修 士 論 文 概 要 書

Master's Thesis Summary

Date of submission: 01/23/2023 (MM/DD/YYYY)

専攻名 (専門分野) Department	Computer Science and Communications Engineering	氏 名 Name	TszHo Man	指 導 教 員 Advisor	Hiroshi Watanabe 印 Seal
研究指導名 Research guidance	Research on Audiovisual Information Processing	学籍番号 Student ID number	5121FG09-4		
研究題目 Title	Research on Upper Body Limbs Position Estimation in Real-time Using Inverse Kinematics				

1. Introduction

Human motion capture and pose estimation is an important topic in computer vision, as it allows for the accurate conversion of human body motion from 2D images to 3D coordinates. Accurate motion capture is essential in these contexts, as it allows for the creation of realistic and natural-looking human body animations.

However, accurate motion capture can be difficult to achieve in marker-less tracking, particularly in cases of occlusion, as it may result in ambiguous or incomplete tracking. The human joints might also be blocked by the environment in the 2D image or be occluded by the human body.

The purpose of this research is to improve the tracking accuracy of the upper body motion by using the Inverse Kinematics (IK) method in the occluded cases. This method allows for the fast recovery of pose using fewer joints than traditional tracking methods by taking into account the specific transformation and rotation constraints of human joints.

2. Related Work

The recent research of IK in computer vision are mostly employing data-driven method. Such as Grochow [1], who optimize the human pose by only a few joints from the encoded data, Kim [2] employs transformer and RNN related neural networks to solve the problem. Although these mentioned methods provide a solid pose estimate result, they often come with a high demand on computation cost, which is difficult to be used in a real-time application.

3. Proposed method

In our research, we propose a lightweight model using MLP to solve the IK problem. By using this method, it will be possible to compute the IK in real-time. This will enable us to achieve accurate pose estimation while also maintaining a high level of efficiency and speed.

The objectives of our research are as follow:

- Estimate the occluded joint from the upper body using IK in real-time.
- Achieve high accuracy of the estimated joint's position.
- Eliminate the requirement of manually set up joint constraints.

We proposed an IK solver using MLP, the model is able to estimate the occluded joint's position based on the kinematic chain's transformation, resulting in more accurate pose estimation. The model consists of three hidden layers, each of them are using Softmax as the activation function. The network structure is inspired by Gholami [2], which controls a delta robot's position by calculating the IK. We have conducted multiple experiments to test the combination of different hyperparameters and adjusting them in order to get a best training performance.

Our proposed workflow is divided into two main stages, as illustrated in Fig. 1:

1. Keypoints extraction: Using an existing real-time pose detection model to extract the 3D coordinates of keypoints
2. IK computation: Using the extracted keypoints as input, the MLP network performs IK computation to estimate the occluded joint poses.

4. Dataset

Our research utilizes the 3D Poses in the Wild (3DPW) [3] dataset as a benchmark for evaluating the performance of our 3D human pose estimation algorithm. The 3DPW dataset is a comprehensive dataset that includes 3D human pose annotations for a wide range of activities and viewpoints. This dataset is commonly used in research to evaluate the performance of different 3D human pose estimation methods.

For our research, we conducted pre-processing to the dataset motion sequences. The motion sequences with more than one model are separated into individual sequences. This decision was made because the focus of our research is not on detecting multiple people in an image, and separating the sequences simplifies the training and testing process. Additionally, we converted the train dataset from world coordinates to local

coordinates, using the hip joint of the model instead of the world origin as the origin of the model.

5. Experiment

Our research is using mean square error (MSE) loss as the loss function to evaluate the model. The 3DPW dataset suggests researchers to use joint error metric for the evaluation, which is the mean Euclidean distance between predicted joints and the ground truth joints. While MSE loss is the square of joint error metric. Since the MSE loss and joint metric error both provide the same geometry information, we are using MSE loss for the metric evaluation.

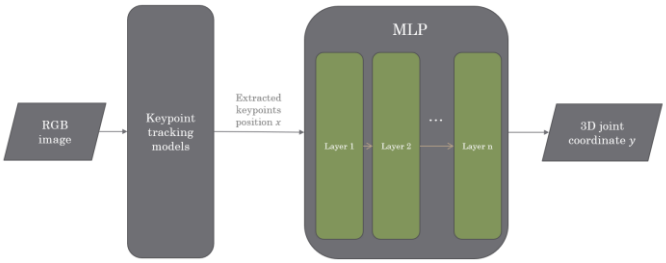


Fig. 1. Proposed model structure

The accuracy of the proposed model is evaluated by the test dataset and the evaluation dataset. Therefore, we conducted multiple experiments to find out the best training condition, such as the input dataset representation.

6. Analyze

The proposed model can be utilized with existing pose tracking models. We extract the joint coordinates using BlazePose [4] (Fig. 2). As BlazePose is able to provide the visibility value of each joint, we can determine how confident it is in detecting the position of each joint. In this particular example, that the sample image is taken from 3DPW dataset, BlazePose has a low visibility value for the right elbow because it is occluded by the body.

We then extract the joint coordinates of the right arm, input them into our proposed model, then the estimated joint position of right arm elbow is computed. For this example, the result is illustrated in Fig. 3. With the purple line indicates the new estimated position of the occluded elbow. Due to the scale difference along each axis, the result might be distorted. However, the MSE loss is similar to the training result.

We also took the same sample image's 3D joint coordinates, and directly predict the right elbow joint. The result has a better performance than the result that generated based on the BlazePose detection. We assume that this is caused by the insufficient variant of dataset.

7. Conclusion

This research presented a lightweight machine learning model as a real-time IK solver, which can provide accurate results for upper body limb position estimation. The proposed method is based on MLP network and has been trained on the 3DPW dataset. The model has been tested and demonstrated its practical application in combination with existing 3D body tracking models.

Fig. 2. Keypoints captured by BlazePose [4]

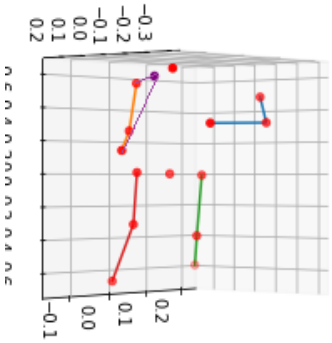


Fig. 3. Estimated keypoint result utilizing with existing BlazePose [4]

8. References

1. K. Grochow, S.L.Martin, A.Hertzmann, and Z. Popivic, "Style-based Inverse Kinematics," ACMSiggraph, pp.523-531, Aug.2004.
2. A. Gholami, T. Homayouni, R. Ehsani, and J.-Q. Sun, "Inverse Kinematic Control of a Delta Robot Using Neural Networks in Real-Time," Robotics, vol. 10, no. 4, p. 115, Oct. 2021, doi: 10.3390/robotics10040115.
3. T. von Marcard, R. Henschel, M. J. Black, B. Rosenhahn, and G. Pons- Moll, "Recovering Accurate 3D Human Pose in the Wild Using IMUs and a Moving Camera," in Computer Vision – ECCV 2018, pp. 614–631.
4. V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann," BlazePose: On-device Real-time Body Pose tracking," CV4ARVR, 2020. (accessedJan.10,2023).

Research on Upper Body Limbs Position Estimation in Real-time Using Inverse Kinematics

A Thesis Submitted to the Department of Computer Science and Communications Engineering,
the Graduate School of Fundamental Science and Engineering of Waseda University
in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Submission Date: January 23rd, 2023

TszHo Man
(5121FG09-4)
Advisor: Prof. Hiroshi Watanabe
Research guidance: Research on Audiovisual Information Processing

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my advisor, Professor Hiroshi Watanabe, for his guidance and support throughout my research. His suggestions and insights have been instrumental in the development of this research.

I would also like to thank the seminar members of the Advance Multimedia Systems Lab. for their feedback and advice provided for my research.

I am also grateful to the support of Department of Computer Science and Communications Engineering, for providing the support for my research.

Finally, I would like to extend my gratitude to all of the participants in my study, my family and my friends. Their support and willingness to share their experiences has been invaluable to my research and has helped to make this thesis a success. Thank you for your time and support.

List of Figures

LIST OF TABLES

Abstract

In this research, we proposed a lightweight machine learning method for real-time Inverse Kinematics (IK) computation. IK is a technique used to determine the movement of an object in 3D space, by specifying the desired end effector position and solving for the necessary joint angles to achieve it. However, IK is a difficult problem to solve due to its non-linearity and the redundancy problem. We studied different existing methods that related to IK computation, based on the existing techniques, we utilize the multi-layer perceptron (MLP) architecture and the 3DPW dataset for training. Our model is capable to estimate upper body limb positions in 3D space. The results demonstrate the proposed model's ability to deliver accurate results with real-time computation performance. We also discussed the limitations of the model and the work we aim to improve in the future.

I. Introduction

Human motion capture and pose estimation is an important topic in computer vision, as it allows for the accurate conversion of human body motion from 2D images to 3D coordinates. This can be useful for various applications, such as motion capture for movie special effects, pose extraction, and even physiotherapy. Accurate motion capture is essential in these contexts, as it allows for the creation of realistic and natural-looking human body animations. It also can provide valuable insights into human movement and posture.

However, accurate motion capture can be difficult to achieve in marker-less tracking, particularly in cases of occlusion. As it may result in ambiguous or incomplete tracking due to occlusion. The human joints might be blocked by the environment in the 2D image or be occluded by the human body.

The purpose of this research is to improve the tracking accuracy of the upper body motion by using the Inverse Kinematics (IK) method in the occluded cases. Occlusion can be a major challenge in marker-less tracking, as it can cause the human joints to be blocked by the environment in the 2D image or occluded by the human body itself. To address this issue, we plan to use the IK method, which is commonly applied in 3D animation. This method allows for the recovery of pose using fewer joints than traditional tracking methods by taking into account the specific transformation and rotation constraints of human joints. However, there may be redundancy result, that multiple IK solutions exist for a given desired task value, therefore, multiple solutions for one joint's constraint are generated, which can lead to inaccurate pose estimates.

The existing research of IK in computer vision are mostly require a high computation power, and long processing time. Such as Grochow [1] optimizes the human pose by only a few joints from the encoded data, Kim [2] uses transformer and RNN related neural networks to solve the IK. The detail of these method will be discussed later in the next section. Although these mentioned methods provide a solid pose estimate result, they often come with a high demand on computation cost, which is difficult to be used in a real-time application. Therefore, in our research, we propose a less computation intensive method for solving the IK problem. By using this method, it will be possible to compute the IK using a neural network in real-time. This will enable us to achieve accurate pose estimates while also maintaining a high level of efficiency and speed.

The objectives of our research are as follow:

- Estimate the occluded joint from the upper body using IK in real-time. Since the upper body is particularly important in daily life applications, such as webcam capture, portrait photography, so being able to accurately track this region is crucial.
- To achieve high accuracy of the estimated joint's position. There is often redundancy occurs while solving IK, and multiple possible solutions may come up. Our method should be able to choose the solution that best matches the ground truth result.
- Eliminate the need for manually set up. In the traditional algorithm IK solver, the manual setup is often required such as joint constraints, joint angle limit, in order to get a more realistic result. Our method should be able to adapt the configuration and constraint of a human body skeleton. Therefore, there is no need for any additional setup or constraints.

Overall, our goal is to develop a method for IK that is accurate, efficient, and easy to use, with the ability to handle occlusion from the environment or body itself. By achieving these objectives, we hope to make human motion capture and pose estimation more reliable and widely applicable, leading to more accurate and realistic representations of human movement and posture for the upper part of human body. We proposed an IK solver using Multilayer Perceptron (MLP) or fully connected layers. Our approach is lightweight, and low computation cost, making it suitable for real-time tracking. The model is able to estimate the occluded joint's position based on the kinematic chain's transformation, resulting in more accurate pose estimation.

The significance of our research lies in the improvement the accuracy of the tracking result, and the efficiency of human motion capture. By combining the IK method with neural network in real-time, we aim to achieve a high level of tracking accuracy in human motion capture especially for the occluded cases. By addressing the occlusion problem, our research aims to make human motion capture more reliable and widely applicable. This can lead to more accurate and realistic representations of human movement and posture.

II. Background

Inverse Kinematics (IK) techniques have been the subject of study for decades, and there are several methods that have been developed to solve this problem. In this section, we will provide a brief overview of these methods, their key features and limitations.

A. Kinematic Chain

A kinematic chain is a series of connected rigid bodies that can move relatively to each other. The movement of a kinematic chain is governed by the constraints between the bodies in the chain. Kinematic chains are commonly used in robotics, computer graphics, 3D animation, and other fields to model the movement of mechanical systems.

The concept of kinematic chain is proposed by Franz Reuleaux in 1876 [3] . He proposed that, in a kinematic chain, each body is referred to as a link, and the connections between the links are called joints. The joints can be rotational or translational, depending on the type of movement they allow. In a human skeleton, there is only rotational joints. The links and joints of a kinematic chain form a hierarchy, with the base link at the bottom and the end effector at the top. The base link is typically fixed in place, while the end effector is free to move within the constraints of the chain.

Kinematic chains are useful for modeling and analyzing the movement of mechanical systems, as they allow for the consideration of the constraints and interactions between the links and joints. They are commonly used in robotics to design and control the movement of robotic arms and other mechanical systems. Kinematic chains are commonly used in computer graphics to model and animate the movement of characters and other objects. In 3D animation, kinematic chains are used to represent the hierarchy of bones in a character's skeleton, with the bones serving as the links and the joints representing the points of rotation or translation (Fig. 1). This hierarchy allows for the manipulation of the character's pose and movement by manipulating the rotations and translations of the joints in the kinematic chain. Therefore, it is possible to create realistic and natural-looking character animations.

In a kinematic chain, the complete configuration is specified by the scalars θ_n, and each joint angle's value is θ_j. Certain points on the links of the kinematic chain are called end effectors, the positions of the end effectors are denoted as s_k. Each end effector is a function of the joint angles

$$\vec{s} = [s_1, s_2, ..., s_k]^T. \tag{1}$$

In addition, kinematic chains are also used for IK, which involves computing the joint angles necessary to achieve a desired end effector position. This can be useful for tasks such as character animation, where the animator may specify the desired position of the end effector (e.g., a hand or foot) and the IK solver computes the joint angles necessary to achieve that pose.

B. Forward Kinematics (FK)

Forward kinematics (FK) is a technique used in robotics and computer graphics to process the skeleton joints, also known as end effectors, in 3D space given the position and orientation based on the joint angles [4].

By specifying the joint angles of each joint, the FK solver can calculate the position and orientation of the end effector, such as the hand or gripper. It is performed by rotating the joints of a kinematic chain

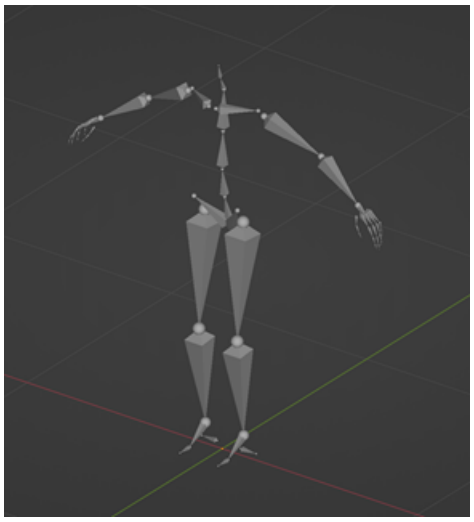


Fig. 1. A kinematic chain of human skeleton created in Blender.

from the root joint to the ending in the leaf joints recursively. This can be used to control the movement of robots in manufacturing, assembly, and other applications. And is defined by [4]

$$p^n = p^{parent(n)} + R^n s^{-n}, \tag{2}$$

where p^n is the updated 3D position of the n-th joint. $p^{parent(n)}$ is the position of its parent n-th joint. R^n is the rotation of the n-th joint with respect to its parent. s^{-n} is the transform offset of the end effectors.

This method is also applied in computer graphics, FK is used to control the movement of characters and objects, such as human, animals, and mechanical parts. By specifying the joint angles, the FK solver can calculate the position and orientation of the end effector. This allows for more natural and realistic motion, as the character's joints will move in a way that is anatomically plausible.

1) Jacobian matrix: The transformation of a kinematic chain is often represented in Jacobian matrix. The Jacobian matrix is a matrix of partial derivatives that represents the relationships between different variables in the function. It does not represent the whole function on its own, but rather describes how the variables are related to one another. Assume there is a function

$$\begin{aligned} x &= \gamma cos(\varphi), \\ y &= \gamma sin(\varphi). \end{aligned} \tag{3}$$

In order to study the relationship between x, y, we can take the partial derivative of x and y. The Jacobian matrix will become

$$J_F(\gamma, \varphi) = \begin{bmatrix} \frac{\partial x}{\partial \gamma} & \frac{\partial x}{\partial \varphi} \\ \frac{\partial y}{\partial \gamma} & \frac{\partial y}{\partial \varphi} \end{bmatrix}. \tag{4}$$

In a kinematic chain, Jacobian matrix J provides the relation between joint velocities q and velocities of end effector X [5]. And the relation is given as

$$X = Jq. \tag{5}$$

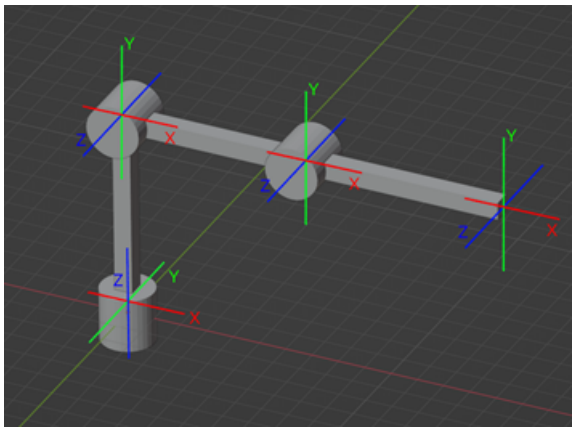


Fig. 2. The example kinematic chain for the Jacobian matrix.

Jacobian matrix is combined with the Linear velocity J_ν and Angular velocity J_ω [6] . J_ν is a $3 * n$ matrix that represents the displacement of the joints and J_ω is another $3 * n$ matrix that represents the rotation of the joint, where n is the number of joints

$$J = \begin{bmatrix} J_\nu \\ J_\omega \end{bmatrix}_{6*n} . \tag{6}$$

To complete the Jacobian matrix, the matrix can be rewrite as [7]

$$J = \begin{bmatrix} J_\nu \\ J_\omega \end{bmatrix} = \begin{bmatrix} R^0_{i-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} (d^0_n - d^0_{i-1}) \\ R^0_{i-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix}, \tag{7}$$

where i is the number of frames, $d^0_n - d^0_{i-1}$ is the displacement vector from frame 0 to frame n. It can be retrieved by the upper right part of the homogeneous transformation matrix from frame 0 to frame n, which also represents the displacement vector. R^0_{i-1} is the rotation matrix from frame 0 to frame i. The rotation matrix of R^0_0 is the identical matrix since it is the rotation matrix from frame 0 to itself (frame 0). The upper half of the matrix is the linear components (displacement), while the bottom part is the rotational components. For the part $R^0_{i-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ means the joint is moving or rotating along that z axis. Therefore, if the joint is along x axis, the matrix would become $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$.

For example, R^0_3 is the rotation matrix from frame 0 to frame 3. Then

$$\begin{aligned} R^0_3 &= R^0_0 . R^0_1 . R^0_2 . R^0_3 \\ &= R^0_1, R^0_2, R^0_3 \end{aligned} . \tag{8}$$

Assume there are total 3 joints (Fig. 2), the Jacobian matrix J could be represented as

$$J = \begin{bmatrix} J_\nu \\ J_\omega \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} (d_3^0 - d_0^0) & R_1^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} (d_3^0 - d_1^0) & R_2^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} (d_3^0 - d_2^0) \\ \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} & R_1^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} & R_2^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix}. \tag{9}$$

Overall, forward kinematics is a method for calculating and controlling the movement of characters and objects in robotics and computer graphics. It allows for the automation control of robotic movement, and also can be used to create more natural and realistic motion in computer graphics.

C. Inverse Kinematics (IK)

Inverse Kinematics (IK) is a reverse process to compute the joint orientations for the end effectors in a kinematic chain to be placed at the target position. In a kinematic chain, the end effector is the link at the top of the hierarchy that is free to move within the constraints of the chain. The purpose of IK is to determine the joint angles necessary to achieve a desired position and orientation for the end effector [4]

$$R^{1:N} = IK(P^{1:N}, p_0^{1:N}). \tag{10}$$

1) Redundancy: In IK computation, redundancy may occurs. Redundancy refers to multiple solutions exist for a given desired task value. There are 2 types of redundancy, discrete redundancy and continuous redundancy [8].

Discrete redundancy refers to the phenomenon where there are multiple solutions for a given desired task value, but the solutions can only be chosen from a finite set of possibilities. Assume there is a 2 DOF manipulator. For the given desired end effector position value $(x_{des}, y_{des}, z_{des})$, there may be multiple solutions for the joint angles that can achieve that position. For example, there are two distinct IK solutions, $(\theta_1^*, \theta_2^*, \theta_3^*)$ and $(\theta_1^{'*}, \theta_2^{'*}, \theta_3^{'*})$, that can reach the target position, while having different joint rotations. This phenomenon occurs when multiple solutions exist for a given task and is of the same dimension as the configuration space and task space. The multiple solutions can be seen as different options or alternatives to reach a same target position, it is important to choose the solution which is the closest to the current configuration or that avoids joint limits or other physical constraints.

Another type of redundancy is continuous redundancy. Continuous redundancy refers to the phenomenon where there are multiple solutions for a given desired task value, and the solutions can be continuously varied within a certain range. This can occur when there are more degrees of freedom in the kinematic chain than are needed to achieve the desired end effector position. Consider there is a gripper trying to place on a vertical line $x = x_{des}$. The desire values for x_{des} only has 1 dimension (vertical line). Therefore, there are infinite number of IK solutions for every desired value.

The difference between continuous redundancy and discrete redundancy, as in the latter case, there are only a limited number of solutions, while in continuous redundancy, the solutions can be varied within a certain range.

2) IK Computation: There are 2 different aspects of IK computation methods. Mathematical algorithm and deep learning models.

For the algorithm, it is often calculated in real-time. And it is divided into 2 types, analytical solutions and approximate solutions [9].

In analytic solution, there is a closed-form expression which represents the IK (joint variables) as a function of the end-effector pose. Therefore, equations for each joint parameter can be calculated. Then we can assign the known values such as end effector's target position, kinematic chain displacement, to the equations in order to get the joint parameters required to achieve that target pose. The equation becomes

$$\begin{aligned} q_1 &= f_1(x) \\ q_2 &= f_2(x) \\ q_3 &= f_3(x) \\ &\vdots \\ q_n &= f_n(x), \end{aligned} \tag{11}$$

where q are the joint parameters, $f(x)$ are functions of the know values.

For the approximate solutions, the joint angles corresponding to the end-effector pose is calculated numerically. Then an iterative optimization is employed to solve the IK problem. One of the approaches is Jacobian Inverse technique, which will be discussed in the next section.

Overall, the advantages of analytical solution are faster and often able to compute the solutions when approximate solution fails. But, when the DOF is higher, it becomes more difficult to solve. While approximate solution is able to handle high DOF IK. But it is slower than analytical solution.

III. Related Work

A. Selectively Damped Least Squares

The Jacobian transpose method is a technique to solve IK problem numerically. Buss [10] further developed this method to solve it effectively. They proposed the selectively damped least squares (SDLS), which is a refinement of the damped last squares method. The SDLS is an extension of the numeric filtering, which is applied to the smallest non-zero singular value of the Jacobian. A kinematic chain is controlled by the position of the end effectors, they can be denoted as vector $\vec{t} = [t_1, t_2, t_3, \ldots, t_k]^T$, where t_i is the i-th end effector's target position. Represent the joint angles as $\theta = [\theta_1, \theta_2, \theta_3, \ldots, \theta_n]^T$ and the end effector's position as $\vec{s} = [s_1, s_2, s_3, \ldots, s_k]^T$, so that the end effector positions can be denoted as $\vec{s} = \vec{s}(\theta)$. The IK problem is to find the value of θ_j, so that the target position of each end effector is $t_i = s_i(\theta)$.

However, it is possible that a solution to the function $\vec{s}$ does not exist, or there may be multiple solutions. Therefore, using an iterative method with a Jacobian matrix may be more effective. Suppose the values for $\theta, \vec{t}, \vec{s}$ are provided, the Jacobian matrix $J(\theta)$ can be calculated. Then, by keep increasing the joint angles by $\Delta\theta$, the joint angles can be estimated as

$$\vec{e} = J\Delta\theta, \tag{12}$$

where $\vec{e}$ is approximately equal to $\Delta\vec{s}$.

The Jacobian transpose method is to transpose the Jacobian matrix to find $\Delta\theta$, by adding a scalar α. Buss [11] proves that the transposed Jacobian matrix can achieve a similar result as the inversed Jacobian matrix. Using transpose instead of inverse can save a lot of processing time, since it is less computation intensive. Therefore, the function becomes

$$\Delta\theta = \alpha J^T \vec{e}. \tag{13}$$

Since the number of columns of a Jacobian matrix is determined by the DOF of the kinematic chain, it may not be able to perform matrix transpose. Therefore, pseudoinverse method is employed instead of transpose. In this case, the function becomes

$$\Delta\theta = J^T (JJ^T)^{-1} \vec{e}. \tag{14}$$

However, the pseudoinverse methods may experience stability problems near the singularities. Therefore, damped least square is used to avoid the singularities problems with pseudoinverse. Instead of finding the minimum of vector $\Delta\theta$, it finds the values of $\Delta\theta$ that minimizes the quantity by adding a damping constant λ. The damping constant is decided by the details of the kinematic chain and the target positions of end effectors. The function becomes

$$\Delta\theta = J^T (JJ^T + \lambda^2 I)^{-1} \vec{e}. \tag{15}$$

The SDLS chooses a damping value for the damped least square method depends on the difficulty of reaching the target position rather than only considering the distance to the target and the configuration of the kinematic chain. The SDLS method compares the distance between the end effector and targe position to the motion of each joint individually. After that, it adjusts the motion of the joint with more damping if the former distance is much greater than the latter distance. This is implemented by limiting the maximum change in joint angles.

Overall, the numerical IK solver using Jacobian matrix is an efficient and versatile method to solve a wide range of IK problems. It represents the relationship between the velocities of the joints and

the end-effector's position and orientation. The Jacobian transpose method is a relatively simple and computationally efficient approach that is often used as a first-order approximation for the solution. However, it has limitations particularly in the presence of singularities or near-singularities. While the SDLS method is a more sophisticated and robust approach compare with the Jacobian transpose method. However, it takes a slightly longer time to process the result.

B. Style-Based Inverse Kinematics

Style-based IK [1] proposes a learned model system to solve the IK problem. The system is designed to produce the most likely pose that satisfies a set of constraints in real-time. By training the model on different input data, different styles of IK can be produced. The system can be applied for interactive character posing, trajectory keyframing, and reconstruct pose from a 2D image.

The model is represented as a probability distribution over the latent space of all possible poses. By studying the latent space, with a preference for poses that are most similar to the latent space of poses in the training data, an accurate and realistic pose can be generated. The probability is represented using a model called a Scaled Gaussian Process Latent Variable Model (SGPLVM), which is similar to an autoencoder, parameters can be learned automatically.

1) Feature vectors: In order to provide feature vectors for IK solving, they convert the pose vector to a feature representation. The feature representation includes of the global position and orientation of the root of the kinematic chain, and the joint orientations. This process is similar to the encoding stage in autoencoder.

2) Scaled Gaussian Process Latent Variable Model (SGPLVM): The SGPLVM model maps the x values to y values, by giving the training data x_i, y_i, the model predicts the likelihood of a new y given a new x, which is similar to the encoding process of an autoencoder. To measure the similarity of x and x', a kernel function is

$$k(x, x') = \alpha \exp(-\frac{\lambda}{2} \parallel x - x' \parallel^2) + \delta_{x,x'}\beta^{-1}). \tag{16}$$

The $\delta_{x,x'} = 1$, if $x = x'$, else $\delta_{x,x'} = 0$. Therefore, the function becomes $\alpha + \beta^{-1}$ if x and x' are the same. The kernel function describes how correlated y and y' are, based on the corresponding x and x' values. There are three parameters of the kernel function. λ controls the spread of the similarity function, α controls the general correlation of pairs of points, and β controls the amount of noise in predictions.

The learning process of SGPLVM is based on the model parameters, by minimizing the function

$$L_{GP} = \frac{D}{2} \ln |K| + \frac{1}{2} \sum_k w_k^2 Y_k^T K^{-1} Y_k + \frac{1}{2} \sum_i \parallel x_i \parallel^2 + \ln \frac{\alpha\beta\gamma}{\prod_k w_k^N}, \tag{17}$$

where $x_i, \alpha, \beta, \gamma$ and w_k are unknown parameters to be optimized.

3) Pose synthesis: After the SGPLVM model learned the feature vectors, it optimized the model parameters, and generates a latent space coordinate for each pose. To generate a new pose, the model optimizes $L_{IK}(x, y(q)))$, where q is the new pose.

Ma [12] has implemented a demo in Blender using Python based on the concept of SGPLVM.

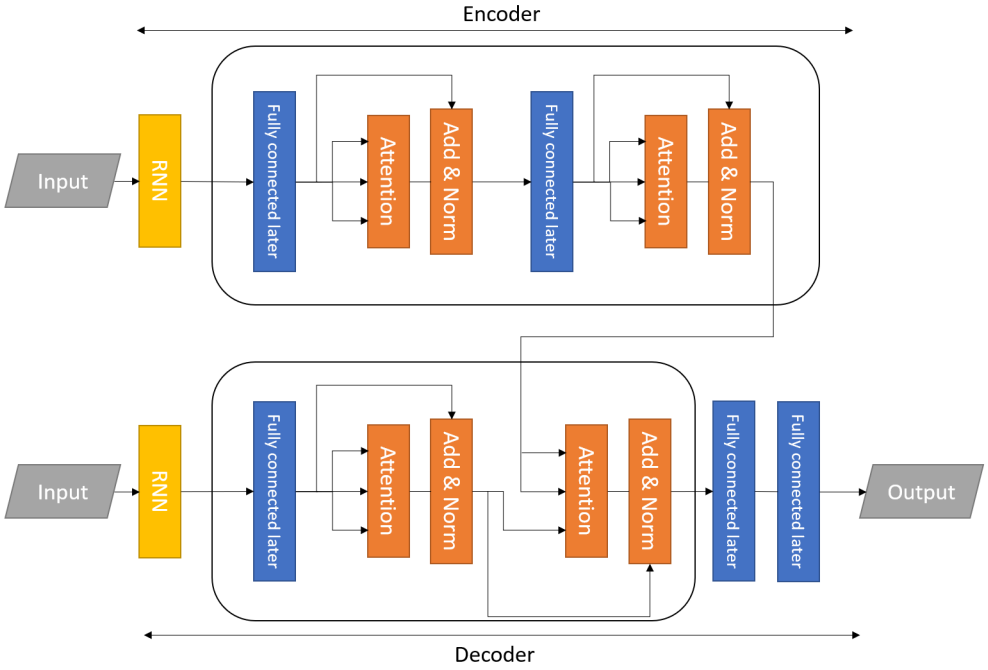


Fig. 3. Architecture of the deep transformer network proposed by Kim [2].

C. Human Motion Reconstruction using Deep Transformer Networks

Kim [2] proposed a pose reconstruction system that aims to reconstruct the human pose by only a limited number of body skeleton constraints. They developed a deep learning framework, which utilizes motion capture data and an attention mechanism network model to map only a few numbers of body constraints into human motion in a realistic manner. The attention networks, similar to the transformer networks [13] , is able to achieve accurate results.

The deep transformer network takes in only six constraints (six keypoints) on the human body. The input matrix includes the translation and rotation of human joints in 3D space. The decoder of the network uses this input to reconstruct the entire human body joint structure.

The model implements a stacked attention-based deep autoencoder system (Fig. 3). The autoencoder consists of an encoder and decoder, both of which have two identical layers with residual connections and layer normalization. For the attention layer, they use eight attention heads for the layer. The attention mechanism is used to selectively focus on the more relevant elements of the data and help to look back further from lengthy data, which leads to better prediction. The attention mechanism is similar to the transformer network, but they have several difference parts compare to the original transformer.

1) Position embedding: The authors compared their network model to the original transformer model, they found that using positional embedding (or positional encoding), which is a common method for sentence representation, caused noticeable discontinuity artifacts between consecutive frames in the motion capture data. To solve this problem, they replaced the positional embedding with a recurrent neural network (RNN) which improved the results without any discontinuity issues. The reason is that, while the transformer model works well for natural language processing (NLP) tasks, it is not well suited for motion reconstruction tasks. The positional embedding of transformer model works well in NLP tasks where the input sequence length is small, and the characteristics of each input token greatly differ from those of its nearby tokens in the

sequence. However, in motion reconstruction tasks, the input motion length is much longer, and the human motion at nearby frames is usually similar. Positional embedding in this case makes adjacent poses in the input motion discontinuous and the transformer model may regard it as noise to remove. They found that the transformer model produced good-quality motion only up to a certain frame and after that, the error increased drastically, resulting in a reconstructed motion that was different from the original motion.

2) Position embedding: The authors also found that using word embedding, a popular method for providing vector representations of words often used in transformers, was not feasible due to the large range of possible motions. Therefore, they added a fully connected layer to the network for encoding motions, instead of following the concept of word embedding.

Overall, the deep transformer network presents an attention mechanism for natural human motion reconstruction. The authors demonstrated that the proposed attention model outperforms previous RNN-based methods and the transformer network. It can observe long-term information by applying the well-designed attention layers to motion capture data.

D. HybrIK

Hybrid Analytical-neural Inverse Kinematics Solution (HybrIK) is another IK solution using neural network proposed by Li et al. [14]. It reconstructs a full 3D mesh of the human body from 2D images. The main challenge with this task is that IK problem is a non-linear process, and it is difficult to learn the abstract parameters that govern the 3D shape and pose of the body. In addition, there is often a misalignment between the image and the model, which leads to poor performance. HybrIK bridges the gap between body mesh estimation and 3D keypoint estimation. It directly transforms 3D joints to relative body-part rotations for 3D body mesh reconstruction, using the technique called twist-and-swing decomposition. The swing rotation is analytically solved with 3D joints, and the twist rotation is derived from visual cues through a neural network. This approach preserves both the accuracy of 3D pose and the realistic body structure of the parametric human model, resulting in a 3D body mesh that is aligned to the 2D image, and a more accurate 3D pose than previous methods.

The proposed network first generates a heatmap for 3D joints regression. The shape parameters β and the twist angle ϕ are learned from visual cues through fully connected layers. Then, the results are passed on to the HybrIK network, which solves for the relative rotation or the pose parameters. Finally, using both the pose and shape parameters, the framework is able to reconstruct the body mesh and the pose using Forward Kinematics or Linear Regression.

Since the movement of human joints is very complex, it often consists of high DOF. In order to simplify the problem, the authors proposed a method called Twist-and-Swing Decomposition to decompose the movement into twist and swing. After applying the decomposition, the DOF can be lowered to 1 to 2 DOF, so that the complexity is of the IK problem can be lowered.

Overall, HybrIK is able to bridge the gap between 3D keypoint estimation and body mesh estimation by employing a hybrid analytical-neural IK solution to transform 3D joint locations into accurate human body mesh, and then obtains a more accurate and realistic 3D skeleton from the reconstructed 3D mesh.

IV. Proposed method

In our research, we aim to develop a real-time Inverse Kinematics (IK) solution for estimating the 3D poses of occluded joints. To accomplish this, we propose using a multi-layer perceptron (MLP) network as the IK solver. Our proposed workflow is divided into two main stages:

1) Keypoints extraction: For this research, we use an existing real-time pose detection model, BlazePose, to extract the 3D coordinates of human skeleton keypoints from the input image.
2) IK computation: Using the extracted keypoints as input, our MLP network performs IK computation to estimate the occluded joint poses. The network consists of multiple layers of neurons that process the input data and generate the final output.

As the graph shows in Fig. 4, we first use an existing real-time pose detection model BlazePose to extract the human skeleton keypoints coordinates in 3D space. This model is trained to detect and estimate the position of different joints in a human body, such as the head, shoulders, elbows, wrists, hips, knees, and ankles. The output of this model is a set of keypoints in 3D space, which represent the locations of the joints in the XYZ dimensions. It will be introduced later in this section. Then we input the detected keypoints x in order to get the output y, which is the estimated joint's 3D coordinates, using MLP.

In this section, we will provide an overview of the multi-layer perceptron (MLP) algorithm, explain the reasoning behind our selection of BlazePose as the real-time pose detection model, and detail the 3DPW dataset that we utilized in our experiment. Furthermore, we will detail the workflow we implemented to address the problem at hand, including the specific steps and techniques used.

A. Multi-layer Perceptron (MLP)

A MLP is a type of feed forward network that consists of multiple layers of interconnected neurons or nodes [15]. These layers are often referred to as the input, hidden, and output layers.

The input layer receives the input data, the hidden layers process the data, and the output layer produces the final output or prediction. The MLP is trained using a supervised learning algorithm, such as backpropagation, to adjust the weights of the connections between the nodes in order to minimize the difference

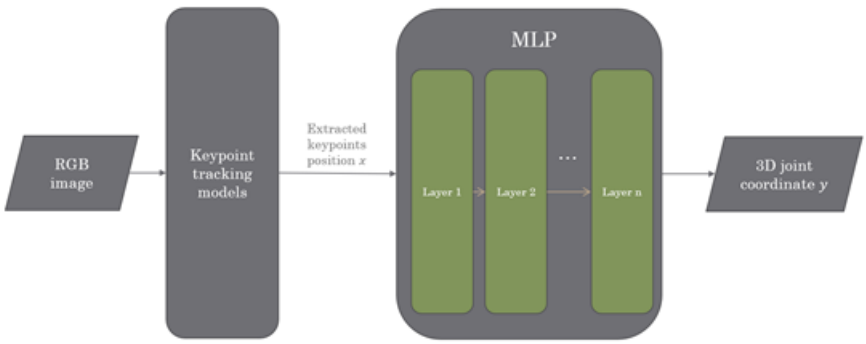


Fig. 4. Proposed network structure.

between the predicted output and the actual output. MLPs are commonly used for tasks such as image recognition, speech recognition and natural language processing.

The structure of MLP is illustrated in Fig. 5. A MLP network typically consists of an input layer, one or more hidden layers, and an output layer. The input layer receives the input vector x, and passes it through to the neurons in the hidden layers. Each layer in the MLP is fully connected, meaning that each neuron in a given layer is connected to every neuron in the next layer. The number of neurons in each layer can be vary, allowing for flexibility in the design of the network. The final output of the MLP is represented by the output vector y. This network architecture is capable of approximating complex, non-linear functions and can be trained using supervised learning techniques.

1) Back Propagation: The fully connected network relies back propagation heavily for training. The purpose of training is to find the combination of weights that result in the smallest error when trying to model a relationship. The back propagation is a common and straightforward algorithm to train the MLP. It uses gradient descent to locate the minimum point of the error surface.

Back propagation consists of a series of steps [16], such as initializing the network weights, providing input, calculating error, propagating error back through the network, adjusting weights, and repeating the process until the overall error is satisfactorily small. It also consists of 2 adjustable parameters, learning rate and momentum term, to avoid the optimization algorithm getting trapped in a local minimum.

The learning rate controls the step size taken during the iterative gradient descent process. If the parameter is set too large, the network error will change erratically and may miss the global minimum. If it is set too small, training will take a longer time. The momentum term is used to help the gradient descent process escape local minima by adding a proportion of the previous weight change to the current weight change. It helps the network to escape from local minima and move towards the global minimum.

2) Neuron: The structure of a neuron in MLP is illustrated in Fig. 6. A neuron is a fundamental unit that represents a single node in the MLP network. Each neuron receives input vector from other neurons or from the input layer, processes that input using an activation function, and produces an output that is passed to other neurons or to the output layer.

The structure of a neuron in an MLP typically consists of the following components:

- Inputs: Each neuron receives input from other neurons or from the input layer in the form of a vector of values.
- Weights: Each input to a neuron is multiplied by a weight, which is a scalar value that represents the strength of the connection between the input and the neuron.
- Bias: A bias term is added to the weighted input. This is a scalar value that allows the neuron to shift the input along the activation function.
- Summation: A scalar value that sums up the weighted input and bias.
- Activation function: The output of the neuron is computed as a non-linear function of the sum of the weighted input and the bias.
- Output: The output of the neuron. It is passed to the next layer of neurons or to the output layer of the network.

The overall process of a neuron can be represented by the following equation

$$y = \sum_{i=1}^{n} (x_i w_i) + b, \tag{18}$$

where x is the input vector, w is the weight vector calculated by the back propagation, and b is the bias.

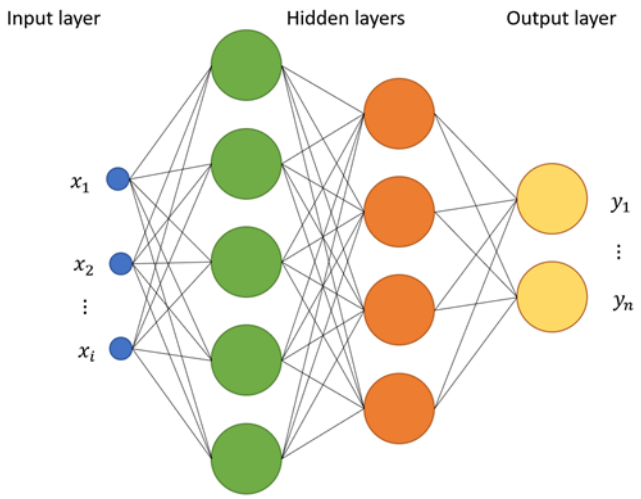


Fig. 5. Structure of MLP.

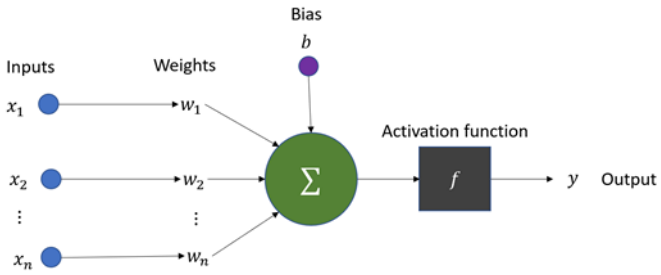


Fig. 6. Structure of a neuron in MLP.

3) Activation Function: Activation functions allow MLPs to model complex relationships between input and output, as they can output any kind of mathematical function. As Fig. 6, the activation function is applied to the output of each neuron, determining whether or not the neuron should be activated and pass its output to the next layer.

Commonly used activation functions include the sigmoid, ReLU, and tanh functions. Each activation function has its own unique properties and it's important to choose the suitable activation function according to the specific problem. For example, the sigmoid function is often applied for binary classification problems and produces output between 0 and 1, while the ReLU function is commonly used in deep networks and produces output only when the input is positive.

Overall, MLP is a network designed for prediction, regression, and classification. It can accurately reproduce any measurable relationship and is particularly useful in applications where a full theoretical model cannot be constructed, especially when dealing with non-linear problems.

B. BlazePose

BlazePose is a real-time human pose estimation model developed by Google's MediaPipe team [17] that utilizes a single-stage, bottom-up approach for detecting the 2D and 3D positions of multiple body joints in an image or video. It is one of the pre-built models in MediaPipe, a cross-platform framework for building multimodal pipelines that can handle tasks such as image, video, and audio. In our research, we are utilizing BlazePose as the first stage of our workflow to extract the 3D keypoints coordinates for further computation. This section will provide a brief introduction to the BlazePose model and its capabilities.

1) Pipeline and NMS detector: The pipeline of BlazePose consists of a lightweight body pose detector followed by a pose tracker network. The pipeline first uses the detector to detect the bounding box of a relatively rigid body part, such as the human face or torso. If the detector indicates that there is no human present in the current frame, the pipeline will re-run the detector network on the next frame. Once a human is detected, the pose tracker network is used to predict keypoint coordinates, the presence of the person on the current frame, and the refined region of interest for the current frame. The pipeline continues to use the tracker network to track the human pose until the tracker indicates that the person is no longer present in the frame, at which point the detector is run again.

The model employs the Non-Maximum Suppression (NMS) algorithm to detect the bounding box of a human body in an image. However, this approach is not suitable for scenarios with highly articulated human poses, such as waving or hugging, as it results in multiple ambiguous boxes that satisfy the Intersection over Union (IoU) threshold for the NMS algorithm. To overcome this limitation, the model makes the assumption that the head of the person should always be visible in order to accurately determine the presence of a human in a complex scene. To achieve this, the model uses a fast on-device face detector which is used as a proxy for detecting the presence of a person in the image.

2) Neural Network Architecture: The architecture of the model is illustrated in Fig. 7. It involves a combination of heatmap, offset, and regression approach to predict the position of 33 keypoints on a human body. A heatmap and offset loss are used during the training stage, but the corresponding output layers are removed before running the inference. This approach uses the heatmap to supervise a lightweight embedding, which is then utilized by the regression encoder network. The model also includes skip-connections between all the stages to balance high-level and low-level features, and gradient-stopping connections to improve the heatmap predictions and increase the accuracy of coordinate regression.

In summary, BlazePose provides high accuracy and real-time human keypoint detection in both static images and videos. It can detect only half of the human body correctly, which some other existing pose recognition models could not. Additionally, its ability to estimate both 2D and 3D positions of multiple body joints provides additional flexibility for our research. Therefore, we selected BlazePose for the keypoint detection for our own research.

C. Dataset

Our research utilizes the 3D Poses in the Wild (3DPW) dataset as a benchmark for evaluating the performance of our 3D human pose estimation algorithm. The 3DPW dataset is a comprehensive dataset that includes high-quality 3D human pose annotations for a wide range of activities, viewpoints, and body shapes. The dataset was created by capturing motion capture data of actors performing various actions and then synthesizing images of the actors in different poses and settings. These images come with 3D keypoint annotations, which can be used to assess the accuracy of various 3D human pose estimation algorithms. This dataset is commonly used in research to evaluate the performance of different 3D human pose estimation methods.

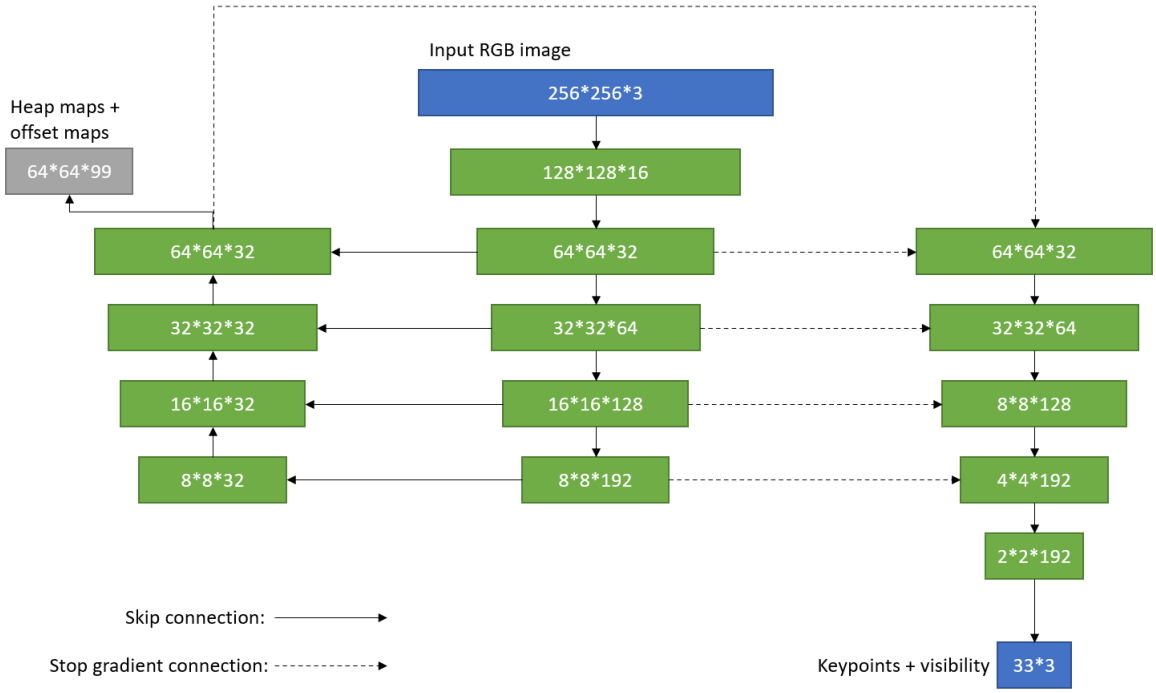


Fig. 7. Network architecture of BlazePose [17].

The 3DPW dataset was collected using a method proposed by Marcard [18] . The method uses a moving camera and inertial measurement units (IMUs) to recover accurate 3D human pose, which was then evaluated using the 3DPW dataset. The dataset contains over 51,000 frames with 3D pose accurate to 26mm, making it a highly accurate benchmark for image-based 3D pose estimation.

1) Data content: The 3DPW dataset contains total 60 motion sequences. The sequences are organized as imageFiles and sequenceFiles folder. The imageFiles contains 2D RGB images of the motion sequence, while the sequenceFiles contains data of the motion sequences. Each sequence is stored as a pkl file.

According to Dean [19] , each pkl file is a dictionary in Python, and it contains with the following fields:

- sequence: name of the sequence
- betas
- poses
- trans
- poses_60hz
- trans_60hz
- betas_clothed
- v_templates_clothed
- gender: actor genders
- texture_maps
- poses2d
- jointPositions: List of $M \times N \times (24 \times 3)$. M actors of N frames of 24 joints in XYZ coordinates
- img_fram_ids

TABLE I: Different list format of the training data.

1	actor	frame	XYZ coordiantes	
2	actor × frame	XYZ coordiantes		
3	actor × frame	X coordinate	Y coordinate	Z coordinate
4	actor × frame	X coordinate (local)	Y coordinate (local)	Z coordinate (local)

- cam_poses: camera extrinsic for each image frame (I frames $\times$ homogenous transformation matrix of the camera)
- campose_valid
- cam_instrinsics: camera intrinsic matrix

Each sequence consists of either one or two actors as model. We are only using the 60Hz data in order to get more data for training.

2) Dataset pre-processing: For our research, we have chosen to separate motion sequences with multiple models in the 3DPW dataset into individual sequences (Algorithm 1). This decision was made because our focus is not on detecting multiple people in an image, and separating the sequences simplifies the training and testing process. Additionally, as 3D human pose datasets are limited, we aim to maximize the amount of training data available by separating the sequences.

The 3D joint positions in the dataset are presented in a complex format, a list of $M \times N \times (24 \times 3)$ float values (row 1 of TABLE I), which can be difficult to manipulate and use in training models. In order to simplify the data and make it more suitable for training, we pre-processed it by concatenating the first and second dimensions of the list, resulting in a shape of $(M, N, 72)$ (row 2 of TABLE I). Next, we separate the list into XYZ coordinates for ease of calculation, resulting in a shape of $(M, N, 24, 3)$ (row 3 of TABLE I). Finally, we convert the joint positions from world coordinates to local coordinates with respect to each individual model (row 4 of TABLE I) to compare the training performance with models in world coordinates.

Algorithm 1 Dataset pre-process

```
data = pickle.load(file)
frame = data.get('jointPositions')
if convert to local == true then
    convert coordinates to local coordinates
    combined = combineDimension(input dimension)
    reshaped = combined.reshape(length(combined), 24, 3)
```

To convert from world coordinates to local coordinates, the keypoint of hip is taken as the origin of the model

$$\begin{aligned} P_l &= [x_i', y_i', z_i']^T, \\ P_w &= [x_i, y_i, z_i]^T, \\ O &= [x_o, y_o, z_o]^T, \end{aligned} \tag{19}$$

$$\begin{aligned} P_l &= Pw - O, \\ [x_i', y_i', z_i']^T &= [x_i - x_0, y_i - y_0, z_i - z_0]^T, \end{aligned}$$

where P_l is the position of the local coordinates, P_w is the position of the world coordinates, $[x_o, y_o, z_o]$ is the XYZ coordinates of the hip since it is the first keypoint in the 3DPW dataset. Therefore, the local coordinates are the offset subtracted by the world coordinates.

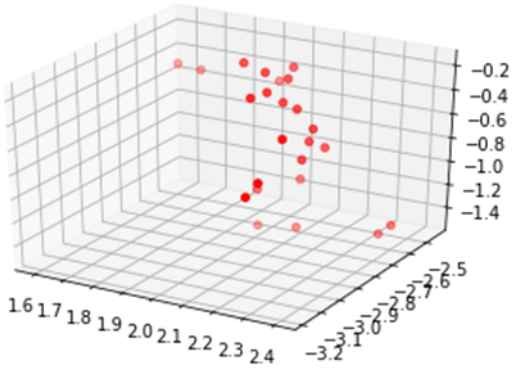


Fig. 8. 3D coordinates of one of the frame in dataset

3) Data loader: The 3DPW dataset pre-processed data is split into train, test, and validation datasets. The train set is used to train the model, where the model learns and sets the weights and biases of the network. The test set is used for frequent evaluation of the model, to test its accuracy and adjust the learning. The validation set provides a standard to evaluate the model. This allows us to evaluate the model's generalization ability, how well it can generalize to new unseen data.

It is important to have a separate validation set, as it can help to prevent overfitting, which is when the model performs well on the training data but poorly on unseen data. This helps to ensure that the model is generalizable and can perform well on new, unseen data.

The dataset contains 60 different sequences, total of 149240 frames. We split 80438 frames for train dataset, 47086 frame for test dataset, and 21716 frames for validation dataset.

The train dataset is shuffled in every training loop. Shuffling the training data has several advantages:

- Preventing overfitting: Shuffling the data breaks up any such patterns and forces the model to focus on the underlying patterns.
- Improving generalization: Exposes the model to a wider variety of examples, which can help it generalize better to new data.
- Stochastic optimization: When the data is passed to the model in a specific order, the model will follow the same path of optimization every time. Shuffling the data ensures that the model will see different examples in different orders.
- Making use of all the data: If the data is sorted in a way that the same class or label is grouped together, the model will not be able to learn the patterns of the other classes or labels. Shuffling the data ensures that the model sees all the classes or labels.

Additionally, the train dataset set "drop_last" to true, so that it ignores the last batch if the number of samples in the dataset is not divisible by the batch size. It ensures that all batches have the same size and preventing any potential issues., such as incorrect gradients or unstable training.

D. Proposed workflow

Our proposed model is designed to be lightweight and efficient, making it well-suited for real-time applications. By using existing keypoint detection models, our model is able to obtain the upper body joints

Algorithm 2 Application workflow for one frame

```
input RGB image
extract all keypoints coordinates
if arm is occluded == true then
    occludedJoints = model(occluded joint)
    finalized joints 3D coordinates
```

coordinates in an RGB image. If the visibility or confidence value of the elbow keypoint is too low, our model is able to produce accurate coordinate estimation by using the obtained 3D coordinates.

As the pseudo code in Algorithm 2 shows, the proposed method utilizes 3D keypoint coordinates to estimate the position of the keypoints. This allows for integration with any existing keypoint detection model, allowing the proposed network to predict keypoint coordinates based on the extracted keypoints. However, in practical applications, it is recommended to only activate the proposed model when the confidence level or visibility of the elbow joint is low, in order to maintain the accuracy of tracking in normal situations. This is because the IK process is only an estimation without 100% accurate solution and may not always match the original ground truth results.

MLP is applied to the model as a neural network to estimate the position of a keypoint. The network takes in two 3D vectors as input, which are processed through a series of hidden layers, and outputs a single 3D vector as the final result. The input, represented as x, has a dimension of 3×2, while the output, represented as y, has a dimension of 3. The input is flattened into 1 dimension in order to feed into the hidden layer for the process.

The hyperparameters of the model are referenced from Gholami [20], they control the position of the delta robot by computing the IK of it. We take the parameters and model layer structure as reference and modified the hyperparameters by conducting several experiments. A total of six different networks has been trained in order to find out the best combination of hyperparameters.

The hyperparameters of the model, such as the number of hidden layers and neurons, were referenced from a previous study by Gholami [20], the paper controls a delta robot's position be computing the IK. However, we conducted multiple experiments to fine-tune these parameters to optimize the performance of the model. This involved training a total of six different networks with different combinations of hyperparameters.

The hyperparameters of our model are as follow:

- Number of inputs: 6
- Number of outputs: 3
- Number of hidden layers: 3
- Number of hidden layers: [1, 2, 3]
- Number of neurons in each layer: [12, 24, 12]
- Activation function: [Softmax, Softmax, Softmax]
- Batch size: 16
- Learning rate: 0.00015

We have tried to adjust the learning rate during the training using a learning rate scheduler in Pytorch. However, the result was not as good as using a fixed learning rate. Therefore, we keep the learning rate as 0.00015 for the training process. Our proposed model architecture is displaye d in Fig. 9.

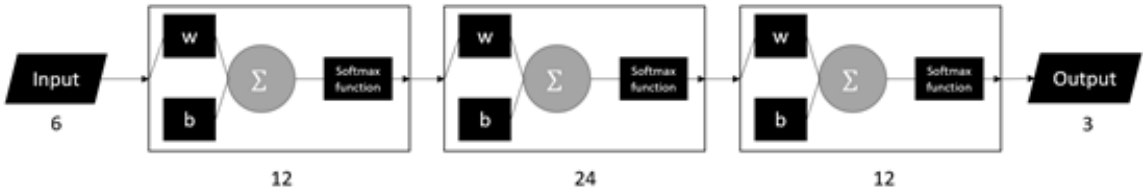


Fig. 9. Layer structure of the MLP.

V. RESULTS

In this study, we aimed to evaluate the accuracy of our proposed model for solving the IK problem in real-time. To accomplish this, we conducted a series of experiments using the 3DPW dataset as our test data. The results of these experiments are presented below, along with a discussion of the key findings.

A. Loss function

The 3DPW dataset suggests us to use joint error metric, mesh error metric, mesh error metric unposed, and orientation error metric for the evaluation. For our research, we are using joint error metric for the evaluation of the model.

Joint error metric is the mean Euclidean distance between predicted joints and the ground truth joints. The implementation is as

$$Euclidean\ distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}. \tag{20}$$

We can simplify the function to $Y_i = [x_i, y_i, z_i]$, where Y_i is a vector in 3 dimensions. Therefore, it can be represented as

$$\begin{aligned} Euclidean\ distance &= \sqrt{(\widehat{Y}_i - Y_i)^2}, \\ Joint\ metric\ error &= \frac{1}{n}\sum_{i=1}^{n}\sqrt{(\widehat{Y}_i - Y_i)^2}. \end{aligned} \tag{21}$$

Since it is similar to the mean square error loss (MSE loss) but without the square root, we use MSE loss instead of joint metric error because they provide the same geometry information. The function of MSE loss can be denoted as

$$MSELoss = \frac{1}{n}\sum_{i=1}^{n}(\widehat{Y}_i - Y_i)^2. \tag{22}$$

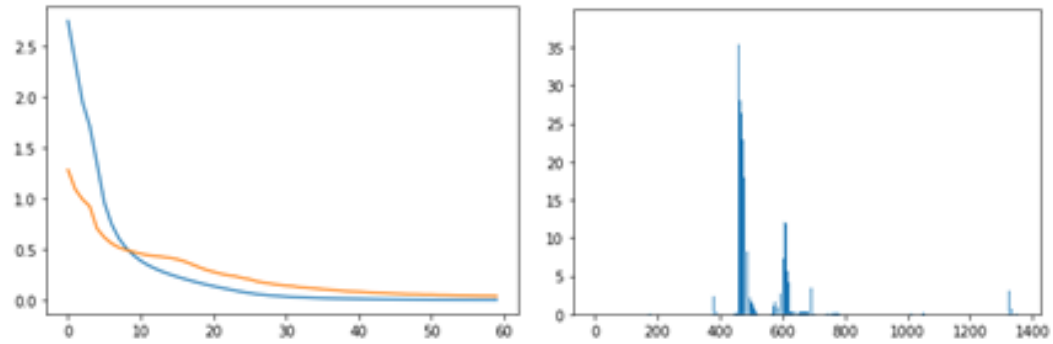
B. Single arm or both arms

In the initial experiment, we evaluated the model accuracy of training each arm of the model separately or both arms together simultaneously. The network architecture was the same for both methods, however, we doubled the number of neurons in the model of both arms as it was processing double the amount of input data.

The results from TABLE II indicate that training each arm of the model separately results in lower loss compared to training both arms together. This is evident from the test loss and validation loss values, which show that the loss is roughly double when training both arms as compared to training a single arm. This

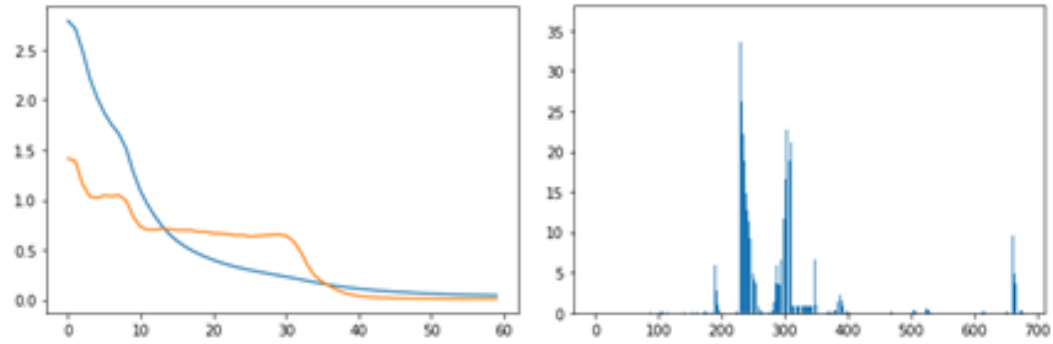
TABLE II: Loss values between training single arm and both arms together.

	Train loss	Test loss	Validation loss
Single arm	0.006	0.092	0.0466
Both arms	0.046	0.175	0.0827



(a) Train loss (blue) & test loss (yellow) of single arm

(b) Validation loss of single arm



(c) Train loss (blue) & test loss (yellow) of both arms

(d) Validation loss of both arms

Fig. 10. Loss value graphs between training single arm and both arms together.

is expected as the output of the network increases from 3 to 6 when training both arms. Additionally, the validation loss graphs in Fig. 10 (b) and (d) illustrate that the error values are similar for both the corresponding patches. This also proves that the learning process of the network between two methods are similar.

We finally decided to use a single arm for training the network. This decision was based on the fact that it is more common and practical for the model to take only a single arm as input and output, as it provides simpler geometry information in the 3D space for the model to learn. Additionally, due to the rarity of 3D human pose datasets, we sought to maximize the use of our training data by separating two arms coordinates from an actor and using a single arm for training.

C. Local coordinates or world coordinates

As we have mentioned in section ”Data pre-processing”, we converted the coordinates in dataset from world coordinates to local coordinates. We then compare the network’s training result between world coordinates and local coordinates.

The keypoint of the hip is taken as the origin of the model, instead of the world origin $[0, 0, 0]$. Since in the practical applications, the input data often locate in the world coordinates, we did not convert the validation dataset into local coordinates. We tested three combinations of the local and world data: local coordinates train dataset and local coordinates test dataset, local coordinates train dataset and world coordinates test dataset, world coordinates train dataset and world coordinates test dataset.

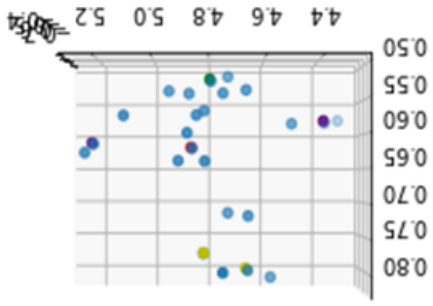


Fig. 11. Visualization of the image frame after the view matrix transformation.
(It is upside-downed and left-right flipped since it is multiplied by the camera transform matrix).

The results from TABLE III shows that training with local coordinates and testing with local coordinates results in the highest validation loss, indicating that the model is not able to fully adapt to the different coordinate systems, resulting in a decrease in performance. Overall, training with local coordinates and testing with world coordinates results in the best accuracy in validation dataset. Therefore, this combination is applied for our training. However, the differences between the three different combinations are not significant and fall within the margin of error. They would not have a significant impact on the final accuracy.

1) View matrix transformation: The view matrix is a transformation matrix that is used to convert the coordinates of a 3D point from world coordinates to view coordinates. The view matrix represents the position and orientation of the virtual camera in the 3D world. It defines a transformation that moves the entire scene so that the camera is located at the origin and is looking along the negative z-axis (or y-axis in some 3D software). This means that any point in the scene that is transformed by the view matrix will end up in a coordinate system where the camera is at the origin, looking down the negative z-axis. The visualized result is displayed in Fig. 11.

Since the 3DPW dataset provides the camera extrinsic and intrinsic of every frame, we can compute the view matrix transform by taking dot product with the camera extrinsic, which is the homogeneous transformation of the camera in 3D world

$$\widehat{P}_i - P_i.E, \tag{23}$$

where $\widehat{P}_i$ is the point converted into view transform, and E is the 4×4 homogeneous transformation matrix.

We also considered transforming the dataset to a view-projection matrix, but ultimately decided to drop it. This is because the dataset would become too generalized, losing valuable variations, such as orientation of the model. Additionally, in practical applications, obtaining camera transformation information may not always be possible. Therefore, we chose not to use this transformation for the training process.

TABLE III: Loss values between training with local coordinates dataset and world coordinates dataset.

	Localtrain + local test	Localtrain + world test	Worldtrain + world test
Train loss	0.006	0.007	0.007
Test loss	0.092	0.043	0.059
Validation loss	0.0466	0.0296	0.0353

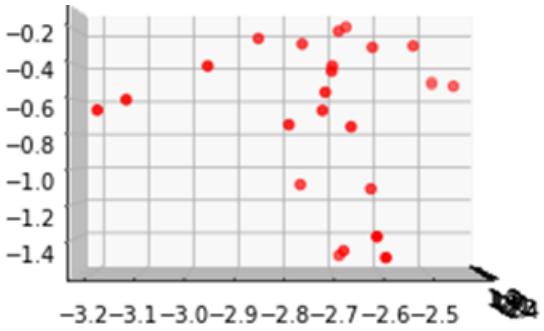
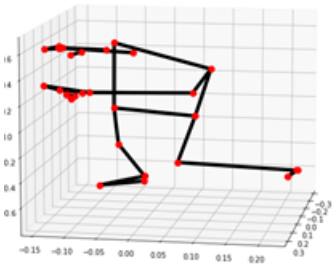
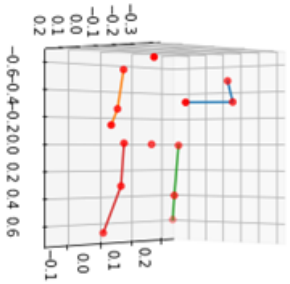


Fig. 12. Front view of the 3D coordinates.



(a) 3D keypoints captured using BlazePose [17]

(b) 2D keypoints captured by BlazePose [17]



(c) 3D keypoints captured using BlazePose [17] (The figure is rotated for better visualization)

Fig. 13. Visualize the keypoints captured by BlazePose.

D. Applications

The proposed model can be utilized with existing pose tracking models. We are using one of the image sequence from 3DPW validation dataset as the example for demonstration.

The joint coordinates from the example sequence are illustrated in Fig.12. The coordinates captured by BlazePose in 3D space are illustrated in Fig. 13. As BlazePose is able to provide the visibility value of each joint, TABLE IV shows the visibility of the joints in right arms, we can determine how confident BlazePose is in detecting the position of each joint. In this particular example, BlazePose has a low visibility value for the right elbow because it is obscured by the body, as can be seen in the accompanying 2D image. This is a common problem in human keypoint detection, particularly when the subject is in a complex pose or partially obscured by other objects.

TABLE IV: Visibility of the right arm joints.

Joint	Visibility
RIGHT_SHOULDER	0.9999679327011108
RIGHT_ELBOW	0.19555442035198212
RIGHT_WRIST	0.5551260709762573

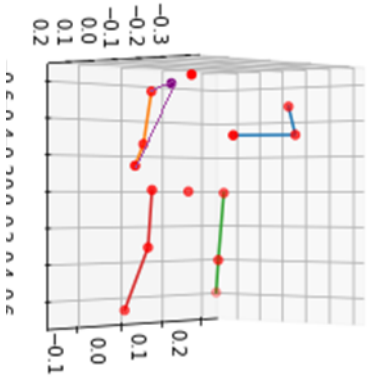


Fig. 14. Result utilizing with existing body tracking model.
(The figure is rotated for better visualization)

We then extract the joint coordinates of the right arm, input them into our proposed model, then the estimated joint position of right arm elbow is computed. For this example, the result is illustrated in Fig. 14. The purple line indicates the new estimate position of the occluded elbow. Due to the scale difference along each axis, the result might be distorted. However, the MSE loss value is 0.0418, which is similar to the training result.

We also took the same frame from the validation dataset, and directly predict the right elbow joint. The result is illustrated in Fig. 15. The red line is the ground truth value, while the purple line is the predicted value. In this case, the loss value is 0.013.

The proposed model's loss value is higher when using keypoints detected by BlazePose compared to the original keypoints from the dataset. This is likely due to the fact that 3DPW and BlazePose utilize different body topologies. BlazePose utilizes a minimal number of keypoints (33 points, including the face) while 3DPW employs 24 points based on SMPL [21], to represent the human body. As the proposed model is only trained on the 3DPW dataset, it may not have sufficient generalization capabilities for other cases.

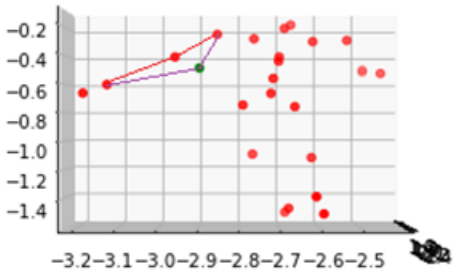


Fig. 15. Predicted result directly using the coordinates from the validation dataset.

E. Computation performance

The proposed model is optimized for real-time performance, which is crucial for practical applications. We have designed a lightweight and efficient model that can run on both CPU and GPU, while maintaining high accuracy.

This is in contrast to existing IK solvers, which are computationally intensive and can take a long time to process even a short video. For example, the popular HybrIK model can take up to 10 minutes to process a 40-second, 30 fps video using a GTX 1080 desktop PC, which is only processing around 2 frames per second.

However, our model can process up to 1000 frames per second on a standard Intel Core i7 CPU. It processes one frame in about 0.001 second in CPU. This high performance makes our model well suited for real-time applications such as robotics, animation, and motion capture.

VI. Conclusion

The research presented in this thesis aimed to investigate the upper body limbs position estimation in real-time using Inverse Kinematics (IK) through neural networks. We are able to study the concept of kinematic chain, and different existing IK solving methods. We were also able to reconstruct an accurate result according to the training loss and experiments carried in the previous section. These results contribute to our understanding of IK by computing it in real-time with a low error value, and without manually set up constraints.

In this research, we investigated various methods for solving the IK problem, including both numerical and data-driven machine learning approaches. To address the slow processing time problem of most of the existing deep learning methods, we proposed a lightweight machine learning model as a real-time IK solver. We introduced the use of the 3DPW dataset for training, as well as the pre-processing steps applied to the data. We also discussed the multi-layer perceptron (MLP) architecture applied in the proposed network and the hyperparameters used in the model. Additionally, we presented the results of our model's training and demonstrated its practical application in combination with existing 3D body tracking models.

Our proposed method was able to achieve real-time IK computation and provide accurate results for the upper body. We achieved our objective of providing fast computation performance, with our model processing one frame in approximately 0.001 seconds on a standard Intel Core i7 CPU. The network architecture is based on MLP, which is efficient and able to deliver accurate results. Additionally, our proposed method has the advantage of not requiring manual setup of joint constraints, such as rotation angle limits or joint orientation direction, unlike traditional IK solvers in 3D software.

Although the proposed method has been able to achieve real-time IK computation and provide accurate results for the upper body, there are still areas for improvement. One limitation is that the network is currently only focused on the upper body, despite it can estimate the lower body joints, the accuracy is not as high. Additionally, while the proposed method has been able to achieve good results, the improvement of accuracy is still possible. For example, incorporating more dataset and fine-tuning the network architecture could lead to more accurate results. Moreover, the proposed method is currently trained on a specific dataset, which may not generalize well to other datasets or real-world scenarios.

In future work, we plan to explore encoding the dataset into a latent space in order to gain a better understanding of its distribution and improve generalization. Additionally, we aim to expand the model's capabilities to estimate full body joints. To improve the accuracy, we also plan to integrate training with other 3D human pose data sets, estimation with different network architectures, and using other network training techniques. Furthermore, we will also explore the integration of temporal information in the model to improve the real-time performance of IK solver.

Overall, the research presented in this thesis has successfully proposed a lightweight machine learning model as a real-time IK solver, which can provide accurate results for upper body limb position estimation. The proposed method is based on MLP network and has been trained on the 3DPW dataset. The model has been tested and demonstrated its practical application in combination with existing 3D body tracking models. Additionally, the proposed method is able to perform fast computation, processing one frame in approximately 0.001 seconds on a standard CPU and does not require manual setup of joint constraints. While there are still areas for improvement, such as expanding the model's capabilities to estimate full body joints and increasing accuracy, the results of this research provide a solid foundation for our further exploration and development in this area.

REFERENCES

[1] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popivic, "Style-based Inverse Kinematics," ACM Siggraph, pp.523-531, Aug. 2004.

[2] S. U. Kim, H. Jang, H. Im, and J. Kim, "Human motion reconstruction using deep transformer networks," Pattern Recognition Letters, vol. 150, pp. 162–169, Oct. 2021, doi: 10.1016/j.patrec.2021.06.018.

[3] F. Reuleaux and A. B. W. Kennedy, "The kinematics of machinery: Outlines of a theory of machines," London: Macmillan, 1876.

[4] R. Villegas, J. Yang, D. Ceylan, and H. Lee, "Neural kinematic networks for unsupervised motion retargetting," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018.

[5] S. Krishna, "Jacobian," ROBOTICS. https://www.rosroboticslearning.com/jacobian (accessed Jan. 10, 2013).

[6] A. L. Sears-Collins, "The Ultimate Guide to Jacobian Matrices for Robotics," Automatic Addison. https://automaticaddison.com/the-ultimate-guide-to-jacobian-matrices-for-robotics/ (accessed Jan. 10, 2013).

[7] D. E. Whitney, "Resolved Motion Rate Control of Manipulators and Human Prostheses," IEEE Transactions on Man Machine Systems, vol. 10, no. 2, pp. 47–53, Jun. 1969, doi: 10.1109/tmms.1969.299896.

[8] "Inverse kinematics," Introduction to Open-Source Robotics. http://osrobotics.org/osr/kinematics/inverse_kinematics.html (accessed Jan. 10, 2023).

[9] S. Krishna, "Inverse Kinematics", ROBOTICS. https://www.rosroboticslearning.com/inverse-kinematics (accessed Jan. 10, 2013).

[10] S. R. Buss and J.-S. Kim, "Selectively Damped Least Squares for Inverse Kinematics," Journal of Graphics Tools, vol. 10, no. 3, pp. 37–49, Jan. 2005, doi: 10.1080/2151237x.2005.10129202.

[11] S. R. Buzz, "Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods." Oct. 2009.

[12] Y. Ma, "Latent Pose," GitHub, Nov. 10, 2022. https://github.com/maajor/latent-pose (accessed Jan. 10, 2023).

[13] A. Vaswani et al., "Attention Is All You Need," Advances in Neural Information Processing Systems, vol. 2017, pp. 5999–6009, Dec. 2017.

[14] J. Li, C. Xu, Z. Chen, S. Bian, L. Yang, and C. Lu, "HybrIK: A Hybrid Analytical-Neural Inverse Kinematics Solution for 3D Human Pose and Shape Estimation," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3383–3393, Jun. 2021.

[15] M. W. Gardner and S. R. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," Atmospheric Environment, vol. 32, no. 14–15, pp. 2627–2636, Aug. 1998, doi: 10.1016/s1352-2310(97)00447-0.

[16] C. M. Bishop, Neural Networks for Pattern Recognition. Clarendon Press, 1995.

[17] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, "BlazePose: On-device Real-time Body Pose tracking," CV4ARVR, 2020. https://xr.cornell.edu/workshop/2020/papers (accessed Jan. 10, 2023).

[18] T. von Marcard, R. Henschel, M. J. Black, B. Rosenhahn, and G. Pons-Moll, "Recovering Accurate 3D Human Pose in The Wild Using IMUs and a Moving Camera," in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 614-631, Sep. 2018.

[19] Dean, "Dean/3DPW-3D_Pose_Data," DAGsHub, 2022. https://dagshub.com/Dean/3DPW-3D_Pose_Data (accessed Jan. 22, 2023).

[20] A. Gholami, T. Homayouni, R. Ehsani, and J.-Q. Sun, "Inverse Kinematic Control of a Delta Robot Using Neural Networks in Real-Time," Robotics, vol. 10, no. 4, p. 115, Oct. 2021, doi: 10.3390/robotics10040115.

[21] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: a skinned multi-person linear model," ACM Transactions on Graphics, vol. 34, no. 6, pp. 1–16, Nov. 2015, doi: 10.1145/2816795.2818013.

APPENDIX

TABLE V: Sequence name of each dataset.

Train	Test	Validation
downtown_arguing_00	courtyard_arguing_00	courtyard_basketball_01
downtown_bar_00	courtyard_backpack_00	courtyard_dancing_00
downtown_bus_00	courtyard_basketball_00	courtyard_drinking_00
downtown_cafe_00	courtyard_bodyScannerMotions_00	courtyard_hug_00
downtown_car_00	courtyard_box_00	courtyard_jumpBench_01
downtown_crossStreets_00	courtyard_capoeira_00	court yard_rangeOfMotions_01
downtown_downstairs_00	courtyard_captureSelfies_00	downtown_walkDownhill_00
downtown_enterShop_00	courtyard_dancing_01	outdoors_crosscountry_00
downtown_rampAndStairs_00	courtyard_giveDirections_00	outdoors_freestyle_01
downtown_runForBus_00	courtyard_golf_00	outdoors_golf_00
downtown_runForBus_01	courtyard_goodNews_00	outdoors_parcours_00
downtown_sitOnStairs_00	courtyard_jacket_00	outdoors_parcours_01
downtown_stairs_00	courtyard_laceShoe_00	
downtown_upstairs_00	courtyard_rangeOfMotions_00	
downtown_walkBridge_01	courtyard_relaxOnBench_00	
downtown_walking_00	courtyard_relaxOnBench_01	
downtown_walkUphill_00	courtyard_shakeHands_00	
downtown_warmWelcome_00	courtyard_warmWelcome_00	
downtown_weeklyMarket_00	outdoors_climbing_00	
downtown_windowShopping_00	outdoors_climbing_01	
flat_guitar_01	outdoors_climbing_02	
flat_packBags_00	outdoors_freestyle_00	
office_phoneCall_00	outdoors_slalom_00	
outdoors_fencing_01	outdoors_slalom_01	