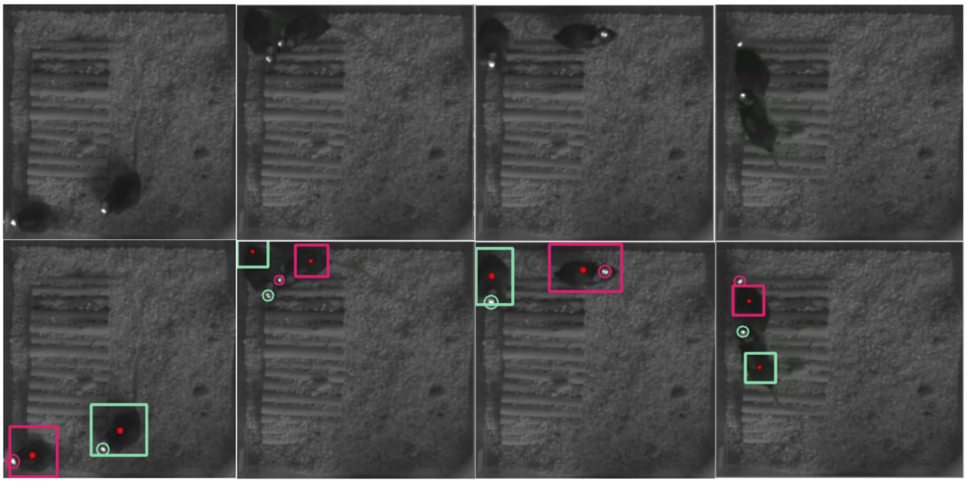
MICE TRACKING USING INFRARED SUBCUTANEOUS IMPLANTS FOR ERROR DETECTION



Master Thesis

To obtain the title of
Master in Science, Visual Computing and Communication
at the Technische Universiteit Delft
to be defended publicly on 6 April, 2021, 10:45.

by

Javier GUINEA PÉREZ

Faculty of Electrical Engineering, Mathematics and Computer Science

Thesis supervised by:

Supervisor: Prof. dr. AJ. van der Veen
Cosupervisor: M. Sc. F. Nassirina

Thesis committee:

Prof. dr. AJ. van der Veen,	Technische Universiteit Delft
M.Sc. F. Nassirina,	Erasmus MC
Prof. dr. R. Marroquim,	Technische Universiteit Delft

Science is a wonderful thing
if one does not have to earn one's living at it.

Albert Einstein

Contents

Summary

The objective of this thesis was to develop a rodent tracker using the FlashTrack implants, and allow for tracking data to be used for behavioural research. The task was split into three main problems: detection, tracking and error detection.

The detection was solved with basic image processing. The first step was background removal, using median filtering. Later, blob detection after some processing was done. Detections were differentiated into cases where mice are together in contact events and where they are alone. Bounding boxes were generated for the contours of lone mice, while the distance transform was employed to detect the joint ones.

To track the moving targets, Kalman filtering was used on the bounding boxes of the detector. This approach was based on the Simple Online Realtime Tracking framework, adapting it to the particularities of mice. Other approaches were tried, but SORT was the chosen one.

The infrared subcutaneous implants of FlashTrack allow for identity verification through code detections. To process the codes of each track, a Gaussian Mixture Model is trained to be the classifier of the detections. A track handling module was built to monitor the estimated tracks and code detections, and verify correct assignments. Detected erroneous tracks were discarded.

Synthetic data was used to evaluate the tool. Artificial datasets were developed in Blender. Common metrics for evaluation of multiple object trackers were gathered and discussed, as well as a comparison with one of the state of the art animal trackers.

LIST OF FIGURES

1 Introduction

"I've got ninety-nine problems."

Jay-Z

Researchers need measurable and accurate descriptions of animal behaviour both to analyse the structure of the behaviour itself, as well as for revealing underlying neural or genetic mechanisms. Computational tools are becoming more and more prevalent in the gathering of such data.

The aim of the project is to study the social interactions between multiple rodents within one environment. It consists in the development of a software solution that tracks and stores the rodents' position with the future prospect of gathering statistics based on this.

This chapter highlights the motivation behind the project, gives an overview of the objectives, and briefly describes the experimental setup and resources employed in the development of the solution.

1.1. Motivation

In biology the concept of behaviour is usually linked with movement and goals. It is the response of whole living organisms (both individuals and groups) to external or internal stimuli [1].

Behavioural neuroscience studies the brain mechanisms underlying human or animal behaviour. It tackles problems such as the causes and treatments of autism, depression and other diseases, the evolutionary basis of behaviour or the difference between animal behaviours depending on external signals.

A measurable description is required for the study of behaviour. Traditionally two distinct methods have been employed: manual descriptions of the behaviour (video tagged by human observers) and simple assays that capture an aspect of some complex behaviour (such as priority of right of way to assert dominance in mice) [2]. Nowadays, computational methods have broadened the reach of behavioural biology. Video tracking is used to automatically tag and analyse an otherwise insurmountable amount of data.

The raw data of the trajectories, on its own, is not a useful representation of behaviour. Instead, behavioural tracking methods categorize the behaviour of each subject at each time point into different categories (grooming, chasing, fleeing, etc). This segmentation is then statistically interpreted in order to reach meaningful conclusions, measuring changes in frequency and manner of such behaviours.

1.1.1. Animal Video Tracking

Video tracking is a data association problem in which pixels at each frame have to be tagged as either belonging to the background or to one of the targets, generating a positional sequence that conveys the trajectory of the animal throughout the video sequence. Further characteristics of the tracked objects can be discerned on a pixel-level, such as the different body parts, orientation or area. Both the trajectory- and pixel-based features are used to classify the behaviour, which is then done automatically either with a rule-based, supervised or unsupervised classifier.

1.1.2. Rodent Behavioural Testing

Mice are the most commonly used mammalian research model. There are many advantages to working with mice: size, costs, genetic diversity or sample size requirements. The mouse and the human genome have a very similar gene content, so mice can show many of the symptoms of human disease, making them good models for research [3].

Transgenic technology is very advanced in mice, it is possible to insert segments of human alleles into mice, as to provoke particular genetic syndromes [4]. Phenotyping tools make assessing the genome of particular mice simple. This has allowed the development of many strains of laboratory mice that model a particular human behavioural disorder, such as autism or Williams syndrome, allowing for novel lines of research that are not constrained by human testing.

In long term experiments, mice must be identified throughout the research. This means that each time the subjects are tested, they must retain their identity.

1.2. Problem Statement and Thesis Goals

The project being presented has been developed with the aim of creating an automatic behavioural testing solution that measures social interaction between several mice in an enclosure. Social interaction in mice encompasses all behaviours that include more than one rodent, such as group making, mating, following or fleeing. The project is constrained to analyse positional statistics regarding the time spent at different areas in the enclosure.

1.2.1. Location

The neuroscience department of Erasmus MC is responsible for the FlashTrack project, where this Thesis fits into. Before developing the software solution in-house, Erasmus MC worked with Noldus IT to create a tracker using their proprietary tool EthoVision [5]. After a couple of months, Erasmus changed focus and decided to bring someone in to create a solution that would be open source and built in python. This is where the project begins.

1.2.2. Setup

A tracker was developed to work with implants that facilitate the task of detection and error correction. The mice employed for testing are fitted with an implant above their skull or between the shoulder blades, below the skin, that carries two infrared (IR) LED lights. The two lights give information regarding the position and direction of the head, the latter by creating a vector from one light to the other. Of the two LEDs one is always powered on, while the other blinks at 15 Hz with a unique 8 bit code that identifies the mouse. The codes employed are permutationally unique, meaning that no matter when you start the reading of the code, it is always distinguishable from the rest.

Mice cannot sense infrared frequencies, so the lights do not disturb them, and the location of the implants allow for pose estimation of the head's direction.

The experimental setup consists in a 30 by 30 cm enclosure where two mice fitted with wireless implants move freely and are recorded by an infrared camera. A video feed of a 30 by 30 cm enclosure is gathered by a 30 frames per second camera, with a quality of

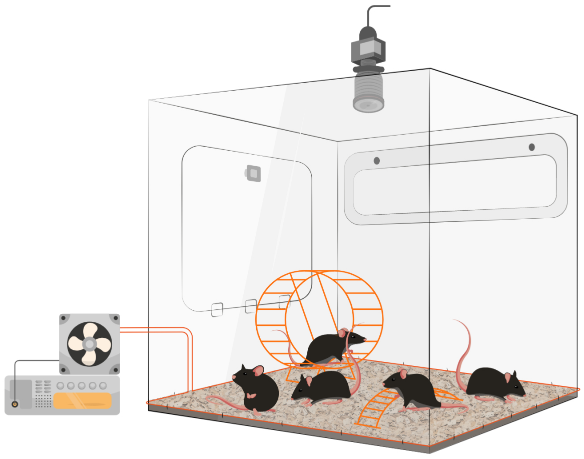


Figure 1.1: Diagram of the experimental setup (courtesy of FlashTrack)

1280x1024 pixels, positioned on top of the cage. Figure 1.1 shows the setup, while figure 1.2 shows a frame from one of the videos.

1.2.3. GOALS

The objective of the thesis can be summarized as follows:

"The development of an open-source sustainable tracking solution for free moving mice with the FlashTrack implants, with the aim of correcting misdetections and miss-labelings and some behavioural information is gathered."

This problem can be expanded into more concise and measurable goals:

- Study the detection methods available in the literature for behavioural research and identify the methods which are suitable for detecting the high occlusion case that are free moving mice in the enclosure.
- Implement a detection algorithm in order to optimally detect the mice and their IR lights.
- Study the tracking algorithms available in the literature and identify the methods which are suitable for detecting the high occlusion and quick moving mice case.
- Implement a mice-tracking system which can track the rodents and assign their lights.
- Create a code detection algorithm that detects the code of a given a sequence of detections of the same light.
- Create an error detection-correction scheme based on the code detections.
- Gather positional statistics of the rodents that are useful in behavioural research.

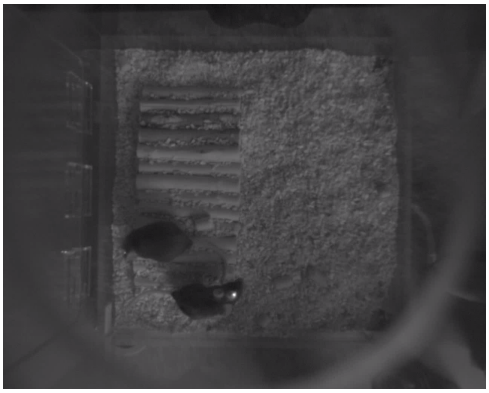


Figure 1.2: Raw frame from the video feed.

- Develop the software solution using python and, where possible, use design patterns that allow for easy improvement of the code.
- Create a labeled dataset to evaluate the detection accuracy of the mice images and their ID.

1.3. THESIS ORGANIZATION

THIS thesis starts with a literature review of the different methods used in rodent tracking, and the approaches employed in the wider multiple object tracking problem. The choice for the project is presented with its two main components: detector and tracker. Each warrants its own chapter.

The next two chapters present each component (detector and tracker) and the reasoning behind that choice of solution.

A novel method for error detection is presented in the fifth chapter, along with the data structure that enables its use. A short discussion on possible error correcting approaches ends this chapter.

The last chapter presents the evaluation of the solution. The complexity of the task and the workaround to some of its problems begins this section. The numerical results are then presented and discussed, and the chapter ends with a comparison between these results and those of a state of the art mice tracker.

Lastly, the conclusions of the Thesis give an overview of the work done and that which is left undone: the future lines of research.

The methods and results of the multiple statistical analyses done to judge different properties of the videos and of the tool are summarized in the appendix.

REFERENCES

[1] Daniel A. Levitis, William Z. Lidicker, and Glenn Freund. Behavioural biologists don't agree on what constitutes behaviour. *Animal behaviour*, 78(1):103–110, 07 2009. 20160973[pmid].

[2] S.E. Roian Egnor and Kristin Branson. Computational analysis of behavior. *Annual Review of Neuroscience*, 39(1):217–236, 2016. PMID: 27090952.

[3] Lucy R. Osborne. Animal models of williams syndrome. *American journal of medical genetics. Part C, Seminars in medical genetics*, 154C(2):209–219, 05 2010. 20425782[pmid].

[4] Douglas Wahlsten. *Mouse Behavioral Testing*. Academic Press, 2010.

[5] Lucas P. J. J. Noldus, Andrew J. Spink, and Ruud A. J. Tegelenbosch. EthoVision: A versatile video tracking system for automation of behavioral experiments. *Behavior Research Methods, Instruments, & Computers*, 33(3):398–414, August 2001.

2

Literature Review

"All the reading she had done had given her a view of life that they had never seen."

Roald Dahl, Matilda

Rodent tracking is the next step toward large scale behavioural research. It is part of the wider multiple object tracking problem, where moving targets are identified across a video sequence and their trajectories stored. The most common way of doing it is by first detecting the targets at every frame and then linking the detections into trajectories. Therefore, both a detector and a tracking strategy must be implemented. Once there are defined tracks per mouse, behavioural inference is a matter of geometrical constraints.

This chapter is intended to be an introduction to the problem of multiple object tracking, an overview of the image processing tools employed in the development of such a tracker and a brief survey on the current tools employed in rodent behavioural research.

First, a high level description of the problem of multiple object tracking is given. Next, a per-component description of a tracking solution. A deeper explanation of the detector of a multiple object tracker follows this breakdown, going into detail on some of the image processing techniques employed. Finally a comparison of different research papers that have tackled the issue of rodent tracking. As the conclusion of the chapter a summary of the available options, their drawbacks and advantages, as well as the chosen approach.

2.1. MULTIPLE OBJECT TRACKING

TRACKING the real world position and orientation of moving targets is a common problem in the field of computer vision. It has many applications, such as pedestrian tracking (for surveillance and autonomous locomotion), surgical robotics or livestock management.

Visual multiple object tracking (MOT) aims at locating multiple targets in a visual sequence, inferring their trajectories and maintaining their identities. MOT has intrinsic issues that complicate this task: targets might disappear from the field of view or they might occlude each other, especially in crowded environments [1]. Prediction of the position of the objects is useful to handle such problems of misdetection.

The most common approach to MOT is tracking-by-detection, also known as detection based tracking (DBT) [2], where the problem is solved by following two steps: first detection, in which a pre-trained object detector is applied to each frame of the video to segment the objects from the background, and then a tracking step, where different observations are linked to different targets.

Targets (also known as tracklets) are described in the system by the *observation model*, which captures their appearance as well as their motion. This model is employed by the detector, generating the observations. A *tracking system* links each observation to the targets, possibly by generating a prediction of where to find them, thus reducing the search space. There are two main classes of tracking systems: Bayesian and data association methods, depending on the way they link the observations and the targets.

2.2. COMPONENTS

THERE are three major issues to be considered in MOT: the detection of objects frame by frame, the measurement of similarity between objects across frames, and the recovery of identity based on these similarity measures. The first two problems depend on the modeling of appearance, motion, interaction, exclusion and occlusion of the targets. The last one, linking observations to maintain identity, is the inference problem, where approaches can be clustered as Bayesian and data association methods.

2.2.1. OBSERVATION MODEL

Targets are described in the observation model. Not only is the appearance of the object important, but also its motion and its interaction with other targets. This information is then aggregated to create a distance formulation between observations across frames, which allows the tracker to link them.

APPEARANCE MODEL

The appearance model describes the characteristics of an object by using some features. Through a statistical measure, a computation of the similarity between two observations is calculated.

The **visual representation** uses both local features and region features. *Local features* describe the objects pixelwise frame by frame. An example is the Kanade–Lucas–Tomasi Feature Tracker [3], that searches "good" local features for tracking [4]. Another example of local features is the optical flow, the apparent movement of brightness patterns on an image, if taking the pixel as the finest local range [1], employed both for motion information retrieval [5] and for linking detection responses before data association. *Region features* are extracted from a wider range (i.e. a bounding box). They can be sorted depending on the order of discrepancy used when computing the representation.

Zero order features, particularly the color histogram [6] [7] [8], are the most used in MOT [1] and are very efficient [2]. First order or gradient based methods are also commonly used, like the histogram of oriented gradients [9] [10] or level-set formulations [11] [12] [9]. Higher order features are less common, but nonetheless spatio-temporal information [13] and the regional covariance matrix [6] have successful implementations.

The comparison of observations yields a value that is used to compare similarity and later link different observations in tracklets. Common distance measures are employed over one cue (distance, color, shape, depth, etc), as well as several cues aggregated to complement each other by boosting, concatenating or summing. A simple, yet very effective, measure of similarity between bounding boxes is the intersection over union (IOU), the ratio of the intersection of two bounding boxes and their union [14].

MOTION MODEL

The motion model estimates the position of the tracklet in future frames, reducing the search space, helping correct noisy detections, and adding useful information for the linking stage. In general, objects are assumed to move smoothly in the image [1] [2].

Linear motion models assume the tracklet's motion can be estimated by linear regression, by assuming that the real motion is linear over short time periods [14] [10].

Non-linear motion models can work in situations where targets move freely, detections occur sparsely or objects move very quickly [15] [11].

EXCLUSION MODEL

Real world objects cannot occupy the same physical space. The exclusion model handles this type of collisions. Spatio-temporal constraints ensure that neither two close detections are identified with different trajectory labels and that neither two different detections in the same frame are assigned to the same target.

2.2.2. TRACKING METHODS

Formally, tracking can be viewed as a multi-variable estimation problem. Given a video, s_t^i is defined as the state of the i-th object in the t-th frame. $\boldsymbol{S}_t = \{s_t^1, s_t^2, ..., s_t^{M_t}\}$ is then defined as the states of all the M_t objects in the t-th frame and $\boldsymbol{S}_{1:t} = \{\boldsymbol{S}_1, \boldsymbol{S}_2, ..., \boldsymbol{S}_t\}$ as all the states of all the objects form the first frame to the t-th frame.

Correspondingly and following the DTB paradigm, o_t^i are the collected observations at frame t of object i, with $\boldsymbol{O}_t = \{o_t^1, o_t^2, ..., o_t^{M_t}\}$ the collected observations of all M_t objects at time t, and $\boldsymbol{O}_{1:t} = \{\boldsymbol{O}_1, \boldsymbol{O}_2, ..., \boldsymbol{O}_t\}$ all the observations of all the objects form the first frame to the t-th frame.

The objective of MOT is to find the "best" sequential states of all the objects, by performing the maximum a posteriori estimation of the conditional distribution of the states given the observations:

$$\hat{\boldsymbol{S}}_{1:t} = \underset{\boldsymbol{S}_{1:t}}{\operatorname{argmax}} P(\boldsymbol{S}_{1:t}|\boldsymbol{O}_{1:t}) \tag{2.1}$$

Different tracking methods can be distinguished by their approach to this problem, as either probabilistic or deterministic-based.

BAYESIAN THEORY-BASED

Probabilistic inference approaches, also known as Bayesian approaches usually solve equation 2.1 by using a two-step iterative procedure:

Predict: $P(\boldsymbol{S}_t|\boldsymbol{O}_{1:t-1}) = \int P(\boldsymbol{S}_t|\boldsymbol{S}_{t-1})P(\boldsymbol{S}_{t-1}|\boldsymbol{O}_{1:t-1})d\boldsymbol{S}_{t-1}$

Update: $P(\boldsymbol{S}_t|\boldsymbol{O}_{1:t}) \propto P(\boldsymbol{O}_t|\boldsymbol{S}_t)P(\boldsymbol{S}_t|\boldsymbol{O}_{1:t-1})$

Where $P(\boldsymbol{S}_t|\boldsymbol{S}_{t-1})$ is the dynamic model (the tracking strategy) and $P(\boldsymbol{O}_t|\boldsymbol{S}_t)$ the observation model.

Kalman filters are part of the G-H family of filters, relying on two scalings to derive predictions from measurements. They are an appropriate way to address MOT when the number of objects is not too large. As the number of targets increases, identity switches become frequent and errors are propagated and difficult to correct due to the recursive nature of the method [16]. The single motion model is also limited in problems where the object to be tracked undergoes abrupt changes (*maneuvering objects*) [17].

Kalman filters are very efficient and simple to implement, and can achieve very good results in real time if coupled with good detectors[14]. They assume a linear system with Gaussian-distributed object states, but can be generalized to approximate a non-linear system through the Taylor expansion in the extended Kalman filter [18].

Particle filters, also known as condensation or sequential Monte Carlo, systematically deals with nonlinearity and non-Gaussianity. These methods do not work well with high dimensional state representations [19] [20]. One of their advantages is that they do not require any assumptions about the state-space model or the state distributions. Particle filters have had success in environments with static backgrounds [21] as well as changing backgrounds [22] [19].

DETERMINISTIC OPTIMIZATION

Deterministic based approaches maximize a delegate of $P(\boldsymbol{S}_{1:t}|\boldsymbol{O}_{1:t})$ over the set of available observations $\{\hat{\boldsymbol{O}}^n_{1:t}\}$. This delegate can be an energy function $E(\boldsymbol{S}_{1:t}|\boldsymbol{O}_{1:t})$ to be minimized or a likelihood function $L(\mathbf{O}_{1:t}|\mathbf{S}_{1:t})$ to maximize:

$$\hat{\mathbf{S}}_{1:t} = \underset{\boldsymbol{S}_{1:t}}{\operatorname{argmax}} P(\boldsymbol{S}_{1:t}|\boldsymbol{O}_{1:t}) = \underset{\boldsymbol{S}_{1:t}}{\operatorname{argmax}} L(\boldsymbol{O}_{1:t}|\mathbf{S}_{1:t}) = \underset{\boldsymbol{S}_{1:t}}{\operatorname{argmax}} \prod_n P(\hat{\boldsymbol{O}}^n_{1:t}|\mathbf{S}_{1:t}) \tag{2.2}$$

Most deterministic approaches boil down to graph problems. Bipartite graph matching is solved through either greedy algorithms or the optimal Hungarian algorithm [14], where two disjoint graphs (existing trajectories and new detections) are matched. These approaches can also be used to connect tracklets offline after creating them online through Bayesian methods [23].

2.2.3. CAUSALITY

Tracking systems, as estimation problems, can take either causal (online) or non-causal (offline) approaches. The difference between them is whether observations from future frames are taken into account when handling the current frame.

Online tracking relies solely on information from past frames up to the current frame. Their main advantage is efficiency and speed, and they are the most used in practice because of it. They go through the frames in a step-wise manner, sequentially processing them and gradually extending existing trajectories with current observations, producing results after one iteration per frame.

Offline tracking links all observations into trajectories. In theory, they are able to reach a global optimal solution, but suffer from high computational cost, algorithmic complexity and a delay in results. Sometimes, not all frames can be handled by the system, and subsequently the video is split into shorter batches.

Mixed methods are an interesting combination of online and offline tracking. Tracking can be done online while error correction and longer track matching is handled by an offline algorithm [23].

2.3. DETECTOR

THE detector method must be able to distinguish individual targets in a frame. This problem is encompassed by the wider set of problems known as **image segmentation**. Segmentation subdivides the image into regions of interest, in this case individual targets and background. In statistics, the problem is known as *cluster analysis*. In computer vision, this is one of the oldest and most widely studied problems [24].

The more measure of control over the image environment, the simpler the segmentation becomes, as it is the case with fixed cameras and fixed backgrounds [25]. Segmentation algorithms are mostly based on discontinuity and similarity of intensity values of an image, and range in complexity from simple or adaptive thresholding to convolutional neural networks.

Target segmentation usually begins by processing the image to remove the background, leaving only the objects to be tracked. Then, a shape detector is applied to find the desired individual targets.

2.3.1. BACKGROUND REMOVAL

The task of background removal can be trivial, as in the case of static backgrounds, or very complex, for example in driving footage. The result of a background retrieval algorithm is a foreground mask where every background pixel is set to a value of zero, as in Figure 2.1. Non binary masks are also possible with uncertainty encoded into the intensity values of the pixels. Noise, moving objects in the background, shadows and slow movement or even stopping of targets, are the main issues in background subtraction [26].

In controlled environments with static backgrounds, subtracting a background image will yield reasonable segmentation, through which every non-background element will be present, including shadows.

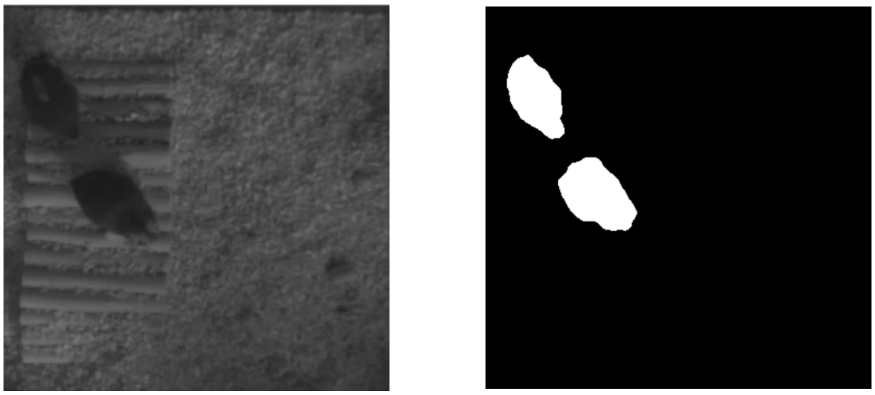
Different background subtraction schemes are popular. The main ones are based on estimating a Gaussian mixture model (GMM) for each pixel, and then calculating the probability of the pixel belonging to the background or foreground in each frame [27] [28] [29]. A mixture model assumes that a pixel has a combination of values coming from different Gaussian clusters, which are estimated by using expectation maximization (similar to k-means clustering) [24]. In the case of foreground to background subtraction, pixel clusters are estimated by using a window of frames, and then a pixel is classified as background if it has a high probability of belonging to the largest clusters, and vice versa. Pixel brightness and color are used to create the clusters.

The main differences between GMM algorithms are the update equations of the model parameters, such as the number of clusters, and the learning rate at each pixel,which can be a global or local parameter [30].

2.3.2. PRIMITIVE DETECTION

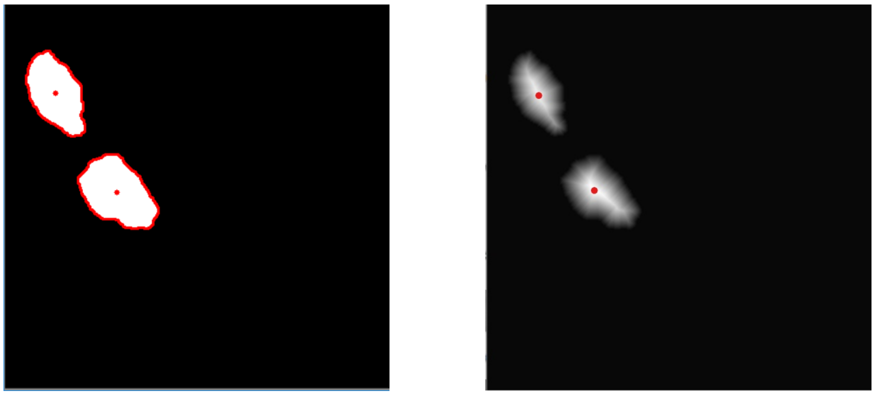
After background subtraction, the detector fits some descriptor to each individual target. Descriptors can be as simple as coordinates or bounding boxes [14] or as complex as multiple geometrical primitives (like ellipses) [31] or contours [11].

Bounding boxes are efficient, easy to implement and quite robust. They are very common, and are fitted to some form of either blob or contour detection. More complex shapes, such as ellipses are fitted by employing neural networks [32], or, if possible, by employing algorithms such as the elliptical Hough transform [33].



(a) Original image (b) Foreground mask

Figure 2.1: Background subtraction



(a) Countours and their centroid (b) Distance transform and its peaks

Figure 2.2: Centroid detection in a binary image

CENTROID CALCULATION

Several techniques are employed to find the center of a detected object. In a situation such as the one in Figure 2.1b, where the objects are clearly differentiated, finding the contour of the shapes, and then the centroid is sufficient. Finding contours of binary images is done by following and linking pixels that can be considered borders (white pixels in the neighbourhood of black pixels) [34]. The centroid of the curve can then be calculated by taking the moments [35], where if the image $I(x,y)$ has only values at the contour, the ij moment is defined as:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y), \tag{2.3}$$

and the centroid as:

$$\{x,y\} = \left\{ \frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}} \right\}. \tag{2.4}$$

A different approach is to take the distance transform of the binary image and then find its peaks. This method, if constrained to do so, finds the desired number of centroids even when the blobs overlap and only one contour is found. The distance transform image is a transformation where every pixel inside a shape is mapped as its distance to the closest boundary pixel [36]. Figure 2.2 shows both methods applied to the foreground mask in Figure 2.1b.

2.4. PREVIOUS WORK

ANIMAL tracking in lab environments for behavioural research is a well studied problem in computer vision, as the complexity of the task is beyond the capacity of human inspection [31]. A number of commercial and public solutions exist, with mice being the most common animal to study.

The appearance model of mice in rodent trackers is created for both the best tracking performance and to gather cues for behavioural inference. The direction of the head, shape, and position of the mice are important in both behaviour and tracking, so mice are commonly described by one [12] or several shape primitives [31] (commonly ellipses) and a vector storing the direction of the head [32].

Detection in rodent trackers is highly dependent on the complexity of the appearance model and the resources employed. A simple solution, which requires a limited amount of computing power and no input from the user, is contour detection. This is done after some preprocessing with the goal of background removal and difference highlighting [32]. More cues aid in the segmentation process, such as depth information [37], subcutaneous RFID tags [38] or some kind of fur marking [39]. The main drawback of such approaches is their limitations regarding their ability to be generalized [12].

To overcome low generalization, convolutional neural networks (CNNs) have started becoming more prevalent for the detection step [12] [40] [41] [42]. CNNs come with their own set of problems. They require a high number of training samples that need to be gathered and labeled by the user. Also, although they work well with a single subject, when extending them to several the number of errors quickly grows [40]. All CNNs require a training step before using, and this takes time as well as a lot of GPU processing power. After training though, detection can be achieved real time.

Tracking approaches differ widely. Neural networks have been employed [12], physics modeling [31], [37], Kalman filtering [40] among other solutions. Table 2.1 summarized the differences in the current state of the art and most cited papers regarding mice tracking.

Interesting behavioural data for medical research includes time spent at different regions of the enclosure (such as dark corners or social areas), contact events between mice (body, nose to nose, nose to tail), grouping of mice (following each other or mating) and self grooming [31]. Simple vector operations and geometrical constraints characterize all of these behaviours with small numbers of mice. As the numbers grow, the behaviour of following each other becomes complex to gather, similar to the grouping of pathlines in vector flow visualizations [43], where machine learning clustering techniques are useful.

The main issues in multiple rodent tracking are occlusion and identity switches due to proximity. Because of the nature of interaction between mice, no environmental constraints can be set to handle these problems, so robust trackers are needed. To handle occlusion, detectors are either trained on the shape of two mice overlapping or have complex occlusion handling routines included.

No current solution maintains the identity of the mice between trials, meaning that individual mice have to be re-tagged manually after each trial to preserve their identity.

2.5. CONCLUSION

In this chapter, the multiple object tracking problem has been presented both in a formal mathematical way (2.2.2) and through a high level overview of its different components (2.2). Figure 2.3 schematizes this description, showing both the approaches and components of MOT.

Multiple object tracking is a cross cutting field where aspects of image processing, machine learning and graph theory are combined. Adding behavioural inference further magnifies the complexity of the task, as descriptors do not just need to be efficient at tracking but have to be useful for this application.

Currently, open problems include multiple camera scenarios, 3D object tracking, scene understanding, and general purpose adaptive systems [1]. As MOT matures, so will rodent behaviour trackers: multiple cameras and 3D tracking have already been included in state of the art rodent tracking systems [37].

For the current solution, the observation model employs an object descriptor based on a bounding box of the body of the mice, keeping with the requirements for behavioural inference described in the previous section (2.4).

An online detector has been chosen. For the online tracking, Kalman filtering of a bounding box enclosing the mouse has been selected, based on the work done in Simple Online Real-time Tracking (SORT) [14]. This high dimensional descriptor of the target is robust to the frequent occlusions.

The nature of the system allows for periodical error detections, and an offline heuristically built error corrector is used to match previously known correct estimations with

Paper	**Detector**	**Behaviour**	**Tracking**	**Notes**
An unsupervised learning approach for tracking mice in an enclosed area [32].	Active contours.	Positional and angular data.	Maximum overlap between shapes of successive frames.	Includes an annotation software for matlab.
Automated measurement of mouse social behaviors using depth sensing, video tracking, and machine learning [37].	Image segmentation based on the depth information. Each segmented animal at each frame was fit with an ellipse.	Random decision forest (TreeBagger).	The identities of the animals were determined by their fur colors (black vs. white).	Multiple cameras: Top, depth, side.
Computerized video analysis of social interactions in mice [31].	A set of geometrical primitives describes the mouse, along with some constraints and physical forces that are solved by a physics engine.	Position, speed, previous behaviour and orientation.	Prediction by computing the instantaneous displacement vector of each primitive and applying it to find the new position.	They give the viewing angle of each mouse (-120º - 120º) and calculate the time the other mice are inside it.
Robust mouse tracking in complex environments using neural networks [12].	Neural network to segment the mouse from the background, different network to fit an ellipse to the mice and get the direction of the ellipse.	The solution only gathers positional data.	Neural net predicts the ellipse values in the current frame.	Only one mouse at a time. They test all other solutions and conclude theirs is the most general purpose one.
Deep Lab Cut [41].	CNN for bottom up estimation of the pose	Only positional and pose tracking	Kalman filtering of each skeleton node	A lot of emphasis on detection but not much information about tracking.
SLEAP [40].	CNN for bottom up or top down estimation of the pose	Only positional and pose tracking.	Optical flow or Kalman filtering	On mice the best results were with the bottom up detector and Kalman filtering.

Table 2.1: Some of the current open-source mice tracking solutions.

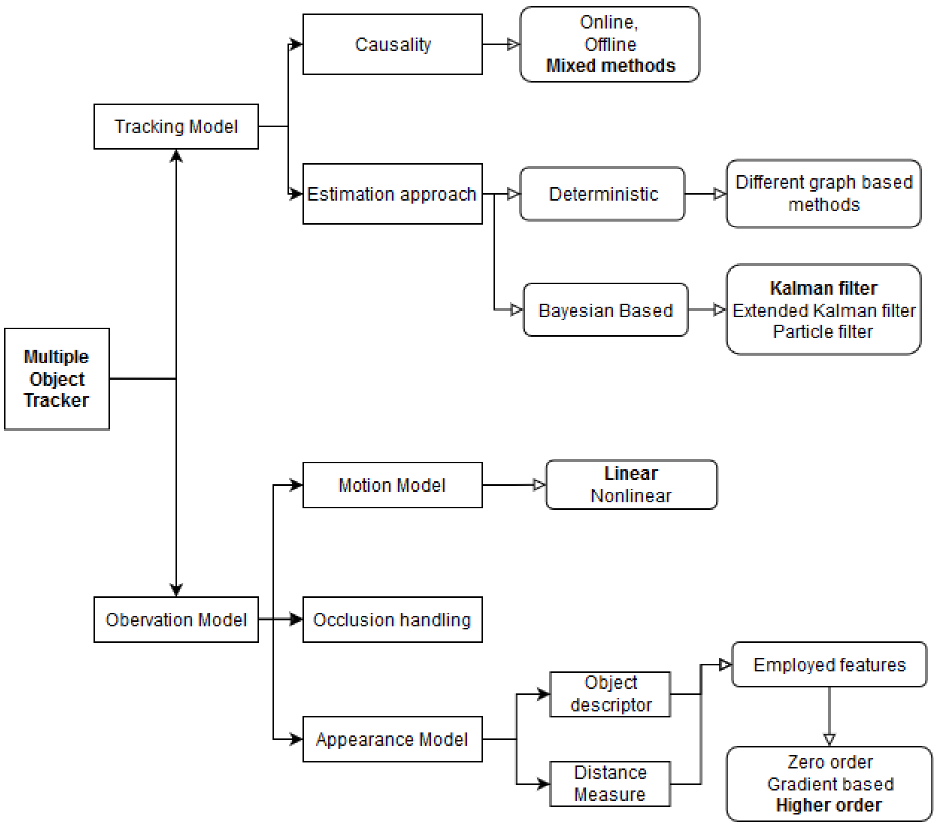


Figure 2.3: Summary of the components of a multiple object tracker (with right angles and filled arrows) and the different approaches to them (with rounded corners and white arrows). In bold, the approach taken in this work.

the current error detected one, as well as discard any incorrect tracks that are not possible to correct.

REFERENCES

[1] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, Xiaowei Zhao, and Tae-Kyun Kim. Multiple object tracking: A literature review, 2014. arXiv:1409.7618.

[2] L. Fan, Z. Wang, B. Cail, C. Tao, Z. Zhang, Y. Wang, S. Li, F. Huang, S. Fu, and F. Zhang. A survey on multiple object tracking algorithm. In *2016 IEEE International Conference on Information and Automation (ICIA)*, pages 1855–1862, 8 2016.

[3] C. Tomasi and T. Kanade. Shape and motion from image streams: a factorization method. *Proceedings of the National Academy of Sciences of the United States of America*, 90(21):9795–9802, 11 1993. 11607434[pmid].

[4] Jianbo Shi and Carlo Tomasi. Good features to track. *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 600, 03 2000.

[5] M. Rodriguez, S. Ali, and T. Kanade. Tracking in unstructured crowded scenes. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1389–1396, 09 2009.

[6] Cheng-Hao Kuo, Chang Huang, and Ramakant Nevatia. Multi-target tracking by on-line learned discriminative appearance models. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 685 – 692, 07 2010.

[7] Weiming Hu, Wei Li, Xiaoqing Zhang, and Stephen Maybank. Single and multiple object tracking using a multi-feature joint sparse representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 37:816–833, 04 2015.

[8] D. Riahi and G. Bilodeau. Multiple object tracking based on sparse generative appearance modeling. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 4017–4021, 09 2015.

[9] Dennis Mitzel, Esther Horbert, Andreas Ess, and Bastian Leibe. In *Computer Vision – ECCV 2010*, pages 397–410, 09 2010.

[10] Hamid Izadinia, Imran Saleemi, Wenhui Li, and Mubarak Shah. (mp) 2 t: Multiple people multiple parts tracker. In *ECCV*, 10 2012.

[11] Natan Peterfreund. Robust tracking of position and velocity with kalman snakes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21:564 – 569, 07 1999.

[12] Brian Q. Geuther, Sean P. Deats, Kai J. Fox, Steve A. Murray, Robert E. Braun, Jacqueline K. White, Elissa J. Chesler, Cathleen M. Lutz, and Vivek Kumar. Robust mouse tracking in complex environments using neural networks. *Communications Biology*, 2(1):124, 2019.

[13] A. Milan, L. Leal-Taixé, K. Schindler, and I. Reid. Joint tracking and segmentation of multiple targets. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5397–5406, 06 2015.

[14] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. 02 2016. arXiv:1602.00763.

[15] Bo Yang and Ram Nevatia. Multi-target tracking by online learning of non-linear motion patterns and robust appearance models. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1918–1925, 06 2012.

[16] Jerome Berclaz, Francois Fleuret, Engin Turetken, and Pascal Fua. Multiple object tracking using k-shortest paths optimization. *IEEE transactions on pattern analysis and machine intelligence*, 33, 09 2011.

[17] S. Challa, M.R. Morelande, D. Mušicki, and R.J. Evans. *Fundamentals of Object Tracking*. Cambridge books online. Cambridge University Press, 2011.

[18] D. Mitzel and B. Leibe. Real-time multi-person tracking with detector assisted structure propagation. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 974–981, 11 2011.

[19] Andrei Vatavu, Radu Danescu, and Sergiu Nedevschi. Stereovision-based multiple object tracking in traffic scenarios using free-form obstacle delimiters and particle filters. *IEEE Transactions on Intelligent Transportation Systems*, 16:1346–1351, 10 2015.

[20] Mohammad Azari, Ahamd Seyfi, and Amir Rezaie. Real time multiple object tracking and occlusion reasoning using adaptive kalman filters. *2011 7th Iranian Conference on Machine Vision and Image Processing, MVIP 2011 - Proceedings*, 11 2011.

[21] M. Isard and J. MacCormick. Bramble: a bayesian multiple-blob tracker. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 34–41 vol.2, 07 2001.

[22] Kenji Okuma, Ali Taleghani, De Freitas, J.J. Little, and David Lowe. A boosted particle filter: Multitarget detection and tracking. In *Computer Vision - ECCV 2004, Lecture Notes in Computer Science*, volume 3021, 05 2004.

[23] Bi Song, Ting-Yueh Jeng, Elliot Staudt, and Amit K. Roy-Chowdhury. A stochastic graph evolution framework for robust multi-target tracking. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 605–619, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[24] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[25] Rafael Gonzalez and Zahraa Faisal. *Digital Image Processing Fourth Edition*. Pearson, 06 2018.

[26] Tibor Trnovszký, Peter Sýkora, and Róbert Hudec. Comparison of background subtraction methods on near infra-red spectrum video sequences. *Procedia Engineering*, 192:887 – 892, 2017.

[27] Pakorn Kaewtrakulpong and Richard Bowden. An improved adaptive background mixture model for realtime tracking with shadow detection. *Proceedings of 2nd European Workshop on Advanced Video-Based Surveillance Systems; September 4, 2001; London, U.K*, 05 2002.

[28] Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction. *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, 2:28–31 Vol.2, 2004.

[29] Andrew Godbehere, A. Matsukawa, and Kenneth Goldberg. Visual tracking of human visitors under variable-lighting conditions for a responsive audio art installation. In *Proceedings of the American Control Conference*, pages 4305–4312, 06 2012.

[30] Kalpana Goyal and Singhai Jyoti. Review of background subtraction methods using gaussian mixture model for video surveillance systems. *Artificial Intelligence Review*, 50, 01 2017.

[31] Fabrice Chaumont, Renata Coura, Pierre Serreau, Arnaud Cressant, Jonathan Chabout, Sylvie Granon, and Jean-Christophe Olivo-Marin. Computerized video analysis of social interactions in mice. *Nature methods*, 9:410–7, 03 2012.

[32] Jakob Unger, Mike Mansour, Marcin Kopaczka, Nina Gronloh, Marc Spehr, and Dorit Merhof. An unsupervised learning approach for tracking mice in an enclosed area. *BMC Bioinformatics*, 18:272, 05 2017.

[33] P.S. Nair and A.T. Saunders. Hough transform based ellipse detection algorithm. *Pattern Recognition Letters*, 17(7):777 – 784, 1996.

[34] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32 – 46, 1985.

[35] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8(2):179–187, 02 1962.

[36] Azriel Rosenfeld and John L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13(4):471–494, 10 1966.

[37] Weizhe Hong, Ann Kennedy, Xavier Burgos-Artizzu, Moriel Zelikowsky, Santiago Navonne, Pietro Perona, and David Anderson. Automated measurement of mouse social behaviors using depth sensing, video tracking, and machine learning. *Proceedings of the National Academy of Sciences*, 112, 09 2015.

[38] Tatiana Peleh, Xuesheng Bai, Martien J.H. Kas, and Bastian Hengerer. RFID-supported video tracking for automated analysis of social behaviour in groups of mice. *Journal of Neuroscience Methods*, 325:108323, September 2019.

[39] Shay Ohayon, Ofer Avni, Adam L. Taylor, Pietro Perona, and S.E. Roian Egnor. Automated multi-day tracking of marked mice for the analysis of social behaviour. *Journal of Neuroscience Methods*, 219(1):10–19, September 2013.

[40] Talmo D. Pereira, Nathaniel Tabris, Junyu Li, Shruthi Ravindranath, Eleni S. Papadoyannis, Z. Yan Wang, David M. Turner, Grace McKenzie-Smith, Sarah D. Kocher, Annegret L. Falkner, Joshua W. Shaevitz, and Mala Murthy. SLEAP: Multi-animal pose tracking. September 2020.

[41] Alexander Mathis, Pranav Mamidanna, Kevin M. Cury, Taiga Abe, Venkatesh N. Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 21(9):1281–1289, August 2018.

[42] Jacob M Graving, Daniel Chae, Hemal Naik, Liang Li, Benjamin Koger, Blair R Costelloe, and Iain D Couzin. Deepposekit, a software toolkit for fast and robust animal pose estimation using deep learning. *eLife*, 8:e47994, 2019.

[43] Monique Meuschke, Samuel Voß, Bernhard Preim, and Kai Lawonn. Exploration of blood flow patterns in cerebral aneurysms during the cardiac cycle. *Computers Graphics*, 72, 02 2018.

3 DETECTOR

"Life is no way to treat an animal, not even a mouse"

Vonnegut Kurt

The detector module in a tracker is in charge of differentiating between individual targets and background. The complexity of the segmentation problem depends on the variability of the image environment and targets and the models used to describe them.

In the case concerning this Thesis, the detections used in the tracker are bounding boxes encompassing the back of the mice, with accompanying lights that are used in error correction. The bounding boxes are computed in two ways, depending on whether the mice are clearly distinguishable as separate blobs or not.

In the following chapter the appearance model choice and the reasoning behind it is described. The process through which the model is detected is detailed by first explaining the background removal step, and then the identification of each mouse.

3.1. APPEARANCE MODEL

MANY different appearance models have been proposed for rodents in behavioural tracking. They range from very complex, for example a connected graph skeletonizing the anatomy of the mouse [1] [2], to more simple geometric primitives, such as ellipses [3] [4].

Two factors influence the decision of how to build the appearance model: the cues needed for further analysis of the tracks, which is task specific, and the complexity of the underlying segmentation algorithm.

Current state of the art systems have very resource hungry detectors that require huge amounts of training data and gather lots of pose information about the targets. They work through CNNs that either first detect the individual mouse parts and then cluster them into different mice or first detect the individual mice and then segment each one into its parts (bottom-up or top down approaches).

Other trackers use more subtle methods based on shape matching, through active contours or ellipse fitting. In these cases, the trackers require either no training data or a smaller set of tagged samples, but the solutions lose in generalization [5].

The most important behavioural events that should be tracked when working with rodents are: self grooming, contact events (i.e. mouse A goes to mouse B and they touch nose to nose, mating), positional tracking (i.e. how much time has mouse A spent in the feeding area) and dynamic events (i.e. chasing) [4]. These events do not need a full description of the rodent's anatomy to be computed, as most require only positional data, angular data about the orientation of the mouse or shape characteristics (i.g. in the case of grooming) [6].

With the limited scope of the Thesis, the tracker only gathers positional behaviour statistics. That being the case, to keep with the goal of creating a sustainable and scalable solution, it must still be able to expand and encompass the necessary cues for the other commonly tracked behavioural events. A vector storing the direction of the mouse's head, the position of the head and of the center of the mouse, and a measure of the area of the mouse when it is alone are sufficient to compute all the mentioned events:

- Dynamic and contact events need the position and velocity of each mouse, gathered with the coordinates of the mouse's head and body.

- Grooming can be assessed through the area occupied by a lone mouse.

Beyond study-worthy features of the mice, to take full advantage of the error correction capabilities provided by the implants, the code should also be detected.

So, the descriptor chosen is one in which the mice are described by: **the current bit being transmitted by the LEDs, the area of their body, and the position of their center.**

3.2. DETECTIONS

To create the appearance model descriptor, several features of the mice are detected: a **contour of the light**, a **contour of the body** and a **bounding box** centered in the center of mass of the mouse. Though these are the detected items for a complete description of the mouse, an incomplete detection will still allow for tracking of the mice, as the tracking method only uses the bounding box. Incomplete detections are a frequent event, as explained later. Relying on full descriptions of the mice for tracking makes the sparsity of detections a problem when estimating the position of mice. The longer the mouse has not been detected, the bigger the uncertainty about its position will be.

To arrive at the descriptors from the detection, a few computations need to be done:

- **Light position**: Using the contour of the light, the position can be calculated as the centroid, calculated by using the first moments of the curve, as explained in equations 2.3 and 2.4.

- **Bit encoded**: To detect the current bit being encoded by the implant, the area occupied by the light is used. To calculate this area, again image moments are used, where M_{00} corresponds to the total amount of pixels inside the contour. The detected area is then processed by a classifier that decides whether it corresponds to a 1 or 0.

- **Area of the mouse**: Simply using M_{00} of the contour of the mouse gives the value of the area. Not only is this area useful for research purposes, but it can also identify misdetections or joint detections.

- **Centroid of the mouse**: First, the number of correctly sized contours is assessed. If it corresponds to the number of mice, then only one mouse is contained inside each contour, and the same process as with the light position is employed: the moments of the contour are used to find its center.

  If the number of detected contours is smaller, the larger contours are candidates for encompassing multiple mice. Using the distance transform, as explained briefly in 2.3.2, and more in detail in 3.4.2, the centroids of the incorrectly detected mice are found. The rest of the centers are calculated as usual, as each of the remaining contours has only one mouse inside.

- **Bounding box**: The bounding box around the centroid is also detected depending on whether the mouse's contour was clearly differentiated from the rest or not. In the first case, a bounding box enveloping the contour is generated. In the second case, when several centroids are detected in one contour, a square bounding box of fixed area is generated around the centroids. This simple approach is robust inside the tracker, and as the bounding box is not used for research purposes, accuracy of the feature itself is not important, while the accuracy of the tracker is.

STATISTICAL ANALYSIS OF LIGHT AVAILABILITY

The first idea when approaching the detection step of the solution was to employ only the lights, with a descriptor that would include head direction, encoded bit and position and that would not give the area of the mouse. This would result in a minimal loss of research data as the area is only used when identifying self grooming events.

The advantage of this approach is the simplicity of detection. The lights are very easy to identify, as they are captured as white by the camera in an otherwise gray and black image. Simple thresholding and morphological operations suffice to detect the lights. With such a simple detector, if the tracker worked correctly, more effort could be allocated to the behavioural analysis of the data.

So, a proof of concept Kalman filter tracker employing the simplified detections was built. Without the need of rigorous mathematical evaluation, the system was deemed inoperable. The time between detections was too long and therefore the estimations became completely wrong, as the linear movement model of the Kalman filter could not predict with certainty what the mouse would do in such large time periods. An example of the model failing can be seen in figure 3.1.

The failure was attributed to the time between detections, caused by frequent occlusions. The light was not detectable whenever the head of the mouse tilted, for example when digging or looking up or down. The position of the light atop the skull, as well as the anisotropic nature of the implant's radiation were identified as the main problems with the setup.

If digging were the main cause of the occlusion, by removing the bedding the enclosure, which removes the incentive to dig, would solve the problems, even though it would create a more artificial habitat for the mice.

A new position for the implants could be between the shoulder blades. To evaluate this, a mouse was fitted with the implants in this position and recorded. This video, along with two different videos of mice with and without bedding with the implants on their skull were statistically analyzed, with the results in the Appendix 8.1. The main takeaway from the analysis was that the light on its own is too unreliable to be the only cue of the tracker, as well as to be the only descriptor of the targets. Thus, a more complex approach was employed.

3.3. BACKGROUND REMOVAL

DETECTION is a data classification problem in which the targets are differentiated from the background and between each other. Therefore, the first step is foreground detection, where the background is subtracted from the image. The result of background subtraction is a foreground mask, a binary image that has intensity value zero in the background and maximum intensity in the foreground.

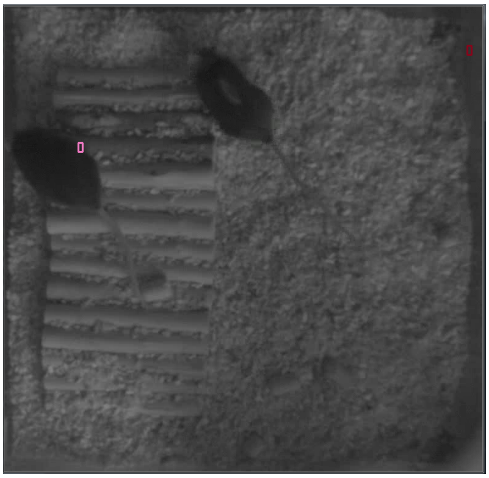


Figure 3.1: Example of the Kalman filter failing while the mice have their lights occluded if the observations are based uniquely on the lights. Notice the light's estimated positions (the pink and red rectangles) and how far they are from the head of the mice.

3.3.1. BACKGROUND ESTIMATION

The setup provides a fixed camera angle with a static background. Differences in camera angle or camera position between recordings made taking a picture of the enclosure without mice as the background not feasible. It would not be adequate for background removal, as these small differences would be presented as part of the foreground, or would distort the mask. So, the background must be decided through an estimation process dependent on the video.

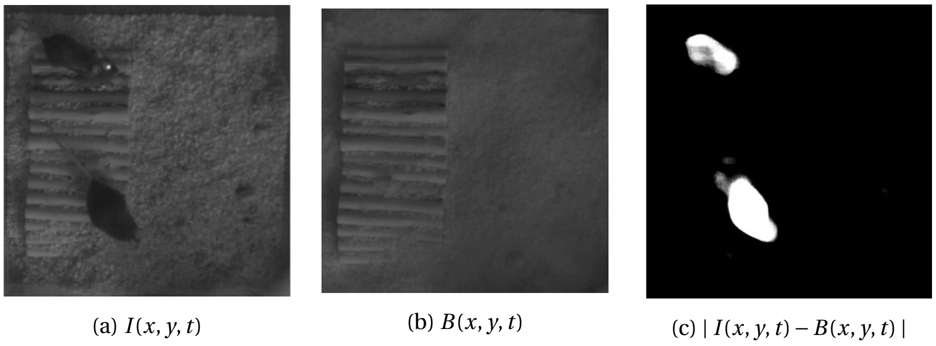
The approach employed was a median background subtraction model [7]. First an estimate of the background ($B(x,y,t)$) is found using median temporal filtering. This is then subtracted from each frame ($I(x,y,t)$). Lastly, a threshold (γ) to the absolute difference is applied to get the foreground mask:

$$|\, I(x,y,t) - B(x,y,t)\,| \geq \gamma, \tag{3.1}$$

where the background is found through spatial median filtering of a random subsampled subset of the video. The subsampling reduces buffer space and has been argued to provide an adequate background model [8]. Each background pixel is estimated as the median pixel of 100 random frames of the video sequence at frames t_i:

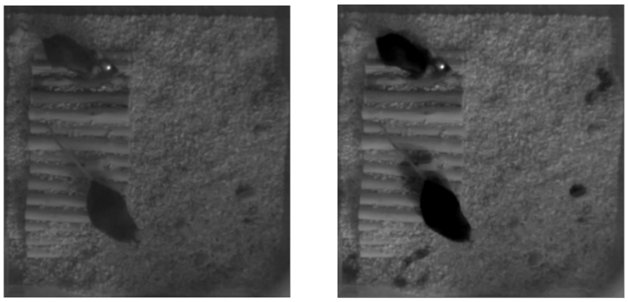
$$B(x,y,t) = \underset{i}{\text{median}}\{I(x,y,t_i)\} \tag{3.2}$$

Background estimation through median filtering is fast and simple, and works well in setups where the background occurs much more frequently than the foreground and is well behaved (reasonably constant). This is the case in the setup, though in very long videos changes in lighting conditions occur. The solution to this is to reestimate the



(a) $I(x, y, t)$ (b) $B(x, y, t)$ (c) $| I(x, y, t) - B(x, y, t) |$

Figure 3.2: Frame, calculated background and foreground



(a) Frame from the video (b) Darkened fur.

Figure 3.3: Example of the darkening of the fur through the subtraction of the foreground

background after 10000 frames, approximately 6 minutes. This keeps the subsampling factor in the order of magnitude required [8].

Figure 3.2 shows the mask gotten from a frame. Before applying the threshold, spatial mean filtering is used to remove noisy pixels due to bedding changes.

3.3.2. DIFFERENCE ADDITION

In figure 3.2c it can be seen that the foreground mask incorporates undesired elements such as spurious differences in the background or shadows. The element to be distinguished is the fur of each mouse, which is clearly darker than the rest of the image. With a simple thresholding of the frame it is not sufficient for differentiation of the mice and background, as other parts of the image can have the same darkness as the fur.

By taking the foreground mask, applying it to the image and then subtracting this "foreground" to the original frame, we will have a meaningful impact on the darkest foreground elements, notably the fur. It will effectively reduce its brightness, making it darker and easier to identify. In figure 3.3 the results of subtracting the foreground can be seen.

Instead of hardcoding a value to threshold this image and find the mice's fur, the user is prompted to select a mouse's fur at the beginning of each video sequence. Then, the mean value of this selection is taken as an approximate value for the fur's brightness. To account for the darkening process, the threshold is selected as 30% of the value that the user chose.

Algorithm 1: Calculation of the foreground mask

```
Result: Foreground mask
Data: User selection of fur and video
FurColor = mean(user selected bounding box);
if BackgroundNotInitialized then
    1.Choose 100 random $t_i$ form within the next 10000 frames.
    2. Calculate background pixels as:
    $B(x,y,t) = \underset{i}{\text{median}}\{I(x,y,t_i)\}$
    3.BackgroundNotInitialized=True
end
1. Calculate the foreground as $F(x,y,t) = \mid I(x,y,t) - B(x,y,t) \mid$
2. Add the foreground mask to the image to darken fur:
$I_d(x,y,t) = 1.2 \cdot I(x,y,t) - 0.2 \cdot F(x,y,t)$
3. Threshold with the chosen color:
$Mask = (I_d(x,y,t) > FurColor \cdot 0.3)$
4. if NframesAfterLastBackground>10000 then
    BackgroundNotInitialized = True
end
```

3.3.3. MORPHOLOGICAL OPERATIONS

To remove small artifacts from the mask, as well as to unify each mouse's blob, morphological operations are used.

In image processing, mathematical morphology is a non linear processing technique in which pixels of a binary image are evaluated with a simple kernel and conclusions are drawn on how this kernel fits or misses the shapes of the image. The two building blocks of morphology are erosion and dilation.

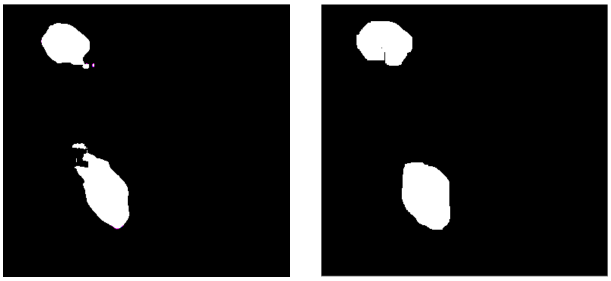
In **erosion**, the image A is eroded by kernel B as in equation 3.3, where B_z is the translation of kernel B by vector z and E is the Euclidean space in which image A is defined. For every pixel, B is superimposed to the image A. If B is completely contained by non-zero values of A, the pixel is retained, otherwise it is deleted.

$$A \ominus B = \{z \in E | B_z \subseteq A\} \tag{3.3}$$

Dilation is defined by equation 3.4, where image A is again transformed by kernel B, and A_b denotes the translation of A by b. For every non-zero pixel of A, kernel B is superimposed, and every pixel of the kernel is included in $A \oplus B$.

$$A \oplus B = \bigcup_{b \in B} A_b \tag{3.4}$$

By applying the same kernel B to first erode and then dilate an image A, we perform the operation of **opening**. Opening removes objects smaller than B with the erosion



(a) Mask before processing (b) Mask after processing

Figure 3.4: Morphological operations applied to a mask

while restoring the size and shape of the remaining objects (approximately) with the dilation operation. For the case at hand, closing followed by an additional dilation restores removes not only artifacts but holes in the detected blobs, as seen in figure 3.4

3.4. FINDING EACH MOUSE

WITH the foreground mask it is now the moment to distinguish between individual mice and create the detections required by the tracker. The data association problem now becomes that of finding the centroid of each mouse and a useful bounding box to feed to the tracker.

3.4.1. CONTOUR FINDING

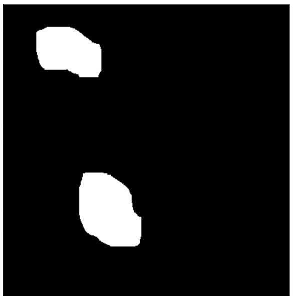
The structural outline of a shape in an image is identified in the process of contouring. It is helpful to identify the shapes of objects inside the image.

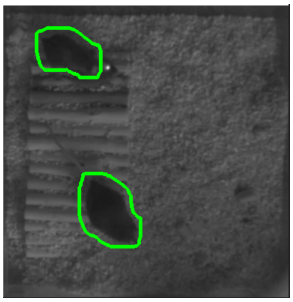
When a mouse is clearly differentiable from the rest (as in figure 3.4b) finding the contours of its shape is enough to achieve the characterization required. To find the contours of the binary image, the built-in function of OpenCV was used. The approach is based on the algorithm proposed by Satoshi Suzuki and Keiichi Abe [9], where pixels are identified not just as border pixels but they are ordered in a hierarchy based on the nesting of the borders (whether they are outer or inner borders). With the contour of a mouse, the bounding box calculation is trivial, and the cetroid of the mouse can be found using image moments (see 3.2).

3.4.2. DISTANCE TRANSFORM

When two or more mice overlap in the image (see Figure 3.6), using contours is not sufficient for identifying individual mice. The area of the contour *does* indicate with high certainty the amount of mice enclosed. By analyzing the shape of the contour and its area, it is possible to differentiate each of the mice.

In this case, the approach is reversed. The centroids of the mice are calculated first, and afterward a bounding box is generated using the average width of a mouse (Appendix 8.2 for statistics on the areas). The goal of the bounding box is not behavioural inference, but correct tracking. The difference between bounding boxes using the first


(a) Foreground mask


(b) Contours of the mask

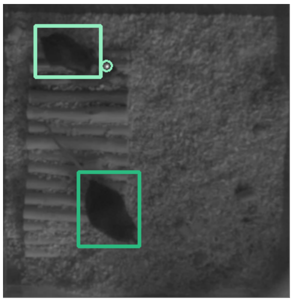
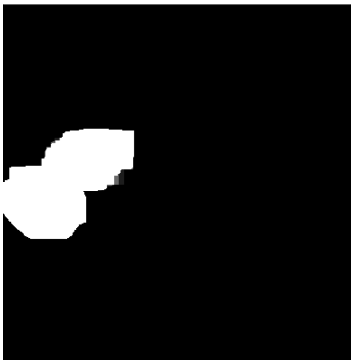
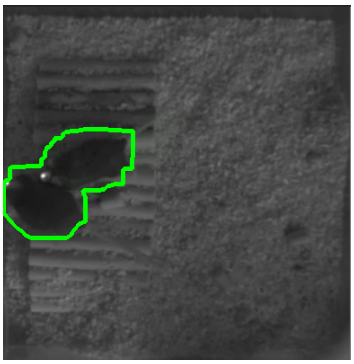
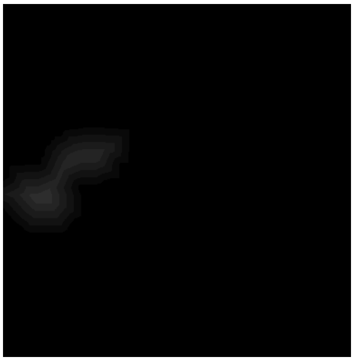

(c) Bounding boxes

Figure 3.5: Bounding box creation using contour detection


(a) Foreground mask


(b) Contours of the mask


(c) Distance transform

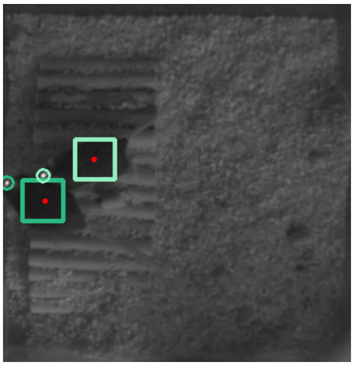

(d) Peaks and bounding boxes

Figure 3.6: Bounding box creation using contour detection

and second approach are designed intentionally to reduce identity switches when mice are overlapping, as explained in Section 4.2.1.

The distance function D on a binary image f associates each pixel x of the definition domain D_f of f with its distance to the nearest zero-valued pixel:

$$[D(f)](x) = \min\{d(x,y) \mid f(y) = 0\} \tag{3.5}$$

The distance transform will depend on the distance function employed and its domain. The typical choice is the euclidean distance, which is the one used in the implementation of OpenCV.

PEAK FINDING

The peaks generated by the distance transform correspond to the centers of overlapping objects in the image [10], in our case the centroids of the mice.

A peak of the distance transform image is a local maximum of the image. The python library scikit-image [11] uses a non linear maximum filter and then morphological dilation to merge the closest peaks. Due to the varying nature of the shape of mice together, more than one peak per centroid is usually detected, so processing of the peak coordinates must find the distinct centers.

RANSAC

The result of peak finding is a set of coordinates P corresponding to the peaks of the distance transform:

$$P = \{(x,y) \mid f(x,y) \equiv \underset{f}{\mathrm{localmax}}\} \tag{3.6}$$

Several peaks are usually detected per centroid of a mouse. To separate each center, the peaks are evaluated to find the subset of P with size N, corresponding to the number of mice, where each of the coordinates are as distant to the rest as possible. For the coordinates to be usable for tracking, though, they do not need to be the exact N points that satisfy the statement. It is only important that each point is in one distinct mouse. So, an approximate solution to the problem is sufficient.

To guarantee that each of the N points is in a distinct mouse, the function to be evaluated is the minimum distance between two of the N points of the subset. By maximizing the minimum distance, the two closest points of the subset will be in the centroids of the two closest mice of the blob. Then, all points in the subset are separated by at least the distance between the two closest mice, and thus are at different mice. Equation 3.7, where $d(x_i, x_j)$ is the distance between two points (chosen as the euclidean distance), summarizes this approach:

$$C \subseteq P, \quad C = \{x_1, ..., x_N \mid \max\{\min\{d(x_i, x_j) \mid \forall x_i \neq x_j; x_i, x_j \in C\}\}\} \tag{3.7}$$

Instead of solving 3.7 analytically, an iterative method is used: RANSAC. With this method, the peaks are sampled iteratively, each sample is evaluated and after a certain amount of tries, the best sample set is chosen.

In RANSAC, the parameters of a distribution are estimated by random sampling of the observed data. First, a model is fitted to a random subset of the data. Then, a cost function is evaluated for this model. This is repeated until the cost function reaches a certain threshold or a number of tries are attempted. If the latter is the case, the model with the lowest cost function is the selected model. Otherwise the first model to reach the threshold value is the selected model.

Applying RANSAC to the problem at hand, N coordinates are taken from the set of all peak coordinates. The distance between all of them is calculated, and the smallest one taken as the cost function. The subset selected will be the one with the largest minimum distance between all points after 50 tries.

BOUNDING BOX CREATION

With the centroids differentiated, a bounding box must be created around each one to feed to the tracker. This bounding box is generated as a square centered in the mouse's centroid with side length the average diameter of a mouse's body. This bounding box will be robust to the switch from the larger detected when using the full contour of the mouse, as explained in 4.2.1. In Appendix 8.2 the reasoning behind the area used when creating the bounding box is presented.

3.5. LIGHT DETECTION

THE brightness of the lights is such that nothing else in the video feed reaches that colour. Therefore, simple binary thresholding of the image is sufficient for the differentiation of the lights.

The close position of the LEDs inside the implants, together with the fact that the LEDs radiate quite diffusely make the differentiation of the two impossible. The light is detected as a single blob whether the bit coding light is on or off. Therefore, the *area* of the detection is what is used for code detection. This process is detailed in Chapter 5.

3.6. CONCLUSION

BEFORE creating an algorithm for the detection, it was interesting to analyze the videos to be able to generate a useful and accurate appearance model. This appearance model ended up as a bounding box centered in the center of mass of the mouse and the contour of the light. To arrive at the descriptor from a frame, two separate image segmentation problems were solved: mice and LED segmentation.

To differentiate the mice, the background of the video is calculated using median filtering. The result from subtracting this estimation to each frame is added to the original frame. This step darkens the fur of the animals, and with some adaptive thresholding the background is removed and a binary image generated. To separate the mice, first the contours of the thresholded image are found, and then their bounding boxes. If two mice are detected within the same contour, the contour is analyzed using the distance transform. The peaks of the function are found, and they are deemed the centers of each mouse. A fixed size bounding box is generated for these cases.

Light detection is as simple as thresholding and contour finding, as the brightness from the infrared implants is beyond anything else in the image.

With the detector completed, next is the selection of a movement model and tracking strategy using these cues to complete the tracker.

REFERENCES

[1] Talmo D. Pereira, Nathaniel Tabris, Junyu Li, Shruthi Ravindranath, Eleni S. Papadoyannis, Z. Yan Wang, David M. Turner, Grace McKenzie-Smith, Sarah D. Kocher, Annegret L. Falkner, Joshua W. Shaevitz, and Mala Murthy. SLEAP: Multi-animal pose tracking. September 2020.

[2] Alexander Mathis, Pranav Mamidanna, Kevin M. Cury, Taiga Abe, Venkatesh N. Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 21(9):1281–1289, August 2018.

[3] Weizhe Hong, Ann Kennedy, Xavier Burgos-Artizzu, Moriel Zelikowsky, Santiago Navonne, Pietro Perona, and David Anderson. Automated measurement of mouse social behaviors using depth sensing, video tracking, and machine learning. *Proceedings of the National Academy of Sciences*, 112, 09 2015.

[4] Fabrice Chaumont, Renata Coura, Pierre Serreau, Arnaud Cressant, Jonathan Chabout, Sylvie Granon, and Jean-Christophe Olivo-Marin. Computerized video analysis of social interactions in mice. *Nature methods*, 9:410–7, 03 2012.

[5] Brian Q. Geuther, Sean P. Deats, Kai J. Fox, Steve A. Murray, Robert E. Braun, Jacqueline K. White, Elissa J. Chesler, Cathleen M. Lutz, and Vivek Kumar. Robust mouse tracking in complex environments using neural networks. *Communications Biology*, 2(1):124, 2019.

[6] Jakob Unger, Mike Mansour, Marcin Kopaczka, Nina Gronloh, Marc Spehr, and Dorit Merhof. An unsupervised learning approach for tracking mice in an enclosed area. *BMC Bioinformatics*, 18:272, 05 2017.

[7] M. Piccardi. Background subtraction techniques: a review. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, volume 4, pages 3099–3104 vol.4, 2004.

[8] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati. Detecting moving objects, ghosts, and shadows in video streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1337–1342, 2003.

[9] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32 – 46, 1985.

[10] Pierre Soille. *Morphological Image Analysis : Principles and Applications.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[11] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

4

TRACKER

"He sees you when you're sleeping
And he knows when you're awake"

Santa Claus Is Coming To Town

After detecting the objects in a frame, they must be linked to the different targets from previous frames. This data association problem is solved through estimating the position of the targets and matching estimations to detections. So, both a movement model and a data association metric have to be chosen when designing a tracking algorithm.

In the case of the project, a Kalman filter was used to estimate the position of the mice in each frame, by using the previous position. To assign the detections, the estimations are compared to the detections using the intersection over union metric. The matching is solved by formulating the problem as an optimization problem of a bipartite graph.

This chapter first formalizes and gives an overview of the Kalman algorithm as applied to tracking, and then explains the motivation for the choice of the tracking method. Next, a summary of the previous work that this tracker is based on is given, as well as the implementation details and changes made to accommodate the problem.

4.1. SELECTING THE TRACKING METHOD

THERE are many approaches to the estimation of the position of the objects in a frame, as explained in Section 2.2.2. The selection of an appropriate estimation method for the tracker is linked to the descriptors generated from detections. Both components have to be chosen together as to maximize the performance of the solution.

In the literature of mice tracking solutions, several different approaches have been successfully employed (see Table 2.1). They mostly depend on the complexity of the descriptors and the availability of training data. Very detailed models of the targets allow for the use of complex physical movement estimations [1]. With large amounts of training data, neural networks can be employed to predict the positions and variations of the targets [2]. In between, allowing for a wide range of descriptors and requiring no training data, Kalman filtering is a very widespread approach in rodent tracking, as well as the general MOT problem.

4.1.1. KALMAN FILTER

The estimation method chosen was Kalman filtering. This method has low resource requirements, both in training data (not necessary) and computationally. It is easy to implement and allows for more complex descriptors than particle filters [3].

The simple implementation of Kalman filters allowed for different appearance models to be tested before committing to one. Additionally, knowing that other working mice trackers employ Kalman filters with a linear movement model, if the detections employed in the solution are appropriate, the estimations of the filter will hold.

The Kalman filter is an estimation algorithm that uses a sequence of noisy and inaccurate measurements (i.e. the detections) to produce estimates of variables (i.e. the position of the mice) using a joint probability distribution over the variables for each timeframe. It is recursive in nature, and estimates the internal state of a linear dynamic system using only the current measurement and the previous one. It first predicts the state (*a priori estimate*) using the previous estimation and some internal variables and then corrects this based on the current measurement (*a posteriori estimate*).

The assumption of a linear process and measurement model allows all computations to be done through matrix vector multiplication. The noise in the process and measurement are assumed to be additive Gaussian.

The prediction equations that govern the filter are:

$$\hat{\boldsymbol{x}}_k^- = F\hat{\boldsymbol{x}}_{k-1}^+ \tag{4.1}$$

$$P_k^- = FP_{k-1}^+F^T + Q, \tag{4.2}$$

while the update step is:

$$\tilde{\boldsymbol{y}}_k = \hat{\boldsymbol{z}}_k - H\hat{\boldsymbol{x}}_k^- \tag{4.3}$$

$$K_k = P_k^- H^T(R + HP_k^- H^T)^{-1} \tag{4.4}$$

$$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + K_k\tilde{\boldsymbol{y}}_k \tag{4.5}$$

$$P_k^+ = (I - K_kH)P_k^-. \tag{4.6}$$

In this notation, the hat operator, ˆ , denotes an estimate of a variable. So $\hat{\boldsymbol{x}}$ is an estimation of $\boldsymbol{x}$. The superscripts - and + are used for prior (predicted) and posterior (updated) estimates, respectively.

Equation 4.1 predicts the current state ($\hat{\boldsymbol{x}}$) evolving the previous updated estimate with the dynamics model (matrix F). The error covariance of this estimate is encoded in the matrix P, and predicted in Equation 4.2.

The update stage begins by calculating the measurement residual $\tilde{\boldsymbol{y}}_k$ (4.3), the difference between the true measurement z_k and the estimated measurement $H\hat{\boldsymbol{x}}_k^-$. H is the measurement matrix that encodes the relationship between the state to be predicted and what is actually measured. This residual is then multiplied by the Kalman gain (calculated as in Equation 4.4) to provide the correction to the predicted estimate, in Equation 4.5. Lastly, the error covariance is updated for the next step, Equation 4.6.

To initialize the filter an initial guess of the state estimate and error covariance matrix are needed. Commonly, initial ignorance is employed, meaning a large P_0^+ is used for quick convergence. The process noise (Q) and measurement noise (R) have to be estimated as well. These two are typically obtained from studies or measurements of the system.

In the case of Kalman filtering applied to video tracking, the measurement vector ($\hat{\boldsymbol{z}}$) is the position of the target: a vector described by the horizontal and vertical pixel locations at the center of the targets (u and v).

$$\hat{\boldsymbol{z}} \equiv [u, v]^T \tag{4.7}$$

For tracking, real motion can be viewed as linear over short time short time periods [4]. Thus, the dynamics model of the matrix F is typically chosen to be linear constant velocity model, where targets are modeled by the state:

$$\hat{\boldsymbol{x}} \equiv [u, v, \dot{u}, \dot{v}]^T \tag{4.8}$$

The measurement (Equation 4.7) is projected onto a state vector through the measurement matrix (H). No velocity can be calculated solely from the position, so it is left as zero:

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.9}$$

To evolve the state, the velocity is maintained constant while the locations are moved by their velocity multiplied by the time between frames (Δ_t), making dynamics model matrix:

$$F = \begin{bmatrix} 1 & 0 & \Delta_t & 0 \\ 0 & 1 & 0 & \Delta_t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$

ASSIGNMENT PROBLEM

Before the update step of the Kalman filter, each of the detections must be assigned to its corresponding track by looking at the distance to the estimated positions. This is a combinatorial optimization problem, as each detection cannot be individually assigned to a track. If not taking into account the previous assignments, collisions where several detections are mapped to the same track can occur. Therefore, all detections must be assigned at the same time. This is done by minimizing the sum of all distances between detections and their assigned track, which is analogous to the assignment problem:

Given two sets of equal size A and T, together with the weight function $C : A \times T \to \mathbb{R}$, find a bijection $f : A \to T$ such that the cost function $\sum_{a \in A} C(a, f(a))$ is minimized.

In the case at hand, A and T are the detections and the estimated position of each track, respectively. The cost function is the distance measure used to compare and evaluate similarity between the two.

A polynomial time algorithm to solve the assignment problem is the Hungarian algorithm [5], commonly used for the MOT problem [6] [7] [8], and readily implemented in most machine learning libraries.

4.1.2. KALMAN FILTER ON LIGHTS

As explained in Section 3.2, light detections are too infrequent to make reliable estimations of the path they take between detections. Figure 4.1 again shows the model failing. In Appendix 8.1, statistics detailing the frequency of occlusions of the lights are detailed.

Before gathering the statistics of light detections, a trial solution applying the simple Kalman filter with a linear movement model on the centers of the lights was built. The simplicity of the design allowed for a heuristic evaluation of such a solution to be a realistic possibility that required less time than data analysis of the recordings. Plus, it was a good first dive into the structure and logic of a MOT, building the experience required for the project.

The state vector was given by the position of the centers of each light, and the distance measure between estimations and detections used in the assignment problem was the euclidean distance between two points.

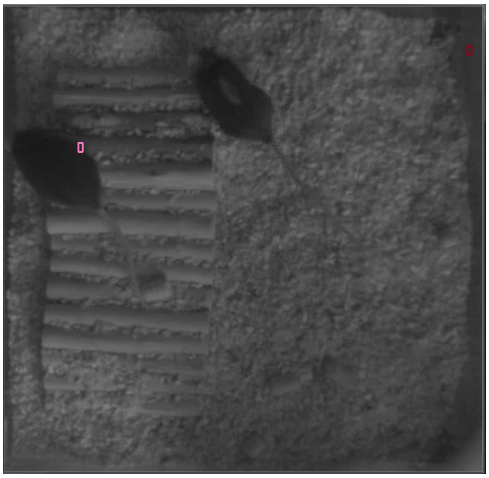


Figure 4.1: Example of the Kalman filter failing while the mice have their lights occluded.

The failure of the model was due to the dynamics model. A linear constant velocity can only hold in short time periods, and so with the frequent and long intervals of occlusion of the lights, the assumption fails.

The main takeaway from the first attempt was that an appearance model based on the lights would never be enough for a functioning MOT. So, more preprocessing was needed in the detector in order to identify mice with other cues apart from the LEDs.

4.1.3. KALMAN FILTER ON CENTERS OF MASS

The position of the centers of mass of the mice were identified as possible candidates for the state vectors of the Kalman filters. Following the steps detailed in sections 3.3 and 3.4, a rough detection of the center of mass of each mouse was possible in *every frame*. Thus, the frequency of detections of this variable would not be a problem.

The Kalman filter designed in section 4.1.2 was repurposed to work with these descriptors, with not very positive results. The centers of mass change position very sharply when the detector switches from contour centroids to distance transform peaks, that is, in contact events. This is the most sensitive moment in tracking. Identity switches are more likely to occur in such events, as it is when the detections are the closest. Outside of contact events, the tracker seemed fine, but the frequency of contact events is very high even when working with only two mice, as assessed statistically in Appendix 8.2, so it did not seem like the best option.

Another problem encountered was that mice can be very quick and unpredictably, and even at 30fps a linear movement model can fail. This is a common problem faced when encountering maneuvering objects in tracking [9].

A solution for the last problem is to break the tracks when the distance between detection and estimation is bigger than a certain threshold, and to create new tracks for unassigned detections. The LEDs of the system allow for the identification of the mice when decoding the sequence of blinks detected. If the tracks last for longer than the fre-

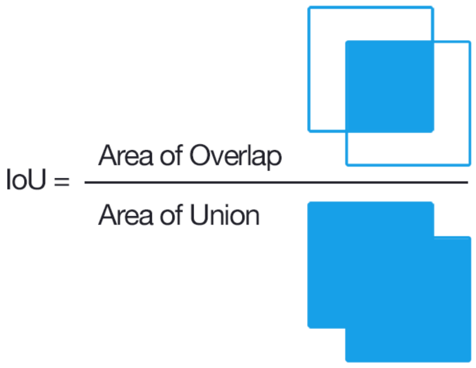


Figure 4.2: Visualization of the Jaccard Index between two bounding boxes.

quency of code detections, broken tracks can be reassigned with certainty to the correct subject, even when several tracks are discontinued simultaneously.

4.1.4. Simple Online Realtime Tracking

Following the experiences described in the previous subsections, it is clear that the euclidean distance between two coordinates is not the best distance measure for dealing with occlusions. Hence, some handling of misdetections when the movement model falls short is required. Bewley et al. proposed an elegant and pragmatic object tracker using both the Kalman filter and the Hungarian algorithm as building blocks. This was put into practice in Simple Online and Realtime Tracking [6] (from now on SORT).

The tracker was designed as a multipurpose tool, but with pedestrian tracking in mind. It only handles the estimation and assignment problem within MOT, and leaves the detector to be selected by the implementer. Detections become key here, as changing the detector can improve tracking by 18.9%.

The simplicity of the design comes not only from the estimation and assignment algorithms, but also from the descriptors. All features beyond the detections are ignored in the tracker, and the objects are described simply by a bounding box. The bounding box is evolved by the Kalman filter and the distance between estimations and detections are computed as the intersection-over-union, also known as the Jaccard index (Equation 4.11). This measures the ratio of similarity between sets as the size (defined as cardinality i.e. number of pixels) of the intersection divided by the size of the union of the sets 4.2.

$$d(A,B) = \mathrm{IoU}(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.11}$$

The choice of this distance metric implicitly handles short term occlusions by passing targets (in contact events). It does so as when a target is occluded by a moving target only the occluder is detected, since the IoU favors detections with similar scale.

SORT also tackles the problem of track handling. Having been designed as a multi purpose tool, the issue of deletion and creation of targets comes up when targets leave or come into the frame. In the setup this is not a possibility, but the limited capabilities of the movement model can be handled similarly. In SORT, tracks are terminated when they are not detected for T_{lost} frames. New trackers are created when a detection with an overlap less than IOU_{min} exists for a probationary period of T_{found} frames.

4.2. ADAPTING SORT TO MICE TRACKING

To begin with, SORT was implemented to work solely on the bounding boxes of the contours. The results looked promising but the amount of times the contours were not differentiated meant that contact events were not tracked at all. Discontinuation of the detections in contact events terminated the tracks. So the bounding box creation step was added to the detector using the distance transform peaks. A small change in the state vector and dynamics model was made to incorporate the particularities of mice body shape. Also, the tracker parameters IOU_{min}, T_{lost} and T_{min} were to be determined.

4.2.1. DETECTOR CHANGES

As previously stated, using the bounding boxes of the detected contours terminated the tracks when contact events occured. To solve this, a bounding box needed to be created during the contact event that would maintain the track identity using the IoU metric. The solution was to create a small bounding box centered in the centroid of the mouse. By making the bounding box sufficiently small, it does not overlap with the estimated bounding box of other mice, while it is fully contained inside the correct mouse's prediction. Therefore, the IoU of this bounding box is only nonzero with the mouse it corresponds to.

The bounding boxes also allow for tracking through contact events, as the centroids calculated using the distance transform are good detections of the mice position throughout the event. The centroids do have small variations in position that do not account for movement of the mouse but for the changes of the overall shape of the blob formed by all the mice in the contact event. These variations change the position of the bounding boxes by small amounts, and are handled correctly by the distance metric.

4.2.2. TRACKER CHANGES AND PARAMETERS

The state vector of the targets was defined as:

$$\hat{\boldsymbol{x}} \equiv [u, v, s, r, \dot{u}, \dot{v}, \dot{s}, \dot{r}] \tag{4.12}$$

Where (u, v) correspond to the center coordinates of the mouse's enclosing bounding box, s to its size and r to the aspect ratio. The aspect ratio is not constant as in the original paper as when mice stand up or curl into a ball they change substantially their shape. With the high framerate a constant velocity for the aspect ratio is appropriate.

No orientation was added to the bounding boxes as the changes in orientation when using this parameter are too abrupt. The causes for variation of orientation of the bounding boxes of the contours are not only changes in the orientation of the mouse, but changes in its posture. The contour of mouse standing up resembles a circle, and the orientation of the bounding box in this position changes frame to frame as many similar bounding boxes can fit the shape.

To be able to reidentify targets, the creation and deletion of tracks must happen less frequently than the detection of the blinking sequence of each mouse. To evaluate the performance of different parameters on the tracks period, several videos were fed to the tracker and the number of tracks and their length were noted. Being the constant velocity model a poor predictor of the true dynamics of targets, T_{lost} could not be too long, but it must also allow some leeway otherwise the codes would not be detectable. The parameters employed where $IOU_{min} = 0.1$, $T_{lost} = 45$ and $T_{min} = 10$.

4.3. CONCLUSION

AFTER having selected an appearance model and created a detector, the tracking strategy was developed. Kalman filtering was deemed the best choice as it is simple in its implementation, robust to errors and has low resource requirements. Several ways of approaching the tracking were tested, by using different movement models and descriptors. In the end, an adaptation of SORT was used, where not just the centers of the targets are evolved by the estimator, but full bounding boxes. This allows better distance metrics regarding occlusion.

Still, the movement model choice was not complex enough as to handle all of the types of motion patterns of mice. To overcome this, the tracks were discontinued and restarted. For this solution to work, tracks must be reidentified quicker than they are generated, by using some sort of code detection scheme. The development of this is what follows.

REFERENCES

[1] Fabrice Chaumont, Renata Coura, Pierre Serreau, Arnaud Cressant, Jonathan Chabout, Sylvie Granon, and Jean-Christophe Olivo-Marin. Computerized video analysis of social interactions in mice. *Nature methods*, 9:410–7, 03 2012.

[2] Brian Q. Geuther, Sean P. Deats, Kai J. Fox, Steve A. Murray, Robert E. Braun, Jacqueline K. White, Elissa J. Chesler, Cathleen M. Lutz, and Vivek Kumar. Robust mouse tracking in complex environments using neural networks. *Communications Biology*, 2(1):124, 2019.

[3] Mohammad Azari, Ahamd Seyfi, and Amir Rezaie. Real time multiple object tracking and occlusion reasoning using adaptive kalman filters. *2011 7th Iranian Conference on Machine Vision and Image Processing, MVIP 2011 - Proceedings*, 11 2011.

[4] L. Fan, Z. Wang, B. Cail, C. Tao, Z. Zhang, Y. Wang, S. Li, F. Huang, S. Fu, and F. Zhang. A survey on multiple object tracking algorithm. In *2016 IEEE International Conference on Information and Automation (ICIA)*, pages 1855–1862, 8 2016.

[5] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[6] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. 02 2016. arXiv:1602.00763.

[7] D. Nandashri and P. Smitha. An efficient tracking of multi object visual motion using hungarian method. *International journal of engineering research and technology*, 4, 2015.

[8] B. Sahbani and W. Adiprawita. Kalman filter and iterative-hungarian algorithm implementation for low complexity point tracking as part of fast multiple object tracking system. In *2016 6th International Conference on System Engineering and Technology (ICSET)*, pages 109–115, 2016.

[9] S. Challa, M.R. Morelande, D. Mušicki, and R.J. Evans. *Fundamentals of Object Tracking*. Cambridge books online. Cambridge University Press, 2011.

5 CODE DETECTION

"I wanna turn those blue lights into strobe lights
Not blue flashing lights, maybe fairy lights"

Jorja Smith, Blue Lights

The proposed system has one main advantage: the possibility of error detection and correction *automatically*. This is thanks to the individual identification of each mouse by a code that is transmitted by the IR lights in the implants. To be able to identify each mouse, a system that detects and keeps track of the blinking of each implant is needed.

The system presented detects the lights by thresholding the raw camera frames, and assigns each light to the closest mouse by solving the assignment problem. Each target has a classifier that determines whether a detection corresponds to a one or a zero, by analyzing the area of the light detected. This classifier is based on a Gaussian Mixture Model so as to not hardcode the value correspondence between detected area and code, allowing for variation between mice. The light detections are then analyzed and compared to the available codes, and fed to a system that handles the mislabeling and errors.

The following chapter first explains how lights are detected and assigned to targets. Then, the classification paradigm used to discern bit values is detailed. Lastly, the processing of the detection sequence to arrive at the bit code encoded is presented.

5.1. LIGHT DETECTION AND ASSIGNMENT

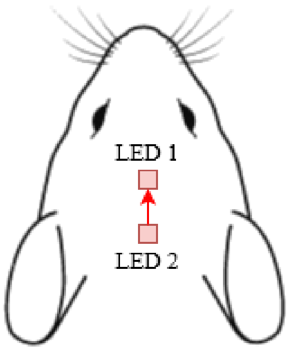
THE setup does not have any direct source of light aimed at the camera except for the implants. This makes the IR light from the implants the brightest element detected by the camera with a very big margin. The IR lights reach the saturation capacity of the detector of the camera [1] when aimed directly at the sensor, and are therefore encoded with a maximum pixel value.

Because of this clear differentiation between the bright light pixels and the rest of the frame, simple thresholding of the image is sufficient to separate the lights from the background. The available videos of live mice with implants were analyzed to gather data to be able to choose an appropriate threshold. The brightness values of the lights varied between mice and within the mice due to the orientation changes of the implants. So, instead of hardcoding a value, each frame is thresholded by using its maximum brightness and values close to it, with a minimum possible value in case no lights are clearly visible in the image. This minimum was selected as the minimum value encountered while analyzing the videos for light brightness values.

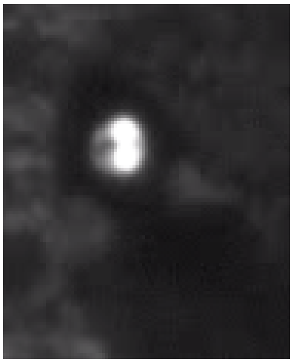
Figure 5.1 shows the light placement on a mouse's head and the result from the camera. As it is evident, the resolution of the camera is not high enough to differentiate the two lights in all orientations, so it is best to analyze the light detection as a single blob. To guarantee that the lights are located in a single blob in detection, an opening morphological operation is performed to the thresholded image. The results can be appreciated in figure 5.2.

As explained in 3.2, the lights are better placed in between the shoulder blades. The same logic applies to detecting the lights when placed there, but orientation changes are less prevalent.

[1]The camera is a Basler acA2000-165umNIR USB 3.0 camera with the CMOSIS CMV2000 CMOS sensor, with a typical saturation capacity of 11.8 ke^{-} [1].

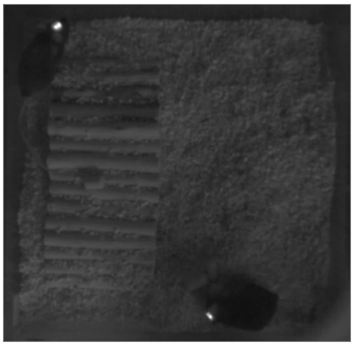


(a) Head placement of the lights

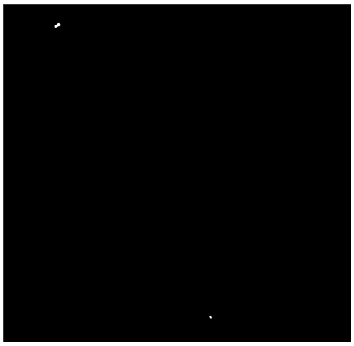


(b) Light detection with the head implant

Figure 5.1: Original head placement of the lights and the detection



(a) Raw frame



(b) Thresholded frame

Figure 5.2: Example of the light detection adaptive thresholding.

5.1.1. LIGHT ASSIGNMENT

The problem of assigning the lights to the different mice can be formulated as the assignment problem previously discussed when assigning detections to estimations (4.1.1). Each mouse and each light is a vertex on a fully connected graph. The lights edges have weights corresponding to the euclidean distance between the lights centroid and the calculated centroid of each mouse. The issue is to find the matching of mice to lights that minimizes the sum of distances between the two sets. As with the other assignment problems, the Hungarian algorithm is employed to solve the problem.

5.2. CODE DETECTION

AFTER assigning the light detections to the different mice, the goal is to be able to distinguish what code is being transmitted by each mouse. These are unique 8 bit codes that are broadcast at 15Hz. By storing previous code detections and comparing the detected code to the previous codes, it is possible to identify and correct mistakes in the tracking.

5.2.1. CODES

The codes employed are permutationally unique, meaning that not only are each of the 8 bit codes different from each other, but their permutations as well. So, another way to look at it, is that each of the mice are represented by not just one 8 bit code, but also for its 7 unique permutations, as any of the 8 possible codes are different to the rest.

This gives rise to a very interesting property of the system: any 8 bit detection is sufficient to identify a mouse, there is no need for synchronization of the detections.

To be able to assign the permutations together, a lookup table is employed, where the permutations generated by the same code all share the same key.

5.2.2. BLINK CLASSIFIER

Two LEDs are present in each implant (as seen in figure 5.1). One of the two lights always stays on, while the other blinks according to the unique sequence at 15 Hz. So if the implant is visible there will always be a light detection, but the area will be larger when a 1 is being transmitted.

Up until this point, each target is stored in the system by a sequence of frame data consisting of the contour of the fur, the bounding box around it, its centroid and the contour of the light. To be able to distinguish the value of each bit, each target must also have a classifier that assigns the light contours to values of one or zero.

Even though the LED lights in the implants are the same between mice, when analyzing the videos it was evident that there was variation between mice in the area that is detected for each bit. This can be attributed to the differences in the skin on top of the implant, the depth of the implant and the darkness of the fur. So, no fixed value can be set to distinguish between a 1 or 0 detection, it must be individual for each mouse.

Because of this, a classifier must be trained in order to identify the boundary between a zero and a one for each mouse. The chosen classifying paradigm is based on a Gaussian mixture model (GMM).

GAUSSIAN MIXTURE MODEL

A mixture model is a probabilistic model that describes the subpopulations of a distribution with a probability density distribution. It clusters data into classes described by probability distributions. A GMM models each probability distribution as a linear combination of Gaussian distributions, which are estimated with available data.

When creating the model, the objective of GMM is to maximize the likelihood ($p(X)$) of the available data X with regards to the parameters of the K Gaussians (mean and covariance matrix). By assuming a mixture of K Gaussians to have generated the data, we can write the likelihood as the marginalized probability, summed over all K clusters for all data points:

$$p(X) = \prod_{i=1}^{N} p(x_i) = \prod_{i=1}^{N} \sum_{k=1}^{K} p(x_i|c_k)p(c_k) \tag{5.1}$$

To maximize this likelihood, the *expectation-maximization algorithm* is employed, an iterative method to find local maximum estimates of probability distributions.

After creating a model, it can be used as a classifier. Each new data-point is labeled as belonging to the subpopulation with the highest posterior probability. This is known as *soft classification*, as it not only assigns a class to a new data-point, but returns the probability of it belonging to each class.

GMM was chosen over the similar k-means clustering algorithm for its soft classification property, as in the future development of the project the probabilities of each blink detection could be employed to make the code identification more robust.

In the project, each track has its own GMM classifier, which is trained after 2 continuous code detections (32 continuous frames where a light was assigned to the track). With the model, the 32 detections are labeled and the codes are processed to assign the identity of the mouse. After training, new light detections are evaluated with the model to distinguish whether they correspond to ones or zeros.

CODE PROCESSING

Every 16 continuous detections of light contours, a code can be computed (as the frame rate is twice the blinking frequency and the codes are 8 bits).

Because the blinking is twice the frame rate, two light detections encode one bit, and with 16 frames the code has to be analyzed by pairs. There is no guarantee that the sequence being processed started at the same time as a bit started to be emitted so it is possible for the pairs to be shifted. A short processing step is needed taking all of this into account and combining the pairs of bits into the transmitted sequence. The code used and a brief explanation can be found in the appendix (8.4).

Whenever a sequence is detected, the track notifies an error correcting module that handles all targets and stores target-track assumptions. In this module errors are detected and verified tracks are stored into memory.

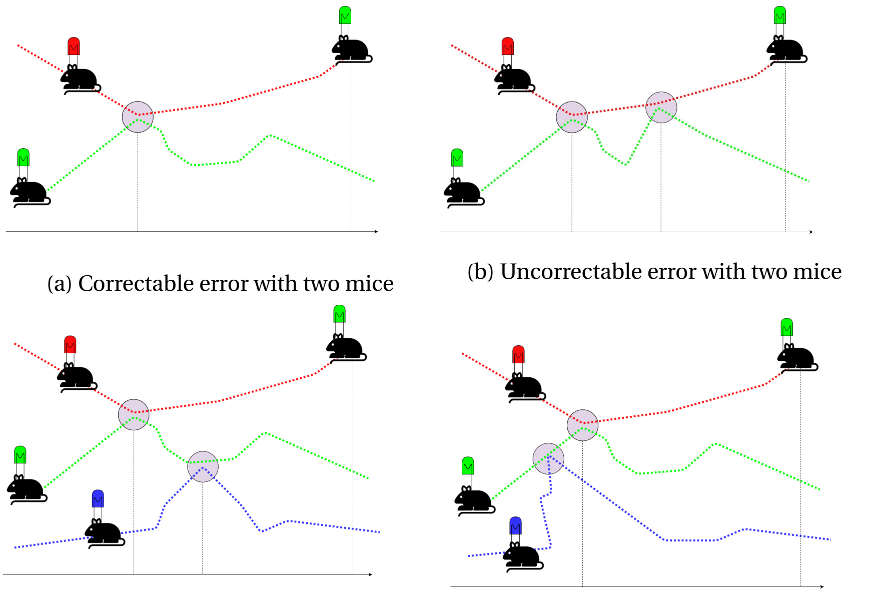
5.3. DATA STRUCTURE

THE handling of code detections and correcting of the stored trajectories is made up of three building blocks: the classifier, a track object and a track handler.

The classifier is an implementation of the GMM classifier algorithm using the model of sklearn [2]. The classifier takes a sequence of detected areas to fit the model to and then predicts the probability of a class according to the estimated distribution.

Track objects are instances of tracks (each mouse) that have their own classifier and the historical data of the light.

The track handler is the module in charge of verifying the codes and trackers, and handling miss-matches between expected code to be detected and detected code. It has a memory buffer where all tracks are stored with their expected code. Each time a code is detected in one of the tracks, the handler is notified. If there is a miss-detection the incorrect tracking data is discarded (here is where error correction is to be implemented).



(a) Correctable error with two mice
(b) Uncorrectable error with two mice
(c) Partly correctable error with three mice
(d) Fully correctable error with three mice

Figure 5.3: Examples of possible code miss-matches. The x axis is a time axis, while the y axis shows the displacement of the mouse (simplified to one dimension). Different codes are shown by the different colors, contact events are shown with a circle, and the mouse with the LED icon represents a code detection of the corresponding light.

When a track is verified with the code detection, its correct historical data is transferred to the main memory, which at the end of the execution is stored in a separate file for future processing.

TRACK HANDLER

The track handler is the event handler in charge of reacting to code detections. It has two memories, a buffer with the tracks between code detections and a main memory where correctly identified tracks are stored. When a new track is identified through a code detection, this pairing is added to the buffer. When the track updates its code, if it coincides with the previously detected code, the data between the detections is transferred to the main memory. If there is a miss-match, or if a new track is detected with a code that is in the buffer, all buffered data is considered wrong and discarded. When miss-matches between expected code of a track and detected code occur is where error correction is to be implemented. In the prototypical implementation of the solution, it was deemed not necessary, as the complexity of error correction is significant. The tool would be evaluated while discarding the non confirmed tracks and the frequency of these would determine the necessity of error correction.

Nevertheless, to reach the conclusion that error correction was too complex to pursue, some high level analysis of errors was performed.

ERRORS AND CONTACT EVENTS

The first step toward creating an error correction scheme is to identify the types of errors that exist and which errors can be handled by the proposed system. Errors can be separated into lost tracks, misdetections and identity switches.

Lost tracks occur when a track is not detected through several frames. The system, explained in 4.2.2, breaks tracks and creates new ones whenever a track is not detected or its distance to its estimated position is too large through several frames. So, the tracker already takes care of these errors.

Misdetections take place whenever a new track is detected where there is no track. If these erroneous tracks do not merge into a true track, they will never have a code detection, and therefore they will not be verified and moved into the track handler.

An **identity switch** happens when tracked objects come into contact and their trackers are changed. Contact events are very common in rodents, so it is these type of errors that are the most interesting.

Identity switches can be corrected trivially by the system if they are simple enough (Figure 5.3a). This means that the switch must occur in a contact event between two mice, and none of the mice involved are involved in another contact event before the miss-match is notified to the track handler.

When introducing more mice, errors of identity switches start to increase in complexity exponentially. To think of the complexity of the task, the set of all trackers in the buffer can be seen as a directed graph, where contact events and code detections are nodes, and the edges always point in the opposite direction of the passing of time. A track can be stored into memory if only one path can be found between the last confirmed identity and the new code detection. This way, the more mice, and unavoidably the more numerous the contact events, the more complex the graph and the more difficult the analysis. Figure 5.3 shows examples with up to three mice.

5.4. CONCLUSION

THE tracking system does not incorporate into its movement model all of the complexities of mice motion. This means that it fails in limit cases, where the mice move very abruptly. To overcome this, tracks are discontinued and restarted. New tracks are reassigned to their targets by employing the novel infrared implants. The implants blink certain unique patterns that identify the mice. They are distinguished frame by frame and assigned to their closest mouse.

To detect the patterns being emitted, each mouse is fitted with a Gaussian mixture model classifier that distinguishes blinks as either ones or zeros. After a sequence, some preprocessing is done to arrive at the encoded bit pattern. A buffer stores the tracks between code detections, and whenever a track is verified (its code corresponds to the expected mouse), it is stored into the main memory of the program.

This concludes the prototype of the tracker, and next is testing to numerically assess the functionality of the solution.

REFERENCES

[1] Basler AG. Basler ace aca2000-165umnir - area scan camera, Dec 2020.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

6
Evaluation

"Strictness, rigor and precision.
I can't afford to make a mistake
I develop a strong methodology
Inspired by old astronomy
I need conclusive evidence
Results need to be concrete"

Pupa Jim, My Research

The last step to attain the complete set of goals stated in 1.2.3 is to create a labeled dataset and evaluate the detection accuracy and precision of the tracker. This chapter explains the process.

The first section of the chapter discusses the resources that were available to work with through the project. Videos and the capacity to record new ones were limited during the process. A workaround found for this problem was to create artificial datasets that mimic real conditions, and then evaluate the project with them. The computer graphics software Blender was used in the creation of these videos and their corresponding datasets.

Next, the metrics employed are presented, as well as the reasoning for their choice. The metrics are selected for their capacity to judge the trackers precision in determining the location of the objects and to trace the correct trajectories through time.

The numerical results are shown, with regards to the number of mice in each video. A discussion of the results follows. In the discussion, each metric is analyzed in detail.

Lastly, the current work is compared to a state of the art tracking system, showing comparable metrics and differences in the approaches.

6.1. RESOURCES

THE usefulness of the evaluation of a system depends on the quality, quantity and variability of the data employed. In multiple object tracking, the datasets employed must be a significant representation of all of the different subject configurations and background conditions. This means it is necessary to evaluate the tracker with a varying number of targets, light conditions and backgrounds.

There were many limitations in this regard for the present work. The FlashTrack implants were fitted to mice and videos were recorded before the tracker was started. Because of this, some basic conditions were not kept constant throughout the videos: there was no fixed camera angle or position and no fixed enclosure conditions, the cages had toys and bedding in some videos that changed the behaviour of the mice and created more occlusions (by encouraging digging).

Another factor that influenced the outcome was the position of the implants. In Ap-

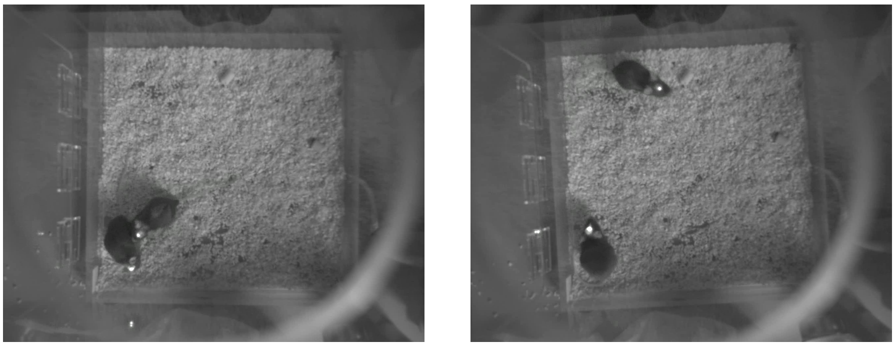
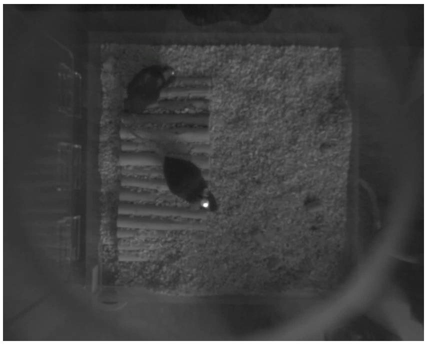
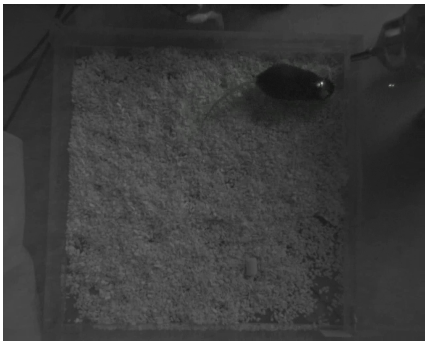


Figure 6.1: Example of camera angle variation within the same video.


(a) Two mice with bedding and a toy.

(b) One mouse with little bedding.


(c) One mouse with the shoulder implant.

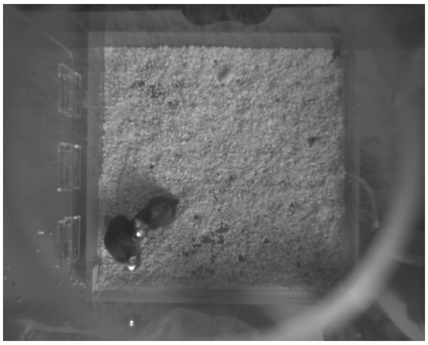

(d) Two mice without a toy.

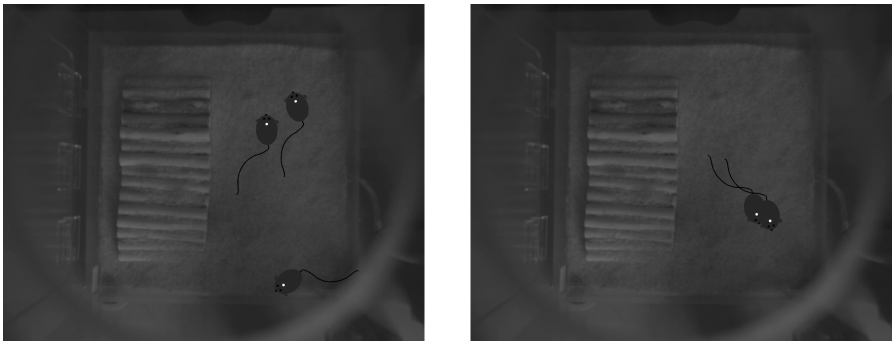
Figure 6.2: Examples of varied conditions between recordings.

pendix 8.1, the results of gathering statistics regarding the LED light position are presented. Because of the difficulty of generating new data, it was not possible to explore the conclusions of the study: that implants between the shoulder blades are easier to detect.

The process of generating new data is not trivial. Surgery is needed to fit the implant to the mouse. This makes changing the amount of mice or the position of the implant challenging.

All in all, 7 recordings of either one or two mice with very different lighting, camera angle and enclosure conditions were available. The total sum of recording time was just under 5 hours, but the quality of the recordings varied widely, even within the same video. The recordings were not tagged, meaning that if they were to be used for numerical evaluation, a previous video analysis and tagging step was needed.

These setbacks motivated the search for a different resource to recordings that allows for easier evaluation and fast video generation. The solution found was to generate artificial data.



(a) Example of following behaviour. (b) Occlusion in a synthetic video.

Figure 6.3: Two of the artificial videos.

SYNTHETIC DATA

Data not obtained by measurement is deemed synthetic or artificial. The use of synthetic data is widespread in machine learning [1], advances in generative adversarial networks are based on producing new data to train the model. Synthetic data is faster to produce, is generated tagged and can lead to good generalization to real data (transfer learning) as has been shown in some object detectors [2].

In the case at hand, the generation of artificial recordings is useful as it remedied most of the issues stated beforehand. It removes the need for surgery when changing the number of mice or position of implants. The data is ready for evaluation without the time consuming step of tagging. Furthermore, specific situations that can be difficult to find in recordings can be tested i.e. chasing, some kinds of occlusion or other limit cases.

To generate videos that emulated the desired conditions of the system, the computer graphics software Blender [3] was used.

The videos were generated by using the background of one of the recordings (as calculated using median filtering, explained in 3.3). A geometrical representation of a mouse was built on an axis that could be constrained to follow a path. Then, paths were created and linked, and the mouse object was made to follow them with different velocities. To emulate the behaviour of mice, events such as following, occlusion caused by contact and swift trajectory changes were added into the paths.

In addition to the geometrical representation of the mouse, another primitive was added to the axis object to create the implant lights. Blender allows for scripting in python, and this was used to make the LEDs blink with the codes through the video.

When rendering, each video was made to generate a json file with a python dictionary storing the normalized position of the axis of the mice frame by frame, to be used as the ground truth.

6.2. METRICS

SEVERAL evaluation methods have been proposed in order to assess the performance of multiple object trackers. Many kinds of errors can occur: identity swapping by switching objects, identity swapping by switching tracks, noisy detections, long intervals without tracking, spurious misdetections, etc.

Some metrics in the literature try to encompass all errors and offer very detailed insight into the tracker while losing easy interpretation [4]. In order not to clutter the evaluation process and offer a concise and understandable conclusion of the performance, the metrics chosen combine the different errors and behave according to human intuition. The multiple object tracking precision (**MOTP**) and multiple object tracking accuracy (**MOTA**) metrics proposed by K. Bernardin et al. [5] were chosen.

The case at hand makes correspondences between created tracks and objects trivial, as all targets appear in every frame and all tracks are identified with the code.

The *M*ultiple *O*bject *T*racking *P*recision (MOTP) is defined as the total position error, rhat is, the distance (pixelwise) between a track and object in every matched frame, averaged by the total number of matches made (Equation 6.1):

$$P = \frac{\sum_{i,t} d_{i,t}}{\sum_t c_t}. \tag{6.1}$$

The *M*ultiple *O*bject *T*racking *A*ccuracy is defined as:

$$A = 1 - \frac{\sum_t (m_t + f_t + k_t)}{\sum_t g_t}, \tag{6.2}$$

where m_t, f_t and k_t are different types of errors: misses, false positives and mismatches respectively, which are averaged by the total number of objects across frames (g_t).

Additionally, the average missed frames per target (**MF**) was chosen as a good indicator of the capabilities of the light assignment method and motion model. This indicator simply averages the amount of frames where the ID of a track does not have a stored position. As all mice are always in frame, whenever one is not in the tracker it means that either there was no detection of that mouse (detection error) or, more frequently, that the track corresponding to the mouse in that frame was not verified. The last case can be because there was an identity switch, because the light was occluded while transmitting a sequence or because the track was never validated as it lasted shorter than a blinking detection.

6.3. RESULTS

THE evaluation was performed with synthetic data. The videos generated had a length of two minutes. All of the limit cases were present: occlusion, chasing, mating, climbing up the walls and standing. Videos had one to six mice in the 30x30cm enclosure. The metrics for each case are shown in Table 6.1.

When too many mice are present in the small enclosure (more than 4) all indicators point to a sharp decrease in the performance of the tracker. With more mice, more interaction and thus more occlusion. If the enclosure is too small for the amount of mice, the occlusion can be frequent enough for the tracker not to be able to detect codes as quickly as the motion model fails and new tracks are created. This means that the mice tracks will not be verified and thus will be discarded with the current error processing.

A more detailed discussion of each indicator follows.

6.3.1. ACCURACY

As explained in 6.2, the accuracy of the tracker is measured using the MOTA metric. This approximates the intuitive understanding of accuracy with that of the stat. MOTA aggregates several types of errors: misses, false positives and mismatches. Its range is from 1, perfectly accurate, to 0, completely wrong. Figure 6.4 shows the evolution of the metric with regards to the number of mice.

With only one mouse, there is virtually no occlusion. Therefore the mouse is always detected and the only chance it has of its track not being stored into memory is if it breaks the movement model twice in quick succession, generating a short new track that is too brief to be verified.

When increasing the number of mice, the accuracy stays above 0.85. This is as long as the mice have enough space to move without occluding each other too frequently, which happens when there are five and six mice. If the threshold is surpassed, MOTA decreases sharply, reaching as low as 0.22 with six mice.

The exaggerated failure of the system with six mice is due to the size of the enclosure. Even with artificially small mice, such a number means that contact events are ever-present in the video (as seen in figure 6.5). Contact events are critical as it is where most mismatches occur. Most errors are due to misses, as if a track is not verified the current system discards it. This is evident when looking at the MF metric, as mice are not detected in more than 70% of the frames.

6.3.2. PRECISION

The precision of the tracker is only measured when a match between track and object is made. It measures the capabilities of the detector and movement model estimation as the distance between the stored position and the ground truth. A plot of this measure against the number of mice in a video can be seen in figure 6.7.

Up to five mice, the metric stays roughly constant (with a 10% margin) around 20 pixels. This discrepancy is approximately 8mm, with a variance between videos of only 0.6mm. The reason for this constant error is due to a discrepancy between how the centroid of the mouse is calculated and how the ground truth is stored. The ground truth has the position of the axis of each mouse collection of blender, which is slightly shifted with regards to the center of mass of the fur, which is the calculation of the tracker (see figure 6.6).

The mice created in blender have a length of 4cm, so the precision is around 20% of the length of a mouse, meaning that the centroids are on average inside the mouse. The precision can be deemed acceptable because of this.

With six mice, the precision falls drastically, to more than 250 pixels. As misses ac-

	One Mice	Two Mice	Three Mice	Four Mice	Five Mice	Six Mice
MOTA	0.9965	0.9229	0.8907	0.8662	0.6217	0.2261
MOTP	20.98	22.66	18.88	19.59	20.95	269.3
MF	0.34%	7.71%	10.91%	13.35%	37.82%	73.06%

Table 6.1: Benchmarks of the tool with different amounts of mice.

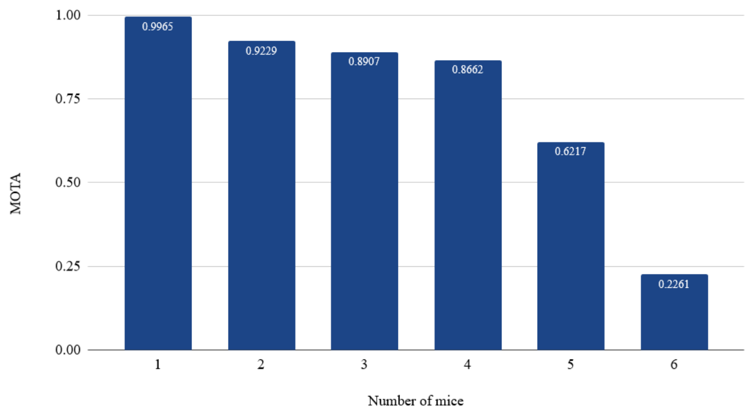


Figure 6.4: MOTA of the tracker for different amounts of mice.

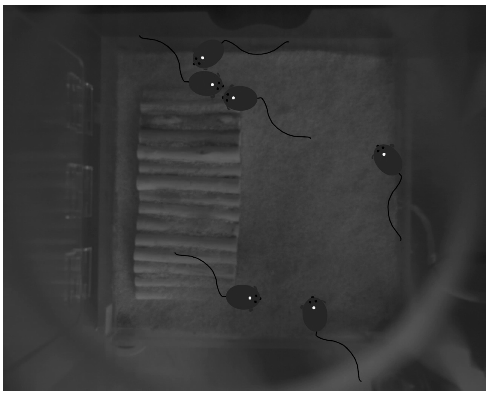


Figure 6.5: Six artificially small mice in the enclosure.

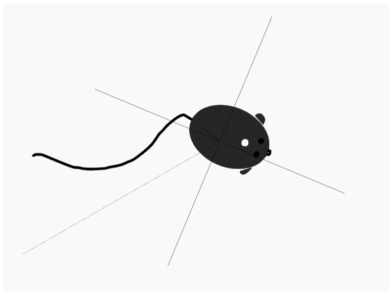


Figure 6.6: Shifted collection axis in Blender with regards to the center of the mouse's fur.

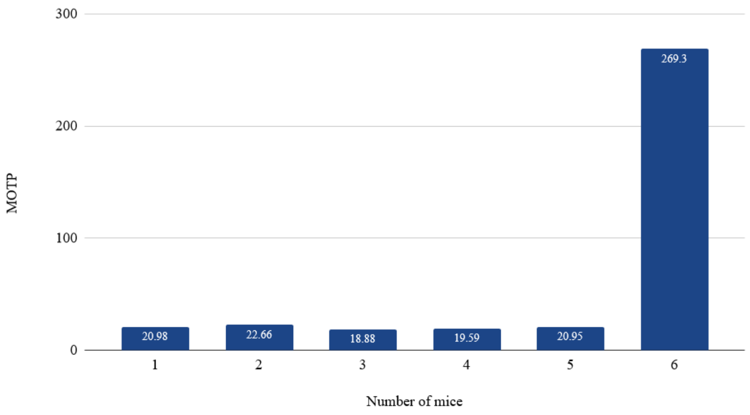


Figure 6.7: MOTP of the tracker for different amount of mice.

count to around 73% of the frames and the MOTA is 23%, there is a 3% error discrepancy that can be attributed to false positives that are verified through code detection. This has a very small chance of occurring, but as contact events become more frequent, so do these errors.

6.3.3. MISSED FRAMES

The amount of missed frames per mouse is negligible when only one mouse is present, as occlusion is almost non-existent and the track is always being verified.

When increasing the number of mice, and with it the amount of contact events, MF starts to increase as well. Contact events are the most sensitive of all limit cases, as mismatches can occur, as well as erroneous light assignments. With five and six mice, the frequency of contact events is such that mice spend more than 35 and 70 % of the time without a stored track. This is not acceptable, and so a more precise solution for mislabeled tracks should be found. Results for 6 mice should not be taken into account, as clearly the enclosure is too small to maintain such a population.

6.4. COMPARISON WITH SLEAP

THE solution is compared here to Social LEAP Estimates Animal Poses (SLEAP), a state of the art tracking framework for multiple animals using deep learning. A brief summary of SLEAP can be found in 2.4. As a multi-purpose tool, in its evaluation it was tested against several types of animals. One of them were mice, videos of two mice recorded in a cage with bedding at 40fps, 1280x1080p for 5 minutes.

6.4.1. TECHNIQUES

SLEAP is a general purpose animal tracking tool. It uses complex body-part descriptors for each animal. To create the descriptor, the researcher using SLEAP must first define

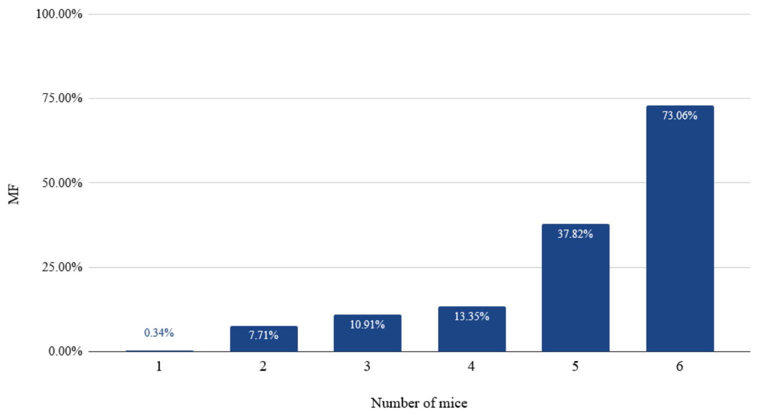


Figure 6.8: MF of the tracker for different amounts of mice.

the model and then tag a series of frames (1474 in their example case with mice). This is then fed into a deep learning algorithm to create the detector. For this last step, a powerful computer is needed in order to train the model.

In contrast, the work being presented uses a set environment and subject type, and thus can pinpoint the detector without the need for training data or powerful computers.

The descriptor is simpler, with the downside that pose information is not gathered, unlike SLEAP.

This work also maintains identity through tests, as the implants will continue to use the same codes throughout all experiments. In SLEAP, no such feature is present.

With regards to tracking, the framework proposes two movement models: linear Kalman filtering and particle filters. Both models are applied to each body part, and the chosen one depends on the type of animal. For mice they use Kalman. This solution is very similar in both SLEAP and this work. The main difference is in the complexity of the descriptors, as in this case Kalman filtering is applied only to bounding boxes instead of multiple descriptors.

6.4.2. METRICS

In SLEAP, precision is measured differently. This is because as their approach has more complex descriptors, they cannot be simplified into only one distance metric for MOTP. So, SLEAP uses a different metric (Object Keypoint Similarity). OKS has a domain of 0 to 1, where 1 is perfect precision. For the dataset with mice, they achieve only 0.535 mean average precision. This shows that overly complicated descriptors are not optimal with animals with blob like features. With the simple bounding box descriptor, precision was high in the work.

Before measuring the accuracy, the researchers proofread the tracking results, correcting the identity switches between the two mice. On average identity switches occur 1.26 times per minute. After proofreading, they achieve a 99.98% MOTA, 6% higher than in the current work, which is not proofread.

The amount of switches, as well as the need for manual supervision is not present in the current solution, albeit at a 6% loss of accuracy.

6.5. CONCLUSION

To assess the validity of the tracker, synthetic data was generated. Using this synthetic data, statistics were gathered of the performance with videos with different amounts of mice. The tracker is accurate and precise with a limited number of mice. It ends up holding its ground against an established solution such as SLEAP with two mice in the enclosure.

Problems begin when more mice are introduced into the enclosure. No error correction scheme was created and erroneous tracks are simply discarded. As contact events between mice increase, so do the amount of discarded frames, reaching amounts that make the tracker unusable with more than 5 mice. Clearly, a more nuanced error management approach is needed.

The external validity of the study is not high, as synthetic data is simpler than real data, with no shadows or pose changes. It is therefore clear that the statistics will decrease with a further necessary evaluation with real data, which is the next step in the development of the system.

REFERENCES

[1] N. Patki, R. Wedge, and K. Veeramachaneni. The synthetic data vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 399–410, 2016.

[2] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1278–1286, 2015.

[3] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam,

[4] Kevin Smith, D. Gatica-Perez, Jean-Marc Odobez, and Sileye Ba. Evaluating multi-object tracking. pages 36–36, 07 2005.

[5] Keni Bernardin, Alexander Elbs, and Rainer Stiefelhagen. Multiple object tracking performance metrics and evaluation in a smart room environment. *Proceedings of IEEE International Workshop on Visual Surveillance*, 01 2006.

7 Conclusion

Every real story is a never ending story.

Michael Ende, The Neverending Story

This last chapter gives a brief overview of the work done, a section on the contributions of the thesis and a discussion with future lines of research to continue working on.

7.1. THESIS OVERVIEW

THE objective of this thesis was to develop a multiple object tracker for mice with infrared implants that could detect errors and process them. The final goal of the project is to infer behavioural data from the tracker. This was taken into account in the design process of the solution.

Part of the project was dedicated to gathering data about positioning of the implants and enclosure conditions. This was only explored theoretically, as no videos were recorded following the conclusions of the study.

Various approaches were investigated for detecting the mice. The main issue was what descriptor should be chosen to attain a balance between tracking capabilities and behavioural cues. Finally, a simple bounding box based descriptor was chosen, discarding more expensive ellipse fitting methods.

Regarding tracking, Kalman filtering was deemed optimal for the task as it is simple to implement and has a good track record with animal tracking. Some adjustments were made to take into account the particularities of fixed camera rodent tracking.

To detect errors, the infrared implants came into play. Each mouse has a unique code with which it blinks for track validation. The codes were chosen to be permutationally unique in favour of other coding schemes as it made detections quicker with simple tools at the framerate required. The problem of error correction was studied but considered out of the scope of the thesis, so tracks were only validated or discarded. To evaluate the work, synthetic data was generated. Common multiple object tracker metrics were calculated using the ground truth, and the result was compared with other animal trackers.

7.2. DISCUSSION AND FUTURE WORK

This project concludes with an error detecting mice tracker with promising results in some particular conditions evaluated with synthetic data.

The generalization of the results of the project to real data is to be assessed. The external validity of the research is therefore questionable, and it is necessary to use the statistical analysis of the enclosure and light positioning to record real data with the correct conditions to evaluate the tool with.

With the positional data of the mice, it is possible to obtain behavioural indicators such as time spent in different areas of the enclosure, amount of contact events or dominance. In the work presented, none of these indicators are gathered, and it will be an important line of research to do so.

When the tracker was employed to analyze the behaviour of too many targets, the performance lowered considerably. The main reason for that was that if a track is not validated by the light codes, it is simply discarded. Graph theory based error correction schemes using the detected codes are possible, and an elegant albeit complex solution to the subpar performance.

Employing a GMM classifier to distinguish the codes generated by the implants associates a probability to each of the detections. This probability was built in to the tracker for the possibility of expanding the correction of the tracks by using some probabilistic method. In the current iteration of the solution, all code detections are treated as equally correct, but by using each blink certainty, the codes can have an associated likelihood score. This certainty measure can be used for a more refined error correction, where a very low probability mismatch between detected and expected code would need several repeated occurrences to be flagged as a mismatch.

8
APPENDIX

To avoid clutter in the main body of work, the statistical analysis of the videos of real mice are grouped here in the Appendix. For the same reason, the preprocessing code explained in 5.2.2 is presented here.

8.1. LIGHT POSITIONAL DATA

Three videos were analyzed in order to assess the value of the lights in both detection and error correction. The limited number of videos and varying conditions mean that the data is not fully generalizable, but the analysis still proved useful to the research, as it demonstrated that the lights on their own are not sufficient for detection, and the lights on the scalp have to much cadence and infrequent detections that even identification is an issue.

To analyze the three videos, a script run frame by frame checking whether the lights were detectable by thresholding. In the case of the video with two mice, the lights were assigned by the euclidean distance between the current and last detections. This data was gathered in a comma separated value file (csv) and then processed using pandas [1]. To interpret the data from the tables, the following is the legend:

- *Continuous frames with both mice*: The number of consecutive frames in which the lights of both mice were detected before one or both stop being detected.
- *Continuous frames with no mice*: The number of consecutive frames in which no light was detected.
- *Continuous frames with mouse A*: Number of consecutive frames in which mouse A's light is detected.
- *Continuous frames without mouse A*: The number of consecutive frames in which mouse A's light was not detected.
- *Continuous frames with mouse A*: Number of consecutive frames in which mouse A's light is detected.

Table 8.1: Data from an 80 minute video of two mice with the implant on their scalp and bedding

2 mice	Average	Stdev	Mode	# mode	% of mode	# >Code	% >Code	# breaks	Max
Cont. frames $\forall m_i$	5.8819	7.3876	1	1331	25.3041				94
Cont. frames $\nexists m_i$	7.5453	15.601	1	812	24.7108				321
Cont. frame m_0	16.632	31.078	1	644	14.7604	460	10.543	4363	611
Cont. frames m_1	15.373	32.859	3	1063	21.2218	371	7.4066	5009	842
Cont. frames $\nexists m_0$	16.2371	44.137	1	916	20.9899				1062
Cont. frames $\nexists m_1$	13.2672	34.760	1	1054	21.0379				1114
Frames between code m_0	42.8527	85.1182	15	2117	63.2317				1508
Frames between code m_1	40.3412	78.250	15	2315	65.1195				1447

- *Frames between code in mouse A*: Number of consecutive frames between detections of length 15 or more.
- *% > Code*: Percentage of continuous detections that have a length longer than the code's length.
- *% of mode*: Frequency of the mode in the distribution

TWO MICE WITH IMPLANTS ON THE SCALP (TABLE 8.1)

On average a mouse light is detected every 15 frames, though it has a huge variance, and it can stay undetected for 1000 frames (30 seconds).

The detections on average last 15 frames, exactly one code detection, but this also has a huge variance. Only 8% of detections last long enough to identify the code, meaning that 92% of detections are not valuable for detection purposes, and that detections usually occur in long continuous intervals.

The time between possible detections is on average 42 frames (more than 1s). This value also has a huge variance, as it is common for several detections to happen continuously (as illustrated by the mode of the time between detections being 15 frames). In this particular video, a mouse went for over 1500 frames with its light not being visible (almost one minute).

A wild mouse has a top speed of 3.6 m/s, so even if the mouse moves at 10% this speed, on the average time between detections (0.5s) it can move 18 cm, almost half of the enclosure. This is between detections, not between code detections. Between code detections, with more than 1s on average, the mouse is likely to move 40 cm, almost the length of the enclosure. The most common occurrences are very short detections (of 1 to 3 frames) followed by also short intervals of non detections (1 frame).

Each mouse, in total, has its light detected in a little over 50% of the frames. Because of the infrequency of the detections, again, 92% of this time is not usable to find the code. This means that the lights are only useful 4% of the time for detection purposes.

Table 8.2: Data gathered from a 20 minute video of one mouse with the implant in the scalp and no bedding

1 mouse scalp implant	Average	Stdev	Mode	# mode	% of mode	# >Code	% >Code	# breaks	Max
Continous frames with m	42.9941	84.7805	1	93	13.6764	301	44.26470588	680	825
Continous frames without m	18.5212	73.3755	1	142	20.85168869				1481
Frames between code	23.9728	94.4966	15	881	77.4846				2872

Table 8.3: Data gathered from a 30 minute video of one mouse with the implant between the shoulders.

1 mouse with shoulder implant	Average	Stdev	Mode	# mode	% of mode	# >Code	% >Code	# breaks	Max
Continous frames with m	38.2469	83.3969	1	178	15.6414	455	39.9824	1138	820
Continous frames without m	9.07550	18.8759	1	297	26.0755				327
Frames between code	21.5306	20.6073	15	868	76.2071				480

ONE MOUSE WITH THE IMPLANT ON ITS SCALP (TABLE 8.2)

The mouse is detected in just short of 70% of the frames. Of this, 44% are long enough for identification. The average time between detections is 23 frames, though it has a huge variance as the mouse can stay undetected for long intervals (up to 47 seconds). 77% of detections came right after a previous detection. The number of very short detections is low, 14%.

It is very important to note that in this setting, the mouse did not have bedding in the enclosure, limiting the amount of times it lowered its head.

ONE MOUSE WITH THE IMPLANT ON ITS SCALP (TABLE 8.3)

The mouse is detected in 80% of the frames. Of these detections, 40% are long enough for identification to happen. The average time between code detections is only 21 frames, 76% of detections coming right after another detection (those after 15 frames). The number of spurious detections is low, 15%.

The most important improvements with regards of the scalp case are:

- A 14% increase in frames detected ($70 \rightarrow 80$) even though the enclosure in this case did have bedding, meaning that even though the mouse tilted its head often, it stayed detectable.
- An 83% reduction in the maximum amount of time the mouse stays undetected ($2872 \rightarrow 480$).
- A 50% reduction in the average time the mouse spends undetected.
- A 78% decrease in the variance of both the time between detections and between code detections.

8.2. AREA OF A MOUSE

To calculate the area of a mouse, the image moments of the contour of a well differentiated mouse was used. All 6 videos with live mice were employed to arrive at an average of around 10000 pixels.

As a mouse is not round, but oval, with an average aspect ratio of 11:3, a bounding box centered on the mouse of 25 pixels was created to give certainty that it stays within the fur of the mouse.

8.3. CONTACT EVENT FREQUENCY

The average time mice spent together was calculated by counting the number of distinguishable blobs detected after the thresholding in the detector. As all videos analyzed had only 2 mice, if only one blob was detected, it would mean that a contact event was happening. This occurred on average 26% of the frames, with a deviation of between videos of 0.15. Three videos of 1.3 hours, 22 minutes and 30 seconds were used.

8.4. CODE FOR PROCESSING THE PAIRED BITS

The following code analyzes a sequence of bits detected at twice the blinking rate. Bits are therefore encoded in pairs. Because the sequence can start mid bit detection, this must be taken into account by checking whether the first and last bit are the same in case of an uneven number of equal detections in the beginning of the code. The code is as follows:

```
def process_code(code2process):
    if len(code2process) % 2 != 0:
        return
    code = []
    shifted_bit = False
    step = 1
    for bit_ind in range(int(len(code2process)/2)):
        bit_ind = 2*bit_ind
        bit_pair_first = code2process[bit_ind]
        bit_pair_second = code2process[bit_ind + step]
        if bit_pair_first == bit_pair_second:
            code.append(bit_pair_first)
        else:
            if shifted_bit:
                return
            if bit_pair_first != code2process[-1]:
                return
            if bit_ind > 0:
                for bit in code:
                    if bit_pair_first != bit:
                        return
            shifted_bit = True
            code.append(bit_pair_first)
            step = -1
    return code
```

REFERENCES

[1] Wes McKinney. Data structures for statistical computing in python. pages 51 – 56, 2010.

ACKNOWLEDGEMENTS

I have to start by thanking both of my supervisors for their incredible patience and support through this thesis. I am grateful to have worked with them.

Thank you to Farnaz for allowing me to work on your wonderful project and sticking with me through this difficult times.

Another big thank you to Alle-Jan for helping me overcome this process and giving me his fantastic feedback and support. I have not been the easiest student to work with and he was so patient, which I greatly appreciate.

I could have never arrived to this point without the help from my family, for whom I am very lucky, their support and kind words pushed me here.

I am very fortunate to have shared this experience, along with so so much more with my partner Celia. Thanks to her for being there for me. It is incredible to keep learning and growing with her.

I have to thank my shrink Patricia for her help in showing me the way forward and the tools I needed to stay in the state of mind that would allow me to work in this project.

My time in the Netherlands and Sweden was as good as the company I kept, which I mean in the best of ways. I am grateful to all of my friends for their help, I could not have asked for a better group of people to accompany me these years.