

TDK-thesis

Gergely Dinya

Multi-view instance tracking of swimming birds in a system of camera pairs with limited overlap

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPT. OF ARTIFICIAL INTELLIGENCE



Author:

Gergely Dinya

Computer Science MSc

1. semester

Supervisors:

Dr. Anna Gelencsér-Horváth

Research Fellow

Dr. habil. András Lőrincz

Senior Researcher

Budapest, 2024

Contents

List of Figures	2
List of Tables	3
1 Introduction	1
2 Related works	3
3 Methods	6
3.1 Data collection	6
3.2 Synchronization	7
3.3 Intrinsic camera parameters and distortion	8
3.4 Orientation estimation	10
3.5 Tracking	12
3.6 Eye-state extraction	17
4 Results and conclusion	19
Bibliography	27

List of Figures

1.1	The same scene looked at from four different cameras with the red box highlighting a manually re-identified goose.	2
3.1	The schematic overview of the aviary.	7
3.2	Illustration of the timestamp refinement. If 25 frames have the same stamp 10:00:00, they will receive the fractional timestamps with an increment of 40 milliseconds based on their relative order.	7
3.3	Illustration of the frame correspondence mapping. For each "tick" of the baseline synchronization signal, we pick the corresponding frames from videos A and B with the closest fractional timestamp. These frames are then handled as depicting the same moment in time.	8
3.4	Top row: The original picture and the same picture with the area discarded after undistortion marked with red. Bottom row: The undistorted picture. .	9
3.5	Top row: The original and the undistorted image. Bottom row: The predicted edge of the aviary, marked with red. A rendered mesh visualizing the predicted surface of the water.	10
3.6	A list of the 16 collar patterns used by the biologists. The similarity of certain patterns combined with the birds' black and white neck and chest plumage makes identification considerably more difficult.	13
3.7	The results of the pooling-pairing process visualized on top of a schematic overview of a segment of the aviary	15
3.8	An overview of the tracking pipeline's key components.	17
3.9	Visualization of the tracking output overlayed onto frames extracted from three neighbouring camera pairs depicting the same synchronized moment.	17
3.10	Illustration of the main steps of the eye-state extraction process.	18

List of Tables

4.1	This table shows a breakdown of the tracking pipeline’s output. For every batch, the accuracy of identification of the instances is over 99%, which ensures reliable tracking.	20
4.2	This table compares the averaged results of the DLC pipeline and the results of our method.	21
4.3	This table shows a breakdown of the eye-extraction pipeline’s output. The rows represent the goose images with a certain eye state. The columns represent the output classes and the number of images the pipeline classified as belonging to that class.	21

Abstract

Research in biology often requires data analysis on a scale that would be unfeasible or too slow without the use of automated approaches. In their research looking into the sleeping and group behaviour of wild geese, the biologists at the Max Planck Institute for Biological Intelligence have to annotate hundreds of hours of video footage that could take years using manual methods. Automated multi-animal, multi-view instance tracking in their aviary setup is challenging due to frequent occlusion of both the animals and their identifying features, the narrow overlap between camera pairs, and the semi-calibrated, semi-synchronized nature of the setup. These challenges within animal tracking differ significantly from those in human tracking, making existing solutions unsuitable for addressing the unique difficulties posed by the aforementioned setup. In this paper, I present a composite tracking system capable of overcoming these challenges. To combat the uneven frame rates, the videos are linked to a baseline synchronization signal using predicted fractional timestamps. An automated orientation estimation algorithm ensures precise position prediction on the surface of the water despite the shifting nature of the setup. Both visual and positional tracking elements are utilized to supplement the shortcomings of each approach, with deep neural networks employed on the visual side to detect, validate, and, when possible, identify the animals. Problem specific logic is applied, to determine the position of the birds and merge the detections from different cameras into a single global system, where they can be efficiently tracked. Finally, a post-processing step assembles the broken subtracklets into full-length tracks. The results show that the combination of these methods yields a highly precise and robust tracking algorithm, reducing the data processing time by an order of magnitude, moreover, only a small fraction of this reduced time requires human attention.

Chapter 1

Introduction

Visual data processing and annotation play key roles in the research process in fields like biology and behavioural ecology that often require the analysis of vast quantities of video and other visual data. The ability to track individual animals within this visual data can provide essential insights into their behavioural patterns and social interactions [1, 2, 3, 4, 5]. Traditional methods of animal tracking and observation, based on manual annotation, are labour-intensive and time-consuming, especially in environments where large groups of animals are observed in a multi-camera setup. As demands for detailed biological data increase, the limitations of manual methods highlight an urgent need for more efficient, automated tracking systems capable of handling the volume and complexity of observed data. Yet, accurate and consistent multi-animal, multi-view tracking, particularly with instances that share highly similar physical characteristics, remains a significant challenge.

Researchers at the Avian Sleep Group of the Max Planck Institute for Biological Intelligence are exploring the sleep patterns of migratory birds such as Canada geese, aiming to shed light on their unique ability to engage in unihemispheric sleep, while retaining control of both halves of their body. During the analysed sessions, a group of wild geese follow a boat going around in circles inside a floating aviary, while pairs of cameras film the animals from pillars on both sides. The birds follow the boat as they have been imprinted on the biologists from an early age. The lack of a strong motivator means that the number of birds in the observed flock fluctuates during the session. Tracking these animals as they swim and interact presents a considerable challenge. Beyond the ones posed by limited overlap, camera distortion, frequent occlusion of identifying features and the generally high visual similarity of the animals; achieving accurate timestamp

alignment and spatial positioning adds further complexity to data synchronization across cameras. Manual annotation quickly becomes unmanageable, while existing tracking solutions struggle with the setup. Moreover, any solution must be accurate enough to assess the open or closed state of each eye independently.

We address these challenges by proposing an advanced composite pipeline for multi-animal, multi-view instance tracking, specifically targeting swimming birds observed in a semi-calibrated, partially synchronized camera setup. Our approach utilizes both visual and positional tracking elements, to supplement the short-comings of each separate system and performs pre and post-processing on the data, to provide a robust solution to the problem.

The challenges are illustrated in Fig. 1.1.

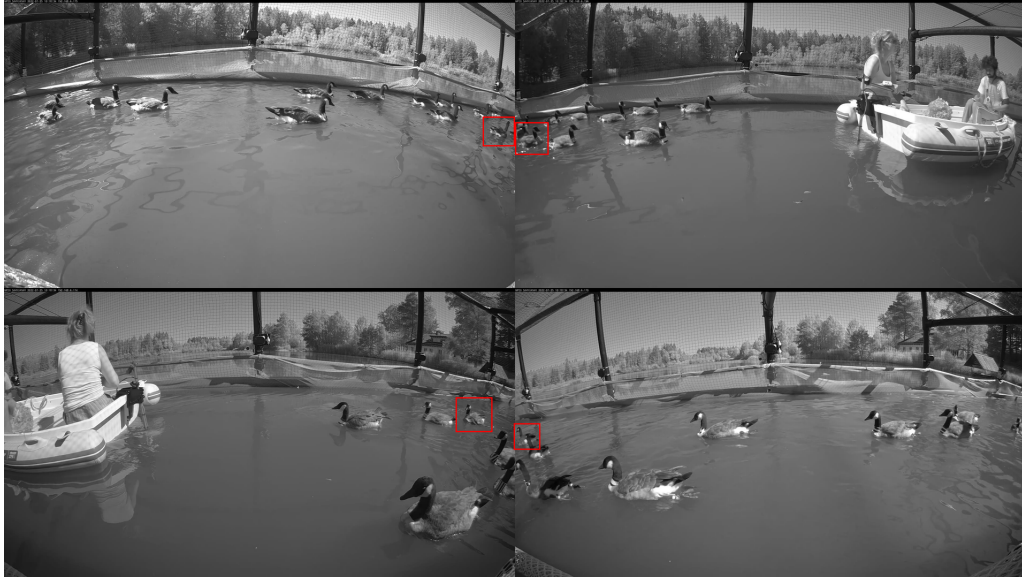


Figure 1.1: The same scene looked at from four different cameras with the red box highlighting a manually re-identified goose.

Chapter 2

Related works

Automated multi-animal tracking has seen significant advancements [6, 7, 8, 9, 10, 11, 12, 13], driven by the development of computer vision models that are highly effective in handling complex animal behaviours and interactions. SLEAP (Social LEAP Estimates Animal Poses)[14] is a versatile open-source tool for multi-animal pose tracking that offers significant improvements in the field of behavioural analysis. Unlike traditional tracking tools, SLEAP is designed to handle the complexities of multi-animal interactions, including occlusions and overlapping, making it well-suited for studies involving group behaviours. Anipose [15] introduces a toolkit that integrates SLEAP for 3D pose estimation in animals, demonstrating its application in various species. LiftPose3D [16] employs SLEAP for 2D pose estimation, which is then transformed into 3D poses using the LiftPose3D framework, enhancing the analysis of animal movements. In [17], the authors present a system for studying the behavioural dynamics of songbirds within a 3D aviary. It addresses challenges in tracking multiple visually similar birds in three dimensions, such as occlusion and orientation-dependent appearances. The authors introduce a novel dataset for evaluating multi-view trackers and analyze ethogram data to demonstrate how social context influences sequential interactions among birds. In addition to these pose-based tracking methods, there are also object detection-centric approaches. ChickTrack[18] utilizes a deep learning model based on the YOLOv5[19] algorithm to detect and track chickens in video footage, handling complex backgrounds and varying lighting conditions. The system then employs a Kalman Filter[20] to predict and update the positions of individual chickens across consecutive frames, enabling accurate multi-target tracking even in the presence of occlusions or overlapping birds. The paper "MBW: Multi-view Bootstrapping in the Wild" [21] presents a novel approach that leverages mul-

multiple uncalibrated camera views to handle frequent occlusions and viewpoint changes in unconstrained, real-world environments. The MBW framework first performs single-view tracking on each camera feed independently, then uses an optimization process to refine the tracks by propagating information across the different viewpoints, maintaining consistent identification of the target animal even when it is temporarily occluded by other individuals in the scene. While this method is specifically designed for single-animal tracking in complex, naturalistic settings, its underlying approach of combining partial tracklets from multiple views is promising and could potentially be extended to enable more robust multi-animal tracking as well.

In their study, [22] introduces an improved HRNet model [23] for pose estimation and behaviour classification of white ducks, showcasing strong results. While effective, the model's performance can be influenced by factors such as occlusions, highlighting the need for adaptation to scenarios involving multiple animals.

Some studies have also explored the use of DeepLabCut [24], a popular open-source toolbox for markerless pose estimation, in the context of multi-animal tracking. For example [25, 11, 26] have presented methods that build upon DeepLabCut to address challenges in tracking multiple, visually similar animals.

The above-mentioned frameworks often leverage the fact that object detection and segmentation is a widely researched topic. There are many different frameworks, models, and approaches one can choose from and combine to best fit the problem at hand. Detectron2 [27], built by Facebook AI Research, is a framework designed to support the fast implementation of object detection and segmentation algorithms. With a PyTorch [28] backbone, it allows users to quickly build and train models from scratch or use transfer learning to fine-tune pretrained weights. Mask-RCNN [29] is an extension of the Faster-RCNN [30] model and a member of the region-based convolutional-network family. Unlike Fast and Faster-RCNN it is designed specifically for instance segmentation outputting segmentation masks in addition to classes and bounding boxes. Detectron2 gives access to several pretrained instances of this model. Beyond deep networks, Detectron2 also offers visual-transformer based methods, such as Mask2Former [31]. Visual transformers are promising in the field of video instance segmentation [32, 33], although their computational need is higher compared to deep networks. YOLO [34] is a fast and efficient approach to object detection and segmentation. The latest versions YOLOv9 and YOLOv11 [35, 36, 37], are state-of-the-art algorithms in the field of object detection, segmentation and pose estimation. YOLO uses a single forward pass through the data, as

opposed to the two-stage approach of Mask-RCNN. Its real-time object detection capabilities, coupled with its ability to handle occlusions well, make it a suitable candidate for dealing with the dynamic and variable nature of animal tracking.

There are several different approaches to tracking. Modified IoU-based methods such as SORT [38] and DeepSORT [39, 40]. Optical flow based methods such as mmflow [41] and others. The SORT [38] algorithm is designed for multi-object tracking. It is based on a Kalman filter [42] that predicts the trajectories. The detections of each frame and the predicted trajectories are matched using intersection over union [43] (IoU) on the bounding boxes. Application of the Hungarian algorithm [44] ensures the global optimality of the assignment. DeepSORT [39, 40], an extension of the SORT algorithm. It includes extra appearance descriptors in the tracking process. These are extracted using deep learning and help to reduce the number of identity switches, thus making the tracking algorithm more accurate. However, it is a great challenge for appearance-based methods to track objects that are highly similar in appearance and occluded frequently. Modular Interactive Video Object Segmentation (MiVOS) [45] is an algorithm designed for segmenting out and tracking specific objects in a video, commonly used for annotation purposes. Interactive VOS methods take an iterative sequence of user actions as input, such as scribbles or clicks, to generate a segmentation mask of the target object in the video's first frame.

It is worth mentioning that in the field of multi-camera multi-object tracking, there are solutions for tracking people both outdoors and indoors [46, 47, 48, 49, 50]. However, due to the varying features of different individuals, the differences in the number and positions of the cameras and thus, the nature of camera overlaps, these solutions fundamentally address different problems compared to the setups used in biological research. Since animals do not differ as much as humans do in terms of clothing or appearance, re-identification becomes more challenging. Additionally, pedestrian tracking setups are often not analogous to configurations used in environments like a floating aviary.

Chapter 3

Methods

3.1 Data collection

The experiments took place in a floating aviary¹. The outdoor structure measures 75m x 3.5m, split between a platform and a swimming area. 23 Axis M2026 MkII cameras were installed to observe the animals; of these, 20 are looking over the water arranged in pairs on opposite sides of the aviary, each pair mounted 6m apart. Due to space limitations, the boat crossed the field of view of only eight camera pairs during the experimental sessions. The geese followed the boat as they had been imprinted on the biologists from an early age. The lack of a strong motivator means that the size of the flock following the boat varies widely even within the same session, as geese get tired or otherwise lose interest. Each session was 2 hours long, with roughly 15 cycles around the aviary. These video blocks were chopped into smaller segments for processing. The chopped segments form the input for our software. An overview of the aviary layout can be seen in Fig. 3.1.

¹On the EßSee, Starnberg, Germany

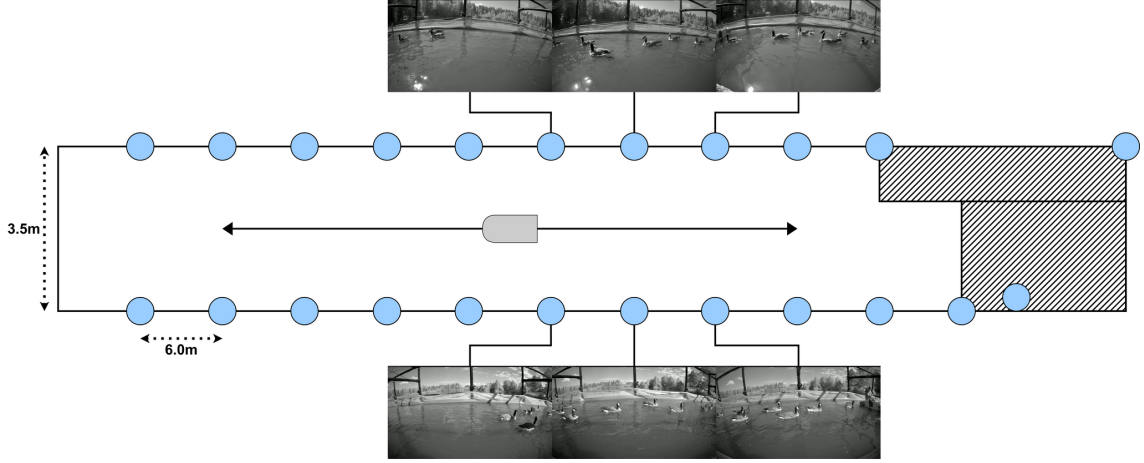


Figure 3.1: The schematic overview of the aviary.

3.2 Synchronization

To deal with the varying framerate and uniform length of the input videos, we need to synchronize them, otherwise the position predictions made by the different cameras would quickly slip out of sync making cross camera tracking more difficult.

Our synchronization approach comprises two parts: the timestamp refinement and the subsequent frame correspondence mapping. Within the series of frames, each frame's position is fixed, but the exact millisecond within the second is not specified, as the computer-generated timestamps are only represented to a second precision. Thus, a sequence of frames is formed for each second, with multiple frames sharing the same timestamp. Due to the camera settings, the number of captured frames is not guaranteed to be the same within each second. To achieve a better level of synchronization, we refine the original computer-generated timestamp by assigning a fractional timestamp to every frame based on its location among frames with the same stamp (this location is determined by their order in the video), effectively assigning a predicted millisecond value to each. See Fig. 3.2 for illustration.

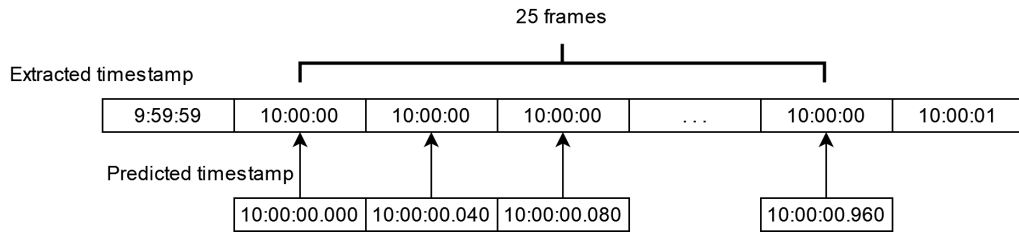


Figure 3.2: Illustration of the timestamp refinement. If 25 frames have the same stamp 10:00:00, they will receive the fractional timestamps with an increment of 40 milliseconds based on their relative order.

To find the corresponding frames among the separate videos, we create a uniform 25 FPS baseline synchronization signal spanning from the earliest to the last timestamp. For each "tick" of the baseline signal, we choose the frames with the closest fractional timestamp from each video to create the final list of synchronized frame IDs. By picking the closest frames, we effectively distribute the insertions/deletions across the entire second, as the difference between the length of time a frame represents compared to a "tick" of the baseline signal slowly accumulates, until a frame is skipped (or duplicated depending on the sign of the difference). Fig. 3.3 provides a visual illustration of the process. This allows us to work with cameras from different ends of the swimming lane with little to no overlap in time or space and considerably different frame rates.

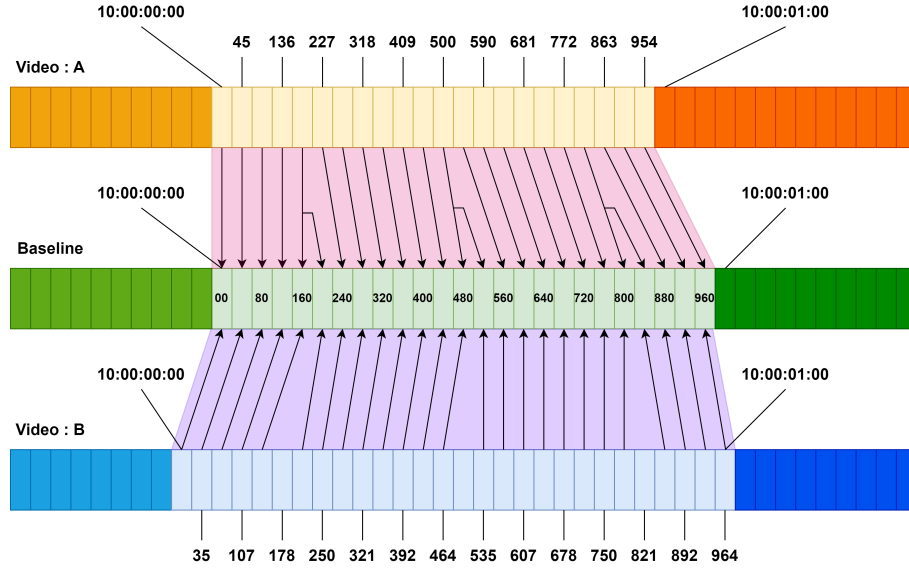


Figure 3.3: Illustration of the frame correspondence mapping. For each "tick" of the baseline synchronization signal, we pick the corresponding frames from videos A and B with the closest fractional timestamp. These frames are then handled as depicting the same moment in time.

3.3 Intrinsic camera parameters and distortion

To determine the intrinsic matrix and distortion coefficients of the cameras, we used the Zhang calibration algorithm [51] on a set of extracted frames with a 9×7 chessboard pattern moved around in a boat. The proper calibration video was available for only one pair of cameras, however as the type and model of the cameras are identical, this didn't effect precision in any significant capacity.

As the width of the aviary is very small compared to its length and the distance between camera pairs, the biologists had to use *fisheye* lenses on the cameras, which introduced significant amounts of distortion into the images we had to correct for. This includes radial and tangential distortions. To express these, we use [52] the Brown-Conrady distortion model [53], with Eq. 3.1 and Eq. 3.2 to express the offset along the x and y axis, respectively.

$$\delta x = (x - c_x)(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) + [p_1 \cdot (r^2 + 2(x - c_x)^2) + 2p_2 \cdot (x - c_x) \cdot (y - c_y)] \quad (3.1)$$

$$\delta y = (y - c_y)(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) + [2p_1 \cdot (x - c_x) \cdot (y - c_y) + p_2 \cdot (r^2 + 2(y - c_y)^2)] \quad (3.2)$$

The undistortion process straightens out the center of the image at the cost of the warping of the sides. For this reason, we select an appropriate region of interest from the full undistorted image, rescale it, and use the new intrinsic matrix (corresponding to the full undistorted image) to match the original input size. We will continue referring to this newly calculated matrix as the camera's intrinsic matrix in later steps. The undistorted image and the discarded region are illustrated in Fig. 3.4. The loss of the edges means a loss of overlap between camera pairs, however from an application standpoint in these areas, the geese are usually too far away from the camera for either identification or eye-state extraction, thus it is a worthwhile tradeoff in exchange for the ability to predict positions more precisely.

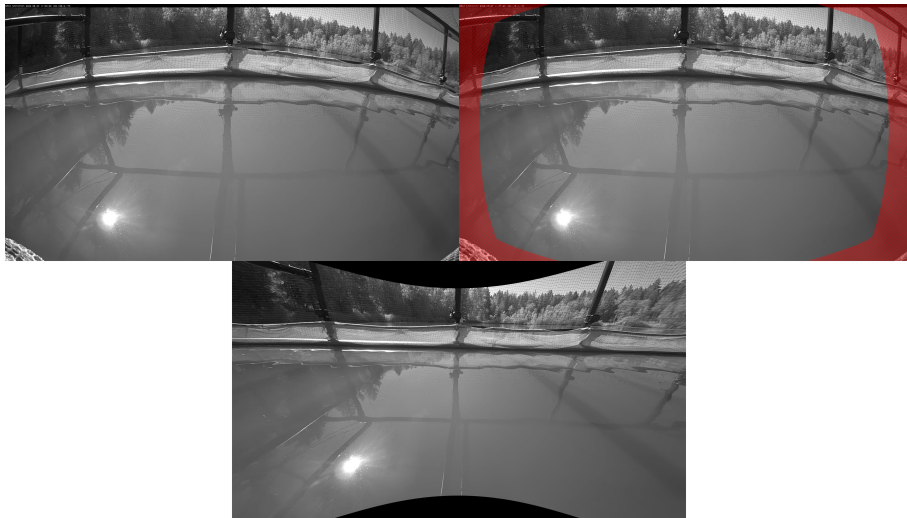


Figure 3.4: Top row: The original picture and the same picture with the area discarded after undistortion marked with red. Bottom row: The undistorted picture.

3.4 Orientation estimation

After the intrinsic parameters of the cameras have been calculated, we still need to find their extrinsic values to be able to predict the position of the geese swimming on the water accurately. The position of the cameras can be considered fixed within a margin of error, given by the aviary's layout. However, their orientation is unknown and not constant, as the entire structure has been designed to be relatively flexible to adjust to the waves. Thus, there is an amount of randomness we have to deal with. To address this, for each input we recalculate the orientation of the cameras.

We start from an undistorted image and use four things we know about the setup.

- The intrinsic parameters of the camera.
- The distance of the mounted camera from the water. As the whole structure is floating, this can be considered a constant value.
- The fact that the opposite side of the aviary forms an approximately straight line.
- The width of the aviary.

Based on these, we estimate the roll and pitch of the cameras so that the predicted plane of the water aligns with the opposing edge of the aviary (where the structure meets the water), illustrated in Fig. 3.5.

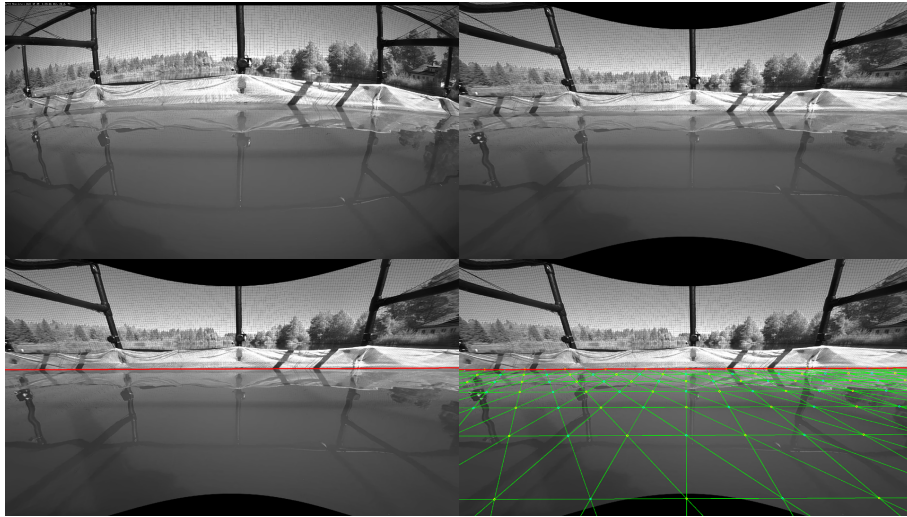


Figure 3.5: Top row: The original and the undistorted image. Bottom row: The predicted edge of the aviary, marked with red. A rendered mesh visualizing the predicted surface of the water.

Here, we only care about the orientation along two axes. This is because we aim only to align the predicted plane with a single line and don't care about where its origin falls on

that line. If we would like to calculate the orientation’s third value (yaw) in this restricted system, we would need a second perpendicular line, something we do not have in our setup, as there is no reliable structure present on the water’s surface (due to waves, the reflections are unreliable at the best of times).

The first step in calculating the orientation is determining where the opposing edge of the aviary is. For this, we find the lower edge of the white net on the opposing side, as it is a more consistent feature than the water’s surface. We prompt SAM with a fine-tuned nano-size YOLOv8 segmentation model. We use SAM as during inference, YOLO operates on scaled down images and thus the segmentation masks are not pixel-precise, this makes them insufficient for determining the exact line where the water meets the structure.

Algorithm 1 Orientation Calculation for Restricted System

- 1: **Input:** Extracted frame, intrinsic camera parameters, YOLOv8 fine-tuned model
 - 2: **Output:** Target line for orientation fitting (L_{final}), orientation parameters (pitch, roll)
 - 3: **Aviary edge detection:**
 - 4: Use the YOLO model, to prompt SAM with a box and five positive point prompts.
 - 5: Predict a mask (M) for the white net with SAM.
 - 6: Extract the largest connected component to reduce noise.
 - 7: **Repeat**
 - 8: Fit a line L to the lowest pixels of M .
 - 9: Update M by removing pixels below L (with a 10px grace zone).
 - 10: **Until** Number of pixels removed is low
 - 11: Distribute additional point prompts across L , above and below the predicted line.
 - 12: Generate a new mask M_{new} .
 - 13: Extract the largest connected component of M_{new} .
 - 14: Fit the final line L_{final} to the lowest points of M_{new} .
 - 15: Repeat 6 – 9. to converge to a stable position.
 - 16: **Orientation Parameter Fitting:**
 - 17: Use the center pixel of L_{final} to calculate pitch (Eq. 3.7).
 - 18: Use two opposing pixels where L_{final} meets image edges to calculate roll (Eq. 3.4).
 - 19: **Return** L_{final} and orientation parameters (pitch, roll)
-

This initial mask prediction gives us a good result in most cases; however, if the ambient lighting is very strong, the contrast between the white net and the edge of the water is reduced, and the mask predicted by SAM tends to become very noisy, skewing the results of the line fitting. We take several steps to combat this. Only the largest connected component in the predicted SAM mask is considered, to reduce noise caused by small disconnected islands around the edge of the main mask. We take the initial mask prediction, fit a line on its lowest pixels, and remove points below the fitted line (with a 10px grace zone). We fit a new line onto the lower edge of the new mask and repeat until con-

vergence (the number of pixels removed is very small). Using the new line, we give SAM ten additional point prompts (five positive and five negative ones) distributed uniformly across the width of the line, with positive prompts going above and negatives below the line. Using these new prompts, a new mask is predicted, the largest connected component is selected and converged to a stable position. The line fitted to the lowest points of this final prediction will form the target line to which we fit the orientation parameters, see Alg. 1 for an overview.

We want to adjust the pitch and roll parameters so that projection of the line a camera height down along the y-axis and an aviary width away along the z-axis is as close to the observed edge of the aviary as possible. We use the center pixel of the observed aviary edge (the previously predicted line stretching from one side of the image to the other), to calculate the pitch as this pixel isn't affected by the roll of the camera. We use Eq. 3.3 to calculate the exact pitch parameter (up to a rounding error, as pixels, by their nature, are discrete values). We then pick the two opposing pixels where the line meets the edge of the frame and calculate the roll to ensure that the inverse projected coordinates of these two pixels are as close along the z-axis as possible. See Eq. 3.4. After simplification, this turns out to be independent of the pitch parameter and essentially just the angle of the perceived edge of the aviary. For the detailed calculations, see Appendix A.

$$\theta = \tan^{-1} \left(\frac{c_0 - z_0 \cdot \frac{y - c_y}{f_y}}{(z_0 + c_0 \cdot \frac{y - c_y}{f_y})} \right) \quad (3.3)$$

$$\lambda = \tan^{-1} \left(\frac{\frac{y_1 - c_{y1}}{f_{y1}} - \frac{y_2 - c_{y2}}{f_{y2}}}{\frac{x_1 - c_{x1}}{f_{x1}} - \frac{x_2 - c_{x2}}{f_{x2}}} \right) \quad (3.4)$$

3.5 Tracking

To overcome the difficulties mentioned in the Introduction, we created a composite tracking system that relies on a combination of visual and positional tracking to supplement the shortcomings of each approach, as each proved to provide suboptimal results separately.

To segment the geese, we fine-tuned a YOLOv8 model (*yolov8l-seg*), pretrained on the COCO dataset [54], using approximately 2 500 manually annotated images.

Based on the bounding boxes, the identifier collars present at the base of the geese'

neck is segmented out. In case more than one collar falls within the bounding box because of the proximity of the animals, we pick the collar with the highest overlap with the segmentation mask of the goose (at least a 50% overlap is required, to consider the collar to prevent misidentification of geese swimming in and out of the frame). The collars are classified using a *yolov8x-cls* model fine-tuned on 32 000 collar boxes (2000/*category*). See Fig. 3.6 for the collar patterns used by the biologists. To combat the emergence of

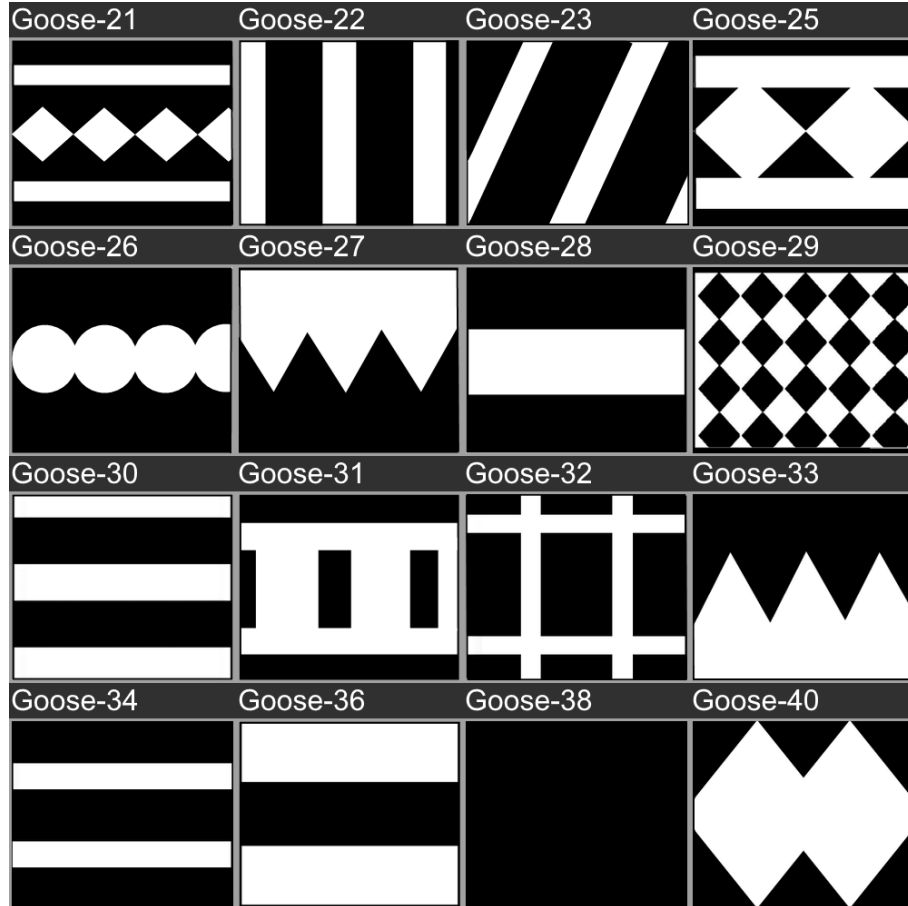


Figure 3.6: A list of the 16 collar patterns used by the biologists. The similarity of certain patterns combined with the birds’ black and white neck and chest plumage makes identification considerably more difficult.

false-positive bounding boxes whose presence could introduce further noise into the positional tracking parts of the algorithm, we use a surface-level SORT tracking to validate them. This receives as input all of the bounding boxes detected for each frame, and constructs the corresponding tracklets. Bounding boxes belonging to tracklets that are at least 25 frames (1 second) long are considered valid; the rest are thrown away, filtering out possible false-positive and duplicate detections and reducing noise in the subsequent steps.

Every valid detection is then assigned a position value, utilizing the fact that the geese are swimming on the surface of the water, we intersect the 3D ray going through their

bottom-center pixel with the predicted plane of the water (as the aviary is floating and the size of the waves are insignificant compared to the size of the birds, we can approximate the water with a plane located a fixed distance below the camera). This yields us a relative position value for each animal.

As the geese swim in and out of the frame, their bounding boxes get squished, thus distorting the center point, resulting in the predicted position temporarily stalling around the edges. To deal with this, if a bounding box touches the edge, we apply a rough corrective measure on the corresponding predicted position, by offsetting it along the x – axis so that the predicted positional difference between the center and edge of the bounding box is equal to half the average length of the birds.

These steps transform the bounding box predictions made by individual cameras into a global positional representation.

As each camera predicts a position value for each animal it sees, we need to merge these predictions not only to pool the animals’ respective detections in different videos, but also to mitigate certain errors in the position prediction. These errors stem primarily from three sources: imprecisions in the orientation estimation of the respective cameras, slight inconsistencies in the bounding box detection, and the fact that we predict positions for the side of the goose closest to the camera.

The algorithm that merges the individual positional representations operates on two levels. The lower level is performed repeatedly, while the higher level modifies the underlying position predictions each time to achieve a better result.

The lower level can be split into two phases: pooling and pairing. Pooling deals with detections made by cameras on the same side of the aviary, while pairing connects detections made by cameras on opposing sides. In the pooling phase, we form clusters from the positions, by linking together any position that falls within a certain (half-a-geese length) threshold. After calculating the center point of these initial clusters, any point that is more than a goose-length away is discarded from the cluster. These discarded points are then reclustered one last time. We create the predicted points from the resulting grouped (or singular) points, by calculating their center of mass and pooling the associated detections.

These predicted points are passed along to the pairing step. During the pairing step, we calculate the Euclidean distance of every point from all points coming from the opposing camera side. The ones that fall within a threshold (this threshold is larger for points with matching IDs) are considered *close* to that point. Based on the number of *close* points, they are split into three categories. See Fig. 3.7

- A/B-only: The points that fall within this category don't have any points close enough to form a valid pair.
- Certain matches: Points that mutually have a single point from the other side close enough to form a pair. We handle these as if they belong to the same animal and merge their data together, with the position averaged out between them.
- Inconclusive: If a point has multiple associated *close* points, a clear pairing cannot be established, and we discard them as inconclusive, as false-positive pairs are far more damaging to the integrity of the overall tracking than the false-negative ones.

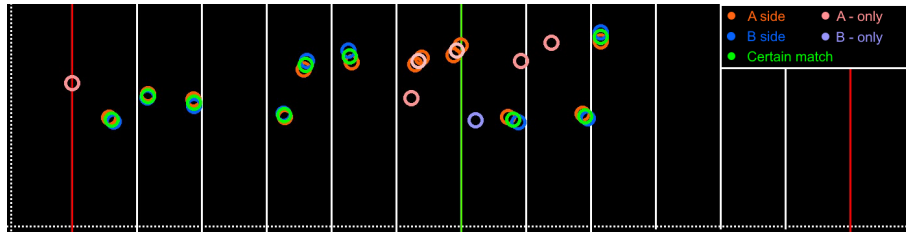


Figure 3.7: The results of the pooling-pairing process visualized on top of a schematic overview of a segment of the aviary

To mitigate the small errors caused by inaccuracies in the calibration, detection and position prediction. We apply average offset correction to the data in the higher level of the merging algorithm. This means that after every iteration of the pooling-pairing algorithm, we modify the predicted positions along one axis. First, we take the certain matches from each camera pair and calculate the average difference between the two points that form that pair, each of these points are then offset by that difference, reducing the gap between them and making the next set of pooling-pairing more precise. This is done both along the x and y-axis. Finally, we calculate the average difference between pooled points on the x-axis and cumulatively shift the predicted positions in each pool (pair of cameras). This helps clean the result of the merging algorithm and smooth out the perceived trajectory of the points, making their tracking easier.

The merged points serve as input to SORT, which produces our initial set of tracklets. However, these are usually the broken-up pieces of the full-length tracks, thus we need to perform post-processing on them. In a post-processing step, we assemble these pieces (subtracklets from here on out) into full-length tracks. The post-processing step handles the input set of subtracklets as internally consistent pieces to be paired together, for this reason, we first need to insure their internal consistency. This is done using the identifying

collars, however, as they are not visible all of the time, we take the most numerous known identity (we denote the animal as *unknown* in the absence of any known identity), and name it as that subtracklets leading identity. All other known identities are either discarded to reduce noise, if their overall multiplicity doesn't exceed 5 (0.2 seconds of visibility) or cause the subtracklet to be flagged for user confirmation. The user can cut the subtracklets and redesignate each piece individually.

The problem of linking the subtracklets together can be viewed as a linear assignment problem, where we want to connect subtracklet ends to subtracklet beginnings. The cost of connecting the end of subtracklet A with the beginning of subtracklet B is calculated as follows. The distance of the endpoint of A and the first point of B in both time and space (v_{diff}) is multiplied by the 2 minus the cosine similarity of the spatial difference between A and B (v_{diff}) and the direction vector of A's smoothed-out trajectory during the last 100 frames (V_A) assigning a lower cost to pairs, where subtracklet A would need to continue swimming along a similar trajectory to get to B. This value is multiplied by the DL-ratio 3.5, the distance of the jump divided by the minimum of the temporal length of A and B, penalizing pairs, where we would need to jump a large distance, to connect to a small tracklet. Finally, if we enable identifiers, we weigh the cost based on the similarity of the leading ID of A and B (*leading_id_modifier*), increasing cost for differing IDs, keeping them the same, if either one is unknown and reducing them if their IDs match 3.6. See Eq. 3.7.

$$DL_ratio = v_{diff} * \min(length_A, length_B) \quad (3.5)$$

$$leading_id_modifier = \begin{cases} 1, & leading_id_A = unknown \vee leading_id_B = unknown \\ \frac{1}{1000}, & leading_id_A = leading_id_B \\ 1000, & leading_id_A \neq leading_id_B \end{cases} \quad (3.6)$$

$$C_{AB} = \|v_{diff}\| \cdot \left(2 - \frac{v_{diff} \cdot V_A}{\|v_{diff}\| \cdot \|V_A\|} \right) \cdot DL_ratio \cdot leading_id_modifier \quad (3.7)$$

We introduce difference, minimum-length and cost thresholds to force sparsity over the bipartite graph and prevent highly improbable pairs from forming. These thresholds (except for the cost) are not affected by the *leading_id_modifier*.

To prevent small, *unknown* subtracklets being skipped by the *leading_id_modifier* induced cost reduction. We form the pairs in iterative steps, loosening the thresholds in each step. After each iterative step, we merge the paired subtracklets, recalculating their

values but saving the underlying indices of unprocessed tracklets for the manual correction step. We continue working with these subtracklets in the next iteration. We use curve corrected moving average [55] to smooth out the tracklets and fill the gaps caused by occlusion using position interpolation. Finally based on the leading identity of the assembled tracks and their underlying subtracklets, we flag possible post-processing errors. For an overview of the tracking pipeline and the output, see Fig. 3.8 and Fig. 3.9.

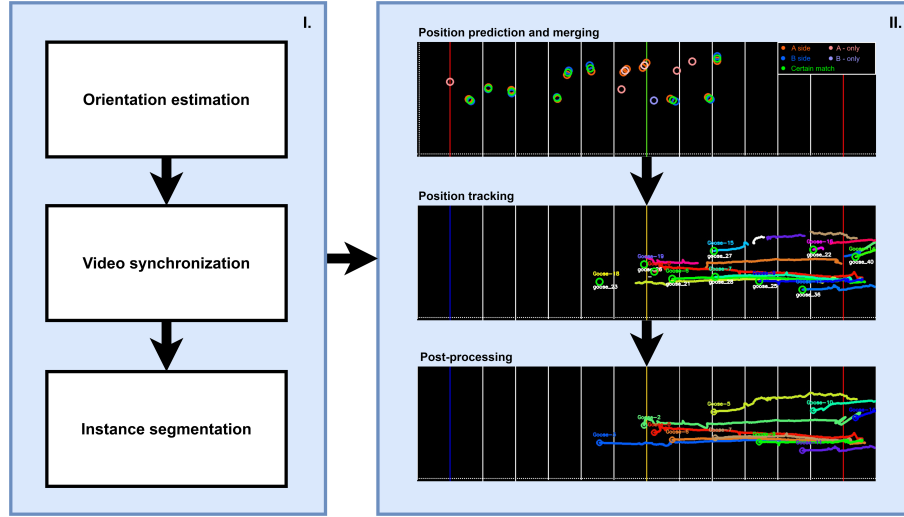


Figure 3.8: An overview of the tracking pipeline’s key components.

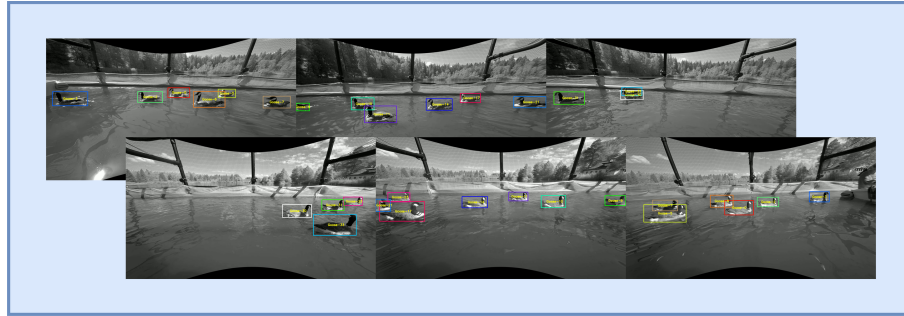


Figure 3.9: Visualization of the tracking output overlaid onto frames extracted from three neighbouring camera pairs depicting the same synchronized moment.

3.6 Eye-state extraction

The end goal of the pipeline is to provide the biologists with the extracted eye states of the geese. The geese can sleep without their eyes being fully closed, and vice versa, they can be awake with a part of the eyelid visible. For this reason, we try to classify the eyes using a threshold. If $\sim 1/2$ of the eyelid or less is visible, we consider the eye open,

otherwise it is classified as closed. To overcome the challenges of differing perspectives, lighting and resolution of the eye, we provide a robust, multi-step pipeline to extract the eye-states. The pipeline is comprised of four steps. First, we segment the heads from the extracted bounding boxes. If more than one head is visible in the box due to the proximity of the birds, we use the one with the highest overlap with the segmentation mask corresponding to the bounding box. If no head with a sufficiently large overlap with the segmentation mask of the goose can be detected, we automatically handle it as not visible.

Afterwards, in a preliminary classification step, we sort the head bounding boxes into three categories.

- Open: If no eyelid is visible. In this case, we might not necessarily know where the eye of the animal is located, but we can be certain that the eye is open.
- Partially closed: If we can see any part of the eyelid, we classify it as partially closed. At this stage, we don't yet know whether the eye is closed enough to be considered properly closed or not.
- Not-visible: Heads that are turning away from the camera and thus none of the eyes are visible belong to this category.

We continue working with the partially closed category, by segmenting out the eyelids. If more than one eyelid is detected, we discard the frame as unknown. Otherwise, even though we don't know whether enough of the eyelid is visible, to consider it closed, but we know where it is located, giving us the region of interest for the final classification step.

For the last step of the pipeline, we cut out an area based on the eyelid-segmentation mask and classify it into either open or closed.

For an overview of the eye-extraction process, see Fig. 3.10.

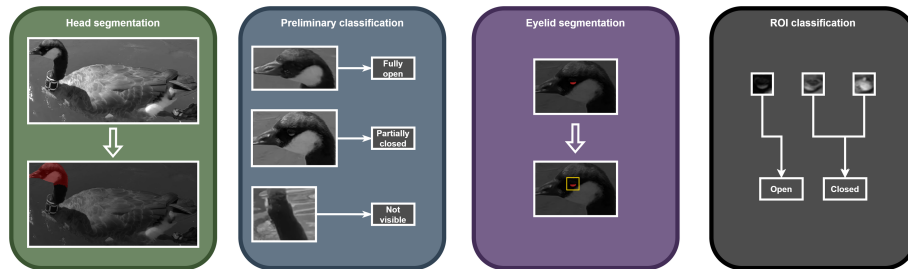


Figure 3.10: Illustration of the main steps of the eye-state extraction process.

Chapter 4

Results and conclusion

Evaluation of tracking performance

We evaluated our pipeline’s performance on videos taken from two days (2022/08/22 and 2022/08/25). A pass-through consists of a batch of 16 videos. Each video in the batch is chopped to the same length (~ 3 minutes), encompassing the entire length of the pass-through (the boat entering at one side of the observation area and leaving at the other). This means that each video contains a buffer zone before and after the flock reaches that particular camera. The results displayed below were measured after the manual correction steps were performed. The first manual correction step takes a fairly heavy-handed approach as only $\sim 19\%$ of the flagged subtracklets actually contain errors. However, the priority was on the precision of the end result, and even with this approach, the flagged subtracklets represent only a fraction ($\sim 17.6\%$) of the total number of assigned detections, with an even smaller fraction ($\sim 3.8\%$) of all detections being actually marked as requiring attention.

As the most relevant quality for the biologists is the tracking precision for the sake of eye extraction, we focus on the number of incorrectly assigned detections (the tracklet they were assigned to isn’t the tracklet they belong to) as our key metric. Furthermore, we measured the number of temporary tracklet switches. This indicates the total number of frame sequences assigned to the wrong tracklet. See Table 4.1 for the results.

Tracking results				
	Number of assigned detections	Incorrectly assigned detections	Temporary tracklet switch within incorrect intervals	Correctly assigned detection ratio
08/22				
Batch 1.	71 450	206	3	99.71 %
Batch 2.	74 165	281	8	99.62 %
Batch 3.	65 067	377	5	99.42 %
Batch 4.	75 504	37	3	99.95 %
Batch 5.	49 799	201	4	99.59 %
08/25				
Batch 1.	102 683	43	3	99.95 %
Batch 2.	56 331	25	3	99.95 %
Batch 3.	77 604	587	8	99.24 %
Batch 4.	66 965	68	4	99.89 %

Table 4.1: This table shows a breakdown of the tracking pipeline’s output. For every batch, the accuracy of identification of the instances is over 99%, which ensures reliable tracking.

Comparison with the previous approach

As a baseline, we present a comparison measured on videos from 2022/08/22 between the performance of our approach and the purely DeepLabCut-based tracking previously used by the biologists. The DeepLabCut model was trained on 1241 frames annotated by the biologists, who used it for each camera separately. Then, a manual re-identification was performed across the different videos of the same two hour long session. However, the processing of this output proved too time-consuming for the accuracy it provided. Thus, only a few camera pairs and certain days ended up being processed in this manner. This means that there isn’t a full ground truth dataset available for comparison.

As there is no synchronization of labels between videos in the DeepLabCut output, we compare the results on a video-by-video basis rather than on all of the videos combined. We generated a camera-by-camera breakdown of the tracking results obtained from the full camera setup and used this breakdown to compare with the results obtained by DeepLabCut. 4.2.

	DLC results			Our results		
	Temp. tracklet switches	Incorrectly assigned detections	Number of assigned detections	Temp. tracklet switches	Incorrectly assigned detections	Number of assigned detections
Cam. 6170	2	97	3931	1	27	4768
Cam. 6175	3	130	3544	1	1	3993
Cam. 6174	5	123	4167	0	0	4367
Cam. 6196	4	122	3108	0	0	3793
	Extrapolated to full batch			Averaged over evaluated batches		
16 cameras	56	1888	59000	5	220	67197

Table 4.2: This table compares the averaged results of the DLC pipeline and the results of our method.

Eye-state extraction

We evaluated the accuracy of the eye state extraction pipeline on a ground truth dataset of 3000 extracted goose images. These images were split between *open*, *closed* and *not visible* with a ratio of 3 : 2 : 1. Each image was sorted into one of four categories, these being *open*, *closed*, *not visible* and *unknown*. The last category is reserved for images for which a clear classification could not be established. The version we used for performance evaluation differs from the one integrated into the main pipeline in a single point. As the test dataset only contains images without the corresponding segmentation masks, during the head segmentation step it doesn't consider the masks of the geese. The pipeline achieved a precision of 89.06%, which falls within the acceptable range. Both because the relatively small size of the eyes makes classification difficult even for a human and because almost half of the incorrectly labelled frames were classified as *not visible* or *unknown*, classes that do not affect the biological conclusion. See Table 4.3 for a more detailed breakdown.

Eye-state classification breakdown				
	Open	Closed	Not visible	Unknown
Open	1386	42	30	42
Closed	127	790	13	70
Not visible	4	0	496	0

Table 4.3: This table shows a breakdown of the eye-extraction pipeline's output. The rows represent the goose images with a certain eye state. The columns represent the output classes and the number of images the pipeline classified as belonging to that class.

Conclusion

In their study investigating the possibility of unihemispheric sleep in actively moving birds, the biologists at the Max Planck Institute for Biological Intelligence have to analyze a considerable amount of visual data. Using manual annotation methods, the processing of only the 13, two-hour long boat sessions would take over a year.

Thus, developing a method that can automatically process this visual data by tracking and re-identifying the geese across the eight camera pairs is absolutely crucial to allow the proper and timely progress of their research project.

There are numerous difficulties associated with the problem at hand, from distorted, unsynchronized videos, the high similarity of the birds, frequent occlusion and the small overlap between neighbouring cameras.

We have designed a composite tracking system that can solve the problem with high precision. With the help of several deep neural networks, we detect, segment, and validate the geese instances. Their 3D position is determined using the calibration data, fine-tuned with the help of an automatic orientation estimation algorithm. Through pooling and pairing steps, we have successfully reduced the problem from the original visual-based tracking and re-identification across multiple videos to a 2D positional tracking on a single dataset, a task that can be solved much more precisely and efficiently. Through several layers of post-processing on the proposed tracklets, we have further increased the consistency of the tracking, reducing the number of identity switches and allowing a flag to be raised to let the user know of potential errors in the algorithm's output. The remaining error is less than 1%, which is comparable to the error and inconsistency of a human annotator both in terms of the number of switched ID labels and number of incorrect detections.

At the same time, the processing time has been significantly reduced. For the boat sessions, we estimate the total computational time of the not-optimised and not-parallelised code to be around three weeks for the entire season, even on a mid-range GPU. Moreover, the real speed-up is reflected in terms of human attention and time, which has yet to be measured precisely. However, based on experiences so far, it is approximately a 10-12x improvement compared to the DeepLabCut-assisted semi-manual method.

Appendix

A.1 The starting problem: Given a camera's intrinsic parameters and a plane located at a set distance c_0 below the camera, we want to find a (not necessarily distinct) camera orientation where the predicted plane lines up with the perceived plane (water surface) so that we can accurately predict the positions of swimming objects. Given an image, the K intrinsic matrix of the camera, the c_0 distance along the y axis and a perceived line (edge of the aviary) and its distance along the z-axis, we can accurately calculate a pitch and roll value that solves the original problem. The problem can be split in two parts. First, we find the roll value for which the point of the perceived line in the middle of the image (no relative translation along the x-axis) is estimated to be z_0 distance away from the camera along the z-axis. Afterwards, we need to find a pitch value so that every point on the line has the same predicted distance along the z-axis.

A.2 First, let's define the inverse-projection that takes a 2D-pixel coordinate, the camera's intrinsic matrix, rotation vector and the distance of the constraining $xz - plane$ and returns the coordinates obtained by intersecting the homogeneous ray going through the pixel with the containing plane.

$$h_ray = \begin{bmatrix} \frac{x-c_x}{f_x} \\ \frac{y-c_y}{f_y} \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad (1)$$

$$ir_ray = R^{-1} \times \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} \quad (2)$$

$$p_constraint = c_0/y'' \quad (3)$$

$$relative_point_on_the_plane = \begin{bmatrix} x'' \cdot p_constraint \\ p_constraint \\ z'' \cdot p_constraint \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (4)$$

We want to find a specific rotation (θ) along the x-axis for which the z-coordinate of

the inverse projection of the pixel in the middle of the perceived line (thus not affected by the roll) is z_0 .

Let's modify the original equations by replacing the 3D rotation matrix R with R_x that only deals with rotation along the x-axis.

$$R_x^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \quad (5)$$

This gives us the following equation for ir_ray .

$$\begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} \frac{x-c_x}{f_x} \\ \frac{y-c_y}{f_y} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x-c_x}{f_x} \\ \cos\theta \cdot \frac{y-c_y}{f_y} + \sin\theta \\ \cos\theta - \sin\theta \cdot \frac{y-c_y}{f_y} \end{bmatrix} \quad (6)$$

The relative point on the plane becomes:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \frac{x-c_x}{f_x} \cdot \frac{c_0}{\cos\theta \cdot \frac{y-c_y}{f_y} + \sin\theta} \\ c_0 \\ (\cos\theta - \sin\theta \cdot \frac{y-c_y}{f_y}) \cdot \frac{c_0}{\cos\theta \cdot \frac{y-c_y}{f_y} + \sin\theta} \end{bmatrix} \quad (7)$$

We only care about the Z coordinate; we want to find θ for which $Z = z_0$.

$$z_0 = (\cos\theta - \sin\theta \cdot \frac{y-c_y}{f_y}) \cdot \frac{c_0}{\cos\theta \cdot \frac{y-c_y}{f_y} + \sin\theta} \quad (8)$$

$$z_0 = \frac{c_0 \cdot \cos\theta}{\cos\theta \cdot \frac{y-c_y}{f_y} + \sin\theta} - \frac{c_0 \cdot \sin\theta \cdot \frac{y-c_y}{f_y}}{\cos\theta \cdot \frac{y-c_y}{f_y} + \sin\theta} \quad (9)$$

$$z_0 \cdot \cos\theta \cdot \frac{y-c_y}{f_y} + z_0 \cdot \sin\theta = c_0 \cdot \cos\theta - c_0 \cdot \sin\theta \cdot \frac{y-c_y}{f_y} \quad (10)$$

$$z_0 \cdot \frac{y-c_y}{f_y} + z_0 \cdot \tan\theta = c_0 - c_0 \cdot \tan\theta \cdot \frac{y-c_y}{f_y} \quad (11)$$

$$z_0 \cdot \tan\theta + c_0 \cdot \tan\theta \cdot \frac{y-c_y}{f_y} = c_0 - z_0 \cdot \frac{y-c_y}{f_y} \quad (12)$$

$$\tan\theta \cdot (z_0 + c_0 \cdot \frac{y-c_y}{f_y}) = c_0 - z_0 \cdot \frac{y-c_y}{f_y} \quad (13)$$

$$\tan\theta = \frac{c_0 - z_0 \cdot \frac{y-c_y}{f_y}}{(z_0 + c_0 \cdot \frac{y-c_y}{f_y})} \quad (14)$$

$$\theta = \tan^{-1} \left(\frac{c_0 - z_0 \cdot \frac{y-c_y}{f_y}}{(z_0 + c_0 \cdot \frac{y-c_y}{f_y})} \right) \quad (15)$$

A.3 Given the θ rotation along the x-axis, we want to find a rotation (λ) along the z-axis, for which any two points on the perceived line will have the same z coordinate. As we will see, this is essentially just the angle of the line.

Let's modify the original equations by replacing the 3D rotation matrix R with R_{xz} that only deals with rotation along the x-axis.

$$R_{xz}^{-1} = \begin{bmatrix} \cos\lambda & -\sin\lambda \cos\theta & \sin\lambda \sin\theta \\ \sin\lambda & \cos\lambda \cos\theta & -\cos\lambda \sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}^{-1} = \begin{bmatrix} \cos\lambda & \sin\lambda & 0 \\ -\sin\lambda \cos\theta & \cos\lambda \cos\theta & \sin\theta \\ \sin\lambda \sin\theta & -\cos\lambda \sin\theta & \cos\theta \end{bmatrix} \quad (16)$$

This gives us the following equation for ir_ray .

$$\begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} = \begin{bmatrix} \cos\lambda & \sin\lambda & 0 \\ -\sin\lambda \cos\theta & \cos\lambda \cos\theta & \sin\theta \\ \sin\lambda \sin\theta & -\cos\lambda \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} \frac{x-c_x}{f_x} \\ \frac{y-c_y}{f_y} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\lambda \cdot \frac{x-c_x}{f_x} + \sin\lambda \cdot \frac{y-c_y}{f_y} \\ -\sin\lambda \cdot \cos\theta \cdot \frac{x-c_x}{f_x} + \cos\lambda \cdot \cos\theta \cdot \frac{y-c_y}{f_y} + \sin\theta \\ \sin\lambda \cdot \sin\theta \cdot \frac{x-c_x}{f_x} - \cos\lambda \cdot \sin\theta \cdot \frac{y-c_y}{f_y} + \cos\theta \end{bmatrix} \quad (17)$$

The relative point on the plane becomes.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} (\cos\lambda \cdot \frac{x-c_x}{f_x} + \sin\lambda \cdot \frac{y-c_y}{f_y}) \cdot \frac{c_0}{-\sin\lambda \cdot \cos\theta \cdot \frac{x-c_x}{f_x} + \cos\lambda \cdot \cos\theta \cdot \frac{y-c_y}{f_y} + \sin\theta} \\ c_0 \\ (\sin\lambda \cdot \sin\theta \cdot \frac{x-c_x}{f_x} - \cos\lambda \cdot \sin\theta \cdot \frac{y-c_y}{f_y} + \cos\theta) \cdot \frac{c_0}{-\sin\lambda \cdot \cos\theta \cdot \frac{x-c_x}{f_x} + \cos\lambda \cdot \cos\theta \cdot \frac{y-c_y}{f_y} + \sin\theta} \end{bmatrix} \quad (18)$$

We only care about the Z coordinate.

$$Z = \frac{c_0 \cdot (\sin\lambda \cdot \sin\theta \cdot \frac{x-c_x}{f_x} - \cos\lambda \cdot \sin\theta \cdot \frac{y-c_y}{f_y} + \cos\theta)}{-\sin\lambda \cdot \cos\theta \cdot \frac{x-c_x}{f_x} + \cos\lambda \cdot \cos\theta \cdot \frac{y-c_y}{f_y} + \sin\theta} \quad (19)$$

$$\begin{aligned}
 & -Z \cdot \sin\lambda \cdot \cos\theta \cdot \frac{x-c_x}{f_x} + Z \cdot \cos\lambda \cdot \cos\theta \cdot \frac{y-c_y}{f_y} + Z \cdot \sin\theta = \\
 & c_0 \cdot \sin\lambda \cdot \sin\theta \cdot \frac{x-c_x}{f_x} - c_0 \cdot \cos\lambda \cdot \sin\theta \cdot \frac{y-c_y}{f_y} + c_0 \cdot \cos\theta
 \end{aligned} \tag{20}$$

$$\begin{aligned}
 \sin\lambda \cdot (Z \cdot \cos\theta \cdot \frac{x-c_x}{f_x} + c_0 \cdot \sin\theta \cdot \frac{x-c_x}{f_x}) - \cos\lambda \cdot (Z \cdot \cos\theta \cdot \frac{y-c_y}{f_y} + c_0 \cdot \sin\theta \cdot \frac{y-c_y}{f_y}) = \\
 Z \cdot \sin\theta - c_0 \cdot \cos\theta
 \end{aligned} \tag{21}$$

We want to find a λ for which two points on the line $(x_1, y_1), (x_2, y_2)$ will have the same Z coordinate. This means we have to solve the following equation:

$$\begin{aligned}
 \sin\lambda \cdot (Z \cdot \cos\theta \cdot \frac{x_1-c_{x1}}{f_{x1}} + c_0 \cdot \sin\theta \cdot \frac{x_1-c_{x1}}{f_{x1}}) - \cos\lambda \cdot (Z \cdot \cos\theta \cdot \frac{y_1-c_{y1}}{f_{y1}} + c_0 \cdot \sin\theta \cdot \frac{y_1-c_{y1}}{f_{y1}}) = \\
 \sin\lambda \cdot (Z \cdot \cos\theta \cdot \frac{x_2-c_{x2}}{f_{x2}} + c_0 \cdot \sin\theta \cdot \frac{x_2-c_{x2}}{f_{x2}}) - \cos\lambda \cdot (Z \cdot \cos\theta \cdot \frac{y_2-c_{y2}}{f_{y2}} + c_0 \cdot \sin\theta \cdot \frac{y_2-c_{y2}}{f_{y2}})
 \end{aligned} \tag{22}$$

$$\begin{aligned}
 \sin\lambda \cdot (Z \cdot \cos\theta \cdot \frac{x_1-c_{x1}}{f_{x1}} + c_0 \cdot \sin\theta \cdot \frac{x_1-c_{x1}}{f_{x1}} - Z \cdot \cos\theta \cdot \frac{x_2-c_{x2}}{f_{x2}} + c_0 \cdot \sin\theta \cdot \frac{x_2-c_{x2}}{f_{x2}}) = \\
 \cos\lambda \cdot (Z \cdot \cos\theta \cdot \frac{y_1-c_{y1}}{f_{y1}} + c_0 \cdot \sin\theta \cdot \frac{y_1-c_{y1}}{f_{y1}} - Z \cdot \cos\theta \cdot \frac{y_2-c_{y2}}{f_{y2}} + c_0 \cdot \sin\theta \cdot \frac{y_2-c_{y2}}{f_{y2}})
 \end{aligned} \tag{23}$$

$$\begin{aligned}
 \sin\lambda \cdot (\frac{x_1-c_{x1}}{f_{x1}} \cdot (Z \cdot \cos\theta + c_0 \cdot \sin\theta) - \frac{x_2-c_{x2}}{f_{x2}} (Z \cdot \cos\theta + c_0 \cdot \sin\theta)) = \\
 \cos\lambda \cdot (\frac{y_1-c_{y1}}{f_{y1}} \cdot (Z \cdot \cos\theta + c_0 \cdot \sin\theta) - \frac{y_2-c_{y2}}{f_{y2}} \cdot (Z \cdot \cos\theta + c_0 \cdot \sin\theta))
 \end{aligned} \tag{24}$$

$$\sin\lambda \cdot (\frac{x_1-c_{x1}}{f_{x1}} - \frac{x_2-c_{x2}}{f_{x2}}) = \cos\lambda \cdot (\frac{y_1-c_{y1}}{f_{y1}} - \frac{y_2-c_{y2}}{f_{y2}}) \tag{25}$$

$$\tan\lambda = \frac{\frac{y_1-c_{y1}}{f_{y1}} - \frac{y_2-c_{y2}}{f_{y2}}}{\frac{x_1-c_{x1}}{f_{x1}} - \frac{x_2-c_{x2}}{f_{x2}}} \tag{26}$$

$$\lambda = \tan^{-1} \left(\frac{\frac{y_1-c_{y1}}{f_{y1}} - \frac{y_2-c_{y2}}{f_{y2}}}{\frac{x_1-c_{x1}}{f_{x1}} - \frac{x_2-c_{x2}}{f_{x2}}} \right) \tag{27}$$

This is essentially just the angle of the perceived line.

Bibliography

- [1] A. Jones et al. “Advances in computer vision for wildlife monitoring: Applications in tracking and behavior analysis”. In: *Nature Communications* (2021).
- [2] L. Smith et al. “Deep learning for automated tracking of animal movement from camera feed”. In: *Computers and Electronics in Agriculture* (2022).
- [3] H. Perez et al. “Camera-based monitoring of wildlife behavior using computer vision techniques”. In: *Applied Sciences* (2023).
- [4] T. Nguyen et al. “Automated multi-animal tracking using deep learning: Behavioral insights and ecological applications”. In: *Ecological Informatics* (2021).
- [5] Y. Chen et al. “Tracking animal behavior using advanced video analytics”. In: *Journal of Animal Ecology* (2020).
- [6] Edoardo Fazzari et al. “Animal Behavior Analysis Methods Using Deep Learning: A Survey”. In: *arXiv preprint arXiv:2405.14002* (2024).
- [7] Q. Fan et al. “Bottom-up Cattle Pose Estimation Via Concise Multi-Branch Network”. In: *Computers and Electronics in Agriculture* 211 (2023), p. 107945.
- [8] A. Nasiri et al. “Pose Estimation-Based Lameness Recognition in Broiler Using CNN-LSTM Network”. In: *Computers and Electronics in Agriculture* 197 (2022), p. 106931.
- [9] S. Wu et al. “Research Progress of Animal Behavior Recognition Based on Pose Estimation”. In: *Journal of China Agricultural University* 28 (2023), pp. 22–35.
- [10] X. Dai et al. “Animal Pose Estimation Based on 3D Priors”. In: *Applied Sciences* 13 (2023), p. 1466.
- [11] Cheng Fang et al. “Pose estimation and behavior classification of broiler chickens based on deep neural networks”. In: *Computers and Electronics in Agriculture* 180 (2021), p. 105863.

- [12] F. Zhang, M. Wang, and Z. Wang. “Construction of the Animal Skeletons Keypoint Detection Model Based on Transformer and Scale Fusion”. In: *Transactions of the Chinese Society of Agricultural Engineering* 23 (2021), pp. 179–185.
- [13] Y. Xue et al. “Posture Change Recognition of Lactating Sow by Using 2D-3D Convolution Feature Fusion”. In: *Transactions of the Chinese Society of Agricultural Engineering* 9 (2021), pp. 230–237.
- [14] Talmo D Pereira et al. “SLEAP: A deep learning system for multi-animal pose tracking”. In: *Nature methods* 19.4 (2022), pp. 486–495.
- [15] Pierre Karashchuk et al. “Anipose: A toolkit for robust markerless 3D pose estimation”. In: *Cell reports* 36.13 (2021).
- [16] Adam Gosztolai et al. “LiftPose3D, a deep learning-based approach for transforming two-dimensional to three-dimensional poses in laboratory animals”. In: *Nature methods* 18.8 (2021), pp. 975–981.
- [17] Shiting Xiao et al. “Multi-view tracking, re-id, and social network analysis of a flock of visually similar birds in an outdoor aviary”. In: *International Journal of Computer Vision* 131.6 (2023), pp. 1532–1549.
- [18] Suresh Neethirajan. “ChickTrack – A quantitative tracking tool for measuring chicken activity”. In: *Measurement* 191 (2022), p. 110819. ISSN: 0263-2241. DOI: <https://doi.org/10.1016/j.measurement.2022.110819>. URL: <https://www.sciencedirect.com/science/article/pii/S0263224122001154>.
- [19] Glenn Jocher. *ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements*. <https://github.com/ultralytics/yolov5>. Version v3.1. Oct. 2020. DOI: 10.5281/zenodo.4154370. URL: <https://doi.org/10.5281/zenodo.4154370>.
- [20] Rudolph Emil Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.
- [21] Mosam Dabhi et al. “MBW: Multi-view Bootstrapping in the Wild”. In: *Advances in neural information processing systems* 35 (2022), pp. 3039–3051.
- [22] Shida Zhao et al. “Pose Estimation and Behavior Classification of Jinling White Duck Based on Improved HRNet”. In: *Animals* 13.18 (2023), p. 2878.

- [23] Ke Sun et al. “Deep high-resolution representation learning for human pose estimation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA, 2019, pp. 5693–5703.
- [24] Jessy Lauer et al. “Multi-animal pose estimation and tracking with DeepLabCut”. In: *bioRxiv 2021.04.30.442096* (2021). DOI: 10.1101/2021.04.30.442096. eprint: <https://www.biorxiv.org/content/early/2021/04/30/2021.04.30.442096.full.pdf>.
- [25] Jacob M Graving. “Computer Vision and Deep Learning Methods for Measuring and Modeling Animal Behavior”. In: (2021).
- [26] István Fodor et al. “Automated pose estimation reveals walking characteristics associated with lameness in broilers”. In: *Poultry science* 102.8 (2023), p. 102787.
- [27] Yuxin Wu et al. *Detectron2*. <https://github.com/facebookresearch/detectron2>. 2019.
- [28] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [29] Kaiming He et al. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV].
- [30] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].
- [31] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. “Per-Pixel Classification is Not All You Need for Semantic Segmentation”. In: 2021.
- [32] Bowen Cheng et al. *Mask2Former for Video Instance Segmentation*. 2021. arXiv: 2112.10764 [cs.CV].
- [33] Shusheng Yang et al. *Temporally Efficient Vision Transformer for Video Instance Segmentation*. 2022. arXiv: 2204.08412 [cs.CV].
- [34] Joseph Redmon et al. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

- [35] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: <https://github.com/ultralytics/ultralytics>.
- [36] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. *YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information*. 2024. arXiv: 2402.13616 [cs.CV]. URL: <https://arxiv.org/abs/2402.13616>.
- [37] Glenn Jocher and Jing Qiu. *Ultralytics YOLO11*. Version 11.0.0. 2024. URL: <https://github.com/ultralytics/ultralytics>.
- [38] Alex Bewley et al. “Simple online and realtime tracking”. In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.
- [39] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. “Simple Online and Realtime Tracking with a Deep Association Metric”. In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962.
- [40] Nicolai Wojke and Alex Bewley. “Deep Cosine Metric Learning for Person Re-identification”. In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 748–756. DOI: 10.1109/WACV.2018.00087.
- [41] MMFlow Contributors. *MMFlow: OpenMMLab Optical Flow Toolbox and Benchmark*. <https://github.com/open-mmlab/mmdetection>. 2021.
- [42] Rudolph Emil Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Transactions of the ASME—Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.
- [43] Mark Everingham et al. *The PASCAL Visual Object Classes (VOC) challenge*. 2010.
- [44] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.
- [45] Ho Kei Cheng, Yu-Wing Tai, and Chi-Keung Tang. “Modular Interactive Video Object Segmentation: Interaction-to-Mask, Propagation and Difference-Aware Fusion”. In: *CVPR*. 2021.
- [46] Andreas Specker and Jürgen Beyerer. “Reidtrack: Reid-only multi-target multi-camera tracking”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 5442–5452.

- [47] Alexandru Niculescu-Mizil, Deep Patel, and Iain Melvin. “MCTR: Multi Camera Tracking Transformer”. In: *arXiv preprint arXiv:2408.13243* (2024).
- [48] Temitope Ibrahim Amosa et al. “Multi-camera multi-object tracking: A review of current trends and future advances”. In: *Neurocomputing* 552 (2023), p. 126558. ISSN: 0925-2312. DOI: <https://doi.org/10.1016/j.neucom.2023.126558>. URL: <https://www.sciencedirect.com/science/article/pii/S0925231223006811>.
- [49] Alejandro López-Cifuentes et al. “Semantic-driven multi-camera pedestrian detection”. In: *Knowledge and Information Systems* 64.5 (2022), pp. 1211–1237.
- [50] Olly Styles et al. “Multi-camera trajectory forecasting: Pedestrian trajectory prediction in a network of cameras”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 1016–1017.
- [51] Zhengyou Zhang. “A flexible new technique for camera calibration”. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.11 (2000), pp. 1330–1334.
- [52] Learning OpenCV. by Gary Bradski and Adrian Kaehler. 2008.
- [53] DC Brown. *Close-range camera calibration Photogeom*. 1971.
- [54] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV]. URL: <https://arxiv.org/abs/1405.0312>.
- [55] Thomas Steinecker and Hans-Joachim Wuensche. “A Simple and Model-Free Path Filtering Algorithm for Smoothing and Accuracy”. In: *2023 IEEE Intelligent Vehicles Symposium (IV)*. 2023, pp. 1–7. DOI: 10.1109/IV55152.2023.10186704.