Noi Bar & Noy Shabtay 12th September 2021

3D Pose Estimation of Bearded Dragons for Visual Exploration Research

Chapter 1 – Introduction

General background

Like humans, bearded dragons are foveate animals that utilize eye movements to explore space in high resolution. This offers insights into internal neuronal processes[1,2]. Although almost all animal eye movement research is done while the animal's head is fixed, it was found that animals have a strong connection between eye movements and head and body movements. For example, mice, where 97% of their eye movements came along with head and body movements[3].

Realizing that makes clear that fixing the animal's head during a visual task might unfaithfully present its sensing process in the real world. Thus, our goal is to help create the technology to examine the dragons' 3D visual scanning process while they are free to move. The general motivation is that our technological solution and methods will help understand animals' visual attention process and infer from it about additional inner processes (e.g., space sampling process, ranking of the importance of objects and features in space). We think that dragons are a good animal model choice for visual acquisition experiments due to their slight and distinct movements, unlike mice which are often too energetic and frantic.

The final goal of this project is to build a research tool that produces a 3D gaze trace over time of the dragons while they exhibit a more natural behaviour when compared to their behaviour when measured with today's eye-tracking tools. This new system will allow acquisition of a complete visual exploration behaviour profile of the animal, by the construction of three stages: (1) locating the eyes of the animal in space, (2) modelling the angle of view through changes in the pupil, and (3) connecting all the information together to build an accurate gaze vector for each eye. This paper focuses on solving the first stage of this process.

The system settings

Experiments are held inside an arena consisting of various stimuli such as a large screen, various objects, a ramp, a food system that releases food according to the animal's desired behaviours and more. The arena is surrounded by four IR sensitive cameras: two in the front wall (left and right), one in the back and a wide-angle camera above the center of the arena.

Inside the arena, the dragon can move freely while a special device is mounted to its head. This device, consisting of a 3D printed body, four IR LEDs, tiny cameras, and mirrors, continuously records the dragon's pupils. IR spectrum is used to not interfere with the animal's vision while filming (IR is outside the visible light spectrum of dragons). Each dragon's head is 3D scanned by Photogrammetry in order to adjust the 3D printed body to the Dragon's head via a thread. The thread allows attaching the assembly properly when needed. The advantage of this method is that there is no need for animal surgery. The disadvantage is when the skin regenerates, the thread falls and must be re-glued. All the cameras that capture the arena, as well as those on the dragon's head, are synchronized by a GPU based filming system.

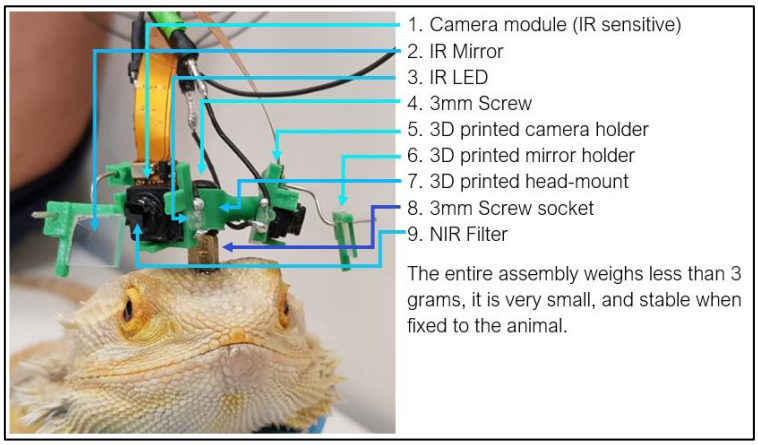


Fig. 1: Details of the assembly components

Chapter 2 – Review of literature and related works

3D pose estimation is generally regarded as the task of predicting the articulated joint locations of an object from an image or a sequence of images of that object. Due to its wide range of potential applications, 3D pose estimation is a fundamental and active research direction in the area of computer vision. In our context, this problem is also extended to 3D location estimation of the device inside the arena. That is, our goal is to describe the four LEDs locations inside the arena relative to an agreed coordinate system and calculate the dragon's head pose in terms of orientation relative to an agreed baseline reference pose.

In such problems, the image data from which the pose of an object is determined is usually a single image, a stereo image pair (Computer Stereo Vision) or more (Triangulation), or an image sequence where the camera is typically moving with a known velocity. The objects which are considered can be rather general, including a living being or body parts, e.g., a head or hands. The methods which are used for determining the pose of an object, however, are usually specific for a class of objects and cannot generally be expected to work well for other types of objects.

For 3D location and depth estimation there are various techniques from hardware solutions to software solutions using classical methods and Machine Learning. Software solutions mostly include multiple images of the same scene with slight displacements such as Computer Stereo Vision[4] while creating a disparity map and Triangulation (also referred as Reconstruction or Intersection)[5] solving a more general correspondence problem. The common idea behind multiple image methods is that by matching key points that are common with each image, we can reconstruct a 3D model of the scene. For example, a best-known algorithm for this task is Scale-Invariant Feature Transform (SIFT)[6].

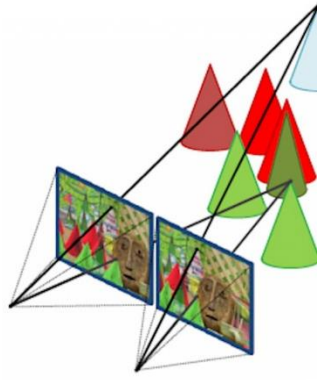


Fig. 2: Visual illustration of Stereo Vision

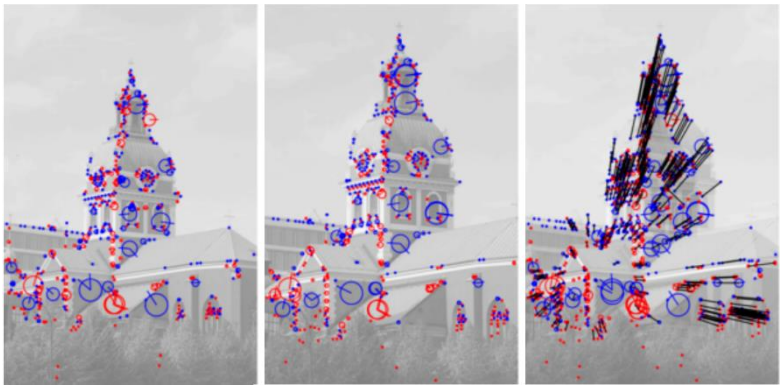


Fig. 3: A SIFT demonstration - Interest points detected from two images of the same scene with the computed image matches drawn as black lines between corresponding interest points. The blue and red arrows at the centers of the circles illustrate the orientation estimates obtained from peaks in local orientation histograms around the interest points.

There are several single-image depth estimation methods as well. These methods usually involve a neural network trained on pairs of images and their depth maps. Although such methods can be easy to construct and provide decent accuracy sometimes, they are rarely used in a multiple-image setting such as ours.

For 3D orientation estimation the most used methods are triangulation ones, probably because they provide a general solution that also helps finding the 3D location at the same time. At a quite late stage in our project we were acquainted with the most relevant work for our purposes – Anipose[7] – an new open-source unpublished-yet toolkit for markerless 3D tracking of animal behaviour from multiple camera views. It leverages the machine learning toolbox DeepLabCut to track key-points in 2D (as we independently decided to use it also), then triangulates across camera views to estimate 3D pose. An advantage of this method is that after triangulation it is possible to compute the reprojection error given the 2D coordinates associated with each key-point from each camera and the 3D coordinates determined from all the cameras. The reprojection error describes how well the 2D projections of a triangulated 3D point match its corresponding 2D key points in every camera view.

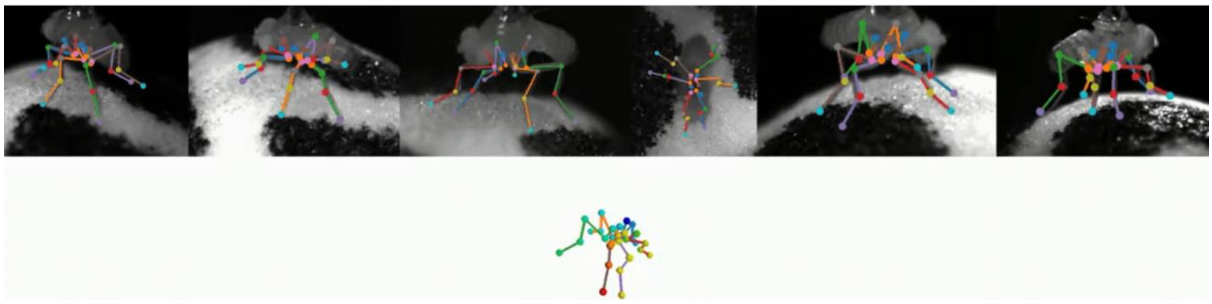


Fig. 4: An Anipose demonstration on a fly video.

Nowadays there are new and powerful deep learning driven techniques for pose estimation, especially in humans (including a recently collected large-scale datasets for human 3D pose estimation). These techniques have continued making great progress in the problem of human pose estimation, especially from 2D images. However, performance of these methods is not yet satisfactory, largely due to the lack of sufficient "in the wild" datasets, depth ambiguities, a still unmet demand for rich posture information (such as translations and rotations), and a large searching state space for each joint (representing a discretised 3D space)[8].

In our setting we don't have enough data to use deep learning only for estimating the dragon's head pose. Additionally, we reduced the problem of determining the head pose to locating the 4 LEDs in the assembly, thus our problem focuses on 3D location and 3D rotations only (without considering the variability of the animal's body in each position). We started with triangulation-based exploration, but in practice it didn't go well for us, thus we stopped trying these ones at an early stage.

Our final solution relies on the understanding that we have additional information, that the above-described methods don't have – an accurate 3D model of the assembly. We found in the literature a problem called Perspective-n-Points (PnP) for estimating the pose of a calibrated camera given a set of *n* 3D points in the world and their corresponding 2D projections in the image[9]. The camera's placement in space consists of 6 degrees-of-freedom (DOF) which are made up of the orientation (roll, pitch, and yaw) and 3D location (translation) of the camera with respect to the world. This problem originates from camera calibration and has many applications in computer vision. In our case, we solve P4P (n = #LEDs = 4) problem by transferring from the assembly coordinates to camera ones for each camera video. Later we discovered this approach is similar to other ones uses a 3D model such as POSIT[10] and others[11,12].

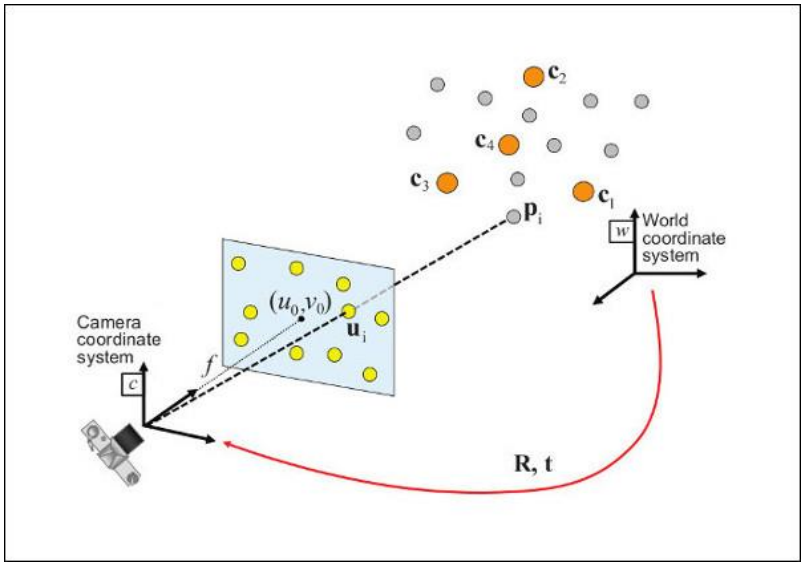


Fig. 5: PnP problem formulation: Given a set of correspondences between 3D points p_i expressed in a world reference frame, and their 2D projections u_i onto the image, we seek to retrieve the pose (R and t) of the camera w.r.t. the world and the focal length f.

As far as we have seen, PnP problems are usually solved from a single image with respect to the camera's coordinates. As a final step for solving our problem, we solve another PnP problem transferring each video from its camera coordinates to agreed arena coordinates, then averaging them smartly (see Methods p. 7-9). We found this solution to be the most accurate and sufficient one to be included in the lab's pipeline (see Results p. 11).

Chapter 3 – Methods

2D LEDs Detection & Tracking for each camera

Our first attempts were with traditional computer vision tools powered by the OpenCV library, such as detection based on brightness threshold, morphological transformations, and tracking techniques. We found that with these approaches there are too many edge cases with specific angles (some LEDs are sometimes concealed , two close LEDs sometimes detected as one, light smearing, etc.) and dealing with them properly was hard.

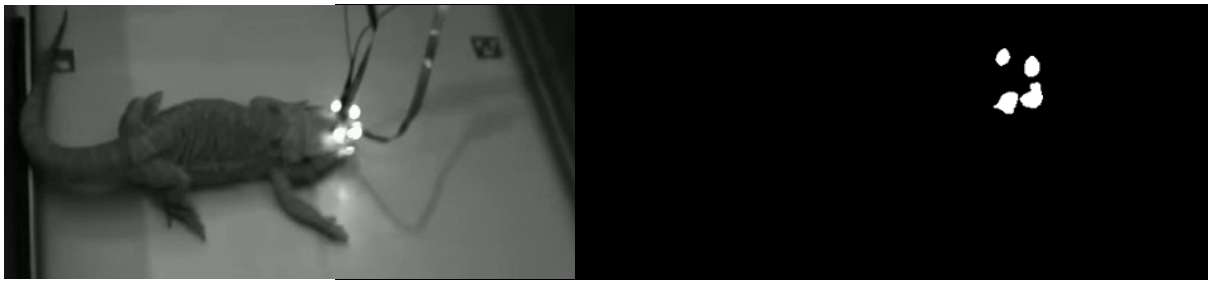


Fig. 6: An experiment frame with it's our early-stage approach.

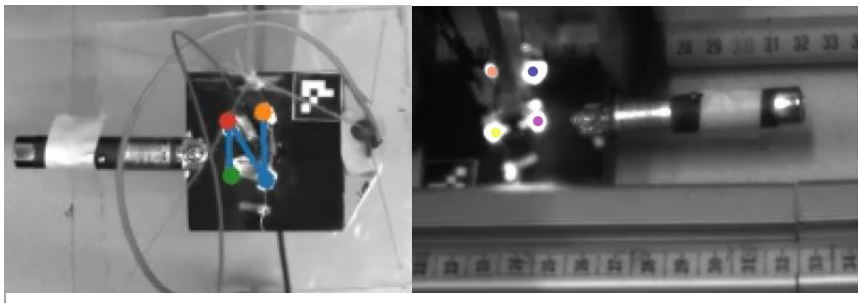


Fig. 7: A DeepLabCut detection of the four LEDs structure (left), and human labeling of the LEDs under conditions of light pollution and partial concealment (right).

Our current solution uses a supervised deep learning approach using DeepLabCut (DLC) – a software package for animal 2D pose estimation. The DLC toolbox is widely known and commonly used for quantifying behaviour for neuroscience purposes and its major strength is achieving excellent tracking performance on test frames, comparable to human accuracy with minimal training data[13]. Although we first thought that using a deep Learning approach might be an overkill for this problem, we found it much more flexible, and it significantly outperformed our previous approach by making it much easier to distinguish between each one of the LEDs. In its protocol, it is even suggested that DLC should be good for 3D pose estimation via multi-camera use[14], and it was chosen also as the detection implementation of Anipose – an unpublished-yet toolkit for animal 3D pose estimation.

About DeepLabCut Architecture

DeepLabCut is a deep convolutional network combining two key ingredients from algorithms for object recognition and semantic segmentation: pretrained ResNets and deconvolutional layers[15]. The network consists of a variant of ResNets, whose weights were trained on a popular, large-scale object recognition benchmark called ImageNet, on which it achieves excellent performance[16]. Instead of the classification layer at the output of the ResNet, deconvolutional layers are used to up-sample the visual information and produce spatial probability densities. For each body part, its probability density represents the 'evidence' that a body part is in a particular location. To fine-tune the network for a particular task, its weights are trained on labelled data, which consist of frames and the accompanying annotated body part locations (or other objects of interest in the frame).

During training, the weights are adjusted in an iterative manner such that for a given frame the network assigns high probabilities to labelled body part locations and low probabilities elsewhere. Thereby, the network is rewired and 'learns' feature detectors for the labelled body parts. As a result of the initialization with the ResNet pretrained on ImageNet, this rewiring is robust and data efficient.

Cameras' calibration

An essential step in accurate 3D pose estimation is camera calibration, which precisely determines the relative location and various parameters for each camera (i.e., the focal length and distortion). We implemented an automated procedure that calibrates the cameras from simultaneously acquired videos of a standard checkerboard moved by hand through the cameras' field of view. We chose the checkerboard method instead of other methods (such as ArUco markers or ChArUco board) because it's a known technique used in the lab before, and a well-known OpenCV standard.

Fig. 8: An original distorted frame from the experiment arena.

Fig. 9: An undistorted frame from the arena using our calibration module.

This calibration needs to happen only once, unless changes are made to the camera setup itself (e.g. focus change, replacing cameras). Although cameras of the same model should in theory have the same intrinsic and extrinsic properties, we found that in practice there are differences between the cameras in the parameters needed for each camera. Thus, we perform the calibration calculations for each camera individually.

The quality of the calibration depends on the number of frames with clear checkerboard view at different angles, and we were able to measure it using measurement videos we made with known locations and differences. The calibration is calculated by detecting key points on the calibration board automatically using OpenCV, based on the board's geometric regularities (checkerboard grid pattern).

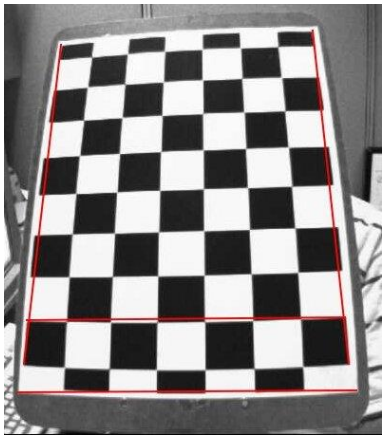


Fig. 10: Demonstration of a distorted image of a checkerboard using OpenCV.

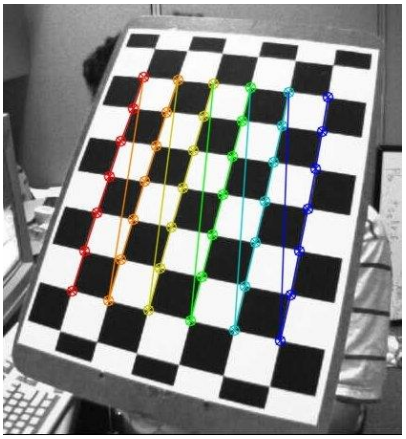


Fig. 11: a chessboard 7x6 grid pattern detected and drawn using OpenCV.

3D Positioning

The topic of 3D pose estimation is reviewed at length in the Related Works chapter, describing what led us to choose a 3D-model-based approach. Our solution consists of a series of solutions to Perspective-n-Points (PnP) problems. PnP is the problem of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. In our solution, we solve two PnP problems: first we solve a P4P (n = #LEDs = 4) problem by translating from the assembly coordinates to camera coordinates for each camera video. Then we solve another PnP problem translating each video from its camera coordinates to agreed arena coordinates by using markers with known locations that appear in the calibration video. Those markers are set by using 2 checkerboards placed at two opposite corners of the arna as described in our Arena Experiment Calibration Manual (see Appendix 1 p. 13-14). If there was a change in the position of the cameras, it is necessary to perform a camera-to-arena recalibration. We found this solution to work best for us (see Results p. 11).

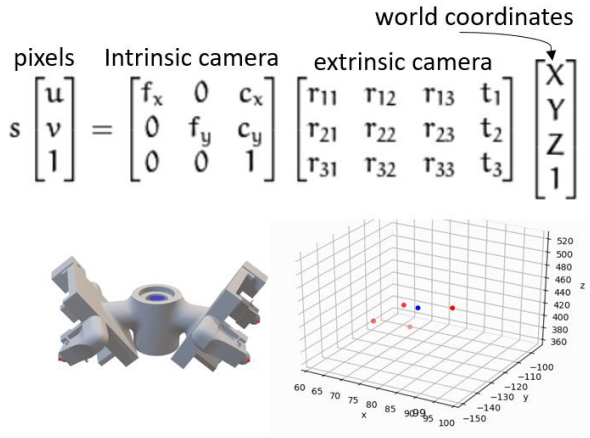


Fig. 12: an assembly PnP solution. Pixels known from the detection phase, Intrinsic and Extrinsic matrices from the cameras' calibration.

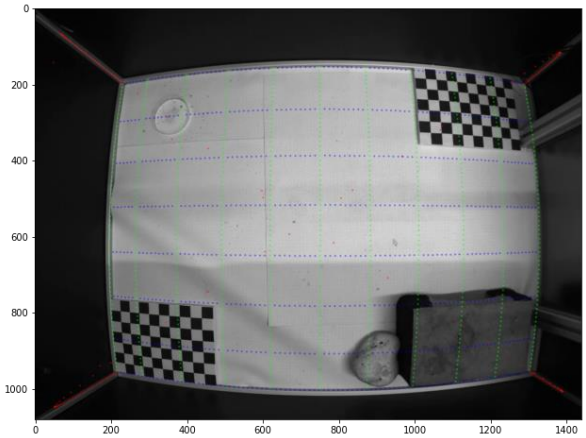


Fig. 13: Simulation of the arena 3D space drawn on an original top-camera video frame.

After performing this series of PnP solutions, we get from each camera a set of 3D coordinates for all the LEDs. The four LEDs are enough to determine a plane in the 3D space, and with the assembly center location, the dragon's head location and direction vector is determined as a single unique solution. We chose to define the direction vector as the normalized vector in space which connects the assembly center location point and the middle point of the line containing the two front LEDs' location.

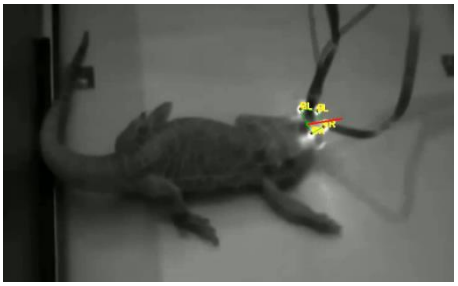


Fig. 14: a frame of dragon with 4 LEDs detected with a drawn of its head direction.

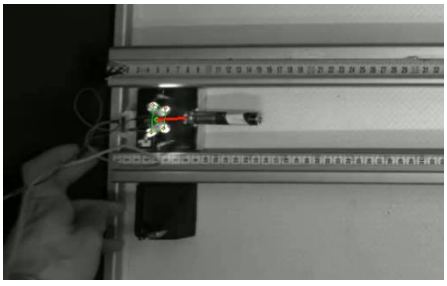


Fig. 15: a calibration video frame with a drawn direction vector, following the assembly movement in space.

Combining the cameras

Since the location retrieved from the PnP process (3D Positioning part) tends to be noisy for each camera in practice, we combine estimations from all the cameras to a final more accurate estimation before applying the last phase - filtering (see Methods p. 10).

At first, we tried to produce less noisy results for each camera thanks to better calibration process, but our effort in this direction didn't lead to a major breakthrough – even the best calibrated camera we had was too noisy at some points, also when all LEDs detected in a very high likelihood (above 0.95). Also fine tuning the calibration process can be exhausting and unrealistic for real daily use in the lab (where cameras in the arna tends to move and get out of focus). Saying that makes it clear that our final output needs to be somehow an average of all cameras' PnP solutions.

In this section we describe four different averaging ideas and their impact on the result compared to the ground truth location we have from the measurement videos we made. All versions assume a Gaussian noise channel as it is commonly used in image processing tasks.

For each frame, the data to be considered by all the methods is only the one whose camera detection process found all the four LEDs with a certain confidence controlled by a user-defined threshold relative to DLC detection likelihood outputs. Thus, all samples in the combination are defined as good and reasonable detections.

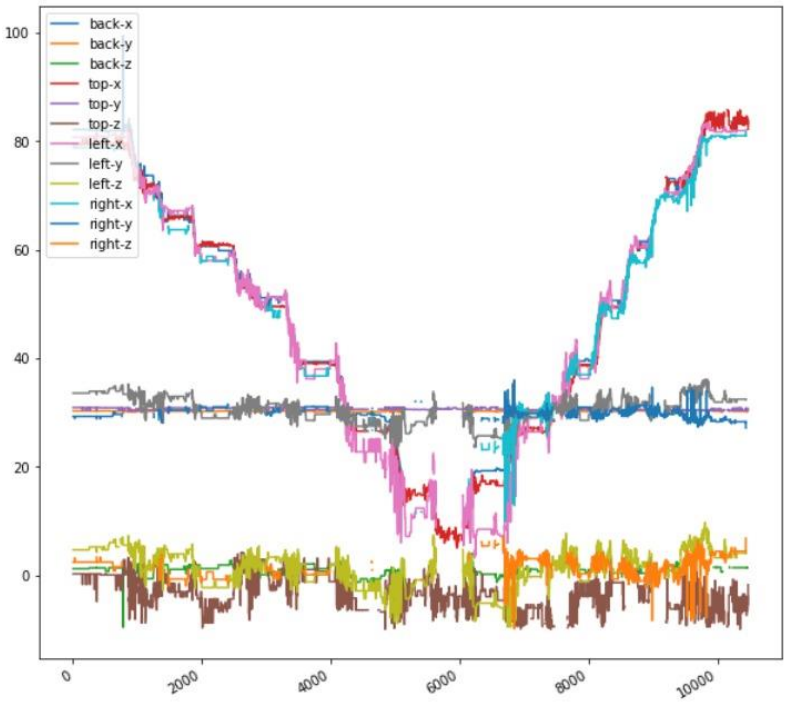


Fig. 16: a plot presenting all cameras' 3D estimations for the assembly's center for each axis (cm) over time (frames) for a length (X axis) movement video.

Simple Mean

Arithmetic mean is performed over the set of all good and reasonable 3D estimations in each frame. This method found to be the most accurate one (See results p. 11) and smooths the data the most over time.

Axis Standard Deviation Threshold

Points that are above or below one standard deviation from the mean are removed for each axis independently. This solution sometimes significantly reduced noise and sometimes increased it. We less recommend this method, as it treats each axis individually, thus in theory and practice each coordinate of the final estimation can be determined by a different combination of cameras.

Distance from Pseudo Centroid Threshold

In this method a pseudo-point is created as in the Simple Mean method. For each 3D estimation we calculate its Euclidean distance from the pseudo-point. Estimateions that are far from the pseudo-point by more than the distance mean plus the standard deviation are removed from the calculation. Then an arithmetic mean is performed over the remaining estimations. We found this method to be less smooth than the methods described above but it still noticeably reduces noise.

Distance from Pseudo Centroid Threshold - converges iterative version

The same process as in the last method is performed, but in an iterative way for multiple times till convergence (where none of the last remaining cameras is being removed). This process is promised to convergence after at most 3 iterations because we use four cameras. Empirical data shows that the combination for each frame converges to one or two cameras, and the final outcome is the noisiest between all the above-described methods.

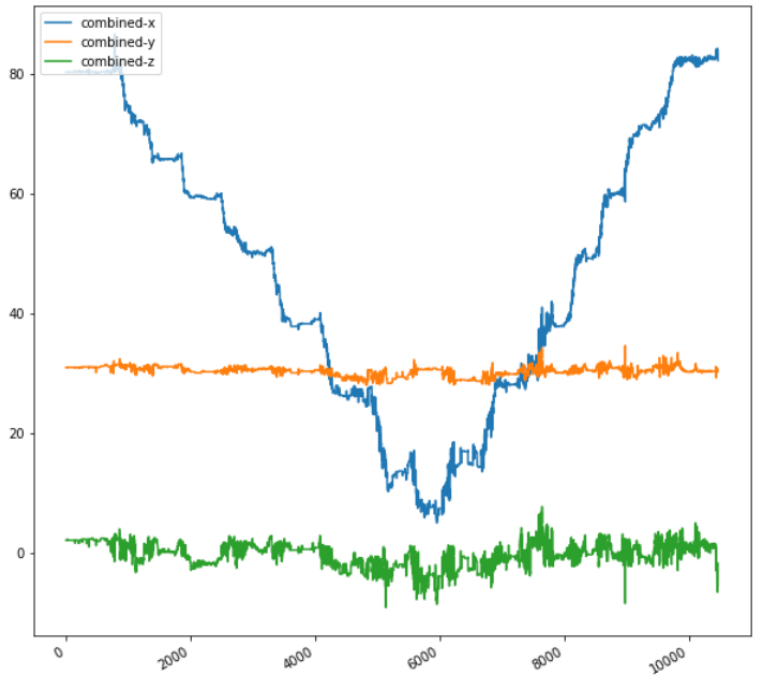


Fig. 17: a plot presenting the mean combination of all estimations in fig. 16.

Filtering

Filters are widely used in computer vision in many fields such as detection, movement tracking, and 3D estimations. It is commonly suggested to reduce noise and to produce coherent results over time by software toolkits like Google MediaPipe, and in toolkits we examined such as in DeepLabCut and Anipose.

2D Filters over the LEDs detections in the videos

We tried 3 different 2D filters on the detections of the LEDs overall: (1) **Median** and (2) **ARIMA** (AutoRegressive Integrated Moving Average) filters using the DLC toolkit itself, and (3) **Viterbi** filter via the Anipose toolkit. All of them shown similar results in our measurement videos – a modest contribution to accuracy and noise reduction.

The Meidan and ARIMA filters were performed on the data itself, but the use of the Viterbi filter was different – it functiones as a threshold filter. Predicted labels that do not fall within the threshold are removed and replaced with a new prediction that is determined by interpolating using a cubic spline. Although this method was chosen to be the filtering implementation in the Anipose toolkit, in our calibration data it didn't seem to be more effective than other filters. Therefore, due to the desire to avoid using software that has not yet been officially released, we decided to keep the existing implementations using DLC.

3D Filters over the final location estimation

As a final stage the cameras-combined estimation is being filtered by a 3D Savitzky–Golay (savgol) filter as it is used in image smoothing[17] and for 3D usages[18]. Applying this 3D filter found to be helpful for improving the accuracy (see Results p. 11).

Since most the verification was done on our measurement videos, and due to the fact that we straggled getting real animal videos, it is important to emphasize that it’s soon to determine which combination method and usage of filtering is the best for future real use.

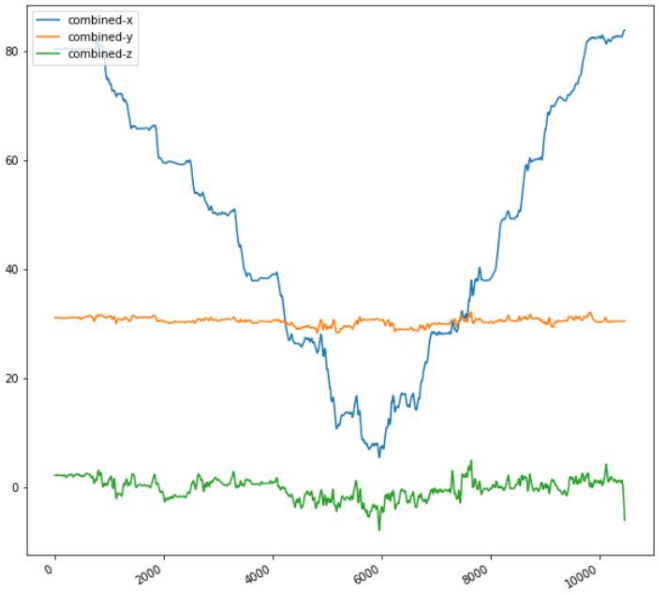


Fig. 18: a plot presenting the final filtered estimations for fig. 16’s video with mean cameras combination (such as in fig. 17).

Chapter 4 – Results

Through the working on the project, we came up with several ideas to verify our results, and we produce measurement videos of the device. These videos start filming the assembly from a known pre-measured location, then the assembly is forced to move on one axis at a time (arena's length, width, and height) where measuring tapes let us know the position on the movement axis. Also, we created videos of the device located in one place and spinning with a known drift. We created such videos twice, after changes in the cameras' location and focus, to check that our calibration process works sufficiently. In these videos we used other measurement tools such as accelerometer, laser mimicking the head's looking vector, and also consider working with a gyroscope, but from all these methods, following the measuring tape in the undistorted videos was the best verification way we found to work with (gyroscope isn't sensitive enough; accelerometer is too noisy; laser is problematic for determining the device's 3D location).

Our accuracy is measured by Euclidean distance from the measured ground truth point to the device's center estimated point. In our measurement videos we found that the average Euclidean distance error in randomly selected frames is:

- 2.19 cm (X=0.85, Y=0.79, Z=1.62) for 2D-3D filtered data with mean cameras combination.
- 2.21 cm (X=0.82, Y=0.77, Z=1.67) for unfiltered data with mean cameras combination.
- 2.64 cm (X=0.94, Y=0.80, Z=2.15) for unfiltered data with Euclidean cameras combination.

For comparison, the dimensions of the arena are X=97cm, Y=69 cm, Z=45 cm.

The filtered data with the mean average provides the most accurate results with the best noise cancelling and smoothing over time, although it's soon to determine that this is the best configuration to work with in real experiments settings.

We noticed that Z axis is always the major error contributor. It makes sense since our calibration process focuses more on Axis X and Y as most of the calibration takes place on checkerboards on the arena floor (see Appendix 1 p. 13-14). We believe this issue can be addressed by expending the calibration process to the Z axis (e.g., putting another checkerboard on an arena wall).

In addition, we discover that our LEDs detections were pretty good, and if not – the DLC network can be re-trained to achieve great results. That is, very-good-enough 2D detection is easily and likely to get. This understanding reveals that most of the noise is created when transferring from 2D to 3D through the PnP solutions process. Therefore, we developed a way to calculate some kind of a Reprojection Error as used in Triangulation implementations. After a camera-to-arena calibration is done, a sanity check is running for determine each camera location. We found that the average error of the measurement videos is around the cameras' location reconstruction error – 2.85 cm (Right=1.77, Left=2.68, Top=2.74, Back=4.2). These findings are in correlation to the focus seen in each camera - all cameras were poorly focused in our measurement videos except the right one. Thus, we believe that with better focus and a new calibration accordingly, it's possible to get significantly lower error (< 2 cm).

Chapter 5 – About the System

General

All code in the system is written in Python (https://www.python.org/) and provided in a Jupyter Notebook (https://jupyter.org/) allowing easy-to-use interface and modular structure for future use (a plug & play environment for each phase in the process). The project code consists of 12 python files, one DLC configured detection network and 2 notebooks (one for DLC training and use and one for rest of the pipeline). Videos are always processed using the OpenCV library (https://opencv.org/).

Detection

A DLC network trained over 608 real animal experiment frames is provided. The Training notebook enables the user to re-train the network more on new labeled data if needed. The likelihood threshold for the LEDs detection phase can be controlled in the Pose Analysis notebook. This phase output is a csv file for each video containing the LEDs location with likelihood for each frame.

Cameras' Calibration

We find each camera's intrinsic and extrinsic properties using the checkerboard method via OpenCV. When the camera setup changes, the user can use our pipeline's calibration module. Inside the module one can enter a path to a calibration video which follows the Arena Experiment Calibration Manual we created and to get frames sampled from it for each camera. Then the user selects the relevant frames which have a clear and full view of the checkerboard. When the user finishes selecting the frames, the pipeline starts by detecting key-points on the calibration board automatically based on the board's geometric regularities (checkerboard grid pattern) using OpenCV and outputs all camera's intrinsic and extrinsic matrices to a file the pipeline would later use for undistortion and 3D positioning estimations.

3D Estimation

A single PnP problem is solved by the OpenCV implementation. The camera-to-arena calibration is made with a UI we designed enabling automated calibration if a checkerboard was detected, or a manual one asking the user to mark certain points in the arena. After this calibration the user get a sanity check result that is the estimations of the cameras' location, enabling to check if the calibration is satisfying. After the calibration everything is done automatically, creating a csv file containing each camera's estimations.

Filtering

Savgol filter option uses the SciPy.signal API (https://docs.scipy.org/doc/scipy/reference/signal.html).

Median and Autoregressive integrated moving average (ARIMA) filters options for the 2D LEDs detections are done through the DLC API.

The experiments with the Viterbi 2D filter and Cubic Spline Interpolation, which were not implemented in the final system, were done using AniPose (https://anipose.readthedocs.io/).

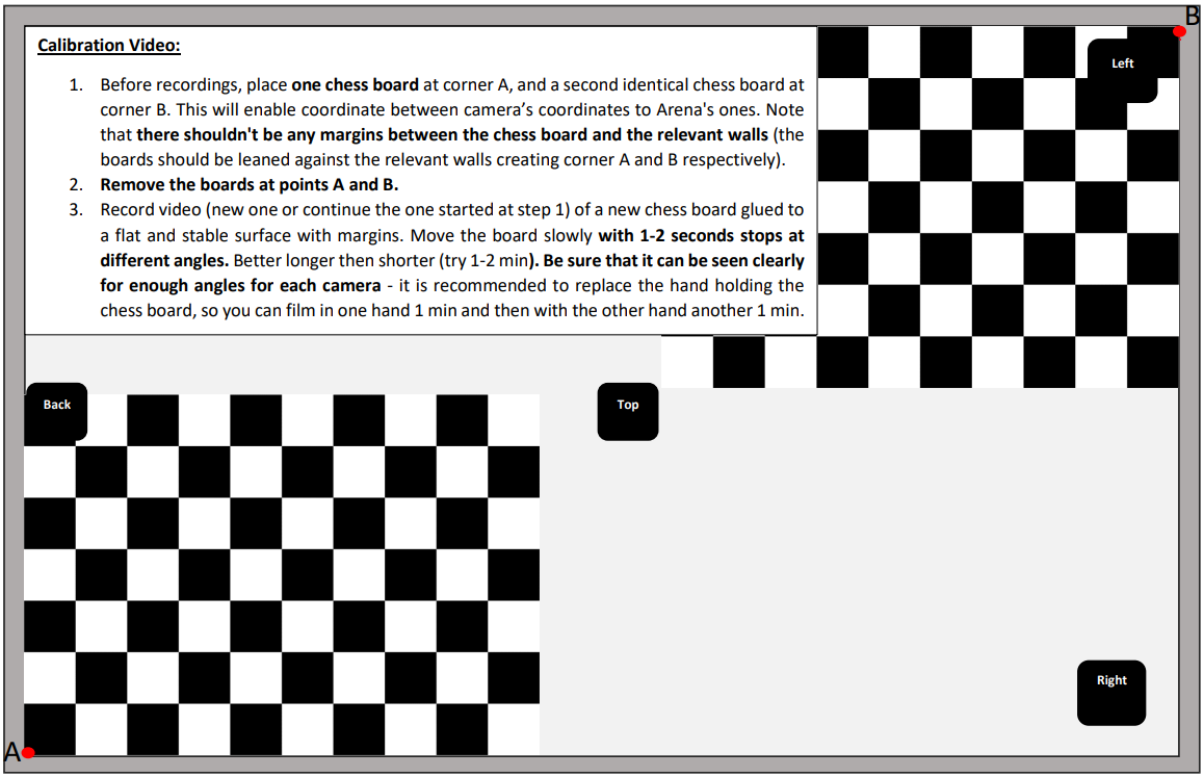
Analytics

Analytical analysis and manipulations on the data, as well as the work with the csv files and the combination of all the estimations were made using Pandas (https://pandas.pydata.org/) and NumPy (https://numpy.org/) libraries.

The analytical module (plots of the data from the DLC detections, 3D Head's location and vector, filter effects etc.) was built using Matplotlib library (https://matplotlib.org/).

Appendix

1. Arena Experiment Calibration Manual



Experiment measurements (all in cm)

(see figure on the next page)

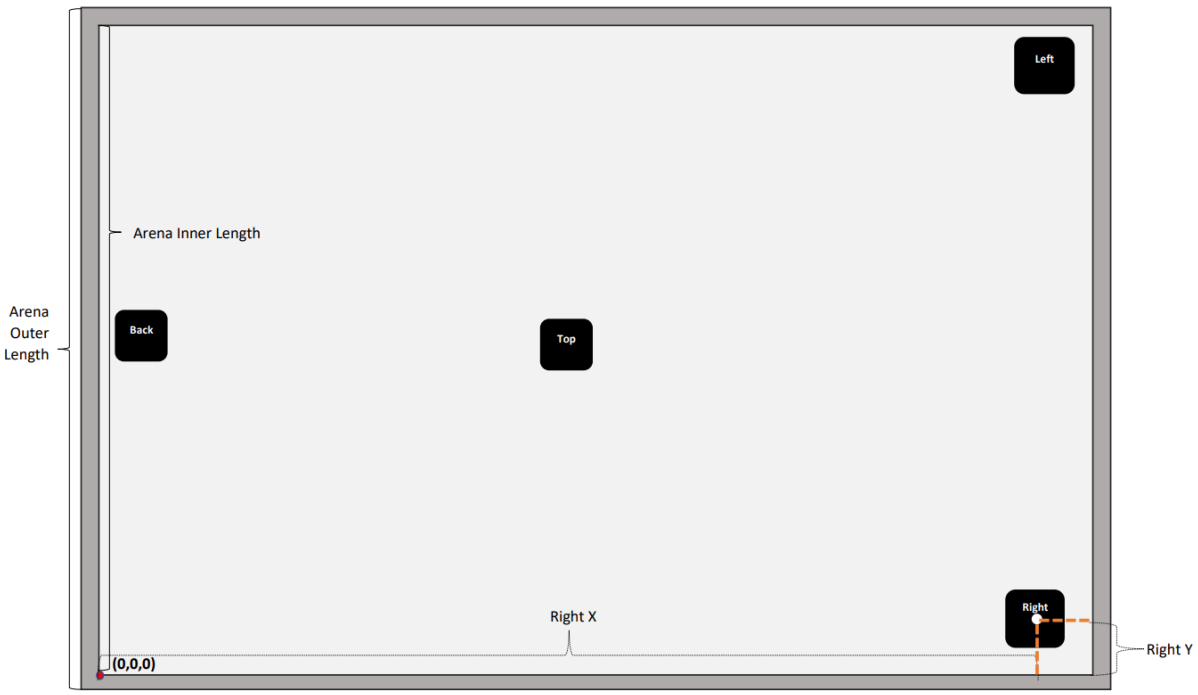
Date: _____

Time: _____

Arena	Inner	Outer
Width		
Length		
Height		

Cameras	Back	Top	Left	Right
X				
Y				
Z				

Corner Chessboards	Size
Side length of a square	
Whole board size (#length_squares X #width_squares)	


Left
Arena Inner Length
Arena
Outer
Length
Back
Top
Right X
Right
Right Y
(0,0,0)

Bibliography

1. Engbert, R. *Fixational Eye Movements*. *Dynamical Models In Neurocognitive Psychology* (2021). doi:10.1007/978-3-030-67299-7_2.

2. Martinez-Conde, S., Macknik, S. L. & Hubel, D. H. The role of fixational eye movements in visual perception. *Nat. Rev. Neurosci.* **5**, 229–240 (2004).

3. Wallace, D. J. *et al.* Rats maintain an overhead binocular field at the expense of constant fusion. *Nature* **498**, 65–69 (2013).

4. Clapworthy, G., Viceconti, M., Coveney, P. V. & Kohl, P. Philosophical Transactions of the Royal Society: Editorial. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* **366**, 2975–2978 (2008).

5. Hartley, R. I. & Sturm, P. Triangulation. *Comput. Vis. Image Underst.* **68**, 146–157 (1997).

6. Lindeberg, T. Scale Invariant Feature Transform. *Scholarpedia* **7**, 10491 (2012).

7. Dickinson, E. S., Rupp, K. L., Brunton, B. W., Azim, E. & Tuthill, J. C. Anipose : a Toolkit for Robust Markerless. (2020).

8. Wang, J. *et al.* Deep 3D human pose estimation : A review. *Comput. Vis. Image Underst.* **210**, 103225 (2021).

9. Wu, Y. PnP problem revisited. *J. Math. Imaging Vis.* **24**, 131–141 (2006).

10. Bertok, K. & Fazekas, A. A robust head pose estimation method based on POSIT algorithm. (2016).

11. Grest, D., Petersen, T. & Kr, V. A Comparison of Iterative 2D-3D Pose Estimation Methods for Real-Time Applications. 706–715 (2009).

12. Tang, M., Peng, X., Zhang, Z., Zhang, Y. & Duan, Y. 3D Pose estimation by matching CAD data to single image. *MIPPR 2005 SAR Multispectral Image Process.* **6043**, 60431L (2005).

13. Mathis, A. *et al.* DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. *Nat. Neurosci.* **21**, 1281–1289 (2018).

14. Nath, T. *et al.* Using DeepLabCut for 3D markerless pose estimation across species and behaviors. *Nat. Protoc.* **14**, 2152–2176 (2019).

15. Insafutdinov, E., Pishchulin, L., Andres, B., Andriluka, M. & Schiele, B. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* **9910 LNCS**, 34–50 (2016).

16. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* **2016-Decem**, 770–778 (2016).

17. Rajagopalan, S. & Robb, R. A. Image smoothing with Savtizky-Golai filters. *Med. Imaging 2003 Vis. Image-Guided Proced. Disp.* **5029**, 773 (2003).

18. Toonkum, P., Suwanwela, N. C. & Chinrungrueng, C. Reconstruction of 3D ultrasound images based on Cyclic Regularized Savitzky-Golay filters. *Ultrasonics* **51**, 136–147 (2011).