FPT UNIVERSITY

MOTION CAPTURE OPTIMIZATION

by

Anh. Phan Ngoc (A.P.N.),
Nguyen. Nguyen Trung (N.N.T.)

Supervisors: PhD. Nguyen. Le Phu

THE FPT UNIVERSITY HO CHI MINH CITY

December 2023

MOTION CAPTURE OPTIMIZATION

by

Anh. Phan Ngoc (A.P.N.),
Nguyen. Nguyen Trung (N.N.T.)

Supervisors: PhD. Nguyen. Le Phu

A final capstone project submitted in partial fulfillment of the requirement for the Degree of Bachelor of Artificial Intelligence in Computer Science

DEPARTMENT OF ITS

THE FPT UNIVERSITY HO CHI MINH CITY

December 2023

ACKNOWLEDGEMENTS

We extend our sincere gratitude to our teacher - our project supervisor, **PhD. Nguyen. Le Phu**, for his unwavering support throughout the ”Motion Capture Optimization” project. His guidance, valuable insightsand solutions were instrumental in shaping the direction of our work, from the initial proposal to the final report.

Our heartfelt appreciation goes to **HiPNUC Company**, the esteemed manufacturer of accelerometers. Their generous sponsorship of 2 sets of 6 wired sensors and 6 wireless sensors significantly contributed to the success of our project. The support from HiPNUC played a crucial role in the implementation and optimization of motion capture technology.

Special thanks are due to **Thanh Tin**, an invaluable contributor from **Comosix Media**. Thanh Tin shared crucial insights into the market's demands within the realms of animation and gaming. His perspectives on the Motion Capture application's relevance for creating animations in cartoons were indispensable, providing us with a real-world understanding of industry needs.

We are grateful to everyone who played a role, directly or indirectly, in the realization of this project. Your support and collaboration have been instrumentaland we are truly appreciative of the opportunity to work with such dedicated individuals.

We would also like to express our deep gratitude to **FPT University** for providing a nurturing academic environment that encourages innovation and hands-on learning. Our gratitude extends to all the teachers at FPT University who have shared their knowledge and expertise, fostering our intellectual growth.

Thank you so much!

AUTHOR CONTRIBUTIONS

Methodology direction, A.P.N and N.N.T; Dataset preparation, A.P.N; Dataset preprocessing, A.P.N; Algorithm collection, A.P.N and N.N.T; Framework establishment, N.N.T; Experiments, A.P.N and N.N.T; Results analysis, N.N.T; Report writing, A.P.N and N.N.T; Support materials, figures and appendix, A.P.N and N.N.T; Report revision, A.P.N and N.N.T. All authors have read and agreed to the Final Capstone Project document.

ABSTRACT

Motion Capture (MC) technology, prevalent in film production and virtual reality gaming, captures and reproduces human skeletal movements. This study seeks to operationalize Motion Capture in practical settings by employing six HiPNUC accelerometer sensors to forecast the motion of 18 SMPL skeleton joints. Our primary task involves harnessing the potential of HiPNUC's sensors and optimizing the computational engine inspired by Stanford University's Yifeng Jiang and their work on "Transformer Inertial Poser." Our key accomplishment lies in the successful integration of HiPNUC's accelerometer sensors. We have implemented these sensors, established an Engine ONNX Runtime and streamlined processing times through multiprocessing. Achieving real-time performance, our system operates at 60 fps on GPUs like RTX2060 and an impressive 90fps in offline mode on GPUs such as T4 and M40. This level of efficiency is essential for seamless integration into applications requiring high-performance motion capture. The adaptation of the Transformer Inertial Poser's computational engine to our context ensures precise predictions of SMPL joint movements. This adaptation maximizes the strengths of HiPNUC's sensor capabilities, providing a sturdy framework for real-time motion capture. Additionally, we have developed a data export mechanism through a dedicated socket, facilitating the seamless transmission of motion data to external applications, including Unity, Blender, and others. Our system's versatility ensures compatibility with a variety of creative and development environments, making it a valuable tool for animators, game developers, and virtual reality content creators. In conclusion, our research successfully bridges theoretical advancements and practical implementation in Motion Capture. The integration of six HiPNUC accelerometer sensors, coupled with an optimized computational engine, delivers high-fidelity real-time motion capture capabilities. The achieved frame rates on various GPUs highlight the scalability and efficiency of our system. With the added feature of data export to external applications, our system not only meets the demands of real-time performance but also opens avenues for diverse creative applications, reshaping the landscape of motion capture technology.

Keywords: Motion Capture (MoCap), Inertial Measurement Unit (IMU), Human Pose Estimation, Transformers, ONNX Runtime, Computing Parallelism.

Contents

ACKNOWLEDGMENTS **3**

AUTHOR CONTRIBUTIONS **4**

ABSTRACT **5**

1 INTRODUCTION **11**

1.1 Literature review 11

1.2 The necessity of the research 12

1.3 The feasibility of research 12

1.4 Research scope 13

2 RELATED WORK **14**

2.1 Capturing motion with computer vision 14

2.2 Capturing motion with digital signals 15

3 PROJECT MANAGEMENT PLAN **17**

4 MATERIALS AND METHODS **18**

4.1 Building System Architecture 19

4.1.1 System Workflow 19

4.1.2 Model Architecture 20

4.1.3 PyBullet Library 22

4.2 Process Data from IMU Sensors 23

4.2.1 Read and Merge Raw Data from Sensors 23

4.2.2 Signal Denoising Using Kalman Filter 25

4.2.3 Convert Data from Geographic Coordinate System to Virtual Coordinate System 26

4.3 Deploy Engine with ONNX Runtime 30

4.3.1 Converting Model from PyTorch to ONNX 30

4.3.2 Using ONNX Runtime for Inference 31

4.3.3 Optimizing the ONNX Model 32

4.4 Optimize with Multi-Processor 34

4.5 Scale Up with Multi-Character 35

5 RESULTS **36**

5.1 Evaluation Datasets 36

5.2 Quality of Quantization and Input Length 37
5.3 Performance on Types of Hardware 39
5.4 Hardware Responsiveness Analysis 42

6 DISCUSSIONS **43**
6.1 Application of Motion Capture 43
6.2 Future of Motion Capture 44

7 CONCLUSIONS **45**

8 APPENDIX **49**
8.1 Rotation Matrices and Rotate Vector 49
8.1.1 Rotation Matrix 49
8.1.2 Rotate Vector 50
8.1.3 Quaternion 50
8.2 A Skinned Multi-Person Linear Model (SMPL) 51
8.2.1 Key Features of SMPL 51
8.2.2 Structure of SMPL Data 52

List of Figures

1.1 Setup for Computer Vision (left) and setup with IMU sensors (right). 11

1.2 Idea of processing flow. 13

4.1 Illustration from input to expected output and connect with other applications. . 18

4.2 Hi299 inertial sensor series on hand and dock. 18

4.3 System workflow from sensor data to SMPL outputs. 19

4.4 Data shape input and output of model. 20

4.5 Model architecture with Transformer blocks and RNN layer. 21

4.6 Drift in motion capture 22

4.7 A 3D articulated model in PyBullet simulation for Mocap. 22

4.8 Sample signal values of sensor 0 following by time series. Top is value of accelerations and bottom is value of quaternion. 24

4.9 Denoising signal of Acceleration of axis Y from root sensor using Kalman filter. 26

4.10 Setup of T-pose in GCS and VCS. At left is the character at T-pose in simulation, center is human at T-pose and right is set 6 IMU sensors when defining front direction. 27

4.11 Preprocess data flow from raw data to input data for model prediction. 29

4.12 Shape of data from raw data to input data for model prediction. 29

4.13 Normal system workflow. 34

4.14 System workflow using multi-process. 34

4.15 Multiples simulations. 35

5.1 AMASS dataset. 36

5.2 Compare model results (yellow character) with ground-truth (green character) . 37

5.3 Processing with 2 characters 42

8.1 Present of quaternion vector 51

8.2 Relationship between SMPL and 18 joints. Left is SMPL and right is the skeleton system with all 18 joints defined. 52

List of Tables

3.1 Project Timeline 17

4.1 ONNX Runtime Graph Optimization Levels 33

5.1 Comparison of model quality on evaluation datasets. 38

5.2 Hardware information used for benchmark. 39

5.3 Comparison of model performance, benchmark on Tesla T4 40

5.4 Comparison of model performance, benchmark on RTX 2060 Super 40

5.5 Comparison of model performance, benchmark on Tesla M40 40

5.6 Comparison of model performance, benchmark on GTX 1060 40

5.7 Model performance when increasing the number of characters. 42

List of Abbreviations

AMASS	Archive of Motion Capture as Surface Shapes
AR	Augmented Reality
CGI	Computer-Generated Imagery
CPU	Central Processing Unit
DNN	Deep Neural Network
DOF	Degrees of Freedom
DSP	Digital Signal Processing
FPS	Frame Per Second
GCS	Geographic Coordinate System
GPU	Graphics Processing Unit
GTX	Giga Texel Shader eXtreme
IMU	Inertial Measurement Unit
IPS	Indoor Positioning Systems
LSTM	Long Short-Term Memory
MoCap	Motion Capture
ONNX	Open Neural Network Exchange
RNN	Recurrent Neural Network
RTX	Ray Tracing Texel eXtreme
SMPL	Skinned Multi-Person Linear
SOTA	State of The Art
TIP	Transformer Inertial Poser
USB	Universal Serial Bus
UWB	Ultra-Wideband
VCS	Virtual Coordinate System
VR	Virtual Reality

1. INTRODUCTION

1.1 Literature review

Currently, tracking to reconstruct the shape and motion of the human body is becoming increasingly important. Before 2014, the reconstruction of human shape and motion [1] mainly relied on flex sensors and then the joints were reconstructed through calculations based on the sensor's variation values. Since 2014, with the development of Deep Learning, human pose estimation [2] through Computer Vision has become popular, as illustrated in Figure 1.1. From 2017, researchers began using inertial sensors to predict body movements. Last June, researchers in China [3] used 6 IMUs combined with an LSTM network [4] to predict the SMPL skeleton of the human body.

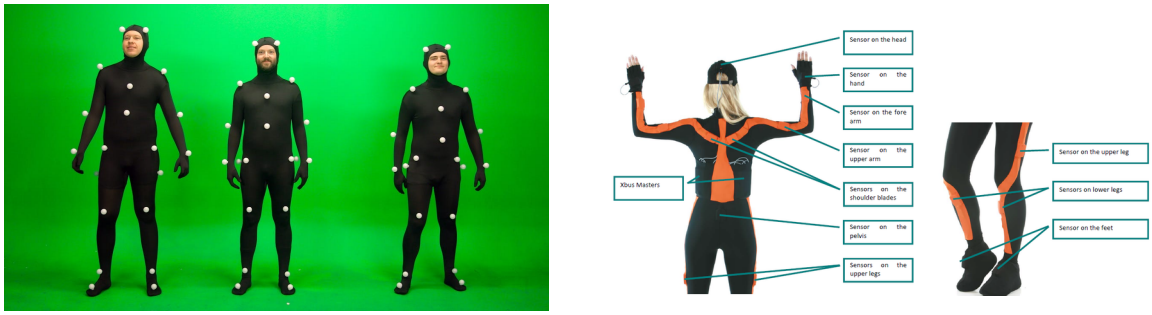

Fig. 1.1 Setup for Computer Vision (left) and setup with IMU sensors (right).

For prediction using Flex sensors, the accuracy is high but the cost is quite expensive to invest in and difficult for users to operate. Research on Computer Vision provides high stability but is limited by the camera's view and heavily affected by external factors such as lighting or background. Studies on using inertial sensors for predicting human body shape show relatively high accuracy. Moreover, the use of inertial sensors has advantages in terms of operating space and overcoming the barrier of the camera's viewing angle.

The use of human pose through Computer Vision is mainly for analyzing user behavior such as user actions analysis and providing notifications, evaluating user workouts and giving feedback. When using Computer Vision, motion can be recognized as commands to control devices. Computer Vision also helps to transform motions into movements of 3D characters.

Research using sensors aimed at virtual environments and large moving spaces will not be limited by space and angles. They are often used in virtual reality games, as well as interactions in augmented reality. In addition, this field is moving towards creating motion records of characters for filmmakers.

As previous studies have demonstrated the feasibility of using IMUs to predict human shape, the accuracy has not yet been high. Research is focusing on improving accuracy through deep learning models with powerful new architectures such as transformers.

1.2 The necessity of the research

The research aims to increase the accuracy and stability of the model with more advanced architectures. In addition, optimizing the speed and performance to run in real-time is also crucial. The data from IMU sensors fed into the model for prediction has a time series format, which is often lost in traditional recurrent neural networks [5]. Currently, transformer architecture with self-attention [6] can help the model retain more information from the past, improving the model's learning and inference capabilities. However, the increasing complexity of the model and the need for more computing power may increase the cost. Therefore, optimizing the engine to reduce latency in real-time and operating costs is essential. This can allow the model to run on machines with moderate configurations.

1.3 The feasibility of research

We have built a model based on the transformer architecture with 4 encoder blocks. The input of the model is data from 6 IMU sensors placed at the pelvis, left arm, right arm, left leg, right leg and head. Each sensor has 12 values, with 3 values for acceleration and 9 values for rotation matrix. The output of the model is the estimated velocity of 18 joints and the corresponding bone shape. The output values will be combined with data from the sensors to serve as input for the model, allowing the model to have more context for prediction.

The data for training and evaluating the model is exported from the AMASS dataset [7]. The SMPL skeleton [8] of the data will be the label and the input data will be the acceleration and Euler angles extracted from the corresponding sensor positions on the body. The final task is

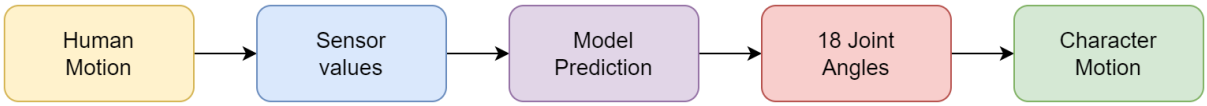

Fig. 1.2 Idea of processing flow.

optimizing the model from the PyTorch framework. Open Neural Network Exchange (ONNX) [9] is a powerful and optimized framework for NVIDIA GPUs and here we use RTX2060 Super. ONNX helps reduce the latency of Encoder Only architectures by 2x with FP32 and 2.5x with FP16 on Turing architecture GPUs [10]. We aim to optimize the engine and quantize the parameters to achieve real-time performance with 90-120 fps.

1.4 Research scope

This research focuses on applying AI to build a system that simplifies the process of creating motions for virtual characters in the film industry. The system utilizes inertial sensors to record human motions, which are then analyzed and transformed into realistic movements for virtual characters. The goal is to optimize the system to run in real-time on a personal computer, allowing for greater flexibility and ease of use. By automating the motion capture process with AI technology, filmmakers can save time and resources while still producing high-quality content.

This research aims to apply many sensor lines to Motion Capture and specifically the Hi299 sensor line first. We build synthesis, transformation, and preprocessing methods for sensors. We optimize the calculation process to help MoCap run faster on low-GPU hardware and can run realtime on mid-range hardware. Besides, we also develop output to expand the network and connect to many applications.

2. RELATED WORK

2.1 Capturing motion with computer vision

Vision-based motion analysis involves extracting information from sequential images in order to describe movement [11]. This field dates back to the late 19th century, initiated by the groundbreaking efforts of Eadweard Muybridge [12], who was among the first to devise methods for capturing sequences of images. Since then, the field of motion analysis has undergone significant evolution, driven by major technological breakthroughs and a growing need for quicker and more complex methods of capturing movement. These techniques are now applied in various domains, from assessing human gait in clinical settings [13] to creating animations in video games [14].

The evolution of motion capture in computer vision has seen significant advancements but also faces certain challenges [11]. This field has progressed from manual methods to sophisticated systems incorporating computer vision and machine learning. Markerless motion capture systems use advanced computer vision algorithms to estimate body pose directly from image data [15, 16]. A number of reviews [17, 18] have been previously published detailing these developments, targeting specific application areas such as security, forensics and entertainment.These systems are complex, requiring sophisticated camera systems, body models, and algorithms for pose estimation. While promising, they are still in the developmental stage and have not yet achieved widespread use in biomechanics with only a small number of companies providing commercial systems. However, it remains unclear exactly what precision these systems can achieve in comparison to the other, more established motion analysis systems available on the market. Certainly, the technology is under rapid development with modern computer vision algorithms improving the robustness, flexibility and accuracy of markerless systems.

2.2 Capturing motion with digital signals

In our capstone, we primarily focuses on optimize Inertial Measurement Unit (IMU) sensors either as the primary or a supplementary input source. Additionally, we explore the application of Transformer models, specifically the one introduced by Vaswani et al. [6] with version Transformer Inertial Poser, as they form the foundational basis of our reconstruction approach.

A typical IMU sensor includes an accelerometer measuring 3-axis linear acceleration, a gyroscope measuring 3-axis angular velocity, and a magnetometer identifying the vector pointing Earth's magnetic north. From these raw signals, sensor fusion algorithms based on Kalman filter or its extended version are used to provide more robust measures of the orientation [19, 20, 21, 22] [Bachmann et al. 2001; Del Rosario et al. 2018; Foxlin 1996; Vitali et al. 2021]. IMU sensors have been used along with vision-based sensors such as RGB or RGB-D cameras for motion estimation.

With IMU sensors getting more compact and inexpensive, they have received increasing attentions from both industry and research communities for a standalone body tracking solution. Popular commercial products such as Xsens [23] and Rokoko [24] can generate high-quality human motions ready to be used in real-time game engines. However, requiring a sophisticated full-body setup with at least 17 IMUs hinders their accessibility to everyday users. Researchers have therefore proposed body tracking systems with a small number of IMUs sparsely placed on the body, usually utilizing statistical body models and/or high quality optical mocap data as prior to mitigate input signals being under-specified. Marcard et al. [25] developed an offline system (SIP) with only six IMUs, which optimizes poses and the parameters of the SMPL body model [26] to fit the sparse sensor input. Huang et al.[27] learned a deep neural-net model (DIP) from a large amount of motion capture data to directly map the IMU signals to poses. Their model is based on bidirectional recurrent neural networks (BRNN), so the system can run in an online manner while considering both the past and future sensor inputs with a negligible latency, outperforming previous non-learning online methods. An ensemble of BRNNs was further adopted by Nagaraj et al. [28] to improve upon the results. However, the two real-time solutions mostly focus on reconstructing the local joint motion without global translation. Yi et al.[29] proposed a new neural model (TransPose) where the progressive upscaling of joint position estimation showed more accurate pose estimation. The model can additionally generate

accurate global root motions by combining a supporting-foot heuristics and a small learned deep network, similar to [30]. Recently, an extension of this system (PIP)[31] has been introduced, which is concurrent to our paper, where the predicted motions are further optimized to reduce violations of physics laws [32]. We explore this problem domain with a set of drastically different techniques, and with much more relaxed assumptions on the environment geometry, while producing comparable or better reconstruction results.

3. PROJECT MANAGEMENT PLAN

Table 3.1 Project Timeline

Week	Date	Project Phase	Detail Task	Note
1	09/04/2023	Initiation	Review the paper and the current model Set up development environment	Completed
2	09/11/2023	Analysis	Plan in detail for optimization requirements Coordinate transformation	Completed
3	09/18/2023	Design	Design improvements for the model Solutions for coordinate transformation	Completed
4	09/25/2023	Development	Coding improvements with ONNX	Completed
5	10/02/2023	Development	Model development and refinement	Completed
6	10/09/2023	Development	Finalize coordinate system transformation	Completed
7-8	10/16/2023	Testing	Testing of the model Bug fixing and performance optimization	Completed
9-10	10/30/2023	Evaluation	Evaluate model performance real-time	Completed
11-13	11/13/2023	Report	Prepare report and usage guidelines	Completed
14	04/12/2023	Revision	Overall project evaluation	Completed

4. MATERIALS AND METHODS

In this project, we employ six inertial measurement unit (IMU) sensors strategically positioned on a model character at six key locations: 0 (smallest ID) - pelvis, 1 - left wrist, 2 - right wrist, 3 - left knee, 4 - right knee, 5 (largest ID) - head. The computer receives data from these six sensors and utilizes a deep learning model in conjunction with a physics simulation tool to predict 18 human joints. Subsequently, the output is stored in memory or transmitted in real-time to graphic software applications such as Unity, Unreal, Blender.

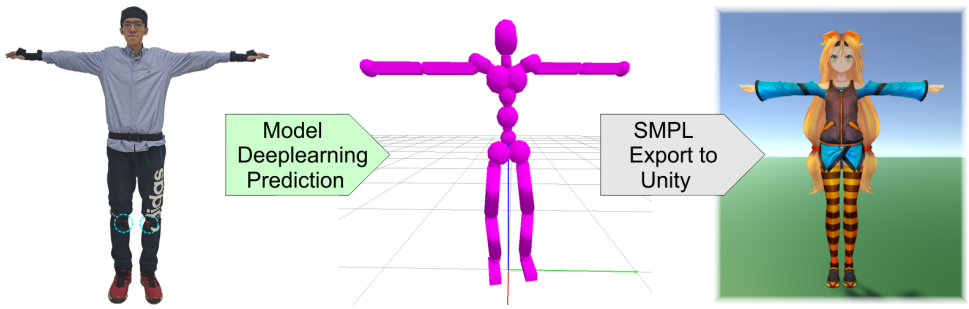

Fig. 4.1 Illustration from input to expected output and connect with other applications.

The project utilizes the Hi299 inertial sensor series with wireless connectivity support from HiPNUC, featuring six-axis inertial measurement and three-axis magnetic field measurement. This sensor configuration enables accurate and dynamic tracking of the character's movements, facilitating the precise capture of intricate joint motions. The wireless capability enhances flexibility, allowing for unencumbered movement during data capture sessions.

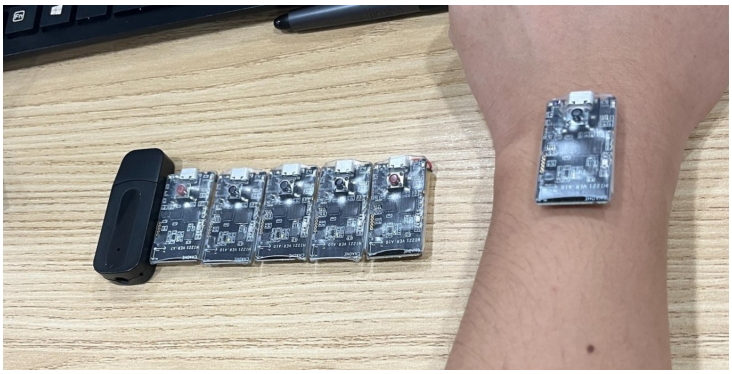

Fig. 4.2 Hi299 inertial sensor series on hand and dock.

4.1 Building System Architecture

4.1.1 System Workflow

Flowchart 4.3 describing our Motion Capture System. It outlines the process flow from sensor data collection to the final update of joint points, involving various data processing stages, prediction models and calibrate with tools like PyBullet.

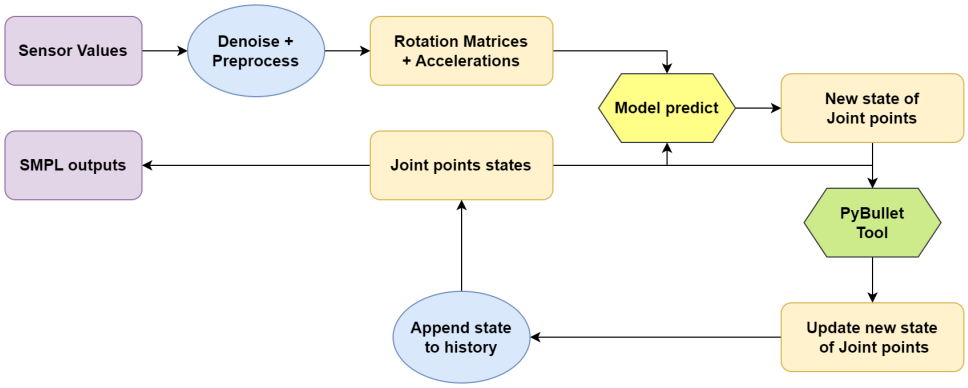

Fig. 4.3 System workflow from sensor data to SMPL outputs.

The system starts with sensor values that presumably capture motion data from a real-world environment. These sensor values are then denoised to improve the quality of the data. The cleaned data is then converted into rotation matrices and accelerations data, which are crucial for understanding the orientation and movement dynamics of the captured points. A predictive model uses the processed data to forecast the new state of joint points. This model is trained to interpret sensor data and predict movement. The predicted new state of joint points is the output from the predictive model, detailing the positions of joints in the next frame or time step. The new state of joint points is then processed by PyBullet, which is a physics engine used for simulations in robotics and animation. After being processed by PyBullet, the state of the joint points is updated, which could mean it is refined to be more realistic. The updated state of the joint points is appended to the history, a log of all the states for the motion capture system to learn from the sequence. Finally, the output is likely in the form of SMPL (Skinned Multi-Person Linear model) parameters, which is a widely used model for representing human bodies in motion in real-time.

4.1.2 Model Architecture

In this project, we reused a model from TIP [33] with architecture containing transformer blocks and RNN layer. Transformers are a perfect combination for parallel computation between states thanks to matrix multiplication by self-attention instead of the sequential model of RNNs. This significantly increases computation speed when used on GPUs.

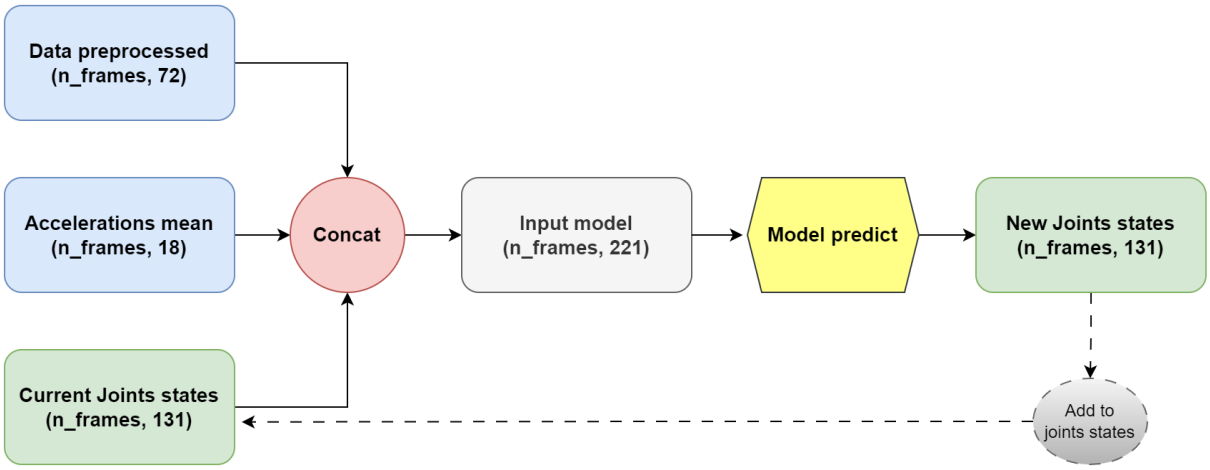

Fig. 4.4 Data shape input and output of model.

The proposed model's input consists of sensory data and character state information. The sensory data encompasses six rotation matrices and six acceleration vectors, forming a 72-dimensional vector. Additionally, the mean of acceleration values from the five most recent states is incorporated, adding 18 dimensions to the input vector.

Character joint states are represented using 131 dimensions. The first three dimensions encode the root position of the character, while the remaining 128 dimensions capture the joint angles of 18 joints. Each joint is represented by 6 values, with the first two columns corresponding of rotation matrix to the fixed point of the joint.

The model's output is a prediction of the character's new state, adhering to the same format as the character state information in the input. To ensure accuracy and compatibility with the character state representation, the output is calibrated using PyBullet and then appended to the character joint states for use in subsequent predictions.

The input of the model is a list of frames with shape=(batch_size, n_frames, 221) and the output of model also is a list of frames with shape=(batch_size, n_frames, 131). Model has 2 linear layers after input and before output, it is used to perform linear and nonlinear transformations on the input and output of the model. Inside of the model we have 4 transformer blocks with 16 heads of multihead attention, attention helps find correlations between states and helps transform embedding vectors for states more effectively. After transformer blocks and before the last linear layer is RNN layer, this layer helps model predict output of state n using information from n-1 state before that.

Figure 4.5 below illustrates the detailed architecture of our model. The total of parameters used in this model is about 3.5M parameters.

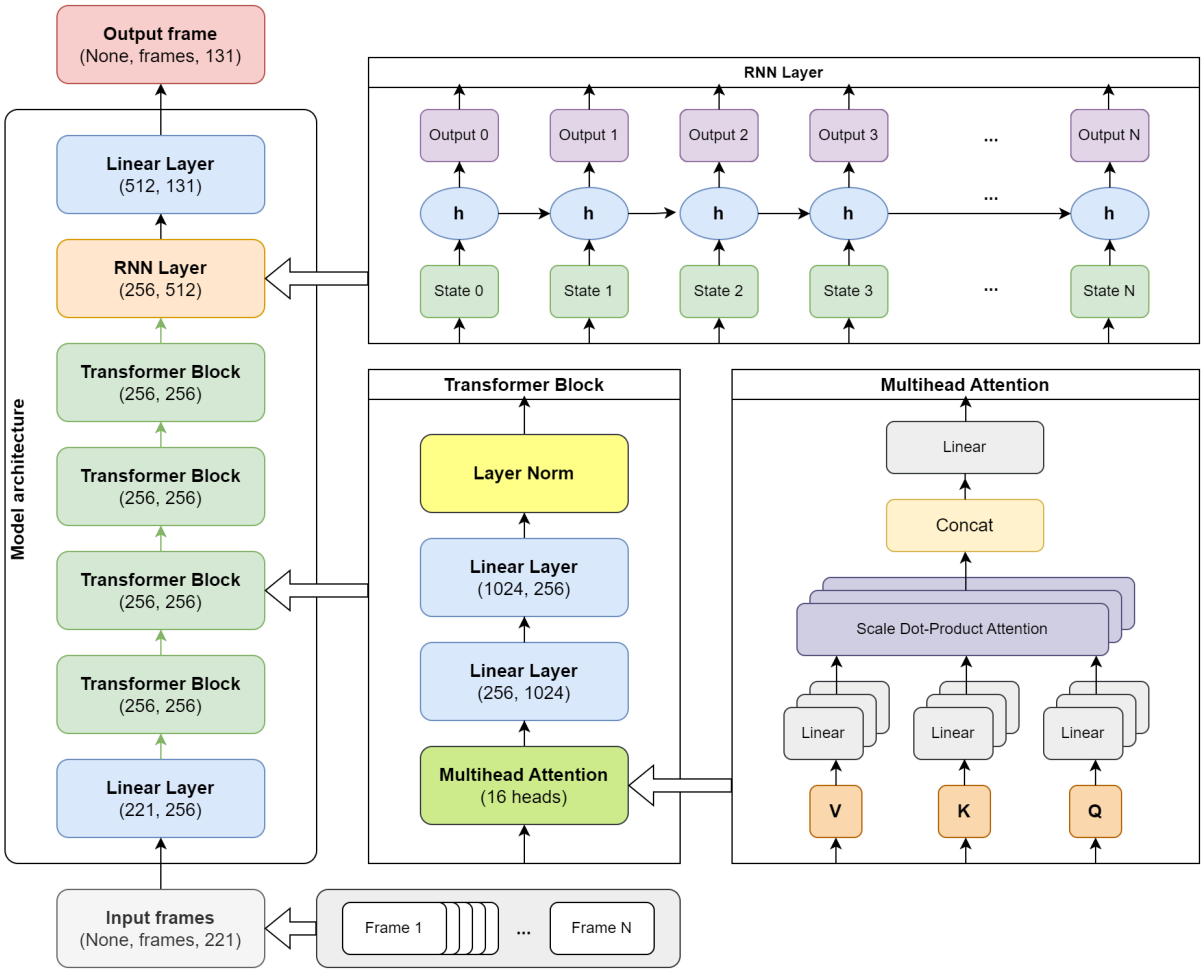

Fig. 4.5 Model architecture with Transformer blocks and RNN layer.

4.1.3 PyBullet Library

Motion capture are sensitive to drift, significantly compromise the quality and accuracy of the captured movements. Drift in motion capture refers to the gradual accumulation of errors over time, resulting in a misalignment of sensor position and orientation data compared to actual movements. This can be caused by factors such as measurement inaccuracies from the hardware, sensor instability, or environmental changes. Drift leads to inaccuracies in capturing movements, making the final simulated output potentially unnatural or inconsistent.

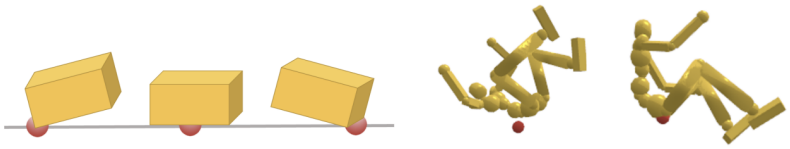

Fig. 4.6 Drift in motion capture

PyBullet can help mitigate this issue by using algorithms to calibrate and reconcile mocap data with physical models. PyBullet is capable of simulating complex physical interactions and apply physical constraints to keep movements within realistic. When mocap data is fed into PyBullet, it can utilize the physical model to predict and correct movements, minimizing the impact of drift by ensuring that motion adheres to the laws of physics. This improves the accuracy of motion reconstruction, even when the original mocap data is imperfect.

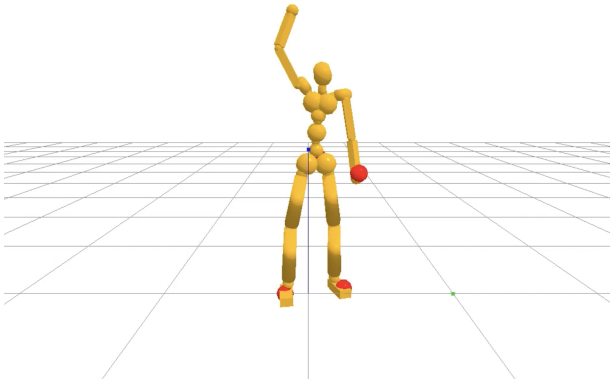

Fig. 4.7 A 3D articulated model in PyBullet simulation for Mocap.

4.2 Process Data from IMU Sensors

4.2.1 Read and Merge Raw Data from Sensors

The current project leverages the capabilities of six Inertial Measurement Unit (IMU) sensors developed by HiPNUC to gather user action data. The data acquisition process involves the utilization of these IMU sensors, each of which transmits information to a USB docking station connected to our computer. The transmitted data is structured in the form of a packet comprising six JSON objects. Each JSON object encapsulates key parameters, including GWD, id, timestamp, acc (acceleration), gyr (gyroscope readings), mag (magnetic field data), euler (Euler angles), and quat (quaternion values).

The data received on our computer is organized as a set of six JSON packets, each encapsulating the sensor readings. The structure of each JSON packet is defined as follows:

```
{
    'GWD': [{'': 0}],
    'id': [{'': 0}],
    'timestamp': [{'(s)': 0.0}],
    'acc': [{'X': 0.0, 'Y': 0.024, 'Z': 1.012}],
    'gyr': [{'X': 0.0, 'Y': 0.2, 'Z': 0.0}],
    'mag': [{'X': -38, 'Y': 13, 'Z': -1}],
    'euler': [{'Roll': 1.27, 'Pitch': 0.11, 'Yaw': -64.98}],
    'quat': [{'W': 0.843, 'X': 0.01, 'Y': -0.005, 'Z': -0.537}]
}
```

For the specific objectives of this project, our analysis concentrates on three key parameters within each JSON packet: 'id,' 'acc,' and 'quat.' The 'id' serves as a unique identifier for individual sensors, while 'acc' represents the acceleration readings, and 'quat' encapsulates quaternion values, crucial for capturing motion and orientation dynamics. To streamline the analysis process, the data from all six IMU sensors is integrated into a unified data frame. The consolidated format of the data frame is as follows:

```
{
    'acc': [acc_0, acc_1, acc_2, acc_3, acc_4, acc_5],
    'quat': [quat_0, quat_1, quat_2, quat_3, quat_4, quat_5]
}
```

For each sensor, the acceleration values are represented as an array with three real numbers, corresponding to the acceleration along the X, Y, and Z axes. The quaternion values are also represented as an array with four real numbers, denoting the rotation angles of the sensor. These four values are typically labeled as X, Y, Z, and W. The following charts illustrate the values captured by sensor 0 (root) .

For each sensor, the acceleration values are represented as an array with three real numbers, corresponding to the acceleration along the X, Y, and Z axes. The quaternion values are also represented as an array with four real numbers, denoting the rotation angles of the sensor. These four values are typically labeled as X, Y, Z, and W. The following charts illustrate the values captured by sensor 0 (root) .

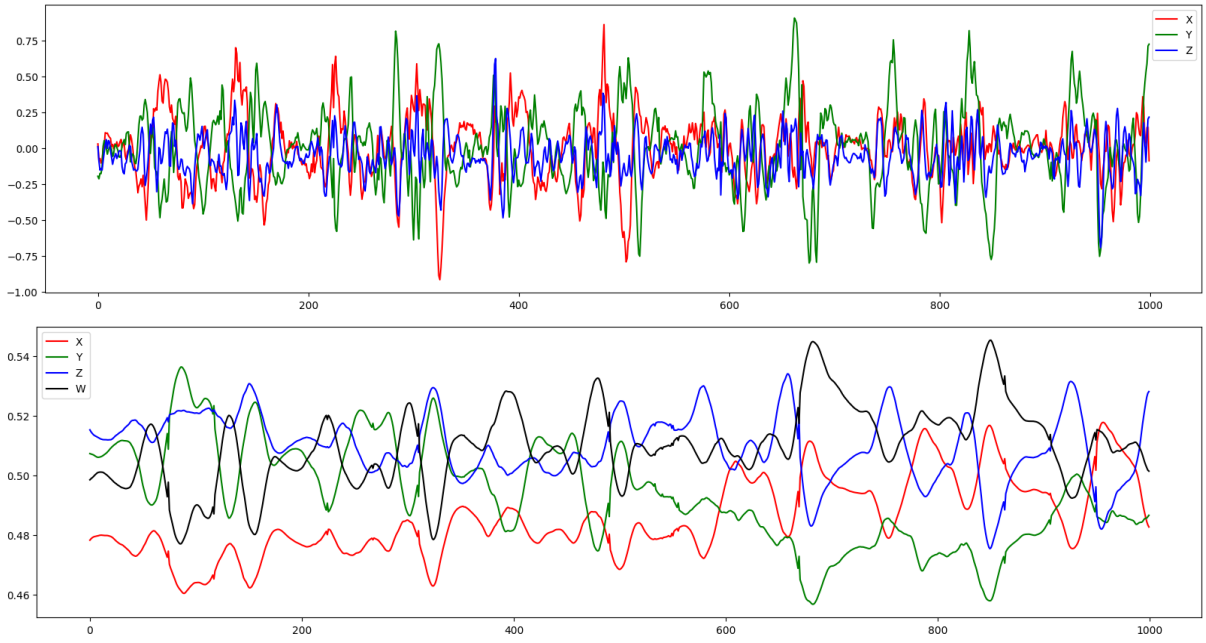

Fig. 4.8 Sample signal values of sensor 0 following by time series.
Top is value of accelerations and bottom is value of quaternion.

4.2.2 Signal Denoising Using Kalman Filter

In this project, we employ the Kalman Filter as a robust technique for signal denoising, particularly in the context of data collected from six IMU sensors. IMU sensors are instrumental in capturing various aspects of motion and orientation, but their measurements are often susceptible to noise. The Kalman Filter is well-suited for this task, as it is capable of estimating the true underlying state of a system by recursively processing noisy measurements. The filter combines a prediction model, based on the system dynamics, with real-time measurements to generate an optimal estimate.

Mathematically, the Kalman Filter is represented by the following equations:

Predict	**Update**
$\hat{x}_{k\|k-1} = A_k\hat{x}_{k-1\|k-1} + B_k u_k$	$K_k = P_{k\|k-1}H_k^T(H_k P_{k\|k-1}H_k^T + R_k)^{-1}$
$P_{k\|k-1} = A_k P_{k-1\|k-1}A_k^T + Q_k$	$\hat{x}_{k\|k} = \hat{x}_{k\|k-1} + K_k(z_k - H_k\hat{x}_{k\|k-1})$
	$P_{k\|k} = (I - K_k H_k)P_{k\|k-1}$

Where:

- $\hat{x}_{k|k}$ is the updated state estimate at time k,
- $P_{k|k}$ is the updated error covariance at time k,
- A_k is the state transition matrix,
- B_k is the control-input matrix,
- u_k is the control input,
- Q_k is the process noise covariance,
- H_k is the measurement matrix,
- R_k is the measurement noise covariance,
- K_k is the Kalman gain,
- z_k is the measurement at time k.

This application of the Kalman Filter aims to enhance the accuracy and reliability of signal measurements obtained from the six IMU sensors by effectively mitigating the impact of inherent noise in the sensor data. Figure 4.9 below present acceleration signal of axis Y from root sensor with blue is raw value and orange is value after using Kalman filter.

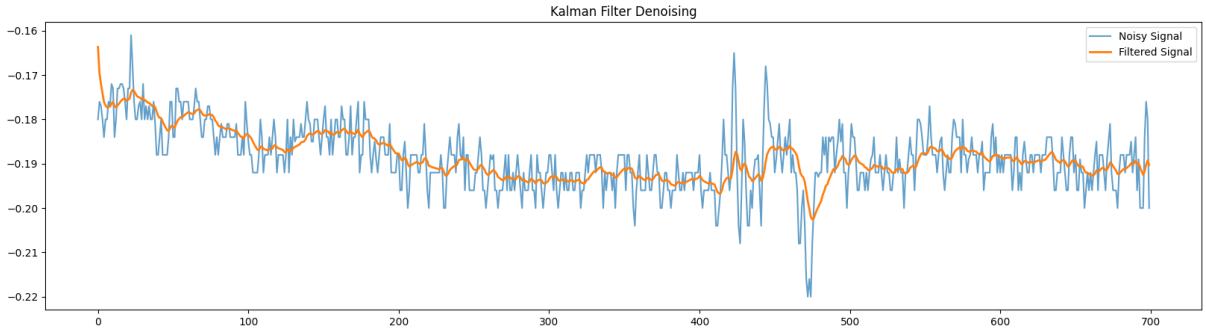

Fig. 4.9 Denoising signal of Acceleration of axis Y from root sensor using Kalman filter.

4.2.3 Convert Data from Geographic Coordinate System to Virtual Coordinate System

This section will explain why we must convert data from sensor values in geographic coordinate system (GCS) to virtual coordinate system (VCS) used in simulation and how we convert them.

A Geographic Coordinate System is a three-dimensional reference system that locates points on the Earth's surface. The Values of sensors are obtained from GCS with three fixed axes: X axis corresponds to North, Y axis corresponds to West and Z axis corresponds to the upwards. But in the simulation program, we fixed X axis to indicate the specified front of character, Y axis corresponds to the left and Z axis corresponds to the upwards. The simplest way, we start up at T-Pose with the face facing North and GCS will overlap with VCS. But it is difficult to determine the north direction in actual use. Therefore we need the first phase to determine the relative orientation of the character in GCS. In this phase we set the X-axis of the sensors to face the character's front direction on ground, Z-axis up to sky and value of the sensors to help define the angle between human direction and North direction.

Fig. 4.10 Setup of T-pose in GCS and VCS. At left is the character at T-pose in simulation, center is human at T-pose and right is set 6 IMU sensors when defining front direction.

We have some conventions:

- V: Virtual coordinate system used in simulation
- G: Geographic coordinate system is base for get IMU values
- I: IMU coordinate system
- R: Rotation matrix
- $0, T, t$: State at define front direction, T-pose, and at any time t

V: virtual **A**: arm **I**: IMU **G**: global **D**: rotation matrix

Matrix global to virtual:

$$D_V = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Rotation matrix of IMU at T Pose:

$$[\widehat{w_t}, \widehat{x_t}, \widehat{y_t}, \widehat{z_T}] \xrightarrow{quaternion2matrix} D_{T-I}$$

Rotation matrix of IMU at Pose 1:

$$[\widehat{w_1}, \widehat{x_1}, \widehat{y_1}, \widehat{z_1}] \xrightarrow{quaternion2matrix} D_{1-I}$$

Goal:

$$\overrightarrow{V} = D.\overrightarrow{A_1} \tag{4.1}$$

When T Pose we have:

$$\overrightarrow{V} = \overrightarrow{A_T} = D_{A-I}.\overrightarrow{I_T} \tag{4.2}$$

When Pose 1 we have:

$$\overrightarrow{A_1} = D_{A-I}.\overrightarrow{I_1} \tag{4.3}$$

In Virtual coordinate:

$$\overrightarrow{V} = D_V.\overrightarrow{G} \tag{4.4}$$

In Global coordinate:

$$\overrightarrow{G} = D_{T-I}.\overrightarrow{I_T} = D_{1-I}.\overrightarrow{I_1} \tag{4.5}$$

From (4.2) and (4.4) we have:

$$D_V.\overrightarrow{G} = \overrightarrow{V} = D_{A-I}.\overrightarrow{I_T} \tag{4.6}$$

From (4.5) and (4.6) we have:

$$D_V.D_{T-I} = D_{A-I} \tag{4.7}$$

From (4.3) and (4.7) we have:

$$\overrightarrow{A_1} = D_V.D_{T-I}.\overrightarrow{I_1} \tag{4.8}$$

From (4.1), (4.4), and (4.5) we have:

$$D.\overrightarrow{A_1} = D_V.D_{1-I}.\overrightarrow{I_1} \tag{4.9}$$

From (4.8) and (4.9) we have:

$$D.D_V.D_{T-I} = D_V.D_{1-I} \tag{4.10}$$

Therefore:

$$D = D_V.D_{1-I}.(D_V.D_{T-I})^{-1} \tag{4.11}$$

Formula 4.11 is used to calculate the rotation matrix and compute the acceleration for the model input.

In summary, process data from 6 sensors including 2 stages is calibrated and preprocessed. At the calibrated stage, we get a mean value of 3 second, 3 second for getting Acc and Quat at Root and 3 second for getting Acc and Quat at T-pose. After that, we find an offset of accelerations and 2 rotation matrices used to calculate rotation matrices and accelerations of 6 points for input of the model in simulation.

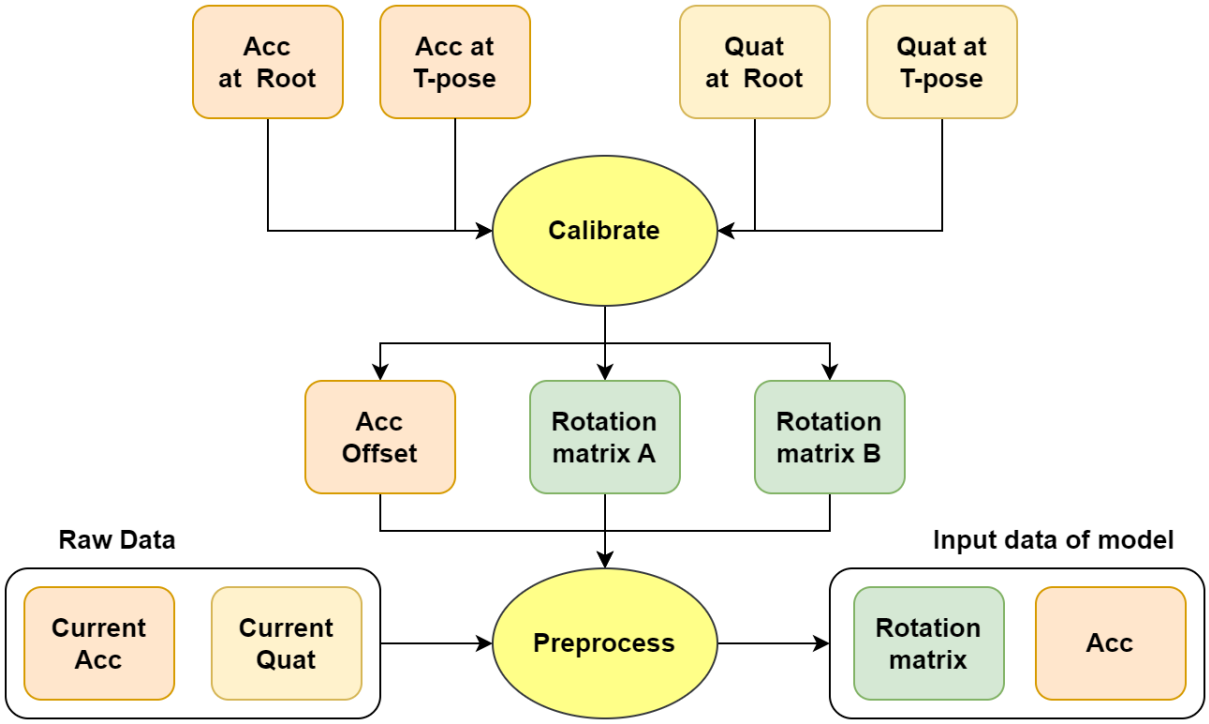

Fig. 4.11 Preprocess data flow from raw data to input data for model prediction.

Raw data is a pack of 6 JSON data including 3 dims of acceleration and 4 dims of quaternion. After preprocessing, we have a pack of 6 JSON data including 9 dims of rotation matrix after flatten and 3 dims of acceleration. We will concatenate them to a data frame with 72 dims, the first 54 dimensions are the concatenation of 6 rotation matrices in order, and the following 18 dimensions are the concatenation of 6 sets of acceleration values in order. Shape of data through processing data present in Figure 4.8.

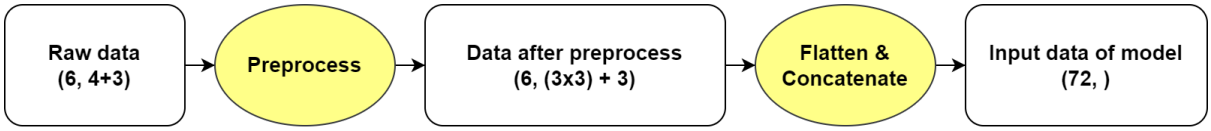

Fig. 4.12 Shape of data from raw data to input data for model prediction.

4.3 Deploy Engine with ONNX Runtime

ONNX, which stands for Open Neural Network Exchange, is an open format for machine learning models. It's designed to facilitate engineers and data scientists in transferring models between different machine learning platforms and optimize the deployment process. We provide an overview of how ONNX operates and its workflow in my system:

4.3.1 Converting Model from PyTorch to ONNX

Exporting PyTorch Model: Use the *torch.onnx.export* function to convert the PyTorch model to ONNX format.

```
torch.onnx.export(
    model=model,
    (dummy_input_imu, dummy_input_s),
    onnx_path=onnx_path,
    dynamic_axes={
        'imu_input': {1: 'seq_len'},
        's_input': {1: 'seq_len'}
    },
    input_names=['imu_input', 's_input'],
    output_names=['output'],
    opset_version=14,
    export_params=True,
    do_constant_folding=True
)
```

Some parameters of the export function:

- `model`: The PyTorch model you want to export.
- `(dummy_input_imu, dummy_input_s)`: A tuple containing multiple tensors representing the model's input.
- `onnx_path`: The path where the model will be saved after conversion.

- `dynamic_axes`: Specifies dynamic axes for input tensors.
- `input_names`: Names assigned to input parameters.
- `output_names`: Names assigned to the returned values.
- `opset_version`: Specifies the ONNX operator set version to use. In this case, it's version 14.
- `export_params`: Determines whether to use pretrained weights or not. Set to True to use pretrained weights.
- `do_constant_folding`: Enables constant folding optimization during export, which can simplify the exported ONNX model.

4.3.2 Using ONNX Runtime for Inference

We use ONNX Runtime, an optimized execution engine, to load the ONNX model. ONNX Runtime supports multiple platforms and is optimized for various hardware types.

```
onnxruntime.InferenceSession(
      onnx_path,
      providers=['CUDAExecutionProvider']
)
```

- `onnxruntime.InferenceSession`: Create an inference session.
- `onnx_path`: Path to the ONNX file.
- `providers`: The execution providers used to perform inference. In this case, we have specified 'CUDAExecutionProvider', indicating that we want to use the GPU to accelerate the inference process.

4.3.3 Optimizing the ONNX Model

Create Float16 Models

Converting a model to use float16 instead of float32 can decrease the model size and improve performance. There may be some accuracy loss, but in many models the new accuracy is acceptable.

```
convert_float_to_float16(
    model,
    min_positive_val=1e-7,
    max_finite_val=1e4,
    keep_io_types=False,
)
```

- `model`: The ONNX model to convert.
- `min_positive_val`, `max_finite_val`: Constant values will be clipped to these bounds. 0.0, nan, inf, and -inf will be unchanged.
- `keep_io_types`: Whether model inputs/outputs should be left as float32. Set to `False` to left as float16.

Graph Optimizations

ONNX Runtime provides various graph optimizations to improve performance. Graph optimizations are essentially graph-level transformations, ranging from small graph simplifications and node eliminations to more complex node fusions and layout optimizations.

Graph optimizations are divided into three levels:

- Basic Graph Optimizations: These are semantics-preserving graph rewrites which remove redundant nodes and redundant computation.
- Extended Graph Optimizations: These optimizations include complex node fusions.

- Layout Optimizations: These optimizations change the data layout for applicable nodes to achieve higher performance improvements.

ONNX Runtime defines the *GraphOptimizationLevel* enum to determine which of the aforementioned optimization levels will be enabled. Choosing a level enables the optimizations of that level, as well as the optimizations of all preceding levels.

Optimization Level	**Description**
`ORT_DISABLE_ALL`	Disables all optimizations
`ORT_ENABLE_BASIC`	Enables basic optimizations
`ORT_ENABLE_EXTENDED`	Enables basic and extended optimizations
`ORT_ENABLE_ALL`	Enables all available optimizations

Table 4.1 ONNX Runtime Graph Optimization Levels

Quantize ONNX Models (Future Features)

Quantization in ONNX Runtime refers to 8 bit linear quantization of an ONNX model. During quantization, the floating point values are mapped to an 8 bit quantization space of the form :val_fp32 $=$ scale $\times$ (val_quantized $-$ zero_point)

Where *scale* is a positive real number used to map the floating point numbers to a quantization space. It is calculated as follows:

$$\text{scale} = \frac{\text{data\_range\_max} - \text{data\_range\_min}}{\text{quantization\_range\_max} - \text{quantization\_range\_min}} \tag{4.12}$$

zero_point represents zero in the quantization space. It is important that the floating point zero value be exactly representable in quantization space. If it is not possible to represent 0 uniquely after quantization, it will result in accuracy errors.

There are two ways of quantizing a model: dynamic and static. Dynamic quantization is preferred to use dynamic quantization for transformer-based models. Dynamic quantization calculates the quantization parameters for activations dynamically. Python API for dynamic quantization is in the module `onnxruntime.quantization.quantize`, function `quantize_dynamic()`.

4.4 Optimize with Multi-Processor

In the simple system, as outlined in section 4.1.1, processing occurs sequentially, progressing from the beginning to the end of each frame and then onto subsequent time intervals. This approach results in underutilization of GPU computational time after prediction, as the system awaits the completion of other processing tasks, as illustrated in Figure 4.13 below.

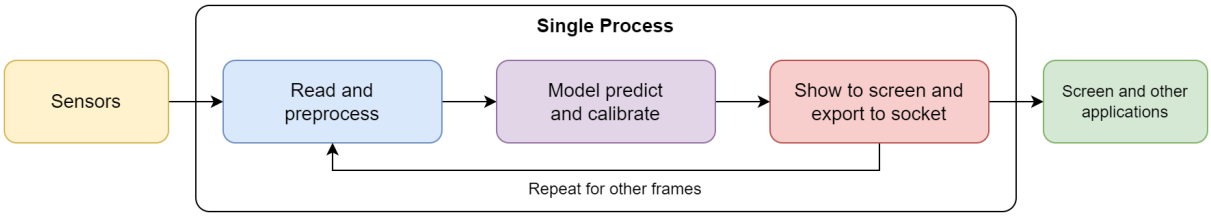

Fig. 4.13 Normal system workflow.

To maximize GPU computational capabilities, we parallelized computational processes into independent tasks, necessitating the allocation of additional processors. Follow Figure 4.14, this approach allows the ”Read and preprocess” phase to seamlessly read and compute upon data arrivals without waiting for the predict and export processes to conclude. Similarly, the prediction task runs continuously on the GPU, leveraging preprocessed data from the preceding phase, eliminating the need to stall for data processing. While this strategy significantly boosts computational speed, it entails a trade-off by requiring a higher number of processors. This optimization ensures efficient utilization of GPU resources, enhancing the overall system's computational throughput.

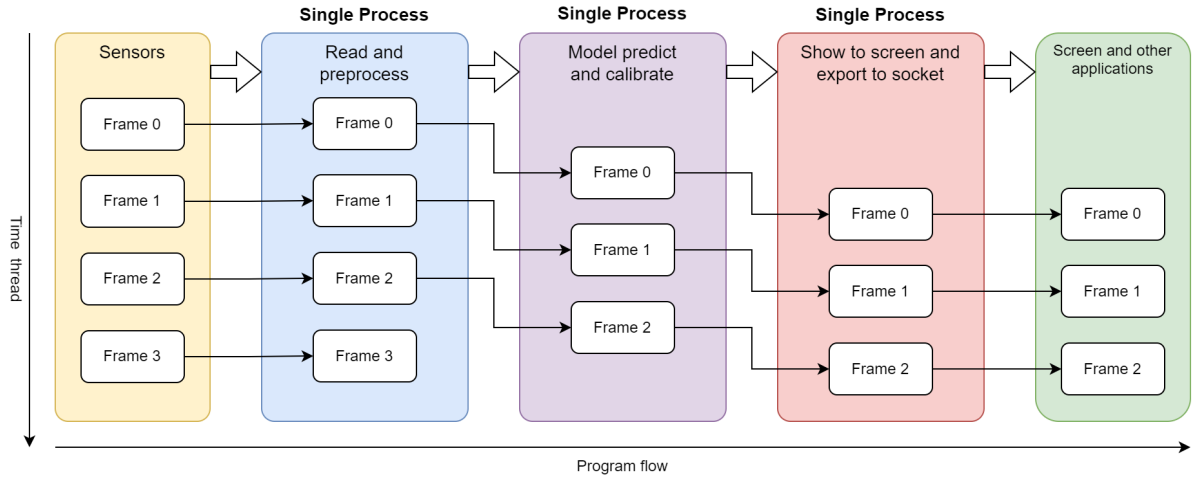

Fig. 4.14 System workflow using multi-process.

4.5 Scale Up with Multi-Character

Increasing the number of characters, synonymous with augmenting batch size, proves instrumental in reducing computation time for the system. This optimization strategy enhances the efficiency of character-based processes, mitigating computational demands and expediting overall system performance. The amplified batch size not only streamlines calculations but also contributes to the system's scalability, rendering it more adept at handling diverse computational workloads. This report delves into the impact of character quantity augmentation on batch size, illustrating its pivotal role in optimizing system efficiency and laying the groundwork for improved computational capabilities.

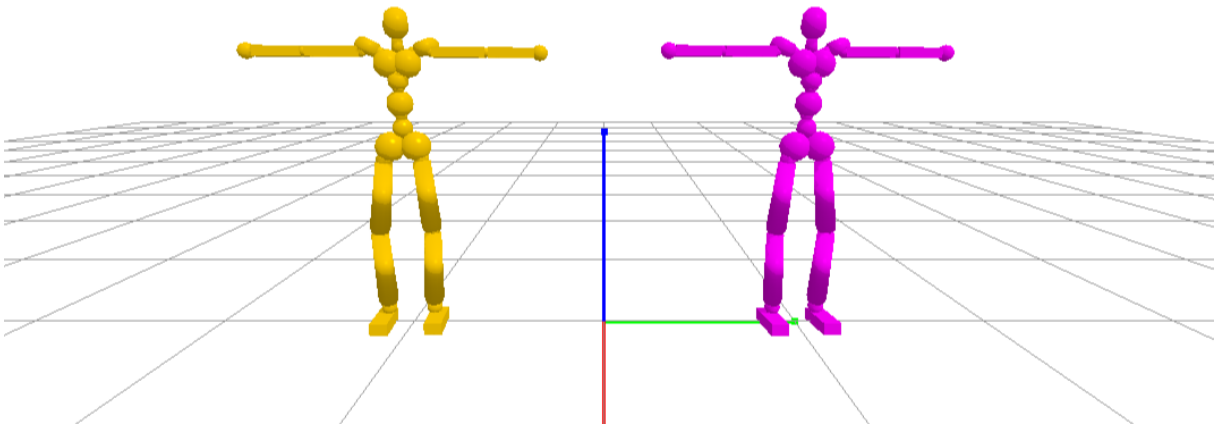

Fig. 4.15 Multiples simulations.

Simultaneously, this enhancement enables the system to process multiple characters and movements concurrently, significantly reducing recording time. The heightened capacity to handle diverse character interactions and motions concurrently not only optimizes efficiency but also proves instrumental in expediting the overall recording process.

5. RESULTS

5.1 Evaluation Datasets

Our model is trained on established AMASS dataset [7], which encompass a variety of motion types and are matched with verified full-body motion data for comparative analysis. AMASS encompasses over 40 hours of motion data, covering more than 300 subjects and over 11000 motions, making it richer than previous human motion collections.

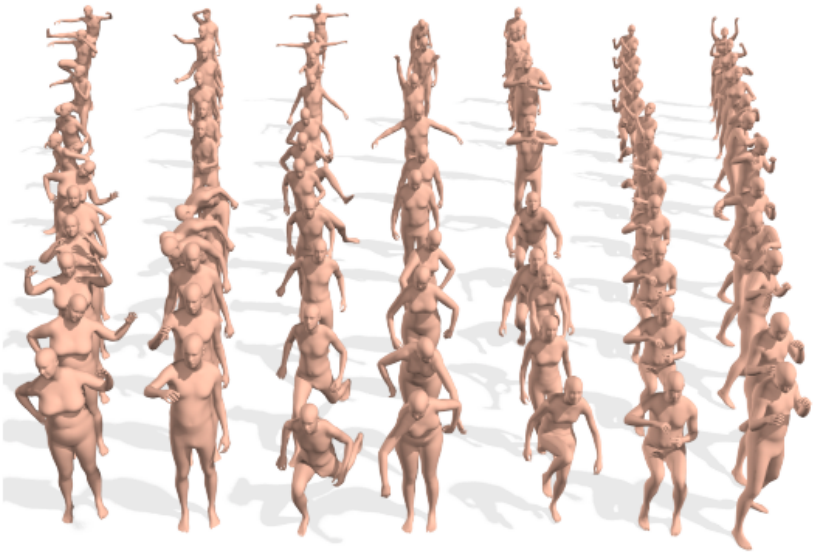

Fig. 5.1 AMASS dataset.

To evaluate the model, we evaluate using DanceDB, a large dataset of contemporary dances, therefore containing unique motion types to any other training dataset. DanceDB is part of AMASS but the model intentionally keeps it out from training data. Previous works likely did not include them in training either since they predate DanceDB's release in AMASS [33].

In our evaluation, we employ common metrics and focus on three models: PyTorch, ONNX FP32, and ONNX FP16. Additionally, our methodology includes an assessment of each model across four different frame lengths: 40, 60, 90, and 120 frames. To ensure a comprehensive, we randomly select 5000 consecutive frames from each motion in the evaluation datasets. This selection process is crucial to prevent skewing the results with excessively long motions, and to avoid the evaluation of root translation error at the beginning of motions, which typically start from a stationary standing position.

5.2 Quality of Quantization and Input Length

In Figure 5.2, our metrics are mainly based on the comparison between model prediction results and actual ground-truth.

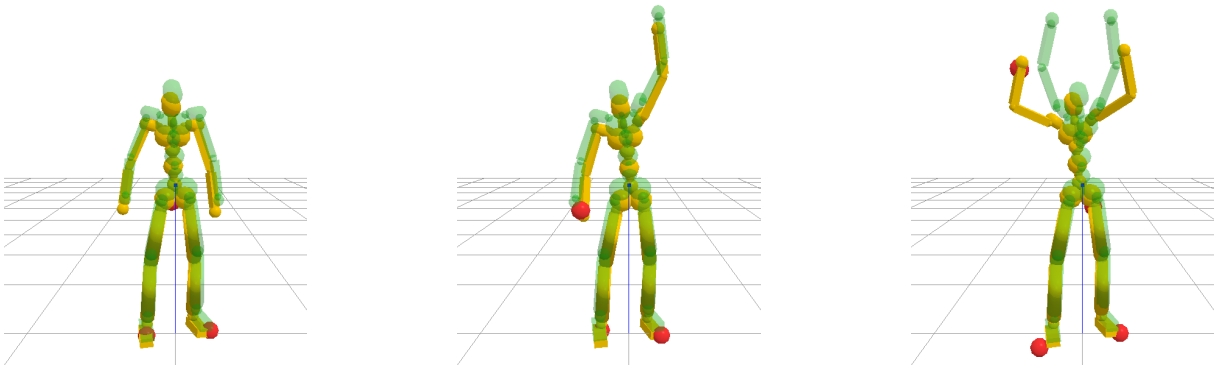

Fig. 5.2 Compare model results (yellow character) with ground-truth (green character)

- **Mean Joint Angle Error (in degrees):** Joint angle (represented in axis-angles) difference between reconstruction and ground-truth, averaged over all joints.

  With n joints, and θ_i^{recon}, θ_i^{gt} represent the reconstructed and ground-truth angles for the i^{th} joint, then the Mean Joint Angle Error is calculated as equation 5.1:

$$\text{Mean Joint Angle Error} = \frac{1}{n}\sum_{i=1}^{n}\left\|\theta_i^{\text{recon}} - \theta_i^{\text{gt}}\right\|_2^2 \tag{5.1}$$

- **Mean Root-Relative Joint Position Error (in centimeters):** Global joint Cartesian position difference (Euclidean norm) between the reconstruction and ground-truth by aligning at the root, averaged over all joints.

  If p_i^{recon} and p_i^{gt} represent the reconstructed and ground-truth positions of the i^{th} joint then the error for that joint is the Euclidean distance between these two points. The Mean Root-Relative Joint Position Error is then the average of these errors across all n joints:

$$\text{Mean Squared Relative Joint Position Error} = \frac{1}{n}\sum_{i=1}^{n}\left\|p_i^{\text{recon}} - p_i^{\text{gt}}\right\|_2^2 \tag{5.2}$$

- **Root Error 2s/5s/10s (in meters):** Root translation error measured in Euclidean norm during a continuous period of 2s/5s/10s. Since the formulas are similar, here we only

present the formulas of Root Error 2s. At each time point within this 2-second interval, if R_t^{recon} and R_t^{gt} represent the root positions at time t in the reconstructed and ground-truth motions, respectively, then the difference at time t is:

$$\Delta R_t = (R_t^{\text{recon}} - R_t^{\text{gt}})^2 \tag{5.3}$$

Then average these Euclidean norms over all the time points in the 2-second interval. If you have n time points in your 2-second interval, the Root Error 2s is calculated as:

$$\text{Root Error 2s} = \frac{1}{n}\sum_{t=1}^{n} \|\Delta R_t\| \tag{5.4}$$

We present our result quantitative metrics on the evaluation dataset in Table 5.1:

Table 5.1 Comparison of model quality on evaluation datasets.

Engine	PyTorch			ONNX FP32			ONNX FP16		
	40	60	120	40	60	120	40	60	120
Joint Angle Error	10.008	9.833	**5.385**	6.609	6.474	6.431	6.604	6.429	6.591
Joint Position Errors	6.144	6.064	3.289	3.370	3.335	3.184	3.358	3.323	**3.193**
Root Errors in 2s	0.022	0.016	0.025	0.012	**0.010**	0.011	0.012	0.010	0.011
Root Errors in 5s	0.032	0.017	0.012	**0.012**	0.013	0.014	0.013	0.014	0.014
Root Errors in 10s	0.047	0.024	0.029	0.011	**0.008**	0.009	0.011	0.009	0.009

The results indicate that converting from a PyTorch model to an ONNX model does not degrade the model's quality. The Joint Angle Error metric is the only indicator showing that the PyTorch model is performing better in its task. However, across all other metrics, the victory goes to the ONNX models in both FP32 and FP16 formats. Notably, the ONNX FP16 model has halved in size, yet its model quality remains reasonably high. This suggests that this model can be deployed on hardware-constrained devices while still achieving promising results.

The successful conversion from PyTorch to ONNX without substantial quality loss indicates the compatibility and adaptability of the model across different frameworks and hardware setups. Particularly, the reduced size and maintained performance of the ONNX FP16 model demonstrate its potential for deployment in resource-constrained environments, opening up possibilities for its use in scenarios with limited hardware capabilities.

5.3 Performance on Types of Hardware

We provide a detailed description of each metric below:

- Model runtime (s): The amount of time an AI model takes to process or infer a single batch of data.
- Engine runtime (s): This measures the performance of the software running model.
- Process speed (FPS): The number of frames that the system can process or display each second. The formula to calculate Process speed is:

$$\text{Process speed} = \frac{\text{total frames}}{\text{engine runtime}} \tag{5.5}$$

  With the *total frames* fixed at 5000 frames.

- Memory usage (MB): This refers to the amount of VRAM (Video Random Access Memory) being utilized.

In Table 5.2, for benchmark performance of engine on types of hardware, we used some option, with the following specific configuration :

Table 5.2 Hardware information used for benchmark.

GPU Name	VRAM	TFLOPS	CPU cores	CPU clock	RAM
Tesla T4	16GB	8.14	2	2.5 GHz	12GB
RTX 2060	6GB	6.45	12	4.3 GHz	16GB
Tesla M40	24GB	6.83	48	2.5 GHz	32GB
GTX 1060	4GB	1.86	8	4.0 GHz	16GB

We have presented the results in Table 5.3, Table 5.4, Table 5.5, Table 5.6:

Table 5.3 Comparison of model performance, benchmark on Tesla T4

Engine	PyTorch			ONNX FP32			ONNX FP16		
Num of frames	40	60	120	**40**	**60**	120	**40**	**60**	120
Model runtime (s)	44.1	46.7	53.5	**17.8**	**19.7**	28.4	**17.7**	**19.3**	27.0
Engine runtime (s)	114.8	126.1	163.7	**63.0**	**74.8**	113.8	**62.7**	**74.2**	111.5
Process speed (fps)	43.6	39.6	30.5	**79.3**	**66.8**	43.9	**79.7**	**67.4**	44.8
Memory usage (MB)	172	176	184	**152**	**156**	184	**156**	**160**	184

Table 5.4 Comparison of model performance, benchmark on RTX 2060 Super

Engine	PyTorch			ONNX FP32			ONNX FP16		
Num of frames	40	60	120	**40**	**60**	120	**40**	**60**	120
Model runtime (s)	56.1	58.7	66.2	**20.4**	**25.0**	28.7	**20.0**	**23.2**	33.2
Engine runtime (s)	89.9	95.1	123.7	**47.6**	**64.0**	97.5	**49.1**	**58.0**	88.6
Process speed (fps)	55.6	52.6	40.4	**105.0**	**78.1**	51.3	**101.8**	**86.2**	56.4
Memory usage (MB)	176	180	188	**156**	**160**	188	**160**	**164**	188

Table 5.5 Comparison of model performance, benchmark on Tesla M40

Engine	PyTorch			ONNX FP32			ONNX FP16		
Num of frames	40	60	120	**40**	**60**	120	**40**	**60**	120
Model runtime (s)	52.4	55.5	63.6	**21.2**	**23.4**	33.8	**21.0**	**22.9**	32.1
Engine runtime (s)	86.1	94.6	122.8	**47.3**	**56.1**	85.3	**47.1**	**55.6**	83.6
Process speed (fps)	58.1	52.9	40.7	**105.8**	**89.1**	58.6	**106.3**	**89.9**	59.8
Memory usage (MB)	184	188	200	**168**	**172**	200	**168**	**172**	200

Table 5.6 Comparison of model performance, benchmark on GTX 1060

Engine	PyTorch			ONNX FP32			ONNX FP16		
Num of frames	40	60	120	40	60	120	40	60	120
Model runtime (s)	183.4	194.2	222.7	74.2	81.9	118.1	73.5	80.3	112.2
Engine runtime (s)	210.4	225.5	270.1	95.1	108.1	159.4	94.3	106.4	153.5
Process speed (fps)	23.8	22.2	18.5	52.6	46.3	31.4	53.0	47.0	32.6
Memory usage (MB)	220	256	268	210	224	236	212	216	232

Based on the model performance benchmark tables provided, we can conduct an evaluation based on the crucial criterion of processing speed (fps - frames per second). This metric is essential because, to achieve real-time performance, the model must process at least 60 frames per second.

In Tesla T4, referring to table 5.3, models using ONNX FP32 and ONNX FP16 with frames of 40 and 60 can demo in real-time with respective speeds of 79.3 and 79.7 fps (ONNX FP32) and 101.8 and 86.2 fps (ONNX FP16). PyTorch does not meet the real-time requirements in any case. Similar results also occurred in M40 (table 5.5) and RTX2060 Super (table 5.4). On table 5.6, no case in GTX 1060 achieves the fps threshold for real-time performance. The highest speed is 53.0 fps with ONNX FP16 at 40 frames.

While encountering the challenge of the ONNX model falling short of reaching 60 fps on the older GTX1060 device, it can not diminish the value of the conversion. Conversely, on devices like the Tesla T4, Tesla M40, and particularly the RTX 2060 Super – our targeted device – they all significantly surpassed the 60 fps threshold. This superiority even extended up to an impressive 105 fps, showcasing the potency of the ONNX model on hardware.

For an real-time demo, we suggest to utilize the ONNX FP16 model due to its superior performance in achieving high frame rates across various tests. In terms of hardware, the Tesla T4, RTX 2060 Super and Tesla M40 GPUs stand out as process speed, with the Tesla M40 slightly edging out in performance at higher frame rates. However, we highly recommend the RTX 2060 Super as it is more accessible to non-IT professionals. Employing the ONNX FP16 model in conjunction with either the RTX 2060 Super will ensure that the demo not only stable at 60 fps threshold but also maintains a robust performance buffer to handle more complex tasks or any additional computational load that may arise during the demonstration. In the future, if our input devices become more modern and our models become even more optimized, we believe we can run real-time demo at 120 fps.

With such impressive outcomes, we were exhibiting the project's real-time capabilities and confirming the accomplishment of our objectives. Importantly, the ONNX model has proven its capacity to enhance performance, even on devices considered less advanced. This holds significant implications for expanding the project's applicability across various devices and diverse environments.

5.4 Hardware Responsiveness Analysis

To evaluate, we benchmarked on the RTX 2060 Super with ONNX FP32 model and number of frames is 40 when increasing the number of characters to get this as close to a real-time demo as possible. The results are presented in Table 5.7:

Table 5.7 Model performance when increasing the number of characters.

Num of characters	**1**	**2**	4	8
Model runtime (s)	**20.35**	**24.43**	29.31	35.17
Engine runtime (s)	**47.62**	**71.43**	107.14	160.71
Process speed (fps)	**105.0**	**70.0**	46.7	31.1
Memory usage (MB)	**184**	**280**	520	960

The model displays optimal efficiency with a single character, achieving a high process speed of 105 fps and maintaining low memory usage at 184 MB. Doubling the character count to two results in a slight performance decrease, with a process speed of 70 fps, yet remaining within real-time processing capabilities. However, as characters increase to four and eight, the process speed falls significantly below the real-time threshold (46.7 fps and 31.1 fps, respectively).

A notable observation is the model's efficient scaling, with the runtime increasing by a factor of approximately 1.3 to 1.5 when doubling the character count. This scaling behavior suggests the potential efficiency gains of a shared computation engine for increased batch sizes. Nevertheless, beyond a certain point, the model encounters diminishing returns, underscoring the importance of a nuanced approach in character count considerations.

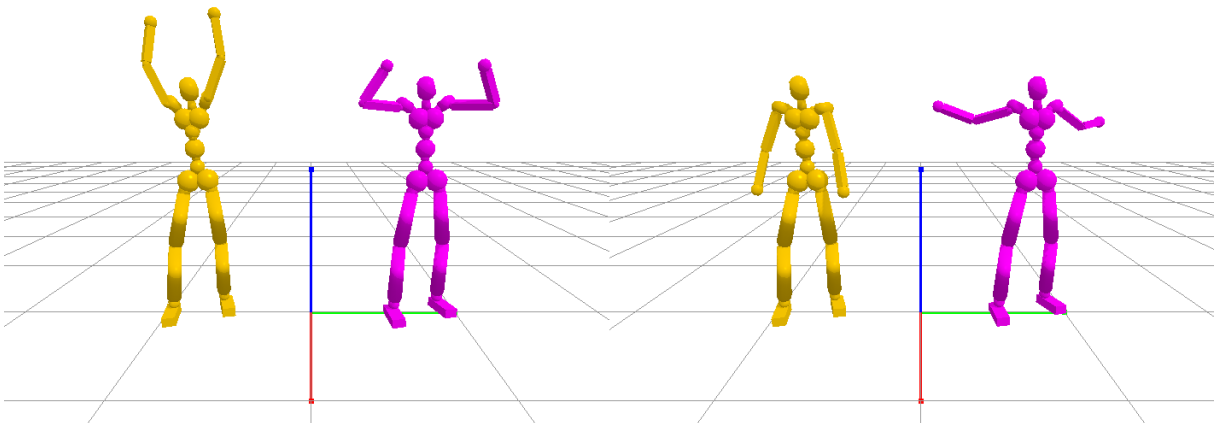

Fig. 5.3 Processing with 2 characters

6. DISCUSSIONS

6.1 Application of Motion Capture

Our project not only succeeded in optimizing the Transformer Inertial Poser (TIP) engine with ONNX Runtime but also went above and beyond by enhancing its usability and versatility. The native TIP program showcased remarkable real-time performance, achieving 65 fps on the RTX 3090 GPU, a testament to its prowess in pose estimation. However, the exclusive reliance on the Ampere architecture limited its accessibility to a broad user base.

To address this challenge, our optimization efforts resulted in a significant performance boost—TIP now runs 2-3 times faster, making it compatible with a spectrum of older GPUs, including the Turing architecture (RTX 2000 series, Tesla T4), Pascal architecture (GTX 1000 series, P4 series), and Volta architecture (Tesla M40). This strategic expansion of compatibility democratizes access to TIP, ensuring that its cutting-edge pose estimation capabilities are available to users with diverse hardware configurations.

Moreover, our project brings practicality to the forefront by seamlessly integrating the HiPNUC's IMU sensor product line. Beyond performance enhancements, this integration introduces a cost-effective solution without compromising accuracy. By doing so, our project acknowledges the importance of real-world applications, making TIP an economically feasible choice for various industries.

In a forward-looking move, we implemented a socket-based real-time export feature in SMPL format. This functionality transforms TIP into a versatile tool, allowing users to connect with external applications such as Unity, Blender, and VR Chat. The adoption of the SMPL format ensures compatibility and interoperability, unlocking a multitude of possibilities in virtual reality experiences, animation, and interactive simulations.

In essence, our project transcends the traditional boundaries of pose estimation. It not only optimizes performance but strategically aligns TIP with practical needs, broadens its accessibility, and fosters interdisciplinary collaborations through real-time export capabilities. TIP emerges not just as a cutting-edge technology but as a solution that empowers users across diverse domains to redefine possibilities in human pose estimation.

6.2 Future of Motion Capture

While our current project focused on optimizing the speed of the ”Transformer Inertial Poser” (TIP) engine, the horizon presents exciting possibilities for enhancing both speed and accuracy. It’s important to note that our optimization efforts concentrated on engine speed, and we did not make modifications to improve the model’s accuracy. As a result, there are certain poses that the current model may struggle to estimate accurately.

In the near future, we envision a refinement of TIP’s accuracy by incorporating data from six IMU sensors and scaling up the number of sensors for each character to a range of 12 to 18 sensors. This expansion aims to provide a more comprehensive and detailed understanding of the user’s motion, enabling TIP to accurately capture poses that may currently pose challenges.

Beyond sensor augmentation, our future endeavors include integrating Indoor Positioning Systems (IPS), such as Ultra-Wideband (UWB), to endow characters with absolute positioning. This feature is particularly valuable for applications in virtual reality (VR) games with expansive spaces or the development of digital museums. Absolute positioning not only enhances the immersive experience in VR but also opens avenues for the creation of highly detailed computer-generated imagery (CGI) in various high-resolution applications.

By combining increased sensor density, enhanced accuracy and absolute positioning, we anticipate elevating TIP to a new standard of precision. This trajectory aligns with the demands of VR gaming, CGI applications, and innovative projects in the digital realm. The pursuit of accuracy and absolute positioning not only enriches user experiences but also lays the foundation for pushing the boundaries of what is achievable in the dynamic field of motion capture.

In summary, the future of motion capture with TIP unfolds as a journey toward heightened accuracy, expanded sensor capabilities, and the integration of absolute positioning technologies, promising a new era of possibilities in virtual environments and advanced CGI solutions.

7. CONCLUSIONS

In conclusion, our project has achieved significant milestones in optimizing the ”Transformer Inertial Poser” (TIP) engine [33], thereby enhancing its speed and accessibility. The integration of the HiPNUC IMU sensor product line not only bolstered the cost-effectiveness of TIP [33] but also broadened its applicability, making advanced pose estimation technology accessible to a wider audience. The strategic decision to implement ONNX Runtime over PyTorch played a crucial role in reducing processing time and ensuring compatibility across various hardware systems. The success of ONNX Runtime in streamlining TIP's execution represents a pivotal aspect of our optimization efforts.

Looking forward, our roadmap envisions a two-pronged approach: refining accuracy and expanding application domains. With more time and additional computational resources, we propose extending the size of the model to address specific pose scenarios where TIP currently encounters challenges in accurate estimation. This enhancement involves incorporating more parameters and features, making TIP more adept at capturing intricate nuances of human motion. Additionally, our intent is to continue fortifying TIP's capabilities by addressing pose-specific challenges through meticulous model refinement and parameter tuning.

Furthermore, our success extends to the seamless integration of multiple characters and improved connectivity to various applications through the utilization of the SMPL format. The ability of TIP to handle multiple characters and export in SMPL format not only underscores its scalability but also opens new possibilities for collaborative ventures and innovative applications in virtual reality, animation, and beyond. This multifaceted success positions TIP as a versatile and dynamic solution, ready to meet the demands of diverse applications and future advancements in the field of motion capture.

References

[1] R. J. Roesthuis, M. Kemp, J. J. van den Dobbelsteen, and S. Misra, "Three-dimensional needle shape reconstruction using an array of fiber bragg grating sensors," *IEEE/ASME Transactions on Mechatronics*, vol. 19, pp. 1115–1126, 2014. [Online]. Available: https://doi.org/10.1109/tmech.2013.2269836

[2] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[3] X. Yi, Y. Zhou, M. Habermann, S. Shimada, V. Golyanik, C. Theobalt, and F. Xu, "Physical inertial poser (pip): Physics-aware real-time human motion tracking from sparse inertial sensors," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[4] A. Graves, *Long short-term memory*, 2012, pp. 37–45.

[5] L. J. LR Medsker, "Recurrent neural networks," 2001.

[6] A. Vaswani *et al.*, "Attention is all you need," arXiv.org, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[7] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. Black, "Amass: Archive of motion capture as surface shapes," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[8] M. Loper *et al.*, *SMPL: A skinned multi-person linear model*, 2019, pp. 851–866.

[9] J. Bai, F. Lu, K. Zhang *et al.*, "Onnx: Open neural network exchange," https://github.com/onnx/onnx, 2019.

[10] J. S. Zhe Jia, Marco Maggioni and D. P. Scarpazza., "Dissecting the nvidia turing t4 gpu via microbenchmarking," arXiv.org, 2019. [Online]. Available: https://arxiv.org/pdf/1903.07486.pdf

[11] S. L. Colyer, M. Evans, D. P. Cosker *et al.*, "A review of the evolution of vision-based motion analysis and the integration of advanced computer vision methods towards developing a markerless system," *Sports Medicine - Open*, vol. 4, no. 24, 2018. [Online]. Available: https://doi.org/10.1186/s40798-018-0139-y

[12] E. Muybridge, *Complete Human and Animal Locomotion: All 781 Plates from the 1887 Animal Locomotion*, A. Cappozzo, M. Marchetti, and V. Tosi, Eds. Rome: Promograph, 1979.

[13] V. Cimolin and M. Galli, "Summary measures for clinical gait analysis: A literature review," *Gait & Posture*, vol. 39, no. 4, pp. 1005–1010, 2014.

[14] D. Cosker, P. Eisert, and V. Helzle, "Facial capture and animation in visual effects," in *Digital Representations of the Real World: How to Capture, Model, and Render Visual Reality*, M. A. Magnor, O. Grau, O. Sorkine-Hornung, and C. Theobalt, Eds. Boca Raton, FL: CRC Press, 2015, pp. 311–321.

[15] T. B. Moeslund, A. Hilton, and V. Krüger, "A survey of advances in vision-based human motion capture and analysis," *Computer Vision and Image Understanding*, vol. 104, no. 2-3, pp. 90–126, 2006.

[16] L. Mündermann, S. Corazza, and T. P. Andriacchi, "The evolution of methods for the capture of human movement leading to markerless motion capture for biomechanical applications," *Journal of NeuroEngineering and Rehabilitation*, vol. 3, no. 1, 2006.

[17] M. B. Holte, C. Tran, M. M. Trivedi, and T. B. Moeslund, "Human pose estimation and activity recognition from multi-view videos: Comparative explorations of recent developments," *IEEE Journal of Selected Topics in Signal Processing*, vol. 6, no. 5, pp. 538–552, 2012.

[18] S. X. M. Yang, M. S. Christiansen, P. K. Larsen, T. Alkjær, T. B. Moeslund, E. B. Simonsen, and N. Lynnerup, "Markerless motion capture systems for tracking of persons in forensic biomechanics: An overview," *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, vol. 2, no. 1, pp. 46–65, 2013.

[19] E. R. Bachmann, R. B. McGhee, X. Yun, and M. J. Zyda, "Inertial and magnetic posture tracking for inserting humans into networked virtual environments," in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '01, 2001, pp. 9–16.

[20] M. B. Del Rosario, H. Khamis, P. Ngo, N. H. Lovell, and S. J. Redmond, "Computationally efficient adaptive error-state kalman filter for attitude estimation," *IEEE Sensors Journal*, vol. 18, no. 22, pp. 9332–9342, 2018.

[21] E. Foxlin, "Inertial head-tracker sensor fusion by a complementary separate-bias kalman filter," in *Proceedings of the IEEE 1996 Virtual Reality Annual International Symposium*, 1996, pp. 185–194.

[22] R. V. Vitali, R. S. McGinnis, and N. C. Perkins, "Robust error-state kalman filter for estimating imu orientation," *IEEE Sensors Journal*, vol. 21, no. 3, pp. 3561–3569, 2021.

[23] Xsens, "Xsens https://www.xsens.com/," n d, last visited: 08/26/2022.

[24] Rokoko, "Rokoko https://www.rokoko.com/," n d, last visited: 08/26/2022.

[25] T. von Marcard, B. Rosenhahn, M. Black, and G. Pons-Moll, "Sparse inertial poser: Automatic 3d human pose estimation from sparse imus," *Computer Graphics Forum 36(2), Proceedings of the 38th Annual Conference of the European Association for Computer Graphics (Eurographics)*, pp. 349–360, 2017.

[26] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *ACM TOG*, vol. 34, no. 6, pp. 248:1–248:16, Oct. 2015.

[27] Y. Huang, M. Kaufmann, E. Aksan, M. J. Black, O. Hilliges, and G. Pons-Moll, "Deep Inertial Poser: Learning to reconstruct human pose from sparse inertial measurements in real time," *ACM TOG*, vol. 37, no. 6, 12 2018.

[28] D. Nagaraj, E. Schake, P. Leiner, and D. Werth, "An rnn-ensemble approach for real time human pose estimation from sparse imus," in *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*, ser. APPIS 2020, 2020.

[29] X. Yi, Y. Zhou, and F. Xu, "TransPose: Real-time 3d human translation and pose estimation with six inertial sensors," *ACM TOG*, vol. 40, no. 4, 8 2021.

[30] D. Rempe, T. Birdal, A. Hertzmann, J. Yang, S. Sridhar, and L. J. Guibas, "Humor: 3d human motion model for robust pose estimation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 11 488–11 499.

[31] X. Yi, Y. Zhou, M. Habermann, S. Shimada, V. Golyanik, C. Theobalt, and F. Xu, "Physical inertial poser (pip): Physics-aware real-time human motion tracking from sparse inertial sensors," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[32] S. Shimada, V. Golyanik, W. Xu, and C. Theobalt, "PhysCap: Physically plausible monocular 3d motion capture in real time," *ACM TOG*, vol. 39, no. 6, 12 2020.

[33] Y. Jiang, Y. Ye, D. Gopinath, J. Won, A. W. Winkler, and C. K. Liu, "Transformer inertial poser: Real-time human motion reconstruction from sparse imus with simultaneous terrain generation," in *SIGGRAPH Asia 2022 Conference Papers*, ser. SA '22 Conference Papers, 2022.

Appendix

8.1 Rotation Matrices and Rotate Vector

Rotation matrices are fundamental mathematical tools used to represent rotations in n-dimensional spaces, particularly in three-dimensional space (3D). They play a crucial role in various fields, including computer graphics, robotics, physics, and engineering.

In the realm of three-dimensional space, the ability to manipulate and transform objects is fundamental to various fields, ranging from computer graphics to robotics. Rotation, a pivotal transformation, allows us to reorient objects in space, opening avenues for a multitude of applications. Central to the mathematics of rotation is the concept of rotation matrices, elegant mathematical constructs that encapsulate the essence of spatial reorientation.

8.1.1 Rotation Matrix

A rotation matrix is a mathematical tool used to perform rotations in three-dimensional space. It represents a rotation transformation by describing how the coordinates of points in space change as a result of the rotation. The elegance of rotation matrices lies in their simplicity and efficiency in expressing complex spatial transformations.

A 3D rotation matrix is a 3x3 matrix of the form:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Here, θ represents the angle of rotation. The structure of the matrix allows us to seamlessly perform rotations around the three axes of the coordinate system (X, Y, and Z).

8.1.2 Rotate Vector

To rotate a point $P(x,y,z)$ using the rotation matrix R, the rotated point P' is obtained through matrix multiplication:

$$P' = R \cdot P$$

This concise formula embodies the transformative power of rotation matrices, enabling precise and efficient rotations in three-dimensional space.

8.1.3 Quaternion

In mathematics, the quaternion extends the complex numbers. Quaternions were first described by Irish mathematician William Rowan Hamilton in 1843 and applied to mechanics in three-dimensional space. Hamilton defined a quaternion as the quotient of two directed lines in a three-dimensional space, or, equivalently, as the quotient of two vectors. Multiplication of quaternions is noncommutative $i \times j \neq j \times i$.

Quaternions are generally represented in the form:

$$a + bi + cj + dk \quad \text{and} \quad i^2 = j^2 = k^2 = ijk = -1$$

where a, b, c, and d are real numbers; whilst i, j, and k are the basic quaternions.

Quaternions are used in pure mathematics, but also have practical uses in applied mathematics, particularly for calculations involving three-dimensional rotations, such as in three-dimensional computer graphics, computer vision, and crystallographic texture analysis. They can be used alongside other methods of rotation, such as Euler angles and rotation matrices, or as an alternative to them, depending on the application.

The quaternion representation of the rotation may be expressed as

$$q = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)(u_b i + u_c j + u_d k)$$

, where θ is the angle of rotation and $[u_b, u_c, \text{ and } u_d]$ is the axis of rotation.

With quaternion vector $[W, X, Y, Z]$

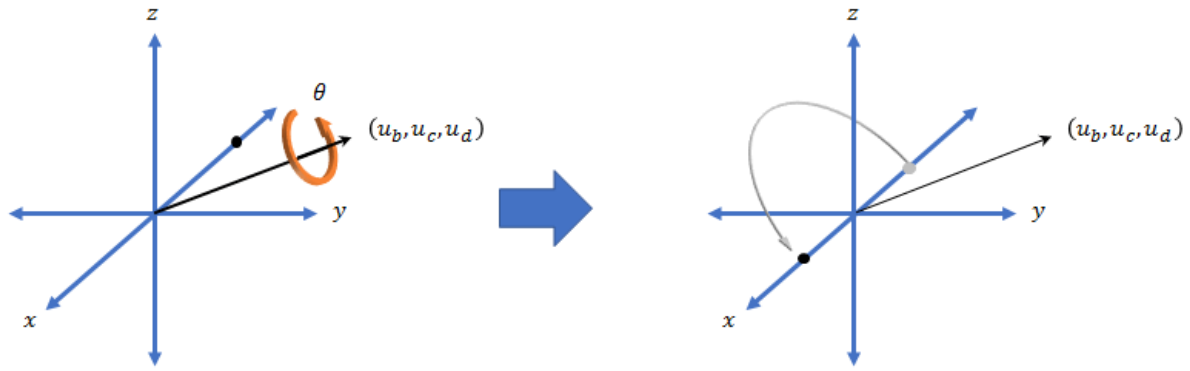

Fig. 8.1 Present of quaternion vector

$$q = \left(\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right) a\right) = (w,(x,y,z))$$

Convert to quaternion to rotation matrix

$$R_q = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy & 0 \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx & 0 \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

8.2 A Skinned Multi-Person Linear Model (SMPL)

SMPL is a realistic 3D model of the human body that is based on skinning and blend shapes and is learned from thousands of 3D body scans. This site provides resources to learn about SMPL, including example FBX files with animated SMPL models, and code for using SMPL in Python, Maya and Unity.

8.2.1 Key Features of SMPL

- **Parameterization:** SMPL represents the human body through a set of parameters that control its shape and pose. This allows for compact and efficient storage of 3D data compared to storing raw mesh vertices.
- **Blend Shapes:** The model incorporates blend shapes to capture variations in body shape

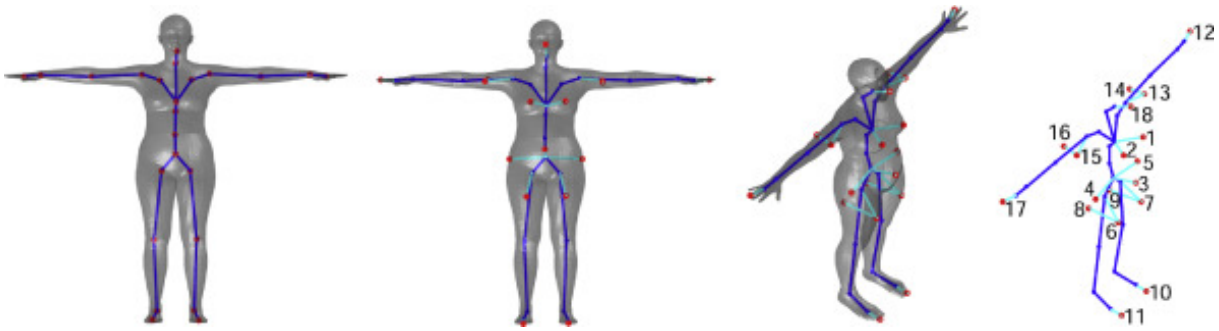

Fig. 8.2 Relationship between SMPL and 18 joints. Left is SMPL and right is the skeleton system with all 18 joints defined.

across different individuals. These blend shapes allow the model to adapt to a diverse range of body types.

- **Skinning:** SMPL utilizes skinning to bind the body mesh to a skeleton, enabling realistic animation and pose deformation.
- **Multi-Person:** The model can represent multiple people simultaneously, making it suitable for applications involving groups or interactions.

8.2.2 Structure of SMPL Data

The SMPL data format typically consists of the following components:

- **Template mesh:** This is a base mesh representing the average human body shape.
- **Blend shape coefficients:** These coefficients control the amount of influence each blend shape has on the final body shape.
- **Pose parameters:** These parameters specify the rotations of the skeleton joints, determining the body pose.
- **Joint weights:** These weights indicate how much influence each joint has on the deformation of nearby vertices.