

# 1 TRex, a fast multi-animal tracking 2 system with markerless 3 identification, and 2D estimation of 4 posture and visual fields

5 **Tristan Walter<sup>1,2,3\*</sup> and Iain D Couzin<sup>1,2,3\*</sup>**

\*For correspondence:

[twalter@ab.mpg.de](mailto:twalter@ab.mpg.de) (TW);  
[icouzin@ab.mpg.de](mailto:icouzin@ab.mpg.de) (IDC)

6 <sup>1</sup>Max Planck Institute of Animal Behavior, Germany; <sup>2</sup>Centre for the Advanced Study of  
 7 Collective Behaviour, University of Konstanz, Germany; <sup>3</sup>Department of Biology,  
 8 University of Konstanz, Germany

9

10 **Abstract** Automated visual tracking of animals is rapidly becoming an indispensable tool for  
 11 the study of behavior. It offers a quantitative methodology by which organisms' sensing and  
 12 decision-making can be studied in a wide range of ecological contexts. Despite this, existing  
 13 solutions tend to be challenging to deploy in practice, especially when considering long and/or  
 14 high-resolution video-streams. Here, we present TRex, a fast and easy-to-use solution for  
 15 tracking a large number of individuals simultaneously [using background-subtraction](#) with  
 16 real-time (60Hz) tracking performance for up to approximately 256 individuals and estimates 2D  
 17 [visual-fields, outlines, and head/rear of bilateral animals](#), both in open and closed-loop contexts.  
 18 Additionally, TRex offers highly-accurate, deep-learning-based visual identification of up to  
 19 approximately 100 unmarked individuals, where it is between 2.5-46.7 times faster, and requires  
 20 2-10 times less memory, than comparable software (with relative performance increasing for  
 21 more organisms/longer videos) and provides interactive data-exploration within an intuitive,  
 22 platform-independent graphical user-interface.

## 24 Introduction

25 Tracking multiple moving animals (and multiple objects, generally) is important in various fields of  
 26 research such as behavioral studies, ecophysiology, biomechanics, and neuroscience ([Dell et al.](#)  
 27 [\(2014\)](#)). Many tracking algorithms have been proposed in recent years ([Ohayon et al. \(2013\)](#), [Fuku-](#)  
 28 [naga et al. \(2015\)](#), [Burgos-Artizzu et al. \(2012\)](#), [Rasch et al. \(2016\)](#)), often limited to/only tested with  
 29 a particular organism ([Hewitt et al. \(2018\)](#), [Branson et al. \(2009\)](#)) or type of organism (e.g. pro-  
 30 [tists](#), [Pennekamp et al. \(2015\)](#); fly larvae and worms, [Risse et al. \(2017\)](#)). Relatively few have been  
 31 tested with a range of organisms and scenarios ([Pérez-Escudero et al. \(2014\)](#), [Sridhar et al. \(2019\)](#),  
 32 [Rodríguez et al. \(2018\)](#)). Furthermore, many existing tools only have a specialized set of features,  
 33 struggle with very long or high-resolution ( $\geq 4K$ ) videos, or simply take too long to yield results. Existing  
 34 fast algorithms are often severely limited with respect to the number of individuals that can be  
 35 tracked simultaneously; for example xyTracker ([Rasch et al. \(2016\)](#)) allows for real-time tracking at  
 36 40Hz while accurately maintaining identities, and thus is suitable for closed-loop experimentation  
 37 (experiments where stimulus presentation can depend on the real-time behaviors of the individ-  
 38 uals, e.g. [Bath et al. \(2014\)](#), [Brembs and Heisenberg \(2000\)](#), [Bianco and Engert \(2015\)](#)), but has a  
 39 limit of being able to track only 5 individuals simultaneously. ToxTrac ([Rodríguez et al. \(2018\)](#)), a

40 software comparable to xyTracker in its set of features, is limited to 20 individuals and relatively  
 41 low frame-rates ( $\leq 25\text{fps}$ ). Others, while implementing a wide range of features and offering high-  
 42 performance tracking, are costly and thus limited in access (**Noldus et al. (2001)**). Perhaps with  
 43 the exception of proprietary software, one major problem at present is the severe fragmentation  
 44 of features across the various software solutions. For example, experimentalists must typically  
 45 construct work-flows from many individual tools: One tool might be responsible for estimating  
 46 the animal's positions, another for estimating their posture, another one for reconstructing visual  
 47 fields (which in turn probably also estimates animal posture, but does not export it in any way)  
 48 and one for keeping identities – correcting results of other tools post-hoc. It can take a very long  
 49 time to make them all work effectively together, adding what is often considerable overhead to  
 50 behavioral studies.

51 TRex, the software released with this publication (available at [trex.run](http://trex.run) under an Open-Source  
 52 license), has been designed to address these problems, and thus to provide a powerful, fast and  
 53 easy to use tool that will be of use in a wide range of behavioral studies. It allows users to track  
 54 moving objects/animals, as long as there is a way to separate them from the background (e.g.  
 55 static backgrounds, custom masks, as discussed below). In addition to the positions of individuals,  
 56 our software provides other per-individual metrics such as body shape and, if applicable, head-  
 57 /tail-position. This is achieved using a basic posture analysis, which works out of the box for most  
 58 organisms, and, if required, can be easily adapted for others. Posture information, which includes  
 59 the body center-line, can be useful for detecting e.g. courtship displays and other behaviors that  
 60 might not otherwise be obvious from mere positional data. Additionally, with the visual sense  
 61 often being one of the most important modalities to consider in behavioral research, we include  
 62 the capability for users to obtain a computational reconstruction of the visual fields of all individuals  
 63 (**Strandburg-Peshkin et al. 2013, Rosenthal et al. 2015**). This not only reveals which individuals are  
 64 visible from an individual's point-of-view, as well as the distance to them, but also which parts of  
 65 others' bodies are visible.

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

77 The fast tracking described above employs information about the kinematics of each organ-  
 78 ism in order to try to maintain their identities. This is very fast and useful in many scenarios, e.g.  
 79 where general assessments about group properties (group centroid, alignment of individuals, den-  
 80 sity, etc.) are to be made. However, when making conclusions about *individuals* instead, main-  
 81 taining identities perfectly throughout the video is a critical requirement. Every tracking method  
 82 inevitably makes mistakes, which, for small groups of two or three individuals or short videos, can  
 83 be corrected manually – at the expense of spending much more time on analysis, which rapidly  
 84 becomes prohibitive as the number of individuals to be tracked increases. To make matters worse,  
 85 when multiple individuals stay out of view of the camera for too long (such as if individuals move  
 86 out of frame, under a shelter, or occlude one another) there is no way to know who is whom  
 87 once they re-emerge. With no baseline truth available (e.g. using physical tags as in **Alarcón-Nieto**  
 88 **et al. (2018), Nagy et al. (2013)**; or marker-less methods as in **Pérez-Escudero et al. (2014), Romero-**  
 89 **Ferrero et al. (2019), Rasch et al. (2016)**), these mistakes can not be corrected and accumulate over  
 90 time, until eventually all identities are fully shuffled. To solve this problem (and without the need

91 to mark, or add physical tags to individuals), TRex can, at the cost of spending more time on analy-  
 92 sis (and thus not during live-tracking), automatically learn the identity of up to approximately 100  
 93 unmarked individuals based on their visual appearance. This machine-learning based approach,  
 94 herein termed *visual identification*, provides an independent source of information on the identity  
 95 of individuals, which is used to detect and correct potential tracking mistakes without the need for  
 96 human supervision.

97 In this paper, we will describe the most important functions of the software in a typical order  
 98 of execution. This is followed by an evaluation of these functions in terms of speed and reliability  
 99 using a wide range of experimental systems, including termites, fruit flies, locusts and multiple  
 100 species of schooling fish (although we stress that our software is not limited to such species).

101 Specifically regarding the visual identification of unmarked individuals in groups, *idtracker.ai*  
 102 is currently state-of-the-art, yielding high-accuracy (>99% in most cases) in maintaining consistent  
 103 identity assignments across entire videos (*Romero-Ferrero et al. (2019)*). Similarly to TRex, this is  
 104 achieved by training an artificial neural network to visually differentiate between individuals, and  
 105 using identity predictions from this network to avoid/correct tracking mistakes. Both approaches  
 106 work without human supervision, and are limited to approximately 100 individuals. Given that  
 107 *idtracker.ai* is the only currently available tool with visual identification for such large groups of  
 108 individuals, and also because of the quality of results, we will use it as a benchmark for our visual  
 109 identification system. Results will be compared in terms of both accuracy and computation speed,  
 110 showing TRex' ability to achieve the same high level of accuracy but typically at far higher speeds,  
 111 and with a much reduced memory requirement.

112 TRex is platform-independent and runs on all major operating systems (Linux, Windows, macOS)  
 113 and offers complete batch processing support, allowing users to efficiently process entire sets  
 114 of videos without requiring human intervention. All parameters can be accessed either through  
 115 settings files, from within the graphical user interface (or *GUI*), or using the command-line. The  
 116 user interface supports off-site access using a built-in web-server (although it is recommended  
 117 to only use this from within a secure VPN environment). Available parameters are explained in  
 118 the documentation directly as part of the GUI and on an external website (see below). Results  
 119 can be exported to independent data-containers (*NPZ*, or *CSV* for plain-text type data) for further  
 120 analyses in software of the user's choosing. We will not go into detail regarding the many GUI  
 121 functions since albeit being of great utility to the researcher, they are only the means to easily  
 122 apply the features presented herein. Some examples will be given in the main text and appendix,  
 123 but a comprehensive collection of all of them, as well as detailed documentation, is available in the  
 124 up-to-date online-documentation which can be found at [trex.run/docs](http://trex.run/docs).

## 125 Methods

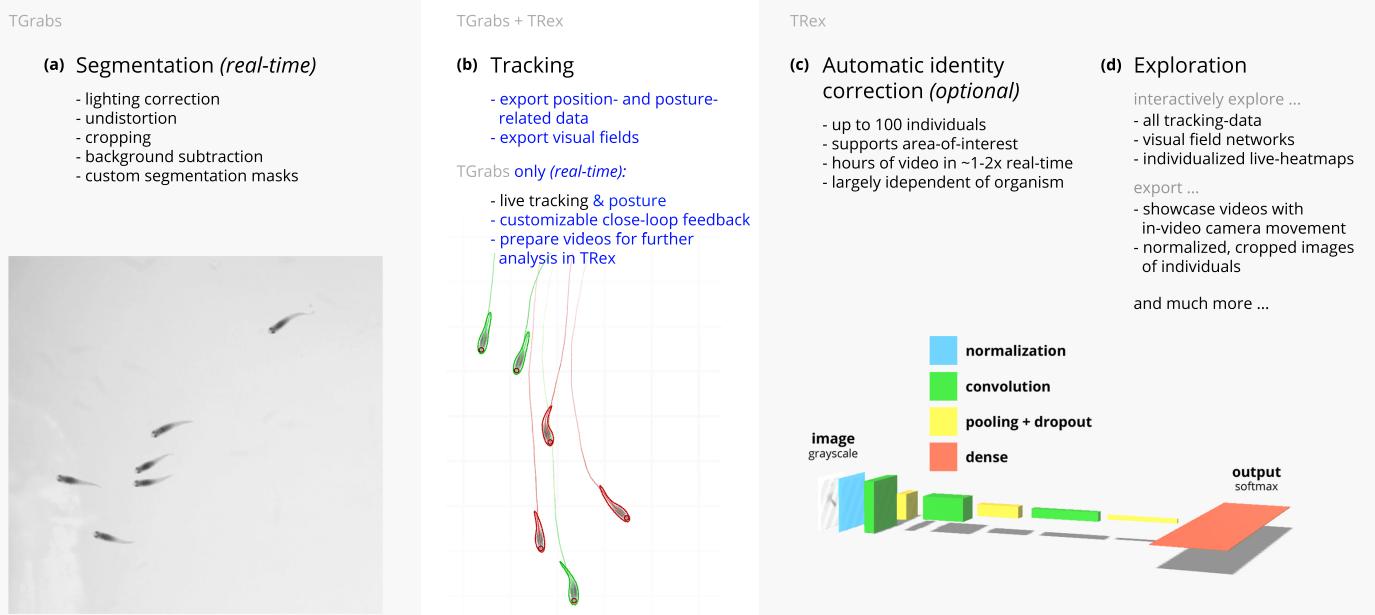
126 In the following sections we will explore some of the methods implemented in TRex and TGrabs,  
 127 as well as their most important features in a typical order of operations (see *Figure 2* for a flow  
 128 diagram), starting out with a raw video. We will then describe how trajectories are obtained and end  
 129 with the most technically involved features. The workflow for using our software is straightforward  
 130 and can be summarized in four stages:

- 131 1. **Segmentation** in TGrabs. When recording a video or converting a previously recorded file (e.g.  
 132 MP4, .AVI, etc.), it is segmented into background and foreground-objects (*blobs*). Results are  
 133 saved to a custom, non-proprietary video format (*PV*) (*Figure 1a*).
- 134 2. **Tracking** the video, either directly in TGrabs, or [in TRex after pre-processing](#), with access to  
 135 customizable visualizations and the ability to change tracking parameters on-the-fly. Here, we  
 136 will describe two types of data available within TRex, 2D posture- and visual-field estimation,  
 137 as well as real-time applications of such data (*Figure 1b*).
- 138 3. We then go into detail on how to train a neural network to perform visual identification of indi-  
 139 viduals, used in the process of **automatic identity correction** (*Figure 1c*). 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 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.

In summary, 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). This speed and its highly specialized nature comes at the cost of not having access to the rich graphical user interface and slower processing steps, such as deep-learning based identification, that TRex provides. TRex focusses 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 work-flow.



**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/g9EOi7FZHM0>

**160 Segmentation**

161 When an image is first received from a camera (or a video file), the objects of interest potentially  
 162 present in the frame must be **found** and cropped out. Several technologies are available to sep-  
 163 arate the foreground from the background (segmentation). Various machine learning algorithms  
 164 are frequently used to great effect, even for the most complex environments (**Hughey et al. 2018**,  
 165 **Robie et al. 2017**, **Francisco et al. 2019**). These more advanced approaches are typically beneficial  
 166 for the analysis of field-data or organisms that are very hard to see in video (e.g. very transparent  
 167 or low contrast objects/animals in the scene). **In these situations, where integrated methods might**  
 168 **not suffice, it is possible to segment objects from the background using external, e.g. deep-learning**  
 169 **based, tools (see next paragraph)**. However, for most laboratory experiments, simpler (and also  
 170 much faster), classical image-processing methods yield satisfactory results. **Thus, we provide as a**  
 171 **generically-useful capability *background-subtraction*, which is the default method by which objects**  
 172 **are segmented.** This can be used immediately in experiments where the background is relatively  
 173 static. Backgrounds are generated automatically by uniformly sampling images from the source  
 174 video(s) – different modes are available (min/max, mode and mean) for the user to choose from.  
 175 More advanced image-processing techniques like luminance equalization (which is useful when  
 176 lighting varies between images) image undistortion, and brightness/contrast adjustments are avail-  
 177 able in **TGrabs** and can enhance segmentation results – but come at the cost of slightly increased  
 178 **processing time.** Importantly, since many behavioral studies rely on  $\geq 4K$  resolution videos, we  
 179 heavily utilize the GPU (if available) to speed up most of the image-processing, allowing **TRex** to  
 180 scale well with increasing image resolution.

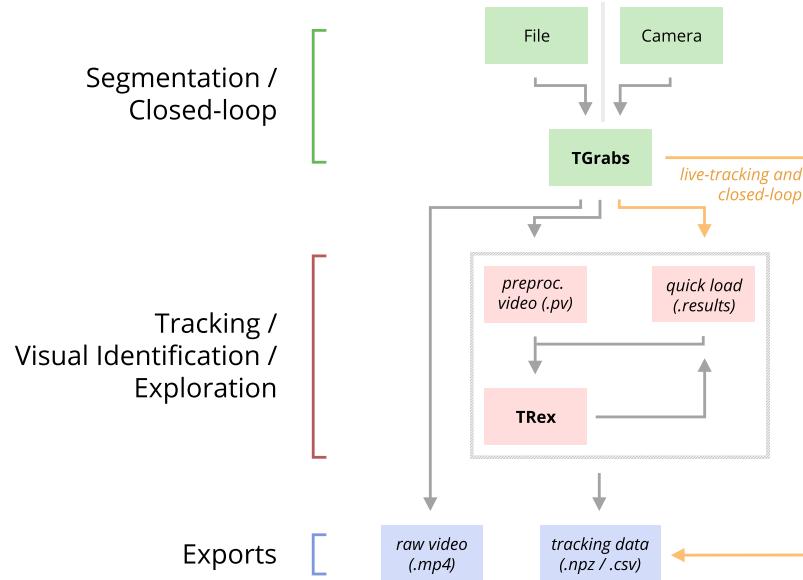
181 **TGrabs** can generally find any object in the video stream, and subsequently pass it on to the  
 182 tracking algorithm (next section), as long as either (i) the background is relatively static while the  
 183 objects move at least occasionally, (ii) the objects/animals of interest have enough contrast to the  
 184 background or (iii) the user provides an additional binary mask per frame which is used to separate  
 185 the objects **of interest** from the background, the typical means of doing this being by deep-learning  
 186 based segmentation (e.g. **Caelles et al. 2017**). These masks are expected to be in a video-format  
 187 themselves and correspond 1:1 in length and dimensions to the video that is to be analyzed. They  
 188 are expected to be binary, marking individuals in white and background in black. Of course, these  
 189 binary videos could be **used** on their own, but would not retain grey-scale information **of the ob-**  
 190 **jects.** There are a lot of possible applications where this could be useful; but generally, whenever  
 191 individuals are really hard to detect visually and need to be recognized by a different software (e.g.  
 192 a machine-learning-based **segmentation** like **Maninis et al. 2018**). Individual frames can then be  
 193 connected using our software as a second step.

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

**201 Tracking**

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

208 Note that this approach alone is very fast, but, as will be shown, is subject to error with respect  
 209 to maintaining individual identities. If that is required, there is a further step, outlined in Auto-



**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.

210      matic Visual Identification Based on Machine Learning below, which can be applied at the cost of  
 211      processing speed. First, however, we will discuss the general basis of tracking, which is common  
 212      to approaches that do, and do not, require identities to be maintained with high-fidelity. Tracking  
 213      can occur for two distinct categories, which are handled slightly differently by our software:

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

216      The first case assumes that the number of tracked objects in a frame cannot exceed a certain  
 217      expected number of objects (calculated automatically, or set by the user). This allows the algorithm  
 218      to make stronger assumptions, for example regarding noise, where otherwise "valid" objects (con-  
 219      forming to size expectations) are ignored due to their positioning in the scene (e.g. too far away  
 220      from previously lost individuals). In the second case, new objects may be generated until all viable  
 221      objects in a frame are assigned. While being more susceptible to noise, this is useful for tracking  
 222      a large number of objects, where counting objects may not be possible, or where there is a highly  
 223      variable number of objects to be tracked.

224      For a given video, our algorithm processes every frame sequentially, extending existing trajec-  
 225      tories (if possible) for each of the objects found in the current frame. Every object can only be  
 226      assigned to one trajectory, but some objects may not be assigned to any trajectory (e.g. in case the  
 227      number of objects exceeds the allowed number of individuals) and some trajectories might not  
 228      be assigned to any object (e.g. while objects are out of view). To estimate object identities across  
 229      frames we use an approach akin to the popular Kalman filter (Kalman, 1960) which makes pre-  
 230      dictions based on multiple noisy data streams (here, positional history and posture information).  
 231      In the initial frame, objects are simply assigned from top-left to bottom-right. In all other frames,  
 232      assignments are made based on probabilities (see appendix Matching an object to an object in the  
 233      next frame) calculated for every combination of object and trajectory. These probabilities repre-  
 234      sent the degree to which the program believes that "it makes sense" to extend an existing trajectory

235 with an object in the current frame, given its position and speed. Our tracking algorithm only con-  
 236 siders assignments with probabilities larger than a certain threshold, generally constrained to a  
 237 certain proximity around an object assigned in the previous frame.

238 Matching a set of objects in one frame with a set of objects in the next frame is representative  
 239 of a typical assignment problem, which can be solved in polynomial time (e.g. using the Hungarian  
 240 method *Kuhn 1955*). However, we found that, in practice, the computational complexity of  
 241 the Hungarian method can constrain analysis speed to such a degree that we decided to imple-  
 242 ment a custom algorithm, which we term tree-based matching, which has a better *average-case*  
 243 performance (see evaluation), even while having a comparatively bad *worst-case* complexity. Our  
 244 algorithm constructs a tree of all possible object/trajectory combinations in the frame and tries to  
 245 find a compatible (such that no objects/trajectories are assigned twice) set of choices, maximizing  
 246 the sum of probabilities amongst these choices (described in detail in the appendix Matching an  
 247 object to an object in the next frame). Problematic are situations where a large number of objects  
 248 are in close proximity of one another, since then the number of possible sets of choices grows ex-  
 249 ponentially. These situations are avoided by using a mixed approach: tree-based matching is used  
 250 most of the time, but as soon as the combinatorical complexity of a certain situation becomes too  
 251 great, our software falls back on using the Hungarian method. If videos are known to be prob-  
 252 lematic throughout (e.g. with >100 individuals consistently very close to each other), the user may  
 253 choose to use an approximate method instead (described in the appendix *section 4*), which simply  
 254 iterates through all objects and assigns each to the trajectory for which it has the highest proba-  
 255 bility and subsequently does not consider whether another object has an even higher probability  
 256 for that trajectory. While the approximate method scales better with an increasing number of in-  
 257 dividuals, it is "wrong" (seeing as it does not consider all possible combinations) – which is why it is  
 258 not recommended unless strictly necessary. However, since it does not consider all combinations,  
 259 making it more sensitive to parameter choice, it scales better for very large numbers of objects and  
 260 produces results good enough for it to be useful in very large groups (see *Appendix 4 Table A2*).

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

277 By default, objects suspected to contain overlapping individuals are split by thresholding their  
 278 background-difference image (see appendix *section 11*), continuously increasing the threshold un-  
 279 til the expected number (or more) similarly sized objects are found. Greyscale values and, more  
 280 generally, the shading of three-dimensional objects and animals often produces a natural gradi-  
 281 ent (see for example *Figure 4*) making this process surprisingly effective for many of the species  
 282 we tested with. Even when there is almost no visible gradient and thresholding produces holes in-  
 283 side objects, objects are still successfully separated with this approach. Missing pixels from inside  
 284 the objects can even be regenerated afterwards. The algorithm fails, however, if the remaining  
 285 objects are too small or are too different in size, in which case the overlapping objects will not be

286 assigned to any trajectory until all involved objects are found again separately in a later frame.  
 287 After an object is assigned to a specific trajectory, two kinds of data (posture and visual-fields)  
 288 are calculated and made available to the user, which will each be described in one of the follow-  
 289 ing subsections. In the last subsection, we outline how these can be utilized in real-time tracking  
 290 situations.

291 Posture Analysis

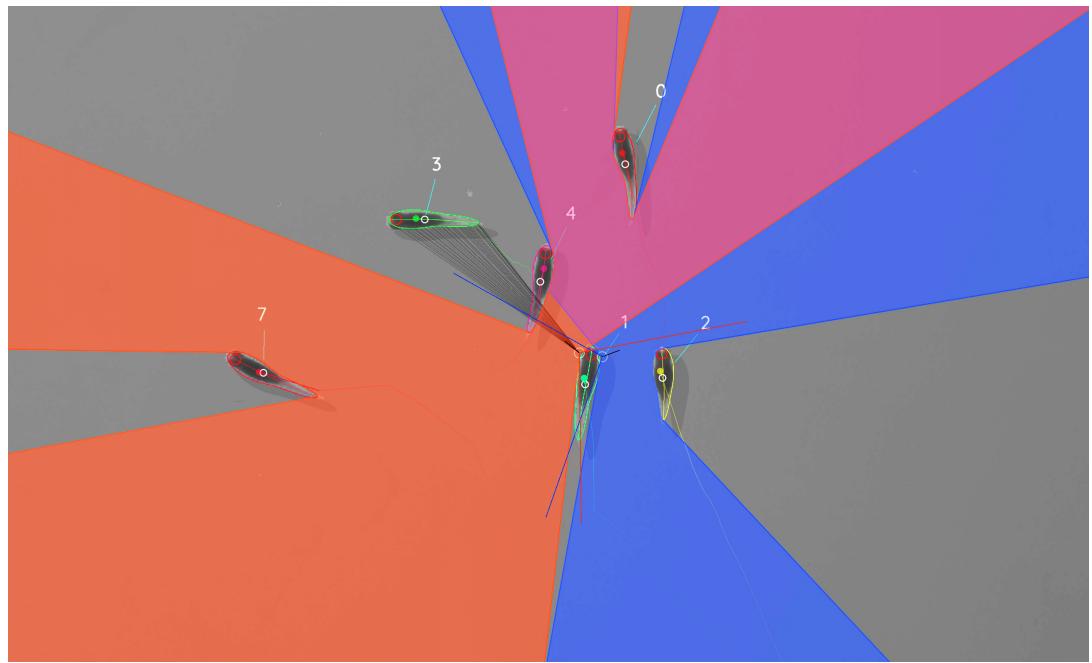
292 Groups of animals are often modeled as systems of simple particles (*Inada and Kawachi 2002, Cav-*  
*293 agna et al. 2010, Pérez-Escudero and de Polavieja 2011*), a reasonable simplification which helps  
 294 to formalize/predict behavior. However, intricate behaviors, like courtship displays, can only be  
 295 fully observed once the body shape and orientation are considered (e.g. using tools such as Deep-  
 296 PoseKit, *Graving et al. 2019*, LEAP *Pereira et al. (2019)*/SLEAP *Pereira et al. (2020)*, and DeepLab-  
 297 Cut, *Mathis et al. 2018*). TRex does not track individual body parts apart from the head and tail  
 298 (where applicable), but even the included simple and fast 2D posture estimator already allows for  
 299 deductions to be made about how an animal is positioned in space, bent and oriented – crucial  
 300 e.g. when trying to estimate the position of eyes/antennae as part of an analysis, where this is  
 301 required (e.g. *Strandburg-Peshkin et al. 2013, Rosenthal et al. 2015*). When detailed tracking of  
 302 all extremities is required, TRex offers an option that allows it to interface with third-party soft-  
 303 ware like DeepPoseKit (*Graving et al. 2019*), SLEAP (*Pereira et al. 2020*), or DeepLabCut (*Mathis*  
 304 *et al. 2018*). This option (`output_image_per_tracklet`), when set to true, exports cropped and (op-  
 305 tionally) normalised videos per individual that can be imported directly into these tools – where  
 306 they might perform better than the raw video. Normalisation, for example, can make it easier for  
 307 machine-learning algorithms in these tools to learn where body-parts are likely to be (see *Figure 7*)  
 308 and may even reduce the number of clicks required during annotation.

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

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

323 ii. The pointiest end of the outline is assumed, by default, to be either the tail or the head  
 324 (based on curvature and area between the outline points in question). Assignment of head vs. tail  
 325 can be set by the user, seeing as some animals might have "pointier" heads than tails (e.g. termite  
 326 workers, one of the examples we employ). Posture data coming directly from an image can be very  
 327 noisy, which is why the program offers options to simplify outline shapes using an Elliptical Fourier  
 328 Transform (EFT, see *Iwata et al. 2015, Kuhl and Giardina 1982*) or smoothing via a simple weighted  
 329 average across points of the curve (inspired by common subdivision techniques, see *Warren and*  
 330 *Weimer 2001*). The EFT allows for the user to set the desired level of approximation detail (via the  
 331 number of elliptic fourier descriptors, EFDs) and thus make it "rounder" and less jittery. Using an  
 332 EFT with just two descriptors is equivalent to fitting an ellipse to the animal's shape (as, for example,  
 333 xyTracker does), which is the simplest supported representation of an animal's body.

334 iii. The reference-point chosen in (ii) marks the start for the midline-algorithm. It walks both left  
 335 and right from this point, always trying to move approximately the same distance on the outline



**Figure 3.** 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° 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 3-video 1.** A clip from video 15, showing TRex' visual-field estimation for Individual 1.

[https://youtu.be/yEO\\_3lpZlzU](https://youtu.be/yEO_3lpZlzU)

336 (with limited wiggle-room), while at the same time minimizing the distance from the left to the right  
 337 point. This works well for most shapes and also automatically yields distances between a midline  
 338 point and its corresponding two points on the outline, estimating thickness of this object's body at  
 339 this point.

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

#### 344 Reconstructing 2D Visual Fields

345 Visual input is an important modality for many species (e.g. fish *Strandburg-Peshkin et al. 2013*,  
 346 *Bilotta and Saszik 2001* and humans *Colavita 1974*). Due to its importance in widely used model  
 347 organisms like zebrafish (*Danio rerio*), we decided to include the capability to conduct a 2-dimensional  
 348 reconstruction of each individual's visual field as part of the software. The requirements for this  
 349 are successful posture estimation and that individuals are viewed from above, as is usually the  
 350 case in laboratory studies.

351 The algorithm makes use of the fact that outlines have already been calculated during posture  
 352 estimation. Eye positions are estimated to be evenly distanced from the "snout" and will be spaced  
 353 apart depending on the thickness of the body at that point (the distance is based on a ratio, relative  
 354 to body-size, which can be adjusted by the user). Eye orientation is also adjustable, which influ-  
 355 ences the size of the stereoscopic part of the visual field. We then use ray-casting to intersect rays  
 356 from each of the eyes with all other individuals as well as the focal individual itself (self-occlusion).  
 357 Individuals not detected in the current frame are approximated using the last available posture.  
 358 Data are organized as a multi-layered 1D-image of fixed size for each frame, with each image prep-

359 resenting angles from  $-180^\circ$  to  $180^\circ$  for the given frame. Simulating a limited field-of-view would  
 360 thus be as simple as cropping parts of these images off the left and right sides. The different layers  
 361 per pixel encode:

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

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

### 373     Realtime Tracking Option for Closed-Loop Experiments

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

379     During development we worked with a mid-range gaming computer and Basler cameras at  
 380 90fps and  $2048^2$ px resolution, where drawbacks did not occur. [Running the program on hardware](#)  
 381 [with specifications below our recommendations \(see Results\), however, may affect frame-rates as](#)  
 382 [described below.](#)

383     *TRex* loads a prepared *Python* script, handing down an array of data per individual in every  
 384 frame. Which data fields are being generated and sent to the script is selected by the script. Avail-  
 385 able fields are:

- 386         • Position
- 387         • Midline information
- 388         • Visual field

389     If the script ([or any other part of the recording process](#)) takes too long to execute [in one frame](#),  
 390 [consecutive frames may be](#) dropped until a stable frame-rate can be achieved. This scales well for  
 391 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,  
 393 since even untracked frames are saved to disk, these inaccuracies can be fixed in *TRex* later. Alter-  
 394 natively, if live-tracking is enabled but closed-loop feedback is disabled, the program maintains  
 395 detected objects in memory and tracks them in an asynchronous thread (potentially introducing  
 396 wait time after the recording stops). When the program terminates, the tracked individual's data  
 397 are exported – along with a `results` file that can be loaded by the `tracker` at a later time.

398     In order to make this interface easy to use for prototyping and to debug experiments, the script  
 399 may be changed during its run-time and will be reloaded if necessary. Errors in the *Python* code  
 400 lead to a temporary pause of the closed-loop part of the program (not the recording) until all errors  
 401 have been fixed.

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

**406 Automatic Visual Identification Based on Machine Learning**

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

419 The only way to solve this problem is by providing an independent source of information from  
 420 which to infer identity of individuals, which is of course a principle we make use of all the time in  
 421 our everyday lives: Facial identification of con-specifics is something that [is easy for most humans](#),  
 422 to an extent where we sometimes recognize face-like features where there aren't any. Our natural  
 423 tendency to find patterns enables us to train experts on recognizing differences between animals,  
 424 even when they belong to a completely different taxonomic order. Tracking individuals is a de-  
 425 manding task, especially with large numbers of moving animals ([Liu et al. 2009](#) shows humans to  
 426 be effective for up to 4 objects). Human observers are able to solve simple memory recall tasks for  
 427 39 objects at only 92% correct (see [Humphrey and Khan 1992](#)), where the presented objects do not  
 428 even have to be identified individually (just classified as old/new) and contain more inherent varia-  
 429 tion than most con-specific animals would. Even with this being true, human observers are still the  
 430 most efficient solution in some cases (e.g. for long-lived animals in complex habitats). Enhancing  
 431 visual inter-individual differences by attaching physical tags is an effective way to make the task  
 432 easier and more straight-forward to automate. RFID tags are great at maintaining identities and  
 433 useful in many situations, but are also limited since individuals have to be in very close proximity  
 434 to a sensor in order to be detected ([Bonter and Bridge, 2011](#)). Attaching [fiducial markers](#) (such as  
 435 [QR codes](#)) to animals allows for a very large number of individuals to be uniquely identified at the  
 436 same time ([Gernat et al. 2018](#), [Wild et al. 2020](#), [Mersch et al. 2013](#), [Crall et al. 2015](#) *theoretically*  
 437 up to 7000) – and over greater distance than RFID tags. Generating codes can also be automated,  
 438 generating tags with optimal visual inter-marker distances ([Garrido-Jurado et al., 2016](#)), making it  
 439 feasible to identify a large number of individuals with minimal tracking mistakes.

440 While physical tagging is often an effective method by which to identify individuals, it requires  
 441 animals to be caught and manipulated, which can be difficult ([Mersch et al., 2013](#)) and is subject to  
 442 the physical limitations of the respective system. Tags have to be large enough so a program can  
 443 recognize it in a video stream. Even worse, especially with increased relative tag-size, the animal's  
 444 behavior may be affected by the presence of the tag, and there might be no way for experimenters  
 445 to necessarily know that it did. In addition, for some animals, like fish and termites, attachment of  
 446 tags that are effective for discriminating among a large number of individuals can be problematic  
 447 ([but possible, see e.g. Crall et al. 2015, Gernat et al. 2018, Wild et al. 2020](#)), or impossible.

448 Recognizing such issues, ([Pérez-Escudero et al., 2014](#)) first proposed an algorithm termed *id-*  
 449 *tracker*, generalizing the process of pattern recognition for a range of different species. Training an  
 450 expert program to tell individuals apart, by detecting slight differences in patterning on their bod-  
 451 ies, allows the correction of identities without any human involvement. Even while being limited  
 452 to about 15 individuals per group, this was a very promising approach. It became much improved  
 453 upon only a few years later by the same group in their software *idtracker.ai* ([Romero-Ferrero](#)  
 454 *et al.*, 2019), implementing a paradigm shift from explicit, hard-coded, color-difference detection  
 455 to using more general machine learning methods instead – increasing the supported group size

456 by an order of magnitude.

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

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

#### 480 Preparing Tracking-Data

481 Visual identification starts out only with the trajectories that the Tracking provides. Tracking, on  
 482 its own, is already an improvement over other solutions, especially since (unlike e.g. idtracker.  
 483 ai) TRex makes an effort to separate overlapping objects (see the Algorithm for splitting touching  
 484 individuals) and thus is able to keep track of individuals for longer (see **Appendix 4 Figure A2**).  
 485 Here, we – quite conservatively – assume that, after every problematic situation (defined in the  
 486 list below), the assignments made by our tracking algorithm are wrong. Whenever a problematic  
 487 situation is encountered as part of a trajectory, we split the trajectory at that point. This way,  
 488 all trajectories of all individuals in a video become an assortment of trajectory snippets (termed  
 489 "segments" from here on), which are clear of problematic situations, and for each of which the  
 490 goal is to find the correct identity ("correct" meaning that identities are consistently assigned to  
 491 the same *real* individual throughout the video). Situations are considered "problematic", and cause  
 492 the trajectory to be split, when:

- 493 • **The individual has been lost for at least one frame.** For example when individuals are  
 494 moving unexpectedly fast, are occluded by other individuals/the environment, or simply not  
 495 present anymore (e.g. eaten).
- 496 • **Uncertainty of assignment was too high (> 50%)** e.g. due to very high movement speeds  
 497 or extreme variation in size between frames. With simpler tracking tasks in mind, these seg-  
 498 ments are kept as *connected* tracks, but regarded as separate ones here.
- 499 • **Timestamps suggest skipped frames.** Missing frames in the video may cause wrong as-  
 500 signments and are thus treated as if the individuals have been lost. This distinction can only  
 501 be made if accurate frame timings are available (when recording using TGrabs or provided  
 502 alongside the video files in separate npz files).

503 Unless one of the above conditions becomes true, a segment is assumed to be consecutive  
 504 and connected; that is, throughout the whole segment, no mistakes have been made that lead to

505 identities being switched. Frames where all individuals are currently within one such segment at  
 506 the same time will henceforth be termed *global segments*.

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

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

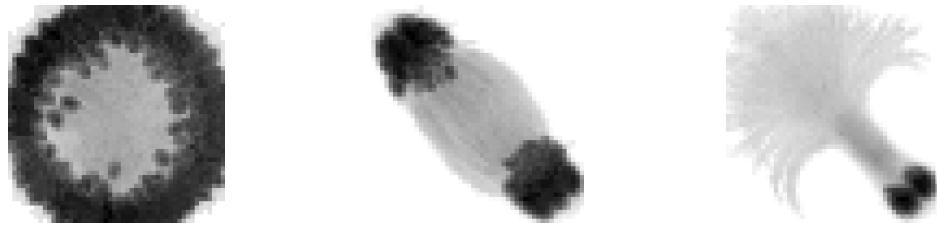
- 522 1. To capture as much as possible the variation due to an individual's own movement, as well as  
 523 within the background that it moves across, a "good" segment should be a segment where  
 524 all individuals move as much as possible and also travel as large a distance as possible. Thus,  
 525 we derive a per-individual *spatial coverage descriptor* for the given segment by dissecting the  
 526 arena (virtually) into a grid of equally sized, rectangular "cells" (depending on the aspect ratio  
 527 of the video). Each time an individual's center-point moves from one cell to the next, a counter  
 528 is incremented for that individual. To avoid situations where, for example, all individuals but  
 529 one are moving, we only use the lowest per-individual spatial coverage value to represent a  
 530 given segment.
- 531 2. It is beneficial to have more examples for the network to learn from. Thus, as a second sorting  
 532 criterion, we use the average number of samples per individual.

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

536 The number of visited cells may, at first, appear to be essentially equivalent to a spatially nor-  
 537 malized *distance travelled* (as used in `idtracker.ai`). In edge cases, where individuals never stop  
 538 or always stop, both metrics can be very similar. However, one can imagine an individual continu-  
 539 ously moving around in the same corner of the arena, which would be counted as an equally good  
 540 segment for that individual as if it had traversed the whole arena (and thus capturing all variable en-  
 541 vironmental factors). In most cases, using highly restricted movement for training is problematic,  
 542 and worse than using a shorter segment of the individual moving diagonally through the entire  
 543 space, since the latter captures more of the variation within background, lighting conditions and  
 544 the animals movement in the process.

#### 545 Minimizing the Variance Landscape by Normalizing Samples

546 A big strength of machine learning approaches is their resistance to noise in the data. Generally,  
 547 any machine learning method will likely still converge - even with noisy data. Eliminating unnec-  
 548 essary noise and degrees of freedom in the dataset, however, will typically help the network to  
 549 converge much more quickly: Tasks that are easier to solve will of course also be solved more ac-  
 550 curately within similar or smaller timescales. This is due to the optimizer not having to consider  
 551 various parts of the possible parameter-space during training, or, put differently, shrinking the  
 552 overall parameter-space to the smallest possible size without losing important information. The



(a) No normalization.

(b) Using the main body-axis (moments).

(c) Using posture information.

**Figure 4.** Comparison of different normalization methods. Images all stem from the same video and belong to the same identity. The video has previously been automatically corrected using the visual identification. Each object visible here consists of  $N$  images  $M_i, i \in [0, N]$  that have been accumulated into a single image using  $\min_{i \in [0, N]} M_i$ , with min being the element-wise minimum across images. The columns represent same samples from the same frames, but normalized in three different ways: In (a), images have not been normalized at all. Images in (b) have been normalized by aligning the objects along their main axis (calculated using *image-moments*), which only gives the axis within 0 to 180 degrees. In (c), all images have been aligned using posture information generated during the tracking process. As the images become more and more recognizable to us from left to right, the same applies to a network trying to tell identities apart: Reducing noise in the data speeds up the learning process.

553 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.

554 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 (Hu, 1962), yielding an angle within  $[0 - 180]^\circ$ . Of course, this means that

- 561 1. circular objects have a random (noisy) orientation
- 562 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 4**)
- 563 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.

564 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 4**). 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 7**).

#### 574 Guiding the Training Process

575 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.

576 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;

**Box 1. Calculating uniqueness for a frame**

```

588  Data: frame  $x$ 
589  Result: Uniqueness score for frame  $x$ 
590  uids = map{}
591   $\hat{p}(i \mid b)$  is the probability of blob  $b$  to be identity  $i$ 
592   $f(x)$  returns a list of the tracked objects in frame  $x$ 
593   $E(v) = (1 + \exp(-\pi)) / (1 + \exp(-\pi v))$  is a shift of roughly +0.5 and non-linear scaling of
594  values  $0 \leq v \leq 1$ 
595
596  foreach object  $b \in f(x)$  do
597    maxid =  $\arg \max_i \hat{p}(i \mid b)$  with  $i \in$  identities
598    if maxid  $\in$  uids then
599      | uids[maxid] =  $\max(\text{uids}[\text{maxid}], \hat{p}(\text{maxid}, b))$ 
600    else
601      | uids[maxid] =  $\hat{p}(\text{maxid}, b)$ 
602    end
603  end
604  return  $|\text{uids}|^{-1} |f(x)| * E(|\text{uids}|^{-1} (\sum_{i \in \text{uids}} \text{uids}[i]))$ 

```

**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  $|\text{uids}|^{-1} |f(x)|$  by the (scaled) mean probability deals with cases of low accuracy, where individuals switch every frame (but uniquely).

584 images of individuals within the same frame are required to be assigned different identities by the  
 585 networks predictions.

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

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

623 The closest example of such a procedure in `idtracker.ai` is the termination criterion after  
 624 *protocol 1*, which states that individual segments have to be consistent and certain enough in all  
 625 global segments in order to stop iterating. While this seems to be similar at first, the way accu-

racy 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).

#### 643 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 (*Lin et al., 2020*). Focal loss down-weights classes that are already reliably predicted by the network and in turn emphasizes neglected classes. An Adam optimizer (*Kingma and Ba, 2015*) 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 *Fukushima (1988)*, inspired originally by the work of Hubel and Wiesel on cats and rhesus monkeys (*Hubel and Wiesel 1959, Hubel and Wiesel 1963, Wiesel and Hubel 1966*), describing receptive fields and their hierarchical structure in the visual cortex. Soon afterward, in *LeCun et al. (1989)*, 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 *Romero-Ferrero et al. (2019)*, and is, as in most typical CNNs, (reverse-)pyramid-like. However, key differences between `TREx`'s 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 (*Perez and Wang, 2017*). We do not flip the image in 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 4*). Here, in fact, normalization of object orientation (during training and predictions) could be seen as a superior alternative to data

676 augmentation.

677 The input data for TReX' network is a single, cropped grayscale image of an individual (see *Fig-*  
 678 *ure 1c*). This image is first passed through a "lambda" layer (blue) that normalizes the pixel values,  
 679 dividing them by half the value limit of  $255/2 = 127.5$  and subtracting 1 – this moves them into the  
 680 range of  $[-1, 1]$ . From then on, sections are a combination of convolutional layers (kernel sizes of  
 681 16, 64 and 100 pixels), each followed by a 2D (2x2) max-pooling and a 2D spatial dropout layer  
 682 (with a rate of 0.25). Within each of these blocks the input data is reduced further, focussing it  
 683 down to information that is deemed important. Towards the end, the data are flattened and flow  
 684 into a densely connected layer (100 units) with exactly as many outputs as the number of classes.  
 685 The output is a vector with values between 0 and 1 for all elements of the vector, which, due to  
 686 softmax-activation, sum to 1.

687 Training commences by performing a stochastic gradient descent (using the Adam optimizer,  
 688 see *Kingma and Ba 2015*), which iteratively minimizes the error between network predictions and  
 689 previously known associations of images with identities – the original assignments within the initial  
 690 frame segment. The optimizer's behavior in the last five epochs is continuously observed and  
 691 training is terminated immediately if one of the following criteria is met:

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

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

- 702 • plateauing at/circling around a certain `val_loss` level
- 703 • plateauing around a certain uniqueness level

## 704 2. Accumulation of Additional Segments and Stopping-Criteria

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

- 710 1. Average uniqueness index (rounded to an integer percentage in 5% steps)
- 711 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
- 714 3. Minimum *cells visited* per individual (larger is better for the same reason as 2)
- 715 4. Minimum average samples per individual (larger is better)
- 716 5. Whether its image data has already been generated before (mostly for saving memory)
- 717 6. The uniqueness value is smaller than  $U_{prev}^2$  after 5 steps, with  $U_{prev}$  being the best uniqueness  
     value previous to the current accumulation step

719 With the help of these values, the segment list is sorted and the best segment selected to be  
 720 considered next. Adding a segment to a set of already active samples requires us to correct the  
 721 identities inside it, potentially switching temporary identities to represent the same *real* identities  
 722 as in our previous data. This is done by predicting identities for the new samples using the network

723 that has been trained on the old samples. Making mistakes here can lead to significant subsequent  
 724 problems, so merely plausible segments will be added - meaning only those samples are accepted  
 725 for which the predicted IDs are *unique* within each unobstructed sequence of frames for every  
 726 temporary identity. If multiple temporary individuals are predicted to be the same real identity,  
 727 the segment is saved for later and the search continues.

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

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

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

750 Training is considered to be successful globally, as soon as either (i) accumulative individual  
 751 gaps between sampled regions is less than 25% of the video length for all individuals, or (ii) unique-  
 752 ness has reached a value higher than  $1 - \frac{0.5}{N_{\text{id}}}$  (1) so that almost all detected identities are present  
 753 exactly once per frame. Otherwise, training will be continued as described above with additional  
 754 segments - each time extending the percentage of images seen by the network further.

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

### 758 3. The Final Training Unit

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

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

773 which our uniqueness metric was designed to detect.

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

#### 779 4. Assigning Identities Based on Network Predictions

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

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

$$R_{\max} = \max \left\{ \frac{P(i | S_j)}{P(i | S_k)}, \forall S_{j \neq k} \in \text{overlapping segments} \right\}$$

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

- 798 1. count the number of samples  $N_{me}$  in the current segment, and the number of samples  $N_{he}$  in  
 799 the other segment that this segment is compared to  
 800 2. calculate average probability vectors  $P_{me}$  and  $P_{he}$   
 801 3. if  $S(P_{me}, N_{me}) \geq S(P_{he}, N_{he})$ , then assign the current segment with the ID in question. Otherwise  
 802 assign the ID to the other segment. Where:

$$\begin{aligned} \text{norm}(x) &= \frac{x}{N_{me} + N_{he}}, \quad \text{sig}(x) = (1 + e^{2\pi(0.5-x)})^{-1} \\ S(p, x) &= \text{sig}(p) + \text{sig}(\text{norm}(x)). \end{aligned} \tag{2}$$

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

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

817 **Results**

818 Below we assess TRex' performance regarding three properties that are most important when us-  
 819 ing it (or in fact any tracking software) in practice: (i) The time it takes to perform tracking (ii) the  
 820 time it takes to perform automatic identity correction and (iii) the peak memory consumption when  
 821 correcting identities (since this is where memory consumption is maximal), as well as (iv) the accu-  
 822 racy of the produced trajectories after visual identification. While accuracy is an important metric  
 823 and specific to identification tasks, time and memory are typically of considerable practical im-  
 824 portance for all tasks. For example, tracking-speed may be the difference between only being  
 825 able to run a few trials or producing more reliable results with a much larger number of trials.  
 826 In addition, tracking speed can make a major difference as the number of individuals increases.  
 827 Furthermore, memory constraints can be extremely prohibitive making tracking over long video  
 828 sequences and/or for a large number of individuals extremely time-consuming, or impossible, for  
 829 the user.

**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.

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

**Table 1-source data 1.** Videos 7 and 8, as well as 13-11, are available as part of the original idtracker paper ([Pérez-Escudero et al. \(2014\)](#)). Many of the videos are part of yet unpublished data: Guppy videos have been recorded by A. Albi, videos with sunbleak (*Leucaspis 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 [videos.trex.run](#).

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

- 832 • Intel Core i9-7900X CPU  
 833 • NVIDIA Geforce 1080 Ti

- 834 • 64GB RAM
- 835 • NVMe PCIe x4 hard-drive
- 836 • Debian bullseye ([debian.org](https://www.debian.org))

837 As can be seen in the following sections (memory consumption, processing speeds, etc.) using  
 838 a high-end system is not necessary to run TRex and, anecdotally, we did not observe noticeable  
 839 improvements when using a solid state drive versus a normal hard drive. A video card (presently  
 840 an NVIDIA card due to the requirements of TensorFlow) is recommended for tasks involving visual  
 841 identification as such computations will take much longer without it – however, it is not required.  
 842 We decided to employ this system due to having a relatively cheap, compatible graphics card, as  
 843 well as to ensure that we have an easy way to produce direct comparisons with [idtracker.ai](#)  
 844 – which according to their website requires large amounts of RAM (32-128GB, [idtrackerai online](#)  
 845 [documentation](#), accessed 05/21/2020) and a fast solid-state drive.

846 **Table 1** shows the entire set of videos used in this paper, which have been obtained from multi-  
 847 ple sources (credited under the table) and span a wide range of different organisms, demonstrating  
 848 TRex' ability to track anything as long as it moves occasionally. [In response to some suggestions,](#)  
 849 [we extended this range to mammals and included two additional videos as part of our supplemen-](#)  
 850 [tary material. An analysis \(analogous to \*\*Table 3\*\*, see sections below\) of TRex' performance in these](#)  
 851 [videos can be found in the Appendix \(Stability for Highly Deformable Bodies\).](#) Videos involving a  
 852 large number (>100) of individuals are all the same species of fish since these were the only or-  
 853 ganisms we had available in such quantities. However, this is not to say that only fish could be  
 854 tracked efficiently in these quantities. We used the full dataset with up to 1024 individuals in one  
 855 video (video **0**) to evaluate raw tracking speed without visual identification and identity corrections  
 856 (next sub-section). However, since such numbers of individuals exceed the capacity of the neu-  
 857 ral network used for automatic identity corrections (compare also [Romero-Ferrero et al. \(2019\)](#)  
 858 who used a similar network), we only used a subset of these videos (videos **7** through **16**) to look  
 859 specifically into the quality of our visual identification in terms of keeping identities and its memory  
 860 consumption.

### 861 **Tracking: Speed and Accuracy**

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

869 Trials were run without posture/visual-field estimation enabled, where tracking generally, and  
 870 consistently, reaches speeds faster than real-time (processing times of 1.5-40% of the video du-  
 871 ration, 25-100Hz) even for a relatively large number of individuals (77-94.77% for up to 256 indi-  
 872 viduals, see [Appendix 4 Table A1](#)). Videos with more individuals (>500) were still tracked within  
 873 reasonable time of 235% to 358% of the video duration. As would be expected from these re-  
 874 sults, we found that combining tracking and recording in a single step generally leads to higher  
 875 processing speeds. The only situation where this was not the case was a video with 1024 individu-  
 876 als, which suggests that live-tracking (in TGrabs) handles cases with many individuals slightly worse  
 877 than offline tracking (in TRex). Otherwise, 5% to 35% shorter total processing times were measured  
 878 (14.55% on average, see [Appendix 4 Table A4](#)), compared to running TGrabs separately and then  
 879 tracking in TRex. These percentage differences, in most cases, reflect the ratio between the video  
 880 duration and the time it takes to track it, suggesting that most time is spent – by far – on the conver-  
 881 sion of videos. This additional cost can be avoided in practice when using TGrabs to record videos,  
 882 by directly writing to a custom format recognized by TRex, and/or using its live-tracking ability to

883 export tracking data immediately after the recording is stopped.

884 We also investigated trials that were run with posture estimation *enabled* and we found that real-  
 885 time speed could be achieved for videos with  $\leq 128$  individuals (see column "tracking" in **Appendix 4**  
**Table A4**). Tracking speed, when posture estimation is enabled, depends more strongly on the size  
 887 of individuals in the image.

888 Generally, tracking software becomes slower as the number of individuals to be tracked in-  
 889 creases, as a result of an exponentially growing number of combinations to consider during match-  
 890 ing. Comparing our mixed approach (see *Tracking*) to purely using the *Hungarian method* (also  
 891 known as the *Kuhn-Munkres algorithm*) shows that, while both perform similarly for few individu-  
 892 als, the Hungarian method is easily outperformed by our algorithm for larger groups of individuals  
 893 (as can be seen in **Appendix 4 Figure A3**). This might be due to custom optimizations regarding local  
 894 cliques of individuals, whereby we ignore objects that are too far away, and also as a result of our  
 895 optimized pre-sorting. The Hungarian method has the advantage of not leading to combinatorical  
 896 explosions in some situations – and thus has a lower *maximum* complexity while proving to be less  
 897 optimal in the *average* case. For further details, see the appendix: Matching an object to an object  
 898 in the next frame.

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

### 917 **Visual Identification: Accuracy**

918 Since the goal of using visual identification is to generate consistent identity assignments, we eval-  
 919 uated the accuracy of our method in this regard. As a benchmark, we compare it to manually  
 920 reviewed datasets as well as results from *idtracker.ai* for the same set of videos (where possi-  
 921 ble). In order to validate trajectories exported by either software, we manually reviewed multiple  
 922 videos with the help from a tool within *TRex* that allows to view each crossing and correct possi-  
 923 ble mistakes in-place. Assignments were deemed incorrect, and subsequently corrected by the  
 924 reviewer, if the centroid of a given individual was not contained within the object it was assigned  
 925 to (e.g. the individual was not part of the correct object). Double assignments per object are im-  
 926 possible due to the nature of the tracking method. Individuals were also forcibly assigned to the  
 927 correct objects in case they were visible but not detected by the tracking algorithm. After manual  
 928 corrections had been applied, "clean" trajectories were exported – providing a per-frame baseline  
 929 truth for the respective videos. A complete table of reviewed videos, and the percentage of re-  
 930 viewed frames per video, can be found in **Table 3**. For longer videos ( $>1$ h) we relied entirely on  
 931 a comparison between results from *idtracker.ai* and *TRex*. Their paper (*Romero-Ferrero et al.*  
 932 *(2019)*) suggests a very high accuracy of over 99.9% correctly identified individual images for most

**933** videos, which should suffice for most relevant applications and provide a good baseline truth. As  
**934** long as both tools produce sufficiently similar trajectories, we therefore know they have found the  
**935** correct solution.

**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**) 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.

video	# ind.	N <code>TREx</code>	% similar individuals	ø final uniqueness
<b>7</b>	100	5	99.9522 ± 0.2536	0.9758 ± 0.0018
<b>8</b>	59	5	99.7249 ± 0.5586	0.9356 ± 0.0358
<b>13</b>	10	5	99.9907 ± 0.3668	0.9812 ± 0.0013
<b>12</b>	10	5	99.9565 ± 0.8381	0.9698 ± 0.0024
<b>11</b>	10	5	99.9218 ± 1.116	0.9461 ± 0.0039
<b>14</b>	8	5	98.8185 ± 5.8095	0.9192 ± 0.0077
<b>15</b>	8	5	99.241 ± 4.2876	0.9576 ± 0.0023
<b>16</b>	8	5	99.8063 ± 1.9556	0.9481 ± 0.0025

**936** A direct comparison between `TREx` and `idtracker.ai` was not possible for videos **9** and **10**,  
**937** where `idtracker.ai` frequently exceeded hardware memory-limits and caused the application to  
**938** be terminated, or did not produce usable results within multiple days of run-time. However, we  
**939** were able to successfully analyse these videos with `TREx` and evaluate its performance by com-  
**940** paring to manually reviewed trajectories (see below in Visual Identification: Accuracy). Due to the  
**941** stochastic nature of machine learning, and thus the inherent possibility of obtaining different re-  
**942** sults in each run, as well as other potential factors influencing processing time and memory con-  
**943** sumption, both `TREx` and `idtracker.ai` have been executed repeatedly (5x `TREx`, 3x `idtracker.ai`).

**944** The trajectories exported by both `idtracker.ai` and `TREx` were very similar throughout (see  
**945** **Table 2**). While occasional disagreements happened, similarity scores were higher than 99% in  
**946** all cases (i.e. less than 1% of individuals have been differently assigned in each frame on average).  
**947** Most difficulties that *did* occur were, after manual review, attributable to situations where multiple  
**948** individuals cross over excessively within a short time-span. In each case that has been manually  
**949** reviewed, identities switched back to the correct individuals – even after temporary disagreement.  
**950** We found that both solutions occasionally experienced these same problems, which often occur  
**951** when individuals repeatedly come in and out of view in quick succession (e.g. overlapping with  
**952** other individuals). Disagreements were expected for videos with many such situations due to the  
**953** way both algorithms deal differently with them: `idtracker.ai` assigns identities only based on the  
**954** network output. In many cases, individuals continue to partly overlap even while already being  
**955** tracked, which results in visual artifacts and can lead to unstable predictions by the network and  
**956** causing `idtracker.ai`'s approach to fail. Comparing results from both `idtracker.ai` and `TREx` to  
**957** manually reviewed data (see **Table 3**) shows that both solutions consistently provide high accuracy  
**958** results of above 99.5% for most videos, but that `TREx` is slightly improved in all cases while also  
**959** having a better overall frame coverage per individual (99.65% versus `idtracker.ai`'s 97.93%, where  
**960** 100% would mean that all individuals are tracked in every frame; not shown). This suggests that the  
**961** splitting algorithm (see appendix, Algorithm for splitting touching individuals) is working to `TREx`'  
**962** advantage here.

**963** Additionally, while `TREx` could successfully track individuals in all videos without tags, we were

**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.

video metrics		review stats		% correct	
video	# ind.	reviewed (%)	of that interpolated (%)	TRex	idtracker.ai
7	100	100.0	0.23	$99.97 \pm 0.013$	$98.95 \pm 0.146$
8	59	100.0	0.15	$99.68 \pm 0.533$	$99.94 \pm 0.0$
9	15	22.2	8.44	$95.12 \pm 6.077$	N/A
10	10	100.0	1.21	$99.7 \pm 0.088$	N/A
13	10	100.0	0.27	$99.98 \pm 0.0$	$99.96 \pm 0.0$
12	10	100.0	0.59	$99.94 \pm 0.006$	$99.63 \pm 0.0$
11	10	100.0	0.5	$99.89 \pm 0.009$	$99.34 \pm 0.002$

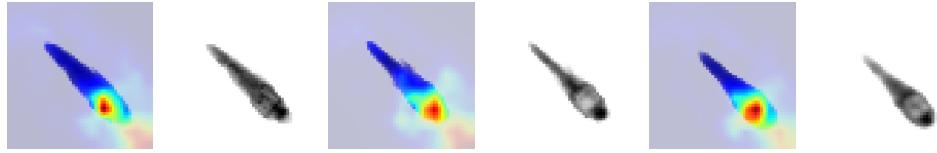
interested to see the effect of tags (in this case QR tags attached to locusts, see *Figure 5a*) on network training. In *Figure 5* we visualise differences in network activation, depending on the visual features available for the network to learn from, which are different between species (or due to 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, as shown in *Figure 4c*), 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 5a*/*Figure 5b*) 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 5d*) 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 5a*). 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.

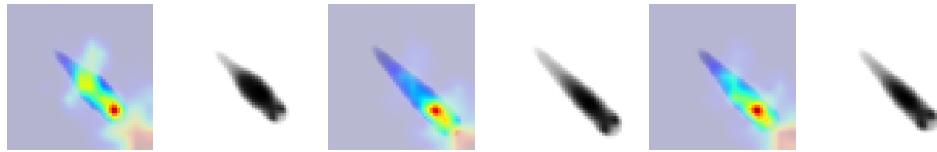
### Visual Identification: Memory Consumption

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

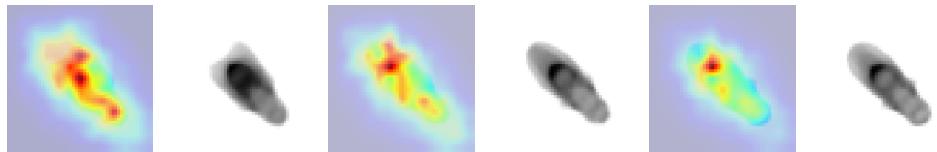
**Figure 5.** Activation differences for images of randomly selected individuals from four videos, next to a median image of the respective individual – which hides thin extremities, such as legs in (a) and (c). The captions in (a-d) detail the species per group and number of samples per individual. Colors represent the relative activation differences, with hotter colors suggesting bigger magnitudes, which are computed by performing a forward-pass through the network up to the last convolutional layer (using [keract](#)). The outputs for each identity are averaged and stretched back to the original image size by cropping and scaling according to the network architecture. Differences shown here are calculated per cluster of pixels corresponding to each filter, comparing average activations for images from the individual's class to activations for images from other classes.



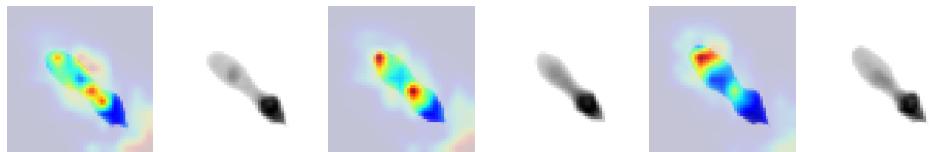
**(a)** Locusts from video 9 with 15 tagged individuals (N: 5101, 7942, 9974) – the only video with physical tags. The network activates more strongly in regions close to the tag, as well as the bottom right corner.



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



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



**(d)** 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.

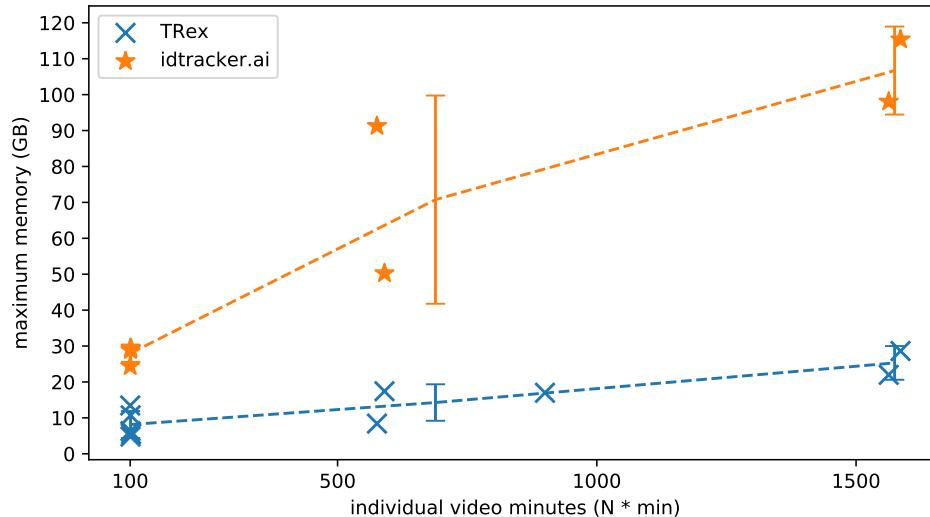
995 accurate measurements for Swap, Shared and Private separately, using [ps\\_mem](#).

996 As expected, differences in memory consumption are especially prominent for long videos (4-7x  
 997 lower maximum memory), and for videos with many individuals (2-3x lower). Since we already ex-  
 998 perienced significant problems tracking a long video (>3h) of only 8 individuals with `idtracker.ai`,  
 999 we did not attempt to further study its behavior in long videos with many individuals. However, we  
 1000 would expect `idtracker.ai`'s memory usage to increase even more rapidly than is visible in **Figure 6**  
 1001 since it retains a lot of image data (segmentation/pixels) in memory and we already had to "allow"  
 1002 it to relay to hard-disk in our efforts to make it work for Videos **8**, **14** and **16** (which slows down  
 1003 analysis). The maximum memory consumption across all trials was on average  $5.01 \pm 2.54$  times  
 1004 higher in `idtracker.ai`, ranging from 1.81 to 10.85 times the maximum memory consumption of  
 1005 `TREx` for the same video.

1006 Overall memory consumption for `TREx` also contains posture data, which contributes a lot to  
 1007 RAM usage. Especially with longer videos, disabling posture can lower the hardware needs for run-  
 1008 ning our software. If posture is to be retained, the user can still (more slightly) reduce memory  
 1009 requirements by changing the outline re-sampling scale (1 by default), which adjusts the outline

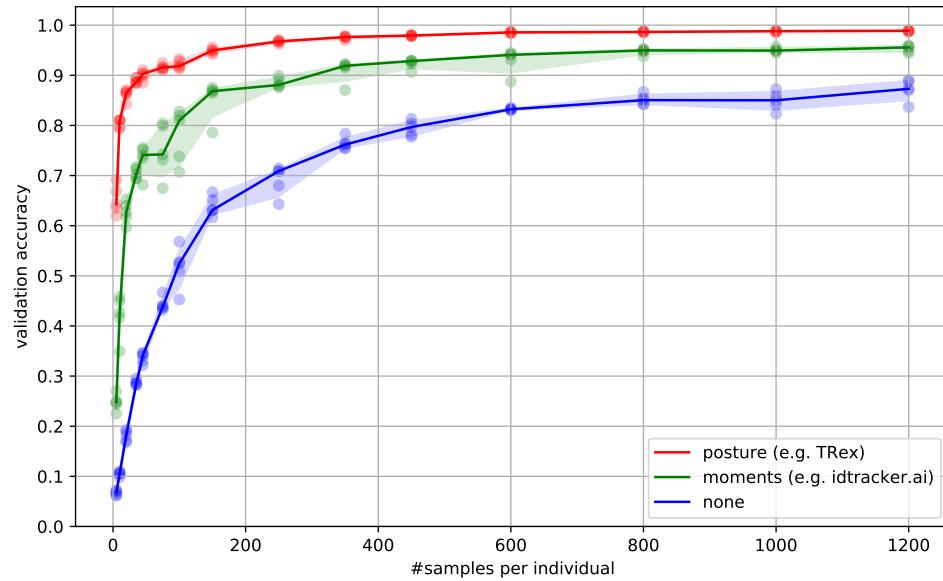
**Table 4.** Both TRex and idtracker.ai analysed the same set of videos, while continuously logging their memory consumption using an external tool. Rows have been sorted by video\_length \* #individuals, which seems to be a good predictor for the memory consumption of both solutions. idtracker.ai has 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.

video	#ind.	length	max.consec.	TRex memory (GB)	idtracker.ai memory (GB)
<b>12</b>	10	10min	26.03s	$\varnothing 4.88 \pm 0.23$ , max 6.31	$\varnothing 8.23 \pm 0.99$ , max 28.85
<b>13</b>	10	10min	36.94s	$\varnothing 4.27 \pm 0.12$ , max 4.79	$\varnothing 7.83 \pm 1.05$ , max 29.43
<b>11</b>	10	10min	28.75s	$\varnothing 4.37 \pm 0.32$ , max 5.49	$\varnothing 6.53 \pm 4.29$ , max 29.32
<b>7</b>	100	1min	5.97s	$\varnothing 9.4 \pm 0.47$ , max 13.45	$\varnothing 15.27 \pm 1.05$ , max 24.39
<b>15</b>	8	72min	79.4s	$\varnothing 5.6 \pm 0.22$ , max 8.41	$\varnothing 35.2 \pm 4.51$ , max 91.26
<b>10</b>	10	10min	1.91s	$\varnothing 6.94 \pm 0.27$ , max 10.71	N/A
<b>9</b>	15	60min	7.64s	$\varnothing 13.81 \pm 0.53$ , max 16.99	N/A
<b>8</b>	59	10min	102.35s	$\varnothing 12.4 \pm 0.56$ , max 17.41	$\varnothing 35.3 \pm 0.92$ , max 50.26
<b>14</b>	8	195min	145.77s	$\varnothing 12.44 \pm 0.8$ , max 21.99	$\varnothing 35.08 \pm 4.08$ , max 98.04
<b>16</b>	8	198min	322.57s	$\varnothing 16.15 \pm 1.6$ , max 28.62	$\varnothing 49.24 \pm 8.21$ , max 115.37



**Figure 6.** 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.

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 [Appendix 4](#) [Figure A3](#)), users of the visual identification might experience a decrease in training accuracy or speed (see [Figure 7](#)).



**Figure 7.** 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 4](#)): 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  $\geq 75$  samples. This property can help significantly in situations with more crossings, when longer global segments are harder to find.

#### 1014 Visual Identification: Processing Time

1015 Automatically correcting the trajectories (to produce consistent identity assignments) means that  
 1016 additional time is spent on the training and application of a network, specifically for the video  
 1017 in question. Visual identification builds on some of the other methods described in this paper  
 1018 (tracking and posture estimation), naturally making it by far the most complex and time-consuming  
 1019 process in `TRex` – we thus evaluated how much time is spent on the entire sequence of all required  
 1020 processes. For each run of `TRex` and `idtracker.ai`, we saved precise timing information from start  
 1021 to finish. Since `idtracker.ai` reads videos *directly* and preprocesses them again each run, we used  
 1022 the same starting conditions with our software for a direct comparison:

1023 A trial starts by converting/preprocessing a video in `TGrabs` and then immediately opening it in  
 1024 `TRex`, where automatic identity corrections were applied. `TRex` terminated automatically after sat-  
 1025 isfying a correctness criterion (high uniqueness value) according to equation (1). It then exported  
 1026 trajectories, as well as validation data (similar to `idtracker.ai`), concluding the trial. The sum of  
 1027 time spent within `TGrabs` and `TRex` gives the total amount of time for that trial. For the purpose of  
 1028 this test it would not have been fair to compare only `TRex` processing times to `idtracker.ai`, but  
 1029 it is important to emphasize that conversion could be skipped entirely by using `TGrabs` to record  
 1030 videos directly from a camera instead of opening an existing video file.

1031 In [Table 5](#) we can see that video length and processing times did not correlate directly. In-  
 1032 deed, conversion times eventually overtook processing times with increasing video-length if the

**Table 5.** Evaluating time-cost for automatic identity correction – comparing to results from `idtracker.ai`. Timings consist of preprocessing time in `TGrabs` plus network training in `TRex`, which are shown separately as well as combined (*ours (min)*,  $N = 5$ ). The time it takes to analyse videos strongly depends on the number of individuals and how many usable samples per individual the initial segment provides. The length of the video factors in as well, as does the stochasticity of the gradient descent (training). `idtracker.ai` timings ( $N = 3$ ) contain the whole tracking and training process from start to finish, using its `terminal_mode` (v3). Parameters have been manually adjusted per video and setting, to the best of our abilities, spending at most one hour per configuration. For videos **16** and **14** we had to set `idtracker.ai` to storing segmentation information on disk (as compared to in RAM) to prevent the program from being terminated for running out of memory.

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

number of individuals remained the same. Furthermore, the time it took to track and correct a video was shorter when the initial segment (column "sample" in the table) was longer (and as such likely of higher quality/capturing more visual intra-individual variation). Conversion times correlated strongly with the total video-length (in frames) and not the number of individuals, suggesting conversion was only constrained by video-decoding/reading speeds and not by (pre-)processing.

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

## Conclusions

`TRex` provides a comprehensive, powerful and easy to use software solution for tracking animals. Its tracking accuracy is at the state-of-the-art while typically being 2.57x to 46.74x 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  $\leq 256$  without posture estimation ( $\leq 128$  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.

`TRex` is a versatile and fast program, which we have designed to enable researchers to study 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 and other features. 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. Not only does the increased processing-speeds benefit researchers, but the contributions we provide to data exploration should not be

**1061** underestimated as well – merely making data more easily accessible right out-of-the-box, such as  
**1062** visual fields and live-heatmaps, has the potential to reveal features of group- and individual be-  
**1063** haviour which have not been visible before. TRex makes information on multiple timescales of  
**1064** events available simultaneously, and sometimes this is the only way to detect interesting proper-  
**1065** ties (e.g. trail formation in termites).

### **1066 Future extensions**

**1067** Since the software is already actively used within the Max Planck Institute of Animal Behavior, re-  
**1068** ported issues have been taken into consideration during development. However, certain theoreti-  
**1069** cal, as well as practically observed, limitations remain:

- 1070** • Posture: While almost all shapes can be detected correctly (by adjusting parameters), some  
**1071** shapes – especially round shapes – are hard to interpret in terms of "tail" or "head". This  
**1072** means that only the other image alignment method (moments) can be used. However, it  
**1073** does introduce some limitations e.g. calculating visual fields is impossible.
- 1074** • Tracking: Predictions, if the wrong direction is assumed, might go really far away from where  
**1075** the object is. Objects are then "lost" for a fixed amount of time (parameter). This can be  
**1076** "fixed" by shortening this time-period, though this leads to different problems when the soft-  
**1077** ware does not wait long enough for individuals to reappear.
- 1078** • General: Barely visible individuals have to be tracked with the help of deep learning (e.g.  
**1079** using *Caelles et al. (2017)*) and a custom-made mask per video frame, prepared in an external  
**1080** program of the users choosing
- 1081** • Visual identification: All individuals have to be *visible* and *separate* at the same time, at least  
**1082** once, for identification to work at all. Visual identification, e.g. with very high densities of  
**1083** individuals, can thus be very difficult. This is a hard restriction to any software since find-  
**1084** ing consecutive global segments is the underlying principle for the successful recognition of  
**1085** individuals.

**1086** We will continue updating the software, increasingly addressing the above issues (and likely  
**1087** others), as well as potentially adding new features. During development we noticed a couple of  
**1088** areas where improvements could be made, both theoretical and practical in nature. Specifically,  
**1089** incremental improvements in analysis speed could be made regarding visual identification by us-  
**1090** ing the trained network more sporadically – e.g. it is not necessary to predict every image of very  
**1091** long consecutive segments, since, even with fewer samples, prediction values are likely to converge  
**1092** to a certain value early on. A likely more potent change would be an improved "uniqueness" algo-  
**1093** rithm, which, during the accumulation phase, is better at predicting which consecutive segment  
**1094** will improve training results the most. This could be done, for example, by taking into account the  
**1095** variation between images of the same individual. Other planned extensions include:

- 1096** • (Feature): We want to have a more general interface available to users, so they can create  
**1097** their own plugins. Working with the data in live-mode, while applying their own filters. As  
**1098** well as specifically being able to write a plugin that can detect different species/annotate  
**1099** them in the video.
- 1100** • (Crossing solver): Additional method optimized for splitting overlapping, solid-color objects.  
**1101** The current method, simply using a threshold, is effective for many species but often pro-  
**1102** duces large holes when splitting objects consisting of largely the same color.

**1103** To obtain the most up-to-date version of TRex, please download it at [trex.run](http://trex.run) or update your  
**1104** existing installation according to our instructions listed on [trex.run/docs/install.html](http://trex.run/docs/install.html).

### **1105 Software and Licenses**

**1106** TRex is published under the GNU GPLv3 license (see [here](#) for permissions granted by GPLv3). All of  
**1107** the code has been written by the first author of this paper (a few individual lines of code from other

1108 sources have been marked inside the code). While none of these libraries are distributed alongside  
 1109 TRex (they have to be provided separately), the following libraries are used: OpenCV ([opencv.org](http://opencv.org))  
 1110 is a core library, used for all kinds of image manipulation. GLFW ([glfw.org](http://glfw.org)) helps with opening applica-  
 1111 tion windows and maintaining graphics contexts, while DearImGui ([github.com/ocornut/imgui](https://github.com/ocornut/imgui))  
 1112 helps with some more abstractions regarding graphics. pybind11 ([Jakob et al. \(2017\)](https://jakob.github.io/pybind11/)) for Python  
 1113 integration within a C++ environment. miniLZO ([oberhumer.com/opensource/lzo](http://oberhumer.com/opensource/lzo)) is used for com-  
 1114 pression of PV frames. Optional bindings are available to FFmpeg ([ffmpeg.org](http://ffmpeg.org)) and libpng libraries,  
 1115 if available. (optional) GNU Libmicrohttpd ([gnu.org/software/libmicrohttpd](http://gnu.org/software/libmicrohttpd)), if available, can be  
 1116 used for an HTTP interface of the software, but is non-essential.

### 1117 Acknowledgments

1118 We thank A. Albi, F. Nowak, H. Hugo, D. E. Bath, F. Oberhauser, H. Naik, J. Graving, I. Etheredge  
 1119 for helping with their insights, by providing videos, for comments on the manuscript, testing the  
 1120 software and for frequent coffee breaks during development. The development of this software  
 1121 would not have been possible without them. **Furthermore, we thank D. Mink and M. Groettrup for**  
 1122 **so quickly coming to our aid during the review process and providing additional video material of**  
 1123 **mice.** IDC acknowledges support from the NSF (IOS-1355061), the Office of Naval Research grant  
 1124 (ONR, N00014-19-1-2556), the Struktur- und Innovationsfunds für die Forschung of the State of  
 1125 Baden-Württemberg, the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)  
 1126 under Germany's Excellence Strategy-EXC 2117-422037984, and the Max Planck Society.

### 1127 References

- 1128 AbuBaker A, Qahwaji R, Ipson S, Saleh M. One Scan Connected Component Labeling Technique. In: *2007 IEEE International Conference on Signal Processing and Communications*; 2007. p. 1283–1286. doi: <https://doi.org/10.1109/ICSPC.2007.4728561>.
- 1131 Alarcón-Nieto G, Graving JM, Klarevas-Irby JA, Maldonado-Chaparro AA, Mueller I, Farine DR. An automated  
 1132 barcode tracking system for behavioural studies in birds. *Methods in Ecology and Evolution*. 2018; 9(6):1536–  
 1133 1547. doi: <https://doi.org/10.1111/2041-210X.13005>.
- 1134 Bath DE, Stowers JR, Hörmann D, Poehlmann A, Dickson BJ, Straw AD. FlyMAD: rapid thermogenetic  
 1135 control of neuronal activity in freely walking Drosophila. *Nature Methods*. 2014; 11(7):756–762. doi:  
 1136 <https://doi.org/10.1038/nmeth.2973>.
- 1137 Bertsekas DP. A new algorithm for the assignment problem. *Mathematical Programming*. 1981; 21(1):152–171.  
 1138 doi: <https://doi.org/10.1007/BF01584237>.
- 1139 Bianco IH, Engert F. Visuomotor transformations underlying hunting behavior in zebrafish. *Current Biology*.  
 1140 2015; 25(7):831–846. doi: <https://doi.org/10.1016/j.cub.2015.01.042>.
- 1141 Bilotta J, Saszik S. The zebrafish as a model visual system. *International Journal of Developmental Neuro-  
 1142 science*. 2001; 19(7):621–629. doi: [https://doi.org/10.1016/S0736-5748\(01\)00050-8](https://doi.org/10.1016/S0736-5748(01)00050-8).
- 1143 Bonter DN, Bridge ES. Applications of radio frequency identification (RFID) in ornithological research: a review.  
 1144 *Journal of Field Ornithology*. 2011; 82(1):1–10. doi: <https://doi.org/10.1111/j.1557-9263.2010.00302.x>.
- 1145 Branson K, Robie AA, Bender J, Perona P, Dickinson MH. High-throughput ethomics in large groups of  
 1146 Drosophila. *Nature Methods*. 2009; 6(6):451–457. doi: <https://doi.org/10.1038/nmeth.1328>.
- 1147 Brembs B, Heisenberg M. The operant and the classical in conditioned orientation of *Drosophila melanogaster*  
 1148 at the flight simulator. *Learning & Memory*. 2000; 7(2):104–115. doi: [10.1101/lm.7.2.104](https://doi.org/10.1101/lm.7.2.104).
- 1149 Burgos-Artizzu XP, Dollár P, Lin D, Anderson DJ, Perona P. Social behavior recognition in continuous  
 1150 video. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition* IEEE; 2012. p. 1322–1329. doi:  
 1151 <https://doi.org/10.1109/CVPR.2012.6247817>.
- 1152 Caelles S, Maninis K, Pont-Tuset J, Leal-Taixé L, Cremers D, Van Gool L. One-Shot Video Object Segmen-  
 1153 tation. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*; 2017. p. 5320–5329. doi:  
 1154 <https://doi.org/10.1109/CVPR.2017.565>.

- 1155 Cavagna A, Cimarelli A, Giardina I, Parisi G, Santagati R, Stefanini F, Tavarone R. From empirical data to inter-  
 1156 individual interactions: unveiling the rules of collective animal behavior. Mathematical Models and Methods  
 1157 in Applied Sciences. 2010; 20(supp01):1491–1510. doi: <https://doi.org/10.1142/S0218202510004660>.
- 1158 Chang F, Chen C. A Component-Labeling Algorithm Using Contour Tracing Technique. In: 2013 12th Inter-  
 1159 national Conference on Document Analysis and Recognition, vol. 3 Los Alamitos, CA, USA: IEEE Computer  
 1160 Society; 2003. p. 741. <https://doi.ieeecomputersociety.org/10.1109/ICDAR.2003.1227760>, doi: 10.1109/IC-  
 1161 DAR.2003.1227760.
- 1162 Clausen J, Branch and bound algorithms-principles and examples. University of Copenhagen; 1999. [Online;  
 1163 accessed 22-Oct-2020]. <http://www2.imm.dtu.dk/courses/04232/TSPtext.pdf>.
- 1164 Colavita FB. Human sensory dominance. Perception & Psychophysics. 1974; 16(2):409–412. doi:  
 1165 <https://doi.org/10.3758/BF03203962>.
- 1166 Crall JD, Gravish N, Mountcastle AM, Combes SA. BEEtag: a low-cost, image-based track-  
 1167 ing system for the study of animal behavior and locomotion. PloS One. 2015; 10(9). doi:  
 1168 <https://doi.org/10.1371/journal.pone.0136487>.
- 1169 Dell AI, Bender JA, Branson K, Couzin ID, de Polavieja GG, Noldus LP, Pérez-Escudero A, Perona P, Straw AD,  
 1170 Wikelski M, et al. Automated image-based tracking and its application in ecology. Trends in Ecology & Evolution.  
 1171 2014; 29(7):417–428. doi: <https://doi.org/10.1016/j.tree.2014.05.004>.
- 1172 Francisco FA, Nührenberg P, Jordan AL. A low-cost, open-source framework for tracking and behavioural  
 1173 analysis of animals in aquatic ecosystems. bioRxiv. 2019; p. 571232. doi: <https://doi.org/10.1101/571232>.
- 1174 Fredman ML, Tarjan RE. Fibonacci heaps and their uses in improved network optimization algorithms. Journal  
 1175 of the ACM (JACM). 1987; 34(3):596–615. doi: <https://doi.org/10.1145/28869.28874>.
- 1176 Fukunaga T, Kubota S, Oda S, Iwasaki W. GroupTracker: video tracking system for multiple ani-  
 1177 mals under severe occlusion. Computational Biology and Chemistry. 2015; 57:39–45. doi:  
 1178 <https://doi.org/10.1016/j.combiolchem.2015.02.006>.
- 1179 Fukushima K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural  
 1180 Networks. 1988; 1(2):119–130. doi: [https://doi.org/10.1016/0893-6080\(88\)90014-7](https://doi.org/10.1016/0893-6080(88)90014-7).
- 1181 Garrido-Jurado S, Muñoz-Salinas R, Madrid-Cuevas FJ, Medina-Carnicer R. Generation of fiducial marker  
 1182 dictionaries using mixed integer linear programming. Pattern Recognition. 2016; 51:481–491. doi:  
 1183 <https://doi.org/10.1016/j.patcog.2015.09.023>.
- 1184 Gernat T, Rao VD, Middendorf M, Dankowicz H, Goldenfeld N, Robinson GE. Automated monitoring of behavior  
 1185 reveals bursty interaction patterns and rapid spreading dynamics in honeybee social networks. Proceedings  
 1186 of the National Academy of Sciences. 2018; 115(7):1433–1438. <https://www.pnas.org/content/115/7/1433>,  
 1187 doi: 10.1073/pnas.1713568115.
- 1188 Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Teh YW,  
 1189 Titterington M, editors. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics,  
 1190 vol. 9 of Proceedings of Machine Learning Research Chia Laguna Resort, Sardinia, Italy: PMLR; 2010. p.  
 1191 249–256. <http://proceedings.mlr.press/v9/glorot10a.html>.
- 1192 Graving JM, Chae D, Naik H, Li L, Koger B, Costelloe BR, Couzin ID. DeepPoseKit, a software toolkit  
 1193 for fast and robust animal pose estimation using deep learning. eLife. 2019; 8:e47994. doi:  
 1194 <https://doi.org/10.7554/eLife.47994>.
- 1195 He L, Chao Y, Suzuki K, Wu K. Fast connected-component labeling. Pattern recognition. 2009; 42(9):1977–1987.  
 1196 doi: <https://doi.org/10.1016/j.patcog.2008.10.013>.
- 1197 Hewitt BM, Yap MH, Hodson-Tole EF, Kennerley AJ, Sharp PS, Grant RA. A novel automated rodent tracker  
 1198 (ART), demonstrated in a mouse model of amyotrophic lateral sclerosis. Journal of neuroscience methods.  
 1199 2018; 300:147–156. doi: <https://doi.org/10.1016/j.jneumeth.2017.04.006>.
- 1200 Hu MK. Visual pattern recognition by moment invariants. IRE Transactions on Information Theory. 1962;  
 1201 8(2):179–187. doi: <https://doi.org/10.1109/TIT.1962.1057692>.
- 1202 Hubel DH, Wiesel TN. Receptive fields of single neurones in the cat's striate cortex. The Journal of Physiology.  
 1203 1959; 148(3):574. doi: [10.1113/jphysiol.1959.sp006308](https://doi.org/10.1113/jphysiol.1959.sp006308).

- 1204 Hubel DH, Wiesel TN. Receptive fields of cells in striate cortex of very young, visually inexperienced kittens.  
 1205 Journal of Neurophysiology. 1963; 26(6):994–1002. doi: <https://doi.org/10.1152/jn.1963.26.6.994>.
- 1206 Hughey LF, Hein AM, Strandburg-Peshkin A, Jensen FH. Challenges and solutions for studying collective animal behaviour in the wild. Philosophical Transactions of the Royal Society B: Biological Sciences. 2018; 373(1746):20170005. <https://royalsocietypublishing.org/doi/abs/10.1098/rstb.2017.0005>, doi: 10.1098/rstb.2017.0005.
- 1210 Humphrey GK, Khan SC. Recognizing novel views of three-dimensional objects. Canadian Journal of Psychology/Revue canadienne de psychologie. 1992; 46(2):170. doi: <https://doi.org/10.1037/h0084320>.
- 1212 Inada Y, Kawachi K. Order and Flexibility in the Motion of Fish Schools. Journal of Theoretical Biology. 2002; 214(3):371 – 387. <http://www.sciencedirect.com/science/article/pii/S002251930192449X>, doi: <https://doi.org/10.1006/jtbi.2001.2449>.
- 1215 Iwata H, Ebana K, Uga Y, Hayashi T. Genomic prediction of biological shape: elliptic fourier analysis and kernel partial least squares (PLS) regression applied to grain shape prediction in rice (*Oryza sativa* L.). PloS One. 2015; 10(3). doi: <https://doi.org/10.1371/journal.pone.0120610>.
- 1218 Jakob W, Rhinelander J, Moldovan D, pybind11 – Seamless operability between C++11 and Python. Wenzel Jakob; 2017. [Online; accessed 22-Oct-2020]. <https://github.com/pybind/pybind11>.
- 1220 Kalman RE. A New Approach to Linear Filtering and Prediction Problems. Journal of Basic Engineering. 1960 03; 82(1):35–45. <https://doi.org/10.1115/1.3662552>, doi: 10.1115/1.3662552.
- 1222 Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. In: Bengio Y, LeCun Y, editors. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*; 2015. <http://arxiv.org/abs/1412.6980>, arXiv:1412.6980.
- 1225 Kuhl FP, Giardina CR. Elliptic Fourier features of a closed contour. Computer Graphics and Image Processing. 1982; 18(3):236–258. doi: [https://doi.org/10.1016/0146-664X\(82\)90034-X](https://doi.org/10.1016/0146-664X(82)90034-X).
- 1227 Kuhn HW. The Hungarian method for the assignment problem. Naval Research Logistics Quarterly. 1955; 2(1-2):83–97. doi: <https://doi.org/10.1002/nav.3800020109>.
- 1229 Land AH, Doig AG. In: Jünger M, Liebling TM, Naddef D, Nemhauser GL, Pulleyblank WR, Reinelt G, Rinaldi G, Wolsey LA, editors. *An Automatic Method for Solving Discrete Programming Problems* Berlin, Heidelberg: Springer Berlin Heidelberg; 2010. p. 105–132. [https://doi.org/10.1007/978-3-540-68279-0\\_5](https://doi.org/10.1007/978-3-540-68279-0_5), doi: 10.1007/978-3-540-68279-0\_5.
- 1233 LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD. Backpropagation applied to handwritten zip code recognition. Neural Computation. 1989; 1(4):541–551. doi: <https://doi.org/10.1162/neco.1989.1.4.541>.
- 1236 Lin T, Goyal P, Girshick R, He K, Dollár P. Focal Loss for Dense Object Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2020; 42(2):318–327. doi: <https://doi.org/10.1109/TPAMI.2018.2858826>.
- 1238 Little JD, Murty KG, Sweeney DW, Karel C. An algorithm for the traveling salesman problem. Operations Research. 1963; 11(6):972–989. doi: <https://doi.org/10.1287/opre.11.6.972>.
- 1240 Liu T, Chen W, Xuan Y, Fu X. The effect of object features on multiple object tracking and identification. In: *International Conference on Engineering Psychology and Cognitive Ergonomics* Springer; 2009. p. 206–212. doi: [https://doi.org/10.1007/978-3-642-02728-4\\_22](https://doi.org/10.1007/978-3-642-02728-4_22).
- 1243 Maninis KK, Caelles S, Chen Y, Pont-Tuset J, Leal-Taixé L, Cremers D, Van Gool L. Video Object Segmentation Without Temporal Information. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI). 2018; doi: <https://doi.org/10.1109/TPAMI.2018.2838670>.
- 1246 Mathis A, Mamidanna P, Cury KM, Abe T, Murthy VN, Mathis MW, Bethge M. DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. Nature Neuroscience. 2018; 21(9):1281–1289. doi: <https://doi.org/10.1038/s41593-018-0209-y>.
- 1249 Mersch DP, Crespi A, Keller L. Tracking individuals shows spatial fidelity is a key regulator of ant social organization. Science. 2013; 340(6136):1090–1093. doi: 10.1126/science.1234316.
- 1251 Munkres J. Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics. 1957; 5(1):32–38. doi: <https://doi.org/10.1137/0105003>.

- 1253 Nagy M, Vásárhelyi G, Pettit B, Roberts-Mariani I, Vicsek T, Biro D. Context-dependent hierarchies  
 1254 in pigeons. *Proceedings of the National Academy of Sciences*. 2013; 110(32):13049–13054. doi:  
 1255 <https://doi.org/10.1073/pnas.1305552110>.
- 1256 Noldus LP, Spink AJ, Tegelenbosch RA. EthoVision: a versatile video tracking system for automation of be-  
 1257 havioral experiments. *Behavior Research Methods, Instruments, & Computers*. 2001; 33(3):398–414. doi:  
 1258 <https://doi.org/10.3758/BF03195394>.
- 1259 Ohayon S, Avni O, Taylor AL, Perona P, Egnor SR. Automated multi-day tracking of marked mice  
 1260 for the analysis of social behaviour. *Journal of Neuroscience Methods*. 2013; 219(1):10–19. doi:  
 1261 <https://doi.org/10.1016/j.jneumeth.2013.05.013>.
- 1262 Pennekamp F, Schtickzelle N, Petchey OL. BEMOVI, software for extracting behavior and morphology  
 1263 from videos, illustrated with analyses of microbes. *Ecology and Evolution*. 2015; 5(13):2584–2595. doi:  
 1264 <https://doi.org/10.1002/ece3.1529>.
- 1265 Pereira TD, Aldarondo DE, Willmore L, Kislin M, Wang SSH, Murthy M, Shaevitz JW. Fast animal pose estimation  
 1266 using deep neural networks. *Nature Methods*. 2019; 16(1):117–125. doi: <https://doi.org/10.1038/s41592-018-0234-5>.
- 1268 Pereira TD, Tabris N, Li J, Ravindranath S, Papadoyannis ES, Wang ZY, Turner DM, McKenzie-Smith G, Kocher  
 1269 SD, Falkner AL, Shaevitz JW, Murthy M. SLEAP: Multi-animal pose tracking. *bioRxiv*. 2020; <https://www.biorxiv.org/content/early/2020/09/02/2020.08.31.276246>, doi: [10.1101/2020.08.31.276246](https://doi.org/10.1101/2020.08.31.276246).
- 1271 Perez L, Wang J. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *CoRR*.  
 1272 2017; abs/1712.04621. <http://arxiv.org/abs/1712.04621>.
- 1273 Pérez-Escudero A, de Polavieja G. Collective animal behavior from Bayesian estimation and probability match-  
 1274 ing. *Nature Precedings*. 2011; p. 1–1. doi: <https://doi.org/10.1038/npre.2011.5939.2>.
- 1275 Pesant G, Quimper CG, Zanarini A. Counting-based search: Branching heuristics for constraint  
 1276 satisfaction problems. *Journal of Artificial Intelligence Research*. 2012; 43:173–210. doi:  
 1277 <https://doi.org/10.1613/jair.3463>.
- 1278 Pérez-Escudero A, Vicente-Page J, Hinz RC, Arganda S, de Polavieja GG. idTracker: tracking individuals  
 1279 in a group by automatic identification of unmarked animals. *Nature Methods*. 2014; 11(7):743. doi:  
 1280 <https://doi.org/10.1038/nmeth.2994>.
- 1281 Ramshaw L, Tarjan RE. A Weight-Scaling Algorithm for Min-Cost Imperfect Matchings in Bipartite Graphs.  
 1282 In: *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*; 2012. p. 581–590. doi:  
 1283 <https://doi.org/10.1109/FOCS.2012.9>.
- 1284 Ramshaw L, Tarjan RE, On Minimum-Cost Assignments in Unbalanced Bipartite Graphs. HP Labs, Palo Alto, CA,  
 1285 USA; 2012. <https://www.hpl.hp.com/techreports/2012/HPL-2012-40.pdf>, Technical Report, HPL-2012-40R1,  
 1286 [Online; Accessed 22-Oct-2020].
- 1287 Rasch MJ, Shi A, Ji Z. Closing the loop: tracking and perturbing behaviour of individuals in a group in real-time.  
 1288 *bioRxiv*. 2016; p. 071308. doi: <https://doi.org/10.1101/071308>.
- 1289 Risse B, Berh D, Otto N, Klämbt C, Jiang X. FIMTrack: An open source tracking and locomotion  
 1290 analysis software for small animals. *PLoS Computational Biology*. 2017; 13(5):e1005530. doi:  
 1291 <https://doi.org/10.1371/journal.pcbi.1005530>.
- 1292 Robie AA, Seagraves KM, Egnor SR, Branson K. Machine vision methods for analyzing social interactions. *Journal*  
 1293 of Experimental Biology. 2017; 220(1):25–34. doi: <https://doi.org/10.1242/jeb.142281>.
- 1294 Rodriguez A, Zhang H, Klaminder J, Brodin T, Andersson PL, Andersson M. ToxTrac: a fast and ro-  
 1295 bust software for tracking organisms. *Methods in Ecology and Evolution*. 2018; 9(3):460–464. doi:  
 1296 <https://doi.org/10.1111/2041-210X.12874>.
- 1297 Romero-Ferrero F, Bergomi MG, Hinz RC, Heras FJ, de Polavieja GG. idtracker.ai: tracking all indi-  
 1298 viduals in small or large collectives of unmarked animals. *Nature Methods*. 2019; 16(2):179. doi:  
 1299 <https://doi.org/10.1038/s41592-018-0295-5>.
- 1300 Rosenthal SB, Twomey CR, Hartnett AT, Wu HS, Couzin ID. Revealing the hidden networks of interaction in mo-  
 1301 bile animal groups allows prediction of complex behavioral contagion. *Proceedings of the National Academy*  
 1302 of Sciences. 2015; 112(15):4690–4695. doi: <https://doi.org/10.1073/pnas.1420068112>.

- 1303 Sridhar VH, Roche DG, Gingins S. Tracktor: Image-based automated tracking of animal movement and  
1304 behaviour. Methods in Ecology and Evolution. 2019; 10(6):815–820. doi: <https://doi.org/10.1111/2041-210X.13166>.
- 1306 Strandburg-Peshkin A, Twomey CR, Bode NW, Kao AB, Katz Y, Ioannou CC, Rosenthal SB, Torney CJ, Wu HS,  
1307 Levin SA, et al. Visual sensory networks and effective information transfer in animal groups. Current Biology.  
1308 2013; 23(17):R709–R711. doi: <https://doi.org/10.1016/j.cub.2013.07.059>.
- 1309 Suzuki K, Horiba I, Sugie N. Linear-time connected-component labeling based on sequential local operations.  
1310 Computer Vision and Image Understanding. 2003; 89(1):1–23. doi: [https://doi.org/10.1016/S1077-3142\(02\)00030-9](https://doi.org/10.1016/S1077-3142(02)00030-9).
- 1312 Thomas DJ, Matching Problems with Additional Resource Constraints. Universität Trier; 2016. <https://doi.org/10.25353/ubtr-xxxx-7644-a670/>, doi: 10.25353/ubtr-xxxx-7644-a670/, Doctoral Thesis.
- 1314 Warren J, Weimer H. Subdivision Methods for Geometric Design: A Constructive Approach. 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2001. ISBN: 1558604464.
- 1316 Weixiong Z, Branch-and-Bound Search Algorithms and Their Computational Complexity. University of Southern California/Marina Del Rey Information Sciences Institute; 1996. <https://apps.dtic.mil/sti/citations/ADA314598>, Technical Report, ISI/RR-96-443, [Online; Accessed 22-Oct-2020].
- 1319 Wiesel TN, Hubel DH. Spatial and chromatic interactions in the lateral geniculate body of the rhesus monkey.  
1320 Journal of Neurophysiology. 1966; 29(6):1115–1156. doi: <https://doi.org/10.1152/jn.1966.29.6.1115>.
- 1321 Wild B, Dormagen DM, Zachariae A, Smith ML, Traynor KS, Brockmann D, Couzin ID, Landgraf T. Social networks  
1322 predict the life and death of honey bees. bioRxiv. 2020; <https://www.biorxiv.org/content/early/2020/09/01/2020.05.06.076943>, doi: 10.1101/2020.05.06.076943.
- 1324 Williams L. Casting curved shadows on curved surfaces. In: *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*; 1978. p. 270–274. doi: <https://doi.org/10.1145/800248.807402>.

## 1326 Appendix 1

1327 **Interface and installation requirements**

1328 While all options are available from the command-line and a screen is not required, TReX  
 1329 offers a rich, yet straight-forward to use, interface to local as well as remote users. Accom-  
 1330 panied by the integrated documentation for all parameters, each stating purpose, type and  
 1331 value ranges, as well as a comprehensive online documentation, the learning curve for new  
 1332 users is usually quite steep. Especially to the benefit of new users, we evaluated the pa-  
 1333 rameter space on a wide dataset (fish, termites, locusts) and determined which parameters  
 1334 work best in most use-cases to set their default values (evaluation not part of this paper).

1335 Compiled, read-to-use binaries are available for all major operating systems (Windows,  
 1336 Linux, MacOS). However, it should be possible to compile the software yourself for any Unix-  
 1337 or Windows-based system ( $\geq 8$ ), possibly with minor adjustments. Tested setups include:

- 1338 • Windows, Linux, MacOS
- 1339 • A computer with  $\geq 16\text{GB}$  RAM is recommended
- 1340 • OpenCV<sup>o</sup> libraries  $\geq v3.3$
- 1341 • Python libraries  $\geq v3.6$ , as well as additional packages such as:
- 1342 • Keras  $\approx v2.2$  with one of the following backends installed
  - 1343 – Tensorflow  $< v2^b$  (either CPU-based, or GPU-based)
  - 1344 – Theano <sup>c</sup>
- 1345 • GPU-based recognition requires an NVIDIA graphics-card and drivers (see Tensorflow
- 1346 documentation)

1347 For detailed download/installation instructions and up-to-date requirements, please re-  
 1348 fer to the documentation at [trex.run/install](#).

1349 The interface is structured into groups (see **Appendix 1 Figure A1**), categorized by the  
 1350 typical use-case:

- 1351 1. The main menu, containing options for loading/saving, options for the timeline and  
 1352 reanalysis of parts of the video
- 1353 2. Timeline and current video playback information
- 1354 3. Information about the selected individual
- 1355 4. Display options and an interactive "omni-box" for viewing and changing parameters
- 1356 5. General status information about TReX and the Python integration

1357 **Workflow**

1358 TReX can be opened in one of two ways: (i) Simply starting the application (e.g. using the  
 1359 operating systems' file-browser), (ii) using the command-line. If the user simply opens the  
 1360 application, a file opening dialog displays a list of compatible files as well as information on  
 1361 a selected files content. Certain startup parameters can be adjusted from within the graph-  
 1362 ical user-interface, before confirming and loading up the file (see **Appendix 1 Figure A3**).  
 1363 Users with more command-line experience, or the intent of running TReX in batch-mode,  
 1364 can append necessary parameter values without adding them to a settings file.

1365 To acquire video-files that can be opened using TReX, one needs to first run TGrabs in  
 1366 one way or another. It is possible to use a webcam (generic USB camera) for recording, but  
 1367 TGrabs can also be compiled with Basler Pylon5 support<sup>d</sup>. TGrabs can also convert existing  
 1368 videos and write to a more suitable format for TReX to interact with (a static background  
 1369 with moving objects clearly separated in front of it). It can be started just like TReX, although

1365

1366

1367

1368

1369

1370

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.

1371

1372

1373

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

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

- `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.
- `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).

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

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 time-line, 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.

## Output

1398

1399

1400

1401

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:

1402

- **output** folder:

1403

- Settings files

1404

- Training weights

1405

- Saved program states

1406

- **data** folder:

1407

- \* Statistics

1408

- \* All exported NPZ files (named [video\_name]\_fish[number].npz – the prefix "fish" can be changed).

1409

- \* ...

1410

- **frames** folder (contains video clips recorded in the GUI, e.g. for presentations):

1411

- \* **[video name]** folder

1412

- clip[index].avi

1413

- ...

1414

- \* ...

1415

1416 At any point in time (except during training), the user can save the current program state  
 1417 and return to it at a later time (e.g. after a computer restart).

1418 **Export options**

1419 After individuals have been assigned by the matching algorithm, various metrics are calcu-  
 1420 lated (depending on settings):

1421 • **Angle:** The angle of an individual can be calculated without any context using image  
 1422 moments (*Hu (1962)*). However, this angle is only reliable within 0 to 180 degrees – not  
 1423 the full 360. Within these 180 degrees it is probably more accurate than is movement  
 1424 direction.

1425 • **Position:** Centroid information on the current, as well as the previous position of the  
 1426 individual are maintained. Based on previous positions, velocity as well as acceleration  
 1427 are calculated. This process is based on information sourced from the respective video  
 1428 file or camera on the time passed between frames. The centroid of an individual is  
 1429 calculated based on the mass center of the pixels that the object comprises. Angles  
 1430 calculated in the previous steps are corrected (flipped by 180 degrees) if the angle  
 1431 difference between movement direction and angle + 180 degrees is smaller than with  
 1432 the raw angle.

1433 • **Posture:** A large part of the computational complexity comes from calculating the  
 1434 posture of individuals. While this process is relatively fast in *TRex*, it is still the main  
 1435 factor (except with many individuals, where the matching process takes longest). We  
 1436 dedicated a subsection to it below.

1437 • **Visual Field:** Based on posture, rays can be cast to detect which animal is visible from  
 1438 the position of another individual. We also dedicated a subsection to visual field fur-  
 1439 ther down.

1440 • **Other** features can be computed, such as inter-individual distances or distance to  
 1441 the tank border. These are optional and will only be computed if necessary when  
 1442 exporting the data. A (non-comprehensive) list of metrics that can be exported follows:

- 1443 – Time: The time of the current frame (relative to the start of the video) in seconds.
- 1444 – Frame: Index of the frame in the PV video file.
- 1445 – Individual components of position its derivatives (as well as their magnitudes, e.g.  
   1446 speed)
- 1447 – Midline offset: The center-line, e.g. of a beating fish-tail, is normalized to be  
   1448 roughly parallel to the x-axis (from its head to a user-defined percentage of a  
   1449 body). The y-offset of its last point is exported as a "midline offset". This is useful,  
   1450 e.g. to detect burst-and-glide events.
- 1451 – Midline variance: Variance in midline offset, e.g. for detection of irregular pos-  
   1452 tures or increased activity.
- 1453 – Border distance
- 1454 – Average neighbour distance: Could be used to detect individuals who prefer to  
   1455 be located far away from the others or are avoided by them.

1456 Additionally, tracks of individuals can be exported as a series of cropped-out images – a  
 1457 very useful tool if they are to be used with an external posture estimator or tag-recognition.  
 1458 This series of images can be either every single image, or the median of multiple images  
 1459 (the time-series is down-sampled).

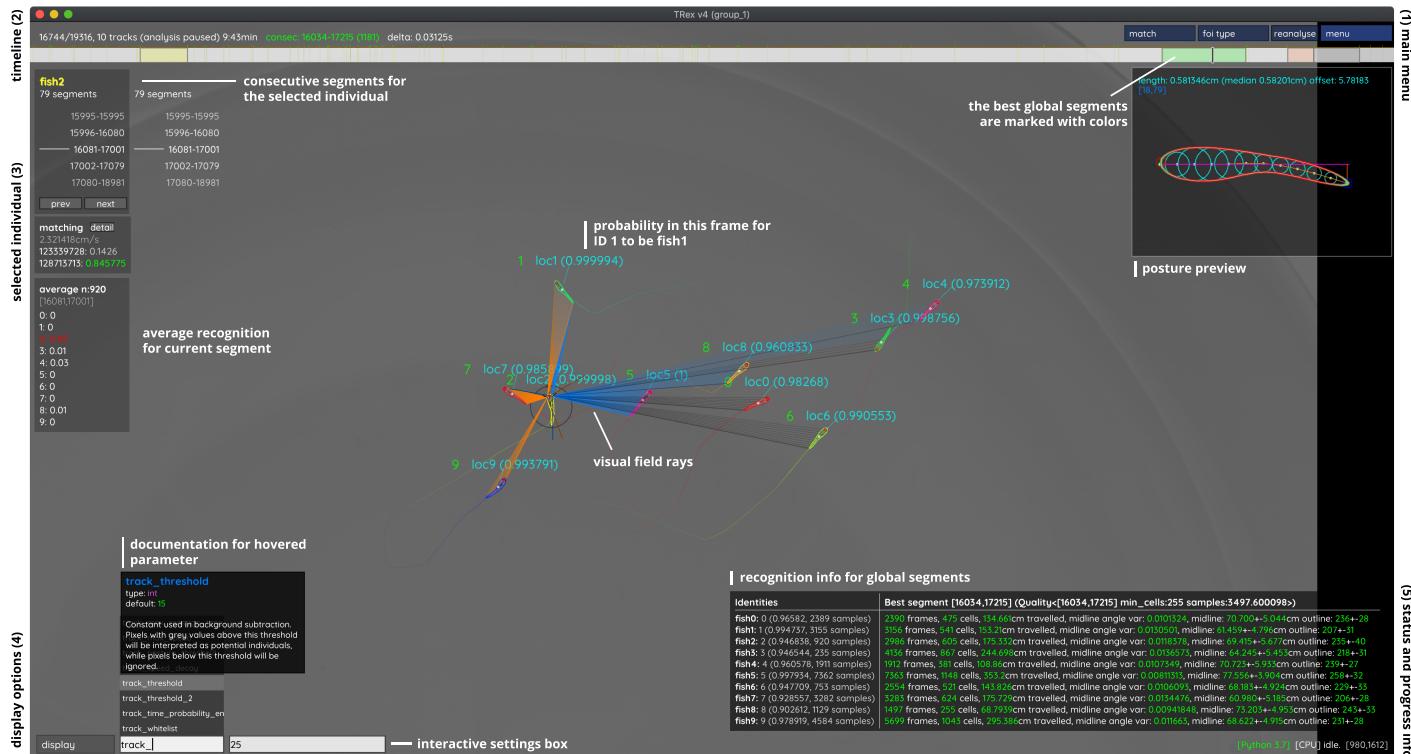
---

<sup>a</sup>[opencv.org](http://opencv.org)

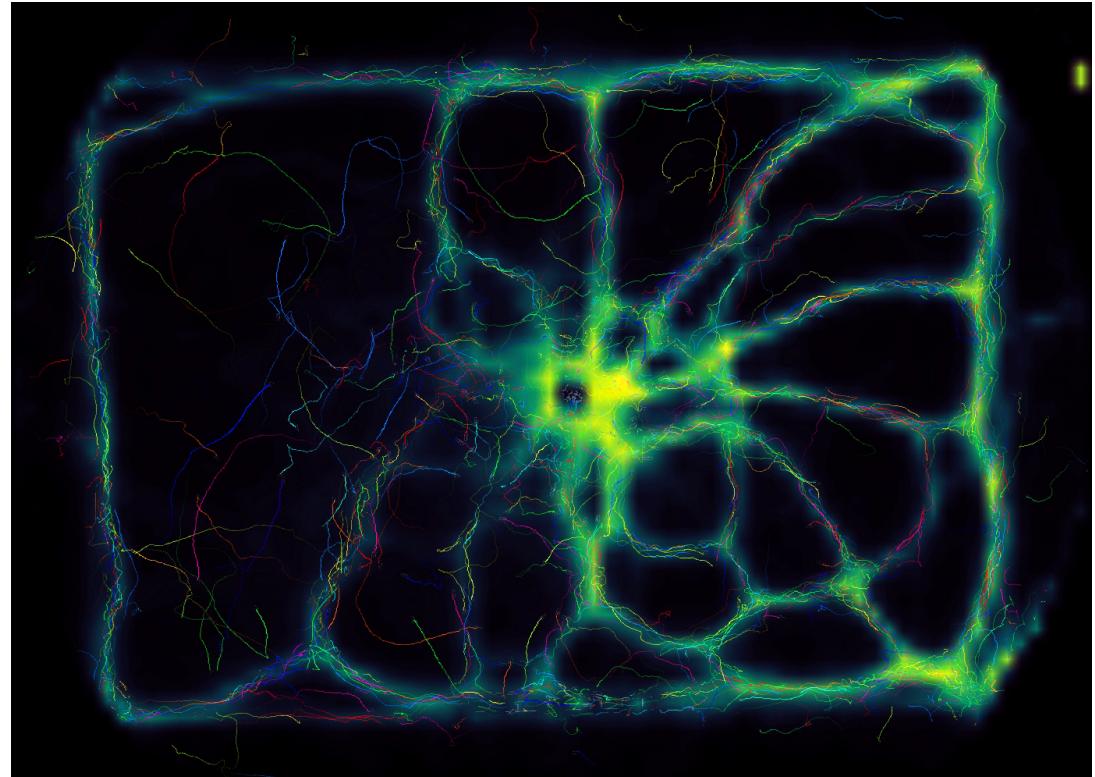
<sup>b</sup>[tensorflow.org](http://tensorflow.org)

<sup>c</sup>[deeplearning.net](http://deeplearning.net)

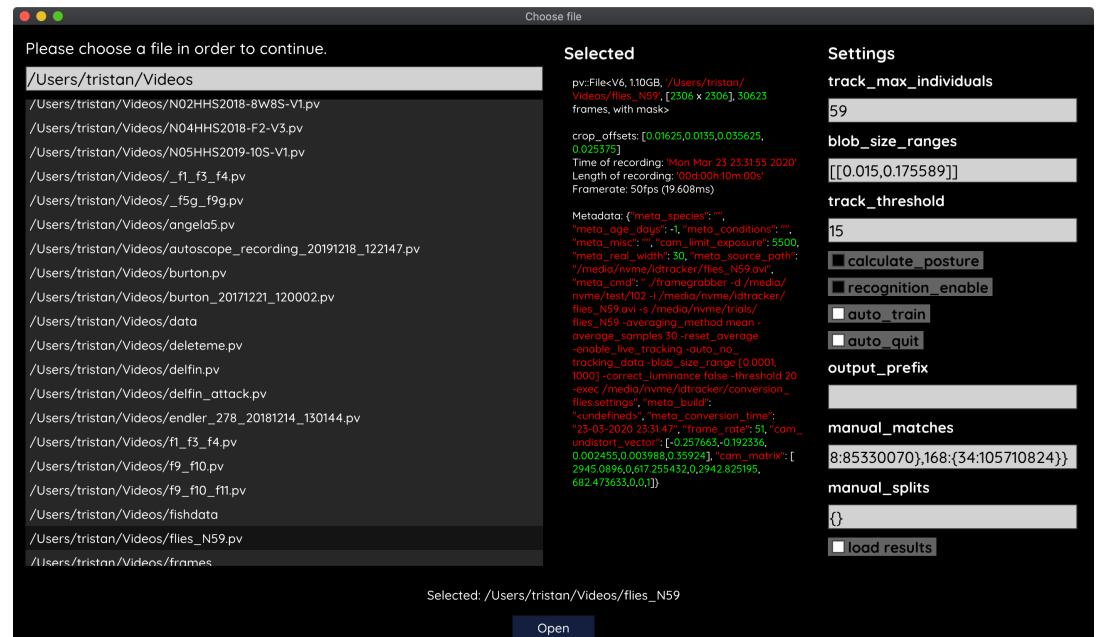
<sup>d</sup>The [baslerweb.com](http://baslerweb.com) Pylon SDK is required to be installed to support Basler USB cameras.



**Appendix 1 Figure A1.** An overview of TRex' the main interface, which is part of the documentation at [trex.run/docs](http://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).



**Appendix 1 Figure A2.** Using the interactive heatmap generator within TRex, the foraging trail formation of *Constrictotermes cyphergaster* (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



**Appendix 1 Figure A3.** 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.

## 1460 Appendix 2

1461 **From video frame to blobs**

1462 Video frames can originate either from a camera, or from a pre-recorded video file saved  
 1463 on disk. `TGrabs` treats both sources equally, the only exception being some minor details  
 1464 and that pre-recorded videos have a well-defined end (which only has an impact on MP4  
 1465 encoding). Multiple formats are supported, but the full list of supported codecs depends  
 1466 on the specific system and OpenCV version installed. `TGrabs` saves images in RAW quality,  
 1467 but does not store complete images. Merely the objects of interest, defined by common  
 1468 tracking parameters such as size, will actually be written to a file. Since `TGrabs` is mostly  
 1469 meant for use with stable backgrounds (except when contrast is good or a video-mask is  
 1470 provided), the rest of the area can be approximated by a static background image generated  
 1471 in the beginning of the process (or previously).

1472 Generally, every image goes through a number of steps before it can be tracked in `TRex`:

1473 1. Images are decoded by either (i) a camera driver, or (ii) OpenCV. They consist of an  
 1474 array of values between 0 and 255 (grayscale). Color images will be converted to  
 1475 grayscale images (color channel or "hue" can be chosen).

1476 2. Timing information is saved and images are appended to a queue of images to be  
 1477 processed

1478 3. All operations from now on are performed on the GPU if available. Once images are  
 1479 in the queue, they are picked one-by-one by the processing thread, which performs  
 1480 operations on them based on user-defined parameters:

1481 • Cropping

1482 • Inverting

1483 • Contrast/brightness and lighting corrections

1484 • Undistortion (see OpenCV Tutorial)

1485 4. (optional) Background subtraction ( $d(x) = b(x) - f(x)$ , with  $f$  being the image and  $b$  the  
 1486 background image), leaving a difference image containing only the objects. This can  
 1487 be an absolute difference  $|b(x) - f(x)|$  or a signed one, which has different effects on  
 1488 the following step. Otherwise  $d(x) = f(x)$

1489 5. Thresholding to obtain a binary image, with all pixels either being 1 or 0:

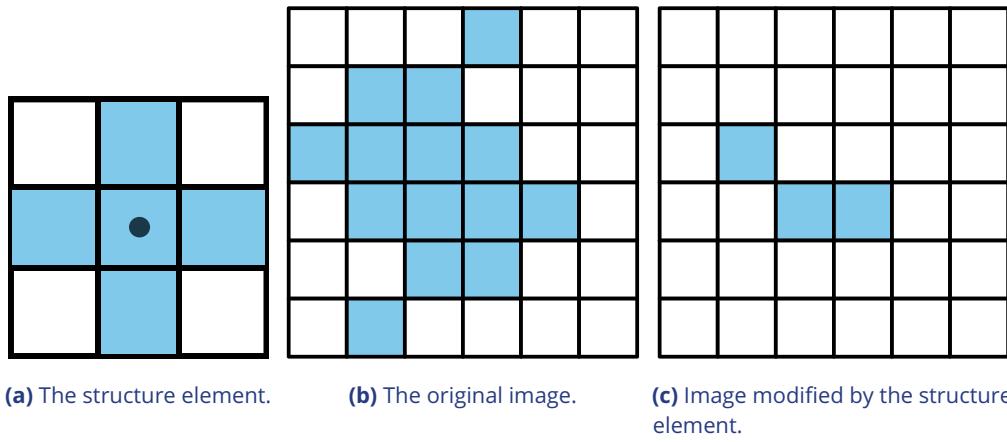
$$t(x) = \begin{cases} 0 & d(x) < T \\ 1 & d(x) \geq T \end{cases}$$

1490 where  $0 \leq T \leq 255$  is the threshold constant.

1491 6. Options are available for further adjustment of the binary image: Dilation, Erosion and  
 1492 Closing are used to close gaps in the shapes, which are filled up by successive dilation  
 1493 and erosion operations (see [Appendix 2 Figure A1](#)). If there is an imbalance of dilation  
 1494 and erosion commands, noise can be removed or shapes made more inclusive.

1495 7. The original image is multiplied by the thresholded image, obtaining a masked grayscale  
 1496 image:  $t(x) \cdot f(x)$ , where  $\cdot$  is the element-wise multiplication operator.

1497 At this point, the masked image is returned to the CPU, where connected components  
 1498 (objects) are detected. A connected component is a number of adjacent pixels with color  
 1499 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 ([AbuBaker et al. \(2007\)](#), [Chang and Chen \(2003\)](#),



**Appendix 2 Figure A1.** 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.

1503

1504

1505 and many others), even capable of real-time speeds (**Suzuki et al. (2003), He et al. (2009)**).  
 1506 However, since we want to use a compressed representation throughout our solution, as  
 1507 well as transfer over valuable information to integrate it with posture analysis, we needed  
 1508 to implement our own (see Connected components algorithm).

1509 MP4 encoding has some special properties, since its speed is mainly determined by the  
 1510 external encoding software. Encoding at high-speed frame-rates can be challenging, since  
 1511 we are also encoding to a PV-file simultaneously. Videos are encoded in a separate thread,  
 1512 without muxing, and will be remuxed after the recording is stopped. For very high frame-  
 1513 rates or resolutions, it may be necessary to limit the duration of videos since all of the images  
 1514 have to be kept in RAM until they have been encoded. RAW images in RAM can take up a lot  
 1515 of space ( $1024 * 1024 * 1000 = 1,048,576,000$  bytes for 1000 images quite low in resolution).  
 1516 If there a recording length is defined prior to starting the program, or a video is converted to  
 1517 PV and streamed to MP4 at the same time (though it is unclear why that would be necessary),  
 1518 TGrabs is able to automatically determine which frame-rate can be maintained reliably and  
 1519 without filling the memory.

## 1520 Appendix 3

1521 **Connected components algorithm**

1522 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  
 1523 saved ( $y, x_0$  and  $x_1$ ). This representation is especially suited for objects stretching out along  
 1524 the x-axis, but of course its worst-case is a straight, vertical line – in which case space  
 1525 requirements are  $O(2 * N)$  for  $N$  pixels. Especially for big objects, however, only a fraction  
 1526 of coordinates has to be kept in memory (with a space requirement of  $O(2 * H)$  instead of  
 1527  $O(W * H)$ , with  $W, H$  being width and height of the object).

1528 Extracting these connected horizontal line segments from an image can be parallelized  
 1529 easily by cutting the image into full-width pieces and running the following algorithm re-  
 1530 peatedly for each row:

- 1531 1. From 0 to  $W$ , iterate all pixels. Always maintain the previous value (binary), as well  
 1532 as the current value. We start out with our previous value of  $\bar{p} = 0$  (the border is  
 1533 considered not to be an object).
- 1534 2. Now repeat for every pixel  $p_i$  in the current row:
  - 1535 (a) If  $\bar{p}$  is 1 and  $p_i$  is 0, set  $\bar{p} := 0$  and save the position as the end of a line segment  
 1536  $x_1 = i - 1$ .
  - 1537 (b) If  $\bar{p}$  is 0 and  $p_i$  is 1, we did not have a previous line segment and a new one starts.  
 1538 We save it as our current line segment with  $x_0$  and  $y$  equal to the current row. Set  
 1539  $\bar{p} := 1$ .
- 1540 3. After each row, if we have a valid current line, we save it in our array of lines. If  $\bar{p} = 1$   
 1541 was set, and the line segment ended at the border  $W$  of the image, we first set its end  
 1542 position to  $x_1 := W - 1$ .

1543 We keep the array of extracted lines sorted by their y-coordinate, as well as their x-  
 1544 coordinates in the order we encountered them. To extract connected components, we now  
 1545 just need to walk through all extracted rows and detect changes in the y-coordinate. The  
 1546 only information needed are the current row and the previous row, as well as a list of active  
 1547 preliminary "blobs" (or connected components). A blob is simply a collection of ordered  
 1548 horizontal line segments belonging to a single connected component. These blobs are pre-  
 1549 liminary until the whole image has been processed, since they might be merged into a single  
 1550 blob further down despite currently being separate (see *Appendix 3 Figure A1*).

1551 "Rows" are an array of horizontal lines with the same y-coordinate, ordered by their x-  
 1552 coordinates (increasing). The following algorithm only considers pairs of previous row  $R_{i-1}$   
 1553 and current row  $R_i$ . We start by inserting all separate horizontal line segments of the very  
 1554 first row into the pool of active blobs, each assigned their own blob. Lines within row  $R_i$   
 1555 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  
 1556 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  
 1557 of rows, three different actions may be required depending on the case at hand. All three  
 1558 actions are hierarchically ordered and mutually exclusive (like a typical `if/else` structure  
 1559 would be), meaning that case 0-2 can be true at the same time while no other combination  
 1560 can be simultaneously true:

- 1561 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 ( $x_1(i, k) + 1 < x_0(i, j)$ ), or (1) y-coordinates of  $R_i$  and

1562

1563

$R_{i-1}$  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 ( $j \geq |R_{i-1}|$ ).  $L_{i,k}$  is assigned with a new blob.

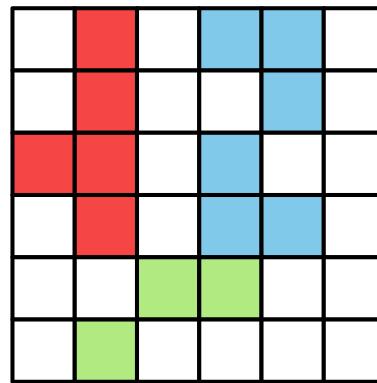
2. **Case 3: Segment in the previous row ends before the segment in the current row starts.** If  $x_0(i, k) > x_1(i-1, j) + 1$ , then we just have to  $j := j + 1$ .

3. **Case 4: Segment in the previous row and segment in the current row intersect in x-coordinates.** If  $L_{i,k}$  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.



**Appendix 3 Figure A1.** 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.

## 1586 Appendix 4

1587 **Matching an object to an object in the next frame**1588 **Terminology**

1589 A graph is a mathematical structure commonly used in many fields of research, such as  
 1590 computer science, biology and linguistics. Graphs are made up of vertices, which in turn  
 1591 are connected by edges. Below we define relevant terms that we are going to use in the  
 1592 following section:

- 1593 • Directed graph: Edges have a direction assigned to them
- 1594 • Weighted edges: Edges have a weight (or cost) assigned to them
- 1595 • Adjacent nodes: Nodes which are connected immediately by an edge
- 1596 • Path: A path is a sequence of edges, where each edges starting vertex is the end vertex  
     of the previous edge
- 1597 • Acyclic graph: The graph contains no path in which the same vertex appears more  
     than once
- 1598 • Connected graph: There are no vertices without edges, there is a path from any vertex  
     to any other vertex in the graph
- 1599 • 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
- 1600 • Tree: A tree is a connected, undirected, acyclic graph, in which any two vertices are  
     only connected by exactly one path
- 1601 • 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
- 1602 • Visited vertex: A vertex that is already part of the current path
- 1603 • Leaf: A vertex which has only one edge arriving, but none going out (in a tree this are  
     the bottom-most vertices)
- 1604 • Depth-first/breadth-first and best-first search: Different strategies to pick the next ver-  
     tex 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 ver-  
     tex. Breadth-first is the opposite of depth-search. Best-first search uses strategies to  
     explore the most promising path first.

1616 **Background**

1617 The transportation problem is one of the fundamental problems in computer science. It  
 1618 solves the problem of transporting a finite number of *goods* to a finite number of *factories*,  
 1619 where each possible transport route is associated with a *cost* (or weight). Every factory has  
 1620 a *demand* for goods and every good has a limited *supply*. The sum of this cost has to be  
 1621 minimized (or benefits maximized), while remaining within the constraints given by supply  
 1622 and demand. In the special case where demand by each factory and supply for each good  
 1623 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 ([Ramshaw and Tarjan \(2012\)](#)). Thankfully, unbalanced assignments can be reduced to balanced assignments, for example using graph-duplication methods or by adding nodes ([Ramshaw and Tarjan \(2012\)](#), [Ramshaw and Tarjan \(2012\)](#)). This makes the widely used Hungarian method ([Kuhn \(1955\)](#); [Munkres \(1957\)](#)) a viable solution to both,

1627

1628

1629

1630

1631

1632

1633

1634

1635

1636

1637

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 (**Fredman and Tarjan (1987)**), 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  $s \ll n$  (**Ramshaw and Tarjan (2012)**).

1638

1639

1640

1641

1642

1643

1644

1645

1646

1647

### 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 of 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, **Land and Doig (2010)**, formalized in **Little et al. (1963)**) 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 (**Clausen (1999)**). 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 (**Weixiong (1996)**, "Graph vs. tree"). So even if the bipartite assignment graph (an example sketched in **Appendix 4 Figure A1**) 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 **Appendix 4 Figure A1**, 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 **Appendix 4 Figure A1** (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

1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730

of pairings between objects (plus `null`) and individuals. Edge weights in  $E$  are equal to the probability  $P_i(t, \tau_i \mid B_j)$  (see equation 7), 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

$$o(\rho) = \sum_{uv \in \rho} P_u(v),$$

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

$$\overline{P}_u = \max_{v \in V} \{P_u(v)\}.$$

Nodes are sorted first by their degree (ascending) and secondly by  $\overline{P}_u$  (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* (**Weixiong (1996)**). Sorting by  $\overline{P}_u$  is only applied whenever nodes of the same degree have to be considered.

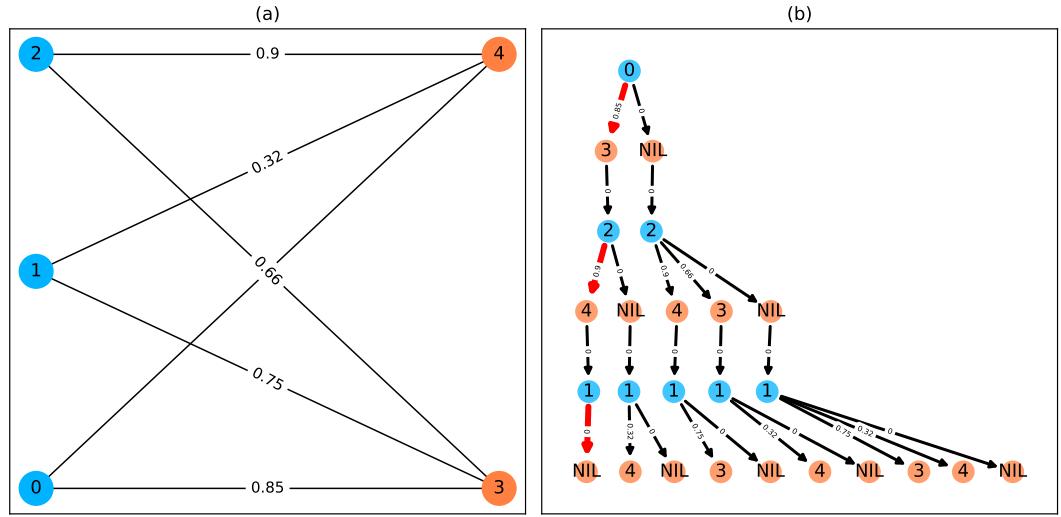
We always follow the most promising paths first (the one with the highest accumulated probability), which is called "best-first search" (BFS) – our selection strategy for (1.) in 4. BFS is implemented using a queue maintaining the list of all currently expanded nodes.

Regarding (2.) in 4, we utilize  $\overline{P}_u$  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  $\text{upper\_limit}(i) = \sum_{j > i \in U} \overline{P}_j$ , with  $j, i$  being indices into our ordered set  $S$  of individuals and  $i$  being 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 previous 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.

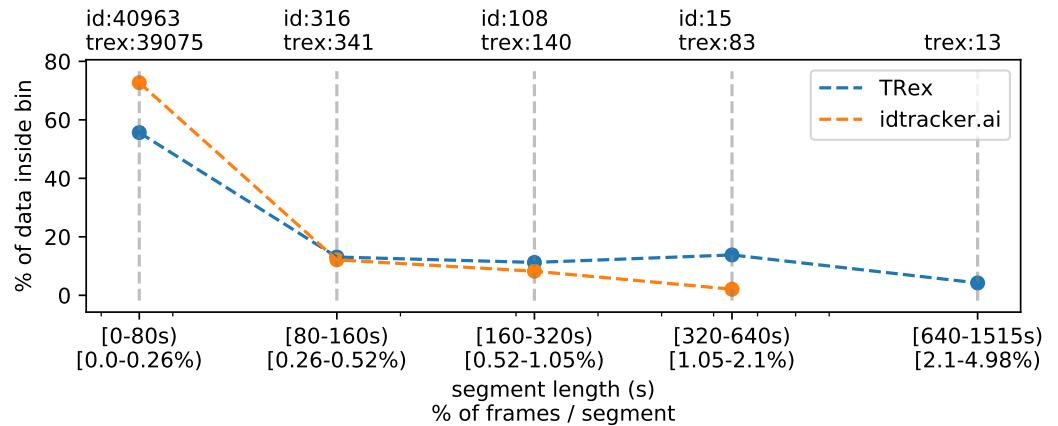
As an example, let us traverse the tree in **Appendix 4 Figure A1b**:

- 1731 • We first calculate  $\overline{P}_u$  for every  $u \in U$  ( $\overline{P}_0 = 0.85$ ;  $\overline{P}_2 = 0.9$ ;  $\overline{P}_1 = 0.75$ ), as well as the  
 1732 hierarchical probability table `upper_limit(i)` for each index  $0 \leq i < N$  ( $0.9 + 0.75$ ;  $0.75$ ;  $0$ ).  
 1733  $P_{\text{best}} := 0$ .  
 1734 • Individual 0 (the root) is expanded, which has one edge with probability  $0.85 + \text{upper\_limit}(0) \geq$   
 1735  $P_{\text{best}}$  to object 3 (plus the `null` case) and is the only node with a degree of 1. We know  
 1736 that our now expanded node is the best, since it has the largest probability due to  
 1737 sorting, plus also is the deepest. In fact, this is true for all expanded nodes exactly in  
 1738 the order they are expanded (depth-first search == best-first search for our case). We  
 1739 set  $P_{\text{best}} := 0.85$ . The edge to `NIL` is added to our queue.  
 1740 • Objects in  $V$  are only virtual and always have zero-probability connections to the next  
 1741 individual in an ordered set ( $f \in F$ ), so they do not add to the overall probability sum.  
 1742 We skip to the next node.  
 1743 • Individual 2 branches off into one or two different edges, depending on which edges  
 1744 have been chosen previously.  
 1745 • We first explore the edge towards object 4 with a probability of  $0.9 + \text{upper\_limit}(1) =$   
 1746  $1.65 \geq P_{\text{best}}$  and add it to  $P_{\text{best}}$ .  
 1747 • Only one possibility is left and we arrive at a leaf with an accumulated probability of  
 1748  $0.85 + 0.9 + 0 = 1.75$ .  
 1749 • We now perform backtracking, meaning we look at every expanded node in our queue,  
 1750 each time observing  $\overline{P}_u + \text{upper\_limit}(i)$ .
- 1751   – `NIL` (from node 2) would be added to the front of our queue, however its proba-  
 1752 bility  $0.85 + 0 + \text{upper\_limit}(1) = 1.6 < 1.75 = P_{\text{best}}$ , so it is discarded.  
 1753   – `NIL` (from node 0) would be added now, but its probability of  $0 + \text{upper\_limit}(0) =$   
 1754  $1.65 < P_{\text{best}}$ , so it is also discarded.
- 1755 We can see that with increased depth, we have to keep track of more and more pos-  
 1756 sibilities. Since our nodes and edges are pre-sorted, our path through the tree is optimal  
 1757 after exactly  $N = |U|$  node expansions (not counting  $v \in V$  expansions since they are only  
 1758 "virtual").
- 1759 **Complexity**
- 1760 Utilizing these techniques, we can achieve very good average-case complexity. Of course  
 1761 having a good worst-case complexity is important (such as the Hungarian method), but the  
 1762 impact of a good average-case complexity can be significant as well. This is illustrated nicely  
 1763 by the timings measured in Table [Appendix 4 Table A3](#), where our method consistently  
 1764 surpasses the Hungarian method in terms of performance – especially for very large groups  
 1765 of animals – despite having worse worst-case complexity. Usually, even in situations with  
 1766 over 1000 individuals present, the average number of leaves visited was approximately 1.112  
 1767 (see Table [Appendix 4 Table A5](#)) and each visit was a global improvement (not shown). The  
 1768 number of nodes visited per frame were around 2844 to 19,804,880 in the same video, which,  
 1769 given the maximal number of possible combinations  $N^M$  for  $M$  edges and  $N$  individuals  
 1770 ([Thomas \(2016\)](#)), is quite moderate. Especially considering the number of calculations that  
 1771 the Hungarian method has to perform in every step, which, according to its complexity, will  
 1772 be in the range of  $N^3 \approx 1e9$  for  $N = 1024$  individuals.
- 1773 The average complexity of a solution using best-first-search BnB is given by [Weixiong](#)  
 1774 ([1996](#)). It depends on the probability of encountering a "zero-cost edge"  $p_0$ , as well as the  
 1775 mean branching factor  $b$  of the tree:
- 1776 1.  $\Theta(\beta^N)$  when  $bp_0 < 1$ , with  $\beta \leq b$  and  $N$  is the depth of the tree  
 1777 2.  $\Theta(N^2)$  when  $bp_0 = 1$

1778	3. $\Theta(N)$ when $bp_0 > 1 \Leftrightarrow b > 1/p_0$
1779	as $N \rightarrow \infty$ .
1780	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.
1781	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, <b>Bertsekas (1981)</b> ), giving a possible explanation for the good results achieved using tree-based matching in TReX on average (Table <b>Appendix 4 Table A3</b> ).
1782	
1783	
1784	
1785	
1786	
1787	
1788	
1789	
1790	
1791	
1792	
1793	
1794	
1795	
1796	
1797	
1798	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" ( <b>Pesant et al. (2012)</b> ). 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.
1799	
1800	
1801	
1802	



**Appendix 4 Figure A1.** 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.



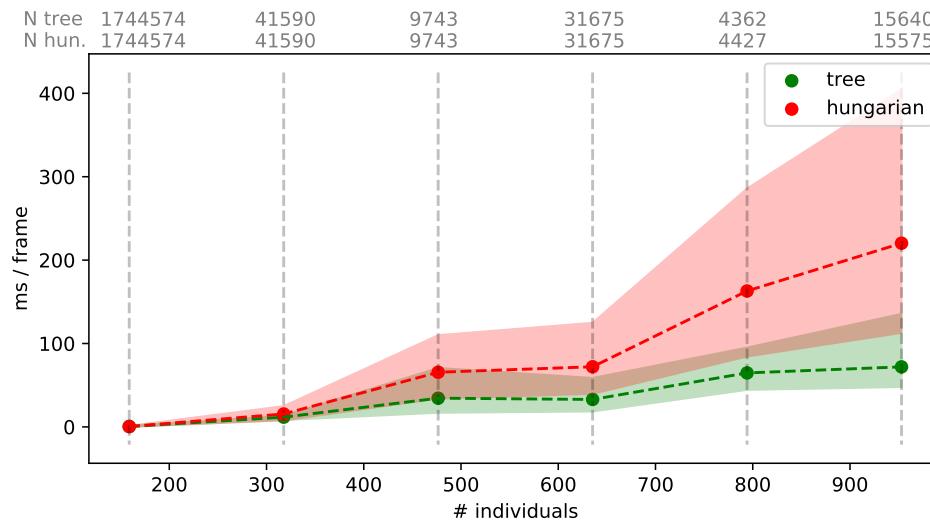
**Appendix 4 Figure A2.** The same set of videos as in **Table 5** pooled together, we evaluate the efficiency of our crossings solver. *Consecutive frame segments* are sequences of frames without gaps, for example due to crossings or visibility issues. We find these *consecutive frame segments* in data 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  $\sum_{k \in [1, V]} \text{video\_length}_k * \# \text{individuals}_k$  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.

**Appendix 4 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  $\leq 256$  individuals can be processed faster than they could be recorded. Videos that can not be processed faster than real-time are underlaid in gray.

video characteristics			ms / frame (processing)					processing time
video	# ind.	ms / frame	5%	mean	95%	max	> real-time	% video length
<b>0</b>	1024	25.0	46.93	62.96	119.54	849.16	100.0%	358.12
<b>1</b>	512	20.0	19.09	29.26	88.57	913.52	92.11%	259.92
<b>2</b>	512	16.67	17.51	26.53	36.72	442.12	97.26%	235.39
<b>3</b>	256	20.0	8.35	11.28	13.25	402.54	1.03%	77.18
<b>4</b>	256	16.67	8.04	11.62	13.48	394.75	1.13%	94.77
<b>5</b>	128	16.67	3.54	5.14	5.97	367.92	0.41%	40.1
<b>6</b>	128	16.67	3.91	5.64	6.89	381.51	0.51%	44.38
<b>7</b>	100	31.25	2.5	3.57	5.19	316.75	0.1%	28.35
<b>8</b>	59	19.61	1.43	2.29	3.93	2108.77	0.19%	16.33
<b>9</b>	15	40.0	0.4	0.52	1.67	4688.5	0.01%	2.96
<b>10</b>	10	10.0	0.28	0.33	0.57	283.7	0.07%	8.08
<b>11</b>	10	31.25	0.21	0.25	0.65	233.7	0.01%	3.48
<b>12</b>	10	31.25	0.23	0.27	0.75	225.63	0.02%	2.82
<b>13</b>	10	31.25	0.22	0.25	0.54	237.32	0.02%	2.64
<b>14</b>	8	33.33	0.24	0.29	0.66	172.8	0.02%	1.8
<b>15</b>	8	40.0	0.22	0.26	0.88	244.88	0.01%	1.5
<b>16</b>	8	28.57	0.18	0.21	0.51	1667.14	0.02%	1.38
<b>17</b>	1	7.14	0.03	0.04	0.06	220.81	0.01%	1.56

**Appendix 4 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 reassessments 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 (**Kuhn (1955)**) 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.

video	# ind.	length		total	excluded	wrong	
7	100	1min		717	755	22	22 45 (6.47%) 65 (8.87%)
8	59	10min		279	312	146	100 55 (41.35%) 32 (15.09%)
9	15	1h0min		838	972	70	111 100 (13.02%) 240 (27.87%)
13	10	10min3s		331	337	22	22 36 (11.65%) 54 (17.14%)
12	10	10min3s		382	404	42	43 83 (24.41%) 130 (36.01%)
11	10	10min10s		1067	1085	50	52 73 (7.18%) 92 (8.91%)
14	8	3h15min22s		7424	7644	1428	1481 1174 (19.58%) 1481 (24.03%)
15	8	1h12min		3538	3714	427	517 651 (20.93%) 962 (30.09%)
16	8	3h18min13s		2376	3305	136	206 594 (26.52%) 1318 (42.53%)
			sum	16952	16754	-2343	-2554 2811 (19.24%) 4374 (27.38%)



**Appendix 4 Figure A3.** Mean values of processing-times and 5%/95% percentiles for video frames of all videos in the *speed dataset* (Table **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 **Appendix 4 Table A3**).

**Appendix 4 Table A3.** Comparing computation speeds of the tree-based tracking algorithm with the widely established Hungarian algorithm *Kuhn (1955)*, 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 [Appendix 4 Table A1](#)), with (1) and (2) even getting close for 512 individuals.

video	# ind.	video metrics		% real-time		
		fps (Hz)	size (px <sup>2</sup> )	tree	approximate	hungarian
<b>0</b>	1024	40	3866 × 4048	35.49 ± 65.94	38.69 ± 65.39	12.05 ± 18.72
<b>1</b>	512	50	3866 × 4140	51.18 ± 180.08	75.02 ± 193.0	28.92 ± 29.12
<b>2</b>	512	60	3866 × 4048	59.66 ± 121.4	65.58 ± 175.51	23.18 ± 26.83
<b>3</b>	256	50	3866 × 4140	174.02 ± 793.12	190.62 ± 743.54	127.86 ± 9841.21
<b>4</b>	256	60	3866 × 4048	140.73 ± 988.15	155.9 ± 760.05	108.48 ± 2501.06
<b>5</b>	128	60	3866 × 4048	318.6 ± 347.8	353.58 ± 291.63	312.05 ± 337.71
<b>6</b>	128	60	3866 × 4048	286.13 ± 330.08	314.91 ± 303.53	232.33 ± 395.21
<b>7</b>	100	32	3584 × 3500	572.46 ± 98.21	611.5 ± 96.46	637.87 ± 97.03
<b>8</b>	59	51	2306 × 2306	744.98 ± 264.43	839.45 ± 257.56	864.01 ± 223.47
<b>9</b>	15	25	1880 × 1881	4626.84 ± 424.8	4585.08 ± 378.64	4508.08 ± 404.56
<b>10</b>	10	100	1920 × 1080	2370.35 ± 303.94	2408.27 ± 297.83	2362.42 ± 296.99
<b>11</b>	10	32	3712 × 3712	6489.12 ± 322.59	6571.28 ± 306.34	6472.0 ± 322.03
<b>12</b>	10	32	3712 × 3712	6011.59 ± 318.12	6106.12 ± 305.96	5549.25 ± 318.21
<b>13</b>	10	32	3712 × 3712	6717.12 ± 325.37	6980.12 ± 316.59	6726.46 ± 316.87
<b>14</b>	8	30	3008 × 3008	8752.2 ± 2141.03	8814.63 ± 2101.4	8630.73 ± 2177.16
<b>15</b>	8	25	3008 × 3008	9786.68 ± 1438.08	10118.04 ± 1380.2	9593.44 ± 1439.28
<b>16</b>	8	35	3008 × 3008	9861.42 ± 1424.91	10268.82 ± 1339.8	9680.68 ± 1387.14
<b>17</b>	1	140	1312 × 1312	15323.05 ± 637.17	15250.39 ± 639.2	15680.93 ± 640.99

**Appendix 4 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  $(\text{prepare} + \text{tracking} - \text{live}) / (\text{prepare} + \text{tracking})$ . 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.

video metrics				minutes				
video	# ind.	length	fps (Hz)	prepare	tracking	live	win (%)	
<b>0</b>	1024	8.33min	40	$10.96 \pm 0.3$	$41.11 \pm 0.34$	$65.72 \pm 1.35$	-26.23	
<b>1</b>	512	6.67min	50	$11.09 \pm 0.24$	$24.43 \pm 0.2$	$33.67 \pm 0.58$	5.24	
<b>2</b>	512	5.98min	60	$11.72 \pm 0.2$	$20.86 \pm 0.47$	$31.1 \pm 0.62$	4.55	
<b>3</b>	256	6.67min	50	$11.09 \pm 0.21$	$7.99 \pm 0.17$	$12.35 \pm 0.17$	35.26	
<b>4</b>	256	5.98min	60	$11.76 \pm 0.26$	$9.04 \pm 0.26$	$15.08 \pm 0.13$	27.46	
<b>6</b>	128	5.98min	60	$11.77 \pm 0.29$	$4.74 \pm 0.13$	$12.13 \pm 0.32$	26.49	
<b>5</b>	128	6.0min	60	$11.74 \pm 0.26$	$4.54 \pm 0.1$	$12.08 \pm 0.25$	25.79	
<b>7</b>	100	1.0min	32	$1.92 \pm 0.02$	$0.47 \pm 0.01$	$2.03 \pm 0.02$	14.88	
<b>8</b>	59	10.0min	51	$6.11 \pm 0.07$	$7.68 \pm 0.12$	$9.28 \pm 0.08$	32.7	
<b>9</b>	15	60.0min	25	$12.59 \pm 0.18$	$5.32 \pm 0.07$	$13.17 \pm 0.12$	26.47	
<b>11</b>	10	10.17min	32	$8.58 \pm 0.04$	$0.74 \pm 0.01$	$8.8 \pm 0.12$	5.66	
<b>12</b>	10	10.05min	32	$8.68 \pm 0.04$	$0.75 \pm 0.01$	$8.65 \pm 0.07$	8.3	
<b>13</b>	10	10.05min	32	$8.67 \pm 0.03$	$0.71 \pm 0.01$	$8.65 \pm 0.07$	7.76	
<b>10</b>	10	10.08min	100	$4.17 \pm 0.06$	$2.02 \pm 0.02$	$4.43 \pm 0.05$	28.3	
<b>14</b>	8	195.37min	30	$110.51 \pm 2.32$	$8.99 \pm 0.22$	$109.97 \pm 2.05$	7.98	
<b>15</b>	8	72.0min	25	$31.84 \pm 0.53$	$3.26 \pm 0.07$	$32.1 \pm 0.42$	8.55	
<b>16</b>	8	198.22min	35	$133.45 \pm 2.22$	$11.38 \pm 0.28$	$133.1 \pm 2.28$	8.1	
						mean	14.55 %	

**Appendix 4 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.

video characteristics		matching stats		
video	# ind.	# nodes visited (5,50,95,100%)	# leafs visited	# improvements
<b>0</b>	1024	[1535; 2858; 83243; 18576918]	$1.113 \pm 0.37$	1.113
<b>1</b>	512	[1060; 8156; 999137; 19811558]	$1.247 \pm 0.61$	1.247
<b>2</b>	512	[989; 2209; 56061; 8692547]	$1.159 \pm 0.47$	1.159
<b>3</b>	256	[452; 479; 969; 205761]	$1.064 \pm 0.29$	1.064
<b>4</b>	256	[475; 496; 584; 608994]	$1.028 \pm 0.18$	1.028
<b>5</b>	128	[233; 245; 258; 7149]	$1.012 \pm 0.12$	1.012
<b>6</b>	128	[237; 259; 510; 681702]	$1.046 \pm 0.25$	1.046
<b>7</b>	100	[195; 199; 199; 13585]	$1.014 \pm 0.14$	1.014
<b>8</b>	59	[117; 117; 117; 16430]	$1.014 \pm 0.2$	1.014
<b>9</b>	15	[24; 29; 29; 635]	$1.027 \pm 0.22$	1.027
<b>10</b>	10	[17; 19; 19; 56]	$1.001 \pm 0.02$	1.001
<b>11</b>	10	[19; 19; 19; 129]	$1.006 \pm 0.1$	1.006
<b>12</b>	10	[19; 19; 19; 1060]	$1.023 \pm 0.23$	1.023
<b>13</b>	10	[19; 19; 19; 106]	$1.001 \pm 0.04$	1.001
<b>14</b>	8	[11; 15; 15; 893]	$1.003 \pm 0.08$	1.003
<b>15</b>	8	[13; 15; 15; 597]	$1.024 \pm 0.23$	1.024
<b>16</b>	8	[15; 15; 15; 2151]	$1.009 \pm 0.17$	1.009
<b>17</b>	1	[1; 1; 1; 1]	$1.0 \pm 0.02$	1.0

## 1803 Appendix 5

1804 **Posture**

1805 Estimating an animals orientation and body pose in space is a diverse topic, where angle  
 1806 and pose can mean many different things. We are not estimating the individual positions  
 1807 of many legs and antennae in TRex, we simply want to know where the front- and the back-  
 1808 end of the animal are. Ultimately, the goal here is to be able to align animals using an  
 1809 arbitrary axis with their head extending in one direction and their tail roughly in the opposite  
 1810 direction. In order to achieve this, we are required to follow a series of steps to acquire all  
 1811 the necessary information:

- 1812 1. Locate objects in the image
- 1813 2. Detect the edge of objects
- 1814 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).
- 1815 4. Calculate a center-line based on local curvature of the outline.
- 1816 5. Calculate head and tail positions.
- 1817

1818 The first point is a given at this point (see Connected components algorithm). We can  
 1819 utilize the format in which connected components are computed in TRex (an ordered array  
 1820 of horizontal line segments), which reduces redundancy by avoiding to look at every individ-  
 1821 ual pixel. These line segments also contain information about edges since every start and  
 1822 end has to be an edge-pixel, too.

1823 Even though we already have a list of edge-pixels, retrieving an *ordered* set of points is  
 1824 crucial and requires much more effort. Without information about a pixels connectivity, we  
 1825 can not differentiate between inner and outer shapes (holes vs. outlines) and we can not  
 1826 calculate local curvature.

1827 **Connecting pixels to form an outline**

1828 We implemented an algorithm based on horizontal line segments, which only ever retains  
 1829 three consecutive rows of pixels (*p* previous, *c* current and *n* next). These horizontal line  
 1830 segments always stem from a "blob" (or connected component). Rows contain (i) their y-  
 1831 value in pixels, (ii)  $x_0, x_1$  values describing the first and last "on"-pixel that has been found  
 1832 in it, (iii) a set of detected border pixels (identified by their x-coordinate). A row is valid,  
 1833 whenever the y coordinate is not -1 – all three rows are initialized to an invalid  $y = -1$ .  $l'$  is  
 1834 the previous row. Using *p, corn* as a function  $c(x)$  returns 1 for on-pixels at that x-coordinate,  
 1835 and 0 for off-pixels.

1836 For each line *l* in the sorted list of horizontal line segments, we detect border pixels:

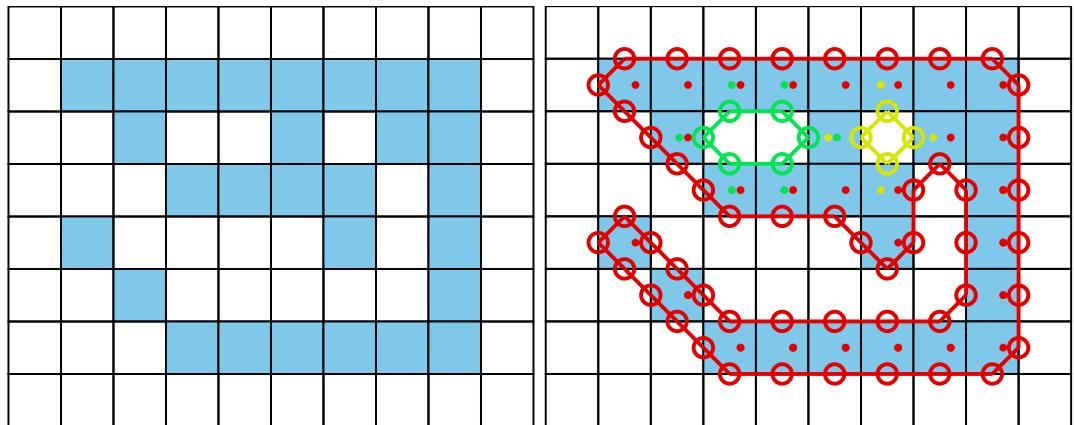
- 1837 1. subtract the blobs position (minimum of all  $l_{x0}$  and  $l_y$  separately) from *l*
- 1838 2. **if**  $n_y \neq l_y$ , a row has ended and a new one starts: call finalize
- 1839     **else if**  $l_{x0} - l'_{x1} \geq 1 \wedge l_{x0} \geq c_{x0}$ , we either skipped a few pixels in *n* or *l* starts before  
     *c* even had valid pixels. This means that all pixels *x* between  $\max\{l'_{x1} + 1; c_{x0}\} \leq x <$   
      $\min\{l_{x0}; c_{x1} + 1\}$  are border pixels in *c*.
- 1840 3. **if**  $l_{x1} < c_{x0}$ , or *c* is invalid, then line *l* ends before the previous row (*c*) even has any  
     "on"-pixels. All pixels *x* between  $l_{x0} \leq x \leq l_{x1}$  are border pixels in *n*.
- 1841     **else**
- 1842         (a)  $s := l_{x0}$
- 1843         (b) **if**  $s < c_{x0}$ , then lines are overlapping in *c* and *n* (line *l*). We can fill *n* up with border  
             while  $x < c_{x0}$  and  $x \leq l_{x1}$ . Set  $s := \min\{c_{x0} - 1; l_{x1}\}$ .
- 1844
- 1845

1846  
 1847      **else if**  $s = 0$  or  $s > 0 \wedge n(s - 1) = 0$ , then  $l$  starts at the image border (which is an automatic border pixel) or there is a gap before  $l$ . Set  $s := s + 1$ .  
 1848  
 1849  
 1850      (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)  $c(x) = 0$ .  
 1851  
 1852      4. Set  $n_{x1} := l_{x1}$ .  
 1853  
 1854      After iterating through all lines, we need two additional calls to **finalize** to populate the lines currently in  $c$  and  $n$  through.  
 1855  
 1856      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 neighbours 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.  
 1857  
 1858  
 1859  
 1860      1. **if**  $n$  is invalid, continue to the next operation.  
 1861      **else if**  $l_y > n_y + 1$ , then we skipped at least one row between  $n$  and the new row – making all on-pixels in  $n$  border pixels.  
 1862  
 1863      **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.  
 1864  
 1865      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).  
 1866  
 1867      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  $n_{x0} = l_{x0}, n_{x1} = l_{x1}, n_y = l_y$ .  
 1868  
 1869  
 1870      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 **finalize** is called, since the middle row ( $c$ ) is fully defined at that point (its entire neighbourhood has been cached).  
 1871  
 1872  
 1873  
 1874      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 half-way 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 **Appendix 5 Figure A1b**). The half-edge-points for each node are generated as follows:  
 1875  
 1876  
 1877  
 1878  
 1879  
 1880  
 1881  
 1882  
 1883      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.  
 1884  
 1885  
 1886  
 1887      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).  
 1888      **else** check the face an additional 45° to the left (e.g. LEFT).  
 1889  
 1890      (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).  
 1891  
 1892      (b) **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).  
 1893  
 1894

1895      Each time an edge is added, more and more of the half-edges are becoming fully-connected  
 1896      (meaning they have two of the allowed two edges). To generate the final result, all we have  
 1897      to do is to start somewhere in the graph and walk strictly in clockwise direction. "Walking"  
 1898      is done using a queue and edges are followed using depth-first search (see Matching an ob-  
 1899      ject to an object in the next frame): Each time a node is visited, all its yet unexplored edges  
 1900      are added to the front of the queue (in clockwise order). Already visited edges are marked  
 1901      (or pruned) and will not be traversed again – their floating-point positions (somewhere on  
 1902      an edge of its parent pixel) are added to an array.  
 1903      After a path ended, meaning that no more edges can be reached from our current node,  
 1904      the collected floating-point positions are pushed to another array and a different, yet unvis-  
 1905      ited, starting node is sought. This way, we can accumulate all available outlines in a given  
 1906      image one-by-one – including holes.  
 1907      These outlines will usually be further processed using an Elliptical Fourier Transform  
 1908      (or EFT, *Kuhl and Giardina (1982)*), as mentioned in the main-text. Outlines can also be  
 1909      smoothed using a weighted average of the  $N$  points around a given point, or resampled to  
 1910      either reduce or (virtually) increase resolution.

1911      **Finding the tail**  
 1912      Given an ordered outline, curvature can be calculated locally (per index  $i$ ):  
 1913  
 1914  
 1915       $C(i) = 4 * \text{triangle\_area}(p_{i-r}, p_i, p_{i+r}) / (\|p_i - p_{i-r}\| * \|p_i - p_{i+r}\| * \|p_{i-r} - p_{i+r}\|)$   
 1916      where  $1 \leq r \in \mathbb{N}$  is a parameter, which effectively leads to more smoothing when in-  
 1917      creased. Triangle area can be calculated as follows:  
 1918  
 1919  
 1920  
 1921       $\text{triangle\_area}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = (\mathbf{b}_x - \mathbf{a}_x)(\mathbf{c}_y - \mathbf{a}_y) - (\mathbf{b}_y - \mathbf{a}_y)(\mathbf{c}_x - \mathbf{a}_x).$   
 1922  
 1923      To find the "tail", or the pointy end of the shape, we employ a method closely related to  
 1924      scipys `find_peaks` function: We find local maxima using discrete curve differentiation and  
 1925      then generate a hierarchy of these extrema. The only major difference to normal differen-  
 1926      tiation is that we assume periodicity to achieve our results – values wrap around in both  
 1927      directions, since we are dealing with an outline here. We then find the peak with the largest  
 1928      integral, meaning we detect both very wide and very high peaks (just not very slim ones).  
 1929      The center of this peak is the "tail".  
 1930      To find the head as well, we now have to search for the peak that has the largest (index-  
 1931      ) distance to the tail-peak. This is a periodic distance, too, meaning that  $N$  is one of the  
 1932      closest neighbours of 0.  
 1933      The entire outline array is then rotated, so that the head is always the first point in it.  
 1934      Both indexes are saved.

1935      **Calculating the center-line**  
 1936      A center-line, for a given outline, can be calculated by starting out at the head and walking  
 1937      in both directions from there – always trying to find a pair of points with minimal distance  
 1938      to each other on both sides. Two indices are used:  $l, r$  for left and right. We also allow  
 1939      some "wiggle-room" for the algorithm to find the best-matching points on each side. This  
 1940      is limited by a maximum offset of  $\omega$  points which is set to  $0.025 * N$  by default, where  $N$  is  
 1941      the number of points in the outline.  $\mathbf{f}(i)$  gives the point on in outline at position  $i$ .  
 1942      Starting from  $l := -1, r := 1$  we continue while  $r < l + N$ :  
 1943      1. Find  $m := \arg\min_i \{\|\mathbf{f}(r+i) - \mathbf{f}(l)\| ; \forall i \leq \omega \wedge r+i < N\}$ . If no valid  $m$  can be found, abort.  
 1944      Otherwise set  $r := m$ .



**Appendix 5 Figure A1.** 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 OpenCVs `findContours` function (for reference) coded in the same colors as ours. Contours calculated by OpenCVs algorithm can not 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.

- ```

1945 2. Find  $k := \operatorname{argmin}_i \{ \|\mathbf{f}(l - i + N) - \mathbf{f}(r)\| ; \forall i \leq \omega \wedge l - i \leq -N \}$ . If no valid  $k$  can be found,
1946   abort. Otherwise set  $l := k$ .
1947 3. Our segment now consists of points  $\mathbf{f}(m)$  and  $\mathbf{f}(k)$ , with a center vector of  $(\mathbf{f}(k) - \mathbf{f}(m)) *$ 
1948     $0.5 + \mathbf{f}(m)$ . Push it to the center-line array. We can also calculate the width of the body
1949    at that point using  $\|\mathbf{f}(k) - \mathbf{f}(m)\|$ .
1950 4. Set  $l := l - 1$ .
1951 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 slimmest 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.

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.

## 1960 Appendix 6

### 1961 Visual field estimation

1962 Visual fields are calculated by casting rays and intersecting them with other individuals and  
 1963 the focal individual (for self-occlusion). An example of this can be seen in **Figure 3**. The  
 1964 following procedure requires posture for all individuals in a frame. In case an individual  
 1965 does not have a valid posture in the given frame, its most recent posture and position are  
 1966 used as an approximation. The field is internally represented as a discretized vector of  
 1967 multi-dimensional pixel values. Depending on the resolution parameter ( $F_{\text{res}}$ ), which sets the  
 1968 number of pixels, each index in the array represents step-sizes of  $(F_{\text{max}} - F_{\text{min}})/F_{\text{res}}$  radians.  
 1969 The  $F$  values are constants setting the minimum and maximum field of view ( $-130^\circ$  to  $130^\circ$   
 1970 by default, which gives a range of  $260^\circ$ ). Each pixel consists of multiple data-streams: The  
 1971 distance to the other individual, the identity of the other individual and the body-part that  
 1972 the ray intersected with.

1973 Eyes are simulated to be located on the outline of the focal individual, near the head.  
 1974 The distance to the head can be set by the user as a percentage of midline-length. To find  
 1975 the exact eye position, the program calculates intersections between vectors going left/right  
 1976 from that midline point, perpendicular to the midline, and the individual's outline. In order  
 1977 to be able to simulate different types of binocular and monocular sight, a parameter for eye  
 1978 separation  $E_{\text{sep}}$  (radians) controls the offset from the head angle  $H_a$  per eye. Left and right  
 1979 eye are looking in directions  $H_a - E_{\text{sep}}$  and  $H_a + E_{\text{sep}}$ , respectively.

1980 We iterate through all available postures in a given frame and use a procedure which is  
 1981 very similar to depth-maps (**Williams (1978)**) in e.g. OpenGL. In the case of 2D visual fields,  
 1982 this depth-map is 1D. Each pixel holds a floating-point value (initialized to  $\infty$ ) which is  
 1983 continuously compared to new samples for the same position – if the new sample is closer to  
 1984 the "camera" than the reference value, the reference value is replaced. This way, after all  
 1985 samples have been evaluated, we generate a map of the objects closest to the "camera" (in  
 1986 this case the eye of the focal individual). For that to work we also have to keep the identity  
 1987 in each of these discrete slots maintained. So each time a depth value is replaced, the same  
 1988 goes for all the other data-streams (such as identity and head-position). When an existing  
 1989 value is replaced, values in deeper layers of occlusion are pushed downwards alongside the  
 1990 old value for the first layer.

1991 Position of the intersecting object's top-left corner is located at  $\hat{P}$ . Let  $E_e$  be the position  
 1992 of each eye, relative to  $\hat{P}$ . For each point  $P_j$  (coordinates relative to  $\hat{P}$ ) of the outline, check  
 1993 the distance between  $E_e$  and the outline segments  $(P_j - P_{j-1})$ . For each eye  $E_e$ :

- 1994 1. Project angles ranging from  $[\text{atan}2(P_{j-1} + E_e), \text{atan}2(P_j + E_e)]$ , where  $\alpha_e$  is the eye orien-  
 1995 tation, using:

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

1996 1997 1998 1999 angle\_normalize( $\beta$ ) normalizes beta to be between  $[-\pi, \pi]$ .

- 2000 2. if either  $\max(R)$  or  $\min(R)$  is inside the visual field ( $0 \leq \Gamma_e(\beta) \leq 1$ ):

2001 (a) We call the first angle satisfying the condition  $\beta$ .

2002 (b) Then the search range becomes  $R := [\lfloor \max\{\beta - 0.5; 0\} \rfloor, \lceil \beta + 0.5 \rceil]$ , where the ele-  
 2003 ments in  $R$  are integers.

2004 (c) Let  $\delta_{j,e} = \|P_{j-1} - E_e\|$ , the distance between outline point at  $j - 1$  and the eye  
 2005 (interpolation could be done here).

2006 (d) Let index  $k \in \mathbb{N}, k \in R$  be our index into the first layer of the depth-map  $\text{depth}_0$ :

|      |                                                                                                                                                                                                     |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2007 | (e) <b>if</b> $\text{depth}_0(k) > \delta_{j,e}$ : Calculate all properties $D_0(k) := \{\text{head\_distance}, \dots \in \text{data\_streams}\}^T$ , and push values at $k$ in layer 0 to layer 1. |
| 2008 |                                                                                                                                                                                                     |
| 2009 | (f) <b>otherwise</b> , if $\text{depth}_1(k) > \delta_{j,e}$ , calculate properties for layer 1 instead and move data from layer 1 further down, etc.                                               |
| 2010 |                                                                                                                                                                                                     |
| 2011 | The data-streams are calculated individually with the following equations:                                                                                                                          |
| 2012 | • <b>Distance</b> : 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 $[0, 255]$ .                       |
| 2013 | • <b>Identity</b> : Is assigned alongside $\text{depth}_i(k)$ for each element that successfully replacing another in the map.                                                                      |
| 2014 | • <b>Body-part</b> : Let $T_i = \text{tail index}$ , $L_{l/r} = \text{number of points in left/right side of the outline given by tail- and head-indexes}$ :                                        |
| 2015 |                                                                                                                                                                                                     |
| 2016 | 1. <b>if</b> $i > T_i$ : $\text{head\_distance} = 1 -  i - T_i /L_l$                                                                                                                                |
| 2017 | 2. <b>else</b> : $\text{head\_distance} = 1 -  i - T_i /L_r$                                                                                                                                        |
| 2018 |                                                                                                                                                                                                     |
| 2019 |                                                                                                                                                                                                     |

2020 Appendix 7

2021  
2022  
2023  
2024  
2025  
2026

## 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.

2027  
2028  
2029  
2030

## 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:

2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043

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

2044  
2045

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:

2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057

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\*)

2058  
2059  
2060  
2061  
2062  
2063

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).

## 2064 Appendix 8

2065 **Automatic visual recognition**2066 **Network layout and training procedure**

2067 Network layout is sketched in *Figure 1c*. Using version 2.2.4 of Keras<sup>g</sup>, weights of densely  
 2068 connected layers as well as convolutional layers are initialized using Xavier-initialization (*Glorot*  
 2069 and *Bengio (2010)*). Biases are used and initialized to 0. The default image size in TRex is  
 2070 80×80, but can be changed to any size in order to retain more detail or improve computation  
 2071 speed.

2072 During training, we use the Adam optimizer (*Kingma and Ba (2015)*) to traverse the loss  
 2073 landscape, which is generated by categorical focal loss. *Categorical* focal loss is an adapta-  
 2074 tion of the original *binary* focal loss (*Lin et al. (2020)*) for multiple classes:

$$2077 \text{cFL}(j) = \sum_{c=1}^N -\alpha (1 - \mathbf{P}_{jc})^\gamma \mathbf{V}_{jc} \log (\mathbf{P}_{jc}),$$

$$2078$$

2079 where  $\mathbf{P}_{jc}$  is the prediction vector component returned by the network for class  $c$  in  
 2080 image  $j$ .  $\mathbf{V}$  is a set of validation images, which remains the same throughout the training  
 2081 process. It comprises 25% of the images available per individual. Images are marked *globally*  
 2082 when becoming part of the validation dataset and are not used for training in the current  
 2083 or any of the following steps.

2084 After each epoch, predictions are generated by performing a forward-pass through the  
 2085 network layers. Returned are the softmax-activations  $\mathbf{P}_{jc}$  of the last layer for each image  $j$   
 2086 in the validation dataset. Simply calculating the mean of

$$2088 \bar{A} = \frac{1}{M} \sum_{j \in [0, M]} \begin{cases} 1 & \text{if } \mathbf{P}_j = \mathbf{V}_j, \\ 0 & \text{otherwise} \end{cases},$$

$$2089$$

$$2090$$

2091 gives the mean accuracy of the network.  $M$  is the number of images in the validation  
 2092 dataset, where  $\mathbf{V}_j$  are the expected probability vectors per image  $j$ . However, much more  
 2093 informative is the per-class (per-identity) accuracy of the network among the set of images  
 2094  $i$  belonging to class  $c$ , which is

$$2097 I_c = \{j; \text{where } \mathbf{V}_{jc} = 1, j \in [0, M]\},$$

$$2098$$

2099 given that all vectors in  $\mathcal{V}$  are one-hot vectors – meaning the vector has length  $N$  with  
 2100  $\mathbf{V}_{j\phi} = 0 \forall \phi \neq c$  and  $\mathbf{V}_{jc} = 1$ .

$$2102 A_c = \frac{1}{|I_c|} \sum_{j \in I_c} \begin{cases} 1 & \text{if } \mathbf{P}_j = \mathbf{V}_j \\ 0 & \text{otherwise} \end{cases}$$

$$2103$$

$$2104$$

2105 Another constant, across training units – not just across epochs, is the set of images used  
 2106 to calculate mean uniqueness  $\bar{U}$  (see Box 1, as well as Guiding the Training Process). Values  
 2107 generated in each epoch  $t$  of every training unit are kept in memory and used to calculate  
 2108 their derivative  $\bar{U}'(t)$ .

2109 **Stopping-criteria**

2110 A training unit can be interrupted if one of the following conditions becomes true:

- 2111  
2113  
2112  
2114
1. Training commenced for at least  $t = 5$  epochs, but uniqueness value  $\bar{U}$  was never above

$$\bar{U}_{\text{best}}^2 > \bar{U}(t) \forall t$$

2117 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.

- 2118  
2120  
2121
2. The worst accuracy value per class has been "good enough" in the last three epochs:

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

- 2122  
2123  
2124  
2125
3. The global uniqueness value has been plateauing for more than 10 epochs.

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

- 2128  
2130  
2131
4. Overfitting: Change in loss is very low on average after more than 5 epochs. Mean loss is calculated as follows:

$$\text{cFL}_\mu(t) = \frac{1}{5} \sum_{k \in [t-6, t-1]} \text{cFL}(k)$$

2134  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152

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

$$\lambda(t) = \lfloor \ln(\text{cFL}(t)) \rfloor - 1$$

$$\frac{1}{5} \sum_{k \in [t-5, t]} \max \left\{ \epsilon; |\text{cFL}(k) - \text{cFL}_\mu(k)| \right\} < 0.05 * 10^{\lambda(t)}$$

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.

<sup>a</sup>See [keras.io](#) documentation for default arguments

## 2153 Appendix 9

2154 **Data used in this paper and reproducibility**

2155 All of the data, as well as the figures and tables showing the data, have been generated  
 2156 automatically. We provide the scripts that have been used, as well as the videos if requested.  
 2157 "Data" refers to converted video-files, as well as log- and NPZ-files. Analysis has been done  
 2158 in Python notebooks, using mostly matplotlib and pandas, as well as numpy to load the data.  
 2159 Since TRex and TGrabs, as well as idtracker.ai have been run on a Linux system, we were  
 2160 able to run everything from two separate bash files:

- 2161     1. run.bash  
 2162     2. run\_idtracker.bash

2163 Where (1) encompasses all trials run using TRex and TGrabs, both for the speed- and  
 2164 recognition-datasets. (2) runs idtracker.ai in its own dedicated Python environment, using  
 2165 only the recognition-dataset. The parameters we picked for idtracker.ai vary between  
 2166 videos and are hand-crafted, saved in individual .json files (see Table *Appendix 9 Table A1*  
 2167 for a list of settings used). We ran multiple trials for each combination of tools and data  
 2168 with  $N = 5$  where necessary:

- 2169     • 3x TGrabs [speed-dataset]  
 2170     • 5x TRex + recognition [recognition-dataset]  
 2171     • 3x idtracker.ai [recognition-dataset]  
 2172     • 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 *Appendix 4 Table A3*)  
     – 3x testing accuracy of basic tracking (see Table *Appendix 4 Table A2*), with posture  
       enabled

2177 A Python script used for *Figure 7*, which is run only once. It generates a series of results  
 2178 for the same video (video 7 with 100 individuals) with different sample-sizes. It uses a single  
 2179 set of training samples and then – after equalizing the numbers of images per individual –  
 2180 generates multiple virtual subsets with fewer images. They span 15 different sample-sizes  
 2181 per individual, saving a history of accuracies for each run. We repeated the same procedure  
 2182 with for the different normalization methods (no normalization, moments and posture),  
 2183 each repeated five times.

2184 As described in the main text, we recorded memory usage with an external tool (syrupy)  
 2185 and used it to measure both software solutions. This tool saves a log-file for each run, which  
 2186 is appropriately renamed and stored alongside the other files of that trial.

2187 All runs of TRex are preceded by running a series of TGrabs commands first, in order  
 2188 to convert the videos in the datasets. We chose to keep these trials separately and load  
 2189 whenever possible, to avoid data-duplication. Since subsequent results of TGrabs are always  
 2190 identical (with the exception of timings), we only keep one version of the PV files (The PV  
 2191 file format) as well as only one version of the results files generated using live-tracking.  
 2192 However, multiple runs of TGrabs were recorded in the form of log-files to get a measure of  
 2193 variance between runs in terms of speed and memory.

2194 **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

**Appendix 9 Table A1.** 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.

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

2195

2196

2197

2198

2199

2200

2201

2202

2203

2204

2205

2206

2207

2208

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).

2209

2210

### Comparison between trajectories from different softwares, or multiple runs of the same software

2211

2212

2213

2214

2215

2216

2217

2218

2219

2220

2221

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.

## 2222 Appendix 10

### 2223 Matching probabilities

2224 One of the most important steps, when matching objects in one frame with objects in the  
 2225 next frame, is to calculate a numerical landscape that can then be traversed by a maximiza-  
 2226 tion algorithm to find the optimal combination. This landscape, which can be expressed  
 2227 as an  $m \times n$  matrix  $\mathbf{P}(t)$ , contains the probability values between [0, 1] for each assignment  
 2228 between individuals  $i$  and objects  $B_j$ .

2229 Below are definitions used in the following text:

- 2230 •  $T_\Delta$  is the typical time between frames (s), which depends on the video
- 2231 •  $\tau_i < t$  is most recent frame assigned to individual  $i$  previous to the current frame  $t$
- 2232 •  $P_{\min}$  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.
- 2233 •  $F(t \in \mathbb{R}) \rightarrow \mathbb{N}$  is the frame number associated with the time  $t$  (s)
- 2234 •  $\mathcal{T}(f \in \mathbb{N}) \rightarrow \mathbb{R}$  is the time in seconds of frame  $f$ , with  $F(\mathcal{T}(f)) = f$
- 2235 •  $\mathbf{x}$  indicates that  $x$  is a vector
- 2236 •  $\mathbf{U}(\mathbf{x}) = \mathbf{x} / \|\mathbf{x}\|$

2237 Some values necessary for the following calculations are independent of the objects in  
 2238 the current frame and merely depend on data from previous frames. They can be re-used  
 2239 per frame and individual in the spirit of dynamic programming, reducing computational  
 2240 complexity in later steps:

$$2241 \mathbf{v}_i(t) = \mathbf{p}'_i(t) = \frac{\delta}{\delta t} \mathbf{p}_i(t)$$

$$2242 \hat{\mathbf{v}}_i(t) = \mathbf{v}_i(t) * \begin{cases} 1 & \text{if } \|\mathbf{v}_i(t)\| \leq D_{\max} \\ D_{\max} / \|\mathbf{v}_i(t)\| & \text{otherwise} \end{cases}$$

$$2243 \mathbf{a}_i(t) = \frac{\delta}{\delta t} \hat{\mathbf{v}}_i(t)$$

2244 Velocity  $\mathbf{v}_i(t)$  and acceleration  $\mathbf{a}_i(t)$  are simply the first and second derivatives of the indi-  
 2245 viduals position at time  $t$ .  $\hat{\mathbf{v}}_i(t)$  is almost the same as the raw velocity, but its length is limited  
 2246 to the maximally allowed travel distance per second ( $D_{\max}$ , parameter `track_max_speed`).

2247 These are then further processed, combining and smoothing across values of multiple  
 2248 previous frames (the last 5 valid ones). Here,  $\bar{f}(x)$  indicates that the resulting value uses  
 2249 data from multiple frames.

$$2250 \bar{s}_i(t) = \text{median}_{k \in [F(\tau)-5, F(t)]} \|\hat{\mathbf{v}}_i(\mathcal{T}(k))\|$$

2251 is the speed at which the individual has travelled at recently. The mean direction of  
 2252 movement is expressed as

$$2253 \bar{\mathbf{d}}_i(t) = \frac{1}{F(t) - F(\tau) + 5} \sum_{k \in [F(\tau)-5, F(t)]} \hat{\mathbf{v}}_i(\mathcal{T}(k))$$

2254 with the corresponding direction of acceleration

$$2255 \bar{\mathbf{a}}_i(t) = \mathbf{U} \left( \frac{1}{F(t) - F(\tau) + 5} \sum_{k \in [F(\tau)-5, F(t)]} \mathbf{a}_i(\mathcal{T}(k)) \right).$$

2275  
2276  
2277  
2278  
2279  
2280  
2281

The predicted position for individual  $i$  at time  $t$  is calculated as follows:

$$\dot{\mathbf{p}}_i(t) = s_i(t) \sum_{k \in [F(\tau_i), F(t)-1]} w(k) (\bar{\mathbf{d}}_i(t) + \mathcal{T}'(k) * \bar{\mathbf{a}}_i(t)),$$

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:

$$\mathbf{P}(t) = \begin{bmatrix} P_0(t | B_0) & \dots & P_n(t | B_0) \\ \vdots & \ddots & \vdots \\ P_0(t | B_m) & \dots & P_n(t | B_m) \end{bmatrix}. \quad (3)$$

Probabilities  $P_i(t | B_j)$  for all potential connections between blobs  $B_j$  and identities  $i$  at time  $t$  are calculated by first predicting the expected position  $\dot{\mathbf{p}}_i(t)$  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  $\bar{s}_i(t)$ , direction  $\bar{\mathbf{d}}_i(t)$ , acceleration  $\bar{\mathbf{a}}_i(t)$  and angular momentum  $\bar{\alpha}'_i(t)$ , the individual's projected position  $\dot{\mathbf{p}}_i(t)$  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  $\mathbf{P}(t)$  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.

$$F_{\min} = \min \left\{ \frac{1}{T_\Delta}, 5 \right\}$$

$$R_i(t) = \left\| \left\{ \mathcal{T}(k) \mid F(t) - T_\Delta^{-1} \leq k \leq t \wedge \mathcal{T}(k) - \mathcal{T}(k-1) \leq T_{\max} \right\} \right\|$$

$$T_i(t) = \left( 1 - \min \left\{ 1, \frac{\max \{0, \tau_i - t - T_\Delta\}}{T_{\max}} \right\} \right) * \begin{cases} \min \left\{ 1, \frac{R_i(\tau_i)-1}{F_{\min}} + P_{\min} \right\} & F(\tau_i) \geq F(t_0) + F_{\min} \\ 1 & \text{otherwise} \end{cases} \quad (4)$$

$S_i(t | B_j)$  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.

$$S_i(t | B_j) = \left( 1 + \frac{\left\| (\mathbf{p}_{B_j}(t) - \dot{\mathbf{p}}_i(t)) / (\tau_i - t) \right\|^2}{D_{\max}} \right)^{-2} \quad (5)$$

2325

2326

2327

2328

2329

2330

2331

2332

2333

2334

2335

2336

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

$$\mathbf{a} = \dot{\mathbf{p}}_i(t) - \mathbf{p}_i(\tau_i)$$

$$\mathbf{b} = \mathbf{p}_{B_j}(t) - \mathbf{p}_i(\tau_i)$$

$$A_i(t, \tau_i | B_j) = \begin{cases} 1 - \frac{1}{\pi} |\text{atan2}\{\|\mathbf{a} \times \mathbf{b}\|, \mathbf{a} \cdot \mathbf{b}\}| & \text{if } \|\mathbf{a}\| > 1 \wedge \|\mathbf{b}\| > 1 \\ 1 & \text{otherwise} \end{cases} \quad (6)$$

2337

2338

2339

2340

2341

2342

2343

2344

2345

2346

2347

2348

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 (4), (5) and (6) into a weighted probability product yields:

$$P_i(t, \tau_i | B_j) = S_i(t | B_j) * (1 - \omega_1 (1 + A_i(t, \tau_i | B_j))) * (1 - \omega_2 (1 + T_i(t, \tau_i))) \quad (7)$$

Results from equation (7) 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.

2349 **Appendix 11**

2350           **Algorithm for splitting touching individuals**

2351           **Data:** image of a blob,  $N_e$  number of expected blobs

2352           **Result:**  $N \geq N_e$  smaller image-segments, or *error*

2353           threshold =  $T_c$ ;

2354           **while** threshold < 255 **do**

2355            blobs = apply\_threshold(image, threshold);

2356            **if** ||blobs|| = 0 **then**

2357            | **break**;

2358            **end**

2359            **if** ||blobs||  $\geq N_e$  **then**

2360            | sort blobs by size in decreasing fashion;

2361            | loop through all blobs  $i$  up to  $i \leq N_e$  and detect whether the size-ratio

2362            | between them is roughly even. until then, we keep iterating.;

2363            **if** min{ratio <sub>$i$</sub> ,  $\forall i \in [0, N_e]$ } < 0.3 **then**

2364            | threshold = threshold + 1;

2365            | **continue**;

2366            **else**

2367            | **return** blobs;

2368            **end**

2369            **else**

2370            | threshold = threshold + 1;

2371           **end**

2372           **end**

2373           **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.  $T_e$  is the starting threshold constant parameter, as set by the user.

## 2375 Appendix 12

2376 **Stability for Highly Deformable Bodies**

2377 The videos used in evaluating our method primarily consist of insects and fish, which are  
 2378 relatively "stiff" compared to animals with more deformable bodies, such as mice. To assess  
 2379 how this would affect the stability of our posture and visual identification algorithms, we  
 2380 were kindly provided with one additional video of four C57BL/6 mice by D. Mink and M.  
 2381 Groettrup, which we included as part of the supplementary material. We were also given  
 2382 access, thanks to G. G. de Polavieja, to another video with four black mice that is part of  
 2383 *Romero-Ferrero et al. (2019)* (linked under [idtrackerai.readthedocs.io/en/latest/data.html](https://idtrackerai.readthedocs.io/en/latest/data.html)  
 2384 [accessed 02/03/2021]).

2385 Both videos, listed in *Appendix 12 Table A1* and previewed in *Appendix 12 Figure A1*,  
 2386 show different groups of four freely behaving mice. The following analysis was performed  
 2387 using the same scripts used to generate *Table 3*, although each video has only been auto-  
 2388 matically tracked once. We manually generated verified trajectories for both videos in full,  
 2389 following the same procedure described in Human validation, and compared them to the  
 2390 automatically generated trajectories. As can be seen in *Appendix 12 Table A1*, TReX provides  
 2391 highly accurate results for both videos ( $\geq 99.6\%$ ).

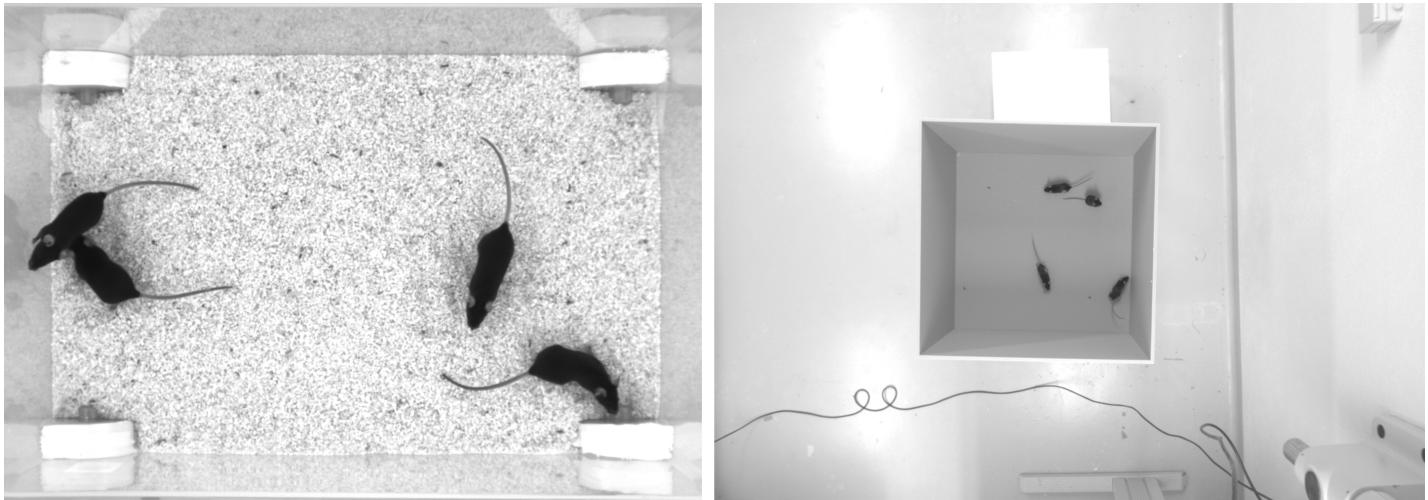
2392 Tracking, in theory and in practice as per our results here, is not generally impacted by  
 2393 the shape of individuals. However, individuals of some species tend to stay close/on top of  
 2394 con-specifics, which may render them impossible to track during periods where traditional  
 2395 image processing methods are unable to separate them. This explains the  $> 6\%$  interpolated  
 2396 frames in V1 (see *Appendix 12 Table A1*), and also gives a reason why there is similarity  
 2397 between video 9 and V1 in that respect – the locusts in video 9 also spend much time either  
 2398 on top of others, or in places where they are harder to see.

2399 Very short segments of mistaken identities (with a maximum length of less than 200ms)  
 2400 occurred whenever individuals "appear" only for a short moment and the segment does  
 2401 not contain enough data to be properly matched with a learned identity. Correct identities  
 2402 were reassigned in all cases after the individuals could be visually separated from each other  
 2403 again. Since such events only make up  $< 1\%$  of the tracked data, we can conclude that TReX  
 2404 is able to reliably identify all individuals in these two additional videos as well.

2405 Furthermore, we found that our method for posture estimation works well despite the  
 2406 more deformable bodies and complex 3D-postures of mice. Head and tail may switch occa-  
 2407 sionally, especially when animals shrink to "a circle" from the viewpoint of the camera. Over-  
 2408 all, however, by far most samples are normalised correctly – as can be seen in *Appendix 12*  
 2409 *Figure A2* and *Appendix 12 Figure A3*.

**Appendix 12 Table A1.** 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%.

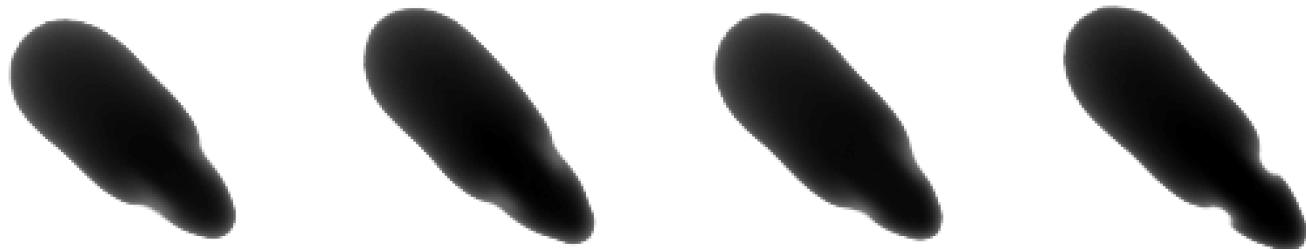
| video                                    | # ind. | reviewed (%) | interpolated (%) | TReX            |
|------------------------------------------|--------|--------------|------------------|-----------------|
| (V1) <i>Romero-Ferrero et al. (2019)</i> | 4      | 100.0        | 6.41             | $99.6 \pm 0.0$  |
| (V2) D. Mink, M. Groettrup               | 4      | 100.0        | 1.74             | $99.82 \pm 0.0$ |



**Appendix 12 Figure A1.** Screenshots from videos V1 and V2 listed in [Appendix 12 Table A1](#). Left (V1), video of four black mice (17min, 1272x909px resolution) from [Romero-Ferrero et al. \(2019\)](#). Right (V2), four C57BL/6 mice (1:08min, 1280x960px resolution) by M. Groettrup, D. Mink.

**Appendix 12 Figure A1-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 [Appendix 12 Table A1](#). <https://youtu.be/UnqRNKrYiR4>

**Appendix 12 Figure A1-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>



**Appendix 12 Figure A2.** Median of all normalised images (N=7161, 7040, 7153, 7076) for each of the four individuals from V1 in [Appendix 12 Table A1](#). 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).



**Appendix 12 Figure A3.** Median of all normalised images (N=1593, 1586, 1620, 1538) for each of the four individuals from V2 in [Appendix 12 Table A1](#). Resolution per animal is lower than in V1, but ears are still clearly visible.