



Palestine Polytechnic University  
Deanship of Graduate Studies and Scientific Research  
Master of Intelligent Systems

# Using Deep Pose Estimation of Football Player Body for Virtual Reality

Submitted by:

Islam Amar

Supervisors:

Dr. Hashem Tamimi

ProF. Jihad El-Sana

Thesis submitted in partial fulfillment of requirements of the degree  
Master of Science in Intelligent Systems

Feb, 2023

The undersigned hereby certify that they have read, examined and recommended to the Deanship of Graduate Studies and Scientific Research at Palestine Polytechnic University the approval of a thesis entitled: **Using Deep Pose Estimation of Football Player Body for Virtual Reality.**, submitted by **Islam Amar** in partial fulfillment of the requirements for the degree of Master in Intelligent Systems.

**Graduate Advisory Committee:**

Dr. Hashem Tamimi (Supervisor), Palestine Polytechnic University.

Signature:\_\_\_\_\_ Date:\_\_\_\_\_

Prof. Jihad El-Sana (Supervisor), Triangle Research and Development Center.

Signature:\_\_\_\_\_ Date:\_\_\_\_\_

Dr. Alaa Halawani (Internal committee member), Palestine Polytechnic University.

Signature:\_\_\_\_\_ Date:\_\_\_\_\_

Dr. Raid Saabni (External committee member), Triangle Research and Development Center.

Signature:\_\_\_\_\_ Date:\_\_\_\_\_

**Thesis Approved**

|                                                                                                               |
|---------------------------------------------------------------------------------------------------------------|
| Dr. Nafeth Nasereddin<br>Dean of Graduate Studies and Scientific Research<br>Palestine Polytechnic University |
|---------------------------------------------------------------------------------------------------------------|

Signature:\_\_\_\_\_ Date:\_\_\_\_\_



# DECLARATION

I declare that the Master Thesis entitled "**Using Deep Pose Estimation of Football Player Body for Virtual Reality**" is my original work, and hereby certify that unless stated, all work contained within this thesis is my own independent research and has not been submitted for the award of any other degree at any institution, except where due acknowledgement is made in the text.

**Islam Amar**

Signature:\_\_\_\_\_

Date:\_\_\_\_\_

# STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for the master degree in Intelligent System at Palestine Polytechnic University, I agree that the library shall make it available to borrowers under rules of the library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from, reproduction, or publication of this thesis may be granted by my main supervisor, or in his absence, by the Dean of Graduate Studies and Scientific Research when, in the opinion of either, the proposed use of the material is for scholarly purposes.

Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

**Islam Amar**

Signature:\_\_\_\_\_

Date:\_\_\_\_\_

# DEDICATION

*To my parents Mrs&Mr Najat and Ayman Amar*

*To My sisters and Bothers*

# ACKNOWLEDGEMENT

As I write the last words of this thesis, I greatly appreciate the thesis's supervisors Prof. Jihad El-Sana and Dr. Hashem Tamimi for their support and time they spent with me in order for this thesis to succeed.

I would like to thank the thesis's examiners Dr. Alaa Halawani and Dr. Raid Saabni for their valuable suggestions and corrections to this work, which greatly helped me to improve in various aspects.

I would like to thank my family for continued support, encouragement and patience from the first step till the end.

# Abstract

Animating 3D character models has attracted the interest of many researchers over the past two decades, and many related practical algorithms have been developed. These algorithms apply various techniques ranging from physics-based animation and inverse kinematics to 3D skeletal animation and rigging. In this study, we demonstrate a framework to reconstruct the 3D models of the players in sport game views. Initially, pose features are extracted from each player’s body using a set of deep neural networks. These networks are pre-trained on 3D player data. Next, these poses are applied to the 3D model of the player. Eventually, The poses and positions of the players in the virtual field will match the actual ones. The view can be displayed via a 3D viewer or virtual reality devices. Our proposed framework consists of different deep neural networks, including convolutional and, recurrent neural networks, for estimating human poses that form the main body. In this study, we were able to reconstruct the body models of real players and transform them into avatars. In addition, we outperformed the rendering process of an existing research.

# Table of Contents

|                   |                                                |    |
|-------------------|------------------------------------------------|----|
| Permission        | iii                                            |    |
| Acknowledgments   | v                                              |    |
| Abstract          | vi                                             |    |
| Table of Contents | viii                                           |    |
| List of Figures   | ix                                             |    |
| List of Tables    | xi                                             |    |
| Chapter 1         | Introduction                                   | 1  |
| 1.1               | Research objectives . . . . .                  | 3  |
| 1.2               | Thesis structure . . . . .                     | 4  |
| Chapter 2         | Background                                     | 6  |
| 2.1               | Machine learning . . . . .                     | 6  |
| 2.2               | What are Neural Networks? . . . . .            | 7  |
| 2.2.1             | Cost Function . . . . .                        | 8  |
| 2.3               | Deep learning . . . . .                        | 9  |
| 2.3.1             | How deep learning works . . . . .              | 10 |
| 2.3.2             | Deep Neural Networks . . . . .                 | 11 |
|                   | Convolutional Neural Networks (CNNs) . . . . . | 11 |
|                   | Recurrent Neural Networks (RNNs) . . . . .     | 13 |

|                     |                                                                   |           |
|---------------------|-------------------------------------------------------------------|-----------|
|                     | Long Short Term Memory Networks (LSTMs)                           | 14        |
|                     | Generative Adversarial Networks (GANs) . . .                      | 15        |
| 2.3.3               | Deep learning applications . . . . .                              | 17        |
| 2.3.4               | Deep Learning Platforms . . . . .                                 | 17        |
| 2.3.5               | Human pose estimation . . . . .                                   | 19        |
| 2.3.6               | Classical approaches to 2D Human Pose Esti-<br>mation . . . . .   | 20        |
| 2.3.7               | Human Pose Estimation using Deep Neural<br>Networks . . . . .     | 22        |
|                     | Deep learning models for multi-human pose<br>estimation . . . . . | 23        |
| 2.3.8               | Transfomration . . . . .                                          | 34        |
| <b>Chapter 3</b>    | <b>Literature review</b>                                          | <b>36</b> |
| 3.1                 | Object detection Algorithms related works . . . . .               | 36        |
| 3.2                 | Human Pose Estimation And 3D Rendering . . . . .                  | 39        |
| <b>Chapter 4</b>    | <b>Our Approach</b>                                               | <b>43</b> |
| <b>Chapter 5</b>    | <b>Experiments and Results</b>                                    | <b>57</b> |
| 5.1                 | Performance . . . . .                                             | 65        |
| <b>Chapter 6</b>    | <b>Conclusions and Future Work</b>                                | <b>69</b> |
| <b>Appendix A</b>   | <b>Cameras Equations</b>                                          | <b>72</b> |
| A.1                 | Camera-1 equations . . . . .                                      | 72        |
| A.2                 | camera-2 equations . . . . .                                      | 73        |
| A.3                 | camera-4 equations . . . . .                                      | 74        |
| <b>Appendix B</b>   | <b>Code</b>                                                       | <b>76</b> |
| B.1                 | Virtual reality code . . . . .                                    | 76        |
| <b>Bibliography</b> |                                                                   | <b>77</b> |

# List of Figures

|      |                                                                                                           |    |
|------|-----------------------------------------------------------------------------------------------------------|----|
| 2.1  | Simple neural networks . . . . .                                                                          | 8  |
| 2.2  | Cost Function . . . . .                                                                                   | 9  |
| 2.3  | Convolutional Neural Networks . . . . .                                                                   | 12 |
| 2.4  | Recurrent Neural Networks (RNNs) . . . . .                                                                | 13 |
| 2.5  | Google’s autocompleting feature . . . . .                                                                 | 14 |
| 2.6  | Long Short Term Memory Networks . . . . .                                                                 | 15 |
| 2.7  | Generative Adversarial Networks architecture . . . . .                                                    | 16 |
| 2.8  | Human Body Models [1]. . . . .                                                                            | 20 |
| 2.9  | OpenPose Architecture . . . . .                                                                           | 24 |
| 2.10 | OpenPose pipeline . . . . .                                                                               | 24 |
| 2.11 | bounding box localization error [2]. . . . .                                                              | 25 |
| 2.12 | multiple poses detection of single person using AlphaPose method [2]                                      | 26 |
| 2.13 | joint subset partition and labeling for multi person pose estimation<br>using DeepCut method [3]. . . . . | 27 |
| 2.14 | result of using Mask R-CNN on coco test image [4] . . . . .                                               | 28 |
| 2.15 | Mask R-CNN Architecture . . . . .                                                                         | 29 |
| 2.16 | SMPL shape blend shape [5]. . . . .                                                                       | 31 |
| 2.17 | SMPL pose blend shapes [5]. . . . .                                                                       | 32 |
| 2.18 | SMPL skinning [5]. . . . .                                                                                | 32 |
| 2.19 | SMPL template mesh where white points indicate 24 joints and<br>color indicate blend weights [5]. . . . . | 33 |



|     |                                                                                                                                                       |    |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.1 | 3D object detection for pose estimation using object as points detection method [6]. . . . .                                                          | 38 |
| 3.2 | the General architecture of MobileNetv2 . . . . .                                                                                                     | 39 |
| 4.1 | Mapping of real game to virtual environment . . . . .                                                                                                 | 44 |
| 4.2 | The General architecture of our framework . . . . .                                                                                                   | 49 |
| 4.3 | Mapping between SMPL joint id's and joint names . . . . .                                                                                             | 51 |
| 4.4 | Dura staduim . . . . .                                                                                                                                | 54 |
| 4.5 | Camera-1 Projective Transformation . . . . .                                                                                                          | 55 |
| 4.6 | Camera-2 Projective Transformation . . . . .                                                                                                          | 56 |
| 4.7 | Camera-3 Projective Transformation . . . . .                                                                                                          | 56 |
| 5.1 | Camera-2 and Camera-3 reconstruction for Frame-1 . . . . .                                                                                            | 60 |
| 5.2 | Camera-2 and Camera-3 reconstruction for Frame-2 . . . . .                                                                                            | 61 |
| 5.3 | 3D model generated by our framework for person standing behind the table as a test for partial occlusion . . . . .                                    | 63 |
| 5.4 | human pose skeleton for MobileNetV2 for a person stand behind the table as a test for partial occlusion . . . . .                                     | 63 |
| 5.5 | 3D models generated by our framework for each player in a soccer match . . . . .                                                                      | 63 |
| 5.6 | 3D model generated by our framework for person standing behind the table as a test for partial occlusion . . . . .                                    | 64 |
| 5.7 | 3D models generated by our framework for two pedestrians standing behind the sofa with different poses as a test for partial occlusion . . . . .      | 64 |
| 5.8 | 3D models generated by our framework for two pedestrians standing behind the sofa with different hand poses as a test for partial occlusion . . . . . | 64 |
| 5.9 | Performance of our framework according to different numbers of detected players . . . . .                                                             | 68 |

# List of Tables

|     |                                                                                                                |    |
|-----|----------------------------------------------------------------------------------------------------------------|----|
| 5.1 | the number of detected and undetected players in their respective re-<br>sponding frames in Camera-2 . . . . . | 58 |
| 5.2 | the number of detected and undetected players in their respective re-<br>sponding frames in Camera-3 . . . . . | 58 |
| 5.3 | the number of detected and undetected players in their respective re-<br>sponding frames in Camera-1 . . . . . | 59 |
| 5.4 | detection and rendering elapsed time for both the traditional and our<br>proposed approach. . . . .            | 65 |
| 5.5 | Time analysis of our framework for different 565 frames videos . . . . .                                       | 66 |

# Chapter 1

## Introduction

Watching sports competitions is among the main entertainment avenues attracting millions of people worldwide. Currently, people follow these games on-site or remotely over 2D displays. In both cases, the observer often has a limited view of the match, which is dictated by the seat in the stadium or the camera's current view. The user's ability to select or influence the view parameters, such as direction, field of view, and zoom, is limited. Several approaches equip the match field with many cameras to generate novel views, synchronize them and reconstruct the game area with the players [7, 8, 9]. These approaches create views that were not captured by any of the cameras. However, to reach accurate construction, there is a need for many cameras at high resolution that are finely synchronized. Nevertheless, the reconstruction is not always complete. Free Viewpoint Video (FVV)[10] is a sophisticated visual media format that provides variable perspective movement in

3D space and time from multi-view filmed video. The main advantage of FVV is interactivity, which allows users to control the ideal viewing angles and locations rather than broadcasters or content developers.

For making a typical FVV system, the system should contain many modules, such as the acquisition module, where the primary role is to acquire the 3D scene. It comprises several capturing devices in general (e.g., sensors and camera arrays) and a processing module that converts collected data signals (such as image data) into a data format suited for 3D scene representation. In addition, it should have a transmission module that provides a concise 3D data representation format for streaming and/or storing. The rendering module synthesizes the decoded 3D data for display and navigation at the specified viewing and time scales. The Display module presents the rendered video to increase the user's immersion in the situation. It also includes user interface (UI) capabilities for navigating viewpoints. Additionally, it has user interface (UI) features for switching between views.

Fixed cameras and cable-suspended cameras are used in traditional stadium systems to give video feeds from limited angles. Meanwhile, this system lets viewers observe the action on the pitch from any position or aspect of the stadium for the first time. one may watch the same scene from numerous angles, for as from the perspective of an athlete on the field or any number of other perspectives. Furthermore, spectators have complete control over the perspective and the game duration. For example, when watching the scene in slow motion, the viewpoint can be adjusted. This breakthrough technology is changing the way sports

are perceived, and this match was the moment it became a reality. The method through which video is created may be seen as the future of video capture. High-resolution cameras mounted throughout the stadium record visual data, which is subsequently transformed into 3D data and saved on computers. When a user selects or modifies the virtual camera's location, the video they view is created from 3D data to present footage from the specified camera angle. This video data can then be shown. Rugby and soccer matches [11], among other sporting events, are currently being tested with the FVV system. Each camera needs to start recording at the same precise moment to get the correct 3D data. The data cannot be appropriately created if even one camera's time is wrong. Since they were aware of this problem throughout the design process, developers created algorithms to completely synchronize and manage the start of shooting for various cameras. The requirement to instantly analyze vast amounts of data to provide Free Viewpoint Video data presents an additional barrier. High-definition photos are being produced at quicker rates using techniques like parallel distributed processing.

## 1.1 Research objectives

This thesis presents a framework to view sports via a virtual reality interface and give spectators the ability to view the field and the players from any point of view at any zoom anytime. Our framework should extract pose features using deep learning models from every player in the video stream and apply these features to

predefined 3D models of the respective players within a synthesized sports field, forcing them to have the same pose and position in the virtual field according to the actual one.

## 1.2 Thesis structure

Chapter 1, discusses the scope of the research, the goals and objectives of the research, and the methodology. It also explains the thesis structure and contribution to knowledge. Chapter 2 presents theories, concepts, terms, and ideas that focus on machine learning, neural networks, deep learning platforms, well-known deep learning neural networks, and classical and deep learning approaches in human pose estimation. Chapter 3 presents the results of the comprehensive review of the most significant studies relevant to the current research. The review focuses on the techniques and algorithms applied to enhance the performance of object detection systems and the methods used to predict human pose. In addition, this chapter describes the published results and discussions on famous 3D human Models. Chapter 4 provides a basic and detailed overview of our framework, pose feature extraction, and avatars for player models. Additionally, this chapter provides a summary of the primary tools that were utilized to develop this project, as well as the environment specification that we worked on during this project. Chapter 5 shows the experiments and results of this study. Chapter 6 highlights the major findings and presents the main conclusion from the study conducted

---

and recommendations for future investigations.

# Chapter 2

## Background

### 2.1 Machine learning

One of the main branches of artificial intelligence(AI) is machine learning(ML). Without explicit programming, ML programs learn from experience (or, to be precise, data) as people do [12]. These systems learn, grow, alter, and expand on their own when presented with additional data. In other words, machine learning entails computers discovering valuable knowledge on their own. The capacity to autonomously and repeatedly adapt to new input is the essence of machine learning. Applications employ "pattern recognition" to generate trustworthy and informed outcomes by learning from prior calculations, and transactions [13]. Due to its complexity, machine learning has been split into two main categories: supervised learning and unsupervised learning. Each has a distinct goal and course



of action that produces outcomes and uses different data types [14].

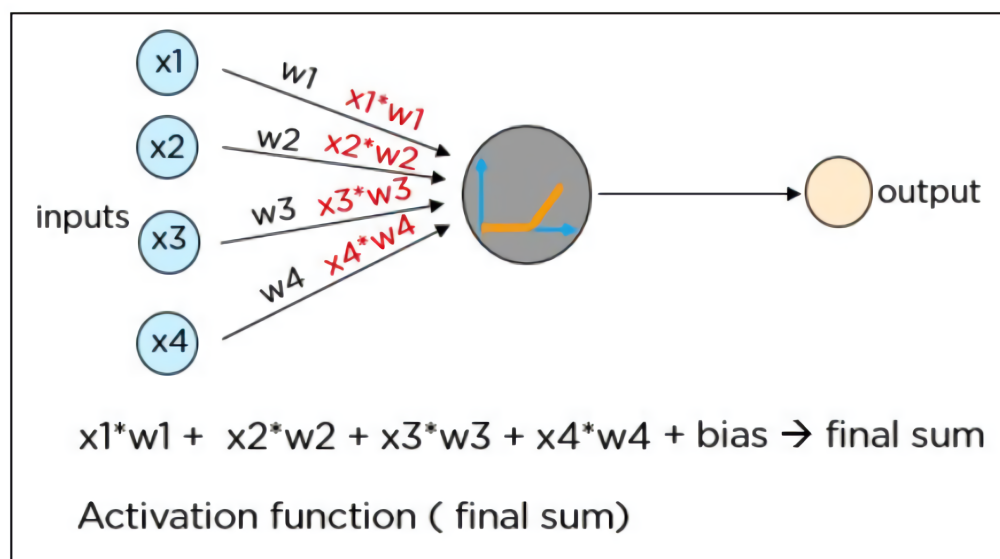
## 2.2 What are Neural Networks?

A neural network is a ML system composed of an input layer, many hidden layers, and an output layer that is modeled based on the human brain. Data is delivered into the neurons as input. Using proper weights and biases, the information is passed to the next layer. The final value anticipated by the artificial neuron is the output [15]. In a neural network, each neuron performs the following tasks as shown clearly in Figure 2.1:

- The weight of the channel it is sent across is calculated as the product of each input.
- The weighted sum, which is the sum of the weighted products, is calculated.
- The weighted sum is expanded by the neuron's bias value.
- The activation function is a specific function that is applied after the final sum.

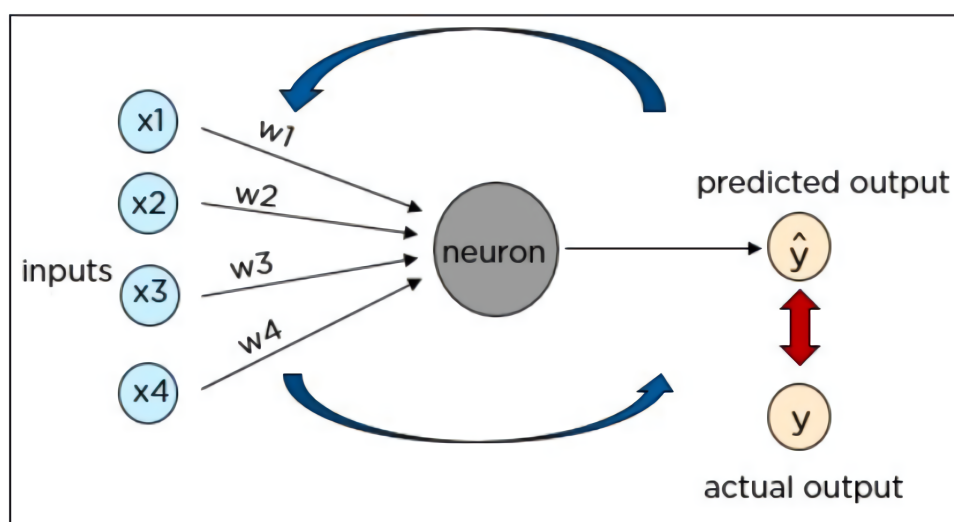
---

<sup>1</sup>Courtesy:<https://www.simplilearn.com/tutorials/deep-learning-tutorial/introduction-to-deep-learning>.

Figure 2.1: Simple neural networks<sup>1</sup>.

### 2.2.1 Cost Function

The cost function is an important component of a neural network. The cost value is the difference between the anticipated output of a neural network and the actual output of a batch of labeled training data. The least-cost value is produced by iteratively adjusting the weights and biases throughout the training phase, as illustrated in Figure 2.2.

Figure 2.2: Cost Function<sup>2</sup>.

## 2.3 Deep learning

Deep learning can be thought of as a subset of machine learning. In contrast to machine learning, deep learning works with artificial neural networks that aim to mimic human thinking and learning. Until recently, the complexity of neural networks was limited due to limited computing power. However, advances in big data analytics have made computers able to monitor, understand, and respond to complex situations faster than humans. Image classification, language translation, and speech recognition all benefit from deep learning. You can solve pattern recognition problems without the need for human intervention. Deep learning utilizes a multi-layered artificial neural network. Deep Neural Networks (DNNs) are

---

<sup>2</sup>Courtesy:<https://www.simplilearn.com/tutorials/deep-learning-tutorial/introduction-to-deep-learning>.

multi-layer networks that can perform complex operations such as representations and abstractions to understand images, sounds, and texts. Deep learning, the fastest-growing area of machine learning, is a truly disruptive digital technology that is being used by more and more companies to develop new business models [16, 17, 18]

### 2.3.1 How deep learning works

Just as the human brain is made up of neurons, neural networks consist of layers of nodes. Individual layer nodes are linked to nodes in adjacent layers. The number of layers in the network indicates that it is deeper. In the human brain, one neuron receives hundreds of impulses from other neurons. The signal travels between the nodes of the artificial neural network and weights the nodes. Nodes with higher weights have a greater impact on the nodes below them. The weighted inputs are assembled in the final layer to produce the output [19]. Deep learning systems process large amounts of data and perform multiple difficult mathematical calculations, requiring powerful hardware. However, even with such advanced hardware, training neural networks can take weeks. Deep learning algorithms require large amounts of data to provide correct results. As a result, the information is provided as a large data set. Artificial neural networks can use very complex mathematical calculations while processing data to classify data using answers to a series of binary yes or false questions [20]. For example, facial recognition computers learn to identify and recognize the edges and lines of a face, then the more important as-

pects of the face, and finally the entire facial expression. The algorithm learns and improves over time, increasing the chances of a correct answer. The face detection algorithm correctly identifies the face over time in this situation [21].

### 2.3.2 Deep Neural Networks

Deep neural networks are regarded as one of the most important research topics due to their superior performance in a wide range of benchmark problems and applications. DNN can be implemented in general by stacking multiple layers. This section demonstrates the most well-known type of DNN.

#### Convolutional Neural Networks (CNNs)

CNNs, often called ConvNets, have several layers and are mainly used for object detection and image processing. When it was still known as LeNet in 1988, Yann LeCun created the first CNN[22]. It was used to identify characters such as ZIP codes and numbers. The identification of satellite photos, processing of medical imaging, forecasting of time series, and anomaly detection all make use of CNNs. CNNs analyze and extract features from data using a number of layers [23]:

**Convolution Layer** : A convolution layer on CNN uses a number of filters to carry out the convolution process.

**Rectified Linear Unit (ReLU)** : ReLU is a layer that CNNs have to manipulate elements. ReLU does an operation element-by-element by setting all the negative values to 0. ReLU is generally used for adding non-linearity to the network, where the result from ReLU is a corrected feature map that just contains positive values as they are and negative values as 0.

**Pooling Layer** : The pooling layer receives the rectified feature map after that. Pooling is a downsampling procedure that shrinks the feature map's dimensions. The resultant two-dimensional arrays from the pooled feature map are then flattened by the pooling layer into a solitary lengthy continuous linear vector.

**Fully Connected Layer** : When the flattened matrix from the pooling layer is supplied as an input, a fully linked layer that classifies and IDs the images forms.

Figure 2.3 illustrates a CNN-processed image.

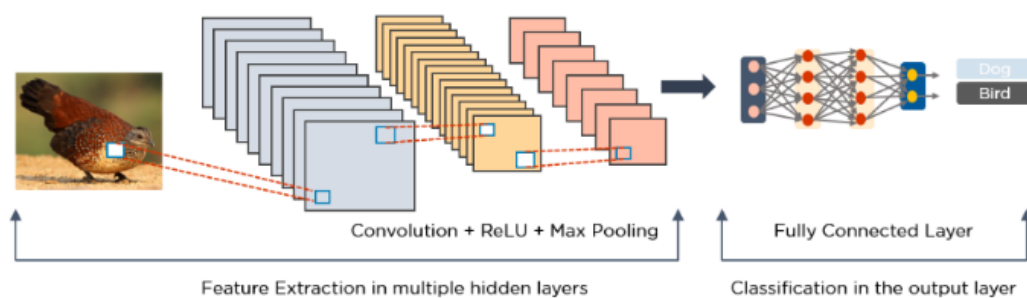


Figure 2.3: Convolutional Neural Networks<sup>3</sup>

<sup>3</sup>Courtesy:<https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network>

## Recurrent Neural Networks (RNNs)

The outputs from the Long Short Term Memory Networks (LSTM) can be sent as inputs to the current phase thanks to RNNs' connections that form directed cycles. Due to its internal memory, the LSTM's output can remember prior inputs and is used as an input in the current phase. Natural language processing, time series analysis, handwriting recognition, and machine translation are all common applications for RNNs [24].

An unfolded RNN appears as follows in Figure 2.4:

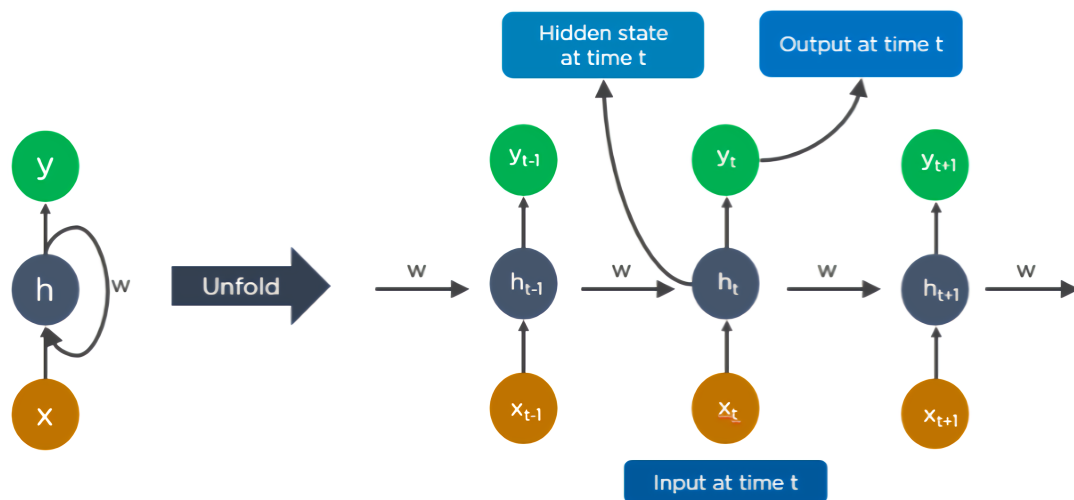


Figure 2.4: Recurrent Neural Networks (RNNs)<sup>4</sup>

How Do RNNs work?

- The output at time  $t-1$  feeds into the input at time  $t$
- Similarly, the output at time  $t$  feeds into the input at time  $t+1$ .

<sup>4</sup>courtesy:<https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn>

- RNNs can process inputs of any length.
- The computation incorporates past data, but the model size is independent of the input size.

Figure 2.5 illustrates of how Google’s autocomplete function operates:

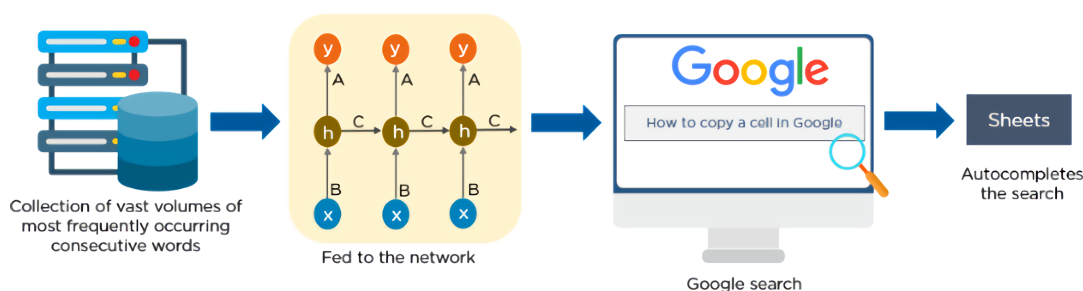


Figure 2.5: Google’s autocompleting feature<sup>5</sup>

## Long Short Term Memory Networks (LSTMs)

Recurrent neural networks (RNNs) with LSTMs have the capacity to learn and remember long-term dependencies. The default behavior is to recall past information for extended periods of time [25].

Over time, LSTMs preserve information. Due to their ability to recall prior inputs, they are helpful in time-series prediction. In LSTMs, four interacting layers connect in a chain-like structure to communicate in a special way. LSTMs are frequently employed for voice recognition, music creation, and drug research, in addition to time-series predictions.

<sup>5</sup>courtesy:<https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm>



### How Do LSTMs Work?

- They start by forgetting pointless details of the prior condition.
- Then, they update the cell-state values in a selected manner.
- Lastly, the output of specific cell state components

Figure 2.6 shows how LSTMs work:

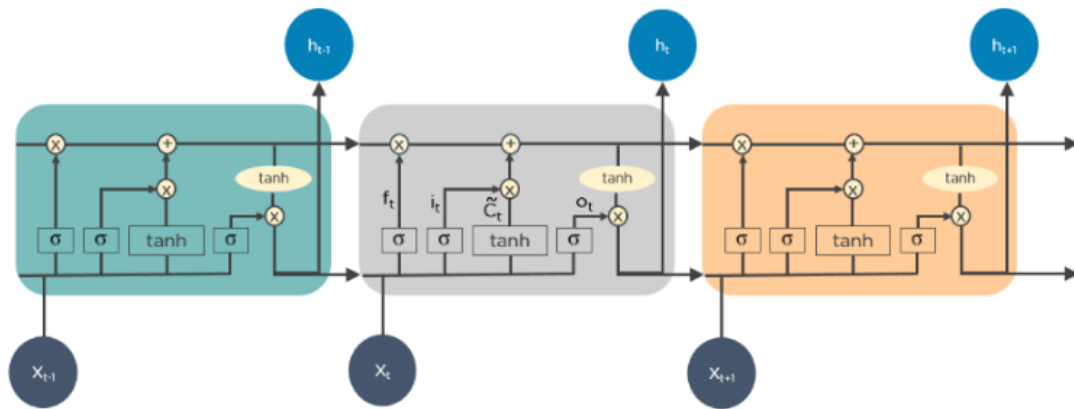


Figure 2.6: Long Short Term Memory Networks<sup>6</sup>

### Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) produce new data instances that mimic the training data. GAN consists of two parts: a generator that learns to produce new data and a discriminator that absorbs the false data into its learning process [26].

<sup>6</sup>Courtesy: <https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm>

Over time, GANs have become more often used. They can be employed for dark-matter studies to replicate gravitational lensing and enhance astronomy photos. Using picture training, GANs are used by video game producers to recreate low-resolution, 2D textures in older video games in 4K or greater resolutions. GANs are used to render 3D objects, produce photos of human faces, and produce realistic visuals and cartoon characters.

How GANs works?

- The discriminator gains the ability to discriminate between the real sample data and the generator's fake data.
- The generator creates fake data during early training, and the discriminator quickly picks up on the fact that it is untrue.
- In order to update the model, the GAN delivers the results to the generator and discriminator.

Figure 2.7 showing how GANs work:

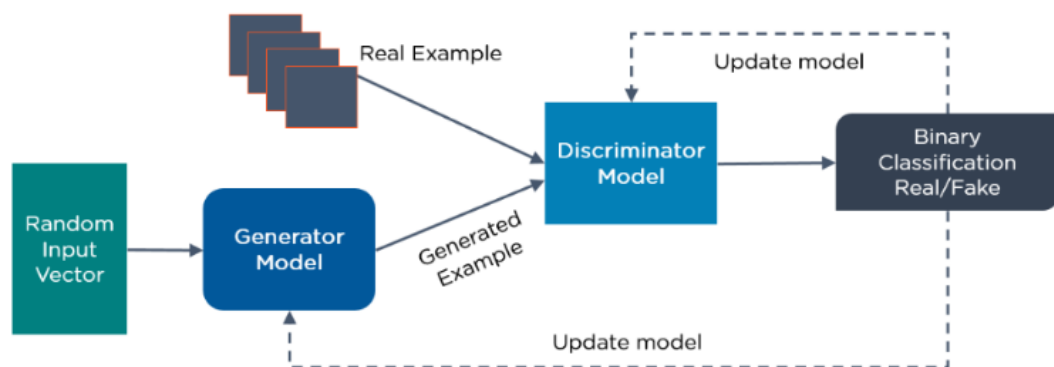


Figure 2.7: Generative Adversarial Networks architecture<sup>7</sup>

### 2.3.3 Deep learning applications

Deep learning is commonly used to forecast rain, earthquakes, and tsunamis. It aids in the implementation of appropriate safeguards. Machines can understand speech and produce the needed output thanks to deep learning. It allows machines to distinguish between people and things in photographs. Advertisers may also use deep learning models to execute real-time bidding, and targeted display advertising [27].

### 2.3.4 Deep Learning Platforms

**Torch** PyTorch is an efficient deep learning tensor library that was first released in 2016 based on Python and Torch, mainly used for applications that require GPU and CPU. PyTorch uses dynamic computational graphs and is completely Pythonic, so it takes precedence over other deep learning frameworks such as TensorFlow and Keras. This allows scientists and engineers to run and evaluate code sections in real-time. PyTorch simplifies the construction of highly complex neural networks. It is easy to understand and use on both Windows and Linux. PyTorch introduces a new hybrid front end that offers flexibility and ease of use in Eager mode while moving to graph mode for speed, optimization, and functionality of the C++ runtime [28].

---

<sup>7</sup>courtesy:<https://www.simplilearn.com/tutorials/deep-learning-tutorial/generative-adversarial-networks-gans>

**Keras** We use a neural network that uses multiple operators placed on the node to divide the problem into smaller pieces and solve each individually. However, implementing a neural network can be very difficult. Keras, a deep learning framework, handles this issue. Keras is a high-level Neural Network library developed by Google to implement neural networks. Written in Python, it is used to facilitate the implementation of neural networks. It also supports the calculation of multiple back-end neural networks. Keras is built into TensorFlow and provides built-in modules for all neural network calculations so you can use it to perform deep learning quickly. Initially, it had more than 4,800 contributors at launch but has since grown to 250,000 developers. It's growing every year, so it's doubling. Large companies such as Microsoft, Google, NVIDIA, and Amazon have actively contributed to the development of Keras [29].

**TensorFlow** First released in late 2015. It is considered an open source end-to-end platform for building machine learning applications. The Tensorflow architecture consists of three parts: data preprocessing, model building, model training and estimation. The word TensorFlow consists of two words. Tensor and Flow, where Tensor is a multidimensional array, Flow is used to define the flow of data in operation. TensorFlow incorporates several API to develop a large-scale deep learning architecture such as CNN or RNN . it is based on graph computation and allows the developer to build a neural network with Tensorboard. This tool aids in the debugging of our software. It is powered by a CPU (Central Processing Unit) and a GPU (Graphical Processing Unit)[30].

### 2.3.5 Human pose estimation

Human Pose Estimation (HPE) is a method of recognizing and categorizing human body joints. It is a method of capturing a collection of coordinates for each joint (arm, head, torso, etc.) known as a key point that may define a person's position. A pair is a relationship between these two places. Because the link between the points must be substantial, not all points can form a pair. HPE aims to create a skeleton-like model of a human body and then process it for task-specific applications [31].

There are three methods for modeling the human body [1], as illustrated in Figure 2.8, these are :

1. Skeleton-based model
2. Contour-based model
3. Volume-based model

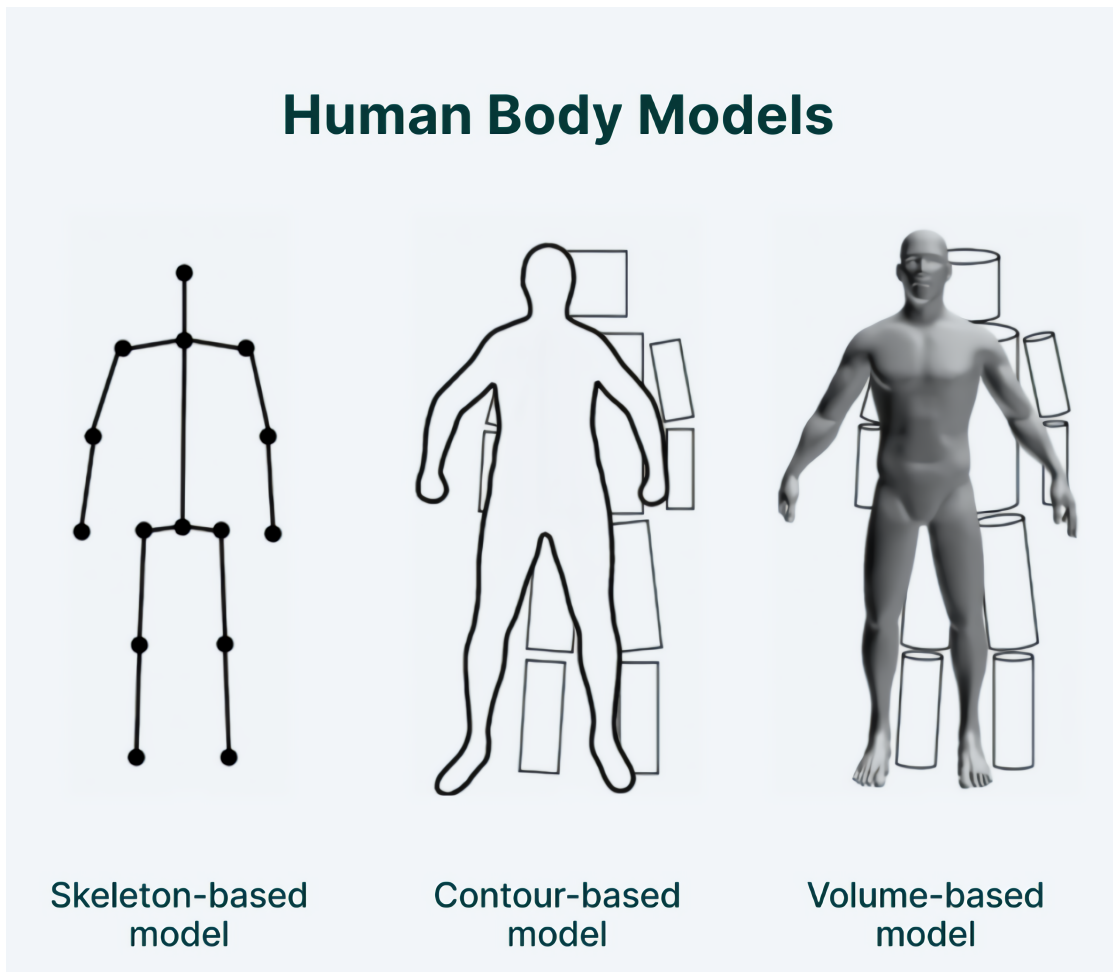


Figure 2.8: Human Body Models [1].

There are two approaches for Human pose estimation, classical and deep learning. Next, we overview each category.

### 2.3.6 Classical approaches to 2D Human Pose Estimation

Classical approaches are strategies and procedures that use traditional machine learning algorithms. For example, earlier attempts to estimate human position incorporated random forests inside a pictorial structural framework. This was

utilized to forecast human body joints. One of the conventional ways of estimating human posture is the pictorial structural framework (PSF). PSF consisted of two parts:

- Discriminator: It simulates the possibility of a specific part being present at a given place. In other words, it recognizes bodily components.
- Prior: It is known as modeling the probability distribution over posture using the discriminator output; the modeled pose should be realistic.

In essence, the PSF goal is to represent the human body in a given input picture as a collection of coordinates for each body part. Nonlinear joint regressors, ideally a two-layered random forest regressor, are used in PSF[32]. These models perform well when the input picture contains distinct and visible limbs; nevertheless, they fail to capture and model limbs that are concealed or not visible from a specific angle. To address these challenges, feature construction approaches such as histogram oriented gaussian (HOG), contours, histograms, and so on were applied [33]. Despite applying these approaches, the classical model lacked accuracy, correlation, and generalization capabilities. Therefore adopting a new strategy was only a question of time. The capacity of deep learning-based techniques to generalize any function is well known (if a sufficient number of nodes are present in the given hidden layer). Deep convolutional neural networks (CNN) outperform all other methods in computer vision tasks, and this is also true in HPE. CNN can extract patterns and representations from a given input picture with greater

precision and accuracy than any other method, making it ideal for classification, detection, and segmentation tasks. In contrast to the traditional technique, in which the features were handmade, CNN can learn complicated features when given enough training-validation-testing data.

### 2.3.7 Human Pose Estimation using Deep Neural Networks

As HPE's research and development began to take off, new issues arose, such as dealing with the multi-person pose estimation problem. DNNs are quite good at predicting single-human poses, but they struggle with multi-human poses due to the following:

- An image can feature a large number of individuals in various situations.
- As the number of individuals rises, so does their interaction, leading to increased computing difficulties.
- In real-time inference, as computing complexity rise, inference time rises as well.

The researchers proposed two techniques to deal with this issues [\[33\]](#):

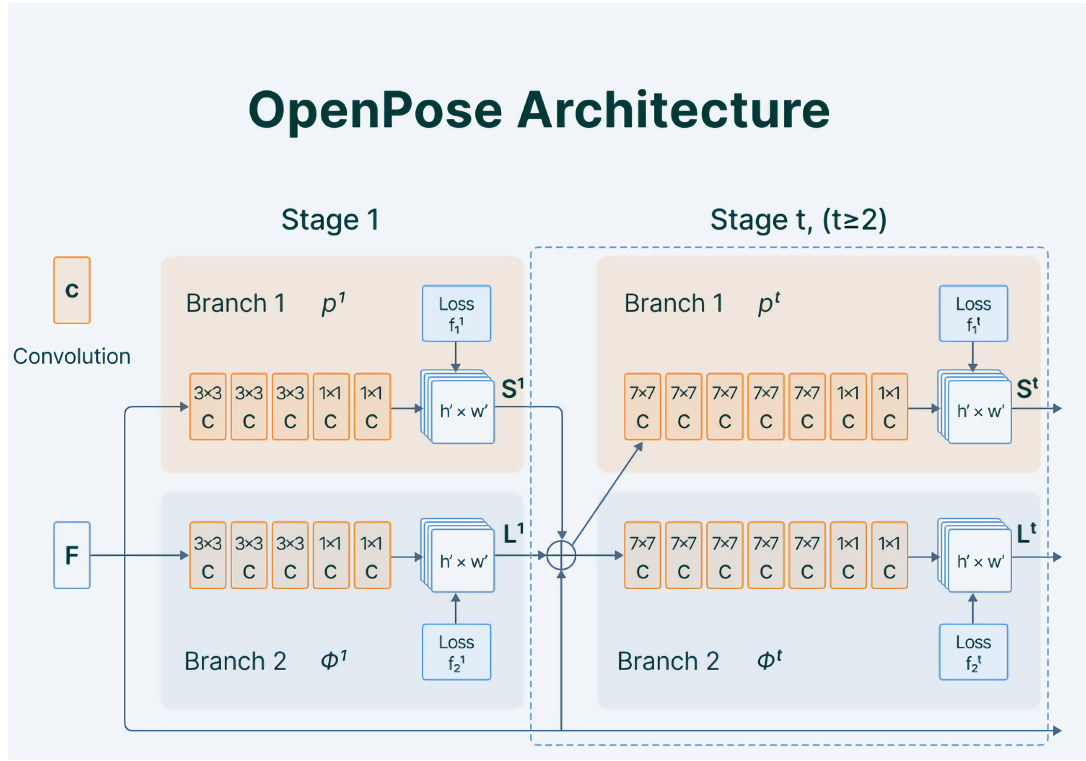
- Top-down: Identify the persons in the image or video, estimate their components, and then calculate their stance.



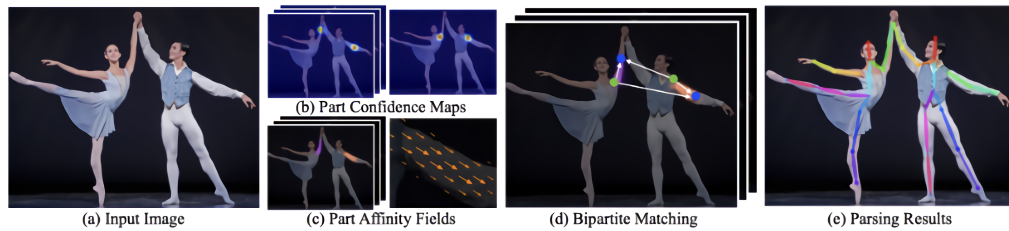
- Bottom-up: Calculate the posture after estimating the human body components in the image.

### Deep learning models for multi-human pose estimation

**OpenPose** Cao *et al.*[34] presented OpenPose. The network recognizes body components or key points in the picture first, then maps relevant key points to form pairings in a bottom-up manner. The main architecture of OpenPose is CNN. It is made up of a VGG-19 convolutional network that extracts patterns and representations from the input. The VGG-19's output is split into two convolutional network branches. The first network predicts a set of confidence maps for each body component, while the second predicts Part Affinity Fields (PAFs), which is a collection of flow fields for encoding pairwise, unstructured interactions between body components and establishing a degree of relationship between parts. Pruning the weaker links in bipartite graphs is also beneficial.

Figure 2.9: OpenPose Architecture<sup>8</sup>.

According to the number of humans present in the input, the predictions from the two branches are essentially concatenated with the traits for the following step to create a human skeleton as shown in Figure 2.9. CNNs are employed in successive phases to improve the prediction .

Figure 2.10: OpenPose pipeline<sup>9</sup>.

<sup>8</sup>Courtesy:[https://www.researchgate.net/figure/OpenPose-methodology-for-the-2D-pose-detection-of-several-people-in-an-image-and-neural\\_fig1\\_332147850](https://www.researchgate.net/figure/OpenPose-methodology-for-the-2D-pose-detection-of-several-people-in-an-image-and-neural_fig1_332147850).

Figure 2.10 depicts OpenPose's overall process.

**AlphaPose (RMPE)** The top-down method to HPE is implemented via Regional Multi-person Pose Estimation (RMPE) or AlphaPose. The top-down approach to HPE introduces a great deal of inaccuracy in localization and mistakes during prediction, making it a difficult task.

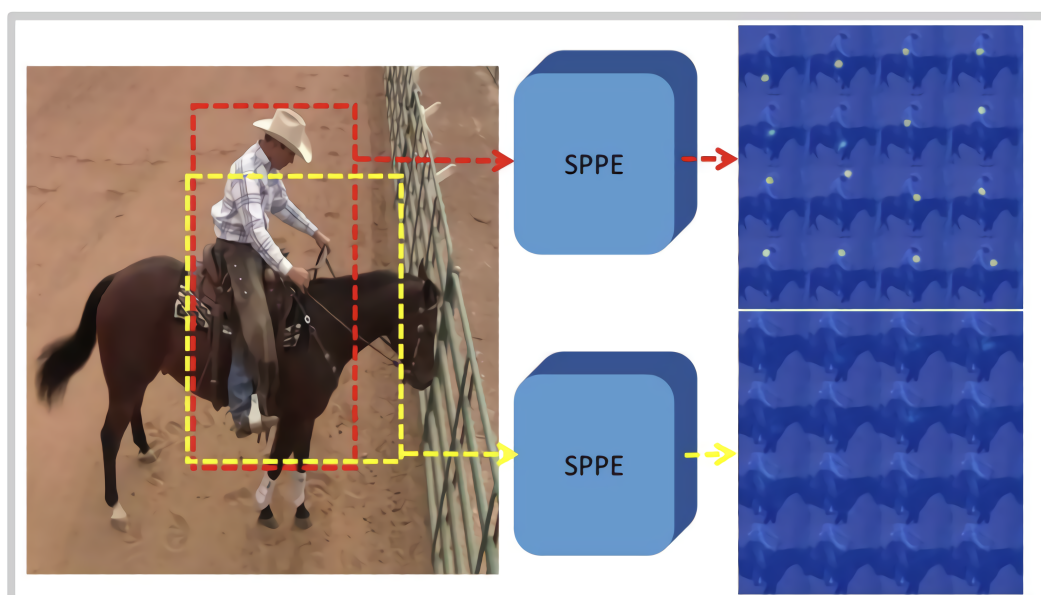


Figure 2.11: bounding box localization error [2].

For example, Figure 2.11 the red box represents the ground truth, while the yellow box represents the anticipated bounding box. When it comes to classification, however, the yellow bounding box will be regarded as the "proper" bounding box for classifying a human. Even with the "proper" bounding box, however, the human position cannot be calculated. The creators of AlphaPose devised a two-step

<sup>9</sup>Courtesy: [https://www.researchgate.net/figure/OpenPose-methodology-for-the-2D-pose-detection-of-several-people-in-an-image-and-neural\\_fig1\\_332147850](https://www.researchgate.net/figure/OpenPose-methodology-for-the-2D-pose-detection-of-several-people-in-an-image-and-neural_fig1_332147850).

methodology to address the problem of poor human detection. They incorporated two networks into this framework as shown in Figure 2.12:

1. Symmetric Spatial Transformer Network (SSTN): It aids in cropping out the relevant region in the input, simplifying the classification process and resulting in improved performance.
2. SPPE (Single Person Pose Estimator) is a tool for extracting and estimating human poses

The goal of AlphaPose is to link SSTN to the SPPE and extract a high-quality single-person area from an erroneous bounding box. By addressing invariance, this technique improves classification performance while also offering a solid foundation for estimating human posture [2].

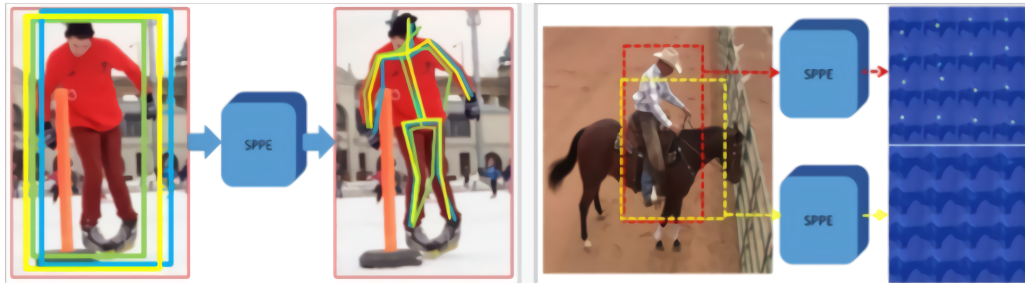


Figure 2.12: multiple poses detection of single person using AlphaPose method [2]

**DeepCut** Pishchulin *et al.*[3] presented DeepCut in 2016 with the goal of completing the tasks of detection and posture estimation concurrently. It's a bottom-up method for calculating human position. The goal was to find all potential

body parts in a given image, then classify them as a head, hands, legs, and so on, before separating the body parts that belonged to each individual as clarified in Figure 2.13.



Figure 2.13: joint subset partition and labeling for multi person pose estimation using DeepCut method [3].

The network uses an Integral Linear Programming (ILP) modeling to implicitly group all recognized essential points in the provided input, resulting in an output that resembles a human skeleton.

**Mask R-CNN** Mask R-CNN is particularly common for instance segmentation. It establishes a bounding box around the item and a segmentation mask. The model may simultaneously locate and categorize the object as shown in Figure 2.14.



Figure 2.14: result of using Mask R-CNN on coco test image [4]

Human Pose Estimation tasks may be readily added to the basic architecture. Fast R-CNN extracts features and representations from the input using CNN. The retrieved characteristics are then utilized through a Region Proposal Network to propose where the object could be present (RPN). The bounding box might have different sizes, as seen in Figure 2.14, therefore, a layer named RoIAlign is used to normalize the extracted features and make them all the same size as shown in Figure 2.15. The network's parallel branches use the retrieved characteristics to refine the suggested region of interest (RoI) and build bounding boxes and segmentation masks[4] .



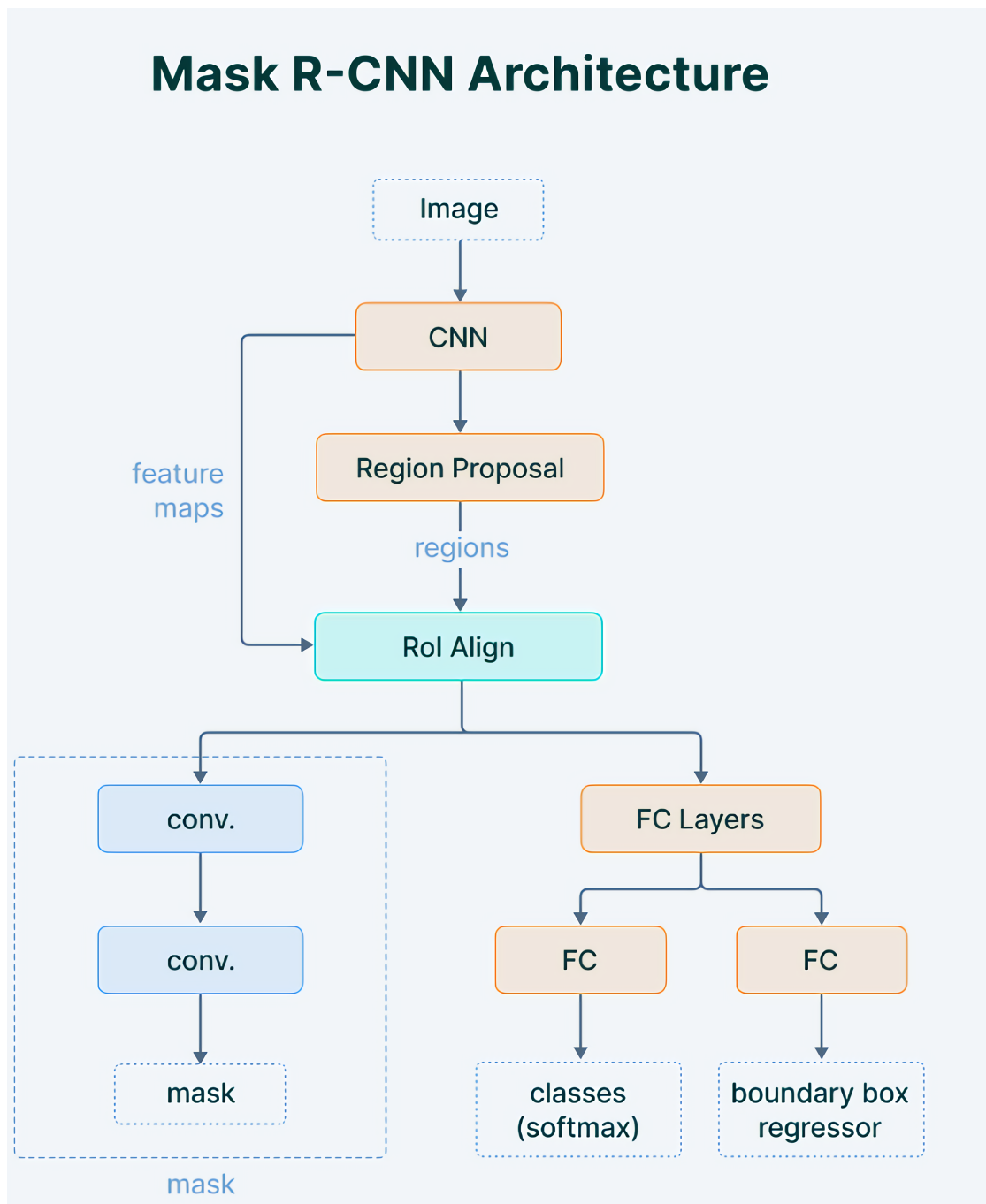


Figure 2.15: Mask R-CNN Architecture

To estimate human posture, the network's mask segmentation is utilized to recognize humans in the input image/frame. Since mask segmentation is accurate at detecting objects, such as humans, the human pose may be approximated quite

readily. This method is similar to the top-down approach where the person detection step occurs concurrently with the part detection stage. In other words, the keypoint and person detection stages operate apart from one another.

**SMPL** Loper *et al.* [5] proposed SMPL, which is a mathematical model used in many human pose estimation applications. The main characteristics of SMPL that make it common in human pose applications are its ability to generate 3d meshes that consist of about 21000 vertices. This complex structure of a large number of vertices creates a 3D mesh based on the AMASS dataset. SMPL contains two main parameters, which are pose and shape parameters. The main human form in SMPL can be demonstrated by 10 scalar values that demonstrate the shrinking and expansion of the human body in different directions. SMPL can be introduced by posing  $(24 \times 3)$  relative rotations of joints, where each relative rotation can be encoded as a 3D vector in axis-angle presentation.

**Pipeline of synthesis human body** There are 3 stages or phases to synthesizing a human body in the SMPL model, as demonstrated below:

**Shape Blend Shapes:** In this stage, a template (or mean) mesh  $T$  is created with vertex displacements that reflect how distant the subject shape is from the mean shape as shown in Figure 2.16.



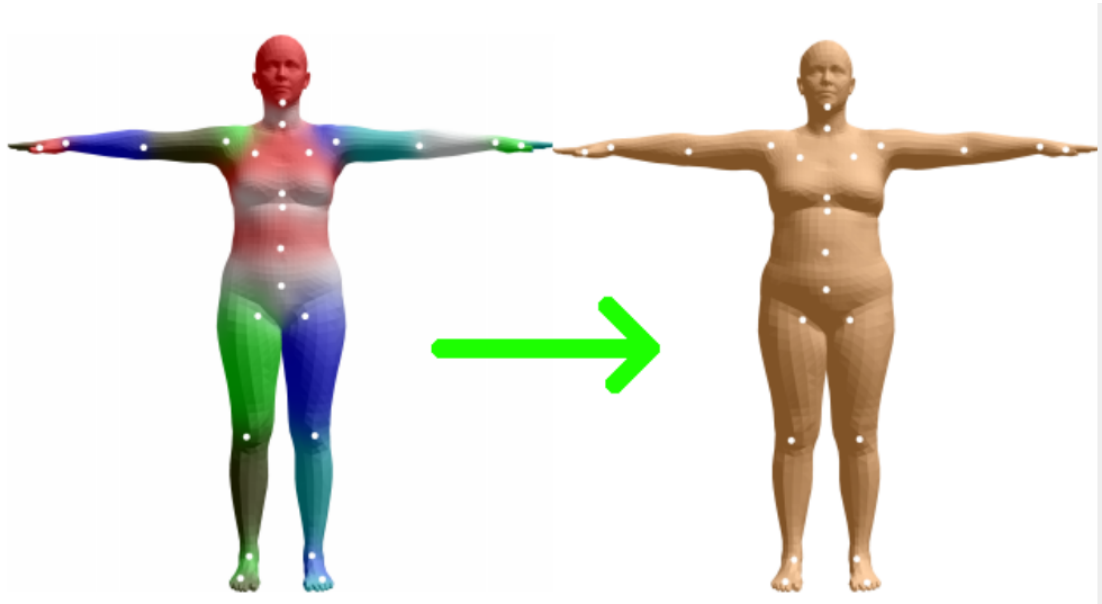


Figure 2.16: SMPL shape blend shape [5].

**Pose Blend Shapes:** After the identity mesh is generated in the rest posture, it is further enhanced by vertex displacements that describe deformation correction induced by a certain pose. In other words, the posture in the following stage, "Skinning" causes some deformation in the rest position as illustrated in Figure 2.17.

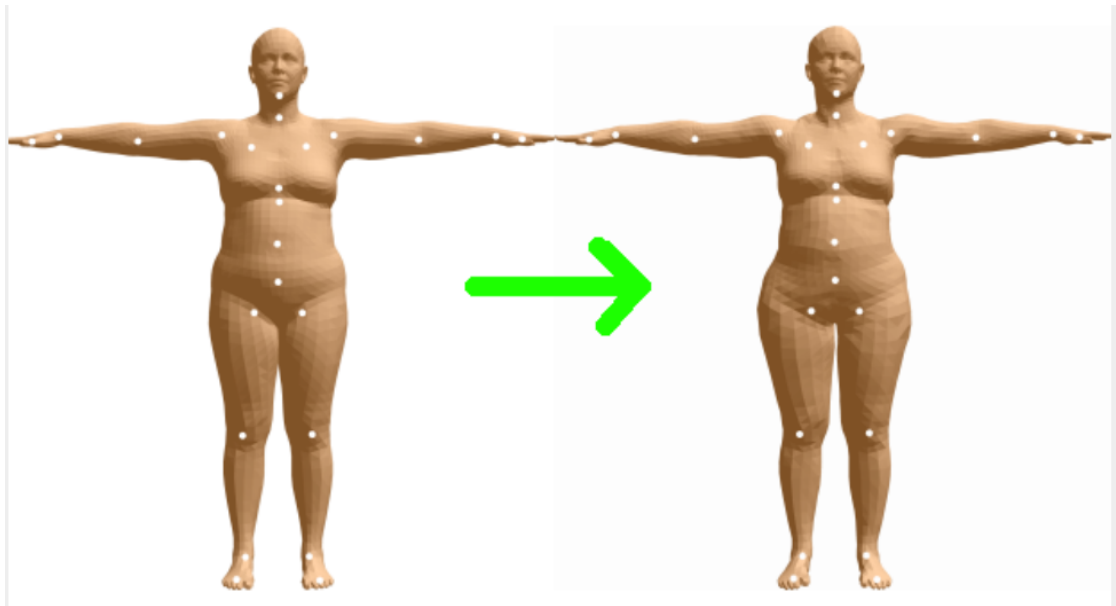


Figure 2.17: SMPL pose blend shapes [5].

**Skinning:** A weighted combination of joint deformation transforms each mesh vertex from the previous phase, i.e. the closer a joint is to a vertex, the stronger the joint rotates/transforms the vertex as clarified in Figure 2.18.

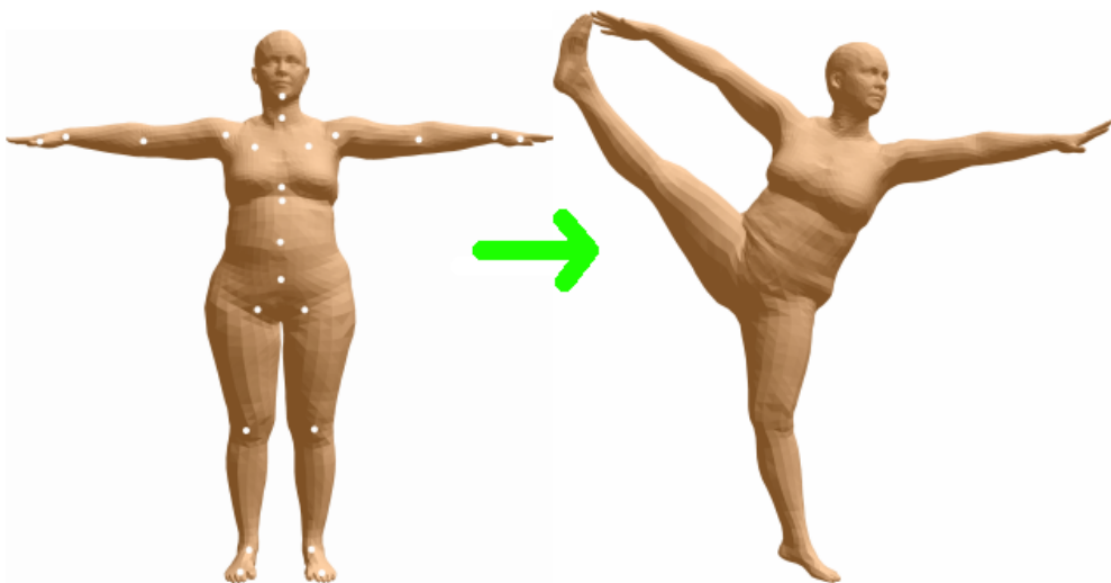


Figure 2.18: SMPL skinning [5].

As illustrated in Figure 2.19, which demonstrates white dots represent 24 joints, it is necessary to build the main form of the body skeleton based on a specific kinematic tree that is needed to determine the human pose accurately.



Figure 2.19: SMPL template mesh where white points indicate 24 joints and color indicate blend weights [5].

Each joint in 24 joints is represented by a  $(23 \times 3)$  matrix, i.e. each one of those 23 joints has 3 scalar values, which is a relative rotation of joints.

### 2.3.8 Transformation

Linear transformation is one of the most widely used fields in mathematics. It can be considered a function that maps one vector to another. A particular vector that can be reoriented (x,y) pixel values by a matrix that represents transformation can be multiplied by a particular matrix. Generally, there are two types of linear transformation (affine and projective) transformation. A projective transformation is the composition of a pair of projective projections. It discusses what occurs to the perceived position or location of seen objects when the observer's point of view changes. Transformation can be represented using the following matrix [35] :

$$\begin{pmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{pmatrix}$$

where:

$$\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix}$$

is rotation matrix that specifies the type of transformation to be performed: scaling, rotation, and so on.. .

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

is the translation vector. It simply moves the points

$$\begin{pmatrix} c_1 & c_2 \end{pmatrix}$$

is the projection vector.

If  $x$  and  $y$  are the coordinates of a point, the transformation can be done by the simple multiplication

$$\begin{pmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$$

# Chapter 3

## Literature review

This chapter presents the results of the literature review on object detection algorithms as well as human pose estimation and 3D rendering algorithms. The chapter discusses the outcomes of previous studies on the design and configuration of object detection systems, and methods of human pose estimation.

### 3.1 Object detection Algorithms related works

Classifiers or localizers are repurposed for detection in other detection systems such as SSD321, SSD513[36], DSSD321, DSSD513[37], R-FCN [38], and Retinanet-50-500 [39]. They use the model to manipulate images at various scales and locations. Image detections are high-scoring areas of the image. However, Joseph Redmon et al. [40] proposed You Only Looked Once "YOLO," which uses a single network

on the entire image, as a different approach. This network divides the image into regions and assigns bounding boxes and probabilities to each. The projected probabilities are used to weight the bounding boxes. Compared to classifier-based systems, the YOLO model has significant advantages. At test time, it examines the entire image; thus, its predictions are influenced by the image's overall context. In contrast to systems like R-CNN, which require thousands of network evaluations for a single image [41], it provides predictions with just one. This makes it a thousand times quicker than R-CNN and a hundred times faster than Fast R-CNN[42].

Traditionally, object detection identifies objects and marks them with bounding boxes in an image. However, this approach or strategy is considered wasteful, inefficient, and requires further post-processing. In their paper, [6]. Zhou et al. proposed a new approach by modeling objects as a single point. To discover center points, this detector employs keypoint estimation and regresses to all other object parameters, including size, 3D position, orientation, and even pose. This approach, which CenterNet considers end-to-end differentiable, is simpler, faster, and more precise than traditional bounding box-based algorithms. This method takes an image as input for CNN in order to generate heatmaps where the peak for each heatmap forms a center. The image features in these points are necessary for predicting the width and height of the bounding box. This method is generally used for object detection but can be used in other aspects, such as multi-person human pose estimation, as illustrated in Figure 3.1.



Figure 3.1: 3D object detection for pose estimation using object as points detection method [6].

MobileNetV2 is an extended version of MobileNetV1, a computer vision algorithm based on neural networks designed for mobile devices to support object detection and classification. MobileNetV1 [43] is appropriate for mobile devices in regards to the depthwise separable convolution that the MobileNetV1 algorithm depends on. The first layer in MobileNetV1 is a depthwise convolution, which applies a single convolution filter per input channel to perform lightweight filtering. The second layer is a  $1 \times 1$  convolution (pointwise convolution), which creates new features by estimating linear combinations of input channels. MobileNetV2 [44] introduces features such as little bottlenecks between layers and shortcut connections between the bottlenecks. It consists of two blocks. The first is a residual block with stride 1, and the second is a block with stride 2 for downsizing. Each of the two blocks in MobileNetV2 contains three layers. The first layer is  $1 \times 1$  convolution with ReLU, the second is depthwise convolution, and the last is  $1 \times 1$  convolution without non-linearity. The main structure of MobileNetv2 is illustrated in Figure 3.2:



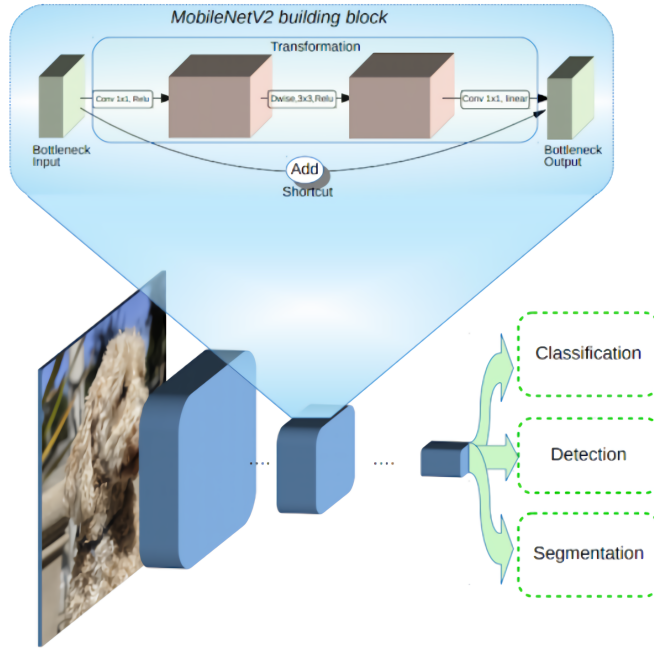


Figure 3.2: the General architecture of MobileNetv2<sup>1</sup>.

## 3.2 Human Pose Estimation And 3D Rendering

Xie *et al.* [45] presented a novel approach for training motion synthesis models from raw video pose estimates without using motion capture data in this study. This method improves noisy pose estimates by applying physics constraints via contact invariant optimization, which includes contact force calculation. The revised postures are then used to train a time-series generative model, which synthesizes future motion and contact forces. Their results showed considerable improvements in both posture estimation and motion synthesis outcomes from video using physics-based refinement.

<sup>1</sup>Courtesy:<https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html>

Kocabas *et al.*[46] proposed Video Inference for Body Pose and Shape Estimation (VIBE), a video pose and shape estimation method. It predicts the parameters of the SMPL body model for each frame of an input video. VIBE uses an existing large-scale motion capture dataset (AMASS) together with unpaired in-the-wild 2D keypoint annotations. VIBE takes a video of a single person and extracts the features of each frame in the video by using a convolutional neural network. These features are used to regress the parameters of the SMPL model at each instance, such as body pose and shape parameters. The pose parameters include body rotation and the relative rotation of 23 joints, as well as the shape parameters, which are the first 10 coefficients of a PCA shape space. SMPL is considered a differentiable function that takes the body pose and shape parameters as input and the pose of a 3D mesh as output. To extract visual characteristics, VIBE employs CNNs. The CNN output is passed into the recurrent neural network, which analyzes the sequential aspect of human movement. The body parameters for the whole input sequence are then predicted using a temporal encoder and regressor. This entire section is known as the Generator ( $G$ ) model. Realistic human motion is realized through adversarial training and the construction of a motion discriminator using the AMASS 3D dataset. The motion discriminator considers both predicted pose sequences and AMASS sampled pose sequences. The discriminator attempts to distinguish between false and genuine movements by assigning a real or fake probability to each input sequence, which aids in the production of realistic motion. This method generates a standard SMPL body model format with a sequence of postures and shapes parameters.

Rematas *et al.* in their paper[47] offer a method that converts a monocular video of a soccer game into a moving 3D reconstruction in which the players and field may be displayed interactively using a 3D viewer or with an Augmented Reality device. They construct a neural network to assess the depth of every new soccer player's image. The hourglass network architecture is used in this approach: the input is processed by a series of hourglass modules—a series of residual blocks that decrease the input resolution and then upscale it—and the output is depth estimations. The Adam solver trained the network for 300 epochs with a batch size of 6 and a learning rate of 0.0001. They use many stages for 3D reconstruction from soccer videos. The first phase is camera pose estimation, which is used to estimate the frame-by-frame settings of the actual game camera. The global origin is aligned with the center of the synthetic soccer field on the  $y = 0$  plane. The field line is the most consistent element on the pitch (e.g., touchline, penalty area around the goal). As a result, the edge point E for each frame is extracted to identify these properties by finding the camera settings  $w$  (focal length, rotation, and translation) to match the generated synthetic field line to the acquired edge point. Use a structured forest to identify the lines. The second stage in video analysis is to recognize the players in each frame. While recognizing soccer players may appear simple owing to the generally consistent background, most previous person detectors nevertheless struggle when, for example, players from the same team obscure each other or the players are too small. They begin with a collection of bounding boxes for detecting the video sequence, where each track has a beginning and finishing point. They begin by treating all identified neck

keypoints from all frames as distinct tracks and calculating their pairwise distances. The third phase is temporal instance segmentation by using a segmentation mask that helps the depth estimation network by supplying a segmentation mask in which the monitored player is in the foreground and the field, other players, and the stadium are in the background. The last phase is mesh generation, where the foreground mask is combined with a cropped image and fed to the depth estimation neural network. The output of the network is the relative quantized signed distance per pixel between the player's surface and the virtual plane. The depth map is projected into world coordinates using the camera settings, resulting in a 3D player point cloud. Each pixel corresponds to a 3D point, and it uses pixel connections to identify faces. Finally, the mesh is a texture with the input image.

## Chapter 4

### Our Approach

Mixed (Augmented/Virtual) reality interfaces allow the viewer to determine the view parameters freely. However, generating content for these technologies is often challenging; it is comparable to preparing content for a video game or a movie. To obtain high-quality content, one needs to write a script, design scenes, objects, and characters, and animate them according to the script. In addition, these objects and characters must be registered on the real-world coordinate. This research presents a novel approach to enable users to view sports via a virtual reality interface. Within the proposed framework, spectators can view the field and the players from any point of view, at any zoom anytime, including playback of previous events. Our framework extracts pose features from every player in the video stream and applies these features to predefined 3D models of the respective players within a synthesized sports field. We assume the availability of an accurate

3D model of every player in the game and the appropriate area. Current modeling and animation technologies can model and animate sports satisfactorily, e.g., FIFA games from Electronic Arts [1]. Viewing these games within a virtual reality framework in real time is feasible. Our novel framework extracts appropriate and accurate pose features from each player and applies these features to the model of the player leading it to have the same pose and position in the virtual field according to the actual one in real time as illustrated in Figure 4.1.

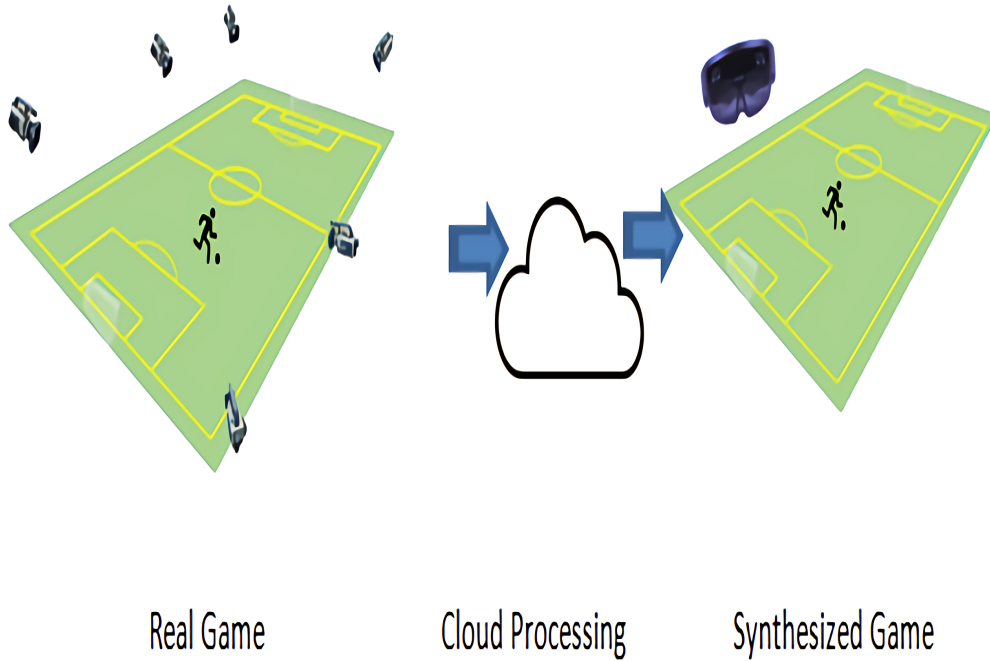


Figure 4.1: Mapping of real game to virtual environment

Currently, it is feasible to generate the model of every player in the match with high accuracy. In our framework, we extract players' skeletal and skin features from the video stream using deep-learning models and apply these features to animate their respective 3D avatars. Toward this goal, we detect each player and track his

movements over the video stream. A player may appear and disappear within the video stream across the game, i.e., we need to detect and track players continuously. We detect each individual player and extract two classes of pose features: skeletal and skin. The skeletal features are extracted using a deep learning model, which determines key points that form the character skeleton. For skin, we can apply a deep learning model that obtains pose features from the player’s skin, including the deformation of the clothes. To animate the 3D model of a player through, its avatar, according to its performance within the video stream, we utilize the extracted pose features and the character’s animation model. The extracted pose features may not be enough to determine the avatar’s pose at each frame due to partial occlusion or false pose detection of the respective player. In such cases, the animation model will fill the gap and provide smooth animation of the avatar.

**Pose Feature Extraction** Detection and analysis of human characters have attracted the interest of researchers, and an enormous body of research has been developed. Deep learning technologies have led to impressive improvement in character detection [48], pose estimation[49, 3], and tracking [50]. We apply multi-character detection and tracking to track the movements of every player over the available set of cameras and select the best view in terms of visibility and coherence. We then apply character pose estimation and extract skeletal and skin pose features using deep learning. We developed a deep-learning model which generates a sequence of the 3D skeleton and skin features from a sequence of frames for each player. The architecture of the model includes extracting features from

each player using CNN from each frame and then applying Transformers blocks to map these features to the skeleton and skin features. The transformer block depends on GANs, where generative adversarial networks have a large effect on image modeling and synthesis. Many research studies have used GANs with recurrent networks for sequence-to-sequence tasks. Many researchers combine GANs and RNN for estimating and predicting future motion sequences based on previous ones or using them to generate human motion sequences. In this study, adversarial training was used to refine predicted poses based on sequential input data. Our framework used a motion discriminator with recurrent architecture and an adversarial network with 3D motion capture data to do so. The transformer block contains a generator and a discriminator. The generator depends on a recurrent neural network, where the idea behind employing a recurrent architecture is that previous video pose data might be useful for upcoming frames. When a person's pose is unclear or their body is partially obscured in a frame, this is helpful. In this case, historical data can assist clarify and limit the pose estimate. The generator produces the relevant poses and shape parameters in each frame given a series of frames  $I_1, \dots, I_T$ . A convolutional network,  $f$ , is fed a sequence of T frames and acts as a feature extractor, producing a vector  $f_i$  for each frame,  $f(I_1), \dots, f(I_T)$ . A Gated Recurrent Unit (GRU) layer receives these and produces a latent feature vector  $g_i$  for each frame,  $g(f_1), \dots, g(f_T)$ , depending on the preceding frames. Then, we input  $g_i$  to regressors with iterative feedback. The regressor receives the current pose as input and is initialized with the mean pose. The generator overall loss is made up of 2D (x), 3D (X), pose, and shape losses when such data are



available as in the following Equation.

$$L_G = L_{3D} + L_{2D} + L_{SMPL} + L_{adv}$$

where each loss can be calculated as :

$$L_{3D} = \sum_{t=1}^T ||X_t - \hat{X}_t||_2$$

$$L_{2D} = \sum_{t=1}^T ||x_t - \hat{x}_t||_2$$

$$L_{SMPL} = ||\beta - \hat{\beta}||_2 + \sum_{t=1}^T ||\theta - \hat{\theta}||_2$$

$$L_{adv} = \mathbb{E}_{\theta \sim p_G} [(D_M(\hat{\theta}) - 1)^2]$$

To determine if the created series of poses match a genuine sequence, we use a motion discriminator called  $D_M$ . A multi-layer GRU model is given the generator's output as its input. Then a linear layer predicts a value  $[0, 1]$  for the likelihood that belongs to the manifold of plausible human movements, which is the final prediction. The main loss in Discriminator can be calculated as shown in the following equation :

$$L_{D_M} = \mathbb{E}_{\theta \sim p_R} [(D_M(\theta) - 1)^2] + \mathbb{E}_{\theta \sim p_G} [(D_M(\hat{\theta}))^2]$$

where  $p_R$  is a real motion sequence from the AMASS dataset, while  $p_G$  is a gen-

erated motion sequence.  $D_M$  is designed to utilize temporal coherence among the features of each player over the span of the sequence. Figure 4.2 presents the model architecture.

**Player Models (Avatars)** In this section, we overview the generation and animation of player models, in which we denote player avatars. To generate a player’s model, we leverage available art in this domain, which ranges from manual modeling to 3D scanning and model fitting. We represent the generated model using the SMPL body model, as illustrated in Figure 4.3, which supports the dynamic animation of a body by manipulating its skeleton [5]. To display the game within an augmented reality framework, we register the field on a large enough indoor region, which could range from the floor of a room to a wall or tabletop. The stream includes the position of each player and its pose features, which include enough information to position an avatar in the same place at the same pose as his respective player.

**Unity Engine** We created a 3D model in the Unity engine that is similar to the SMPL model, with 24 joints, where the pelvis is the root joint and the other joints have a relative rotation joint based on the root joint. Our system works as a client-server architecture, where extracting features, finding poses, detecting, and extracting players are all done on a Python server that generates an array of JSON object frames. Each frame consists of many people, and each person has a (person\_id, bbox, poses). Many clients received JSON over HTTP connections

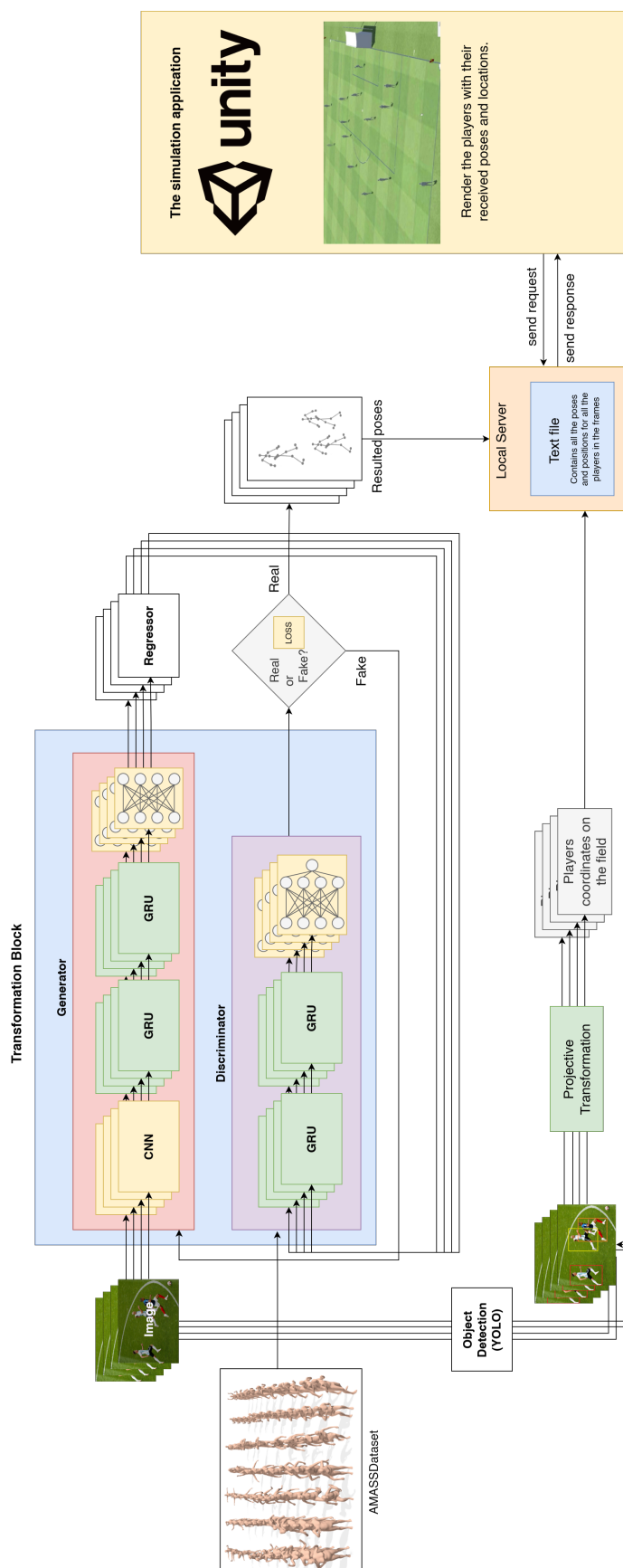


Figure 4.2: The General architecture of our framework

and attempted to parse it into a real object in C# with the same attributes as the JSON object. With the availability of relative rotation poses for each joint, we can render these poses to a 3D model as the real players' poses. The location of each player in the synthesized field should be the same as in the actual field, but our approach so far gives us the position according to the image coordinates. To solve this problem, we use projective transformation for each frame and take the updated location of each player after transformation. To use projective transformation, we must designate four source points and four destination points into which the image should be transformed. To differentiate one player's location from another, we created a color map with 22 distinct RGB color codes as the maximum number of players that may exist at the same time in a stadium. Each player has an ID that corresponds to a specific color in the color map that is located in a specific position in a black image that has the same dimensionality as the real image after the projective transformation. In this way, we extract the position of each player in the image by the specific color that reflects the player's ID.

|            |      |               |      |
|------------|------|---------------|------|
| 'Pelvis',  | # 0  | 'Neck',       | # 12 |
| 'L_Hip',   | # 1  | 'L_Collar',   | # 13 |
| 'R_Hip',   | # 2  | 'R_Collar',   | # 14 |
| 'Spine1',  | # 3  | 'Head',       | # 15 |
| 'L_Knee',  | # 4  | 'L_Shoulder', | # 16 |
| 'R_Knee',  | # 5  | 'R_Shoulder', | # 17 |
| 'Spine2',  | # 6  | 'L_Elbow',    | # 18 |
| 'L_Ankle', | # 7  | 'R_Elbow',    | # 19 |
| 'R_Ankle', | # 8  | 'L_Wrist',    | # 20 |
| 'Spine3',  | # 9  | 'R_Wrist',    | # 21 |
| 'L_Foot',  | # 10 | 'L_Hand',     | # 22 |
| 'R_Foot',  | # 11 | 'R_Hand',     | # 23 |



Figure 4.3: Mapping between SMPL joint id's and joint names

**Football Environment** We have chosen Dura Stadium for our experiments, knowing that its dimensions are similar to the standard stadiums in the world, with a 110m length and 68m width. We have created a video for a game scenario between Dura and the Samu Team. In the beginning, we checked all the dimensions of Dura Stadium, including the length and width of the entire stadium, in addition to checking the length and width of the middle and goalkeeper areas, as illustrated in Figure 4.4. Before making videos in the stadium, we thought theoretically that we should need 4 cameras in the corners to cover the entire stadium and all the players, but we were surprised that there is no stadium bench in the east of the stadium that allows us to put cameras in each corner. Because of that, we have distributed cameras along the entire west bench of the stadium. That makes the cameras cover the entire stadium but in an inaccurate way. For that reason, we have excluded camera-4 and have used the 3 cameras, which are camera-1, camera-2, and camera-3. For implementing projective transformation, we need 4 points, which are corners, as source points to be transformed later into destination points that identify the entire dimension of the stadium. That will be the 3D stadium model's input into Unity. Figure 4.5 , Figure 4.6 and Figure 4.7 depict projective transformation for cameras-1, camera-2, and camera-3 respectively. We used some mathematical equations to calculate these points for camera-1, camera-2, and camera-3, which are shown clearly in Appendix A.

**virtual reality** We have implemented virtual reality in our project using Unity Engine through Google Cardboard, which is an inexpensive device that powers

virtual reality, which is commonly used in smartphone devices. So all we need to create a virtual reality simulation of our framework is cardboard, a smartphone, and Android apps. We have applied virtual reality to our project by enabling the Extended Reality (XR) plugin in Unity, which is considered an umbrella that encapsulates Augmented Reality (AR), Virtual Reality (VR), Mixed Reality (MR), and everything in between. So we use the XR plugin in Unity as a tool that helps us give the user a good experience in a free-viewpoint video by providing variable perspective movement in 3D space. For that purpose, we have placed many beacons in the field that allows the player to move from different positions in the stadium by gazing at these objects, so if the player gazes correctly at this beacon by trying to make the beacon in the center of the phone screen, the player moves to that location. So, to do this, we cast rays toward the camera in a forward direction to detect if the beacon is gazed at or not, and we have two types of beacons, one for movement and the other for starting the game, so if a user with a phone device gazes at a beacon with a start type, the game will start; otherwise, when gazing at a beacon with a movement type, it will move to the specific location that beacon has.

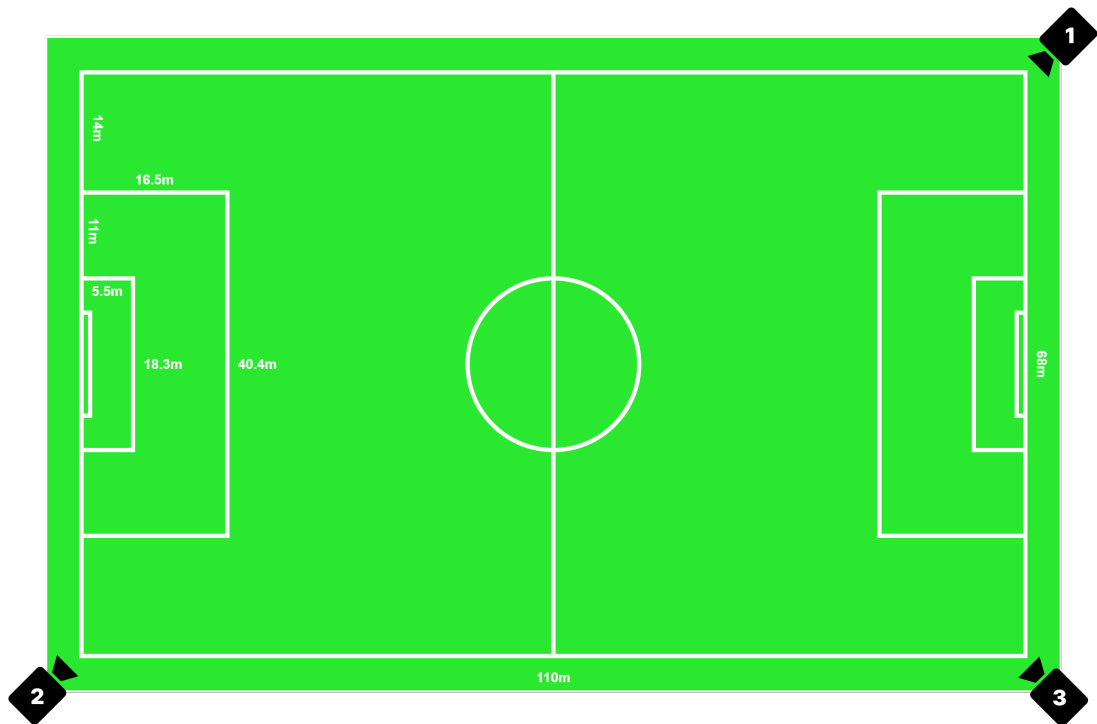
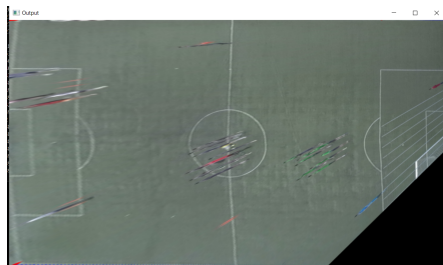


Figure 4.4: Dura staduim

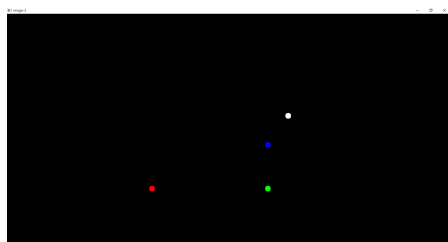




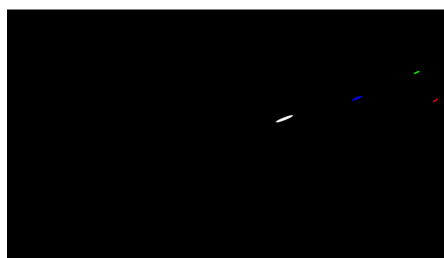
(a) Input Image



(b) Output image



(c) coordinates of image before transformation



(d) coordinates of image after transformation

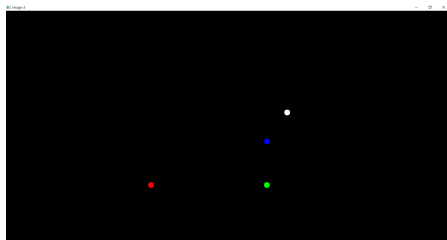
Figure 4.5: Camera-1 Projective Transformation



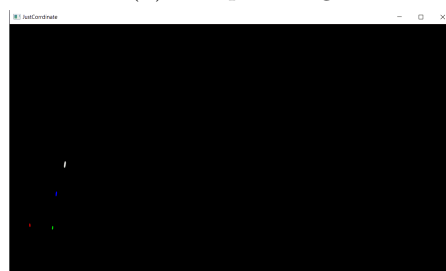
(a) Input Image



(b) Output image



(c) coordinates of image before transformation

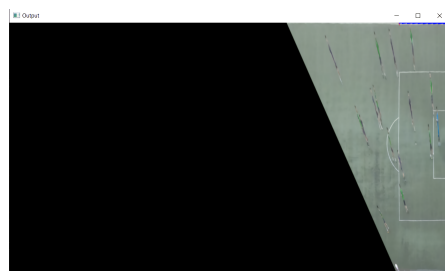


(d) coordinates of image after transformation

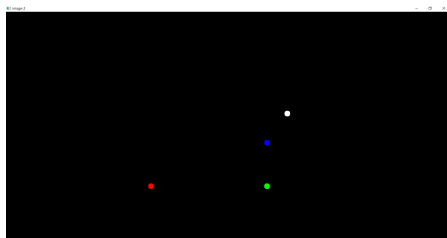
Figure 4.6: Camera-2 Projective Transformation



(a) Input Image



(b) Output image



(c) coordinates of image before transformation



(d) coordinates of image after transformation

Figure 4.7: Camera-3 Projective Transformation

## Chapter 5

# Experiments and Results

This chapter displays the experiments for reconstructing a 3D player and for calculating the time of player detection and rendering. This chapter compares our suggested framework with the conventional method for estimating human poses and contains virtual reality data, by allowing the user to observe a match from several viewpoints and perspectives. We have shown in this chapter the number of detected and non-detected players in each frame. for the two videos that have been taken from cameras 1 and 2.

We applied our experiments to video that had been taken from three cameras in Dura Stadium. The video has 375 frames. Each frame has a different number of players depending on the camera from which it was captured. As an example, camera 1 contains nearly 24 players as shown in Table [5.3](#); camera 2 contains nearly

Table 5.1: the number of detected and undetected players in their respective responding frames in Camera-2

| Frame No | Actual | Not Detected | Detected |
|----------|--------|--------------|----------|
| 1-34     | 7      | 0            | 7        |
| 34-44    | 7      | 1            | 6        |
| 44-68    | 10     | 1            | 9        |
| 69-176   | 8      | 0            | 8        |
| 177-179  | 10     | 1            | 9        |
| 200-215  | 9      | 1            | 8        |
| 215-235  | 9      | 0            | 9        |
| 236-243  | 9      | 1            | 8        |
| 244-375  | 9      | 0            | 9        |

10 players as clarified in Table 5.1; and camera 3 contains 15 players as illustrated in Table 5.2. The number of players that have been detected in a single frame is higher than the number of undetected players in that specific frame, as can be seen from Table 5.1, which displays the frame number interval and associated detection or non-detection for players in the stadium. As previously said, from frame 1 to frame 34, there are 7 players who have all been 100% successfully identified in our frame. However, there are 10 players between 44 and 68. Nine players have been detected, while one has gone unreported. Therefore, there are 90% of observable humans in these frames.

Table 5.2: the number of detected and undetected players in their respective responding frames in Camera-3

| Frame No | Actual | Not Detected | Detected |
|----------|--------|--------------|----------|
| 1-7      | 15     | 3            | 12       |
| 44       | 15     | 11           | 4        |
| 68       | 15     | 5            | 10       |
| 75       | 15     | 1            | 14       |
| 88       | 15     | 6            | 9        |
| 128      | 15     | 5            | 10       |

Table 5.2 illustrates the result of a camera that has been captured in the south-

west corner. It shows that from frame 1 to frame 7, our framework successfully detected 12 out of 15 players in the stadium, implying that the percentage of detected players among all players is 80%. However, in frame number 44, the number of detected players in a specific frame changed dramatically to 4, and the number of undetected players changed to 11 from 15, which means the rate of the number of detected players in frame 44 is 26.6%. However, the number of detected players in frames 88 and 128 is nearly 9 or 10, with a nearly 60-66.6% success rate.

Table 5.3: the number of detected and undetected players in their respective responding frames in Camera-1

| Frame No | Actual | Not Detected | Detected |
|----------|--------|--------------|----------|
| 1-68     | 24     | 19           | 5        |
| 68       | 24     | 16           | 8        |
| 82       | 24     | 13           | 11       |
| 94       | 24     | 15           | 9        |
| 105      | 24     | 16           | 8        |
| 125      | 24     | 16           | 8        |
| 142      | 24     | 14           | 10       |

Table 5.3 shows the camera captures taken from the stadium's southeast corner. As shown in the preceding table, the rate of detected players from this camera is 20.8% because from frame 1 to frame 68, there were 5 detected players while there were 19 not detected, which is more than twice as many as there were detected players. Nearly the same number of players were found in frames 105 and 125 using our framework with a 32% detected rate. According to the position, distance, and length of our camera's view from the players, we have deduced from this camera that the number of undiscovered players is often twice as high as the number of identified players.

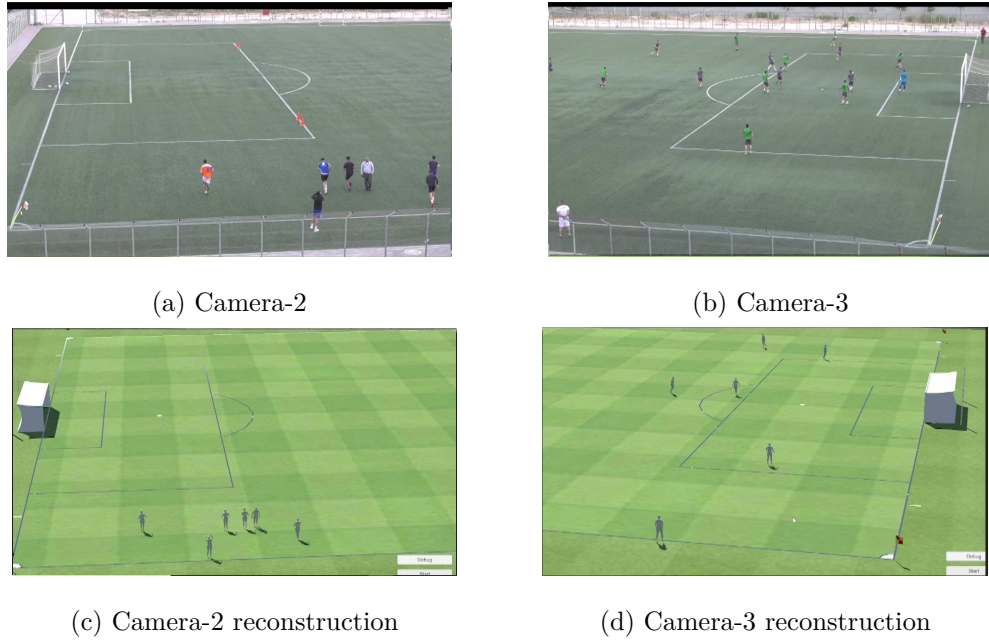


Figure 5.1: Camera-2 and Camera-3 reconstruction for Frame-1

Figure 5.1 has been taken from the final video, which consists of the videos from two selected cameras that have the highest detection rate. For that reason, we have excluded camera-1 from 3D player reconstruction according to the lowest player detection rate caused by the long distance between the player position in the stadium and the camera, but we have used it to determine the dimensions of the synthesis field that are related to and similar to the actual one. Regarding Figure 5.1, especially Figure a and c, our framework detects six players and reconstructs 3D players with the same elevation and pose rotations. In addition, our framework has the ability to identify the positions of each player precisely in a synthesis field that is similar to the position in a real field. However, in Figure b and d, in the real stadium image, we can observe that the number of real players is greater than the number of avatars in the constructed stadium and synthesis

field, and that refers to some frames that have been captured from this camera that have a higher rate of undetected players compared with another camera. As seen in Figure 5.1, our framework constructs 6 players from a total of 13 players.

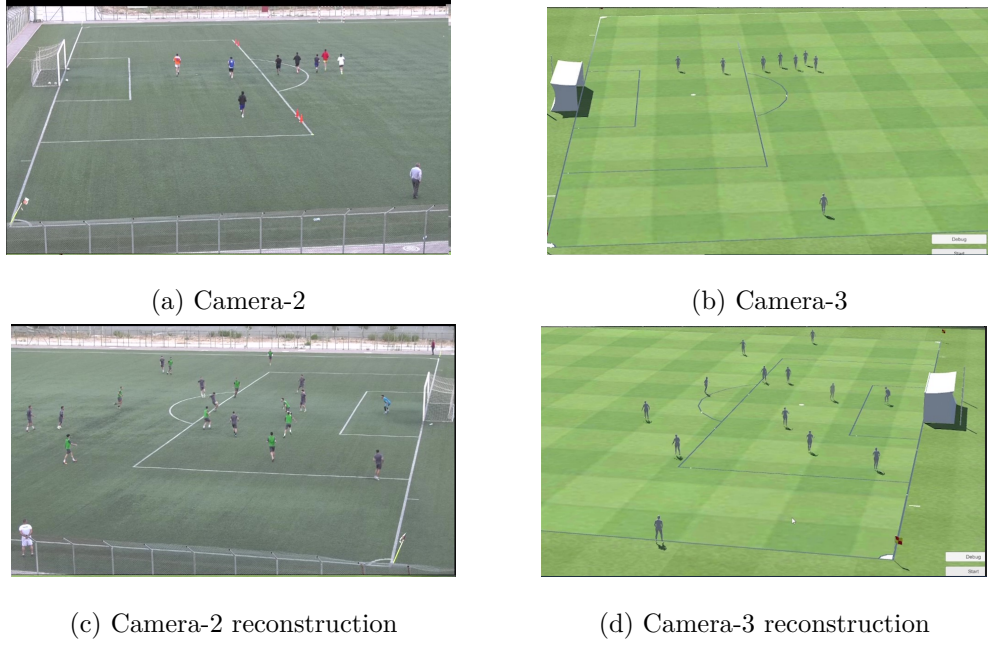


Figure 5.2: Camera-2 and Camera-3 reconstruction for Frame-2

Figure 5.2 depicts the avatar reconstruction for almost every player in the stadium, either in the left or right camera. Moreover, as observed from the image of this frame, the high accuracy in the position of each avatar corresponds to the real player in the stadium.

**Partial Occlusion** Human pose estimation is regarded as one of the most difficult tasks due to changes in body appearance caused by clothing, arbitrary and partial occlusion, the viewing angle, and background context. Pose estimation must be resistant to difficult real-world changes like illumination and occlusion.

Therefore, to detect the fine-grained joint coordinates, image-processing models must overcome this difficulty. It is particularly challenging to follow tiny, almost perceptible joints. For that purpose, we have detected each player using the YOLO algorithm. When there is no detection for a specific player in YOLO, then there is no estimation for the poses. However, when there is detection in YOLO and we find that there is partial occlusion, in this situation our framework estimates poses and fitting for the 3D model and fills the occlusion as continuity of the predicted joint in the visible 3D model part by assuming that we are dealing with a fully human body and the algorithm fits this general body to the visible part, but we can't guarantee if this is exactly correct or not according to the hidden part position. It just makes the prediction that the poses in the hidden one should be connected to the visible joints and infers the poses of child's joints according to their roots by taking advantage of the fact that all joints are relative rotation joints. In the future, we will try to detect the player's time sequence, reconstruct it, and validate it. In Figures 5.3 and 5.4, we make a comparison between the MobileNetV2 model, which is used for estimating human poses by using just CNN, and our framework, which uses GANs in addition to CNN for refining the estimated poses to be similar to and more close to the real ones. Figure 5.5, 5.6, 5.7, and 5.8 clearly show that our framework was successful in reconstructing a 3D model in all situations. Even if there is partial occlusion, our framework will fill the gap and predict the invisible pose for the hidden body parts. In contrast, Figure 5.4 shows poor results, especially when dealing with partial occlusion, which is evident clearly in the joints of the legs. MobileNetV2 doesn't have the ability to estimate the right poses for the



leg's joints when the legs are invisible.

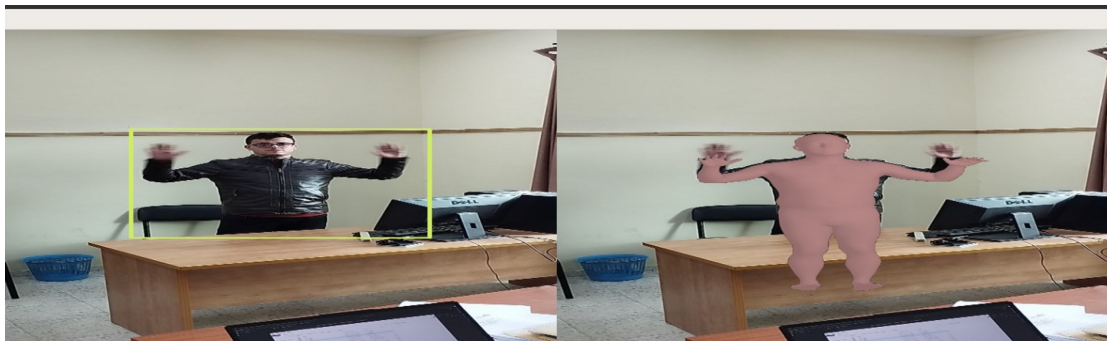


Figure 5.3: 3D model generated by our framework for person standing behind the table as a test for partial occlusion



Figure 5.4: human pose skeleton for MobileNetV2 for a person stand behind the table as a test for partial occlusion

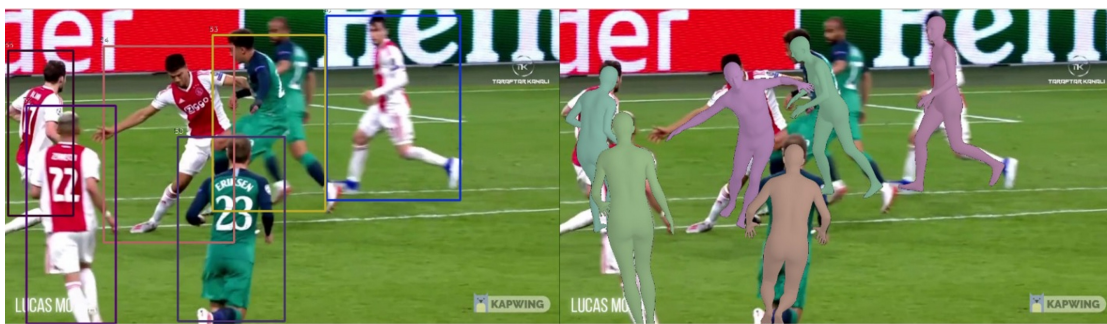


Figure 5.5: 3D models generated by our framework for each player in a soccer match



Figure 5.6: 3D model generated by our framework for person standing behind the table as a test for partial occlusion



Figure 5.7: 3D models generated by our framework for two pedestrians standing behind the sofa with different poses as a test for partial occlusion



Figure 5.8: 3D models generated by our framework for two pedestrians standing behind the sofa with different hand poses as a test for partial occlusion

Table 5.4: detection and rendering elapsed time for both the traditional and our proposed approach.

| <b>Camera Number</b>   | <b>CAMERA 1</b> |              | <b>CAMERA 2</b> |              |
|------------------------|-----------------|--------------|-----------------|--------------|
| VIBE/ Proposed Results | VIBE            | Proposed     | VIBE            | Proposed     |
| Detection time         | 2.5 minutes     | 5.64 minutes | 3.4 minutes     | 6.68 minutes |
| Rendering time         | 7 minutes       | 27 seconds   | 8.85 minutes    | 27 seconds   |
| Total time             | 9.5 minutes     | 5.91 minutes | 12.25 minutes   | 6.95 minutes |

## 5.1 Performance

Our framework takes time to detect multiple people at the same time based on the number of players on the field or stadium. As an example, if we apply our framework for extracting features, rendering 3D models, and making avatars for the players in a 300-frame video that contains 1 person or player, the elapsed time would be less than 0.5 minutes for detection and nearly 4 minutes to render all the frames in the video in the VIBE approach. However, according to our framework, the detection would take 1 minute, and rendering this frame would take less than 25 seconds. Our framework takes less time than the VIBE approach because the VIBE approach needs to render 21,000 vertices, 24 joints, and 24 poses every time and tries to draw the SMPL model above each player, so rendering all these processes takes a long time. However, in our approach, we have a predefined 3D model, like the SMPL model, and just need to give the poses of all players in all frames to Unity in order to render it directly with a time that is similar to the actual video time. We have applied our framework to the two videos taken from cameras 1 and 2, and the detection time and rendering time for the VIBE

approach on camera 1 are 2.5 minutes and 7 minutes, respectively, with a total of 9.5 minutes. However, the time for our approach is 5.64 minutes and 27 seconds, respectively, with a total time of 5.91 minutes. The time the VIBE approach takes for detecting and rendering all the players in camera 2 is 3.4 and 8.85 minutes, respectively, with a total time of 12.25 minutes. In our approach, however, the detection and rendering times for all players are 6.68 minutes and 27 seconds, for a total of 6.95 minutes as shown clearly in Table 5.4. All these results confirm that our approach is two times faster than VIBE approaches.

Table 5.5: Time analysis of our framework for different 565 frames videos

| Number of players | Average total time | Standard deviation |
|-------------------|--------------------|--------------------|
| 1                 | 110.23 seconds     | 3.55 seconds       |
| 2                 | 129.615 seconds    | 1.19 seconds       |
| 4                 | 159.441 seconds    | 1.45 seconds       |
| 6                 | 200.381 seconds    | 1.16 seconds       |
| 8                 | 231.868 seconds    | 2.38 seconds       |
| 10                | 258.686 seconds    | 1.43 seconds       |
| 12                | 281.684 seconds    | 1.58 seconds       |

We ran 10 experiments in our framework for each of seven videos. Each video has 565 frames with different detected players, as shown in Table 5.5. We calculate the average total time for each video, considering that each video contains a different number of players. Total time is the time needed for tracking, detecting, and generating pose and shape parameters for each player. Generally, the total time increases linearly with increasing numbers of detected players, as shown in Figure 5.9.

We make all these experiments on a personal computer that has a core i7 6700

3.40 GHZ, 16 GB of memory, and an 8 GB Nvidia Geforce GTX 1070. Currently, we concentrate on proofing the concept. The running time on the server side is determined by detecting the players and computing the pose estimation. YOLO runs in real-time and we can pass the pose estimation to a multi-process machine that can compute this in parallel. Bare in mind that pose estimation for one player is done in real-time. On the client side, we receive the pose and project to each player and render the scene. This could be done in real-time because the number of polygons on all the players is small and projecting the pose to an SMPL model includes several matrix multiplications that could be done in real-time on a modest machine with GPU. This is a very light hardware requirement compared to the computation power required by FreeD, for example.

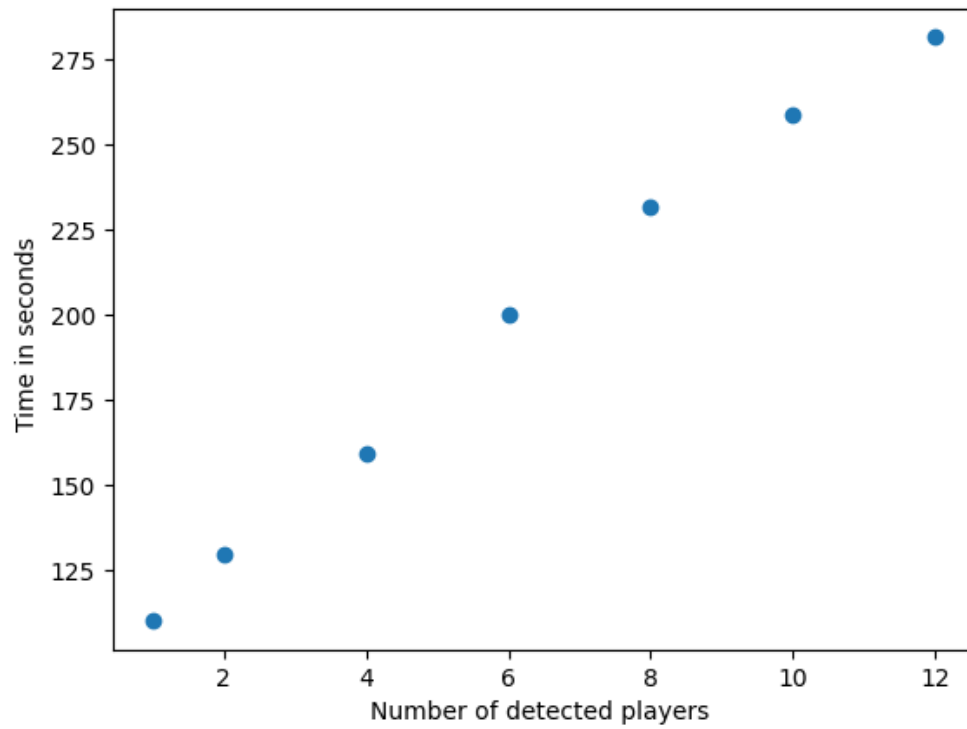


Figure 5.9: Performance of our framework according to different numbers of detected players

## Chapter 6

# Conclusions and Future Work

In this thesis, we have designed and implemented a virtual reality framework that has the ability to predict human poses and body models using deep learning models that extract features from each player using CNN and feed these features to RNN to regress all the human body joint poses. We have used GANs networks in this model in order to refine the resulting poses and shapes for each player. For this purpose, our framework contains a generator and discriminator that compete with each other to generate poses that are similar to real human poses. The idea behind using recurrent neural networks is the ability of our framework to use past frames for predicting future frames, especially in partial occlusion situations. The combination of GANs and recurrent networks makes our framework robust against partial occlusions and generates reliable and plausible human motions, as has been shown clearly in the Figure [5.5](#), [5.6](#), [5.7](#), and [5.8](#). We store these

---

pose and shape parameters in a file and pass these parameters to the client by making use of the HTTP service, where the client makes the request and the server responds to that request. Generally, we succeeded in reconstructing 3D models for each player with the same poses as the actual ones by rendering an SMPL model for each player and feeding it with the 72 poses for relative rotation joints and 10 principal component analysis shape parameters that come from the server. We have used projective transformation to map the position of each player in the image to their real position on the field. Using projective transformation gives consistency between the player's position in the real and synthesis fields. Moreover, we build a framework that allows users to view sports via a virtual reality interface by activating the XR plugin in the Unity engine and using Google Cardboard, which is suitable for and compatible with virtual reality devices and mobile phones. This framework has the ability to view the field and the players from any point of view by adding many movement beacons in the stadium. All the user needs to do is gaze for these beacons, which give him the ability to move freely inside the stadium from any angle and point of view. We analyze the performance of our framework both in detection and rendering time. We calculate the detection time for the main components in our framework, and in addition, we determine the stage where bottlenecks happen in order to solve that problem in the future. We conducted many experiments with different numbers of players and found that the detection time was generally directly proportional and increased with an increasing number of players. We outperformed the rendering process of existing research, where the total time in our framework is faster than VIBE,



especially rendering time. We explored the performance of this system using a real soccer game in Dura Stadium and were able to detect most of the players that were close to the camera. We are confident that adding cameras on all sides will enable the detection of all the players and determine their poses accurately. In the end, We have designed a client-server architecture in which pose feature extraction and deep learning models are performed on the server, and 3D models for each player are rendered on the client.

In future work, we will address many issues such as finding the depth estimation of the flying ball and trying to reconstruct the ball using stereo vision. In addition, we will try to make our framework run in Realtime by following the same methodology of video streaming such as web Real-Time Communications (WebRTC), HTTP Live Streaming (HLS), and Dynamic Adaptive Streaming over HTTP (MPEG-DASH). Similarly, the features stream will be split into chunks of several seconds, compressed, and transmitted. Each chunk will have its own index and additional attribute to enable collecting them correctly on the client side and adapting to various display sizes, client processing power, and the quality of the communication channel. Moreover, we will try to make accurate detection for joints and completion for missing frames.

# Appendix A

## Cameras Equations

### A.1 Camera-1 equations

Points : (1674,888) (1755, 628)

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (\text{A.1})$$

$$D = \sqrt{(1674 - 1755)^2 + (888 - 628)^2}$$

D=281.5 pixels

point 2 and right point (bottom)

(1647 , 888) (840 , 758)

$$D = \sqrt{(1674 - 840)^2 + (888 - 758)^2}$$

D= 821 pixels

Point 2 and right point (top)

$$y - y_1 = m(x - x_1) \quad (\text{A.2})$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}, m = \frac{888 - 628}{1674 - 1755}, m = \frac{260}{-108}, m = -2.4$$

by substitute in equation 4.2

$$y - 888 = -2.4 (x - 1674)$$

$$\sqrt{(x - 1674)^2 + (y - 888)^2} = 1876 \text{ pixels}$$

$$XD = \frac{821 \times 68}{17.5} = 3190 \text{ pixels}$$

$$y - y_1 = m(x - x_1)$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}, m = \frac{888 - 735}{1674 - 840}, m = \frac{153}{807}, m = 0.19$$

$$\sqrt{(x - 1674)^2 + (y - 888)^2} = 3190 \text{ pixels}$$

possible solutions (point 1 is done)

$$x = 925.4, y = 2619 \text{ (not approved)}$$

$$x = 2368.5, y = -843.6 \text{ (approved)}$$

## A.2 camera-2 equations

$$y - y_1 = m(x - x_1)$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}, m = \frac{90 - 80}{335 - 930}, m = \frac{-10}{595}, m = -0.16$$

$$y - 90 = -0.16 (x - 335)$$

$$\sqrt{(x - 335)^2 + (y - 90)^2} = 3500 \text{ pixels}$$

Solve 1 and 2 equations (point 2 done)

$$x = -3164.55, y = 145.993 \text{ (not approved)}$$

$$x = 3834.55, y = 34.0072 \text{ (approved)}$$

Point 3 and right point

$$y - y_1 = m(x - x_1)$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}, m = \frac{974 - 890}{25 - 1600}, m = \frac{-57}{1575}, m = -0.036$$

$$y - 947 = -0.036(x - 25)$$

$$\sqrt{(x - 25)^2 + (y - 947)^2} = 10506 \text{ pixels}$$

Possible solutions (point 4 is done)

$$x = -10474.2, y = 1324.97 \text{ (not approved)}$$

$$x = 10524, y = 569 \text{ (approved)}$$

## A.3 camera-4 equations

Point 4 and left point

$$(1645, 1020) (105, 915)$$

$$D = \sqrt{(1645 - 105)^2 + (1020 - 915)^2}$$

$$D_1 = 1543 \text{ Pixels } (1850, 130) (1220, 120) D = \sqrt{(1850 - 1220)^2 + (130 - 120)^2}$$

$$D_2 = 630 \text{ Pixels}$$

Point 4 and left point

$$XD_1 = \frac{1543 \times 110}{16.5} = 10286 \text{ pixels}$$

Point 2 and right point

$$XD_2 = \frac{630 \times 110}{16.5} = 4200 \text{ pixels}$$

$$y - y_1 = m(x - x_1)$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}, m = \frac{1020 - 915}{1645 - 105}, m = \frac{105}{1540}, m = 0.068$$

$$y - y_1 = m(x - x_1)$$

$$y - 1020 = 0.068(x - 1645)$$

$$\sqrt{(x - 1645)^2 + (y - 1020)^2} = 10286$$

Possible solutions (point 3 is done)

$$x = -8617.3, y = 322.1 \text{ (approved)}$$

$$x = 11907, y = 1717 \text{ (not approved)}$$

$$y - y_1 = m(x - x_1)$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}, m = \frac{130 - 120}{1850 - 1220}, m = \frac{10}{630}, m = 0.015$$

$$y - y_1 = m(x - x_1)$$

$$y - 130 = 0.015(x - 1850)$$

$$\sqrt{(x - 1850)^2 + (y - 130)^2} = 4200$$

Possible solutions (point 1 is done)

$$x = -2349.53, y = 67 \text{ (not approved)}$$

$$x = 6049, y = 192.99 \text{ (not approved)}$$

# Appendix B

## Code

### B.1 Virtual reality code

# Bibliography

- [1] Y. Chen, Y. Tian, and M. He, “Monocular human pose estimation: A survey of deep learning-based methods,” *Computer Vision and Image Understanding*, vol. 192, p. 102897, 2020.
- [2] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu, “Rmpe: Regional multi-person pose estimation,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2334–2343.
- [3] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. V. Gehler, and B. Schiele, “Deepcut: Joint subset partition and labeling for multi person pose estimation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4929–4937.
- [4] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.
- [5] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, “Smpl: A skinned multi-person linear model,” *ACM transactions on graphics (TOG)*, vol. 34, no. 6, pp. 1–16, 2015.
- [6] X. Zhou, D. Wang, and P. Krähenbühl, “Objects as points,” *arXiv preprint arXiv:1904.07850*, 2019.
- [7] O. Grau, A. Hilton, J. Kilner, G. Miller, T. Sargeant, and J. Starck, “A free-viewpoint video system for visualization of sport scenes,” *SMPTE motion imaging journal*, vol. 116, no. 5-6, pp. 213–219, 2007.
- [8] O. Grau, G. A. Thomas, A. Hilton, J. Kilner, and J. Starck, “A robust free-viewpoint video system for sport scenes,” in *2007 3DTV conference*. IEEE, 2007, pp. 1–4.
- [9] J.-Y. Guillemaut, J. Kilner, and A. Hilton, “Robust graph-cut scene segmentation and reconstruction for free-viewpoint video of complex dynamic scenes,” in *2009 IEEE 12th International Conference on Computer Vision*. IEEE, 2009, pp. 809–816.

- [10] C.-C. Lee, A. Tabatabai, and K. Tashiro, “Free viewpoint video (fvv) survey and future research direction,” *APSIPA Transactions on Signal and Information Processing*, vol. 4, 2015.
- [11] “Free viewpoint video system.” [Online]. Available: <https://global.canon/en/technology/frontier18.html>
- [12] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. ” O’Reilly Media, Inc.”, 2022.
- [13] Z.-H. Zhou, *Machine learning*. Springer Nature, 2021.
- [14] T. O. Ayodele, “Types of machine learning algorithms,” *New advances in machine learning*, vol. 3, pp. 19–48, 2010.
- [15] C. M. Bishop, “Neural networks and their applications,” *Review of scientific instruments*, vol. 65, no. 6, pp. 1803–1832, 1994.
- [16] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [18] S. K. Khanna, “Machine learning v/s deep learning,” *International Research Journal of Engineering and Technology. Volume 6 Issue*, vol. 2, 2019.
- [19] S.-C. Wang, *Artificial Neural Network*. Boston, MA: Springer US, 2003, pp. 81–100. [Online]. Available: [https://doi.org/10.1007/978-1-4615-0377-4\\_5](https://doi.org/10.1007/978-1-4615-0377-4_5)
- [20] H. Kukreja, N. Bharath, C. Siddesh, and S. Kuldeep, “An introduction to artificial neural network,” *Int J Adv Res Innov Ideas Educ*, vol. 1, pp. 27–30, 2016.
- [21] S. Balaban, “Deep learning and face recognition: the state of the art,” *Biometric and surveillance technology for human and activity identification XII*, vol. 9457, pp. 68–75, 2015.
- [22] Y. Le Cun, “Convolutional nets.”
- [23] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015.
- [24] R. M. Schmidt, “Recurrent neural networks (rnns): A gentle introduction and overview,” *arXiv preprint arXiv:1912.05911*, 2019.
- [25] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [26] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, “Generative adversarial networks: An overview,” *IEEE signal processing magazine*, vol. 35, no. 1, pp. 53–65, 2018.



- [27] L. Deng, D. Yu *et al.*, “Deep learning: methods and applications,” *Foundations and trends® in signal processing*, vol. 7, no. 3–4, pp. 197–387, 2014.
- [28] A. D. I. Pytorch, “Pytorch,” 2018.
- [29] N. Ketkar, “Introduction to keras,” in *Deep learning with Python*. Springer, 2017, pp. 97–111.
- [30] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “{TensorFlow}: a system for {Large-Scale} machine learning,” in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.
- [31] N. Sarafianos, B. Boteanu, B. Ionescu, and I. A. Kakadiaris, “3d human pose estimation: A review of the literature and analysis of covariates,” *Computer Vision and Image Understanding*, vol. 152, pp. 1–20, 2016.
- [32] M. Dantone, J. Gall, C. Leistner, and L. Van Gool, “Human pose estimation using body parts dependent joint regressors,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 3041–3048.
- [33] Q. Dang, J. Yin, B. Wang, and W. Zheng, “Deep learning based 2d human pose estimation: A survey,” *Tsinghua Science and Technology*, vol. 24, no. 6, pp. 663–676, 2019.
- [34] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, “Realtime multi-person 2d pose estimation using part affinity fields,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7291–7299.
- [35] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [36] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *European conference on computer vision*. Springer, 2016, pp. 21–37.
- [37] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, “Dssd: Deconvolutional single shot detector,” *arXiv preprint arXiv:1701.06659*, 2017.
- [38] J. Dai, Y. Li, K. He, and J. Sun, “R-fcn: Object detection via region-based fully convolutional networks,” *Advances in neural information processing systems*, vol. 29, 2016.
- [39] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.
- [40] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

- [41] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Region-based convolutional networks for accurate object detection and segmentation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 1, pp. 142–158, 2015.
- [42] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *Advances in neural information processing systems*, vol. 28, 2015.
- [43] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.
- [44] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.
- [45] K. Xie, T. Wang, U. Iqbal, Y. Guo, S. Fidler, and F. Shkurti, “Physics-based human motion estimation and synthesis from videos,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 11 532–11 541.
- [46] M. Kocabas, N. Athanasiou, and M. J. Black, “Vibe: Video inference for human body pose and shape estimation,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 5253–5263.
- [47] K. Rematas, I. Kemelmacher-Shlizerman, B. Curless, and S. M. Seitz, “Soccer on your tabletop,” *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4738–4747, 2018.
- [48] W. Lan, J. Dang, Y. Wang, and S. Wang, “Pedestrian detection based on yolo network model,” in *2018 IEEE international conference on mechatronics and automation (ICMA)*. IEEE, 2018, pp. 1547–1551.
- [49] G. Pavlakos, X. Zhou, K. G. Derpanis, and K. Daniilidis, “Harvesting multiple views for marker-less 3d human pose annotations,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6988–6997.
- [50] S. Liu, D. Liu, G. Srivastava, D. Połap, and M. Woźniak, “Overview and methods of correlation filter algorithms in object tracking,” *Complex & Intelligent Systems*, vol. 7, no. 4, pp. 1895–1917, 2021.