Markerless 3D tracking and reconstruction of subjects with multiple stereo cameras

Can Hicabi Tartanoglu

Thesis submitted for the degree of
Master in Molecular Biosciences
(Physiology)
60 credits

Department of Biosciences
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2019

Markerless 3D tracking and reconstruction of subjects with multiple stereo cameras

Can Hicabi Tartanoglu

Markerless 3D tracking and reconstruction of subjects with multiple stereo cameras

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abbreviations

The abbreviations will also be mentioned on the right margin whenever they are used.
ANN: Artificial Neural Network
DNN: Deep Neural Network
LED: Light Emitting Diode
D (in 2D and 3D): Dimension
FPS: Frames per second
ttl: transistor-transistor logic
png: Portable network graphics
hdf: Hierarchical data format
LTS: Long Term Support
VRAM: Video Random Access Memory
CPU: Central Processing Unit
GPU: Graphical Processing Unit

Contents

Acknowledgments

The project that is being presented in this thesis was performed at the physiology program at the Department of Biosciences, University of Oslo, under the supervision of professor Dr. Marianne Fyhn and associate professor Dr. Torkel Hafting.

I would like to thank Marianne for giving me the opportunity to experience research in neuroscience while still being an undergraduate student. I would also like to thank Torkel for helping me grow as a researcher as well as giving me the freedom to shape my project according to what I feel mostly passionate about.

To my co-supervisors, Alessio Buccino and Malin Benum Røe, thank you for endless hours of debugging, conducting experiments and writing all the way to the end; I am eternally thankful for your help. My co-workers at the Center for Integrative Neuroplasticity, thank you for the support and enthusiasm, especially Tristan Manfred Stöber. I would also like to thank my study advisor Torill Rørtveit, and my previous study advisors Kyrre Grøtan and My Hanh Tu for their efforts in structuring my studies and providing me with great job opportunities during my studies.

To my peer and soulmate, Dejana Mitrovic, thank you for your eternal academic and moral support in my dire straits, and thank you for always bein a believer. To my mother and father, Filiz Ersan-Boberg and Osman H. Tartanoglu, and the rest of my family, thank you for your love, support and care, you have been there since the first day of my journey. Lastly, my friends Ahmad Tsjokajev, Eirik Sunde Brekke, Imen Belhaj, Jeanne Corrales, Markéta Chlubnová, Roger Karlsen and Øystein Magnus Sørebø. You are all irreplaceable, and I am grateful for having you accompany me in this academic journey of mine.

Abstract

Physical markers are useful for tracking and reconstructing the behavior of experimental animals; however, they are often a major source of interference. Although recent efforts to miniaturize these objects have been promising, we argue that the method needs to change fundamentally. With the introduction of convolutional neural networks, artificial neural network technology have finally fulfilled its 60 year old promise to solve tasks that require complex pattern recognition in domains such as language, speech, and object recognition.

Object recognition technology is an excellent candidate for replacing markers, and has, therefore, laid the foundation for application such as DeepLabCut (DLC), which enables states of the art markerless pose estimation. DLC also supports 3D localization by stereo vision and 3D reconstruction. Stereo vision estimates the depth of a point in a unique subspace of $\mathbb{R}^3$; therefore, multiple stereo cameras can not be used in tandem. This is unsatisfactory in cases when reconstructions are partial across stereo cameras, but complete if they were merged. This is especially advantageous when the temporal resolution of the recordings are low with respect to the experiment.

The aim of the project is to accomplish this feature by mapping the coordinates from each stereo camera to a common subspace. The solution algorithms are organized and stored in a Python package that we named DeepCage. In the results we show how DeepCage can be used for multi-camera tracking of objects in 3D and we discuss its use against other approaches.

1. Introduction

Experiments are a means to support or refute a hypothesis through the exploration of correlations and causal relations between behavior and biological readouts. Behaviour is one of the, if not the most important, output of neural activity in the CNS. Understanding the relationship between behaviour and neural computations, however, is far from trivial, as behaviour is the culmination of context and subject, and the results of a study are most likely invalid if these categorical factors are left unchecked. It is, therefore, important to distill behaviour down to definitive metrics.

CNS: **C**entral **N**ervous **S**ystem

The study of behaviour, ethology, have been around for a very long time, and the study of animal. The main motivation of these studies were to understand the animal's *umwelt* (Tinbergen 1963; Bernstein 1966; Von Uexküll and Kriszat 1970). Tinbergen, Lorenz, and von Frisch were the first ethologists who were awarded the Nobel Prize in Physiology or Medicine for their pioneering work on the patterns of individual and social group behavior in 1973 (*Nobel Lecture. Ethology and Stress Diseases.* 1973).

umwelt: the world as it is experienced by a particular organism

The explorations of these relationships require insight that often requires instruments that are the means to accurately measure behavior-related variables, such as kinematic and dynamic variables; moreover, the use of external instruments can also cause interference with behaviour.

Manual methods are powerful in any scientific study; however, they require investment in terms labour and time, and thereby, funding. The progression of human technology has, therefore, been a huge boost in recent developments in the field. Integrating technology into ethological studies is a non-trivial problem (Schaefer and Claridge-Chang 2012; Anderson and Perona 2014); moreover, the potential of these efforts is a great source of motivation. The use of technological instruments may have disadvantages such as decreased resolution in imperative measurements or increased signal to noise ratio. The primary goal of newer methods is to eliminate sources of interference and maximize the resolution of instruments.

There being many methods to record behaviour (Dell et al. 2014; Egnor and Branson 2016; Camomilla et al. 2018), videography is a non-invasive method for recording a subject in motion. Depending on the complexity of the application, we can estimate the pose of the animal and transform the data into information about kinetics, dynamics, and actions (Bernstein 1966; Schaefer and Claridge-Chang 2012; Anderson and Perona 2014; Dell et al. 2014; Gomez-Marin et al. 2014).

1.1 Pose Estimation and tracking of animal's position

Animals, and their postures have been documented back to cave paintings. Computer storage and power has paved the way for methods that enable the automation of the capture and analysis of this phenomena. Posture, until recently, have been primarily measured by physical markers. The primary goal of these instruments is to reduce the signal to noise ratio surrounding the region of interest.

In behavioural experiments, a significant interference to natural behaviour could be the weight

the animal is carrying[1]. Moreover, during acceleration, the extra object on the body can also cause an impulse in the opposite direction of the acceleration on its mounting point (Newtons third law of motion), which can cause unwanted sensations. These mechanical interference can also cause disadvantageous psychological responses, such as stress, which can interfere with the biological measures. Psychophysical studies are the ones most affected by the sum of the interferences as they aim to understand the relation between sensations, stimuli, and perception.

Recent experimental efforts utilise physical markers to track the position of subjects in experiments. The purpose of these markers is to create an area that is distinguishable from the color-space of the image; thereto, computer algorithms are used to track these regions of interest, optionally, in real time. It is common to use LEDs to track subjects in dark conditions (Bonnevie et al. 2013), and reflective markers in luminous conditions (Finkelstein et al. 2015).

LED: **L**ight **E**mitting **D**iode

Markers (including LEDs and reflective markers) are also used for a procedure referred to as reconstruction. The procedure consists of geometrically representing a physical object in a computer, and using the representation to extract geometric features. A reconstruction can be created in any number of dimensions; moreover, the most common dimension is the number of spatial dimension, which is three to make the computer representation realistic. *3D reconstruction* consists of four logical steps: (1) acquire an image or a video frame; (2) find the 2D coordinates of the points of interest; (3) map the coordinates $\mathbb{R}^2 \rightarrow \mathbb{R}^3$ using a fitting method (since images are two dimensional); (4) build a 3D model out of 3D points (e.g. ears and nose can make up a head model). 2D reconstruction is useful when the third dimension is negligible (Bonnevie et al. 2013), for example, when tracking rodents in planar arenas. There are many categories of applications utilizing 3D reconstruction outside research, including virtual reality, visual effects, 3D modeling.

There have been different solutions to track the animal positions in 3D using external instrumentation:

- Postural features are extracted from reflective markers that are mounted on a head-stage (Mimica et al. 2018) (Figure 1.1 A)
- LED markers on flying bats may be used for 3D localization
- 3D dot patterns are used to define the tagential plane that touches the cranium (Vanzella et al. 2019) (Figure 1.1 C)

These studies enabled the researchers to extract 3D trajectories of the animal behaviour, which they in turn used to demonstrate essential correlations between behaviour and neural activity. Although these solutions have been effective, they present some limitations.

Light from LEDs can cause visual sensations either directly, or indirectly through reflection. The supporting elements[2] of these markers has be to assembled and maintained, which also requires additional time. The use of markers is not congruent with the scientific method as they can be a significant source of interference because their effect on the outcome of experiments.

Replacing markers as a means for pose estimation is, indeed, attractive; however, unlike distinguishing the region of interest by color, we were requiring the computer to see the world through human abstractions of objects (nose, right eye, etc). Computational pattern recognition, required unfeasible human interventions and computational power; however, recent advances in artificial intelligence and machine learning, have represented a revolution in the field of computer vision.

[1] Rats and mice can carry up to 25g and 5g on the cranium without being overly burdened by the extra weight (Fee and Leonardo 2001)

[2] Head-stages, electrical power, etc

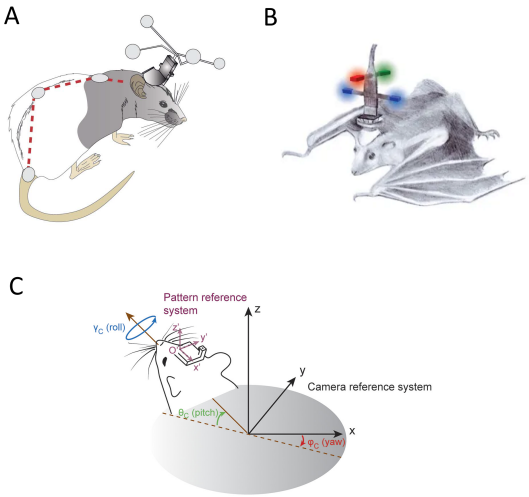


Figure 1.1: **(A)** Schematic for the head and three markers along the back, subfigure adapted from Mimica et al. 2018; **(B)** LED markers used for positioning flying bats in three dimensions, subfigure adapted from Finkelstein et al. 2015; **(C)** The dots are used to define the planar axes (r_1/x, r_2/y). The third axis, r_3/z, is computed by taking the cross product of r_2 and r_1 (in that order), subfigure adapted from Vanzella et al. 2019

1.2 Machine Learning and Artificial Intelligence

Machine Learning is one of the most active subfields of Artificial Intelligence (AI). The field of AI research had its name conceived at Dartmouth College in 1956 (Crevier 1993), where the term "Artificial Intelligence" was given by John McCarthy; however, the field had been growing since the early 1940s after being mentioned as a subject by Warren McCulloch and Walter Pitts in 1943 (McCulloch and Pitts 1943). The field was specifically boosted by the neurophysiological findings of Hebb modeling the mechanism of neural plasticity, and was referred to as Hebbian Learning (Hebb 1962). Hebb suggested that neurons that fire simultaneously often wire together, this way of thinking was the central connectionist model, artificial neural networks (ANN) are modeled after. ANNs such as multi-layer perceptron consist of perceptrons, which are the information processing units of the network. These are optimised for firing when the signal they receive has a value above the threshold of the perceptron. This threshold and the strength of the signal is tweaked by the training algorithm until a desired state is reached.

In addition to the connectionist model, there is also another model referred to as symbolic artificial intelligence. There are only two models. The ideal of the model is that intelligence networks can be defined with the use of logic; thereby, the system is defined by rules, and search trees. The model does have some utility, but is dwarfed by the success story of the connectionist paradigm. This does make sense as developing symbolic AI requires a deep philosophical understanding of intelligence that humanity does not currently posses.

There are many types of ANNs. Each type is optimal for a certain category of pattern recognition. An ANN can have multiple *layers* of neurons for different levels of abstractions. ANNs consist of neurons, which are sensitive. An ANN with many layers is usually referred to as a deep

neural network or DNN(s); however, the term is often used loosely as a substitute for ANN(s).

The AI movement had a difficult time living up to its hype in the 1960s during the Cold War that lead to loss in funding and interest. Firstly a major fiasco had occurred, in which AI had failed to translate Russian documents and scientific reports to English because of issues with grammar with a funding of approximately 20 million USDs (Hutchins 2005); secondly, in 1969, when Marvin Minsky and Seymour Papert published the book Perceptrons (Minsky and Papert 2017), they scrutinized the most robust connectionist model as being inefficient. This period of pessimism towards AI lasted until the late 1980s, which ended by virtue of three key developments in the industry:

- **Hardware:** Development in graphical processing units or GPUs thanks to consumer demand, paved the way for research and development that AI research could not fund by itself. AI technology basically got a free ride[3].
- **Big data:** Data that had been collected for purposes other than AI could be re-purposed for training AI algorithms.
- **Algorithms:** Early applications of artificial neural networks such as the multi-layer perceptron (MLP) was limited and inefficient by itself. Newer types of neural network such as convolutions neural networks (CNN) sought to solve these limitations.

In the mid 2010s, Google openly recognised the potential for the world-wide adoption of AI, and open sourced a more streamlined version of their closed library named DistBelief (Oremus 2015). The name of the new library was Tensorflow (Abadi et al. 2016). In the meantime, Facebook merged several machine learning libraries including Caffe2 into PyTorch (Paszke et al. 2017). The open sourcing of these libraries boosted the adoption of ANNs and the rate of debugging.

The culmination of these events is what has brought us here, to the brink of eliminating markers from experimental studies. The introduction of enterprise-ready libraries such as Tensorflow made it possible to create reliable tools for computer vision.

1.2.1 2D and 3D pose estimation

In 2014, DeepPose was the first project to apply machine learning to human 2D pose estimation (Toshev and Szegedy 2014), there were many immediate efforts to improve the effort with the translation invariant model (Jain et al. 2014), and CNNs along with geometric constraints (J. J. Tompson et al. 2014; J. Tompson et al. 2015). As any scientific boom, the years that followed there have been approximately 4000 papers on human pose estimation, and standardized benchmarks for comparisons of state of the art performance (Andriluka et al. 2018). Within a decade, the mistake rate on human labelings by ANNs have gone from 56% to 4% with the most popular networks being just few percentages of each other (Newell, Yang and Deng 2016; Insafutdinov, Pishchulin et al. 2016; Insafutdinov, Andriluka et al. 2017; Cao et al. 2017; Kreiss, Bertoni and Alahi 2019; Sun et al. 2019).

The transition from 2D to 3D, is a story that is still being written. There have been massive improvements lately (Sarafianos et al. 2016) in which stereo vision is the primary means of acquiring 3D data (Mehta et al. 2016; Martinez et al. 2017). In the meantime, there have new exotic ideas that seek to extract the information necessary from single images with the use of ANNs (**\cite {tome2017lifting}; \cite {chen20173d}; \cite {martinez2017simple}**).

[3]Note that GPU technology was important for the early development of AI as large scale implementations of the technology most often utilize application specific integrated circuits or ASIC(s) as they are cheaper

1.3 DeepLabCut: Generalized paradigm for pose esitmation beyond humans

DeeperCut was one of the first human pose estimation algorithm utilizing a convolutional neural network written in Tensorflow (Insafutdinov, Pishchulin et al. n.d.). A subset of DeeperCut algorithms were augmented for experimental use, and packaged together with application specific algorythms in new a Python package named DeepLabCut (Mathis et al. 2018). The aim of DeepLabCut is to streamline the acquisition and analysis regions of interest across time. DeepLabCut also integrates functions from the Python package OpenCV for 3D reconstruction. OpenCV (Bradski and Kaehler 2000; Bradski and Kaehler 2008) is an open source computer vision library that is written in C++, and has bindings in Python, Java, and Matlab/Octave.

Many studies have attempted to implement DeepLabCut, such as the 3D reconstruction of rodents' paw (Bova et al. 2019). Other methods have been utilized for 3D pose estimation in humans (Tome, Russell and Agapito 2017; Li et al. 2019) in addition to rodents both offline (Machado et al. 2015) and in real time (Vanzella et al. 2019).

1.4 3D reconstruction of freely moving rodent within a constricted space

The experimental laboratory at the Centre for integrative neural plasticity (CINPLA) conducted visual experiments on freely moving rodents. These experiments included visual stimulation, electrophysiological recordings, and 2D tracking using LEDs mounted on the head of the animal. The experiment was performed inside a closed system constricted by the glass walls of a repurposed, cuboid aquarium. In addition to repeating these experiments, the goal of this project was to augment the behavioral readout of the animal's activity, by tracking its pose in 3D[4].

We wanted to model the relation between activities in the visual cortex with movement features of the head during visual stimulation. This correlation could lay the groundwork for understanding the relation between electrophysiological activity in the visual cortex (by extension the visual pathway) and the vestibular system. Our solution was to use a surround camera setup consisting of eight Logitech C930e webcameras, which both tracked and 3D reconstructed the rodent. The 3D reconstruction could be used for calculating features related to head movement.

The use of a combination of pairs of cameras (stereo camera) makes it complicated to integrate the acquired data. Each stereo camera, in fact, reconstructs 3D positions in its own subspace of $\mathbb{R}^3_i$; therefore, multiple stereo cameras can not be utilized in tandem for completing 3D reconstructions before mapping them to the same space, $\mathbb{R}^3_i \rightarrow \mathbb{R}^3_{global}$. We therefore needed to identify a global subspace in order to integrate the multiple cameras adopted in the data acquisition phase.

1.4.1 Aim of study

The project required an automated method for collecting data from eight stereo cameras, and a method for mapping the coordinates to the same subspace of $\mathbb{R}^3$. DeepLabCut also lacked a feature to combine data from stereo cameras that were acquired within a reasonable time period. This feature is used to combine several incomplete *reconstructions* into one complete reconstruction. This is especially useful in cases when a stereo camera observes the front of the rodent while other stereo cameras observes the sides and the back, which is quite often.

[4]The method being presented can also be scaled for recording eye movement, but this is not in the scope of the project

Nonetheless, DeepLabCut 3D does not have such analytical algorithms; therefore, the goal of this project is to create analytical and methodological tools to tackle the above-mentioned issues.

2. Materials and methods

A person who never made a
mistake never tried anything new

Albert Einstein

2.1 Approvals

The experiments comprising the project were approved by the Norwegian Animal Research committee (FDU) and the European Union. The staff that worked on this study had completed the legally required training, and had been certified to conduct animal trials by the Norwegian Food Safety Authority, *Mattilsynet*.

2.2 Design of the experimental setup

2.2.1 Workstation

Our experiment station consisted of two workstations (or computers) to ensure a balanced CPU load. The workstations were identical, and consisted of the following components:

- CPU: Intel® Core™ i7-8700 with 6 cores and 12 threads that has a base clock of 3.20 GHz
- RAM: 2x16GB, 32 GB 2667MHz
- Storage: SSD with 512 GB capacity

RAM: **R**andom **A**ccess **M**emory

SSD: **S**olid **S**tate **D**rive

FoV: **F**ield **o**f **V**iew

fps: **f**rames **p**er **s**econd

Each workstation had four Logitech c930e webcameras with 90°FoV connected to them, making it eight cameras in total. The C930e can capture 30 FPS with a resolution of 1920x1080 pixels. The cameras had an accompanying software named Camera Settings to enable configuration of the internal settings. Our experimental setup required us to turn auto focus and color correction off (discussed in **??**).

There were two cameras on each side of the glass cage, as depicted in figure 2.1. These were mounted to a custom aluminium frame using custom-made 3D-printed arms. These arms were able to move perpendicularly with respect to the cage side to which they were attached. Each camera was in a group with its two neighbors, making up eight groups in total.

We used compass directions for assigning camera identities. With reference to figure 2.1, #1 and #2 are located on north, camera #3 and #4 are located on east, etc. We used compass direction to distinguish between the location of cameras that are on the same side, which made

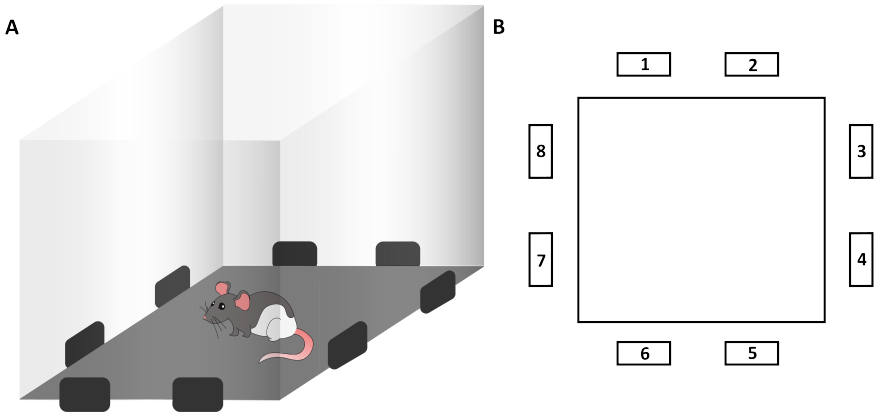


Figure 2.1: The cage assembly: (A) A depiction of the glass cage with cameras and rodent **(B)** Diagram of the camera placement relative to the glass cage along with the number identification of each camera. Each camera was paired with its two neighbors. Camera #1 for instance, was in paired with camera #2 and #8: (1, 2), (8, 1).

camera #1 NorthWest and #2 NorthEast, etc. The following is a list of the camera IDs that were used:

```
CAMERA_NAMES = (
    'NorthWest', 'NorthEast', 'EastNorth', 'EastSouth',
    'SouthEast', 'SouthWest', 'WerstSouth', 'WestNorth'
)
```

The glass cage and the cameras were surrounded by four 17 inch monitors with a resolution of 1920x1080 and refresh rate of 60 Hz (Dell workstations, Limerick, Ireland). The monitors were used to deliver the visual stimulation to the experimental animal. The monitors surrounding the cage received the same display input; thereby, they were recognized as a single monitor by the workstation.

2.2.2 Electrophysiology: Open Ephys

Electrophysiological activity was recorded with the use of 17 micron platinum-iridium electrodes with gold-plating. Four such electrodes are intertwined to form a *tetrode*. Each electrode was individually mounted on an Axona microdrive, and implanted into the visual cortex area V1 or V2 in rats. Each microdrive is connected to an Intan headstage through a custom-made adapter. Each rat had two headstages with half of the channels (16) utilized; both were connected to the acquisition board with the use of SPI cables.

Extracellular electric potentials recorded by the electrodes are generated by ionic currents flowing in to and out of neuronal membranes. These signals propagate to the Intan headstage, where they are amplified, converted to digital from analog, and relayed to an Open Ephys acquisition board (Siegle et al. 2017) through an Intan SPI cable. The acquisition board uses an FPGA to send the digital signals to the workstation through USB 3.0.

FPGA: **F**ield **P**rogrammable **G**ate **A**rray

The signals were recorded in the Open Ephys GUI (Siegle et al. 2017). The software was an open source visual programming framework developed by the Open Ephys foundation. The Open Ephys acquisition board was also connected to an I/O board, which was utilized for receiving TTL signals for synchronization purposes.

TTL: **T**ransistor **T**ransistor **L**ogic

2.2.3 Visual stimulation: PsychoPy

PsychoPy is an open-source software designed for behavioral and psychophysical experiments (Peirce et al. 2019). We used PsychoPy[1] to present pseudo-random grating stimuli with different orientations. The presentation of a visual stimulus, triggered a synchronization TTL event sent through a parallel port mounted on the workstation and recorded by Open Ephys system.

2.2.4 Video recording: Bonsai

We used a modular visual programming framework named Bonsai (Lopes et al. 2015), which allowed the user to create asynchronous processing and/or reactive sequences initiated by events. For instance, during video capture, an event would be the reception of a frame from a webcamera.

2.3 Markerless pose estimation with DeepLabCut

The following methods were adapted from the GitHub repository of the DeepLabCut project, and the scientific article that presented the package 3D markerless pose estimation (**dlc-3d**).

In DeepLabCut, files related to a DeepLabCut session were stored and organized in directories referred to as *DeepLabCut projects*. We used two types of projects: DeepLabCut markerless pose estimation project (2D) and DeepLabCut 3D project.

Each project had a configuration file named `config.yaml`, which stored the project settings. The `config.yaml` file was modified with the use of text-editing software to adjust the settings for the specific experiment. DeepLabCut functions that interact with projects used the path of the `config.yaml` file to read the settings through an auxiliary function.

2.3.1 Installing DeepLabCut to a `conda` environment

We used `conda` as the Python package manager of the project. We installed the 64-bit version of Miniconda with Python 3.7, which included the `conda` package manager. We used MSVC – VS 2019 C++ x64/x86 build tools for compiling C++ packages that had Python bindings.

With the 3rd party requirements installed, we proceeded to install DeepLabCut into a dedicated virtual environment using `conda`. This environment was used to run the Python code in the project.

We used `conda` to setup our environment using the environment `yaml` file shipped with the DeepLabCut repository. Installing this way ensures that the packages work in tandem. MSVC was used to compile some of the dependencies. Additional packages were installed after the environment creation with the use of `pip` (used to install Python packages available on the PyPi – Python Package Index – repository).

Following the creation of the `conda` environment, we created a root projects directory to store all DeepLabCut projects used for this thesis. Each specific project consists of a separate folder that stores configuration files, videos, and trained models.

2.3.2 DeepLabCut Jupyter Notebook example

The following steps are also explained in a (Jupyter) notebook file along with output from the presented code. In the following, we will briefly introduce the different steps involved in creating and running a DeepLabCut project for markerless pose extimation.

[1] version 2

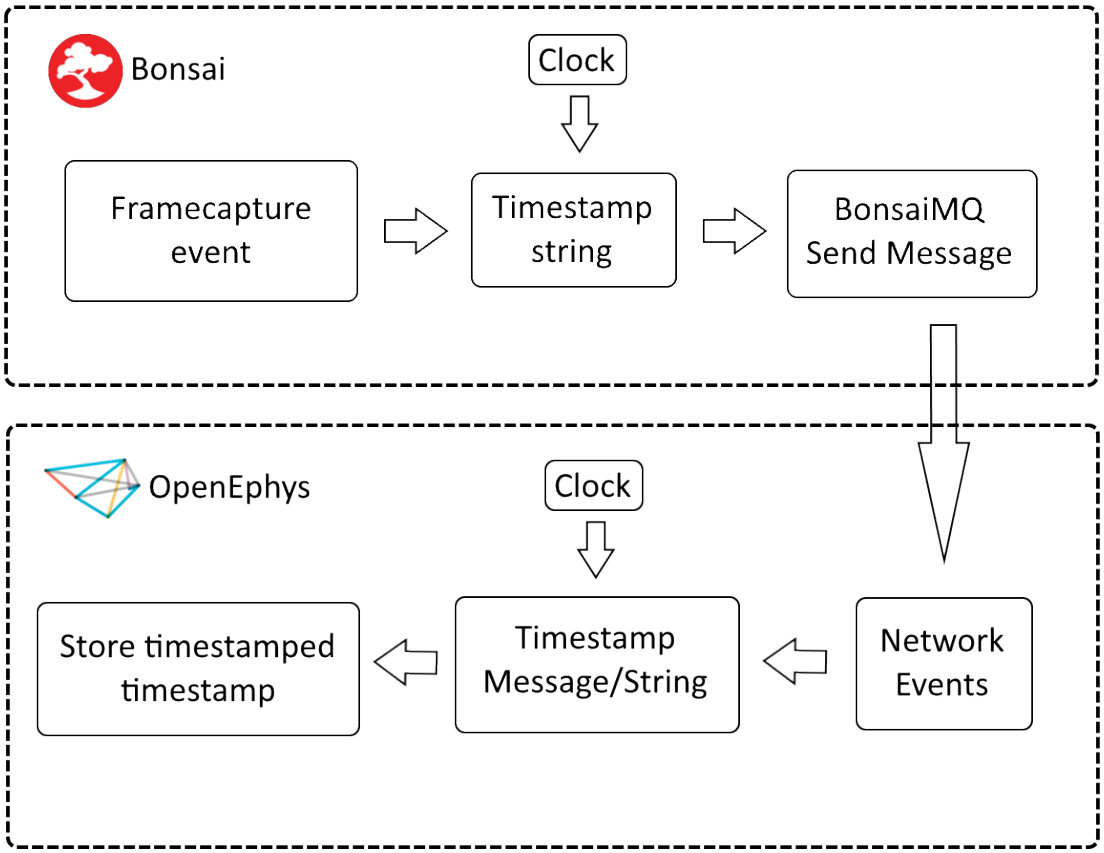


Figure 2.2: BonsaiMQ schematic: Bonsai records a timestamp whenever a frame is captured. The timestamp is sent to Open Ephys with the use of BonsaiMQ. When Open Ephys receives the message, it timestamps the message retrieval with its own clock, and stores the message along with its timestamp. The difference between the two timestamps is the time it took for Bonsai to send the message, and for Open Ephys to receive and record a timestamp for the message, usually < 1ms.

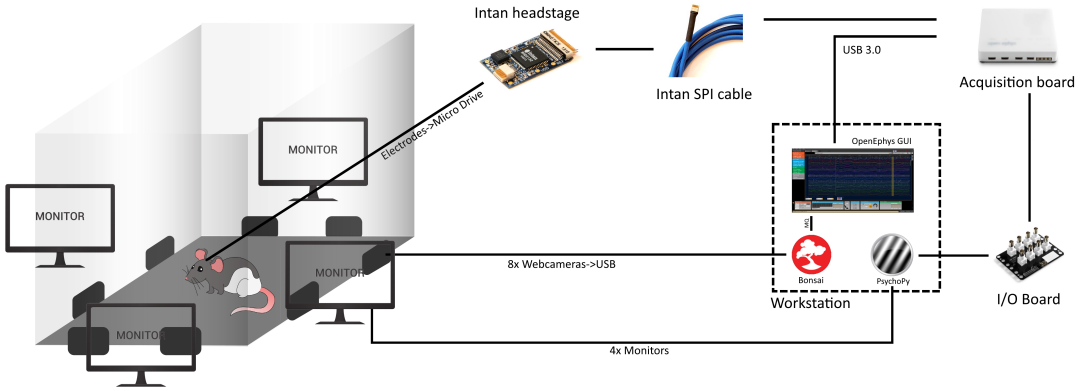


Figure 2.3: Diagram of the experiment setup. The rat had two microdrives + headstages with half of the channels, 16, utilized mounted to its cranium; both were connected to the Open Ephys acquisition board with the use of SPI cables; the acquisition board was connected to the workstation with USB 3.0. The workstation was also connected to an analog I/O board, which was utilized for sending TTL signals to the Open Ephys acquisition board. The monitors were connected to an adapter that would split a single monitor signal to four through DVI cables.

Creating and configuring a DeepLabCut markerless pose estimation project

DeepLabCut[2] organises its project files into a standardised directory structure (Nath et al. 2019). By using the function `deeplabcut.create_new_project()` a project directory was created. This function also creates several sub-directories "dlc-models"; "labeled-data"; "training-datasets"; "videos", and a configuration file named "config.yaml".

Annotating the regions of interest

The `deeplabcut.extract_frames('path-to-config.yaml')` function was ran to extract uniformly distributed frames from the videos. The frames would be stored under the **labeled-data** directory in a sub-directory dedicated to the respective video. The frames were used for manually labeling body parts with a graphical user interface (GUI) initialised by running `deeplabcut.label_frames('path-to-config.yaml')`. A screenshot of the GUI is shown in figure 2.4. The labeled data was saved in a hdf5-file (Folk, Cheng and Yates 1999) and were used to train the DNN.

DNN training

A deep neural network (DNN) was trained using the data manually annotated as training dataset, which is created with `deeplabcut.create_training_dataset('path-to-config.yaml')`. The function downloads a pre-trained network that has been trained with the ImageNet dataset (Mathis et al. 2018), which is a massive dataset used for object recognition (Deng et al. 2009). Pre-training the network allows for transfer learning, which makes it possible to tune the model to user-specific cases with very little labeled data.

[2]version: 2.0

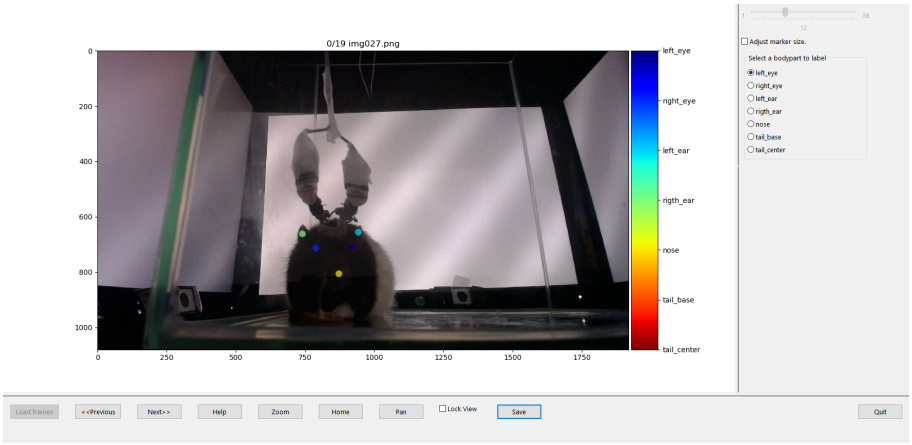


Figure 2.4: A screenshot of the graphical user interface for annotating regions of interest in DeepLabCut.

The training was initiated following the preparations, and was kept running until the mistake rate was below 0.2%, which required approximately 4 hours on our GTX 1080 TI. We used the same DNN for all markerless pose estimation in all cameras.

Evaluation of the trained DNN

DeepLabCut includes a function for evaluating the performance of the trained network. The evaluation includes the computation of the *mean average Euclidean error* (MAE) between the user annotations and DNN predictions. It also created a visual representation of the two labels on the label images `evaluation-results`. Each prediction comes with a likelihood score that is based on the probability of the estimator, in this case the DNN, of being correct. This probability is based on how well the current frame *fits* the model that the DNN uses. The predicted labels that had a likelihood score $>$ p-cutoff were colored with circles, and the predicted labels that had likelihood score $<$ p-cutoff were colored with "X" (Figure 3.5). A refinement step (explained below) can be performed if the predictions are not satisfactory.

Video analysis

After the evaluation, videos can be analyzed to extract labels from all frames with `deeplabcut.analyze_videos()`. The analyzed data is stored as an hdf5-file (Folk, Cheng and Yates 1999) in the same directory as the video files. This data included the network name, the body parts, the (x, y) positions in pixels, and the likelihood score for each annotation on regions of interest per frame. The plots that were created using `deeplabcut.plot_trajectories`, deeplabcut function, were showing trajectories for each region of interest (Figure 2.6).

Refinement

With DeepLabCut 2.0, a refinement process was introduced (Nath et al. 2019). This automatically extracts frames from a user-defined video that the current DNN struggled annotating (low likelihood score). The user can re-annotate these *outlier* frames and continue the DNN training. The refinement step improves the overall performance of the DNN predictions.

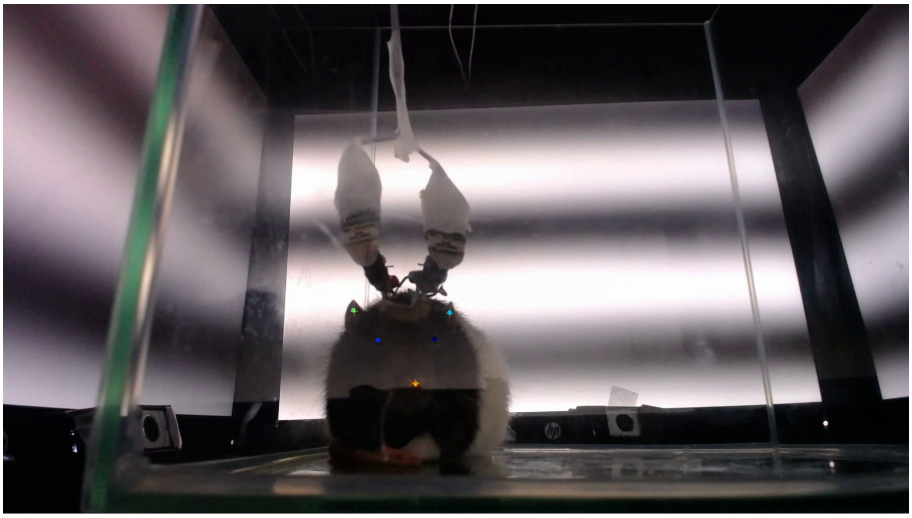


Figure 2.5: Depiction of an evaluated annotation set. The plus ('+') sign indicates the user annotation; the circle ('o') indicates an DNN prediction with a likelihood score above the p-cutoff, and the cross ('X') an DNN prediction with a likelihood score below the p-cutoff

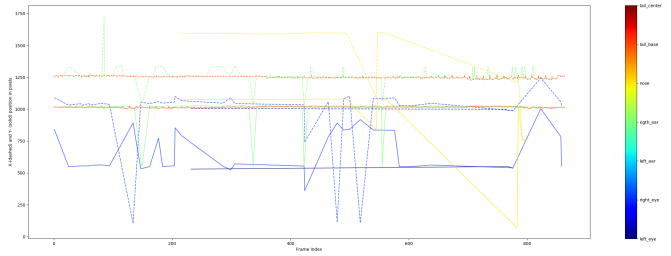


Figure 2.6: Depiction of a 2D trajectory of regions of interest.

The outlier frames were extracted by calling the `deeplabcut.extract_outlier_frames('path-to-config.yaml', [Full path of videos])` function. The next step was running the refinement GUI, which is quite similar to the labeling GUI displayed in figure 2.4. The main difference was the placement of the predicted label. This predicted label could be removed or altered during refinement. The refinement GUI was initiated by calling `deeplabcut.refine_labels('path-to-config.yaml')`. In the GUI, we chose a set of images by clicking on "Load Labels", and selecting the directory in which the set of frames were stored. During the refinement process, labels with a likelihood below the likelihood threshold were presented with an empty circle instead of a full circle.

Labels that were previously refined, but needed further adjustment, were further refined in the GUI by ticking the "Adjust original label?" checkbox. Corrected labels were merged into the previous data set by calling `deeplabcut.merge_datasets('path-to-config.yaml')`.

After the refinement is complete the user repeats the training steps and can evaluate the performance of the network after the refinement procedure. Moreover, the refinement step could be repeated several times to improve the DNN performance.

2.4 DeepLabCut 3D project

2.4.1 Creating and configuring 3D projects

The workflow for mapping the 3D coordinates of a stereo camera was implemented in a DeepLabCut 3D project. Concurrently, we created one DeepLabCut 3D project for each camera pair, i.e. 8 3D projects in total. The 3D project is created with `deeplabcut.create_new_project_3d()`. Following the creation of the project, we had to define the name of camera pair, which is, for instance, NorthWest and NorthEast for the first pair.

2.4.2 Calibration of the cameras

Following the configuration of the project, the next step was to calibrate the cameras.

The calibration was performed with the use of one of standardised patterns from OpenCV. We used a checkerboard pattern as DeepLabCut had native support for these kind of calibration patterns. Using of a simple Bonsai workflow, which included a camera capture stream and an image saver, we acquired several synchronous images for each camera pair containing the checkerboard pattern in different orientations; the images were saved as .PNG files[3].

The calibration process itself was performed with the use of the `deeplabcut.calibrate_cameras()` function. Initially the function was run in inspect mode, which allowed us to inspect the detection of the checkerboard corners before committing to the computation of a stereo camera model. These images were saved to the `corners` directory in the 3D project. These images were manually inspected for imperfections, images with imperfect detection were deleted.

The calibration process yielded the radial distortion and tangential distortion coefficients, which were used for undistorting the images. Additionally, the cameras had to be calibrated pairwise to compute the relative rotation and position between the cameras. The focal length, and the principle point of the camera are also estimated during the calibration. These values were used during triangulation.

Focal length: The length between the camera sensor and the lens

[3]the images were in .PNG format, and had to be converted to .JPEG for compatability reasons. We used our own function, `deeplabcut.png_to_jpg()`, to automate the process.

Principle point: The point on the image plane intersected by the line perpendicular that passes through the pinhole (principal axis)

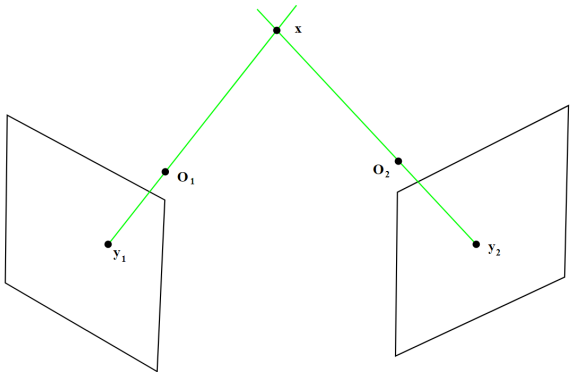


Figure 2.7: Depiction of triangulation in ideal conditions Adepted from https://en.wikipedia.org/wiki/File:TriangulationIdeal.svg

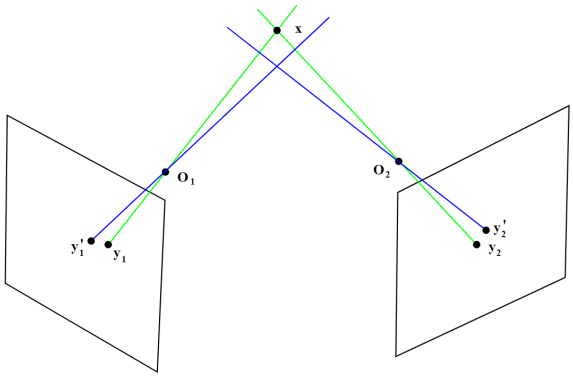


Figure 2.8: Depiction of triangulation in realistic conditions the primary source of interference in these cases is the imperfection of the camera lens and sensor causing distortion. Adapted from https://en.wikipedia.org/wiki/File:TriangulationReal.svg#filelinks

The quality of the distortion parameters was then inspected. DeepLabCut automates this through the `deeplabcut.check_undistortion()` function. It should be noted that this quality assurance function is only for testing the intrinsic and extrinsic calibration of the individual cameras, and does not ensure the quality of the stereo camera calibration.

2.4.3 Triangulation

In ideal conditions (Figure 2.7), the projections od 2D points from the two camera planes meet in a single 3D point; however, the imperfections of the cameras, and, by extension, their representations of the world, introduce an error term into the solution. The source of these errors are the camera lenses, which can introduce radial , and tangential distortion. The degree of these distortions vary across cameras, which made the problem device-specific. The imperfections had to be rectified. This problem is referred to as the *stereo correspondence problem* (Figure 2.8).

Radial distortion: Are caused by the lens being too convex or concave

Tangential distortion: Are caused by the lens and sensor not being parallel

Conveniently, the imperfections of the lenses can be identified using OpenCV. The authors of DeepLabCut augmented these algorithms, and fit them to the DeepLabCut 3D workflow, which was demonstrated through the 3D reconstruction of sprinting cheetahs from 6 GoPro videos (**dlc-3d**).

Before triangulation , the 2D coordinates from the camera frames were undistored using the calibration parameters estimated during calibration. This yields a new coordinate that is better estimate of the real coordinate. The triangulation can be then performed by `deeplabcut.triangulate(config_path3d, video_path)`.

The FoV of each stereo camera can be interpreted as a subspace that is continuous, and closed; therefrom, each stereo camera is in its FOV; thereby, in its projective subspace in $\mathbb{R}^3$. Each stereo camera subspace is injective with respect to the others. Moreover, each coordinate has to be in the same subspace for merging the incomplete reconstructions. We solved this by mapping the coordinates to a commonly injective subspace, with respect to each stereo camera, that we defined as the *volume* of the experimental system . This volume was defined, and mapped to using tools that will be presented in the result section 3.2, DeepCage.

FoV: **F**ield **o**f **V**iew

Volume: The volume of the experimental system refers to the 3D space spanned by the system

The steps mentioned in the DeepLabCut 3D section (2.4), were repeated for each stereo camera. Alternatively, the steps until calibration image selection can also be automated with the use of deepcage.project.create_dlc_dc_projects().

3. Results

The aim of the project was to create the features that the visual experiment pipeline at CINPLA needed. Initially, we augmented the physical experiment environment to make it compatible with our new experiment design. We also created the Python package named DeepCage for expanding DeepLabCut, giving us the ability to merge incomplete reconstructions, testing tools, and quality of life tools such as automating the assembly of project environments to DeepLabCut. We had to add features to other parts of our experimental pipeline as well, which includes giving Bonsai the capability to directly communicate with other parts of our pipeline via ZeroMQ. We made several contributions such as bug fixes and generalized features to the DeepLabCut project.

3.1 Cage assembly

The glass cage setup that we used had to be augmented for DeepLabCut related camera support. This included means for attaching webcameras, webcameras, and a workstation to handle the ingestion of 240 frames per second (30 fps * 8 webcameras).

The glass cage is mounted to an aluminium frame, the frame has eight 3D printed arms (RaffoSan 2017) that are evenly spaced on each side. The base of the webcameras are augmented with a 3D printed mount (Felwat 2018), which are used to attach them to the arms.

Although we have assembled the glass cage and provided support for the eight pairs of cameras, most of the results will be shown on a test dataset in which the cameras where set on a laboratory table outside the glass box. These kinds of test datasets were used to develop the software tools necessary to the project, which can be thought as the main contribution of this thesis.

3.2 DeepCage

We created a Python package named DeepCage[1] to organize the function that we made for the project. This includes the creation of DeepLabCut 3D projects; finding the new origin; creating linear maps, for each stereo camera. Finally, after mapping each coordinate to the same vector space, DeepCage was also designed for extracting features from the movement of the experiment animal.

3.2.1 Creating a DeepCage project

DeepCage projects are used to organise DeepLabCut projects for analysis. The project has a hierarchy for file storage similar to DeepLabCut projects, and stores results; tests; videos; DeepCage calibration images. A DeepCage project can be initialised in many ways.

[1] `https://github.com/caniko2/DeepCage`

The user can create a DC project with `deepcage.project.create_dc_project()`, or migrate the calibration parameters along with DeepLabCut 3D projects to a new project with `deepcage.project.create_project_old_cage()`.

Alternatively, the user can also use the automation function, `deepcage.project.create_dlc_dc_projects()`, bundled with DeepCage, which automates the following steps:

1. Creates a DeepCage project
2. Assigns a DeepLabCut markerless pose estimation project to it (optionally created a new one)
3. Creates DeepLabCut 3D projects equal to the number of stereo cameras, and assigned them to the DeepCage project
4. Detects checkerboard calibration images in a given root folder, converts them from .PNG to .JPEG, and moves the converted version to their respective DeepLabCut 3D project

3.2.2 Calibrating stereo cameras

As introduced in the Methods section, each pair of cameras needs to be calibrated to become a stereo camera. This is done by using a checkerboard pattern, that enables automatic algorithms to easily recognize corners and to fit a model to correct for radial and tangential distortions, as well as to estimate the relative position and rotation between the two cameras. In order to acquire the necessary amount of calibration images, videos were acquired and a Bonsai script was used to ensure synchronization between frames from the same pair. Moreover, after the automatic algorithm detected the corners, badly detected frames were removed from the calibration data, in the attempt to improve the calibration step, which is crucial for 3D reconstruction. In figure 3.1 are some examples of well and bad detected frames, which were includes/excluded for calibration.

3.2.3 Theory for mapping the coordinates to the same subspace

As previously mentioned, the coordinates of each stereo camera can not be used in tandem without being in the same vector space. As depicted in figure **??**, in fact, different camera pairs have different coordinate systems, which need to be remapped to the same subspace for integrating data from multiple stereo cameras.

We solve this problem by projecting the 3D points from different stereo cameras onto the same basis. In mathematical notation:

$$\begin{bmatrix}\vec{B}_{r_1} & \vec{B}_{r_2} & \vec{B}_{r_3}\end{bmatrix}^T \left([p_1, p_2, p_3] - \vec{O}_{new}\right) = \mathbf{B}^T\left(\vec{p} - \vec{O}_{new}\right) = \vec{p}_B$$

Where $\vec{B}$ is a standardized column basis vector defined by the experimenter; p_i is one of the components belonging to a coordinate; $\vec{O}_{new}$ is the vector from the old origin to the new origin also defined by the experimenter; $\mathbf{B}$ is the linear map comprised of column basis vectors; $\vec{p}$ is a vector composed of p_i; $[\vec{p}]_B$ is the new coordinate in the common vector space spanned by $\vec{B}$.

The general solution for changing basis is as follows:

$$\begin{bmatrix}\vec{B}_{r_1} & \vec{B}_{r_2} & ... & \vec{B}_{r_n}\end{bmatrix}^T [p_1, p_2, ..., p_n] = \mathbf{B}^T\vec{p} = \vec{p}_B$$

The dot product maps $\vec{p}$ to another vector space with $[\vec{B}_{r_1}\vec{B}_{r_2}...\vec{B}_{r_n}]$ as basis. The general solution for changing basis in $\mathbb{R}^3$ is as follows:

$$\begin{bmatrix}\vec{B}_{r_1} & \vec{B}_{r_2} & \vec{B}_{r_3}\end{bmatrix}^T [r_1, r_2, r_3] = \mathbf{B}^T\vec{p} = \vec{p}_B$$

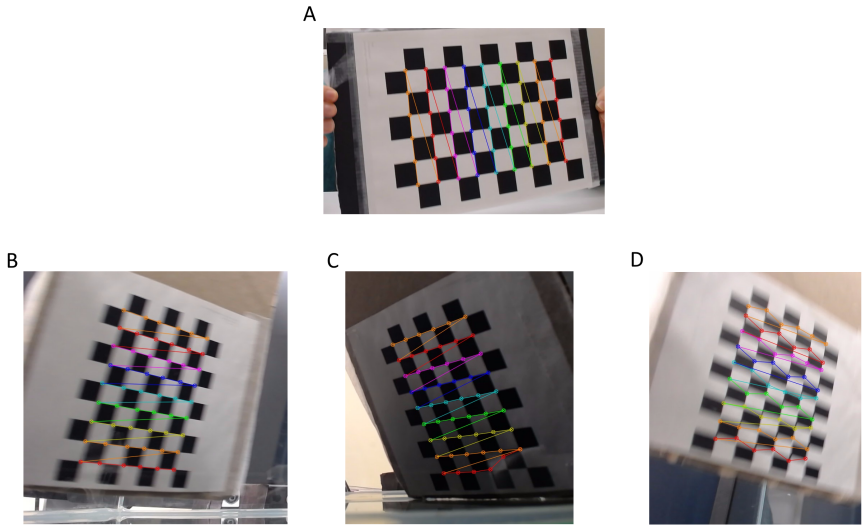


Figure 3.1: Examples of corner detection attempts on the checkerboard. A) successful detection; **B)** potentially inaccurate because of motion blur (fail); **C)** failed detection caused by lack of lighting; **D)** another failed corner detection because of severe motion blur

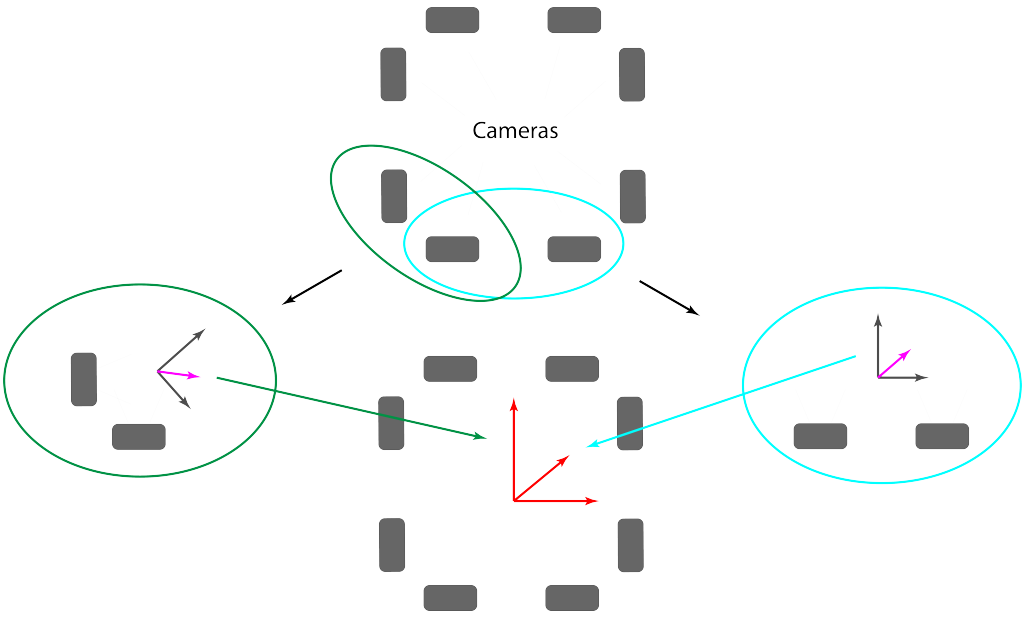


Figure 3.2: Mapping coordinates to the same subspace. Different stereo cameras have different coordinate systems. In order to integrate reconstruction from several stereo cameras, the coordinate systems must be remapped to the same subspace (in red).

r_1, r_2, and r_3 can also be referred to as x, y, and z. From here we must remember that each stereo camera has its own origin, and that these also have to be taken into account when mapping the coordinates to the same space. We, therefore, subtract the $\vec{OO}_{new}$ vector from each coordinate, where O is the stereo camera origin, and O_{new} is the point where the experimenter defined origin is located. Each stereo camera has a unique O_{new}. Concurrently we end up with the mentioned solution:

$$\mathbf{B}^T\left(\vec{p} - \vec{O_{new}}\right) = [\vec{p}]_B$$

In the following subsection, we will present the methods for defining $\mathbf{B}$ and O_{new}.

3.2.4 Defining B and O_{new}

Annotation

Each component of $\mathbf{B}$ and O_{new} had to be defined manually by annotation. The annotation was performed on images that had an object referred to as the *calibration tool* (shown in figure 3.3C), which standardized the position of the new basis vectors across the stereo cameras. The cone shaped structures at the ends of the calibration tool assists the user in the upcoming steps of the calibration.

The calibration tool was placed in the centre of the cage with the planar cones directed perpendicular to the sides of the cage. The planar cones represent the planar axes that are r_1 and r_2. A simple Bonsai workflow was used to capture images for calibration (shown in figure 3.3A).

The cameras were not moved after this step; otherwise, the calibration had to be repeated. The images were fed into the workstation, and placed inside the calibration_images folder in the DeepCage project. The calibration tool was, subsequently, removed from the cage.

The images were annotated manually by mouse clicks using a GUI within DeepCage, initiated by running `deepcage.basis_label(config_path)`. The GUI showed the calibration images acquired previously and allowed the user to manually annotate the points mentioned in the figure title. The annotation includes annotating the three visible cone structures, and the estimated location of the origin, which are 4 annotations per camera; 32 annotations in total (Figure **??**). The annotations were saved in the DeepCage project as a pickle file (`https://docs.python.org/3.6/library/pickle.html`).

3.2.5 Computation of the basis vectors and estimation of the linear map

The annotations from the previous section were triangulated with the use of the camera calibration matrices from DeepLabCut 3D project of the stereo camera. This was automated with the use of an augmented function from the DeepLabCut package, which supported manually assigned points, instead of points from the trained ANN.

The direction of each axis was determined by us with convenience in mind. The x-axis or r_1 was directed towards the east side of the cage, the y-axis or r_2 was directed towards the north side of the cage, and, lastly, the z-axis or r_3 was normal to r_1 and r_2, which meant it was normal to the floor of the cage (pointing to the ceiling). Recalling that each tip on the calibration tool signifies a point on one of the three axes, we can determine the basis vectors with the use of linear algebra.

3.2.6 Defining the new axes for each camera pair

Following are the calculations for defining the horizontal axes (Figure 3.4):

$$r_{h+} - O_{new} = \vec{r_{h+}}$$

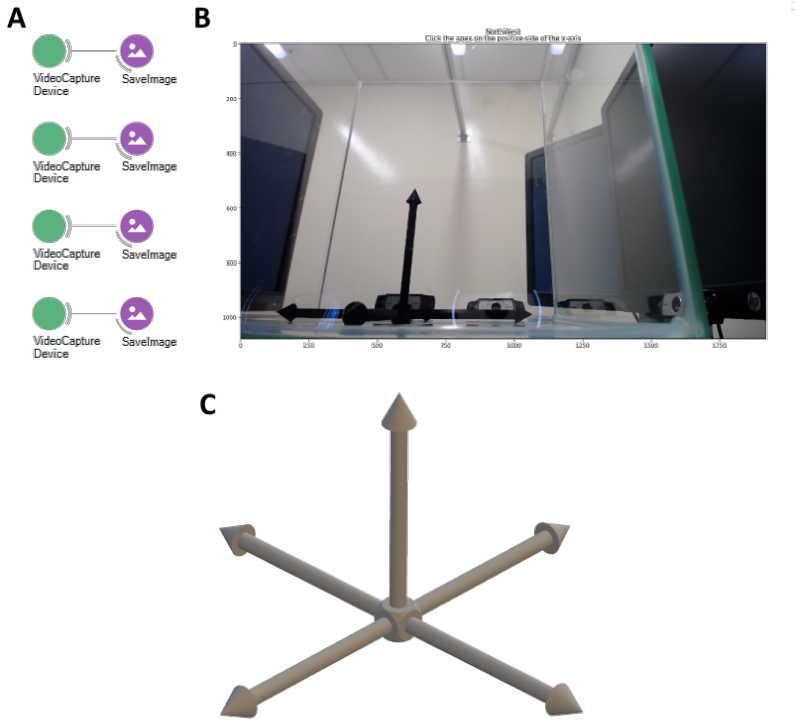


Figure 3.3: Calibrating the cage (A) The Bonsai workflow included a capture and save nodes for each camera, making it eight in total (only four of the eight are depicted in the figure). **(B)** Depiction of the interface used for annotating images for cage calibration. The point that we recently annotated in the current case is marked in red; the positive apex of the x-axis. The image was one of the eight cage calibration pictures taken with Bonsai. **(C)** The 3D model of the calibration tool.

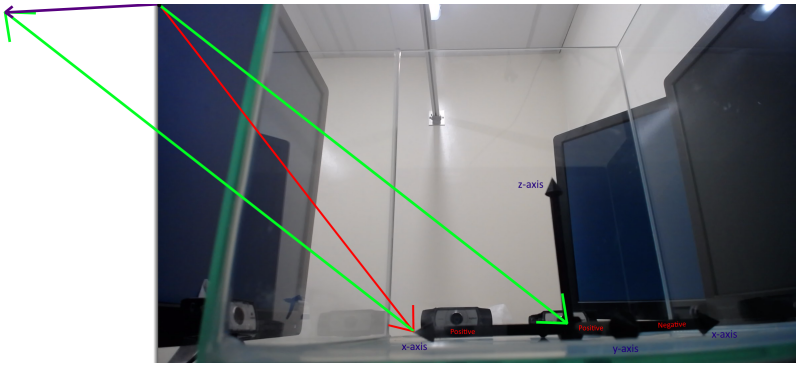


Figure 3.4: Camera NorthEast calculation of the horizontal basis vector, the x-axis: The red arrow is the vector from the origin to the tip of the positive side of the horizontal axis ($\vec{x}$); the green arrow is the vector from the origin to the new origin ($\vec{O_{new}}$); the violet arrow is $\vec{x} - \vec{O_{new}}$ which is the new horizontal basis vector or, in the case of north, x-axis

Where O_{new} is the point of the new origin (i.e. the center of the calibration tool); r_{h+} is a point on the positive side of the horizontal axis. $\mp$
$\pm$ signifies that the positive side may shift sides with respect to O_{new} across sides; thereby, shifting the direction of the positive axis. This shift occurs as we go from north to east, and south to west; $left \rightarrow right \rightarrow left$. The vertical axis, which is the z-axis in the experiment system is always $r_v - O_{new} = \vec{r}$.

The orthogonal axis with respect to the camera, which can also be interpreted as the depth also changes direction across sides; thereby, also changing the orthogonal axis on the calibrator tool. The orthogonal axis for the north side is the y-axis. The visible side of this axis is the positive side, as depicted in figure 3.4.

We created a variable that stores the information about horizontal and vertical axis, and their direction for each camera:

```
CAMERAS = {
    'NorthWest': (('x-axis', 'left'), ('y-axis', 'positive'), 1),
    'NorthEast': (('x-axis', 'left'), ('y-axis', 'positive'), 1),
    'EastNorth': (('y-axis', 'right'), ('x-axis', 'positive'), 2),
    'EastSouth': (('y-axis', 'right'), ('x-axis', 'positive'), 2),
    'SouthEast': (('x-axis', 'right'), ('y-axis', 'negative'), 3),
    'SouthWest': (('x-axis', 'right'), ('y-axis', 'negative'), 3),
    'WestSouth': (('y-axis', 'left'), ('x-axis', 'negative'), 4),
    'WestNorth': (('y-axis', 'left'), ('x-axis', 'negative'), 4)
}
```

Here is an explanation of each element in the dictionary presented above:

```
'<Camera Name>': {
    (Axis with visible negative and positive side,
     Relative distance of positive side of the axis to the origin),
    (Axis with one side, Direction of the respective axis side),
    (North: 1; East: 2; South: 3; West: 4)
}
```

`CAMERAS['NorthEast'][0]`, explains that the horizontal axis, being the x-axis, is to the *left* of the new origin (O_{new}); `CAMERAS['NorthEast'][1]` explains that the orthogonal axis, being the y-axis, has its possible side visible from the camera perspective.

We define the orthogonal axis in two ways. (1) we can either take the cross product of the horizontal axis and the vertical axis (z-axis), or (2) use the annotations of the orthogonal cone to compute it similarly to the other axes. In method 1, since the plane spanned by $\{\vec{r_1}, \vec{r_3}\}$ is orthogonal to $\vec{r_2}$, we can then compute the third axis $\vec{r_3}$ as:

$$\vec{r_1} \times \vec{r_3} = \pm\vec{r_2}$$

The $\pm$ signifies that the positive side may shift sides with respect to O_{new} across cameras. The new basis for each camera pair was constructed so that $\vec{r_1}$, $\vec{r_2}$, and $\vec{r_3}$ were consistent ($\vec{r_1}$ pointing to the north, $\vec{r_2}$ to the east, and $\vec{r_3}$ to the ceiling). We can find the positive direction with the second right-hand rule, which states that $\vec{r_3} \times \vec{r_1} = \vec{r_2}$ in the case of north and east, and $\vec{r_1} \times \vec{r_3} = \vec{r_2}$ in the case of south and west.

The basis vectors were finally normalized to obtain unit basis vectors:

$$\frac{\vec{r_i}}{||\vec{r_i}||} = \hat{r_i}$$

$\hat{r_i}$ is a unit basis vector.

The first step is to compute the new origin, O_{new}. For pairs that are on the same side - NorthWest-NorthEast for instance - the new origin was the center of the negative and positive tips of the horizontal axis on the calibration tool. Corner pairs, however, were different. As a consequence of the 90°rotation between cameras in a corner-pair, they did not share the identical horizontal axis (Figure 2.1). We used the annotation of the origin for computing the origin in these situations.

Corner pairs share one side of each axis. In the case of NorthEast and EastNorth, the positive side of the x and the y axis are shared; every camera shares the same z-axis side, which is positive. We use these shared sides to compute the new basis vectors. When the sides are negative they are simpley inverted to become their positive counterpart.

Summary of how basis vectors are calculated

- New origin: $\pm(r_{h+} + \frac{r_{h-} - r_{h+}}{2}) = O_{new}$
- Horizontal axis (x-axis): $r_{h+} - O_{new} = \vec{r_h} = \vec{r_1}$
- Verical axis (z-axis): $r_{v+} - O_{new} = \vec{r_v} = \vec{r_3}$
- Orthogonal axis (y-axis):

$$r_{o+} - O_{new} = \vec{r_o} = \vec{r_2}$$
$$\vec{r_1} \times \vec{r_3} = \pm\vec{o} = \pm\vec{r_2}$$

Each basis vector is normalized, and used to create the linear map:

$$\begin{bmatrix}\vec{r_1} & \vec{r_2} & \vec{r_3}\end{bmatrix}^T = \mathbf{B}^T$$

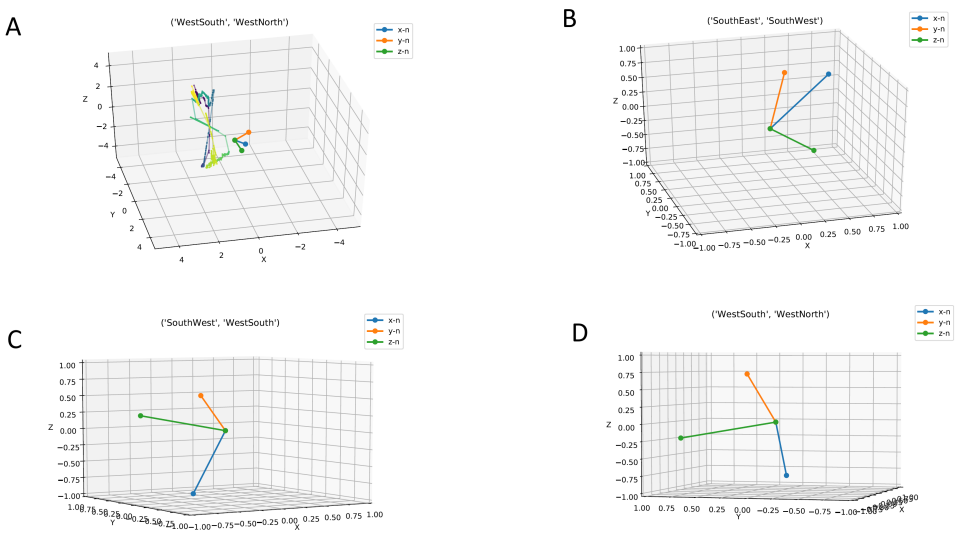


Figure 3.5: Depiction of the reconstruction of the new axes. A The reconstructed axes along with the trajectory of a pen where color codes for time. The colormap is viridis, which starts at blue and goes to yellow; **B, C, D** the reconstructed axes for the respective stereo cameras

3.2.7 Mapping coordinates to the same vector space, $\mathbb{R}^3 \rightarrow \mathbb{R}^3$

Following the definition of the linear maps (**B**) and $\vec{O_{new}}$, we were ready to map the coordinates to the same subspace. `deepcage.basis_label(config_path)` would detect the coordinate sets belonging to the DeepCage project, and map them to the subspace of the basis vectors projected by the calibration tool. This was done by `deepcage.create_stereo_cam_origmap(config_path)`.

3.3 Synchronization of video capture and electrophysiology

We needed to synchronize the camera capture in Bonsai and electrophysiological recordings by OpenEphys in order to correlate the data. The Open Ephys GUI's network events module supports receiving ZeroMQ messages from other ZeroMQ enabled applications. With that in mind, we created a Bonsai package named *BonsaiMQ* (https://github.com/caniko2/BonsaiMQ), that enabled sending messages in string format from Bonsai to any other ZeroMQ/NetMQ applications including Open Ephys. The package was uploaded as a Bonsai community package to the Bonsai community nuget repository.

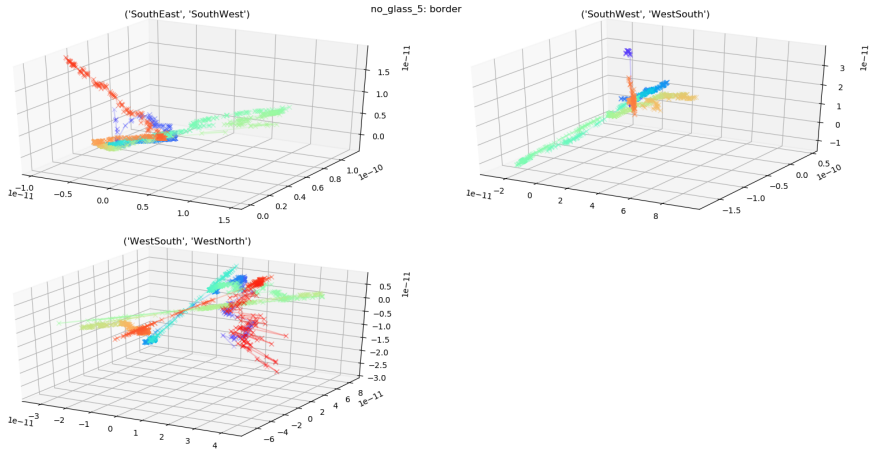
MQ: Message Queuing

ZeroMQ is a library that enables bidirectional communication by the TCP protocol , which is what we use BonsaiMQ. Moreover, ZeroMQ also supports UDP, which is monodirectional protocol. We chose to use TCP to ensure that signals sent from Bonsai are received in OpenEphys.

TCP: **T**ransmission **C**ontrol **P**rotocol
UDP: **U**ser **D**atagram **P**rotocol

In Bonsai, a timestamp is created for each frame encoded by a video capture stream, and sent to OpenEphys with BonsaiMQ. The timestamps are also stamped by the network events module on OpenEphys on arrival, which is what we need, comparing the clock of OpenEphys with Bonsai throughout a trial. The timestamps between Bonsai and OpenEphys are <1ms apart.

A



B

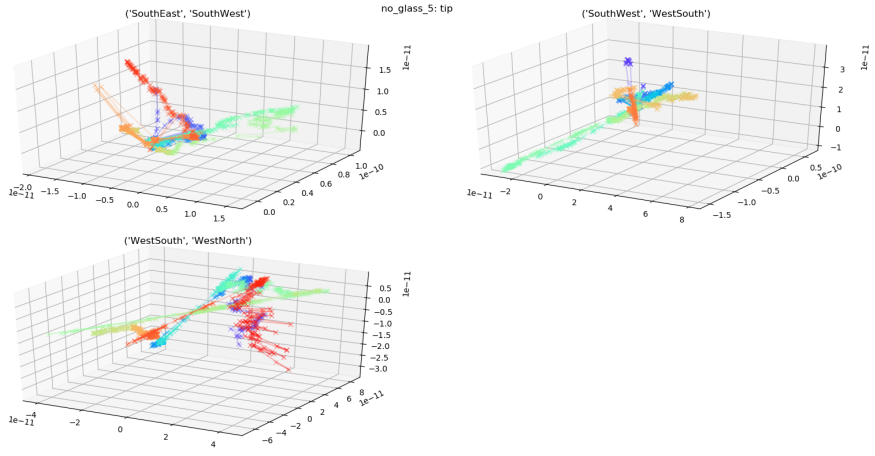


Figure 3.6: Depiction of the trajectory of parts of pen The rainbow color-map indicates time, which goes from blue to red in the . **A** The trajectory of the portion of the pen that is

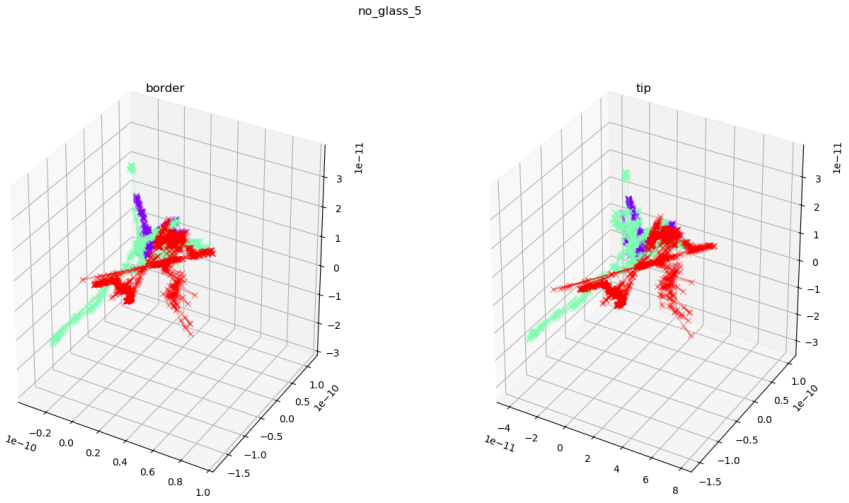


Figure 3.7: Depiction of the trajectories mapped to the new subspace. The color depicts the origin of the respective trajectory.

3.4 DeepLabCut contribution

The DeepLabCut package is maintained by the Mathis Lab[2] - Adaptive Motor Control - at Harvard. The project accepts contributions to the package. We made the following contributions:

- Improved method for selecting image pairs that have bad corner from the DLC 3D camera calibration.
- Found and fixed a bug that causes a crash when FPS metadata is 0. The FPS values is used for calculating the duration of a video, and when the FPS value is undefined the function returns 0, which leads to a `ZeroDivisionError`.
- Found and fixed a bug that did not save the triangulation results in csv-format if the triangulation had already been run for the respective dataset

The status of these contributions can be seen on the github pull request submitted by the project author.

[2] `http://www.mousemotorlab.org/`

4. Discussion

DeepLabCut, which is a relatively new package that aims to provide the means necessary for markerless pose-estimation and 3D reconstruction; moreover, the package was not feature complete for our case. Therefore, developed a Python package named DeepCage, which allows users to automate the creation of several 3D projects, to identify and reconstruct a common subspace, and to rebase the 3D trajectories from different stereo cameras onto the same global subspace. Moreover, the package also contains several functions that facilitate certain steps of the pipeline, such as the creation of project directories, and converting files to compatible formats.

It should be noted that DeepCage is not limited to the experiment design that is being presented in this project. DeepCage could be easily modified to deal with different experimental setups with multiple stereo cameras to perform 3D reconstruction of objects and pose.

Whereas DeepCage provide several functionalities to facilitate the creation and handling of several DeepLabCut 3D projects, the combination of the several reconstruction into a single model is still at an early stage and requires more tests and validation stages. Throughout this thesis, plenty of time have been spent in solving issues related to the entire data acquisition pipeline, such as synchronization of stereo cameras frames, and external synchronization of videos and electrophysiology (using `BonsaiMQ`). In the following, we discuss advantages and disadvantages of the adopted methodologies, as well as technical considerations on hardware and software choices.

4.1 DeepLabCut and DeepCage

DeepLabCut is a library that streamlines markerless pose estimation, and DeepCage is library that is built with the aim of expanding its capabilities. I believe that the opportunity cost of not using these frameworks is taxing on the quality of respective experimental results. The following section will outline the reason behind the position.

4.1.1 Markerless pose estimation

Tools such as DeepLabCut, make it far simpler for groups with little programming/technical backgrounds to utilize ANNs for purposes of tracking and reconstructing; however the tools does have its disadvanges[1]. DeepLabCut and its usage requires preparations, setup, and the disadvantages of relying on an ANN.

Our experimental setup required us to craft 3D printed arms, aluminum frames, and the purchase of 8 medium quality webcameras (Logitech C930e). This setup also need to be calibrated, and the calibration itself also required tests to assure its quality. DeepLabCut is a new package, and its lack in features also required us to create new ones, which we are presenting in this project,

[1] In an attempt to *steelman* our argument, we believe it is suitable to start by presenting the disadvantages of DeepLabCut

which took the full time of a master student, and mind space and part-time from two doctorates and the supervision of an Associate processor for approximately a year.

The tool also required investment in computer hardware; however, alternative methods such as cloud computing discussed in section 4.4.1.

; moreover, if the user requires means beyond the offers of the package, insight into the inner workings of the framework may be needed. This was certainly true in our case when we had to augment the triangulation function for our subpackage `deepcage.compute`.

Different methods can be used for analysis of behaviour of different organisms in 3D. In one study, multiple depth cameras were needed for obtaining the 3D data of rodents, as well as physics engine software for simulation of skeleton movements (Matsumoto et al. 2013). This, however, is limited to tracking the skeleton movement in addition to being obliged to use expensive cameras (around $200 per camera; Matsumoto et al. 2013) compared to what we have tried to accomplish in this thesis project. More advanced cameras also require more advanced PC properties. Identifying different parts of primary motor cortex have allowed to track movements of different body parts based on the electrical activity of neurons in the distinct brain area (Vargas-Irwin et al. 2010; Mimica et al. 2018); however, this makes it difficult to track subject's movement concomitantly with neural activity in other brain areas of interest, such as visual cortex.

With that in mind, DeepLabCut is the state of the art for tracking any type of object. If the ANN is trained properly, the mistake-rate may be as low as $\approx 0.1\%$, which is far beyond what physical markers have been able to achieve in poor conditions. As previously mentioned, the new paradigm that we are presenting in this project not only removes the need for markers in tracking, but we believe that it is suitable for any experiment that also requires reconstruction of objects such as the head or the body of an animal.

4.1.2 3D reconstruction

The combination of several cameras to make stereo cameras, and their combination with markerless pose estimation, can in principle provide 3D tracking and pose estimation without interference with the animals being recorded. However, during this project, we encountered some limitations in the reconstruction.

The calibration of the cameras and possibly the quality of the cameras themselves seemed to have largely affected the reconstructions. When reconstructing the calibration tool, in fact, we expected the reconstruction to yield a perfectly orthogonal basis, bu this was not exactly the case. We found that the 3D reconstruction of the calibration had angles between the axes deviating 10-20 degrees from the 90 degrees that ensures orthogonality. The same inaccuracy can be expected for the reconstruction of trajectories of body parts extracted with DeepLabCut. We identify two possible reasons for this: on one hand, the cameras were not perfectly fixed, and therefore they were subjected to misalignments and small shifts, that would be amplified by the undistortion procedure. Second, we believe that the quality of the cameras could play a role. For higher quality cameras, in fact, radial and tangential distortions are more uniform and easier to correct with undistortion techniques. All these factors have probably affected the quality of the presented results, but we still believe that the approach introduced in this thesis could be further explored and investigated by taking the above-mentioned precautions (e.g. using higher quality cameras and fixing more carefully the cameras to avoid unwanted movements). Alternatively, another solution would be to substitute the single cameras with built-in stereo cameras (e.g. a set of two cameras built on the same support). We believe that, moving forward and looking backwards, this could be the best solution to ensure a high-quality 3D reconstruction.

Glass and other mediums between the camera and the experiment It is important to note that recordings that are refracted are not compatible with the DeepLabCut 3D undistortion and the

triangulation step. In these steps, in fact, it is assumed that the ray/projection being traced is on a straight line, which is not the case for rays that have been refracted. Having different mediums between the experiment and the camera is, therefore, not recommended.

If differing mediums are not avoided, the user can deactivate undistortion to minimize the error caused by the refraction; however, this will increase the error from the imperfections of the camera of the resulting triangulation as it is dysfunctional. Fortunately, in our case the cameras are identical and have more or less the same biases during capture. However, the presence of the glass should be take into account when applying the triangulation projections.

This issue has been brought up in the context of OpenCV checkerboard calibration on StackOverflow (https://stackoverflow.com/questions/15419659/calibrating-camera-with-a-glass-covered-checkerboard). Here, it has been proposed that Automatic Differentiation may be the proper means for modelling the refraction by media such as glass. This method would account for the derivative of the refraction as we move within the image plane, and is most likely advantageous in the context of both undistortion and triangulation. This can certainly be added to our pipeline.

4.2 Experiment setup

4.2.1 Hardware considerations

Central processing unit and video capture + encoding

Receiving 240 frames per second (30 frames from 8 cameras) requires a lot of work from the central processing unit (CPU); consequently, we used a set of 2 computers to balance the workload. Moreover, with hindsight, we believe that a monolithic workstation with a CPU that has ≥ 18 cores is an ideal setup. The centralization of the computations conducted during experiments are easier to organize and store. Additionally, as of 2019, CPUs with corecounts above 18 is cheaper thanks to recent advancements in CPU technology.

Video recording device

We used Logitech C930e webcameras that capture 30 FPS. Because of a lack of intrinsic synchronization, we used Bonsai as the means for synchronizing them. This resulted in the FPS dropping to about 20-23 FPS, which may be inadequate for some experiments. Other cameras with a higher FPS count or the capability to synchronize may be advantageous over the C930e; however, it should be noted that C930e has native support for streaming frames over USB to the recording device, which many camera devices do not have. Our experimental setup, moreover, required us to turn auto focus and color correction off in order to prevent focus shifts during the video acquisition, and therefore, to avoid problems in the triangulation.

4.3 Softeare considerations

Selecting operating system

Picking the correct operating system (OS) for any enterprise level application would almost always conclude with a *distribution*, also referred to as *distro*, of Linux. A Linux distribution with a large community such as Ubuntu would almost always be the most effective and reliable OS. The creators of DeepLabCut used Ubuntu 16.04 Long Term Support (LTS) for their experiments. However, the thesis project relied on Bonsai that was only compatible with Windows.

Migrating the project to Linux could be justified for the reliability of the OS; however, the investment could be argued to be unnecessary as Windows 10 is stable to an extent. Ultimately, we chose Windows 10.

In a scenario which includes moving the project to Linux, a replacement for Bonsai would be necessary. The best solution, in that case, would be to utilize OpenCV in conjunction with C++. This setup requires the new code to make sure the stereo cameras are synchronized (capture frames simultaneously), and storing the frames in a video file such as AVI.

Using conda to create a virtual environment to run DeepLabCut

The manual installation of dependencies for DeepLabCut or any package that has multiple dependencies is a tedious task; consequently, it is sensible to utilize conda, which allows the author to automate the installation of software that is not available in pip. The DeepLabCut authors maintain several conda environments in the DeepLabCut repository.

The differences between these files were important to consider as they should be chosen with emphasis on the operating system or OS, and the type of processor to be used for the computational operations during the training of the deep neural network. We used Windows 10 as our OS, and used GPU for computing, and, therefore, chose *dlc-windowsGPU.yaml*, located in the DeepLabCut repository.

There are other alternatives to using conda, which can be found on the DeepLabCut repository; however, several of these methods were incompatible with Windows 10 at the time of the thesis project. The DeepLabCut Docker container, for instance, works only on Linux because Docker on Windows 10 does not support mounting the GPU. The Docker container solution would be the preferred method if not for the incompatibility with the OS as Docker containers are very easy to setup compared to conda, which require installation of several 3rd party software.

Anaconda vs Miniconda

Anaconda and Miniconda are quite similar software packages that install conda along with the given version of Python (2.7.2 or 3.x).

The difference between the two is that Miniconda is a minimal package that only includes the conda command line interface (CLI), while Anaconda is bundled with a GUI and other software used in scientific programming. The DeepLabCut authors picked Anaconda as their method for installing conda to their project computer. We, keeping the aforementioned facts in mind, decided to use Miniconda as our conda installer.

4.3.1 Video recording software

We used Bonsai, a visual programming software (Lopes et al. 2015), to program the video recording workflow. The asynchronous nature of the recording made it convincing to use Bonsai because of its simplicity; thereby, shortening development cycles until a proper method is found.

The setup could be taken a step further by writing software that is specific to our needs to increase performance, and possibly improved output. The software would be the same as the one mentioned in section 4.3.

4.4 DeepLabCut

DeepLabCut was both the pioneering and the only fully complete package in its category during project design. It was used for estimating the location of regions of interest throughout individual

recordings. These coordinates were triangulated with DeepLabCut 3D.

Tensorflow version

DeepLabCut and DeeperCut used the TensorFlow machine learning library (Abadi et al. 2016) developed and maintained by Google. The TensorFlow versions that have been tested by the authors were 1.0 to 1.4, 1.8, 1.10 and 1.11. Assuming that a hypothetical upgrade takes 10 minutes, or less, we decided to use the latest version of TensorFlow 1, which was 1.13.

4.4.1 Cloud computing

Laboratories that may not want to invest in expensive GPUs can utilize cloud computing services for training their ANN. These services are relatively easy to set up, and some, like Amazon Web Services (AWS), has native support for Jupyter Notebooks. DeepLabCut in light mode has experimental support for cloud computing, which has been promising in many cases.

This can be especially advantageous for laboratories that want to test the paradigm before investing into expensive hardware for the long term.

ANN training: GPU vs CPU

A decent GPU was always faster than its CPU counterpart in operations that were parallelizable. This phenomenon stemmed from the fact that these processors had different architectures. GPUs were designed for performing operations in parallel, and as a trade off required time to initialise. While CPUs were designed for performing input and output or I/O operations, very fast operation initialization. Consequently, GPUs were able to perform floating point operations per second (FLOPS) in the magnitude of 10^{12} or teraflops (TFLOPS); CPUs are able to do the same operations in the magnitude of 10^{9} or gigaflops (GFLOPS).

Every PC, including the project computer, requires a CPU to operate common operating systems such as Windows 10 and Linux. DeepLabCut users can, therefore, choose to invest in GPU provided they need to train their networks often. We chose to invest in a Nvidia GTX 1080 Ti, a GPU that was considered to have a good performance/price trade-off. The card had 11 GB of video random access memory (VRAM), and could perform 11.34 TFLOPS. The GPU training was conducted usually overnight, but sometimes for 4-6 hour, which yielded about 600 000 or 400 000 iterations; more than enough.

4.4.2 Pose estimation

Reconstruction, or pose estimation, can add extensive value to experiments. In fact, it enables one to accurately measure movement, which can then be related to other biological measurements (e.g. brian activity). As it makes extrinsic futures accessible for extraction from otherwise raw data. The main drawback of including such a tool would be the required workflow and computation. Not to mention the added layer of complexity.

2D vs 3D reconstruction

The spatial world has three dimensions. The information about the third spatial dimension may sometimes be redundant. For example, for the study of spatial navigation, rodents usually explore planar open-field or arenas, which makes tracking the third dimension unnecessary, i.e. not useful. In some applications, even the second dimension might be redundant. In some experiments, rodents

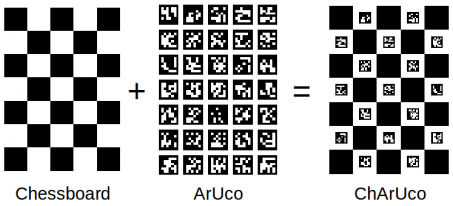


Figure 4.1: The depiction of how ChArUco board, the superior board for calibration was designed. The design of ChArUco utilizes a best of both world approach and includes elements from both the chessboard and the ArUco board

are trained to run back and forth on a linear track, and in that case 1D tracking is enough such as in the case of linear tracks.

Method for mapping 3D coordinates from 2D

Since we decided to perform 3D reconstruction, we had to determine the method for mapping 3D coordinates from 2D. There were several solutions for mapping the 2D coordinates to 3D. The most common solutions were depth estimation with a standard camera, and triangulation with stereo cameras.

Depth estimation is a paradigm that is gaining increasing levels of adoption in computer vision, and depth estimation of frames from a standard camera was a viable option. The depth is estimated by an ANN that utilizes the intrinsic information in the image to map the 2D points to 3D space. The categories of data include the borders between objects, and the individual brightness of objects. The biggest drawback of the solution was its layer of complexity, and computational requirements.

Alternatively, one can use stereo cameras and triangulation techniques. Two or more cameras that were spatially calibrated capture an image of the same area. Images taken simultaneously have their identical points of interest labeled, and their coordinates extracted; these coordinates are triangulated. The triangulation would yield a 3D coordinate. This method was repeated for every point of interest.

Video recording

The eight cameras stream 30 fps each with a resolution of 1920x1080 (Full-HD). The data had to be compressed into either a video or images where each frame would be one image. That meant that the computer compressed 240 fps where each frame is full HD.

Board for calibration

There were alternatives to using chessboard for calibration such as ArUco and ChArUco boards. These alternative boards increase the precision of the calibration.

The ChArUco board was not implemented into the 3D workflow of DeepLabCut, therefore; the initial design of the experiment utilized the supported chessboard instead. The test yielded optimistic results.

4.5 Conclusions

DeepLabCut represents the state-of-the-art pose estimation algorithm for use in animals' tracking. The recent addition of 3D reconstruction capabilities, however, has not been largely adopted by the community. In this project, we have attempted to use several cameras for a complete 3D tracking of animal behavior with the final aim to monitor vestibular modulation of visual neural activity. We have developed an experimental setup with 8 cameras and laid the theoretical background for remapping the 3D reconstructions of the different stereo cameras to the same coordinate system, defined by a 3D-printed calibration tool. Our 3D reconstructions, in all honesty, were not as accurate as we expected. We believe that a use of higher quality cameras or built-in stereo cameras could improve the performance of this method and enable a full 3D reconstruction of rodents' pose. In the process of combining several stereo cameras, several software tools were developed. BonsaiMQ is a Bonsai package for sending ZMQ messages for synchronization purposes. DeepCage is a Python package designed for easing the creation of and management of several DeepLabCut 3D projects. While the project necessitates more work in order to be deployed for experiments, we believe that this thesis work has set the basis for the use of deep learning augmented 3D pose estimation of rodents for neuroscience experiments in the CINPLA laboratory.

References

Abadi, Martın et al. (2016). 'Tensorflow: A system for large-scale machine learning'. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283.

Anderson, David J and Pietro Perona (2014). 'Toward a science of computational ethology'. In: *Neuron* 84.1, pp. 18–31.

Andriluka, Mykhaylo et al. (2018). 'Posetrack: A benchmark for human pose estimation and tracking'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5167–5176.

Bernstein, Nikolai (1966). 'The co-ordination and regulation of movements'. In: *The co-ordination and regulation of movements*.

Bonnevie, Tora et al. (2013). 'Grid cells require excitatory drive from the hippocampus'. In: *Nature neuroscience* 16.3, p. 309.

Bova, Alexandra et al. (2019). 'Automated Rat Single-Pellet Reaching with 3-Dimensional Reconstruction of Paw and Digit Trajectories.' In: *Journal of visualized experiments: JoVE* 149.

Bradski, Gary and Adrian Kaehler (2000). 'OpenCV'. In: *Dr. Dobb's journal of software tools* 3.

— (2008). *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc."

Camomilla, Valentina et al. (2018). 'Trends supporting the in-field use of wearable inertial sensors for sport performance evaluation: A systematic review'. In: *Sensors* 18.3, p. 873.

Cao, Zhe et al. (2017). 'Realtime multi-person 2d pose estimation using part affinity fields'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7291–7299.

Crevier, Daniel (1993). *AI: the tumultuous history of the search for artificial intelligence*. Basic Books, pp. 102–105.

Dell, Anthony I et al. (2014). 'Automated image-based tracking and its application in ecology'. In: *Trends in ecology & evolution* 29.7, pp. 417–428.

Deng, Jia et al. (2009). 'Imagenet: A large-scale hierarchical image database'. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Egnor, SE Roian and Kristin Branson (2016). 'Computational analysis of behavior'. In: *Annual review of neuroscience* 39, pp. 217–236.

Fee, Michale S and Anthony Leonardo (2001). 'Miniature motorized microdrive and commutator system for chronic neural recording in small animals'. In: *Journal of neuroscience methods* 112.2, pp. 83–94.

Felwat (2018). *Logitech C920 Adapter for RaffoSan's Camera Arm*. URL: `https://www.thingiverse.com/thing:3013280`.

Finkelstein, Arseny et al. (2015). 'Three-dimensional head-direction coding in the bat brain'. In: *Nature* 517.7533, p. 159.

Folk, Mike, Albert Cheng and Kim Yates (1999). 'HDF5: A file format and I/O library for high performance computing applications'. In: *Proceedings of supercomputing*. Vol. 99, pp. 5–33.

Gomez-Marin, Alex et al. (2014). 'Big behavioral data: psychology, ethology and the foundations of neuroscience'. In: *Nature neuroscience* 17.11, p. 1455.

Hebb, Donald Olding (1962). *The organization of behavior: a neuropsychological theory*. Science Editions.

Hutchins, John (2005). 'The history of machine translation in a nutshell'. In: *Retrieved December* 20, p. 2009.

Insafutdinov, Eldar, Mykhaylo Andriluka et al. (2017). 'Arttrack: Articulated multi-person tracking in the wild'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6457–6465.

Insafutdinov, Eldar, Leonid Pishchulin et al. (2016). 'Deepercut: A deeper, stronger, and faster multi-person pose estimation model'. In: *European Conference on Computer Vision*. Springer, pp. 34–50.

— (n.d.). 'DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model'. In: URL: `http://arxiv.org/abs/1605.03170`.

Jain, Arjun et al. (2014). 'Modeep: A deep learning framework using motion features for human pose estimation'. In: *Asian conference on computer vision*. Springer, pp. 302–315.

Kreiss, Sven, Lorenzo Bertoni and Alexandre Alahi (2019). 'Pifpaf: Composite fields for human pose estimation'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11977–11986.

Li, Xiu et al. (2019). '3D Pose Detection of Closely Interactive Humans Using Multi-View Cameras'. In: *Sensors* 19.12, p. 2831.

Lopes, Gonçalo et al. (2015). 'Bonsai: an event-based framework for processing and controlling data streams'. In: *Frontiers in neuroinformatics* 9, p. 7.

Machado, Ana S et al. (2015). 'A quantitative framework for whole-body coordination reveals specific deficits in freely walking ataxic mice'. In: *Elife* 4, e07892.

Martinez, Julieta et al. (2017). 'A simple yet effective baseline for 3d human pose estimation'. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2640–2649.

Mathis, Alexander et al. (2018). *DeepLabCut: markerless pose estimation of user-defined body parts with deep learning*. Tech. rep. Nature Publishing Group.

Matsumoto, Jumpei et al. (2013). 'A 3D-video-based computerized analysis of social and sexual interactions in rats'. In: *PloS one* 8.10, e78460.

McCulloch, Warren S and Walter Pitts (1943). 'A logical calculus of the ideas immanent in nervous activity'. In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.

Mehta, Dushyant et al. (2016). 'Monocular 3d human pose estimation using transfer learning and improved CNN supervision'. In: *arXiv preprint arXiv:1611.09813* 1.3, p. 5.

Mimica, Bartul et al. (2018). 'Efficient cortical coding of 3D posture in freely behaving rats'. In: *Science* 362.6414, pp. 584–589.

Minsky, Marvin and Seymour A Papert (2017). *Perceptrons: An introduction to computational geometry*. MIT press.

Nath, Tanmay et al. (2019). 'Using DeepLabCut for 3D markerless pose estimation across species and behaviors'. In: *Nature protocols* 14.7, pp. 2152–2176.

Newell, Alejandro, Kaiyu Yang and Jia Deng (2016). 'Stacked hourglass networks for human pose estimation'. In: *European conference on computer vision*. Springer, pp. 483–499.

Nobel Lecture. Ethology and Stress Diseases. (1973). [Nobel Media AB 2019]. Retrieved from https://www.nobelprize.org/prizes/medicine/1973/tinbergen/lecture/. URL: `https://www.nobelprize.org/prizes/medicine/1973/tinbergen/lecture/`.

Oremus, Will (2015). *What Is "TensorFlow," and Why Is Google So Excited About It?* URL: `https://slate.com/technology/2015/11/google-s-tensorflow-is-open-source-and-it-s-about-to-be-a-huge-huge-deal.html`.

Paszke, Adam et al. (2017). 'Automatic differentiation in pytorch'. In:
Peirce, Jonathan et al. (2019). 'PsychoPy2: Experiments in behavior made easy'. In: *Behavior research methods* 51.1, pp. 195–203.
pickle — Python object serialization (n.d.). URL: `https://docs.python.org/3.6/library/pickle.html`.
RaffoSan (2017). *T-slot V-slot Universal camera mount*. URL: `https://www.thingiverse.com/thing:2477180`.
Sarafianos, Nikolaos et al. (2016). '3d human pose estimation: A review of the literature and analysis of covariates'. In: *Computer Vision and Image Understanding* 152, pp. 1–20.
Schaefer, Andreas T and Adam Claridge-Chang (2012). 'The surveillance state of behavioral automation'. In: *Current opinion in neurobiology* 22.1, pp. 170–176.
Siegle, Joshua H et al. (2017). 'Open Ephys: an open-source, plugin-based platform for multichannel electrophysiology'. In: *Journal of neural engineering* 14.4, p. 045003.
Sun, Ke et al. (2019). 'Deep high-resolution representation learning for human pose estimation'. In: *arXiv preprint arXiv:1902.09212*.
Tinbergen, Niko (1963). 'On aims and methods of ethology'. In: *Zeitschrift für tierpsychologie* 20.4, pp. 410–433.
Tome, Denis, Chris Russell and Lourdes Agapito (2017). 'Lifting from the deep: Convolutional 3d pose estimation from a single image'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2500–2509.
Tompson, Jonathan J et al. (2014). 'Joint training of a convolutional network and a graphical model for human pose estimation'. In: *Advances in neural information processing systems*, pp. 1799–1807.
Tompson, Jonathan et al. (2015). 'Efficient object localization using convolutional networks'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 648–656.
Toshev, Alexander and Christian Szegedy (2014). 'Deeppose: Human pose estimation via deep neural networks'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1653–1660.
Vanzella, Walter et al. (2019). 'A passive, camera-based head-tracking system for real-time, three-dimensional estimation of head position and orientation in rodents'. In: *Journal of neurophysiology*.
Vargas-Irwin, Carlos E et al. (2010). 'Decoding complete reach and grasp actions from local primary motor cortex populations'. In: *Journal of neuroscience* 30.29, pp. 9659–9669.
Von Uexküll, Jakob and Georg Kriszat (1970). *Streifzüge durch die Umwelten von Tieren und Menschen: Ein Bilderbuch unsichtbarer Welten. Bedeutungslehre*. S. Fischer.