AI-Based Video Motion Capture

Joseph Aguilar
joseph.aguilar@yale.edu
Yale University
New Haven, Connecticut, USA

Figure 1: Joseph Aguilar performing in a musical, motion captured by the methods described, 2025.

Abstract

This paper details a method for generating human motion capture data using only visual input. This method utilizes a pipeline of different convolutional neural networks to read in photo and video data and generate 3D coordinates aligned with the body parts of the people in the photo/video. This paper is a synthesis and extension of two other papers, [1, 2]. This paper recycles code and methods from these papers into an automatic, speedy, and efficient pipeline, achieving similar results and speed solely on CPU, rather than utilizing GPU.

The method, using 3 convolutional neural networks, is able to process through videos at a rate of one frame every two seconds and generate accurate results.

Keywords

CNN, MoCap, Video, Visualization, Graphics, GPU

ACM Reference Format:
Joseph Aguilar. 2025. AI-Based Video Motion Capture. In . ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

As a Computing and the Arts major on the theater track, I approached this project from the perspective of of a theater director. I have wondered about the possibilities of introducing advanced computing techniques into the realm of live technical theater. Computing and live performance is a relatively unexplored cross-section. This is mainly because computing techniques until relatively recently have had a hard time interacting with a visual medium such as a live performance. With the rise of computer vision techniques and AI algorithms, like convolutional neural networks, that make visual processing more feasible, this cross-section is able to be better explored.

Many video games and movies utilize motion capture technology to create realistic "digital twins" of humans. A "digital twin" in performance is the synthesized recreation of human movement through special effects. This allows these mediums to create and animate fantastical-looking characters with ease and realism. However, for these mediums to achieve good results with this technology, expensive, high-maintenance motion capture suits and studios are necessary. This prohibits lower-budget productions and studios from being able to use this technology. In the context of theater, "digital twins" would be a very interesting mode of performance for many actors and technicians, but this idea is limited by the necessity of sensory cameras and a suit. When watching a hypothetical performance utilizing this "digital twin" effect, the suit and the cameras and the studio severely limit the amount of creative freedom and look of the live portion of the show. From this, there exists a need for a less hardware-dependent method for motion capture.

In this paper, I utilize previous approaches to this problem and adapt their work for streamlined use on custom video input.

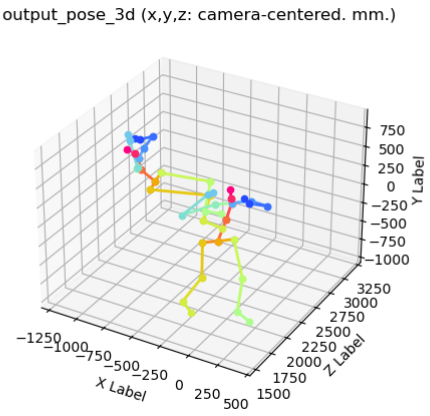

Figure 2: Figure 1 on a 3D plot of the pure coordinates generated by my method. Notice that the model captures the cut-off portion of a person in the background and attempts to map limbs to them.

2 Related Work

As mentioned above, this paper is a synthesis of two other papers. This project utilizes code and pre-trained models from these papers, as their complexity and accuracy are both already very high. Being able to synthesize these papers' work allowed me to focus more on the scope of the project: the motion-capture of video data. When I was first working on this project, I figured that the first place to jump to most accurately reflect motion capture through purely visual input would be through 3D pose estimation of humans. This drew me to a paper by Gyeongsik et al., Camera Distance-aware Top-down Approach for 3D Multi-person Pose Estimation from a Single RGB Image [1]. This paper contained the code-base from which I derived much of this project. This paper introduced two neural networks, RootNet and PoseNet.

2.1 RootNet and PoseNet

RootNet is a convolutional neural network capable of taking in an image of a human and locating a "root" point in 3D space from a 2D picture. More simply, it estimates the distance from the camera that the human is standing, estimates the X and Y coordinate for their center of mass or "root" point, and then places their root point in a 3D coordinate space. This is necessary, since when PoseNet needs to generate the pose data for the human, it needs to be able to place the joints in a distinct place in the 3D space, especially when multiple people are in the picture.

PoseNet is the second convolutional neural network introduced in this paper. This one takes in the root point from RootNet and the visual input, then maps a series of joints to the different parts of the person's body. Both of these neural networks were trained on a blend of publicly available datasets, but the main datasets used were Human3.6M (a collection of 3.6 million photos of humans in studios matched to 3D coordinate data attained by sonar data), MSCOCO, and MuPoTS-3D.

For custom use, and especially custom video use on lower-budget hardware, both of these neural networks had limitations. First of all, these neural networks were programmed to make use of high-end GPUs. For lower-budget studios or amateur artists who only have access to a laptop or similarly inexpensive equipment, this makes these models inaccessible for use. Another is that these models required 2D coordinates of bounding boxes drawn around each human in the picture. The paper itself mentions the use of their own DetectNet model for this task, but this model provided the coordinates in a slow and clunky manner: first writing the coordinates to a JSON file, then reading them into the models. To solve this, I turned to another paper, You Only Look Once: Unified, Real-Time Object Detection [2] by Redmon et. al.

2.2 YOLO for Bounding Box Detection

Redmon et. al. details in their paper a "You Only Look Once" approach to object classification from image data. Instead running multiple passes of a classification model and cross-checking the guesses for accurate object identification, the paper runs a regression model over an image only once, and generates probabilities of different object classes for each object identified in the photo. This convolutional neural network is trained to output the dimensions of a bounding box around each identified object in the image, along

with the different probabilities of it being different objects. The YOLO neural network has many different classes of objects that it has been trained on, so it is able to assign a probability to an object from each class. The YOLO models assigns a class to a detected object by the highest probability. This model is much faster and suited for this specific task, as it has a direct coordinate output for each object which you are able to sort by classification. It also doesn't require multiple, slow passes to classify an image. It's a single model that is able to have a quick output off one pass.

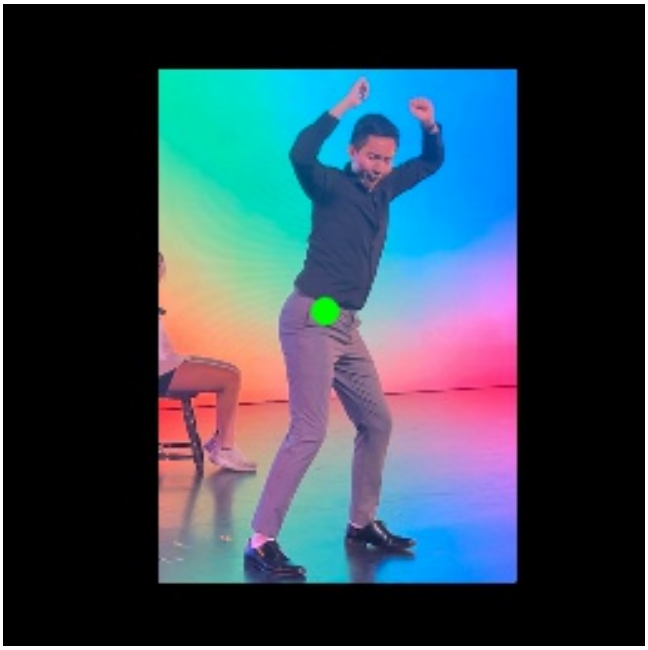

Figure 3: The first part of the output of RootNet on the image of Joseph Aguilar dancing.

3 Methods

To start, here are the limitations and goals that I had in mind:

- I wanted to run one script and have the "motion-capture" data of a given input video.
- I was working with neural networks designed for image data, not video.
- I was working off a code-base of neural networks that I was unable to use because they required GPUs, and my laptop only had CPU.
- Despite not using GPUs, I wanted similar speed and accuracy from only using CPU.
- To use PoseNet to get poses, I first needed root depths from RootNet. To use RootNet to get root depths, I first needed bounding boxes

Knowing this, I started working on the YOLO object classification neural network to make sure it was able to provide useable bounding boxes for the next two neural networks.

YOLO was not trained on GPUs, so it did not require them. I ran a rudimentary weighted snapshot that was provided by the paper

Figure 4: The second part of the output of RootNet on the image of Joseph Aguilar dancing. It managed to find the person sitting in the back.

```
>>> Using CPU
Root joint depth: 1932.1128 mm
Root joint depth: 2763.571 mm
```

Figure 5: The distances assigned to each root point by RootNet. It placed Joseph closer to the camera at a depth of 1932, where the person in the back is at a depth of 2763.

[2] on my own images. This means that I used the weights of their model after a certain iteration of training that they had previously done. This provided me with hundreds of different bounding boxes for each possible object in the frame. I modified the output to only provide bounding boxes for each objec that was given the "human" class by YOLO. This limited the bounding boxes, but I still had a list of boxes far longer than amount of people in the photo. YOLO does not have an understanding if it has already identified an object in the photo, so it will often provide multiple bounding boxes for a single object. I wrote a script that went down the list of bounding boxes, compared the dimensions, and merged their dimensions into a bounding box of best fit if they overlapped too much. This finally got me a consistent and accurate number of bounding boxes for each human in a given photo.

Next, I scrubbed the tools and code provided by Gyeongsik et. al. for use of GPUs, and converted them to have a CPU option. If any functions required CUDA or NVIDIA GPU, I simply made an alternate to it with standard torchvision functions that did not

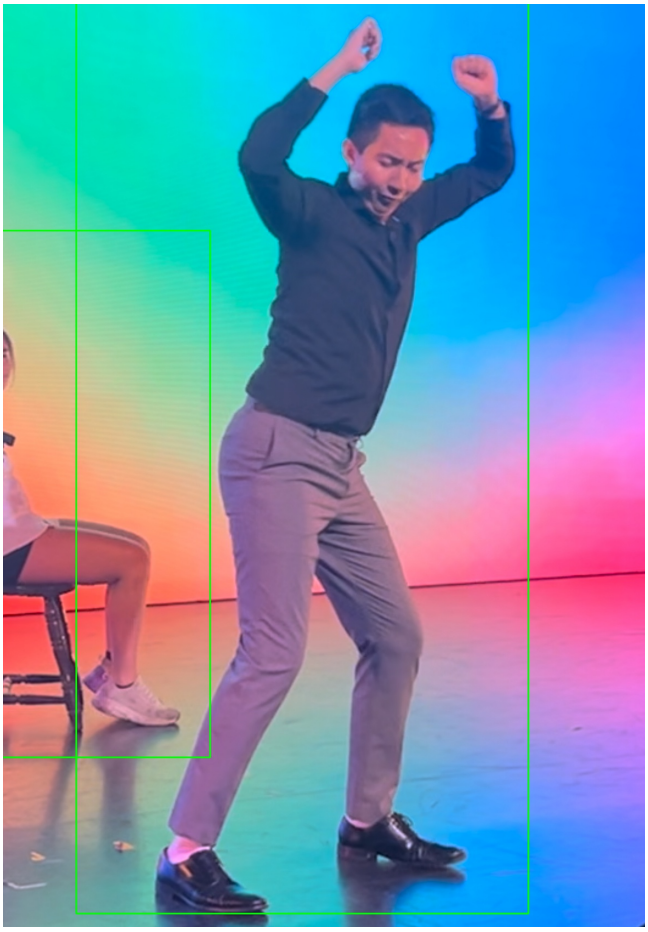

Figure 6: Figure 1 on a 3D plot of the pure coordinates generated by my method. Notice that the model captures the cut-off portion of a person in the background and attempts to map limbs to them.

require them, as there was often a GPU-less equivalent. Then, I put these into a series of if statements, checking if the user was running PyTorch with CUDA enabled and if the user was using any GPUs. If they weren't the program switched to using CPU.

Now that I was able to use PoseNet and RootNet without GPU, I wanted to synthesize Gyeongsik et. al.'s code-base into one script containing all the processes necessary for pose extraction. This meant taking a lot of their scripts and code and turning them into a series of functions, whose inputs and outputs were able to interact with each other automatically. I also fit YOLO with bounding box consolidation into this pipeline to provide the bounding boxes. After this, I had a series of functions that, when run sequentially, would provide pose data on any photo with bounding boxes, roots, and pose all automatically generated. This process in the original papers was divided into multiple scripts and different processes, or in some cases needed manual input or data read from another file (bounding boxes).

Table 1: Speed of GPU vs. CPU

Processing Unit	Avg. Time per Frame (seconds)	Total Time (s)
2 GPUs	2.77	58.14
1 GPU	2.72	57.03
CPU	2.01	42.24

Next, I need to run this process on video instead of solely image input. This was a simple iterative process, where the functions above would all read in the same frame, perform their processes on it, extract the pose data from one frame, then reassemble it into a video with the poses outlined. The code for visualizing the poses on the video was lifted from [1] and modified to fit in the pipeline. If I had wanted, I could also have it log the points sequentially into a file for use in a separate animation program. To make this speedy without the use of a GPU, a few corners were cut and assumptions were made. I assumed the dimensions of the video would not change for the duration of the video, allowing me to only load the dimensions of the video once before looping over the frames. I also changed each function so that I only had to load the frame once. I once I loaded the frame, the data was copied to each function, instead of loading the frame each time each function was run. This allowed me to save processing power and time rendering the photo for each function. I also loaded each model once, since I figured reweighting before each frame would not be useful and would only waste time and processing power. Once these adjustments were made, the speed of running on CPU was comparable to the speeds of the GPU, sometimes faster. The accuracy of the poses was comparable as well. I elaborate on this in the results section.

4 Results

4.1 Speed

I tested the GPU version of my script on the Yale Grace Cluster, using two NVIDIA GPUs for comparison. In Table 1, you will see the timing comparison of the different processing units on a short video with only 24 frames and one person on screen.

As you can see, it's a surprising time difference of GPU vs CPU, even making the speed modifications for both versions of the script. I believe this is because GPUs initially take a longer time to initialize and start when being used. CPUs are built-in, and can be accessed on demand. When working on a very short video (less than a second), this initial time difference would make the CPU seem faster. I hypothesize that on longer videos with more people to process on the screen, GPUs would be faster long-term than CPUs.

I was also curious about how the number of people on screen would affect the speed of the script. I ran the program on similarly lengthed videos of 450 frames (about 15 seconds) with one person on screen, three people, and a large varying amount amount of people that changes throughout the video. In Table 2, you can see the resuls of that test.

As you can see, the number of people influences the speed at which the frames are processed. This makes sense, as the neural networks are now having to make multiple outputs per frame instead of just one, which impacts the speed of making the video.

Table 2: Speed vs. People

Number of People	Avg. Time per Frame (seconds)	Total Time (s)
1	2.68	1224.07
3	3.54	1619.04
Many	4.29	1932.65

4.2 Visual Accuracy

Overall, the neural networks are able to generate very accurate pose data over the course of a video. The next few figures are dedicated to showing screenshots of the varying tests, but you can see a more accurate representation of my work in video form in output videos folder. I noted a few things from the output that, in the future, I would like to fix:

- The poses jitter in unnatural ways, as the PoseNet neural network does not remember the pose from the previous frame.
- The skeletons in the 3D space jitter unnaturally back and forth, as the RootNet neural network does not remember the depth that the skeletons were placed in the previous frame.
- All of the neural networks are able to recognize humans too well, even when they cannot see the full body. This results in point clouds of people as the neural networks struggle to map a skeleton on an incomplete body. It also clouds the data with points that are not from the subject of the video.

I am impressed with the robustness of the models being able to capture any and all humans in the photo, even if they are not the subject or they are heavily obscured. I am also surprised the models are robust to different dimensions of video, as well resolutions of video. The quality and size of the video do not affect the quality of the predictions too heavily.

5 Discussion

I am glad to know that video-based motion capture is possible. In the hands of a talented but underfunded 3D animator, I imagine this technology could greatly enhance and/or speed up their creations. I am also happy with modifications I made to the neural networks to make them accessible to those who do not have access to high-end GPUs and still achieve similar speeds and performance. While the focus was making an accessible form of motion capture for those without the equipment to make it possible, I imagine this technology would also be useful in the world of professional filmmaking as well. If this method were to be enhanced by high-end GPUs long term and made more efficient and faster, I imagine that movie studios could use this technology on set to get a test run of how the characters and 3D models look in the special effects world before they start official filming, motion capturing, and post processing the footage. This would save directors a lot of time and headache if they could see a rough estimation of how the final product might look live on set. I am also excited to see how I can use this in theater productions that I direct, as motion capture technology in the context of live theater is very interesting to me.

Figure 7: A screenshot of the point skeleton mapped onto the person from the end of the one-person test video, with it capturing a second person in the background.

6 Future Work

As mentioned before, I noted a few things from the output of the models that, in the future, I would like to fix. In addition, I would also like to improve the speed in some way. Here is a list of future possible extensions.

- The poses jitter in unnatural ways, as the PoseNet neural network does not remember the pose from the previous frame. One solution I thought of was making poses of best fit, where the pose from one frame and the estimated pose from the next frame are merged together into points of best fit. This makes transitions smooth and the jittering less prevalent.

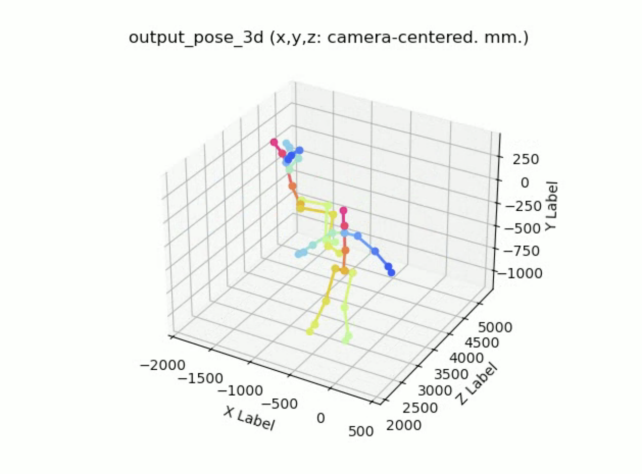

Figure 8: A screenshot from raw point data generated from the one-person test video, with it capturing a second person in the background.

Another solution is to take a pose every 5th frame or so and "tween" between the poses. That is, generate smooth transitions between the frames. This will eliminate the jittering, look very close to accurate and natural, and cut down on processing time since there are less frames to process.

- The skeletons in the 3D space jitter unnaturally back and forth, as the RootNet neural network does not remember the depth that the skeletons were placed in the previous frame. I think the solution would be similar for the jittering poses: I think making a root depth of best fit from frame to frame would solve this, giving RootNet a de facto memory.
- All of the neural networks are able to recognize humans too well, even when they cannot see the full body. This results in point clouds of people as the neural networks struggle to map a skeleton on an incomplete body. It also clouds the data with points that are not from the subject of the video. I imagine this comes from YOLO's probabilistic model making educated guesses at incomplete bodies and RootNet and PoseNet being confused with bounding box input. I gave YOLO a generous threshold for what it considers human, so finetuning its guesses to only get the most certain humans in the photo may solve this.
- Ultimately, I'd like to find ways to further make these neural networks faster. The goal is to get prediction times down to less than a four-tenths of a second, which is the length of one frame of a video shot in 24 frames per second, the standard frame rate of most cinematic cameras. This would allow this model to work on real time footage with good accuracy, and would open up possibilites of working this into a live performance.

Figure 9: A screenshot of the point skeletons mapped onto the people from the end of the three-person test video.

Acknowledgments

To the cast and crew of the Yale production of The Guy Who Didn't Like Musicals in the fall semester of the 2024-2025 school year, for providing great test footage for this project and allowing me to be in a very fun production. To the Yale Alley Cats a cappella group, for also providing fun test footage for this project and being a great group of people to sing with. To CPSC 479, for being a great class and letting me have a very fun semester.

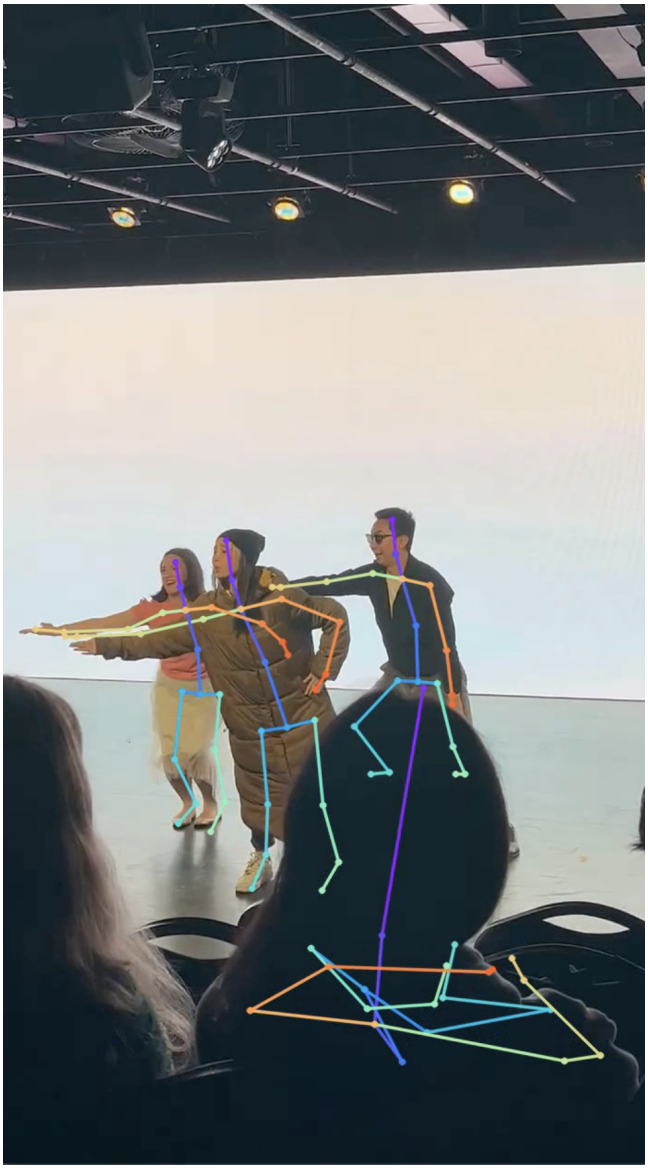

Figure 11: A screenshot of the point skeletons mapped onto the people from the end of the group test video. Notice the strange point cloud as the neural network struggles with the person sitting the foreground, and the inaccurate leg length of the person obscured by the person in front.

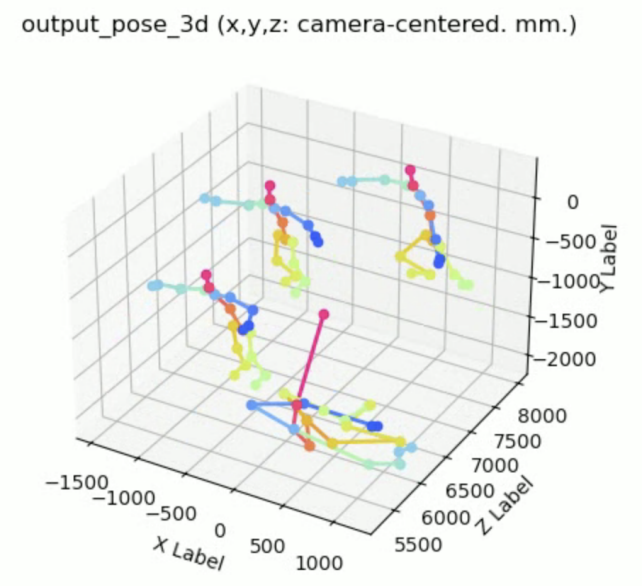

Figure 12: A screenshot from raw point data generated from the three-person test video.

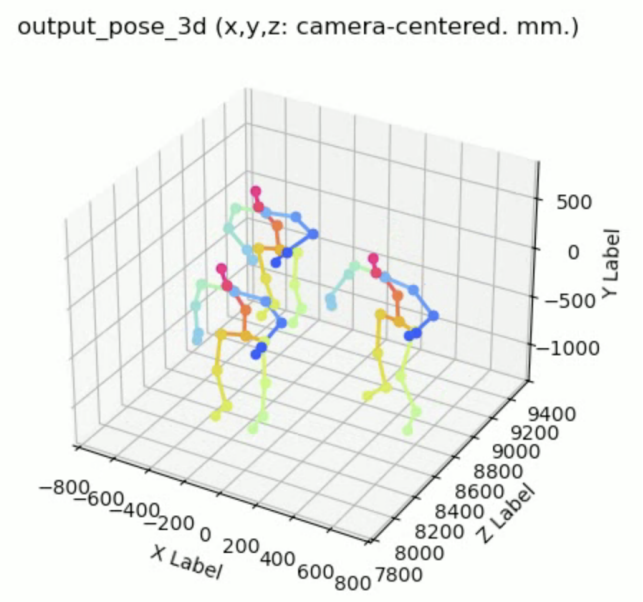

Figure 10: A screenshot from raw point data generated from the three-person test video, with an inaccuracy of the root depths from RootNet

References

[1] Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. 2019. Camera Distance-aware Top-down Approach for 3D Multi-person Pose Estimation from a Single RGB Image. https://arxiv.org/abs/1907.11346

[2] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. 779–788 pages. https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf