DEEP LEARNING AND TRIGONOMETRIC ADJUSTMENT IN ESTIMATION OF LOWER EXTREMITY ANGLES

Pouria Chalangari

A thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science (Electrical and Computer Engineering)
Concordia University
Montréal, Québec, Canada

August 2020

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Pouria Chalangari**

Entitled: **Deep Learning and Trigonometric Adjustment in Estimation of Lower Extremity Angles**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

_____ Chair
Dr. Jun Cai

_____ Examiner
Dr. Sudhir Mudur (CSE)

_____ Examiner
Dr. Jun Cai

_____ Co-supervisor
Dr. Hassan Rivaz

_____ Co-supervisor
Dr. Thomas Fevens (CSE)

Approved by: _____
Dr. Y.R. Shayan, Chair
Department or Electrical and Computer Engineering

_____ 20 _____ _____
Dr. Mourad Debbabi, Interim Dean
Gina Cody School of Engineering and
Computer Science

Abstract

Deep Learning and Trigonometric Adjustment in Estimation of Lower Extremity Angles

Pouria Chalangari

An Anterior Cruciate Ligament (ACL) injury can cause a severe burden, especially for athletes participating in relatively risky sports. This risk raises a growing incentive for designing injury-prevention programs. For this purpose, for example, the analysis of the drop vertical jump test can provide a useful asset for recognizing those who are more likely to sustain knee injuries. Landing Error Score System (LESS) provides an excellent opportunity to predict the level of vulnerability for each individual who participates in the drop jump test process. Knee flexion angle plays a key role within these test scenarios. Multiple research efforts have been conducted on engaging existing technologies such as the Microsoft Kinect sensor and Motion Capture (MoCap) to investigate the connection between the lower limb angle ranges during jump tests and the injury risk associated with them. Even though these technologies provide sufficient capabilities to researchers and clinicians, they need certain levels of knowledge to enable them to utilize these facilities in an effective manner. Moreover, these systems demand special requirements and setup procedures, which make them limiting. Due to recent advances in the area of Deep Learning, numerous powerful pose estimation algorithms have been developed over the last few years. Having access to relatively reliable and accurate 3D body keypoint information can lead to the successful detection and prevention of injury.

The idea of combining temporal convolutions in video sequences with deep Convolutional Neural Networks (CNNs) offers a substantial opportunity to tackle the challenging task of accurate 3D human pose estimation. Utilizing a fast and accurate 2D pose estimation approach has also enabled us to develop a better and real-time solution for the problem of 3D knee flexion angle estimation. Using the Microsoft Kinect sensor as our ground truth, we analyzed the performance of CNN-based 3D

human pose estimation and our proposed method based on a CNN-based 2D pose estimation method in everyday settings. The qualitative and quantitative results are convincing to give an incentive to pursue further improvements, especially in the task of lower extremity kinematics estimation. In addition to the performance comparison between Kinect and CNN, we have also verified the high-margin of consistency between two Kinect sensors.

Acknowledgments

I would like to use this opportunity and convey my best gratitude to my supervisors, Dr. Thomas Fevens and Dr. Hassan Rivaz. Their constant support and extent of knowledge made it possible for me to continuously pursue the improvements and results of my project. I could not ask for better intellectuals who have maintained mentoring me and have never stopped caring about my progress through rough days during my studies at Concordia University.

I would also like to thank my friends and fellow lab members at IMPACT who have helped me in various possible ways.

My warmest and deepest gratitude goes to my parents, Nader and Zobeideh, who have always got my back from thousands miles away back home. None of these were feasible without their heartwarming and gracious supports. And last but not least, I want to thank my caring wife, Fatemeh, who has always been with me through ups and downs and has constantly reminded me that I can rely on her, no matter what.

The work of this thesis was partially supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grants RGPIN-04136, RGPIN-2020-04612, and RGPIN-04929.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

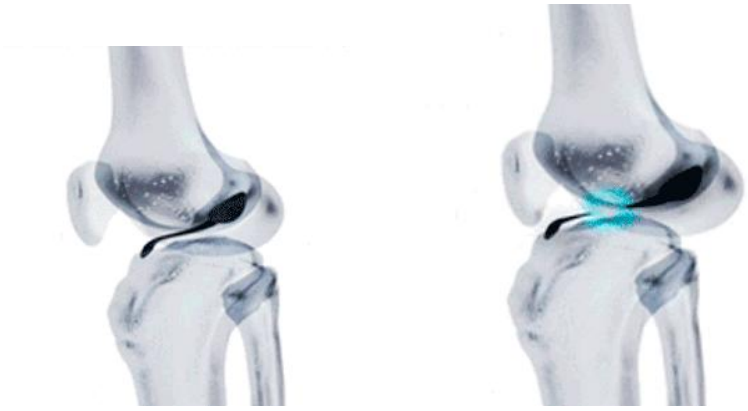
In this chapter, we begin with the clinical background and the motivation behind the thesis (Section 1.1). In Section 1.2, we go through the available solutions to the outlined problem. Then, we briefly explain the objective and organization of the thesis in Sections 1.3 and 1.4, respectively. We conclude the chapter by mentioning the paper published as the outcome of the research conducted in the thesis in Section 1.5.

1.1 Clinical Background

1.1.1 Anterior Cruciate Ligament Injury

The importance of lower extremity kinematic analysis can be identified from the extensive research conducted in sports medicine [1], [2], [3], [4], [5], [6]. The Anterior Cruciate Ligament (ACL) injury is one of the most common lower limb injuries that has attracted many researchers and physicians. Twisting or tearing of ACL may occur due to sudden movements during a game which is categorized as a non-contact injury [7], [8]. Figure 1 depicts a before and after shot of an ACL injury case.

The unbearable pain caused by a torn ACL and financial burden of rehabilitation and surgery is inevitable for the patients and athletes suffering from this injury. This has motivated scientists to design several screening tools and prevention programs to understand the likelihood of the occurrence of such injuries [10], [11], [12], [13], [14]. The population of women vulnerable to this injury is surprisingly multi-fold compared to men [10]. Conducting the drop vertical jump test, Hewett *et al.* [10] studied the role of knee motion and knee loading in ACL injury risk in women athletes. They



(a) Before injury (b) After injury

Figure 1: Demonstration of Anterior Cruciate Ligament injury (from [9]).

used lower body 3-Dimensional (3D) joint angles and joint moments as proof of their hypothesis related to risk factors. Figure 2 illustrates the procedure of conducting the drop vertical jump test. Initially, the subject is standing straight on top of a box with a $31cm$ height. Then a vertical drop off the box on the force plates happens (initial contact), followed by a maximum vertical jump. Several data points are collected during this test.

1.1.2 Landing Error Score System

Research has shown that the Landing Error Scoring System (LESS) is effectively able to recognize the risk of ACL injury in elite-youth soccer athletes [15], [16], [11]. Through analyzing the Jump-Landing task, this screening tool identifies the athletes at higher risk of sustaining ACL injury by assigning a risk score. The higher the score, the more probable that the subject may sustain an ACL injury. Each participant is required to conduct three trials of the test to minimize the possibility of considerable discrepancy in the results. Most of the items involved in the LESS score are qualitative, and the screening needs to be done offline, meaning an operator has to monitor the recordings and decide on the outcome of each item in the list. One of the principal quantitative items in the LESS score is the maximum knee flexion angle at initial contact. To have a visual clue, Figure 3 depicts the knee flexion angle in a sagittal plane view. The maximum flexion angle refers to the position where the hip joint has the minimum vertical distance to the ground.

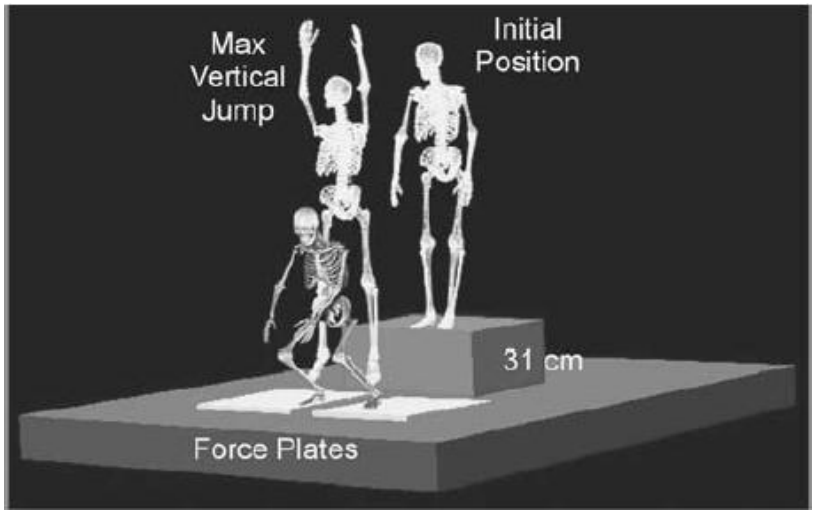


Figure 2: An illustration of drop vertical jump test (from [10]).

In a nutshell, we need a way to measure the lower extremity angle to complete the screening task. Throughout the next section, we try to address this concern.

1.2 Problem Solutions

1.2.1 Gait Analysis

In order to extract 3D kinematics information, gait analysis laboratories could be used. Tao *et al.* [18] discussed the use of wearable sensors in gait analysis labs and their applications in health-related problems, for instance, rehabilitation. These labs provide their services at prohibitive cost, and the process involves the attachment of markers, which can take many hours to complete. Traditional Motion Capture (MoCap) systems are commercially available, which offer accurate and reliable 3D skeletal tracking. Apart from the high cost, it requires a dedicated lab environment and complicated camera setup and synchronization. To avoid this complexity and to have more reachable solutions, we shall look for other options.

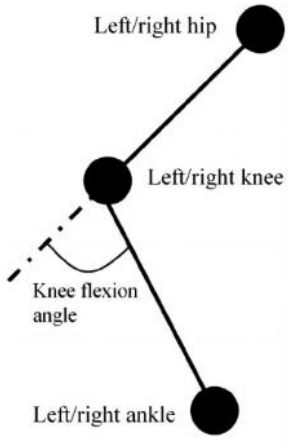


Figure 3: Knee flexion angle (from [17]).

1.2.2 Microsoft Kinect®

The second version of the Microsoft Kinect sensor was released in 2014 for the Xbox One console. It is an affordable and portable device used for many Computer Vision applications, coming with an official Software Development Kit (SDK) provided by Microsoft. Figure 4 presents a Microsoft Kinect Version 2.

Eltoukhy *et al.* [19] validated the Kinect's performance in terms of sagittal plane kinematics by comparing it to a motion analysis system. Their study suggested a rather acceptable consistency between the two technologies. Others [20], [21] also assessed the reliability of the Kinect in clinical and sports medicine applications and confirmed its effectiveness by a negligible deviation from measurements using a Mo-Cap system.

The ultimate goal of this thesis is to offer a solution that is independent of an extra hardware setting and only utilizes one simple RGB camera. Ideally, this camera is our cellphone, which does not require additional utilities, and the tests can be performed in real-time.

1.2.3 Artificial Neural Networks

Artificial neural networks (ANNs) are systems motivated by the distributed, massively parallel computation in the brain that enables it to be so successful at complex

Figure 4: A Microsoft Kinect Version 2

control and recognition/classification tasks [22]. These networks are motivated by the functionality of the biological neural units, also known as neurons. The combination of these neurons in a multi-layered structure with specific weights creates an ANNs. Their main purpose is to learn from experiences (training data) and apply this new knowledge to a set of unseen data. The capability of modeling non-linear tasks enables these networks to participate in solving many critical problems in numerous applications. Xiong *et al.* [23] used Artificial Neural Networks to predict the human lower extremity joint movement using a limited amount of inputs for training in an unconstrained manner.

1.2.4 Convolutional Neural Networks

Deep learning discovers intricate structure in large data sets using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer [24]. A Convolutional Neural Network (CNN) is a Deep ANN that has shown superior performance in learning from images, audio, etc., inputs. They reduce the number of connections and parameters inside the network and apply a kernel convolution to the input of each layer. CNNs are widely used in Computer Vision problems, e.g., 3D human pose estimation, which will be discussed in detail through the rest of the chapters.

1.3 Contributions of the Thesis

In this thesis, we present new approaches to estimate the sagittal-plane knee flexion angles using CNN-based human pose estimation methods. We chose the work of Cao *et al.* [25] (aka, OpenPose) and Pavllo *et al.* [26] (aka, VideoPose3D) to estimate the joint positions and then we carry out the angle calculations using these predictions since it has shown a relatively promising performance on in-the-wild data. We propose a novel approach of calculating a 3D quantity (knee flexion angle) by combining a CNN-based 2D pose estimation method (OpenPose) with a trigonometric adjustment approach. For the purpose of comparison, we also used the Kinect frames as the input sequence to CNN and Kinect 3D keypoints as the quantitative analysis reference. Although these methods do not offer the ultimate and the most accurate solution, they open doors to more robust opportunities. More details are coming through the next chapters.

1.4 Organization of the Thesis

The thesis is organized in the following order: In Chapter 2, we present the technical details about Microsoft Kinect. We continue the chapter by reviewing the literature in human pose estimation and discussing different 2D/3D pose estimation approaches. Then we talk about the details of OpenPose, Detectron, and VideoPose3D. In Chapter 3, we give an explanation of our data collections using Microsoft Kinect and then, we introduce our CNN-based approach for knee flexion angle estimation, which we name DeepLEAD. Chapter 4 is where we evaluate the methods we talked about in previous chapters, both qualitatively and numerically, by conducting several experiments. Finally, we wrap up the thesis with conclusions and future work in Chapter 5.

1.5 Publication

Part of the research in this thesis is accepted as a full contributed paper named "3D Human Knee Flexion Angle Estimation using Deep Convolutional Neural Networks" to be presented at IEEE EMBC 2020.

Chapter 2

Background and Literature Review

The outline of this chapter begins with a brief introduction to Microsoft Kinect and some of its technical details in Section 2.1 followed by a quick glance at 2D pose estimation in Section 2.2. In Section 2.3, we talk about OpenPose, some of its key features that allow us to choose it as our 2D pose estimation method, along with its architecture details. We continue the background with introducing Detectron as the 2D keypoint detection package used by VideoPose3D in Section 2.4. Section 2.5 provides a literature review of 3D pose estimation methods. In Section 2.6, we covered the basic building blocks of a Recurrent Neural Networks. Section 2.7 explains the architecture used by VideoPose3D, the concept of temporal convolution, the dataset preparation and the training procedure of VideoPose3D on Human3.6m dataset. We conclude the chapter by a summary in Section 2.8.

2.1 Microsoft Kinect®: How Does It Work?

Microsoft Kinect can primarily be used as a motion detector in many applications, such as physical therapy, Virtual Reality (VR) and gaming, Robotics, 3D reconstruction, and many more. It contains two separate cameras: RGB and infrared (IR). Using these two cameras, Kinect is able to dump color, IR, and depth images to a computer using Microsoft Kinect SDK. Kinect V2, compared to Kinect V1, offers improved performance and quality, in terms of hardware specifications and sensing technologies. Figure 5 highlights some of these enhancements. Due to an extension

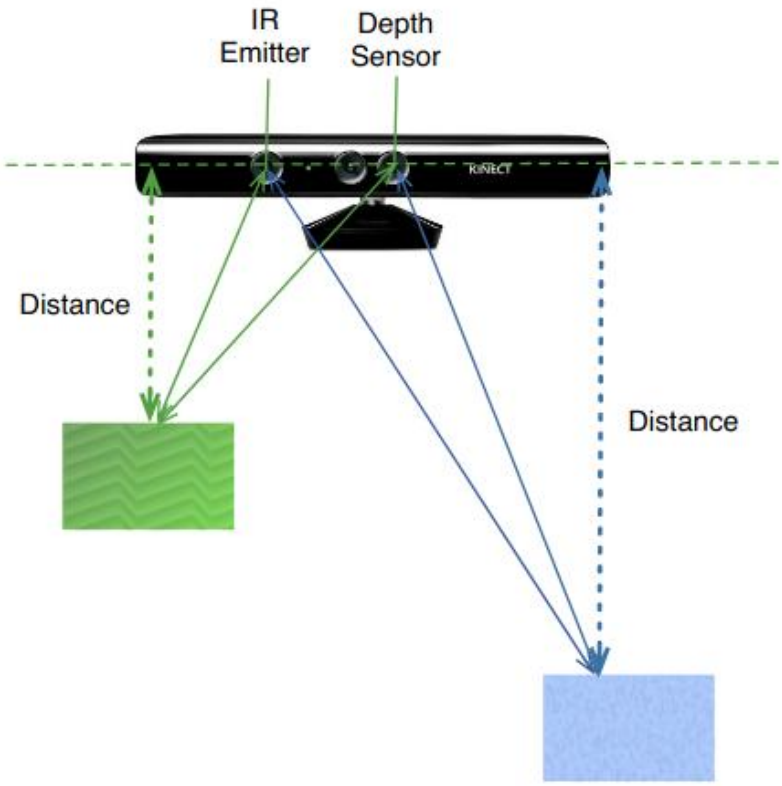
Feature	Kinect v1	Kinect v2
Depth Sensing Technology	Triangulation with structured light	Time of flight
Color Image Resolution	640x480 30fps 1280x960 12fps	1920x1080 30fps (12fps low light)
IR Image Resolution	640x480 30fps	512x424 30fps
Depth Sensing Resolution	640x480 30fps 320x240 30fps 80x60 30fps	512x424 30fps
Field of View	43° vertical 57° horizontal	> 43° vertical 70° horizontal [60]
Depth Sensing Range	0.4m - 3m (near mode) 0.8m - 4m (default mode)	0.5m - 4.5m Up to 8m without skeletonization
Skeleton Tracking (with full skeleton)	Up to 2 subjects 20 joints per skeleton	Up to 6 subjects 25 joints per skeleton
Built-in Gestures	None	Hand state (open, close, lasso) Hand pointer controls; lean
Unity Support	Third party	Yes
Face APIs	Basic	Extended massively
Runtime Design	Can run multiple Kinect sensors per computer; One app per Kinect	At most one Kinect per computer; Multiple apps share same Kinect
Windows Store	Cannot publish to	Yes

Figure 5: A comparison of main features between Kinect V1 and Kinect V2 (from [27]).

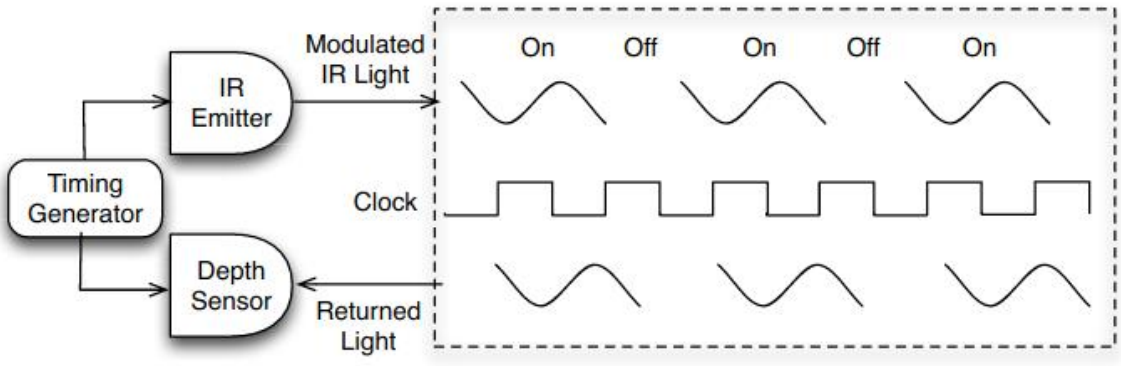
of Field of View (FOV), both in horizontal and vertical directions, along with utilizing a higher quality camera has enabled Kinect V2 to achieve better resolution (1920×1080) in a color image. Skeleton tracking capability with full-body skeleton, as one of Kinect's major features, tripled in terms of maximum number of subjects it can detect simultaneously. Kinect V2, in spite of its companion, offers hand pointers and controls as well.

The technology that Kinect V1 employs to capture depth information is fundamentally different than Kinect V2's Time-of-Flight (ToF). A structured light technique that uses an IR emitter and the depth sensor enables Kinect V1 to calculate a pixel-wise depth map. Figure 6-(a) depicts the details of the triangulation approach in Kinect V1. Using this method, the distance of an object to the device surface plane can be estimated.

Kinect V2 uses an IR laser diode that sends modulated signals that are emitted in a controlled manner with the depth sensor. The depth of each pixel can be calculated based on the phase shift between the emitted light and the reflected light [27]. As



(a) Structured light triangulation in Kinect V1



(b) Time-of-Flight in Kinect V2

Figure 6: A comparison of depth sensing technologies in (a) Kinect V1 versus (b) Kinect V2 (from [27]).

demonstrated in Figure 6-(b), a timing generator is responsible for synchronizing the IR emitter and the depth sensor. For the duration that the IR light is on, the depth sensor receives both the IR light combined with the ambient light, while in the period that the IR light is off, the received signal at the depth sensor only contains the ambient light. By subtracting these two, the remained IR light can be used to calculate the depth accurately.

Considering these technology changes from one generation to another, the depth sensing range has shown an approximately 50% improvement in favor of Kinect V2. Moreover, the quality of the depth image and IR image has experienced a considerable boost.

2.2 2D Pose Estimation

The process of extracting and detecting pose information from a single image is one of the most challenging problems in Computer Vision. When this effort leads to a successful outcome, it can be used in countless other application, from markerless motion capture to autonomous driving systems, to name a few. This has attracted many researchers and engineers to work on this task and come up with multiple solutions.

Toshev *et al.* [28] used Deep Neural Networks to regress the body joints from images. As another example of pose estimation methods, a CNN-based hourglass design was proposed by Newell *et al.* [29]. CNNs have shown great potential in extracting numerous patterns in the images and obviously, can achieve superior performances when it comes to images. The person's orientation, the arrangement of their limbs, and the relationships of adjacent joints are among the many cues that are best recognized at different scales in the image [29]. Convolutional Pose Machines (CPMs) [30] utilize a multi-stage architecture to continuously iterate the process of refining the confidence maps and offers the position of important pixels in the image (joints).

2.3 OpenPose

OpenPose [25] is a real-time multi-person 2D pose estimation method. OpenPose is an open-source project that provides the code for Ubuntu, Microsoft Windows, macOS

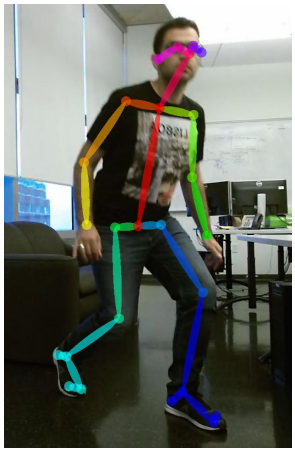


Figure 7: Sample output frame from OpenPose

and Nvidia TX2 Operating Systems. It accepts many types of inputs, e.g., images, videos, webcam, etc., which can be added to the target project. The output can be in multiple forms as well. From images/videos overlaid by the 2D skeleton (Figure 7) to well-known structured data-serialization formats such as *json*, *YAML*, and *XML*, to name a few. For a better performance, it is better to install the GPU-based version of the program, rather than the CPU-based one. OpenPose requires CUDA and Caffe, a deep learning framework made with expression, speed, and modularity in mind [31], for the GPU-based installation. We used a Dell AlienWare 13-inch laptop with 16GB of RAM and a GeForce GTX 1060 GPU with 6GB memory, for all of our model trainings, data collection and simulations.

Apart from the very well-documented repository on Github, it also provides easy-to-use APIs for C++ and Python. The combination of all the above-mentioned features and the level of accuracy and speed compared to its rivals have convinced us to choose OpenPose for our approach.

2.3.1 Network Architecture

In order for the algorithm to detect all the body parts and joints of all the subjects in the input image, OpenPose introduces Part Affinity Fields (PAFs) for each limb and confidence maps for each joint. PAFs are a set of 2D vector fields that encode the location and orientation of limbs over the image domain [25]. The importance of PAF refinement dominates the body part detection while the goal is to maximize the accuracy. By eliminating the body part refinement and increasing the network depth,

both accuracy and speed improve significantly. Figure 8 demonstrates a sample image and its corresponding PAFs and confidence maps.

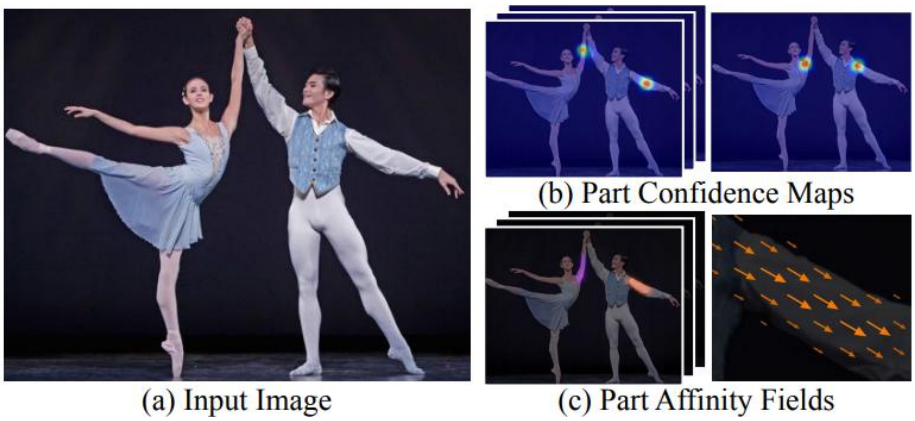


Figure 8: Demonstration of PAFs and confidence maps of an input image (from [25]).

To better understand the network architecture of OpenPose, Figure 9 provides a visualization of different components in that structure. The input to the network, $\mathbf{F}$, are the feature maps extracted by a VGG-19 [32] CNN with 10-layers.

The refinement process happens over multiple stages, t, motivated by the approach in CPM [30]. T_P and T_C refer to the total number of PAF and confidence map stages, respectively. The process of PAF refinement in the blue box needs to reach the T_P stages, before the L^t can be passed to the confidence map stage, in the beige box. The pink rectangles represent a convolution block with 3×3 kernels. At the end of the process, the network produces $\mathbf{L}$ and $\mathbf{S}$ sets which contain the PAF of each body part and keypoint confidence maps of each joint, respectively. By combining these two sets, we achieve the joint keypoints.

2.4 Detectron

In [26], the task of 3D pose estimation is followed by an initial 2D keypoint detection that was done by Detectron [33], an off-the-shelf object detection platform that includes multiple object detection algorithms, e.g., Mask R-CNN [34].

Regardless of the type of the input to the Detectron, i.e., individual frames or videos, it produces the same results in terms of 2D keypoint detection. It exports the 2D keypoints of each frame into an array, by which we can create a custom dataset as a reference for 3D keypoint detection.

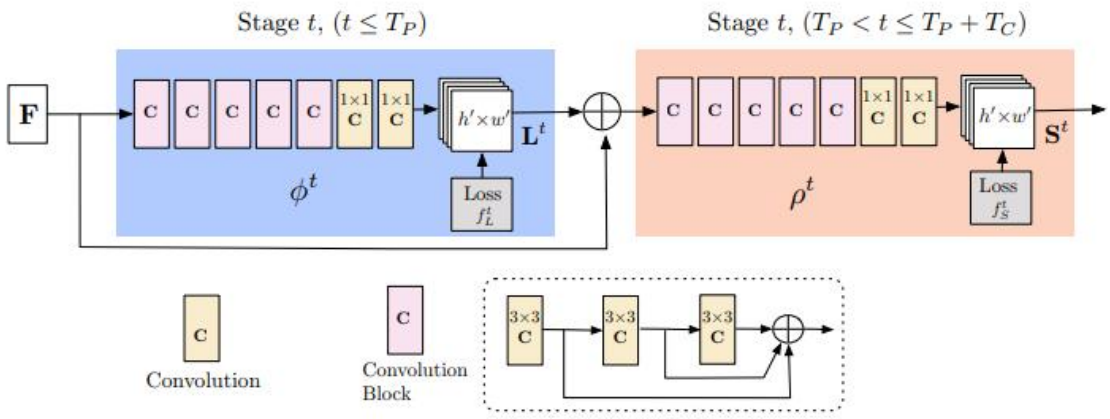


Figure 9: Multi-stage CNN-based network architecture of OpenPose (from [25]).

Detectron2 is a ground-up rewrite of the previous version, Detectron, and it originates from Mask R-CNN benchmark [35]. It adds new features, e.g., rotated bounding boxes, etc., to Detectron. As we do not need those added features and could not spot meaningful difference in our results after using Detectron2, we decided to keep using Detectron as the 2D keypoint extractor for VideoPose3D (2.7).

2.5 3D Pose Estimation

In order to have a sense of depth in a single frame, for every pixel, at least two frame from different angles are required. This is one of the fundamental aspects in Computer Vision. However, what if we only have one frame and we want to extract the depth information of each pixel using that frame? For the case of 3D human pose estimation, we are only interested in the 3D coordinates of each body joint. The 3D positions of these joints can be regressed either in the camera space or the world coordinate space. Figure 10 shows a sample frame and its corresponding 3D reconstructed skeleton in camera space. The 3D positions, in this case, are with respect to a reference joint.

With the advances of Machine Learning and Deep Learning algorithms, 3D human pose estimation methods have been extensively developed for dozens of applications. Due to a limited amount of training data, these tasks are extremely challenging. These methods can be further sub-categorized based on different approaches and most of them utilize the abundant potential of Convolutional Neural Networks as their core.

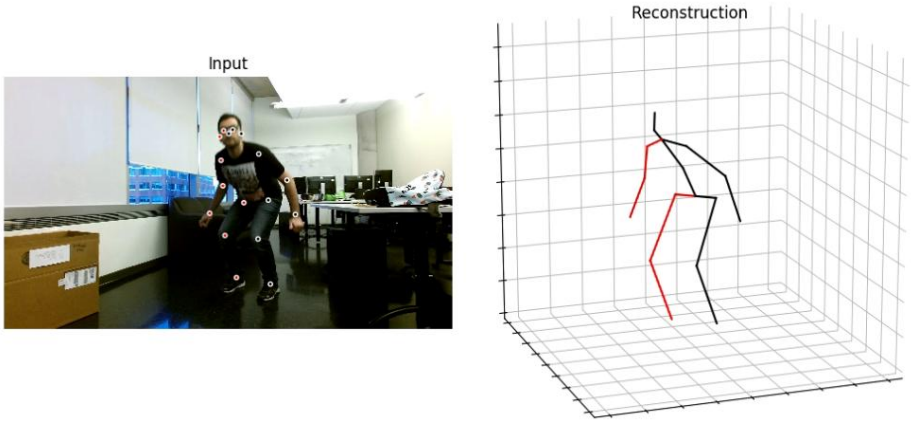


Figure 10: Sample frame overlaid by 2D keypoints from Detectron and its corresponding 3D reconstruction in camera space

The input to these networks could either be a monocular image or a video sequence and in terms of their method, they could either implement an end-to-end input-to-3D joints [36], [37], or lifting 2D joint detection to 3D predictions [26], [38], [39], [40], [41], [42]. Here we provide a quick glance of these works:

- Mehta *et al.* [36] developed *VNECT* as a real-time 3D pose estimation method that directly extracts the joint locations using CNN. They use root relative joint positions and bone lengths as a control parameter during training and inference. By the help of 2D heatmaps for each joint, they predict the 3D joint positions. We have tested this method on our in-the-wild data and could not achieve reliable and consistent results. Although it is real-time, it does not offer jitter-free and precise estimations.

- Li *et al.* [37], without explicitly including any kind of constraints about different body parts, found a meaningful dependency between them while training their CNN network. They use two strategies for the training phase:

  1. training the network on the task of regression, that was pre-trained on the task of body part detection.
  2. training the network jointly for the task of regression and body part detection using a multi-task framework.

- Zhou *et al.* [38] proposed a weakly-supervised transfer learning method that uses mixed 2D and 3D labels in a unified Deep Neural Network that presents

two-stage cascaded structure [38]. They adopted stacked hourglass network [29] as part of their 2D pose estimation module. To mitigate the lack of abundant 3D ground truth data for training, they proposed a weakly-supervised algorithm by incorporating a geometric constraint on 2D labeled training data.

- Martinez *et al.* [39] used an off-the-shelf 2D pose estimation and tried to lift it to 3D predictions by using a simple multi-layer Deep Neural Network. Their network architecture consist of a deep linear block followed by batch normalization, Rectified Linear Unit (ReLU), and a 50% dropout block and this block is concatenated with its replica to form a bigger block. In this block, there is a residual connection that inter-connects the input to the first linear block to the output of this big block. This whole structure is repeated again to finalize the network design. Due to the simple connections and building blocks, the authors proved that their work can be used for real-time applications. But the dependency to the off-the-shelf 2D detector introduces a soft spot to their implementation that can be affected by the probable failures of that 2D detector.

- Tome *et al.* [40] proposed a complementary approach for 2D and 3D poses. Their multi-stage method used 2D belief-maps regressed at every stage for each landmarks (keypoints) and a 3D-to-2D projection to refine the 2D predictions at every stage. After each stage, they showed that the error rate was decreasing. Their solution to the problem of insufficient 3D ground truth data was to ignore some limiting criteria in the process of training. For instance, rather than imposing a length constraint on each limb, they decided to normalize these lengths and only considered the sum of squared normalized lengths to be equal to 1.

- Chen *et al.* [41] offered to use a separated approach for 2D and 3D poses, meaning they used an intermediate 2D detector to predict the 3D keypoints. They mentioned accomplish this 3D detection by exemplar matching. To better accomplish the task of matching exemplars to 2D estimations, instead of introducing an optimization problem, they use a simple warping method which makes the whole process way easier.

- Pavlakos *et al.* [42] decided to avoid approaching the pose estimation problem

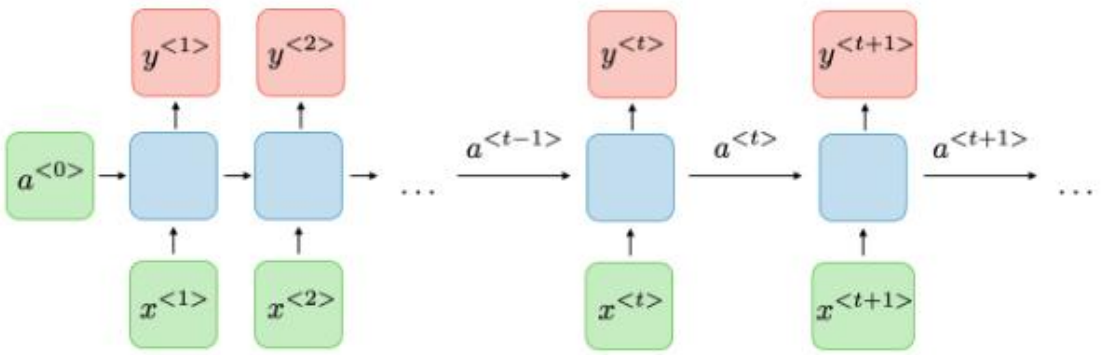


Figure 11: Architecture of a traditional RNN (from [43]).

by a 2D-to-3D prediction paradigm. They discretize the area around the subject to voxels (volumetric pixels) and perform the task of training the network for predicting a likelihood for each joint, in each voxel. In other words, they proposed a volumetric method that performs end-to-end training and prediction. They could increase the accuracy by an acceptable margin but since the probabilistic approach needs to consider all of the voxels for every single joint, the processing time and computation overhead is inevitable.

Not all the methods above in the bullet points offer a free version of their inference/training code or trained models. For the ones that do, we tested them on our in-the-wild data. The reason we decided not to use them relates closely to their poor and unacceptable performance. In Section 2.7, we specifically go through the selected method named VideoPose3D from Facebook AI Research (FAIR).

2.6 Recurrent Neural Networks

Recurrent Neural Networks (RNNs), by their nature, are used to model sequential data. It means that there is some kind of temporal correlation between consecutive data samples. Speech, text, and video (sequence of frames) are examples of sequential data. End-to-end training methods such as connectionist temporal classification make it possible to train RNNs for sequence labelling problems where the input-output alignment is unknown [44]. Figure 11 depicts the traditional structure for a RNN.

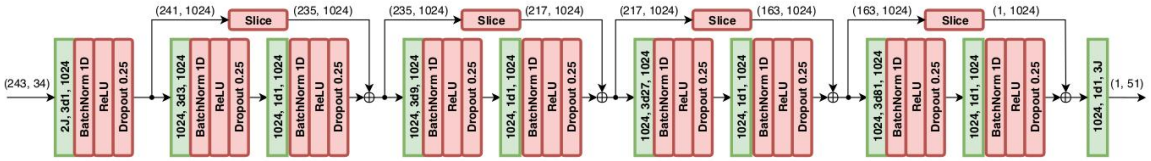


Figure 12: Fully convolutional 3D pose estimation architecture (from [26]).

The sequence of data is fed to the network, one sample at a time. The current state is a function of the previous state and the input in the same time frame.

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \tag{1}$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y) \tag{2}$$

Equations 1 and 2 define the current state and the current output of the network in which g_1 and g_2 are the activation functions of current state and current output and W_{aa}, W_{ax}, b_a, and b_y are the weights and bias values for the current state and the current output, respectively.

Since the parameters of the network are shared throughout the time and the network is able to remember previous inputs (to some extent- not very long time though), the complexity of the model reduces and the network can handle any input length [43].

2.7 VideoPose3D

2.7.1 Architecture

The basic network architecture (model and its weights) for the the 2D detector that we used in our simulations was adopted from Feature Pyramid Networks (FPN) [45], with ResNet-101 backbone [46] trained on Common Objects in Context (COCO) dataset [47]. Although employing RNNs alleviates the ambiguity imposed by the two-stage pose estimation [48], VideoPose3D proves the superiority of the performance of CNNs over RNNs in handling those difficulties. Plus, CNNs are capable of handling multi-frame parallel processing, which cannot happen with RNNs.

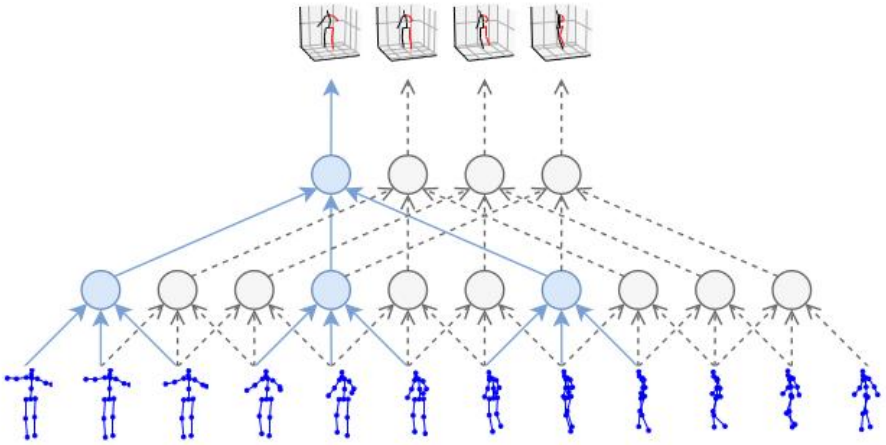


Figure 13: Temporal convolution implementation scheme which maps dilated 2D keypoints to 3D poses (from [26]).

The combination of residual links (Figure 12) and the dilated temporal convolution (Figure 13) successfully maintains the long-term information flow which is required for effective sequence processing in this specific task.

Figure 12 indicates the default number of receptive fields, which is 243 frames. The green rectangle at the beginning of the first convolution layer denotes $2 \times J$ input channels, where J refers to the number of joints. Since our 2D detection network is trained on COCO, we have $J = 17$. In $3d1$, etc., as seen in Figure 12, the kernel size is 3 and $d1$ means the convolutions are calculated with dilation equal to 1. And finally, 1024 denotes the number of channels in the output. This convolution layer is followed by 1D batch normalization, a Rectified Linear Unit (ReLU) and a 25% dropout layer. This pattern, with some minor variations, is replicated throughout the architecture. The tensor dimensions are specified in parentheses. For instance, the input tensor is of shape 243 (frames) by 34. To comply with the subsequent tensors, the residual tensors are sliced before being used in the pipeline.

2.7.2 Temporal Convolutions

In Figure 13, the blue skeletons at the bottom depict the 2D keypoints. The dilated combination of these keypoints are mapped to 3D poses. Since by default, the input to the system is a video, the temporal characteristic of this video is being used properly by this convolution scheme.

2.7.3 Training on Human3.6m Dataset

Human3.6M [49], [50] is one of the most famous datasets in the literature of the human pose estimation. The models (including 6 males and 5 females) are professional actors. Usually for training a model, 7 subjects (4 males and 3 females) are used and the rest of 4 subjects will be considered for the testing phase. Each subject is recorded during 17 different scenarios, e.g., walking, sitting on chair, etc. The whole scene is covered by 4 calibrated and synchronized digital cameras with known intrinsic parameters. Recording the video at 50 Hz produces more than 3.6 Million of frames which provides an extraordinary amount of data for data-hungry tasks like pose estimation. Although each frame is associated with 24 keypoints in this dataset, since the 2D detector is trained on COCO (with 17 joints), the camera-space 3D joints in the output of the architecture contains a flattened 3×17 tensor.

In order to make the dataset ready to use, we applied preprocessing and data re-ordering. For each frame in the sequence, we take the bounding box (determined by the 2D keypoints found by Detectron) with the highest probability. In case of a missing bounding box for a frame, an interpolation of keypoints from adjacent frames will be considered. Data re-ordering only applies to frame inputs, not videos, which is basically stacking all of the keypoints into a single *.npz* file.

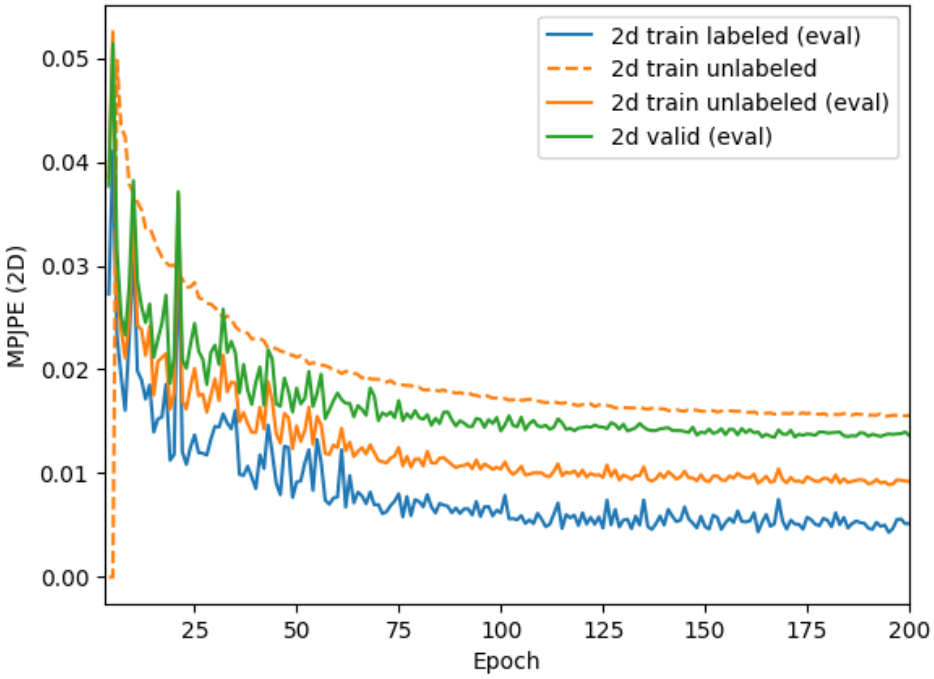
We trained the 3D network on this dataset with a fine-tuned CNN model as the 2D detector for 200 epochs. Figure 14 depicts Mean Per Joint Position Error (MPJPE) as one of the most commonly used evaluation metric in human pose estimation. Per Joint Position Error is simply the Euclidean distance between the ground truth and the network prediction.

In Figure 14-(a), the MPJPE is reported for the 2D predictions for each epoch in meters. As we expected, the error for the labeled training data (blue) is the lowest one since this is easier than the unlabeled training task. In Figure 14-(b), the MPJPE is reported for the 3D predictions for each epoch in meters. The final value of MPJPE always falls below $8cm$ during training or validation. It is vital to remember that having the best performance and lowest possible error almost always comes at a price. This compensation, such as speed, might not be affordable in some applications that require real-time solutions. Therefore, as long as the validation error is within the scope of the error tolerance of the problem in-hand, the algorithm remains an option to be considered.

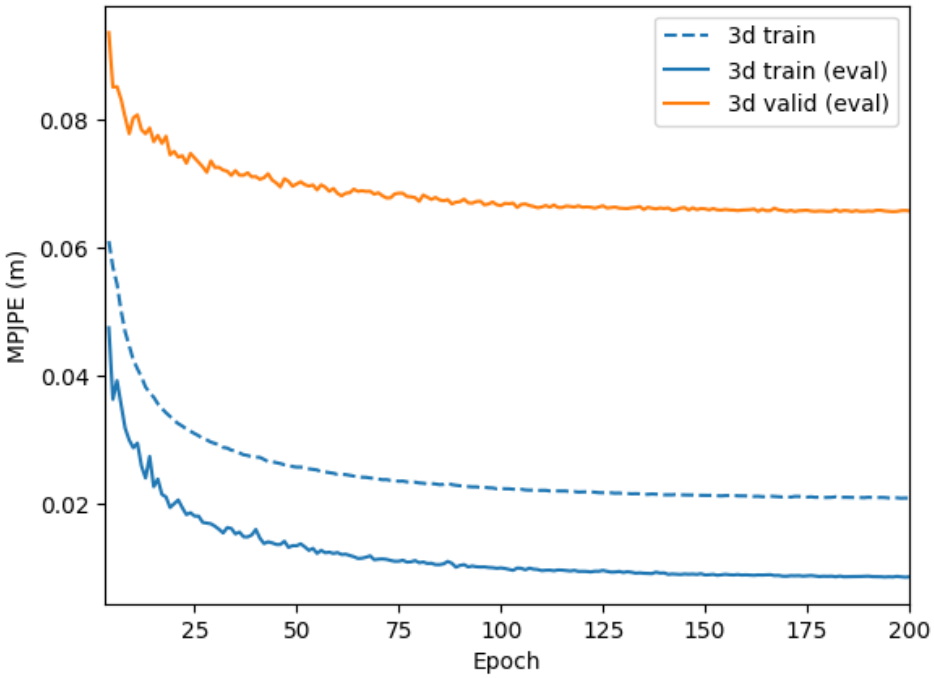
By looking at both figures and paying close attention to the validation error levels, it is notable that the final MPJPE for the 3D predictions is almost 4 times the MPJPE for the 2D predictions. This is, again, compatible with theoretical concepts. The task of the 3D pose estimation is always harder that the 2D pose estimation, therefore it is more error-prone.

2.8 Summary

In this chapter, we covered a wide range of background materials, from details about Microsoft Kinect to 2D/3D pose estimation methods. We reviewed multiple 3D human pose estimation methods in which the 3D keypoints can either be extracted directly from an input image or by lifting an initial 2D detection to 3D estimation. Although these methods offer rather acceptable performances but almost all of them not only require power-intensive hardware with considerable amount of RAM and GPU but they compromise the running time and cannot be used for real-time applications (like lower extremity angles). On the other hand, 2D pose estimation algorithms offer much simpler solutions in terms of CNN network architecture and run-time delays. Using one of these methods as the base to our proposed algorithm, we want to mitigate the need for sophisticated implementations and methods while avoiding losing significant performance and accuracy. Through the coming chapter, we try to address this issue by presenting our new approach and conducting several experiments to back its empirical feasibility.



(a) 2D MPJPE



(b) 3D MPJPE

Figure 14: MPJPE for VideoPose3D training and validation

Chapter 3

Methods and Algorithms

In this chapter, we provide some details about the data collection procedure using Microsoft Kinect V2 (Section 3.1). Then, we introduce the methods we want to evaluate in the next chapter (Section 3.2). In Section 3.3, we explain the details of our proposed method using OpenPose as the basic component combined with trigonometric adjustments. We summarize DeepLEAD in an algorithm representation in Section 3.4. Finally, we conclude the chapter and summarize the findings in Section 3.5.

3.1 Data Collection using Microsoft Kinect

In order to evaluate the methods and algorithms on our own in-the-wild data, we need to have a unified method of data collection to create unbiased results. Using a Microsoft Kinect V2 sensor, we exported RGB and 3D skeleton keypoints with a single person standing in the middle of the scene in an indoor configuration. The toolbox can be found in [51]. The RGB frames from the Kinect were affected by radial distortion imposed by the Kinect RGB camera's intrinsic characteristics. This distortion somehow affects the performance of the CNN on in-the-wild data (also mentioned by Pavllo *et al.* [26] in paper's Github repository) since it should infer the depth information from these distorted frames. Kim *et al.* [52] designed an experiment to estimate the Kinect's intrinsic parameters. As these intrinsic parameters are completely dependent on the physical structure of the lens, the parameters slightly change from one camera to another. In order to investigate this, we used two sets

of parameters from two different Kinect and the effect on the 3D estimations were negligible. As we mentioned earlier, the CNN was trained on Human3.6M dataset which contains 50 frames per second (fps) videos. To avoid performance degradation, these undistorted frames from the Kinect were converted to a sequence data with the same frame rate.

It is worth mentioning that the Kinect toolbox that we used outputs each joint's information in different coordinate systems. In our case, the origin is considered to be the IR laser diode. Considering Figure 4, x and y unit vectors are in parallel with the sensor's plane and z increases in the direction the Kinect is faced. All of these values are exported in meters. Generally, since our goal is to estimate a 3D angle that is calculated using 3D points relative to the system's origin, it does not matter which coordinate system we consider as the reference.

During the data collection process, the subject performs kneeling and standing up several times. The amount of displacements towards or away from camera (z direction) is always maintained to be negligible compared to the subject's distance from the sensor (almost 15 times smaller). This enables us to impose a fundamental assumption we make in Section 3.3 on the length of lower limbs (femur and tibia). To clarify this, suppose the subject is standing in a reference location. The closer she/he gets to the camera, the larger become the length of lower limb vectors, and vice versa. Therefore, this controlled movement needs to be considered during the data collection.

3.2 Methods

In previous chapter, we introduced two CNN-based pose estimation methods. OpenPose offers a highly-reliable 2D pose estimation with great documentation and code support. In Section 3.3, we come up with a novel algorithm that utilizes OpenPose as a companion to predict the 3D angles. VideoPose3D, on the other hand, provides a 3D keypoint detection that can be used to calculate the 3D angle that should be estimated.

In brief, we evaluate two methods in Chapter 4:

1. VideoPose3D
2. Using OpenPose and trigonometric adjustments (Section 3.3)

3.3 Our Approach

In this section, our goal is to overlay the steps we took to calculate 3D angles using OpenPose and trigonometric adjustments. We have introduced OpenPose in previous chapter. OpenPose only provides us the value of 2D coordinates. Figure 15 shows a sample sketch of a knee flexion angle. To make the explanation process easy, we index the 3D hip, knee, and the ankle joints as h, k, and a, respectively.

In order to calculate the angle between two 3D space vectors, we need to use Equation 3, where $\boldsymbol{f}$ (Equations 4), $\boldsymbol{t}$ (Equation 5), l_f (Equation 6) and l_t (Equation 7) represent the 3D femur vector, 3D tibia vector, femur vector's length and tibia vector's length, respectively.

$$\begin{aligned}\theta = & \arccos(\frac{\boldsymbol{f}\cdot\boldsymbol{t}}{l_f l_t}) \\ = & \arccos(\frac{(x_k - x_h)(x_a - x_k) + (y_k - y_h)(y_a - y_k) + (z_k - z_h)(z_a - z_k)}{l_f l_t})\end{aligned} \tag{3}$$

$$\boldsymbol{f} = [(x_k - x_h), (y_k - y_h), (z_k - z_h)] \tag{4}$$

$$\boldsymbol{t} = [(x_a - x_k), (y_a - y_k), (z_a - z_k)] \tag{5}$$

$$l_f = \sqrt{(x_k - x_h)^2 + (y_k - y_h)^2 + (z_k - z_h)^2} \tag{6}$$

$$l_t = \sqrt{(x_a - x_k)^2 + (y_a - y_k)^2 + (z_a - z_k)^2} \tag{7}$$

At the beginning of the calculations using OpenPose, we suppose that the target is standing straight in front of the camera (which is the case in a drop vertical jump test). With this assumption, all of the joints have the same value of z and Equations 6 and 7 reduce to Equations 8 and 9, respectively.

$$l_f = \sqrt{(x_k - x_h)^2 + (y_k - y_h)^2} \tag{8}$$

$$l_t = \sqrt{(x_a - x_k)^2 + (y_a - y_k)^2} \tag{9}$$

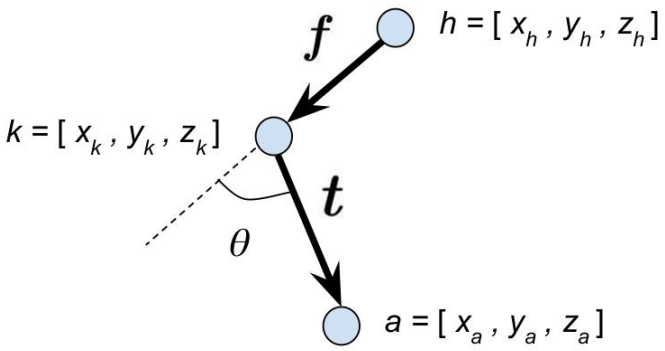


Figure 15: Knee flexion angle and the corresponding point and vector definitions from sagittal plane view. Figure corresponds to knee joint shown in Figure 3.

Now, we have l_f and l_t and we can use these values for the next frames. In the case of our drop vertical jump test, the subject gets closer to the camera by a very small amount (with respect to the distance to the camera, which is at least 3 meters in our experiments) and assuming the length of femur and tibia to be the same for the rest of the frames does not affect the results. This assumption is backed by several tests we conducted.

For the next frame, the only unknown quantities in Equation 3 are the differences in the depth information. Therefore, we rearrange Equations 6 and 7 and calculate these depth differences. Note that, for every frame, we have the values of x and y.

$$(z_k - z_h) = \sqrt{l_f^2 - (x_k - x_h)^2 - (y_k - y_h)^2} \tag{10}$$

$$(z_a - z_k) = \sqrt{l_t^2 - (x_a - x_k)^2 - (y_a - y_k)^2} \tag{11}$$

Both values of depth differences in Equations 10 and 11 are considered to be positive. These quantities, despite the depth values in Kinect which are estimated from the sensor to the subject, are in camera space. It means that we only care about the absolute value of the depth difference and use them in Equation 3 to estimate the angle. To verify this, we considered one of the values in Equations 10 and 11 to be negative. This change affects the results of the experiments in an unacceptable and wrong manner, meaning that these depth differences are always positive.

And finally, we can now use Equation 3 and calculate the knee flexion angle for each leg. Once again, we should mention that the initial estimated length for femur

Algorithm 1: DeepLEAD

Input: A sequence of frames or a video with total number of N frames
Output: $N \times 2$ array containing knee flexion angles of both legs
N json files containing 2D keypoints $\leftarrow$ *OpenPose*
$angles \leftarrow zeroes(N, 2)$
$frame \leftarrow 0$
for *each json file* **do**
    $OpenPoseData(frame, :, :) \leftarrow (6 \times 2)$ *keypoints*
    if $(frame = 0)$ **then**
        2DLengths $\leftarrow$ *2D limb lengths*
    else
        calculate DepthDifferences
        angles $\leftarrow$ *calculate angles using DepthDifferences and 2DLengths*
    end if
    $frame \leftarrow frame + 1$
end for

and tibia is used for all the respective frames. This procedure is applied to both legs separately.

3.4 DeepLEAD

In Section 3.3, we explained an approach by which we employed the 2D keypoints extracted by a deep learning method and using some trigonometric adjustments, we could estimate the lower extremity angle (knee). We name this algorithm as *DeepLEAD*, in which the first half, *Deep*, indicates the Deep Learning backbone of the algorithm (OpenPose). The second half, *LEAD*, stands for *Lower Extremity Angle Determination*.

Using this novel approach is totally effective and the reason for that is three-fold. First, we are using a 2D pose estimation method (OpenPose) that is very robust and extremely powerful. It can easily be used for real-time applications (like our problem) and it does not require exhaustive hardware and processing backbone. Second, the simple trigonometric-based idea behind projecting the 2D information into a 3D angle is quite easy to digest and comprehend. And finally, we can use a straightforward code to implement it in Python.

Algorithm 1 summarizes the steps in DeepLEAD. As the algorithm indicates,

the input is the image sequence, which comes from Kinect in our case. In general, the input images could be from any source, e.g., webcams or smartphone cameras. Following the procedure, we will have an array of size $N \times 2$, in which N refers to the number of total frames in the sequence. This array contains the knee flexion angles of the right and left legs, estimated by DeepLEAD, for every single input frame. The 6×2 keypoints in line 6 refers to the 6 lower extremity keypoints, each having 2 coordinates (x, y).

In Algorithm 1, after deriving the 2D keypoints from OpenPose in line 3, the rest of the algorithm contains what we propose as the novelty and our contribution to the problem. The for loop in line 6 performs all the processing on each $json$ file separately and saves the estimated angles in the appropriate array. This procedure checks if the current frame corresponds to the first frame in the sequence. In this case, the depth differences are considered to be zero and therefore, the 2D length of the lower limb vectors will be saved for the next frames. Otherwise, when we are not dealing with the first frame, the algorithm calculates the depth differences using the 2D lengths from the first round of the for loop and carries on the angle estimations.

The coordinate system in OpenPose originates at the top-left pixel $(0, 0)$ and ends at the bottom-right pixel $(1919, 1079)$ in an exported RGB image. Both x and y are in the image plane and they grow in width and height directions, respectively. They can take either a real pixel value or a value relative to the width and the height of the image (width=1920, height=1080). We can specify the output format by a keyword when running the code for inference. In order to avoid using large numbers in the order of 1000, e.g., values between 0 and 1919 for x, and also to keep the consistency in our implementation, we use normalized coordinate value which are between 0 and 1.

3.5 Summary

In this section, we introduced DeepLEAD, which is a novel approach towards using a 2D pose estimation method based on deep learning and combining it by a trigonometric adjustment to estimate the 3D angle of the knee joint. To investigate the feasibility of this algorithm, we will present our experimental setup and provide the results in the next chapter.

Chapter 4

Experiments and Results

We start this chapter by introducing our numerical error metrics in Section 4.1. For the sake of verification, we conduct an experiment with two Kinects in Section 4.2. In Section 4.3, we present the quantitative and qualitative results of two sets of experiments. Section 4.4 provides an insight about the execution time of each method and finally, we conclude the chapter in Section 4.5 by a brief discussion about the outcome of the experiments.

4.1 Numerical Error Metrics

In this section, we will introduce the error metrics that we use to quantitatively evaluate the performance of our implementations with respect to the reference, which is the Microsoft Kinect. Since the output values we are dealing with are the angles (in degree), we can consider our target as a regression problem and use the numerical metrics that are usually used for these types of situations. It is also very important to select the relevant specific metrics according to the problem. For example, in situations where the error in each data point is greater than 1, Mean Squared Error (MSE) might not be a relevant choice, since the value of error grows significantly by a factor of 2 and it does not offer a meaningful comparison. Having that in mind, we chose the following metrics.

4.1.1 Mean Absolute Error

Mean Absolute Error (MAE), as the name suggests, is simply the mean of absolute errors between the two quantities in comparison and it is defined by Equation 12. MAE can tell us what magnitude of error we should expect on average.

$$MAE = \frac{1}{N}\Sigma_{i=1}^{N}|\alpha_i - \theta_i| \tag{12}$$

In Equation 12, N denotes the total number of frames in the sequence, α_i and θ_i represent the calculated knee flexion angles of the estimator (VideoPose3D or DeepLEAD, in our case) and Kinect.

4.1.2 Root Mean Squared Error

It is worth mentioning that MAE considers errors on an average scale and it can obviously understate the infrequent large errors that exist. To better accommodate these outliers and to offer a more robust metric, we nominate Root Mean Squared Error (RMSE). Equation 13 defines RMSE.

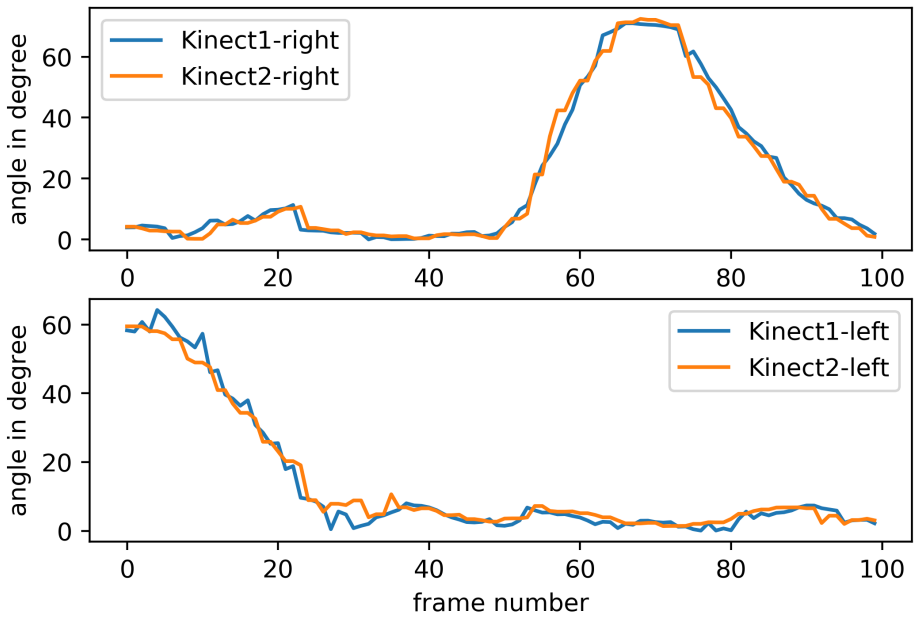
$$RMSE = \sqrt{\frac{1}{N}\Sigma_{i=1}^{N}(\alpha_i - \theta_i)^2} \tag{13}$$

Statistically speaking, RMSE is able to pinpoint the occasional, infrequent but big errors by assigning a relatively bigger weight to them. In other words, we can recognize the occurrence of these outliers by comparing RMSE against MAE. This enables us to effectively evaluate our implementations.

4.2 Double-Kinect Verification

Although we have highlighted some papers that investigated the margin of reliability and accuracy of Microsoft Kinect on real-world data in chapter 1 ([19], [20], [21]), we decided to demonstrate a simple test to verify that. For that matter, we used two separated Kinect V2s (named Kinect1 versus Kinect2, for simplicity). One Kinect is connected to a laptop and the other one to a PC. The Kinects were placed on top of each other. Then we carried on the calculations for 3D knee flexion angles of the same synchronized frames. Therefore, we are only dealing with the extracted data from the two Kinects.

(a) A sample frame



(b) Knee flexion angle of the right legs (top) and the left legs (bottom) from Kinect1 and Kinect2

Figure 16: Double-Kinect demonstration

Table 1: Error metric for knee flexion angle of Kinect1 versus Kinect2.

Metric	Right leg	Left leg
MAE (deg)	1.871	1.361
RMSE (deg)	2.688	2.672

4.2.1 Qualitative Evaluation

In Figure 16, we can see a sample at the top and the calculated knee flexion angles for both legs at the bottom. As we expected, they produce very similar results. It is not possible that both devices produce exactly the same results, with no jitter. By its nature, Kinect is not a perfect device and several factors, such as the PC configuration, the fluctuation in each Kinect's framerate, etc. can affect the Kinect keypoint prediction, and eventually, the angles. Therefore, this is acceptable for the purpose of comparison.

4.2.2 Quantitative Evaluation

When comparing two Kinects under relatively similar circumstances, we do not expect to have big jitters. Therefore, it is totally reasonable to choose MAE as the numerical metric. We also added the RMSE to have a consistency between our tables. Table 1 reflects the amount of discrepancy between our two Kinect devices. As explained in the previous subsection, this magnitude of error between the two devices is almost negligible.

4.3 Experiments and Results

In all of our experiments, we consider the Kinect as the reference. All of our data collections are done using a Kinect V2. For the purpose of comparing VideoPose3D and DeepLEAD to Kinect, we conducted two separate experiments with 70 (named experiment 1) and 85 (named experiment 2) consecutive frames exported by Kinect. We fed these sequences to both VideoPose3D and DeepLEAD to measure their performance. These sequences start with a frame where the subject is standing straight in front of the camera to comply with the fundamental assumption we made for the

proposed algorithm, DeepLEAD.

4.3.1 Experiment 1

Figure 17-(a) depicts a randomly-selected test frame from the sequence. The corresponding OpenPose output of that sample frame is included in Figure 17-(b) where the 2D skeleton is overlaid on top of the original frame. The left picture in Figure 17-(c) depicts the 2D keypoints detected by Detectron and the right picture shows the corresponding 3D skeleton. The model that we used to predict the 3D keypoints is the one that we talked about in Section 2.7. We need to mention that the 3D position of the predictions for VideoPose3D are exported with respect to a reference joint, meaning that those numbers are in camera space. As discussed before, this does not affect the result of angle calculations.

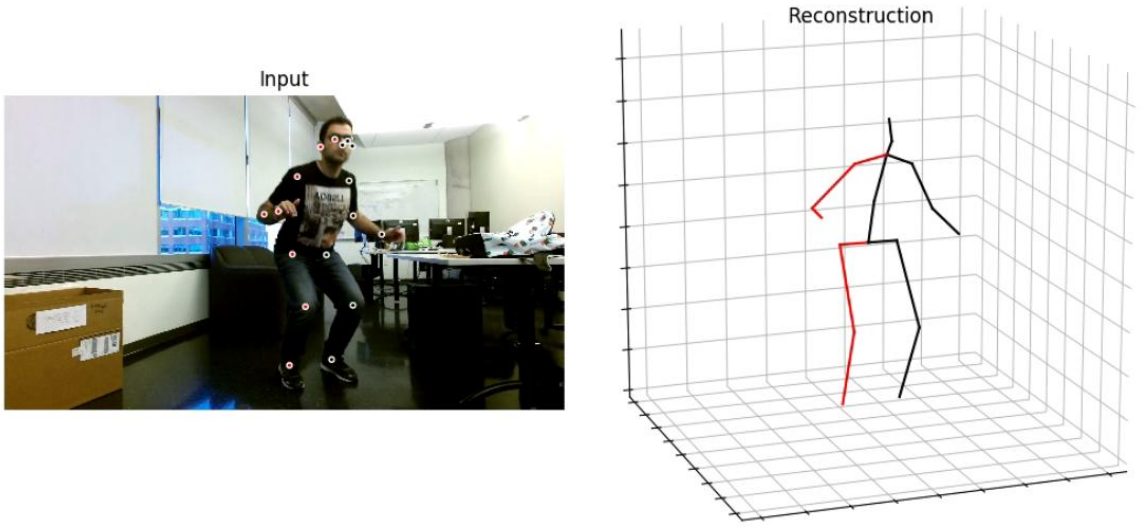
In Figure 18, we can see the estimated knee flexion angles from DeepLEAD and VideoPose3D for both legs where each of them are compared against Kinect, separately. By looking at Figure 18-(a), we realize that DeepLEAD is trying to follow the pattern of Kinect's output. In case of Figure 18-(b), VideoPose3D fails to predict the angles for frame 20 and its neighbor frames. It seems that VideoPose3D is not responding effectively to sudden changes of the knee angle.

Figure 19 combines the two figures in Figure 18 and demonstrates the performance of DeepLEAD and VideoPose3D versus Kinect, all in the same figure and for both legs.

To have a sense of numerical performance versus Kinect, Tables 2 and 3 present the MAE and RMSE of both legs for DeepLEAD and VideoPose3D, respectively. In both tables, the 2 to 4 degrees difference between MAE and RMSE implies that we have infrequent large errors for some frames. That was the main motivation behind choosing RMSE to reflect these kinds of errors.

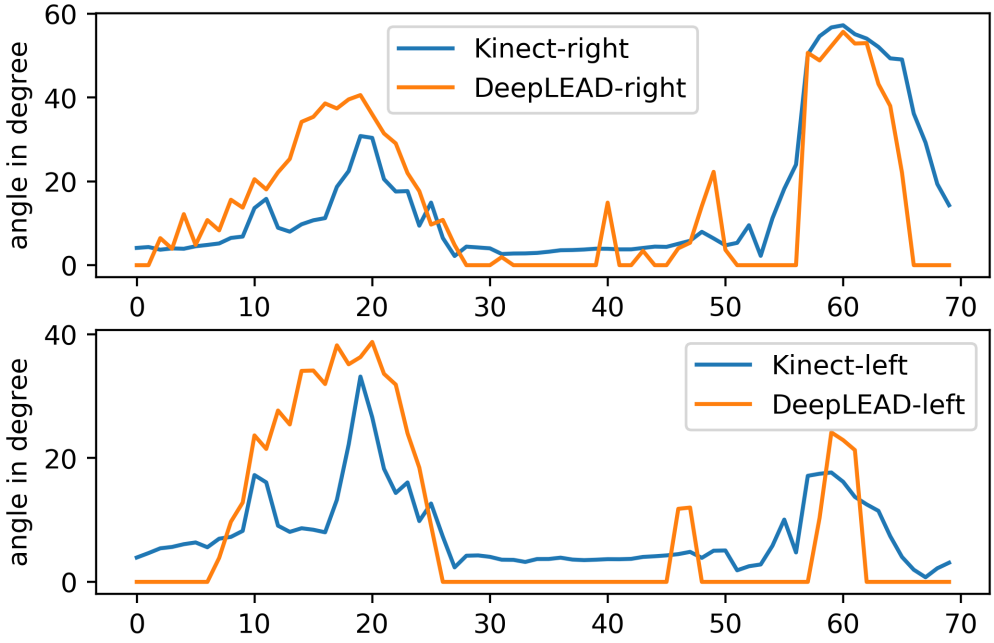
(a) Original test frame

(b) OpenPose output

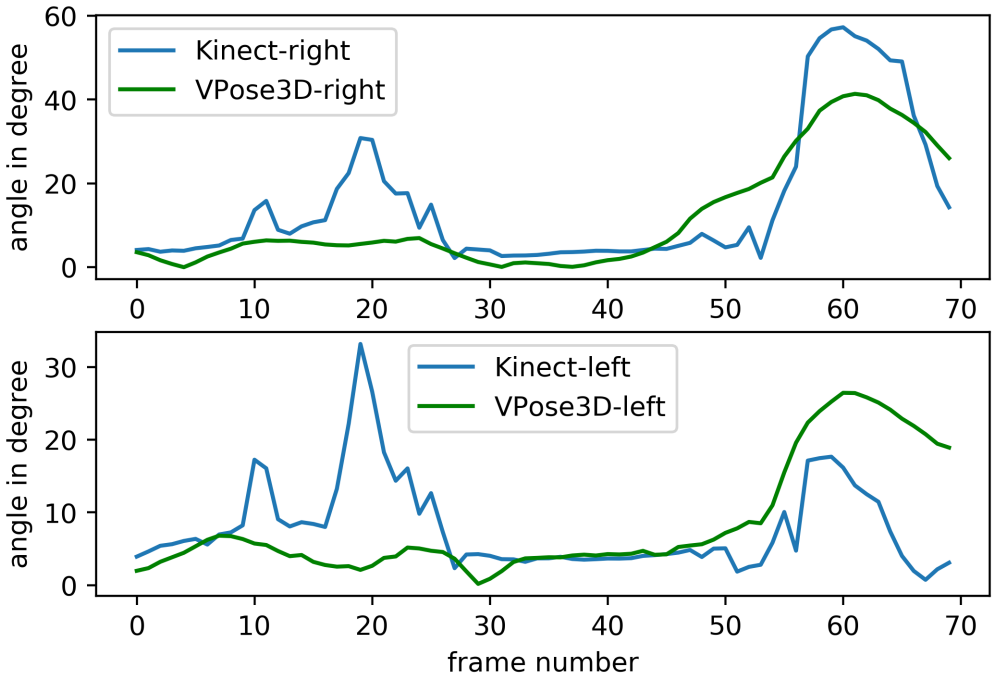


(c) VideoPose3D output along with its 3D reconstruction

Figure 17: Experiment 1 to compare the visual output of OpenPose and VideoPose3D



(a) Kinect versus DeepLEAD



(b) Kinect versus VideoPose3D

Figure 18: Knee flexion angle comparison of DeepLEAD and VideoPose3D versus Kinect for both legs in experiment 1

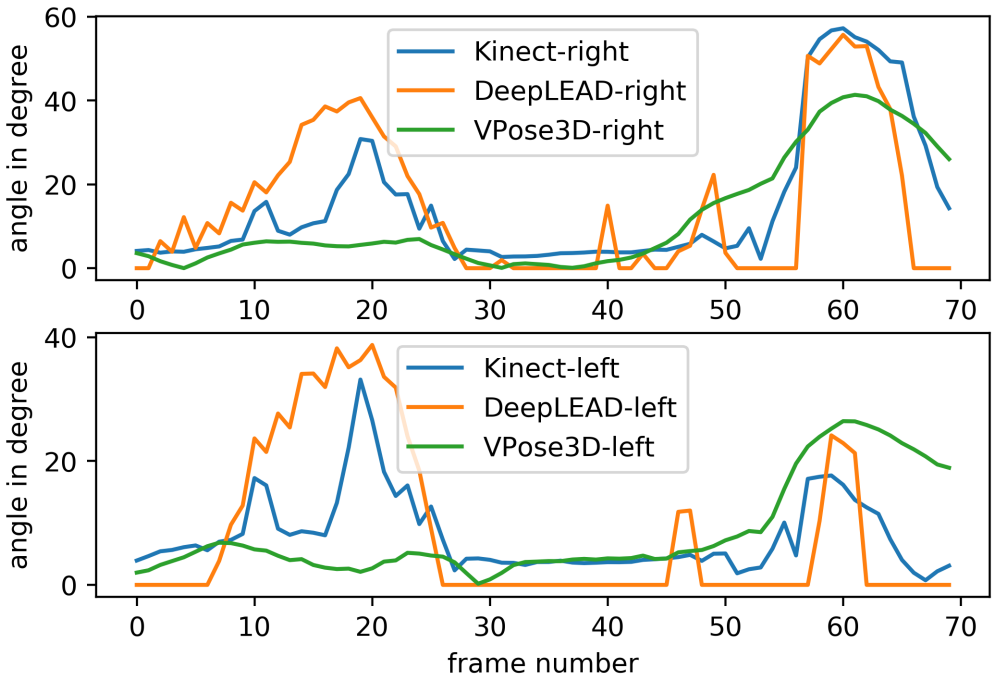


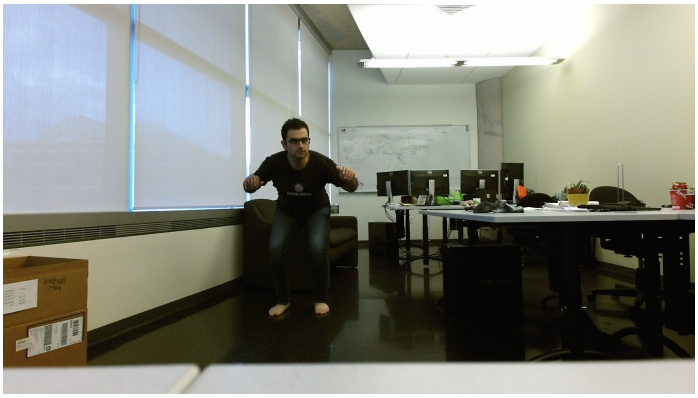
Figure 19: Knee flexion angle comparison of DeepLEAD, VideoPose3D, and Kinect in experiment 1

Table 2: Error metrics for knee flexion angle of DeepLEAD versus Kinect in experiment 1

Metric	Right leg	Left leg
MAE (deg)	8.352	7.138
RMSE (deg)	11.627	9.270

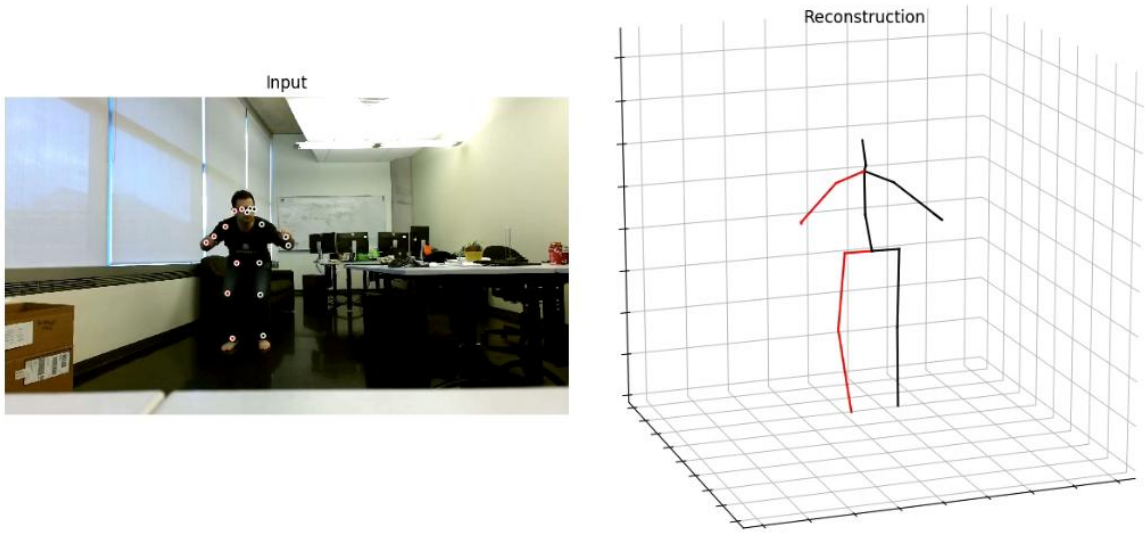
Table 3: Error metrics for knee flexion angle of VideoPose3D versus Kinect in experiment 1

Metric	Right leg	Left leg
MAE (deg)	6.957	6.306
RMSE (deg)	9.344	9.216



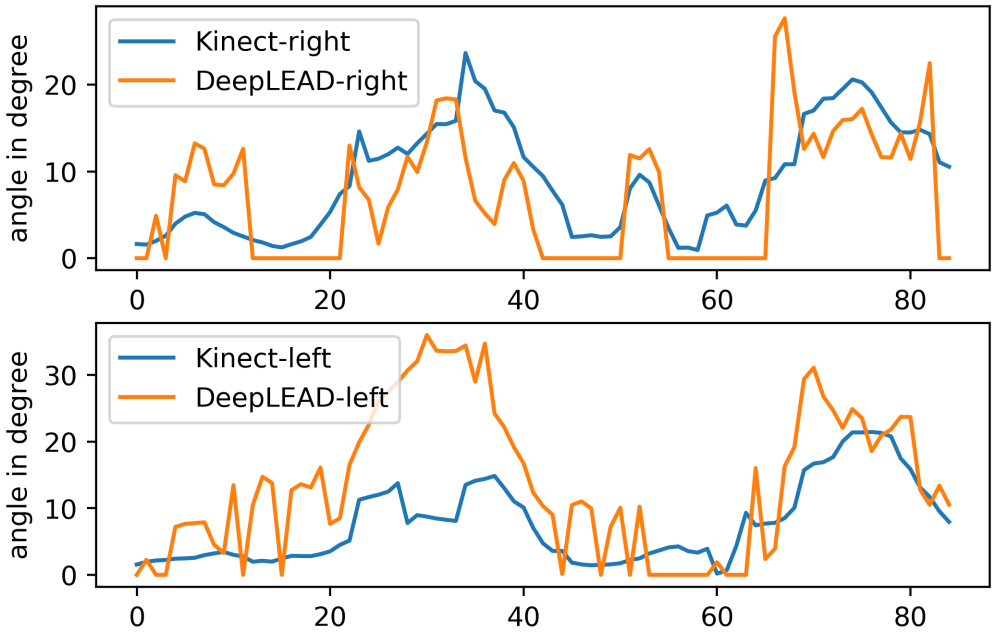
(a) Original test frame

(b) OpenPose output

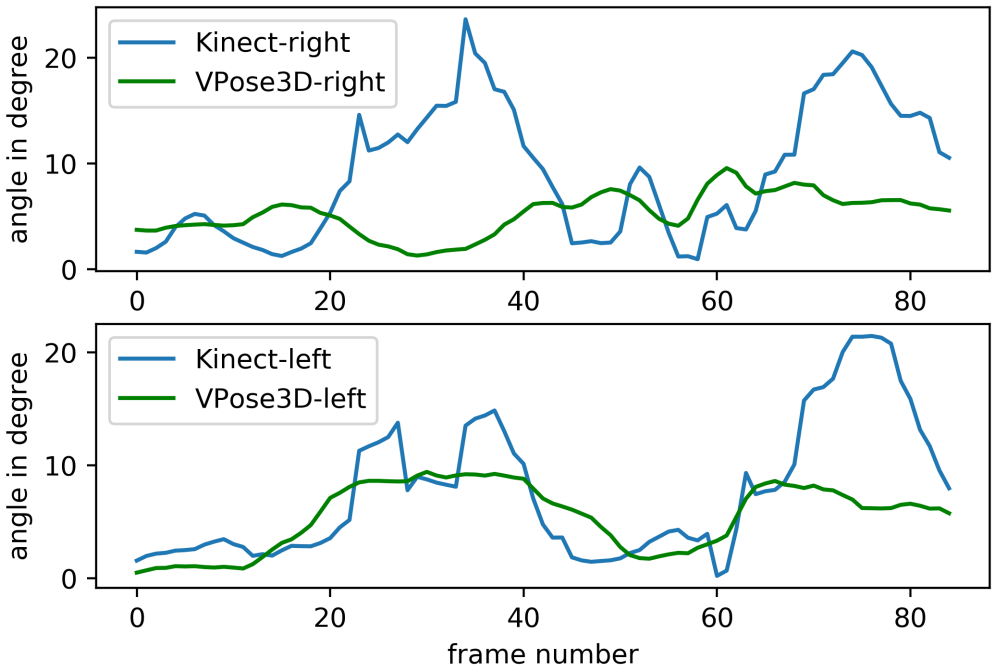


(c) VideoPose3D output along with its 3D reconstruction

Figure 20: Experiment 2 to compare the visual output of OpenPose and VideoPose3D



(a) Kinect versus DeepLEAD



(b) Kinect versus VideoPose3D

Figure 21: Knee flexion angle comparison of DeepLEAD and VideoPose3D versus Kinect for both legs in experiment 2

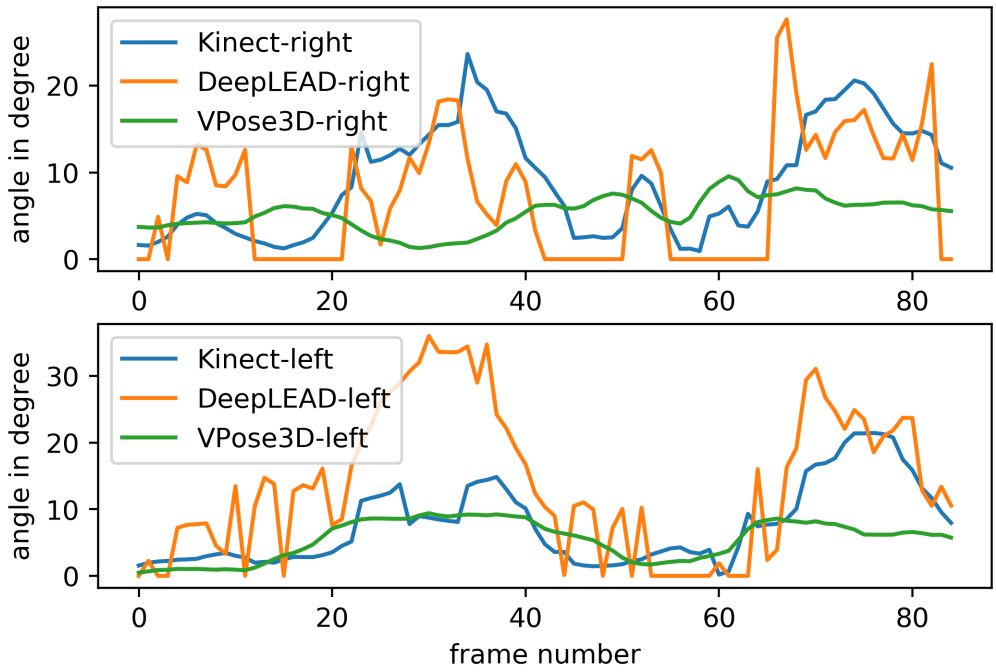


Figure 22: Knee flexion angle comparison of DeepLEAD, VideoPose3D, and Kinect in experiment 2

4.3.2 Experiment 2

To have a rigorous verification of our results, in this subsection, we present figures and tables for a new sequence similar to experiment 1. In Figure 20, we can find a sample frame, the OpenPose 2D skeleton and the corresponding output of VideoPose3D, from top to bottom, respectively.

To visually examine the performance in experiment 2, we refer to Figure 21. DeepLEAD seems to have a good grasp of the changes, except for the frames between 20 to 40 of the left leg where we find a rather considerable error. The same happens with VideoPose3D in right leg.

Overall, as Tables 4 and 5 shows, we are experiencing a better performance in terms of MAE and RMSE for VideoPose3D.

The subject's orientation in front of the camera seems to have an impact on the results. For example, in experiment 2, the subject is looking at the camera with no rotation whereas in experiment 1, the subject turns slightly away from the camera. Therefore, it is important to consider this when designing the experiment setup.

Table 4: Error metrics for knee flexion angle of DeepLEAD versus Kinect in experiment 2

Metric	Right leg	Left leg
MAE (deg)	5.059	7.877
RMSE (deg)	6.211	10.201

Table 5: Error metrics for knee flexion angle of VideoPose3D versus Kinect in experiment 2

Metric	Right leg	Left leg
MAE (deg)	6.127	3.541
RMSE (deg)	7.859	5.202

4.4 Running Time

In order to investigate which method works more efficiently in terms of running time, we measured the time it took for both DeepLEAD and VideoPose3D from receiving an input video with 50 frames per second (fps) to calculate the knee flexion angles of all the frames. The end-to-end runtimes are reported in seconds in Table 6. As the numbers in the table suggest, VideoPose3D is almost 8 times slower than DeepLEAD. This automatically rules out VideoPose3D for real-time processing and gives DeepLEAD a great margin over VideoPose3D.

Table 6: Runtime comparison of DeepLEAD versus VideoPose3D in experiments 1 and 2

Experiment	DeepLEAD	VideoPose3D
Exp. 1 (sec)	9.435	72.741
Exp. 2 (sec)	11.188	87.217

4.5 Discussion

The numbers in Tables 2, 4, 3, and 5 are proving a slightly better performance of VideoPose3D against DeepLEAD. We should remember that in the case of DeepLEAD, we are predicting a 3D value (angle) without having any kind of clue of the depth but VideoPose3D is doing a 3D(keypoints)-to-3D(angles) mapping. Intuitively, this can justify the numbers and figures in the results. To achieve this, VideoPose3D compromises the speed versus DeepLEAD. On the other hand, VideoPose3D can accomplish the task in-hand utilizing a much more complicated and compute-hungry procedure. In addition, Section 4.4 gives us an empirical insight about this issue. This indicates that VideoPose3D might not be the best choice for a rather real-time purpose, e.g., smart phones, where the processing resources are limited. Therefore, DeepLEAD seems to be a better choice considering both performance and execution time.

We can also realize that for both DeepLEAD and VideoPose3D, RMSE is about 2 to 4 degrees greater than MAE. According to the discussion in section 4.1, this implies that we have infrequent large errors. We can also verify this by looking at Figures 19 and 22.

Due to the basic assumptions we made in DeepLEAD, for some frames, the argument inside the arccos function exceeds the permissible range ($-1 < \cos(\theta) < 1$). To alleviate this situation, we had to clip the values to the maximum valid quantities. This suppressed the angle prediction to 0 degree for some middle frames. There might be a better solution for this issue which opens another door for improvement of the algorithm.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we studied two CNN-based methods for estimating lower extremity angles for the first time. We validated the possibility of using CNNs as a more convenient alternative to older technologies such as Microsoft Kinect and motion analysis systems, particularly in the task of lower limb injury prediction. These CNN-based methods can eliminate the need of using non-portable traditional MoCap systems for accurate skeletal data collection in sports medicine. Thanks to the extraordinary capabilities of CNNs in predicting body keypoints from image sequences, they considerably simplify the whole process – as easy as taking a video from the subject and feeding them to the network. While VideoPose3D, as an already established method for 3D pose estimation is widely available as an open-source package with multiple pre-trained models, it offers a solution that is computationally expensive and time-consuming. Therefore, we designed another algorithm that is also based on CNNs. We used OpenPose as the backbone and developed few strategies to mitigate the need for an end-to-end 3D skeletal tracking system. We named our proposed method DeepLEAD wherein LEAD stands for Lower Extremity Angle Determination. It works at a frame rate of almost 8 fps with rather acceptable performance which makes it a better choice compared to VideoPose3D. Although DeepLEAD, like any other novel approaches has its own presumptions that might limit its capabilities, the improvement doors always remain open. It started a novel journey towards using deep learning in that specific application in sports medicine and is capable of

motivating others to shift their work towards using CNNs and machine learning.

5.2 Future Work

The work presented in this thesis can be improved on different fronts. Here are some suggestions for future avenues of research:

- We have seen that Microsoft Kinect might not offer the best accuracy and reliability, when it comes to comparing it to a newly proposed approach. In that case, for the sake of having a very accurate reference while developing a new method, MoCap systems in a dedicated lab environment can be the alternative way to go.

- DeepLEAD shows a promising performance and might be able to surpass VideoPose3D in terms of performance while maintaining the low computational complexity. By applying and testing more conditions on the algorithm we proposed in DeepLEAD, this might become a reality.

- In some situations in DeepLEAD, the argument of *arccos* becomes greater than 1 or less than -1. To mitigate this, we had to round it to 1 or -1, respectively, which is the underlying reason we had 0 degree values in DeepLEAD figures in the previous chapter. There might be some other way to alleviate this problem rather than rounding the angle to 0.

- The task of 3D pose estimation has been remaining as a highly challenging problem and the extensive attention continues to grow among researchers. By designing new algorithms that do not compromise speed over accuracy, or vice versa, these methods can become a great candidate for the task of automating the lower extremity angle predictions.

- As the current methods presented in the thesis have acceptable performances, they can be ported to smart phones or tablets to offer a great degree of portability and ease of use.

- We fed the full size (1920×1080) images exported by Kinect as input to the methods which might become an overhead for the CNN to process it. The images could potentially be down-sampled to achieve a faster implementation.

- By improving DeepLEAD in terms of accuracy, the study of knee injury prediction programs could eventually be shifted to a solution where deep learning plays an undeniable role in these studies. Although most of the items in LESS system's checklist relies on an operator to meticulously monitor the process of the vertical drop jump test, the items that require precise angle calculations could be delegated to the algorithms. The only input these algorithms need from the subject is multiple frames during the test procedure.

- One of the assumptions we made when we came up with DeepLEAD was to consider the length of femur and tibia (Equations 6 and 7) to be constant throughout the whole sequence. Although VideoPose3D is a general 3D pose estimation method which does not imply any kind of constraint on the depth information of the subject, imposing a fixed length criterion on lower limbs might improve the accuracy in VideoPose3D.

The work presented in this thesis is a novel approach to lower extremity angle estimation that uses deep learning. Given that CNNs has shown very promising solutions when the input to the algorithm is an image, it motivated us to go through that path and try to make use of their capabilities in our own project. As technology evolves over time, improving different components of this work, such as employing better CNN models or using deeper networks with more efficient training strategies might offer further improvements.

Bibliography

[1] J. Isear, J. Erickson, and T. Worrell, "Emg analysis of lower extremity muscle recruitment patterns during an unloaded squat," *Medicine & Science in Sports & Exercise*, vol. 29, no. 4, pp. 532–539, 1997.

[2] J. Noble, C. A. Munro, V. S. Prasad, and R. Midha, "Analysis of upper and lower extremity peripheral nerve injuries in a population of patients with multiple injuries," *Journal of Trauma and Acute Care Surgery*, vol. 45, no. 1, pp. 116–122, 1998.

[3] D. T. Leetun, M. L. Ireland, J. D. Willson, B. T. Ballantyne, and I. M. Davis, "Core stability measures as risk factors for lower extremity injury in athletes," *Medicine & Science in Sports & Exercise*, vol. 36, no. 6, pp. 926–934, 2004.

[4] S. McLean, K. Walker, and A. J. van den Bogert, "Effect of gender on lower extremity kinematics during rapid direction changes: An integrated analysis of three sports movements," *Journal of Science and Medicine in Sport*, vol. 8, no. 4, pp. 411–422, 2005.

[5] M. Hägglund, M. Waldén, and J. Ekstrand, "Risk factors for lower extremity muscle injury in professional soccer: The uefa injury study," *The American journal of sports medicine*, vol. 41, no. 2, pp. 327–335, 2013.

[6] J. B. Taylor, K. R. Ford, A.-D. Nguyen, L. N. Terry, and E. J. Hegedus, "Prevention of lower extremity injuries in basketball: A systematic review and meta-analysis," *Sports Health*, vol. 7, no. 5, pp. 392–398, 2015.

[7] B. P. Boden, L. Y. Griffin, and W. E. Garrett Jr, "Etiology and prevention of noncontact acl injury," *The Physician and sportsmedicine*, vol. 28, no. 4, pp. 53–60, 2000.

[8] B. Yu and W. E. Garrett, "Mechanisms of non-contact acl injuries," *British journal of sports medicine*, vol. 41, no. suppl 1, pp. i47–i51, 2007.

[9] P. Gelber, *What is the anterior cruciate ligament?* `https://www.drgelber.com/en/ligamento-cruzado-anterior/`, Online; accessed June 01 2020.

[10] T. E. Hewett, G. D. Myer, K. R. Ford, R. S. Heidt Jr, A. J. Colosimo, S. G. McLean, A. J. Van den Bogert, M. V. Paterno, and P. Succop, "Biomechanical measures of neuromuscular control and valgus loading of the knee predict anterior cruciate ligament injury risk in female athletes: A prospective study," *The American journal of sports medicine*, vol. 33, no. 4, pp. 492–501, 2005.

[11] D. A. Padua, L. J. DiStefano, A. I. Beutler, S. J. De La Motte, M. J. DiStefano, and S. W. Marshall, "The landing error scoring system as a screening tool for an anterior cruciate ligament injury–prevention program in elite-youth soccer athletes," *Journal of athletic training*, vol. 50, no. 6, pp. 589–595, 2015.

[12] L. E. Imwalle, G. D. Myer, K. R. Ford, and T. E. Hewett, "Relationship between hip and knee kinematics in athletic women during cutting maneuvers: A possible link to noncontact anterior cruciate ligament injury and prevention," *Journal of strength and conditioning research/National Strength & Conditioning Association*, vol. 23, no. 8, p. 2223, 2009.

[13] S. Norouzi, F. Esfandiarpour, S. Mehdizadeh, N. K. Yousefzadeh, and M. Parnianpour, "Lower extremity kinematic analysis in male athletes with unilateral anterior cruciate reconstruction in a jump-landing task and its association with return to sport criteria," *BMC musculoskeletal disorders*, vol. 20, no. 1, p. 492, 2019.

[14] J. Hashemi, R. Breighner, N. Chandrashekar, D. M. Hardy, A. M. Chaudhari, S. J. Shultz, J. R. Slauterbeck, and B. D. Beynnon, "Hip extension, knee flexion paradox: A new mechanism for non-contact acl injury," *Journal of biomechanics*, vol. 44, no. 4, pp. 577–585, 2011.

[15] D. A. Padua, S. W. Marshall, M. C. Boling, C. A. Thigpen, W. E. Garrett Jr, and A. I. Beutler, "The landing error scoring system (less) is a valid and reliable clinical assessment tool of jump-landing biomechanics: The jump-acl study," *The American journal of sports medicine*, vol. 37, no. 10, pp. 1996–2002, 2009.

[16] D. A. Padua, M. C. Boling, L. J. DiStefano, J. A. Onate, A. I. Beutler, and S. W. Marshall, "Reliability of the landing error scoring system-real time, a clinical assessment tool of jump-landing biomechanics," *Journal of sport rehabilitation*, vol. 20, no. 2, pp. 145–156, 2011.

[17] B. Dai, S. Leigh, H. Li, V. S. Mercer, and B. Yu, "The relationships between technique variability and performance in discus throwing," *Journal of sports sciences*, vol. 31, no. 2, pp. 219–228, 2013.

[18] W. Tao, T. Liu, R. Zheng, and H. Feng, "Gait analysis using wearable sensors," *Sensors*, vol. 12, no. 2, pp. 2255–2283, 2012.

[19] M. Eltoukhy, A. Kelly, C.-Y. Kim, H.-P. Jun, R. Campbell, and C. Kuenze, "Validation of the microsoft kinect® camera system for measurement of lower extremity jump landing and squatting kinematics," *Sports Biomechanics*, vol. 15, no. 1, pp. 89–102, 2016.

[20] E. Nyman Jr, "Validity of a kinect-based tracking system for clinical assessment of knee kinematics," *International Journal of Sports and Exercise Medicine*, vol. 2, Jan. 2016.

[21] S. Phommahavong, D. Haas, J. Yu, S. Krüger-Ziolek, K. Möller, and J. Kretschmer, "Evaluating the microsoft kinect skeleton joint tracking as a tool for home-based physiotherapy," *Current Directions in Biomedical Engineering*, vol. 1, no. 1, pp. 184–187, 2015.

[22] M. H. Hassoun *et al.*, *Fundamentals of artificial neural networks*. MIT press, 1995.

[23] B. Xiong, N. Zeng, H. Li, Y. Yang, Y. Li, M. Huang, W. Shi, M. Du, and Y. Zhang, "Intelligent prediction of human lower extremity joint moment: An artificial neural network approach," *IEEE Access*, vol. 7, pp. 29 973–29 980, 2019.

[24] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[25] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[26] D. Pavllo, C. Feichtenhofer, D. Grangier, and M. Auli, "3d human pose estimation in video with temporal convolutions and semi-supervised training," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[27] R. Lun and W. Zhao, "A survey of applications and human motion recognition with microsoft kinect," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 29, no. 05, p. 1 555 008, 2015.

[28] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1653–1660.

[29] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *European conference on computer vision*, Springer, 2016, pp. 483–499.

[30] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *CVPR*, 2016.

[31] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[33] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He, *Detectron*, `https://github.com/facebookresearch/detectron`, 2018.

[34] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[35] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, `https://github.com/facebookresearch/detectron2`, 2019.

[36] D. Mehta, S. Sridhar, O. Sotnychenko, H. Rhodin, M. Shafiei, H.-P. Seidel, W. Xu, D. Casas, and C. Theobalt, "Vnect: Real-time 3d human pose estimation with a single rgb camera," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 1–14, 2017.

[37] S. Li and A. B. Chan, "3d human pose estimation from monocular images with deep convolutional neural network," in *Asian Conference on Computer Vision*, Springer, 2014, pp. 332–347.

[38] X. Zhou, Q. Huang, X. Sun, X. Xue, and Y. Wei, "Towards 3d human pose estimation in the wild: A weakly-supervised approach," in *IEEE International Conference on Computer Vision*, 2017, pp. 398–407.

[39] J. Martinez, R. Hossain, J. Romero, and J. J. Little, "A simple yet effective baseline for 3d human pose estimation," in *IEEE International Conference on Computer Vision*, 2017, pp. 2640–2649.

[40] D. Tome, C. Russell, and L. Agapito, "Lifting from the deep: Convolutional 3d pose estimation from a single image," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2500–2509.

[41] C.-H. Chen and D. Ramanan, "3d human pose estimation= 2d pose estimation+ matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7035–7043.

[42] G. Pavlakos, X. Zhou, K. G. Derpanis, and K. Daniilidis, "Coarse-to-fine volumetric prediction for single-image 3d human pose," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7025–7034.

[43] A. Amidi and S. Amidi, *Recurrent neural network cheatsheet*, `https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks`, Online; accessed June 01 2020.

[44] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, 2013, pp. 6645–6649.

[45] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *IEEE conference on computer vision and pattern recognition*, 2017.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[47] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, 2014.

[48] M. Rayat Imtiaz Hossain and J. J. Little, "Exploiting temporal information for 3d human pose estimation," in *European Conference on Computer Vision (ECCV)*, 2018, pp. 68–84.

[49] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1325–1339, 2014.

[50] C. Ionescu, F. Li, and C. Sminchisescu, "Latent structured models for human pose estimation," in *International Conference on Computer Vision*, 2011.

[51] Y. Zhu, Y. Zhao, and S.-C. Zhu, "Understanding tools: Task-oriented object modeling, learning and recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[52] C. Kim, S. Yun, S.-W. Jung, and C. S. Won, "Color and depth image correspondence for kinect v2," in *Advanced Multimedia and Ubiquitous Engineering*, J. J. (H. Park, H.-C. Chao, H. Arabnia, and N. Y. Yen, Eds., 2015, pp. 111–116, ISBN: 978-3-662-47487-7.