, on its own, is already an improvement over other solutions, especially since (unlike e.g. idtracker.ai) TRex makes an effort to separate overlapping objects (see the Algorithm for splitting touching individuals) and thus is able to keep track of individuals for longer (see A6). Here, we – quite conservatively – assume that, after every problematic situation (defined in the list below), the assignments made by our tracking algorithm are wrong. Whenever a problematic situation is encountered as part of a trajectory, we split the trajectory at that point. This way, all trajectories of all individuals in a video become an assortment of trajectory snippets (termed "segments" from here on), which are clear of problematic situations, and for each of which the goal is to find the correct identity ("correct" meaning that identities are consistently assigned to the same real individual throughout the video). Situations are considered "problematic", and cause the trajectory to be split, when:

- **The individual has been lost for at least one frame.** For example when individuals are moving unexpectedly fast, are occluded by other individuals/the environment, or simply not present anymore (e.g. eaten).
- **Uncertainty of assignment was too high** (> 50%) e.g. due to very high movement speeds or extreme variation in size between frames. With simpler tracking tasks in mind, these segments are kept as *connected* tracks, but regarded as separate ones here.

- **Timestamps suggest skipped frames.** Missing frames in the video may cause wrong assignments and are thus treated as if the individuals have been lost. This distinction can only be made if accurate frame timings are available (when recording using TGrabs or provided alongside the video files in separate npz files).

Unless one of the above conditions becomes true, a segment is assumed to be consecutive and connected; that is, throughout the whole segment, no mistakes have been made that lead to identities being switched. Frames where all individuals are currently within one such segment at the same time will henceforth be termed *global segments*.

Since we know that there are no problematic situations inside each per-individual segment, and thus also not across individuals within the range of a global segment, we can choose any global segment as a basis for an initial, arbitrary assignment of identities to trajectories. One of the most important steps of the identification algorithm then becomes deciding which global segment is the best starting point for the training. If a mistake is made here, consecutive predictions for other segments will fail and/or produce unreliable results in general.

Only a limited set of global segments is kept – striking a balance between respecting user-given constraints and capturing as much of the variance as possible. In many of the videos used for evaluation, we found that only few segments had to be considered – however, computation time is ultimately bounded by reducing the number of qualifying segments. While this is true, it is also beneficial to avoid auto-correlation by incorporating samples from all sections of the video instead of only sourcing them from a small portion – to help achieve a balance, global segments are binned by their middle frame into four bins (each quarter of the video being a bin) and then reducing the number of segments inside each bin. With that goal in mind, we sort the segments within bins by their ”quality” – a combination of two factors:

1. To capture as much as possible the variation due to an individual’s own movement, as well as within the background that it moves across, a ”good” segment should be a segment where all individuals move as much as possible and also travel as large a distance as possible. Thus, we derive a per-individual *spatial coverage descriptor* for the given segment by dissecting the arena (virtually) into a grid of equally sized, rectangular ”cells” (depending on the aspect ratio of the video). Each time an individual’s center-point moves from one cell to the next, a counter is
incremented for that individual. To avoid situations where, for example, all individuals but one are moving, we only use the lowest per-individual spatial coverage value to represent a given segment.

2. It is beneficial to have more examples for the network to learn from. Thus, as a second sorting criterion, we use the average number of samples per individual.

After being sorted according to these two metrics, the list of segments per bin is reduced, according to a user-defined variable (4 by default), leaving only the most viable options per quarter of video.

The number of visited cells may, at first, appear to be essentially equivalent to a spatially normalized distance travelled (as used in idtracker.ai). In edge cases, where individuals never stop or always stop, both metrics can be very similar. However, one can imagine an individual continuously moving around in the same corner of the arena, which would be counted as an equally good segment for that individual as if it had traversed the whole arena (and thus capturing all variable environmental factors). In most cases, using highly restricted movement for training is problematic, and worse than using a shorter segment of the individual moving diagonally through the entire space, since the latter captures more of the variation within background, lighting conditions and the animals movement in the process.

1.4.3.2 Minimizing the Variance Landscape by Normalizing Samples

A big strength of machine learning approaches is their resistance to noise in the data. Generally, any machine learning method will likely still converge - even with noisy data. Eliminating unnecessary noise and degrees of freedom in the dataset, however, will typically help the network to converge much more quickly: Tasks that are easier to solve will of course also be solved more accurately within similar or smaller timescales. This is due to the optimizer not having to consider various parts of the possible parameter-space during training, or, put differently, shrinking the overall parameter-space to the smallest possible size without losing important information. The simplest such optimization included in most tracking and visual identification approaches is to segment out the objects and centering the individuals in the cropped out images. This means that (i) the network does not have to consider the whole image, (ii) needs only to consider one individual at a time and (iii) the corners of the image can most likely be neglected.

Further improving on this, approaches like idtracker.ai align all objects along their most-elongated axis, essentially removing global orientation as a degree of freedom. The orientation of an arbitrary
object can be calculated e.g. using an approach often referred to as image-moments [100], yielding an angle within $[0 - 180]^{\circ}$. Of course, this means that

1. circular objects have a random (noisy) orientation

2. elongated objects (e.g. fish) can be either head-first or flipped by $180^{\circ}$ and there is no way to discriminate between those two cases (see second row, Figure 8)

3. a C-shaped body deformation, for example, results in a slightly bent axis, meaning that the head will not be in exactly the same position as with a straight posture of the animal.

Each of these issues adds to the things the network has to learn to account for, widening the parameter-space to be searched and increasing computation time. However, barring the first point, each problem can be tackled using the already available posture information. Knowing head and tail positions and points along the individual’s center-line, the individual’s heads can be locked roughly into a single position. This leaves room only for their rear end to move, reducing variation in the data to a minimum (see Figure 8). In addition to faster convergence, this also results in better generalization right from the start and even with a smaller number of samples per individual (see Figure 5). For further discussion of highly deformable bodies, such as of rodents, please see Appendix (Posture and Visual Identification of Highly-Deformable Bodies).
1.4.3.3 Guiding the Training Process

Per batch, the stochastic gradient descent is directed by the local accuracy (a fraction of correct/total predictions), which is a simple and commonly used metric that has no prior knowledge of where the samples within a batch come from. This has the desirable consequence that no knowledge about the temporal arrangement of images is necessary in order to train and, more importantly, to apply the network later on.

In order to achieve accurate results quickly across batches, while at the same time making it possible to indicate to the user potentially problematic sequences within the video, we devised a metric that can be used to estimate local as well as global training quality: We term this uniqueness and it combines information about objects within a frame, following the principle of non-duplication; images of individuals within the same frame are required to be assigned different identities by the networks predictions.

**Box 1.1: Calculating uniqueness for a frame**

**Data:** frame $x$

**Result:** Uniqueness score for frame $x$

uids = map{}

$\hat{p}(i \mid b)$ is the probability of blob $b$ to be identity $i$

$f(x)$ returns a list of the tracked objects in frame $x$

$E(v) = \frac{1 + \exp(-\pi)}{1 + \exp(-\pi v)}$ is a shift of roughly +0.5 and non-linear scaling of values $0 \leq v \leq 1$

foreach object $b \in f(x)$ do

maxid = $\arg\max_i \hat{p}(i \mid b)$ with $i \in$ identities

if maxid $\in$ uids then

uids[maxid] = max(uids[maxid], $\hat{p}$(maxid, $b$))

else

uids[maxid] = $\hat{p}$(maxid, $b$)

end

end

return $(|uids|^{-1}|f(x)| \ast E \left(|uids|^{-1} \left(\sum_{i \in uids} uids[i]\right)\right))$

**Algorithm 1:** The algorithm used to calculate the uniqueness score for an individual frame. Probabilities $\hat{p}(i \mid b)$ are predictions by the pre-trained network. During the accumulation these predictions will gradually improve proportional to the global training quality. Multiplying the unique percentage $|uids|^{-1}|f(x)|$ by the (scaled) mean probability deals with cases of low accuracy, where individuals switch every frame (but uniquely).

The program generates image data for evenly spaced frames across the entire video. All images of tracked individuals within the selected frames are, after every epoch of the training, passed on to
the network. It returns a vector of probabilities $p_{ij}$ for each image $i$ to be identity $j \in [0, N]$, with $N$ being the number of individuals. Based on these probabilities, uniqueness can be calculated as in Box 1.1, evenly covering the entire video. The magnitude of this probability vector per image is taken into account, rewarding strong predictions of $\max_j \{p_{ij}\} = 1$ and punishing weak predictions of $\max_j \{p_{ij}\} < 1$.

Uniqueness is not integrated as part of the loss function, but it is used as a global gradient before and after each training unit in order to detect global improvements. Based on the average uniqueness calculated before and after a training unit, we can determine whether to stop the training, or whether training on the current segment made our results worse (faulty data). If uniqueness is consistently high throughout the video, then training has been successful and we may terminate early. Otherwise, valleys in the uniqueness curve indicate bad generalization and thus currently missing information regarding some of the individuals. In order to detect problematic sections of the video we search for values below $1 - \frac{0.5}{N}$, meaning that the section potentially contains new information we should be adding to our training data. Using accuracy per-batch and then using uniqueness to determine global progress, we get the best of both worlds: A context-free prediction method that is trained on global segments that are strategically selected by utilizing local context information.

The closest example of such a procedure in idtracker.ai is the termination criterion after protocol I, which states that individual segments have to be consistent and certain enough in all global segments in order to stop iterating. While this seems to be similar at first, the way accuracy is calculated and the terminology here are quite different: (i) Every metric in idtracker.ai’s final assessment after protocol 1 is calculated at segment-level, not utilizing per-frame information. Uniqueness works per-frame, not per segment, and considers individual frames to be entirely independent from each other. It can be considered a much stronger constraint set upon the network’s predictive ability, seeing as it basically counts the number of times mistakes are estimated to have happened within single frames. Averaging only happens afterwards. (ii) The terminology of identities being unique is only used in idtracker.ai once after protocol 1 and essentially as a binary value, not recognizing its potential as a descendable gradient. Images are simply added until a certain percentage of images has been reached, at which point accumulation is terminated. (iii) Testing uniqueness is much faster than testing network accuracy across segments, seeing as the same images are tested over and over again (meaning they can be cached) and the testing dataset can be much smaller due to its locality. Uniqueness thus provides a stronger gradient estimation, while at the same time being more local (meaning it can be used independently of whether images are part of global segments), as well as more manageable in terms of speed and memory size.
In the next four sections, we describe the training phases of our algorithm (1-3), and how the successfully trained network can be used to automatically correct trajectories based on its predictions (4).

1.4.3.4 1. The Initial Training Unit

All global segments are considered and sorted by the criteria listed below in 2. Accumulation of Additional Segments and Stopping-Criteria. The best suitable segment from the beginning of that set of segments is used as the initial dataset for the network. Images are split into a training and a validation set (4:1 ratio). Efforts are made to equalize the sample sizes per class/identity beforehand, but there has to always be a trade-off between similar sample sizes (encouraging unbiased priors) and having as many samples as possible available for the network to learn from. Thus, in order to alleviate some of the severity of dealing with imbalanced datasets, the performance during training iterations is evaluated using a categorical focal loss function \([130]\). Focal loss down-weighs classes that are already reliably predicted by the network and in turn emphasizes neglected classes. An Adam optimizer \([117]\) is used to traverse the loss landscape towards the global (or to at least a local) minimum.

The network layout used for the classification in TRex (see Figure 1c) is a typical Convolutional Neural Network (CNN). The concepts of "convolutional" and "downsampling" layers, as well as the back-propagation used during training, are not new. They were introduced in \([74]\), inspired originally by the work of Hubel and Wiesel on cats and rhesus monkeys (Hubel & Wiesel 101, Hubel & Wiesel 102, Wiesel & Hubel 226), describing receptive fields and their hierarchical structure in the visual cortex. Soon afterward, in \([128]\), CNNs, in combination with back-propagation, were already successfully used to recognize handwritten ZIP codes – for the first time, the learning process was fully automated. A critical step towards making their application practical, and the reason they are popular today.

The network architecture used in our software is similar to the identification module of the network in \([184]\), and is, as in most typical CNNs, (reverse-)pyramid-like. However, key differences between TRex’ and idtracker.ai’s procedure lie with the way that training data is prepared (see previous sections) and how further segments are accumulated/evaluated (see next section). Furthermore, contrary to idtracker.ai’s approach, images in TRex are augmented (during training) before being passed on to the network. While this augmentation is relatively simple (random shift of the image in x-direction), it can help to account for positional noise introduced by e.g. the posture estimation or the video itself when the network is used for predictions later on \([168]\). We do not flip the image in 45
this step, or rotate it, since this would defeat the purpose of using orientation normalization in the first place (as in Minimizing the Variance Landscape by Normalizing Samples, see Figure 8). Here, in fact, normalization of object orientation (during training and predictions) could be seen as a superior alternative to data augmentation.

The input data for TRex’ network is a single, cropped grayscale image of an individual (see Figure 1c). This image is first passed through a "lambda" layer (blue) that normalizes the pixel values, dividing them by half the value limit of $\frac{255}{2} = 127.5$ and subtracting 1 – this moves them into the range of $[-1, 1]$. From then on, sections are a combination of convolutional layers (kernel sizes of 16, 64 and 100 pixels), each followed by a 2D (2x2) max-pooling and a 2D spatial dropout layer (with a rate of 0.25). Within each of these blocks the input data is reduced further, focussing it down to information that is deemed important. Towards the end, the data are flattened and flow into a densely connected layer (100 units) with exactly as many outputs as the number of classes. The output is a vector with values between 0 and 1 for all elements of the vector, which, due to softmax-activation, sum to 1.

Training commences by performing a stochastic gradient descent (using the Adam optimizer, see Kingma & Ba), which iteratively minimizes the error between network predictions and previously known associations of images with identities – the original assignments within the initial frame segment. The optimizer’s behavior in the last five epochs is continuously observed and training is terminated immediately if one of the following criteria is met:

- the maximum number of iterations is reached (150 by default, but can be set by the user)
- a plateau is achieved at a high per-class accuracy
- overfitting/overly optimizing for the training data at the loss of generality
- no further improvements can be made (due to the accuracy within the current training data already being 1)

The initial training unit is also by far the most important as it determines the predicted identities within further segments that are to be added. It is thus less risky to overfit than it is important to get high-quality training results, and the algorithm has to be relatively conservative regarding termination criteria. Later iterations, however, are only meant to extend an already existing dataset and thus (with computation speed in mind) allow for additional termination criteria to be added:

- plateauing at/circling around a certain val_loss level
- plateauing around a certain uniqueness level

46
If necessary, initial training results can be improved by adding more samples to the active dataset. This could be done manually by the user, always trying to select the most promising segment next, but requiring such manual work is not acceptable for high-throughput processing. Instead, in order to translate this idea into features that can be calculated automatically, the following set of metrics is re-generated per (yet inactive) segment after each successful step:

1. Average uniqueness index (rounded to an integer percentage in 5% steps)
2. Minimal distance to regions that have previously been trained on (rounded to the next power of two), larger is better as it potentially includes samples more different from the already known ones
3. Minimum cells visited per individual (larger is better for the same reason as 2)
4. Minimum average samples per individual (larger is better)
5. Whether its image data has already been generated before (mostly for saving memory)
6. The uniqueness value is smaller than $U_{prev}^p$ after 5 steps, with $U_{prev}$ being the best uniqueness value previous to the current accumulation step

With the help of these values, the segment list is sorted and the best segment selected to be considered next. Adding a segment to a set of already active samples requires us to correct the identities inside it, potentially switching temporary identities to represent the same real identities as in our previous data. This is done by predicting identities for the new samples using the network that has been trained on the old samples. Making mistakes here can lead to significant subsequent problems, so merely plausible segments will be added - meaning only those samples are accepted for which the predicted IDs are unique within each unobstructed sequence of frames for every temporary identity. If multiple temporary individuals are predicted to be the same real identity, the segment is saved for later and the search continues.

If multiple additional segments are found, the program tries to actively improve local uniqueness valleys by adding samples first from regions with comparatively low accuracy predictions. Seeing as low accuracy regions will also most likely fail to predict unique identities, it is important to emphasize here that this is generally not a problem for the algorithm: Failed segments are simply ignored and can
be inserted back into the queue later. Smoothing the curve also makes sure to prefer regions close to valleys, making the algorithm follow the valley walls upwards in both directions.

Finishing a training unit does not necessarily mean that it was successful. Only the network states improving upon results from previous units are considered and saved. Any training result - except the initial one - may be rejected after training in case the uniqueness score has not improved globally, or at least remained within 99% of the previous best value. This ensures stability of the process, even with tracking errors present (which can be corrected for later on, see next section). If a segment is rejected, the network is restored to the best recorded state.

Each new segment is always combined with regularly sampled data from previous steps, ensuring that identities don’t switch back and forth between steps due to uncertain predictions. If switching did occur, then the uniqueness and accuracy values can never reach high value regimes – leading to the training unit being discarded as a result. The contribution of each previously added segment $R$ is limited to $\lceil |R_S|/(\text{samples}_\text{max} \times |R|/N) \rceil$ samples, with $N$ as the total number of frames in global segments for this individual and $\text{samples}_\text{max}$ a constant that is calculated using image size and memory constraints (or 1GB by default). $R_S$ is the actual usable number of images in segment $R$. This limitation is an attempt to not bias the priors of the network by sub-sampling segments according to their contribution to the total number of frames in global segments.

Training is considered to be successful globally, as soon as either (i) accumulative individual gaps between sampled regions is less than 25% of the video length for all individuals, or (ii) uniqueness has reached a value higher than $1 - \frac{\sigma}{N_{\text{id}}}$ (1.1) so that almost all detected identities are present exactly once per frame. Otherwise, training will be continued as described above with additional segments – each time extending the percentage of images seen by the network further.

Training accuracy/consistency could potentially be further improved by letting the program add an arbitrary amount of segments, however we found this not to be necessary in any of our test-cases. Users are allowed to set a custom limit if required in their specific cases.

1.4.3.6 3. The Final Training Unit

After the accumulation phase, one last training step is performed. In previous steps, validation data has been kept strictly separate from the training set to get a better gauge on how generalizable the results are to unseen parts of the video. This is especially important during early training units, since "overfitting" is much more likely to occur in smaller datasets and we still potentially need to add samples from different parts of the video. Now that we are not going to extend our training dataset any-
more, maintaining generalizibility is no longer the main objective – so why not use all of the available data? The entire dataset is simply merged and sub-sampled again, according to the memory strategy used. Network training is started, with a maximum of \( \max \{ 3; \max \_epochs \times 0.25 \} \) iterations (\( \max \_epochs \) is 150 by default). During this training, the same stopping-criteria apply as during the initial step.

Even if we tolerate the risk of potentially overfitting on the training data, there is still a way to detect overfitting if it occurs: Only training steps that lead to improvements in mean uniqueness across the video are saved. Often, if prediction results become worse (e.g. due to overfitting), multiple individuals in a single frame are predicted to be the same identity – precisely the problem which our uniqueness metric was designed to detect.

For some videos, this is the step where most progress is made (e.g. video 9). The reason being that this is the first time when all of the training data from all segments is considered at once (instead of mostly the current segment plus fewer samples from previously accepted segments), and samples from all parts of the video have an equal likelihood of being used in training after possible reduction due to memory-constraints.

### 4. Assigning Identities Based on Network Predictions

After the network has been successfully trained, all parts of the video which were not part of the training are packaged together and the network calculates predictive probabilities for each image of each individual to be any of the available identities. The vectors returned by the network are then averaged per consecutive segment per individual. The average probability vectors for all overlapping segments are weighed against each other – usually forcing assignment to the most likely identity (ID) for each segment, given that no other segments have similar probabilities. When referring to segments here, meant is simply a number of consecutive frames of one individual that the tracker is fairly sure does not contain any mix-ups. We implemented a way to detect tracking mistakes, which is mentioned later.

If an assignment is ambiguous, meaning that multiple segments \( S_j \ldots S_M \) overlapping in time have the same maximum probability index \( \arg \max_{i \in [0,N]} \{ P(i \mid S_j) \} \) (for the segment to belong to a certain identity \( i \)), a decision has to be made. Assignments are deferred if the ratio

\[
R_{\max} = \max \left\{ \frac{P(i \mid S_j)}{P(i \mid S_k)}, \forall S_j \neq k \in \text{overlapping segments} \right\}
\]

between any two maximal probabilities is larger than 0.6 for said \( i \) (\( R_{\max} \) is inverted if it is greater than 0.6).
than 1). In such a case, we rely on the general purpose tracking algorithm to pick a sensible option – other identities might even be successfully assigned (using network predictions) in following frames, which is a complexity we do not have to deal with here. In case all ratios are below 0.6, when the best choices per identity are not too ambiguous, the following steps are performed to resolve remaining conflicts:

1. count the number of samples $N_{me}$ in the current segment, and the number of samples $N_{he}$ in the other segment that this segment is compared to

2. calculate average probability vectors $P_{me}$ and $P_{he}$

3. if $S(P_{me}, N_{me}) \geq S(P_{he}, N_{he})$, then assign the current segment with the ID in question. Otherwise assign the ID to the other segment. Where:

\[
\text{norm}(x) = \frac{x}{N_{me} + N_{he}}, \quad \text{sig}(x) = \left(1 + e^{2\pi (0.5 - x)}\right)^{-1}
\]

\[
S(p, x) = \text{sig}(p) + \text{sig}(\text{norm}(x)).
\]

(1.2)

This procedure prefers segments with larger numbers of samples over segments with fewer samples, ensuring that identities are not switched around randomly whenever a short segment (e.g. of noisy data) is predicted to be the given identity for a few frames – at least as long as a better alternative is available. The non-linearity in $S(p, x)$ exaggerates differences between lower values and dampens differences between higher values: For example, the quality of a segment with 4000 samples is barely different from a segment with 5000 samples; however, there is likely to be a significant quality difference between segments with 10 and 100 samples.

In case something goes wrong during the tracking, e.g. an individual is switched with another individual without the program knowing that it might have happened, the training might still be successful (for example if that particular segment has not been used for training). In such cases, the program tries to correct for identity switches mid-segment by calculating a running-window median identity throughout the whole segment. If the identity switches for a significant length of time, before identities are assigned to segments, the segment is split up at the point of the first change within the window and the two parts are handled as separate segments from then on.
1.5 Software and Licenses

TReX is published under the GNU GPLv3 license (see here for permissions granted by GPLv3). All of the code has been written by the first author of this paper (a few individual lines of code from other sources have been marked inside the code). While none of these libraries are distributed alongside TReX (they have to be provided separately), the following libraries are used: OpenCV (opencv.org) is a core library, used for all kinds of image manipulation. GLFW (glfw.org) helps with opening application windows and maintaining graphics contexts, while DearImGui (github.com/ocornut/imgui) helps with some more abstractions regarding graphics. pybind11 ([110]) for Python integration within a C++ environment. miniLZO (oberhumer.com/opensource/lzo) is used for compression of PV frames. Optional bindings are available to FFmpeg (ffmpeg.org) and libpng libraries, if available. (optional) GNU Libmicrohttpd (gnu.org/software/libmicrohttpd), if available, can be used for an HTTP interface of the software, but is non-essential.
Identifying the spatiotemporal locations of anomalies in collective behavior using simplified representations of trajectory data

A common challenge in the study of animal behavior is the identification of unexpected, and by definition rare, sequences in trajectory data. We present methods to (1) track all visible individuals, (2) simplify trajectory data, (3) use machine learning based sequence prediction to detect the temporal and spatial location of anomalous sequences in this simplified representation, (4) compare to a simpler approach using image processing, (5) perform a large-scale meta analysis evaluating the types of models best-suited to the task, (6) return to the trajectories and identify individuals implicated in initiating changes in collective behavior. As an example dataset for the application of our methods we use the trajectories of thousands of individuals in two densely populated videos of freely behaving termites (*Constrictotermes cyphergaster*) in a circular arena. In this dataset, our method identifies a number of anomalies, characterized by a sudden, local acceleration of small groups of individuals.
that sometimes spreads across the entire arena. Some of these events, which are ~1-5 min in duration, were very hard to find for human observers during normal playback and only became visible once automatically labelled as anomalies by our method. Depending on the magnitude of the event, the trail network and clusters may be temporarily dissolved and it takes minutes for the individuals return to their normal behavior. The detected events may be related to alarm responses, but further investigation is required. In this paper, we laid the necessary technological ground work to address such data-driven questions more efficiently, and largely automatically, in future studies.

2.1 Introduction

Changes in animal behavior can be defined on nearly arbitrary scales of time and space, ranging from cells [66] to the level of individuals, and described for entire species and ecosystems (see also [60] for a detailed definition as a multilevel syndrome). Long-term effects, such as seasonal rhythms [1] and weather conditions [30], stretch out over many months or years (e.g. 13 and 17 year cicadas [7]). Here we focus on short-term responses in the range of seconds to minutes, which, in social insect colonies, are often caused by the direct or indirect influence of nest-mates, parasites, and predators. More specifically, we are interested in intrinsic and emergent properties of individual behavior that have consequences for group-level foraging and predator evasion activity. Social insects (especially ants, termites, and bees) are perfect for this task because it is easy to observe almost natural behavior for an entire colony of individuals constrained within a relatively small arena. Many ant and termite species also form pheromone-based trail networks [146, 51], giving us direct access to their two-dimensional collective memory — arguably a sort of “hive mind”. Historically, however, computer memory and other hardware-imposed limitations made it difficult (sometimes impossible) to keep track of thousands of individuals simultaneously over long periods of time. Only recently, facilitated by technological advances in computation (such as graphics processing units, or GPUs), it has become possible to automate almost every step that previously involved days of tedious manual labor (see e.g. [161, 112, 113, 86, 143], and [161] for a recent review of 28 tracking solutions), enabling us to study impossibly large systems in manageable time.

Behavior of termites is severely understudied compared to that of other eusocial insects, which are often preferred for being generally deemed easier to work with. Compared to other social systems, termites are very sensitive to humidity and temperature conditions [44, 68]. Certain species can be difficult to keep alive outside their natural habitat, making it hard to design controlled and reliable experimental setups with captive termite colonies. The effect of this is apparent when a Google
Scholar search for “termite behavior” (without the quotes) returns only 49k results, while an analogous query for ants yields over 2.5 million results, e.g. [43, 219]… and over 1 million results for bees (e.g. [207, 16]…, searched on 28.10.21). There is good reason, however, to overcome these difficulties and study this complex insect more closely. Termites, which are sometimes also called “white ants” [119], seem very similar to Hymenoptera (bees and ants) at first glance. Both are known to be eusocial, and exhibit interesting trail optimisation and building strategies [124, 98, 146]. Despite this, however, they have been separated for over 350 million years of evolution. Hymenoptera appeared more than 50 million years after termites had evolved from cockroaches, during the late Jurassic [93, 23, 129]. This accounts for some of the fundamental, as well as more intricate, differences between them [119]. Termites, for example, are diploid, as opposed to haplodiploid, and so unlike ants, bees, and wasps, workers are not more closely related to their sisters than to their own offspring [156]. They do not have the same genetic motivation to protect or feed other members of the nest, and yet in many ways behave as if they did [211, 119]. Alternative selection pressures have been proposed to help explain this apparent altruism in evolutionary terms [211], but many aspects of their behavior unfortunately remain unexplored [91, 120, 194].

The field-data used here was collected on-site in Brazil of entire termite colonies (Constrictotermes cyphergaster Silvestri 1901) with thousands of individuals exploring an otherwise empty arena (see Materials and methods for details). This neotropical termite species occurs in South America [142] and has been recurrently studied in recent years especially due to its association with nest invaders, such as inquiline termites (Inquilinitermes microcerus; [104, 193, 51]), termophilous beetles (Corotoca melanthos; [185]) and more recently potter wasps (Montezumia termitophila; [105]). Another behavioral component of C. cyphergaster which seems to attract the attention of researchers is the fact that the species forages at night without the protection of covered trails [151]. Despite being a difficult study organism, multiple colonies could be maintained in the laboratory, allowing us to record more than 2h of footage per nest (for details on holding conditions and filming methods see 2.3, and refer to Hugo et al. 2021).

In this paper, we take advantage of modern technology to investigate this interesting example of collective organization, we focus on a common challenge in the study of animal behavior: can we search through complex, densely populated sequences of tracking data, and identify unexpected behaviors, which by their definition are rare, but could be highly consequential to the organism/collective?

Human observers are limited when it comes to tracking groups of animals [132] and fatigue not only influences how many observations can be made, but also which ones and how they are interpreted (see e.g. [27, 132, 33]). Thus, examining hours of footage manually is, especially given high
numbers of individuals, unrealistic but necessary to reveal large-scale and long-term effects (such as alarm responses). Designing an automatic method to pre-filter the data is, therefore, important when dealing with thousands of trajectory pieces in hundreds of thousands of video frames. Reducing the involvement of human observers also weakens the influence of their internal biases, making it an obvious goal for data-intense research. Many recent papers make use of, or suggest using, machine-learning based tools to reduce the manual work load (e.g. in posture estimation [86, 143], tracking and identification of behavioral motifs [229, 14, 113, 224, 184]). There is some reservation as to how interpretable the results of methods based on machine learning are [34], however, within the scope of this paper we use artificial neural networks (ANNs) only for the detection of potentially interesting sequences in large sets of time-series data, not to draw conclusions from them directly. “Anomalies” is the term we use to describe “sudden local (spatially and temporally) changes in density and/or speed distribution of individuals”, which are different from baseline behavior of the collective.

In summary, the main focus of this paper is to explore methods that could help

- simplifying the representation of trajectory data for basic features relevant to collective behavior,
- finding anomalous sequences within this simplified representation of trajectories,
- spatially locating events (e.g. alarm responses), as well as the individuals most likely to be involved,
- and doing so without requiring extensive human supervision.

We apply both a modern machine learning based approach, as well as one based on more traditional computer vision principles, to simplified representations of trajectory data from two videos of *C. cyphergaster* (each more than 2 hours long and showing thousands of individuals simultaneously), evaluate our detections based on manual annotations, and assess the value added by the deep learning approach. Additionally, we document an interesting behavioral feature of *C. cyphergaster* that we discovered in both videos (see “Results”). The example code, images, and animated figures from this paper are available online for free (see website github.com/mooch443/chap3). It is our hope that these will be helpful for related research projects in the future, since we think these methods can be widely applicable to other high-density data — even across fields (e.g. cell and particle movements).
2.2 Related Work

The method presented herein finds itself situated among a large body of other publications on “anomaly detection in time-series” (e.g. [195, 191, 89, 63, 80, 81, 10, 12, 140, 64, 116, 90]) from many different domains. Those research areas include computer/network security [89, 75, 31, 63, 80, 81, 69], detection of fluctuations in power grids [150], databases [123], and many more. More generally, “data mining” is concerned with the discovery and analysis of “interesting knowledge from different types of data” [5], and, being such a diverse topic, can be defined on many different scales of time and space (see [38, 39]) — similarly to animal behavior. Contemporary literature is saturated with specialised approaches, not all of which may be of use here. In the following, we explore the advantages and downsides of different solutions to give additional context for our choices.

“Change point detection” is the problem of discovering sudden, unusual changes in time-series data [116], sometimes on multiple time scales at once. Guralnik & Srivastava [90] proposed an algorithm for “event detection”, essentially the same as “change point detection”, stating that traditional change point detection algorithms often assume a pre-existing model when describing the phenomenon in question — a requirement they removed using an iterative approach in [90]. Their algorithm repeatedly tries to split the time-series into two significantly different parts (until it cannot be split any further), which is particularly interesting in this context since we would like to also discover unknown, and unmodelled, types of events. However, this algorithm can become computationally involved for long time-series and may also, depending on the likelihood function used, not capture non-linearities in the data adequately. An effective solution — especially keeping in mind that we set out to recommend a simple but workable solution by the end of this paper — would support sparse sampling, and be relatively simple to employ (single-pass).

Additionally, since one of our goals was also to locate potential events spatially, we require a detection method that works both spatially and temporally (ST). Most ST-outlier detection methods start by finding interesting spatial regions, and perform temporal detection as a second step [89]. Spatial outliers can be detected using various methods [123, 5, 38, 39], usually prefaced by a clustering method (e.g. DBSCAN [Ester et al.]), followed by comparison and verification steps. While unsupervised methods exist, many such clustering algorithms require manual visual checking of results (e.g. in [39]) and/or can be highly parameterised.

ANNs have been used successfully in many areas of (but not limited to) research and can provide exactly the 1-pass solution we were aiming for. They are not magical, so some may still require manual labor initially, but they can be applied almost without adaptation to different tasks (transfer learn-
ing) and can work without any manual annotations (unsupervised). ANNs are statistical tools, particularly well-suited for predicting common patterns in large datasets, and, analogously, for detecting major deviations from such “normal” behavior. This is commonly done by forecasting future states of complex systems, such as power loads [12], water resources (see [140] for a review), and traffic ([9], see [64] for a review), and comparing these predictions to actual data — revealing anomalies through “error” values: A common approach used to perform machine learning based change point/anomaly detection [89, 10]. Additionally, ANNs have also recently been utilised in various areas of animal behavior research [112, 113, 86, 143], including tracking and automatic pose-estimation and (importantly for our goals) deal well with two- and higher-dimensional data. One of the major criticisms associated with machine learning and ANNs specifically, is that it is hard to understand exactly how they work and what exactly each layer “does” [34]. However, it may still be worthwhile to utilise models despite their low interpretability of outputs in some situations [180], which is an ongoing debate (see e.g. [180, 72]). Valuing comprehensibility over accuracy in network design may just be a form of advanced technophobia, and the inverse merely a symptom of an unjustified trust in numbers, but our paper remains unaffected either way. With our main objective being the identification of possible sequences of interest, interpretability becomes less important than good prediction accuracy. According to [180], decreasing model complexity to make it more comprehensible for humans limits to using simple models that do not suffice for complicated tasks (e.g. language sentiment analysis). Furthermore, since we only aim to identify patterns within single trials and not to compare across trials, we do not need to worry about the reproducibility across videos (as e.g. in [82], “Extrapolating predictions from limited data”) either. Nonetheless, concerns regarding the stability of results within trials, especially after changing meta parameters, can still be sustained and will be addressed as part of our evaluation. The detected anomalies should be relatively consistent after every full execution of the analysis pipeline. However, the potential net benefits of machine learning (e.g. flexibility, and robustness to noise) often outweigh the concerns, especially when risks are known and can be taken into account.

Our main contributions to the available literature are twofold. Firstly, we provide a usable example (and code) for the automatic identification and 2D localization of unusual events in long, densely populated videos; an evaluation of different sizes and types of machine learning models for time-series prediction compared to a more naïve approach. Secondly, we provide a basic description of novel behavior of *C. cyphergaster* as part of an example application of the method. While we focus on termite behavior throughout this paper, the general flexibility of machine learning approaches should allow for an easy adaptation of our methods to different types of time-series data.
In the following sections we will first describe the materials and hardware we used, followed by a step-by-step description of the methods and their evaluation, concluding with a discussion of the results and potential future applications.

2.3 Materials and methods

2.3.1 Video recordings and study site

Termite colonies were collected in November of 2019 and 2020 from a Cerrado area [178] near the municipality of Divinópolis (20°10'19.4"S, 44°49'21.6"W), Southeast Brazil. The recording setup, described in detail in Hugo et al. (2021), was composed of a circular arena connected to a nest box, where whole colonies of termites were kept under suitable conditions of humidity and temperature [44, 68] during behavioral recording. Each arena had a diameter of 60cm (circa 150 body-lengths of the typical individual in our trials). The experiment was designed to investigate the initial exploration of foraging bouts [151], when many individuals exit the nest to start searching for food. Termites could enter the arena by emerging from a hole in its center, which was directly connected to the nest box. They were then allowed to move freely within the space of the arena. All videos (see Table 1) were recorded in high resolution (4K ~58px/mm @ 25fps) with light coming from a LED ring light attached near the camera lens. Markers printed onto the arena floor were used to calibrate the unit conversion between pixels and centimeters.

Since nests were directly collected from nature, they consisted of varying numbers of individuals. As an estimate for relative colony size, the nest volume can be used (calculated as in [49], listed in Table 1). Likely only a fraction of all individuals in the nest emerged during the trials, so an exact total count cannot be given. However, after filtering out all trajectory pieces shorter than 1s (to avoid noise), the number of simultaneously visible individuals from trajectory data of N20 averages 2456±617.4 (with a maximum of 3193), and 838.5±409.2 (with a maximum of 1728) in N07. N20 is thus likely a larger colony than N07 (both according to volume and tracked individuals). The size of individuals on screen varies between 99.2±28.8px (this is the actual number of pixels that make up the individual, not its bounding box).

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>resolution (px)</th>
<th>frames/s</th>
<th>length (h)</th>
<th>nest volume (L)</th>
<th>usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video 1</td>
<td>N20</td>
<td>3840 × 2160</td>
<td>25</td>
<td>02:20:01</td>
<td>20.2</td>
<td>development, analysis</td>
</tr>
<tr>
<td>Video 2</td>
<td>N07</td>
<td>3840 × 2160</td>
<td>25</td>
<td>02:05:19</td>
<td>15.5</td>
<td>analysis</td>
</tr>
</tbody>
</table>
Table 1: A list of the videos used in this paper. Both have been recorded by H. Hugo in Brazil and show a large number of freely behaving C. cyphergaster (typically between 800 and 3000).

2.3.2 Analysis Setup (Hardware & Software)

Only a single, modest-specification computer is required to make this method work (e.g. NVIDIA GeForce GTX 1080 Ti, and an i5 CPU; or similar), but we recommend at least 32GB of RAM for large datasets — alternatively the analysis has to be split into multiple parts.

Weights&Biases [19], an online platform for organising machine learning results, was used to control the hyper-parameter optimization and prime our later analysis with preliminary statistics and graphs. The evaluation of different network layouts and the hyper-parameter analysis was performed using a distributed computing platform (Google Colaboratory) using multiple GPUs and TPUs (tensor processing units).

For tracking we employed TRex ([222], Chapter 1); all analysis code is written in Python 3.8 using the latest TensorFlow versions 2.5.0/2.6.0 with GPU support, which are used for network training, generating predictions, and data analysis.

2.3.3 Preparing the dataset

Our biggest challenge was the large amount of data — so large that even just holding the tracking data for one video in memory was impossible. Multiple metrics for thousands of individuals are accumulated during tracking, most of which are not required here. Since the goal is to detect broad-scale, rather than individual-level effects, we can apply lossy compression to the exported data, reducing all accumulated information to one small 60x60x2 array per frame. Since these arrays are saved in an image format, and each element corresponds to a unique spatial location of the original image, we will refer to these arrays as “images” and to every 60x60 slice (with 2 channels) as “pixels” from here on. Every pixel in each of the heatmap images represents one 64x64px cell in a uniform two-channel grid overlaid on top of the original footage; it stores the sum of speeds in the first and the number of individuals in its second dimension. The data is accumulated from a sliding-window of time-steps surrounding the focal frame (150 frames), so that each pixel additionally summarizes information from multiple frames. As a final step, we removed inactive parts of the heatmap image that are never occupied by any individuals, reducing the heatmap image size to 51x34px (Video 1). Calculated for 200,000 frames, this amounts to $NK \times 1.666\text{ bytes}$ when saving $K \geq 2$ features for $N$ individuals separately, or, in our case, a constant $5.555e9$ bytes for the heatmap images. Using heatmaps in-
creases memory load with low numbers of individuals, but this relationship inverts quickly when $KN \geq 3469$ (as an example: storing X and Y for $\geq 1740$ individuals, which is exceeded in both of our videos). More importantly, it also transforms the data into a more streamlined format, suitably standardised for use as an input to an artificial neural network with fixed input dimensions. In addition, we hoped that using a simple heatmap image format would (1) make the whole process easier to generalise to other video-related problems, and (2) help to naturally maintain the spatial relationships between values.

Primer tests suggested that increasing the heatmap resolution from 60px to 120px only “distracts” the algorithm with smaller events (noise, which we would likely filter out later) and quadruples the memory-load. Of course, it also quadruples the spatial resolution of the detections, but not the process. We decided to employ the lower resolution for the remainder of this paper.

The first step is to extract the required tracking data from raw videos and to store it in 10 separate files (split to fit the RAM in our machine). The manual work required is minimal, since this process can easily be automated using a batch script that performs the following steps in sequence for each video (for parameter choices, see B.1):

- Preprocess the full video using TGrabs, for compatibility with TRex [222].
- The video is long and has periods with more than 4000 individuals visible at the same time, so not all of the tracking data can be held in memory on most computers. It was split into 10 ranges of approx. 20500 frames (~14min) each. These ranges are passed as a parameter (analysis_range) to TRex, which is run separately for each range. Each of the parts has an overlap of at least 500 frames with the next part to remove edge effects caused by the 150 frame sliding-window that runs out of frames there.
- Track each of the 10 ranges using TRex, which calculates the XY positions of all visible individuals and exports them to a binary format (.NPZ).

2.3.4 Normalization and cleaning

Before comparing different methods we need to ensure that all of the data (and all channels of the data) are within (0-1) - especially because speed and density data (the two channels) are generally within entirely different ranges. As mentioned above, all heatmaps are cropped to remove inactive pixels around the edges — and all pixels below the $P_1$ percentile are set to zero, ensuring that noisy “mean” speeds of single samples are not misinterpreted. A percentile $P_q$ is defined as the smallest
value that is larger than q percent of the data. TRex exports the speed channel as a sum of speeds, so all non-zero pixels need to be divided by the density to get the mean.

To get comparable metrics across runs, all heatmap images need to be normalized to the same effect. Given real measurements $r_{ij}(t) \rightarrow \mathbb{R}^c$ with multiple channels $c$ and the two heatmap image dimensions $i$ and $j$, we sample $r$ in regular intervals of $t$ and save this sequence of frame indexes (of 9752 frames) as $G$. The sequence is smoothed temporally, so that each sample is the average of a number of surrounding samples, using the moving average

$$r(t) = m_r^{(u)}(t) = \frac{1}{n} \sum_{i=0}^{n} r(t - n + i)$$

with $n = 5$, of which we take the per-channel maximum vector $\vec{\beta} = \max_{c \in \mathbb{G}} \vec{r}_c(t)$ and minimum ($\vec{\alpha}$). All samples, also for $t \not\in G$, used in the following sections are assumed to be normalized and confined to a value range of $[0, 1]$ by:

$$R_{ij}(t) = \min \left( 1, \max \left( 0, \frac{r_{ij}(t) - \alpha_c}{\beta_c - \alpha_c} \right) \right) = \text{clip} \left( \frac{r_{ij}(t) - \alpha_c}{\beta_c - \alpha_c}, 0, 1 \right).$$
2.3.5 Simple anomaly detection (Computer Vision)

A baseline is required for assessing whether the machine learning based approach is worth the additional effort. Looking for simpler alternatives, an algorithm for detecting changes in a stream of 2D heatmap images is easily defined by interpreting the sequence of images as a (discrete) function and calculating its first per-pixel derivative. This has the advantage of ignoring regions where values are consistently high for a prolonged period of time, detecting only the changes in speed or density. The discrete difference is defined as $R'_{ij}(t) = R_{ij}(t) - R_{ij}(t - 1)$, omitting $\Delta t = 1$ which would make it a derivative. However, this turned out to be very noisy and thus uninformative, likely because the derivative is calculated per pixel and does not take neighboring pixels into account. Smoothing the images before calculating $R'$ can help with this problem by sacrificing accuracy to add context. Alternatively, increasing the step size in a more general form $R'_{ij}(t) = R_{ij}(t) - R_{ij}(t - k)$, for $k \gg 1$ produces similar results (note that division by a constant factor $\Delta t = k$ would be irrelevant in a relative comparison). Here, the last $k$ images in the video cannot be evaluated, which is acceptable for long sequences or small $k$, but not so for shorter video clips. In general, having to choose a particular time-step $k$ in the first place is problematic for generalisation and excluding part of the video is suboptimal. The following is a viable alternative, which is also closer in principle to the error calculation in the next sections:

1. Smooth the image sequence (heavily) in the time domain with a moving average window of a quarter of the number of total images in $G$.

2. subtract the 2D running average from each of the images.

Subtracting the smoothed sequence produces an effect similar to calculating a derivative, where consistently high values would become part of the “background” and later deviations may still be detected. Parameter choices are still important, but have a less direct influence on the result, apart from the downside that both at the end and at the beginning of the video — while not being excluded necessarily — this procedure has a tendency to produce anomalies, simply because the smoothing window is necessarily of a different size there. In effect, this means that they would have to be excluded, which would especially a downside for studies interested in e.g. the formation of trails in the beginning of the video.

$$
\delta_{ij}^{\text{simple}}(t) = \omega \max_{c \in \text{channels}} \left\{ \left( R_{jk}(t) - m_{R_{ij}}^{(0.25+|G|)}(t) \right)^2 \right\}, \text{ where } \omega = 3.5
$$
denotes the entire (slightly modified) mean-square-error (or MSE) function for this method. An adjustable scaling factor $\omega$ is applied and the last channel ($c$) is pruned by taking its maximum. All other image dimensions, which will later be used to spatially locate events, are retained throughout these calculations.

### 2.3.6 Anomaly detection using sequence prediction (machine learning)

While the simple approach described above can capture much of the variation in the sequence of images, it has its limitations. As mentioned above, there is only a weak and artificial relationship between neighboring pixels — which is wrong because the probability for an empty part of the arena to be populated, in the near future, surely must be related to the number of close-by individuals. Deep learning and especially CNNs are made for this kind of task, intrinsically viewing pixels within their spatial (and temporal) context, and introducing additional non-linearities that may help capture more complex relationships. They also bare the possibility of transfer learning, meaning that a trained network can be applied to a previously unseen dataset without the need for manual adjustments.

The general principle we used here is to teach a model to predict future states of the arena based on a short input sequence of length $L_{\text{input}}$, and then calculate the error between predictions and real measurements to find anomalies. Input sequences for the training are randomly selected from the heatmap time-series along with the associated “real” output: Both the speed- and density-channel of a single future image at the distance $L_{\text{input}} + \lambda$, where $\lambda$ is the lag (adjustable per sample). We evaluated multiple values for both lag and the input vector length in our hyper-parameter analysis (see below). While all frames within a sequence are equidistant, the uniform distance between them is another hyper-parameter ($\text{playback\_step}$).

Analogous to the simple method above, we define the two-dimensional error of the real measurement/image $R_{ijc}(t)$ and the prediction $P_{ijc}(t)$ at time $t$, and for time $t + L_{\text{input}} + \lambda$, as:

$$
\delta_{ij}(t) = \omega \max_{c \in \text{channels}} \left\{ \left( R_{ijc}(t + L_{\text{input}} + \lambda) - P_{ijc}(t) \right)^2 \right\}, \quad \omega = 3.5
$$

where $\lambda$ is the lag hyper-parameter that determines how far into the future the network predicts.

Selecting a suitable layout for a network to generate these predictions is difficult, but the search can be narrowed down based on the particular problem: Here, we require a layout that works on image sequences, conserves both temporal and spatial context, and produces a single output image. Hyper-parameters, however, are harder to choose. There are many available tools for efficient model and parameter selection (e.g. [19, 144, 232]).
To find the model layout and hyper-parameters best suited for our specific task, we performed sweeps of over 2000 different layout and parameter combinations (using Weights&Biases [19]). Each run was allowed to choose from an array of possible values for each parameter:

- **lag** $\lambda$ (not including the length of the input sequence): 0, 100, 300, 500, 1000, 2000, 3000
- **playback_step**: 1, 3, 6
- **input sequence length** $L_{\text{input}}$: 1, 3, 5.

The network layout for each run differs, but is always based on the structure presented in Figure 2 with the following modifications:

- the type of primary layer (ConvLSTM2D or Conv3D, both of which retain the temporal and spatial dimensions of the input sequence)
- different activation functions (ReLU, LeakyReLU [138], PReLU [95] and tanh)
- different numbers of filters/units per layer (32-512 filters)
- different numbers of primary layers (2, 3, 4 and 5).

We used the ADAM optimizer [118] for all trials, and each test was terminated after a maximum duration of 35 epochs, or in case of invalid loss values. Having produced over 2500 trained models, it is unfeasible to review predictions of every single one manually, so we need to introduce an automatic scoring system that finds the best model for the task. It would be possible to compare them based on the best val_loss they have achieved within 35 epochs, but perfect predictions may not be beneficial for detecting anomalies because the difference would be essentially zero for all $t$. Instead, we define a more task-specific detection score $S$ that we calculate, based on the manually annotated examples from Table 2, as follows.

### 2.3.7 Manual labels

Manual labelling is the basic building block for the later scoring of both the simple and machine learning based detections, as described in the next section. We tabulate annotations for five distinct anomalies/events in Table 2, listing the start time, the magnitude of the effect (rough estimate of recruited individuals), and an approximate XY centroid position. A new row was added for every few seconds of original video, until the individuals returned to their perceived normal behavior.

The most noticeable event, at 1:10:40h, starts on the bottom-right and eventually spreads to the entirety of the arena (see Figure 1). It is the first event that we were able to find without being aided by
the machine. Going back and forth in the video at different speeds, as opposed to passively watching, and the large magnitude of the event helped to make this event especially obvious to the naked eye.

As can be seen in Figure 1, the increased activity between clusters dissolves them quickly, while the wave continues to travel along the right outer edge towards the top-left of the arena. By just observing the density channel, however, events such as this are often hard to see. This highlights the importance of the speed channel, which motivates its inclusion in our dataset despite the significant additional computation time needed for tracking (see also B.3).

2.3.8 Scoring

First, Gaussian smoothing is applied to the error function (denoted here simply as a general \( \delta(t) \) that could be either the simple or machine learning method), and all of its values are clipped to ensure they remain within \([0, 1]\):

\[
\Delta(t) = \text{clip} \{ \gamma_{11}^{\sigma=2}(\delta(t)), 0, 1 \} \text{ indexable by } ij,
\]

where \( \gamma_{11}^{\sigma=2}(X) \) is the Gaussian blur function that blurs the matrix values in \( X \) with a symmetrical kernel size of 11 and \( \sigma = 2 \). For the score calculation, this function is additionally thresholded with a fixed brightness value of 0.235 (or the fraction 60/255), setting all values below this threshold to zero:

\[
\hat{\Delta}(t) = \begin{cases} 
\text{clip} \{ \gamma_{11}^{\sigma=2}(\delta(t)), 0, 1 \} & \text{where it is } \geq 0.235 \\
0 & \text{otherwise}
\end{cases}
\]

The set \( G \) of manually confirmed anomalies from Table 2 is where we hope the error \( \hat{\Delta}(t) \) is high, while it should be as low as possible for all other \( t \) (to punish models that e.g. only predict one solid color throughout and everything is an anomaly). All \( t \in G \) are then used to calculate the final scalar
score for a particular model by rewarding high values close to the centroid (X and Y in Table 2) and punishing high values outside the manually tracked center:

\[ m_{ij}(t) = \begin{cases} 0 & \text{if } \sqrt{(i - c_x(t))^2 + (j - c_y(t))^2} \geq 8 \times \max(1, \rho(t)/320) \\ 1 & \text{otherwise} \end{cases} \]

which is defined at all \( t \in G \). The radius is calculated the approximate number of recruited individuals \( \rho(t) \) (also in Table 2), which is normalized by a manually chosen constant (320, calibrated by comparing the resulting radii with the raw footage).

Using the masking function \( (m_{ij}) \), the scalar score \( S \) is calculated as a ratio of (rewarded) “inside” pixels \( S_{\text{in}} \) and (punished) “outside” pixels \( S_{\text{out}} \), where

\[ S_{\text{in}} = \frac{1 + \sum_{ij} m_{ij}(t) \Delta_{ij}(t)}{1 + \sum_{ij} m_{ij}(t)}, \quad \text{and} \quad S_{\text{out}} = \frac{1 + \sum_{ij} (1 - m_{ij}(t)) \Delta_{ij}(t)}{1 + \sum_{ij} (1 - m_{ij}(t))}, \quad t \in G. \]

We can now calculate the final scalar score as \( S = \frac{S_{\text{in}}}{S_{\text{out}}} \).

<table>
<thead>
<tr>
<th>event</th>
<th>start (time)</th>
<th>frame</th>
<th>centroid</th>
<th>X (px)</th>
<th>Y (px)</th>
<th>recruited (rough guess)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3m 23s</td>
<td>5075</td>
<td>1320x1388</td>
<td>1320</td>
<td>1388</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3m 25s</td>
<td>5125</td>
<td>1200x1572</td>
<td>1200</td>
<td>1572</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>3m 27s</td>
<td>5175</td>
<td>627x928</td>
<td>627</td>
<td>928</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>3m 50s</td>
<td>5750</td>
<td>N/A</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>14m 53s</td>
<td>22325</td>
<td>1796x1348</td>
<td>1796</td>
<td>1348</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>14m 55s</td>
<td>22375</td>
<td>1700x1548</td>
<td>1700</td>
<td>1548</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>14m 56s</td>
<td>22400</td>
<td>1836x1708</td>
<td>1836</td>
<td>1708</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>15m</td>
<td>22500</td>
<td>1320x1648</td>
<td>1320</td>
<td>1648</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>15m 8s</td>
<td>22700</td>
<td>1540x1128</td>
<td>1540</td>
<td>1128</td>
<td>200</td>
</tr>
<tr>
<td>2</td>
<td>15m 9s</td>
<td>22725</td>
<td>900x1328</td>
<td>900</td>
<td>1328</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>15m 20s</td>
<td>23000</td>
<td>860x900</td>
<td>860</td>
<td>900</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>15m 27s</td>
<td>23175</td>
<td>860x600</td>
<td>860</td>
<td>600</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>15m 31s</td>
<td>23275</td>
<td>270x733</td>
<td>270</td>
<td>733</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>15m 38s</td>
<td>23450</td>
<td>780x960</td>
<td>780</td>
<td>960</td>
<td>1500</td>
</tr>
<tr>
<td>2</td>
<td>15m 45s</td>
<td>23625</td>
<td>780x960</td>
<td>780</td>
<td>960</td>
<td>1500</td>
</tr>
<tr>
<td>2</td>
<td>15m 59s</td>
<td>23975</td>
<td>500x1720</td>
<td>500</td>
<td>1720</td>
<td>1000</td>
</tr>
</tbody>
</table>
Table 2: The record of manually verified events. An event starts if (at any playback speed) the observer notices a sudden increase in activity anywhere in the arena. It ends when the individuals consistently move at “normal” speeds (speeds prior the event) again for at least 10 seconds. The columns X/Y indicate (roughly), where the centroid of the event is located.
2.3.9 Narrowing down the first responders

Since cropping factors and the grid resolution are known, going from heatmap coordinates back to video coordinates is simple: \( C \left( \vec{p}_{\text{grid}} + \vec{p}_{\text{offset}} \right) \Rightarrow \vec{p}_{\text{video}} \) where \( \vec{p}_{\text{grid}} \) is the 2D position in the grid, \( \vec{p}_{\text{offset}} \) is the offset introduced by cropping and \( C \) is the cell size (e.g. 64px in our case). After generating the results for the adjusted MSE in \( \Delta(t) \), one can use the resulting 2D error images to locate events in space. Each part of the image that is above a chosen threshold, such as in \( \hat{\Delta}(t) \), could be interpreted as the centroid of a separate event. By adding the 2D component, it is possible to find multiple separate epicenters at the same time — these may still have the same cause, but can be tracked individually. It may not be enough to sample \( t \in G \) for this purpose, since \( G \) makes up less than 5% of the full dataset and the temporal resolution is very low. Instead, we choose particular times in the mean-MSE graph (see e.g. Figure 5) with high peaks, and generate model predictions for every frame within the given ranges. To help the tracking software’s segmentation algorithm, we increased the contrast by decorating \( \Delta(t) \) to form \( \hat{\Delta}(t) = 10 \cdot \Delta(t)^4 \). The resulting sequence of \( \hat{\Delta}(t) \) images is then put through another tracking loop of (i) conversion in TGrabs and (ii) tracking in TRex, with a threshold that is chosen so that only the peaks remain.

If, in step 2 of Preparing the dataset, individual X and Y positions were also exported along with the original heatmaps, then these can now be correlated with the exact time and place of the tracked objects in the heatmap images according to the above coordinate conversion between \( \vec{p}_{\text{video}} \) and \( \vec{p}_{\text{grid}} \).

2.4 Results

2.4.1 Hyper-parameter analysis

The large number of successfully scored trials (2059) allows for a quantitative assessment of the importance of each variable with respect to the detection score \( S \). A score was calculated for all runs that achieved a best \( \text{val_loss} \neq \infty \), the best epoch is not the first epoch, and did not crash. Infinities and other numerical errors may arise during training due to the dying ReLU problem [136] and the exploding gradient problem [172], both of which are unrecoverable. LeakyReLU [138] (multiply a very small number to negative values, but keep them) and PReLU [95] (Parametric ReLU, where the amount of “leakiness” is learned) are specifically designed to solve the dying ReLU problem. The authors also describe improved loss compared to ReLU, so we expected a positive correlation here as well. Runs with invalid \( \text{val_loss} \) (272), consisting of all activation types to similar degrees; and runs that crashed (170) were filtered out.

68
The influence of all other parameters (independent variables) on the score (the dependent variable) can be measured, for example, in terms of linear correlation. While linear correlation can confirm or reject the hypothesis that “a linear relationship between two variables exists” (not causation, [209]), it does not capture complex relationships between multiple variables well, and has trouble with unnormalized data and outliers [53].

<table>
<thead>
<tr>
<th>variable</th>
<th>pearson</th>
<th>spearman</th>
<th>importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>best_val_loss</td>
<td>-0.112***</td>
<td>0.085***</td>
<td>0.925</td>
</tr>
<tr>
<td>lag (λ)</td>
<td>0.289***</td>
<td>0.445***</td>
<td>0.858</td>
</tr>
<tr>
<td>activation.PReLU</td>
<td>0.254***</td>
<td>0.268***</td>
<td>0.056</td>
</tr>
<tr>
<td>playback_step</td>
<td>-0.03</td>
<td>-0.072**</td>
<td>0.033</td>
</tr>
<tr>
<td>dataset_offset</td>
<td>-0.191***</td>
<td>-0.239***</td>
<td>0.01</td>
</tr>
<tr>
<td>input_length</td>
<td>0.097***</td>
<td>0.077***</td>
<td>0.008</td>
</tr>
<tr>
<td>batch_size</td>
<td>0.043</td>
<td>0.071**</td>
<td>0.006</td>
</tr>
<tr>
<td>num_layers</td>
<td>0.141***</td>
<td>0.136***</td>
<td>0.006</td>
</tr>
<tr>
<td>trainable_count</td>
<td>0.16***</td>
<td>0.271***</td>
<td>0.004</td>
</tr>
<tr>
<td>number_samples</td>
<td>0.163***</td>
<td>0.19***</td>
<td>0.001</td>
</tr>
<tr>
<td>dataset_percentage</td>
<td>0.231***</td>
<td>0.243***</td>
<td>0.001</td>
</tr>
<tr>
<td>primary_layer_type.ConvLSTM2D</td>
<td>0.033</td>
<td>0.009</td>
<td>-0.001</td>
</tr>
<tr>
<td>activation.ReLU</td>
<td>-0.236***</td>
<td>-0.23***</td>
<td>0.001</td>
</tr>
<tr>
<td>activation.LeakyReLU</td>
<td>0.014</td>
<td>-0.002</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Table 3: The correlation coefficients as measured by a RandomForest (N=1000), and permutation importance measures of all hyper-parameters evaluated in our trials. Significance is indicated according to the APA format (*P ≤ 0.05, **P ≤ 0.01, ***P ≤ 0.001).

Random forests (RF) are a model agnostic assortment of random decision trees, that predict outcomes (e.g. score) based on random subsets of features [25]. They do not need to make the same assumptions about the input data, which is preferable for complex inputs, and additionally “provide an assessment of variable importance” [88]. “Importance” expresses a (potentially non-linear) measure of how influential a certain feature is and is calculated as the decrease in prediction quality when a feature’s values are repeatedly permuted in a random manner — the permutation feature importance [25].

An additional metric we use below, the spearman rank [200], is a linear correlation coefficient (a
Pearson correlation coefficient \([164]\)) between ranked variables, modifying the data as to (similarly to RF) not require linearity and yielding a non-parametric assessment of monotonicity. Interestingly, when measured together, a Spearman coefficient for two variables that is much larger than their Pearson coefficient indicates, if significant, a non-linear monotonous relationship between them \([53, 55]\). This is the case for many variables tested here (see Table 3). We trained a RF \((N=1000)\) to predict the detection score \(S\) based on all other metrics (see Figure 5) and calculated their Pearson and Spearman rank correlation coefficients with respect to the score. The results show that \(\text{best\_val\_loss}\) has the highest importance, closely followed by \(\lambda\) — with lower importance, but the largest correlation coefficients. As stated above, the relation of the respective two correlation measures for \(\lambda\) and \(\text{best\_val\_loss}\) indicates complex relationships, which can be confirmed visually by looking at their columns in the parallel coordinates plot (Figure 3); small lag seems to be strongly associated with lower scores (darker colors) and larger lag with higher scores (lighter colors), but \(\lambda = 1000\) appears to be optimal (scores below or beyond 1000 are worse).

There also seems to be a sweet-spot for \(\text{best\_val\_loss}\), between 0.06 and 0.12, where brighter colors in the center dominate more than in the extremes, suggesting that a slight underfit may yield better results than any of the extremes. An overfit \((\text{val\_loss} < 0.06)\) can achieve such low loss values by almost perfectly predicting future time-steps, even where interesting events take place, which makes it unusable for anomaly detection (since error should be \(\text{high}\) during events). Prediction outputs of models with \(\text{val\_loss} > 0.12\) have low mean standard deviation \((0.0728\sigma)\) for \(t \in G\), while the higher scores for \(\text{val\_loss} \leq 0.06\) have higher mean standard deviation \((0.1626\sigma)\). Outputs from high-loss models often still approximate the correct output somewhat, but especially models with \(\text{best\_val\_loss} > 0.2\) often produce static images or even solid colors (hence the low standard deviation). The positive correlation with lag may be related to this, since predictions further into the future are likely more difficult — fewer and flatter global optima in the loss landscape can make local ones more attractive for the gradient descent algorithm.

Initial expectations that PReLU activation could be beneficial were (barely) confirmed by PReLU’s positive correlation with the score, the negative correlation of other activations, as well as its being of the third-highest importance (although there is a large distance between second and third place). Comparing the proportion of PReLU activations in the top vs. the bottom 10% of scores (Figure 4) shows that of the higher scores, more have been achieved with PReLU than is the case in low score regimes — suggesting that PReLU may be the best choice.

The number of trainable parameters, the number of layers, and the type of primary layers do not appear to have had a large effect on the score. While LSTM-based models have performed slightly bet-
Figure 3: A parallel coordinates plot of (nearly) all evaluated hyper-parameters. The missing parameters here are the “dataset offset” (determining the minimum frame for training samples), and the number of samples (more samples were better).

Figure 4: The proportion of activations in the lower and higher score percentiles (correcting for the unequal overall proportion of activations), showing that PReLU’s proportion in the top 10% of scores is larger than in the bottom 10%.

...over all, their (already low) correlation coefficients have low significance, and did not have much impact on the results. The values of other parameters, such as batch_size, were similarly inconsequential.

2.4.2 Comparing the methods: Simple vs. machine learning based

The simple approach detects all manually annotated events (see Figure 5), and achieved a score of $S = 3.45$. This is less than the score of the best machine learning model at $S = 4.19$, but certainly within the top 1% of networks ($S > P_{99} = 3.12$), suggesting little difference in detection rate and visual appearance (due to the nature of the score calculation) when compared to top scoring models. The only distinct dissimilarities are that (i) the ML MSE curve is missing the large bump at the end and beginning of the video and, interestingly, (ii) has an additional second peak before the large one at around frame 100,000 (close to event 4 in Table 2). This short, smaller peak right before the start of the annotated event corresponds to a ring of high error values surrounding the entire arena (see Figure 6).
Figure 5: Mean-$\Delta(t)$ of implementations for the “simple” method (top, orange) and of a single model that achieved the highest score $S$ (top, red), and of the average output of all scored models (bottom). These curves were obtained by subtracting $\Delta(t)$’s sliding-window average from itself, temporal smoothing with a uniform filter of size $N=500$, and averaging across the image dimensions to get a one-dimensional curve.

Figure 6: On the left (frames 99640-100411), showing raw heatmap data before a small peak in the mean error $\Delta(t)$. On the right, 2D mean error during the peak (frames 103696-104467). The outer ring of increased error values on the right is slightly harder to see without moving images, but is brighter when contrasted with the left image.

However, it is difficult to make a direct connection to the raw footage and it shall not be over-interpreted. Many clusters of individuals are framing the arena center, which appear to have slightly higher activity levels right before event 4, which is potentially due to some individuals breaking away from their clusters — this could give some explanation for the peak. One has to mention also that this additional peak is (suspiciously) close to a transition from one part to another, from when the video was split into 10 parts.

The small peak occurs between frames 103750-104750, while the transition to the next part starts in frame 104515. It is possible that, despite our efforts, there is a slight inconsistency in the data due to tracking discontinuity which the model detects. There was an occasion like this during development: A (slight) misalignment during the concatenation of video parts did not produce any obvious
Figure 7: Demonstrating the sensitivity of the simple/machine learning methods to switching one (randomly chosen) data-point with another at a different random position. The Y axis is the difference between methods (simple − ML), when subtracting one data point. Curves have been normalized prior to this by subtracting their mean and dividing by their standard deviation. Both methods show two additional peaks compared to their baseline, but the machine learning based approach is more sensitive.

artefacts in the real measurements (which we checked for), but caused large MSE values until the bug was fixed. This can be reproduced by switching \( R_{ij}(t) \) at two arbitrary \( t \) with each other, and observing the effect this has on the mean-MSE curve. Here, in evaluating MSE spikes for both the simple and the machine learning method, we find that the latter is more sensitive by more than 229% on average (for \( N = 4782 \) random single-frame switches within \( G \), see Figure 7), which may hint at why the mistake only became obvious once we calculated the (more sensitive) machine learning based error values.

2.4.3 Inspecting detections and finding first responders

The trained machine learning models maintain a relatively low error level throughout trials (see Figure 5), but higher levels in sections of the video that contain sudden speed or density changes, such as the manually labelled ones (see Table 2). Since 50% of the dataset consists of speed values, it is not surprising that most detections seem to have been caused by sudden local speed changes. Event 4 (depicted in Figure 1) is an extreme example of this, where a wave of suddenly increased speed spreads through the entire arena within 3 seconds, dissolving most established clusters in the process. It takes about five minutes for the individuals to slow down again, before ultimately reestablishing the clusters. The magnitude of this disturbance makes it easier to find by hand, but other events (such as event 1 in Table 2) last only for a few seconds and do not spread to many other individuals in the surrounding area. Most showed less recruitment than event 4, which makes them almost invisible to human observers — who would be charged with the task to manually integrate speeds of thousands
Figure 8: Potentially affected individuals for the manually annotated event 3 (see Table 2). Each dot represents one individual in one frame. Lighter (or more yellow) colors are more recent in time, and grey dots are not part of the event (temporally and spatially). \(N\) gives an estimate how many individuals were inside the event’s radius based on the number of affected individual IDs (trajectory pieces are assigned an ID, which does not accurately represent individuals). The radius of influence is calculated per frame \(t\), based on the two-dimensional \(\Delta(t)\) output (using track_threshold=10 in TRex). (a) A small sub-event preceding the bigger one in (b), which is the largest “epicenter” that was tracked within this range, (c) a smaller event immediately following (b).

of tiny individuals simultaneously. Most trained models, however, successfully detected every annotated event to varying degrees, along with others that were not part of the set of manual annotations (Figure 5, bottom). Every major event was also manually checked to ensure it is visually identifiable and not merely the product of noise. The detection rate of the highest and lowest scoring runs was assessed by comparing to the \(\Delta(t)\) images for manual annotations to confirm the score computation is functional and tracked the events properly — this fits with what we can tell from the mean curve and percentiles in Figure 5, showing that most runs agree on most peaks.

Using the method described in 2.3.9 Narrowing down the first responders (see also Figure 8), we could successfully identify event 3 and connect MSE peaks to the original trajectory data. The center of the event moved at an average speed of \(0.07 \pm 0.14\) cm/s, while the mean speed of the individuals involved in the event was \(0.94 \pm 0.97\) cm/s (Figure 9 shows kernel density estimates for both the speed of individuals and the event). We can see that, according to \(\overline{\Delta}(t)\), the largest peak of event 3 starts about 18s earlier (frame 70179) than was manually annotated (70625) and the whole event might actually consist of three separate bursts closely following one another that were not labelled. The difference in time could be explained by the 150 frames (6s) window with which the heatmaps were generated, or simply by potential inconsistencies of the human observer during the labelling process. The maximum recruitment, however, that was roughly approximated manually (1000) for event 3 is in a similar range as the automatically obtained value of 1519 (with a large margin of error, given that this number is also an approximation).

Using every successfully scored model, we also tested the same procedure on Video 2 to see if they would be able to transfer what they had learned in one video to a video it has never seen before. The simple method, in comparison, is always specific to the input video and — since we showed its via-
Figure 9: Kernel density estimates for the speed of individuals involved in event 3, and the speed the event’s center (see also Figure 8). The speed of individuals varies greatly, with an upper 99th percentile of 4.2 cm/s, whereas the equivalent value for the event’s center was measured at 0.72 cm/s.

Figure 10: Using models that were trained on Video 1 on data that comes from Video 2, this compares detections by the simple method (which is always specific to the video it is used on) and the machine learning method. This figure is otherwise entirely analogous to Figure 5.

bility in the first video — we were able to use it here as a baseline for Video 2. Firstly, we observe that the beginning and end are problematic for the simple method in both videos, which is “sacrificed” during the smoothing process. Our results (see Figure 10) show also that the ML mean-MSE curves, despite being noisier than for the video they were trained on (larger differences between percentiles, compare Figure 5), follow the results of the simple method. This suggests that a model trained on one video can be used for other videos that are similar in kind without retraining, which is beneficial for non-parametric application in the batch processing of many videos.
2.5 Discussion

We evaluated various parts of potential analysis pipelines for high throughput in difficult datasets, such as data from large nests of freely behaving termites \((C. cyphergaster)\). We ...

- showed an example of lossy, yet sufficiently informative, compression that helps working with very large datasets and many individuals (heatmaps)
- devised two methods for anomaly detection in sequences of this compressed tracking data using (i) a simple classical approach and (ii) sequence prediction using modern machine learning methods, and evaluated their respective advantages
- compared various hyper-parameters and different model architectures, to find the models that performed best in the given time-series prediction scenario
- successfully used the best-scoring model for analysing a different video that it was not trained on (transfer learning)
- successfully tracked the peaks of detected events, connecting the event detections back to the original trajectory data to gain information on potential first responders
- described a set of interesting events in \(C. cyphergaster\) behavior.

Our results show that it is possible to detect anomalies in densely populated videos, even while compressing the complex individual tracking information into a minimal heatmap format. Using advanced machine learning methods does not significantly improve upon simpler methods, which do not require any training, but it may be more sensitive to deviations that classical approaches do not detect. Machine learning methods can also get around other problems that classical approaches might have, for example that fewer images at the end or beginning of the video have to be excluded — these had to be cut for the simpler approach because smoothing is required.

Anomaly detection based on sequence prediction, in a dataset of the type we used, is best done by (i) choosing a medium to large lag (500-2000), (i) slightly underfitting the dataset and thus using a shallow model with few parameters, (ii) choosing PReLU activation throughout the model. While LSTM models may perform slightly better overall, and occupy a larger proportion of the higher score regimes, they are not a significant improvement over Conv3D layers for this specific problem. The complex relationship between \(val\_loss\) and the score, as a post-hoc quality estimate, motivates searching for a better loss function, which is directly proportional to the scoring function. This is especially true since improving loss values beyond 0.06 worsens the quality of detection results. Using

76
a less complex model with fewer parameters (or attempting to predict further into the future) helps mitigating the symptoms, as our hyper-parameter analysis suggests, but does not ultimately solve the underlying problem.

Investigating the detected events, they appear to be much more frequent than we initially thought (at intervals of every ~15min, see Figure 5), with smaller events occurring even more frequently (every few minutes). They also periodically appear in another video, which we did not use for the development of our method (Figure 10). We found no obvious cause for the phenomenon of sudden activity bursts in the video, but there is no way to exclude sound and other vibrations as the source of such behavior without (at least) an audio stream. However, the time-scales of the events and their locality do at least leave room for other interpretations — a loud sound, and other kinds of strong vibrations should be expected to alarm individuals across the entirety of the arena simultaneously and not just locally. This is especially interesting, since the mean speed of event 3 (see Figure 8) was just 0.07±0.14cm/s, while the mean speed of the individuals involved in the event was 0.94±0.97cm/s. Thus, the event moves slowly through the arena, and is likely a consequence of social contagion as opposed to the simultaneous reaction to the same event. Many of the the manually labelled events (e.g. event 2, Table 3) not only show increasing, but also fluctuating recruitment for multiple seconds (and so does Figure 8). This dynamic recruitment throughout an event can only mean that either (i) the labelling method did not start a new event early enough, or that (ii) this behavior must be facilitated at least in part by information spread through social contagion. Even if (i) is true, although it is unlikely that events with a distance of less than 10 seconds are entirely unrelated in this system, the spatial development (or spread) of some events (e.g. 3 and 4) is undeniable.

This may give credibility to another possible explanation: that these events are natural properties of the termite’s social and asocial interactions with the environment and conspecifics (voluntary or involuntary). Sudden activity bursts could help large groups of animals to escape potentially deadly feedback loops that can result in rotational “mills” of ants (such as [189]), also known as the “death spiral” since ants can be trapped in such self-created circles of pheromone until they die of exhaustion. There is (weak) evidence that individual behavior is inherently chaotic [198, 42], the idea is put forth that the “collective mind” of ants may be “the result of the amplification of fluctuations” (from [198]).

Quantitive study of such phenomena is difficult and outside the scope of this paper, but we manually followed some of the individuals in TRex for a number of frames around the annotated events — just to get an idea of what they do. Their behavior appeared to shift especially after the “first responders” started colliding with other individuals, causing them to become agitated as well. Of course,
without further and detailed study (which we want to strongly encourage) this is merely conjecture. Ultimately, we cannot be sure as to the origin of these events and we want to express our hope that future studies will control for factors like (i) sound and vibrations, (ii) wind, and (iii) odor; to make stronger conclusions.

Modern methods, like convolutional neural networks and LSTMs, can be applied and successfully solve almost any sequence prediction problem, but we also want to emphasise that while these computationally complex approaches have many advantages, they should not always be the first item on the list. As we have shown, simpler methods can often do the job similarly well. There is, in any case, the need for examples and an evaluation of different possibilities such as in this paper, to give researchers an idea of what is possible in managing and analysing an ever increasing amount of data on limited systems. More quantitative experiments are important, but they need to be achievable for everyone. Modern cloud computing solves this in part, as it did for the hyper-parameter analysis in this paper, by making it possible to pay for computation time whenever it is required. However, these solutions are often expensive and remain out of reach for many people. The methods presented herein (except the hyper-parameter analysis), which allow for almost any number of splits to reduce memory load, can be used on most moderate systems — especially in the case of the simpler method. Even the machine learning based approach does not require a high-end graphics card or a cluster, which makes us hopeful that this helps researchers across fields to manage their big data with smaller money.

2.6 Future

Many successfully trained networks within the top 5% of scores tend to predict mainly the trails in the density channel, but often are worse at capturing details — this is probably good for detecting anomalies, essentially mimicking a running mean. Where it differs, however, is especially in the beginning and the end of the video because a running mean will always require a very large window to be effective as an anomaly detection method. The machine learning approach only requires 3 or 5 input images to get a good estimate of the trail network, just based on this very short sequence. If prediction and training are possible with shorter input sequences, this would allow for predicting an entire trail network based on noisy three-frame (or five-frame) samples. Furthermore, areas with unusual or fast movement that are part of events often appear as blurry regions in the predictions (see Fig. A3). This could make it possible to detect anomalies by calculating the “blurriness” or “focus” of heatmaps over time and also to localize it inside each frame.

Since the loss function in this study seemingly was not optimal, one could use GANs (generative
adversarial networks) to train the loss function, as in image2image networks (see [Chen et al.]), and find a way to make the loss function more directly correlate with the scoring function. As mentioned above, we encourage further study of this phenomenon and suggest adding the necessary information for determining the origin of these kinds of events. Optimally, these future studies would include 1. synced sound recordings; 2. if not included in the sound recordings, other information on vibrations of the arena; 3. spatial odour information; which could later be related to any detected events.
A deep learning method for the automatic visual categorization of individuals in high-resolution videos

Automatically classifying animals based on their visual appearance has many potential applications in computational biology, neurobiology, ecology and likely other fields of research. We present a machine learning method that can help to speed up the supervised sample selection process with an intuitive graphical user interface, and teach itself to automatically classify thousands of individuals simultaneously in high-resolution videos. Integration into TRex [222], our own multi-purpose tool for tracking related problems, eliminates the need for custom implementations and makes it immediately available for existing users of our software. Existing frameworks for machine learning based segmentation and object recognition are technically superior in specific tasks, but these general purpose algorithms do not have access to the movement history of individuals or their posture. As a consequence, they sometimes intransparently mislabel overlapping individuals (e.g. de-
ecting two when there are three), where our software is more conservative. Applying our method to four >2h videos of termites (*Constrictotermes cyphergaster*) and calculating a number of caste-specific metrics per video, we find a surprising degree of similarity between soldiers and workers of the species in terms of generic kinematics and interactions, with only slight differences in temporal dynamics. Comparing the results for 14 clips (6 min) between the Detectron2 framework (Facebook Research) and our method shows more than 98% agreement between them (at >92% coverage), while TRex more conservatively detects 5±3.43% fewer individuals/frame than Detectron2. Since Detectron2 frequently assigns too many individuals in some cases (according to manual validation), the lower estimate by TRex may more closely represent the true value. Integration within TRex also makes our approach, which is specifically designed for analysing animal behavior experiments, easier to use than the general purpose Detectron2 framework.

### 3.1 Introduction

Many species exhibit inter-individual differences in morphology and/or behavior, most evident in species that have castes, like termites, that can be easily differentiated by morphology. In many species, classifying differences is less clear and misclassifications are likely, but staking out phenotypical borders between castes and classifying the individuals e.g. with respect to their role in the colony, such as “workers” or “soldiers”, is useful to facilitate meaningful analysis. Since these differences between types of individuals (or even species) are likely not only morphological, but potentially also behavioral, it is important to have access to this information for behavioral analyses. Such behavioral differences are known to exist between the castes in termites [152, 155]. Due to the often very large numbers of individuals in study systems like eusocial insects, however, manually labelling entire colonies for a long time is unrealistic. Existing softwares for automatic annotation, specific to research in animal behavior (such as [113, 143, 86]), are not designed to annotate morphological types but behaviors or body postures. Unfortunately, the only currently available option to researchers, general purpose segmentation and classification frameworks like Detectron2 (by Facebook Research, [230]) are, while offering state-of-the-art detection [230, 233], often also complicated to use. They span a wide range of features, but also typically require programming skills and may employ unnecessarily large deep neural networks for this purpose. Especially for studies on individuals who live in large groups or colonies, like termites and ants, the lack of easy-to-use methods of this kind can be a problem already in the design of experiments, since such an experiment seems out of reach for many biologists.
To help researchers, especially those operating outside the realm of computer science, we decided to integrate a much simpler algorithm in our existing tracking software (TRex [222]) that performs similarly well. Our machine learning method, tailored to experiments in animal behavior studies, automatically identifies different “types” of individuals based on manually annotated examples of their visual appearance. Integration into our tracking software makes it immediately available to existing users, without any adjustments on their part, and fills a gap in TRex’ feature set: our method for the visual identification of individuals in [222] has the requirement that, at least once, all individuals must be visible and separated from each other at the same point in time. Here, we adopt the same small network architecture as was used for learning identities in TRex, with minimal modifications, but change the training procedure and sample accumulation completely. During this training phase, the deep neural network (DNN) learns to identify morphological types of individuals using a number of automatically selected, but manually annotated sets of images. This way, with minimal user involvement during a short and guided annotation process, we can classify individuals without any further restrictions, and across videos.

As an example dataset for the application of our methods, we use field data of the Neotropical termite species *Constrictotermes cyphergaster* (Silvestri, Isoptera: Termitidae) collected in Brazil in the wet seasons of 2019 and 2020 (detailed in Recordings). Here, applying our method allows us to identify all visible individuals as “worker” or “soldier”. In termites, soldiers are often the most distinct morphological state of the species [155, 149] and, being a developmental end-point with its focus purely on defense, are typically thought to occupy a behavioral extreme as well [149], [20, chapter 6.1]. This makes them particularly interesting for comparative analysis with other castes/types. Workers’ roles can be much harder to identify and describe because (i) other developmental stages, like Pseudergates, tend to look very similar despite being more similar to larvae behaviorally; and (ii) workers are not entirely specialized and will take up different tasks adaptive to circumstance [155, section 6.5]. However, “true workers” are most likely the only caste actively participating in foraging (as per [20, section 6.1.2.3]) and thus likely the only worker-read termites met outside the nest during foraging experiments.

While the overall ratio of these two castes varies in each nest and per season [151], workers are usually more numerous than soldiers in *C. cyphergaster* (between 1:2.5 and 1:4.5, [220, 193]). Despite the fact that this ratio is already strongly biased towards workers, it is not as strongly biased as in other termite species [157, 94] — in fact, some subterranean species do not have a soldier caste at all [4]. Since *C. cyphergaster* forages outside of protective tunnels, the larger investment in producing many soldiers is likely an adaptation to the strong selection pressures imposed by its many preda-
tors [190, 217, 137, 179]. Studies investigating caste behavior in *C. cyphergaster* focus on one caste only, neglecting the potential differences and interactions between the two. Some interesting exceptions describe observations from the field [151], but we have no knowledge of any quantitative study with continuous observations outside the spatial limits of a petri-dish (or similarly small setups, like [49, 51, 105]).

In this paper, we want to make a step towards filling this void for *C. cyphergaster*. Following the interesting soldier:worker ratio fluctuations in four >2h long recordings of entire nests during the exploratory phase can help quantitatively assess some of the observations made in previous studies [151]. To demonstrate that the same network can be used to classify individuals in multiple videos, even though it has only seen a subset of them, we use a single network to classify individuals in 10 clips of different nests (6 minutes each). These clips were taken from the beginning of our >2h long trials mentioned above. Different tools can find different solutions for the same videos, but we will use the similarity between the findings as a proxy for correctness.

In order to be able to track the large number of individuals in these videos, we also needed to add an improved matching algorithm to TRex, which we call “automatic”. It adaptively chooses the best matching algorithm for each situation, via a divide-and-conquer approach, making it much faster than all methods we previously implemented in TRex. To assess its correctness, we test this “automatic” method by comparing it to the Hungarian algorithm [122] that was already implemented in TRex (see [222], and Chapter 1).

Finally, applying the methods detailed in this paper, we present summary metrics of the two castes for four full-length recordings. These include per-caste speeds, and the caste ratio over time. Since tactile interactions are generally assumed to be important in eusocial insects, especially since workers and soldiers of this species are blind [50], we also include a quantitative analysis of (likely) tactile inter- and intra-caste interactions. These types of interactions are approximated by combining multiple sources of information, including, but not limited to, spatial proximity.

### 3.2 Materials & methods

In this section, we describe the experiments we used to test our methods, the changes that we had to make to TRex to be able to track even larger animal collectives, how users of the software select training data (across multiple videos), how the DNN is trained, and how the comparison between TRex and Detectron2 was made.
3.2.1 Recordings

Nests were collected on-site in Brazil, Divinópolis-MG, in two field-seasons during the wet season of 2019 and 2020. These colonies were kept inside a nest box, where temperature and humidity conditions were controlled following recommendations for working with C. cyphergaster under laboratory conditions (25°C and 85% humidity, see [68]). The experiments were designed to investigate the natural foraging behavior of C. cyphergaster, and so all individuals were permitted to move unrestricted inside the circular arena (with a 30cm radius). Individuals can emerge from a hole in the center of the arena that is directly connected to the nest box.

All videos were recorded in high spatial resolution (4K) and with a frame-rate of 25fps using a SONY ILCE-7SM2 camera. Although all videos are in color, we only used the green channel here, which typically has the highest spatial resolution (due to the way CMOS sensors are built [62]). Markers printed onto the arena floor were used to calibrate the unit conversion between pixels and centimeters (see Figure 1).

The average number of individuals on screen is given in Table 2. The size of individuals on screen, calculated as the number of pixels that make up the individual, varies between 113.1±23.8px for workers and 119.5±21.8px for soldiers (with a resolution of 59.38px/cm).
<table>
<thead>
<tr>
<th>name</th>
<th>date</th>
<th>temperature</th>
<th>humidity</th>
<th>length</th>
<th>resolution (px)</th>
<th>pixel size</th>
<th>nest volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>N01</td>
<td>16.11.20</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:29:51</td>
<td>3840×2160</td>
<td>158.2 µm</td>
<td>15.46 L</td>
</tr>
<tr>
<td>N02</td>
<td>17.11.20</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:30:06</td>
<td>3840×2160</td>
<td>160.5 µm</td>
<td>16.33 L</td>
</tr>
<tr>
<td>N03</td>
<td>18.11.20</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:25:59</td>
<td>3840×2160</td>
<td>160.3 µm</td>
<td>17.54 L</td>
</tr>
<tr>
<td>N04</td>
<td>19.11.20</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:27:36</td>
<td>3840×2160</td>
<td>161.8 µm</td>
<td>17.07 L</td>
</tr>
<tr>
<td>N06</td>
<td>21.11.20</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:02:17</td>
<td>3840×2160</td>
<td>161.3 µm</td>
<td>14.32 L</td>
</tr>
<tr>
<td>N07</td>
<td>22.11.20</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:05:19</td>
<td>3840×2160</td>
<td>162.9 µm</td>
<td>15.5 L</td>
</tr>
<tr>
<td>N08</td>
<td>23.11.20</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:50:22</td>
<td>3840×2160</td>
<td>162.3 µm</td>
<td>21.19 L</td>
</tr>
<tr>
<td>N15</td>
<td>20.11.19</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:21:15</td>
<td>3840×2160</td>
<td>173.0 µm</td>
<td>14.86 L</td>
</tr>
<tr>
<td>N16</td>
<td>20.11.19</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:23:07</td>
<td>3840×2160</td>
<td>168.1 µm</td>
<td>8.7 L</td>
</tr>
<tr>
<td>N17</td>
<td>20.11.19</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:20:06</td>
<td>3840×2160</td>
<td>168.3 µm</td>
<td>15.11 L</td>
</tr>
<tr>
<td>N18</td>
<td>21.11.19</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:20:47</td>
<td>3840×2160</td>
<td>172.1 µm</td>
<td>12.49 L</td>
</tr>
<tr>
<td>N19</td>
<td>21.11.19</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:26:57</td>
<td>3840×2160</td>
<td>173.3 µm</td>
<td>10.77 L</td>
</tr>
<tr>
<td>N21</td>
<td>21.11.19</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:20:47</td>
<td>3840×2160</td>
<td>170.9 µm</td>
<td>15.06 L</td>
</tr>
<tr>
<td>N22</td>
<td>22.11.19</td>
<td>25±1°C</td>
<td>85±1%</td>
<td>02:22:27</td>
<td>3840×2160</td>
<td>168.4 µm</td>
<td>9.01 L</td>
</tr>
</tbody>
</table>

Table 1: Metadata for all videos from field-trips to Divinópolis-MG, Brazil in 2019 and 2020. Temperature and humidity were controlled for in the close-by lab according to [68]. As part of the comparison to Detectron2, and validation of our method, the first 6 minutes of each video were processed. N07, N19, N21, and N22 were then processed in full, and used to evaluate different movement and interaction metrics quantitatively (see subsection 3.3.3).

3.2.2 Pre-/Processing with TRex

The videos N07, N19, N21, and N22 were converted to a preprocessed greyscale format by TGrabs using background subtraction (threshold=60), which is the default segmentation method. We disabled the usage of an absolute threshold, so that only darker shapes would be tracked (enable_absolute_difference=false). Every image of the clips was further enhanced using a sequence of established image processing methods, such as “closing” and luminance equalization (TGrabs parameters correct_luminance, and use_closing). Individuals were then tracked using the new “automatic” matching method (presented in Updates to TRex: An adaptive matching algorithm), with a maximum speed threshold of track_max_speed=10 (cm/s) and a size threshold of blob_size_ranges= \([0.015, 0.07]\). The individuals in each video were first tracked by TRex, and results exported, after
which TRex was launched a second time to predict castes using a pre-trained network\(^1\) (as described in 3.2.6).

While TGrabs was run for the whole video, so that a single pre-processed video in PV format could be used (see [222]), it had to be virtually split for tracking into 5 individual parts since keeping track of so many individuals used up too much memory otherwise. Each part overlapped with the previous part by 500 frames (20s), so that they could be properly merged during analysis, avoiding edge effects.

Additionally, to validate our prediction quality, by comparing to results given by Detectron2, for a much wider range of videos, the first six minutes of every video in Table 1 are processed in the same way as above. For these videos we used a maximum speed of \(\text{track\_max\_speed}=15\) so that assignments were slightly stricter - this does not otherwise affect results since the measured speeds were not used in our analysis (as shown in subsubsection 3.3.3.2, individuals rarely ever move nearly as quickly as this).

Since our later analysis involves the detection of simultaneously disappearing individuals (see 3.3.3) based on “consecutive segments” (defined according to the rules specified in 3.2.6), we had to ensure that individual trajectories are not artificially ended just because it was the end of a part. Thus, all parts were merged according to the reported positions of individuals — each identity within overlaps was checked for a corresponding match in both parts based on its exact position. All were matched successfully. The two identities were then joined and remaining duplicates removed (to ensure e.g. all trajectories that started and ended within the overlap only appear once).

### 3.2.3 Pre-/Processing with Detectron2

A training dataset of 146 images was obtained by randomly sampling amongst the 6 minute video segments of all 15 colonies. To speed up the manual annotation process, a custom python code was used to obtain the bounding boxes of moving termites via background subtraction - incidentally also demonstrating the point made in the introduction, that custom solutions have to be crafted to use these frameworks. The annotations were manually corrected and assigned the corresponding caste (soldier or worker) using the annotation software CVAT ([cvat.org](http://cvat.org)). In total, 146 images containing 50,849 workers and 9543 soldiers were annotated.

The annotations were used to train a pre-trained faster R-CNN model with a ResNet-50 + Feature Pyramid Network backbone ([github.com/facebookresearch/detectron2](https://github.com/facebookresearch/detectron2)). Annotations for training and validation were split 9:1. The trained model was then used for inference on all 6 minute segments, yielding the bounding box and predicted caste of each detected termite. As non-maximum

\(^1\)These two steps can theoretically be joined, and were separated in this case purely due to personal preference.
suppression (keeping only the most likely bounding boxes when they overlap) in Detectron2 [230]
only regards detections within a class, we afterwards assigned the soldier caste whenever the model
predicted both a soldier and worker caste within a 4 pixel radius. Next, all detections with a certainty
below 0.4 were removed.

The thus obtained detections were linked using trackpy’s link function (soft-matter.github.io,
[8]), with a search range of 30 pixels and memory of 10 frames using a k-d tree linking method. We
removed all IDs which were detected only for one frame. To assure stable caste assignment within an
ID, we used a rolling window of 10 frames to correct for single frame switches.

3.2.4 Data normalization

All exported data, both from Detectron2 and TRex, was normalized per video based on the meta
information in Table 1 and more. The coordinate system was centered on the nest entrance, and all
position-based data was converted to centimeters according to the measured pixel sizes. Each dataset
was also filtered so that all detections with positions outside the arena were removed (this was done
based on manual annotations of the arena edges). All of the metadata is available in the supplemen-
tary material files arena_annotation.csv and whole_colony_metadata.csv.

3.2.5 Updates to TRex: An adaptive matching algorithm

More than 1500 individuals were visible simultaneously in many of the videos (see Table 2), which
made adjustments to TRex’ matching algorithm necessary. In the original publication of TRex [222]
we presented three matching algorithms:

- tree-based
- approximate
- Hungarian method [127, 122, 153]

and demonstrated that the tree-based method is accurate and matches the assignments of the Hun-
garian algorithm. Approximate matching was shown to be much faster and to scale well for large
numbers of individuals, but that it produces inaccurate results. It simply assigns objects according
to the maximum probability in a “first come first served” manner, which is also subject to threading-
related inconsistencies and especially problematic in situations where individuals are close.

The Hungarian method, although it does not have this inconsistency problem, has a worst-case
time-complexity of \(O(n^3)\), mainly because it uses an “Individuals x Objects” matrix which grows
rapidly with an increasing number of individuals and objects. The tree-based algorithm outperformed it for most of our test-cases despite having a worse worst-case time-complexity, simply because it has a better average-case complexity. It also likely benefitted from the highly optimized integration with the rest of TRex, a smaller memory footprint, and easier optimization/pruning inside the tree. However, where it is impossible to prune, e.g. when individuals consistently stay close to each other, the tree-search becomes exponentially more costly as more and more options have to be explored, a “combinatorical explosion”, as we have called it in Chapter 1. In such cases, the software automatically switches to the approximate algorithm that scales linearly with the number of individuals. Switching to an inaccurate algorithm is unproblematic if it does not happen often, as in the videos we used for testing in Chapter 1.

Table 2: An overview of different metrics for the first 6 minutes of each video. The first minutes of a trial are the start of the colony’s exploration phase, where the number of individuals is often still rising. Despite that, we find one of the highest number of simultaneously visible individuals (2457) in the beginning N02. The percentage of soldiers varies greatly between videos, but is generally as high as expected, averaging to 96% in two cases.

<table>
<thead>
<tr>
<th>video</th>
<th># individuals</th>
<th>% soldiers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>mean</td>
</tr>
<tr>
<td>N01HHS2020</td>
<td>1015</td>
<td>423.78 ± 249.55</td>
</tr>
<tr>
<td>N02HHS2020</td>
<td>2457</td>
<td>1132.5 ± 546.52</td>
</tr>
<tr>
<td>N03HHS2020</td>
<td>1843</td>
<td>837.57 ± 335.6</td>
</tr>
<tr>
<td>N04HHS2020</td>
<td>1912</td>
<td>1185.48 ± 434.07</td>
</tr>
<tr>
<td>N06HHS2020</td>
<td>681</td>
<td>220.45 ± 125.93</td>
</tr>
<tr>
<td>N07HHS2020</td>
<td>1544</td>
<td>675.89 ± 303.25</td>
</tr>
<tr>
<td>N08HHS2020</td>
<td>1661</td>
<td>834.69 ± 414.9</td>
</tr>
<tr>
<td>N15HHS2019</td>
<td>225</td>
<td>49.57 ± 43.51</td>
</tr>
<tr>
<td>N16HHS2019</td>
<td>306</td>
<td>162.94 ± 57.21</td>
</tr>
<tr>
<td>N17HHS2019</td>
<td>729</td>
<td>377.61 ± 151.33</td>
</tr>
<tr>
<td>N18HHS2019</td>
<td>181</td>
<td>78.12 ± 40.01</td>
</tr>
<tr>
<td>N19HHS2019</td>
<td>19</td>
<td>3.09 ± 1.76</td>
</tr>
<tr>
<td>N21HHS2019</td>
<td>611</td>
<td>297.8 ± 161.93</td>
</tr>
<tr>
<td>N22HHS2019</td>
<td>511</td>
<td>247.55 ± 145.83</td>
</tr>
</tbody>
</table>

However, the large number of termites in our foraging experiments, combined with their species-
specific trail-forming behavior, leads to closer clustering and thus fewer optimization opportunities for our existing matching methods. The Hungarian algorithm, for example, barely achieves more than 1fps in our termite videos (see Results), which is unacceptable for tracking hours of footage.

Each combination of “individual + object” has an assignment score. These scores are largely determined based on an individual’s proximity to objects, or, more specifically, calculated based on each object’s position and direction with respect to an individual’s movement history (see [222] for an in-depth description). Matching, tersely put, is a maximization of the total sum of scores for all chosen assignments.

Consequently, since only the total sum of scores matters, even non-maximum assignments between an individual and an object can improve the total sum if their “sacrifice” (in terms of local optimality) enables higher-scored assignments elsewhere to balance them out. Thus, “correct” matching algorithms scale badly with the number of possibilities because, to ensure that the globally optimal solution has been found, every combination of objects and individuals has to be considered. This is, essentially, what makes matching so difficult - to a point where even an attempt at listing the problem space would be problematic for large numbers of individuals and objects.

To optimize this process, algorithms can be designed to prune improbable/impossible combinations early (in the case of tree-based matching) or mark them as invalid and quickly discard them (Hungarian), but even “quickly dealing with invalid values” becomes inefficient for large problems. Additionally, these algorithms are inherently single-threaded and hard to parallelize in practice. As a consequence, the user is forced to make a difficult decision: either e.g. decrease the “maximum speed” threshold, which can be problematic for tracking individuals that speed up sporadically, or simply resign and use inaccurate matching.

Fortunately, solving this problem in our case, and preventing an exponential growth of possibilities, is possible via “divide and conquer”: if each clique is kept small, and each clique can be processed in parallel, then the algorithm becomes combinatorically stable and more easily scalable. The most critical component of the algorithm then becomes the decision, which individuals and objects need to be considered together (in the same group/clique) and which can be viewed as an independent problem. To this end, individuals with significant assignment scores for the same objects are grouped together, forming a “clique” of individuals that have influence on each other’s assignments (see Figure 2). Importantly, this influence is recursive so that, if both individuals could possibly be assigned to the same object, then they are part of the same matching problem. Conversely, if no such connection exists (i.e. they have no influence on each other), then they can be handled separately. Where the program formerly had to solve a matching problem for the entire arena at the same time, it now
Figure 2: Schematic visualisation of the concept of “cliques”, illustrated as circles around individuals. Who belongs to which clique is, mostly, determined by their proximity (using a maximum distance threshold). In the new matching algorithm described in subsection 3.2.5, instead of considering all individuals at once, each clique is solved separately using the Hungarian algorithm — single individuals (e.g. the green circle) are assigned directly.

decides assignments per clique and adaptively selects the most appropriate algorithm for each case. Single individuals with a clique-size of 1 are assigned directly, while cliques of multiple individuals are solved using the slower (but accurate) Hungarian algorithm. Splitting the arena into entirely independent matching problems allows for thorough parallelization, too.

Under normal circumstances, any division of the objects-to-be-matched remains an approximation of the matching problem, strictly speaking. However, considering that TRex in any case discards individual + object combinations if their score is below a certain threshold (matching_probability_threshold, usually 0.1), this is not the case here\(^2\). Independent cliques truly are always separate and the speed gain is achieved by the additional cost of a calculation of cliques. Turning this algorithm into an approximate algorithm is possible, however, via adjustment of the clique calculation. This adds another tuning possibility if it is still not fast enough.

This algorithm has now been permanently integrated in TRex (under the name “automatic”), which is freely available at trex.run under an open-source license. For an evaluation of this algorithm by way of comparison to the pure Hungarian method, see subsection 3.3.1.

3.2.6 Updates to TRex: Categorization

Visually identifying morphological types of individuals is conceptually very similar to the visual recognition of identities. Thus, reusing the model architecture of TRex’ visual identification mod-

\(^2\)Thus, strictly speaking, this could even be seen as an oversight in the initial implementation of our tracking algorithm which has now been corrected.
ule for this new problem is the most straightforward thing to do - additionally, the number of categories (formerly identities and now morphological types) that have to be recognized is now typically lower. Just two classes (worker and soldier) have to be considered in a binary decision for our example dataset of Table 1, but, to accommodate a wider range of possible applications, this algorithm can be utilised for any number of classes (user’s choice). Likely, the capacity of this network is the same that was measured in [184], where a similar architecture was demonstrated to be sufficient for distinguishing between up to 100 classes. The architecture consists of 5 sequential layers (3x 2D convolutional + pooling layer, 2x densely connected layers), the input consists of a single image and the output is a one-hot vector (a vector with a value $\in [0, 1]$ for each class using softmax activation).

The training process has undergone bigger changes, however. While the network is still ultimately trained by using the Adam optimizer [117] for the gradient descent, the way that training data is sampled from the video is very different. Here, since categories are not predefined as they are for identities, the user chooses images to represent their respective classes. This is done in a selection screen (see Figure 3). However, this screen does not show single images but animations - something that is only possible because tracking information is available.

Conventional approaches, without tracking (such as when purely using Detectron2 to “detect” and score), have to rely merely on the accuracy of their single frame predictions. This is enough in most cases to achieve decent classification accuracy, but, logically, even less noisy results can be achieved by averaging multiple single-frame predictions. Tracking is not reliable in general (unless additional identity information is provided, e.g. by visual identification), but identities do not need to be perfectly maintained in order for it to be advantageous.

Here, the algorithm borrows again from the one for visual identification, where the same assumption was made: TReX calculates “consecutive” segments, or uninterrupted trajectory pieces, which are deemed to likely consist only of detections of a single individual (since they abide by the four rules specified below). These consecutive segments, each consisting of potentially thousands of images\(^3\), are turned into an animation and presented to the expert, who proceeds to choose the appropriate category via the respective button on-screen (see Figure 3). Since consecutive segments are calculated for all individuals, the program has many animations to choose from and repeatedly does so whenever the expert finishes labelling one. Showing animated sequences of samples to the expert has advantages beyond mere visual fidelity, and actively helps both the user and the algorithm in two major ways, as follows.

\(^3\)Sometimes also just one, of course, but a small number of long sequences can be sufficient to train the network.
Figure 3: A screenshot of the selection screen for animated clips of individuals in TRex. After deciding on the number of categories and their names, the user simply clicks one of the buttons representing one of them (e.g. in this case “(W)orker” and “(S)oldier”). Each clip consists of uniformly sampled images of randomly sampled individuals. Albeit being susceptible to bad parameter choices, given proper settings, every such clip is likely to contain images of the same individual. This speeds up manual labelling significantly, considering the alternative is to select individual images.

1. **The annotation bandwidth is maximized.** It is possible to quickly label a large number of individual samples — samples, which are animated clips instead of mere single images. The number of individual samples labelled each time depends on tracking quality (and random chance of being picked), but it can be hundreds or thousands of images per click. For longer segments it is not required to use every single image in the sequence because consecutive images are usually highly auto-correlated. Instead, we sample images uniformly for each clip, so that we end up with maximally 300 images per sequence. The playback speed is anti-proportional to the importance of each sample, and proportional to the number of samples, simply by being fixed to five seconds playback length.
2. **Moving images provide context**, allowing the expert to correctly label noisier images implicitly as part of the sequence (e.g. blurry images taken mid-movement). This has potential to be the most influential aspect, since these images would likely be impossible to annotate when viewed out of context — and the hardest to classify for the machine learning. Labelling them gives the network a chance to learn features, even in these hardest of cases.

To ensure that the image sequences do in fact mainly consist of a single individual, without interruptions by undetected identity switches, consecutive tracking segments are terminated whenever

- the individual is lost for one frame
- the distance between the previous position and the current position is too great
- the change in angle is too great
- the frame timing is too noisy.

In all other cases, tracking is assumed to be reliable (see also [222]; and the later evaluation in this paper in subsection 3.3.1). Since each video provides millions of data-points, it suffices if this assumption is true in most cases. Upon manual inspection, however, the trajectories were only problematic in very dense accumulations of individuals. These clusters are generally handled by the size threshold, where trajectories are consequently split based on the conditions listed above. The start and end of each consecutive sequence of frames is saved by TRex — this information will be of use both during network training, as described in the next section below, and our behavioral analysis later on.

We want to emphasize that this algorithm is not specific to termite castes and can be used more generally for any automatic labelling based on visual features. The user is asked to select the categories in the beginning, and is presented with buttons for each category and a “skip” button (see Figure 3).

### 3.2.7 Analysis: Improving the consistency of caste assignments

In cases with two classes, e.g. workers and soldiers, high per-image classification accuracy can easily be achieved for our network with a training set of fewer than 2000 images per class. However, real-world video data is always noisy, and the network is only trained on a small fraction of the potentially millions of images - each with different body postures and lighting conditions, that it will have to generate predictions for later on. Even with perfect validation accuracy, there will still be objects that the network is unsure, or inadequately confident about. To mitigate this, we added the following two components of the analysis pipeline, making use e.g. again of consecutive segments (see previous section):
1. We wanted to ensure that our network works properly for all videos. Since lighting and camera position/rotation can slightly change between videos (even though we tried to make sure that they do not), and individuals might look different from nest to nest, the network can be more, or less, accurate. Thus, samples were collected from multiple videos, simply by reusing the samples and network weights from one video for the next. We kept adding new samples, until the we did not see any more misclassified individuals for some time and then switched to the next video — until that also did not change prediction accuracy anymore. Usually, a decrease in prediction accuracy shows as a sudden decrease in validation accuracy when adding the first samples from another video.

2. We summarize identity values within each consecutive segment by picking the mean category prediction with the highest-sum probability throughout that segment. Averaging over a number of samples makes the category predictions less sensitive to noise and prevents identity predictions to flicker from frame to frame, even when sub-optimally trained networks are used. We can even allow ourselves to limit the number of predictions to 300 per segment because of the likely high auto correlation of consecutive images — one should generally use as many images as possible, but less correlated images add more information. Thus, a balance can be struck that minimizes computation time (adding more samples) and maximizes accuracy (adding the more valuable of the fewer samples) — simply by sampling the input sequence at regular intervals.

Overall, we annotated 23,528 images of soldiers and 26,948 of workers, totalling to 50,476 annotated detections. While a similar number of images took us many days to annotate in CVAT for the use with Detectron2 (see subsection 3.2.3), collecting these training samples for TRex in pre-tracked videos only took (roughly) thirty minutes (including the training time).

3.2.8 Analysis: Approximating tactile interactions

Investigating the tactile interactions between individuals (e.g. between Worker-Worker, Worker-Soldier, Soldier-Soldier; or WW, WS and SS) can help to draw conclusions about whether individuals may actively seek certain types of interactions out, or whether individuals collide randomly with others. This is especially important in termites, who are blind and for whom tactile interactions provide important physical and chemical clues.

The worker:soldier ratio does of course have an influence on the number of overall interactions, as does the spatial distribution of individuals, simply if they lead to a higher number of possible interac-
tion partners for one or the other caste. A higher number of workers should be expected to correlate with the number of interactions involving workers, merely due to chance encounters. Analogous to this, if individuals prefer being close to their own caste it should follow that we record more intra-caste than inter-caste interactions. The question then becomes whether there is a difference between our measurements and this null hypothesis.

To facilitate this analysis, and since we do not have information about the actual movement of body-parts like the antennae (as could only be obtained for smaller numbers of individuals by e.g. [86]), we define an approximation as follows.

The aforementioned uninterrupted trajectory pieces (or consecutive segments, as described in 3.2.6), were specifically designed to filter out instances where individuals are hidden behind objects, shadows, or intersections with other individuals. We use them, combined with additional information, as a proxy for tactile interactions/very close proximity, specifically by looking at when they end. If individuals are (1) very close to each other and (2) disappear (i.e. their trajectory pieces end) at the same time, then it is like that there is a causal relationship between the two. In combination, these filters are better than using a distance threshold on its own, which does not have the multiple sources of information available to TRex during tracking:

1. The individual movement history indicates (for each individual) where it should be.

2. Combined with the histories of all individuals, this tells the algorithm how many individuals should be in any one place — if there are more or less than expected, then the “consecutive segments” (or trajectory pieces) of all involved individuals are artificially ended.

3. This method uses pixel information as well. It tries to separate individuals using adaptive thresholds, until the resulting objects are too differently sized, large or small.

Individuals will, most likely, only disappear if the dark parts of their bodies intersect and thresholding of lighter pixels does not lead to clean separation. This can clearly be considered to be a tactile interaction, since it is likely that both individuals are even closer than necessary e.g. for antennation.

Interactions between N>2 individuals are also relatively unproblematic, since they would exclusively be matched in pairs and contribute N/2 interactions to the overall numbers — arguably, these are correctly labelled interactions and should not skew the distribution much. For a high number of individuals, the probabilities and the expected distribution of interaction types per worker:soldier ratio can be approximated simply as a die throw, for example:
\[ P(WW) = P(W)^2, \text{ where } P(W) = \frac{N_W}{N_W + N_S} \]

and \[ P(WS) = 2P(W)P(S)\].

However, especially for low numbers of individuals/interactions, the given problem is better represented by the hypergeometric distribution [29, 78] (e.g. "take \( n \) cards from a deck of \( N \) cards and get \( k \) kings"), which we used in our evaluation (see subsubsection 3.3.3.3).

3.3 Results

<table>
<thead>
<tr>
<th>video</th>
<th>% agree</th>
<th>% coverage</th>
<th>compared</th>
<th>miss (TR)</th>
<th>miss (D2)</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>N01HHS2020</td>
<td>98.67</td>
<td>93.92</td>
<td>8699984</td>
<td>526693</td>
<td>36535</td>
<td>9263212</td>
</tr>
<tr>
<td>N02HHS2020</td>
<td>97.68</td>
<td>91.58</td>
<td>24373960</td>
<td>1710320</td>
<td>529904</td>
<td>26614184</td>
</tr>
<tr>
<td>N03HHS2020</td>
<td>96.72</td>
<td>91.68</td>
<td>19673254</td>
<td>1100195</td>
<td>686107</td>
<td>21459556</td>
</tr>
<tr>
<td>N04HHS2020</td>
<td>98.64</td>
<td>90.59</td>
<td>22296736</td>
<td>1555110</td>
<td>760784</td>
<td>24612630</td>
</tr>
<tr>
<td>N06HHS2020</td>
<td>98.53</td>
<td>95.14</td>
<td>6853650</td>
<td>333758</td>
<td>16438</td>
<td>7203846</td>
</tr>
<tr>
<td>N07HHS2020</td>
<td>97.1</td>
<td>95.0</td>
<td>16482168</td>
<td>760136</td>
<td>107731</td>
<td>17350035</td>
</tr>
<tr>
<td>N08HHS2020</td>
<td>99.21</td>
<td>95.1</td>
<td>16267344</td>
<td>714422</td>
<td>124630</td>
<td>17106396</td>
</tr>
<tr>
<td>N15HHS2019</td>
<td>97.39</td>
<td>96.19</td>
<td>1680362</td>
<td>57106</td>
<td>9425</td>
<td>1746893</td>
</tr>
<tr>
<td>N16HHS2019</td>
<td>98.83</td>
<td>95.87</td>
<td>3508600</td>
<td>145846</td>
<td>5278</td>
<td>3659274</td>
</tr>
<tr>
<td>N17HHS2019</td>
<td>99.17</td>
<td>96.95</td>
<td>8537584</td>
<td>249887</td>
<td>18914</td>
<td>8806385</td>
</tr>
<tr>
<td>N18HHS2019</td>
<td>92.94</td>
<td>90.26</td>
<td>1834840</td>
<td>190896</td>
<td>7215</td>
<td>2032951</td>
</tr>
<tr>
<td>N19HHS2019</td>
<td>85.62</td>
<td>84.01</td>
<td>228348</td>
<td>40709</td>
<td>2751</td>
<td>271808</td>
</tr>
<tr>
<td>N21HHS2019</td>
<td>98.37</td>
<td>85.53</td>
<td>5537718</td>
<td>900934</td>
<td>35819</td>
<td>6474471</td>
</tr>
<tr>
<td>N22HHS2019</td>
<td>98.81</td>
<td>95.65</td>
<td>4676764</td>
<td>188578</td>
<td>24121</td>
<td>4889463</td>
</tr>
<tr>
<td>total</td>
<td>98.01</td>
<td>92.84</td>
<td>140651312</td>
<td>8474590</td>
<td>2365652</td>
<td>151491554</td>
</tr>
</tbody>
</table>

Table 3: The agreement on caste predictions between our method and Detectron2, calculated for the first 6 minutes of all 14 videos. Comparison of caste assignments was done for a subset of total detections (total), which was possible only if approximately the same positions were reported for the same number of individuals in the local vicinity. Successful matches are listed as "compared", discarded detections as "miss (TR)" for TRex and "miss (D2)" Detectron2. Since Detectron2 generally reports more detections across all videos, they can more often not all be matched to TRex’ results.
3.3.1 Tracking: Evaluating tracking speed and quality of results

To test the matching algorithm described in subsection 3.2.5 regarding its speed, compared to the other available matching algorithms, we tracked parts (frames 5000-5500, and 100000-100500) of the six minute clip from the beginning of N07 using the (i) Hungarian, (ii) approximate, and (iii) automatic method separately. These specific frame ranges were chosen to represent the most difficult situations for a matching algorithm that were still solvable in manageable time by the Hungarian algorithm (which was already very slow, as pointed out below). While we could have chosen higher-density sequences, both already contain high numbers of (moving) individuals in close proximity (see also Figure 6a) - if there a matching method is inaccurate, then it will fail here. Results produced by (i) are used as a baseline for the other methods to approximate their tracking quality. Note that even with perfect matching, specific assignments can be different because of inherent stochasticity of e.g. memory allocation patterns and multi-threading. We therefore cannot assume that IDs are assigned in a specific order by TRex, which, if we did, would anyway be skewed by an overly strong influence of follow-up errors.

Instead, we matched every detection of (ii) and (iii) to a detection in the same frame in (i). Then, every time an exact match for a detection by (ii) or (iii) is not found in the baseline (i), or an extra detection is reported, an error counter is increased. On average, 9592±246.7 errors were reported for the approximate matching method that was thoroughly outperformed by the new method, for which no errors were found. Different numbers of errors demonstrate the inherent indeterminism of the approximate method, as mentioned above.

In terms of processing speed (note this is only tracking, not yet caste prediction) the new matching algorithm (25.5±7.6us) performed similarly to the approximate (23.0±7.4us), and both much better on average than the Hungarian, which took almost two seconds per frame on average (1708±2254.5us). The large standard deviation for the Hungarian is likely due to its non-linear relation-ship between the number of detections and processing time (time-complexity [127], note that Figure 4 shows log time).

3.3.2 Categorization: Evaluating the quality of caste predictions

Using TRex and Detectron2, we created two datasets for each of the video clips in Table 3. Both tools output many positions and predictions per frame and detection. Data produced by both tools was individually normalized so that the arena center is at (0,0) and the unit is properly expressed in centimeters (see Figure 1).
We summarised the single-frame detections made by TRex as described in “Improving the consistency of caste assignments”, so that the caste is averaged per consecutive segment. Detectron’s predictions are additionally filtered by their prediction score (\(\geq 0.4\)), which constitutes a relatively low threshold (Detectron2 examples in their documentation suggest a threshold of 0.6 for video prediction tasks) since we found that even low-confidence scores were correct in the cases we observed. Being more liberal with specific assignments is also warranted because some of the noise is counteracted by summarizing.

Unlike TRex, Detectron2 and trackpy both have no internal mechanism to take the movement history of individuals into account when assigning identities — this means that single detections may sometimes unknowingly consist of multiple overlapping individuals. Detectron2 has to rely purely on its advanced machine learning based segmentation for this, which can account for overlaps sometimes (and generally tries to detect as much as possible). TRex implements a mechanism that attempts splits where possible, based on individual history, and will prefer to exclude overlapping detections if they cannot be separated cleanly. While both approaches are imperfect in practice, TRex is generally more conservative and Detectron2 more liberal when it comes to what constitutes a valid detection. It should thus be expected that Detectron2 over-estimates the number of detections, which is confirmed by our results: Detectron2 finds 5±3.43% more individuals per frame across all clips. This meant that, to evaluate the agreement between methods, we could not match detections directly and had to account for missing (or additional) detections in each frame. Since different segmentation algorithms could potentially also detect the same individual in a slightly different position...
Figure 5: In (a), visualizing the comparison between TReX’s and Detectron2’s predictions for frame 5000 of N03. If detections are found by both softwares, they are shown as small dots (the caste predictions of which are in >97% agreement for this frame). If a detection by one software does not appear in the list of detections made by the other software it is marked in yellow if it is missing in TReX’ data, and blue if it is missing in Detectron2’s data. If TReX (or Detectron2) “misses” a detection of the other, this does not automatically determine which solution is correct. Sometimes, “missing” a detection can be the correct solution, as is clearly visible by zooming in on the raw video in (b). There, in (b), as an example, the raw video frame is cropped to the area highlighted by a black rectangle in the bottom-left of (a). Clearly, Detectron2 has mistakenly labelled additional, non-existent, individuals for three actual pairs of individuals in the center of (highlighted by arrows).
(e.g. head vs. center position) we allowed detections to be offset by a maximum of 0.5cm. The score of agreement was thus calculated as the fraction of successful matches between detections from both methods and the number of agreeing caste-predictions:

\[
\frac{\text{same caste}}{\text{matches}}, \text{ where matches} \leq \text{total}
\]

Unsuccessful matching attempts constituted 1.76±3.13% of TRex’ detections that could not be found in Detectron’s results, and 10.84±7.57% of Detectron2’s detections that were not found in TRex’ results. Since 10% is a large number we followed our analysis up by a manual investigation into the origin of this (at times) stark difference (see Figure 5). It appears that the filtering mechanism we employed permitted a large number of points on the edge of the arena to “slip through”, of individuals which are only partly visible and their labelling thus of questionable accuracy. These are filtered out by TRex due to their size (compare also Figure 5b, where the head of an individual is separated from its body, but not tracked and thus does not appear in Figure 5a). We also found that our Detectron2 model seems to have identified extraneous individuals, where there are none in the real video - preferably in places where two individuals touch (see Figure 5). These were successfully separated by TRex, however, leading to an overall lower number of (more correctly) estimated individuals. The individuals that are missed by TRex are typically intersecting individuals that are unseparable by thresholding (and thus likely interacting, as described in subsection 3.2.8). Here, machine learning based segmentation certainly has an advantage. All successfully matched detections were in high agreement between the two tools, making the same predictions in more than 98% of the evaluated cases across all videos.

3.3.3 Behavioral analysis

The amount of data that was collected allows for a quantitative comparison between soldiers and workers within videos, but — to an extent — also between videos. In the following, we summarize the most interesting findings directly related to differences and commonalities between the castes as well as other summary statistics. From the analysis and all figures/tables, we excluded the first data-points of each consecutive segment because they contained invalid (speed) values due to faulty interpolation in TRex (will be fixed in a newer version). We also dropped segments shorter than 1s (or 25 frames). Together, all four full-length videos in Table 4 amounted to over 623 million individual single-frame detections after filtering.
Table 4: An overview of different metrics for each video that was analysed in its entirety (only a subset of videos were, see Table 3 for a complete list). The number of individuals does not necessarily reflect the number of individuals inside the nest, only the individuals that emerge can be tracked. Movement speeds were relatively equal between the castes and trials, but individuals in N19 and N21 moved slightly more slowly.

<table>
<thead>
<tr>
<th>video</th>
<th># visible individuals</th>
<th>movement speed (cm/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>mean</td>
</tr>
<tr>
<td>N07HHS2020</td>
<td>1728</td>
<td>838.5 ± 409.22</td>
</tr>
<tr>
<td>N19HHS2019</td>
<td>176</td>
<td>98.33 ± 34.01</td>
</tr>
<tr>
<td>N21HHS2019</td>
<td>2189</td>
<td>1524.72 ± 354.63</td>
</tr>
<tr>
<td>N22HHS2019</td>
<td>931</td>
<td>525.5 ± 206.52</td>
</tr>
</tbody>
</table>

3.3.3.1 Number of individuals and caste ratios

Overall, high numbers of individuals were detected (a maximum of 2189 simultaneously visible individuals in N07), which averaged to hundreds of simultaneously visible individuals throughout the videos. These numbers usually dwindle towards the end of the 2:20h time window we recorded (see Figure 6), during which all videos showed interesting and different temporal dynamics.

In all videos, the number of soldiers remained relatively stable throughout N22, and N21 (see Figure 6d and Figure 6c), and N07. The ratio of workers:soldiers in the first half of N07 looks similar to N22, where workers were clearly in the majority (>70%). This was the expected behavior according to [220, 193]. The most drastic differences were measured in N07, where after an hour of recording a slow decline of workers ultimately leads to the ratio being flipped entirely, until there are almost exclusively soldiers in the arena (see Figure 6a).

In N19, much fewer individuals emerged from the opening — averaging to only 103.5±34.98 individuals per frame (see Table 4). In this video, the termites start forming a single “arm” of foragers shortly after emerging, which also remains the only stable and established trail throughout the video. In fact, this behavior may be more naturalistic compared to the other trials, since C. cyphergaster usually forages along tree trunks in single file. Here, it seems that there is an overall lack of motivation to leave the nest. Lower foraging activity has previously been linked both to lower humidity and season [151], but both were controlled for in this experiment. Since N19’s total nest volume (10.7L) was also larger than e.g. N22’s volume (9.01L), and thus likely N19 has a larger total population size when using the nest volume to approximate it, it is possible that the colony went foraging right before
their nest was collected and individuals are thus not motivated to leave it (see e.g. [151], in which foraging bouts are reported to occur every 1.6 days in the wet season). For details on calculating population size from nest volumes, see e.g. [58], where roughly 150 soldiers/L were reported; [193] reports 4.5 workers/soldier, while we measure proportions of up to 1:1.36 (see Table 4).

### 3.3.3.2 Per-caste movement speed

The average speeds of our own results do not warrant the conclusion that, in freely behaving C. cyphergaster, differences between the castes exist (see Table 4). On average, over the course of all three fully tracked videos, soldiers moved slightly faster than workers with a larger standard deviation (0.95±1.03cm/s), but were not much faster with considerable overlap of their speed distributions (workers: 0.88±0.88cm/s, see also Figure 7). Presumably, intermittent periods of slow movement and stopping impact the standard deviation. Soldier data is especially affected by this, since they seem to occupy a higher percentage of extremes (standing still and moving very fast):

Only 0.6% of worker detections were found in the highest 1% of speeds (>4cm/s), whereas 1.3% of soldier detections fell into this category (compare to Table 4 for per-video analogs). Looking at soldiers and workers separately, 5.2% of soldier detections were found in the lowest speed regimes (<0.01cm/s) and only 2% of workers. This suggests that soldiers move slowly and quickly relatively more often than workers, who stop less often and move at average speeds (this can also be seen in the kernel density estimates of both speed distributions, see Figure 7).

Regarding the actual distance travelled, we found no differences. We calculated the mean dispersion per consecutive segment as the average distance to the mean xy position of each uninterrupted trajectory piece. We calculated these for all sequences that were between 1s and 4s long, yielding the following per-video results:

- **N07**
  - Workers: 0.51±0.53, N=807450
  - Soldiers: 0.66±0.63, N=628177

- **N19**
  - Workers: 0.33±0.5cm, N=88488
  - Soldiers: 0.35±0.52, N=116328

- **N21**
  - Workers: 0.3±0.35cm, N=7406625

102
Figure 6: The per-caste dynamics for the fully tracked videos, showing the #individuals per caste (y) over time (x). In N07, the number of workers peaks early and then continuously declines while the number of soldiers remains stable. Individual numbers fluctuate more strongly in N19 and N22, and a lower percentage of soldiers overall (see also Table 4) but both show similarly declining worker numbers after an early peak.
Figure 7: A kernel density estimate of the speed for N07, N19, N21 and N22 per caste (worker or soldier). Every 100th data point was sampled per video, amounting to 6,124,689 data points in total, and then combined for this joint KDE plot. While the distributions are very similar, we see that both extremes of high and low speeds are more often occupied by soldiers than by workers.

- Soldiers: 0.4±0.49cm, N=2753314
- N22
  - Workers: 0.53±0.56cm, N=830565
  - Soldiers: 0.57±0.62cm, N=147064

3.3.3.3 Approximating tactile interactions between individuals

To make our analysis even more robust, we tested 8 different distance thresholds (which is the only chosen parameter here) in eight 0.25cm steps between 0.25cm and 2cm. The latter may be too far away to accurately filter for proper interactions (>5 body lengths), but no parameter choice in this range changed the results much — as indicated in Figure 8 by the error bars. The lower limit of 0.25cm was chosen because almost no interactions were detected below it.

As can be seen in Figure 8, the distribution of over 503,734 mean interactions (mean for the 8 distance thresholds per video, with a large standard deviation of 197,000) were accurately described by the expected distributions. We can thus conclude that, within the conditions of our trials, interactions between individuals are not biased with respect to their caste.
These graphs show interaction types between individuals for different worker:soldier ratios (Worker-Worker, Worker-Soldier, Soldier-Soldier), which were binned into 12 "situations"/soldier percentages. The expected probability of interactions for each type is sampled from a hypergeometric distribution \([29, 78]\) for the given situation, shown here as dashed lines (\(_*\text{exp}\)). These expected distributions represent the null hypothesis that interactions between individuals happen at random. Each of the 8 conditions per video (different distance thresholds of 0.5 to ~5 times body length) contributes one percentage per interaction type per bin, forming the standard deviations shown. Individuals are only considered “colliding” if they are closer than this threshold right before they disappear at the same time. Percentages for fewer than 5 interactions in a bin were excluded, but they usually comprise tens of thousands.

3.4 Discussion

In the following two subsections we discuss our results, (1) for the updates to TRex and (2) the behavioral results obtained with it, as well as possible future applications of the method and possible improvements to the experimental design for future behavioral essays.

3.4.1 TRex updates

In this paper we described our latest updates to the TRex software, including automatic visual labelling of individuals according to custom-defined categories. The fraction of TRex’ predictions that could be cross-validated with Detectron2's predictions was in agreement for 98% of cases for the clips that were tested. This gives us confidence that most detections are indeed correctly labelled. It is unclear which software was wrong in the remaining 2% of mismatches — without manual evaluation, either result could be correct. We are planning to manually validate a number of detections in the future, especially the 2% of differing detections reported here. A (sparse) manual investigation of the detections revealed that Detectron2 often split two overlapping individuals into three. This
is unsurprising when no individual history is taken into account, but can help to explain - in part - the larger number of total detections in the Detectron2 results. The largest difference between training in Detectron2 and TRex, however, is the way that the training procedure works. In order to use Detectron2, we had to speed up the annotation process via a custom tool. Even when using existing software is used to solve this problem, it is still required to transfer data between the segmentation, tracking, and annotation tools - an additional overhead. Furthermore, since entire clips of individuals are annotated with each click, logically, the training can succeed much more quickly than when using other solutions that work on a single image basis.

It has to be stated that the calculated agreement could be skewed by the worker:soldier ratio. If more than 90% of individuals are workers, then an agreement between detections of both softwares is likely to be less meaningful. However, the changing number of total members of each caste over time and their highly dynamic ratios helps reducing such possible imbalances. We thus conclude that both methods are in high agreement, and that TRex delivers more conservative estimates compared to Detectron2. It is therefore unlikely that both softwares come to the same, wrong conclusions based off separate training datasets, and we can assume the generated classifications are reliable and accurate.

We put forth that it is critical, however, to use tracking information to exclude as many noisy detections as possible. Neural networks tend to classify images according to the categories they are given, and a "none of these" is impossible unless it is part of the training dataset. It has also been shown that faulty injections can even be used to precisely produce intentional misclassifications [133, 231, 17]. For example, mistakenly tracked arena edges are often classified as "soldiers" in TRex. The joint effort of TRex' tracking and the internal machine learning based classification allows for specific, in-depth filtering and various types of post-processing that can be customized to avoid such issues.

Categorization based on visual features in TRex is also not specific to termites. For example, it is theoretically possible to use it to classify animal poses as well. For this, consecutive segments should be broken up artificially after a maximum duration (e.g. 25% the length of the shortest behavior of interest), so that training sample sequences are limited to the correct time scales. These can then be labelled appropriately. If segments are not broken up, then each presented sequence may contain an arbitrary number of random behaviors which makes them impossible to label.

Another possible application is to use categorization as a step prior to the actual tracking: If there is a lot of noise present in the image, which cannot be filtered out based on normal image processing (e.g. due to visual similarity to the studied organism), the network can be taught to differentiate between individuals and objects. In a second tracking step, the noise particles can be ignored based on
the results of that. We are planning to implement similar functions for TRex.

The dataset we generated is very large, and working with such large files (>30GB of filtered positional data for e.g. N21) is still very challenging, but allows for a (previously impossible) quantitative exploration of termite behavior. Improvements in technology will continue increasing data-throughput and -capacity, and we plan on employing larger systems in the short term to analyse even larger datasets. However, only faster algorithms that change the complexity of the problems will help in the long term to significantly reduce the time spent before starting an actual evaluation of results.

3.4.2 Behavior

The termites in our videos behaved similarly to our expectations in many ways, and surprisingly in others. Their movement speed was similar to the speeds reported in previous studies. Movement speeds have been reported in [52] to average 0.87±0.02cm/s for workers and 0.93±0.02cm/s for soldiers, showing workers to be consistently slower than soldiers across treatments. This is largely in agreement with our results, where we measure similar means for soldiers (0.93cm/s vs. 0.95cm/s) and workers (0.87cm/s vs. 0.88cm/s). However, since individuals in their study were confined to petri dishes, and in our trials have much more space available to move in, this difference is not surprising. Since the individuals in [52] were manually tracked using the MouseTracer software (as in [196]), and so were limited in the number of individuals and duration of trials, it is possible that we simply had more samples spanning a vastly increased number of samples of different behaviors in a larger arena.

Comparing between the castes, mean speeds of workers (0.88±0.88cm/s) and soldiers (0.95±1.03cm/s) were very similar. As suggested by the large standard deviations (and Figure 7), individuals are able to move quickly as well (99th percentile of workers: 3.55cm/s; soldiers: 4.14cm/s), and do so intermittently - for example, the individual shown in Figure 1 moves, even when measured visually from the image alone, much faster than the average speed of workers (in this case ~1.9cm/s). It was also unexpected that there was such little difference between the castes in terms of speed. Overall, we were likely unable to sample their entire behavioral repertoire. Termites in an alarmed state for example, for which there was no obvious trigger in our trials, might behave differently. During an alarm response, as in [52], one could imagine soldiers to react more strongly than workers, and thus to move relatively more quickly (as in their study), but we were unable to confirm this here (compare also Figure 7, where both distributions are similar apart from a higher peak at zero for soldiers). However, more aggressive behavior can also more strongly influence other results, like the relative proportions of inter- and intra-caste interactions (WW, WS, and SS), or the recruitment of additional soldiers.
The trial N19 was particularly interesting, since it matched little of what other trials suggested. The ratio of soldiers to workers was flipped compared to the other trials, where workers were usually more numerous, and, mentioned, not many individuals emerged in total. Additionally, logically speaking, when individuals are in close proximity to each other and attempt to move quickly, they should collide with others more often, simply due to chance. This should make high speeds nearly impossible to achieve - and so, as the video containing most individuals, the speed of workers in N21 (0.77cm/s) and their mean distance to the trajectory mean (0.33cm) is slightly lower than in other videos (e.g. 0.51cm). N19, however, counterintuitively, has the lowest number of individuals but they also moved more slowly on average (0.81cm/s, and 0.56cm/s) despite having more room to move in than individuals in more densely populated videos (compare Table 4). If soldiers paused more often than workers, this would give a possible explanation (due to the higher proportion of soldiers in N19), but this is cannot be the case. We do not find differences in the mean dispersion distances of workers and soldiers (= mean distance to the trajectory centroid, see Per-caste movement speed). Especially in N19, both castes were not moving much (similarly to N21). However, we cannot rule out that extremely tight clustering of individuals within a single trail in N19 counter-acts the overall smaller number of individuals here.

The percentage of detections inside the <0.01cm/s and >4cm/s speeds was slightly larger for soldiers than for workers, suggesting that they move at high speeds or very low speeds more often. Higher and more frequent top-speeds in soldiers suggests that they may have much more potential to move even faster, if required. The larger standard deviation in mean dispersion could be interpreted to confirm this — that is, assuming that the distance travelled at high speeds counter-acts more frequent pauses and thus reporting similar means with larger standard deviations.

Interactions between individuals are also unbiased, suggesting that individuals do not interact more often with members of either their own or the other caste, and that there is likely not a preference regarding spatial proximity. This seems to make sense since the apterous lines of C. cyphergaster (i.e. workers and soldiers) do not have eyes to guide them, but also excludes (1) other forms of sensors like odour and (2) caste-specific clusters or patterns of movement, which could theoretically have influenced the probabilities of possible interactions. A possible further implication is that of worker-soldier interaction with regards to nourishment. Since soldiers of C. cyphergaster need to be fed by workers, these results suggest that either (1) the behavior changes specifically when workers have found food or (2) that feeding events occur only when individuals meet randomly. (1) seems unlikely due to the sensory restrictions, but can not be ruled out.

Overall, our results did not demonstrate any large inter-caste differences in the metrics we ob-
Figure 9: “Fencing” by soldiers (in red), seemingly prohibiting individuals in N17 to go towards certain areas. As of yet, we do not have an explanation for this but assume it may be related to the defensive “trail protection” observed in various studies of multiple termite species (see [216, 215, 148, 147]).

...erved, especially not in terms of kinematics. The calculated caste ratios may be an exception to this, pointing towards potential behavioral differences undetected by our set of comparisons: While the ratio between workers and soldiers changes constantly, this is mainly due to the proportion of workers shrinking. The number of soldiers remains remarkably stable throughout all videos, while the number of workers usually peaks early and then decreases shortly after. Soldier trends follow the worker trends, but only to a certain degree and with lag (e.g. supporting a hypothesis that soldier recruitment occurs in reaction to increased worker presence). This is in agreement with studies on the foraging activity of Nasutitermes costalis (in Traniello [216], Miura & Matsumoto [147]), where similarly stable numbers of soldiers were observed for multiple hours. Although there, unlike in our videos, worker numbers steadily increased until the end of the observations. The availability of food in [216], whereas there was no food to be found in our arenas, motivates the conclusion that workers adjust their behavior in response to the availability of food inside the arena (also suggested in [215]). Analogously to this, soldiers might want to occupy the area and stay outside as long as possible to — so to say — secure the area from competing colonies. Miura & Matsumoto [147] similarly report soldier ratios to suddenly increase at the end of a foraging bout, when workers return and soldiers remain. Soldiers in multiple termite species also display similarly defensive behavior, e.g. guarding both sides of the trails ([216, 215, 148]), often being the first to leave and the last to return to the nest.
Termites truly are an interesting model system, and many behavioral features remain yet unexplored — for example we noticed “fencing” shown by the soldiers of this species (see Figure 9), which we want to explore in combination with a more complete set of comparative metrics. Future experimental designs should also, and especially, explore different environments, which can help triggering behaviors that we did not see in our experiments (such as feeding, alarm responses and navigation of more difficult terrain). There, results like the interaction bias and movement speed can be reevaluated, which should also and especially be done with regards to trail formation - an important part of termite behavior that we did not investigate further in this paper. We merely presented the technology required to enable such investigations in the future, as well as a quantitative description of the baseline behavior of this species that, we hope future studies will build upon.
Concluding remarks and perspectives

My thesis has aimed to make both methodological and biological contributions to the field of animal behavior, which I shall summarize below:

In Chapter 1, I presented improvements to some of the most essential tools used in animal behavior research today, united in an open-source and freely available tool called TRex. Tracking is increasingly required for many studies of behavior, cell biology, behavioral ecology, and even particle physics. Due to this need, and the reasons detailed in the introduction, tracking algorithms have been implemented many times. Despite this, we were able to achieve better results across the board, while also adding support for interactive, closed-loop experiments and even larger groups. One of the most important contributions of this chapter, especially for research in animal behavior, was the much improved automatic visual identification of individuals - an essential component that was previously missing in most tracking software. To unify all methods, but also because the only comparable software (in terms of approach and capacity) available at the time was not designed for (and consequently had problems with) trials longer than a few minutes, a reimplementation was necessary. Inspired by idtracker.ai’s algorithm, but changing almost every part of it, my implementation was shown to be superior across the board (both in terms of RAM usage and speed), while maintaining similar accuracy. These upgrades, which allow researchers to analyse hours, instead of just minutes, of footage in reasonable time, are significant improvements over the previous state-of-the-art. Previously, it was simply impossible to maintain consistent identities for hours. Now, given sufficient video quality, the investigation of long-term individual-level effects in large groups (up to 100 individuals) is possible, and also made more accessible by the graphical user interface of TRex. This has a wide range of possible applications, for example to investigate the effects of learning, parasitism, or genetic manipulation on a (mixed) group of individuals. These types of studies are only possible if identities can be reliably maintained throughout multiple hours of footage, and, optimally, across trials.

In Chapter 2, I focused on automating the exploration of large datasets in, as an example dataset,
a set of field videos recorded of termites (*C. cyphergaster*). Mainly, efforts were directed towards the automatic, and quantitative detection of unusual changes in trajectory data (or anomalies), such as alarm responses, to reveal their frequency, duration, and spatial locations. This new dataset, due to its size, made many under-the-hood optimizations for TRex necessary, some of which I described in this chapter. It also motivated the usage of a compressed representation of trajectories, in the form of heatmaps, which was the basis on which I developed two algorithms for anomaly detection: one based on deep neural networks and another one, as a baseline, on more classical image processing principles. A large-scale meta parameter analysis, to find the model architecture best-suited for the task, revealed that all types of models under investigation performed similarly well under the same conditions, and that input-related parameters (such as lag) played a much bigger role in model performance. The results also suggested that the loss function used (mean absolute error) warrants further investigation, and that theoretically a custom-designed function may improve the consistency (and quality) of results. Alternatively, different types of input data should also be considered (e.g. sparse representations of the heatmap, as well as different data channels). Investigating the detections made by both algorithms I was able to not only locate anomalies in time and space, but also to follow the data trail back to the trajectories and identify the individuals involved in these sequences. This revealed spontaneous accelerations of relatively small groups of individuals, which sometimes propagated throughout the entire arena. These bursts of activity are certainly an interesting feature of termite behavior and warrant further inquiry under controlled conditions, especially since, due to our primary focus on methodology, the behavioral analysis was limited by experimental constraints. Nevertheless, our first steps constitute an interesting basis for further investigation and are, due to the use of versatile machine learning methods, also adaptable to entirely different studies and model organisms.

In Chapter 3, I presented an algorithm for the automatic categorization of individuals based on their visual features. Instead of maintaining identities as in Chapter 1, but similarly to it, this algorithm assigns user-defined labels to all individual detections. It builds upon this algorithm from Chapter 1, but with an entirely different, partly-supervised, training procedure. Requiring the user to annotate training data could be interpreted as a “step down” from the unsupervised method in Chapter 1, but it is important not to understate the significance of this addition. Arguably, it was essentially the “missing piece” in TRex because, at the cost of minimal user involvement, it removes the only requirement of its automatic visual identification algorithm: that all individuals are, at least once, visible at the same time. Using this new addition to the toolchain, it becomes possible to label more than 600 million individual detections of *C. cyphergaster* across multiple videos according to their morphological type (i.e. their caste). Since both castes (“workers” and “soldiers”) take up dif-
ferent tasks in the colony, investigations of their presumed behavioral differences are essential for a better understanding of this largely-unexplored species. Over the course of multiple hours of video, we observe interesting dynamics in the relative and absolute numbers of individuals. While there were slight differences in kinematic properties such as speed, both the simple and more the complicated movement metrics (e.g. approximative tactile interactions) show only minor inter-caste differences. The only noticable trends could be found in their temporal dynamics, where soldiers remained almost constant in number when worker numbers were already rapidly decreasing. This suggests that differences between the castes exist, which were undetectable with the metrics that were employed. In future studies, possible differences may be exposed by a more complete set of comparative metrics as well as, more specifically, further investigation into their clusters, and other resting behaviors, and their spatial distribution with respect to trails. This will require the development of additional methods for segmentation and estimation of positions within densely clustered individuals.

Overall, my thesis spans a wide range of methodological topics, which are increasingly becoming key aspects of many studies in collective behavior and other fields of research. The heavy focus on computer science demonstrates how interdisciplinary much of the work in our field has become and, above all, how important this cooperation is. Unsurprisingly, I was unable to created a unified solution to all of our problems. However, I addressed some of the most fundamental problems with a number of new, or greatly improved, methods, combined them into a usable software and, thus, succeeded in a key objective that I set for myself in the beginning: to create a single, practical toolkit and fill it with some of the most vital instruments to conduct and analyse experiments in animal behavior research.

Still, many parts of the methodological map remain unexplored and open for future investigations. Especially important, in my opinion, are more efficient algorithms for (1) machine learning based segmentation in sparsely populated videos (since videos from the field are generally harder to segment), and (2) maintaining identities in situations where the individuals may change. Existing approaches for (1) generally work on the basis of whole images, e.g. repeatedly searching for bounding boxes until there are no more to find, but it should be possible to use faster classical image processing and use machine learning around it to refine detections. Adaptively using machine learning, e.g. in Chapter 1, has proven itself useful as a way of dealing with large numbers of detections and very large datasets in Chapter 3. There are still improvements to be made in terms of implementation, especially in Chapter 3, but just the possibility of such large-scale studies is already a big leap. To maintain identities (2) in contexts where it cannot be guaranteed that all individuals are visible at the same time at least once, since our approach merely learns to differentiate between a group of specific individuals. It does
not constitute a problem if individuals leave, but it does if new individuals join. An example of this could be a macro recording of an ant trail, where individuals move in and out of view and the same ones never appear twice. Here, it is likely that a totally different approach has to be used. Additionally to learning individual identities, which still has to be done if correct identities of e.g. out-of-view individuals need to be recovered, the most important thing for many studies is to better, and quickly, maintain identities in the short-term. For example by using (visual) similarity of overlapping individuals before and after they crossed to resolve them, which is arguably closer to what humans do in manual tracking tasks: looking at each crossing and resolving it based on movement/looks, but not by learning what an individual’s back looks like. Learning the identities of all individuals could be possible for a human, but it would certainly only be the last resort. In the future, I think, it will also increasingly be important to address open questions regarding the study of behavior in the wild (as some methods already do, e.g. herdhover.com), which can also help to forever settle the centuries-old conflict of “laboratory vs. nature” by, essentially, joining the two.

There is certainly much left to do.
A.1 Installation requirements and Usage

Compiled, ready-to-use binaries are available for all major operating systems (Windows, Linux, MacOS). However, it should be possible to compile the software yourself for any Unix- or Windows-based system (≥ 8), possibly with minor adjustments. Tested setups include:

- Windows, Linux, MacOS
- A computer with ≥ 16GB RAM is recommended
- OpenCV\textsuperscript{1} libraries ≥ v3.3
- Python libraries ≥ v3.6, as well as additional packages such as:
  - Keras ≈ v2.2 with one of the following backends installed
    - Tensorflow < v2\textsuperscript{2} (either CPU-based, or GPU-based)

\textsuperscript{1}opencv.org
\textsuperscript{2}tensorflow.org
- Theano

- GPU-based recognition requires an NVIDIA graphics-card and drivers (see Tensorflow documentation)

For detailed download/installation instructions and up-to-date requirements, please refer to the documentation at trex.run/install.

A.1.1 Workflow

TREx can be opened in one of two ways: (i) Simply starting the application (e.g. using the operating systems’ file-browser), (ii) using the command-line. If the user simply opens the application, a file opening dialog displays a list of compatible files as well as information on a selected files content. Certain startup parameters can be adjusted from within the graphical user-interface, before confirming and loading up the file (see A.2). Users with more command-line experience, or the intent of running TREx in batch-mode, can append necessary parameter values without adding them to a settings file.

To acquire video-files that can be opened using TREx, one needs to first run TGrabs in one way or another. It is possible to use a webcam (generic USB camera) for recording, but TGrabs can also be compiled with Basler Pylon5 support. TGrabs can also convert existing videos and write to a more suitable format for TREx to interact with (a static background with moving objects clearly separated in front of it). It can be started just like TREx, although most options are either set via the command-line, or a web-interface. TGrabs can perform basic tracking tasks on the fly, offering closed-loop support as well.

For automatic visual recognition, one might need to adjust some parameters. Mostly, these adjustments consist of changing the following parameters:

- blob_size_ranges: Setting one (or multiple) size thresholds for individuals, by giving lower and upper limit value pairs.
- track_max_individuals: Sets the number of individuals expected in a trial. This number needs to be known for recognition tasks (and will be guessed if not provided), but can be set to 0 for unknown numbers of individuals.

---

3 deeplearning.net
4 The baslerweb.com Pylon SDK is required to be installed to support Basler USB cameras.
- `track_max_speed`: Sets the maximum speed (cm/s) that individuals are expected to travel at. This is influenced by meta information provided to TGrabs by the user (e.g. the width of the tank), as well as frame timings.

- `track_threshold`: Even TRex can threshold images of individuals, so it is beneficial to not threshold away too many pixels during conversion/recording and do finer-grade adjustments in the tracker itself.

- `outline_resample`: A factor that is $>0$, by which the number of points in the outline is essentially "divided". Smaller resample rates lead to more points on the outline (good for very small shapes).

Training can be started once the user is satisfied with the basic tracking results. Consecutive segments are highlighted in the time-line and suggest better or worse tracking, based on their quantity and length. Problematic segments of the video are highlighted using yellow bars in that same timeline, giving another hint to the user as to the tracking quality. To start the training, the user just clicks on "train network" in the main menu – triggering the accumulation process immediately. After training, the user can click on "auto correct" in the menu and let TRex correct the tracks automatically (this will re-track the video). The entire process can be automated by adding the "auto_train" parameter to the command-line, or selecting it in the interface.

A.1.2 Output

Once finished, the user may export the data in the desired format. Which parts of the data are exported is up to the user as well. By default, almost all the data is exported and saved in NPZ files in the output folder.

Output folders are structured in this way:

- **output folder**:
  - Settings files
  - Training weights
  - Saved program states
  - **data folder**:
    - Statistics
    - All exported NPZ files (named `{video_name}_fish[number].npz` – the prefix "fish" can be changed).
frames folder (contains video clips recorded in the GUI, e.g. for presentations):

- [video name] folder
  - clip[index].avi
  - ...

At any point in time (except during training), the user can save the current program state and return to it at a later time (e.g. after a computer restart).

A.1.2.1 Export options

After individuals have been assigned by the matching algorithm, various metrics are calculated (depending on settings):

- **Angle**: The angle of an individual can be calculated without any context using image moments ([100]). However, this angle is only reliable within 0 to 180 degrees – not the full 360. Within these 180 degrees it is probably more accurate than is movement direction.

- **Position**: Centroid information on the current, as well as the previous position of the individual are maintained. Based on previous positions, velocity as well as acceleration are calculated. This process is based on information sourced from the respective video file or camera on the time passed between frames. The centroid of an individual is calculated based on the mass center of the pixels that the object comprises. Angles calculated in the previous steps are corrected (flipped by 180 degrees) if the angle difference between movement direction and angle + 180 degrees is smaller than with the raw angle.

- **Posture**: A large part of the computational complexity comes from calculating the posture of individuals. While this process is relatively fast in TRex, it is still the main factor (except with many individuals, where the matching process takes longest). We dedicated a subsection to it below.

- **Visual Field**: Based on posture, rays can be cast to detect which animal is visible from the position of another individual. We also dedicated a subsection to visual field further down.

- **Other** features can be computed, such as inter-individual distances or distance to the tank border. These are optional and will only be computed if necessary when exporting the data. A (non-comprehensive) list of metrics that can be exported follows:
- Time: The time of the current frame (relative to the start of the video) in seconds.
- Frame: Index of the frame in the PV video file.
- Individual components of position its derivatives (as well as their magnitudes, e.g. speed)
- Midline offset: The center-line, e.g. of a beating fish-tail, is normalized to be roughly parallel to the x-axis (from its head to a user-defined percentage of a body). The y-offset of its last point is exported as a “midline offset”. This is useful, e.g. to detect burst-and-glide events.
- Midline variance: Variance in midline offset, e.g. for detection of irregular postures or increased activity.
- Border distance
- Average neighbour distance: Could be used to detect individuals who prefer to be located far away from the others or are avoided by them.

Additionally, tracks of individuals can be exported as a series of cropped-out images – a very useful tool if they are to be used with an external posture estimator or tag-recognition. This series of images can be either every single image, or the median of multiple images (the time-series is down-sampled).
**Figure A1:** Using the interactive heatmap generator within TRex, the foraging trail formation of *Constrictotermes cypher-gaster* (termites) can be visualized during analysis, as well as other potentially interesting metrics (based on posture- as well basic positional data). This is generalizable to all output data fields available in TRex, e.g. also making it possible to visualize "time" as a heatmap and showing where individuals were more likely to be located during the beginning or towards end of the video. *Video: H. Hugo*
Figure A2: The file opening dialog. On the left is a list of compatible files in the current folder. The center column shows meta-information provided by the video file, including its frame-rate and resolution – or some of the settings used during conversion and the timestamp of conversion. The column on the right provides an easy interface for adjusting the most important parameters before starting up the software. Most parameters can be changed later on from within TRex as well.
A.2 From video frame to blobs

Video frames can originate either from a camera, or from a pre-recorded video file saved on disk. 
TGrabs treats both sources equally, the only exception being some minor details and that pre-recorded 
videos have a well-defined end (which only has an impact on MP4 encoding). Multiple formats are 
supported, but the full list of supported codecs depends on the specific system and OpenCV ver-
sion installed. TGrabs saves images in RAW quality, but does not store complete images. Merely the 
objects of interest, defined by common tracking parameters such as size, will actually be written to 
a file. Since TGrabs is mostly meant for use with stable backgrounds (except when contrast is good 
or a video-mask is provided), the rest of the area can be approximated by a static background image 
generated in the beginning of the process (or previously).

Generally, every image goes through a number of steps before it can be tracked in TRex:

1. Images are decoded by either (i) a camera driver, or (ii) OpenCV. They consist of an array 
of values between 0 and 255 (grayscale). Color images will be converted to grayscale images 
(color channel or “hue” can be chosen).

2. Timing information is saved and images are appended to a queue of images to be processed.

3. All operations from now on are performed on the GPU if available. Once images are in the 
queue, they are picked one-by-one by the processing thread, which performs operations on 
them based on user-defined parameters:

- Cropping
- Inverting
- Contrast/brightness and lighting corrections
- Undistortion (see OpenCV Tutorial)

4. (optional) Background subtraction \(d(x) = b(x) - f(x)\), with \(f\) being the image and \(b\) the 
background image), leaving a difference image containing only the objects. This can be an 
absolute difference \(|b(x) - f(x)|\) or a signed one, which has different effects on the following 
step. Otherwise \(d(x) = f(x)\)

5. Thresholding to obtain a binary image, with all pixels either being 1 or 0:

\[
t(x) = \begin{cases} 
0 & d(x) < T \\
1 & d(x) \geq T 
\end{cases}
\]
where $0 \leq T \leq 255$ is the threshold constant.

6. Options are available for further adjustment of the binary image: Dilation, Erosion and Closing are used to close gaps in the shapes, which are filled up by successive dilation and erosion operations (see A.3). If there is an imbalance of dilation and erosion commands, noise can be removed or shapes made more inclusive.

7. The original image is multiplied by the thresholded image, obtaining a masked grayscale image: $t(x) \cdot f(x)$, where $\cdot$ is the element-wise multiplication operator.

At this point, the masked image is returned to the CPU, where connected components (objects) are detected. A connected component is a number of adjacent pixels with color values greater than zero. Algorithms for connected-component labeling either use a 4-neighborhood or an 8-neighborhood, which considers diagonal neighbors to be adjacent as well. Many such algorithms are available ([2], [36], and many others), even capable of real-time speeds ([205], [96]). However, since we want to use a compressed representation throughout our solution, as well as transfer over valuable information to integrate it with posture analysis, we needed to implement our own (see Connected components algorithm).

**MP4** encoding has some special properties, since its speed is mainly determined by the external encoding software. Encoding at high-speed frame-rates can be challenging, since we are also encoding to a PV-file simultaneously. Videos are encoded in a separate thread, without muxing, and will be remuxed after the recording is stopped. For very high frame-rates or resolutions, it may be necessary to limit the duration of videos since all of the images have to be kept in RAM until they have been encoded. RAW images in RAM can take up a lot of space ($1024 \times 1024 \times 1000 = 1,048,576,000$ bytes for 1000 images quite low in resolution). If there a recording length is defined prior to starting the program, or a video is converted to PV and streamed to MP4 at the same time (though it is unclear why that would be necessary), TGrabs is able to automatically determine which frame-rate can be maintained reliably and without filling the memory.
Figure A3: Example of morphological operations on images: "Erosion". Blue pixels denote on-pixels with color values greater than zero, white pixels are "off-pixels" with a value equal to zero. A mask is moved across the original image, with its center (dot) being the focal pixel. A focal pixel is retained if all of the on-pixels within the structure element/mask are on top of on-pixels in the original image. Otherwise the focal pixel is set to 0. The type of operation performed is entirely determined by the structure element.

A.3 Connected components algorithm

Pixels are not represented individually in TRex. Instead, they are saved as connected horizontal line segments. For each of these lines, only y- as well as start- and end-position are saved (y, x₀ and x₁).

This representation is especially suited for objects stretching out along the x-axis, but of course its worst-case is a straight, vertical line – in which case space requirements are $O(2 \times N)$ for $N$ pixels. Especially for big objects, however, only a fraction of coordinates has to be kept in memory (with a space requirement of $O(2 \times H)$ instead of $O(W \times H)$, with $W, H$ being width and height of the object).

Extracting these connected horizontal line segments from an image can be parallelized easily by cutting the image into full-width pieces and running the following algorithm repeatedly for each row:

1. From 0 to $W$, iterate all pixels. Always maintain the previous value (binary), as well as the current value. We start out with our previous value of $p = 0$ (the border is considered not to be an object).

2. Now repeat for every pixel $p_i$ in the current row:

   (a) If $p$ is 1 and $p_i$ is 0, set $p := 0$ and save the position as the end of a line segment $x_1 = i - 1$. 

124
If $p$ is 0 and $p_i$ is 1, we did not have a previous line segment and a new one starts. We save it as our current line segment with $x_0$ and $y$ equal to the current row. Set $\overline{p} := 1$.

3. After each row, if we have a valid current line, we save it in our array of lines. If $\overline{p} = 1$ was set, and the line segment ended at the border $W$ of the image, we first set its end position to $x_1 := W - 1$.

We keep the array of extracted lines sorted by their y-coordinate, as well as their x-coordinates in the order we encountered them. To extract connected components, we now just need to walk through all extracted rows and detect changes in the y-coordinate. The only information needed are the current row and the previous row, as well as a list of active preliminary "blobs" (or connected components). A blob is simply a collection of ordered horizontal line segments belonging to a single connected component. These blobs are preliminary until the whole image has been processed, since they might be merged into a single blob further down despite currently being separate (see $A_4$).

"Rows" are an array of horizontal lines with the same y-coordinate, ordered by their x-coordinates (increasing). The following algorithm only considers pairs of previous row $R_{i-1}$ and current row $R_i$. We start by inserting all separate horizontal line segments of the very first row into the pool of active blobs, each assigned their own blob. Lines within row $R_i$ are $L_{i,j}$. Coordinates of $L_{i,j}$ will be denoted as $x_0(i,j)$, $x_1(i,j)$ and $y(i,j)$. Our current index in row $R_{i-1}$ is $j$ and our index in row $R_i$ is $k$. We initialize $j := 0$, $k := 1$. Now for each pair of rows, three different actions may be required depending on the case at hand. All three actions are hierarchically ordered and mutually exclusive (like a typical if/else structure would be), meaning that case 0-2 can be true at the same time while no other combination can be simultaneously true:

1. **Case 0,1 and 2: We have to create a new blob.** This is the case if (0) the line in $R_i$ ends before the line in $R_{i-1}$ starts ($x_0(i,k) + 1 < x_0(i,j)$), or (1) y-coordinates of $R_i$ and $R_{i-1}$ are farther apart than 1 ($y(i-1,j) > y(i,k) + 1$), or (2) there are no lines left in $R_{i-1}$ to match the current line in $R_i$ to ($j \geq |R_{i-1}|$). $L_{i,j}$ is assigned with a new blob.

2. **Case 3: Segment in the previous row ends before the segment in the current row starts.**

If $x_0(i,k) > x_1(i-1,j) + 1$, then we just have to $j := j + 1$.

3. **Case 4: Segment in the previous row and segment in the current row intersect in x-coordinates.** If $L_{i,j}$ is no yet assigned with a blob, assign it with the one from $L_{i-1,j}$. Otherwise, both blobs have to be merged. This is done in a sub-routine, which guarantees that lines
within blobs stay properly sorted during merging. This means that (i) y-coordinates increase or stay the same and (ii) x-coordinates increase monotonically. Afterwards, we increase either \( k \) or \( j \) based on which one associated line ends earlier: If \( x_1(i, k) \leq x_1(i - 1, j) \), then we increase \( k := k + 1 \); otherwise \( j := j + 1 \).

After the previous algorithm has been executed on a pair of \( R_{i-1} \) and \( R_i \), we increase \( i \) by one \( i := i + 1 \). This process is continued until \( i = H \), at which point all connected components are contained within the active blob array.

Retaining information about pixel values adds slightly more complexity to the algorithm, but is straight-forward to implement. In TReX, horizontal line segments comprise \( y, x_0 \) and \( x_1 \) values plus an additional pointer. It points to the start of a line within array of all pixels (or an image matrix), adding only little computational complexity overall.

Based on the horizontal line segments and their order, posture analysis can be sped up when properly integrated. Another advantage is that detection of connected components within arrays of horizontal line segments is supported due to the way the algorithm functions – we can just get rid of the extraction phase.

![Figure A4](image.png)

**Figure A4**: An example array of pixels, or image, to be processed by the connected components algorithm. This figure should be read from top to bottom, just as the connected components algorithm would do. When this image is analysed, the red and blue objects will temporarily stay separate within different "blobs". When the green pixels are reached, both objects are combined into one identity.
A.4 Matching an object to an object in the next frame

A.4.1 Terminology

A graph is a mathematical structure commonly used in many fields of research, such as computer science, biology and linguistics. Graphs are made up of vertices, which in turn are connected by edges. Below we define relevant terms that we are going to use in the following section:

- Directed graph: Edges have a direction assigned to them
- Weighted edges: Edges have a weight (or cost) assigned to them
- Adjacent nodes: Nodes which are connected immediately by an edge
- Path: A path is a sequence of edges, where each edges starting vertex is the end vertex of the previous edge
- Acyclic graph: The graph contains no path in which the same vertex appears more than once
- Connected graph: There are no vertices without edges, there is a path from any vertex to any other vertex in the graph
- Bipartite graph: Vertices can be sorted into two distinct groups, without an edge from any vertex to elements of its own group – only to the other group
- Tree: A tree is a connected, undirected, acyclic graph, in which any two vertices are only connected by exactly one path
- Rooted, directed out-tree: A tree where one vertex has been defined to be the root and directed edges, with all edges flowing away from the root
- Visited vertex: A vertex that is already part of the current path
- Leaf: A vertex which has only one edge arriving, but none going out (in a tree this are the bottom-most vertices)
- Depth-first/breadth-first and best-first search: Different strategies to pick the next vertex to explore for a set of paths with traversable edges. Depth-first prefers to first go deeper inside a graph/tree, before going on to explore other edges of the same vertex. Breadth-first is the opposite of depth-search. Best-first search uses strategies to explore the most promising path first.
A.4.2 Background

The transportation problem is one of the fundamental problems in computer science. It solves the problem of transporting a finite number of goods to a finite number of factories, where each possible transport route is associated with a cost (or weight). Every factory has a demand for goods and every good has a limited supply. The sum of this cost has to be minimized (or benefits maximized), while remaining within the constraints given by supply and demand. In the special case where demand by each factory and supply for each good are exactly equal to 1, this problem reduces to the assignment problem.

The assignment problem can be further separated into two distinct cases: the balanced and the unbalanced assignment problem. In the balanced case, net-supply and demand are the same – meaning that the number of factories matches exactly the number of suppliers. While the balanced case can be solved slightly more efficiently, most practical problems are usually unbalanced ([175]). Thankfully, unbalanced assignments can be reduced to balanced assignments, for example using graph-duplication methods or by adding nodes ([175], [176]). This makes the widely used Hungarian method ([122]; [153]) a viable solution to both, with a computational complexity of $O(n^3)$. It can be further improved using Fibonacci heaps (not implemented in TRex), resulting in $O(ms + s^2 \log n)$ time-complexity ([71]), with $m$ being the number of possible connections/edges, $s \leq n$ the number of factories to be supplied and $n$ the number of factories. Re-balancing, by adding nodes or other structures, also adds computational cost – especially when $s \ll n$ ([176]).

A.4.2.1 Adaptation for our matching problem

Assigning individuals to objects in the frame is, in the worst case, exactly that: an unbalanced assignment problem – potentially with $r \neq s$. During development, we found that we can achieve better average-complexity by combining an approach commonly used to solve NP-hard problems. This is a class problems for which it is (probably) not possible to find a polynomial-time solution. In order to motivate our usage of a less stable algorithm than e.g. the Hungarian method, let us first introduce a more general algorithm, following along with remarks for adapting it to our special case. The next subsection concludes with considerations regarding its complexity in comparison to the more stable Hungarian method.

Branch & Bound (or BnB, [125], formalized in [131]) is a very general approach to traversing the large search spaces of NP-hard problems, traditionally represented by a tree. Branching and bounding gives optimal solutions by traversing the entire search space if necessary, but stopping along the
way to evaluate its options, always trying to choose better branches of the tree to explore next or skip unnecessary ones. BnB always consists of three main ingredients:

1. Branching: The division of our problem into smaller, partial problems

2. Bounding: Estimate the upper/lower limits of the probability/cost gain to be expected by traversing a given edge

3. Selection: Determining the next node to be processed

Finding good strategies is essential and can have a big impact on overall computation time. Strategies can only be worked out with insight into the specific problem, but bounding is generally the dominating factor here – in that choosing good selection and branching techniques cannot make up for a bad bounding function ([40]). A bounding function estimates an upper (or lower) limit for the quality of results that can be achieved within a given sub-problem (current branch of the tree).

The "problem" is the entire assignment problem located at the root node of the tree. The further down we go in the tree, the smaller the partial problems become until we reach a leaf. Any graph can be represented as a tree by duplicating nodes when necessary ([225], "Graph vs. tree"). So even if the bipartite assignment graph (an example sketched in $A5$) is a more "traditional" representation of the assignment problem, we can translate it into a rooted, directed out-tree $T = (U, V, E, F)$ with weighted edges. Here, $U$ are individuals and $V$ are objects in the current frame that are potentially assigned to identities in $U$. $E$ are edges mapping from $U \rightarrow V$, while $F : V \rightarrow U$. It is quite visible from $A5$, that the representation as a tree (b) is much more verbose than a bipartite graph (a).

However, its structure is very simple:

Looking at the tree in $A5$ (b), individuals (blue) are found along the y-axis/deeper into the tree while objects in the frame (orange) are listed along on the x-axis. This includes a "null" case per individual, representing the possibility that it is not assigned to any object – ensuring that every individual has at least one edge.

Tree is never generated in its entirety (except in extreme cases), but it represents all possible combinations of individuals and objects. Overall, the set $Q$ of every complete and valid path from top to bottom would be exactly the same as the set of every valid permutation of pairings between objects (plus null) and individuals. Edge weights in $E$ are equal to the probability $P_i(t, \tau_i | B_j)$ (see equation $A.5$), abbreviated to $P_i(B_j)$ here since we are only ever looking at one time-step. $B_j$ is an object and $i$ is an individual, so we can rewrite it in the current context as $P_u(v)$, with $u \in U; v \in V$.

We are maximizing the objective function

129
\[ o(\rho) = \sum_{u \in \rho} P_u(v), \]

where \( \rho \in Q \) is an element of all valid paths within \( T \).

The simplest approach would be to traverse every edge in the graph and accumulate a sum of probabilities along each path, guaranteeing to find the optimal solution eventually. Since the number of possible combinations \(|U|^{|E|}\) grows rapidly with the number of edges, this is not realistic – even with few individuals. Thus, at least the typical number of visited edges has to be minimized. While we do not know the exact solution to our problem before traversing the graph, we can make very good guesses. For example, we may order nodes in such a way that branching (visiting a node leads to \( > 1 \) new edges to be visited) is reduced in most cases. To do that, we first need to calculate the degree of each individual. The degree \( C_u \) of individual \( u \), which is exactly equivalent to the maximum number of edges going out from that individual, we define as

\[ C_u \in \mathbb{N} := \sum_{u \in U} \begin{cases} 1 & \text{if } P_u(v) > P_{\min} \\ 0 & \text{otherwise} \end{cases}. \]

The maximally probable edge per individual also has to be computed beforehand, defined as

\[ P_u = \max_{v \in V} \{ P_u(v) \}. \]

Nodes are sorted first by their degree (ascending) and secondly by \( P_u \) (descending). We call this ordered set \( S \). Sorting by degree ensures that the nodes with the fewest outgoing edges are visited first, causing severe branching to only happen in the lower regions of the tree. This is preferable, because a new branch in the bottom layer merely results in a few more options. If this happens at the top, the tree is essentially duplicated \( C_u \) times – in one step drastically increasing the overall number of items to be kept in memory. This process is, fittingly, called node sorting ([225]). Sorting by \( P_u \) is only applied whenever nodes of the same degree have to be considered.

We always follow the most promising paths first (the one with the highest accumulated probability), which is called “best-first search” (BFS) – our selection strategy for (1.) in A.4.2.1. BFS is implemented using a queue maintaining the list of all currently expanded nodes.

Regarding (2.) in A.4.2.1, we utilize \( P_u \) as an approximation for the upper bound to the achievable probability in each vertex. For each layer with vertices of \( U \), we calculate an accumulative sum

\[ \text{upper\_limit}(i) = \sum_{j > i \in U} P_j, \]

with \( j, i \) being indices into our ordered set \( S \) of individuals and \( i \) be-
ing the current depth in the graph (only counting vertices of $U$). This hierarchical upper limit for
the expected value does not consider whether the respective edges are still viable, so they could have
been eliminated already by assigning the object of $V$ to another vertex of $U$ above the current one.
Any edge with $P_{\text{current}} + \text{upper limit}(i) < P_{\text{best}}$ is skipped since it can not improve upon our previ-
ous best value $P_{\text{best}}$. If we do find an edge with a better value, we replace $P_{\text{best}}$ with the new value and
continue.

As an example, let us traverse the tree in $A5b$:

- We first calculate $P_u$ for every $u \in U$ ($P_0 = 0.85$; $P_2 = 0.9$; $P_1 = 0.75$), as well as the
  hierarchical probability table upper limit($i$) for each index $0 \leq i < N(0.9 + 0.75; 0.75; 0)$. $P_{\text{best}} := 0$.
- Individual 0 (the root) is expanded, which has one edge with probability $0.85 + \text{upper limit}(0) \geq
  P_{\text{best}}$ to object 3 (plus the null case) and is the only node with a degree of 1. We know that our
  now expanded node is the best, since it has the largest probability due to sorting, plus also is
  the deepest. In fact, this is true for all expanded nodes exactly in the order they are expanded
  (depth-first search == best-first search for our case). We set $P_{\text{best}} := 0.85$. The edge to NIL is
  added to our queue.
- Objects in $V$ are only virtual and always have zero-probability connections to the next individ-
  ual in an ordered set ($f \in F$), so they do not add to the overall probability sum. We skip to the
  next node.
- Individual 2 branches off into one or two different edges, depending on which edges have
  been chosen previously.
- We first explore the edge towards object 4 with a probability of $0.9 + \text{upper limit}(1) = 1.65 \geq P_{\text{best}}$ and add it to $P_{\text{best}}$.
- Only one possibility is left and we arrive at a leaf with an accumulated probability of $0.85 +
  0.9 + 0 = 1.75$.
- We now perform backtracking, meaning we look at every expanded node in our queue, each
time observing $P_u + \text{upper limit}(i)$.
  - NIL (from node 2) would be added to the front of our queue, however its probability
    $0.85 + 0 + \text{upper limit}(1) = 1.6 < 1.75 = P_{\text{best}}$, so it is discarded.
  - NIL (from node 0) would be added now, but its probability of $0 + \text{upper limit}(0) =
    1.65 < P_{\text{best}}$, so it is also discarded.
We can see that with increased depth, we have to keep track of more and more possibilities. Since our nodes and edges are pre-sorted, our path through the tree is optimal after exactly \( N = |U| \) node expansions (not counting \( v \in V \) expansions since they are only "virtual").

### A.4.2.2 Complexity

Utilizing these techniques, we can achieve very good average-case complexity. Of course having a good worst-case complexity is important (such as the Hungarian method), but the impact of a good average-case complexity can be significant as well. This is illustrated nicely by the timings measured in Table A3, where our method consistently surpasses the Hungarian method in terms of performance – especially for very large groups of animals – despite having worse worst-case complexity. Usually, even in situations with over 1000 individuals present, the average number of leaves visited was approximately 1.112 (see Table A5) and each visit was a global improvement (not shown). The number of nodes visited per frame were around 2844 to 19, 804, 880 in the same video, which, given the maximal number of possible combinations \( N^M \) for \( M \) edges and \( N \) individuals \((210)\), is quite moderate. Especially considering the number of calculations that the Hungarian method has to perform in every step, which, according to its complexity, will be in the range of \( N^3 \approx 1e9 \) for \( N = 1024 \) individuals.

The average complexity of a solution using best-first-search BnB is given by \([225]\). It depends on the probability of encountering a "zero-cost edge" \( p_0 \), as well as the mean branching factor \( b \) of the tree:

1. \( \Theta(\beta^N) \) when \( bp_0 < 1 \), with \( \beta \leq b \) and \( N \) is the depth of the tree
2. \( \Theta(N^2) \) when \( bp_0 = 1 \)
3. \( \Theta(N) \) when \( bp_0 > 1 \iff b > 1/p_0 \)

as \( N \to \infty \).

In our case the depth of the tree is exactly the number of individuals \( N \), which we have already substituted here. This is the number of nodes that have to be visited in the best case. A "zero-cost edge" is an edge that does not add any cost to the current path. We are maximizing (not minimizing) so in our case this would be "an edge with a probability of 1". While reaching exactly 1 is improbable, it is (in our case) equivalent to "having only one viable edge arriving at an object". \( p_0 \) depends very much on the settings, specifically the maximum movement speed allowed, and behavior of individuals, which is why in scenarios with \( > 100 \) individuals the maximum speed should always be adjusted.
first. To put it another way: If there are only few branching options available for the algorithm to explore per individual, which seems to be the case even in large groups, we can assume our graph to have a probability \( p_0 \) within \( 0 \ll p_0 \leq 1 \). The mean branching factor \( b \) is given by the mean number of edges arriving at an object (not an individual). Averaging at around \( b \approx k + 1 \), with \( k \geq 1 \) being the average number of assignable blobs per individual (roughly 1.005 in video 0) and 1 the null-case, we can assume \( bp_0 \) to be \( > 1 \) on average. An average complexity of \( O(N^2) \), as long as \( b > 1/p_0 \), is even better than the complexity of the Hungarian method (which is also \( O(N^3) \) in the average-case, [15]), giving a possible explanation for the good results achieved using tree-based matching in TRex on average (Table A3).

Further optimizations could be implemented, e.g. using impact-based heuristics (as an example of dynamic variable ordering) instead of the static and coarse maximum probability estimate used here. Such heuristics first choose the vertex ”triggering the largest search space reduction” ([171]). In our case, assigning an individual first if, for example, it has edges to many objects that each only one other individual is connected to.

![Figure A5: A bipartite graph (a) and its equivalent tree-representation (b). It is bipartite since nodes can be sorted into two disjoint and independent sets (\{0, 1, 2\} and \{3, 4\}), where no nodes have edges to other nodes within the same set. (a) is a straight-forward way of depicting an assignment problem, with the identities on the left side and objects being assigned to the identities on the right side. Edge weights are, in TRex and this example, probabilities for a given identity to be the object in question. This graph is also an example for an unbalanced assignment problem, since there are fewer objects (orange) available than individuals (blue). The optimal solution in this case, using weight-maximization, is to assign 0 \( \rightarrow 3 \); 2 \( \rightarrow 4 \) and leave 1 unassigned. Invalid edges have been pruned from the tree in (b), enforcing the rule that objects can only appear once in each path. The optimal assignments have been highlighted in red.](image-url)
Figure A6: The same set of videos as in Table 5 pooled together, we evaluate the efficiency of our crossings solver. Consecutive frame segments are sequences of frames without gaps, for example due to crossings or visibility issues. We find these consecutive frame segments in data exported by TRex, and compare the distribution of segment-lengths to idtracker.ai’s results (as a reference for an algorithm without a way to resolve crossings). In idtracker.ai’s case, we segmented the non-interpolated tracks by missing frames, assuming tracks to be correct in between. The Y-axis shows the percentage of $\sum_{k \in [1,V]} \text{video\_length}_k \times \#\text{individuals}_k$ in $V$ videos that one column makes up for – the overall coverage for TRex was 98%, while idtracker.ai was slightly worse with 95.17%. Overall, the data distribution suggests that, probably due to it attempting to resolve crossings, TRex seems to produce longer consecutive segments.

A6–source data 1. A list of all consecutive frame segments used in A6. In the table, they are indexed by their length, the software they were produced by, the video they originate from, as well as they bin they belong to.

Figure A7: Mean values of processing-times and 5%/95% percentiles for video frames of all videos in the speed dataset (Table 1), comparing two different matching algorithms. Parameters were kept identical, except for the matching mode, and posture was turned off to eliminate its effects on performance. Our tree-based algorithm is shown in green and the Hungarian method in red. Grey numbers above the graphs show the number of samples within each bin, per method. Differences between the algorithms increase very quickly, proportional to the number of individuals. Especially the Hungarian method quickly becomes very computationally intensive, while our tree-based algorithm shows a much shallower curve. Some frames could not be solved in reasonable time by the tree-based algorithm alone, at which point it falls back to the Hungarian algorithm. Data-points belonging to these frames ($N = 79$) have been excluded from the results for both algorithms. One main advantage of the Hungarian method is that, with its bounded worst-case complexity (see Matching an object to an object in the next frame), no such combinatorial explosions can happen. However, even given this advantage the Hungarian method still leads to significantly lower processing speed overall (see also appendix Table A3).

A7–source data 1. Raw data for producing this figure and A3. Each sample is represented as a row here, indexed by method (tree, approximate, hungarian), video and the bin (horizontal line in this figure).
### Table A1: showing quantiles for frame timings for videos of the speed dataset (without posture enabled).

Videos 15, 16 and 14 each contain a short sequence of taking out the fish, causing a lot of big objects and noise in the frame. This leads to relatively high spikes in these segments of the video, resulting in high peak processing timings here. Generally, processing time is influenced by a lot of factors involving not only TRex, but also the operating system as well as other programs. While we did try to control for these, there is no way to make sure. However, having sporadic spikes in the timings per frame does not significantly influence overall processing time, since it can be compensated for by later frames.

We can see that videos of all quantities ≤ 256 individuals can be processed faster than they could be recorded. Videos that can not be processed faster than real-time are underlaid in gray.

<table>
<thead>
<tr>
<th>video</th>
<th># ind.</th>
<th>ms / frame</th>
<th>5%</th>
<th>mean</th>
<th>95%</th>
<th>max</th>
<th>&gt; real-time</th>
<th>% video length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1024</td>
<td>25.0</td>
<td>46.93</td>
<td>62.96</td>
<td>119.54</td>
<td>849.16</td>
<td>100.0%</td>
<td>358.12</td>
</tr>
<tr>
<td>1</td>
<td>512</td>
<td>20.0</td>
<td>19.09</td>
<td>29.26</td>
<td>88.57</td>
<td>913.52</td>
<td>92.11%</td>
<td>259.92</td>
</tr>
<tr>
<td>2</td>
<td>512</td>
<td>16.67</td>
<td>17.51</td>
<td>26.53</td>
<td>36.72</td>
<td>442.12</td>
<td>97.26%</td>
<td>235.39</td>
</tr>
<tr>
<td>3</td>
<td>256</td>
<td>20.0</td>
<td>8.35</td>
<td>11.28</td>
<td>13.25</td>
<td>402.54</td>
<td>1.03%</td>
<td>77.18</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>16.67</td>
<td>8.04</td>
<td>11.62</td>
<td>13.48</td>
<td>394.75</td>
<td>1.13%</td>
<td>94.77</td>
</tr>
<tr>
<td>5</td>
<td>128</td>
<td>16.67</td>
<td>3.54</td>
<td>5.14</td>
<td>5.97</td>
<td>367.92</td>
<td>0.41%</td>
<td>40.1</td>
</tr>
<tr>
<td>6</td>
<td>128</td>
<td>16.67</td>
<td>3.91</td>
<td>5.64</td>
<td>6.89</td>
<td>381.51</td>
<td>0.51%</td>
<td>44.38</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>31.25</td>
<td>2.5</td>
<td>3.57</td>
<td>5.19</td>
<td>316.75</td>
<td>0.1%</td>
<td>28.35</td>
</tr>
<tr>
<td>8</td>
<td>59</td>
<td>19.61</td>
<td>1.43</td>
<td>2.29</td>
<td>3.93</td>
<td>2108.77</td>
<td>0.19%</td>
<td>16.33</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>40.0</td>
<td>0.4</td>
<td>0.52</td>
<td>1.67</td>
<td>4688.5</td>
<td>0.01%</td>
<td>2.96</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10.0</td>
<td>0.28</td>
<td>0.33</td>
<td>0.57</td>
<td>283.7</td>
<td>0.07%</td>
<td>8.08</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>31.25</td>
<td>0.21</td>
<td>0.25</td>
<td>0.65</td>
<td>233.7</td>
<td>0.01%</td>
<td>3.48</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>31.25</td>
<td>0.23</td>
<td>0.27</td>
<td>0.75</td>
<td>225.63</td>
<td>0.02%</td>
<td>2.82</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>31.25</td>
<td>0.22</td>
<td>0.25</td>
<td>0.54</td>
<td>237.32</td>
<td>0.02%</td>
<td>2.64</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>33.33</td>
<td>0.24</td>
<td>0.29</td>
<td>0.66</td>
<td>172.6</td>
<td>0.02%</td>
<td>1.8</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>40.0</td>
<td>0.22</td>
<td>0.26</td>
<td>0.88</td>
<td>244.88</td>
<td>0.01%</td>
<td>1.5</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>28.57</td>
<td>0.18</td>
<td>0.21</td>
<td>0.51</td>
<td>1667.14</td>
<td>0.02%</td>
<td>1.38</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>7.14</td>
<td>0.03</td>
<td>0.04</td>
<td>0.06</td>
<td>220.81</td>
<td>0.01%</td>
<td>1.56</td>
</tr>
</tbody>
</table>

**A1—source data 1.** Raw samples for this table and A5.
<table>
<thead>
<tr>
<th>video</th>
<th># ind.</th>
<th>length</th>
<th>total</th>
<th>excluded</th>
<th>wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>100</td>
<td>1min</td>
<td>717</td>
<td>22</td>
<td>45 (6.47%)</td>
</tr>
<tr>
<td>8</td>
<td>59</td>
<td>10min</td>
<td>279</td>
<td>146</td>
<td>100 (55 (41.35%)</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>1h0min</td>
<td>838</td>
<td>70</td>
<td>22 (110 (13.02%)</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>10min3s</td>
<td>331</td>
<td>22</td>
<td>22 (36 (11.65%)</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>10min3s</td>
<td>382</td>
<td>42</td>
<td>43 (83 (24.41%)</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>10min10s</td>
<td>1067</td>
<td>50</td>
<td>52 (73 (7.18%)</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>3h15min22s</td>
<td>7424</td>
<td>1428</td>
<td>1481 (1174 (19.58%)</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>1h12min</td>
<td>3538</td>
<td>427</td>
<td>517 (651 (20.93%)</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>3h18min13s</td>
<td>2376</td>
<td>136</td>
<td>206 (594 (26.52%)</td>
</tr>
<tr>
<td>sum</td>
<td></td>
<td></td>
<td>16952</td>
<td>2343</td>
<td>2554 (2811 (19.24%)</td>
</tr>
</tbody>
</table>

A2—source data 1. Raw data of trial runs using the hungarian and tree-based matching algorithms, as well as baseline data from manually or automatically corrected trials used in this table is available for download from [221] (in A4T2_source_data.zip).

Table A2: A quality assessment of assignment decisions made by the general purpose tracking system without the aid of visual recognition – comparing results of two accurate tracking algorithms with the assignments made by an approximate method. Here, decisions are reassignments of an individual after it has been lost, or the tracker was too "unsure" about an assignment. Decisions can be either correct or wrong, which is determined by comparing to reference data generated using automatic visual recognition: Every segment of frames between decisions is associated with a corresponding "baseline-truth" identity from the reference data. If this association changes after a decision, then that decision is counted as wrong. Analysing a decision may fail if no good match can be found in the reference data (which is not interpolated). Failed decisions are ignored. Comparative values for the Hungarian algorithm ([122]) are always exactly the same as for our tree-based algorithm, and are therefore not listed separately. Left-aligned total, excluded and wrong counts in each column are results achieved by an accurate algorithm, numbers to their right are the corresponding results using an approximate method.
<table>
<thead>
<tr>
<th>video</th>
<th># ind.</th>
<th>fps (Hz)</th>
<th>size (px²)</th>
<th>tree</th>
<th>approximate</th>
<th>hungarian</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1024</td>
<td>40</td>
<td>3866 × 4048</td>
<td>35.49 ± 65.94</td>
<td>38.69 ± 65.39</td>
<td>12.05 ± 18.72</td>
</tr>
<tr>
<td>1</td>
<td>512</td>
<td>50</td>
<td>3866 × 4140</td>
<td>51.18 ± 180.08</td>
<td>75.02 ± 193.0</td>
<td>28.92 ± 29.12</td>
</tr>
<tr>
<td>2</td>
<td>512</td>
<td>60</td>
<td>3866 × 4048</td>
<td>59.66 ± 121.4</td>
<td>65.58 ± 175.51</td>
<td>23.18 ± 26.83</td>
</tr>
<tr>
<td>3</td>
<td>256</td>
<td>50</td>
<td>3866 × 4140</td>
<td>174.02 ± 793.12</td>
<td>190.62 ± 743.54</td>
<td>127.86 ± 9841.21</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>60</td>
<td>3866 × 4048</td>
<td>140.73 ± 988.15</td>
<td>155.9 ± 760.05</td>
<td>108.48 ± 2501.06</td>
</tr>
<tr>
<td>5</td>
<td>128</td>
<td>60</td>
<td>3866 × 4048</td>
<td>318.6 ± 347.8</td>
<td>353.58 ± 291.63</td>
<td>312.05 ± 337.71</td>
</tr>
<tr>
<td>6</td>
<td>128</td>
<td>60</td>
<td>3866 × 4048</td>
<td>286.13 ± 330.08</td>
<td>314.91 ± 303.53</td>
<td>232.33 ± 395.21</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>32</td>
<td>3584 × 3500</td>
<td>572.46 ± 98.21</td>
<td>611.5 ± 96.46</td>
<td>637.87 ± 97.03</td>
</tr>
<tr>
<td>8</td>
<td>59</td>
<td>51</td>
<td>2306 × 2306</td>
<td>744.98 ± 264.43</td>
<td>839.45 ± 257.56</td>
<td>864.01 ± 223.47</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>25</td>
<td>1880 × 1881</td>
<td>4626.84 ± 424.8</td>
<td>4585.08 ± 378.64</td>
<td>4508.08 ± 404.56</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>100</td>
<td>1920 × 1080</td>
<td>2370.35 ± 303.94</td>
<td>2408.27 ± 297.83</td>
<td>2362.42 ± 296.99</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>32</td>
<td>3712 × 3712</td>
<td>6489.12 ± 322.59</td>
<td>6571.28 ± 306.34</td>
<td>6472.0 ± 322.03</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>32</td>
<td>3712 × 3712</td>
<td>6011.59 ± 318.12</td>
<td>6106.12 ± 305.96</td>
<td>5549.25 ± 318.21</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>32</td>
<td>3712 × 3712</td>
<td>6717.12 ± 325.37</td>
<td>6980.12 ± 316.59</td>
<td>6726.46 ± 316.87</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>30</td>
<td>3008 × 3008</td>
<td>8752.2 ± 2141.03</td>
<td>8814.63 ± 2101.4</td>
<td>8630.73 ± 2177.16</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>25</td>
<td>3008 × 3008</td>
<td>9786.68 ± 1438.08</td>
<td>10118.04 ± 1380.2</td>
<td>9593.44 ± 1439.28</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>35</td>
<td>3008 × 3008</td>
<td>9861.42 ± 1424.91</td>
<td>10268.82 ± 1339.8</td>
<td>9680.68 ± 1387.14</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>140</td>
<td>1312 × 1312</td>
<td>15323.05 ± 637.17</td>
<td>15250.39 ± 639.2</td>
<td>15680.93 ± 640.99</td>
</tr>
</tbody>
</table>

**Table A3:** Comparing computation speeds of the tree-based tracking algorithm with the widely established Hungarian algorithm [122], as well as an approximate version optimized for large quantities of individuals. Posture estimation has been disabled, focusing purely on the assignment problem in our timing measurements. The tree-based algorithm is programmed to fall back on the Hungarian method whenever the current problem "explodes" computationally – these frames were excluded. Listed are relevant video metrics on the left and mean computation speeds on the right side for three different algorithms: (1) The tree-based and (2) the approximate algorithm presented in this paper, and (3) the Hungarian algorithm. Speeds listed here are percentages of real-time (the videos’ fps), demonstrating usability in closed-loop applications and overall performance. Results show that increasing the number of individuals both increases the time-cost, as well as producing much larger relative standard deviation values. (1) is almost always fast than (3), while becoming slower than (2) with increasing individual numbers. In our implementation, all algorithms produce faster than real-time speeds with 256 or fewer individuals (see also appendix Table A1), with (1) and (2) even getting close for 512 individuals.
<table>
<thead>
<tr>
<th>video</th>
<th># ind.</th>
<th>length (min)</th>
<th>fps (Hz)</th>
<th>prepare</th>
<th>tracking</th>
<th>live</th>
<th>win (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1024</td>
<td>8.33</td>
<td>40</td>
<td>10.96 ± 0.3</td>
<td>41.11 ± 0.34</td>
<td>65.72 ± 1.35</td>
<td>−26.23</td>
</tr>
<tr>
<td>1</td>
<td>512</td>
<td>6.67</td>
<td>50</td>
<td>11.09 ± 0.24</td>
<td>24.43 ± 0.2</td>
<td>33.67 ± 0.58</td>
<td>5.24</td>
</tr>
<tr>
<td>2</td>
<td>512</td>
<td>5.98</td>
<td>60</td>
<td>11.72 ± 0.2</td>
<td>20.86 ± 0.47</td>
<td>31.1 ± 0.62</td>
<td>4.55</td>
</tr>
<tr>
<td>3</td>
<td>256</td>
<td>6.67</td>
<td>50</td>
<td>11.09 ± 0.21</td>
<td>7.99 ± 0.17</td>
<td>12.35 ± 0.17</td>
<td>35.26</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>5.98</td>
<td>60</td>
<td>11.76 ± 0.26</td>
<td>9.04 ± 0.26</td>
<td>15.08 ± 0.13</td>
<td>27.46</td>
</tr>
<tr>
<td>6</td>
<td>128</td>
<td>5.98</td>
<td>60</td>
<td>11.77 ± 0.29</td>
<td>4.74 ± 0.13</td>
<td>12.13 ± 0.32</td>
<td>26.49</td>
</tr>
<tr>
<td>5</td>
<td>128</td>
<td>6.0</td>
<td>60</td>
<td>11.74 ± 0.26</td>
<td>4.54 ± 0.1</td>
<td>12.08 ± 0.25</td>
<td>25.79</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>1.0</td>
<td>32</td>
<td>1.92 ± 0.02</td>
<td>0.47 ± 0.01</td>
<td>2.03 ± 0.02</td>
<td>14.88</td>
</tr>
<tr>
<td>8</td>
<td>59</td>
<td>10</td>
<td>51</td>
<td>6.11 ± 0.07</td>
<td>7.68 ± 0.12</td>
<td>9.28 ± 0.08</td>
<td>32.7</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>60</td>
<td>25</td>
<td>12.59 ± 0.18</td>
<td>5.32 ± 0.07</td>
<td>13.17 ± 0.12</td>
<td>26.47</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>10.17</td>
<td>32</td>
<td>8.58 ± 0.04</td>
<td>0.74 ± 0.01</td>
<td>8.8 ± 0.12</td>
<td>5.66</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>10.05</td>
<td>32</td>
<td>8.68 ± 0.04</td>
<td>0.75 ± 0.01</td>
<td>8.65 ± 0.07</td>
<td>8.3</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>10.05</td>
<td>32</td>
<td>8.67 ± 0.03</td>
<td>0.71 ± 0.01</td>
<td>8.65 ± 0.07</td>
<td>7.76</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10.08</td>
<td>100</td>
<td>4.17 ± 0.06</td>
<td>2.02 ± 0.02</td>
<td>4.43 ± 0.05</td>
<td>28.3</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>195.37</td>
<td>30</td>
<td>110.51 ± 2.32</td>
<td>8.99 ± 0.22</td>
<td>109.97 ± 2.05</td>
<td>7.98</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>72.0</td>
<td>25</td>
<td>31.84 ± 0.53</td>
<td>3.26 ± 0.07</td>
<td>32.1 ± 0.42</td>
<td>8.55</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>198.22</td>
<td>35</td>
<td>133.45 ± 2.22</td>
<td>11.38 ± 0.28</td>
<td>133.1 ± 2.28</td>
<td>8.1</td>
</tr>
</tbody>
</table>

**Table A4:** Comparing the time-cost for tracking and converting videos in two steps with doing both of those tasks at the same time. The columns *prepare* and *tracking* show timings for the tasks when executed separately, while *live* shows the time when both of them are performed at the same time using the live-tracking feature of TGrabs. The column *win* shows the time "won" by combining tracking and preprocessing as the percentage \((\text{prepare + tracking - live}) / (\text{prepare + tracking})\). The process is more complicated than simply adding up timings of the tasks. Memory and the interplay of work-loads have a huge effect here. Posture is enabled in all variants.
<table>
<thead>
<tr>
<th>video characteristics</th>
<th>matching stats</th>
</tr>
</thead>
<tbody>
<tr>
<td># ind.</td>
<td># nodes visited (5,50,95,100%)</td>
</tr>
<tr>
<td>0 1024</td>
<td>[1535; 2858; 83243; 18576918]</td>
</tr>
<tr>
<td>1 512</td>
<td>[1060; 8156; 999137; 19811558]</td>
</tr>
<tr>
<td>2 512</td>
<td>[989; 2209; 56061; 8692547]</td>
</tr>
<tr>
<td>3 256</td>
<td>[452; 479; 969; 205761]</td>
</tr>
<tr>
<td>4 256</td>
<td>[475; 496; 584; 608994]</td>
</tr>
<tr>
<td>5 128</td>
<td>[233; 245; 258; 7149]</td>
</tr>
<tr>
<td>6 128</td>
<td>[237; 259; 510; 681702]</td>
</tr>
<tr>
<td>7 100</td>
<td>[195; 199; 199; 13585]</td>
</tr>
<tr>
<td>8 59</td>
<td>[117; 117; 117; 16430]</td>
</tr>
<tr>
<td>9 15</td>
<td>[24; 29; 29; 635]</td>
</tr>
<tr>
<td>10 10</td>
<td>[17; 19; 19; 56]</td>
</tr>
<tr>
<td>11 10</td>
<td>[19; 19; 19; 129]</td>
</tr>
<tr>
<td>12 10</td>
<td>[19; 19; 19; 1060]</td>
</tr>
<tr>
<td>13 10</td>
<td>[19; 19; 19; 106]</td>
</tr>
<tr>
<td>14 8</td>
<td>[11; 15; 15; 893]</td>
</tr>
<tr>
<td>15 8</td>
<td>[13; 15; 15; 597]</td>
</tr>
<tr>
<td>16 8</td>
<td>[15; 15; 15; 2151]</td>
</tr>
<tr>
<td>17 1</td>
<td>[1; 1; 1; 1]</td>
</tr>
</tbody>
</table>

Table A5: Statistics for running the tree-based matching algorithm with the videos of the speed dataset. We achieve low leaf and node visits across the board – this is especially interesting in videos with high numbers of individuals. High values for '# nodes visited' are only impactful if they make up a large portion of the assignments. These are the result of too many choices for assignments – the weak point of the tree-based algorithm – and lead to combinatorical "explosions" (the method will take a really long time to finish). If such an event is detected, TRex automatically switches to a more computationally bounded algorithm like the Hungarian method.
A.5 Posture

Estimating an animal’s orientation and body pose in space is a diverse topic, where angle and pose can mean many different things. We are not estimating the individual positions of many legs and antennae in TRex, we simply want to know where the front- and the back-end of the animal are. Ultimately, the goal here is to be able to align animals using an arbitrary axis with their head extending in one direction and their tail roughly in the opposite direction. In order to achieve this, we are required to follow a series of steps to acquire all the necessary information:

1. Locate objects in the image
2. Detect the edge of objects
3. Find an ordered set of points (the outline), which in sequence approximate the outer edge of an object in the scene. This is done for each object (as well as for holes).
4. Calculate a center-line based on local curvature of the outline.
5. Calculate head and tail positions.

The first point is a given at this point (see Connected components algorithm). We can utilize the format in which connected components are computed in TRex (an ordered array of horizontal line segments), which reduces redundancy by avoiding to look at every individual pixel. These line segments also contain information about edges since every start and end has to be an edge-pixel, too.

Even though we already have a list of edge-pixels, retrieving an ordered set of points is crucial and requires much more effort. Without information about a pixel’s connectivity, we can not differentiate between inner and outer shapes (holes vs. outlines) and we can not calculate local curvature.

A.5.1 Connecting pixels to form an outline

We implemented an algorithm based on horizontal line segments, which only ever retains three consecutive rows of pixels (p previous, c current and n next). These horizontal line segments always stem from a "blob" (or connected component). Rows contain (i) their y-value in pixels, (ii) x₀, x₁ values describing the first and last "on"-pixel that has been found in it, (iii) a set of detected border pixels (identified by their x-coordinate). A row is valid, whenever the y coordinate is not −1 – all three rows are initialized to an invalid y = −1. ′p′ is the previous row. Using p, corn as a function c(x) returns 1 for on-pixels at that x-coordinate, and 0 for off-pixels.
For each line \( l \) in the sorted list of horizontal line segments, we detect border pixels:

1. subtract the blobs position (minimum of all \( l_x \) and \( l_y \) separately) from \( l \)

2. if \( n_y \neq l_y \), a row has ended and a new one starts: call finalize

\[ \text{else if } l_x - l_x' \geq 1 \land l_x \geq c_x \text{, we either skipped a few pixels in } n \text{ or } l \text{ starts before } c \text{ even had valid pixels. This means that all pixels } x \text{ between } \max\{l_x' + 1; c_x\} \leq x < \min\{l_x; c_x + 1\} \text{ are border pixels in } c. \]

3. if \( l_x < c_x \), or \( c \) is invalid, then line \( l \) ends before the previous row \( (c) \) even has any "on"-pixels.

\[ \text{All pixels } x \text{ between } l_x \leq x \leq l_{x1} \text{ are border pixels in } n. \]

\[ \text{else} \]

(a) \( s := l_x \)

(b) if \( s < c_x \), then lines are overlapping in \( c \) and \( n \) (line \( l \)). We can fill \( n \) up with border while \( x < c_x \) and \( x \leq l_{x1} \). Set \( s := \min\{c_x - 1; l_{x1}\} \).

\[ \text{else if } s = 0 \text{ or } s > 0 \land n(s - 1) = 0, \text{ then } l \text{ starts at the image border (which is an automatic border pixel) or there is a gap before } l. \text{ Set } s := s + 1. \]

(c) All pixels at x-coordinates \( s \leq x \leq l_{x1} \) are border in \( n \), if they are either (i) beyond \( c \)'s bounds \( (x \geq c_{x1}) \), or (ii) \( c(x) = 0. \)

4. Set \( n_{x1} := l_{x1}. \)

After iterating through all lines, we need two additional calls to finalize to populate the lines currently in \( c \) and \( n \) through.

A graph is updated each time a row is finalized. This graph stores all border "nodes", as well as all a maximum of two edges per node (since this is the maximum number of neighbors for a line vertex). More on that below. The following procedure (finalize) prepares a row \( (c) \) to be integrated into the graph, using two parameters: A triplet of rows \( (p, c, n) \) and the first line \( l \), which started the new row to be added.

1. if \( n \) is invalid, continue to the next operation.

\[ \text{else if } l_x > n_y + 1, \text{ then we skipped at least one row between } n \text{ and the new row – making all on-pixels in } n \text{ border pixels.} \]
else we have consecutive rows where \( l_y = n_y + 1 \). All on-pixels \( x \) in \( n \) between \( n_{x0} \leq x \leq l_{x0} - 1 \) are border pixels.

2. Now the current row (\( c \)) is certainly finished, as it will in the following become the previous row (\( p \)), which is read-only at that point. We can add every border-pixel of \( c \) to our graph (see below).

3. It then discards \( p \) and moves \( c \rightarrow p \) and \( n \rightarrow c \), as well as reading a new row to assign to \( n \), setting \( n_{x0} = l_{x0}, n_{x1} = l_{x1}, n_y = l_y \).

The graph consists of nodes (border pixels), indexed by their \( x \) and \( y \) coordinates (integers) and containing a list of all on-pixels around them (8-neighbourhood with top-left, top, left, bottom-left, etc.). This information is available when \( \text{finalize} \) is called, since the middle row (\( c \)) is fully defined at that point (its entire neighbourhood has been cached).

After all rows have been processed, an additional step is needed to connect all nodes and produce a connected, clockwise ordered outline. We already marked all pixels that have at least one border.

We can also already mark TOP, RIGHT, BOTTOM and LEFT borders per node if no neighbouring pixel is present in that direction, since these major directions will definitely get a "line" in the end. So all we have left to do now, is check the diagonals. The points that will be returned, are located halfway along the outer edges of pixels. In the end, each pixel can potentially have four border lines (if it is a singular pixel without connections to other pixels, see yellow "hole" in \( A8b \)). The half-edge-points for each node are generated as follows:

1. A nodes list of border pixels is a sparse, ordered list of directions (top, top-right, ..., top-left). Each major direction of these (TOP, RIGHT, BOTTOM, LEFT), if present, check the face of their square to the left of them (own direction - 1, or -45°). For example, TOP would check top-left.

2. if the checked neighbour is on, we add an edge between our face (e.g. TOP) and its 90° rotated face (e.g. own direction + 2 = RIGHT).

else check the face an additional 45° to the left (e.g. LEFT).

(a) if it there is an on-pixel attached to this face, add an edge between the two faces (of the focal and its left pixel) in the same direction (e.g. TOP→TOP).
else we do not seem to have a neighbour to either side, so this must be a corner pixel. Add an edge from the focal face (e.g. TOP) to the side 90° to the left of itself (e.g. LEFT).

Each time an edge is added, more and more of the half-edges are becoming fully-connected (meaning they have two of the allowed two edges). To generate the final result, all we have to do is to start somewhere in the graph and walk strictly in clockwise direction. "Walking" is done using a queue and edges are followed using depth-first search (see Matching an object to an object in the next frame): Each time a node is visited, all its yet unexplored edges are added to the front of the queue (in clockwise order). Already visited edges are marked (or pruned) and will not be traversed again – their floating-point positions (somewhere on an edge of its parent pixel) are added to an array.

After a path ended, meaning that no more edges can be reached from our current node, the collected floating-point positions are pushed to another array and a different, yet unvisited, starting node is sought. This way, we can accumulate all available outlines in a given image one-by-one – including holes.

These outlines will usually be further processed using an Elliptical Fourier Transform (or EFT, [121]), as mentioned in the main-text. Outlines can also be smoothed using a weighted average of the $N$ points around a given point, or resampled to either reduce or (virtually) increase resolution.

A.5.2 Finding the tail

Given an ordered outline, curvature can be calculated locally (per index $i$):

$$C(i) = 4 \cdot \text{triangle}_\text{area}(p_{i-r}, p_i, p_{i+r}) / (\|p_i - p_{i-r}\| \ast \|p_i - p_{i+r}\| \ast \|p_{i-r} - p_{i+r}\|)$$

where $1 \leq r \in \mathbb{N}$ is a parameter, which effectively leads to more smoothing when increased. Triangle area can be calculated as follows:

$$\text{triangle}_\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = (\mathbf{b}_x - \mathbf{a}_x)(\mathbf{c}_y - \mathbf{a}_y) - (\mathbf{b}_y - \mathbf{a}_y)(\mathbf{c}_x - \mathbf{a}_x).$$

To find the "tail", or the pointy end of the shape, we employ a method closely related to scipy’s find_peaks function: We find local maxima using discrete curve differentiation and then generate a hierarchy of these extrema. The only major difference to normal differentiation is that we assume periodicity to achieve our results – values wrap around in both directions, since we are dealing with
an outline here. We then find the peak with the largest integral, meaning we detect both very wide
and very high peaks (just not very slim ones). The center of this peak is the "tail".

To find the head as well, we now have to search for the peak that has the largest (index-) distance to
the tail-peak. This is a periodic distance, too, meaning that \( N \) is one of the closest neighbours of 0.

The entire outline array is then rotated, so that the head is always the first point in it. Both indexes
are saved.

**A.5.3 Calculating the center-line**

A center-line, for a given outline, can be calculated by starting out at the head and walking in both
directions from there – always trying to find a pair of points with minimal distance to each other on
both sides. Two indices are used: \( l, r \) for left and right. We also allow some "wiggle-room" for the
algorithm to find the best-matching points on each side. This is limited by a maximum offset of \( \omega \)
points which is set to 0.025 \( \times N \) by default, where \( N \) is the number of points in the outline. \( f(i) \) gives
the point on in outline at position \( i \).

Starting from \( l := -1, r := 1 \) we continue while \( r < l + N \):

1. Find \( m := \arg\min_i \{ ||f(r + i) - f(l)|| ; \forall i \leq \omega \land r + i < N \} \). If no valid \( m \) can be
found, abort. Otherwise set \( r := m \).

2. Find \( k := \arg\min_i \{ ||f(l - i + N) - f(r)|| ; \forall i \leq \omega \land l - i \leq -N \} \). If no valid \( k \) can be
found, abort. Otherwise set \( l := k \).

3. Our segment now consists of points \( f(m) \) and \( f(k) \), with a center vector of \( (f(k) - f(m)) \times 0.5 + f(m) \). Push it to the center-line array. We can also calculate the width of the body at that
point using \( ||f(k) - f(m)|| \).

4. Set \( l := l - 1 \).

5. Set \( r := r + 1 \).

Head and tail positions can be switched now, e.g. for animals where the wider part is the head. We
may also want to start at the slimmeast peak first, which ever that is, since there we have not as much
space for floating-point errors regarding where exactly the peak was. These options depend on the
specific settings used in each video.
**Figure A8:** The original image is displayed on the left. Each square represents one pixel. The processed image on the right is overlaid with lines of different colors, each representing one connected component detected by our outline estimation algorithm. Dots in the centers of pixels are per-pixel-identities returned by OpenCV’s `findContours` function (for reference) coded in the same colors as ours. Contours calculated by OpenCV’s algorithm cannot be used to estimate the one-pixel-wide "tail" of the 9-like shape seen here, since it becomes a 1D line without sub-pixel accuracy. Our algorithm also detects diagonal lines of pixels, which would otherwise be an aliased line when scaled up.

The angle of the center-line is calculated using `atan2` for a vector between the first point and one point at an offset from it. The specific offset is determined by a midline stiffness parameter, which offers some additional stability – despite e.g. potentially noisy peak detection.

### A.6 Visual field estimation

Visual fields are calculated by casting rays and intersecting them with other individuals and the focal individual (for self-occlusion). An example of this can be seen in Figure 7. The following procedure requires posture for all individuals in a frame. In case an individual does not have a valid posture in the given frame, its most recent posture and position are used as an approximation. The field is internally represented as a discretized vector of multi-dimensional pixel values. Depending on the resolution parameter ($F_{res}$), which sets the number of pixels, each index in the array represents step-sizes of $(F_{max} - F_{min}) / F_{res}$ radians. The $F$ values are constants setting the minimum and maximum field of view ($-130^\circ$ to $130^\circ$ by default, which gives a range of $260^\circ$). Each pixel consists of multiple data-streams: The distance to the other individual, the identity of the other individual and the body-part that the ray intersected with.

Eyes are simulated to be located on the outline of the focal individual, near the head. The distance to the head can be set by the user as a percentage of midline-length. To find the exact eye position, the program calculates intersections between vectors going left/right from that midline point, perpen-
icular to the midline, and the individual’s outline. In order to be able to simulate different types of binocular and monocular sight, a parameter for eye separation $E_{sep}$ (radians) controls the offset from the head angle $H_a$ per eye. Left and right eye are looking in directions $H_a - E_{sep}$ and $H_a + E_{sep}$, respectively.

We iterate through all available postures in a given frame and use a procedure which is very similar to depth-maps ([228]) in e.g. OpenGL. In the case of 2D visual fields, this depth-map is 1D. Each pixel holds a floating-point value (initialized to $\infty$) which is continuously compared to new samples for the same position – if the new sample is closer to the "camera" than the reference value, the reference value is replaced. This way, after all samples have been evaluated, we generate a map of the objects closest to the "camera" (in this case the eye of the focal individual). For that to work we also have to keep the identity in each of these discrete slots maintained. So each time a depth value is replaced, the same goes for all the other data-streams (such as identity and head-position). When an existing value is replaced, values in deeper layers of occlusion are pushed downwards alongside the old value for the first layer.

Position of the intersecting object’s top-left corner is located at $\hat{P}$. Let $E_e$ be the position of each eye, relative to $\hat{P}$. For each point $P_j$ (coordinates relative to $\hat{P}$) of the outline, check the distance between $E_e$ and the outline segments $(P_j - P_{j-1})$. For each eye $E_e$:

1. Project angles ranging from $[\text{atan2}(P_{j-1} + E_e), \text{atan2}(P_j + E_e)]$, where $\alpha_e$ is the eye orientation, using:

$$\Gamma_e(\beta) = \text{angle\_normalize}\left(\beta - \alpha_e - F_{\min}\right) / (F_{\max} - F_{\min}) \times F_{\text{res}}$$

angle\_normalize$(\beta)$ normalizes beta to be between $[-\pi, \pi]$.

2. If either $\max(R)$ or $\min(R)$ is inside the visual field ($0 \leq \Gamma_e(\beta) \leq 1$):
   
   (a) We call the first angle satisfying the condition $\beta$.

   (b) Then the search range becomes $R := [\max\{\beta - 0.5; 0\}, \beta + 0.5]$, where the elements in $R$ are integers.

   (c) Let $\delta_{j,e} = \|P_{j-1} - E_e\|$, the distance between outline point at $j - 1$ and the eye (interpolation could be done here).

   (d) Let index $k \in \mathbb{N}, k \in R$ be our index into the first layer of the depth-map $\text{depth}_0$:
(e) if $\text{depth}_0(k) > \delta_{je}$: Calculate all properties $D_0(k) := \{\text{head\_distance}, \cdots \in \text{data\_streams}\}^T$, and push values at $k$ in layer 0 to layer 1.

(f) otherwise, if $\text{depth}_1(k) > \delta_{je}$, calculate properties for layer 1 instead and move data from layer 1 further down, etc.

The data-streams are calculated individually with the following equations:

- **Distance**: Given already in $\text{depth}_i(k)$. In practice, values are cut off at the maximum distance (size of the video squared) and normalized to $[0, 255]$.
- **Identity**: Is assigned alongside $\text{depth}_i(k)$ for each element that successfully replacing another in the map.
- **Body-part**: Let $T_i = \text{tail index}, L_{l/r} = \text{number of points in left/right side of the outline (given by tail- and head-indexes)}$:
  1. if $i > T_i$: $\text{head\_distance} = 1 - |i - T_i|/L_i$
  2. else: $\text{head\_distance} = 1 - |i - T_i|/L_r$

A.7 The PV file format

Since we are using a custom file format to save videos recorded using TGrabs (MP4 video can be saved alongside PV for a limited time or frame-rate), the following is a short overview of PV6 contents and structure. This description is purely technical and concise. It is mainly intended for users who wish to implement a loader for the file format (e.g. in Python) or are curious.

A.7.1 Structure

Generally, the file is built as a header (containing meta information on the video) followed by a long data section and an index table plus a settings string at the end. The header at the start of the file can be read as follows:

1. version (string): "PV6"
2. channels (uint8): Hard-coded to 1
3. width & height (uint16): Video size
4. crop offsets (4x uint16): Offsets from original image

5. size of HorizontalLine struct (uchar)

6. # frames (uint32)

7. index offset (uint64): Byte offset pointing to the index table for

8. timestamp (uint64): time since 1970 in microseconds of recording (or conversion time if unavailable)

9. empty string

10. background image (byte*): An array of uint8 values of size width * height * channels.

11. mask image size (uint64): 0 if no mask image was used, otherwise size in bytes followed by a byte* array of that size

Followed by the data section, where information is saved per frame. This information can either be in a zip-compressed format, or raw (determined by size), see below:

1. compression flag (uint8): 1 if compression was used, 0 otherwise

2. if compressed:
   
   (a) original size (uint32)

   (b) compressed size (uint32)

   (c) lzo1x compressed data (byte*) in the format of the uncompressed variant (below)

3. if uncompressed:

   (a) timestamp since start time in header (uint32)

   (b) number of images in frame (uint16)

   (c) for each image in frame:

   i. number of HorizontalLines (uint16)

   ii. data of HorizontalLine (byte*)

   iii. pixel data for each pixel in the previous array (byte*)
Files are concluded by the index table, which gives a byte offset for each video frame in the file, and a settings string. This index is used for quick frame skipping in TRex as well as random access. It consists of exactly one uint64 index per video frame (as determined by the number of video frames read earlier). After that map ends, a string follows, which contains a JSON style string of all metadata associated by the user (or program) with the video (such as species or size of the tank).

### A.8 Automatic visual identification

#### A.8.1 Network layout and training procedure

Network layout is sketched in Figure 1c. Using version 2.2.4 of Keras\(^5\), weights of densely connected layers as well as convolutional layers are initialized using Xavier-initialization ([83]). Biases are used and initialized to 0. The default image size in TRex is 80 × 80, but can be changed to any size in order to retain more detail or improve computation speed.

During training, we use the Adam optimizer ([117]) to traverse the loss landscape, which is generated by categorical focal loss. *Categorical focal loss* is an adaptation of the original *binary focal loss* ([130]) for multiple classes:

\[
cFL(j) = \sum_{c=1}^{N} -\alpha (1 - P_{c,j})^\gamma V_{c,j} \log P_{c,j},
\]

where \( P_{c,j} \) is the prediction vector component returned by the network for class \( c \) in image \( j \). \( V \) is a set of validation images, which remains the same throughout the training process. It comprises 25% of the images available per individual. Images are marked *globally* when becoming part of the validation dataset and are not used for training in the current or any of the following steps.

After each epoch, predictions are generated by performing a forward-pass through the network layers. Returned are the softmax-activations \( P_{c,j} \) of the last layer for each image \( j \) in the validation dataset. Simply calculating the mean of

\[
\mathcal{A} = \frac{1}{M} \sum_{j \in [0:M]} \begin{cases} 1 & \text{if } P_j = V_j \\ 0 & \text{otherwise} \end{cases},
\]

gives the mean accuracy of the network. \( M \) is the number of images in the validation dataset, where \( V_j \) are the expected probability vectors per image \( j \). However, much more informative is the

\(^5\)See keras.io documentation for default arguments
per-class (per-identity) accuracy of the network among the set of images $i$ belonging to class $c$, which is

$$I_c = \{ j; \text{ where } V_{jc} = 1, j \in [0, M] \},$$

given that all vectors in $V$ are one-hot vectors – meaning the vector has length $N$ with $V_{j\phi} = 0 \forall \phi \neq c$ and $V_{jc} = 1$.

$$A_c = \frac{1}{|I_c|} \sum_{j \in I_c} \begin{cases} 1 & \text{ if } P_j = V_j \\ 0 & \text{ otherwise} \end{cases}$$

Another constant, across training units – not just across epochs, is the set of images used to calculate mean uniqueness $\bar{U}$ (see Box 1.1, as well as Guiding the Training Process). Values generated in each epoch $t$ of every training unit are kept in memory and used to calculate their derivative $\bar{U}'(t)$.

A.8.2 Stopping-criteria

A training unit can be interrupted if one of the following conditions becomes true:

1. Training commenced for at least $t = 5$ epochs, but uniqueness value $\bar{U}$ was never above

$$\bar{U}^t_{\text{best}} > \bar{U}(t) \forall t$$

where $\bar{U}_{\text{best}}$ is the best mean uniqueness currently achieved by any training unit (initialized with zero). This prevents to train on faulty segments after a first successful epoch.

2. The worst accuracy value per class has been "good enough" in the last three epochs:

$$\min_{c \in [0, N]} \{ A_c \} \geq 0.97$$

3. The global uniqueness value has been plateauing for more than 10 epochs.

$$\sum_{k \in [t-10, t]} \bar{U}'(k) \leq 0.01$$

4. Overfitting: Change in loss is very low on average after more than 5 epochs. Mean loss is calculated as follows:
\[ c_{FL}(t) = \frac{1}{5} \sum_{k \in [t-5, t-1]} c_{FL}(k) \]

Now if the difference between the current loss and the previous loss is below a threshold:

\[ \lambda(t) = \left\lfloor \ln \left(c_{FL}(t)\right) \right\rfloor - 1 \]

\[ \frac{1}{5} \sum_{k \in [t-5, t]} \max \{ \epsilon; |c_{FL}(k) - c_{FL}(k)| \} < 0.05 \times 10^{\lambda(t)} \]

5. Maximum number of epochs has been reached. User-defined option limiting the amount of time that training can take per unit. By default this limit is set to 150 epochs.

6. Loss is zero. No further improvements are possible within the current training unit, so we terminate and continue with the next.

A high per-class accuracy over multiple consecutive epochs is usually an indication that everything that can be learned from the given data has already been learned. No further improvements should be expected from this point, unless the training data is extended by adding samples from a different part of the video. The same applies to scenarios with consistently zero or very low change in loss. Even if improvements are still possible, they are more likely to happen during the final (overfitting) step where all of the data is combined.

A.9 Data used in this paper and reproducibility

All of the data, as well as the figures and tables showing the data, have been generated automatically. We provide the scripts that have been used, as well as the videos if requested. "Data" refers to converted video-files, as well as log- and NPZ-files. Analysis has been done in Python notebooks, using mostly matplotlib and pandas, as well as numpy to load the data. Since TRex and TGrabs, as well as idtracker.ai have been run on a Linux system, we were able to run everything from two separate bash files:

1. run.bash
2. run_idtracker.bash
Where (1) encompasses all trials run using TRex and TGrabs, both for the speed- and recognition-datasets. (2) runs idtracker.ai in its own dedicated Python environment, using only the recognition-dataset. The parameters we picked for idtracker.ai vary between videos and are hand-crafted, saved in individual .json files (see Table A6 for a list of settings used). We ran multiple trials for each combination of tools and data with $N = 5$ where necessary:

- 3x TGrabs [speed-dataset]
- 5x TRex + recognition [recognition-dataset]
- 3x idtracker.ai [recognition-dataset]
- TRex without recognition enabled [speed-dataset]:
  - 3x for testing the tree-based, approximate and Hungarian methods (Tracking), without posture enabled – testing raw speeds (see Table A3)
  - 3x testing accuracy of basic tracking (see Table A2), with posture enabled

A Python script used for Figure 5, which is run only once. It generates a series of results for the same video (video 7 with 100 individuals) with different sample-sizes. It uses a single set of training samples and then – after equalizing the numbers of images per individual – generates multiple virtual subsets with fewer images. They span 15 different sample-sizes per individual, saving a history of accuracies for each run. We repeated the same procedure with for the different normalization methods (no normalization, moments and posture), each repeated five times.

As described in the main text, we recorded memory usage with an external tool (syrupy) and used it to measure both software solutions. This tool saves a log-file for each run, which is appropriately renamed and stored alongside the other files of that trial.

All runs of TRex are preceded by running a series of TGrabs commands first, in order to convert the videos in the datasets. We chose to keep these trials separately and load whenever possible, to avoid data-duplication. Since subsequent results of TGrabs are always identical (with the exception of timings), we only keep one version of the PV files (The PV file format) as well as only one version of the results files generated using live-tracking. However, multiple runs of TGrabs were recorded in the form of log-files to get a measure of variance between runs in terms of speed and memory.

A.9.1 Human validation

To ensure that results from the automatic evaluation (in Visual Identification: Accuracy) are plausible, we manually reviewed part of the data. Specifically, the table in Table 3 shows an overview
of the individual events reviewed and percentages of wrongly assigned frames. Due to the length of videos and the numbers of individuals inside the videos we did not review all videos in their entirety, as shown in the table. Using the reviewing tools integrated in TRex, we focused on crossings that were automatically detected. These tools allow the user to jump directly to points in the video that it deems problematic. Detecting problematic situations is equivalent to detecting the end of individual segments (see Automatic Visual Identification Based on Machine Learning). While iterating through these situations, we corrected individuals that have been assigned to the wrong object, generating a clean and corrected baseline dataset. We assumed that an assignment is correct, as long as the individual is at least part of the object that the identity has been assigned to. Misassignments were typically fixed after a few frames. Identities always returned to the correct individuals afterward (thus not causing a chain of follow-up errors).

A.9.2 Comparison between trajectories from different softwares, or multiple runs of the same software

In our tests, the same individuals may have been given different IDs (or "names") by each software (and in each run of each software for the same video), so, as a first step in every test where this was relevant, we had to determine the optimal pairing between identities of two datasets we wished to compare. This was done using a square distance matrix containing overall euclidean distances between identities is calculated by summing their per-frame distances. Optimally this number would be zero for one and greater than zero for every other pairing, but temporary tracking mistakes and differences in the calculation of centroids may introduce noise. Thus, we solved the matching problem (see Matching an object to an object in the next frame) for identities between each two datasets and paired individuals with the smallest accumulative distance between them. This was done for all results presented, where a direct comparison between two datasets was required.
Table A6: Settings used for idtracker.ai trials, as saved inside the .json files used for tracking. The minimum intensity was always set to 0 and background subtraction was always enabled. An ROI is an area of interest in the form of an array of 2D vectors, typically a convex polygon containing the area of the tank (e.g. for fish or locusts). Since this format is quite lengthy, we only indicate here whether we limited the area of interest or not.

<table>
<thead>
<tr>
<th>video</th>
<th>length (# frames)</th>
<th>nblobs</th>
<th>area</th>
<th>max. intensity</th>
<th>roi</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1921</td>
<td>100</td>
<td>[165, 1500]</td>
<td>170</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>30626</td>
<td>59</td>
<td>[100, 2500]</td>
<td>160</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>19539</td>
<td>10</td>
<td>[200, 1500]</td>
<td>10</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>19317</td>
<td>10</td>
<td>[200, 1500]</td>
<td>10</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>19309</td>
<td>10</td>
<td>[200, 1500]</td>
<td>10</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>90001</td>
<td>8</td>
<td>[190, 4000]</td>
<td>147</td>
<td>Yes</td>
</tr>
<tr>
<td>16</td>
<td>416259</td>
<td>8</td>
<td>[200, 2500]</td>
<td>50</td>
<td>No</td>
</tr>
<tr>
<td>14</td>
<td>351677</td>
<td>8</td>
<td>[200, 2500]</td>
<td>50</td>
<td>No</td>
</tr>
<tr>
<td>15</td>
<td>108000</td>
<td>8</td>
<td>[250, 2500]</td>
<td>10</td>
<td>No</td>
</tr>
</tbody>
</table>

A.10 Matching probabilities

One of the most important steps, when matching objects in one frame with objects in the next frame, is to calculate a numerical landscape that can then be traversed by a maximization algorithm to find the optimal combination. This landscape, which can be expressed as an $m \times n$ matrix $P(t)$, contains the probability values between $[0, 1]$ for each assignment between individuals $i$ and objects $B_j$.

Below are definitions used in the following text:

- $T_\Delta$ is the typical time between frames (s), which depends on the video
- $\tau_i < t$ is most recent frame assigned to individual $i$ previous to the current frame $t$
- $P_{\text{min}}$ is the minimally allowed probability for the matching algorithm, underneath which the probabilities are assumed to be zero (and respective combination of object and individual is ignored). This value is set to 0.1 by default.
- $F(t \in \mathbb{R}) \rightarrow \mathbb{N}$ is the frame number associated with the time $t$ (s)
- $T(f \in \mathbb{N}) \rightarrow \mathbb{R}$ is the time in seconds of frame $f$, with $F(T(f)) = f$
- $x$ indicates that $x$ is a vector
- $U(x) = x/\|x\|$
Some values necessary for the following calculations are independent of the objects in the current frame and merely depend on data from previous frames. They can be re-used per frame and individual in the spirit of dynamic programming, reducing computational complexity in later steps:

\[
\dot{v}_i(t) = \frac{\delta}{\delta t} \ddot{p}_i(t) = \delta p'_i(t) = \delta \frac{\delta}{\delta t} p_i(t)
\]

\[
\tilde{v}_i(t) = v_i(t) \times \begin{cases} 
1 & \text{if } \|v_i(t)\| \leq D_{\text{max}} \\
\frac{D_{\text{max}}}{\|v_i(t)\|} & \text{otherwise}
\end{cases}
\]

\[
a_i(t) = \frac{\delta}{\delta t} \tilde{v}_i(t)
\]

Velocity \(v_i(t)\) and acceleration \(a_i(t)\) are simply the first and second derivatives of the individual's position at time \(t\). \(\tilde{v}_i(t)\) is almost the same as the raw velocity, but its length is limited to the maximally allowed travel distance per second (\(D_{\text{max}}\), parameter \(\text{track\_max\_speed}\)).

These are then further processed, combining and smoothing across values of multiple previous frames (the last 5 valid ones). Here, \(\bar{f}(x)\) indicates that the resulting value uses data from multiple frames.

\[
\bar{\dot{v}}_i(t) = \text{median}_{k \in [F(\tau) - S, F(\tau)]} \|\tilde{v}_i(T(k))\|
\]

is the speed at which the individual has travelled at recently. The mean direction of movement is expressed as

\[
\bar{d}_i(t) = \frac{1}{F(t) - F(\tau) + 5} \sum_{k \in [F(\tau) - S, F(\tau)]} \tilde{v}_i(T(k))
\]

with the corresponding direction of acceleration

\[
\bar{a}_i(t) = U \left( \frac{1}{F(t) - F(\tau) + 5} \sum_{k \in [F(\tau) - S, F(\tau)]} a_i(T(k)) \right)
\]

The predicted position for individual \(i\) at time \(t\) is calculated as follows:

\[
\dot{p}_i(t) = s_i(t) \sum_{k \in [F(\tau), F(t) - 1]} w(k) \left( \bar{d}_i(t) + T'(k) \ast \bar{a}_i(t) \right)
\]

with weights for each considered time-step of
\[ w(f) = \frac{1 + \lambda^4}{1 + \lambda^4 \max \{1, f - F(\tau_i) + 1\} }, \]

where \( \lambda \in [0, 1] \) is a decay rate (parameter track\_speed\_decay) at which the impact of previous positions on the predicted position decreases with distance in time. With its value approaching 1, the resulting curve becomes steeper - giving less weight previous positions the farther away they are from the focal frame.

In order to locate an individual \( i \) in the current frame \( F(t) \), a probability is calculated for each object \( B_j \) found in the current frame resulting in the matrix:

\[
P(t) = \begin{bmatrix}
P_0(t \mid B_0) & \ldots & P_n(t \mid B_0) \\
\vdots & \ddots & \vdots \\
P_0(t \mid B_m) & \ldots & P_n(t \mid B_m)
\end{bmatrix}.
\] (A.1)

Probabilities \( P_i(t \mid B_j) \) for all potential connections between blobs \( B_j \) and identities \( i \) at time \( t \) are calculated by first predicting the expected position \( \hat{p}_i(t) \) for each individual in the current frame \( F(t) \). This allows the program to focus on a small region of where the individual is expected to be located, instead of having to search the whole arena each time.

Based on the individual’s recent speed \( \bar{\dot{v}}_i(t) \), direction \( \bar{\delta}_i(t) \), acceleration \( \bar{\alpha}_i(t) \) and angular momentum \( \bar{\alpha}'_i(t) \), the individual’s projected position \( \hat{p}_i(t) \) is usually not far away from its last seen location for small time-steps. Only when \( \Delta t \) increases, if the individual has been lost for more than one frame or frame-rates are low, does it really play a role.

The actual probability values in \( P(t) \) are then calculated by combining three metrics - each describing different aspects of potential concatenation of object \( b \) at time \( t \) to the already existing track for individual \( i \):

The time metric \( T_i(t) \), which does not depend on the blob the individual is trying to be matched to. It merely reflects the recency of the individuals last occurrence in a way that recently seen individual will always be preferred over individuals that have been lost for longer.

\[ F_{\min} = \min \left\{ \frac{1}{T^{\Delta}}, 5 \right\} \]

\[ R_i(t) = \left\| \left\{ T(k) \mid F(t) - T^{\Delta - 1} \leq k \leq t \land T(k) - T(k - 1) \leq T^{\max} \right\} \right\| \]
\( T_i(t) = \left( 1 - \min \left\{ 1, \frac{\max \{0, \tau_i - t - T_\Delta\}}{T_{\max}} \right\} \right) \cdot \left\{ \begin{array}{ll} 1, & F_i(\tau_i) - 1 + P_{\text{min}} \\ \frac{1}{1}, & F_i(\tau_i) \geq F(t_0) + P_{\text{min}} \end{array} \right. \)  

\( S_i(t \mid B_j) \) is the speed that it would take to travel from the individuals position to the blobs position in the given time (which might be longer than one frame), inverted and normalized to a value between 0 and 1.

\[
S_i(t \mid B_j) = \left( 1 + \frac{\| (p_i(t) - \dot{p}_i(t)) / (\tau_i - t) \|}{D_{\text{max}}} \right)^{-2}
\]

And the angular difference metric \( A_i(t, \tau_i \mid B_j) \), describing how close in angle the resulting vector of connecting blob and individual to a track would be to the previous direction vector:

\[
a = \dot{p}_i(t) - p_i(\tau_i)
\]

\[
b = p_{B_j}(t) - p_i(\tau_i)
\]

\[
A_i(t, \tau_i \mid B_j) = \left\{ \begin{array}{ll} 1 - \frac{1}{\pi} \left| \text{atan2} \left\{ \| a \times b \|, a \cdot b \right\} \right| & \text{if } \| a \| > 1 \land \| b \| > 1 \\ 1 & \text{otherwise} \end{array} \right.
\]

The conditional ensures that the individual travelled a long enough distance, as the atan2 function used to determine angular difference here lacks numerical precision for very small magnitudes. This is, however, an unproblematic case in this situation as the positions are in pixel-coordinates and anything below a movement of one pixel is likely to be due to noise anyway.

Combining (A.2), (A.3) and (A.4) into a weighted probability product yields:

\[
P_i(t, \tau_i \mid B_j) = S_i(t \mid B_j) \cdot \left( 1 - \omega_1 \left( 1 + A_i(t, \tau_i \mid B_j)\right) \right) \cdot \left( 1 - \omega_2 \left( 1 + T_i(t, \tau_i)\right) \right)
\]

Results from equation (A.5) can now easily be used in a matching algorithm, in order to determine the best combination of objects and individuals as in Matching an object to an object in the next frame. \( \omega_1 \) is usually set to 0.1, \( \omega_2 \) is set to 0.25 by default.
### Algorithm for splitting touching individuals

**Data:** image of a blob, \( N_e \) number of expected blobs

**Result:** \( N \geq N_e \) smaller image-segments, or *error*

threshold = \( T_i \);

while threshold < 255 do

blobs = apply_threshold(image, threshold);

if \( \|\text{blobs}\| = 0 \) then

break;

end

if \( \|\text{blobs}\| \geq N_e \) then

sort blobs by size in decreasing fashion;

loop through all blobs \( i \) up to \( i \leq N_e \) and detect whether the size-ratio between them is roughly even. until then, we keep iterating.;

if \( \min \{\text{ratio}, \forall i \in [0, N_e]\} < 0.3 \) then

threshold = threshold + 1;

continue;

else

return blobs;

end

else

threshold = threshold + 1;

end

end

return fail;

**Algorithm 2:** The algorithm used whenever two individuals touch, which is detected by a history-based method. This history-based method also provides \( N_e \), the number of expected objects within the current (big) object. \( T_i \) is the starting threshold constant parameter, as set by the user.

### Posture and Visual Identification of Highly-Deformable Bodies

To evaluate further whether TRex’s posture and visual identification algorithms are broadly applicable, such as to mammals (e.g. rodents) – which have highly-deformable bodies and thus increased
variance per individual, we conducted additional analyses on videos of groups of four freely behaving mice (four C57BL/6 mice provided by D. Mink and M. Groettrup, and four "black mice" from [184] provided to us by G.G. de Polavieja, and now linked under idtrackerai.readthedocs.io).

Both videos, listed in A7 and previewed in A9, were analyzed using the same scripts used to generate Table 3, although each video has only been automatically tracked once (since accuracy of tracking is very high, as detailed below). We manually generated verified trajectories for both videos in full, following the same procedure described in Human validation, and compared them to the automatically generated trajectories. As can be seen in A7, TRex provides highly accurate results for both videos (≥ 99.6%).

Tracking, in theory and in practice as per our results here, is not generally impacted by the shape of individuals. However, individuals of some species tend to stay close/on top of con-species, which may render them impossible to track during periods where traditional image processing methods are unable to separate them. This explains the ∼ 6% interpolated frames in V1 (see A7), and also gives a reason why there is similarity between video 9 and V1 in that respect – the locusts in video 9 also spend much time either on top of others, or in places where they are harder to see.

Very short segments of mistaken identities (with a maximum length of less than 200ms) occurred whenever individuals “appear” only for a short moment and the segment does not contain enough data to be properly matched with a learned identity. Correct identities were reassigned in all cases after the individuals could be visually separated from each other again, and such events only make up < 1% of the tracked data.

Furthermore, we found that our method for posture estimation works well despite the more deformable bodies and complex 3D-postures of mice. Head and tail may switch occasionally, especially when animals shrink to “a circle” from the viewpoint of the camera. Overall, however, by far most samples are normalised correctly – as can be seen in A10 and A11.

<table>
<thead>
<tr>
<th>video</th>
<th># ind.</th>
<th>reviewed (%)</th>
<th>interpolated (%)</th>
<th>TRex</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V1) [184]</td>
<td>4</td>
<td>100.0</td>
<td>6.41</td>
<td>99.6 ± 0.0</td>
</tr>
<tr>
<td>(V2) D. Mink, M. Groettrup</td>
<td>4</td>
<td>100.0</td>
<td>1.74</td>
<td>99.82 ± 0.0</td>
</tr>
</tbody>
</table>

Table A7: Analogous to our analysis in Table 3, we compared automatically generated trajectories for two videos with manually verified ones. Unlike the table in the main text, the sample size per video is only one here, which is why the standard deviation is zero in both cases. Results show very high accuracy for both videos, but relatively high numbers of interpolated frames compared to Table 3, where only the results for video 9 showed more than 8% interpolation and all others remained below 1%.
Figure A9: Screenshots from videos V1 and V2 listed in A7. Left (V1), video of four "black mice" (17min, 1272x909px resolution) from [184]. Right (V2), four C57BL/6 mice (1:08min, 1280x960px resolution) by M. Groettrup, D. Mink.

**A9–video 1.** A clip of the tracking results from V1, played back at normal speed. Although it succumbs to noise in some frames (e.g. around 13 seconds), posture estimation remains remarkably robust to it throughout the video – sometimes even through periods where individuals overlap (e.g. at 27 seconds). Identity assignments are near perfect here, confirming our results in A7. https://youtu.be/UnqRNKrYiR4

**A9–video 2.** Tracking results from V2, played back at two times normal speed. Since resolution per animal in V2 is lower than V1, and contrast is lower, posture estimation in V2 is also slightly worse than in V1. Importantly, however, identity assignment is very stable and accurate. https://youtu.be/OTP4dVSc7Es

Figure A10: Median of all normalised images (N=7161, 7040, 7153, 7076) for each of the four individuals from V1 in A7. Posture information was used to normalise each image sample, which was stable enough — also for TRex — to tell where the head is, and even to make out the ears on each side (brighter spots).

Figure A11: Median of all normalised images (N=1593, 1586, 1620, 1538) for each of the four individuals from V2 in A7. Resolution per animal is lower than in V1, but ears are still clearly visible.
B.1 Reproducing the data

The following sections document technical details for parameter choices that might have been omitted in the main text for conciseness. Variables that change with every call will be denoted by putting them inside brackets.

TGrabs (a part of TReX) performs the initial conversion of the original videos to a format that is usable in the tracking software. We were using version 1.1.4 of the software package. Converting, tracking and generating heatmaps was done using a batch script. For the conversion we passed these parameters when launching the application:

```
./tgrabs -i /path/to/video.MP4 -averaging_method mode -enable_absolute_difference false
     -correct_luminance -use_closing -blob_size_range [0.001,100]
```

Now the length of each part (parts are always set to be 10) is determined by retrieving the full length of the video in frames, e.g. 210036 for Video 1, and dividing it by 10 to get a step/part length of e.g. 21003 frames. The start and end frame for each part is determined using the step length:
\[
\text{start} = \text{step} \times \text{part} - 500 \text{ and}
\]
\[
\text{end} = \begin{cases} 
-1 & \text{if this is the last part} \\
\text{step}(\text{part} + 1) & \text{otherwise}
\end{cases}
\]

A value of -1 tells TRex to track until the end of the video for the last part. We introduce an overlap of 500 frames between parts, at the start of each part \(> 0\).

For the second step, tracking, we used a default.settings file containing these parameters:

- `blob_size_ranges = [[0.015,0.07]]`
- `calculate_posture = false`
- `cm_per_pixel = 0.017241`
- `match_mode = automatic`
- `matching_probability_threshold = 0.3`
- `track_max_reassign_time = 0.25`
- `track_max_speed = 10`
- `track_speed_decay = 1`
- `track_threshold = 80`
- `track_trusted_probability = 0.35`

And called TRex separately for each part:

```
./trex -d rootpath -i videoname -s rootpath/default.settings -auto_quit -auto_no_tracking_data -p part -analysis_range [start,end]
```

The heatmaps were saved using the pvinfo tool, which loaded the .results file generated by TRex and saves one or multiple npz files named data/{videoname}_heatmap_p{#part}_64x60_SPEED.npz, using the following command for each part:

```
./pvinfo -d rootpath -i videoname -s root/default.settings -auto_quit -p part -analysis_range [start,end] -heatmap_source 'SPEED#wcentroid' -heatmap_normalization value -heatmap_resolution 64 -heatmap_frames 150
```

The important parameters here are the `heatmap_` parameters, which define the kind and size of the exported heatmaps. For Figure 8 where we resolved affected individuals, we omitted the `-auto_no_tracking_data` parameter, so that individual trajectories were saved as well. This increases the number and on-disk size of saved archives, so beware.
B.1.1 Notes on the data format of individual NPZ files for trajectory data

No method can currently yield thousands of perfect trajectories. However, TRex is able to detect and keep track of all individuals while they were visually separated from others. Each time an individual overlaps with another, or hides from the camera in a different way, we assume that we lost track of said individual. This provides short segments of consecutive, uninterrupted frames which are used to estimate the speed of individuals, which is especially important for the following section. We assume to be tracking “the same event”, until TRex starts a new frame segment (for more detail on this, refer to the original paper [222]).

B.1.2 Tracking peaks in $\Delta(t)$

As described in the Methods section for “Narrowing down the first responders”, we first transformed a sequence of $\Delta(t)$ images (containing error data) into a movie file. This was done using the iPython notebook MeanCurves_Switching_ExtraBump.ipynb, included in the accompanying archives, which will pick out event 3 (preset frame indexes) from Video 1 and save a video file named n20HHS2019_69013-72988_performance.mov. This file is passed to TGrabs for conversion as:

```
./tgrabs -i N20HHS2019_69013-72988_performance.mov -enable_difference false -threshold 0
  -image_invert false -blob_size_range [0,1000] -o N20HHS2019_model
```

and then to TRex for tracking with the following settings inside n20HHS2019_model.settings:

```
blob_size_ranges = [[1,1000]]
gui_auto_scale_focus_one = false
gui_fish_color = "viridis"
match_mode = automatic
outline_curvature_range_ratio = 0.07
outline_resample = 1
track_do_history_split = false
track_max_individuals = 0
track_end_segment_for_speed = false
track_max_speed = 150
track_posture_threshold = 10
track_speed_decay = 1
track_threshold = 10
track_trusted_probability = 0.2
```

The command is then simply (assuming the settings file is in the same folder as the .pv video):

```
164
```
which saves a few .npz files in data/ with all of the tracked objects.

One can now use the notebook FindFirstResponders.ipynb, after adjusting file paths as in all notebooks, to generate the output found in Figure 8.

### B.2 Heatmap algorithm

TRex has an integrated feature that allowed us to generate dynamic heatmaps from this data, in which each 64px square of the original video is represented by one pixel (or cell). Since it has not been officially documented before, a short summary of the algorithm follows:

Conceptually, the video image is divided in a number of uniform-sized cells. The parameter that determines the size of each cell is `heatmap_resolution` (in our case 64). The grid is always symmetrical, meaning that it will represent an area larger than the original video size (e.g. 60x60 in our case, which would mean 3840x3840px even though the video is only 3840x2160px). This is not a problem because there is not actually a grid. Instead, the program uses an adaptively generated grid, in the form of a sparsely generated 2D binary tree (see Figure B1).

Each branch has four potentially existing children, which are only generated if they contain data. Walking along a path through the tree, one would start with the maximum resolution (3840) in both width and height for the root node (it spans the whole arena), and half the resolution with each reached branch. This is also the reason that the full tree resolution is forced to be the next-closest power of two. Each branch point also holds information about the range, the sum, sum of squared values, and the number of values, as well as the range of frames of the data-points stored under it. These metadata are updated every time a data-point is added or removed, and are also actively used to determine which parts of the tree needs updating in these cases.

When a data-point is added, which consists of (1) the frame it originates from, (2) x and y position, (3) a “value” that can be any metric like “speed”, (4) and an ID that makes it unique within the given frame but connects it to data in other frames (individual ID).

Leafs are special, since they denote the end of a path through the tree — once the path down the tree reaches a branch with the minimum resolution of 1px, we have arrived at a leaf. Each leaf holds actual data-points, along with the meta information that branches also store, and can store multiple of them. While the same pixel is only assigned to one individual per frame, the heatmaps are designed to organise and summarise information from multiple frames.
Accessing data-points inside a fully generated tree is easy. One only has to start at the root node and branch to the left or right, according to criteria that can be evaluated using the available meta information per branch. This is also the mechanism that is used to update an existing tree — which happens every frame, where the earliest frame is removed on the left side of the sliding-window to make room for a new frame on the right side of the sliding-window. One only has to update meta information and remove or add data-points for all branches along a path down to the respective leaf node for a given position — or prune an entire branch if it is empty. To simulate a grid with cell sizes of e.g. 64px, one has to simply stop at a cell size of 64px and return all of the data-points below it.

A final heatmap image is generated by walking through all nodes of the generated tree and accumulating all of the found values within a fixed-size matrix of — in our case — 60x60px with 2 channels. The first channel accumulates the value per cell, the second channel counts the number of samples per cell.
The inclusion of speed in the heatmaps, and thus also the requirement for tracking, is an attempt to provide the network with more information and improve prediction results. While we did introduce the speed channel mainly because we had trouble detecting the events, we did not perform a formal analysis that compares model performance when using only the density channel (which could even be estimated using only the color values of pixels). However, as shown in Figure B2 and Figure B3, the density channel clearly holds different information compared to the speed channel. It has lower standard deviation (speed: 0.257608, density: 0.29970386 for $R_{ij}(t) \forall t \in G, ij \in \text{image dimensions}$) because individuals seem to maintain similar speeds during phases of nominal behavior. This changes during the anomalies that we tried to detect, making the speed channel extra helpful — and it is likely the reason that the simple method works in the first place. The density channel seems better at representing trail networks, while the speed channel seems better for short-term changes.

**Figure B2:** That maximum of every pixel of the difference function $\Delta(t)$ for event 3 (Table 1), $\max_{ij} \Delta_{ij}(t)$, on the left for all channels $c$, on the right only for the density channel.
Figure B3: An example of three frames (43756, 44756, and 49756 of Video 1) and the respective raw heatmap data (leftmost column), where the speed channel is displayed at the top, and the density channel at the bottom. These raw data are presented next to the predictions of a well-performing model for the respective channel in the second column. The third column shows the 2D mean-squared-error for the predictions. The last column is the blurred derivative of the real measurements, which had to be substantially smoothed along the time-axis.
References


[17] Bhagwat, P. D. & Shingare, P. P. (2021). Increasing robustness of deep neural network models against adversarial attacks. *Journal of Physics: Conference Series*, 1797(1). Copyright © 2021. This work is published under http://creativecommons.org/licenses/by/3.0/ (the “License”). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License; Zuletzt aktualisiert - 2021-05-05.


181


Chapter 1

We thank A. Albi, F. Nowak, H. Hugo, D. E. Bath, F. Oberhauser, H. Naik, J. Graving, I. Etheredge for helping with their insights, by providing videos, for comments on the manuscript, testing the software and for frequent coffee breaks during development. The development of this software would not have been possible without them. We thank D. Mink and M. Groettrup providing additional video material of mice. We thank the reviewers and editors for their constructive and useful comments and suggestions.

Chapters 2 and 3

We thank all members of the Department of Collective Behaviour who assisted with the project, especially F. Oberhauser and J. Davidson for constructive discussions and ideas. We thank the IT and administrative staff for providing and helping with the technology that was necessary to realize these computationally, and storage-wise, intense projects.

All chapters

T.W. acknowledges the International Max Planck Research School (IMPRS) for Organismal Biology for providing an excellent graduate school community and access to courses and resources. T.W. and I.D.C. were supported by the DFG Centre of Excellence 2117 “Centre for the Advanced Study of Collective Behaviour” (ID: 422037984). I.D.C. acknowledges support from the NSF (IOS-1355061), the Office of Naval Research grant (ONR, N00014-19-1-2556), the Struktur- und Innovationsfunds für die Forschung of the State of Baden-Württemberg, the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy–EXC 2117-422037984, and the Max Planck Society.
Thesis

So many people contributed in different ways that it is impossible to list them all here. First of all, I want to thank Iain for so much advice. Thank you for listening when it was important, and for helping me through the more difficult parts of this journey wherever you could and, most importantly, for caring. I want to extend my thanks also to Mäggi, who has helped me numerous times in so many ways. All students at the MPI have memories like this, but my first one happened already during “my” first Symposium, which I ended by forgetting literally the only thing I had on me. Mäggi, where would my bag be without you. I also want to thank Francisca, who put up with me despite my “expert” handling of formalities. And Margit, in general and in particular for looking after me and my contracts. Angela, for providing technical support to others, and emotional support to me. To Helder, for the coffees and talks and impersonations. Frederic, for all the time we spent together at work and outside work, for the inspiring thoughts, probably even two or three lines of code, talks and advice. There are so many memories with everyone of our Department that I will never forget, and that I am thankful for. There are also all the others, outside the department, without whom I myself would be impossible and others, who were there for me in so many ways. You know who you are. Thank you.
Author contributions

The following authors contributed to Chapter 1:

**T. Walter**  Conceptualization, Data curation, Software, Formal analysis, Investigation, Visualization, Methodology, Writing - original draft, Project administration

**I. D. Couzin**  Conceptualization, Resources, Supervision, Funding acquisition, Project administration, Review and editing

The following authors contributed to Chapter 2:

**T. Walter**  Conceptualization, Software, Formal analysis, Visualization, Methodology, Writing, Project administration

**H. Hugo**  collected videos of *C. cyphergaster* on-site in Brazil, which were used in this chapter to develop and evaluate the method. He also contributed text to the description of the recording-site and -method in “Materials”, as well as feedback on the manuscript.

**I. D. Couzin**  Conceptualization, Resources, Supervision, Funding acquisition, Project administration, Review and editing

The following authors contributed to Chapter 3:

**T. Walter**  Conceptualization, Software, Formal analysis, Visualization, Methodology, Writing, Project administration

**H. Hugo**  recorded the videos of *C. cyphergaster* that were used in development and analysis, as also in Chapter 2. Again, he provided text for a description of the recording methods in "Materials and methods".

**F. Oberhauser**  labelled the castes in raw video clips using Detectron2, providing the raw data for the inter-software comparison and provided text for the "Materials and methods" section.

**I. D. Couzin**  Conceptualization, Resources, Supervision, Funding acquisition, Project administration, Review and editing