POLITECNICO DI TORINO

Master's Degree in Mechatronics Engineering

Master's Degree Thesis

Body pose estimation in sport science based on sensor fusion algorithm of multiple RGB cameras

Supervisors

Prof. Marcello CHIABERGE

Prof. Gentiane VENTURE

Prof. Vincent HERNANDEZ

Candidate

Amir GAMAHDRID

December 2023

A nonna Elvira.

Abstract

The computer vision field includes tasks like image classification, object recognition, and feature detection. The focus of this work is body pose estimation, where the human motion is analyzed using human activity recognition algorithms through pattern detection principles.
The increasing popularity of cost-effective mobile sensors like Microsoft Kinect has led to the development of various algorithms for activity recognition and tools that enable sport performance analysis and motion rehabilitation at home. These algorithms have the potential to promote a healthy lifestyle, discourage unhealthy habits, and aid in condition tracking, particularly in sports science and healthcare applications.Therefore, in this work we will use data collected with RGB cameras to detect and classify sports movements and exercises involving both the upper and the lower body.
The first step of this work is camera calibration, an essential prerequisite in the world of 3D computer vision, performed using the OpenCV library in Python and a checkerboard. The body pose of the subjects will be estimated using the MediaPipe framework offered by Google, obtaining the skeleton of the subject seen from different orientation. Afterwards, a set of ArUco markers will be used to estimate the pose and position of the cameras with respect to a fixed reference system, which will be used to rotate the 3D joints positions of the skeleton into the same reference system, in order then to fuse the data and obtain a more accurate and robust estimation of the body pose.
Four different fusion methods will be exploited: mean fusion, Kalman filter fusion, mean fusion + DBSCAN and Kalman filter + DBSCAN.
The fusion methods will be then evaluated and compared using the motion capture system developed by OptiTrack as groundtruth ($Prime^X 13$ cameras and Motive software).
Finally, using the data coming from the most accurate fusion algorithm, the joint angles will be computed in order to build a dataset of 24 exercises performed by 4 subjects. The dataset will be then used to train, validate and test a Random forest classifier and a Multi-layer perceptron classifier.
The results show a good improvement of the performances when using the data coming from the proposed sensor fusion method instead of the single cameras, resulting in a satisfactory accuracy in the classification of the exercises and providing valuable insights for further advancements, possibly with the integration of Inertial Measurement Units (IMUs), that could offer a promising enhancement by providing information less susceptible to visual occlusions and better real-time adaptation to dynamic movements.

Acknowledgements

I extend my sincere gratitude to the GV lab at the University of Tokyo. First of all, I would like to thank professor Gentiane Venture, who accepted me as exchange student in her lab and gave me the amazing opportunity to conduct this work in such an innovative and stimulant environment.

Not less important, I am deeply indebted to professor Vincent Hernandez for his constant support and his trust in me. His expertise and collaboration have greatly enriched the outcomes of this research, helping me from the beginning to the end of this run with wise advice and insightful feedback day by day.

I would like to extend my gratitude to professor Marcello Chiaberge for giving me the chance to conduct this work free from worries and constituting the fundamental bridge between Politecnico di Torino and the University of Tokyo.

This thesis symbolizes the culmination of my academic voyage over the last two years, spanning from Italy to Japan. During this time, I've experienced substantial growth both in terms of academics and mostly personally, shaping the individual I have become today.
I want to express my sincere gratitude to everyone who took part to my journey, starting from all the people that have always been by my side to the ones that I got to know during these last years and hopefully will be next to me for a long time.
Lastly, but most importantly, I want to express my infinite gratitude to my mother, for giving me all the means necessary to reach my goals by making me the person I am today, through unconditional and endless love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This introduction will briefly present the motivations, main objectives and implementations of the work conducted during my Erasmus+ project, in collaboration with the mechanical engineering department of the University of Tokyo and more specifically the GV Lab, under the supervision of prof. Gentiane Venture and prof. Vincent Hernandez.
The initial sections present the reasons behind addressing this subject, along with an overview of the principles and techniques applied. Subsequently, the second part outlines the arrangement and layout of the thesis.

1.1 Motivations

Computer Vision involves interpreting image content, including image classification, object recognition, and feature detection like cancer identification in biomedical images.
This work focuses on pose estimation, crucial in several computer vision applications like robot guidance, augmented reality and sport analysis. Human Activity Recognition (HAR) categorizes human motions based on pattern recognition principles, often using machine learning techniques: this procedure revolves around identifying connections between points within the physical world and their two-dimensional image projections.
Cost-effective, mobile sensors like Microsoft Kinect, known for tracking skeletal joints, have gained popularity in recent times, leading to various algorithms for activity recognition and tools that can help to perform sport performance analysis and/or motion rehabilitation at home.
Utilizing data collected from RGB cameras, we will focus on detecting sports movements and classifying various typical exercises related to both upper and lower body, such as shoulder press, biceps curl, lunges, squats etc. Applying these

algorithms in sports science and healthcare can significantly promote a healthy lifestyle, deter unhealthy habits, and facilitate condition tracking.

1.2 Chapters organization

First, camera calibration is an essential prerequisite in the world of 3D computer vision: the accuracy of the detection in fact may be significantly reduced because of the some distortion of the images introduced by the camera itself. The topic of camera calibration will be discussed in *Chapter 2*.

After calibrating multiple RGB cameras using the OpenCV library in Python and a checkerboard, we used a set of ArUco markers to estimate the position of the cameras with respect to a fixed reference system and then estimated the body pose using the MediaPipe framework.
Therefore, in *Chapter 3* we will discuss the estimation analyzing two phases:

- *Camera pose estimation*: A popular approach involves using binary square markers known as ArUco markers. These markers are advantageous because they provide multiple connection points due to their four corners, allowing for pose estimation with just one marker. Furthermore, their internal binary encoding makes them robust and enables the use of error detection and correction methods.

- *Body pose estimation*: The task will be completed using MediaPipe, an open-source framework from Google. MediaPipe provides pre-built, customizable machine learning pipelines for a range of multimedia tasks, such as face detection, hand tracking, pose estimation, and 3D object detection. It simplifies the development of multimedia applications by offering pre-trained models and modules. MediaPipe is written in C++ but supports multiple programming languages, including Python and Java, making it versatile and accessible across various development environments.

After having collected the 3D joints positions from multiple cameras, four sensor fusion algorithm will be deployed in *Chapter 4*:

- Mean
- Mean + DBSCAN
- Kalman filter
- Kalman filter + DBSCAN

Sensor fusion is the process of integrating data from various sensors like cameras, radar or lidar to gain a more precise and complete understanding of a physical situation. The primary objective is to enhance accuracy, ensure redundancy, and boost robustness. By combining information from multiple sensors, errors and inaccuracies associated with any single sensor can be minimized. Additionally, if one sensor malfunctions or provides incorrect data, the system can still operate effectively by relying on data from other sensors, making it more resilient in challenging conditions.

The four sensor fusion algorithm will be compared using a motion capture system as groundtruth. For this reason, in *Chapter 5* an overview of such technologies will be given.
A motion capture system, commonly referred to as "mocap" or "MoCap," represents a technology employed to digitally record and apprehend the motions of objects or living entities. It is primarily utilized for the analysis and recreation of these movements in various applications, including computer-generated imagery (CGI), animation, biomechanical research, sports analysis, and more. In recent years, optical motion capture has gained significant prominence, especially in applications related to body movement animation.
In our case, $Prime^X 13$ cameras by OptiTrack have been used to collect the data, processed then using the software Motive, again by OptiTrack.

In *Chapter 6* we will introduce the Random Forest and the Multi-layer Perceptron classifiers. Using the data coming from the most accurate fusion algorithm, the two classifier will be used to perform classification tasks using a dataset composed of the joint angles during 24 exercises performed by 4 subjects.

The experimental setup for the sensor fusion algorithms and the evaluation through the Mocap system will be described in *Chapter 7*. Moreover we will define the dataset used to train, validate and test the models of the two classifiers. In this chapter, also the results of both the evaluation of the sensor fusion algorithms and the classification accuracy will be presented.

Discussion and conclusion will be finally given in *Chapter 8*.

Chapter 2

Camera calibration

2.1 Overview

Every time we are dealing with cameras and more generally speaking with sensors, a key step to perform is calibration. More specifically, camera calibration is an essential prerequisite in the world of 3D computer vision: the accuracy of the detection in fact may be significantly reduced because of the some distortion of the images introduced by the camera itself. We can distinguish two major kinds of distortion:

- Radial distortion, which causes straight lines to appear curved and becomes larger the farther the points are from the center of the image.
- Tangential distortion, which occurs when the image-taking lens is not aligned perfectly parallel to the imaging plane.

Figure 2.1: Radial and tangential distortion

To eliminate such effects, calibration of the camera is essential. The methods in which a camera can be calibrated can be broadly categorized into two macro-groups: photogrammetric calibration and self-calibration, as reported by [36].

Photogrammetric calibration entails the meticulous observation of a calibration object with a known, highly accurate 3D spatial geometry. Typically, the calibration object comprises two or three mutually perpendicular planes. These approaches necessitate the use of costly calibration equipment and an intricate setup, but the process can be executed with remarkable efficiency.

On the other hand, self-calibration techniques operate without the need for any dedicated calibration object. Simply by moving the camera within a stationary scene, the inherent rigidity of the scene generally imposes two constraints on the camera's internal parameters, relying solely on image data. Consequently, if images are captured using the same camera with fixed internal parameters, correspondences between three images suffice to recover both the internal and external parameters, enabling the reconstruction of the 3D structure with a similarity. Although this approach offers great flexibility, due to the multitude of parameters that need estimation, obtaining consistently reliable results is not always feasible.

For this reason, in this work the photogrammetric approach has been exploited.
The goal of the calibration of cameras is to determine all the characteristics of the cameras themselves: this involves possessing all the necessary parameters concerning the camera, essential for establishing a precise connection between a 3D point in the physical environment and its corresponding 2D representation (pixel) in the image taken by the calibrated camera.
Typically, there are two parameters of interest:

1. Intrinsic parameters: parameters concerning the camera arrangement, such as focal length, optical center, and radial distortion coefficients of the lens.

2. Extrinsic parameters: parameters related to the camera's orientation (rotation and translation) concerning a specific world coordinate system.

In order to extract these parameters, we will analyze in the following sections how the physical world, the camera and the image are related to each other, following the documentation provided by OpenCV at [2] and guidelines provided by [27].

2.2 World coordinate system

To define locations of points in the room we need to first define a world coordinate system.
Thus, we need to define the origin by fixing a corner of the room as (0, 0, 0) and the X, Y, Z axes by defining the X and Y axis of the room along the two dimensions on the floor and the Z axis along the vertical wall.
Using the above, we can find the 3D coordinates of any point in this room by measuring its distance from the origin along the X, Y, and Z axes. In the world coordinate system, the coordinates of P are given by (X_w, Y_w, Z_w), as shown in *Figure 2.2*.

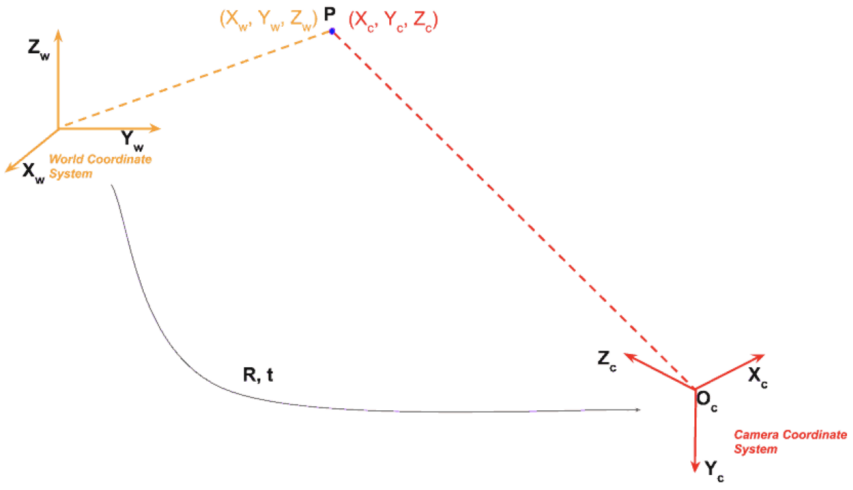

Figure 2.2: Coordinates transformation to camera coordinates

2.3 Camera coordinate system

After having defined the world coordinate system, the next step is to find the relationship between the 3D room (i.e. world) coordinates and the 3D camera coordinates.

Let's say our camera is located at some arbitrary location (t_X, t_Y, t_Z) in the room. The camera may be also looking in some arbitrary direction, that is we can say the camera is rotated with respect to the world coordinate system. The relative rotation of the camera can be expressed through a simple 3x3 rotation matrix R.

The two coordinate values are related by the following equation.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + t \tag{2.1}$$

The above relation can also be expressed in homogeneous coordinates by adding an extra dimension, giving the following representation:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = T \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{2.2}$$
$$T = \begin{bmatrix} R & | & t \end{bmatrix}$$

2.4 Image coordinate system

Once we get a point in 3D coordinate system of the camera by applying a rotation and translation to the points world coordinates, we are in the position to project the point on the image plane, in order to obtain a location of the point in the image, as shown in *Figure 2.3*.

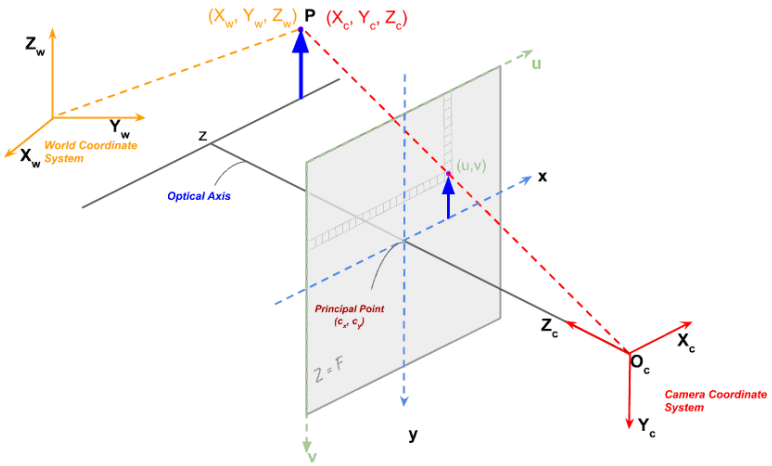

Figure 2.3: Coordinates transformation to image coordinates

The optical center (pin hole) is represented using O_c and the image plane is placed at a distance f (focal length) from it (in reality an inverted image of the point is

formed on the image plane). It can be shown that the project image (x, y) of the 3D point (X_c, Y_c, Z_c) is given by

$$x = f\frac{x_c}{z_c} y = f\frac{y_c}{z_c} \tag{2.3}$$

The above two equations can be rewritten in matrix form as follows

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \tag{2.4}$$

The matrix K shown above is called the *Intrinsic Matrix* and contains the intrinsic parameters of the camera.

The above simple matrix shows only the focal length (in this work f_x and f_y are considered to be equal and the skew γ is considered to be equal to zero).
However, the pixels in the image sensor may not be square, and so we may have two different focal lengths f_x and f_y, and the optical center (c_x, c_y) of the camera may not coincide with the center of the image coordinate system. In addition, there may be a small skew γ between the x and y axes of the camera sensor.
The matrix K thus is more generally an upper triangular matrix composed by:

- (f_x, f_y): focal lengths
- (c_x, c_y): optical center
- γ: skew between x and y axes

Taking all the above into account, the camera matrix can be re-written as.

$$K = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Moreover, in the above equation, the x and y pixel coordinates are referred to the center of the image. Instead, while working with images the origin is at the top left corner of the image, thus we can represent the image coordinates using (u, v) as follows

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \tag{2.6}$$

Therefore, the goal of the camera calibration is to find the 3x3 intrinsic matrix K defined in *Equation 2.5* and the 4x4 extrinsic matrix T defined in *Equation 2.2* using a set of known 3D points (x_w, y_w, z_w) and their corresponding image coordinates (u, v).

2.5 Checkerboard and OpenCV

As explained above, in this work the camera calibration has been performed using the photogrammetric method. For this reason a 7x10 checkerboard has been used as known-geometry object and the OpenCV library in python will be exploited to extract the parameters, following the flowchart show in *Figure 2.4*.

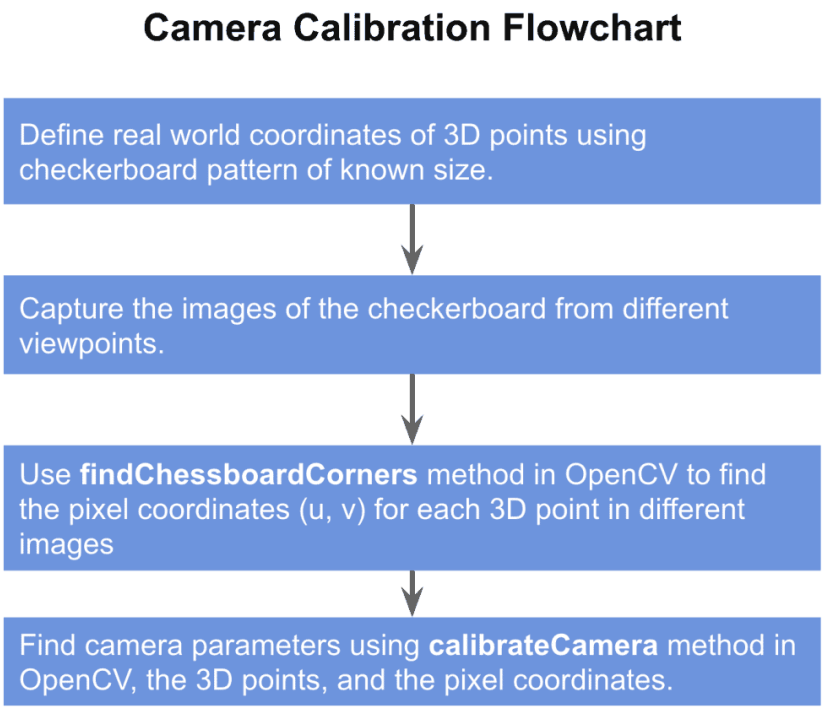

Figure 2.4: Camera calibration flowchart. Figure from [27]

In the process of calibration the camera parameters are computed using a set of know 3D points (x_w, y_w, z_w) and their corresponding pixel location (u,v) in the image. The world coordinates are fixed by the checkerboard pattern, where the 3D points are the corners of the squares.

For the 3D points we photograph a checkerboard pattern with known dimensions at many different orientations, as shown in *Figure 2.5*. Since points are equally spaced in the checkerboard, the (x_w, y_w) coordinates of each 3D point are easily defined by taking one point as reference (0, 0) and defining the remaining ones with respect to that reference point.

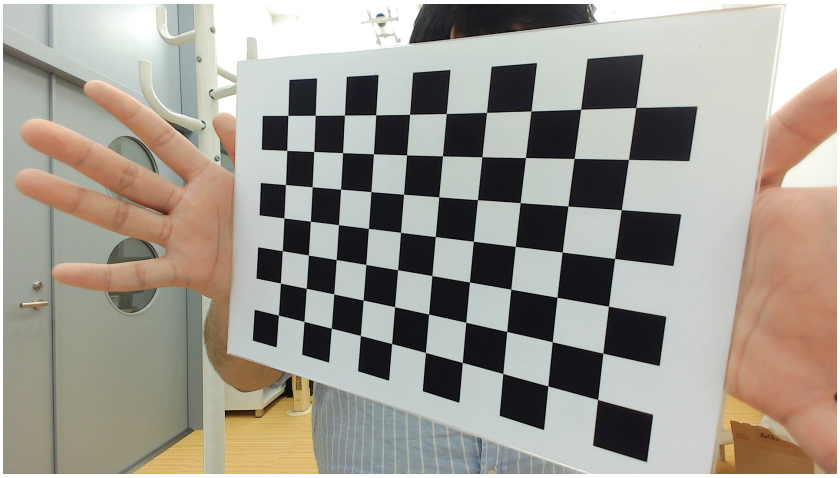

Figure 2.5: Camera calibration using checkerboard

We now have multiple images of the checkerboard and we also know the 3D location of points on the checkerboard in world coordinates. The last thing we need are the 2D pixel locations of these checkerboard corners in the images.
The calibration process has been performed using the OpenCV library in Python: the library provides a builtin function called *findChessboardCorners* that looks for a checkerboard and returns the coordinates of the corners.

```
1 ret, corners = cv2.findChessboardCorners(image, PatternSize,
2                                          flags)
```

where the inputs are:

- image: 8-bit grayscale or colored checkerboard image
- PatternSize: number of inner corners in the checkerboard
- flags: operation flags

and it returns:

- corners: output array of detected corners

Then, OpenCV's function *cornerSubPix* takes as input the original image and the location of corners, and looks for the best corner location inside a small neighborhood of the original location. The algorithm is iterative in nature and therefore we need to specify the termination criteria (e.g. number of iterations and/or the accuracy).

```
corners2 = cv2.cornerSubPix(gray, corners, winSize, zeroZone,
                            criteria_refine_corner)
```

where the inputs are:

- image: 8-bit grayscale or colored checkerboard image
- corners: initial coordinates of the input corners
- winSize: half of the side length of the search window ((11, 11) in this case)
- zeroZone: half of the size of the dead region in the middle of the search zone over which the summation in the formula below is not done ((-1,-1) in this case, which means there is no such a size).
- criteria_refine_corner: criteria for termination of the iterative process of corner refinement

and it returns *corners2*, which are the output refined coordinates.
The final step of calibration is to pass the 3D points in world coordinates and their 2D locations in all images to OpenCV's *calibrateCamera* method.

```
ret, intrinsic_matrix, distortion_coeffs, rvecs, tvecs =
  cv2.calibrateCamera(objPoints, imagePoints, imageSize)
```

where the inputs are:

- objPoints: vectors of 3D points
- imagePoints: vectors of the 2D image points
- imageSize: size of the image

and it returns:

- intrinsic_matrix: intrinsic camera matrix
- distortion_coeffs: lens distortion coefficients
- rvecs: rotation specified as a 3×1 vector. The direction of the vector specifies the axis of rotation and the magnitude of the vector specifies the angle of rotation
- tvecs: 3x1 translation vector

The result of the above method is shown in *Figure 2.6*, where all the corners of the board are detected and the rotation and translation vectors are reported in top left corner.

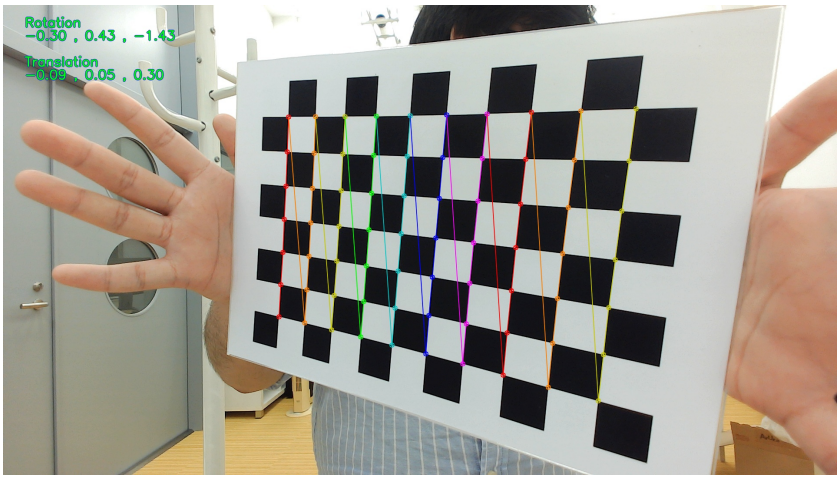

Figure 2.6: Checkerboard image after calibration

Chapter 3

Body pose estimation

3.1 Introduction

Computer Vision is a field that focuses on interpreting image content: it involves tasks like classifying entire images (e.g., on social media platforms like Facebook and Instagram), recognizing objects within images (such as faces or license plates, as seen in Facebook and Google Street View), and detecting specific features or patterns within images, like identifying cancer in biomedical images. The ultimate goal of computer vision is to develop systems that possess similar visual perception abilities as the human visual system, where humans use their eyes to see objects, and the brain interprets the visual information to understand what they see.

In this work, we will focus on the task of body pose detection.
Pose estimation holds great significance in numerous computer vision applications, including robot guidance, augmented reality, and various other contexts. This procedure revolves around identifying connections between points within the physical world and their two-dimensional image projections. This particular phase often presents considerable challenges, prompting the frequent adoption of synthetic or identifiable indicators to simplify the process.

The body pose detection in this work comprehends two tasks:

- *Camera pose estimation*: one widely favored methodology involves employing binary square identifiable markers (ArUco). The principal advantage of these markers lies in their capability to furnish an adequate number of connections (thanks to their four corners) for deriving the camera's pose from just one marker. Additionally, their inner binary encoding enhances their resilience, permitting the implementation of error detection and correction techniques. With the help of the documentation provided by OpenCV at [3], we will

discuss what are and how to use ArUco markers.

- *Body pose estimation*: the task will be carried out using MediaPipe, an open-source framework developed by Google (documentation available at [1]) that offers a collection of pre-built, customizable machine learning (ML) pipelines for various multimedia processing tasks: as explained in [19], it simplifies the development of multimedia applications by providing pre-trained models and modules for tasks like face detection, facial landmark detection, hand tracking, pose estimation, objectron (3D object detection and tracking), and more. Written in C++, It supports various programming languages like Python and Java, making it accessible and usable across different development environments.

3.2 Camera pose

3.2.1 ArUco markers

An ArUco marker is an artificial square marker constructed with a broad black outline and an embedded binary matrix that defines its unique identifier (ID).
The prominent black border streamlines its rapid detection within the image, while the binary encoding enables its recognition and enables the utilization of error identification and rectification methods.
The marker's dimensions dictate the dimensions of the internal matrix. For instance, a 4x4 marker comprises 16 bits. Some examples of ArUco markers are shown in *Figure 3.1*.

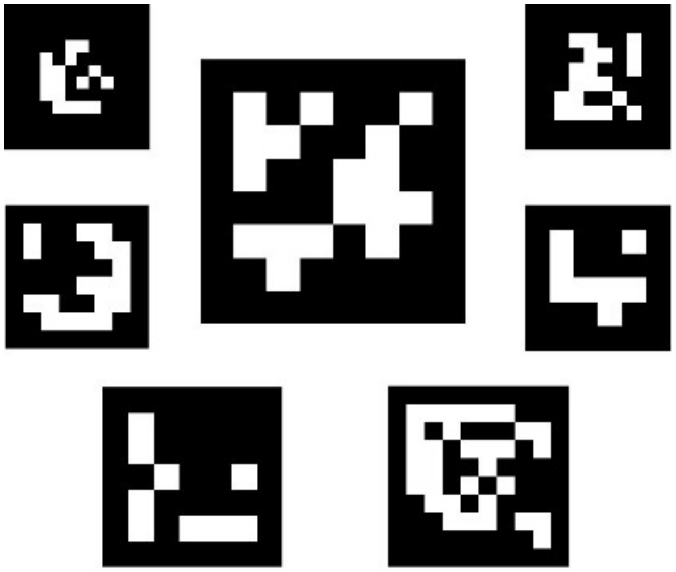

Figure 3.1: ArUco markers. Figure from [3]

A typical problem that may arise is that the marker may appear rotated within the environment. However, the detection process must have the capability to discern its original orientation to ensure unambiguous identification of each corner. This orientation determination also relies on the binary encoding.

A marker dictionary encompasses the collection of markers considered for a particular application. Essentially, it's a compendium of binary codifications for each marker within it. The dictionary's key attributes include its size, denoting the number of markers it comprises, and the marker size, indicating the bit count.

One might assume that the marker ID corresponds to a decimal number derived from converting the binary codification. However, this isn't feasible, especially for markers with a substantial number of bits, as managing such large numbers becomes impractical. Instead, the marker ID is straightforwardly the marker's index within its associated dictionary. For example, in a dictionary, the first five markers would have IDs: 0, 1, 2, 3, and 4.

When dealing with an image containing ArUco markers, the detection process should furnish a roster of recognized markers. Each detected marker comprises:

- The precise positions of its four corners in the image, in their original sequence
- The marker's ID

The marker detection process comprises two primary phases:

1. *Identification of marker candidates*: this stage entails a comprehensive analysis of the image to pinpoint square shapes that might qualify as markers. It commences with adaptive thresholding to segment the markers, followed by contour extraction from the thresholded image. Contours that lack convexity or fail to approximate a square shape are discarded. Additional filters are applied, such as eliminating contours that are excessively small, overly large, or too close to each other.

2. *Post-candidate detection*: the validation step necessitates confirming whether these candidates genuinely constitute markers by scrutinizing their inner encoding. This step commences by extracting the marker bits for each candidate. Achieving this involves applying a perspective transformation to bring the marker into its canonical form. Subsequently, Otsu thresholding is used to distinguish between white and black bits in the canonical image. The image is subdivided into cells according to marker size and border size. The count of black or white pixels within each cell determines the bit type. Finally, an analysis is performed to ascertain if the marker belongs to the specific dictionary. Error correction techniques are applied as needed.

In this work, a box with customized ArUco markers, shown in *Figure 3.2*, was used in order to estimate the camera pose.

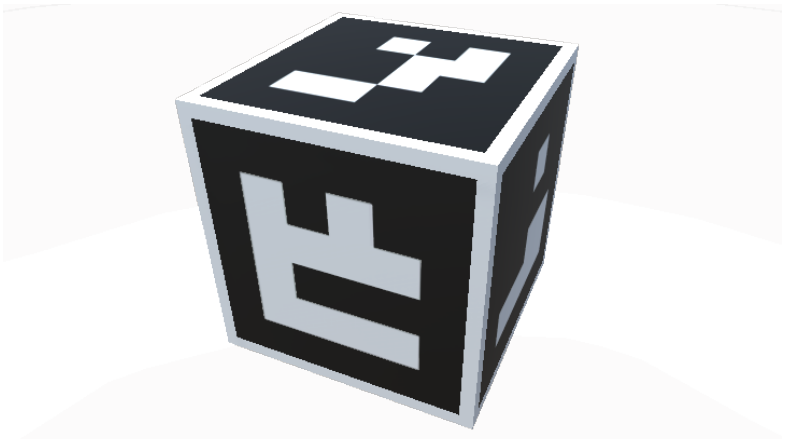

Figure 3.2: ArUco box

3.2.2 ArUco box pose estimation

To perform the box detection the ArUco module inside the OpenCV library in Python has been used. After setting all the parameters they have been stored using

the *DetectorParameters()* function.
Therefore, the detection is performed using the *detectMarkers()* function. This function is the most important in the module, since all the rest of the functionality is based on the detected markers returned by *detectMarkers()*.

```
corners, ids, rejected = aruco.detectMarkers(inputImage, dictionary,
                                             parameters)
```

where the inputs are:

- inputImage: 8-bit grayscale or colored image containing the markers to be detected
- dictionary: the marker dictionary that the function expects to identify
- parameters: The parameters of the markers, obtained using *DetectorParameters()*

and it returns:

- corners: the list of corners of the detected markers. For each marker, its four corners are returned in their original order (which is clockwise starting with top left)
- ids: the list of ids of each of the detected markers in *corners*
- rejected: the list of marker candidates, i.e. shapes that were found and considered but did not contain a valid marker

After the corners and the IDs of the marker have been detected, they can be used to estimate the relative position and pose of the marker with respect to the camera. To perform camera pose estimation, you need to know the camera's calibration parameters (the camera matrix and distortion coefficients). As a result of the calibration, a camera matrix, a matrix of 3x3 elements with the focal distances and the camera center coordinates (a.k.a intrinsic parameters), and the distortion coefficients, a vector of 5 or more elements that models the distortion produced by your camera, are obtained.

All the above have been estimated following the procedures in *Chapter 2*.
In order to estimate the camera pose the function *estimatePoseSingleMarker()* have been used.

```
rvecs, tvecs, _ = aruco.estimatePoseSingleMarkers(
                            detected_corners, marker_length,
                            camera_matrix, distortion_coefficients)
```

where the inputs are:

- detected_corners: the vector of corners detected by *detectMarkers()*
- marker_length: the size of the marker side, in meters
- camera_matrix: the camera matrix obtained during the camera calibration process
- distortion_coefficients: the distortion coefficients obtained during the camera calibration process

and it returns:

- rvecs: the rotation vector of the camera with respect to the detected marker
- tvecs: the translation vector of the camera with respect to the detected marker

The marker coordinate system that is assumed by this function is placed in the center (by default) or in the top left corner of the marker with the Z axis pointing out, as shown in *Figure 3.3*. Axis-color correspondences are X: red, Y: green, Z: blue.

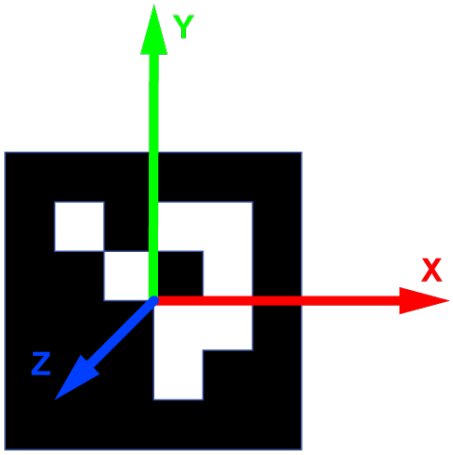

Figure 3.3: ArUco marker system coordinates

Finally, we can use the *drawDetectedMarkers()* function provided by the ArUco module followed by the *drawFrameAxes()* function provided by OpenCV to draw the detected markers in the input image and check whether the markers have been correctly detected.

```
drawn_image = aruco.drawDetectedMarkers(drawn_image,
                                        detected_corners,detected_ids);

drawn_image = cv2.drawFrameAxes(drawn_image, camera_matrix,
                                distortion_coefficients, rvec,
                                tvec, edge_length)
```

where the inputs for *drawDetectedMarkers()* are:

- drawn_image: the input image, where the markers will be drawn to obtain the output image
- detected_corners: the corners of the markers detected
- detected_ids: the id of the marker detected

and the inputs for *drawFrameAxes()* are:

- drawn_image: the input image, where the markers will be drawn to obtain the output image
- camera_matrix: the camera matrix obtained during the camera calibration process
- distortion_coefficients: the distortion coefficients obtained during the camera calibration process
- rvecs: the rotation vector of the camera with respect to the detected marker
- tvecs: the translation vector of the camera with respect to the detected marker

The results of the box detection and the camera pose estimation are shown in *Figure 3.4*.

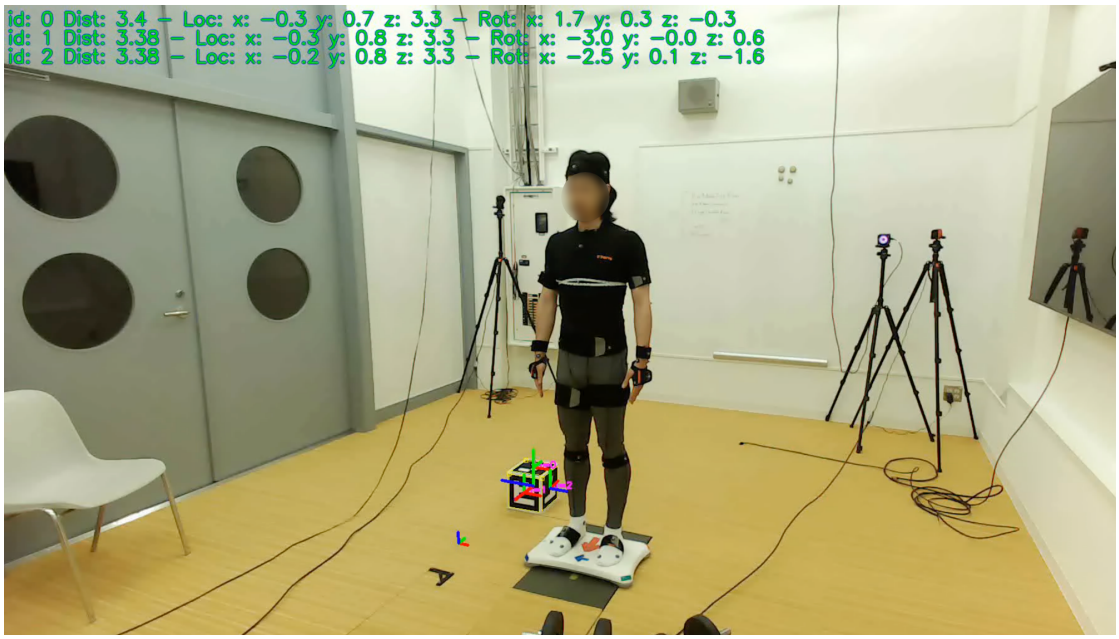

Figure 3.4: Aruco box detection

In this work we will use four cameras, placed in each corner of the room. The rotation matrix and translation vector of the cameras will be used in the following chapters to transform the data coming from all the different cameras into a unique reference system, in order to make them comparable with each other. For this reason, the variable *rvec* has been transformed into a rotation matrix using the OpenCV function *cv2.Rodrigues* and saved, as well as *tvec*, into a dictionary.

3.3 Body pose

3.3.1 MediaPipe

MediaPipe Pose represents an advanced machine learning system designed for precise human pose tracking: it discerns 33 3D landmarks and creates background segmentation masks for the body based on RGB video frames, employing the BlazePose research methodology [7]. Leading-edge methods currently rely predominantly on desktop environments and are compatible with most mobile phones, modern desktops/laptops, Python, and web applications. Possible applications of the MediaPipe Pose framework are shown in [35] and [16].

The MediaPipe algorithm utilizes a two-step detector machine learning pipeline. Initially, the detector identifies the specific person or pose within the frame. Subsequently, the tracker processes the cropped frame containing the pose of interest to predict the pose landmarks and segmentation masks.

Figure 3.5 illustrates the human modeling process using BlazePose's pose detector and landmark model, enabling precise tracking of the 33 key points on the human body from a single frame.

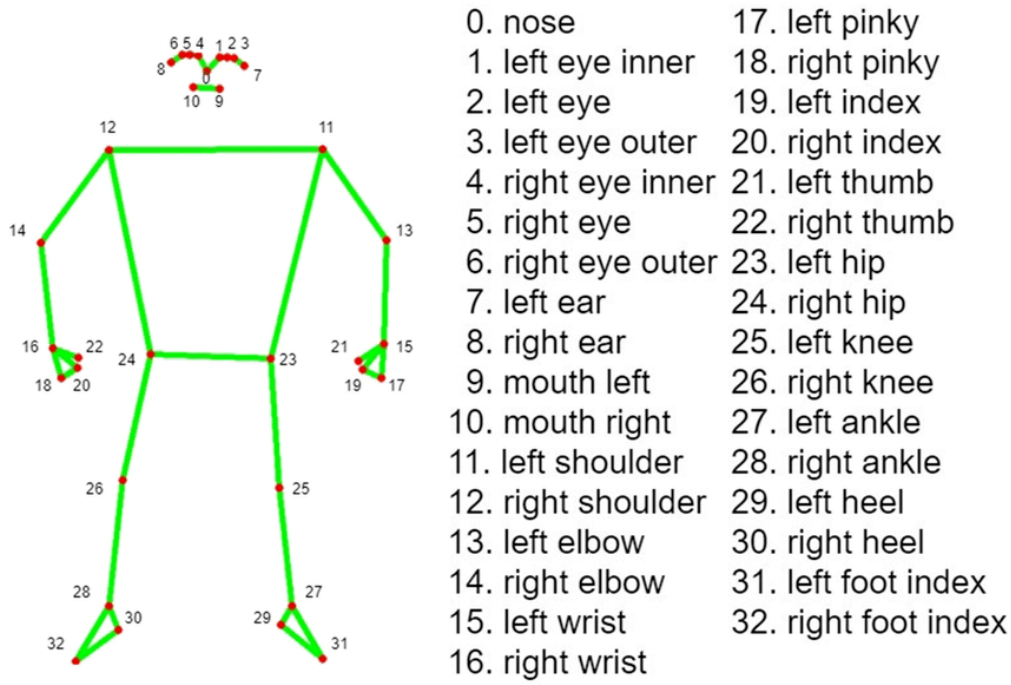

Figure 3.5: MediaPipe landmarks. Figure from [11]

In this work, the MediaPipe framework was used to extract the 33 landmarks of the subject coming from each of the four cameras used. Subsequently, the data has been rotated using the rotation matrices obtained in the *Section 3.2*, in order to get the landmarks position of each camera in the same unique reference system.

First, the *mp_pose.Pose()* object is instantiated as posed the method *pose.process()* is used to extract the pose of the subject:

```
with mp_pose.Pose(static_image_mode=False, min_detection_confidence
    =0.5, min_tracking_confidence=0.5) as pose:

    results_mp = pose.process(images)
```

where the inputs are:

- static_image_mode: whether the image to be processed is static or not
- min_detection_confidence: the minimum confidence score for the pose detection to be considered successful
- min_tracking_confidence: the minimum confidence score for the pose tracking to be considered successful

The attributes of *results_mp* can then be used to extract all the information we need to describe the body pose and store them in the following dictionaries:

- results_hollistic_i: 2D image coordinates
- results_hollistic_w: 3D world coordinates
- results_hollistic_v: Visibility pose

```
results_hollistic_i[i] = _process_mp_to_image()
results_hollistic_w[i] = _process_mp_to_w()
results_hollistic_v[i] = _process_mp_to_visibility()
```

where:

```
def _process_mp_to_image(self):
    results_i = np.array([[p.x * image_width, p.y * image_height] for
    p in results_mp.pose_landmarks.landmark]).astype(dtype=int)
    return results_i

def _process_mp_to_w(self):
    landmarks = results_mp.pose_world_landmarks.landmark
    results_w = np.array([[landmarks[k].x, landmarks[k].y, landmarks[
   k].z] for k in range(n_landmark)])
    return results_w

 def _process_mp_to_visibility(self):
    landmarks = results_mp.pose_landmarks.landmark
    result_visibility = np.array([[landmarks[k].visibility] for k in
   range(n_landmark)])
    return result_visibility
```

After all the landmarks have been extracted, we can proceed drawing the skeletons using the *mp_drawing.draw_landmarks()* function as follows:

```
 Drawing.mp_drawing.draw_landmarks(image=image,
                        landmark_list=results_mp_pl,
                        connections=mp_holistic.POSE_CONNECTIONS,
                        landmark_drawing_spec=Drawing.mp_drawing.
   DrawingSpec(
                        **Drawing.pose_landmarks_parameters),
                        connection_drawing_spec=Drawing.mp_drawing.
   DrawingSpec(
                        **Drawing.pose_connection_parameters))
```

The results of the above computation for one of the four cameras in our setup are shown in *Figure 3.6*.

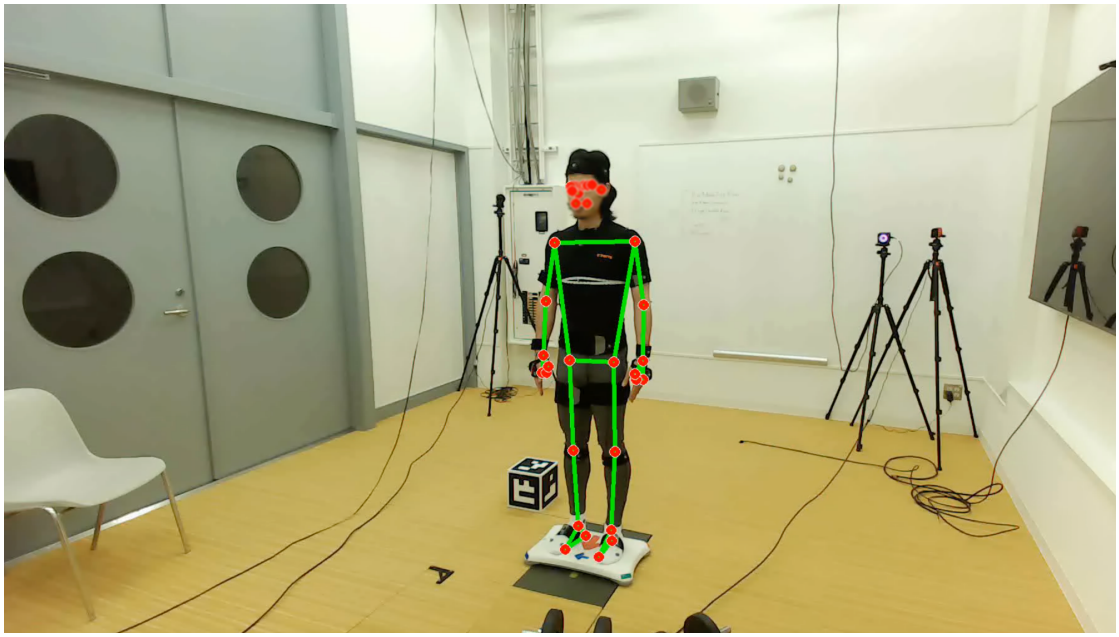

Figure 3.6: Body pose detection

3.3.2 Landmarks rotation

After computing the body pose for each one of the four cameras, we obtain four skeletons of the same subject detected from different points of view. We are only interested in the 3D world coordinates of the 33 skeleton landmarks, which are the variable [x, y, z] contained for each frame in the dictionary *results_hollistic_w*.

Using the rotation matrix R and translation vector t obtained in *Section 3.2.2* for each camera, we rotated the data in order to obtain the 33 joints position all in the same reference system as shown in *Equation 3.1*, which is the one of the ArUco box.

$$\begin{bmatrix} x_{rotated} \\ y_{rotated} \\ z_{rotated} \end{bmatrix} = T \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \qquad (3.1)$$

$$T = \begin{bmatrix} R & | & t \end{bmatrix}$$

The resulting four skeletons of the subject performing an N pose, rotated in the same reference system, are shown in *Figure 3.7*.

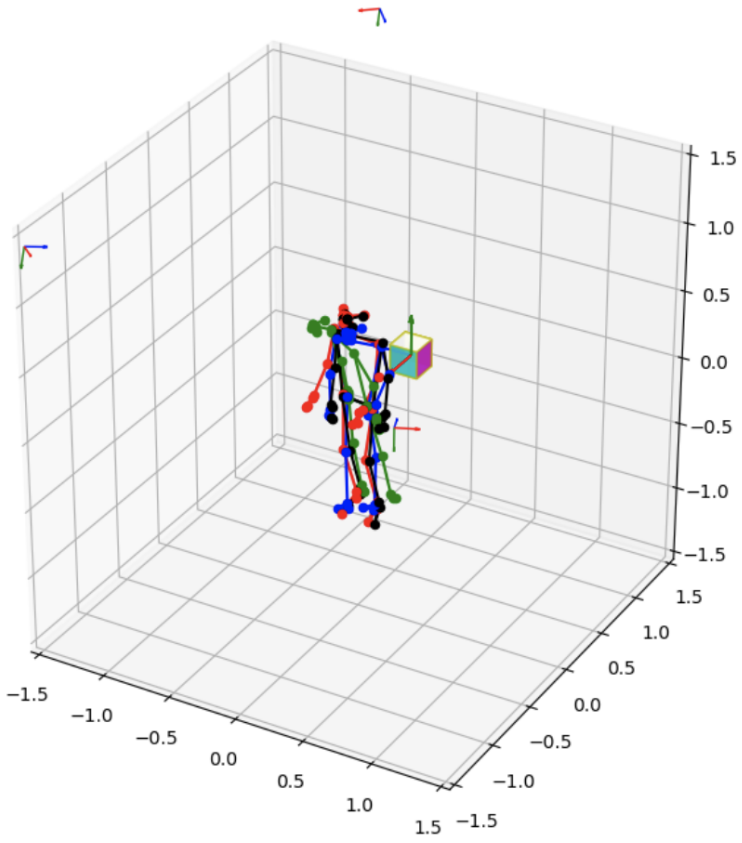

Figure 3.7: Rotated skeletons: ■ Camera 1; ■ Camera 2; ■ Camera 3; ■ Camera 4

The next step will be the fusion of the data coming from the different cameras (*Chapter 4*) in order to obtain an estimation of the pose as accurate as possible.

Chapter 4

Sensor fusion

4.1 Introduction

Sensor fusion is the process of combining data from multiple sensors (cameras, radar, lidar etc.) to obtain a more accurate and comprehensive understanding of a physical phenomenon, environment, or object.
The primary goal of sensor fusion is to improve the accuracy and ensure redundancy and robustness: by combining data from multiple sensors, it is possible to reduce errors and inaccuracies associated with any single sensor, leading to more reliable and precise measurements, as shown in [23] and [13], where skeleton fusion algorithms were deployed. Moreover, if one sensor fails or provides erroneous data, the system can still function using data from the other sensors, making the system more robust in challenging conditions, such as poor visibility or adverse weather, where a single sensor may be unreliable.
Sensor fusion is commonly used in applications like tracking the position and movement of objects, such as in military surveillance, robotics, and autonomous navigation, like the case of [12], which applied sensor fusion algorithms to UAV navigation: in our case, it will be used to fuse the body pose data coming from different cameras, obtained in *Chapter 3*.

The two main approaches to sensor fusion are *hard fusion* and *soft fusion*.
In the first case, the data from different sensors are combined into a single, integrated data-set, involving mathematical operations like averaging or weighted summation. Instead, in the second case, the fusion involves combining data from multiple sensors while retaining their individual characteristics and uncertainties, enhancing techniques such as Bayesian inference or Kalman filtering, as already done in the field of human pose fusion by [26], [24] and [17].

In this work, the fusion of the data will be performed using four different techniques:

- Mean value
- Kalman filter
- Mean value + DBSCAN
- Kalman filter + DBSCAN

In order to test the methods mentioned above, a set of six exercises has been performed by the subject:

- Dumbbell shoulder press
- Hip adduction left
- Hip adduction right
- Lunge step up left
- Lunge step up right
- Squat

4.2 Multi-sensor fusion techniques

Imagine the challenge of tracking a moving target using N diverse sensors, each with distinct measurement characteristics and noise profiles. How can we merge the measurements from these multiple sensors to derive a collective estimation of the target's state that outperforms the individual sensor-based estimates? Various strategies for multi-sensor data fusion exist to address this issue. In this study, we will employ the Mean fusion and Kalman filter fusion techniques.

Specifically, with respect to the Kalman filter approach, we can consider two primary fusion methods: state-vector fusion and measurement fusion.
As shown in [10] and [25], state-vector fusion techniques involve utilizing a set of Kalman filters to generate separate state estimates based on each sensor's data, which are subsequently integrated to produce an enhanced joint state estimate.
On the other hand, measurement fusion methods directly fuse the sensor measurements to obtain a weighted measurement and then use a single Kalman filter to obtain the final state estimate based upon the fused observation.

The block scheme comparing the two techniques is shown in *Figure 4.1*.

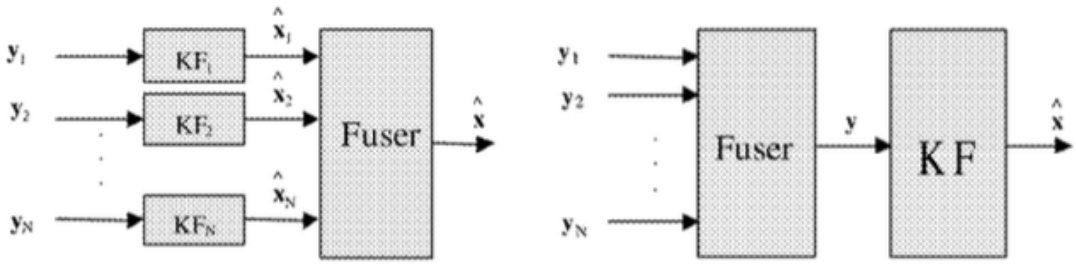

Figure 4.1: State fusion and measurements fusion methods. Figure from [10]

Although it has been demonstrated that the two measurement fusion techniques exhibit functional equivalence when the sensors, each with distinct noise characteristics, possess identical measurement matrices, it's crucial to consider their respective advantages and disadvantages.

In comparative investigations involving state-vector fusion and measurement fusion, it has been observed that measurement fusion methods typically deliver superior overall estimation performance. On the other hand, state-vector fusion methods offer lower computational and communication overhead, along with the benefits of parallel implementation and fault tolerance. It's worth noting that state-vector fusion methods are effective only under the condition of consistent Kalman filters, limiting their practical applicability.
In many real-world scenarios like navigation and target tracking, underlying processes are frequently nonlinear. Consequently, the corresponding Kalman filters rely on linearized process models (e.g., Jacobian linearization or neurofuzzy local linearization) and tend to be inconsistent due to model errors introduced during the linearization process. Therefore, when employing Kalman-filter-based multisensor data fusion in these practical settings, measurement fusion is the preferred choice over state-vector fusion.

In our specific case, the measurement fusion approach will be adopted.

4.3 State-space model

Like the model built in [26], we can consider a linear system. The dynamics of our sensors are modelled by the following discrete-time state-space model:

$$\begin{cases} x_k = A_k x_{k-1} + B_k u_k + w_k \\ y_k = H_k x_k + v_k \end{cases} \tag{4.1}$$

where k represents the discrete-time index and A, B, H, x, y and are the state transition matrix, input transition matrix, measurement matrix, state vector, measurement vector and input control vector, respectively. It is assumed that w is the process noise vector, which has zero mean with a covariance matrix $Q = Eww^T$, and v is the measurement noise vector, which also has zero mean with a covariance matrix $R = Evv^T$. In this work, since we consider an uncorrelated covariance matrix, Q and R become diagonal matrices given by:

$$Q = \begin{cases} Q_{ii} = E\left\{ww^T\right\} \\ Q_{ij} = 0 \end{cases} \tag{4.2}$$

$$R = \begin{cases} R_{ii} = E\left\{vv^T\right\} \\ R_{ij} = 0 \end{cases} \tag{4.3}$$

where the dimension of the measurements is D and the linear dynamic targets are tracked by N sensors. Both y_k and v_k in *Equation 4.4* and *Equation 4.5* are augmented to establish the DN$\times$ 1 observation vector as follows:

$$y_k = \left[\left(y_k^1\right)^T \quad \left(y_k^2\right)^T \quad ... \quad \left(y_k^N\right)^T \right]^T \tag{4.4}$$

$$v_k = \left[\left(v_k^1\right)^T \quad \left(v_k^2\right)^T \quad ... \quad \left(v_k^N\right)^T \right]^T \tag{4.5}$$

4.4 Hard fusion

In the hard fusion case, in this work a simple averaging technique has been adopted. The fusion has been performed computing the mean value of the joint position coming from each one of the four cameras adopted.

$$\bar{y}_k = \frac{1}{N}\left(\sum_{i=1}^{N} y_k\right) \tag{4.6}$$

This method is quite straightforward, nevertheless as shown by *Equation 4.6* it presents as typical drawback the fact that it is not possible to take into account the

different behaviour and the working condition (e.g. self-occlusion, setup dependent noise etc.) of all the sensors in used, that is the measurement noise vector v and the process noise vector w are not considered in the fusion.
In this case, all the sensors that could not detect any joint were excluded from the computation of the mean value.

4.5 Soft fusion

4.5.1 Measurements fusion

As explained in *Section 4.2*, we will enhance and then describe the measurement fusion method, as we will directly combine sensor measurements to generate a weighted fused measurement.
In measurement fusion, the N sensor models can be integrated into a single model, as shown in [26]: the fused measurement covariance, R_k , fused measurement matrix, H_k , and fused measurement vector, y_k , are given as

$$\bar{R_k} = \left(\sum_{i=1}^{N} (R_k^i)^{-1} \right)^{-1} \tag{4.7}$$

$$\bar{H_k} = \bar{R_k} \left(\sum_{i=1}^{N} (R_k^i)^{-1} H_k \right) \tag{4.8}$$

$$\bar{y_k} = \bar{R_k} \left(\sum_{i=1}^{N} (R_k^i)^{-1} y_k \right) \tag{4.9}$$

As depicted in *Equation 4.9*, the combined measurement vector y_k is derived by weighting each individual measurement vector. These weights are determined by the inverse of each measurement covariance matrix, denoted as $(R_k^i)^{-1}$. Therefore, when the observation's variance is substantial, its impact on the fused measurement is relatively feeble. Conversely, if the observation exhibits lower variance, it can exert a more substantial influence on the resulting fused value.

4.5.2 Joint continuity method

Equations 4.7-4.9 provide insight into the crucial aspect of appropriately combining measurements, where the key consideration is establishing the covariance matrix for each measurement. To accomplish this, in this work we followed the approach introduced by [26] to fine-tunes the augmented measurement noise vector, denoted as v in *Equation 4.5* and assign the reliability of each measurement.

In order to assess the reliability of a tracked skeleton, we concentrate the evaluation on the continuity of the human motion: discontinuous motion often results from challenges in estimating the 3D position of a joint and falls into two distinct categories.

In the first case, the joint may be moving rapidly, but the computed joint velocity, based on both current and previous measurements, appears to be slow.
On the other hand, the joint may actually be moving slowly, but the calculated joint velocity, derived from current and past measurements, appears to be fast.
To differentiate between these two cases we employ a straightforward voting mechanism and we assign high measurement noise values to the joints exhibiting discontinuous motion, in such a way that unreliable values have minimal impact on the fused measurement.

Algorithm I outlines our proposed approach for detecting these two types of discontinuous joint motion and assigning high measurement noise values accordingly. In this work, we do not explicitly estimate the velocity of each measurement; instead, we implicitly calculate it by computing the distance between the current observation and the previously estimated 3D joint positions using Kalman filtering. If this distance exceeds the threshold denoted as θ_f, it constitutes a positive case, supporting the hypothesis that the corresponding joint is moving fast. After that, if the number of positive cases exceeds half of the sensors, we categorize the joint as a fast movement joint; otherwise, it is considered a slow movement joint. Once the joint type is determined, we estimate the reliability of measurements from each sensor: if, for instance, a given joint is determined to be fast by the majority of the sensors, we will assign a high measurement noise value to the sensors that instead classified it as slow, and vice versa. The noise value assigned will be the maximum value between an initial value v_{in} and a term directly proportional to the velocity of the joint.

Algorithm 1 Determining Reliability based on Continuity (y_k, y_{k-1}, v_k)

▷ M: Number of joints to be tracked
▷ N: Number of sensors
▷ X[M][N]: M x N 2D array (non-fast joints:0, fast joints:1)
▷ d(y', y"): Euclidian distance between y' and y"
▷ θ_f: Threshold distance(cm) for fast & non-fast joint motion
▷ γ: Scaling factor

for $m \leftarrow 1$ to M **do**
  $count \leftarrow 0$
  for $n \leftarrow 1$ to N **do**
    if $d(y^n_{k,m}, y^n_{k-1,m}) > \theta_f$ **then**
      $X[m][n] \leftarrow 1$
      $count \leftarrow count + 1$
    else
      $X[m][n] \leftarrow 0$
    end if
  end for
  if $count > \frac{N}{2}$ **then**
    for $n \leftarrow 1$ to N **do**
      if X[m][n] = 0 **then**
        $v^n_{k,m,x} \leftarrow max(v_{k,m,x}, \gamma d(y^n_{k,m}, y^n_{k-1,m}))$
        $v^n_{k,m,y} \leftarrow max(v_{k,m,x}, \gamma d(y^n_{k,m}, y^n_{k-1,m}))$
        $v^n_{k,m,z} \leftarrow max(v_{k,m,x}, \gamma d(y^n_{k,m}, y^n_{k-1,m}))$
      end if
    end for
  else
    for $n \leftarrow 1$ to N **do**
      if X[m][n] = 1 **then**
        $v^n_{k,m,x} \leftarrow max(v_{k,m,x}, \gamma d(y^n_{k,m}, y^n_{k-1,m}))$
        $v^n_{k,m,y} \leftarrow max(v_{k,m,x}, \gamma d(y^n_{k,m}, y^n_{k-1,m}))$
        $v^n_{k,m,z} \leftarrow max(v_{k,m,x}, \gamma d(y^n_{k,m}, y^n_{k-1,m}))$
      end if
    end for
  end if
end for

4.5.3 Kalman filter

The Kalman filter, as clearly explained by [31], is a set of recursive mathematical equations that provides an efficient computational way to estimate the state of a dynamical system, often in the presence of noisy measurements. It was developed by Rudolf E. Kalman in the 1960s and has found widespread use in various fields, including control systems, robotics, economics, and more. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown. The Kalman filter addresses the general problem of trying to estimate the state $x \in \mathbb{R}^n$ of a discrete-time controlled process that is governed by the linear stochastic differential equation:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{4.10}$$

with a measurement vector $z \in \mathbb{R}^m$ that is defined as:

$$z_k = Hx_k + v_k \tag{4.11}$$

The random variables w_k and v_k represent respectively the process and measurement noise. They are assumed to be independent, white, and with normal probability distributions:

$$\begin{aligned} p(w) &\sim N(0, Q) \\ p(v) &\sim N(0, R) \end{aligned} \tag{4.12}$$

The Kalman Filter is highly beneficial in scenarios involving imprecise sensor data: it functions through a continual process of refining its estimation by incorporating a forecast of the system's state and the latest measurements collected as time progresses.
The filter operates through a feedback control mechanism, as shown in *Figure 4.2*, where it estimates the state of a process at a particular time and then acquires feedback in the form of measurements, which are often noisy.
Going into details on how the filter works, we can come up with four steps, that will be therefore translated into Python code as instructed by [18]:

1. *Initialization*: the filter starts with an initial estimate of the system's state and an initial estimate of the error in that state.

2. *Prediction*: the filter predicts the next state of the system based on a mathematical model of the system's dynamics. This prediction includes an estimate of how the state will evolve over time and how uncertain that prediction is.

3. *Update*: when new sensor measurements become available, the filter compares them to the predicted state. It calculates the difference between the predicted and measured values, known as the "measurement residual" or "innovation." The Kalman Gain then determines how much the filter trusts the prediction versus the measurement. Afterwards the filter updates its state estimate and its estimate of the state's uncertainty based on the measurement.

4. *Repeat*: prediction and update are repeated as new measurements arrive, continually refining the state estimate based on the latest data. This recursive process is what makes the Kalman Filter powerful for tracking dynamic systems in real-time.

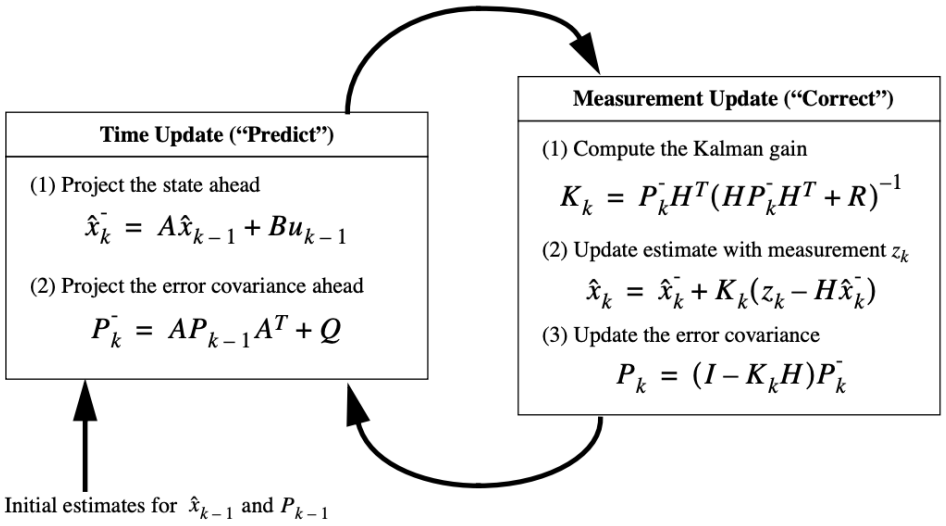

Figure 4.2: Kalman filter loop

The Kalman Filter operates under the assumption that the system's dynamics and measurement noise follow Gaussian distributions, typically having a normal distribution. It efficiently merges predictive and measurement data to yield a state estimate that minimizes the mean squared error between prediction and the true state.

Going more into the mathematics shown in *Figure 4.2*, given the state-space model described by *Equation 4.1*, the Kalman filter provides an unbiased and optimal estimate of the state-vector in the sense of minimizing the estimate covariance. The algorithm works in a two-step process, comprising prediction and update steps.

The prediction step of the Kalman filter is given by:

$$\hat{x}_{k|k-1} = A_k \hat{x}_{k-1|k-1} + B_k u_k \tag{4.13}$$

$$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + Q_k \tag{4.14}$$

where $\hat{x}_{k|k-1}$ is an a-posteriori state estimate at time k given observations up to and including time k-1, and where $P_{k|k-1}$ denotes the covariance matrix at time k given observations up to and including time k - 1.

The update step of the Kalman filter is given by:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \left(\bar{y}_k - \bar{H}_k \hat{x}_{k|k-1} \right) \tag{4.15}$$

$$P_{k|k} = I - \left(K_k \bar{H}_k \right) P_{k|k-1} \tag{4.16}$$

where K_k denotes the Kalman Gain Matrix and $\bar{y}_k - \bar{H}_k \hat{x}_{k|k-1}$ is called Innovation. The Kalman Gain Matrix K_k and Innovation Matrix S_k are described by the following equations:

$$K_k = P_{k|k-1} \bar{H}_k^T S_k^{-1} \tag{4.17}$$

$$S_k = \bar{H}_k P_{k|k-1} \bar{H}_k^T + \bar{R}_k \tag{4.18}$$

4.6 DBSCAN

In the merging process, a key aspect is to consider only in-lier candidates. This can be done by noise filtering the joints based on their position: noisy candidates are recognized by identifying the sensors that had issues recognizing the target joint. In this work, following the work of [17], we used DBSCAN to identify out-liers: in this way we were able to distinguish the noisy candidate based on the distance from the in-lier candidates, as shown in *Figure 4.3*.

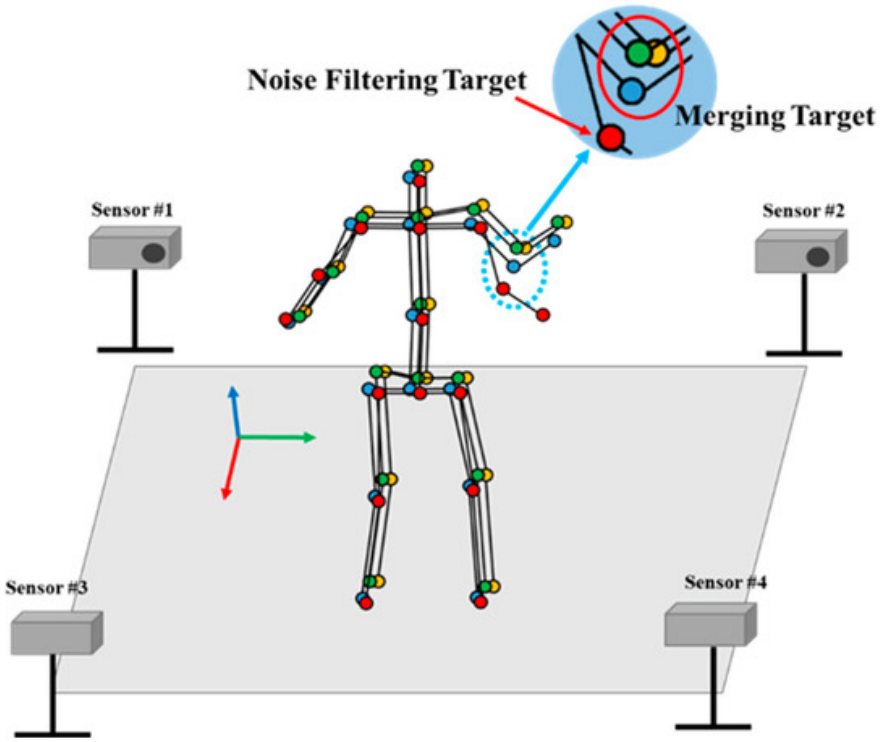

Figure 4.3: Skeleton merging target and noise rejection. Figure from [17]

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm used in machine learning and data mining. Unlike partitioning-based clustering methods like k-means, where you need to specify 'k' (the number of clusters), DBSCAN is a density-based clustering method: it can identify clusters of arbitrary shapes in data based on the data density distribution and can also identify points that don't belong to any cluster (i.e., noise).
On top of that, DBSCAN can identify clusters of varying shapes, whereas methods like k-means typically identify spherical clusters.

The fundamental premise guiding its operation is that candidates within a common cluster will tend to be positioned close to each other. Employing an approach that clusters adjacent data based on data density, DBSCAN demonstrates strong clustering capabilities, even when dealing with data of irregular shapes.

Additionally, DBSCAN possesses the ability to classify noisy data points during the clustering process, thereby diminishing the degradation of the clustering quality that may result from the involvement of outliers. Its operation involves the utilization of hyperparameters, specifically the ϵ parameter, defining the radius for searching neighboring data, and the minimum count of neighboring data points, denoted as N_c.

As a drawback, DBSCAN might not perform well when clusters have significantly different densities: in this case tuning the hyperparameters 'eps' and 'min samples' can be difficult, especially for datasets with varying densities.

Decomposing the algorithm in steps, this is how DBSCAN works:

1. *Parameter selection*: the hyperparameters ϵ (radius of a neghborhood around a data point) and N_c (minimum number of points) are selected.

2. *Neighborhood computation*: for each data point, the algorithm compute how many points lie within an ϵ radius of it.

3. *Cluster assignment*:
   - If a point has at least N_c points within its ϵ radius, it is labeled as a core point. All points within the ϵ radius of a core point are part of the same cluster.
   - If a point is within the ϵ radius of a core point but does not have enough points within its own ϵ radius, it's labeled as a border point.
   - Points that are neither core nor border points are considered noise.

4. *Expansion*: Starting from an arbitrary core point, DBSCAN explores and expands the cluster by checking the neighboring points and determining if they can be added to the same cluster. The process continues recursively until the entire cluster is discovered.

5. *Iterate*: DBSCAN then proceeds to the next unvisited core point to start a new cluster and continues the process until all points are either assigned to a cluster or marked as noise.

The detailed operation procedure of DBSCAN is described in *Algorithm 2* and *Algorithm 3*.

Algorithm 2 DBSCAN

▷ X: Candidates set
▷ ϵ: Searching area
▷ N_c: Minimum number of neighboring data
▷ y: labels
▷ k: number of clusters

Initialize $k = 0$
for all $x \in \mathcal{X}$ **do**
  $y_x \leftarrow UNASSIGNED$
end for
for all $x \in \mathcal{X}$ **do**
  if $y_x = UNASSIGNED$ **then**
    $X_x = SCAN(x, \epsilon)$
    if $|X_x| \geq N_c$ **then**
      $k \leftarrow k + 1$
      $y_x \leftarrow k$
      for all $z \in \mathcal{X}_x$ **do**
        if $y_z = UNASSIGNED$ **then**
          $y_z \leftarrow k$
          $X_z = SCAN(x, \epsilon)$
          if $|X_z| \geq N_c$ **then**
            $X_x \leftarrow X_x \bigcup X_z$
          end if
        end if
      end for
    else
      $y_x \leftarrow NOISE$
    end if
  end if
end for
return y

Algorithm 3 SCAN

▷ X: Data point
▷ ϵ: Searching area
▷ X_x: Neighbors

for all $z \in \mathcal{X}$ **do**
  if $Euclidian_distance(x, z) \leq \epsilon$ **then**
    $X_x \leftarrow X_x \bigcup z$
  end if
end for
return X_x

In this work, the algorithm defines the probability that the data in the cluster are the same as the actual location of the target joint. In other words, it is assumed that the more densely the positions of candidate joints recognized from different cameras belong, the higher the probability that the data constituting the cluster is the same as the actual joint coordinates.
Moreover, there are cases where the movement of the identified joint surpasses the actual joint motion extent, or in some cases, no joint movement is recognized at all. To address this issue, we assigned the position [0, 0, 0] to the untracked data point. This point will subsequently be discarded either during the DBSCAN operation or in the following stages within the sensor fusion process.

Following the application of DBSCAN, we identified the cluster with the highest number of data points as the candidate group representing the target joint.
Among the hyperparameters of DBSCAN described above, N_c was fixed to 1 and ϵ was fixed to 10 cm.

Chapter 5

Motion capture system

5.1 Introduction

A motion capture system, commonly referred to as "mocap" or "MoCap," represents a technology employed to digitally record and apprehend the motions of objects or living entities. It is primarily utilized for the analysis and recreation of these movements in various applications, including computer-generated imagery (CGI), animation, bio-mechanical research, sports analysis, and more. In recent years, optical motion capture has gained significant prominence, especially in applications related to body movement animation, as shown in [29]. Although commercially available systems offer high-quality results, many of them come with prohibitive costs.
More specifically, in sports research the need for motion analysis of the athlets often arises, with applications like the one depicted by [6]. This process involves the recording of human movements, with a particular focus on capturing the overall body position (segments) of the subject.

In this work, an optoelectronic system has been used as groundtruth to evaluate the four fusion methods described in *Chapter 4*.
Specifically, $Prime^X 13$ cameras by OptiTrack have been used to collect the data, processed then using the software Motive, again by OptiTrack.

5.2 Mocap overview

To appreciate why motion capture stands out as one of the preferred choice for animation and body movement analysis, a short overview of the methodologies used by this technology is presented.
Broadly, Mocap can be categorized into *Marker-based Motion Capture* and *Markerless Motion Capture*.

- *Marker-based Motion Capture*: these systems are generally divided into four categories, as well explained by [28] and [29]. The main characteristics of each of these categories are briefly described:

    1. *Acoustic Systems*: a collection of sound emitters is strategically positioned on the key joints of the actor, while sensitive receptors are placed in the vicinity of the capture area. Subsequently, the emitters are activated sequentially, emitting a range of frequencies detected by the receptors. These frequencies are then used to calculate the positions of the emitters in three-dimensional space. While these systems offer certain advantages, they come with a set of challenges. These include difficulties in accurately describing data at specific moments, limitations on movement due to unwieldy cables, a restricted number of transmitters that can be used simultaneously, susceptibility to external noise and interference, which can impact the capture process. However, one notable benefit is that these systems do not face issues related to obstruction or interference from metallic objects.
    2. *Mechanical Systems*: this system is comprised of sliders and potentiometers strategically placed at desired joints, allowing for precise positioning and orientation measurements at a high sampling rate. These devices provide absolute measurements unaffected by magnetic fields or unwanted reflections. However, they tend to be quite obstructive in nature.
    3. *Magnetic Systems*: distinguished by their rapid data processing capabilities, magnetic systems operate at a sampling rate of approximately 100 frames per second (fps). They employ a set of receptors positioned on the actor's joints to measure 3D position and orientation relative to an emitter antenna. The emitter antenna emits a primary signal, with each receptor requiring a connected cable. Noteworthy advantages include low computational costs for data processing, freedom from obstruction (leading to enhanced data accuracy), and cost-effective equipment. Nevertheless, a significant drawback is the substantial use of cables to connect to the antenna, which limits the degree of freedom.

4. *Optical Systems*: the actor wears specialized clothing adorned with reflectors, often LEDs, positioned on key joints. Special cameras are strategically placed to track the movement of these reflectors as the actor moves. These high-resolution cameras capture 2D coordinates from the reflectors through a segmentation process. The collected 2D data from independent cameras are then analyzed to generate the 3D coordinates of the reflectors. The advantages of this approach include a high sampling rate, unrestricted movement, and the ability to use numerous reflectors. Downsides encompass the potential obstruction of one or more markers during the capture process and the necessity of post-processing the camera-obtained data through software, which reduces real-time interactivity.

- *Markerless Motion Capture*: this method eliminates the need for specialized equipment to track an actor's motion. Instead, motion is recorded directly from a video data sequence using motion-based algorithms designed to track and detect objects. This process is accomplished using software, bypassing computational constraints and providing flexibility. For example, Microsoft's Kinect represents a low-cost motion capture system accessible to a broader audience.

5.3 Optoelectronic systems

The utilization of optical motion tracking systems has seen a growing presence across various domains, including the realms of entertainment, bio-mechanics, and sports sciences.
These systems operate based on stereophotogrammetry: they generate three-dimensional coordinates for points on an object undergoing measurements from two or more photographic images captured from distinct points. These systems typically comprise cameras, markers, and processing software.

According to [21], as the adoption of motion analysis has expanded, it's worth noting that besides the leading, high-priced systems like Vicon (Oxford Metrics, UK), more affordable camera setups have emerged that were not initially designed for scientific purposes but have found their way into scientific motion labs. One notable example is OptiTrack (NaturalPoint, Corvallis, OR, USA), which originated in animation motion capture and has since found applications in biomechanics. Its primary uses now encompass virtual reality (VR), robotics, movement sciences, and animations.
The proliferation of cost-effective systems also necessitates validation studies that assess the accuracy of these new systems compared to the established gold standard systems in scientific research. Furthermore, various technical aspects, such as

the capture volume, the smallest detectable marker size, the frequency, and the resolution of the motion capture system, may be crucial factors to consider when evaluating their suitability for specific applications.

5.3.1 OptiTrack

In this work, cameras [4] and software [5] by OptiTrack have been utilized to collect and process data.
OptiTrack is a brand of motion capture technology and systems developed and manufactured by NaturalPoint, a company based in Corvallis, Oregon, USA.

It specializes in high-precision, camera-based motion capture systems: these systems typically use multiple high-speed cameras placed around a capture volume to track the positions of reflective markers placed on the subject. The data collected from these cameras is then processed to create highly accurate 3D motion data.

The cameras used, $Prime^X 13$, is shown in *Figure 5.1*

Figure 5.1: $Prime^X 13$ camera

OptiTrack systems are known for their accuracy and reliability, making them popular in many professional settings that require precise motion capture data. For this reason, the data captured by this system have been used as groundtruth to evaluate the performance of the sensor fusion algorithms.

The software Motive by OptiTrack was used to calibrate the cameras, collect the data and process it by labelling the markers detected in each frame.

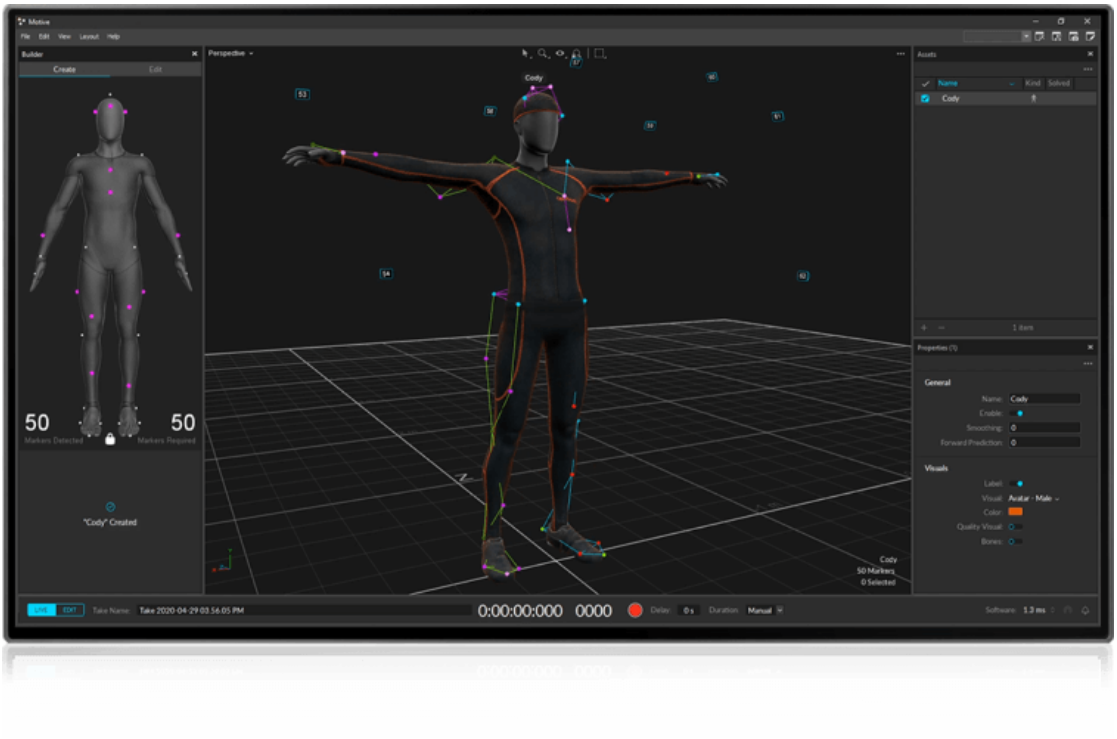

Figure 5.2: Motive software

Chapter 6

Human activity recognition

6.1 Introduction

Human Activity Recognition (HAR) is a field of study using machine learning (ML) models to identify human movements from data provided by sensors such as optical motion capture systems, inertial measurement units or force plates, to name a few. Past research has demonstrated the effectiveness of machine learning techniques in effectively classifying diverse activities from sensor data [22] [37] [34]. Various types of sensors are employed in HAR systems, including digital cameras, depth sensors, wearable sensors, and gyro sensors.
In the context of human assistive devices, this classification can serve as input to the control system for adapting to the assistance ratio of active orthosis or exoskeleton based on the type and intensity of the activity being performed. Alternatively, it can function as a redundant system to verify whether the robot's assistance aligns with the user's intent. In addition, HAR can be implemented in safety systems to determine if the user is approaching a hazard or experiencing loss of stability and respond accordingly. In these scenarios, human motion encompasses both gestures and activities: gestures entail hand movements conveying messages to either another person or a machine, while activities involve general body movements such as walking, running, or playing tennis, among others.

The large amount of data from additional sensing tools in clinical settings can overwhelm clinicians, requiring considerable time and expertise for analysis. To address this issue, the work in [15] Hernandez et al. shows how data dimensionality reduction techniques like Adversarial Autoencoder (AAE) can simplify time-series signals into a 2D latent space, facilitating analysis and identifying signal similarities. This data representation not only facilitates experts but also serves as a valuable self-assessment tool for non-experts, such as patients in home rehabilitation or

athletes in training.

Accurate measurement and estimation of joint kinematics (joint angle) are crucial for developing analysis tools in fields like rehabilitation, sports science, and ergonomics. The gold standard for estimating joint angles is through motion capture systems, but these are expensive and require a dedicated laboratory setting, limiting their accessibility and usability.

Comprehending human activity encompasses both recognizing specific activities and uncovering patterns across different activities such as sitting, standing, walking, and stair climbing, based on input from wearable sensors such as inertial sensors (i.e. IMUs) , or external sensors like motion sensors, cameras, and depth sensors.

Ground reaction force (GRF) sensors offer a valuable approach for Human Activity Recognition (HAR), providing comprehensive data on user movement dynamics [15]. Traditional force plates are costly and require trained personnel, but a cost-effective alternative is the Wii Balance Board (WiiBB), equipped with pressure sensors to estimate the center of pressure variation. The WiiBB, with its Bluetooth connection, allows real-time data acquisition, is affordable, easy to set up, and addresses privacy concerns. Numerous studies have demonstrated the WiiBB's reliability in various applications, including balance assessment, postural instability analysis, weight-bearing distribution, and functional recovery of standing balance in different populations.

Furthermore according to [14] by Hernandez et al., inertial measurement units (IMUs) are now lightweight, affordable, and energy-efficient, providing a practical alternative for joint angle measurement. They can be worn on the body or integrated into clothing, allowing continuous monitoring in various environments. With wireless data transmission, IMUs are suitable for real-time and remote monitoring in sports science, clinical biomechanics, and human/robot interaction. While IMUs don't directly measure joint angles, they can estimate them using 3D acceleration and angular velocity signals. However, challenges such as drift, noise, and movement impact on the body make accurate pose estimation a complex task with IMU systems.

In this work, we utilize data collected from RGB cameras and process it using the methodologies outlined in preceding chapters, with a focus on detecting sports movements and classifying various typical exercises related to both upper and lower body. Applying these algorithms in sports science and healthcare can significantly promote a healthy lifestyle, deter unhealthy habits, and facilitate condition tracking. We will classify the exercises using the random forest algorithm for time series in [8] and the multi-layer perceptron. An overview of the two classifiers is given in *Section 6.2* and *Section 6.3* following the theoretical concepts covered by [20] (Chapter 3 and Chapter 4) and by [9].

6.2 Random forest

Random Forest is a powerful ensemble machine learning algorithm used for both classification and regression tasks and it is considered to be one of the most effective and versatile algorithms in the field of machine learning. The term "ensemble" refers to its ability to combine the predictions of multiple individual models (decision trees) to make more accurate and robust predictions.
Random Forest in fact is built upon a collection of decision trees, which are simple models that make decisions by splitting the data into subsets based on the values of input features and then assigning a label or value to each leaf node.

6.2.1 Decision tree

Decision tree learning is an approach to approximate target functions with discrete values. In this method, the acquired function is portrayed through a decision tree structure. These learned trees can also be presented as collections of if-then rules, enhancing their comprehensibility to humans.
The classification process in decision trees involves the traversal of instances through the tree, starting from the root and ending at a leaf node, which provides the final classification for the instance. Each node within the tree corresponds to a test involving a specific attribute of the instance, and each branch stemming from a node represents one of the potential attribute values. To classify an instance, one commences at the root node, evaluates the attribute mentioned at that node, and subsequently proceeds down the branch corresponding to the attribute value exhibited by the example. This sequential procedure is iteratively applied as one navigates through the sub-tree rooted at each new node encountered along the way.

In *Figure 6.1* a binary classification of whether is it convenient to play football or not based on the weather is shown. Here there are three independent variables (outlook, humidity and wind) to determine the dependent variable (play football or not).
Instances are described by a fixed set of attributes (e.g., Outlook) and their values (e.g., sunny, overcast, rain). The most straightforward scenario for training decision trees occurs when each attribute assumes a limited set of non-overlapping values. The intended function produces distinct discrete result values. Decision tree techniques can readily be expanded to grasp functions with more than just two possible output values. A more substantial adaptation even allows the learning of target functions that yield real numerical results, although the utilization of decision trees in such cases is not as widespread.

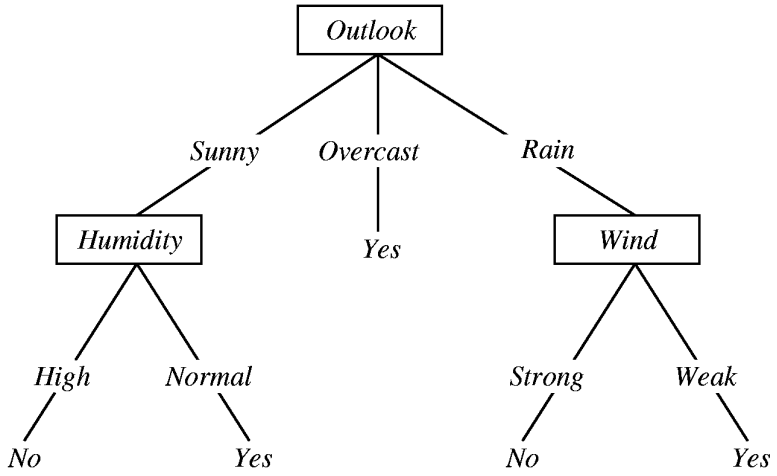

Figure 6.1: Decision tree outlook example

6.2.2 ID3 algorithm

The majority of algorithms devised for acquiring decision trees are variations of a fundamental procedure that employs a top-down, eager exploration of the potential decision tree configurations. An illustration of this approach is found in the ID3 algorithm (Quinlan 1986) and its successor, C4.5 (Quinlan 1993).

ID3 learns decision trees by crafting them from the top down, beginning with the question, "which attribute should serve as the root node's test?" To resolve this query, every attribute of the instances is scrutinized using a statistical evaluation to evaluate its effectiveness in classifying the training examples independently.

The most suitable attribute is chosen and employed as the test criterion at the root node of the tree. Subsequently, a descendant node is generated for each feasible value of this attribute, and the training examples are directed to the appropriate descendant node (i.e., along the branch corresponding to the example's attribute value). This entire procedure is then reiterated using the training examples associated with each descendant node to determine the optimal attribute for testing at that specific point in the tree.

6.2.3 Entropy and information gain

A central point in the ID3 algorithm is to measure how well a given attribute separates the training examples according to their target classification.

For a general discrete distribution of the output values $p_1, ..., p_m$ with $p_i \geq 0$ and $\sum_i p_i = 1$, the entropy is defined as

$$H = -\sum_{i=1}^{m} p_i log_2 p_i \tag{6.1}$$

When working with a data set, denoted as 'S,' the concept of 'Entropy' within the range of [0, log2 m] serves as an indicator of the degree of impurity present in the samples. A low entropy value means that the elements in S tend to predominantly belong to a single class. For instance, if all elements are labeled as "YES," the entropy is reduced to zero.
On the other hand, a high entropy value suggests that the elements in S are mixed. For instance, when the elements are divided equally between "YES" and "NO," the entropy is at its maximum, which is one.
With entropy as a means to evaluate the impurity within a set of training examples, we can now establish a metric for assessing the effectiveness of an attribute in classifying the training data. This metric is referred to as "Information Gain" (IG), and it quantifies the expected reduction in entropy resulting from the partitioning of examples based on a specific attribute.

More precisely, the information gain IG(y,A) of attribute A relative to a collection S is defined as

$$IG(y, A) = H(y) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} H(y_v) \tag{6.2}$$

where values(A) is the set of all possible values for attribute A, and S_v is the subset of S for which attribute A has value v.

The previous formula is related to the conditional entropy of y given A

$$\begin{aligned} H(y|A) &= \sum_{v \in values(A)} Prob\{A = v\} H(y|A = v) \\ H(y|A = v) &= -\sum_{i=1}^{m} Prob\{y = y_i|A = v\} log_2 Prob\{y = y_i|A = v\} \end{aligned} \tag{6.3}$$

Given those information, the information gain is thus defined as:

$$IG(y, A) = H(y) - H(y|A) \tag{6.4}$$

Therefore, IG is the expected reduction in entropy caused by knowing the value of attribute A.

6.2.4 Ensamble methods

Plain DT may overfit and have high variance on the validation data: what it can be done to reduce it is averaging over different models (when predictions are independent):

$$var(\bar{x}) = \frac{var(x)}{N} \tag{6.5}$$

The challenge associated with generating multiple models for subsequent averaging lies in the limitation of having just one training data set. To overcome this constraint, we must introduce additional data sets.
In the case of Random Forest, a methodology known as "Bootstrap Aggregating" or "Bagging" comes into play. This technique entails the individual training of multiple decision trees on randomly selected subsets of the data, followed by the merging of their predictions. This approach effectively mitigates issues related to high variance and overfitting that are commonly encountered when working with decision trees, as shown in *Figure 6.2*.

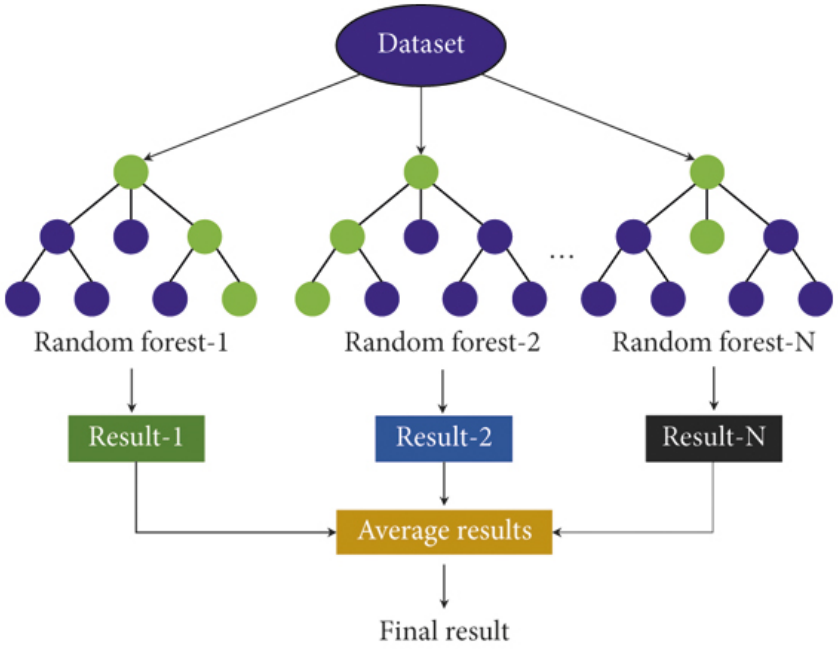

Figure 6.2: Random forest algorithm

For classification tasks, each decision tree in the Random Forest makes a prediction, and the final prediction is determined by a majority vote or averaging among the trees.
The "random" aspect in Random Forest comes from two sources:

- *Bootstrapped Sampling*: when creating each individual decision tree in the forest, a random subset of the training data (with replacement) is used. This means that each tree is trained on a slightly different data set.
- *Feature Randomness*: at each node of a decision tree, a random subset of features (input variables) is considered for splitting the data. This helps to decorrelate the trees and make the ensemble more robust.

The ensemble nature of Random Forest provides several advantages, including improved accuracy, robustness to noisy data, and the ability to handle high-dimensional feature spaces. It's also less prone to overfitting compared to a single decision tree.

6.3 Multi-layer perceptron

A Multi-layer perceptron (MLP) is a type of artificial neural network (ANN) that is widely used in machine learning for various tasks, including classification and regression. In this work the MLP architecture has been used as classifier, so only this task will be taken into account. As MLP is the most basic Neural Networks architecture nowadays, a brief overview of how the classifier works will be given.
Neural networks are loosely inspired on some early theories of human neuron functionality. Our brain has about 10e11 neurons, each of which communicates with (is connected to) about 10e4 other neurons.

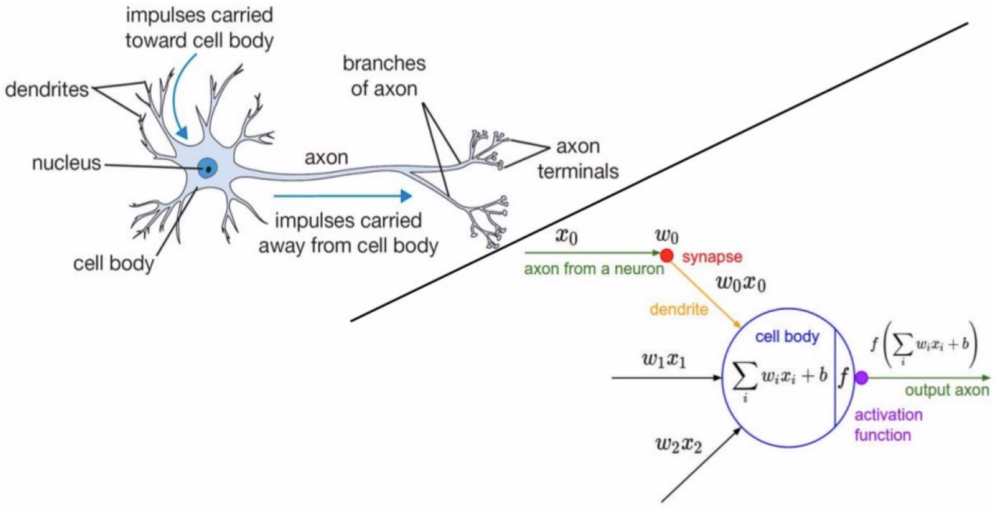

Figure 6.3: Neurons biological similitude

MLP is a supervised learning algorithm with labeled training data $(x^{(i)}, y^{(i)})$, where $x^{(i)}$ is the i-th input and $y^{(i)}$ is the corresponding response.
In classical Neural Networks each neuron is a linear combination of input features , followed by a non-linear activation function, essential for generalization. It gives a way of defining a complex, non-linear form hypotheses $h_{W,b}(x)$, with parameters W, b that we can fit to our data.
The key components of an MLP neural network that will be discussed in the following sections are:

- *Input Layer*: the input layer consists of one or more neurons (nodes) that receive the initial input data. Each neuron in the input layer corresponds to a feature or attribute of the input data.

- *Hidden Layers*: between the input and output layers, there can be one or more

hidden layers. These hidden layers contain neurons that perform transformations on the input data using weighted connections and activation functions.

- *Output Layer*: the output layer produces the final result of the network's computation. The number of neurons in the output layer depends on the specific task. For example, in a binary classification problem, there might be one output neuron that indicates the probability of belonging to one of the two classes. In a multi-class classification problem, there would be one output neuron for each class.
- *Weights*: each connection between neurons in adjacent layers is associated with a weight. These weights are learned during the training process and determine the strength of the connection between neurons. Learning involves adjusting these weights to minimize the error in the network's predictions.
- *Activation Functions*: each neuron in the hidden layers and sometimes in the output layer is associated with an activation function. Activation functions introduce non-linearity into the network, allowing it to model complex relationships in the data. Common activation functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU).

6.3.1 Single neuron

Considering the case of a single neuron shown in *Figure 6.4*, it takes as input $x \in \mathbb{R}^n$ and a "+1" intercept term and outputs

$$y = h_{W,b}(x) = f(b + w^T x) = f(b + \sum_{i=1}^{n} w_i x_i) \tag{6.6}$$

where $f : \mathbb{R} \to \mathbb{R}$ is called the activation function.

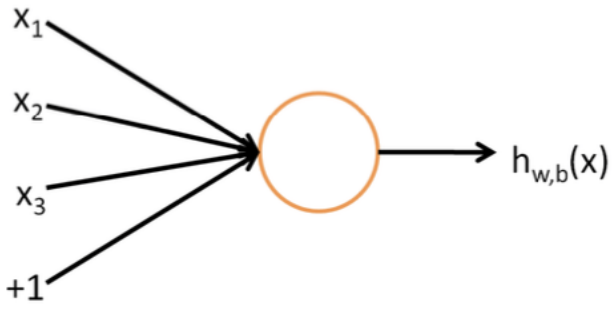

Figure 6.4: Single neuron

The linear combination part is computationally simple, but it contains many coefficients, hence multiplications and additions ($y = wx + b$).

The non-linear (i.e. activation function) is one of several non-linear functions and is essential for generalization. Some typical activation functions are:

- Sigmoid
- Hard threshold
- Hyperbolic tangent
- ReLu
- Soft ReLu

In order to learn any function, it is sufficient to have one hidden layer of neurons. More layers help speed up the training by reducing the overall network size.

6.3.2 Multi-layer architecture

A neural network is formed by hooking together many simple neurons.
The "+1" inputs are called bias units, and correspond to the intercept term.
The leftmost layer is called the input layer and the rightmost layer the output layer, while the middle layer is called hidden layer. A typical architecture is shown in *Figure 6.5.*

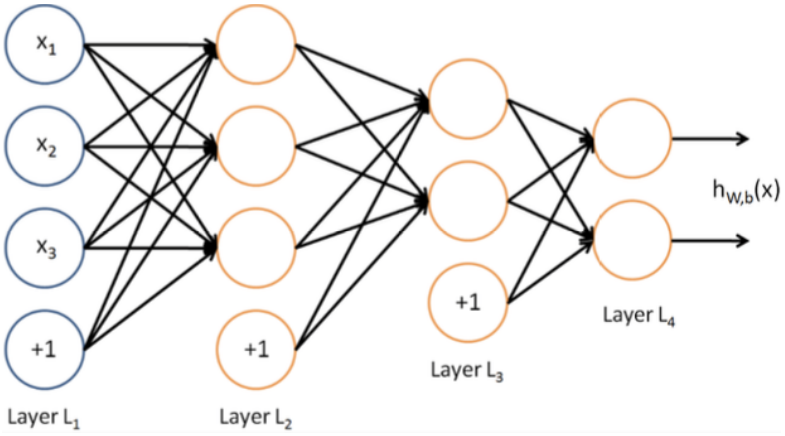

Figure 6.5: Artificial Neural Network architecture

We let n_l denote the number of layers (i.e. the depth of the neural network).
Our neural network has parameters $(W, b) = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$, where we write $W_{ij}^{(l)}$ to denote the parameter (or weight) associated with the connection between unit j in layer l and unit i in layer l+1. Also $b_i^{(l)}$ is the bias associated with unit i in layer l+1.

We also let s_l denote the number of nodes in layer l. This is the number of features of layer l.
We will write $a_i^{(l)}$ to denote the activation of unit i in layer l.
We shall also let $z_i^{(l)}$ denote the total weighted sum of inputs to unit i in layer l, including the bias term (e.g. $z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1} x_j + b_i^{(1)})$) so that $a_i^{(l)} = f(z_i^{(l)})$.
In general, for $l = 1, ..., n_l - 1$ let $W^l \in \mathbb{R}^{s_{l+1}, s_l}$, $b^l \in \mathbb{R}^{s_{l+1}}$ denote the parameters of layer l+1. We can compute activations at any layer via a non-linear recursion, initialized with $a^{(1)} = x$: for $l = 1, ..., n_l - 1$:

$$\begin{aligned} z^{(l+1)} &= W^{(l)} a^{(l)} + b^{(l)} \\ a^{(l+1)} &= f(z^{(l+1)}) \end{aligned} \tag{6.7}$$

The equations provided above outline a typical feedforward neural network (FFNN). This neural network is characterized by its connectivity structure, which lacks any directed loops or cyclic connections. The FFNN has the remarkable capability to effectively approximate a wide range of functions with great precision.

In the context of neural networks, it's crucial that the network's output unambiguously signifies the correct output class. However, the outputs of the neurons in the final layer may not necessarily have uniform magnitudes or weights. To address this concern, the SoftMax function is employed as the output layer in scenarios involving classification problems.

$$h_{W,b}(x)_i = softmax(z^{(n_l)})_i = \frac{exp(z_i^{(n_l)})}{\sum_j exp(z_j^{(n_l)})} \tag{6.8}$$

where $z_i^{(n_l)}$ is the i-th element of $z^{(n_l)} = W^{(n_l)} a^{(n_l)} + b^{(n_l)}$ In this way we can have a vector of arbitrary values (activations from the last layer) and give back as output a vector of values whose sum is always 1.

Given a network topology and parameters W,b we can measure the mismatch, or error, between a training example (x,y) and the network response $h_{W,b}(x)$:

$$e_{W,b}(x, y) = h_{W,b}(x) - y \tag{6.9}$$

6.3.3 Training

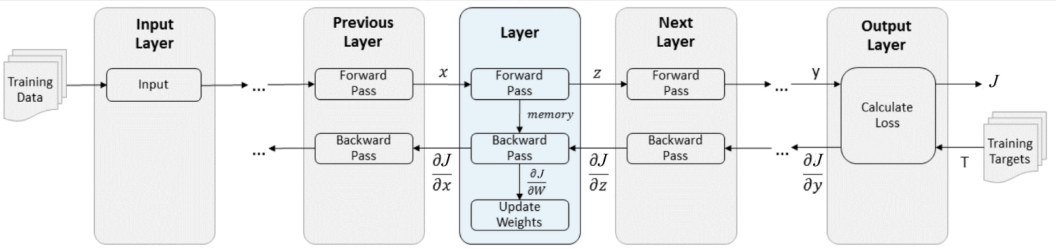

Figure 6.6: FFNN training

Training a FFNN is about finding values of the network parameters (W,b) so to make the training error J(W,b) small (cross-entropy for classifier). The goal of training is to minimize J(W,b) as a function of W and b. To train our neural network we will initialize each parameter W_{ij} and each b_i to a small random value near zero and then apply an optimization algorithm, such as stochastic gradient descent. One iteration of gradient descent updates the parameters W,b as follows:

$$\begin{aligned} W_{ij}^{(l)} &\leftarrow W_{ij}^{(l)} - \alpha \frac{\partial J(X,b}{\partial W_{ij}^{(l)}} \\ b_i^{(l)} &\leftarrow b_i^{(l)} - \alpha \frac{\partial J(X,b}{\partial b_i^{(l)}} \end{aligned} \tag{6.10}$$

where α is called the learning rate (a too low learning rate may result in slow convergence, a too high learning rate may lead to unstable behaviors).

An efficient recursive method for computing the above derivatives is called the back propagation algorithm, which is essentially based on the chain rule for computing derivatives. The algorithm works as follow:

1. *Initialization*: initialize the linear part of the neurons with random weights
2. *Propagate*: after presenting an input data propagate to the output and compute the prediction error.
3. *Adjust*: adjust weights based on estimated impact on error. Use sensitivity i.e. partial derivative of output with respect to the internal and primary inputs to propagate error backwards.
4. *Repeat*: repeat for all the of data in the training set. For efficiency, it can be mini-batched (propagate N input patterns, back-propagate the cumulative error and update the weights).

6.4 Evaluation metrics

In classification, the simplest way to measure the performance is the accuracy, which is defined as:

$$Accuracy = \frac{\text{number of correct predictions}}{\text{total number of predictions}} \tag{6.11}$$

However, accuracy doesn't tells everything about the performance of the classification. For instance, for a binary classification (1 or 0), there may be the case where mistakenly classify 1 as 0 can have way worst consequences than mistakenly classify 0 as 1. Moreover, the population over which we are classifying may be highly unbalanced, resulting in very poor evaluation when using only the accuracy.

For these reasons, we can distinguish between Type I and Type II errors:
- Type I (False Positives, FP): individuals that are classified as positive, while they are negative in reality.
- Type II (False Negatives, FN): individuals that are classified as negative, while they are positive in reality.

Given this distinction, we can define:

- Precision (PPV): the probability that the true state is positive, given that the sample is classified as positive.

$$p = Prob\{x = 1|y = 1\} \tag{6.12}$$

- Sensitivity (TPR, or recall): the probability that the classifier returns positive, given that the sample is positive in reality.

$$r = Prob\{y = 1|x = 1\} \tag{6.13}$$

- Specificity (TNR): the probability that the classifier returns negative, given that the sample is negative in reality.

$$s = Prob\{y = 0|x = 0\} \tag{6.14}$$

These main classification performance criteria can be estimated more easily by constructing the so called "Confusion Matrix", a 2x2 matrix that suitably arranges the number of samples that fall in the four possibilities (TP, FP, FN, TN).

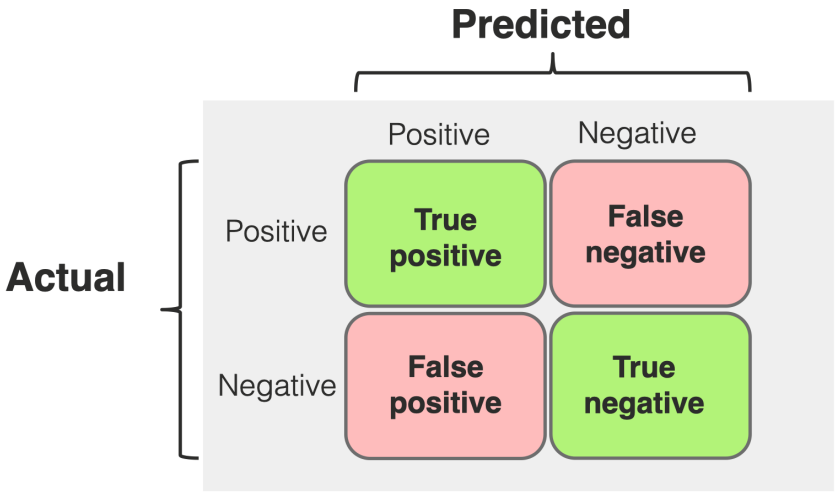

Figure 6.7: Confusion matrix

Using this matrix, we can thus define:

- Precision p: the number of TP divided by the number of all classified positive results.

$$p = Prob\{x = 1|y = 1\} = \frac{TP}{TP + FP} \tag{6.15}$$

- Recall r: the number of TP divided by the number of total actual positives.

$$r = Prob\{y = 1|x = 1\} = \frac{TP}{TP + FN} \tag{6.16}$$

- Specificity s : the number of TN divided by the number of total actual negatives.

$$s = Prob\{y = 0|x = 0\} = \frac{TN}{FP + TN} \tag{6.17}$$

In this work, since we will classify exercises based on the joint angles time-series, a confusion matrix will be built and the accuracy will be considered as performance metric, since we don't have any substantial difference within the wrong classifications.

Chapter 7

Experimental setup and results

7.1 Sensor fusion and Mocap

7.1.1 Sensor fusion setup

The parameters used in this work were threshold distance (cm) for dividing fast and non-fast measurement θ_f and scaling factor γ in *Algorithm I*, set to 3 cm and 3.0 respectively. While the input transition matrix B was considered to be equal to 0, the state transition matrix A, measurement matrix H, process noise matrix Q and measurement noise matrix R, were defined as follows

$$A = \begin{bmatrix} 1 & 0 & 0 & \delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.1}$$

where $\delta t = 30ms$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{7.2}$$

$$Q = \begin{bmatrix} 0.001 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.001 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix} \tag{7.3}$$

$$R = \begin{bmatrix} v & 0 & 0 \\ 0 & v & 0 \\ 0 & 0 & v \end{bmatrix} \tag{7.4}$$

The value of v was set to $v_{in} = 0.005$ when the cameras were coherent identifying the joint as fast or slow and and $v_{high} = 1$ when the joint was not detected by the camera. Any other intermediate values were set following *Algorithm I.*

The fused skeletons resulting from the above methods are shown in *Figure 7.1.*

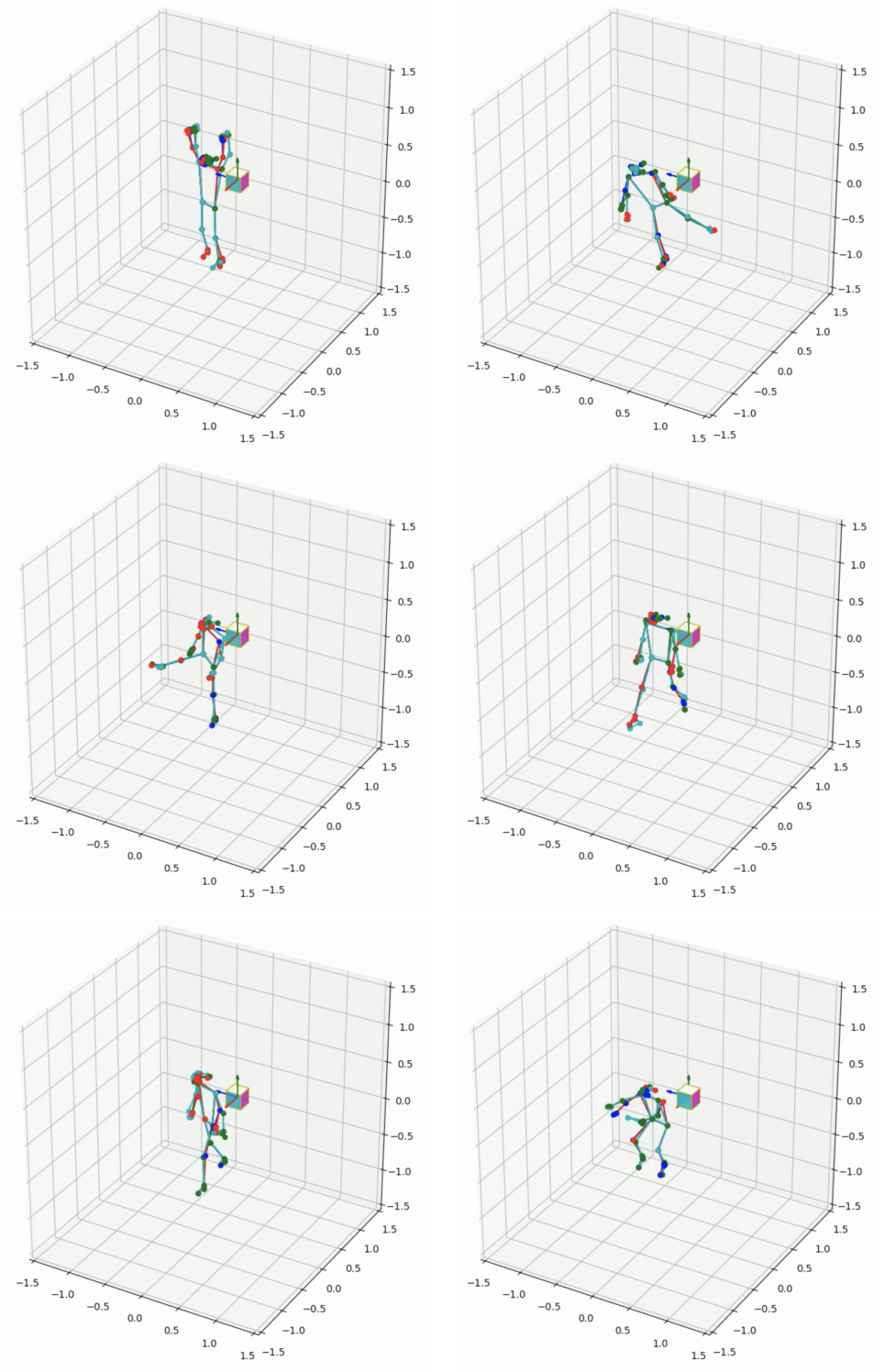

Figure 7.1: Fused skeletons: ■ Mean; ■ Kalman filter; ■ Mean + DBSCAN; ■ Kalman filter + DBSCAN

7.1.2 Mocap setup

In this study, the motion capture system has been used as a groundtruth to evaluate the accuracy of the methods illustrated in *Chapter 4*. In order to do that, the following six exercises illustrated before have been performed by the subject and data from motion capture system and cameras have been acquired simultaneously.

The position of the markers have been chosen in order to respect the human physiognomy following the guidelines provided by [32] and [33] and make the comparison with the 33 landmarks acquired using the MediaPipe framework in *Chapter 3* and fused in *Chapter 4* as reliable as possible. Then, cutting and synchronizing the data was essential to compute the mean squared error between the points in the proper way.

A set of 40 markers was applied as shown in *Figure 7.2* in such way that the corresponding landmark position could be easily computed. For instance, the point corresponding to the elbow landmark was computed as the mean value of the HME and HLE markers. The same reasoning was used for all the other markers.

The configuration of the marker placed on the subject is shown in *Figure 7.3*.

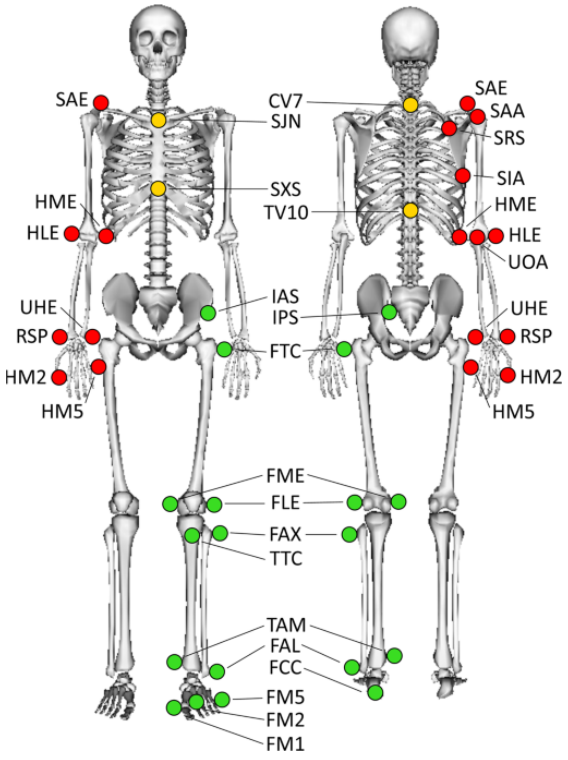

Figure 7.2: Markers set

((a)) Front view

((b)) Raw view

((c)) Side view

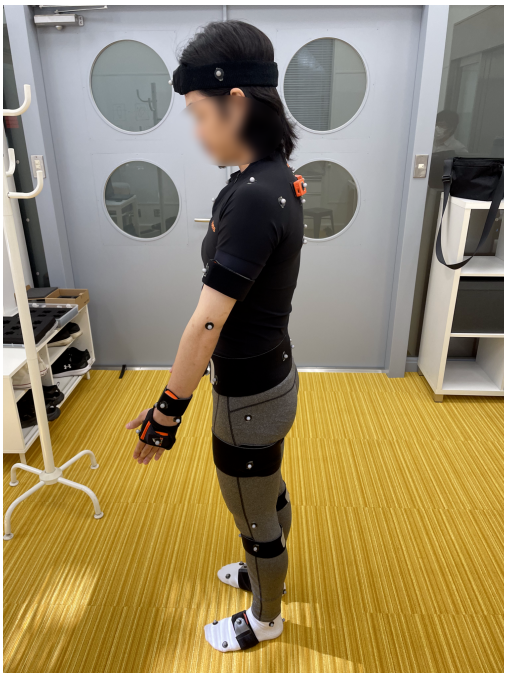

Figure 7.3: Markers configuration

7.1.3 Experimental results

The fusion methods to be evaluated are:

- Mean value
- Kalman filter
- Mean value + DBSCAN
- Kalman filter + DBSCAN

We compared the motion capture system with both the four different fusion method and also with the individual cameras.

The results demonstrates a significant reduction in errors and inaccuracies associated with single sensors, resulting in more reliable and precise measurements. When using individual cameras, the error between the Mocap system and detected joint positions ranged from 10 cm to 25 cm. However, by combining joint positions, these errors decreased to a maximum of 10 cm with a variance of . Notably, the average error was generally smaller (around 5 cm) for joints like elbows, shoulders, hips, and knees, which are crucial for human activity recognition, particularly in sports science. For instance, while in exercises like the dumbbell shoulder press we can notice a smaller variance of the errors, around 3 cm, in other exercises we could have a larger variance. This type of problem though must be interpreted: when considering lower body exercises like the squat, which has larger variance, particular emphasis must be given to the detection of the position of the hips and the knees. In our case these two joints were the ones that gave the smallest error, while the biggest ones, right and left wrist and hand, are way less significant.
This improvement in accuracy will be advantageous in the next phase of the study.

The results of the comparison of each one of the six exercises introduced in *Chapter 4* are shown in *Figure 7.4 - 7.9*.

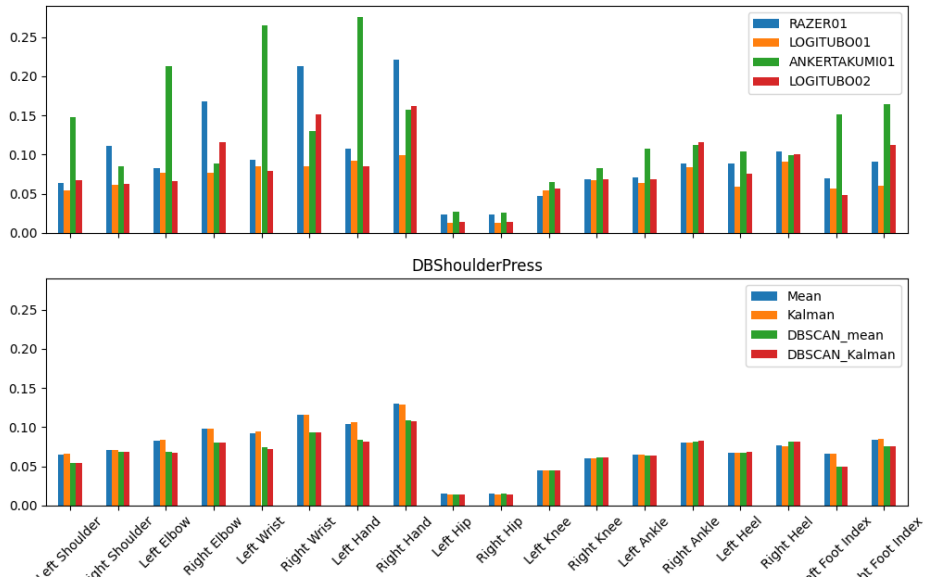

Figure 7.4: Dumbbell shoulder press comparison

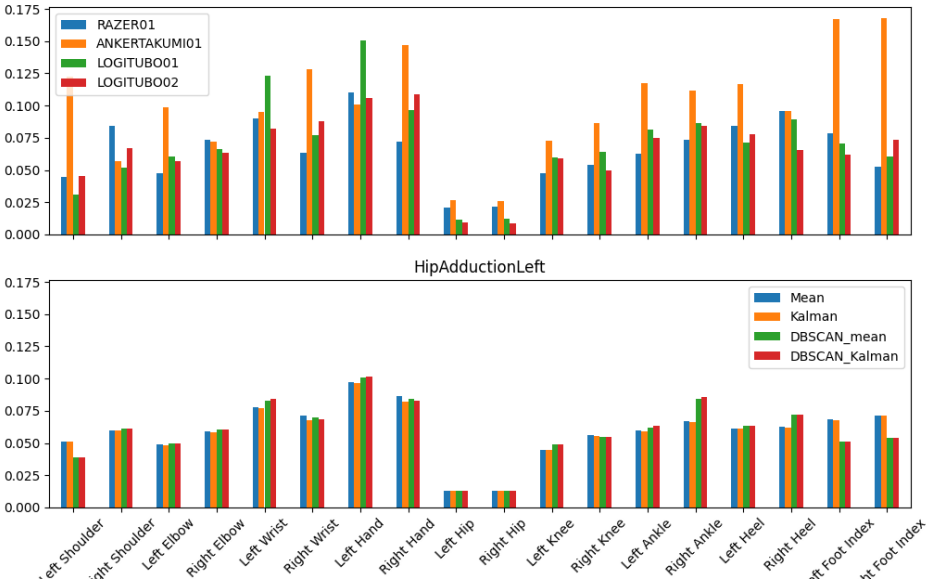

Figure 7.5: Hip adduction left comparison

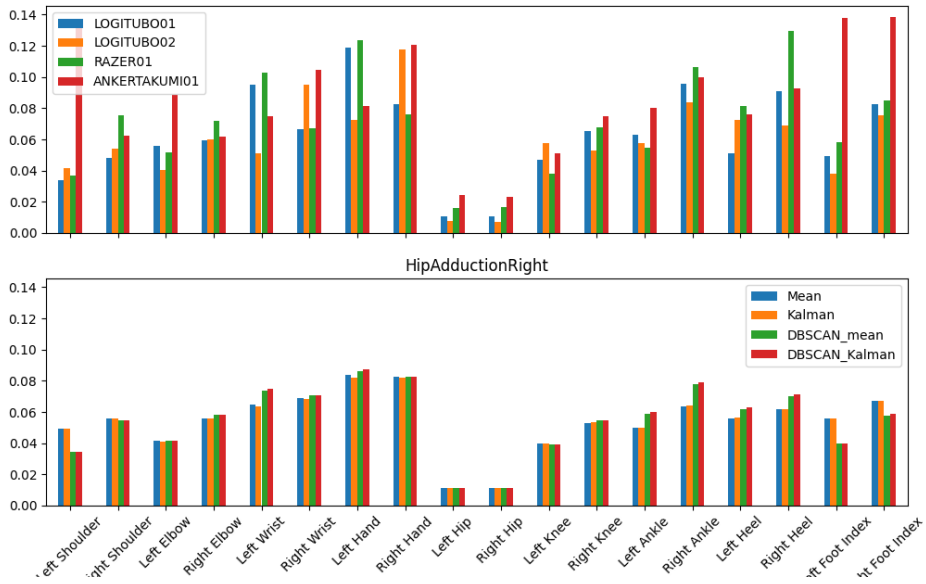

Figure 7.6: Hip adduction right comparison

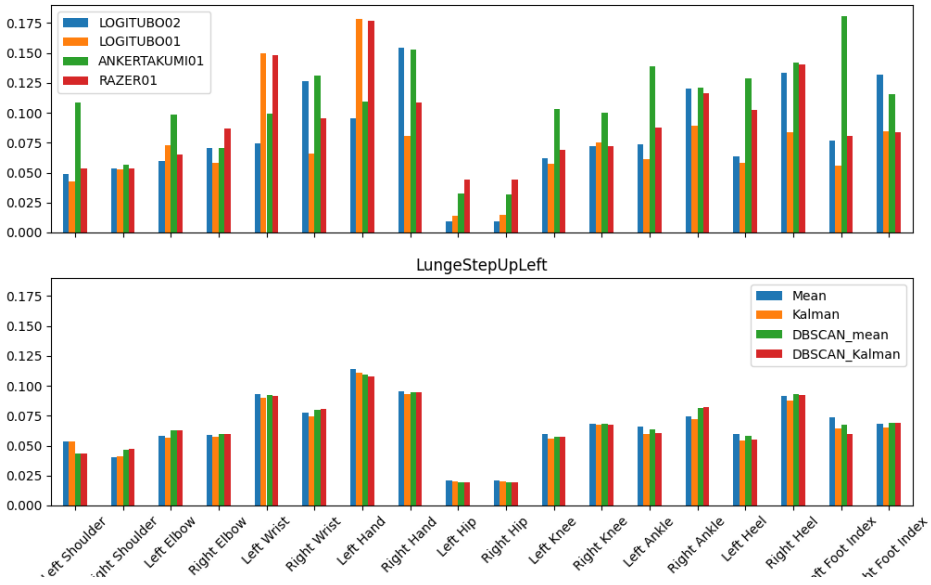

Figure 7.7: Lunge step up left comparison

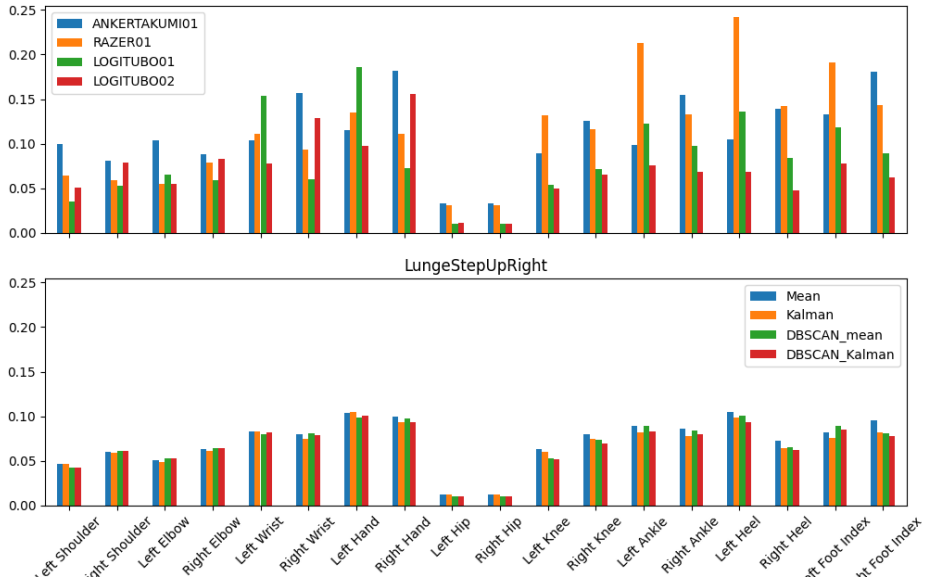

Figure 7.8: Lunge step up right comparison

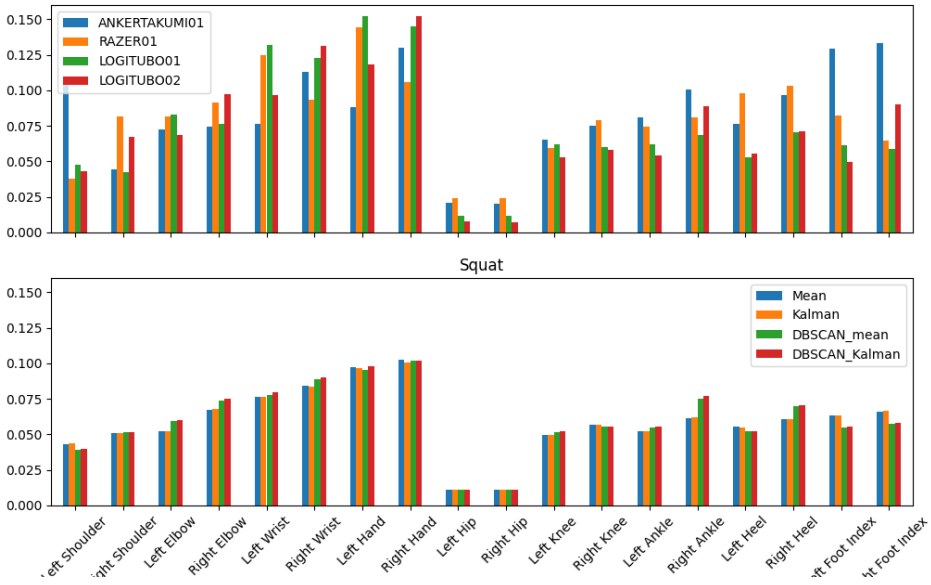

Figure 7.9: Squat comparison

7.2 Human activity recognition

7.2.1 Data set: exercises

The data set used to train, validate and test the human activity classifier were built collecting videos of 24 exercises, performed by 4 subjects. We used 2 subjects to train the model, 1 subject to validate, and 1 subject to test it, using a nested K-fold cross-validation in order to train the models with different partecipant and test them with completely unknown data.

In each video the subject performed 10 repetition of each exercise. The videos have been cut to extract each one of the 10 repetition and re-sampled to 50 frames per rep, resulting in a total of 960 elements composing the data set. We considered the single repetition as feature for our data set and not the whole video: in this way it was possible to eliminate unwanted variability in the features such as different rest time between repetitions and biases due to the characteristics of the different subjects and instead we were able to capture variabilities on the single rep, allowing to make the classifier more efficient. Moreover we overcome possible issues like excessively long signals and the bound of testing the model with a signal that had to be the same length of the training data (10 reps).

The exercises performed by the subjects were the following:

- *Calf raise*
- *Dumbbell bend over raise*
- *Dumbbell bend over row*
- *Dumbbell biceps curl*
- *Dumbbell front raise*
- *Dumbbell lateral raise*
- *Dumbbell shoulder press*
- *Front kick left*
- *Front kick right*
- *Hip adduction left*
- *Hip adduction right*
- *Lunge left*

- *Lunge right*
- *Lunge side left*
- *Lunge side right*
- *Lunge step-up left*
- *Lunge step-up right*
- *Plank jump-in*
- *Push-up knee*
- *Shoulder tap left*
- *Shoulder tap right*
- *Squat*
- *Thigh tap left*
- *Thigh tap right*

7.2.2 Data set: joint angles

The key part of the method is the selection of features to create feature vectors. Using the sensor fusion algorithm introduced in *Chapter 4*, it was possible to extract from each video the position in the 3D space of 33 joints of the human body (e.g. the MediaPipe landmarks).
The work of Uddin and his team demonstrates that the measurement of angles between body parts, such as shoulders, elbows, knees, and the pelvic region, furnishes valuable data for the recognition of 3D human activities [30]. For this reason we did not consider the raw 3D position of the joints to build our data set, but we computed the joint angles, as shown in *Figure 7.10*, only for the most representative joints of the human body out of the 33 detected in the previous chapters. We in fact computed the joint angles of:

- Wrists
- Elbows
- Shoulders
- Hips
- Knees

- Ankles

Moreover, when considering the shoulders and the hips, a particular focus must be given to the fact the these joints, rather then elbows and knees, are spherical joints. For this reason the computation of the joint angles has been performed considering three different planes of motion.

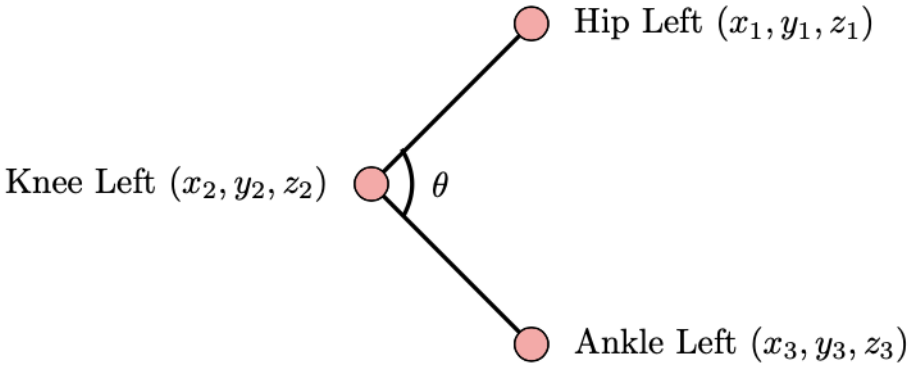

Figure 7.10: Knee angle computation

The value of the angle between two different joints j_1 and j_2 can be computed as done by [37] by estimating the locations of the two joints with respect to a reference joint r in the 3D space, as follows

$$a_{j_1,j_2} = cos^{-1}(\frac{r\vec{j_1}\dot{r}\vec{j_2}}{(||rj_1||)\dot{(}||rj_2||)}) \tag{7.5}$$

In this equation, $r\vec{j_1}$ represents the distance between joint j_1 and the reference joint r in 3D space. Similarly, $r\vec{j_2}$ represents the distance between joint j_1 and the reference joint r in 3D space. Lastly,$||rj_1||$and $||rj_2||$ represent the lengths of these vectors, respectively.

The behaviour of the left knee during 10 reps of squats is shown in *Figure 7.11* clearly shows the behaviour of the left knee angle of the subject performing 10 reps of squats, computed using the above described method. The variation of the angles of each joint during the exercises will be used as feature vectors.

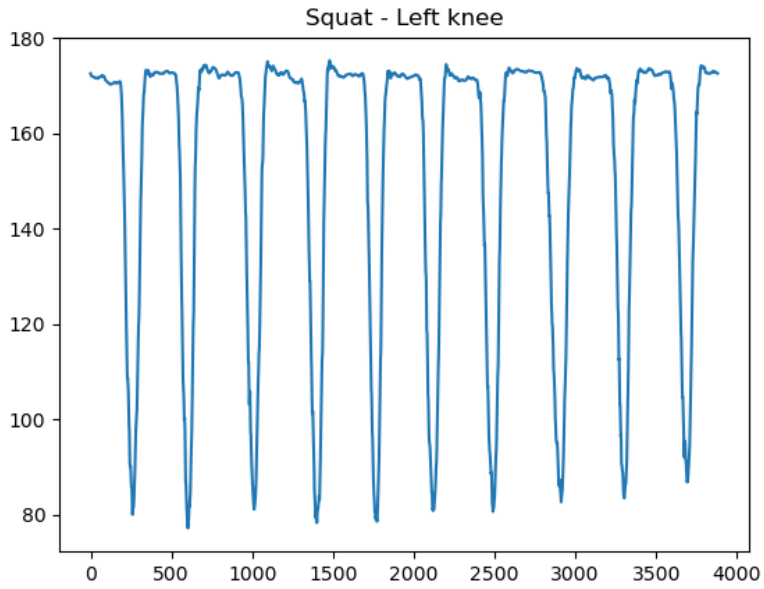

Figure 7.11: Knee angle variation during squat

7.2.3 Experimental results

The two models have been built using the *Scikit Learn* library in Python. The accuracy of the classification has been evaluated using different hyperparameters for the two models, in order to pick the one that could achieve the best performances. In particular, the random forest model has been trained keeping n_estimators = 100 and looping for max_depth = [1, 2, 3, 4, 5, 10, 15, 20, 30]. Selecting the best hyperparameter, which was 10, we then trained keeping max_depth = 10 and looping for n_estimators = [10, 20, 30, 50, 100, 150, 200, 300, 500], finding the best results for n_estimators = 200. Regarding the MLP model, we trained and validated the model combining the following parameters:

- hidden_layer_sizes: [(64, 64, 64), (64, 128, 256), (256, 128, 64)]
- activation: ['relu', 'tanh', 'logistic']
- learning_rate: [0.0005, 0.001, 0.01]

Therefore, tuning the parameters of the classifiers we achieved the best performances using:

- Random Forest:

  - *max_depth* = 10
  - *n_estimators* = 200

- Multi-layer Perceptron:
  - *hidden_layer_sizes* = (64, 128, 256)
  - *activation* = hyperbolic tangent
  - *learning_rate* = 0.0005

The data set used to train, validate and test the human activity classifier were built using 24 exercises, performed by 4 subjects. We used 2 subjects to train the model, 1 subject to validate, and 1 subject to test it, using a nested K-fold cross-validation in order to train the models with different participant and test them with completely unknown data.

Testing the models on each of the four participant, we obtained the results shown in *Table 7.1*.

Classifier	Participant	Accuracy	Mean value	Standard deviation
Random forest	1	82.08%	90.29%	5.1%
	2	94.54%		
	3	94.58%		
	4	89.96%		
Multi-layer perceptron	1	79.58%	87.89%	9.87%
	2	97.48%		
	3	97.92%		
	4	76.57%		

Table 7.1: Classification results

Although the MLP could achieve higher levels of accuracy with some partecipants, the results show how for our data set the Random Forest classifier achieves generally better results, with a mean value within the four participants 2.4 % higher than the MLP, presenting also a smaller standard deviation.

The confusion matrices regarding the classification of each one of the four participants used as test data for both the classifier, shown in *Figure 7.12* and *Figure 7.13*, show how almost all the exercises can be well classified within the four participants in an homogeneous way.
The only exception are made for "Lunge Side Left" and "Lunge Side Right" of participant 1 for which both classifiers showed some difficulties, and "Lunge Step

Up Left" and "Lunge Step Up Right" of participant 4, which the MLP classifier could not correctly classify.

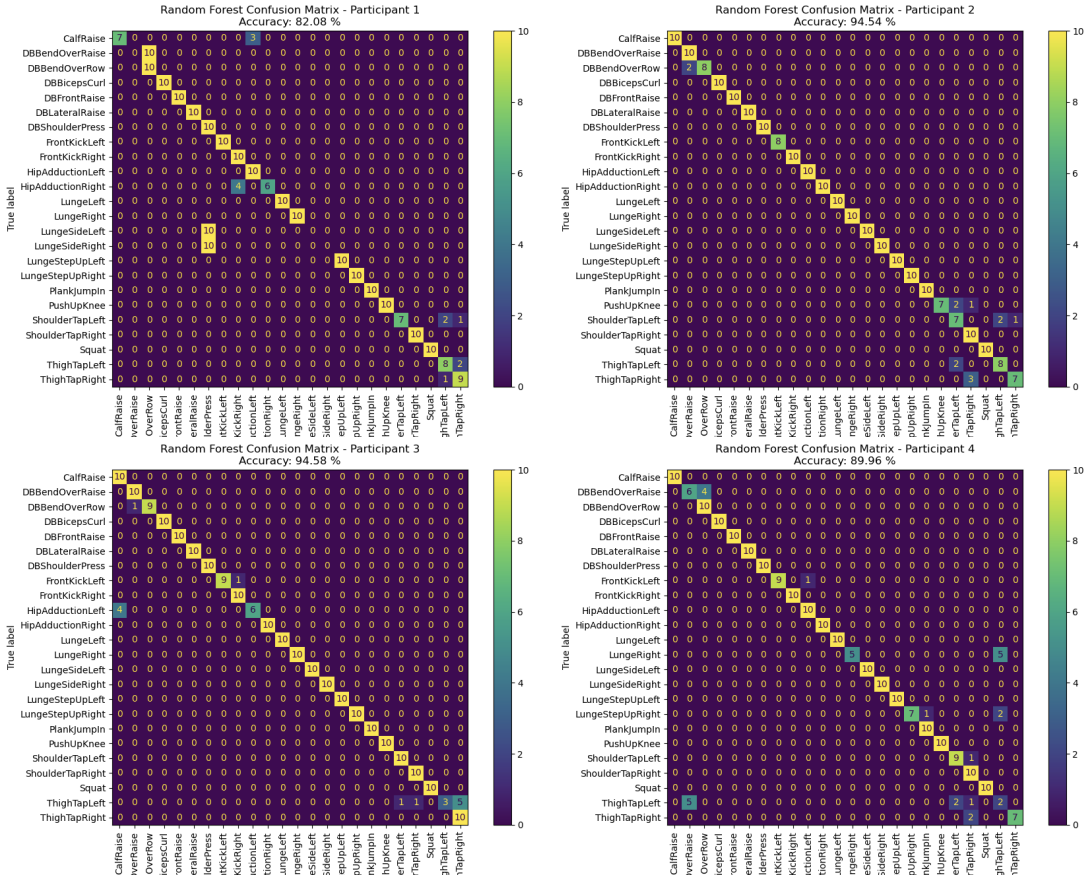

Figure 7.12: Random forest confusion matrices

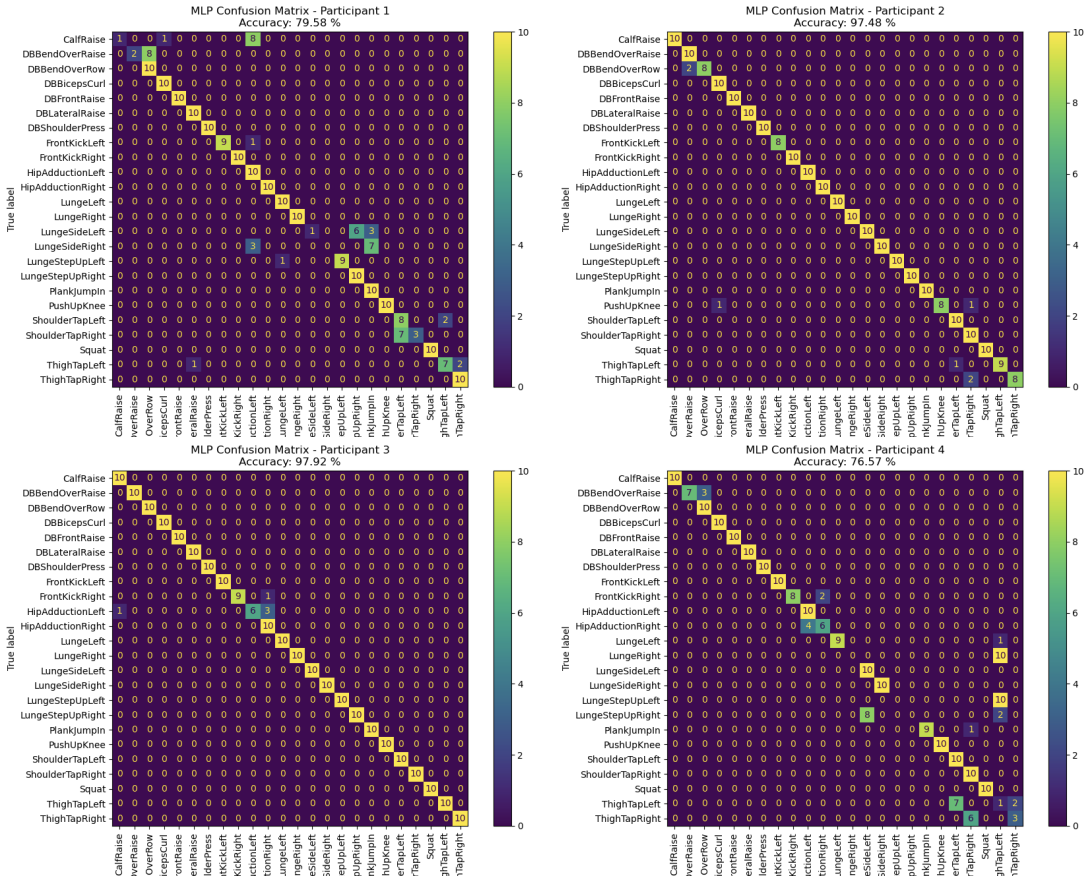

Figure 7.13: Multi-layer perceptron confusion matrices

7.3 Discussion and improvements

Human activity recognition based on 3D joint positions has to face challenges when dealing with different viewpoints, since the complexity arises from the ambiguity introduced by varying perspectives, making it challenging to consistently and accurately estimate the true 3D pose of individuals.
MediaPipe primarily relies on 2D images, which may not capture the full depth information accurately. This limitation can result in inaccuracies, especially in complex poses or when the subject is partially occluded. Even after the camera reference system rotation in fact, we can notice that even a simple T pose there is still some misalignment, especially when dealing with back side views. Moreover, dependencies on camera calibration parameters further contribute to such misalignment.
To address these challenges, the incorporation of a multi-view camera setup is required. Additionally, the integration of Inertial Measurement Units (IMUs) offers

a promising enhancement. IMUs capture accelerations and angular velocities, providing information less susceptible to visual occlusions. Moreover, IMU data can help in real-time adaptation to dynamic movements, improving the system's responsiveness to different human activities. By fusing IMU data with camera-based information, the system gains increased robustness and becomes less sensitive to changes in viewpoint, as inertial sensors capture motion independently of camera perspective, enhancing the overall accuracy of 3D joint position estimation.
A hybrid approach that combines 2D and 3D pose estimation techniques emerges as another viable strategy. Leveraging the strengths of both methods, this hybrid model can compensate for individual limitations, providing a more comprehensive understanding of human poses.

Chapter 8

Conclusion

Human Activity Recognition (HAR) uses pattern recognition and machine learning to classify human movements by linking physical actions to their 2D image representations. Affordable mobile sensors, such as Microsoft Kinect, which tracks skeletal joints, have become popular recently. This has resulted in new algorithms for activity recognition and tools for sports analysis and home-based motion rehabilitation.
In this work we exploited a sensor fusion algorithm to build a data set that has been used to classify exercises based on joint angles.
After calibrating multiple RGB cameras (four, in our case), we computed their relative position with respect to a fixed reference system, composed by ArUco markers. The fact that the only sensors one needs to use the developed algorithms are just RGB cameras was a key point, in order to build a system that could be cheap and easily usable, specially when it comes to motion rehabilitation at home without the help of a clinician.
To collect the data coming from the cameras, the MediaPipe framework was used to extract the 3D positions of the fundamental joints to describe the motion of the body.
Having the body pose with respect to the cameras reference system, we needed to rotate the data coordinates into the same reference, so that we could fuse all the data collected by the cameras. This was the key aspect of the research: the primary goal of sensor fusion is to improve the accuracy and ensure redundancy and robustness by combining data from multiple sensors. In case a sensor malfunctions or gives incorrect data in fact, the system can continue operating by relying on the other sensors. This enhances the system's resilience in difficult conditions like low visibility or bad weather, where a single sensor might not be dependable.

We exploited four different fusion techniques:

- Mean
- Kalman filter
- Mean + DBSCAN
- Kalman filter + DBSCAN

After that, in order to evaluate the precision of the different fusion techniques and also the advantage of fusing the data instead of considering single sensor, we used as groundtruth a motion capture system.
We tested our method collecting videos of a subject performing the 6 exercises described in *Chapter 4*.
The results showed how it has been possible to reduce errors and inaccuracies associated with the single sensor, leading to more reliable and precise measurements: the error between the Mocap system and the detected joint position was between 10 cm and 20 cm using the cameras singularly. Instead, when using the fused joint position, the errors decreased reaching a maximum of 10 cm. Moreover it can be noted that the average error was generally smaller (around 5 cm) when considering joints like elbows, shoulders, hips and knees and higher when considering the joints related to wrists, hands, ankles and feet. This will be an advantage within the next phase of the study, since when it comes to human activity recognition the joint angles of the elbows, shoulders, hips and knees are the most representative of the human motion, specially in sport science.
Regarding the comparison of the four fusion methods, there is a slight advantage using the Kalman filter over the simple mean value, empowered by the fact that the first can be more reliable in terms of robustness and noise rejection. When adding the DBSCAN to preprocess the data, we can notice a slight degradation of the performances within certain joints (more often wrists, hands, ankles, feet) and exercises. More often instead the DBSCAN results in a great improvement of the detection precision. We can conclude that with the use of the DBSCAN in this work it is likely to have improvements, even though it cannot ensure the increasing of the precision of the joint position.
We recorded videos of 4 subjects performing 10 repetitions of each one of the 24 exercises introduced in *Chapter 7*.
Each video has been divided into these 10 repetitions and then resampled to 50 frames per repetition, resulting in a data set of 960 elements. Rather than using the entire video as a feature, only individual repetitions were considered. This approach helps eliminate unwanted variations such as differing rest times and subject-specific biases. It allows for more efficient classifier training and overcomes potential issues

like dealing with excessively long signals or requiring test signals to match the training data length of 10 repetitions.
Using the Kalman filter + DBSCAN method, which was the one that generally gave the best performance, we extracted the 3D joint position and computed the joint angles corresponding to shoulders, elbows, hips and knees to build the data set for the two classifiers exploited: a random forest and a multi-layer perceptron. The classification results indicate that while the Multi-layer Perceptron (MLP) achieved higher accuracy for certain participants, overall, the Random Forest classifier performed better for this data set. On average, the Random Forest classifier had a 2.4% higher accuracy than the MLP across four participants, and it also exhibited less variability (smaller standard deviation) in its performance.

Bibliography

[1] Mediapipe documentation. Available at https://developers.google.com/mediapipe/solutions/vision/pose$_l$*andmarker*.

[2] Opencv: Camera calibration documentation. Available at https://docs.opencv.org/4.x/dc/dbb/tutorial$_p$*y$_c$alibration.html*.

[3] Opencv: Detection of aruco markers. Available at https://docs.opencv.org/4.x/d5/dae/tutorial$_a$*ruco$_d$etection.html*.

[4] Optitrack cameras. Available at https://optitrack.com/cameras/primex-13/.

[5] Optitrack motive software. Available at https://optitrack.com/software/.

[6] J.T. Andersen, A.M. McCarthy, J.A. Wills, J.T. Fuller, G.K. Lenton, and T.L.A. Doyle. A markerless motion capture system can reliably determine peak trunk flexion while squatting with and without a weighted vest. 2023.

[7] Valentin Bazarevsky, Ivan Grishchenko, Karthik Raveendran, Tyler Zhu, Fan Zhang, and Matthias Grundmann. Blazepose: On-device real-time body pose tracking. *CoRR*, abs/2006.10204, 2020.

[8] Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. A time series forest for classification and feature extraction. 2013.

[9] Giulia Fracastoro. Slides of the course of "optimization for machine learning", politecnico di torino, 2022/2023.

[10] Q. Gan and C.J. Harris. Comparison of two measurement fusion methods for kalman-filter-based multisensor data fusion. *IEEE Transactions on Aerospace and Electronic Systems*, 37(1):273–279, 2001.

[11] Shubham Garg, Aman Saxena, and Richa Gupta. Yoga pose classification: a cnn and mediapipe inspired deep learning approach for real-world application. *Journal of Ambient Intelligence and Humanized Computing*, 06 2022.

[12] Shu Ting Goh, Ossama Abdelkhalik, and Seyed A. (Reza) Zekavat. A weighted measurement fusion kalman filter implementation for uav navigation. *Aerospace Science and Technology*, pages 315–323, 2012.

[13] Sumit Hazra, Acharya Aditya Pratap, Dattatreya Tripathy, and Anup Nandy. Novel data fusion strategy for human gait analysis using multiple kinect sensors. *Biomedical Signal Processing and Control*, 67, 2021.

[14] Vincent Hernandez, Davood Dadkhah, Vahid Babakeshizadeh, and Dana Kulić. Lower body kinematics estimation from wearable sensors for walking and running: A deep learning approach. *Gait Posture*, 83:185–193, 01 2021.

[15] Vincent Hernandez, Kulić Dana, and Gentiane Venture. Adversarial autoencoder for visualization and classification of human activity: Application to a low-cost commercial force plate. *Journal of Biomechanics*, 103:109684, 02 2020.

[16] Hustinawaty, Tavipia Rumambi, and Matrissya Hermita. Motion detection application to measure straight leg raise rom using mediapipe pose. In *2022 4th International Conference on Cybernetics and Intelligent System (ICORIS)*, pages 1–5. 2022.

[17] Sang hyub Lee, Deok-Won Lee, Kooksung Jun, Wonjun Lee, and Mun Sang Kim. Markerless 3d skeleton tracking algorithm by merging multiple inaccurate skeleton data from multiple rgb-d sensors. *Sensors*, 22, 2022.

[18] Mohamed Laaraiedh. Implementation of kalman filter with python language. 2009.

[19] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. Mediapipe: A framework for perceiving and processing reality. In *Third Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR) 2019*, 2019.

[20] Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[21] Gergely Nagymáté and Rita Mária Kiss. Application of optitrack motion capture

systems in human movement analysis: A systematic literature review. 1970.

[22] Farhad Nazari, Darius Nahavandi, Navid Mohajer, and Abbas Khosravi. Human activity recognition from knee angle using machine learning techniques. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 295–300. 2021.

[23] Byung-Seo Park, Woosuk Kim, Jin-Kyum Kim, Eui Seok Hwang, Dong-Wook Kim, and Young-Ho Seo. 3d static point cloud registration by estimating temporal human pose at multiview. *Sensors*, 22, 2022.

[24] Chenjian Ran and Zili Deng. Two correlated measurement fusion kalman filtering algorithms based on orthogonal transformation and their functional equivalence. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 2351–2356. 2009.

[25] J.A. Roecker and C.D. McGillem. Comparison of two-sensor tracking methods based on state vector fusion and measurement fusion. *IEEE Transactions on Aerospace and Electronic Systems*, 24(4):447–449, 1988.

[26] Moon S, Park Y, Ko DW, and Suh IH. Multiple kinect sensor fusion for human skeleton tracking using kalman filtering. *International Journal of Advanced Robotic Systems*, 2016.

[27] Kaustubh Sadekar and Satya Mallick. Available at https://learnopencv.com/camera-calibration-using-opencv/.

[28] Antonio Carlos Sementille, Luís Escaramuzi Lourenço, José Remo Ferreira Brega, and Ildeberto Rodello. A motion capture system using passive markers. In *Proceedings of the 2004 ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry*, page 440–447. Association for Computing Machinery, 2004.

[29] Shubham Sharma, Shubhankar Verma, Mohit Kumar, and Lavanya Sharma. Use of motion capture in 3d animation: Motion capture systems, challenges, and recent trends. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 289–294. 2019.

[30] Md. Zia Uddin, Nguyen Duc Thang, and Tae-Seong Kim. Human activity recognition via 3-d joint angle features and hidden markov models. In *2010 IEEE International Conference on Image Processing*, pages 713–716. 2010.

[31] Greg Welch and Gary Bishop. An introduction to the kalman filter. *The Python Papers*, 2006.

[32] Ge Wu, Sorin Siegler, Paul Allard, Chris Kirtley, Alberto Leardini, Dieter Rosenbaum, Mike Whittle, Darryl D D'Lima, Luca Cristofolini, Hartmut Witte, Oskar Schmid, and Ian Stokes. Isb recommendation on definitions of joint coordinate system of various joints for the reporting of human joint motion—part i: ankle, hip, and spine. *Journal of Biomechanics*, 35:543–548, 2002.

[33] Ge Wu, Frans C.T. van der Helm, H.E.J. Veeger, Mohsen Makhsous, Peter Van Roy, Carolyn Anglin, Jochem Nagels, Andrew R. Karduna, Kevin McQuade, Xuguang Wang, Frederick W. Werner, and Bryan Buchholz. Isb recommendation on definitions of joint coordinate systems of various joints for the reporting of human joint motion—part ii: shoulder, elbow, wrist and hand. *Journal of Biomechanics*, 38:981– 992, 2005.

[34] Santosh Kumar Yadav, Kamlesh Tiwari, Hari Mohan Pandey, and Shaik Ali Akbar. Skeleton-based human activity recognition using convlstm and guided feature learning. *Soft computing*, 26:877 – 890, 2022.

[35] Shuo Zhang, Wanmi Chen, Chen Chen, and Yang Liu. Human deep squat detection method based on mediapipe combined with yolov5 network. In *2022 41st Chinese Control Conference (CCC)*, pages 6404–6409. 2022.

[36] Zhengyou Zhang. A flexible new technique for camera calibration. Technical report, Redmond, WA, USA, 1998.

[37] Ömer Faruk İnce, Ibrahim Furkan Ince, Mustafa Eren Yıldırım, Jang Sik Park, Jong Kwan Song, and Byung Woo Yoon. Human activity recognition with analysis of angles between skeletal joints using a rgb-depth sensor. *ETRI journal*, 42:981–992, 2020.