IMPLEMENTATION OF MOCAP SOLVER ALGORITHM ON MOTION CAPTURES TO IMPROVE ACCURACY AND PERFORMANCE

Markos Panayiotou

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

December, 2022

APPROVAL PAGE

Master of Science Thesis

IMPLEMENTATION OF MOCAP SOLVER ALGORITHM ON MOTION CAPTURES TO IMPROVE ACCURACY AND PERFORMANCE

Presented by

Markos Panayiotou

Research Supervisor _____
Aristidou Andreas

Committee Member _____
Pattichis Constantinos

Committee Member _____
Kakas Antonis

University of Cyprus

December, 2022

ABSTRACT

One of the main issues that Motion Capture systems face is to accurately note all markers within the 3D environment. While the accuracy varies according to the system used, there are algorithms that use Deep Learning techniques to place and fix the any offset that a Motion Capture may have, according to the training data given. In this Thesis, I will test the MoCap Solver algorithm on the DanceDB Motion Captures in order to fix the misplaced markers. The results show that the animations can be fixed but a better training set for the neural network is required for better results.

Markos Panayiotou – University of Cyprus, 2022

ACKNOWLEDGEMENTS

I would like to thank my supervisor Andreas Aristidou for guiding me and helping me throughout my research and work of this thesis. His enthusiasm for this project was pivotal for me to keep my spirit high. Additionally, I would like to thank him for finding time to help me with any questions I had and organizing a semi-weekly meeting due to the difficult schedule I had.

I would also like to thank my friends and family for supporting me during the process of this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Contents

1.1 Motivation

A Motion Capture (or motion tracking) describes the process that specific cameras capture and record the movement of objects within a 3D environment. Usually, during sessions, human actors are used in this environment by wearing a suit with markers on the body. The aforementioned cameras pinpoint these markers and calculate their space in the environment creating a series of movement for each point. Later, these points are translated into a 3D animation which can be mapped to a 3D model. Therefore, the model will simulate the actions the human actor has performed. During such a session there is always a possibility of a human error or camera error if any one of the markers is not captured during each of the images taken by the cameras. Such

issue can result into the unwanted scenario that a part of the body of the 3D model will behave differently than how the actor has performed.

To address this issue multiple algorithms have been developed in order to fix and re-track the motion capture to the correct state. One such algorithm is the MoCap Solver[1], a Neural Solver that uses a trained deep learning algorithm to adjust an SMPL file according to the training data given. Since the entire process of solving a Motion Capture and re-targeting the markers is exceedingly time-consuming, the need of an algorithm that can do both in less time would be really helpful.

The process of Motion Capture Solving is a valuable and important area of research and development. The main motivation for pursuing a better solving technique for the DanceDB dataset will help with the problem of the marker occlusion that we have. We aim to find a way to improve the realism and how believable our 3D animations and simulations look. We also need to limit the time spent to do the Motion Capture Solving. The manual way of doing so requires for a person to go through each frame and fix the occluded and misplaced markers, fix joint position and rotation and adjust the animation to be closer to realism. Understandably, this takes a lot of time to do so and having in mind an entire dataset of animations might need solving, this will require a lot of people as well.

In order to avoid the time and resource consuming process, the direction to use an artificial intelligence system is expected. By using one such system we are guaranteed to reduce the cost needed and it can improve the scalability, since an AI system can handle large amounts of data and can be integrated into existing workflows. Additionally, by training the system with the right variables we can improve the accuracy and prediction of it, we will enhance the efficiency of the process and also make it flexible to different kinds of scenarios and requirements. One such system is the MoCap Solver.

Our aim is to use the MoCap Solver on the DanceDB SMPL files and check if the results do increase the accuracy and fix any inconsistencies in the performance. While the manual way of fixing the markers and adjusting them on the actor will always be better when it comes to results, due to the monitoring of the user, an automated way of handling these processes will definitely save a lot of time when handling lots of data.

In this Thesis I will discuss the usage of the SMPL files and why I needed them for the MoCap Solver. I will also analyze the way the algorithm works and discuss the problems adjusting the code to our data. Finally, I will show some of the results generated from the Neural Network and statistics regarding the results. I will also explain the difficulty in generating the results and the plans for future work that need to be done in order for the MoCap Solver to help the DanceDB dataset become more accurate.

1.2 Challenges

Handling Deep Learning algorithms has become relatively easy with the rapid increase of technology that can handle large amount of data. However, there will always be challenges when working with these kind of processes. Firstly, we had to adjust the code of the MoCapSolver accordingly to be able to run with the DanceDB data. The Python code that the algorithm was developed was using old libraries, with some of the code that it used being deprecated. Additionally some of the old versions of the libraries, specifically the cuDNN (CUDA Deep Neural Network) that affects PyTorch, the machine learning framework used in the project, could not use the latest Graphics Cards. We worked around this problem by adjusting the libraries in question to the newer ones while maintaining and changing some parts of the code to fix any inconsistencies.

1.3 Related Work

The problem with marker occlusion has been in the field and has been investigated multiple times. In *Real-time marker prediction and CoR estimation in optical motion capture*[13] we are presented with a method to accurately predict the positions of markers and center of rotation (CoR) in real-time in a motion capture system, even when some markers are occluded and not visible. The approach involves using an unscented Kalman filter with a variable turn model to infer the positions of occluded markers based on information from neighboring markers, and combining these predictions with information from a single camera when some markers are still visible. An inverse kinematics technique is also used to ensure that the lengths of bones remain constant over time, allowing for a continuous flow of data. The proposed method is tested against other popular marker prediction methods and is found to outperform them in estimating both marker and CoR positions.[13]

Another research, the *Robust solving of optical motion capture data by denoising*[14], presents a tool that uses machine learning to process raw optical motion capture data that may contain errors such as occluded markers, mislabeled markers, and high frequency noise or jitter. The tool, a denoising neural network, is trained using a large database of skeletal motion data that has been synthetically corrupted with errors similar to those found in real motion capture data. The neural network can then be used to produce joint transforms directly from the raw marker data, making the process of ”solving” more accurate and efficient. The system is capable of achieving precision within a few millimeters and is fast to compute with low memory requirements, making it suitable for use in production. The tool removes the need for manual clean-up of data and is robust to errors in the input data[14].

In the *MoCap-solver: a neural solver for optical motion capture data*[1], which we will analyze in later Chapters, the authors compare the results of their process with the one suggested in Holden 2018p[14]. Since both use similar techniques in neural networks, the differences in MoCap Solver show that it had better results. In this thesis we will be using the algorithm to solve the DanceDB Motion Captures from any occlusions and unwanted behaviors.

Chapter 2

Literature and Related Work

Contents

2.1 Introduction

There has been an increasing growth in the field of Computer Graphics. With the rising demand of end-user industries like entertainment, healthcare, architecture and many more, these technologies have been on high demand for upgrade and development. One of the sub-fields that got a lot of attention was the field of Motion Capture.

2.2 Motion Capture

Motion Capture describes the process that records the movement of an object or a person and translates it in a 3-D environment. One of the most famous industries that use Motion Capturing is the field of entertainment and specifically Video Game Development and Film-making. In these specific areas, it's used to generate a realistic animation of a human actor into a 3-D animation that will be used by a character in a 3-D environment.

In order to capture the motion of an actor, multiple cameras are set around a specific area and captures their movement, translating it into a file that can be used on a 3-D actor in the environment[2]. Multiple additions and changes have been done to the process since its inception, generating different methods of capturing movement such as Magnetic Motion Capture, Mechanical Motion Capture and Optical Motion Capture.

One common system for motion capture involves the use of 56 markers placed on an actor's body. These markers are typically placed on specific locations, such as the joints, to accurately track the actor's movements. The markers are typically small, reflective spheres that are tracked by cameras or other sensors to capture the movement data. This data is then used to animate a digital character or model in real-time or to create an animation for use in film, television, games, or other applications.

The number of markers used in a motion capture system can impact the accuracy and detail of the captured data, but it can also increase the complexity and cost of the system. As such, the choice of the number of markers to use depends on the specific requirements of the motion capture application and the resources available.

2.2.1 Motion Capture Solving

Motion capture solving is the process of converting raw motion capture data into a usable format, such as a 3D animation or simulation. This typically involves several steps. First step is the *Calibration step*. This involves aligning the motion capture data with a known reference frame, such as a 3D model or the real-world location of the motion capture system. Then, it's the process of *Cleaning*. This involves identifying and correcting any errors or noise in the motion capture data, such as misaligned markers or faulty sensor readings. *Retargeting* involves mapping the motion capture data onto a different 3D model or character, such as a digital avatar or a virtual puppet. This may involve adjusting the timing, spacing, or other characteristics of the motion to fit the new model. Finally, *Refining* involves adjusting the motion capture data to achieve a desired level of realism or believability, such as smoothing out jagged movements or adding natural-looking transitions between different actions.

There are various software tools and techniques that can be used to perform these tasks, depending on the specific needs and goals of the motion capture project. Some common tools include motion capture software packages, 3D animation software, and specialized plug-ins or scripts.

In general, motion capture solving requires a combination of technical skills and artistic judgment, as it involves both processing and manipulating large amounts of data and creating visually appealing and believable motion.

2.3 Data Capture

The Computer Graphics & Virtual Reality Laboratory of the Computer Science Department of the University of Cyprus is equipped with the Phasespace Impulse X2E motion capture system with active LEDs. The hardware is an Optical Motion Capture that uses 24 cameras and

captures 3D motion using modulated LEDs[3]. The Department has used this system to digitally archive traditional and modern dances as Motion Captures to preserve the cultural heritage and increase the awareness of the local community[4]. These are the data that we want to use in the MoCap Solver algorithm and check if the results will improve the performance of the digitized dances. AMASS[7], a large database of human motion capture, has created SMPL versions of the DanceDB Motion Captures. The data they generated are compatible with the input that is required for the MoCap Solver, therefore I used those.

2.4 SMPL

An SMPL, a Skinned Multi-Person Linear model, is a realistic 3D model that is based on blend shapes and skinning of the human body. It's created from thousands of 3D body scans. Since each marker can represent a location on the mesh surface of the body, an SMPL can help obtain information such as pose and shape from Motion Capture markers.[5] The downside of this, is that it's prone to noise and error. Since the body shapes of the human are usually different, or the fact that MoCap suits have different placement for markers, there is an unreliable factor when trying to map the two together.

2.4.1 Mosh++

The data that we are going to use are the SMPL versions of the DanceDB Motion Captures that AMASS[7] created. This is because the input that the MoCap-Solver developers used according to their article, is the CMU SMPL dataset[1]. Since both were created the same way the relation between the original input and the input that we want to add into the MoCap-Solver is strong. Firstly. to understand how the transformation works we need to know how the algorithm that created the SMPLs works. The process and code is described very carefully in the article they made

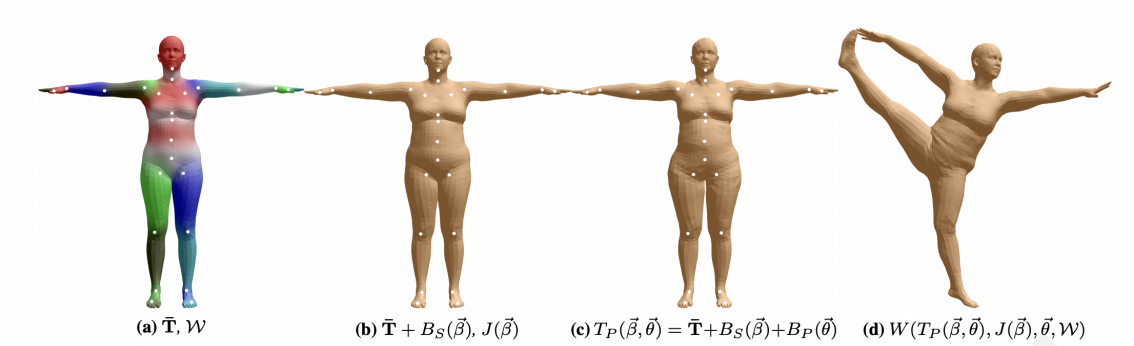


Figure 1: SMPL model. (a) Template mesh with blend weights indicated by color and joints shown in white. (b) With identity-driven blend shape contribution only; vertex and joint locations are linear in shape vector . (c) With the addition of pose blend shapes in preparation for the split pose; note the expansion of the hips. (d) Deformed vertices reposed by dual quaternion skinning for the split pose.[5]

about MoSh++[6,7]. In essence, MoSh++ estimates Body Shape, pose and soft tissue transformations from sparse markers. To simplify using a skeleton, the body shape and the motion capture, Mosh++ creates an SMPL version that has all the information needed with a more accurate way of displaying and performing the actor movement[6,7].

2.4.2 SMPL Data Analysis

The SMPL files from the CMU and DanceDB that we can download from the AMASS database are in .npz file format. Within the numpy file we can find 6 variables, ”trans, gender, mocap framerate, betas, dmpls, poses that form the SMPL. Trans describes the global translation of the body; Gender categorizes the SMPL to male, female or neutral according to the model that was used to generate the file; Mocap Framerate is just the frame rate at which the SMPL has been recorded; Betas represent the body shape and are used to control it; DMPLs are used to control the soft tissue of the actor and finally; Poses is the global root orientation for each frame recorded.

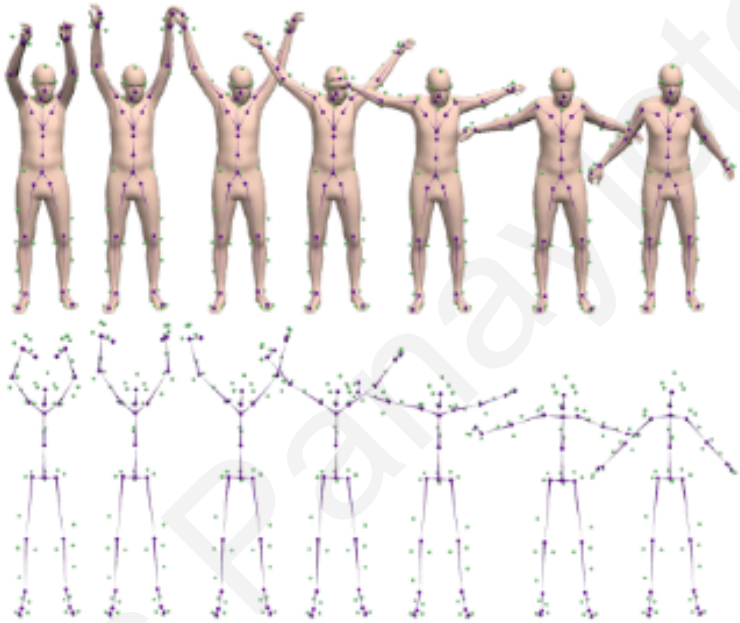


Figure 2: MoSh++ captures body shape, pose, and soft-tissue dynamics by fitting the surface of the SMPL/DMPLbody model to observed mocap markers, while also providing a rigged skeleton that can be used in standard animationprograms (top row). Conventional mocap methods estimateskeletal joints from the markers, filtering out surface motionas noise and losing body shape information (bottom row).Original mocap markers are shown in green[7]

Chapter 3

Data Collection

Contents

3.1 Introduction

In this chapter we will analyze the data that we have and the adjustments done in the MoCap Solver code in order for us to train and use the algorithm. There are a lot of adjustments that needed to be made and a lot of trial and error situations. We were trying different combinations of data until we came with the more appropriate solution to get the most optimal result.

3.2 Data Analysis

As it was explained in Chapter 2, the AMASS database had already created the DanceDB SMPLs in a format similar to the CMU dataset that was used in the MoCap Solver paper[1]. However the MoCap Solver, being a deep learning algorithm, requires a training and testing dataset in order to optimize the SMPL. Firstly, I had to test if I can create a dataset with the already existing motion captures from the DanceDB.

3.2.1 Training Data

The optimal situation was to solve some of the Motion Captures taken in the DanceDB dataset and then create an SMPL version of them. However, this would have taken a lot of time, as the entire process requires careful consideration and handling of a lot of data. In essence, to do this process, one should go frame by frame and reposition all markers that were in wrong positions for multiple files that will be used in the MoCap Solver as Training. While this is the optimal solution, our need for something that can be done easily without the time consuming processes of re-targeting the markers was essential.

The second thought was to use the best motion captures from the DanceDB as Training Data and have the ones that needed a lot of changes as Testing. While this may have been good for the data that needed a lot of adjustments it wouldn't have given us the right solution to what we needed. This left us only with a direction to go and that was to use the CMU dataset similarly to how the MoCap Solver already worked.

Since the CMU is the dataset that the MoCap Solver was already using both for Training and Testing it was an easy adjustment for us to make. The way that it was handling the data as it is described in their paper was that they were separating the entirety of the dataset by 90% of it being Training data and 10% being Testing data. They are also adding noise to both dataset in order to

help it train and at the same time check if the algorithm works to bring the mocap to the right data. The noise was handled by a random seed parameter that was given in the data generation class. Since we didn't need any noise for our Testing dataset, the seed parameter was ignored when generating the data.

3.2.2 Testing Data

As it was described in the previous section the Testing data we used was the SMPL versions of the DanceDB datasets that were generated by AMASS. A problem that we encountered was that, the body shape that was used when generating the Training and Testing datasets was different. However, instead of forming different body shapes for the two datasets, we used the same body shapes for both. As it is described in the Limitations and Future Work chapter of the MoCap Solver paper[1], being a data-driven algorithm, the results might not be as good when it comes data that the algorithm has not been trained with. Therefore, we should have as many similarities as possible. For the body shapes we used the basic male model and basic female model that can be found in the smplpytorch project[8].

3.3 Generating Training And Testing Datasets

The training and testing dataset is essentially a decoded version of the SMPL files. When generating datasets, the algorithm creates new numpy files that will be used for training and testing. This new data, in combination with the body shape given by the basic models consist of 6 files. Starting with the two first, named M and M1, these are the cleaned markers and the raw markers respectively. Cleaned markers being the correct markers and raw being the markers with noise. The rest of the data the newly generated numpy file consist of the J_R and J_t, which represents the global rotation matrix and the global position for each joint, J, being the joint positions for the

T-pose and the marker_config that represents the local position of each marker with respect to the frame of each joint.

3.3.1 Marker and Skeleton Order

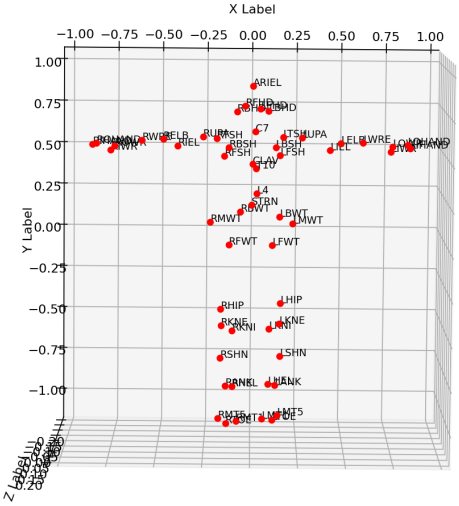
Generally, an SMPL doesn't have markers. As it was described earlier, it consists of the information of the body shape, rotation and soft tissue of the skin[5]. By adding markers to specific positions and handling a skeleton order we can easily adjust and replicate an SMPL back and forth with relation to that position. As we can see in Figure 3, the MoCap Solver uses 56 Markers and 24 Skeleton joints. These variables can be easily changed according to the input data. Since we are using data from the AMASS database there was no need to change any of the Markers used.

As it can be seen in Figure 4, there are similarities to the standard 24 Marker that we use in our Motion Captures that we got from the laboratory when we recorded the dances for the DanceDB.

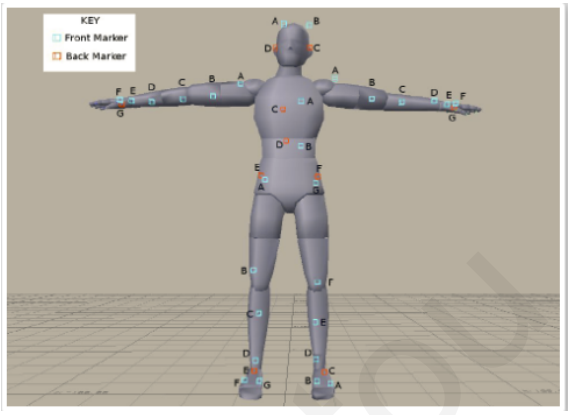
```
Marker_order = {
            "ARIEL": 0, "C7": 1, "CLAV": 2, "L4": 3, "LANK": 4, "LBHD": 5, "LBSH": 6, "LBWT": 7, "LELB": 8, "LFHD": 9,
            "LFSH": 10, "LFWT": 11, "LHEL": 12, "LHIP": 13,
            "LIEL": 14, "LIHAND": 15, "LIWR": 16, "LKNE": 17, "LKNI": 18, "LMT1": 19, "LMT5": 20, "LMWT": 21,
            "LOHAND": 22, "LOWR": 23, "LSHN": 24, "LTOE": 25, "LTSH": 26,
            "LUPA": 27, "LWRE": 28, "RANK": 29, "RBHD": 30, "RBSH": 31, "RBWT": 32, "RELB": 33, "RFHD": 34, "RFSH": 35,
            "RFWT": 36, "RHEL": 37, "RHIP": 38, "RIEL": 39, "RIHAND": 40,
            "RIWR": 41, "RKNE": 42, "RKNI": 43, "RMT1": 44, "RMT5": 45, "RMWT": 46, "ROHAND": 47, "ROWR": 48,
            "RSHN": 49, "RTOE": 50, "RTSH": 51, "RUPA": 52, "RWRE": 53, "STRN": 54, "T10": 55} // The order of markers

Skeleton_order = {"Pelvis": 0, "L_Hip": 1, "L_Knee": 2, "L_Ankle": 3, "L_Foot": 4, "R_Hip": 5, "R_Knee": 6, "R_Ankle": 7,
            "R_Foot": 8, "Spine1": 9, "Spine2": 10, "Spine3": 11, "L_Collar": 12, "L_Shoulder": 13, "L_Elbow": 14,
            "L_Wrist": 15, "L_Hand": 16, "Neck": 17, "Head": 18, "R_Collar": 19, "R_Shoulder": 20, "R_Elbow": 21,
            "R_Wrist": 22, "R_Hand": 23}// The order of skeletons.
```

Figure 3: Marker Order and Skeleton Joint Order used in MoCap Solver[1]



(a) 56 Markers of the MoCap Solver in T-Pose represented in a 3-D environment[1]



(b) The 24 Markers captured in DanceDB [4]

Figure 4: Similarities between the 56 markers of the MoCap Solver and the 24 Markers from the DanceDB Motion Captures

3.4 Generating Shape Data

As we described in chapter 3.2.2 we used the smplpytorch models to generate the shape data that will be used in the MoCap Solver. By unzipping the numpy SMPL data that come with the smplpytorch project, we extract the *maleshape* betas and create a list of json files. These will be used during the creation of the Testing and Training data in order to correlate the data to the body shape used when generating. During the process a random json file is taken and associated with the rest of the files to create the npz file for each of the training and testing data. The betas are a series of 10 numbers that represent the weight and height adjustments of the main body shape. These are added to the SMPL layer class in the MoCap Solver to adjust the variables accordingly, as it can be seen in Figure 5.

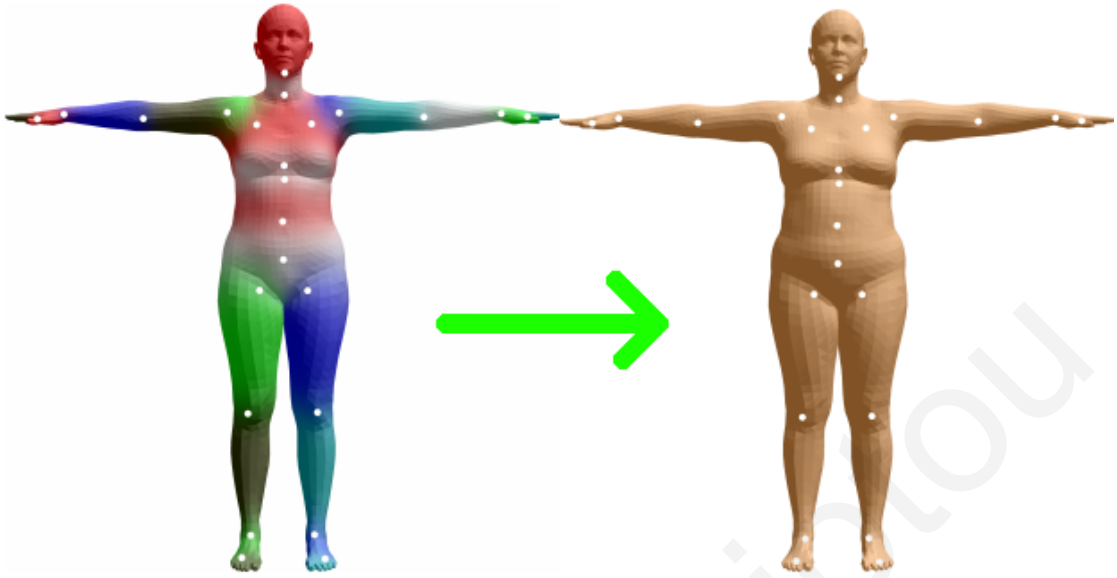


Figure 5: Body shape adjusted by betas

3.5 MoCap Solver Data

By generating the data as it was explained in the previous chapters we now should have a list of the CMU Training datasets and the DanceDB Testing datasets. With these data formatted accordingly we are now able to run the MoCap Solver and train it in order to generate the results that we want to check. The MoCap Solver provides pre-trained datasets to skip the generation of the datasets as it is time-consuming. However, since we are using the DanceDB datasets this process was unavoidable for us. Therefore we had to iterate through the process in order to find the right amount of data to handle as training. With the data being initialized we can now start with training the system so that it will be able to handle and solve the motion captures. Since the MoCap Solver uses a deep learning neural network methodology in order to fix the anomalies in the animation, we need to understand first how these kind of systems work.

Chapter 4

Methodologies

Contents

4.1 Introduction

Before going into the MoCap Solver and see how the process works we need to understand a few things about Neural Networks and specifically Deep Learning which is the main methodology used in the algorithm.

4.1.1 Neural Network

A neural network is a type of machine learning model that is inspired by the structure and function of the human brain. It is made up of layers of interconnected ”neurons,” which process and transmit information. The basic building block of a neural network is the neuron, which is a simple unit that receives input, processes it, and produces an output[9].

The input layer of a neural network takes in the raw input data, which is then processed by one or more hidden layers. The hidden layers use weights, which are numeric parameters that determine the strength of the connections between neurons, to transform the input data into a more useful representation. The output layer produces the final output of the neural network, which can be a prediction, a classification, or some other desired result[9].

One of the key features of neural networks is that they are able to learn from data. When a neural network is trained, it adjusts the weights of its connections based on the input data and the desired output, in order to produce better results. This process is called ”backpropagation,” and it involves adjusting the weights in a way that minimizes the error between the predicted output and the actual output[9].

Neural networks are particularly good at tasks that require the recognition of patterns or the extraction of features from data. They are used in a variety of applications, including image and speech recognition, natural language processing, and even self-driving cars[9].

4.1.2 Deep Learning

Deep learning is a subfield of machine learning. It involves training artificial neural networks on a large dataset, allowing the network to learn and make intelligent decisions on its own[10].

Deep learning algorithms use multiple layers of artificial neural networks to learn and make decisions. Each layer processes the input data and passes it on to the next layer, until the final layer produces the output. The layers in between the input and output layers are called hidden layers, and they are responsible for extracting features and patterns from the input data[10].

The key advantage of deep learning is its ability to learn and make decisions on its own, without the need for explicit programming. Deep learning algorithms are able to learn from large and complex datasets, and they are able to improve their performance over time as they are exposed to more data[10].

Deep learning has been successful in a variety of applications, including healthcare, finance, and many other industries to analyze and make sense of large and complex datasets, similar to the input data that we have.

4.1.3 PyTorch

The MoCap Solver uses the python library PyTorch. PyTorch is an open-source machine learning library for Python that provides a seamless integration of a powerful array computation library (NumPy) and a deep learning library. PyTorch is a popular choice for researchers and developers working on deep learning projects, because it allows them to experiment with different models and approaches quickly and easily[11].

One of the main features of PyTorch is its dynamic computational graph, which allows users to change the structure of their neural network on the fly and recompute the forward pass through

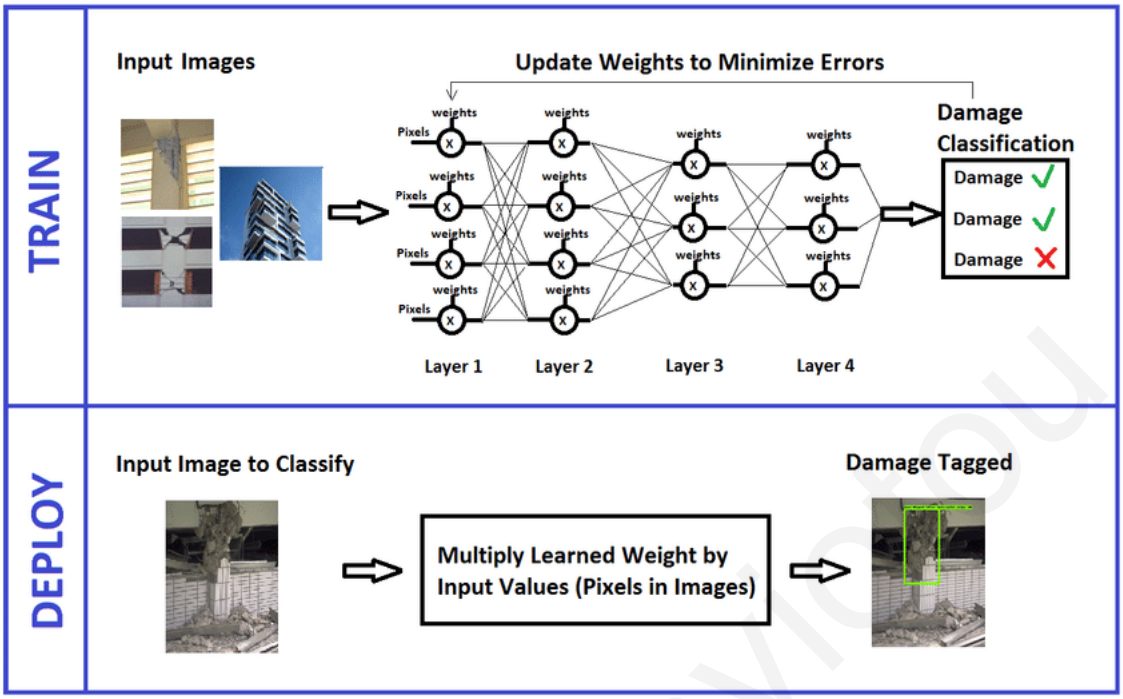


Figure 6: Deep Learning Workflow

the network. This makes it easier to debug and optimize neural networks, as well as to perform tasks such as transfer learning[11].

PyTorch also includes a number of utilities for loading and preprocessing data, as well as for performing common deep learning tasks such as training and evaluation. It is designed to be easy to use and extensible, making it a good choice for both beginners and experienced deep learning practitioners[11].

4.2 Model Definition

Simply put, with the knowledge we have for how the Deep Learning algorithms work, the MoCap Solver builds a network that uses the Training Dataset of the CMU and using that the data passed in from the Training Dataset of the DanceDB will be adjusted. In this chapter we will see exactly how the MoCap Solver extracts this information, build the neural network and how

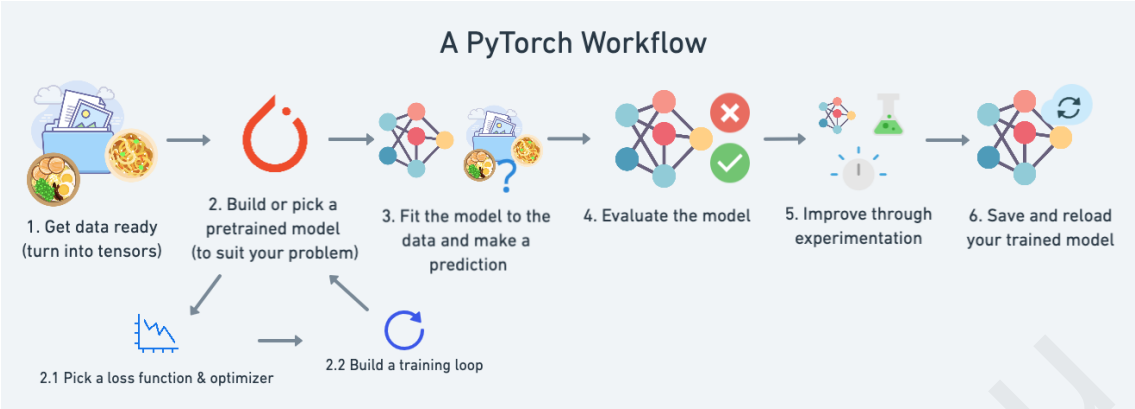


Figure 7: A standard PyTorch workflow

it trains the models for our Training data to pass in. First things first, we need to understand the process that the MoCap Solver uses to adjust and learn.

4.2.1 Observation

The process at which the MoCap Solver will generate the clean markers is based on 3 encoders. As we explained earlier on the definition of the input datasets we care about three things. The template skeleton, the marker configuration and the motion. By separating the data into these 3, creating encoders that will work together and generate a combined result, the MoCap Solver creates a learning algorithm to solve the Motion Captures of the Testing dataset.

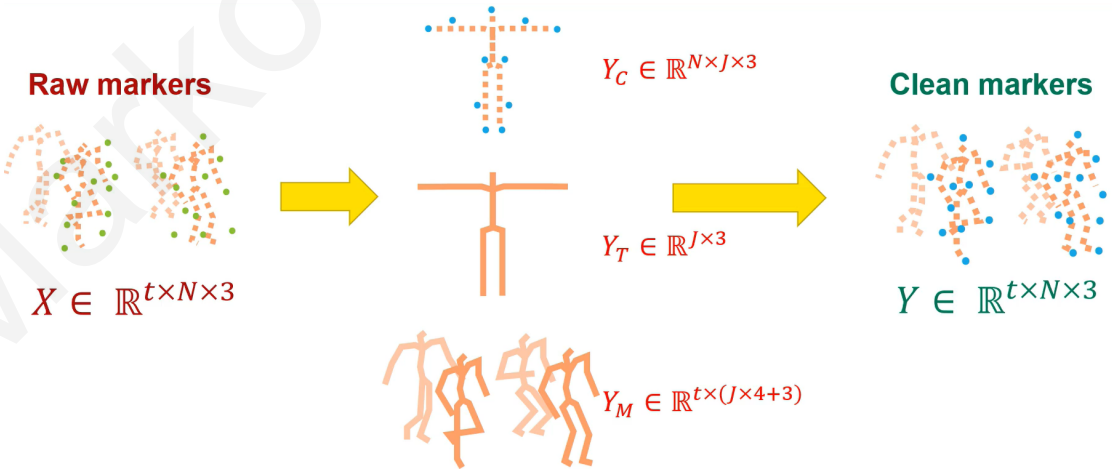


Figure 8: MoCap Solver process[1]

4.2.2 Motion Encoders

The three motion encoders that the MoCap Solver creates are based on the three vital parts of the dataset. To get a better understanding the three encoders will be the Neural Network as it was explained earlier. From the raw markers the MoCap Solver normalizes the data, passes it in the 3 encoders, does the skinning and creates a file with clean markers and skeletons.

From Figure 9 we can see that the Marker Configuration Encoder and the Motion Encoder are build with respect to the template skeleton. In essence, the template skeleton is needed for the relativity in joint rotation and and the marker offsets respectively. While the process follows the network linearly, during the training phase there is a definition of loss that is fed into the encoders in order to train them. When the encoders produce the final result, the last process is to do the skinning and generate a file that contains the appropriate data.

4.2.3 MoCap Solver

After the three components, marker configuration encoder, template skeleton encoder and motion encoder are trained, the algorithm maps the input raw markets to their respective components and decode them to produce the cleaned markers using the LBS(Linear Blend Skinning). This process generates a numpy file that contains multiple information. The numpy file consists of offsets, M, M1, J_t, J_R, RigidR, RigidT, shape, Marker, J, mrk_config, weighted_mrk_config, mc_latent_code, motion_latent, motion, first_rot, offsets_latent. While most of the information is similar to the original files the new data are the most crucial information and what should be used to get the clearer view of the results. Next, we'll see the evaluation that has been done to the MoCap Solver and correlate it to our results.

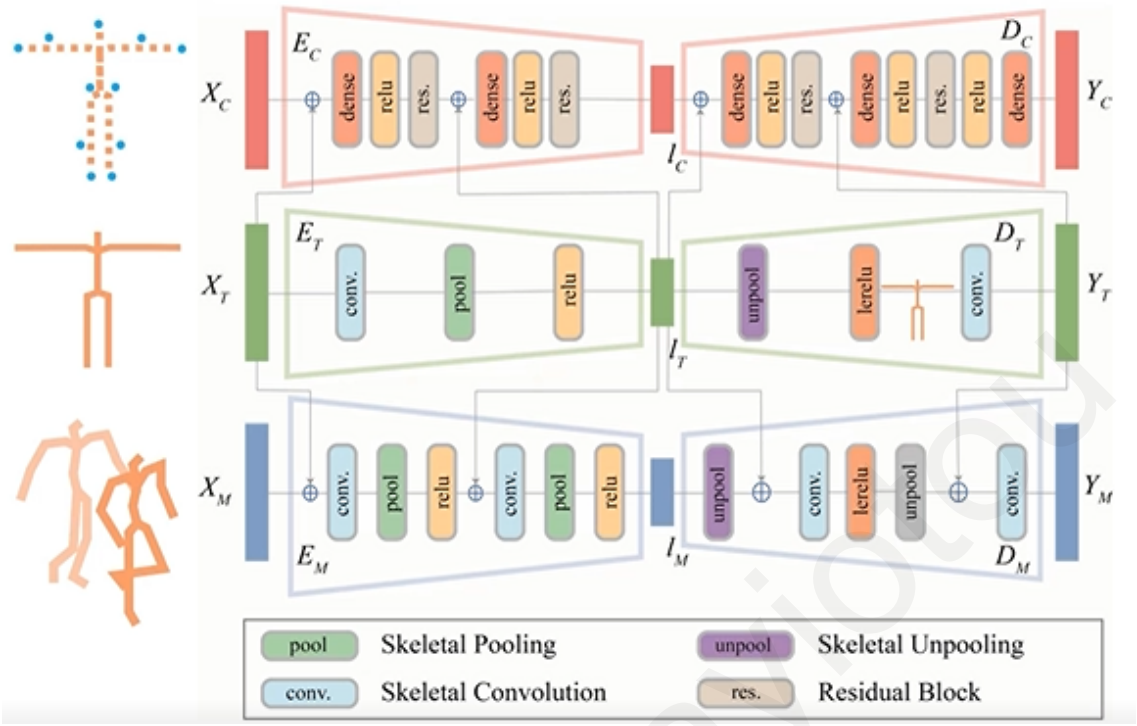


Figure 9: The three MoCap Encoders used by the MoCap Solver[1]

4.3 Evaluation of MoCap Solver

After examining the results the authors found that the reconstruction errors are small enough to be neglected. They state however that the MoCap-Encoders improve as the size of the training dataset increases. They also mention that when using completely different training sets in relation to the testing set the results may be not as good as expected.

4.3.1 Window Sizes

When it comes to the different window sizes, they have trained four versions of the MoCap-Encoders with different sizes 16, 32, 64 and 128. They found that the precision is relatively stable with window sizes up to 64, but decreases with a window size of 128 due to increased complexity and the need for more data to train the encoder. Based on these results, when we ran the MoCap Solver for the DanceDB Testing dataset we used a window size of 64.

4.3.2 Usefulness of MoCap-Encoders

The usage of the MoCap-Encoders also proved useful[1]. As stated, using MoCap-Encoders in the MoCap Solver reduces positional errors of markers and joints by 2mm, and reduces rotational errors of joints by 1.5 degrees compared to an end-to-end network that bypasses the MoCap-Encoders. They also investigated the influence of different normalization strategies on the precision of the MoCap Solver. They found that normalization is important for this problem, and that using their outlier replacement method and pose-dependent marker reliability function in the normalization process significantly improves the precision of the MoCap Solver system compared to other normalization strategies. Specifically, using both of the methods reduces positional errors of markers and joints by over 3.5mm, and reduces rotational errors of joints by 2.56 degrees compared to the other tests they did.

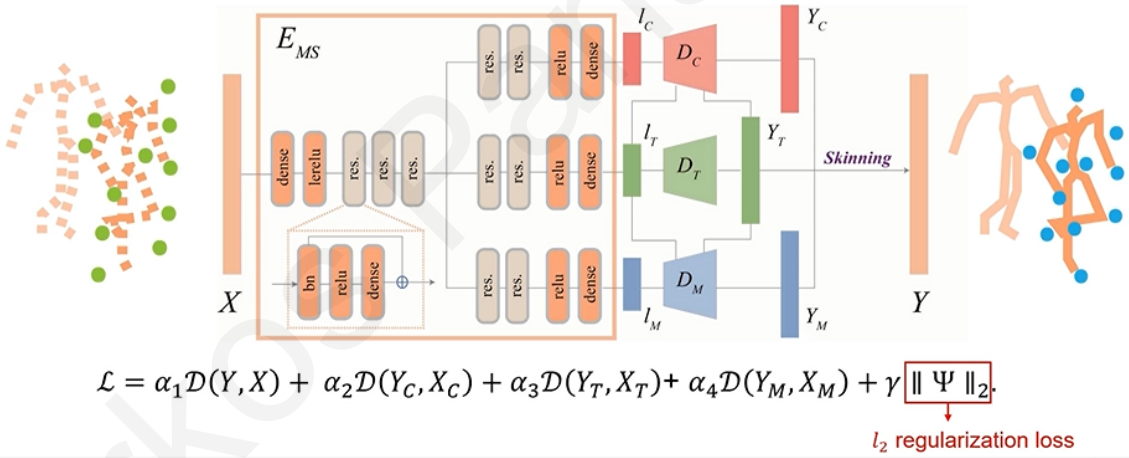


Figure 10: MoCap Solver Process[1]

4.4 Adjustments

Having all of the above in mind we made sure that we follow the same process that the authors of the MoCap Solver used. We had to follow the same process as elaborately as possible in order to

get the right results when using the same process they did. By having the Motion Encoders trained with the same exact data as they have, we will be able to see their influence on the DanceDB datasets. If this proves to have good results then the process can be used to create better Motion Captures for our data and satisfy the motivation of our experiment. In the next chapter we will evaluate the results and provide feedback for future work. We will also discuss a few of the problems faced when running the algorithm with different datasets and the specifications needed to run the MoCap Solver.

Chapter 5

Evaluation

Contents

5.1 Overview

In this section we will evaluate the performance of the trained neural network on DanceDB dataset. Generally when evaluating a program like MoCap Solver we use a subset of the Training set that will not be included in it. This is to make sure that the algorithm runs correctly on the dataset and the result is similar to the appropriate one. Substantially, we would want to check if the neural network is correct but that has already been done using the CMU Dataset[1]. By evaluating the Testing Data with the DanceDB dataset we expect to see higher amounts of errors, due to the

differentiation of the two datasets. But it is important to recognize the errors and analyze if the Motion Encoders can help solve the issue.

5.2 Error Analysis MoCap Solver

Error analysis is a process of identifying and understanding the sources of error in a neural network, which can help to improve its performance. This can be done by comparing the model's predictions on the training and testing datasets. The training dataset is used to train the model, and the model's error on this dataset is known as the training error. The testing dataset, on the other hand, is used to evaluate the model's performance on unseen data. The model's error on the testing dataset is known as the testing error[9].

If the training error is much lower than the testing error, it could indicate that the model is over-fitting to the training data, which means it is not generalizing well to new, unseen data. This can be addressed by increasing the amount of training data, using regularization techniques, or simplifying the model. If the testing error is much higher than the training error, it could indicate that the model is under-fitting, which means it is not able to capture the complexity of the data. This can be addressed by increasing the model's capacity, using more complex features, or using a different model architecture. By analyzing the error on the training and testing datasets, it is possible to identify and address sources of error in a neural network and improve its performance[9].

By analyzing the error on the training and testing datasets, it is possible to identify and address sources of error in a deep learning model and improve its performance.

5.2.1 Error Analysis

After evaluating the results the algorithm displays the error of the testing dataset based on the training set. Since we set the training set to the CMU dataset we expect to be higher than the

results that we can see in the MoCap-Solver paper[1]. It's not really a direct comparison since the datasets they used to evaluate Holden 2018[14] and their algorithm was the CMU datasets. When generating their data they separated them by 90% of the entirety of the dataset as training and 10% as testing. As we explained earlier the algorithm works best for sets that has been trained with. This means that the error will definitely be higher when using 2 different datasets. In Table 1 and Figure 11 we can see that after evaluating the testing dataset it produced a Marker Position Error of 26.48mm, a Joint Orientation Error of 9.46 degrees and a Joint Position Error of 21.30mm.

Datasets	MPE	JOE	JPE
CMU[1]	12.77mm	6.47○	13.66mm
CMU(Training), DanceDB(Testing)	26.48mm	9.46○	21.30mm

Table 1: Error evaluation results. (MPE: Marker Position Error, JOE: Joint Orientation Error, JPE: Joint Position Error)

```
0/186.  current mean error:  marker: 27.304(mm)  angle: 9.632(deg) pos: 19.198(mm)
10/186.  current mean error:  marker: 28.873(mm)  angle: 9.649(deg) pos: 23.636(mm)
20/186.  current mean error:  marker: 27.247(mm)  angle: 9.792(deg) pos: 21.888(mm)
30/186.  current mean error:  marker: 28.066(mm)  angle: 10.040(deg) pos: 22.198(mm)
40/186.  current mean error:  marker: 26.060(mm)  angle: 9.991(deg) pos: 20.632(mm)
50/186.  current mean error:  marker: 25.850(mm)  angle: 9.888(deg) pos: 20.579(mm)
60/186.  current mean error:  marker: 26.013(mm)  angle: 9.844(deg) pos: 20.699(mm)
70/186.  current mean error:  marker: 25.857(mm)  angle: 9.755(deg) pos: 20.611(mm)
80/186.  current mean error:  marker: 26.255(mm)  angle: 9.759(deg) pos: 21.036(mm)
90/186.  current mean error:  marker: 26.176(mm)  angle: 9.681(deg) pos: 21.098(mm)
100/186.  current mean error:  marker: 26.004(mm)  angle: 9.542(deg) pos: 20.918(mm)
110/186.  current mean error:  marker: 25.939(mm)  angle: 9.513(deg) pos: 20.834(mm)
120/186.  current mean error:  marker: 26.038(mm)  angle: 9.481(deg) pos: 20.940(mm)
130/186.  current mean error:  marker: 26.187(mm)  angle: 9.507(deg) pos: 21.068(mm)
140/186.  current mean error:  marker: 26.133(mm)  angle: 9.476(deg) pos: 21.031(mm)
150/186.  current mean error:  marker: 26.238(mm)  angle: 9.504(deg) pos: 21.143(mm)
160/186.  current mean error:  marker: 26.384(mm)  angle: 9.509(deg) pos: 21.229(mm)
170/186.  current mean error:  marker: 26.379(mm)  angle: 9.500(deg) pos: 21.248(mm)
180/186.  current mean error:  marker: 26.478(mm)  angle: 9.487(deg) pos: 21.325(mm)
Total mean error:  marker: 26.481(mm) angle: 9.461(deg)  pos: 21.303(mm)
```

Figure 11: Results as shown in PyCharm when evaluating the results

Firstly we need to note that the evaluation results shown for the CMU Training and Testing is from the MoCap-Solver paper[1]. They used synthetic data for training and real Motion Capture data for testing. Our results, we used similar training dataset but the testing dataset we replaced it with the DanceDB dataset, therefore the margin of the errors is expected.

Furthermore, we decided to test the algorithms' limits when removing some of the values of the X position from our testing data. The 3 Markers we chose to remove the information were the LBWT(Left Back Waist), the LWRE(Left Wrist) and RHIP(Right Hip). This way, we will be able to see the prediction of the algorithm according to the pose, shape and the location of the marker on the Y and Z axis. We would also be able to compare the X-position values between the ones when it previously had an X-position and when we set it to 0. In the next chapter we will see graphs that show the differences for the markers before and after passing through the MoCap Solver.

When we ran the evaluation on the files that had the X-position set as 0, we could see that the total error had increased. The Marker Position Error went up to 30.026mm, the Joint Orientation Error to 10.406 degrees and the Joint Position Error to 24.732mm. This is a fairly, small change when it comes to the bigger picture

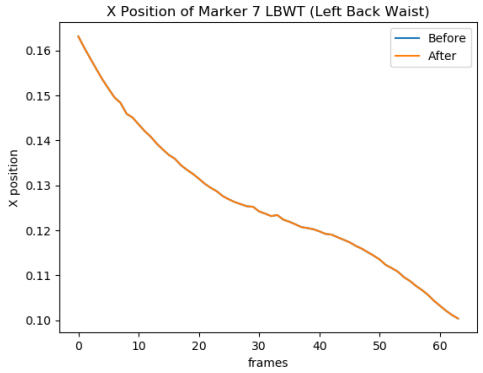
Datasets	MPE	JOE	JPE
CMU(Training), DanceDB(Testing)	30.026mm	10.406◦	24.732mm

Table 2: Error evaluation results. (MPE: Marker Position Error, JOE: Joint Orientation Error, JPE: Joint Position Error) This is the error on the files that we changed the 3 Markers' X position set to 0

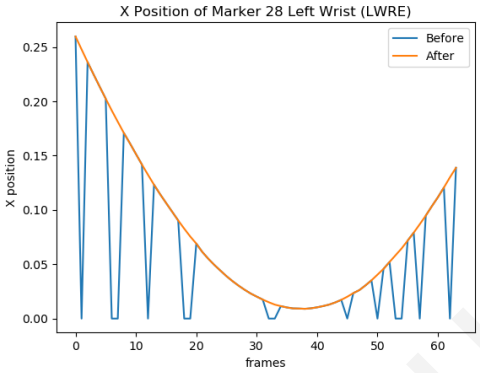
5.2.2 Error Analysis on Marker Position

In this chapter we will check what actually happens to the markers when they pass through the MoCap Solver. Specifically, the 3 Markers that we previously said we would set to 0, are

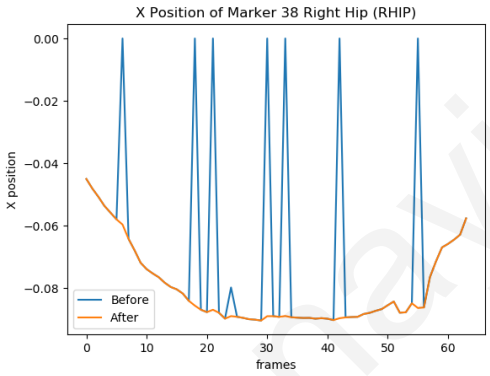
the ones in the Left Back Waist (LBWT), Left Wrist(LWRE) and Right Hip (RHIP). We chose these 3 to demonstrate markers with different kinds of movement. We can expect the Left Back Waist to not have drastic changes to its value as the Left Wrist would have. In the next graphs we will see 3 comparisons specifically for the X position of the files. The first set of graphs will show the changes of the raw markers as they were, to demonstrate how the MoCap Solver works when adjusting the previously added marker. The second set will show the changes when the raw marker's X position was set to 0, to see if the deep learning system was able to predict the location when missing information. Finally, we will compare the two results, the one without the information of the X value on raw markers and the one where X position had its original value. We should note that for these results we got the same set of 64 frames for all the markers. These 64 frames are from the 9720 to 9784 frame of the dance Zeibekiko (Greek) by Vaso Aristeidou[15].



(a) X Position of Left Back Waist



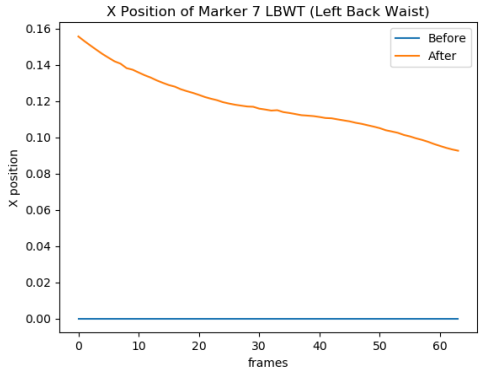
(b) X Position of Left Wrist



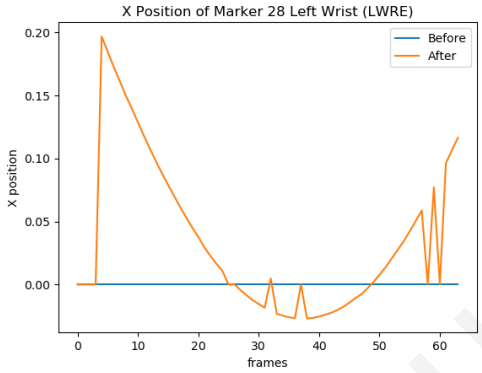
(c) X Position of Right Hip

Figure 12: Changes of the markers before and after passing through the MoCap Solver without changing the original marker location

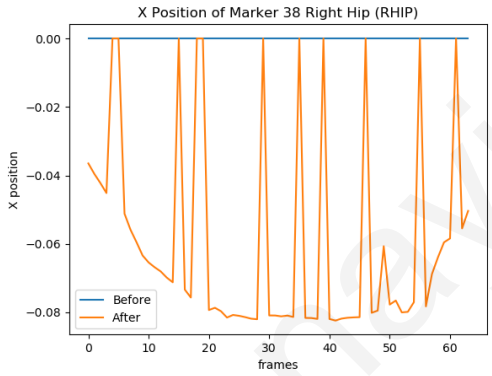
We can see that, for Fig.12 (a) there weren't any changes between the clean and raw markers when it comes to the Left Back Waist marker. The raw markers of the Left Wrist and Right hip show some disturbances with markers going back to 0 and continuing on the original line. The MoCap Solver was able to solve the results by predicting the locations to where the marker should have been. This shows that the MoCap Solver worked well even when the raw input had high disturbances with missing markers or falsely located markers.



(a) X Position of Left Back Waist



(b) X Position of Left Wrist

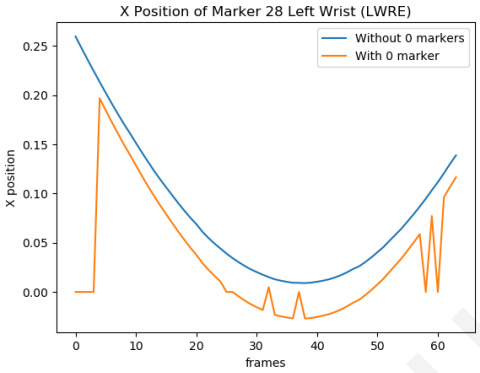


(c) X Position of Right Hip

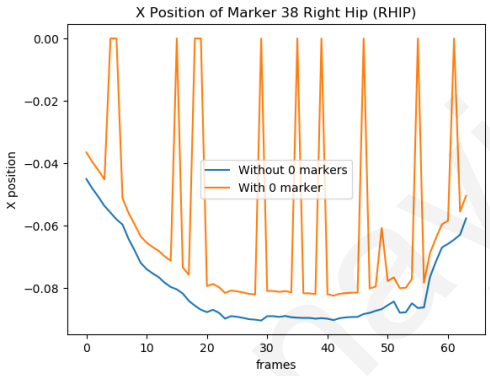
Figure 13: Changes of the markers before and after passing through the MoCap Solver when changing the original marker location of X to 0

In these 3 graphs we can see that the algorithm was able to predict the location of the X even when in the raw markers the value was 0. Especially in the Left Back Waist that didn't have big changes it predicted it flawlessly. However, we can see that for the Left Waist and Right Hip the predictions were not as stable as we would hope. While the MoCap Solver was able to predict the X-location for multiple of these frames there are some disturbances going back to 0 and again back continuing the previous line normally.

(a) X Position of Left Back Waist



(b) X Position of Left Wrist



(c) X Position of Right Hip

Figure 14: Comparison between the results of the MoCap Solver when setting the X-location to 0 and when having the raw markers as is

Examining the Fig.14(a) we can see that there is a significant change to the values of X between our two results. This is a recurring issue in all 3 of the result comparison graphs as we can see that the results are significantly better when the raw markers are as they were captured. Especially in the Left Wrist and Right Hip graphs Fig.14(b,c), the prediction of the non-0 raw marker input is stable and follows a seemingly correct line. On the other hand, the algorithm was unable to correct and predict the 0 values that we set in the raw markers but it follows a right structure.

5.2.3 Addressing the Errors

As explained in the previous section and by examining the marker tracking analysis, the error being higher is expected. The algorithm, like other data-driven methods, is limited in its ability to produce accurate results for cases that are completely ”unseen” in the training set[1]. In order to improve the performance of MoCap-Solver and make it more versatile, it is necessary to build a larger MoCap dataset that includes a greater diversity of human shapes and poses. This will allow MoCap-Solver to be more adaptable and better equipped to handle a wider range of inputs. In short, the next step in improving the performance of MoCap-Solver is to expand the scope and diversity of the training dataset. We could also change some of the training data to contain 0 in some values so that the algorithm will learn to fix even missing errors, as we can see this is something that it lacks after examining Figures 12-14.

Chapter 6

Conclusions, Limitations & Future Work

Contents

6.1 Conclusions

In this chapter we will discuss and show the results. We will see how the algorithm has solved the Motion Captures of the DanceDB. There was a significant change on how the animations perform after passing through the deep learning algorithm. We will also analyze the inconsistencies

in some of the movements and then we will see what changes can be done to use the MoCap Solver appropriately and expect better results. Additionally, we will analyze the limitations that we faced during this research.

6.1.1 Blender

Blender is a 3D graphics software that allows users to create high-quality 3D content for a variety of purposes. Blender has a comprehensive set of tools for creating and editing 3D models, including primitive shapes, deformers, and sculpting tools. It also has a robust set of animation tools that allow users to create keyframe-based animations, rig characters, and create non-linear animations. Blender has advanced rendering capabilities that support ray tracing, global illumination, and physically-based rendering, and it includes a built-in game engine for creating interactive 3D applications and games. Blender has a set of sculpting tools that allow users to create detailed 3D models, and it has a range of compositing and post-processing effects that allow users to enhance the look and feel of their 3D graphics and animations. Blender is used by professionals in various industries and is also popular among hobbyists and 3D enthusiasts.

Blender has the ability to use custom scripts in Python to generate objects, add animations to them and many more. We will use this custom scripting to demonstrate the differences between the old markers and the adjusted by the deep learning algorithm.

Since the result file of the MoCap Solver are in a numpy file, as it was explained in Chapter 4.2.3, we will use some of that data to generate an animation in Blender showing the previous markers and the ones after passing through the algorithm. Therefore we need to extract the markers in the form of one text file per marker for all markers that we want to check. Since the algorithms uses 56 markers, as we saw in the SMPL Chapter, this means we will need 112 text files containing the location of each marker, in X, Y and Z, per frame.

6.1.2 Demonstration in Blender

We have created a Python Script in Blender that reads the 112 text files, one for each marker from the raw motion capture and one for each marker from the cleaned dataset. Then it creates 1 blue sphere in the 3D environment for each raw marker and 1 red for all the adjusted ones and then sets their location as the one in the first frame from the datasets. Then for each key frame of each marker we give them a different location, thus generating an animation. By running the script then we should have an animation at the ready to demonstrate the results.

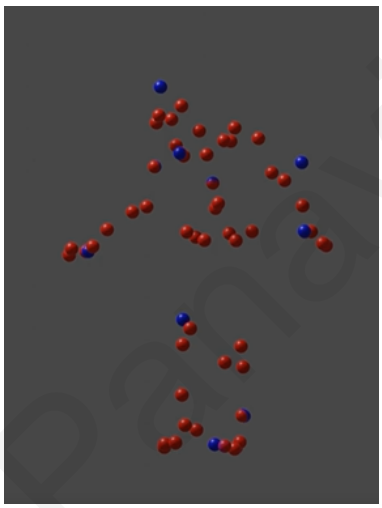


Figure 15: Pose 1: Actor about to make a dance turn

6.1.3 Comments

The poses, as demonstrated in the blue and red markers, show the differences between the 2 datasets. As we can see in Poses 1,2 and 3, the visible blue markers are outside of the standard body shape. This shows that these markers have false information and will be occluded when applied on an actor.

Pose 1 demonstrates the actor in the process of making a turn around in a dance. We can see that there are multiple issues with the raw markers. We can see the blue marker at the right side

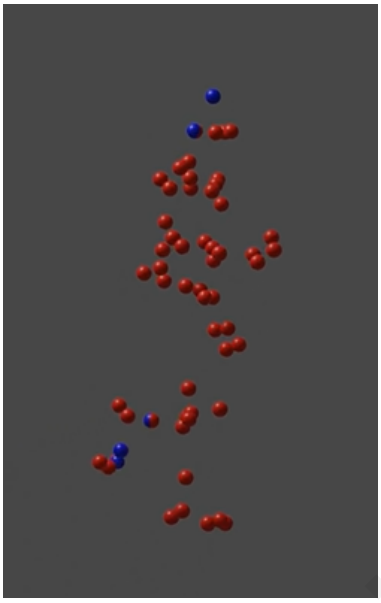


Figure 16: Pose 2: Actor in the motion of running

of the sternum that is occluded compared to its red counterpart. That one and the one on the right hand seem to have a small error margin. However when inspecting the ones on the right hip, left elbow and left foot we can see that there is a big difference there.

Pose 2 demonstrates the actor in the motion of running. Here there aren't many issues with the motion capture. The notable one is that on the left foot, as the error there seems to be far from its' counterpart in red. The one on the head, is close to the red one but it is occluded from the camera view.

Pose 3 demonstrates the actor standing in a pointing upwards pose. The most notable errors on this animation is the one in the hand which is way off where it had to be and we can see that the solving was done right on that one since the red counterpart seems to be in its right place. The ones on the head also seem to have an issue but when perceived in the 3D environment they are not far away from their counterpart. We can see that the rest are close to their equivalent red marker but to the overall scene we can see that the error exists and that the clean markers are more complete to the pose.

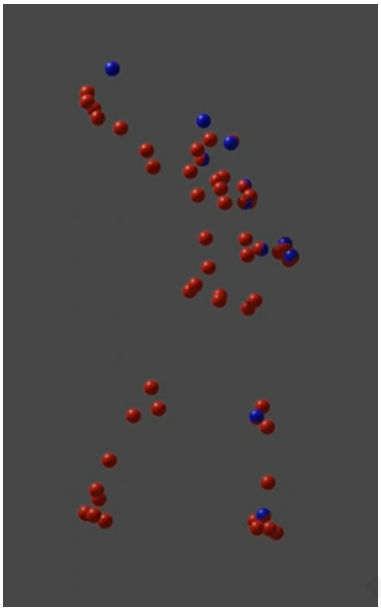


Figure 17: Pose 3: Actor pointing something upwards

6.1.4 Final Results

By analyzing the poses and examining the Figures 12-14 we can understand that there is room for improvement to achieve our motivation. We can understand that the algorithm worked relatively well, even with a dataset completely unseen in the training set. This can also be due to the reason that both datasets used are from the SMPLs that AMASS developed. However, the animations are far from perfect. We will see in the Future Work chapter later, some adjustments that can be made in order to increase the performance of the algorithm and generate better results.

6.2 Limitations

One of the main limitations we had in this process with generating the datasets was the time needed to create them. Being time consuming and having to test different methods to create the optimal universe of data for the process to work with, it took a lot of the time we had for this

project. The system we were working with has an NVIDIA RTX 3080 10GB VRAM GPU, an 11th generation CPU 11700K and 32GB of DDR5 RAM.

The process took a lot longer than expected. Generating the datasets by transforming the SMPL data into the right structure took approximately 30-35 hours. The entirety of the process of the MoCap Solver took 90-100 hours. Since we had to go back and forth changing some of the data given this took a lot of time to come to the right conclusion. Due to the limited time we had when working on this research it was much harder to experiment with different datasets.

Having these in mind, we will see in the Future Work chapter some reworks that might need to be done for better results on the Motion Captures.

6.3 Future Work and Hypotheses

After examining the results of the datasets we can think of various ways to make the performance of the algorithm better and make the results more suitable to what we want. We saw the impact that the CMU dataset had on the DanceDb datasets after the algorithm was trained by the first and tested on the second. The bottom line though, is that we can safely say that there is high probability to fix the occlusions in our target motion captures and more generally use the MoCap Solver to solve even more datasets if needed.

6.3.1 Future Work

One of the main issues as we saw during our research was the inconsistency between the training and testing dataset. As the authors of the MoCap Solver paper[1] described and knowing how the deep learning algorithms work[10], these kinds of algorithms don't work well with data that are totally unseen during the Training phase of the algorithm. The solution to this problem is to manually solve some of the Motion Captures and use those in the training dataset. We need to

note, that to do so, it will take a lot of time, as the manual solve of an animation requires a person to go through every frame and follow the steps of calibration, cleaning, re-targeting and refining as we explained at the beginning of this research.

Another thing to note is that with the continuous rise in the technology of this field more way to solve motion captures will be developed. It is also worthy to note that with the new Graphics Cards and CPUs being released with even better performance and better resources, the tools that will make the algorithm work even better is expected.

6.3.2 Hypotheses

One hypotheses that can be made, since we saw that the results of this research were satisfying enough, is to use the method we described to create a valid training set for the rest of the DanceDB datasets. This means that we will use the CMU dataset similar to what we did during this research to create clean motion captures of the DanceDB. Then we will use the clean motion captures as training to solve more of the same dataset.

This will require a lot of time as we saw in the Limitations chapter. However if we use better equipment and resources this might make the time needed to generate datasets and use the deep learning algorithm even less.

Bibliography

[1] Kang Chen, Yupan Wang, Song-Hai Zhang, Sen-Zhe Xu, Weidong Zhang, and Shi-Min Hu. 2021. MoCap-solver: a neural solver for optical motion capture data. ACM Trans. Graph. 40, 4, Article 84 (August 2021)

[2] G. Cheung, T. Kanade, J.-Y. Bouguet, and M. Holler, *A real time system for robust 3d voxel reconstruction of human motions in Computer Vision and Pattern Recognition*, 2000. Proceedings. IEEE Conference on, vol. 2. IEEE, 2000, pp. 714–720

[3] https://graphics.cs.ucy.ac.cy/lab

[4] Aristidou, A., Shamir, A. and Chrysanthou, Y., 2019. Digital dance ethnography: Organizing large dance collections. Journal on Computing and Cultural Heritage (JOCCH), 12(4), pp.1-27.

[5] Loper, M., Mahmood, N., Romero, J., Pons-Moll, G. and Black, M.J., 2015. *SMPL: A skinned multi-person linear model. ACM transactions on graphics (TOG)*, 34(6), pp.1-16.

[6] Loper, M., Mahmood, N. and Black, M.J., 2014. MoSh: Motion and shape capture from sparse markers. ACM Transactions on Graphics (ToG), 33(6), pp.1-13.

[7] Mahmood, N., Ghorbani, N., Troje, N.F., Pons-Moll, G. and Black, M.J., 2019. AMASS: Archive of motion capture as surface shapes. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 5442-5451).

[8] https://github.com/gulvarol/smplpytorch

[9] Bishop, C.M., 1994. Neural networks and their applications. Review of scientific instruments, 65(6), pp.1803-1832.

[10] LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. nature, 521(7553), pp.436-444.

[11] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. and Desmaison, A., 2019. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32.

[12] Patterson, B. Leone, G. Pantoja, Maria Behrouzi, Anahid. (2018). Deep learning for automated image classification of seismic damage to built infrastructure.

[13] Aristidou, A. and Lasenby, J., 2013. Real-time marker prediction and CoR estimation in optical motion capture. The Visual Computer, 29(1), pp.7-26.

[14] Holden, D., 2018. Robust solving of optical motion capture data by denoising. ACM Transactions on Graphics (TOG), 37(4), pp.1-12.

[15] http://dancedb.eu/main/performances