POLITECNICO DI TORINO

Master's Degree in Biomedical Engineering

HELMHOLTZ PIONEER CAMPUS

Passive Tomography

Tools for extracting three-dimensional information from monocular video

Supervisor

Dr. Thomas BISCHOF

Co-Supervisors

Dr. Oliver BRUNS

Prof. Filippo MOLINARI

Candidate

Ilaria BALBA

October 2020

Abstract

Scientists working with animals aim to improve the health of people by understanding what causes diseases and how they may be treated and prevented. This leads animals under study to be commonly affected by experimental conditions. Moreover, research often goes towards relatively invasive questions, which leads it to adopt invasive methods. For instance, the vascular structure of the brain is a popular topic in research. If we want to follow the blood flow in order to analyze the brain activity, we are forced to adopt invasive techniques, such as the fMRI, which require the animal to be restrained or anesthetized. It is well known that anesthesia affects the acquisition of physiological data such as the heart rate, the breath rate, and the gut movements, as well as restraining the animal.

Investigation into mice is further combined with the use of new imaging techniques. *Short-wavelength infrared region* imaging (SWIR) is an innovative technique for pre-clinical and clinical imaging. Currently, SWIR researchers are working to produce new detection systems, and new contrast agents to perform imaging of mice which are awake and unrestrained. For example, vascular contrast agents can be used to visualize blood vessels through the skin of an animal, which can allow us to follow the blood flow. Since the mouse is unrestrained during the acquisition, we can obtain physiological parameters much more representative than the animal under anaesthesia. Our goal is to track vascular networks in a mouse over time without needing to perform invasive imaging at each time point. By performing SWIR imaging, it may be possible to extract this information as a mouse moving in its cage in which it may show us various parts of its body and we may collect images over time. Is it possible though to correlate those images with a known 3D model of what the mouse looks like?

In this thesis project, we focus on methods for extracting information about the mouse movements over time. First, we investigate how photogrammetry works, state-of-the-art of 3D reconstruction, and how far our research can go exploiting its algorithm and computational geometry. We generate a surface mesh and texture of a mouse, using photogrammetry and SWIR images of a mouse with a vascular contrast agent. In this study, we found out that static scenes and non-deformable objects were adequately reconstructed, but the results changed with objects in motion. Therefore, we divide our efforts into two categories: rigid and deformable bodies in motion.

The skull of a mouse is a rigid body that, even if it moves, the distance between some selected points is fixed in 3D space. In a video, the points may appear closer or further from each other depending on how the mouse rotates its head, since they represent the projections of the 3D model in 2D. We perform motion tracking by using a *deep learning* tool called DeepLabCut to estimate the skull movements over time, a technique known as *pose estimation*. We found out that the orientation of a rigid body can be easily extracted if we know the 2D projections of its points and how they are related in 3D space.

If the object is a deformable body in motion, such as fingers in a moving hand, the distance between its points may change over time. In order to compensate for the body deformation, we perform stereo vision, a process of extracting 3D information from multiple 2D views. We adopt a mirror to have a second view of the object, we track selected points by using DeepLabCut, and we exploit the relation between object and its reflection to triangulate those points. We finally obtain a preliminary 3D deformable object, complete in all its parts.

Acknowledgements

This work has been possible thanks to the great deal of support I have received from wonderful people.

I would like to thank my supervisor, Dr. Thomas Bischof, for his guidance through each stage of this work. Thank you for teaching me how to think and learn from complications. Each of our meetings has enriched me in many aspects, both technical and personal.

Kind thanks to Dr. Oliver Bruns and the Bruns Team of the Helmholtz Pioneer Campus institute, for your help when I needed explanations on your wonderful work. But above all, for making me feel an important team member right from the start. You were fundamental to settle me in a world that is totally new to me.

I would like to acknowledge Prof. Filippo Molinari for inspiring my interest in the world of medical imaging. From year to year, his lectures motivated me to follow this path.

To my friends and colleagues, thank you for making me miss college and desperate student life more and more every day. To my friend Paolo, for having shared with me the adventures (and misadventures) of recent years, thanks for being there. To my forever flatmates, Giulia and Ivana, who I cannot wait to see again, anywhere in the world.

To my brother and sister, and first best friends, Federica and Vincenzo, that since we have been apart, I've really got what having a deep and indelible bond means. Thank you for being a part of me.

To my parents, Stefania and Giovanni, for whom any kind of thanks would never be enough. Thank you for supporting all my choices, my whims, and for giving me a future. I miss you every day.

To my love, Luigi, thank you for loving, raising, and supporting me. Thanks for being my best hiding place, and for building a life with me, even though it seemed impossible. We made it.

Ilaria Balba

Table of Contents

List of Tables vii

List of Figures viii

1 Introduction 1

1.1 Exploring imaging technologies: MRI, NRI, and SWIR 2

1.1.1 High speed imaging to study unrestrained and not anaesthetized mice 4

1.2 Region tracking and pose estimation 6

1.2.1 Deep Learning and ResNet 9

1.3 3D reconstruction 16

1.3.1 Camera calibration 18

2 Photogrammetry in static and non-static objects 22

2.1 Photogrammetry pipeline 23

2.2 3D reconstruction of an horse hoof, mouse belly, and a mouse in motion 30

2.2.1 Still horse hoof and camera in motion 31

2.2.2 Still camera moving around a mouse belly 32

2.2.3 Still camera and mouse in motion 33

2.3 Results and discussion 35

3 3D pose estimation of a mouse skull from a monocular video 36

3.1 SWIR imaging of a mouse with vascular contrast agent 37

3.2 DeepLabCut to track 2D points in a mouse skull 40

3.2.1 Network architecture 42

3.2.2 Network performances 43

3.3 Detection of mouse skull orientation 48

3.3.1 Validation results of skull orientation 50

4 3D pose estimation of an hand in motion with camera-mirror stereo vision 55
4.1 Datasets: calibration and hand skeleton 57
4.1.1 Calibration video: ChArUco board 57
4.1.2 Object in motion: hand skeleton 59
4.2 Pre-processing data 60
4.2.1 Extracting real and virtual views from a single camera system 60
4.3 DeepLabCut to track 2D points in an hand skeleton 62
4.3.1 Networks comparison 63
4.4 Extracting 3D pose by triangulating points with Anipose 73
4.4.1 Calibration of multiple cameras 74
4.4.2 3D skeleton 79

5 Conclusion and future experiment 83

Bibliography 88

List of Tables

3.1 Network parameters. The ResNet-50 model is trained on 20 frames for 500’000 iterations, and tested on other 20. 43

4.1 Dataset split in training and test set. The remaining frames are the validation set 63
4.2 Networks parameters. We train two neural network (ResNet-50 and ResNet-101) by using the same parameters (training and test size, iterations) 63
4.3 MAE averaged on three shuffles for training an test set for ResNet-50 and ResNet-101 72

List of Figures

1.1 SWIR intravital imaging in a mouse with a cranial window. Principal component analysis used to distinguish the tumor (green), arteries (red), from veins (blue) in the brain, information colour-coded to create a multicolour angiograph (a); flow maps of microvascular networks (b-c). Figure adapted from [1] 3

1.2 High-speed SWIR imaging for contact-free monitoring of heart (red ROI) and respiratory rate (blue ROI) in anaesthetized (b) and awake mice (e). The respiratory rate of this anaesthetized mouse was 84 breaths min^{-1} (c) and the heart rate was 130 beats min^{-1} (d). A respiratory rate of 300 breaths min^{-1} (f) and a heart rate of 550 beats min^{-1} (g) was observed in this awake but resting mouse. Figure adapted from [1] 5

1.3 Azure Kinect DK sensor (left) and its health application to track and monitor exercise movements and overall form (right). Figure adapted from [2] 6

1.4 Example of pose estimation result obtained with DeeperCut model on a Multi-Person dataset. Figure adapted from [4] 7

1.5 A regular 3-layer Neural Network: the input layer receives the input data, which are transformed through the network by the hidden layers, and the output layer represents the final class scores 9

1.6 Simple Convolutional Neural Network: convolution, ReLU, pooling operations, and classification layer. For CNN the input is an image that is transformed through the layers to a single vector that represents the final class score. The neurons in a layer are connected only to a small region of the previous layer. 10

1.7 Residual learning: comparison between *plain network* (a) and *short-cut* (b) [6]. 12

1.8 ResNet baseline architecture [7]. The Cfg[i] blocks differ from each other for the size of the input and for the parameters of the used feature maps. they are 4 convolutional layers containing 3 residual blocks each; depending on how many layers we want to add each Cfg can be repeated multiple times (e.g, for 50 layers: $\text{Cfg}[0]\times 3, \text{Cfg}[1]\times 4, \text{Cfg}[2]\times 6, \text{Cfg}[3]\times 3$) 14
1.9 Example of how deconvolution works with a 3x3 kernel on input 2x2, stride 2 to obtain the original size 5x5. The white pixels represent high probability that a body part is there located, and dark pixels low probability. Notice that from the down-sampled to the up-sampled image, there is consistency in probability body part location 15
1.10 Final score-maps assigned for *each* body part on the original image. The outcome of the deconvolutional layers is exactly the input image (with its original size), and a score map associated to each label that suggests where the label is located accordingly to the prediction. . . 16
1.11 Real-time facial tracking system that captures performances at high fidelity. From a monocular input (left), the global shape as well as local details are captured (center and right). Figure adapted from [8] 17
1.12 3D scanner from [9]. This is a cameras structure organized in such way to take in a single shot the higher number of details of the subject place in the middle. The outcome is a 3D body scanning. . 17
1.13 Polynomial curve defined in homogeneous coordinates (blue) and its projection on plane, rational curve (red). *By Wojciech Muła - Own work (Python script, final touches Inkscape), CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1196334* 19
1.14 Illustration of the pinhole camera model: the point P in 3D is projected into a plane with (u, v) coordinates, losing information about its third dimension z 19
1.15 Illustration of the pinhole camera model: second camera seen as rotation and translation of a first camera (left), multiple views of the same camera with respect to the first reference view (right) . . 21

2.1 A typical photogrammetry pipeline for extracting mesh and texture information from a set of still images [10] 23
2.2 Scale-Invariant feature transform [11]: SIFT algorithm detects significant patches of images, and it is able to detect those patches in any images irrespective to the image transformations (translation, rotation, or scaling) 24
2.3 Example of scale-dependent detector. At Scale-1, the window is well detecting the arrow corner, but at Scale-2 the window is too small . 24

2.4 Pyramid downscaled images. The image is blurred with Gaussian filter (LoG), and downsampled to emulate different scaling factor. By computing the Difference of Gaussian (DoG), the research of local maxima is simplified. 25
2.5 Research of local maxima between DoG in different scales. The actual pixel is compared with a neighbourhood of its scale, and other two of the nearby scales 26
2.6 Magnitude and direction of gradient. The above moon [12] is the original image, while the moon below is its gradient. The homogeneous parts of the image are dark, while the regions with high pixel values changes are brighter (see the boarder between the background an the object). The gradient magnitude represents how big is this change, while the gradient direction is in which direction is the highest pixel variation. To the selected pixel in the original image, we can associate an arrow directed to the highest variation, whose length is proportional to the magnitude of this change. 27
2.7 Orientation histogram referred to magnitude and direction gradient of Figure 2.6: each bin represents how many pixels have their direction gradient oriented in a certain range of degrees. Moreover, the frequency is weighted based on the gradient magnitude. 27
2.8 16x16 window oriented with respect to the keypoint orientation (left), 8 bin orientations for each 4x4 sub-block (descriptor vector of 128 elements) [13] 28
2.9 Example of pair of images with matched descriptors. The horse hoof capsule was provided by Tjadina Klein, and imaged by Tjadina Klein and Thomas Bischof 29
2.10 Iterative triangulation. From an initial two views reconstruction, the estimation of the 3D position is iteratively corrected by the addition of multiple views 29
2.11 Example of the depth map, mesh, and texture of an horse hoof. The horse hoof capsule was provided by Tjadina Klein, and imaged by Tjadina Klein and Thomas Bischof 30
2.12 Horse hoof: cameras position estimation and depth map. The horse hoof capsule was provided by Tjadina Klein, and imaged by Tjadina Klein and Thomas Bischof 31
2.13 Acquisition method: rotating mouse (left), rotation imaging of a tumor (right). Figure adapted from [14] 32
2.14 Camera positions around the object, as the object is still and the camera in motion. From the left to the right: set of images, detection of sift points, reconstructed object with estimation of camera positions 33

2.15 Camera positions around the object, as the object is still and the camera in motion. From the left to the right: set of images, detection of sift points, reconstructed object with estimation of camera positions 34

2.16 Reconstructed mouse. Visible errors of reconstruction: double tail and background fused with the mouse 34

3.1 Electromagnetic Spectrum Illustrating SWIR wavelength range. Figure adapted from [15] 37

3.2 Acquisition method and camera setting of SWIR imaging. Detection system composed by the InGaAs camera, and the illumination system composed by the laser, the square engineer diffuser. With the adoption of filter where required. 38

3.3 Absorption (blue) and emission (orange) spectra of Chrome7 dye. The main peak of absorption is 962 nm, while the light emitted is detected from 1150 nm. 39

3.4 Example of frames acquired with SWIR technology. The mouse is awake and it is moving around the chamber. This dataset was acquired by Bernardo Arús and Emily Cosco 39

3.5 DeepLabCut workflow: after the frames extraction, it is necessary to label the body part we want to track. Some of those frames will be used to create a training set and to train the network. At the end of the training, the Neural Network will have learned to correctly label all the remaining frames. If the results are not satisfactory, we can refine the training set and re-train the network. 40

3.6 GUI of DLC toolbox to label each frame [3] 41

3.7 DLC architecture to predict the body-part locations. The training set is used to train a ResNet-50 whose Classification layer is replaced with deconvolutional layers, in order to obtain spatial probability densities for each body part. 42

3.8 Region-based classification with minimum distance between predicted and manual label. The cross is a prediction of the network. We define a criteria of minimum distance to collocate each prediction to a class. The manual label with whom the prediction has the minimum distance is the outcome class. The cross prediction is classified as *EyeR* because it is the nearest manual label. 44

3.9 Confusion matrix example. Multi-class problem reduced to a binary classification problem. 45

3.10 Test set confusion matrix and performances for the ResNet50 with 500’000 iterations 47

3.11 Wrong classification but correct predicted location. The *EyeR* manual label is missing, and *EyeR* prediction is classified as *EyeL* because of the minimum distance . . . 47
3.12 Mouse skull world coordinates. The **x** and **y** coordinates of each label, with respect to the *Nose*, is physically measured by using *Fiji-ImageJ* on two frames: one in which the skull is aligned with *yz* plane, another in which it is aligned with the *xz* plane. Existing sketch adapted to the context. . . . 48
3.13 Mouse skull direction over time. The red dot are the predicted label, the blue line is a visualization of the predicted orientation of the mouse skull. . . . 50
3.14 Angle between two vectors oriented accordingly to their rotation matrices . . . 52
3.15 Comparison between numerical an geometric analysis of skull orientation in some test frames . . . 52
3.16 Predicted and manual skull orientation in 2D and 3D for different frames. The first frames are less illuminated then the last ones, this may cause errors in the prediction. . . . 53

4.1 Individual behaviours of a mouse [17] . . . 55
4.2 Total pipeline for monocular stereo vision experiment using DeepLabCut and Anipose . . . 56
4.3 Charuco board definition [19]: it is a combination between Chessboard and ArUco, in order to detect markers with more accuracy . 58
4.4 Charuco board used for calibrating cameras in our experiment . . . 59
4.5 Frames of moving hand with camera-mirror setting. Real object on the left, and its reflection on the right of each figure. From different frames, not always all fingers are visible . . . 59
4.6 Pre-processing calibration video. The input on the left is the original video (green square indicate real object, blue square the reflection). We first select and crop the real view, then the reflection is selected, cropped and flipped. The coordinates of the manual selection are saved. . . . 61
4.7 Pre-processing object video. On the left, the original input of the moving object (green square indicates the real view, and the blue square the reflection). Without manually selecting, the software automatically crops and flips the video by using the coordinates saved from the pre-processing of the calibration video. . . . 61
4.8 General network architecture pipeline (left), Hand model with 21 keypoint labels (right) . . . 62

4.9 Human and predicted labels with level of confidence 64
4.10 Test set confusion matrix and validation metrics of ResNet-50 . . . 65
4.11 a) Wrong prediction even if all the body parts are well visible (*tip4* predicted as tip5), b) Due to the overlap between the two fingers, the network has misdetected the position of labels *DIP5* and *TIP5*, c) *PIP5* and *DIP5* are well predicted, but because the human labels are missing, the minimum distance is automatically measured with *PIP4* and *DIP4*, d) both *MCP4* and *MCP5* are shifted of a bunch of pixels from their real position, but while *MCP4* has the minimum distance with the real itself, *MCP5* has its minimum distance with *MCP4*, otherwise it might have been considered correct. 67
4.12 Greedy algorithm. In order to resolve the mis-classification caused by the minimum criteria, we propose a greedy algorithm to assign each predicted label to a single class, avoiding multiple assignations 68
4.13 Loss function for ResNet-50 and ResNet-101. ResNet-101 plateaus faster than ResNet-50. 69
4.14 Test set confusion matrix and validation metrics of ResNet-101. . . 69
4.15 *T-test* of three samples for each ResNet, repeated for the *Macro-Precision, Macro-Recall* and *Macro-F1-score* 70
4.16 Comparison between predictions of training frames and test frames: CAM-A (*above*), CAM-B (*below*) 72
4.17 Translation and rotation of the right camera with respect to the left camera 75
4.18 Epipolar geometry of two cameras 75
4.19 Camera-mirror setting 78
4.20 Frames of the 3D reconstruction of the moving hand in time sequence 80
4.21 Lengths of the skeleton, comparison between ground truth and averaged predicions 81
4.22 Men absolute Error calculated for each segment of the skeleton . . . 82

5.1 Camera-mirror setting combined with SWIR imaging. 86

Chapter 1

Introduction

Thanks to the rapid growth of technological innovations over the last decades, biomedical imaging has been making great advances. The introduction of computer vision and artificial intelligence techniques in bio-imaging is speeding up the development of new clinical applications, which until recently were just scientific studies. For example, a great improvement is made on the detection of patterns in images. Nowadays, the detection is faster achieved and supported by *Computer Aided Diagnosis (CAD)* systems, which exploit new algorithms to analyze the common features between images and classify them with respect to a supervisor, such as an expert in the field, or even by automatically learning from data. In fact, improvements are important also in data analysis. Just think of the huge amount of healthcare data that are every day more difficult to interpret due to their exponential increase, they can now be processed at higher speed with much more accuracy by deep learning applications. Scientists are working to improve the adoption of these new technologies in every field, and new challenges arise. These new imaging techniques are often experimented on animals before being adopted in the clinical use. However, animals under study are commonly affected by experimental conditions. Moreover, research often goes toward relatively invasive questions, which leads it to adopt invasive methods. For instance, the vascular structure of the brain is a popular topic in research. If we want to follow the blood flow in order to analyze the brain activity, we are forced to adopt invasive techniques, such as the fMRI, which require the animal to be restrained or anesthetized. It is well known that anesthesia affects the acquisition of physiological data such as the heart rate, the breath rate, and the gut movements, as well as restraining the animal. Studying animals that are not affected by stressed conditions and anesthesia is the first challenge we want to face. If we are able to follow a mouse in its natural behaviour, we can follow at the same time its physiological activity more realistically. Moreover, we want to extend this study to a more complete view: the 3D visualization of the subject. In addition to the study of the blood flow in the vessels, we are also interested in studying the vascular structure of the brain, and so how the network is organised and its properties. This may allow us to answer some questions such as how a tumor affects and increases over time, or more general questions as if there is symmetry between right and left sides, or find relation within different animal species.

In this introduction, we discuss about the *Short-wavelength infrared region (SWIR)* technology, an innovative technique for pre-clinical and clinical imaging. We introduce the advantages of this technique and the researchers goals to improve its applications in studying animals. Afterwards, we introduce the stat-of-the-art of 3D reconstruction, and how its technology, combined with the new deep learning methods, can be adopted for the purposes of our project.

1.1 Exploring imaging technologies: MRI, NRI, and SWIR

Imaging techniques allow us to visualize biological processes taking place in living organisms. Through these techniques we can see things below the skin of an animal, but depending on which technique we are adopting and the problem we want to face, we can have different properties and advantages. *Magnetic Resonance Imaging (MRI)* is a non-invasive imaging technique that produces detailed anatomical images. It allows us to achieve really high depth penetration and to analyze smaller parts such as capillaries in the brain, but at the same time it requires the subject to be totally still during the acquisition since it takes some time to acquire data. On the contrary, *Near-Infrared Region (NIR)* fluorescence imaging is a fast, convenient and non-invasive imaging technique for visualizing deep-tissue structures. Thanks to its high speed acquisition, we can obtain no-blurred images even if the subject is moving during the acquisition since commonly the imaging is faster than the motion. However, NIR does not achieve the same level of depth penetration typical of MRI imaging. A good compromise between the two aforementioned techniques is the *Short-wavelength Infrared Region (SWIR)* imaging. This technology is able to obtain higher level of depth penetration than NIR, but keeping the high speed not reached by MRI. Due to reduction of background signal and scattered light, the SWIR imaging technology has higher contrast, penetration depths, and minimal tissue autofluorescence. Therefore, for the purpose of our experiment, we can list the main advantages:

1. **Imaging at depth**: SWIR achieves higher level of depth penetration, required if we are interested in defining smaller elements, such as the smallest blood vessels of the brain
2. **Imaging at speed:** SWIR is a fast imaging technique, in fact, if we want to acquire non-blurred images, we need the imaging to be faster than the motion.
3. **Natural condition**: thanks to its high speed in acquisition, SWIR allows scientists working with animal to avoid anaesthesia; animals can move freely during the imaging, without losing resolution.

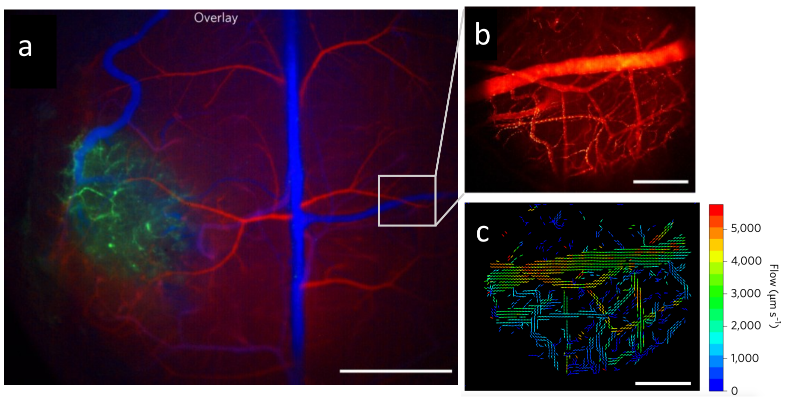


Figure 1.1: SWIR intravital imaging in a mouse with a cranial window. Principal component analysis used to distinguish the tumor (green), arteries (red), from veins (blue) in the brain, information colour-coded to create a multicolour angiograph (a); flow maps of microvascular networks (b-c). Figure adapted from [1]

In [1], these advantages are more explicit. In order to obtain an higher definition of the capillaries, [1] performs intravital microscopy to investigate the brain vascular network. In Figure 1.1 (a-c), Bruns *et al.* [1] used the benefits of SWIR imaging to image a glioblastoma multiform tumour growing in a mouse brain through a transparent cranial window, and thanks to computational tools, it was also possible to distinguish arteries from veins. The blood flow of the brain is of great importance for metabolic activity. SWIR is an higher speed imaging compared to other techniques such as fMRI and CT, which take few seconds per frame, or the scan which takes minutes.

1.1.1 High speed imaging to study unrestrained and not anaesthetized mice

The acquisition speed is one of the main reason why commonly the animal is anesthetized or restrained during the acquisition. Moreover, imaging techniques such as MRI, X-ray scan, and CT, force us to use anaesthesia in animal since the quality of the acquisition depends also on the imaging geometry. For example, to perform MRI imaging, the animal should be positioned within an MRI scanner that forms a strong magnetic field around the area to be imaged. Or, the CT scan can generate a three-dimensional image from a series of two-dimensional images taken by rotation around a fixed axis, while the subject is placed inside the rotating scan. NIR and SWIR have the great advantage of not requiring complex machinery. They consist in simply illuminating the regions of interest and, thanks to excited contrast agents, they receive the fluorescent signal back. Therefore, thanks to the simplicity of the imaging geometry, we can avoid anesthetizing or restraining the animal. This is decisive if we want to perform macroscopy to measure vital signals, because anesthesia greatly affects the results.

In Figure 1.2, [1] shows how the heart and respiratory rate are profoundly affected by anaesthesia. Current approaches for measuring the heart rate in awake mouse adopt telemetric sensors which are surgically implanted. Thanks to SWIR, we can detect signal coming from the heart and the liver, and avoid invasive technique. In Figure 1.2(b,c,d) the mouse is anesthetized during the acquisition. [1] selected two regions of interest to measure heart and liver intensity pixels. Based on the change in intensity over time, it is possible to detect fluctuations which represent heart and breath rate of the animal. Comparing the signals coming from

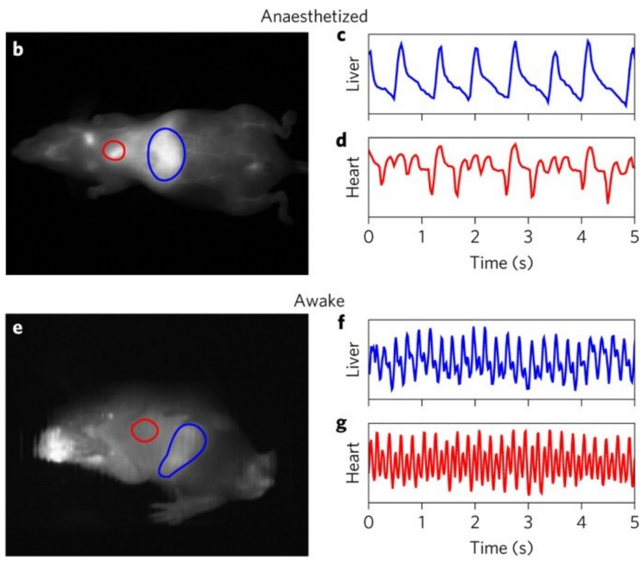


Figure 1.2: High-speed SWIR imaging for contact-free monitoring of heart (red ROI) and respiratory rate (blue ROI) in anaesthetized (b) and awake mice (e). The respiratory rate of this anaesthetized mouse was 84 breaths min^{-1} (c) and the heart rate was 130 beats min^{-1} (d). A respiratory rate of 300 breaths min^{-1} (f) and a heart rate of 550 beats min^{-1} (g) was observed in this awake but resting mouse. Figure adapted from [1]

the two states of the animal (anesthetized in Figure 1.2b and awake in Figure 1.2e), the fluctuations have changed very clearly. In contrast to other techniques such as MRI, CT and ultrasound, this approach allows study of awake mice, and avoid anaesthesia.

We have seen how the interest in studying the brain vascular network leads us to adopt invasive methods. With SWIR imaging we can achieve high level of resolution, and detect even the smallest vessels of the brain, and some vital signals can be easily detected if we look at defined region of the mice body. However, anaesthesia can effect them. In this thesis, we will exploit the advantages of SWIR macroscopy to detect the signal of a vascular dye, which yields relatively transparent mice and allows us to identify blood vessels. The aforementioned mouse will be totally free of anaesthesia or any sort of restraint. We will propose a solution to track a region of interest while the mouse is freely moving, in order to follow, for example, the vital signals we are interested in. Moreover, the tracking method can be extended to a three-dimensional visualization of the interesting body part such as the skull, since it contains useful information of the brain vessels.

1.2 Region tracking and pose estimation

A mouse free of anaesthesia and unrestrained moves during the acquisition. If we want to avoid any kind of restraint, and we want to detect signals that change over time, we have to find a way to detect for each video frame our region of interest. Of course, we do not want to manually select the ROI for each frame, or at least avoid it as much as possible, and for this reason, we investigate new methods to automatically do it.

Region tracking and pose estimation are widely studied problem in computer vision. They are techniques that predict and track the position and orientation of an object in images or videos. Recent developments in deep learning techniques have brought significant progress and remarkable discoveries in this field. Being a technique that concerns the study of humans, animals and objects, pose estimation has applications in many fields such as human activity detection, augmented reality, gaming and robotics. For example, in the field of animation and gaming, characters animation begin to be simplified and automated in many ways, replacing the traditional manual process that was based on expensive capture motion systems. Therefore, character animation is no longer based on markers or specialized suits, but on capturing human player movements and using them to make the actions of virtual characters, making the game experience immersive. Other interesting

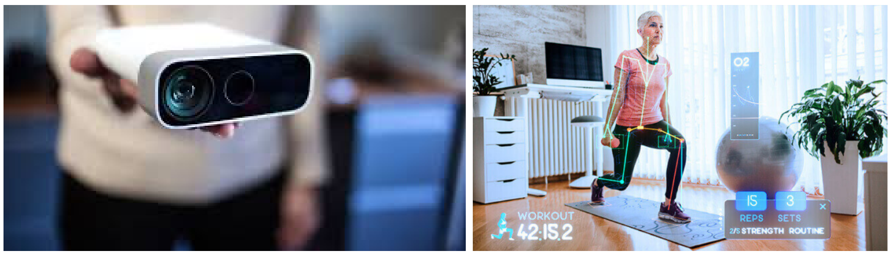


Figure 1.3: Azure Kinect DK sensor (left) and its health application to track and monitor exercise movements and overall form (right). Figure adapted from [2]

applications are based on using computer vision to track and monitor exercise movements and overall form, in order to have, for example, a virtual rehabilitation solution, or prevent and mitigate potential patient accidents (see Figure 1.3). This kind of applications have become popular thanks to Microsoft Kinect depth camera and advances in 3D pose estimation and gesture recognition [2].

In this section, we will provide an overview of real-world use cases for human and animal pose estimation and we will give a general description of the most used methods. We will dive a bit deeper for what concerns the existing tools for animals tracking pose, reporting the 2D pose estimation procedures used by a distinctive open-source toolbox called DeepLabCut [3].

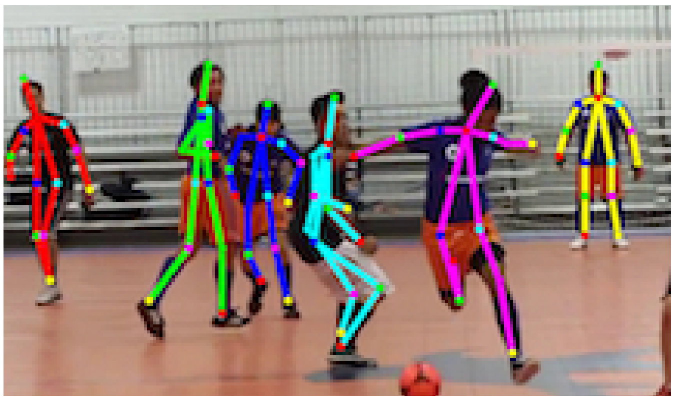


Figure 1.4: Example of pose estimation result obtained with DeeperCut model on a Multi-Person dataset. Figure adapted from [4]

Human pose estimation has been used for human activity, gesture and gait recognition. It is generally based on the generation of body part hypothesis, and so by referring the prediction to a model of what the human body looks like, commonly represented by a skeleton made of segments and nodes. Essentially, all pose estimation methods have a component that detects and estimates the body parts and their positions by using deep learning architectures. Moreover, pose estimation of human activities has recently made strong progress extending the prediction from a single individual to a multiple persons detection, as [4] shown with Figure 1.4, by using their DeeperCut model. Furthermore, the estimated pose are used to analyze variations over a period of time, in order to evaluate, for example, postural issues (e.g scoliosis), or to train a machine to recognize which activity the human is performing. However, these studies are feasible since the subject cooperates with the researchers, and we can easily ask for performing a particular pose or simply for staying still. It is evident that with animals there are some difficulties.

To measure animal behaviour, scientists have recently started adopting these new deep learning tools to automatically track specific part positions of the animal body. One of the main advantage is to avoid the use of physical markers on animals. In fact, traditional techniques involve the combination of video recordings with reflective markers applied in positions of interest that allow to easily track body parts [5]. However, such systems can be expensive and potentially distracting to animals, and markers need to be placed before the recording, and so it is necessary to pre-establish which functions we want to track. Moreover, if we want to track body parts that are under the skin such as blood vessels or organs, we should consider further complications. DeepLabCut is an efficient method for markerless pose estimation of animals, which can help us to evaluate the geometrical configuration of an animal body part without knowing a priori the points locations we want to track. Starting from a monocular video of the mouse, we can perform multiple experiments without being constrained to follow a single solution. Moreover, deep learning architectures commonly require large dataset (e.g ~25,000 extracted frames from all collected videos) to achieve acceptable performances, while DeepLabCut demonstrates that a small training set (~200 frames) can be sufficient to train the network, and achieve high-level of labeling accuracy. This is one of the best features of DeepLabCut, since we want to analyze the whole video, and avoid to manually label a large number of frames, that may render the process infeasible. Moreover, DeepLabCut is optimized to detect poses captured by one or multiple cameras, a feature that is a great advantage for our project. In fact, with this tool we can label points in videos, and we can extend the pose estimation to a three-dimensional space in order to extract the rigid configuration of the body part. DeepLabCut is a deep convolutional network combining pre-trained ResNets and deconvolutional layers. In fact, it makes some changes on the ResNet model to adapt it to the problem: the classification layer at the output of the ResNet is replaced with deconvolutional layers. Further descriptions of deep learning and the adopted network architecture are here provided.

In the end, DeepLabCut allows us to avoid markers to track a specific region of interest in a mouse video. By giving some instructions, manually labeling a few frames, the neural network takes these information and the original video to learn how to detect the correct location of each labeled body part in *unknown frames*. As a result, we will have the original video and the coordinates positions of each selected point for every frame, without physically interfering with the mouse.

1.2.1 Deep Learning and ResNet

Convolutional Neural Network (ConvNet) are a type of Neural Networks which have been used in areas such as image recognition and classification.

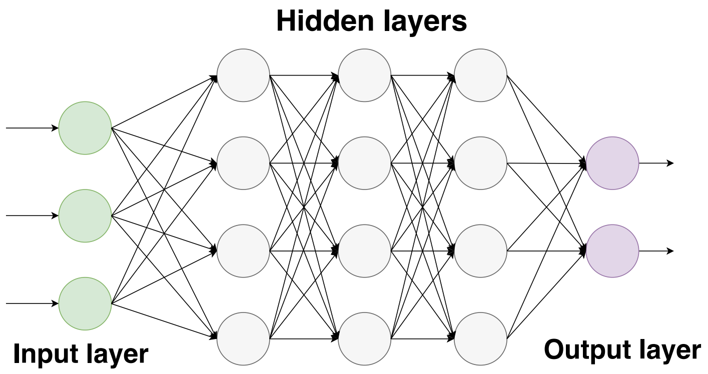


Figure 1.5: A regular 3-layer Neural Network: the input layer receives the input data, which are transformed through the network by the hidden layers, and the output layer represents the final class scores

The typical structure of a regular neural network is shown in Figure 1.5, and it consists of a collection of neurons that are organized into layers: input, hidden, and output layers. The input data is transformed by the neural network, through a certain number of hidden layers, in an output layer that represents the class score. The hidden layers do not share connections between them, but each neuron of the current layer is *fully connected* to all neurons of the previous one.

A Convolutional Neural Network has the same layers organization. The main difference consists in having images as inputs. Therefore, the layers of a ConvNet have neurons arranged in three dimensions: width, height, and depth. Furthermore, the neurons in a layer *are not* fully connected with the neurons of the previous one as in the traditional network, but they are connected only to a small region of the previous layer. The full input image is transformed through the layers into a single vector whose sizes are the dimensions of the final output layer that, even in this network, represents the class scores.

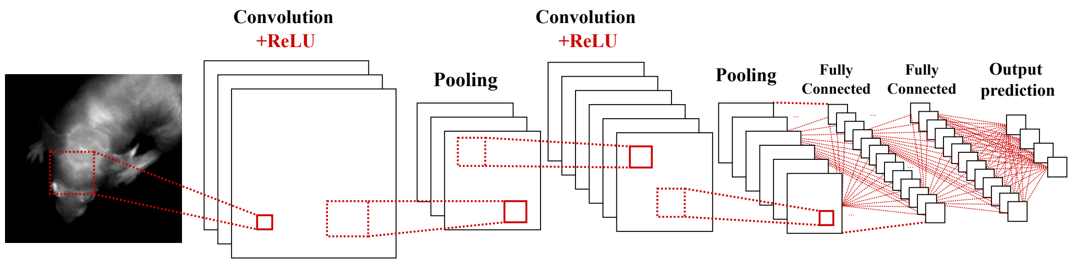


Figure 1.6: Simple Convolutional Neural Network: convolution, ReLU, pooling operations, and classification layer. For CNN the input is an image that is transformed through the layers to a single vector that represents the final class score. The neurons in a layer are connected only to a small region of the previous layer.

A ConvNet has three types of operations and a classification layer, as it is shown in Figure 1.6. The operations are:

- **Convolution**: it has the purpose of extracting features from the input image. We convolve a set of filters across the width and height of the input, producing an output image, called *feature map*. The convolution operation detects type of visual feature such as edges or patterns. The size of the feature map depends on three parameters: depth (number of filters), stride (number of pixels by which we slide out filter matrix), and zero-padding (pad the input matrix with zeros around the border).
- **Non Linearity (ReLU)**: it is used after every convolution operation. It stands for *Rectified Linear Unit*, and it is a non-linear operation. It replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity, since most of the real-world data would be non-linear.
- **Pooling**: it reduces the dimensionality of each feature map, retaining the most important information. It performs a downsampling operation along the spatial dimensions (width, height), reducing the size of the input. This operation is necessary in order to reduce the amount of parameters, make the input representations more manageable in the network, make the network invariant to small transformations, and to also control the overfitting.

- **Classification (Fully-Connected Layer)**: It is called fully-connected because every neuron in this layer are connected to all the neurons in the previous layer. The purpose of this layer is to use the features for classifying the input image into various classes.

While convolution and pooling layers act as block of *feature learning*, the fully-connected layer vests the role of *classification block*. Although, the network learns thanks to its learning parameters (weight w and bias b). Each connection between neurons has an associated weight. The learning process is called *Back Propagation of Errors*, and it is like learning from mistakes. Initially the weights are randomly assigned, and the output of each input is compared with the desired output, that is known. The error made by the network is propagated back to the previous layer, whose weights are adjusted accordingly to the calculated error, and the error is propagated until the first layer. This process is repeated until the output error is reduced. The weight updating is done by using an optimization method such as the *Gradient Descent*, and this is why to reduce the error, we often *compute the gradient*.

Most of the deep learning techniques are based on the use of pre-trained neural networks using particularly numerous sets of data already classified. Hardly ever people train an entire ConvNet from scratch with random initialization because it is rare to have a dataset with a sufficient size. Rather, it is common to pre-train a ConvNet on a large dataset and then use the trained network as initialization for the task of interest. This is what *transfer learning* means. Transfer learning allows to adapt a pre-trained model to a different problem and obtain good results despite having a small amount of data available. However, it is necessary to adapt the pre-trained model to the new classification problem. This could be done by following three different ways:

- Remove the last fully-connected layer
- Continue the backpropagation and repeat the training for some higher-level portion of the network
- Retrain the entire network

Mathis *at al.* in [3] have built DeepLabCut on transfer learning, since they use feature maps trained on extremely deep neural networks (ResNet), which were pre-trained on *ImageNet*, a rich dataset for object recognition. They decided to follow the first way of transfer learning that is to remove the last fully-connected layer and replace it with deconvolutional layers.

Residual Neural Network

Residual Neural Network (*ResNet*) is a pre-trained model developed by Kaiming He *et al* [6]. Traditional CNN models typically contain between sixteen to thirty layers. Deeper network are more difficult to training because of the degradation problems (e.g vanishing and exploding gradient). Since deep networks naturally learn more complex features as depth increases, raising the number of stacked layers is of crucial importance in the filed of object recognition from images. As they empirically shows, ResNet models are able to have up to 152 layers, and this is possible since it addresses the degradation problem introducing the *Residual Block*.

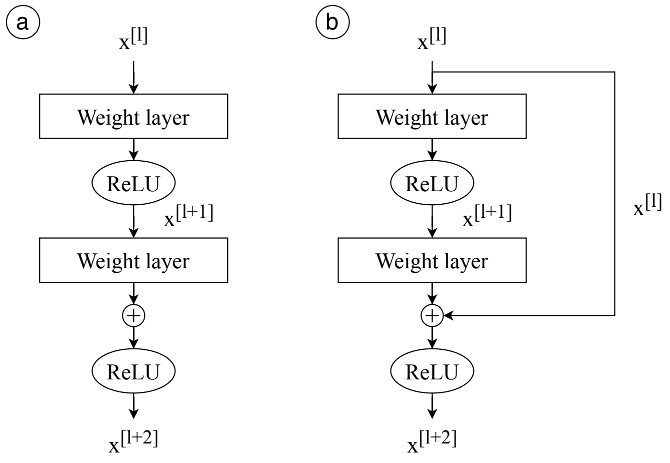


Figure 1.7: Residual learning: comparison between *plain network* (a) and *short-cut* (b) [6].

In the Figure 1.7 is shown a residual block, defined as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{\mathrm{W_i}\}) + \mathbf{x} \tag{1.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the input and the output (in Figure 1.7 $\mathbf{y}$ corresponds to $x^{[l+2]}$)

vectors of the layers, and the function $\mathcal{F}(\mathbf{x}, \{\mathrm{W_i}\})$ is the residual mapping. The solid line carrying the layer input $\mathbf{x}$ to the addition operator is called *shortcut connection* and it skips one or more layers, letting inputs to propagate faster across layers. In this case, the shortcut connection simply performs *identity* mapping, without adding neither extra parameter nor computational complexity to the *plain* network. The paper of [6] provides empirical evidence that it is easier to optimize the residual mapping than optimize the original one. In terms of computation, let's see what changes between a plain network and a residual one. Referring to Figure 1.7a, to obtain $x^{[l+1]}$, we apply a linear function and ReLU operation:

$$\mathrm{z^{[l+1]} = w^{[l+1]} \cdot x^{[l]} + b^{[l+1]}} \rightarrow \text{Linear} \tag{1.2}$$

$$\mathrm{x^{[l+1]}} = \mathcal{F}(\mathrm{z^{[l+1]}}) \rightarrow \text{ReLU} \tag{1.3}$$

such that z is the output from the linear function, w, the weights, x is the input to the function, and b the bias. In a plain network, to be able to yield $x^{[l+2]}$, we would have to go through the process of Equation 1.3 twice. With a residual block as in Figure 1.7b, the skip function reduces the number of times a linear function z is used to achieve the result:

$$\mathrm{x^{[l+2]}} = \mathcal{F}(\mathrm{z^{[l+1]} + x^{[l]}}) \tag{1.4}$$

The intuition behind the residual block is that if we do not have any additional information from a layer, and so $w_i = b_i = 0$ since the residual mapping is optimal $z^{[l+1]} = 0$, thanks to the shortcut connection we have :

$$\mathrm{x^{[l+2]}} = \mathcal{F}(\mathrm{x^{[l]}}) \tag{1.5}$$

which is much more easier for residual block to learn. The gain of adding these short connections between layers is that they *do not hurt the performances* of the plain network, since we are adding an identity map. Of course our goal is not only to not hurt the performances, but also to increase them. In fact, if these added layers learn something useful, they may learn even better of the identity function, and this may happen only if the plain network depth increases. We also know that skip connections help to eliminate vanishing and exploding gradients mentioned before. The *vanishing gradient problem* occurs when the gradient becomes vary small, and the updating weights will not affect the output as we desire. On the contrary, with the *exploding gradient problem*, the gradient becomes exponentially big, and the algorithm cannot train the model. Thanks to the short connections, the number of epochs taken as the algorithm goes deeper are reduced, and the model takes longer to vanish or explode. Therefore, we can resume the main advantages of using skip connections as:

- The model learns an identity function which ensures that the performance cannot get worse, but the following layer will perform at least as good as the previous layer
- They delay the problem of vanishing and exploding gradient.

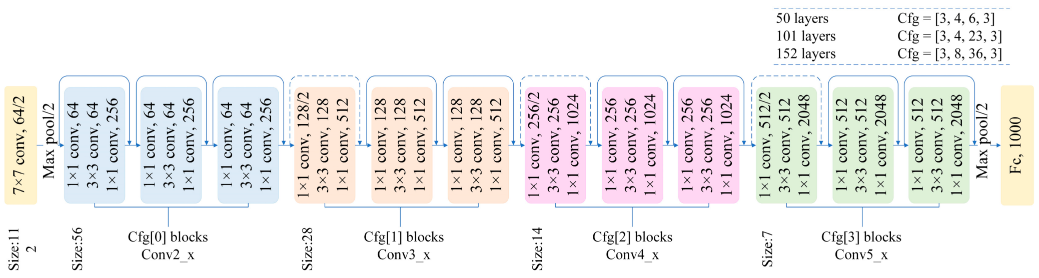


Figure 1.8: ResNet baseline architecture [7]. The Cfg[i] blocks differ from each other for the size of the input and for the parameters of the used feature maps. they are 4 convolutional layers containing 3 residual blocks each; depending on how many layers we want to add each Cfg can be repeated multiple times (e.g, for 50 layers: $\text{Cfg}[0] \times 3, \text{Cfg}[1] \times 4, \text{Cfg}[2] \times 6, \text{Cfg}[3] \times 3$)

In Figure 1.8 is shown the baseline architecture of the ResNet models. The dotted line shortcuts represent the dimensions increase from a residual block to the other. The ResNet model consists of five stages: 1 convolutional layer, 4 convolutional layers which contain 3 residual blocks each, and a Fully Connected layer that reduces its input to the number of classes (not in our interest). The notation for each block

$$n \times n \; conv, N$$

is referred to the size of the filter ($n \times n$), and how many filters are used, that is how many feature maps are extracted (N). In the ResNet-50, the four convolutional-residual blocks are repeated respectively 3, 4, 6, and 3 times to obtain 50 layers, and it has over 23 million trainable parameters.

Deconvolutional layers

The Fully Connected layer at the output of the ResNet model is replaced with *deconvolutional layers* in DeepLabCut. Deconvolution is the opposite operation of

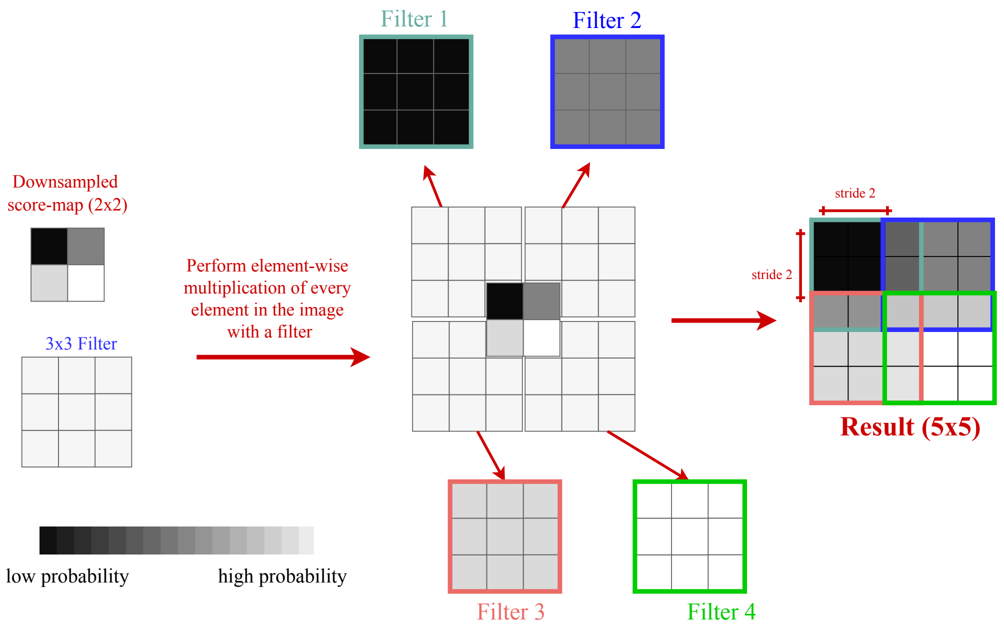

Filter 1
Filter 2
Downsampled score-map (2x2)
Perform element-wise multiplication of every element in the image with a filter
3x3 Filter
stride 2
stride 2
Result (5x5)
low probability
high probability
Filter 3
Filter 4

Figure 1.9: Example of how deconvolution works with a 3x3 kernel on input 2x2, stride 2 to obtain the original size 5x5. The white pixels represent high probability that a body part is there located, and dark pixels low probability. Notice that from the down-sampled to the up-sampled image, there is consistency in probability body part location

convolution. This is necessary since the output we are interested in is the spatial probability densities of each pixel of our input image. In fact, deconvolutional layers are used to up-sample the visual information, and assign to each body part its probability densities, which represent the probability that a body part is placed in a particular location of the image. Since during the training the input size of the image has been down-sampled by the pooling operation, we do want now up-sample it to obtain the original size of the image. During the training the weights are updated iteratively in order to assign for a given frame high probabilities (≈ 1) to region of the original image in which the body part is located, and low probabilities (≈ 0) elsewhere. The output layer of the network is a *down-sampled score-maps*, that is shown in Figure 1.9, and so a down-sampled image that contains brighter pixels if the right location is in nearby. The deconvolutional layers produce a scalar field of activation values with different size with respect to the input score-map, but that contains an expanded probability density corresponding to regions in original image [3], for each body part (see Figure 1.10).

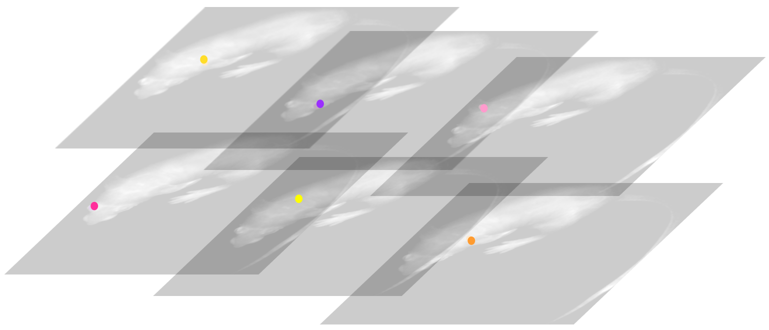


Figure 1.10: Final score-maps assigned for *each* body part on the original image. The outcome of the deconvolutional layers is exactly the input image (with its original size), and a score map associated to each label that suggests where the label is located accordingly to the prediction.

1.3 3D reconstruction

Aiming to have a quantitative representation of the mouse brain vascular network, we investigate how we can extend the 2D tracking to a 3D geometrical visualisation of the region of interest. In computer graphics, controlling facial animation is a well-known challenge. An interesting application of human pose estimation is for a real-time facial capture method [8]. By tracking the variations of human face pose, the rendered graphic naturally fits the facial expressions as they move and, furthermore, modern techniques can deliver high-resolution facial geometry with a reliable motion information (see Figure 1.11). Is it possible to render a mouse fluorescent surface as well?

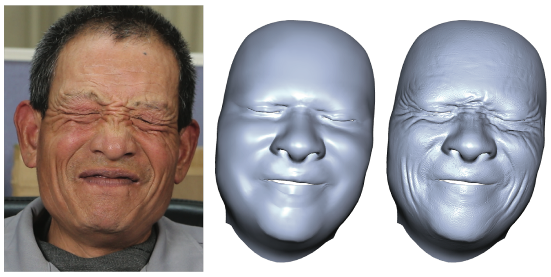


Figure 1.11: Real-time facial tracking system that captures performances at high fidelity. From a monocular input (left), the global shape as well as local details are captured (center and right). Figure adapted from [8]

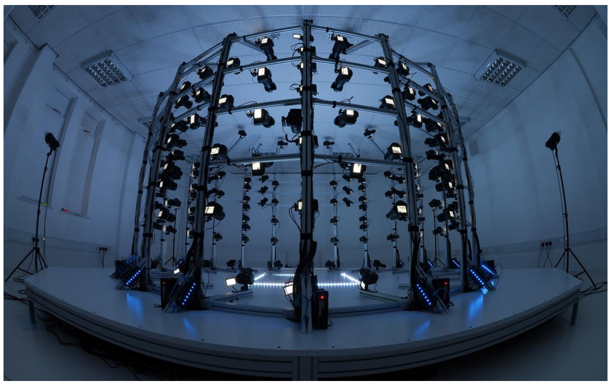


Figure 1.12: 3D scanner from [9]. This is a cameras structure organized in such way to take in a single shot the higher number of details of the subject place in the middle. The outcome is a 3D body scanning.

The most popular online search concerning 3D reconstruction technology nowadays is inevitably related with photogrammetry. Thanks to new image acquisition technologies, the classic 3D model of an object or a body, represented as a skeleton made of arcs and nodes, begins to disappear, thus giving ample space to a more real and all-round construction of the reality. 3D scanner like the one in Figure 1.12 are capable to capture the shape of an object in a single shot. It collects data about a subject that can be an object, an environment or a person (procedure commonly known as *3D body scanning*) and acquires simultaneously shape and color data. 3D scans start to be adopted in 3D printing for numerous fields and applications such as video games, special effects, animation movies as well as automotive, aeronautic, dental. Photogrammetry is a 3D scan that reconstructs a subject from photographs with computational geometry and computer vision algorithms. Reconstructing a rigid object from photographs means to extract three-dimensional information from two-dimensions. The main actor that can help us to perform it is the camera calibration. Essentially, it is necessary to know the properties of a camera to understand the geometrical properties of an object in a picture. An image is not more than a projection of the object in a plane. By converting the object to an image, we lose depth information, and for instance we may need to use further methods to define if a pixel belongs to the object or to the background. *Camera calibration* is the main actor of 3D reconstruction, both if we are talking about photogrammetry and other 3D reconstruction techniques, since it allows us to relate pictures coming from different cameras or views.

1.3.1 Camera calibration

Camera calibration is the first step to convert a 2D image into a 3D model. It consists of estimating the parameters of a *pinhole camera model* which is used to describe the image formation in a camera. In the next equations, the homogeneous coordinates notation is adopted. They are used to solve the difference in appearance of two plane objects viewed from different points of view. In our case, homography is used to refer to the same plane the object and the camera, or the object views from two cameras. In Figure 1.13 is shown how the homogeneous coordinates are adopted to normalize a polynomial curve. In our case, it is like $\mathbf{P}_1$ and $\mathbf{P}_3$ are the cameras, and the blue curve is the object. With the homogeneous notation $\tilde{\mathbf{p}} = [\mathrm{u}, \mathrm{v}, 1]^{\mathrm{T}}$ or $\tilde{\mathbf{P}} = [\mathrm{X}, \mathrm{Y}, \mathrm{Z}, 1]^{\mathrm{T}}$, we are referring all the 2D or 3D elements to the same plane $\mathbf{W} = 1$.

The pinhole camera model is shown in Figure 1.14. Considering a scene view formed by the projections of 3D points into the image plane, the relation between the world coordinates of a 3D point $\tilde{\mathbf{P}} = [X, Y, Z, 1]^T$ and its image plane coordinate

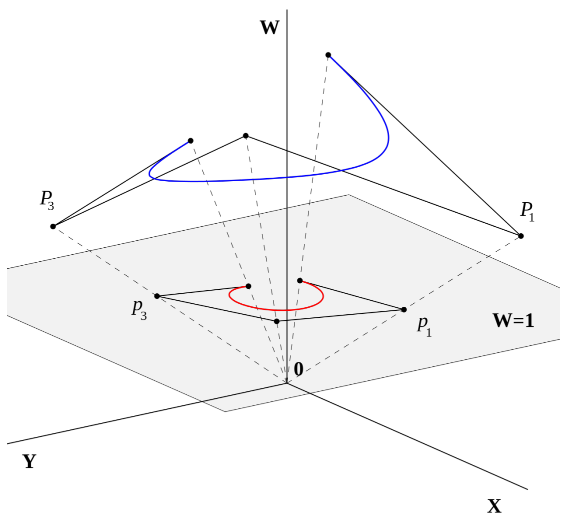


Figure 1.13: Polynomial curve defined in homogeneous coordinates (blue) and its projection on plane, rational curve (red). *By Wojciech Muła - Own work (Python script, final touches Inkscape), CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1196334*

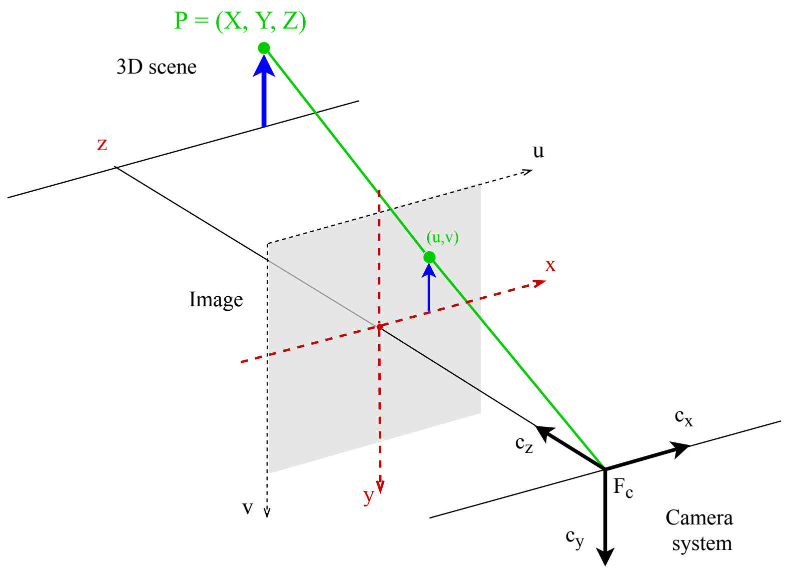


Figure 1.14: Illustration of the pinhole camera model: the point P in 3D is projected into a plane with (u, v) coordinates, losing information about its third dimension z

$\tilde{\mathbf{u}} = [u, v, 1]^T$ is:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1.6}$$

$$\tilde{\mathbf{u}} = \tilde{\mathbf{p}}_{img} = \mathbf{K}\ [\mathbf{R}|\mathbf{t}]\ \tilde{\mathbf{P}} = \mathbf{K}\ \tilde{\mathbf{p}}_{cam} \tag{1.7}$$

$\mathbf{K}$ is the intrinsic matrix, and $[\mathbf{R}|\mathbf{t}]$, rotation matrix and translation vector, respectively, is the extrinsic matrix. Note that intrinsic parameters refer to the optical, geometric and digital properties of the camera, and the matrix $\mathbf{K}$ does not depend on the scene viewed. The extrinsic parameters describe the transformation of the 3D point with respect to the camera view, so they are used to translate coordinates of a point $[X, Y, Z, 1]^T$ to a fixed coordinate system with respect to the camera, obtaining new $\tilde{\mathbf{p}}_{cam} = [x, y, z, 1]^T$ coordinates. Through this model, we can estimate the relation between the image plane and the real scene, since both of them are dependent from the camera view. If we know the intrinsic parameters of the camera, and how the image or the 3D scene is related to the camera, it is possible to define each elements of the Equation 1.7. The coordinate system of the camera is a decisive concept, since we can define two types of coordinate systems: an *absolute* coordinate system, and a *relative* one. If we think of a rigid object, we can always define how each of its points are related to each other based on a *world coordinates* system, which means that those relations between points do not depend on the viewer, since they are *absolute*. If we now take multiple viewers that look at the object from different perspectives, each of them will see the object differently. This is a *relative* coordinate system, and this is exactly what a pinhole camera does: converting the world coordinates of the rigid object to a camera coordinates system. Another important point is what happen if we have multiple cameras. As it is shown in Figure 1.15 on the left, if we have two cameras that look at the same scene, we can always relate a camera to the other, since the second camera can be seen as an object that is translated and rotated with respect to the first camera. Consequently, the same concept can be applied if we have the same camera that moves around the scene (see Figure 1.15 right). We have a initial time t_0, in which the camera had a position p_0, and at different (t, p) the new view can always be related with the original one, since it can be considered as a translated and rotated view with respect to (t_0, p_0). These relation between views will be adopted and described with examples directly in the experiments.

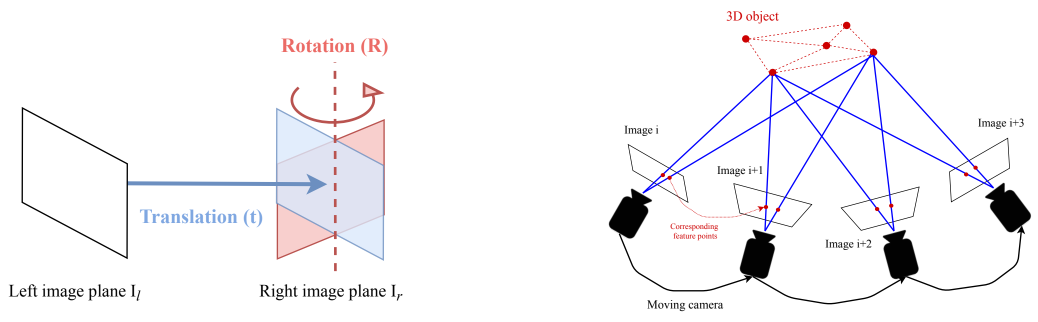


Figure 1.15: Illustration of the pinhole camera model: second camera seen as rotation and translation of a first camera (left), multiple views of the same camera with respect to the first reference view (right)

Chapter 2

Photogrammetry in static and non-static objects

Photogrammetry methods are typically designed for reconstructing imaged surfaces in static scenes. If this technique were applied to photographs or videos of an animal with a fluorescent tracer, it would be possible to have a 3D object with a surface that contains information of phenomena that takes place beyond the skin of the animal. Moreover, the acquisition would be less expensive in comparison to the most sophisticated tomography techniques, because with only high resolution videos it would be possible to follow, for example, vessels through the skin. Based on these intentions, the first pursued approach to the problem is to understand how these methods work and if it is possible to use their computational geometry for the aforementioned purposes. In order to have reliable information, camera and object should respect some constraints: the object cannot move during the acquisition.

In this section, the limits for non-static objects and data constraints of photogrammetry are investigated, and the overall photogrammetry pipeline is described. After, the reconstructing methods are tested to obtain 3D objects of three different datasets:

1. Photographs of an horse hoof

2. Video of a rotating belly of an anesthetized mouse

3. Moving mouse injected with a fluorescent contrast agent

The final models are obtained using a photogrammetry software called Meshroom [10] which is an excellent mean to see how the pipeline is followed in a practical fashion.

2.1 Photogrammetry pipeline

Starting from a set of photographs of an object, photogrammetry adopts images algorithms to extract the most significant elements from pictures. These elements are called *descriptors* or *features*, and allow the algorithm to find out what images have in common. Through these extracted features, photogrammetry is able to correlate images between them, and eventually estimate where the camera was placed with respect to the object during the acquisition. Once the camera positions are defined, the object points are matched between the views, and then triangulated. After, the cloud of points are connected to each other to generate what is called a *mesh*. Finally, the meshed body is "colored" with a *texture* coming from the pixels properties of the original images. In order to better understand how we obtain a textured 3D object from set of images, we follow the pipeline in Figure 2.1.

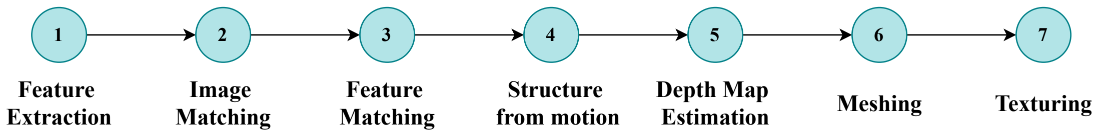


Figure 2.1: A typical photogrammetry pipeline for extracting mesh and texture information from a set of still images [10]

The first step is the **Feature Extraction**. The goal is to extract groups of pixels that are invariant to changing camera viewpoints, or in other words, extract patches that can be detected in different images irrespective of rotation, translation and scale. In Figure 2.2, the extracted patches from the images are listed in the middle of the figure. Otherwise, the yellow squares in the images on the sides point out where those patches have been detected even if they have been rotated, translated, or have been taken away from the camera. Hence, those patches contain some properties, called *feature descriptors*, that are always detectable in all images.

The SIFT (Scale-invariant feature transform) algorithm is the most used features

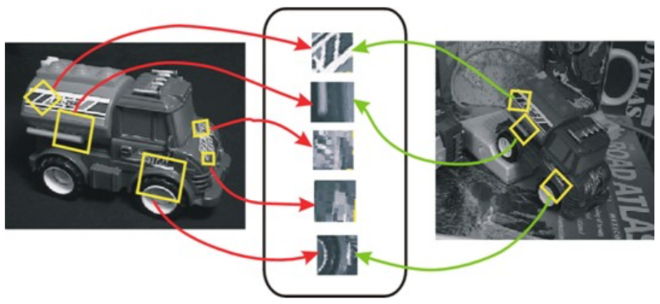


Figure 2.2: Scale-Invariant feature transform [11]: SIFT algorithm detects significant patches of images, and it is able to detect those patches in any images irrespective to the image transformations (translation, rotation, or scaling)

detection method, and the best algorithm to use to deal with image transformations during the acquisition. It is called "scale-invariant" since it is based on the assumption that "a relevant detail only exists at a certain scale [10]".

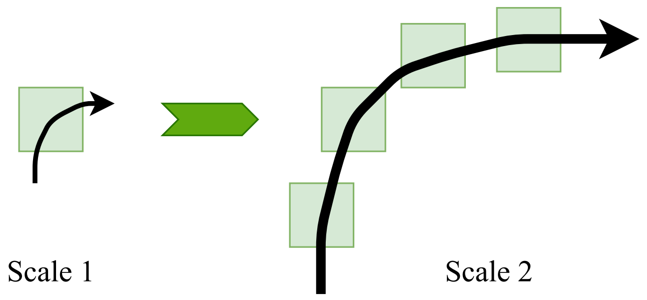


Figure 2.3: Example of scale-dependent detector. At Scale-1, the window is well detecting the arrow corner, but at Scale-2 the window is too small

If we think about a curved object at different scales as in Figure 2.3, and we want to detect its corners, they are not easily detectable with the same window detector in a larger scale, because we would need larger windows. Therefore, the method in Figure 2.3 is scale-dependent. Although, what are the main features that allow us to recognize the object regardless its scale? To make this analysis, SIFT computes a pyramid downscaled images. By downsampling and blurring the image, SIFT emulates this feature changing depending on the scale. SIFT generates what is called a *scale space*, and it is a Laplacian of Gaussian function (*LoG*), obtained with the convolution of a Gaussian kernel G with the input image I as

$$LoG(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{2.1}$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.2}$$

where σ is the standard deviation of the Gaussian distribution. LoG is a detector of blob in different sizes thanks to the scaling parameter σ. Consequently, higher value of σ gives a blob detector for larger corners, while lower value of σ gives a blob detector for smaller corners. However, LoG is a costly function, for this reason making the difference between two scales is more convenient. Therefore, the Differences of Gaussians images (DoG) are computed. The result is a scale space as in Figure 2.4.

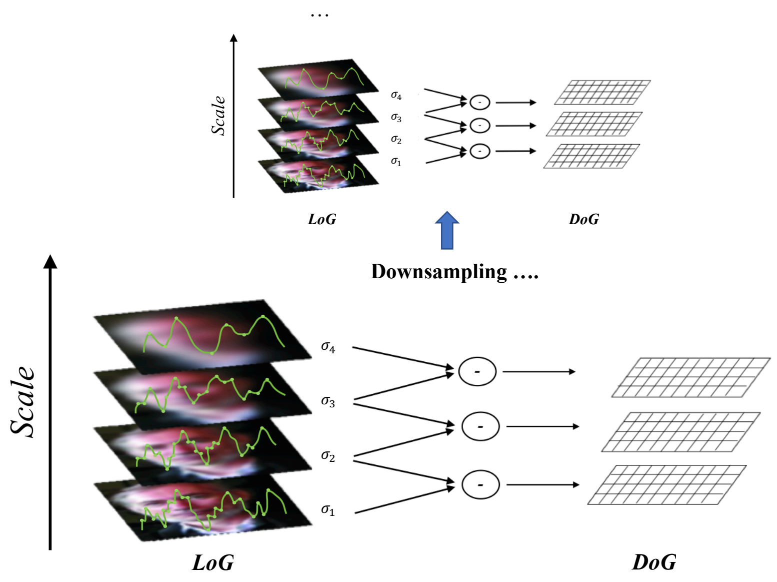


Figure 2.4: Pyramid downscaled images. The image is blurred with Gaussian filter (LoG), and downsampled to emulate different scaling factor. By computing the Difference of Gaussian (DoG), the research of local maxima is simplified.

Once the DoGs are created, we look for the local maxima across the scale to find a list of (x, y, σ) values, which means to find the potential feature descriptors at (x, y), and at σ scale. For example, a pixel of an image at a certain scale is compared between the eight neighbour pixel in the same scale, and with the nine of the previous and the next scale as in Figure 2.5. These found maxima correspond to the location of our points of interest, which are invariant to the scale.

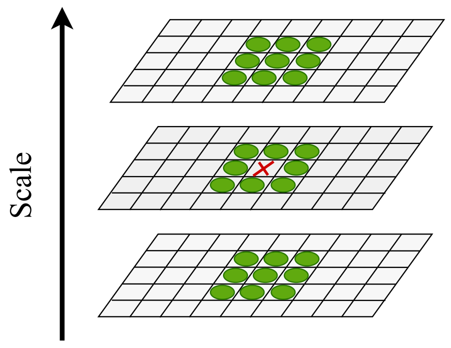
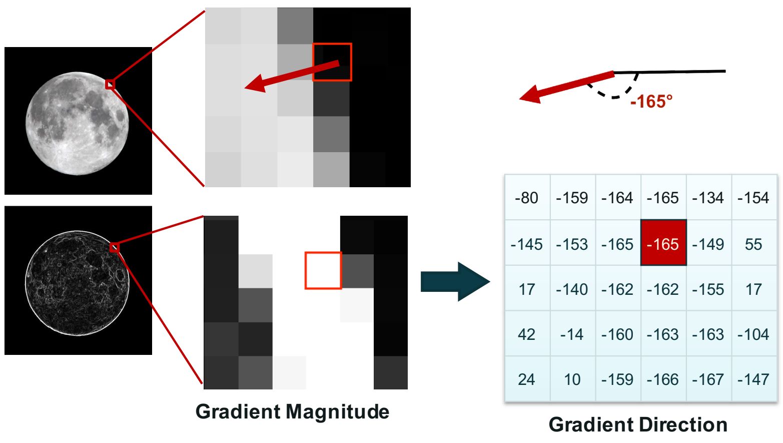


Figure 2.5: Research of local maxima between DoG in different scales. The actual pixel is compared with a neighbourhood of its scale, and other two of the nearby scales

In order to achieve invariance also to image rotation, an orientation is assigned to each point. Therefore, SIFT considers again a neighborhood around each point location, and it computes the gradient magnitude and direction in that region as in Figure 2.6. The goal is to find in which direction, the selected feature point has its greater change. In fact, the gradient is a derivative filter which is commonly used to detect object edges relatively to the background. Then, it creates an orientation histogram with 36 bins (360 covered degrees spaced 10 degrees apart), which are weighted based on the gradient magnitude. The gradient magnitude is a value between 0 and 1 associated to each pixel. Brighter is the magnitude (≈ 1), stronger is the pixel intensity variation, while in homogeneous region the magnitude is smaller (≈ 0). The gradient direction associates to each pixel an angle value in degrees (between (0,180) and $(0, -180)$ degrees), and it represents the strongest variation for the actual pixel. The combination between magnitude and direction of a gradient is an arrow with the gradient direction and its length proportional to the magnitude. If we look at Figure 2.7, the histogram represents the weighted frequencies of the directions found in Figure 2.6 (after having converted the angles values in (0,359) degrees). The highest bin represents the number of pixels whose greater direction is between 190 and 199 degrees, it means that for the investigated neighbour, the greater variation is between that range of angles. SIFT considers



-80	-159	-164	-165	-134	-154
-145	-153	-165	-165	-149	55
17	-140	-162	-162	-155	17
42	-14	-160	-163	-163	-104
24	10	-159	-166	-167	-147

Figure 2.6: Magnitude and direction of gradient. The above moon [12] is the original image, while the moon below is its gradient. The homogeneous parts of the image are dark, while the regions with high pixel values changes are brighter (see the boarder between the background an the object). The gradient magnitude represents how big is this change, while the gradient direction is in which direction is the highest pixel variation. To the selected pixel in the original image, we can associate an arrow directed to the highest variation, whose length is proportional to the magnitude of this change.

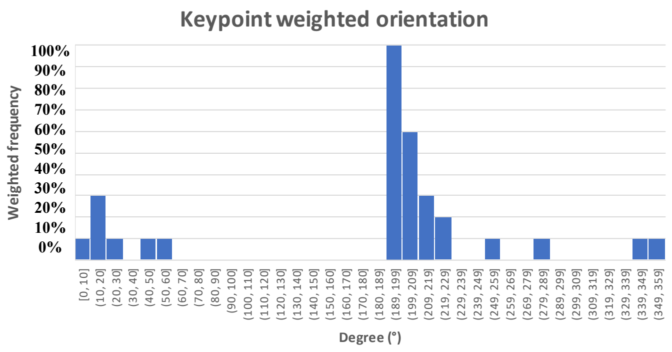


Figure 2.7: Orientation histogram referred to magnitude and direction gradient of Figure 2.6: each bin represents how many pixels have their direction gradient oriented in a certain range of degrees. Moreover, the frequency is weighted based on the gradient magnitude.

each peak between 80% and 100% to calculate the orientation by creating keypoints with the same location, but different directions.

At this point, SIFT has found the location, scale, and orientation of the keypoint.

Now, it associates for each keypoint a *descriptor* in order to have a vector that describes each of them, and to have a mean to make a comparison between them. As in Figure 2.8, SIFT takes a 16x16 window around the keypoint, divided into 16 sub-blocks of 4x4 pixels. The region size is determined by the keypoint scale and the orientation is determined by the dominant axis. Then, for each sub-block, 8 bin orientation histogram is created. Typically, the descriptor consists of 128 bin values (8 bin $\times$ 16 sub-blocks).

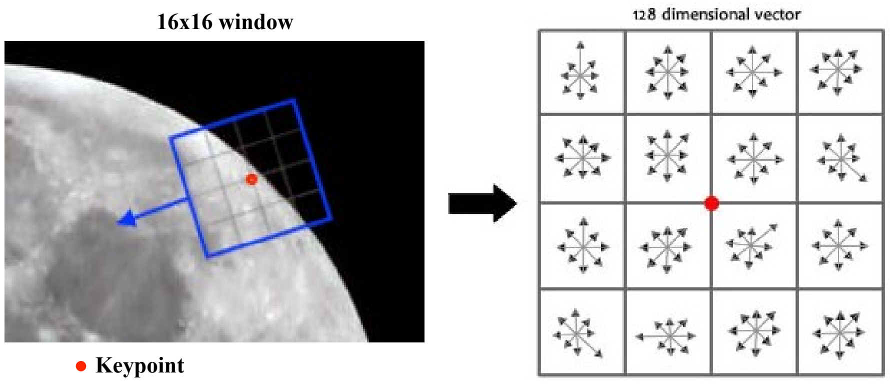


Figure 2.8: 16x16 window oriented with respect to the keypoint orientation (left), 8 bin orientations for each 4x4 sub-block (descriptor vector of 128 elements) [13]

Once the descriptors are extracted for every keypoint of each image, we can go further to the next steps. The **Image Matching** and **Feature matching** have the goal to find images that are looking to the same areas of the object (or the scene). Ideally, we should compare every keypoint of each image between them, but in order to save computational time, Meshroom uses the image retrieval techniques, in order to compare only images that really have something in common. Once pairs of images are matched, it starts to compare all features and look for similar descriptors (see Figure 2.9). For each feature in the first image, we obtain a list of candidate features in a second one. However, not all features are correctly matched. To remove bad candidates, we look for the second closest descriptors. Once we have found the correspondences, we compute the camera model of section 1.3.1 to understand the geometric relationship between the views. This step is the **Structure from Motion (SfM)**. In order to estimate the rigid object structure, based on the positions and the orientations of the camera, it computes an initial two views reconstruction that is iteratively extended by adding new views, as it is shown in Figure 2.10. Each matched point is supposed to represent a point in

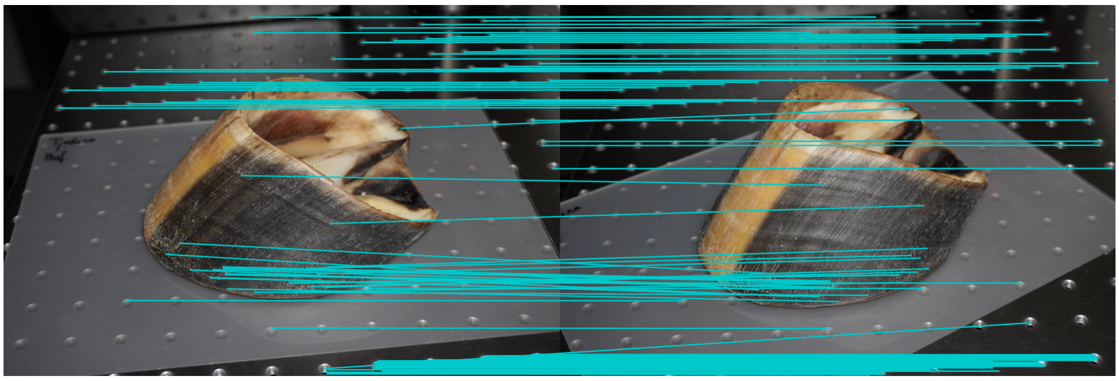


Figure 2.9: Example of pair of images with matched descriptors. The horse hoof capsule was provided by Tjadina Klein, and imaged by Tjadina Klein and Thomas Bischof

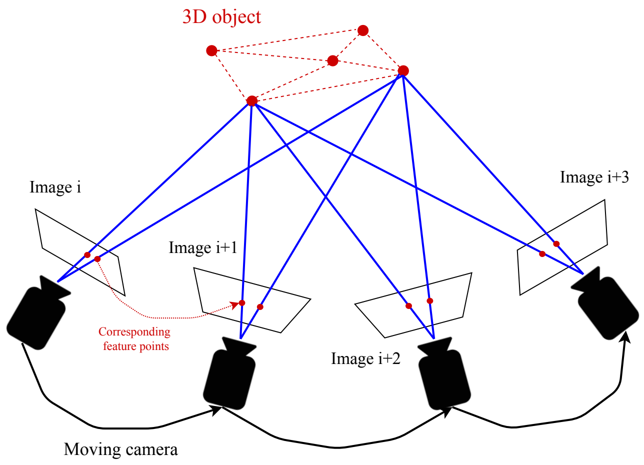


Figure 2.10: Iterative triangulation. From an initial two views reconstruction, the estimation of the 3D position is iteratively corrected by the addition of multiple views

space, visible from multiple views. During this iterative process, the incoherent matched are removed, and the corresponding 2D points are triangulated into 3D points. For the last steps, we are not going deepen into them, since they are not useful yet for the purpose of this thesis project. However, they are here introduced, since they will be subject of future research. The **Depth Map Estimation** has the objective of retrieving the depth value of each pixel for those cameras that have been resolved by SfM, and many approaches are available. The one implemented by AliceVision [10] is based on finding the best matches between the intersection of the optical axis with the pixels of neighbour cameras. This generates a list of many depth candidates, and so the similarity for all of them is estimated. After the application of a series of filters to remove the noise, they look for a compromise to remove bad candidates. The **Meshing** step fuses all the depth maps in order to create a dense geometric surface representation of the object. The result is a mesh, that consists in a cloud of vertices connected to form a tetrahedron structure. For each triangle, the **Texturing** uses the visible information associated with each vertex to retrieve the texture. In Figure 2.11 a visualization of these last three steps is shown.

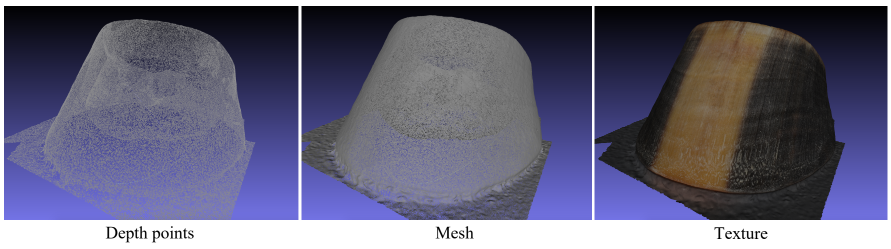


Figure 2.11: Example of the depth map, mesh, and texture of an horse hoof. The horse hoof capsule was provided by Tjadina Klein, and imaged by Tjadina Klein and Thomas Bischof

The described pipeline is the general flow followed by Meshroom in order to reconstruct an object from a set of images. In the next paragraphs, we are going to see how the pipeline works with three different sets of data, underlying the differences between their images, and analyzing the final 3D object.

2.2 3D reconstruction of an horse hoof, mouse belly, and a mouse in motion

Each tested dataset in this section is chosen based on the acquisition mode. Photogrammetry is generally designed to reconstruct static-objects or scenes. In fact, the user typically takes photographs of the subject from different angles and collects them to grasp the greatest number of details, since some parts of the object may be hidden depending on the view. Thus, this is a common set-up in which the camera is in motion and the object is totally held still. This kind of acquisition is inevitable if the object to reconstruct is the interior of an apartment or the wall of a building, but if the subject is small and freely manipulable in space and the user can easily move, rotate, and control it, is it possible to invert the camera-object roles? As this is just a matter of relative viewpoints, it is legitimate to say that if you can rotate the subject at will, then the camera can be held still while it is taking pictures of the rotating object. This is what will be tested in this part, exploiting two sets of data that differ precisely in this acquisition mode.

2.2.1 Still horse hoof and camera in motion

This first set of data consists of 38 photographs of an horse hoof taken with SONY ILCE-6400 in RGB color space, and size of 4240x2832 pixels. The horse hoof is held still in the middle of the scene while the camera takes pictures in order to cover the points of view at 360 degrees. The resolution of the camera allows us to obtain images with an high level of details, strong suit of this dataset.

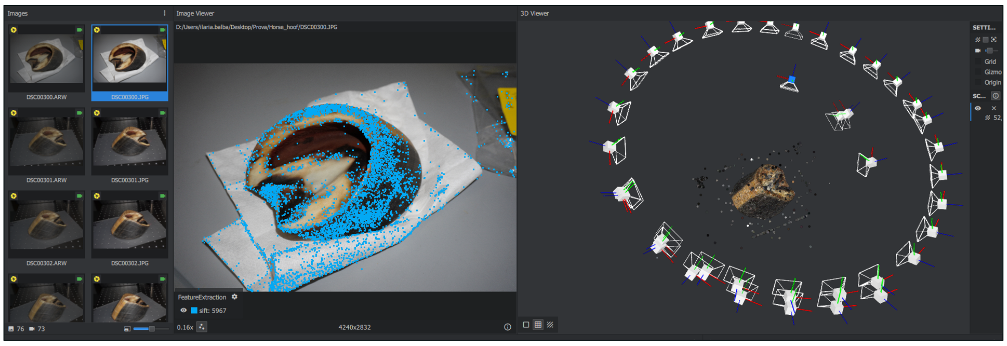


Figure 2.12: Horse hoof: cameras position estimation and depth map. The horse hoof capsule was provided by Tjadina Klein, and imaged by Tjadina Klein and Thomas Bischof

In Figure 2.12, a typical visualization of the Meshroom GUI is shown. From the left to the right, we have the set of images, a particular frame with the sift points detected, and the estimated camera positions. As the circle of cameras suggests, the camera positions are well estimated, and in the middle the depth map of the horse hoof is already visible. This kind of acquisition (rigid object and camera in motion), is perfectly consistent with the use of the photogrammetry pipeline. The object is non-deformable, the pictures have high resolution, and the amount of data are enough to make an efficient reconstruction.

2.2.2 Still camera moving around a mouse belly

The second dataset is a good example of acquiring images of a rotating object while the camera is held still. The mouse is restrained to ensure minimal movement during acquisition. It rotates 360 degrees while being pointed at by a near-infrared camera that captures laser signals (see Figure 2.13). The output is a video that shows a rotation imaging of a tumor of a mouse subjected to an intravenously injection of PbS-aCD8 and ErNPs-aPDL1. This dataset is interesting because we

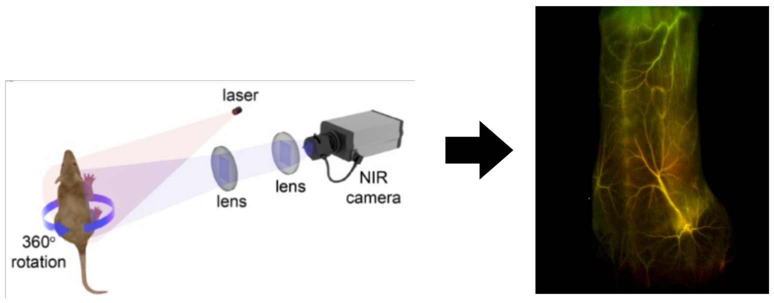


Figure 2.13: Acquisition method: rotating mouse (left), rotation imaging of a tumor (right). Figure adapted from [14]

can assert that we do not need the camera need in motion, but even the object can show different sides of itself. As it is shown in Figure 2.14, it is read as the scenario is reversed: camera in motion and still object. Moreover, thanks to the texture, we can now really understand what we are looking for. This 3D reconstruction of the injected mouse allows us to have a 3D visualization of the vascular network on the surface of the object. However, starting from this 3D object, we may focus the research on tracking the vessels on the surface and estimate what is their depth through the body. Nevertheless, the animal is restrained during the acquisition, not irrelevant for the purpose of our experiment. For that, we proceed with the next dataset, in which the animal is free of anesthesia and restraints.

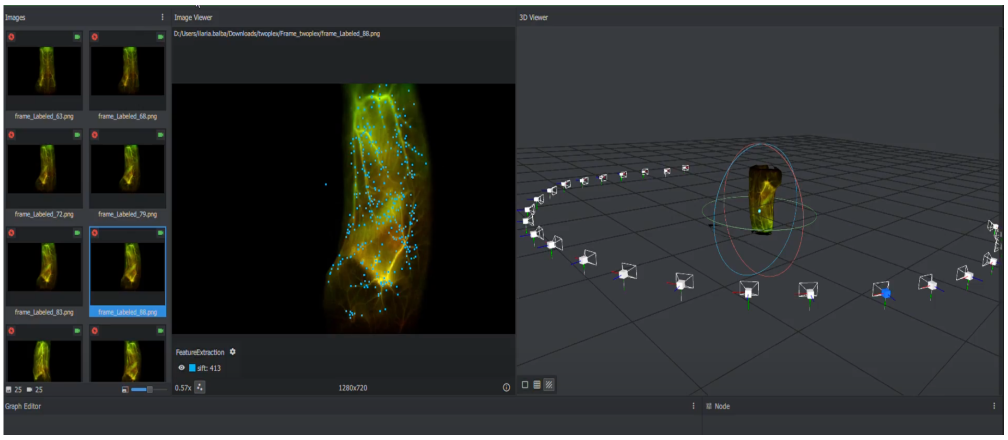


Figure 2.14: Camera positions around the object, as the object is still and the camera in motion. From the left to the right: set of images, detection of sift points, reconstructed object with estimation of camera positions

2.2.3 Still camera and mouse in motion

The last dataset is acquired with a still camera that records a mouse freely to move. The mouse was injected with a fluorescent contrast agent through its vessels, and more details about the experiment will be presented in section 3.1. In Figure 2.15, once again, even if the object is moving and the camera held still, the camera positions are estimated. The photogrammetry algorithm has detected a sort of change in motion of the mouse, since the camera positions follow an irregular shape through the 3D space. It is correctly predicted that the camera could only have been above the mouse, and never from below since the mouse is always facing the floor. The mouse is moving inside the imaging chamber, and it goes around often changing its position through the scene. What Meshroom does is to select only those frames which are coherent between them, and in this case frames are consistent if they show a similar mouse pose. This is why every frame in which the mouse strongly changes its pose with respect to the one in Figure 2.15 (head on top, and tail below) are discarded.

This discard has a series of effects on the reconstruction. First, the animal is not correctly reconstructed, since photogrammetry reconstructs multiple pieces (see the two tails in Figure 2.16). Moreover, the background fuses with the object, probably because from some views the dark parts of the mouse body are confused

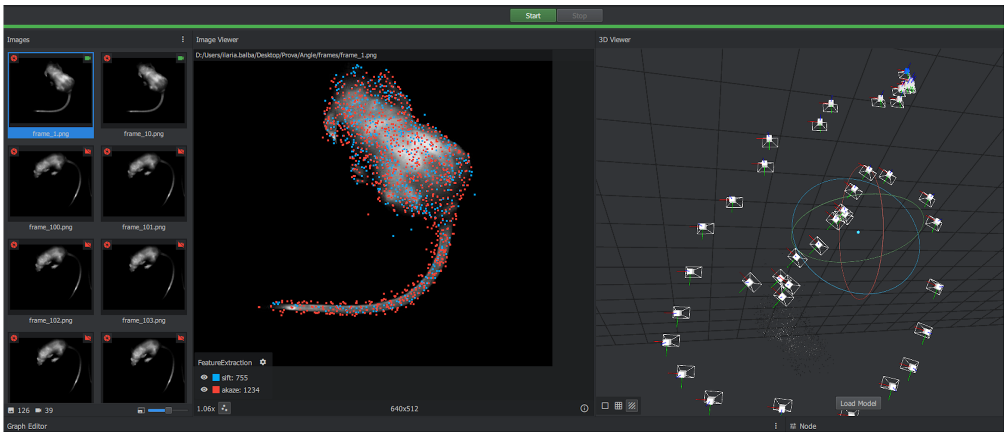


Figure 2.15: Camera positions around the object, as the object is still and the camera in motion. From the left to the right: set of images, detection of sift points, reconstructed object with estimation of camera positions

with the background and vice versa. However, this reconstruction suggests that is possible to obtain a texture of the fluorescent skin of the mouse. What we want to investigate at this point is if the errors may be corrected if we give to the algorithm further information of the orientations that the mouse has over time.

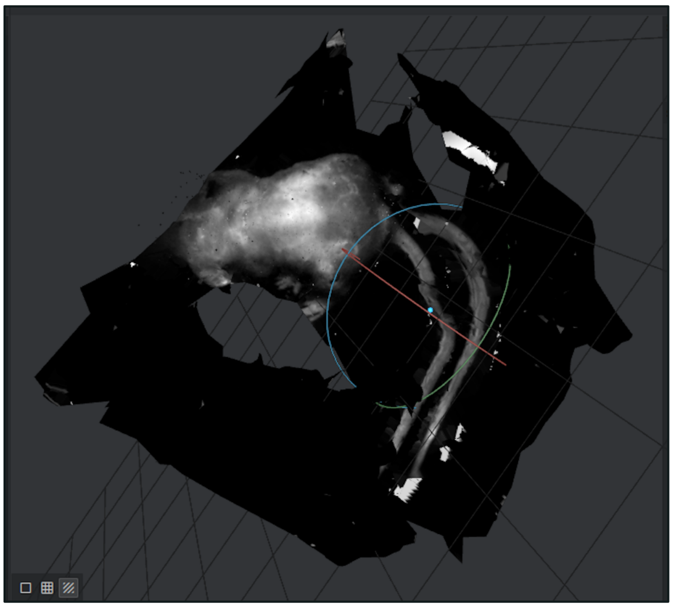


Figure 2.16: Reconstructed mouse. Visible errors of reconstruction: double tail and background fused with the mouse

2.3 Results and discussion

Photogrammetry is commonly used to extract three-dimensional measurements from two-dimensional data (i.e images and videos), which allow the generation of a 3D digital model of the object or scene. We have found out that static scenes and non-deformable objects are adequately reconstructed. Of great interest was the reconstruction of an animal body injected with fluorescent molecules, which allows us to have a 3D visualization through the skin of a mouse organs and vessels. With this 3D model, we may follow the vessels on the surface and extract a preliminary geometrical configuration of what the vascular network looks like, but this dataset consists of a rotating restrained mouse. If we want to leave the animal to freely move, the results change. In fact, we were able to extract an interesting texture of the moving mouse, since it was injected with a fluorescent contrast agent. Our investigation moves forward to find how the motion effects may be solved. Since the reconstruction had parts multiply reconstructed, and the animal were fused to the background, is it possible to define the orientations that the object is taken over time and "suggest" how handle the reconstruction?

Chapter 3

3D pose estimation of a mouse skull from a monocular video

In Chapter 2, we investigated the use of photogrammetry algorithms to reconstruct different objects. We have found that Meshroom is an easy and excellent tool for reconstructing non-deformable objects. The result changes when the object is in motion. With photogrammetry, we are able to extract an interesting texture of the moving mouse. However, the reconstruction has parts multiply reconstructed, and the animal is fused to the background. Our investigation moves forward to find how the motion effects may be solved. If it would be possible to know in which direction the mouse is looking in, would it be possible to detect which part of the body the object is showing to the camera? If so, we may use external measurements of orientation to improve reconstruction?

Here, we turn our attention to *pose estimation* as a method for motion compensation. We propose a first approach to extract the skull orientation of a mouse with a vascular contrast agent through its vessels. We show how *deep learning* technology works on DeepLabCut, a tool extremely useful to estimate the pose of a moving object over time. We investigate the mouse skull as *rigid* object in motion. Based on the assumption that the 2D tracked points are the projections of a rigid 3D model, we use the pinhole camera model described in section 1.3.1 to estimate the skull orientations over time.

3.1 SWIR imaging of a mouse with vascular contrast agent

SWIR technology is a non-invasive fluorescence imaging technique that allows us to visualize biological processes taking place in living organisms. The fluorescence mechanism consists of a molecule, called dye, that absorbs light. After, the dye releases energy, commonly defined as the emitted fluorescent light, it can be detected at a certain wavelength $\lambda_{emission}$. In order to detect light production, the measuring device has to know what wavelength to be set at. This wavelength is inside the SWIR window of the electromagnetic spectrum, typically between 1000 and 2000 nm, and the primary sensors used to detect the signal are the indium gallium arsenide (InGaAs) sensors, since they cover the SWIR range (see Figure 3.1).

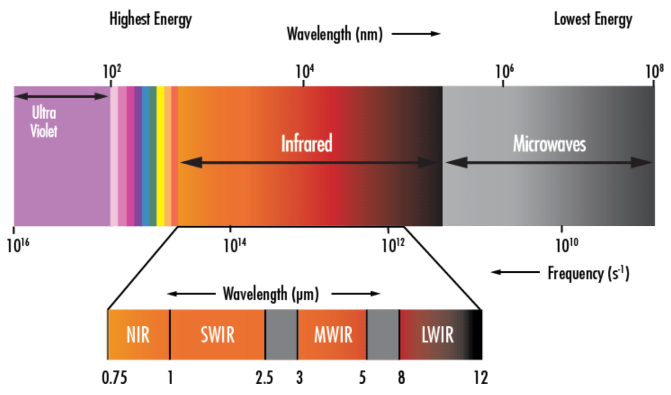


Figure 3.1: Electromagnetic Spectrum Illustrating SWIR wavelength range. Figure adapted from [15]

The mouse we are going to track is freely moving during the acquisition. The dataset we use for this experiment is the video of a mouse with a vascular contrast agent flowing through its vessels. Since it is necessary to inject the dye into the mouse vessels, the animal is anesthetized before imaging. After the injection, the mouse is placed inside an imaging chamber of size 10×10 cm. Depending on the type of anaesthesia, the effects disappear after a certain period of time from the injection, and the mouse starts to wake up. Once enough time has passed, the mouse is totally awake, and it starts to exploring the chamber. Our goal is to follow the mouse when it is awake, going around the chamber and changing its positions over time.

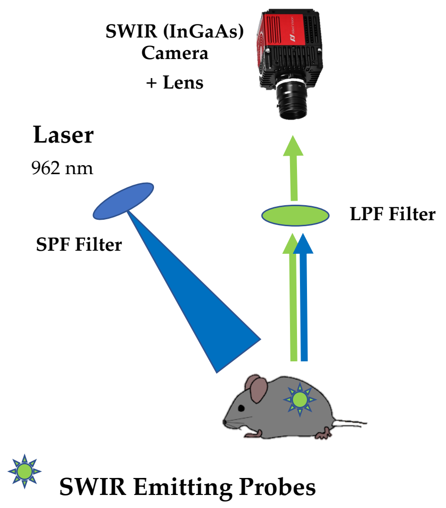


Figure 3.2: Acquisition method and camera setting of SWIR imaging. Detection system composed by the InGaAs camera, and the illumination system composed by the laser, the square engineer diffuser. With the adoption of filter where required.

The camera setting for SWIR imaging is shown in Figure 3.2, and it consists of:

1. Detection system

2. Illumination system

The detection system consists of an InGaAs camera Goldeye 033 of ALLIED-vision that acquires at 300 fps, with exposure time of 3 ms. The aforementioned camera is capable of detecting the emitted signal of a fluorescent probe, excited by a laser. The probe used for this experiment is *Chrome7* dye with an absorption and emission spectra shown in Figure 3.3.

The illumination system consists of a laser that excites the probe with at $\lambda_{laser} = 962$ nm, and an engineered square diffuser Thorlabs ED1-S20-MD diffuses the light with an angle of 20 degrees. The laser with a $\lambda_{laser} = 962$ nm is chosen in order to excite the probe at a wavelength of the *Chrome7*-absorption curve (blue curve in Figure 3.3). The probe absorbs the light and emits the signal. The InGaAs camera detects this signal, and a long-pass-filter (LPF) allows the transmission of only wavelengths $\lambda_{em} > 1150$ nm. With this filter, we can assert that only the

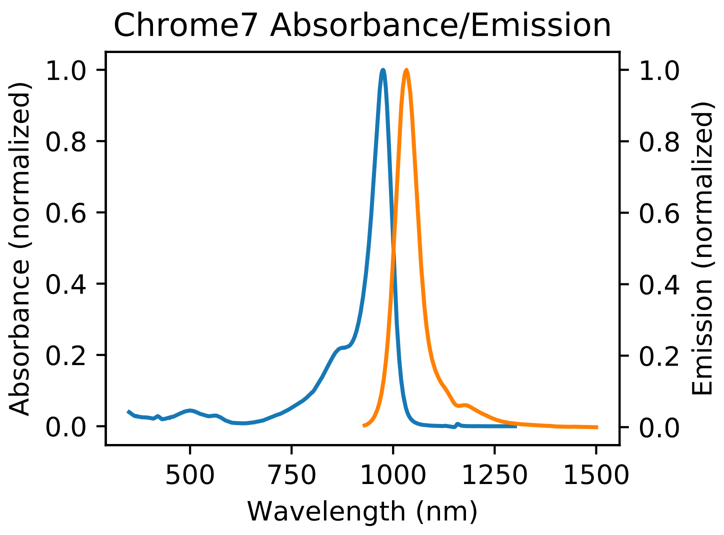


Figure 3.3: Absorption (blue) and emission (orange) spectra of Chrome7 dye. The main peak of absorption is 962 nm, while the light emitted is detected from 1150 nm.

signals with wavelengths of the emission spectra (orange curve in Figure 3.3) are detected. The final result is a video of 1500 frames of 512x640 pixels (see Figure 3.4).

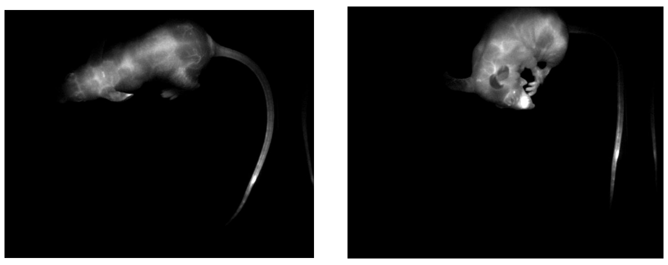


Figure 3.4: Example of frames acquired with SWIR technology. The mouse is awake and it is moving around the chamber. This dataset was acquired by Bernardo Arús and Emily Cosco

3.2 DeepLabCut to track 2D points in a mouse skull

The goal of this experiment is to estimate the skull orientation of a mouse in a 3D space. The skull is a rigid body part that does not change its shape over time, and the 2D video is the projections of this rigid model into a plane. In the video, the mouse can freely move, and based on its orientation some skull points may be hidden. DeepLabCut [3] is a useful toolbox to track animals over time and quantify their behavior. Since it is a markerless technique, it requires a certain amount of time to virtually label the target, but it allows us to avoid physical markers. Moreover, DeepLabCut provides a user-friendly toolbox that generally follows the workflow in Figure 3.5.

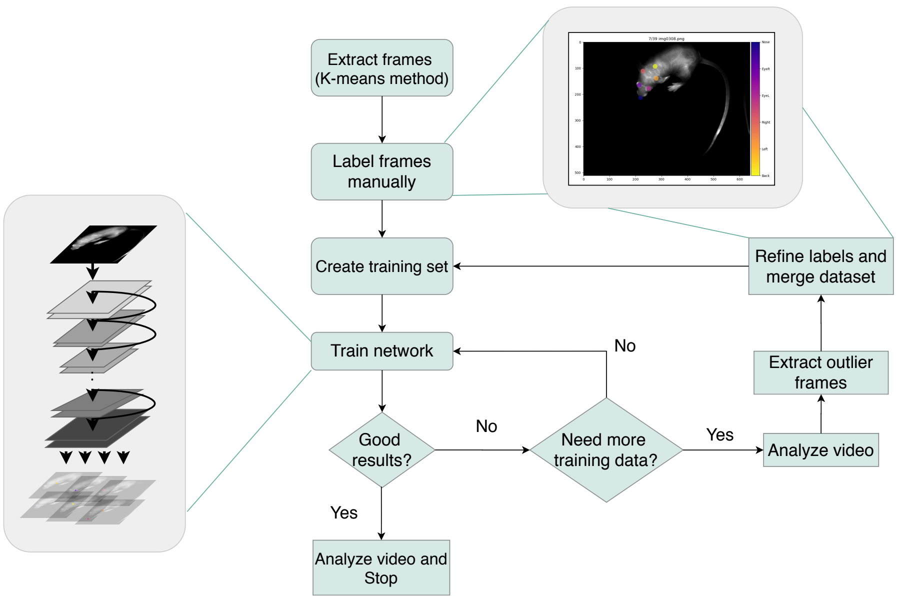


Figure 3.5: DeepLabCut workflow: after the frames extraction, it is necessary to label the body part we want to track. Some of those frames will be used to create a training set and to train the network. At the end of the training, the Neural Network will have learned to correctly label all the remaining frames. If the results are not satisfactory, we can refine the training set and re-train the network.

The input of the workflow is the video, and the first step is to create a training and test set. We are not going to use the whole video to train the network, since an efficient feature of DeepLabCut is to achieve excellent results with minimal training data. Therefore, we extract frames as different as possible from each other. In order to achieve the best performances of the prediction, we provide the largest number of features of the input data, through the minimum number of frames. In fact, we do not want use frames which do not provide any additional information, and which can only slow down the training. Consequently, 40 frames are extracted with a *k-means* algorithm. This is a cluster method that takes the entire set of frames, and splits it into clusters. Each of this cluster contains frames with similar features. So, the extraction consists of selecting frames from different clusters.

The second step is to manually label the extracted frames. Through a specific GUI (see Figure 3.6), we process each extracted frame labeling similar spots and skipping occluded points. For the skull of the mouse six points are selected: nose (*Nose*), right (*EyeR*) and left eye (*EyeL*), right (*Right*) and left ear (*Left*) and a fixed point on the back of the skull (*Back*).

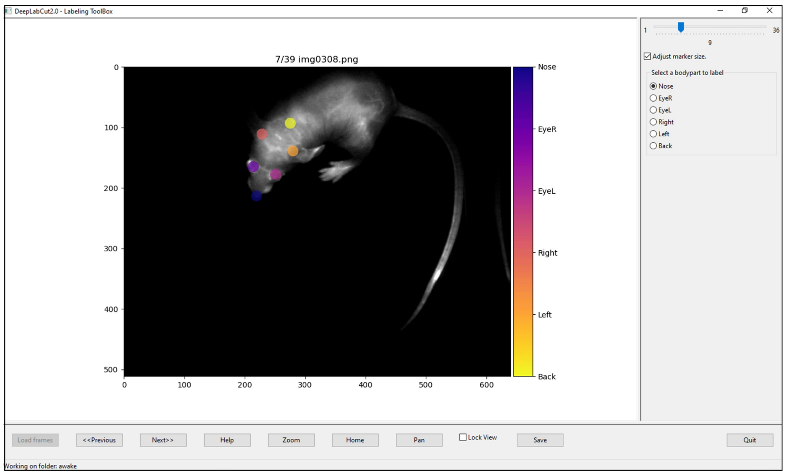


Figure 3.6: GUI of DLC toolbox to label each frame [3]

Once all frames are labeled, we shuffle the dataset and split it in training and test sets. We use a training fraction of 0.5, this means that 50% of the extracted frames will be used to train the network, whereas the remaining dataset will be used during evaluation to test the network.

3.2.1 Network architecture

Once the training set is created, the network starts the training. DeepLabCut is based on transfer learning and it exploits ResNet models (see section 1.2.1) to train the network. However, it makes some changes to adapt the pre-trained model to the problem (Figure 3.7):

1. The weights are trained on labeled data which consists of the object image and the annotated body part locations.

2. During training, the network assigns high probabilities to the labeled locations and low probabilities to the rest of the image.

3. The classification layer at the output of the ResNet is replaced with *deconvolutional layers* in order to produce spatial probability densities to each body part.

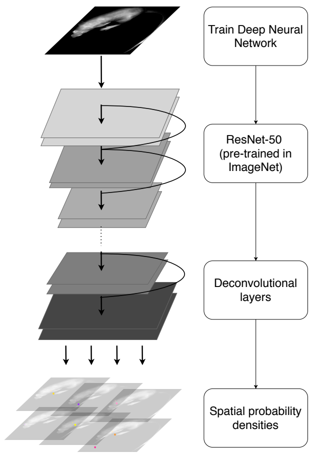

Train Deep Neural Network
ResNet-50 (pre-trained in ImageNet)
Deconvolutional layers
Spatial probability densities

Figure 3.7: DLC architecture to predict the body-part locations. The training set is used to train a ResNet-50 whose Classification layer is replaced with deconvolutional layers, in order to obtain spatial probability densities for each body part.

The network is trained for 500'000 iterations until the loss plateaus and its weights are initialized using the pre-trained ResNet model with 50 layers. The network parameters are summarized in Table 3.1.

Pre-trained model	**Extracted frames**	**Training fraction**	**Iterations**
ResNet-50	40	0.5	500'000

Table 3.1: Network parameters. The ResNet-50 model is trained on 20 frames for 500'000 iterations, and tested on other 20.

If the training results are not satisfactory, an additional functionality provided by DeepLabCut is the extraction of the outlier frames and the consequent network refinement. The output of the training step is a list of frames which contains the following information for each body part:

1. (x, y) label position in pixels
2. The likelihood (value between 0 and 1, implying the level of confidence of the prediction)

In the refinement step, images with low labeling performance are extracted and manually corrected. The frames selected to be refined are those in which one or more body parts jumped more then $\varepsilon = 17$ pixels from the last frame. The frame refined are merged with the previous training set and the network is trained again.

3.2.2 Network performances

In order to evaluate the performance of DeepLabCut predictions, we analyze the ability of the model to correctly identify the body parts. A useful way of visualizing the performance of a prediction model is using **Confusion Matrix** and its validation metrics to have quantified values. Each cell of this table denotes the number of predictions of the model where it classified correctly or incorrectly the classes. Since this is not a binary classification problem, the confusion matrix here below works on multi-class problems. Each label represents a class, for a total amount of six classes: *Nose, EyeR, EyeL, Right, Left, Back*. Since we have the coordinates for both manual and predicted labels, the distance between each predicted label and all six manual labels is computed. The manual label with which the current predicted label has the shortest distance represents its region-based prediction. In Figure 3.8, the predicted label represented by a + is classified as

EyeR, because it has the shortest distance with *EyeR*. The confusion matrix is built based on these allocations.

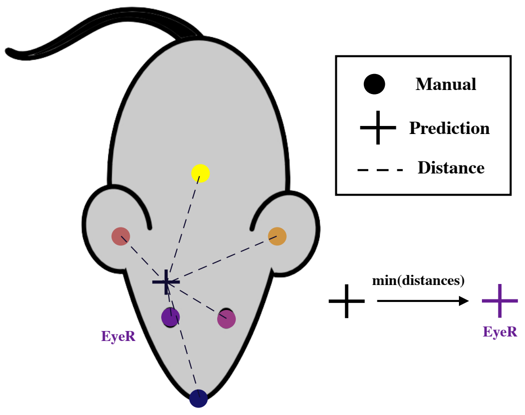


Figure 3.8: Region-based classification with minimum distance between predicted and manual label. The cross is a prediction of the network. We define a criteria of minimum distance to collocate each prediction to a class. The manual label with whom the prediction has the minimum distance is the outcome class. The cross prediction is classified as *EyeR* because it is the nearest manual label.

If we look at the first row of the left table in Figure 3.9, the predicted *Nose* label has the shortest distance with the manual *EyeR* label twice, this means that it happens in two frames. However, the label *Nose* is 27 times well predicted, 2 times is predicted as *EyeR* and 3 times as *EyeL* and so on. This is how a confusion matrix looks like: the number of times that a prediction is correctly or incorrectly made. Confusion matrix it is useful also because it gives efficient performance measures for the model. The values of the confusion matrix in Figure 3.9 take on the following meanings:

- **True Positive (TP)**: number of predictions where the classifier correctly predicts a label for the body part it really belongs to.
- **True Negative (TN)**: number of predictions where the classifier does not have to predict the current body part and it really does not predict it.
- **False Positive (FP)**: number of predictions where the classifier incorrectly predicts other labels as the current body part.

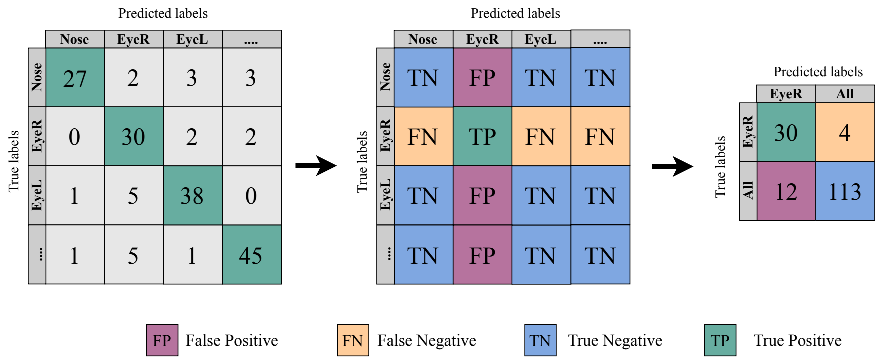


Figure 3.9: Confusion matrix example. Multi-class problem reduced to a binary classification problem.

- **False Negative (FN)**: number of predictions where the classifier incorrectly predicts the current body part as belonging to other labels.

From these values, the following validation metrics are calculated for each frame:

- *Precision*: overall the label predictions, the precision represents how many of those predictions predicted the body part correctly.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- *Recall (or True Positive Rate, TPR)*: overall the human label samples, how many of those were correctly predicted by the classifier.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- *F1-score*: It combines precision and recall into a single measure.

$$\text{F}_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

If we now want to compute these metrics for our confusion matrix, we have to define which class in our multi-class problem is "positive" and which one is "negative". In a multi-class model, it is a common practice to reduce the problem in a binary classification problem. Let's take a specific class, such as the *EyeR* highlighted in Figure 3.9. This class is for a moment considered as the "positive" class while the rest of the table as the "negative". This assumption makes easier to compute the performances metrics, because now the problem has just two classes and the same formulations can be applied. This simplification can be repeated for all the classes. What really matters in this simplification is the effect of reducing the whole matrix in a double class. Everything that *is not EyeR* becomes a unique class and, depending on how many classes are joined in one, the amount of True Negative of the confusion matrix can be really high. This causes an unbalance of the confusion matrix that can misrepresent the model (see Figure 3.9 and $\mathrm{TN_{EyeR}} = 30$ against $\mathrm{TN_{All}} = 113$). To avoid this unbalance, we choose F1-score as the most convenient metric in order to evaluate the model. Since it is a combination of *Recall* and *Precision*, it has a lesser dependence on the True Negative. Therefore, F1-score gives more relevant information about the variation of the other two values, False Negative and False Positive, much more significant in this context. F1-score of each class can be combined to have a single measure for the whole model:

- *Micro F1 (micro-averaged F1-score)*: It does not consider each class individually, but it calculates the metrics globally.

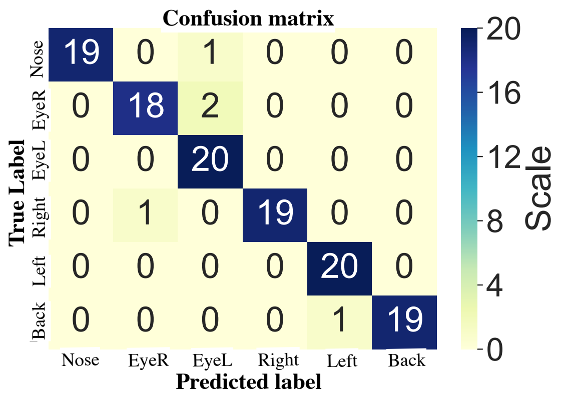
$$\mathrm{MicroF_1} = \frac{\mathrm{TP_{Total}}}{\mathrm{TP_{Total}} + \mathrm{FP_{Total}}} = \frac{\mathrm{TP_{Total}}}{\mathrm{TP_{Total}} + \mathrm{FN_{Total}}}$$

- *Macro F1*: it calculates F1-score for each class individually and takes the mean of the measures.

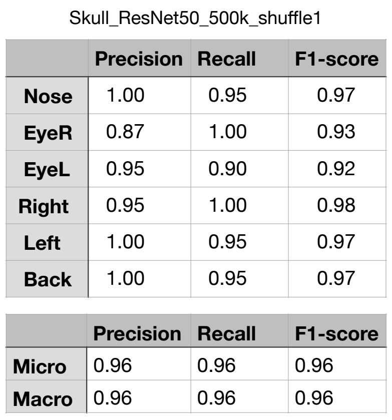
$$\mathrm{MacroF_1} = \frac{1}{\mathrm{N}} \sum_{\mathrm{i=1}}^{\mathrm{N}} (\mathrm{F_1})_\mathrm{i}$$

In Figure 3.10, it is shown the confusion matrix and its performances calculated for the test set. The overall performances are quite excellent, and the network predicts mostly ever the correct label location. However, the network makes a double mistake in predicting the *EyeR* label as *EyeL*. If we now look at the frames in which this mistake is made (see Figure 3.11), we can assert that the error is not made by the prediction, but the minimum distance criteria as region-based classification is wrong-classifying the predicted label. During the manual labeling step, we tried to not label hidden body parts such as the *EyeR* in this frame.

Consequently, by computing the distance of *EyeR* predicted with each manual label, necessarily the shortest distance of *EyeR* is with *EyeL*, while the predicted location is correct.



	Nose	EyeR	EyeL	Right	Left	Back
Nose	19	0	1	0	0	0
EyeR	0	18	2	0	0	0
EyeL	0	0	20	0	0	0
Right	0	1	0	19	0	0
Left	0	0	0	0	20	0
Back	0	0	0	0	1	19



Skull_ResNet50_500k_shuffle1

	Precision	Recall	F1-score
Nose	1.00	0.95	0.97
EyeR	0.87	1.00	0.93
EyeL	0.95	0.90	0.92
Right	0.95	1.00	0.98
Left	1.00	0.95	0.97
Back	1.00	0.95	0.97

	Precision	Recall	F1-score
Micro	0.96	0.96	0.96
Macro	0.96	0.96	0.96

Figure 3.10: Test set confusion matrix and performances for the ResNet50 with 500'000 iterations

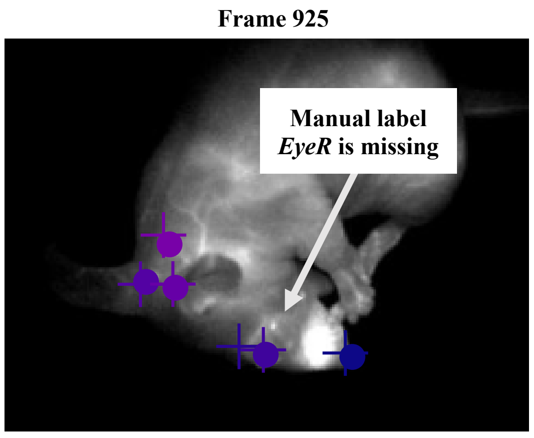


Figure 3.11: Wrong classification but correct predicted location. The *EyeR* manual label is missing, and *EyeR* prediction is classified as *EyeL* because of the minimum distance

Therefore, we can assert that the network achieves excellent results with these set of parameters. The training is performed by using a NVIDIA GeForce RTX 2070 GPU, and for 500'000 iterations it requires a little bit more than *8 hours*.

3.3 Detection of mouse skull orientation

In the previous section, we have trained the network and for each frame the 2D coordinates have been provided. So, it becomes possible to predict the 2D skull labels for the entire video by using the trained weights. From this analysis, DeepLabCut provides a file that contains the predicted x and y pixel coordinates of each labels. It is a *.csv* file easy to be manipulated.
These two-dimensional coordinates can now be used to extract the overall 3D movement of the skull over time. In fact, the 3D skull is considered as a rigid body whose movements in the space are possible to be estimated through Equation 1.7.

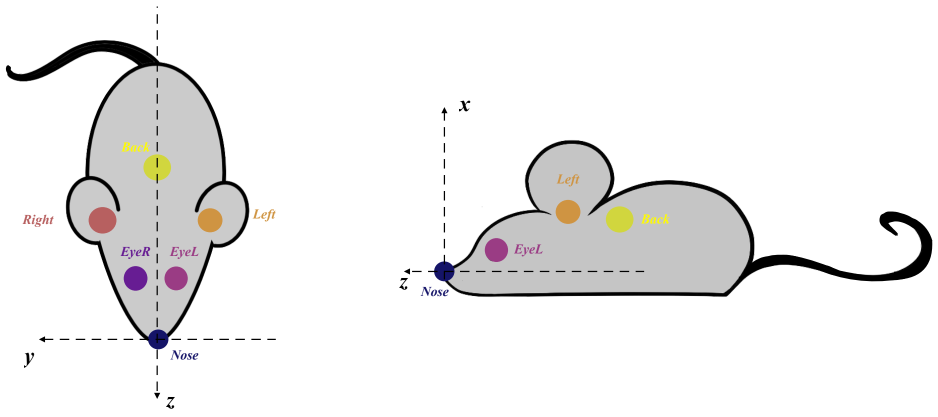


Figure 3.12: Mouse skull world coordinates. The **x** and **y** coordinates of each label, with respect to the *Nose*, is physically measured by using *Fiji-ImageJ* on two frames: one in which the skull is aligned with *yz* plane, another in which it is aligned with the *xz* plane. Existing sketch adapted to the context.

The 3D skull *world coordinates* need to be defined. In Figure 3.12 is shown the adopted reference system: the arbitrary origin point is the *Nose*, so the location of each label is determined with respect to this one. Thus, we take two frames: the first in which the skull is approximately aligned with the plane yz, and the second in which the skull shows only one side aligned with the xz plane. Through *Fiji Image-J*, we measure the (x, y, z) coordinates of each label.

Starting from the 2D labels location $\tilde{\mathbf{p}}_{img}$ and the 3D skull coordinate $\tilde{\mathbf{P}}$, we can estimate the 3D skull direction in each frame by using an OpenCV function called *SolvePnP*. This function reverse the Equation 1.7 to obtain the projection matrix

M, and eventually $[\mathbf{R}]$, which will then inform us about the 3D skull orientation. Thus,

$$\tilde{\mathbf{p}}_{\mathrm{img}} = \mathrm{M}\ \tilde{\mathbf{P}} \tag{3.1}$$

where the projection matrix M is

$$\mathrm{M} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \tag{3.2}$$

$\mathbf{K}$, $[\mathbf{R}|\mathbf{t}]$ (see section 1.3.1 and Equation 1.6 to the explicit notation) are respectively the intrinsic matrix (specific parameters depending on the camera model), and extrinsic matrix (it describes how the object and the camera are rotated and translated with respect to each other). For the intrinsic matrix, we are assuming that the camera is pointing exactly to the center of the image and that there is not distortion. Therefore, we can approximate

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

with

$$\mathrm{f} = \mathrm{f_x} = \mathrm{f_y},$$

$$(\mathrm{c_x}, \mathrm{c_y}) = \left(\frac{\mathrm{I_{weight}}}{2}, \frac{\mathrm{I_{height}}}{2}\right)$$

where f_x and f_y are the focal lengths of the camera, and I_{weight} and I_{height} are the sizes of whether frame. Assuming that the intrinsic matrix never changes during the video because we are not changing the optics or zooming, we must handle only the extrinsic camera matrix. On the other hand, the extrinsic parameters can change, for instance, if we rotate the camera (consequently, the matrix $\mathbf{R}$ will change) or we change its location (the vector $\mathbf{t}$ will change). We can reverse this scenario if we now move the object pose rather than the camera, and the approach does not change. Since we have the rigid model of the skull in 3D space and its 2D projections on the image plane (output of DeepLabCut), we can easily solve the Equation 3.1 by replacing these coordinates respectively with $\tilde{\mathbf{p}}_{img}$ and $\tilde{\mathbf{P}}$, and compute $\mathbf{R}$ and $\mathbf{t}$. These two parameters tell us how the skull is oriented in the 3D space. In Figure 3.13, examples of the estimated directions based on the predicted labels.

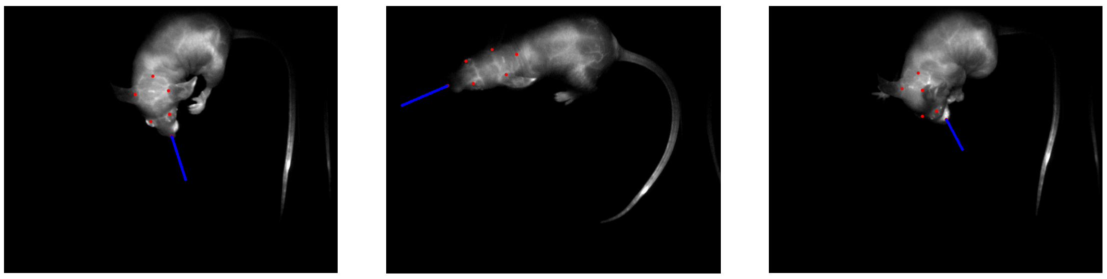


Figure 3.13: Mouse skull direction over time. The red dot are the predicted label, the blue line is a visualization of the predicted orientation of the mouse skull.

3.3.1 Validation results of skull orientation

In order to analyze the results of the predictions, we evaluate the relative error of the skull orientation between predicted and manual measurements. We calculate the rotation matrix of the 2D projections coming from both manual and predicted labels. To evaluate the accuracy of the skull orientation, we calculate the *Rotation Matrix Relative Error*, and the *angle* between the two estimated directions in a 3D space.

Since the estimation of the skull orientation is strictly related to the 2D location of the skull labels, we calculate the rotation matrix by using the Equation 3.1 with both manual and predicted labels. For each frame we have:

$$[\mathbf{R}]_{\text{man}} = \begin{bmatrix} \mathrm{r}_{11} & r_{12} & r_{13} \\ \mathrm{r}_{21} & r_{22} & r_{23} \\ \mathrm{r}_{31} & r_{32} & r_{33} \end{bmatrix} \tag{3.4}$$

$$[\mathbf{R}]_{\text{pred}} = \begin{bmatrix} \mathrm{r}'_{11} & r'_{12} & r'_{13} \\ \mathrm{r}'_{21} & r'_{22} & r'_{23} \\ \mathrm{r}'_{31} & r'_{32} & r'_{33} \end{bmatrix} \tag{3.5}$$

where $[\mathbf{R}]_{\mathrm{man}}$ is obtained by using the manual labels and $[\mathbf{R}]_{\mathrm{pred}}$ by using the predicted ones. To make a comparison between the two matrices, we consider a numerical and a geometric point of view. To numerically evaluate the accuracy, we compute the **Matrix Rotation Relative Error** as

$$\mathrm{Err}_{\mathrm{rel}} = \frac{||\ [\mathbf{R}]_{\mathrm{man}} - [\mathbf{R}]_{\mathrm{pred}}\ ||_2}{||\ [\mathbf{R}]_{\mathrm{man}}\ ||_2} \tag{3.6}$$

where $||A||_2$ is the L_2 *norm*, that is equal to

$$||\mathrm{A}||_2 = \left(\sum_{\mathrm{ij}} |\mathrm{a}_{\mathrm{ij}}|^2\right)^{1/2} \tag{3.7}$$

and a_{ij} is the element of the matrix A. We refer to LAPACK documentation for the computation of the relative error between matrices [16]. The matrix norm helps us to correlate a matrix to a real number in order to have a numerical comparison between matrices. The norm gives us information about the matrix magnitude, and by using it in an relative error, we can say if the difference between two vectors is large or small with respect to the correct measure. For small difference, we have small error. From the Equation 3.6, we evaluate the accuracy of the predicted orientation compared with the manual one.

Geometrically, the distance between two 3D rotation matrices is often evaluated by measuring the θ angle between the two rotations. In this analysis we look only at the rotation matrices. This means that, since we are not considering the translation, the predicted and manual skull segmented models have the same origin. Therefore, the angle θ between our two skull represents the difference between the relative rotation matrices. The orientation of the skull can be represented by a vector whose direction is defined by the relative rotation matrix. The angle θ in Figure 3.14 between two 3D vectors is calculated as:

$$\overrightarrow{u} \cdot \overrightarrow{v} = ||\overrightarrow{u}||\ ||\overrightarrow{v}||\ \cos\theta \tag{3.8}$$

$$\overrightarrow{u} \cdot \overrightarrow{v} = \mathrm{u_x v_x} + \mathrm{u_y v_y} + \mathrm{u_z v_z} \tag{3.9}$$

$$\cos\theta = \frac{\overrightarrow{u} \cdot \overrightarrow{v}}{||\overrightarrow{u}||\ ||\overrightarrow{v}||} \tag{3.10}$$

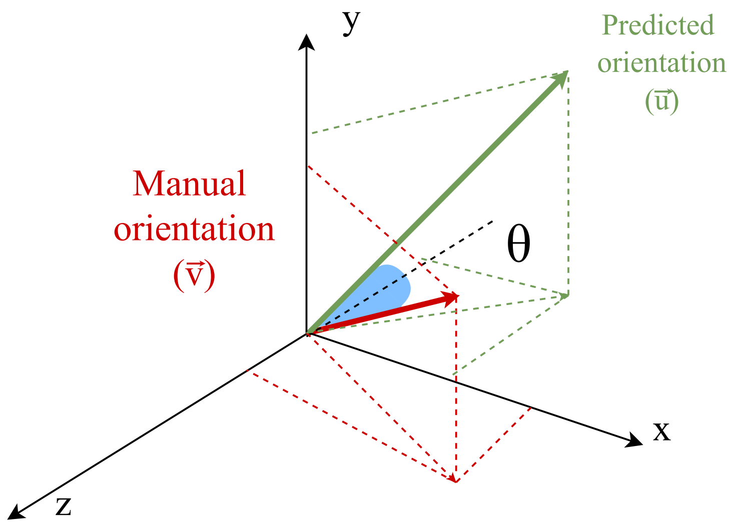

y
Predicted orientation ($\vec{u}$)
Manual orientation ($\vec{v}$)
θ
x
z

Figure 3.14: Angle between two vectors oriented accordingly to their rotation matrices

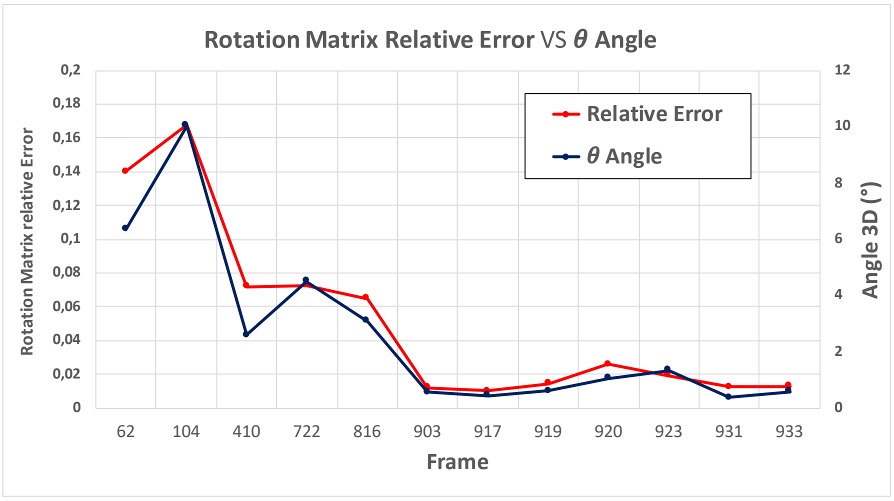


Figure 3.15: Comparison between numerical an geometric analysis of skull orientation in some test frames

In Figure 3.15, both numerical and geometric point of views are visualized. From the metrics, we can assess that to a smaller angle corresponds a smaller relative error of the rotation matrices. We can also observe that in the last frames the error is smaller than the initial frames. While in the first few frames, the classifier is less accurate in predicting. If we look up at the frames, we can understand the reason of these inaccuracies. In Figure 3.16, both 2D and 3D skull orientations for some of

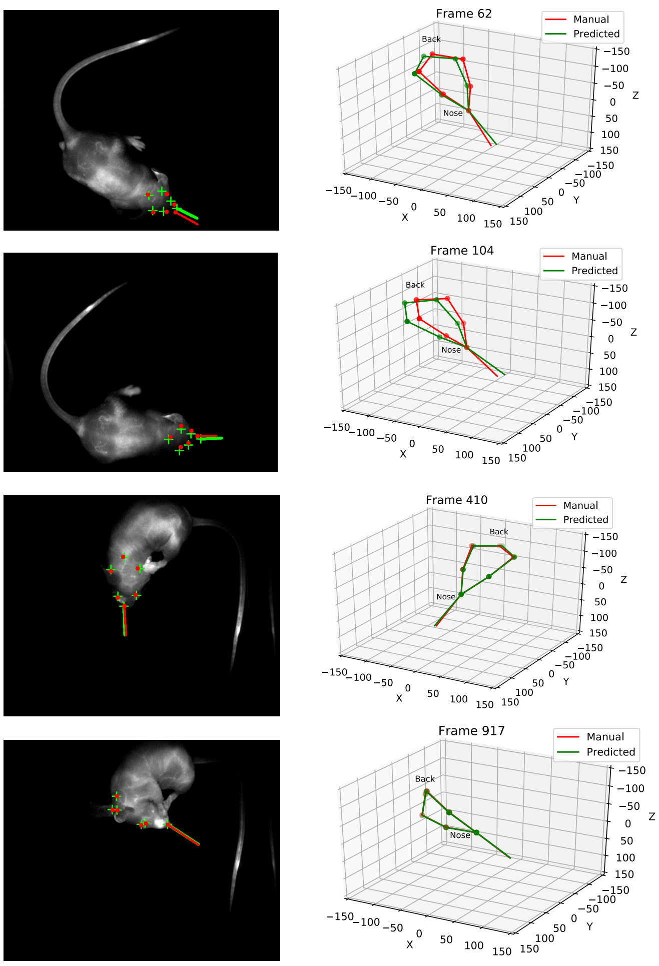


Figure 3.16: Predicted and manual skull orientation in 2D and 3D for different frames. The first frames are less illuminated then the last ones, this may cause errors in the prediction.

those test frames are shown. It is evident that in frame 62 and 104 the 2D predicted labels are less accurate than the other two frames, and this outcome is evident in the orientation predictions. This may be caused by the lack of light during the acquisition. In fact, the predicted points are trying to guess between dark pixels. While in the last frames, pixels are clearer thanks to the good illumination.

With this experiment, we have shown that it is feasible to extract 3D information from a monocular video. The predicted skull orientations are consistence with the expectations, since there is an evident correspondence between the real video and the 3D skeleton. Moreover, the overall angle error in a 3D space is acceptable. We may even evaluate the actions performed by the mouse in each frame, by visualizing its orientations. Certainly, we can say just looking at the 3D skeleton if the mouse is looking up or down, while is moving around the chamber. Further researches may include, for example, the next steps of the photogrammetry pipeline. Once the main points of the object are well-located in a 3D space, as the Structure From Motion does in Meshroom (see section 2.1), would be possible to generate a dense surface and compute the pixel depths around the newly skeleton? If yes, we may go ahead with creating a Mesh, and a final Texture of the skull. We will look for answering these question in further researches. For now, we focus on understand how this three dimensional information may be extracted, not only for a rigid body part of the mouse, but for the entire body, including deformable parts.

Chapter 4

3D pose estimation of an hand in motion with camera-mirror stereo vision

In the previous chapter, the third dimension of an object in a 2D video was predicted by making a convenient choice: tracking a rigid body part. The aforementioned object was the skull of a mouse whose six fixed points were selected, and their relative positions were measured. The projection matrix and the 2D tracked labels allowed us to estimate the skull orientation over time, and to obtain a first 3D model in motion. The above method was useful for monitoring a specific and rigid body part, the skull, and to extract general information of that, the gaze direction. How about those body parts that break the rule of being rigid? Is it possible to track their movements and extract more complex information? For instance, knowing how and in which direction the ventral region changes its shape would be useful to understand if the mouse is standing up, crouching or having other similar behaviours like those represented in Figure 4.1.

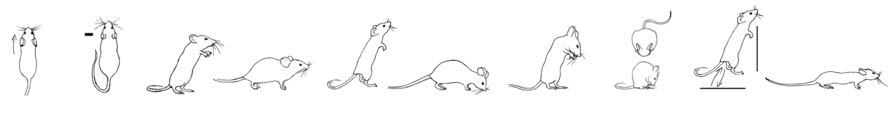


Figure 4.1: Individual behaviours of a mouse [17]

Here, we focus on tracking a non-rigid body in motion. First, tracking movements in three dimensions without knowing a prior model of the object requires multiple views of the same scene. Therefore, a stereo vision system is proposed, which consists in recording the object from two camera views. However, we use only a camera combined with a mirror, in order to use its reflection as second view to avoid multiple cameras.

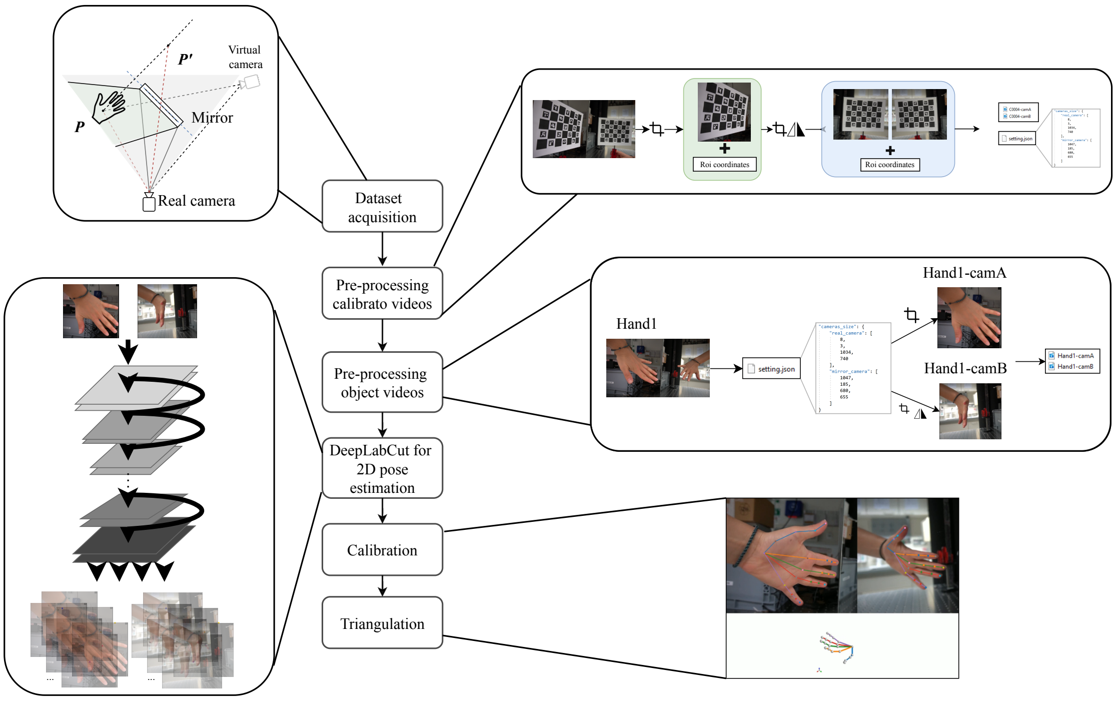


Figure 4.2: Total pipeline for monocular stereo vision experiment using DeepLab-Cut and Anipose

The pipeline of this experiment is shown in Figure 4.2, and each step will be explained in detail in the next sections. In such a system, we use a camera to film an object and its reflection at once. After, it needs to be cropped to obtain two different videos in order to emulate two different cameras: the real and the virtual one. This cropping is repeated for two different subjects: a calibration board chess and an hand in motion. Then, the two hand videos are used to train a neural network in DeepLabCut, while the calibration videos are used as a starting point for the usage of Anipose. This latter is a toolkit for markerless 3D pose estimation, and consists of a series of filters to resolve 2D tracking errors coming from DeepLabCut, and a 3D calibration module to extract the third dimension by performing triangulation [18].

4.1 Datasets: calibration and hand skeleton

The new camera setting proposed represents a preliminary experiment that is conducted on a generic object in motion: an hand in motion. This is a preliminary dataset useful to check the mirror-camera setting before applying it to a dataset acquired through SWIR technology. Once this part will be validated, the same procedure should be adopted to track and evaluate different deformable body parts of a mouse such as its ventral region, the tail and/or the fluorescent vessels.

First, this third experiment requires the acquisition of two distinct videos recorded with the identical mirror-camera system:

- A **calibration video**: to automatize the calibration of both cameras, a particular board chess is recorded while is moving and showing to the camera different inclinations of its squares.
- An **hand in motion video**: an object capable of moving, rotating and changing its shape to record various poses over time.

The videos are recorded in 50fps with a SONY ILCE-6400 in RGB color space, and frame size of 1920x1080 pixels. The farthest point of the mirror has a distance of $\approx$ 40 cm from the camera lens to include both the subject and its reflection into the field of view. In the following two paragraphs, the calibration and the moving hand videos are described in detail.

4.1.1 Calibration video: ChArUco board

An important step for 3D reconstruction is to correct the typical distortions introduced by the pinhole camera (see section 1.3.1). The distortion essentially makes straight lines appear curved within an image and it depends on many factors such as the optical design of lenses or the relative position of the camera to the subject. However, in applications that use stereo images, the distortions need to be corrected. The distortion D is modeled with three parameters k_1, k_2, k_3 as

$$D([x,y]) = \begin{bmatrix} x + x(k_1(x^2+y^2) + k_2(x^2+y^2)^2 + k_3(x^2+y^2)^4) \\ y + y(k_1(x^2+y^2) + k_2(x^2+y^2)^2 + k_3(x^2+y^2)^4) \end{bmatrix} \tag{4.1}$$

where x and y are the 2D coordinates of the object within the image. To discover these parameters, the easiest way is to provide a well defined object whose dimensions are known, and by using its coordinates in the 2D plane matched with its dimensions in 3D space, the distortion coefficients can be calculated. The above-mentioned object is the ChArUco board in Figure 4.3.

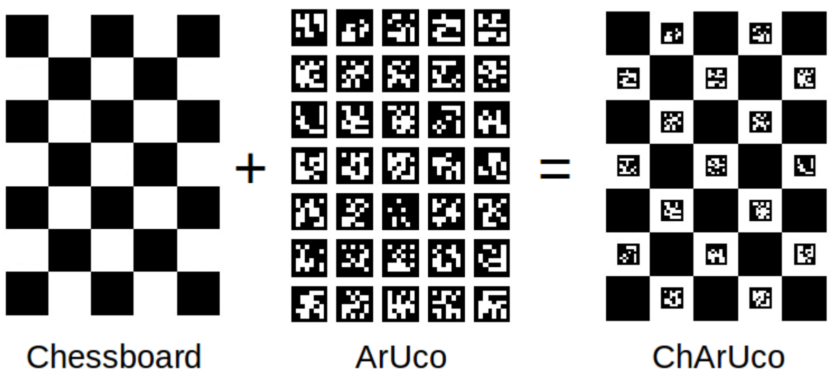


Figure 4.3: Charuco board definition [19]: it is a combination between Chessboard and ArUco, in order to detect markers with more accuracy

A ChArUco board tries to combine the benefits of two other calibration objects: a common chessboard and the ArUco markers. The latter are useful because their versatility and its easily detection. In fact, they can be detected even if they are occluded or not completely visible, but the accuracy of their corner positions is low. Thus, in ChArUco board, ArUco is combined with a common chessboard whose patterns can be easily refined because they are surrounded by black squares. In Figure 4.4 is shown the ChArUco board adopted for this experiment. Its pattern has 7 × 5 squares, with 4 bit markers. The size of each marker is 15 mm, the size of each square is 26 mm, and the length separation between marker and the square board is 1 mm. In the calibration video, the ChArUco board and its reflection are filmed for less than one minute while we manually rotate the board to show different angles to the two field of view of the cameras (real and virtual).

In section 4.4.1, an overall explanation of how the distortion parameters are calculated will be explained.

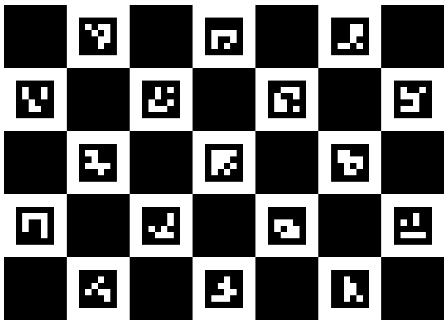


Figure 4.4: Charuco board used for calibrating cameras in our experiment

4.1.2 Object in motion: hand skeleton

The object we are going to reconstruct is an hand in motion. The hand moves, rotates, and shows to the camera different poses in some of which itself occludes or completely hides other parts (see Figure 4.5). Fingers sometimes overlap and the palm rotation causes a reversal of the position from the thumb to the little finger, which could confuse the detection, and the correct hand pose.

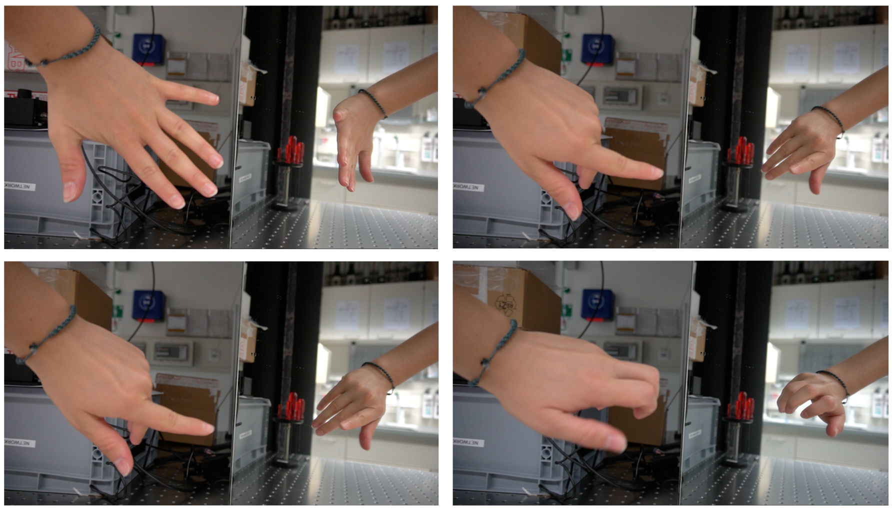


Figure 4.5: Frames of moving hand with camera-mirror setting. Real object on the left, and its reflection on the right of each figure. From different frames, not always all fingers are visible

At this point, the mechanical goal is to estimate the three-dimensional pose of the hand skeleton, and see if there is eventually a coherence in the movement between the 3D estimated pose and the 2D videos. Furthermore, what is worth evaluating in terms of classification and regression validation is:

- The accuracy in finger detection even when two of them overlap
- The level of resemblance to reality

To answer to these questions, simple geometric procedures are adopted, exploiting results that are obtained from the DeepLabCut and Anipose tools. Based on its duration and the acquisition rate of the camera, the hand video consists of 1435 frames.

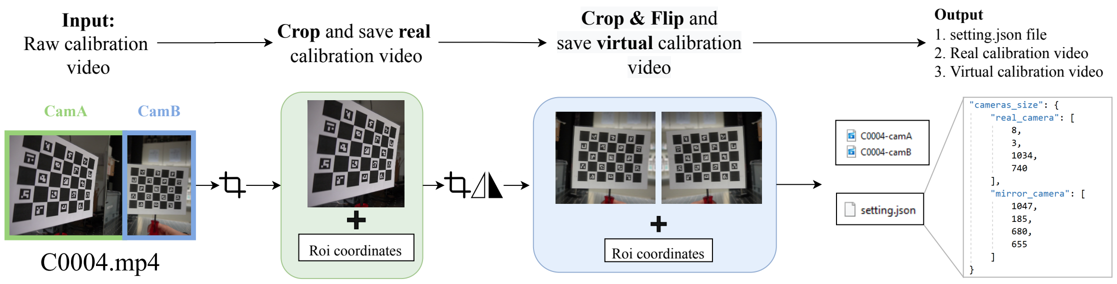
4.2 Pre-processing data

In the course of the video recording, the adoption of a second camera might introduce a synchronization problem (e.g. double acquisition not exactly in phase, interoperability issues between different cameras, etc.). This phenomenon is totally avoided if a mirror is used. However, in order to obtain two different views from a single one, a pre-processing step is required.

4.2.1 Extracting real and virtual views from a single camera system

In our video, we have two distinct regions of interest. One is the direct view of the object, and the other is its reflection. The pre-processing of the videos consists in cropping the original video to emulate a double camera system. These steps are shown in Figure 4.6.

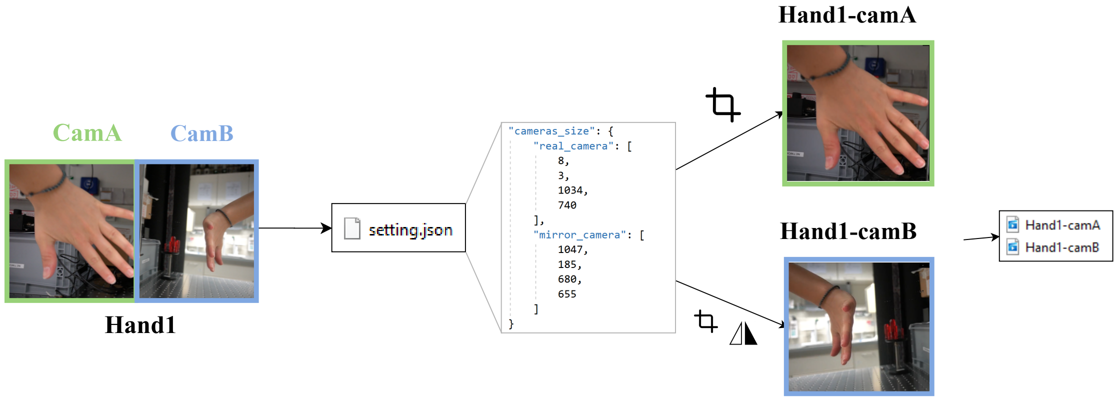
We primarily process the raw calibration video by manually selecting the regions of the two views to be cropped. The "direct" view (*camA*) of the object is taken by cropping it, while the "virtual" view (*camB*) is taken by cropping and flipping it. Since both calibration and object video are recorded by the same system, it is necessary to have a coherence between them. Therefore, the software asks the user for selecting a *Region Of Interest (ROI)* and afterwards it uses the same ROI coordinates to crop the object in the identical fashion. With this purpose, a


Input:
Raw calibration video
Crop and save real calibration video
Crop & Flip and save virtual calibration video
Output
1. setting.json file
2. Real calibration video
3. Virtual calibration video
CamA
CamB
C0004.mp4
Roi coordinates
Roi coordinates
C0004-camA
C0004-camB
setting.json

```
"cameras_size": {
    "real_camera": [
        8,
        3,
        1034,
        740
    ],
    "mirror_camera": [
        1047,
        185,
        680,
        655
    ]
}
```

Figure 4.6: Pre-processing calibration video. The input on the left is the original video (green square indicate real object, blue square the reflection). We first select and crop the real view, then the reflection is selected, cropped and flipped. The coordinates of the manual selection are saved.

setting.json file is created that contains the real and the virtual camera coordinates information ($[x, y, width, height]$ of a rectangle). The outputs of this pre-processing step are two videos and a json file.



```
"cameras_size": {
    "real_camera": [
        8,
        3,
        1034,
        740
    ],
    "mirror_camera": [
        1047,
        185,
        680,
        655
    ]
}
```

Figure 4.7: Pre-processing object video. On the left, the original input of the moving object (green square indicates the real view, and the blue square the reflection). Without manually selecting, the software automatically crops and flips the video by using the coordinates saved from the pre-processing of the calibration video.

Once the calibration videos are extracted, we proceed with the extraction of the object videos (see Figure 4.7). This time, the two videos are automatically created by using the coordinates saved in the *setting.json*. This is necessary because the calibration videos need to be coherent in terms of frame sizes to the object ones, and by using the saved coordinates, it is legitimate to say that the dimension constraints are satisfied.

4.3 DeepLabCut to track 2D points in an hand skeleton

The two videos extracted in the previous section (*Hand1-camA* and *Hand1-camB*) are the next step inputs: train a Convolutional Neural Network exploiting DeepLab-Cut (see section 3.2.1 and Figure 4.8).

Based on the same workflow in Figure 3.5, from both *camA* and *camB*, 80 frames are extracted using k-means method, for a total amount of 160 frames. The object is labeled with an amount of 21 labels that follow the skeleton shape, whether the palm is facing up or down. The notations adopted are reported in Figure 4.8, the same user interface of Figure 3.6 and the same criteria in labeling are respected.

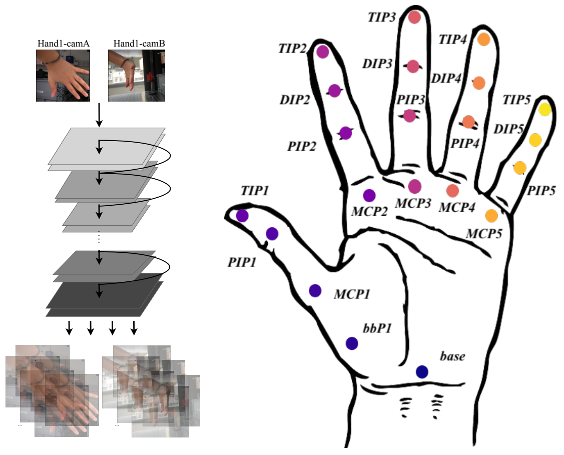


Figure 4.8: General network architecture pipeline (left), Hand model with 21 keypoint labels (right)

Once each frame is labeled, DeepLabCut shuffles the two 80-datasets and splits them in training and test set with a training fraction of 0.5. Then, it joins the training sets coming from the two views to generate a unique training set, and the same procedure is done for the test set (see Table 4.1).

The weights of the network are again trained on labeled data which consists of the object images (this time recorded from two views) and their annotated body

	CAM A	**CAM B**	**TOTAL**
Training set	40 frames	40 frames	80 frames
Test set	40 frames	40 frames	80 frames
Validation set	1355 frames	1355 frames	2710 frames
Total dataset	1435 frames	1435 frames	2870 frames

Table 4.1: Dataset split in training and test set. The remaining frames are the validation set

part locations. Due to the large amount of labels and frames extracted, two different Residual Neural Networks are tested. Investigating multiple neural networks aims to exploit a neural network more efficiently in terms of results and time. These neural networks are trained to account for a wide variety of changes in the studied frames, it is worth though exploring if deeper networks achieve better results even if they spend more time or if the final result does not meet the expectations.

The network is trained for 700’000 iterations and its weights are initialized using two pre-trained ResNet models with 50 layers and 101 layers (see Table 4.2).

Pre-trained model	**Training set size**	**Test set size**	**Training fraction**	**Iterations**
ResNet-50	80	80	0.5	700’000
ResNet-101	80	80	0.5	700’000

Table 4.2: Networks parameters. We train two neural network (ResNet-50 and ResNet-101) by using the same parameters (training and test size, iterations)

4.3.1 Networks comparison

In order to evaluate the performance of DeepLabCut predictions, we consider classification and regression separately. To measure classification, we analyze the ability of the models to correctly identify the body parts. Moreover, two different deep network architectures are tested to evaluate the advantage of using a deeper network rather than a shallower. To measure regression, we evaluate the pixel error between paired observations (real and predicted).

Once the network is trained, videos are analyzed using the trained weights. From this automatic analysis, DeepLabCut provides files that contain the predicted

x and y pixel coordinates of each labeled body part and the network likelihood, per frame. To make comparisons between the networks, two main *.csv* files are used which contain a list of:

- labels manually defined by human (*ground truth*)
- labels predicted by the network.

To have a qualitative visualization of human labels against the network predictions, a useful image is given as in Figure 4.9. The human labels are indicated with a colored dot, while the predictions are divided in:

- points labeled with an high level of confidence (+)
- points labeled with a low level of confidence (×)

where the level of confidence depends on the likelihood calculated by the network.

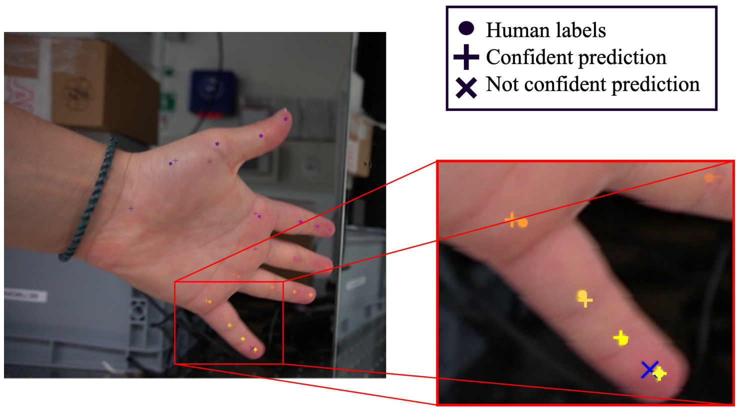
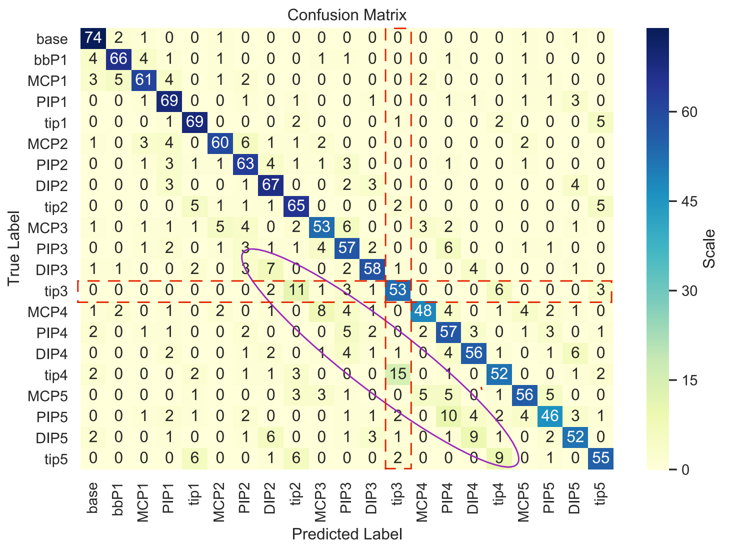


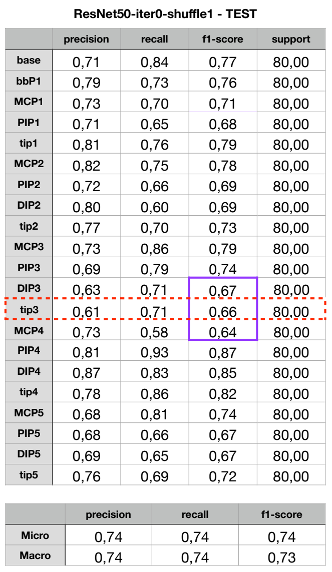
Figure 4.9: Human and predicted labels with level of confidence

For instance, as it is clear from the Figure 4.9, the yellow and orange labels are almost correctly predicted (+), while the blue label is a cross (×) because it should be placed on the thumb instead of on the little finger. This type of error is a *region-based* misclassification, in the sense of misdetecting one finger rather than another. However, even if the yellow and orange labels are correctly placed, inaccuracies are detectable, and we can consider this analysis as a *pixel-based* misclassification, and thus how large is the network error in terms of the pixel distance between the predicted label and the *ground truth.* The region-based validation is carried out on test set by using *Confusion Matrix.* For the pixel-based validation we use the *Mean Absolute Error (MAE)* on the test and training set to evaluate the similarity and accuracy between the predicted and manual labels. Moreover, further evaluations are accomplished to also take into account the duration of the training and how the prediction errors vary when some parameters change.

Classification validation

A useful way of visualizing the performance of a prediction model is using a **Confusion Matrix**, already described in section 3.2.2. Even in this experiment, we deal with a multi-class problems. Each label represents a class, for a total amount of 21 classes (*base, bbP1, MCP1 ... tip5*), and the criteria of the minimum distance is adopted to evaluate if a prediction is correct or incorrect. Since we are dealing with 21 classes, the unbalance of the multi-class model is more evident. For this reason, considering the F1-score is again the most convenient choice to evaluate the network performances.





ResNet50-iter0-shuffle1 - TEST

	precision	recall	f1-score	support
base	0,71	0,84	0,77	80,00
bbP1	0,79	0,73	0,76	80,00
MCP1	0,73	0,70	0,71	80,00
PIP1	0,71	0,65	0,68	80,00
tip1	0,81	0,76	0,79	80,00
MCP2	0,82	0,75	0,78	80,00
PIP2	0,72	0,66	0,69	80,00
DIP2	0,80	0,60	0,69	80,00
tip2	0,77	0,70	0,73	80,00
MCP3	0,73	0,86	0,79	80,00
PIP3	0,69	0,79	0,74	80,00
DIP3	0,63	0,71	0,67	80,00
tip3	0,61	0,71	0,66	80,00
MCP4	0,73	0,58	0,64	80,00
PIP4	0,81	0,93	0,87	80,00
DIP4	0,87	0,83	0,85	80,00
tip4	0,78	0,86	0,82	80,00
MCP5	0,68	0,81	0,74	80,00
PIP5	0,68	0,66	0,67	80,00
DIP5	0,69	0,65	0,67	80,00
tip5	0,76	0,69	0,72	80,00

	precision	recall	f1-score
Micro	0,74	0,74	0,74
Macro	0,74	0,74	0,73

Figure 4.10: Test set confusion matrix and validation metrics of ResNet-50

The confusion matrix obtained for the test set with both ResNets are shown in

Figure 4.10 and in Figure 4.14. Because of the number of classes in the model, they are 21x21 arrays. The performances detailed in section 3.2.2 have been calculated for each label and then a global measures (micro and macro metrics) have been determined. As we can immediately see in Figure 4.10, the overall classification performance of the network looks valuable due to the high number of True Positives (values on the matrix diagonals) with an Micro and Macro F1-score of 0.74 and 0.73. However, some metrics values in the performances table deviate from the others. If we look up to the matrix, we can finally see *where* the performances decrease. For instance, if we consider the ResNet-50 table in Figure 4.10, the lowest *F1-score* values belong to those classes between *DIP3* and *MCP4*. If we now investigate those labels on the confusion matrix (Figure 4.10 region surrounded in purple), we can clearly see that around those labels are located the highest number of False Positives and False Negatives. As the purple diagonal suggests, there is a sort of "shifted" misdetection, as if the common error of the network is to mix-up a finger with the one that precedes it or follow it (e.g, *tip3* predicted as *tip2*, *tip4* as *tip3*, *tip5* as *tip4*). Let's consider now the *tip3* class. It has an *F1-score* of 0.66, since it is effected by the row and column values of the confusion matrix squared in dashed red line. The column represents the False Positive, and *tip4* label is predicted as *tip3* for an amount of 15 predictions. To understand why this happens, we look up at the frames in which this error was made. We have found that the main errors made by the network can be summarized in four groups:

1. Despite the body part is well seen, the network makes the wrong prediction. In Figure 4.11a, *tip4* is predicted as *tip2*.

2. The real body part is completely hidden by the body part that was in its place predicted and this confuses the network. In Figure 4.11b), the little finger is completely hidden, and its labels are confused with those of the ring finger.

3. The real body part is completely hidden, but the prediction seems to be well located. However, the human label is missing and the nearest label is the output of the final prediction. In Figure 4.11c, *PIP5* and *DIP5* are well-located, but their manual labels are missing.

4. The predicted region is almost correct, but the criteria of the minimum distance

does not allow a correct evaluation of the result. In Figure 4.11d, *MCP5* has the shortest distance with MCP4.

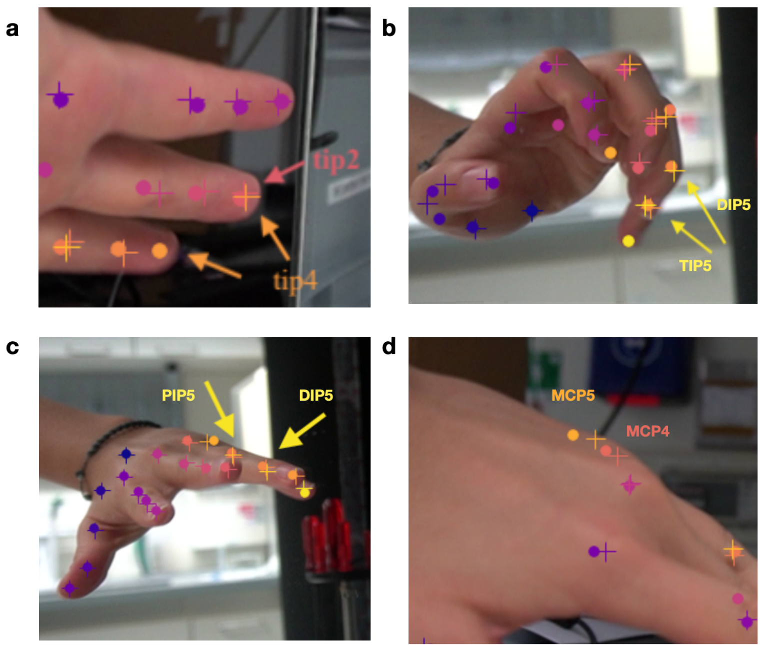
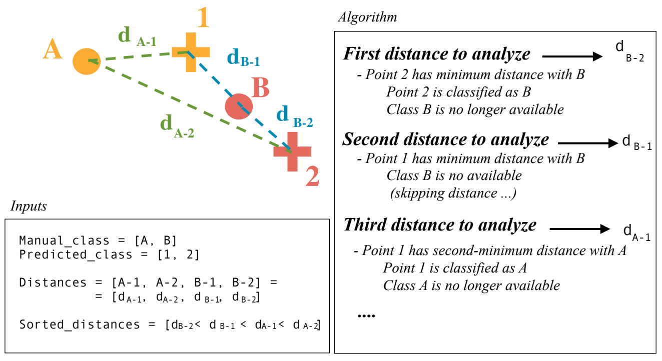


Figure 4.11: a) Wrong prediction even if all the body parts are well visible (*tip4* predicted as tip5), b) Due to the overlap between the two fingers, the network has misdetected the position of labels *DIP5* and *TIP5*, c) *PIP5* and *DIP5* are well predicted, but because the human labels are missing, the minimum distance is automatically measured with *PIP4* and *DIP4*, d) both *MCP4* and *MCP5* are shifted of a bunch of pixels from their real position, but while *MCP4* has the minimum distance with the real itself, *MCP5* has its minimum distance with *MCP4*, otherwise it might have been considered correct.

If we now think how these errors can be solved, we realize that some of them are strictly related to the training of the network and some others are caused by our choices. For the errors coming from the group 1 and 2, they are clearly networks misclassification and they can be solved in different ways. For instance, one could be to act directly on the network and exploit the opportunity that DeepLabCut gives us to refine some frames and re-train the network. In this case, DeepLabCut asks for randomly selecting frames or specifying which one we want to refine. The best choice can be to select and add to the training set directly those frames in which we already know that the network makes mistakes. Another could be to improve the minimum distance criteria by using a *greedy algorithm*, proposal useful also to solve the error coming from group 4. In Figure 4.11d, *MCP5* is classified as *MCP4* because of the distance, and at the same time *MCP4* is correctly classified as itself. This means that two predicted labels are assigned to the same *MCP4* class. By using the greedy algorithm this double assignment can be avoided. This



Inputs

```
Manual_class = [A, B]
Predicted_class = [1, 2]

Distances = [A-1, A-2, B-1, B-2] =
          = [dA-1, dA-2, d B-1, dB-2]

Sorted_distances = [dB-2< d B-1 < dA-1< d A-2]
```

Algorithm

First distance to analyze $\longrightarrow$ d_{B-2}
- *Point 2 has minimum distance with B*
  Point 2 is classified as B
  Class B is no longer available

Second distance to analyze $\longrightarrow$ d_{B-1}
- *Point 1 has minimum distance with B*
  Class B is no available
  (skipping distance ...)

Third distance to analyze $\longrightarrow$ d_{A-1}
- *Point 1 has second-minimum distance with A*
  Point 1 is classified as A
  Class A is no longer available

....

Figure 4.12: Greedy algorithm. In order to resolve the mis-classification caused by the minimum criteria, we propose a greedy algorithm to assign each predicted label to a single class, avoiding multiple assignations

algorithm looks for the locally optimal choice at each stage, and it assigns each prediction to a *single* output class. In order to see how this algorithm may operate, we copy the configuration of Figure 4.11d, and analyze it in Figure 4.12. Once the network returns these labels coordinates, the algorithm would creates two lists: real and predicted class. Then, it calculates the distance that each + has from each ·, and it searches for the minimum distance. It now sorts the distances and, starting from the shortest, it assigns each predicted label to each class. However, at the end of each iteration, it removes from the real list the last assigned class in order to avoid a multiple assignment. The greedy algorithm and the refinement of the network are potential solutions that will be performed in future works.

Although slightly worse, the same consideration can be made for the ResNet-101 test set. The initial hypothesis of the inspection between ResNet-50 and ResNet-101 was that a deeper network might achieve better results then a shallower. Moreover, we have seen that during the training, the deeper network ResNet-101 plateaus faster and to a lower loss value than ResNet-50 (see Figure 4.13). However, it is worth taking into account the duration of the training. The training is performed

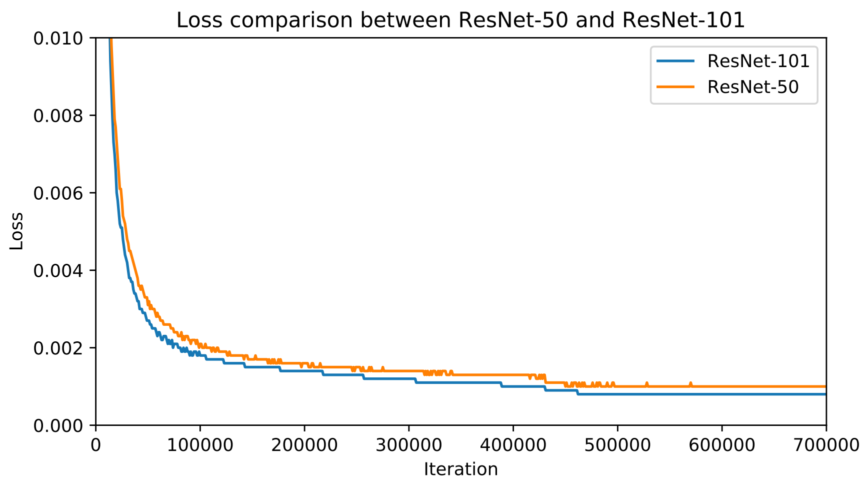
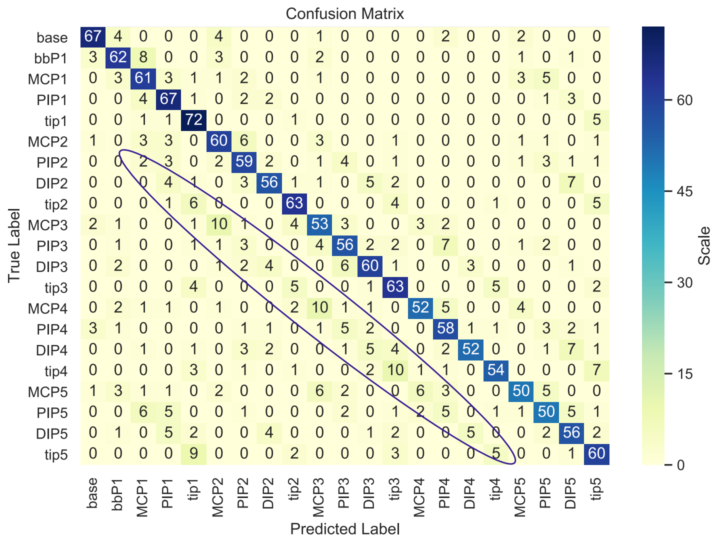
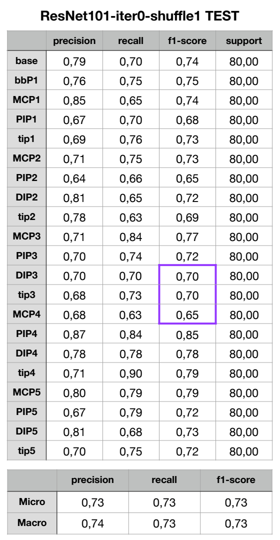


Figure 4.13: Loss function for ResNet-50 and ResNet-101. ResNet-101 plateaus faster than ResNet-50.




ResNet101-iter0-shuffle1 TEST

	precision	recall	f1-score	support
base	0,79	0,70	0,74	80,00
bbP1	0,76	0,75	0,75	80,00
MCP1	0,85	0,65	0,74	80,00
PIP1	0,67	0,70	0,68	80,00
tip1	0,69	0,76	0,73	80,00
MCP2	0,71	0,75	0,73	80,00
PIP2	0,64	0,66	0,65	80,00
DIP2	0,81	0,65	0,72	80,00
tip2	0,78	0,63	0,69	80,00
MCP3	0,71	0,84	0,77	80,00
PIP3	0,70	0,74	0,72	80,00
DIP3	0,70	0,70	0,70	80,00
tip3	0,68	0,73	0,70	80,00
MCP4	0,68	0,63	0,65	80,00
PIP4	0,87	0,84	0,85	80,00
DIP4	0,78	0,78	0,78	80,00
tip4	0,71	0,90	0,79	80,00
MCP5	0,80	0,79	0,79	80,00
PIP5	0,67	0,79	0,72	80,00
DIP5	0,81	0,68	0,73	80,00
tip5	0,70	0,75	0,72	80,00

	precision	recall	f1-score
Micro	0,73	0,73	0,73
Macro	0,74	0,73	0,73

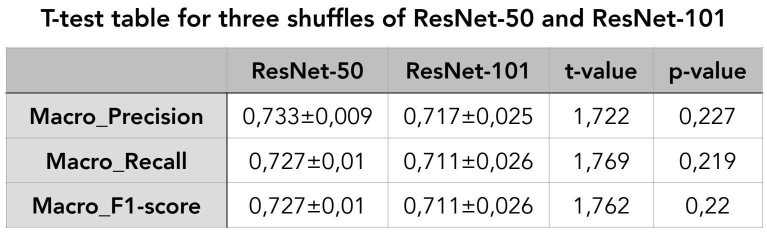
Figure 4.14: Test set confusion matrix and validation metrics of ResNet-101.

using a NVIDIA GeForce RTX 2070 GPU and for 700'000 iterations it requires *1 day* to train the ResNet-50, and *1 day and 16 hours* to train the ResNet-101. From the performances table (Figure 4.14), the classes with the worst values are in some case the same of the ResNet-50. If we look at the same classes of ResNet-50 analysis, *DIP2, tip3, MCP4* seem to be improved, but the overall change cannot be considered a worthy gain. Therefore, if we look at the confusion matrix (see Figure 4.14), the overall behavior is comparable to the previous one. The displaced diagonal is re-identified even it is longer. The first impression may be that exploiting a deeper network is not improving the research result. However, this comparison needs to be further investigated.

In order to asses if the achieved performances are stable, the dataset is shuffled three times before being split into training and test sets, and the networks are trained on each division. In order to measure whether the average score differs significantly across samples, a *t-test* is performed. This test help us to compare two averages. It tells if they are different from each other and how significant the differences are. The t statistic score is calculated as

$$t = \frac{\overline{X}_d}{s_D/\sqrt{n}}$$

where $\overline{X}_d$ and s_D are the average and standard deviation of the differences between all pairs, and n is the number of pairs [20]. In other words, larger is the t-score, more difference there is between samples. The *t-test* is performed for each metric (Precision, Recall, and F1-score). Since we have shuffled the networks three times, we have three samples for each metric (e.g, the *Precision* values of ResNet50 are [$p50_1$, $p50_2$,$p50_3$], which are paired with [$p101_1$, $p101_2$,$p101_3$] of ResNet101). Every t-value is paired with a p-value, that is the probability that the results from our sample occurred by chance. If we observe a *p-value* greater than 0.05 or 0.1 then we cannot reject the null hypothesis of identical average scores. The values obtained are in Table 4.15.



T-test table for three shuffles of ResNet-50 and ResNet-101

	ResNet-50	ResNet-101	t-value	p-value
Macro_Precision	0,733±0,009	0,717±0,025	1,722	0,227
Macro_Recall	0,727±0,01	0,711±0,026	1,769	0,219
Macro_F1-score	0,727±0,01	0,711±0,026	1,762	0,22

Figure 4.15: *T-test* of three samples for each ResNet, repeated for the *Macro-Precision, Macro-Recall* and *Macro-F1-score*

From the *t-values*, we cannot say that the networks are significantly different. A *t-value* of 1.7 means that the groups are 1.7 times different from each other, but to understand if this difference is big enough we look at the *p-values*. The *p-values* are remarkably high to reject the hypothesis that the networks are similar, since we cannot say that the results are occurred by chance. This may be caused by the limited number of samples used for the analysis, and so we should increase the amount of shuffles and train the networks more times. Based on the collected data, we agreed that the benefit of adopting a deeper network is not relevant for our goal to the extent of spending a huge amount of days to train other shuffled data.

Regression validation

In order to evaluate the pixel-based errors of the networks, we make a comparison between test and training set. In Figure 4.16, we show how the ResNet-50 has predicted the hand pose in a training, test, and validation frame. The validation is a set of frame that is neither included in the training set nor in the test set, this is why the manual labels are missing. However, verifying how the model behaves with the validation set is what really matters to understand if the model was well trained. Even if we cannot perform any automatic analysis, from both validation frames (*camA* and *camB*) we can only assess that the model has correctly predicted the *region* labels.

On the contrary, from test and training frame we can measure the accuracy level because we can physically compare predicted and manual labels. To evaluate this accuracy, the **Mean Absolute Error** is calculated pairwise per body part. In statistics, it is a measure of errors between paired observations expressing the same phenomenon or, as in this case, a comparisons of predicted versus observed. To calculate the difference between prediction and observation, we use the Euclidean distance since the data are points in two dimensions. Thus,

$$\mathbf{MAE} = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=0}^{\mathrm{n}-1} \sqrt{(\mathrm{x_i} - \mathrm{x'_i})^2 + (\mathrm{y_i} - \mathrm{y'_i})^2} \tag{4.2}$$

where n is the amount of total labels (21 $\times$ number of frames of the test set), (x_i, y_i) are the manual coordinates, and (x'_i, y'_i) the predicted coordinates (*not a number*

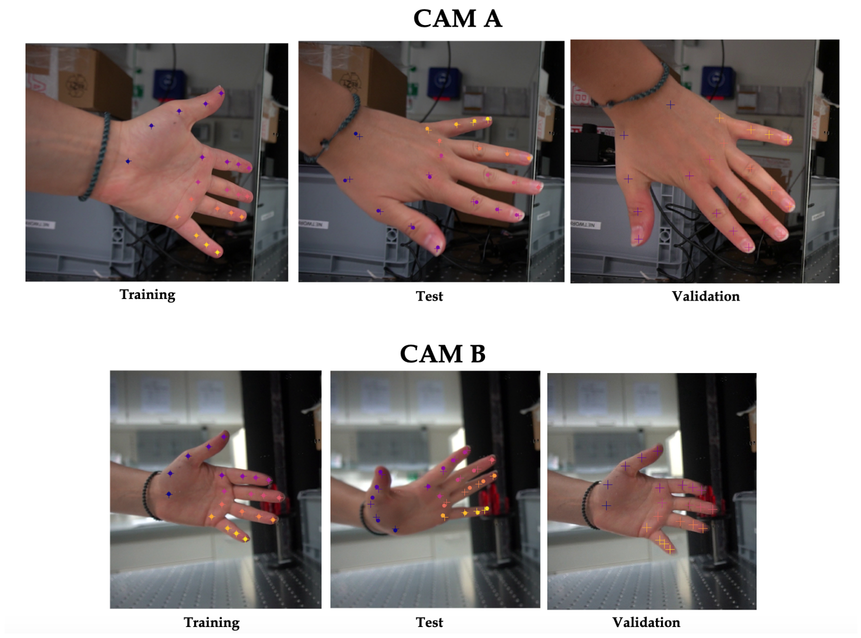


Figure 4.16: Comparison between predictions of training frames and test frames: CAM-A (*above*), CAM-B (*below*)

NaN values are not considered). The MAE is calculated for the three shuffles of the dataset, and in Table 4.3 it is shown their averaged values. Accordingly with Figure 4.16, the pixel-error is more evident in the test set rather than the training set, which demonstrates the efficiency of the model in well-detecting labels used for training the network. Moreover, the standard deviation suggests that the error is more stable in the training set, since it is < 1, rather than in the test set, in which it is > 5. Comparing the two networks, ResNet-101 achieves pixel-error comparable with the ResNet-50 in the training set, even if it is less stable through the shuffles (its standard deviation is equal to 0,22 against the 0,05 of ResNet-50). At the same time, the averaged pixel-error in the test set is greater for the ResNet-101.

	ResNet-50	**ResNet-101**
Training Set	0,90 ± 0,05	0,90 ± 0,22
Test Set	36,11 ± 5,41	42,75 ± 5,51

Table 4.3: MAE averaged on three shuffles for training an test set for ResNet-50 and ResNet-101

We can conclude that, based on classification and regression validation, the overall predictions in terms of region and pixel positions are similar for both networks. However, for our purpose, ResNet-50 achieves acceptable results in less time and with more precision than ResNet-101. For these reasons, further research will be carried out exploiting this architecture.

4.4 Extracting 3D pose by triangulating points with Anipose

The final step of this experiment is to extract the 3D model of the hand skeleton exploiting the 2D predictions coming from DeepLabCut. The 3D tracking toolkit implemented by the authors of Anipose [18] helps us to reconstruct the skeleton, and it consists in the following steps:

1. Estimation of calibration parameters
2. Detection and refinement of the 2D predictions coming from DeepLabCut (both for the real and virtual camera videos)
3. Triangulation and refinement of keypoints to obtain 3D pose estimation.

The advantages of using this toolkit consist in having a good refinement of bad predictions coming from DeepLabCut, and a suitable visualization of the final 3D reconstruction.

Since [18] deals with experiments that require the adoption of more than two cameras, Anipose proposes an initial optimization by performing the calibration of multiple cameras. Furthermore, in our single-camera case, we would have to calibrate only a real and a virtual camera. Then, two approaches might be adopted:

1. calibration based on geometrical features of the mirror
2. calibration proposed by [18].

After having analyzed both calibration approaches, the second one has been chosen. Nevertheless, the first approach might be an interesting evolution for future researches.

4.4.1 Calibration of multiple cameras

In order to perform a 3D reconstruction starting from 2D points detected from different cameras, calibration is a crucial step to matched points between different views. This becomes even more difficult if we use multiple cameras which can affect images with different distortions and optical properties. Therefore, the estimation of the point position in a 3D space may be more accurate if these camera effects are handled and corrected. Calibrate cameras exactly means to determine the intrinsic and the structural parameters of a camera. The model defined in section 1.3.1 is described as

$$\tilde{\mathbf{p}}_{img} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tilde{\mathbf{P}}$$

$$\tilde{\mathbf{p}}_{img} = \mathbf{K}\ [\mathbf{R}|\mathbf{t}]\ \tilde{\mathbf{P}} = \mathbf{K}\ \tilde{\mathbf{p}}_{cam}$$

where $\mathbf{K}$ is the intrinsic matrix and consists in five parameters: f_x and f_y focal length terms, γ is the skew parameter, and c_x and c_y are the offset terms. At this time, we cannot easily approximate these parameters, since we have a multiple camera system, and each camera requires to be well considered. In this paragraph, we analyze the properties of our camera-mirror stereo vision system, and how its reflection properties can help us to relate the two cameras. Afterwards, we describe in a general way how Anipose estimates these parameters, and how they are used to reconstruct the 3D hand.

Calibration in camera-mirror stereo vision system

A traditional stereo vision system is composed by two cameras taking images of a 3D object from different angles. Essentially, the pixels from the right image are matched with pixels of the left one, and the epipolar geometry is used to calculate where the object is located in a 3D space.

As it is shown in Figure 4.17, the right camera in a pinhole camera model (see section 1.3.1) can be described as an object that is translated and rotated with respect to the left camera.

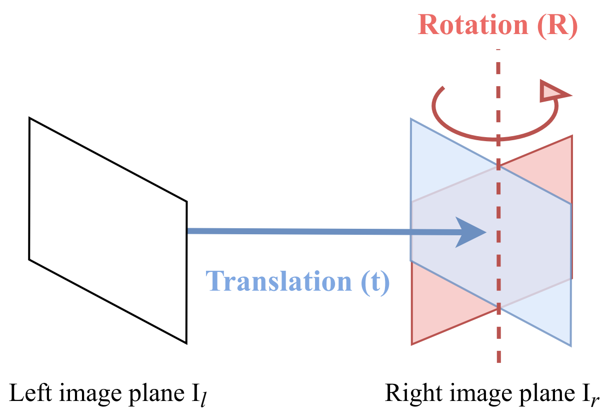


Figure 4.17: Translation and rotation of the right camera with respect to the left camera

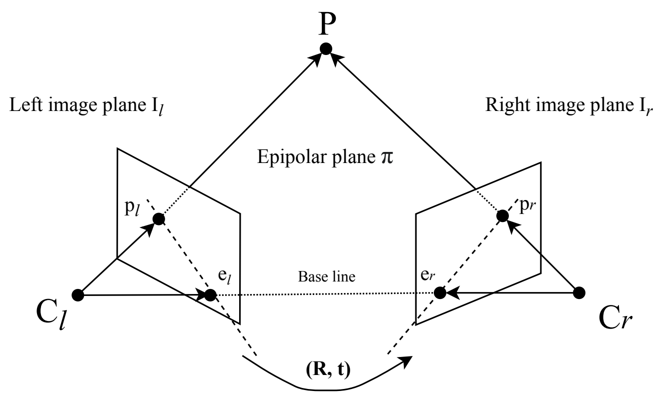


Figure 4.18: Epipolar geometry of two cameras

The resulting system is shown in Figure 4.18. It is called epipolar geometry model, and it determines the geometric relation between two images in a stereo vision system. The 3D point $\mathbf{P}$ has $\mathbf{p}_l$ and $\mathbf{p}_r$ as its 2D projections onto image I_l and image I_r, $\mathbf{C}_l$ and $\mathbf{C}_r$ are the optical centers of left and right cameras, respectively. The projection of the 3D point $\tilde{\mathbf{P}}$ onto a 2D image in homogeneous coordinates is

$$\tilde{\mathbf{p}}_{img} = \mathbf{K}\ [\mathbf{R}|\mathbf{t}]\ \tilde{\mathbf{P}} = \mathbf{K}\ \tilde{\mathbf{p}}_{cam} \tag{4.3}$$

$$\tilde{\mathbf{p}}_{cam} = \mathbf{K}^{-1}\ \tilde{\mathbf{p}}_{img} \tag{4.4}$$

where $\tilde{\mathbf{p}}_{cam}$ is the product between $[\mathbf{R}|\ \mathbf{t}]$ and $\tilde{\mathbf{P}}$, and it represents the transformation of the world coordinates of $\tilde{\mathbf{P}}$ into camera coordinates. Then, if we multiply $\tilde{\mathbf{p}}_{cam}$ to the intrinsic matrix $\mathbf{K}$ we can get the points description in the image plane in terms of pixel, that is $\tilde{\mathbf{p}}_{img}$. However, Equation 4.3 is referred to a single camera system. Based on this model, we want to obtain an equation which describes the relation between the two cameras.

If we go back to the Figure 4.18, $\mathbf{P}$, $\mathbf{C}_l$ and $\mathbf{C}_r$ define the epipolar plane π. Since they are on the same plane, $\overline{\mathbf{C}_\mathrm{l}\mathbf{p}_\mathrm{l}}$, $\overline{\mathbf{C}_\mathrm{r}\mathbf{p}_\mathrm{r}}$ and $\overline{\mathbf{C}_\mathrm{l}\mathbf{C}_\mathrm{r}}$ are three coplanar vectors. To geometrically ensure that three vectors are coplanar, they must satisfy the scalar triple product

$$\overline{\mathbf{C}_\mathrm{l}\mathbf{p}_\mathrm{l}}\ \cdot\ (\overline{\mathbf{C}_\mathrm{l}\mathbf{C}_\mathrm{r}} \times \overline{\mathbf{C}_\mathrm{r}\mathbf{p}_\mathrm{r}}) = 0 \tag{4.5}$$

Considering that $\mathbf{p}_l$ and $\mathbf{p}_r$ are on the same line of the vectors $\overline{\mathbf{C}_l\mathbf{p}_l}$ and $\overline{\mathbf{C}_r\mathbf{p}_r}$, we can reformulate the Equation 4.5 as

$$\tilde{\mathbf{p}}_\mathrm{l} \cdot (\mathbf{t} \times \mathbf{R}\ \tilde{\mathbf{p}}_\mathrm{r}) = 0 \tag{4.6}$$

where $\tilde{\mathbf{p}}_\mathrm{l} = (\mathrm{x}_\mathrm{l},\ \mathrm{y}_\mathrm{l},\ 1)^\mathrm{T}$and $\tilde{\mathbf{p}}_\mathrm{r} = (\mathrm{x}_\mathrm{r},\ \mathrm{y}_\mathrm{r},\ 1)^\mathrm{T}$ are the homogeneous coordinates of the projection of $\tilde{\mathbf{P}}$ with respect to camera $\mathbf{C}_\mathrm{l}$ and $\mathbf{C}_\mathrm{r}$, $\mathbf{t}$ is the translation of camera r with respect to camera l, and $\mathbf{R}\ \tilde{\mathbf{p}}_\mathrm{r}$ is the rotation of camera r with respect to camera l. Moreover, we can reduce the Equation 4.6 in a more compact formula as

$$\tilde{\mathbf{p}}_\mathrm{l}^\mathrm{T}\ [\mathbf{t}]_\mathrm{x}\ \mathbf{R}\ \tilde{\mathbf{p}}_\mathrm{r} = 0 \tag{4.7}$$

$$[\mathbf{t}]_\mathrm{x} = \begin{bmatrix} 0 & -t_3 & t_2 \\ \mathrm{t}_3 & 0 & -t_1 \\ -\mathrm{t}_2 & t_1 & 0 \end{bmatrix}$$

$$\tilde{\mathbf{p}}_\mathrm{l}^\mathrm{T}\ \mathbf{E}\ \ \tilde{\mathbf{p}}_\mathrm{r} = 0 \tag{4.8}$$

where $[\mathbf{t}]_x$ is called the skew symmetric matrix. The equation 4.8 defines the relation between two image planes and so the relation between the image of a point in one camera to the image of the same point in the other camera. $[\mathbf{t}]_x\mathbf{R}$ is also called *essential matrix* $\mathbf{E}$.

Another and more important relation can be obtained if we go back to Equation 4.7. We can assert that $\tilde{\mathbf{p}}_l$ and $\tilde{\mathbf{p}}_r$ are the 2D projections of point $\tilde{\mathbf{P}}$ to each camera ($\tilde{\mathbf{p}}_\mathrm{l} = \tilde{\mathbf{p}}_{\mathrm{cam_l}}$ and $\tilde{\mathbf{p}}_\mathrm{r} = \tilde{\mathbf{p}}_{\mathrm{cam_r}}$). Thus,

$$\tilde{\mathbf{p}}_\mathrm{L} = \mathbf{K}_\mathrm{l}\ \tilde{\mathbf{p}}_\mathrm{l} \Rightarrow \tilde{\mathbf{p}}_\mathrm{l} = \mathbf{K}_\mathrm{l}^{-1}\ \tilde{\mathbf{p}}_\mathrm{L}$$

$$\tilde{\mathbf{p}}_\mathrm{R} = \mathbf{K}_\mathrm{r}\ \tilde{\mathbf{p}}_\mathrm{r} \Rightarrow \tilde{\mathbf{p}}_\mathrm{r} = \mathbf{K}_\mathrm{r}^{-1}\ \tilde{\mathbf{p}}_\mathrm{R}$$

where $\tilde{\mathbf{p}}_L$ and $\tilde{\mathbf{p}}_R$ are the $\tilde{\mathbf{p}}_{img}$ of Equation 4.3. If we replace these two terms in Equation 4.8,

$$(\mathbf{K}_\mathrm{l}^{-1}\ \tilde{\mathbf{p}}_\mathrm{L})^\mathrm{T}\ \mathbf{E}\ (\mathbf{K}_\mathrm{r}^{-1}\ \tilde{\mathbf{p}}_\mathrm{R}) = 0$$

$$\tilde{\mathbf{p}}_\mathrm{L}^\mathrm{T}\ \mathbf{K}_\mathrm{l}^{-\mathrm{T}}\ \mathbf{E}\ \mathbf{K}_\mathrm{r}^{-1}\ \tilde{\mathbf{p}}_\mathrm{R} = 0$$

$$\tilde{\mathbf{p}}_\mathrm{L}^\mathrm{T}\ \mathbf{F}\ \tilde{\mathbf{p}}_\mathrm{R} = 0 \tag{4.9}$$

where $\mathbf{F}$ is the *fundamental matrix* of the two cameras. $\mathbf{F}$ prescribes that for the point $\mathbf{p}_\mathrm{R}$, the analogous point in the left image is located on the corresponding epipolar line. Geometrically, $\mathbf{F}\ \tilde{\mathbf{p}}_\mathrm{R}$ defines the epipolar line of point $\tilde{\mathbf{p}}_L$ on the right image. The intersection between the epipolar line and the base line gives an epipole $\mathbf{e}$. Finally, determining the fundamental matrix $\mathbf{F}$ means defining the intrinsic parameters $\mathbf{K}$ of the two cameras and their structural parameters $[\mathbf{R}|\mathbf{t}]$. Therefore, given $\mathbf{F}$ we can always relate between each other the two cameras.

According to [21], the epipolar constraint model is also applicable for mirror images. However, compared to traditional stereo camera system with two real cameras, the model has different properties since the target point is taken from different fields of view by a single camera. In a general case, the fundamental matrix $\mathbf{F}$ has 7 degrees of freedom, but as [22] shows, in a camera-mirror system can be reduced to 6. Since the real camera and the virtual camera are symmetric, epipolar lines and epipoles are identical in the real and virtual images. Consequently, the

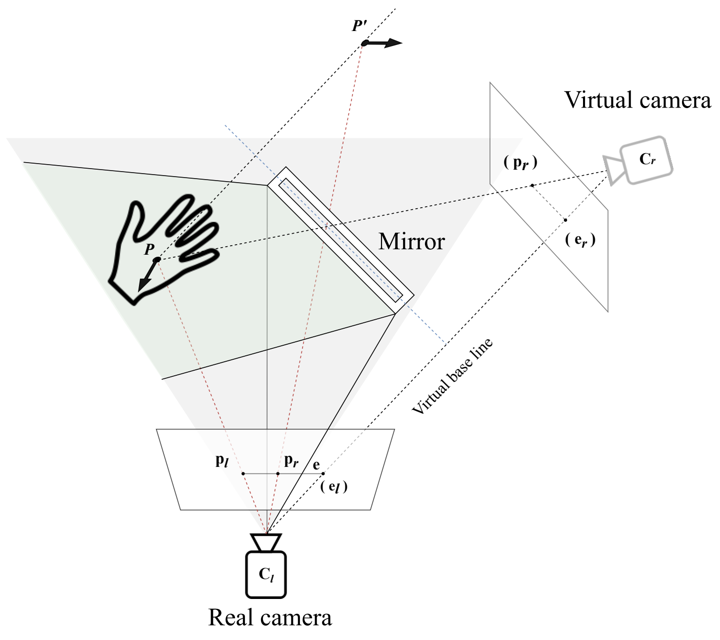


Figure 4.19: Camera-mirror setting

fundamental matrix **F** can be described as

$$\mathbf{F} = \begin{bmatrix} 0 & -e_3 & e_2 \\ \mathrm{e}_3 & 0 & -e_1 \\ -\mathrm{e}_2 & e_1 & 0 \end{bmatrix} \tag{4.10}$$

where $[\mathrm{e}_1, \mathrm{e}_2, \mathrm{e}_3]^\mathrm{T} = \mathbf{e}$ is an epipole between a real and virtual cameras. Therefore, the epipolar points e_l and e_r coincide in the real image at point **e** (see Figure 4.19), due to the existence of only a real camera and the symmetry between the real camera and its reflection.

This geometrical analysis of the camera-mirror system is a potential solution to improve the current calibration method. We have found that with Equation 4.9, we can relate two matched points from the cameras. Since the camera-mirror is a compact and flexible system rather than a multiple real cameras one, we may further investigate these properties, and lighten the calibration.

Calibration performed by Anipose

Anipose performs an automatic calibration of every camera is used for the experiment. This is a great beneficial if we want to perform a reconstruction with more than a mirror. In order to calibrate, the camera model used by [18] introduces the distortion function D, presented in Equation 4.1, and make the full model equal to

$$\tilde{\mathbf{p}} = \mathrm{D}(\mathbf{K}\ [\mathbf{R}|\mathbf{t}]\ \tilde{\mathbf{P}}) \tag{4.11}$$

The Equation 4.11 is a model proposed by [23]. Essentially, [18] reduces the number of parameters setting $\mathrm{f} = \mathrm{f_x} = \mathrm{f_y}$, $\gamma = 0$, and set the camera in order to have $(\mathrm{c_x}, \mathrm{c_y})$ at the center of the image. Similarly, they found that in Equation 4.1, that is

$$\mathrm{D}([\mathrm{x}, \mathrm{y}]) = \begin{bmatrix} \mathrm{x} + \mathrm{x}(\mathrm{k_1}(\mathrm{x}^2 + \mathrm{y}^2) + \mathrm{k_2}(\mathrm{x}^2 + \mathrm{y}^2)^2 + \mathrm{k_3}(\mathrm{x}^2 + \mathrm{y}^2)^4) \\ \mathrm{y} + \mathrm{y}(\mathrm{k_1}(\mathrm{x}^2 + \mathrm{y}^2) + \mathrm{k_2}(\mathrm{x}^2 + \mathrm{y}^2)^2 + \mathrm{k_3}(\mathrm{x}^2 + \mathrm{y}^2)^4) \end{bmatrix}$$

the terms $\mathrm{k_2}$ and $\mathrm{k_3}$ are always small for modern cameras. Therefore, they estimate only 8 parameters per camera: 6 for the extrinsic matrices, 1 for the intrinsic matrices and 1 for the distortion function.

The intrinsic parameter is estimated by using the ChArUco board of paragraph 4.1.1. OpenCV automatically detects the keypoints of the calibration board simultaneously captured from the two cameras and the intrinsic parameter are set based on the board's geometric regularities. Then, to make a valuable calibration they estimate these 8 parameters by implementing a bundle adjustment to minimize the reprojection error. In other words, they take a keypoint of the board, detected by both views, and estimate its 3D position. Then, this 3D point is 2D-projected in both cameras planes and they quantify how closely this re-projections match the respective keypoints of the board. The calculated error is iterated until a minimum error is obtained (for details, see [18] p.13-14).

4.4.2 3D skeleton

We finally triangulate the 2D predictions to obtain a preliminary 3D model of the moving hand (see Figure 4.20).

Triangulation is the process of determining the position of a point in a 3D space

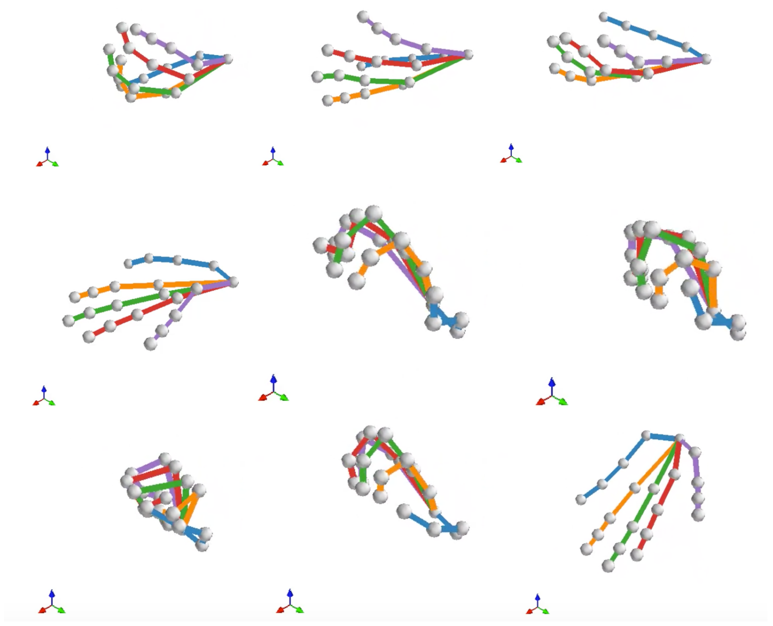


Figure 4.20: Frames of the 3D reconstruction of the moving hand in time sequence

by generating a triangle to it from two known points. Thanks to the filtering operation of Anipose [18], the hand movements are smoothed in each frame and the overall poses are most of the times coherent with the 2D videos. However, this filtering causes some poses not to be very accurate especially when the hand changes its shape rapidly.

In order to evaluate how good is the 3D prediction, we physically measured the distances between points while the hand is completely fully supported by a flat surface. Through *Fiji - ImageJ* we physically measure the distance between each label previously shown in Figure 4.8. In order to have a comparable unit, the obtained distances are normalized with respect to the longest segment of the skeleton. After, we calculate the segment length reconstructed by the 3D predictions for each frame, and we normalize each row with respect to the same length measured. At this point, we generate two complementary graphs. In Figure 4.21 is shown the averaged length of each segment compared to the ground truth. The overall predictions suggest that the 3D model follows the ground truth behaviour. The inaccuracy seems to occur regularly:

- The predicted segments *MCPx-PIPx* are longer than the real one

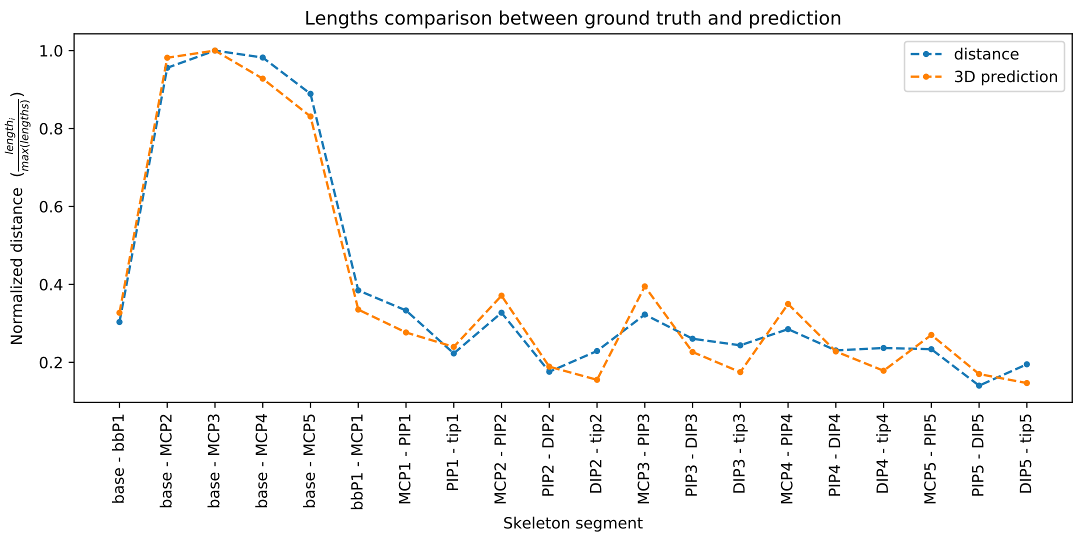


Figure 4.21: Lengths of the skeleton, comparison between ground truth and averaged predicions

- The predicted segments *DIPx-tipx* are shorter than the real one.

As well as, the length of *PPIx-DIPx* is predicted more accurately with the same regularity. What can really affects the length of the skeleton segments is manually labeling both views. Since the reflection appears far from the camera with respect to the real view, the points are difficult to be labeled with high accuracy, therefore they are to be considered as approximated positions. An interesting test to do would be to automatically label the second view points by exploiting the SIFT algorithm described in section 2.1, and see to what extent can we match points with similar descriptors. However, if we look at Figure 4.22, the MAE (see section 4.3.1) is constantly < 0.1 for every segments. Therefore, the overall error is low enough to say that the pose estimation is consistence.

Even this is a preliminary result, the camera-mirror stereo system can be considered a potential solution to perform total body pose estimation. The 3D model reflects the reality, predicts the geometrical relation between components, and if the body rotates, it correctly suggests which part is in front and which one is behind, missing information in a monocular video. We want to repeat this experiment for the whole mouse, and obtain a 3D skeleton of every part, deformable

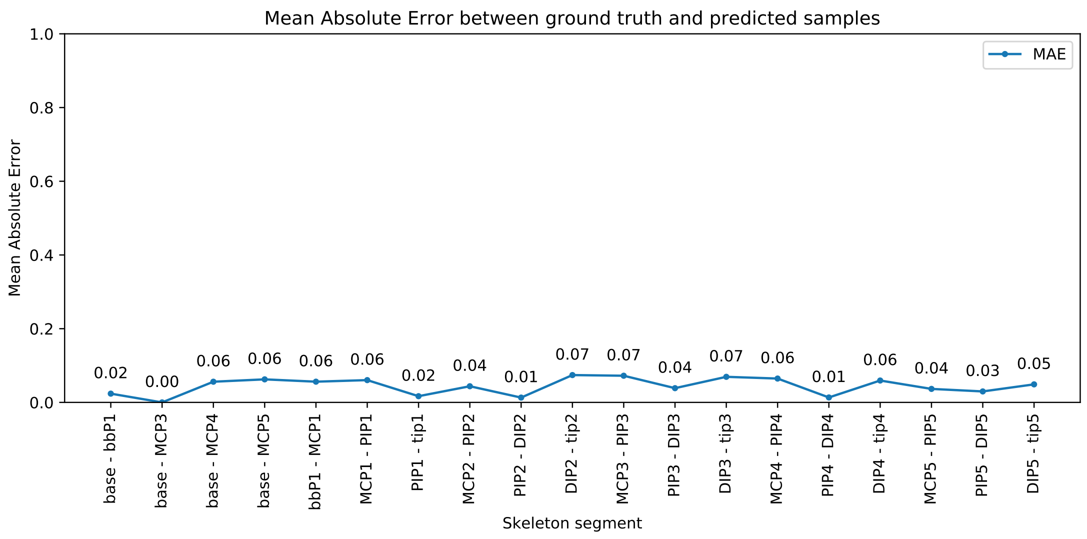


Figure 4.22: Men absolute Error calculated for each segment of the skeleton

and non-deformable. It may be a starting point to a total body reconstruction, and through photogrammetry steps we may correct the results obtained in section 2.2.3, get a dense reconstruction, and a texture with the vascular structure. Otherwise, we may start again and directly label precise vessels that we can see through SWIR imaging. Obtaining a 3D structure of the vessels network may allow us to first see how deep and accurate can be their tracking (e.g, can we track capillaries, or only larger vessels can be detected?), we may try to evaluate the symmetry between right and left side, or compare it between different mice. Nevertheless, we deem worth investigating how the algorithm works if applied to a dataset acquired through SWIR technology.

Chapter 5

Conclusion and future experiment

Examining animals without affecting their natural physiology has always been of great interest in the field of laboratory research. Since animals hardly cooperate, scientists are often led to restrain or anaesthetize them, causing the acquisition of data often being compromised by their doped and stressed state. In order to bypass these conditions, we investigate how to obtain a 3D reconstruction of the examined mouse while it is moving freely in its environment. Moreover, the acquisition is made by using a new imaging technique, called SWIR, which allows us to see through the mouse skin.

The purpose of the first experiment was to investigate the existing reconstruction technique: photogrammetry. We have tried to figure out how its algorithm might help us to image the mouse fluorescent body. Since the SWIR technology allows us to see the mouse vessels, we have looked for a 3D object with a *SWIR texture*. We wanted to obtain a mouse surface in three-dimensions which we would use to follow the vascular network. We have found that photogrammetry easily reconstructs rigid objects, but the results change when the subject is moving or changing its shape. Regarding static objects, we have reconstructed a mouse video, in which vessels and a tumor where well-visible thanks to an NIR imaging. So, we definitely can obtain a texture with vessels information in three-dimensions which may be tracked and analyzed. Moreover, what we have learnt from this experiment is that we can reconstruct whether the camera rotates around the object, or the object rotates around itself as the camera takes photographs. This has been helpful since we really start to understand how camera calibration works. Regarding moving objects, we have obtained a preliminary texture of the fluorescent body of a mouse freely to move, but during the reconstruction some errors occurred. One was the multiple reconstruction of the same part of the body, since the movement might confuse the algorithm.

As a motion compensation, a second experiment has been conducted. We wanted to understand if by tracking a specific and rigid body part, we would have been capable of extracting its orientation in 3D space over time. We have used a deep learning technique to track in each frame of the video the 2D direction of the mouse skull. After, we have used the camera information combined with the absolute coordinates of the skull to estimate the third dimension. We have found that, based on the 2D predictions, we can obtain a 3D skeleton of the skull while is taking different orientations on the whole video with minimum error. We have learnt how to manipulate the camera calibration to follow the object we are recording, and to extract three dimensional information from that. Moreover, we analyzed the architecture of the neural network and its performances in estimating the 2D skull positions. We have used DeepLabCut tool whose predicted labels were excellent if compared with the manual ones, and this is decisive since they certainly affect the 3D predictions. We were excited with this experiment, as the results supported our expectations, and excellent outcomes have been obtained using only a monocular video. For this reason, we have decided to extend the reconstruction to the whole body, since until now we have been focused on tracking just the skull.

In the third and last experiment, our goal was to find a way to track the whole body of an object, whether it was rigid or deformed. In order to extract the third dimension of the body without knowing a prior 3D model, we had to perform a stereo vision system which consists in recording the object from two different points of view. This has been necessary in order to triangulate corresponding points of two images and estimating their positions in a 3D space. This has been feasible also because we have already learnt how to calibrate one camera, even if the concept may appear different with two cameras. The first thing that we have learnt from this experiment is that we can calibrate two cameras, since one could be considered as rotated and translated with respect to the first one. The second important thing has been the adoption of a mirror to avoid a second camera. In fact, we wanted the system to be as simple as possible, keeping a monocular system. The adoption of a mirror has allowed us to record an object, a moving hand, from to different points of view: the real object and its reflection. We have used again DeepLabCut to estimate the points positions in both views, but this time we have analyzed two different neural network architectures. This comparison has been done in order to understand if a deeper network could have provided more accurate results. Since a deeper network did not improve the initial performances, after a careful analysis we have agreed that a simpler system was convenient for the purposes of our research. After obtaining the same labeled points for both real and virtual cameras, we have triangulated each matched points to obtain a 3D skeleton of the hand in motion. The result was consistent and reflected the reality, since the hand skeleton followed the movements recorded in the monocular video.

Therefore, fundamental findings arise from each experiment, and each of these approaches can bring something useful to the others. Future research may follow in order to improve some features:

1. **Automatic labeling**: since the accuracy of the predictions strongly depends on the number of labeled frames used in the training set, we want to find a way to automatize this process, and make it faster. One could be using particular filters to pre-process the image. For example, if we are interested in labeling points of vessels, we can filter in order to obtain only those white pixels that for sure belongs to the vessels. This kind of image-processing would require the adoption of strict constraints in the algorithm, since we cannot accept ambiguous points.

2. **Finding matched points**: instead of manually labeling the second camera, we may investigate other ways to find the matched points in the second view. One could be applying what we have learnt with the photogrammetry pipeline, and so using the SIFT algorithm to match points in the two views. If the match is correct, it would be definitely more consistent than the manual one, since in the SIFT algorithm we use statistical descriptors to find matches. Another way would be to exploit the properties of the mirror: we know that each point in the real view has its symmetrical correspondent point in the reflection.

3. **Simplify the calibration**: as introduced in section 4.4.1, the adoption of

a mirror may simplify the calibration method. We already know how the two cameras are related to each other, since one is the reflection of the other, therefore a point in the real image has its correspondence exactly in symmetry.

4. **Combining tracking and photogrammetry**: thanks to the tracking method described in our experiments, what we can obtain is a skeleton of the object of interest. Since we are interested in studying things such as the changes of the vessels over time, it could be worth having information about the vessel sizes (e.g, diameters). We want to investigate if we can obtain a dense reconstruction, and so a texture, around our skeleton segments whose positions in a 3D space are already known. In other words, we want to exploit photogrammetry algorithm to physically build pieces of texture around our 3D skeleton. Otherwise, we can think to quantify vessels sizes by associating a proportional weight to each arch of the skeleton.

These are only a few things that we have listed while performing the experiments, and of course there are many solutions to be tested. If the improvement is well-addressed, we can think of performing the mirror-camera system applied on SWIR imaging, and then see what kind of images we are capable of acquiring by introducing a mirror.

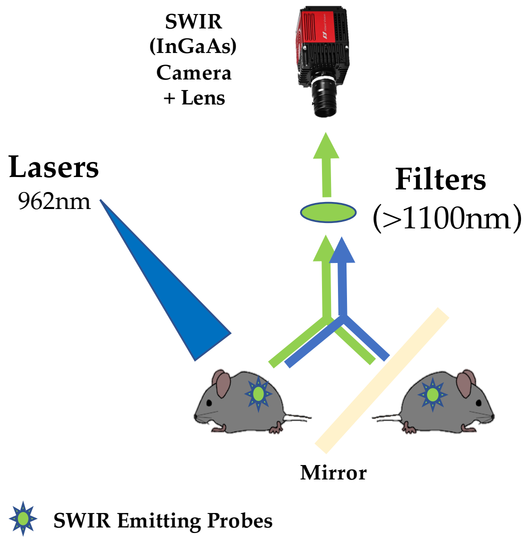


Figure 5.1: Camera-mirror setting combined with SWIR imaging.

The camera system we are referring to will be something similar to the one in Figure 5.1. Exciting the probes with a laser, as described in section 1.1, the InGaAs camera will acquire a monocular video whit both real and reflection views of the mouse. We will try to repeat the process of the third experiment by tracking selected points from both views and triangulate them for the whole body reconstruction. Moreover, we will look for alternative technologies, in order to understand if we are using the right methods for reconstructing. For example, we may try to use a marker technology that can solve the tracking of a body part, and see how the reconstruction changes. Or we can totally change the reconstruction method, and look for other techniques. Another one could be using the illumination to detect the 3D shape of an object by detecting the curvatures of the structured light [24], or improving the stereo system by performing a stereo sensing system and exploiting *neural depth refinement* [25] on SWIR imaging.

In conclusion, the results obtained in this thesis are potential starting points to achieve an efficient and innovative reconstruction of animals body, without being forced to anaesthetize them.

Bibliography

[1] Oliver T. Bruns et al. «Next-generation in vivo optical imaging with short-wave infrared quantum dots». In: *Nature Biomedical Engineering* 1.4 (Apr. 2017), p. 0056. ISSN: 2157-846X. DOI: `10.1038/s41551-017-0056`. URL: `https://doi.org/10.1038/s41551-017-0056` (cit. on pp. 3–5).

[2] Wikipedia contributors. *Azure Kinect — Wikipedia, The Free Encyclopedia.* [Online; accessed 23-July-2020]. 2020. URL: `https://en.wikipedia.org/w/index.php?title=Azure_Kinect&oldid=945489630` (cit. on pp. 6, 7).

[3] Alexander Mathis, Pranav Mamidanna, Kevin M. Cury, Taiga Abe, Venkatesh N. Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. «DeepLabCut: markerless pose estimation of user-defined body parts with deep learning». In: *Nature Neuroscience* 21.9 (Sept. 2018), pp. 1281–1289. ISSN: 1546-1726. DOI: `10.1038/s41593-018-0209-y`. URL: `https://doi.org/10.1038/s41593-018-0209-y` (cit. on pp. 7, 11, 16, 40, 41).

[4] Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. «DeeperCut: A Deeper, Stronger, and Faster Multi-person Pose Estimation Model». In: *Lecture Notes in Computer Science* (2016), pp. 34–50. ISSN: 1611-3349. DOI: `10.1007/978-3-319-46466-4_3`. URL: `http://dx.doi.org/10.1007/978-3-319-46466-4_3` (cit. on p. 7).

[5] Omid Haji Maghsoudi, A. Vahedipour Tabrizi, B. Robertson, and Andrew Spence. «Superpixels based marker tracking vs. hue thresholding in rodent biomechanics application». In: *2017 51st Asilomar Conference on Signals, Systems, and Computers* (Oct. 2017). DOI: `10.1109/acssc.2017.8335168`. URL: `http://dx.doi.org/10.1109/acssc.2017.8335168` (cit. on p. 8).

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition.* 2015. arXiv: `1512.03385 [cs.CV]` (cit. on pp. 12, 13).

[7] Deng Lu; Chu Hong-Hu; Shi Peng; Wang Wei; Kong Xuan. «Region-Based CNN Method with Deformable Modules for Visually Classifying Concrete Cracks». In: (2020). URL: `https://www.mdpi.com/2076-3417/10/7/2528#cite` (cit. on p. 14).

[8] Chen Cao, Derek Bradley, Kun Zhou, and Thabo Beeler. «Real-Time High-Fidelity Facial Performance Capture». In: *ACM Transactions on Graphics* 34 (July 2015), 46:1–46:9. DOI: 10.1145/2766943 (cit. on pp. 16, 17).

[9] 3D scan store. URL: https://www.3dscanstore.com/about-us (cit. on p. 17).

[10] AliceVision. *Meshroom: A 3D reconstruction software.* 2018. URL: https://github.com/alicevision/meshroom (cit. on pp. 23, 24, 30).

[11] Dr. Mubarak Shah. *Computer Vision Fall 2012 - Lecture 5, SIFT feature matching.* 2012 (cit. on p. 24).

[12] GettyImages. *Here's How to Watch a Full Moon, Lunar Eclipse and Comet Light Up the Sky on Friday.* URL: https://time.com/4666332/full-moon-lunar-eclipse-comet-watch/ (cit. on p. 27).

[13] Medium. *Introduction to SIFT(Scale Invariant Feature Transform).* 2019. URL: https://medium.com/data-breach/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40 (cit. on p. 28).

[14] Yeteng Zhong et al. «In vivo molecular imaging for immunotherapy using ultra-bright near-infrared-IIb rare-earth nanoparticles». In: *Nature Biotechnology* 37.11 (Nov. 2019), pp. 1322–1331. ISSN: 1546-1696. DOI: 10.1038/s41587-019-0262-4. URL: https://doi.org/10.1038/s41587-019-0262-4 (cit. on p. 32).

[15] Edmund Optics GmbH Germany. *What is SWIR?* URL: https://www.edmundoptics.eu/knowledge-center/application-notes/imaging/what-is-swir/ (cit. on p. 37).

[16] Susan Blackford. *How to Measure Errors.* Oct. 1999. URL: https://www.netlib.org/lapack/lug/node75.html (cit. on p. 51).

[17] Torquet N. Chaumont F. Ey E. «Real-time analysis of the behaviour of groups of mice via a depth-sensing camera and machine learning». In: *Nat Biomed Eng* (May 2019). DOI: https://doi.org/10.1038/s41551-019-0396-1 (cit. on p. 55).

[18] Pierre Karashchuk, Katie L. Rupp, Evyn S. Dickinson, Elischa Sanders, Eiman Azim, Bingni W. Brunton, and John C. Tuthill. «Anipose: a toolkit for robust markerless 3D pose estimation». In: *bioRxiv* (2020). DOI: 10.1101/2020.05.26.117325. eprint: https://www.biorxiv.org/content/early/2020/05/29/2020.05.26.117325.full.pdf. URL: https://www.biorxiv.org/content/early/2020/05/29/2020.05.26.117325 (cit. on pp. 56, 73, 79, 80).

[19] Open Source Computer Vision. URL: https://docs.opencv.org/3.4/df/d4a/tutorial_charuco_detection.html (cit. on p. 58).

[20] The SciPy community. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html (cit. on p. 70).

[21] Xinghua Chai, Fuqiang Zhou, and Xin Chen. «Epipolar constraint of single-camera mirror binocular stereo vision systems». In: *Optical Engineering* 56 (Aug. 2017), p. 1. DOI: 10.1117/1.OE.56.8.084103 (cit. on p. 77).

[22] Joshua Gluckman and S. Nayar. «Planar catadioptric stereo: geometry and calibration». In: *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)* 1 (1999), 22–28 Vol. 1 (cit. on p. 77).

[23] Zhengyou Zhang. «A Flexible New Technique for Camera Calibration». In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22 (Dec. 2000), pp. 1330–1334. DOI: 10.1109/34.888718 (cit. on p. 79).

[24] Ryo Furukawa, Ryusuke Sagawa, and Hiroshi Kawasaki. *Depth estimation using structured light flow – analysis of projected pattern flow on an object's surface –*. 2017. arXiv: 1710.00513 [cs.CV] (cit. on p. 87).

[25] uDepth Software Lead Michael Schoenberg and Adarsh Kowdle. *uDepth: Real-time 3D Depth Sensing on the Pixel 4*. 2020. URL: https://ai.googleblog.com/2020/04/udepth-real-time-3d-depth-sensing-on.html (cit. on p. 87).