Human Pose Estimation from 3D Mesh: A Synthetic Egocentric Pipeline

Matteo Bordignon, Cristian Murtas, Marco Wang
University of Trento
{matteo.bordignon, cristian.murtas, marco.wang}@studenti.unitn.it

Abstract

In this report, we present a synthetic pipeline for human pose estimation using a 3D mesh model in Blender. First, we simulate an egocentric viewpoint by placing a virtual camera above the subject's head, extracting 3D joint coordinates and projecting them onto a 2D image plane. We then create a stereo camera rig to reconstruct the 3D pose from two 2D projections, highlighting how controlled virtual environments can facilitate accurate pose recovery.

1. Introduction

Human pose estimation is a fundamental task in computer vision, with applications ranging from motion capture and activity recognition to virtual reality and robotics. In this work, we propose a synthetic pipeline for human pose estimation based on a single 3D human mesh. We focus specifically on an egocentric viewpoint, placing a virtual camera directly above the head of the subject, facing downwards. This viewpoint allows us to simulate use cases such as wearable cameras (helmet- or glasses-mounted). Furthermore, we extend the setup to a stereo rig to exploit the geometry of two camera views for 3D reconstruction. The objectives of this project are:

- Import the mesh and set up blender environment;
- Derive the 2D image plane coordinates corresponding to 3D joint coordinates using perspective projection;
- Reconstruct the original 3D coordinates using the images captured from a stereo camera rig.

The remainder of this report is organized as follows. In Section 2, we describe our methodology, including the 3D mesh setup in Blender and the camera configuration. Section 3 presents experimental results on pose accuracy and reconstruction. Finally, Section 4 concludes the report with comments about our results.

2. Methodology

Our pipeline begins with setting up the Blender environment, where we prepare the 3D scene and configure the necessary parameters. Subsequently, we extract relevant information, including the 3D data and camera parameters. From the camera(s) in the setup, we capture a screenshot of the scene. Next, we apply a perspective projection to map the 3D points onto their corresponding 2D pixel coordinates in the captured image(s). When two cameras are available, we use their data to estimate the original 3D coordinates through triangulation. Finally, we compare the estimated 3D coordinates with the original ones to analyze and quantify the error. Below, we detail each step.

2.1. Blender's Setup

We begin with a pre-rigged 3D human mesh, complete with a skeletal structure consisting of several bones (neck, spine, hips, shoulders, elbows, wrists, knees, ankles, etc.). The model is imported into Blender, where we ensure the mesh is scaled to realistic proportions (approximately 1.75 meters tall). Next, we add a virtual camera to the Blender scene. This camera is positioned directly above the head bone of the mesh with a y-axis offset of -0.2 m and slightly rotated on the x-axis, pointing downward (Fig. 1). The camera has a focal length of 50 mm and a sensor width of 36 mm, which are the default settings of a Blender camera.

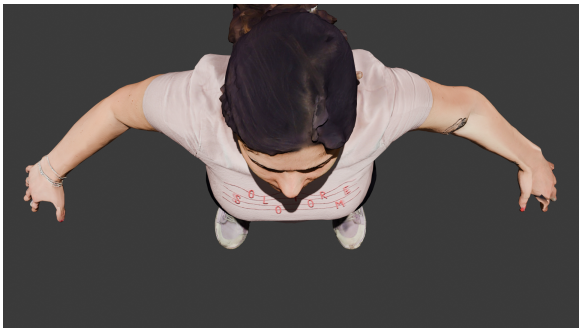


Figure 1: Illustration of the render of the egocentric camera placed above the human mesh. `python main.py -n 1 -l True`

From this setup, we extract:

- The 3D coordinates of each joint (in Blender's world coordinate system);
- A rendered image from the egocentric viewpoint.

2.2. Perspective projection

To project a 3D coordinate onto a 2D image plane, we need to calibrate the camera to obtain its projection matrix, which provides insight into the transformation applied to a 3D point by the camera, and then use it to perform the projection. Here, we discuss the theoretical principles underlying this process and explain its practical implementation in Blender.

2.2.1 Camera Calibration for Intrinsic and Extrinsic Parameters

Camera calibration is a process used to determine the intrinsic and extrinsic parameters of a camera, which are essential to accurately map 3D world to 2D image points.

Intrinsic Parameters The intrinsic parameters define the internal characteristics of the camera, such as:

- **Focal Length** (f_x, f_y): Determines how real-world distances are scaled to pixels;
- **Principal Point** (c_x, c_y): The optical center of the image, usually near the image center;
- Skew, Distortion Coefficient etc.

Extrinsic Parameters The extrinsic parameters define the position and orientation of the camera in the world coordinate system. These include:

- **Rotation Matrix** (R): Describes the camera's orientation;
- **Translation Vector** (t): Describes the camera's position in the world.

Calibration Process To determine these parameters, calibration involves:

1. Capturing multiple images of a known pattern (e.g., a checkerboard) from different viewpoints;
2. Identifying key features (e.g., corner points) in both 3D and 2D;
3. Estimating the parameters using optimization algorithms (e.g., least squares approach) to minimize the error.

2.2.2 Projection

Let the 3D point be:

$$P_w = (X, Y, Z)$$

where P_w represents our 3D coordinates for the joint in the world coordinate system.

Transform to Camera Coordinates Then we apply a transformation using the extrinsic parameters of the camera:

$$\mathbf{P}_{\text{camera}} = \mathbf{R} \cdot \mathbf{P_w} + \mathbf{t}$$

where: $\mathbf{R}$ is the rotation matrix (3×3), $\mathbf{t}$ is the translation vector (3×1) and $\mathbf{P}_{\text{camera}} = (X_c, Y_c, Z_c)$ are the coordinates of the point in the camera's coordinate system. In homogeneous coordinates, this can be written as:

$$\mathbf{P}_{\text{camera}} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0^T} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P_w} \\ 1 \end{bmatrix}$$

Project to the Image Plane Use the intrinsic matrix to project the 3D point onto the image plane:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z_c} \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

where: f_x, f_y are the focal lengths along the x and y axes, c_x, c_y are the principal point offsets and u, v are the 2D coordinates on the image plane:

$$u = f_x \cdot \frac{X_c}{Z_c} + c_x, \quad v = f_y \cdot \frac{Y_c}{Z_c} + c_y$$

Implementation Since we are working in a simulated environment, both the calibration and projection processes were significantly easier to implement. For the calibration, we utilized Blender's API to directly access the intrinsic and extrinsic parameters of the camera(s). For the projection, we employed the `world_to_camera_view()` function, which returns the normalized device coordinates (NDC) for a given 3D point. In this coordinate space, (0,0) represents the bottom-left corner of the camera view, and (1,1) represents the top-right corner, with all values constrained within the [0,1] range. To map the NDC coordinates to pixel values, we used the resolution of the rendered image, and flipped the Y-coordinate to match the Blender's space origin ensuring the 3D points were accurately projected onto the 2D image plane.

2.3. 3D Reconstruction

To precisely estimate the depth coordinate, we used two cameras configured in a parallel and aligned setup. Since we already have the corresponding points in the

stereo images, we can compute the disparity, as the difference in the positions between these points along the horizontal axis.

$$\textbf{disparity} = u_L - u_R \tag{1}$$

where u_L and u_R are the $\mathbf{x}$ coordinates of the two points in the pixel space. The disparity, combined with the known distance between the cameras and their intrinsic calibration parameters (obtained through Blender APIs), allows us to calculate the 3D coordinates using triangulation.

$$\mathbf{Z} = \frac{fb}{\text{disparity}} \tag{2}$$

$$\mathbf{X} = \frac{(v_L - c_y)\mathbf{Z}}{f} \tag{3}$$

$$\mathbf{Y} = \frac{(u_L - c_x)\mathbf{Z}}{f} \tag{4}$$

where u_L and v_L are the 2D point coordinates converted in pixel space, f is the focal length, b is the baseline distance, c_x and c_y are the principal component coordinates.

3. Results

We conducted the tests on the figure above (Fig. 1). During the test, the following data has been used:

1. Rendered images from the camera(s);
2. Projected 2D keypoints from the egocentric camera;
3. Stereo-based 3D triangulated points;
4. Ground-truth 3D coordinates (directly from the Blender rig).

To assess reconstruction accuracy, we compute the Mean Per Joint Position Error (MPJPE):

$$\text{MPJPE} = \frac{1}{N}\sum_{j=1}^{N}\|\hat{\mathbf{X}}_j - \mathbf{X}_j\|$$

where $\hat{\mathbf{X}}_j$ is the reconstructed 3D position of joint j, $\mathbf{X}_j$ is the ground-truth position, and N is the total number of joints. Table 1 summarizes the results obtained.

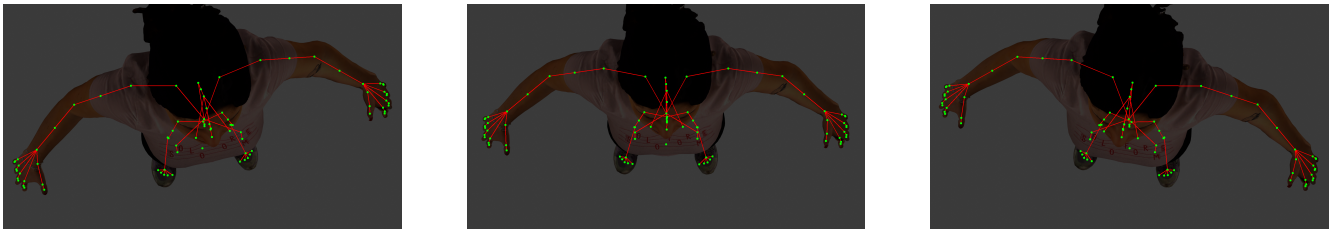
Error	**Value (mm)**
Mean Absolute Error X	$3.57 \cdot 10^{-5}$
Mean Absolute Error Y	$1.19 \cdot 10^{-5}$
Mean Absolute Error Z	0.00045
MPJPE:	0.00049

Table 1: MPJPE and X, Y, Z mean absolute error with 2 cameras and a baseline distance of 2cm.

The average MPJPE is negligible, indicating that the stereo rig yields fairly accurate reconstructions in this controlled environment. The errors are primarily influenced by the Z-coordinate, which exhibits the largest error due to being the missing coordinate requiring estimation. Figure 2 shows a sample frame with projected 2D keypoints (circles) overlaid on the rendered image.

4. Conclusions

We presented a synthetic pipeline in Blender for human pose estimation, emphasizing an egocentric camera configuration and extending it to a stereo rig for 3D joint reconstruction. Our results show that the outcomes are nearly perfect. However, we are aware that achieving such precision in a real-world scenario is very unlikely due to several factors. First, we did not have to estimate the intrinsic and extrinsic parameters, since Blender's API allowed us to extract exact values for those parameters without taking into account any distortion or noise. Second, we also avoid the process of matching pixels between images, as the correspondences were computed by us in a previous step within the controlled environment. These simplification notably contributed to the near-perfect result observed. Even though we used ideal values for our environment, Blender can be setup to recreate plausible real-world conditions. For this reason, virtual environments like Blender can be used to produce datasets for training and testing models designed to perform HPE.



`-n 1, -t -4,0,0, -r 0.13,0,-0.2` `-n 1` `-n 1, -t 4,0,0, -r 0.13,0,0.2`

Figure 2: Results of perspective projection with different parameters using *python main.py*