Group 95: Reproducibility Project Towards Single Camera Human 3D-Kinematics

Course Info

CS4240 Deep Learning (2022/2023 Q3)
TU Delft

Supervisor:
Xucong Zhang, xucong.zhang@tudelft.nl

Students:
Farhad Azimzade 4788206, F.Azimzade@student.tudelft.nl
Adriaan Keurhorst, 4550994, A.F.Keurhorst@student.tudelft.nl
Floris Pauwels, 4606000, F.Pauwels@student.tudelft.nl

In this project we try to reproduce results from the paper
Towards single camera human 3D-kinematics Sources:
https://www.mdpi.com/1424-8220/23/1/341

We use the provided code the authors shared on GitHub:
https://github.com/bittnerma/Direct3DKinematicEstimation

Table of Contents

Project Overview

Topics: Biomemenic model, OpenSim, Human Pose

Results to reproduce: Augment the testing data to check for robustness of the model

This blog aims to reproduce the deep learning research by Bittner, M., et al. (2023) on markerless estimation of 3D Kinematics (3DKE). 3D human kinematics relates to measuring joint angles between different parts of the human body. These joint angles are vital for professional physicians to give precise advice to, for example, athletes to perform better in their respective sport. This information can also aid in diagnosing neurodegenerative diseases more efficiently. Accurately analysing joint angles of a subject requires extensive training of the human evaluator, as it involves fine distinctions in the visual eception of the subject, as well as some expert knowledge. As such, efforts have been made in automating the process of such kinematics analysis.

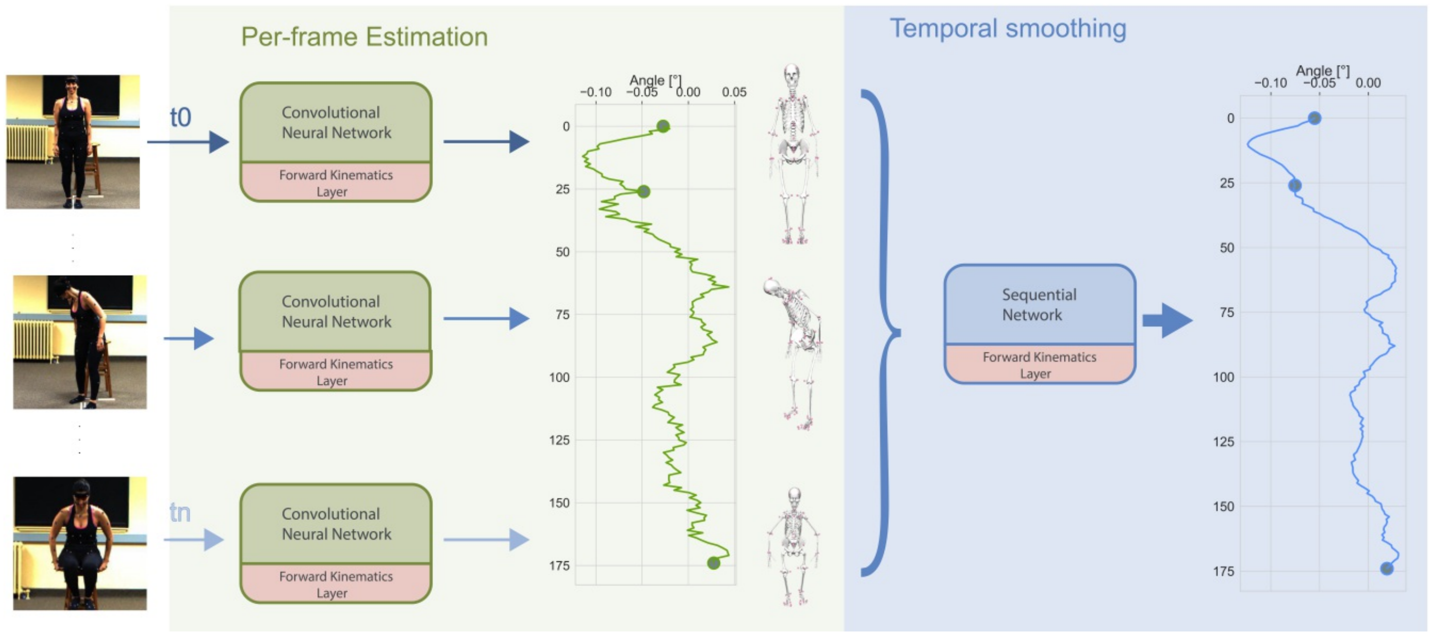
One such approach is the multi-step approach. This approach predicts the 3D pose, fits it to a model, and estimates the kinematics. Applied to a dataset of videos of various human subjects, the multi-step approach has already been shown to be highly performant in the estimation of human kinematics. However, most kinematic estimation algorithms cannot correct for the errors in the estimated pose because they all make use of the multi-step approach, or variations thereof.

In the paper by Bittner et al., an alternative to the multi-step approach for human kinematics is presented, which involves direct application of Deep Learning in an end-to-end fashion on the video dataset to estimate human kinematics. The algorithm presented in this paper shows great potential for usage in combination with mobile phones, and has been shown to be an improved approach compared to the multi-step approach.

The algorithm presented in this paper uses deep learning algorithm which directly learns from a video to joint angles and scales using deep neural networks.

Theory

The method presented in this paper, simplified, goes as follows: it takes videos from a single camera as input and directly estimates joint angles. This is done by using a convolutional neural network (CNN) for each individual frame for instantaneous kinematic estimations, and then using a sequential neural network with temporal relations to relate the instantaneous kinematic estimations with the temporal relations of the frames to refine the output. The general method is shown in the following image:



A standard pre-trained ResNeXt-50 is used as the convolutional backbone, with three different sequential networks tested. These sequential networks are LSTM, TCN and a Transformer.

The overall objective function is

$$L = \lambda_1L_{joint} + \lambda_2L_{marker} + \lambda_3L_{body} + \lambda_4L_{angle}$$

with λ_1, λ_2, λ_3, λ_4 as weights of the losses.

The training requirements are the joint angle and the scales of individual bones, a rotation matrix of the pelvis to the ground as well as the marker positions corresponding to them. Joint angles are generated using the OpenSim software. The resulting ground truth values are the calculated joint angles, the scaling factors and the virtual marker positions.

The ResNetXt algorithm used has the following hyperparameters:

- Adam optimizer with weight decay of 0.001
- Batch size = 64
- Learning rate = exponentially decays in two steps from 5 × 10−4 to 3.33 × 10−5 over 28 epochs and from 3.33 × 10−6 to 10−6 over 2 epochs

The sequential and convolutional networks have lambda values of:

- λ_1 = 1.0
- λ_2 = 2.0
- λ_3 = 0.1
- λ_4 = 0.06

These values were determined experimentally by the authors of the paper.

Dataset

The 3DKE algorithm was trained and tested on the BML-MoVi Database (https://www.biomotionlab.ca/movi/). The database contains 90 actors performing 20 different kind of everyday movements, and also a random movement. The motions of the actor's body parts were captured using inertial measurement units, combined with a Qualisys optical motion capture system (https://www.qualisys.com/). Two camera mounting positions are used, PG1 and PG2. The camera view captured by PG1 is the frontal- and PG2 as the sagittal camera view. The ground truth virtual markers were generated by the 3D mesh representations of the Qualisys data that is provided in the AMASS dataset. AMASS is a large database of human motion unifying different optical marker-based motion capture datasets by representing them within a common framework and parameterization. AMASS is readily useful for animation, visualization, and generating training data for deep learning (https://amass.is.tue.mpg.de/).

Method

The reproduction project was aimed at exploring the robustness of the D3KE method to input data augmentation. In particular, we would like to evaluate the performance of the D3KE algorithm on an augmented testing data set.

The data augmentation we considered were flipping, resizing, cropping, altering the brightness, changing the contrast of the frames in the input videos. Such augmentation is usually applied to the training set, as part of inducing robustness toward certain changes of the input. In fact, augmentation of the training data was performed in the training of the original D3KE model, as is stated in the paper.

Data augmentation can also allow the training dataset to effectively increase in size. When applied to images, a mirrored image of a cat is still a cat. Applying mirroring to videos of people, however, changes things when the model is created of the person in the video. A mirrored video of someone moving their left arm will show someone moving their right arm. Augmenting the test set and thereby creating 'new' testing data will exploit several aspects of the algorithm.

Testing the algorithm will give insights into its generalisability. The study *Do ImageNet Classifiers Generalize to ImageNet?* by Recht B. et al. has shown that presenting different ImageNet classifiers with new but comparable testing data, decreases the accuracy, indicating that the test set has been relied upon too much in the training of the algorithm.

This study will aim to do something similar, but with videos. By presenting augmented versions of the original test set, the algorithm's generalisability will be explored.

Installation Requirements

Installation requirements to run the algorithm presented by the paper are the following:

- Python 3.8.0
- PyTorch 1.11.0
- OpenSim 4.3+

The authors of the paper created an environment file with the necessary libraries to install.
In addition, they provided an installation guide-line to:

1. Install and activate the correct conda environment
2. Download the source datasets from the provided links
3. How to unpack the resources
4. How to generate and prepare the dataset
5. How to download, run, evaluate, and train the models

Problems Encountered

Errors from Running the Original Code

It was difficult to get the correct datasets in the correct folders, as there were many files from multiple sources to download and put into the correct folders in the project. There was no system like a *checksum system* that checks whether all files are complete and correctly stored. With a bit of trial-and-error we got this working.

The *generate_opensim_gt.py* and the *prepare_dataset.py* had to be changed to be able to run them.

The first error we got from running *generate_opensim_gt.py* was:
[warning] Couldn't find file 'videoMuscle_sacrum.obj' This warning can be solved with:

```
npzPathList=gtGenerator.traverse_npz_files("")
```

This solution was found by a fellow student on *Mattermost* after we inquired about the issue.

A second error in this script can be suppressed by adding:

```
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
```

See https://stackoverflow.com/questions/74217717/what-does-os-environkmp-duplicate-lib-ok-actually-do for more info on the issue.

The next error was "RuntimeError: std:: exception in 'bool OpenSim::ScaleTool::run() const'" in *prepare_dataset.py*. A solution was once again found on Mattermost, where the paths need to be set as absolute, e.g.:

```
sys.path.append(str(Path(__file__).absolute().parent / "ms_model_estimation"))
parent_dir = Path(__file__).absolute().parent
```

See https://github.com/opensim-org/opensim-gui/issues/448 for more info on this issue.

In addition to changing the scripts to remove errors, some paths needed to be changed to make the code use the correct files, such as adding the following line to the *prepare_dataset.py* file:

```
outputFolder = "D:/_dataset"
```

This code depends on where you want the dataset stored. Beware that the processed dataset contains ~550GB of data. We needed to reprocess all files multiple times as we were often short of data. More on that in the next section.

We also once had an unknown error that we fixed by creating a new environment and retracing all steps. This may work for some errors.

With the *run_inference.py* and *run_training.py* files, we had the following error:
DLL load failed while importing _multiarray_umath: The specified module could not be found.
We might have been able to fix this error in time, but time was getting short, and we knew the original plan was out of reach with the constant accumulation of unforseen issues with running the code. We instead chose to create code that makes the data more manageable.

Issues with Efficiency

A GPU can increase processing speed by more than 10x. The code does not automatically use the GPU, so in each file that we ran we needed to include the following code:

```
import torch
COMP_DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Now the code automatically uses the GPU if there is one installed for GPU specific tasks.

Still, the processing speed was slower than expected, even with the GPU enabled. On the original GitHub, it is stated that the processes of the *generate_opensim_gt.py* and *prepare_dataset.py* code "might take several hours!".
Our experience was on a high-end PC with a *AMD Ryzen 9 5950X* processor and a *Nvidia GTX 1070*, and it took **several days** for these processes to finish with GPU enabled. We assume the original authors processed the data on a powerful server, instead of a high-end PC. This was not mentioned in the README.

As said before, the *prepare_dataset.py* creates an additional total file size of over 500 GB of storage.
This created an issue for us, as we were out of storage multiple times, as there was no mention nor indication how big the eventual dataset would be.
An additional problem was that the code does not check if the files are already correctly processed, but reprocesses all files again when the code is re-run. This took a lot of time and computing, as each re-run took a couple days.

An easy way to skip files that are already processed in the *generate_opensim_gt.py* script is done with the following code:

```
for path in npzPathList:
    if os.path.exists(path):
        continue
    gtGenerator.generate(path)
```

A similar solution can be used to skip files from reprocessing in the *prepare_dataset.py* script. A more robuust method that also checks the integrity of the files is to use a checksum system and check if the hashes correspond. This system is added to the GitHub files as the *hash_check.py* and *hash_create.py* files.

Cloud Computing Issues

Given that the original research project made extensive use of GPUs for developing the model, deploying GPUs for the reproduction appeared to be a reasonable step. In order to achieve this, a popular cloud computing platform, Google Cloud, was decided upon. The general plan involved running a virtual machine with access to GPUs on the cloud to augment the data and re-test the model on the augmented data. However, there have certain roadblocks we have run into with respect to this approach.

Firstly, the allocation of physical resources on Google Cloud turned out to be more challenging than expected. It has taken a while to arrive at a server that met the needs, despite our requirements being relatively limited. This has taken a while to resolve, as the allocation has to be manually performed on individual regional servers.

Secondly, there have been issues when attempting to build the project on the cloud virtual machine instance. In particular, certain packages prescribed in the conda environment file could not be installed by conda. These conda environment packages appeared to halt the creation of the environment. After numerous attempts, this issue has not been resolved.

As a result of the afore-mentioned issues, cloud computing was not utilised for the project due to time efficiency considerations. Instead, the work on the data generation for the model was performed locally.

Results

We encountered several errors during the process of reproducing the original results. These errors originated from incompatible code and insufficient hardware. This is why we were unable to reproduce the original results, and abandoned our original plan to test the generalizability and robustness of the model by data augmentation.

Due to the issues we faced, we decided to create a hashing checksum system that:

1. Creates hashes of the files in specified folders (*hash_check.py*)
2. Uses these hashes to check if all files are stored in the correct location (*hash_create.py*)
3. Check if a file is already processed or generated, and skips these files from being unnecessarily recreated (Rewrite *generate_opensim_gt.py* and *prepare_dataset.py* as described above)

These systems make it easier to find which files are missing, and removes the issue of reprocessing files that are already processed. This can greatly improve the efficiency of the code if some files have to be reprocessed, for example when the PC is out of storage.

Individual Contribution

Farhad Azimzade: Cloud computing / Report
Adriaan Keurhorst: Research / Report
Floris Pauwels: Reproducing / Hashing