CS231a Final Report: 3D Reconstruction in Blender using Stereoscopic Vision

Kegan Kawamura
Stanford University
Aero/Astro
kegank@stanford.edu

Neha Konakalla
Stanford University
Computer Science
nkona@stanford.edu

Sudeep Narala
Stanford University
Electrical Engineering
sudeepn@stanford.edu

Abstract

Given that human pose estimation has numerous applications in robotics, self-driving vehicles, and even motion capture for animations, we aim to create an accurate, affordable method for 3D pose estimation. Specifically, we work on going from a pair of stereo images of a single human in a particular pose to a rendering of a 3D model with the same pose. Key points are collected by using OpenPose, a 2D pose estimation library that detects up to 25 keypoint coordinates of key human joints in an image. We then utilize a non-linear method for triangulation from each pair of stereo keypoints, optimizing the 3D position estimate from the 2D coordinates, based on our solved projection matrices. Upon obtaining the 3D position estimates for the 25 joints, we pass these 3D coordinates into Blender, a powerful open-source modeling tool, in order to render 3D models of a human using Python. We demonstrate the pipeline with our own stereo, iPhone photographs and provide successful reconstructions in Blender, even with occluded points. We then experiment with using our pipeline on a stereoscopic pair of videos, which highlights some of the challenges encountered in 3D reconstruction from stereoscopic videos in general. We demonstrate that this technique requires several improvements as it does not account for noisy data and the propagation of key points with time, resulting in inconsistent reconstructions between consecutive video frames. Future work may include improving our triangulation algorithm, handling occlusions more precisely, and improving the qualitative accuracy of bone rotations within the Blender rendering.

1. Introduction

Human pose estimation is a technology that has wide ranging applications. It can be used to predict or classify actions of people in images, and could lead to a greater understanding of human behavior and motion. Applications of human pose estimation can be utilized in human-robot interactions and improving safety of autonomous vehicles. There has been a lot of focus on 2D human pose estimation from images, including OpenPose and other keypoint detection methods, but not as much for 3D human pose estimation from images.

Stereoscopic systems employ two cameras from different angles that take an image of an object at the same time. By identifying the image coordinates of key points in each image, and knowing the relative position and orientation of the cameras, it is possible to triangulate the 3D position in space of the key point.

By combining 2D human pose estimation models and concepts from stereoscopic vision, we would like to recreate the 3D pose of a person through a pair of images, and ultimately extend it to recreating 3D motion through pairs of videos. This study could lead to affordable methods for motion capture of people, without the use of cumbersome gear such as motion capture suits and multiple cameras.

2. Background/Related Work

2.1. OpenPose

In the first stage of our project pipeline, we will be using OpenPose [1] to detect keypoints of human joints in our image. OpenPose is the first realtime approach to detect the 2D pose of multiple people in an image (and video) via keypoints for joints. OpenPose is a bottom-up approach to capture the coordinates of each joint, meaning that the pipeline calculates each body joint first and then joins them together as a skeleton to create a pose estimation. When an image is fed into OpenPose, first the baseline convolutional neural network (CNN) extracts the feature map from the image. Then, the feature map is fed into another multi-stage CNN pipeline to obtain Confidence Maps, a 2D belief estimation representation of body points being located at certain pixels, and Part Affinity Fields, which are 2D vector fields encoding coordinates and orientations of limbs in the image. More specifically, one CNN obtains the Part Affinity Fields, which are then passed into another CNN to obtain the Confidence Maps which are then optimized using an L2 loss function to ground truth fields and maps. OpenPose has outperformed prior state of the art methods in 2D keypoint detection, including DeeperCut [6] and ArtTrack [5].

2.2. Triangulation

2.2.1 Theory of Triangulation methods

Although there are several methods for triangulation, we will employ a non-linear method which involves solving a minimization problem:

$$\min_{\hat{P}} ||M_1\hat{P} - p_1||^2 + ||M_2\hat{P} - p_2||^2 \quad (1)$$

The above equation attempts to find the best 3D position estimate $\hat{P}$ from the given projection matrix M_i and the corresponding 2D image coordinate p_i. The subscripts indicates the specific camera used to take the image. If we define an error function $e_i = M_i\hat{P} - p_i$, then we can restructure this problem as a linear least squares problem. We want to find an update parameter δ_P to iteratively update our estimate of $\hat{P}$.

$$\hat{P}_{i+1} = \hat{P}_i + \delta_P \quad (2)$$

We would like $e(\hat{P} + \delta_P) < e(\hat{P})$. Therefore, if we linearize our error function we can approximate $e(\hat{P} + \delta_P) \approx e(\hat{P}) + \frac{\partial e}{\partial P}\delta_P$. This leads to to a typical linear least squares minimization problem:

$$\min_{\delta_P} ||J\delta_P - (-e(\hat{P}))||^2 \quad (3)$$

$$\begin{aligned} e &= \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \\ &= \begin{bmatrix} M_1\hat{P} - p_1 \\ M_2\hat{P} - p_2 \end{bmatrix} \\ J &= \begin{bmatrix} \frac{\partial e_1}{\partial \hat{P}_1} & \frac{\partial e_1}{\partial \hat{P}_2} \\ \frac{\partial e_2}{\partial \hat{P}_1} & \frac{\partial e_2}{\partial \hat{P}_2} \end{bmatrix} \end{aligned}$$

Our solution to this minimization problem then becomes

$$\delta_P = -(J^TJ)^{-1}J^Te \quad (4)$$

To initialize this, we can solve a linear version of triangulation, which takes advantage of the fact that $p(MP) = 0$. With two image points in homogeneous coordinates, $p_1 = [x_1y_11]^T$ and $p_2 = [x_2y_21]^T$, this leads to the following system of equations:

$$\begin{bmatrix} x_1M_1^3 - M_1^1 \\ y_1M_1^3 - M_1^2 \\ x_2M_2^3 - M_2^1 \\ y_2M_2^3 - M_2^2 \end{bmatrix} P = 0 \quad (5)$$

M_i^j is defined to be the j-th row of projection matrix i. P can be solved for easily through singular value decomposition.

2.2.2 Solving for Projection Matrices

The projection matrix M maps 3D points in the world frame to 2D image coordinates, and is constructed in the following manner:

$$M = K\begin{bmatrix} R & T \end{bmatrix} \quad (6)$$

K is the camera matrix, R is the rotation matrix defining the change in coordinates from the world to camera frame, and T is a translation vector to account for change in origin between the camera and world reference frame. For our two cameras, we assume that both camera matrices are known and the same, and that the world reference frame is defined to be camera 1's reference frame. Therefore, we still need to find the projection matrix of the second camera by solving for R_2 and T_2. We can do this through the Essential Matrix E.

Given at least eight correspondences between images, we can use the eight-point algorithm to solve for the fundamental matrix F. Then, our essential matrix is simply $E = K^TFK$. We can compute four different pairs of R and T from this. Taking the singular value decomposition of $E = U\Sigma V^T$, the candidates for R and T are

$$\begin{aligned} R_1 &= (\det UWV^T)UWV^T \\ R_2 &= (\det UW^TV^T)UW^TV^T \\ T_1 &= U\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ T_2 &= -U\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ W &= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

The proof for this is somewhat cumbersome and will be excluded. After reconstructing every 3D point in the frame, the RT pair that produces the most 3D point reconstructions with positive z-coordinate values will be chosen as the correct pair.

2.3. Blender

Blender is an open-source software which can be used for animations or modeling. [4] We will be using Blender to model the human from their pictures.

Since we want to automate the modeling of the human, we will be using bpy, a Python library to interact with Blender. Bpy provides all the functionality we will be using in this project, which involves manipulating a rendering a model into an image. The model can be create (a one-time process) using the Blender GUI. However, as discussed later, we will be using a model we found online.

Blender has many features, but for this project we are mostly concerned with the idea of *bones* and *pose mode*. One of the main use cases of mode **??**bones in Blender is is to mimic human bones. For example, if your upper arm bone (shoulder joint), moves, your lower arm moves alongside it. We use a model that has bones in it to get human-like movement from the model. The bones initial positions are configured in edit mode, but they can be modified in pose mode. Using bpy set in pose mode, we can move the bones (effectively moving bones along joints) in order to mimic a real-life person.

One more thing to note is the different own coordinate systems within which we will have to work. Each bone has its own local coordinate system. The coordinate system's y-axis points in the direction of bone length (in edit mode so it might look a little off in pose mode) and a roll which determines the orientation of the x-axis and z-axis. In addition, Blender also has a world coordinate system. As we will see later, we will have to go between these two coordinate systems.

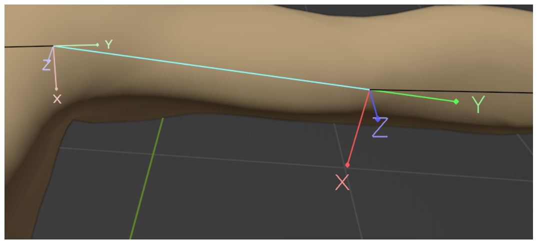

Figure 1. Bone coordinates in Blender

3. Approach

Our approach will be split into roughly three parts:

1. Key Point estimation through OpenPose
2. Triangulation of Key Points through estimation of the fundamental matrix and structure from motion
3. Compute orientation of human pose bones to recreate pose in Blender

Our images and videos have been taken with two iPhone 12 cameras. We will generally assume that the camera matrices between the two cameras are the same.

3.1. OpenPose for Point Correspondence

As mentioned, we take pictures from two different angles of the same person frozen in the same pose. Our first goal is to detect the keypoints, the human body joints (e.g. knees, shoulders, wrists, eyes), in each of the two images. We choose to use the OpenPose BODY-25 Model, which identifies the 25 joints in each person view, as shown in Figure 2. In this case, our input data (iPhone images) are two-dimensional, so OpenPose is a fitting library because it performs 2D pose estimation, where the outputs for each joint are its x- and y- coordinates and a confidence score (0 through 1) for each coordinate prediction. In order to detect keypoints, for each experiment, we first took two pictures, as described above, of a teammember, with an iPhone, and obtained the image metadata for the camera matrix in the Files app. Then, we cloned the Github Repository (https://github.com/CMUPerceptual-Computing-Lab/openpose.git) and installed all necessary dependencies, using Google Colab with GPU to speed up some of the build processes. Then, we ran pose detection prediction, which outputted a json file of the 25 keypoints for each of the two images. We then converted these keypoints from json format to numpy arrays, as this is the required format for triangulation. As we have 25 keypoints in each image, we have 25 2D point correspondences for the matching joints between the images. For the task of videos, we ran OpenPose on left and right videos, which was then able to split those videos into frames and run the same procedure to detect keypoints on each frame treated as an image.

3.2. Triangulation from Stereoscopic images

Much of the theory behind triangulation in this section has been taken from the Stanford course CS231a, taught by Silvio Savarese and Jeanette Bohg.

3.2.1 Implementing Triangulation

The inputs for triangulation are (1) two $N \times 2$ numpy arrays that represent the 2D keypoints identified in the two images, and (2) data for the intrinsic parameters. The output will be $N - 1$, bone orientations represented as a relative rotation about an axis in 3D space. We have used quaternions as an intermediate state to make computations of relative rotations between any two bones easier. Quaternions were chosen due to their compact representation of relative rotations and their immunity to gimbal lock. Furthermore, quaternions may easily be transformed into Euler Angles if necessary.

The 2D keypoint arrays come directly from the output of the previous subsection, which is essentially a reformatted version of OpenPose's output. The camera's intrinsic parameters may be determined through image metadata and a few assumptions. We will assume that our images have square pixels and no-skew, which is a reasonable assumption for a well-tested, mass-produced product such as an iPhone camera. Finally, the image center and focal length

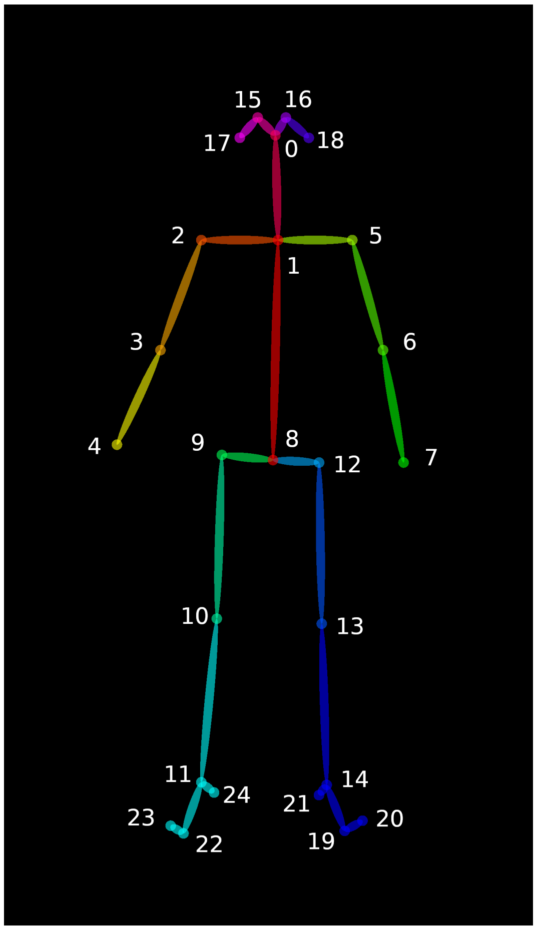

Figure 2. A physical interpretation of the Body_25 model used in OpenPose.

can be found within the image metadata. For images, the metadata indicates a focal length of 4.25mm. Some research on the internet indicates that the sensor used for iPhone cameras have a pixel pitch of 1.4 μm. To find the focal length in units of pixels, we divide the focal length by the pixel pitch, resulting in a focal length of about 3035.7 pixels. The image dimensions are 40323024. Therefore, our camera matrix is the following:

$$K = \begin{bmatrix} 3035.7 & 0 & 1512 \\ 0 & 3035.7 & 2016 \\ 0 & 0 & 1 \end{bmatrix} \quad (7)$$

For video, the camera matrix changes slightly. A difficulty associated with video in iPhones is that the metadata does not provide any information about the focal length. Based on the magnification and new aspect ratio, a rough calculation indicates that the equivalent focal length would be about 1.28 times the original camera focal length. Furthermore, the image dimensions are now 1920×1080. The aspect ratio has changed, but in addition the number of pixels has been severely reduced. This is likely an effort to reduce the overall file size of the video. Based on the change in aspect ratio and number of pixels, we assume that the effective pixel pitch has been increased by a factor of 2. This leads to a focal length of about 1942.9 pixels. Therefore, the camera matrix for videos is the following:

$$K_{vid} = \begin{bmatrix} 1942.9 & 0 & 960 \\ 0 & 1942.9 & 540 \\ 0 & 0 & 1 \end{bmatrix} \quad (8)$$

After constructing our camera matrix, we will perform the normalized eight-point algorithm to compute the fundamental matrix F of the image pair. Since OpenPose provides up to 25 points, even though we have the possibility of occlusions in an image in the majority of cases we should have more than enough points for our fundamental matrix estimation. From here, we can extract the essential matrix E to estimate our projection matrices. We construct the projection matrices as the following from our RT pair candidates:

$$M_1 = K \begin{bmatrix} I & 0 \end{bmatrix}$$
$$M_2 = K \begin{bmatrix} R^T & -R^T T \end{bmatrix}$$

```
Nose
├── Neck
│   ├── RShoulder
│   │   └── RElbow
│   │       └── RWrist
│   ├── LShoulder
│   │   └── LElbow
│   │       └── LWrist
│   └── MidHip
│       ├── RHip
│       │   └── RKnee
│       │       └── RAnkle
│       │           ├── RBigToe
│       │           │   └── RSmallToe
│       │           └── RBigToe
│       └── LHip
│           └── LKnee
│               └── LAnkle
│                   ├── LBigToe
│                   │   └── LSmallToe
│                   └── LBigToe
├── REye
├── LEye
├── REar
└── LEar
```

Figure 3. Tree Structure for Joints

R and T define the transformation from camera 1 to camera 2; hence, we must perform the reverse transformation for camera 2 to construct M_2. With these two projection matrices, we are ready to use them in our non-linear, triangulation algorithm. We will reconstruct all points into 3D before determining which RT pair is most likely to be correct.

3.2.2 Converting key points to bone vectors.

Our 3D points are stored in a tree structure 3 to establish the physical relations between points (example shown in figure 2), and from here we can compute the vector in 3D space that represents each bone in our model by subtracting the appropriate 3D keypoint locations.

One more step we had to go through is to get the vectors in the "blender world" coordinate system. We had to do a quick linear transformation in order to get from camera 1 coordinate system from the images (as interpreted by the OpenPose pipeline) to Blender world coordinate system. This involved switching and flipping axes.

3.3. Using Blender to Mimic Human Pose

We used the Blender software in order to take this information and actually apply it to a 3D model of a human. We are using a model we found online with bones, so we didn't have to create the model of the human from scratch. The model at rest state in pose mode is depicted in 4. In order to actually move the model to the position we

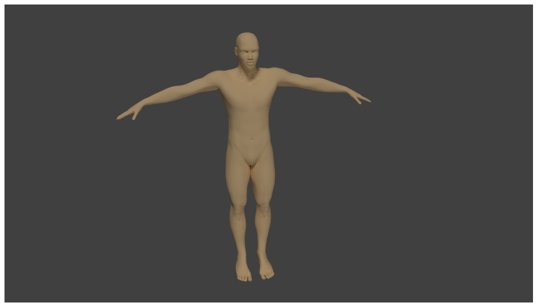
Figure 4. Rest Position Rendering Using Bpy

desired, we used axis-angle rotation because it ended up being more intuitive to implement using bpy. Following is the pseudocode which we had to implement in bpy in order to render the model appropriately.

```
$\vec{c} \leftarrow$ current vector,
$\vec{d} \leftarrow$ desired vector
$\vec{w} \leftarrow \vec{c} \times \vec{d}$
$\theta \leftarrow \arccos\left(\frac{\vec{c}\cdot\vec{d}}{|\vec{c}||\vec{d}|}\right)$
$\overrightarrow{axis} \leftarrow {}_{\text{world}}M_{\text{bone local}}\vec{w}$
return( axis,$\theta$)
```

The main idea in the pseudocode shown is to do computations in world coordinates and transform them to bone local coordinates. This is because we receive our input in the world system, but Blender requires transformations to be done in the bone local coordinate system.

4. Experiment

There are 3D human pose datasets that provide annotations for angles and orientations between joints, such as the Human3.6M dataset [3], that would have provided an appropriate quantitative metric for our algorithm. However, we decided to first take our own images and use them as our benchmark. Our aim is to test the feasibility of our pipeline for easy use and access to the everyday user. We expect such a user to not have access to specialized equipment for human pose estimation, so we opted to test our algorithm with basic cameras (such as those on iPhones), which we used to create our custom (tiny) dataset. Additionally, our pipeline does not require any training or finetuning with a large dataset as we leverage several pre-trained technologies such as OpenPose, trivializing the need for a large dataset at this initial stage.

4.1. Testing on Images

Testing our pipeline's effectiveness and robustness with images lead to two broad categories of images. We first tested a baseline pose, for which all key points are clearly visible in both images, and then an occlusion pose, for which some key points are occluded in one or more images.

We will demonstrate our results here. First, we used stereo images of a model sitting and passed it through the OpenPose Pipeline, which detects keypoints as shown in 6.

Next, we triangulate the positions of each key point. We were able to generate a plot of the key points in 3D space as a sanity check. An example of such a plot is shown in figure 5. Finally, we perform the last step, where we model a human using our computed axis vectors and relative rotations in Blender. Our final result is shown in figure 6.

We went through the same pipeline for a stereo image pair with occluded keypoints. The ground truth and resulting rendering is shown in 7. As we can see, as of right now, our pipeline ignores the case where there are occluded points and simply defaults to the rest pose position in Blender. However, it is important to note that we were still able to successfully reconstruct the remaining key points even with the absence of a few points. At minimum, we need eight points for our estimate of the fundamental matrix; since OpenPose provides up to 25 points, the probability of more than 17 points being occluded is unlikely (at this point, we probably won't be able to glean any useful pose information anyways!). This gives us confidence that, when we apply this to video, our algorithm will not error whenever a few key points are missing in a few frames. Accounting for the 3D position of key points in blender can be a study left for future work.

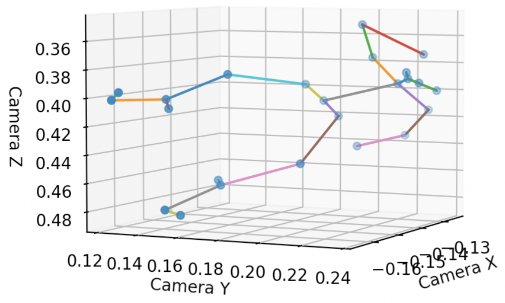

Figure 5. Plot of Keypoints in 3D space. The general orientation of the model sitting is visible.

4.2. Testing on Video

Next, to test our pipeline on videos, we took a video of a teammate from a left view and a right view, attempting a motion that occludes as few points as possible throughout the video, in order to test our pipeline on the most simple case.

As our results for pairs of images were promising, we had assumed that translating the algorithm from images to video would have been an easy task. However, a new set of challenges arose with the use of video.

Since the 3D reconstructions were performed for each frame independently, each reconstruction has no knowledge of the previous frame's estimates. Due to a large amount of noise between each frame, our reconstructions varied greatly from frame to frame, making it nearly impossible to reliably obtain a smooth, 3D motion of key points. The amount of noise between frames can be seen in the position estimates in figure 8 and the relative axis angles in figure 9.

We suspect our major source of error stems from our estimates of R and T, the rotation and translation matrices defining the relative position and orientation of the two cameras. We noticed that our estimate of R and T between the cameras change non-trivially between frames. This should not be the case since we kept our cameras stationary while taking video. This can be observed in figure 10. Further investigation shows that this could indicate a deficiency in our algorithm for estimating the fundamental matrix, which seemed to also change non-trivially between the subsequent video frames. Again, since our cameras are stationary, this matrix should not be significantly changing between frames.

Furthermore, we noticed some issues with our non-linear triangulation algorithm. Although not extremely common, some of our key point estimates resulted in gradient explosion, making our 3D point estimate for some key points unusable in some frames. Although it is entirely clear why this occurs, one theory is that our initial estimate for our optimization problem was simply not good enough, leading to a large reprojection error, and in turn leading to an unstable system. This could, again, be caused by noisy measurements of the key points in the images by OpenPose.

A potential fix that could be expanded upon in future work is to take advantage of our measurements and 3D estimates from previous frames. Therefore, this problem would be a great candidate for state estimation technique such as Kalman filters. One suggestion for our state would be to estimate our fundamental matrix. The process noise could involve vibrations that the cameras may experience due to the environment or shaky hands, and our measurement noise will stem from OpenPose's potentially noisy output of key point locations in images, camera distortions, and discretization due to pixels. Assuming our cameras stay stationary, our update matrix would simply be the identity matrix, since the fundamental matrix should not be changing with time. Our fundamental matrix can also predict the locations of 2D key points in one image given the location of 2D key points in the other image, leading to a viable measurement model with a tunable measurement noise parameter Other candidates that may be included in our state include 3D positions of key points, 2D coordinates of the key points, and the optical flow of key points. By applying state estimation techniques, we may also be able to propagate points if they are occluded and predict outliers, possibly leading to smoother motion in our final result.

4.3. Evaluating Results

We don't have any quantitative metrics for our image results. As discussed before, we could obtain these using an annotated dataset. Qualitatively, we think the results look pretty accurate. They seem to capture the idea of the pose very well. However, some work needs to be done on rotating joints correctly and dealing with occluded points.

5. Conclusion

Some areas of improvement that we have identified were issues with our non-linear triangulation algorithm and issues with our rotation and translation matrices, which could all potentially contribute to the significant noise present between frames in the video. As a next step, we will need to identify the cause of gradient explosion in triangulation, by further analyzing the code and testing more refined initial estimates. Additionally, we will need to make our algorithm for calculating the fundamental matrix more robust to variations in frames, such that our estimate of R and T becomes more consistent across frames. The goal of such improvement would be to improve the result of the end-to-end pipeline for videos discussed above. A potential solution that could be studied in future work is the application of

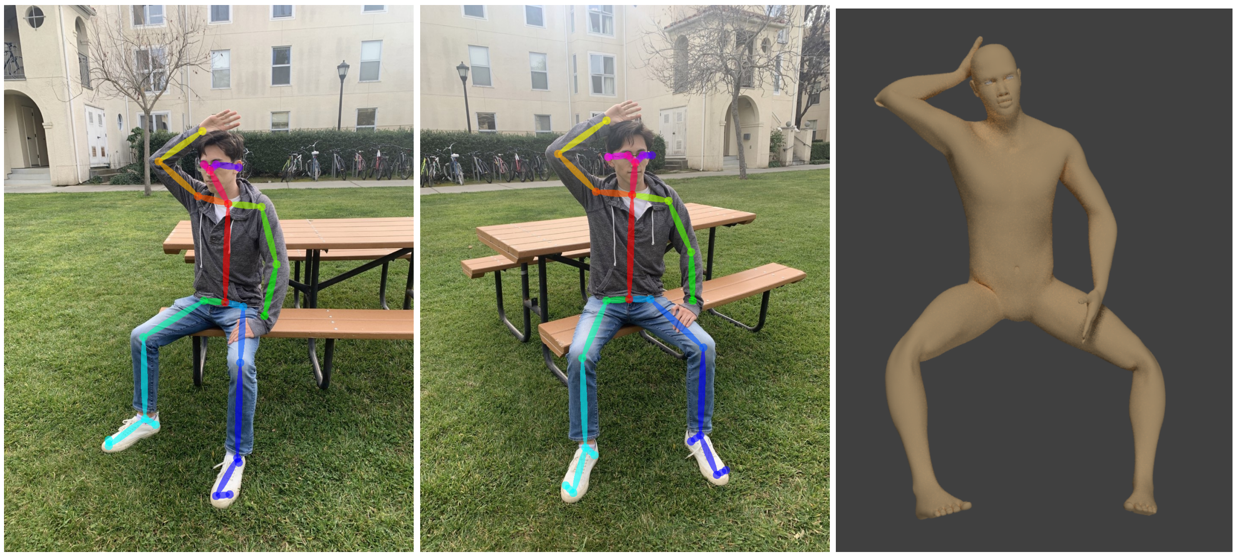
Figure 6. Results of Experiment 1: Keypoints on Pose with No Occluded Points and the associated Blender render

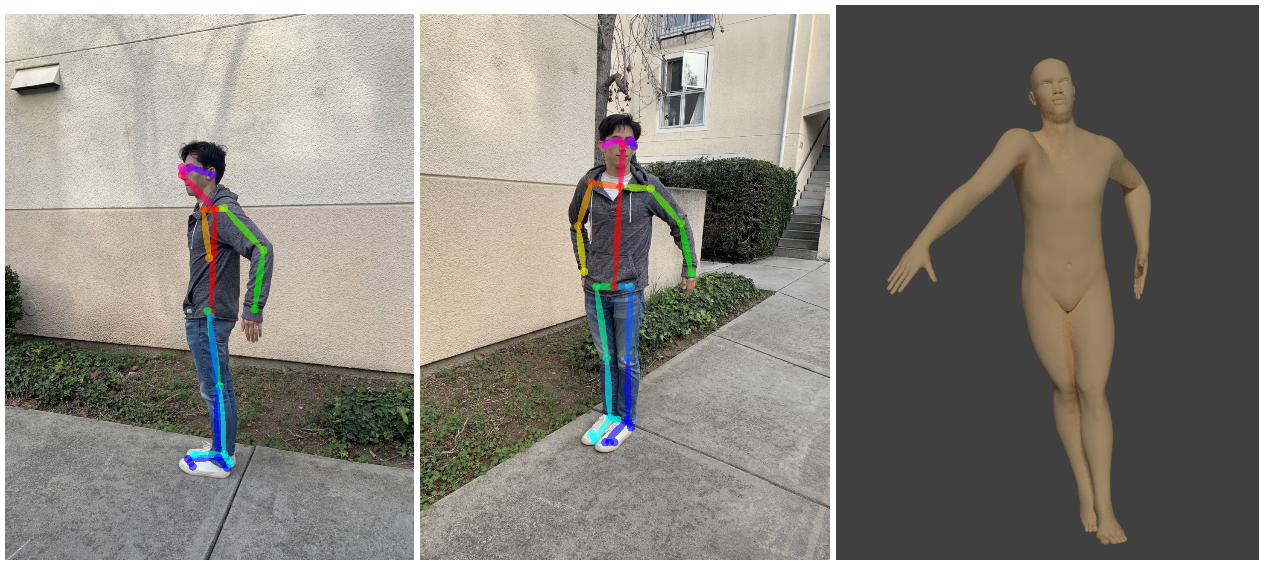
Figure 7. Results of Experiment 2: Keypoints on Pose with Occluded Points and the associated Blender render

state estimation techniques to reduce noisy states over time. This would allow us to take advantage of the priors present in video frames, something that we do not do currently.

As for improvements on images, we would like to handle the case of occluded points better than just ignoring the joint as a whole. As an image is a single, independent entity, state estimation techniques would not be applicable for our solution. Future solutions may take advantage of our knowledge of the human body, allowing us to restrict orientations in certain directions. The lack of key points from certain camera angles may also provide clues to the true location of the points; e.g. if the right elbow and right wrist are not present, but the left elbow and left wrist is, its possible that the image is the side profile of a person and the right arm is hiding behind the body. We may also utilize information of key points that are present in one image but occluded in the other.

Another way we could improve is to focus on improving our estimates of bone rotations within the blender rendering. However, right now we have no idea how the joints are rotated since we only know the position of the keypoints. This can be clearly seen in figure 6. This may be improved by creating a blender model that is designed for use with OpenPose, since our current model is a generic model downloaded off of the internet.

Finally, to collect more quantitative measures of the performance of our algorithm pipeline, we can use annotated datasets such as Human3.6M [3] [2] or 3DPW [7], which

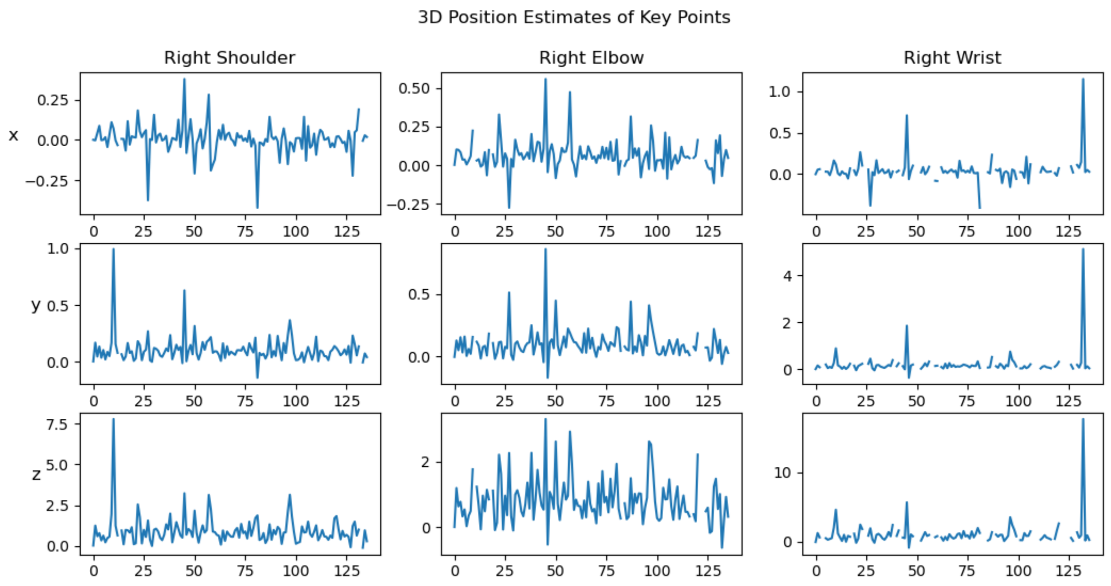

Figure 8. The estimated 3D positions of the right shoulder, right elbow, and right wrist for a punching motion. The gaps in data indicate points with bad estimation or bad data.

provides ground truth data on poses for a variety of poses.

References

[1] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. 1

[2] F. L. Catalin Ionescu and C. Sminchisescu. *International Conference on Computer Vision*. 7

[3] V. O. Catalin Ionescu, Dragos Papava and C. Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7), 2014. 5, 7

[4] B. O. Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 2

[5] E. Insafutdinov, M. Andriluka, L. Pishchulin, S. Tang, E. Levinkov, B. Andres, and B. Schiele. Articulated multi-person tracking in the wild. *CoRR*, abs/1612.01465, 2016. 2

[6] E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. *CoRR*, abs/1605.03170, 2016. 2

[7] T. von Marcard, R. Henschel, M. Black, B. Rosenhahn, and G. Pons-Moll. Recovering accurate 3d human pose in the wild using imus and a moving camera. In *European Conference on Computer Vision (ECCV)*, sep 2018. 7

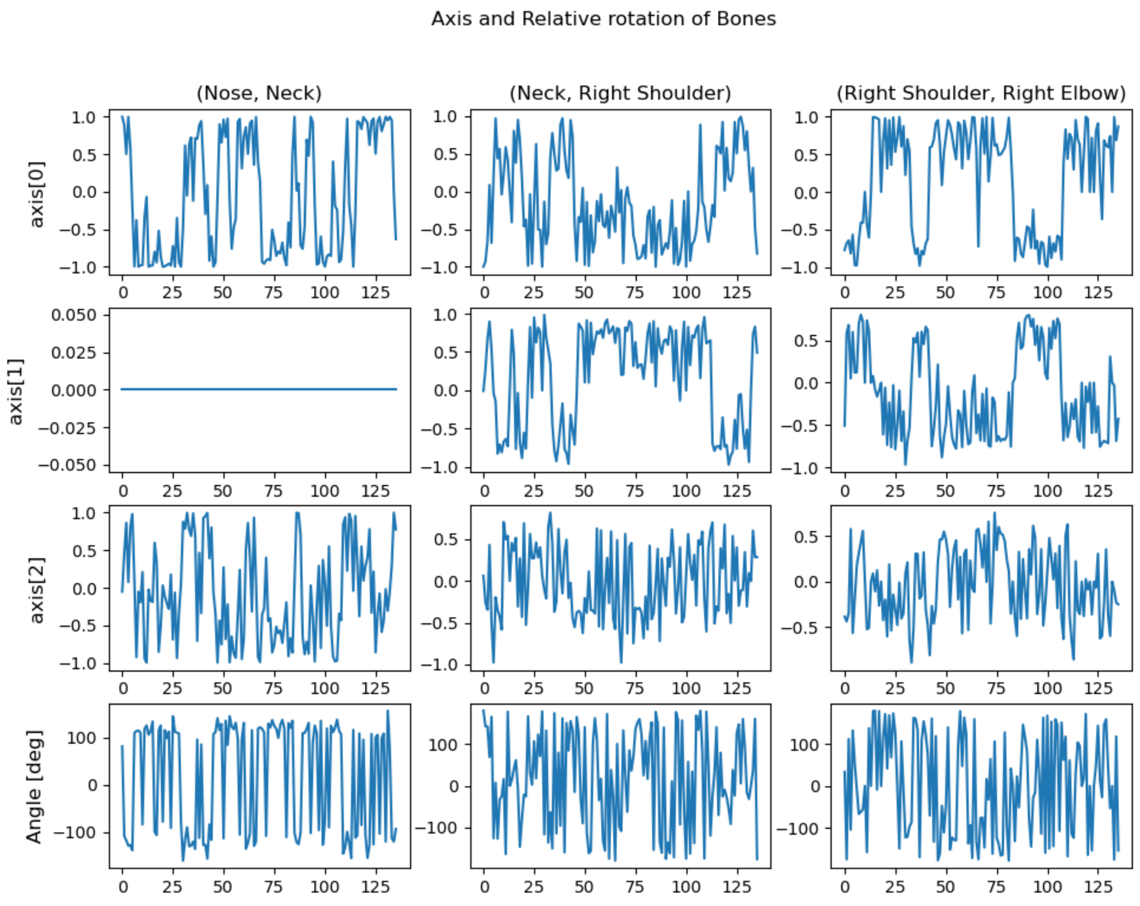

Figure 9. The estimated axis and rotation angles between select bones. Angles have been constrained to lie between -180 and 180 degrees. (Nose, Neck) is the only bone that is described relative to the world y-axis, which can be seen in the plot.

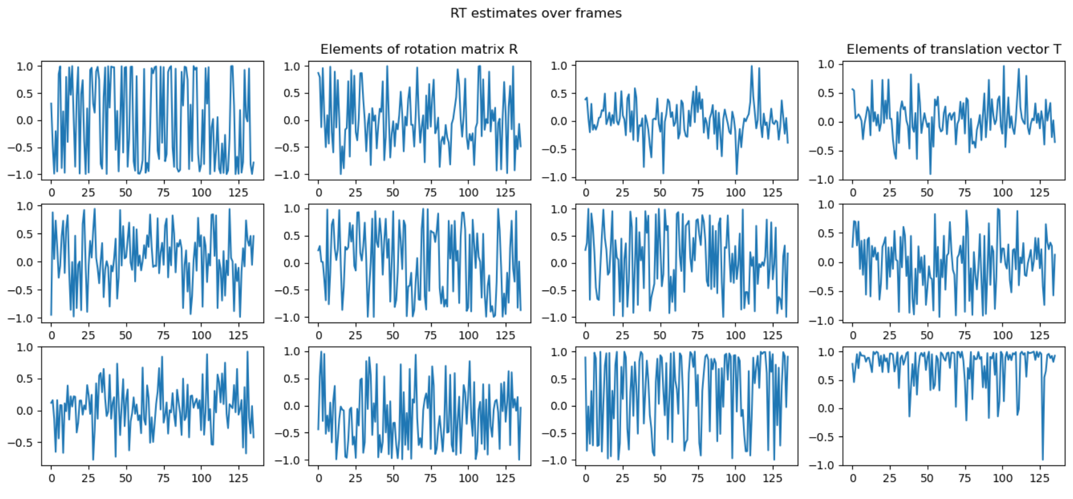

Figure 10. The elements of our estimates of rotation matrix and translation vectors over time. We expect our plots to be nearly flat.