Hand Tracking and Object Detection for

Assembly Process and Steps Verification

YE QING, A0304458A, National University of Singapore

Supervisor: Prof. Ong , Prof. Nee

National University of Singapore

College of Design and Engineering

4/13/2024

Abstract

Ensuring procedural correctness and quality in manual assembly remains critical in flexible manufacturing, yet traditional monitoring is prone to error and lacks objective quantification. This work addresses the need for robust, automated verification of assembly tasks by proposing an integrated vision-based system that monitors operator hand movements and assembly state progression. The system utilizes a calibrated multi-camera setup (RGB-D + stereo RGB) for comprehensive data capture, enabling robust 3D hand keypoint reconstruction via sensor fusion, with initial 2D detection performed by MediaPipe. A novel Multi-View Sliding YOLO Group architecture is introduced for accurate assembly stage identification, enhancing discrimination between sequential states compared to conventional approaches. Furthermore, operator motion conformity is quantitatively assessed using a Transformer-based Siamese network, evaluating the similarity between captured 3D hand kinematic sequences and standard action templates. These components are integrated within a PyQt-based graphical user interface for operator guidance and feedback. Experiments validated the proposed YOLO Group's enhanced state classification accuracy, particularly for adjacent stages, and confirmed the Transformer model's effectiveness in motion similarity assessment (~84% accuracy on frame-filtered data), while also highlighting the impact of sequence length normalization. This study contributes a comprehensive framework for automated assembly verification, advancing quality assurance capabilities in human-centric manufacturing environments. All related codes in this project have been open-sourced in github.com/0123YHYQ1129/ME5001_Hand_tracking_and_objection_detetction.

Keywords: Assembly Process Verification, Hand Tracking, YOLO-based Object Detection, Motion Similarity Assessment, Computer Vision.

List of Abbreviations

Abbreviations	Definition
QA	Quality assurance
ML	Machine learning
UI	User interface
LSTM	Long Short-Term Memory
HRC	Human-robot collaboration
CM	Cellular manufacturing
YOLO	You Only Look Once
MCT	Multi-camera multi-object tracking
DTW	Dynamic Time Warping
GCNs	Graph convolutional networks
DLT	Direct Linear Transformation
SVD	Singular Value Decomposition
CNN	Convolutional neural network
Q(transformer)	Query
K(transformer)	Key
V(transformer)	Value
BCE	Binary Cross-Entropy
GUI	Graphical User Interface
MHSA	Multi-head self-attention
FFN	Position-wise feed-forward network
MLP	Multi-Layer Perceptron
P(YOLO)	Precision
R(YOLO)	Recall
BCE	Binary Cross-Entropy
GUI	Graphical user interfaces

List of Figures

List of Tables

Contents

1. Introduction

For decades, the drive towards smart manufacturing focused on achieving high quality, stability, and low production costs, largely through the replacement of human workers by machines and robots [1]. Nevertheless, the emergence of advanced technologies like Big Data and IoT, alongside increasing demands for personalized services, has revealed that full automation may not always satisfy dynamic production requirements or prove optimally cost-effective. This has prompted a redefinition of the human-machine relationship in manufacturing environments. Meanwhile, strategies where human operators, aided by equipment, execute assembly tasks while also contributing to quality assessment are now widely acknowledged.

For many years, quality assurance (QA) in industrial manufacturing has relied on human inspectors to identify defects as parts move along the production assembly line. However, the repetitive nature of assembly operations makes human quality assurance both time-consuming and prone to errors, potentially resulting in defective or incomplete products reaching customers. To address these challenges, this study proposes the application of machine learning (ML) technologies to develop a vision-based QA system. The system monitors workers' hand movements to ensure correct hand postures and adherence to the prescribed assembly sequence.

Machine learning has a wide range of applications in manufacturing, including predictive maintenance, quality assurance, and intelligent automation. The primary contribution of this work is the integration of multiple ML methods to enable hand tracking and object detection, thereby ensuring assembly quality. The proposed hand-tracking method uses computer vision to monitor workers' hand movements, ensuring compliance with the required assembly sequence. Concurrently, the detection algorithm identifies the current assembly stage and detects key points on workers' hands. By constructing time-series tensors of hand key points, the system evaluates whether the current hand movements meet predefined standards.

In this project, MediaPipe is utilized to extract hand keypoints, while YOLO is employed for assembly action recognition and stage identification. A novel sliding-stage YOLO classifier has been designed specifically to assess assembly stages, demonstrating superior performance compared to standard single-stage YOLO models. For the assessment of hand movement similarity, a similarity model based on advanced Transformer architecture is utilized. This model evaluates the similarity between the 3D temporal sequences of detected hand keypoints and those of standard action templates. To integrate these functionalities, a user interface (UI) based on the Qt framework has been developed, providing a comprehensive platform for quality assurance (QA) tasks.

2. Related work

2.1 Machine Learning-Driven Movement Analysis and Quality Assurance in Industrial Applications

In the context of Industry 4.0, visual control systems have become indispensable for ensuring assembly quality. Machine learning (ML)-based systems integrating hand tracking and object detection have been employed to monitor workers' hand movements and assembly sequences, thereby reducing human errors and increasing efficiency [2]. For instance, one recent approach proposed a real-time fine-grained assembly task recognition system combining You-Only-Look-Once (YOLO) for detecting potential task start points and a Long Short-Term Memory (LSTM) based classifier operating on 3D hand skeleton data to identify specific fine-grained actions, emphasizing hand-object interactions critical for operator efficiency assessment [3].

Jhen-Jia Hu developed a monitoring system for human-robot collaboration (HRC) of hybrid automation enabling avoidance of product defects and real-time warnings in cellular manufacturing (CM) [4]. Al-Amin et al. developed a multimodal sensor system combining the Myo armband (IMU and EMG signals) with the Kinect camera (skeleton tracking), using fused predictions from multiple CNN models to improve accuracy in assembly tasks [5]. Further advancing human-centric intelligent manufacturing, another line of research developed deep learning-based end-to-end models capable of not only recognizing actions during assembly processes but also predicting action completion times, leveraging comprehensive perception via sensors like Azure Kinect combined with algorithms like MediaPipe and YOLOv5, and using autoencoders for efficient process representation [6]. The broader field of movement analysis and quality assurance has witnessed significant advancements with the integration of ML and deep learning techniques, with movement similarity, trajectory analysis, and

action recognition emerging as pivotal areas. Collaborative human-robot efforts have also been explored to improve product assembly quality and reduce labor intensity.

2.2 Object Detection and Tracking in Industrial and Multi - Camera Scenarios

Object detection can be broadly categorized into two major paradigms: one-stage and two-stage methods. Among one-stage methods, the YOLO (You Only Look Once) family of algorithms is the most widely applied in industrial domains. YOLO has played a pivotal role in advancing real-time object detection. The original YOLO model [7] proposed a unified framework, with subsequent iterations like YOLOv2 [8], YOLOv3 [9], YOLOv4 [10], YOLOv5 [11], and YOLOv7 [12] progressively improving accuracy, speed, and efficiency through various architectural and optimization strategies.

In contrast, two-stage methods, such as the R-CNN series [13], while typically slower, provide higher detection accuracy by first generating region proposals and then classifying them. Subsequent advancements include novel approaches like class-agnostic object detection using discriminative embeddings and few-shot learning [14].

In multi-camera multi-object tracking (MCT), significant advances have addressed challenges like occlusion and data association, often integrating appearance and trajectory information across views [15]. Oyekan et al., utilizing RGB-D cameras, explored tracking for manual assembly, demonstrating the efficacy of such sensors for object recognition in industrial settings [16].

2.3 Movement Analysis and Similarity Measures

The study of movement similarity spans various domains. Methods for comparing movement trajectories often focus on spatial, temporal, and spatiotemporal parameters, assessing aspects like path resemblance, velocity profiles, and temporal synchronization [17]. Dynamic Time Warping (DTW) has been particularly effective for analyzing temporal data due to its resilience to outliers and its utility in quantifying similarities between sequences [18], with dynamic variants optimizing it for real-time updates. Addressing challenges like viewpoint variation and skeletal differences in motion comparison, recent work has explored deep metric learning approaches. One such method employs an autoencoder trained with a triplet loss function to map 2D skeleton sequences into a latent space where feature vectors are invariant to camera view and skeleton size, subsequently using DTW for temporal alignment to achieve more robust and subjectively accurate similarity evaluation [19].

2.4 Action Recognition and Human Motion Understanding

The advent of 3D skeleton-based action recognition has brought new insights into human motion analysis, often outperforming traditional RGB-based approaches by providing precise action representations. Foundational advancements in sequence modeling, such as the Transformer architecture which relies solely on attention mechanisms, dispensing with recurrence and convolutions, have significantly influenced this field [20]. Recent developments leveraging these principles include networks like MotionAGFormer, which integrate transformers with graph convolutional networks (GCNs) to enhance the understanding of 3D human poses [21]. Similarly, models like TemPose have

been designed for fine-grained motion recognition, prioritizing temporal and spatial relationships within action sequences.[22]

2.5 Synthesis and Future Directions

These advancements highlight the growing importance of ML-driven methods in movement analysis, trajectory classification, and industrial quality assurance. However, challenges remain in integrating multimodal data, adapting to unseen scenarios, and optimizing real-time performance. Future research should explore hybrid approaches that combine state-of-the-art techniques, such as transformers and GCNs [26], with robust similarity measures like DTW to further advance the field.

3. Research Methodology

3.1 Non-contact Data Acquisition System

3.1.1 Multi-sensor Configuration

To integrate depth information into the model and address data incompleteness caused by single-view occlusion, a customized mechanical framework is designed. This framework integrates a hybrid sensor system comprising an Azure Kinect depth sensor and dual RGB cameras (Logitech HD Pro C920), which synchronously captures 2D and 3D data from three orthogonal perspectives (top, left, and right views). The depth sensor is mounted atop the framework, while the RGB cameras are positioned bilaterally with adjustable positioning to ensure full coverage of the workspace. Detailed structural parameters of the framework are provided in the camera_structure.zip supplementary material, and the SOLIDWORKS schematic of the mechanical design is illustrated in Figure 1.

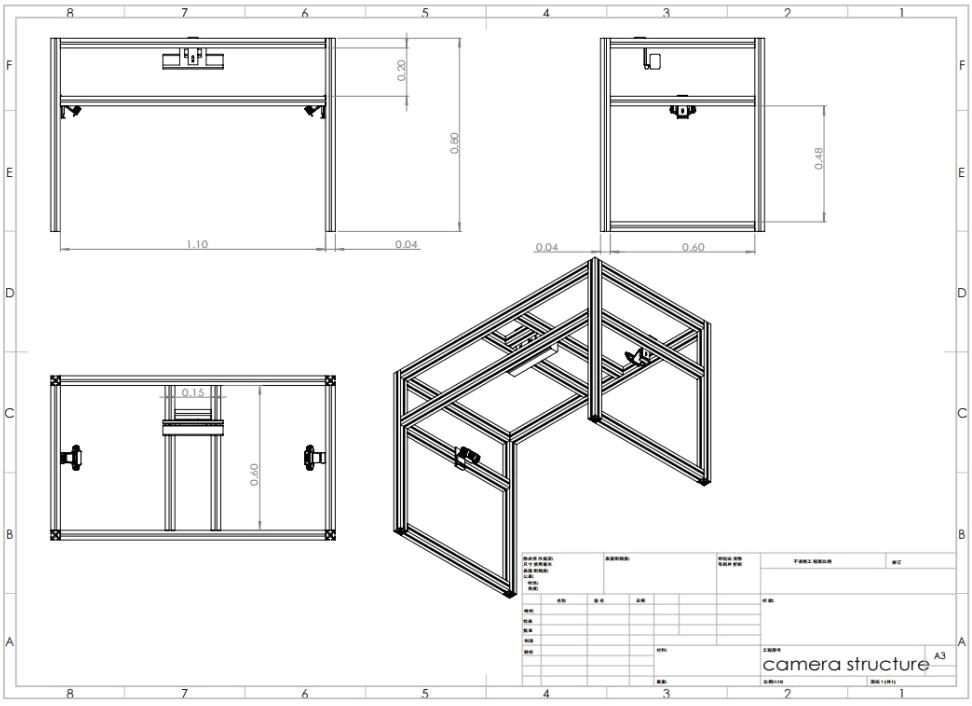


Figure 1 Mechanical Design of the Multi-Sensor Data Acquisition Framework.

3.1.2 Calibration and Data Fusion

Metrically accurate three-dimensional (3D) reconstruction and the coherent integration of data from heterogeneous sensors necessitate a rigorous calibration of the multi-camera system. This calibration process establishes the intrinsic characteristics of each sensor and their precise spatial interrelations within a unified coordinate framework, thereby providing the geometric foundation for subsequent data fusion and 3D coordinate estimation.

System Calibration. The calibration procedure is executed for the complete sensor suite, comprising two standard RGB cameras and one RGB-D Kinect sensor ($i \in \{1,2,3\}$). Employing established techniques, typically utilizing a planar checkerboard target observed from diverse viewpoints, the following parameter sets are estimated for each camera i:

1. **Intrinsic Parameters:** These parameters model the internal projective geometry of the camera, encompassing the focal lengths ($f_{x,i}, f_{y,i}$) and principal point coordinates ($c_{x,i}, c_{y,i}$), typically represented by the intrinsic matrix K_i. Additionally, lens distortion coefficients D_i (modeling radial and tangential distortions) are determined to enable correction of image coordinates or incorporation into the projection model. Figure 2 illustrates intrinsic parameter calibration using the chessboard method.

$K_i = \begin{bmatrix} f_{x,i} & s_i & c_{x,i} \\ 0 & f_{y,i} & c_{y,i} \\ 0 & 0 & 1 \end{bmatrix}$ (where skew s_i is often assumed negligible).

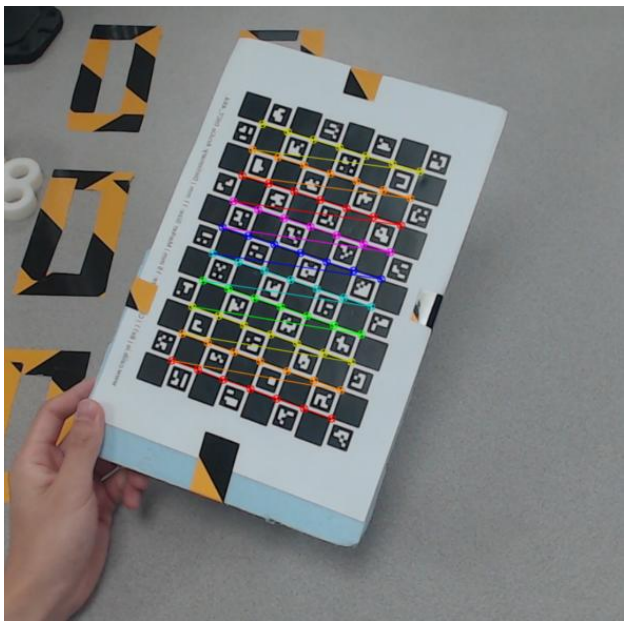


Figure 2 intrinsic parameter calibration using the chessboard method.

Extrinsic Parameters: These define the pose of each camera i relative to a common, arbitrarily chosen world coordinate system W. The Kinect camera's coordinate system is designated as the reference world coordinate frame W. The pose is parameterized by a 3×3 rotation matrix R_i and a 3×1 translation vector t_i, forming the rigid body transformation $[R_i|t_i]$ from W to the camera's local coordinate system $\mathrm{C_i}$.

The concatenation of intrinsic and extrinsic parameters yields the 3×4 Projection Matrix $\mathrm{P_i} = K_i[R_i|t_i]$ for each camera. This matrix encapsulates the complete perspective projection transformation, providing the mathematical model relating a homogeneous 3D world point $[X_j, Y_j, Z_j, 1]^T$ to its corresponding homogeneous image projection $(\mathrm{u_{ij}}, \mathrm{v_{ij}}, 1)^{\mathrm{T}}$ according to the fundamental projection equation $S_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = K_i \cdot [R_i|T_i] \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = P_i \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$. Here, is a scaling factor, denotes the intrinsic matrix, and (u_i, v_i) are normalized image coordinates. Figure 3 illustrates chessboard images captured from three different perspectives, which serve as input data for determining the extrinsic parameters.

The experimental methodology involved acquiring distinct image sets for calibration. Specifically, 15 images were initially captured from each viewpoint to compute the intrinsic parameters. Furthermore, a separate set of 15 images per viewpoint was designated for the calculation of extrinsic parameters. Post-acquisition, a data curation step was performed whereby images with inadequate focus or substantial chessboard recognition inaccuracies were excluded to improve the precision of the computed extrinsic parameters. The image dataset employed in this study is located within the 'frame' folder.

Figure 3 Chessboard images from three camera perspectives

Data Fusion Framework. The calibrated system parameters (K_i,D_i,R_i, t_i and P_i)are instrumental in enabling the fusion of information derived from the multiple sensors. For each identified hand keypoint j, the objective is to estimate its 3D coordinates $[X_j, Y_j, Z_j]^T$ in the world frame W by optimally integrating the available observations: (1) the 2D image coordinates $(u_{ij}, v_{ij})^T$ detected in each camera view i, and (2) the depth measurement d_{3j} provided by the Kinect sensor.

The fusion strategy capitalizes on the geometric constraints imposed by the multi-view setup and the direct range information from the RGB-D sensor. Multi-View Geometric Fusion: The primary fusion mechanism leverages the geometric consistency constraints across multiple views. Given the 2D observations $m_{ij} = (u_{ij}, v_{ij})^T$ and the corresponding projection matrices P_i, the 3D point $M_j = [X_j, Y_j, Z_j]^T$ can be estimated via triangulation. While linear methods like the Direct Linear Transformation (DLT) can provide an initial estimate by solving an overdetermined system of linear equations derived from the projection equations, superior accuracy is generally achieved through non-linear optimization. This typically involves minimizing the sum of squared reprojection errors over all available views:$\min_{M_j} \sum_{i=1}^{3} w_{proj,i} \left\| m_{ij} - \pi(P_i, M_j) \right\|^2$, where $\pi(P_i, M_j)$ denotes the projective function mapping M_j onto the i-th image plane and $w_{proj,i}$ represents weighting factors, often uniform initially. This formulation inherently integrates geometric information from all views, providing robustness through redundancy against noise and partial occlusions, provided the keypoint is reliably detected in at least two views.

Depth Information Integration: The framework is enhanced by incorporating the direct depth measurement d_{3j} from the Kinect sensor. This value provides an estimate of the point's depth in the Kinect's local coordinate

system, $Z_{c,3j}$. This depth relates to the world coordinates M_j via the Kinect's extrinsic parameters: $Z_{c,3j} = \left(R_3 M_j + t_3\right)_z$, where $(\cdot)_z$ denotes the Z-component. The depth information is typically integrated as an additional residual term within the non-linear least-squares optimization:

$$\underset{M_j}{\operatorname{argmin}} \left(\sum_{i=1}^{3} w_{proj,i} \left\| m_{ij} - \pi\left(P_i, M_j\right) \right\|^2 + w_{depth} \left\| \left(R_3 M_j + t_3\right)_z - d_{3j} \right\|^2 \right)$$

The weighting factor quantifies the influence of Kinect depth measurement error on the objective error function. Its value is critical and should ideally reflect the uncertainty associated with the depth measurement d_{3j}, which can vary significantly depending on factors like distance, surface reflectivity, and incidence angle. Empirical validation in this study adopts a default weighting factor of 0.2, determined through sensitivity analysis balancing measurement noise characteristics and system observability. It is noted that the intrinsic and extrinsic parameters fundamental to this experiment were obtained through a calibration procedure utilizing RGB images from three perspectives along with their concurrently captured depth maps. The resulting parameters are detailed in Table 1, 2, 3, 4.

Cameras	Intrinsic Matrix
Camera left	$\begin{bmatrix} 1414.58 & 0 & 972.63 \\ 0 & 1412.83 & 484.50 \\ 0 & 0 & 1 \end{bmatrix}$
Kinect	$\begin{bmatrix} 1071.36 & 0 & 960.58 \\ 0 & 1071.72 & 556.24 \\ 0 & 0 & 1 \end{bmatrix}$
Camera right	$\begin{bmatrix} 1400.25 & 0 & 936.72 \\ 0 & 1389.37 & 503.83 \\ 0 & 0 & 1 \end{bmatrix}$

Table 1 Intrinsic matrices of the cameras

Cameras	Distortion Coefficients

Camera left	$\begin{bmatrix} 0.061 \\ 0.26 \\ -0.001 \\ 0.00061 \\ -1.2774 \end{bmatrix}$
Kinect	$\begin{bmatrix} -0.0073 \\ 0.1262 \\ -0.00039 \\ -0.00252 \\ -0.14407 \end{bmatrix}$
Camera right	$\begin{bmatrix} 0.0295 \\ -0.15176 \\ 0.0056 \\ -0.011 \\ -0.1074 \end{bmatrix}$

Table 2 Distortion coefficient matrices of the cameras

Cameras	Rotation Matrix
Camera left	$\begin{bmatrix} -0.0085 & 0.9956 & 0.09297 \\ -0.832 & 0.0446 & -0.553 \\ -0.5547 & -0.082 & 0.828 \end{bmatrix}$
Kinect	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Camera right	$\begin{bmatrix} 0.0148 & -0.9977 & -0.0668 \\ 0.81178 & 0.0510 & -0.5817 \\ 0.5838 & -0.0456 & 0.8106 \end{bmatrix}$

Table 3 Rotation matrices of the cameras

Cameras	Translation Matrix
Camera left	$\begin{bmatrix} -9.2844 \\ 61.7920 \\ 13.3231 \end{bmatrix}^T$
Kinect	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}^T$

Camera right	$\begin{bmatrix} 8.4644 \\ 45.9886 \\ 0.6840 \end{bmatrix}^T$

Table 4 Translation matrices of the cameras

3.1.3 Data Annotation and Data Preprocessing

Keypoint Detection and Depth Acquisition. The primary annotation of 2D hand keypoints was performed using the MediaPipe Hand Landmarker framework [24]. Applied to the RGB video streams from all three cameras, the framework identified 21 distinct landmarks per detected hand in each frame. A detection confidence threshold of 0.9 was employed, discarding landmarks with lower confidence scores. To ensure the dataset captured only the right hand's motion, subjects wore an opaque black glove on the left hand, and the detection framework was configured for a maximum detection count of one hand (max_ num_ hands= 1). For the Kinect sensor(designated as camera $i = 3$) , after obtaining the 2D landmark coordinates (u_{3jt}, v_{3jt}) in its 1920×1080 resolution RGB image at frame t for keypoint j, the corresponding metric depth value d_{3jt} was retrieved. This was achieved by querying the spatially and temporally aligned depth map provided by the Kinect sensor at the pixel location (u_{3jt}, v_{3jt}). It is noted that this depth retrieval may yield invalid values (e.g,zero or specific error codes) in cases of occlusion or if the point lies outside the sensor's operational range. The two auxiliary RGB cameras operated at a resolution of 960×540 pixels.

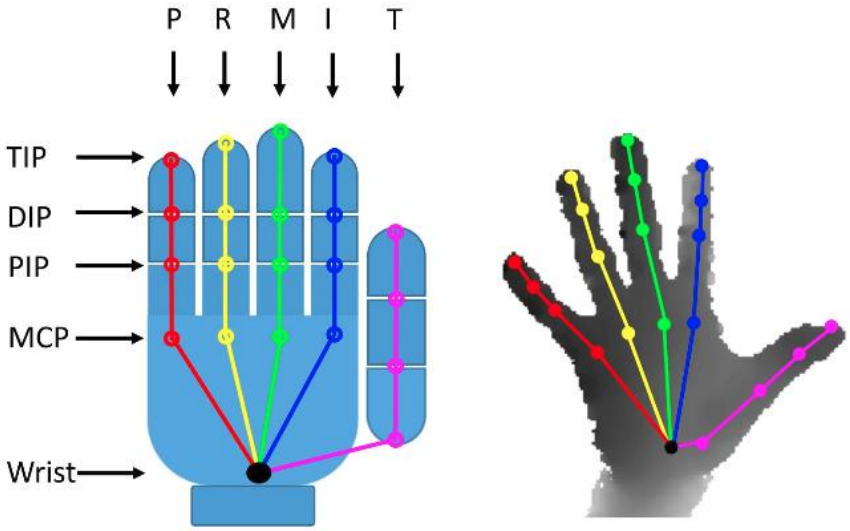


Figure 4 Diagram of the 21 hand landmarks detected by MediaPipe

Synchronized Data Acquisition. A critical component of the acquisition process was the temporal synchronization of all three cameras by opencv control. The system was configured to capture frames simultaneously across all sensors at a consistent frame rate of 30 fps. This synchronization ensured precise temporal alignment between the 2D keypoint data extracted from the RGB streams and the depth measurements from the Kinect sensor. Consequently, the acquisition process yielded three parallel, time-aligned data sequences for each recorded motion:

- Stream 1 (RGB Cam 1): $D_1 \in R^{T\times K\times 2}$ - providing (u_{1jt}, v_{1jt})
- Stream 2 (RGB Cam 2): $D_2 \in R^{T\times K\times 2}$ - providing (u_{2jt}, v_{2jt})
- Stream 3 (Kinect RGB + Depth): $D_3 \in R^{T\times K\times 3}$ - providing $(u_{3jt}, v_{3jt}, d_{3jt})$

  where T represents the number of frames in the sequence, $K = 21$ is the number of hand keypoints, and the final dimension indicates the coordinate data (2D for RGB, 2D + depth for Kinect).

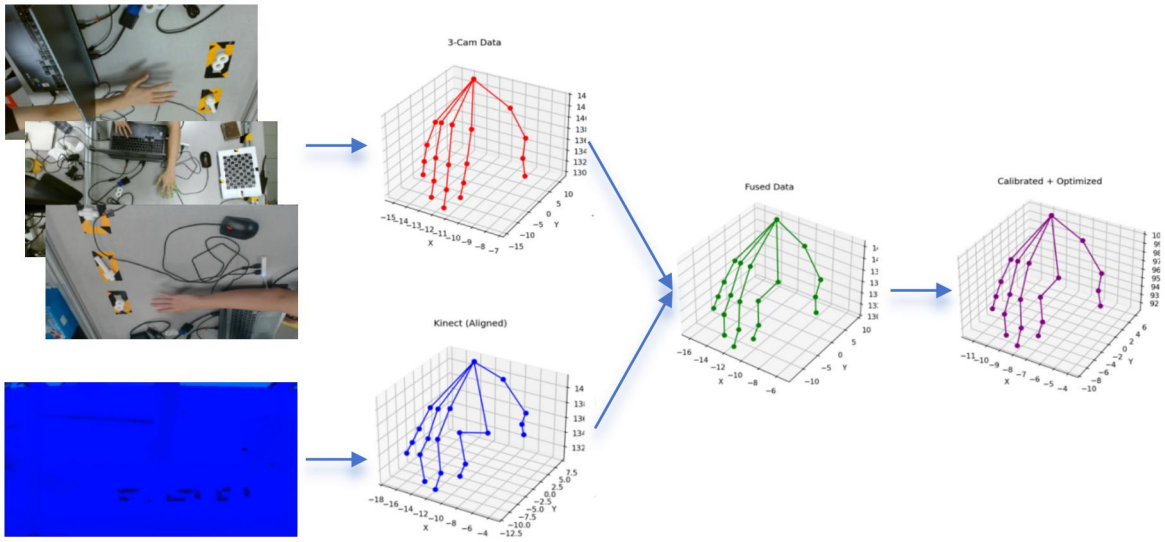


Figure 5 Flowchart of 3D Coordinate Generation via Fusion.

3D Coordinate Generation via Fusion. The synchronized, multi-modal data streams (D_1,D_2,D_3) constitute the input for the 3D reconstruction process detailed in Section 3.1.2 (Calibration and Data Fusion). Employing the pre-determined camera calibration parameters and the described fusion algorithm, the system integrates the multiple 2D views (u_{1jt}, v_{1jt}), (u_{2jt}, v_{2jt}), (u_{3jt}, v_{3jt}) and the direct depth measurement d_{3jt} for each keypoint j at each time step t. This fusion process resolves the 3D position $M_{jt} = [X_{jt}, Y_{jt}, Z_{jt}]^T$ of each keypoint in the common world coordinate system W. The output is a consolidated time series representing the dynamic 3D hand pose: $D_{3D} \in R^{T\times K\times 3}$.

3.1.4 SVD and temporal smooth algorithm

In Section 3.1.3, the DLT method constructs an overdetermined homogeneous linear system (Equation 3-11) to solve for projection matrix parameters. However, due to measurement noise and camera calibration errors, the solution of the system may become unstable [1]. To enhance robustness, this system introduces Singular Value Decomposition (SVD) to optimize the solution, ensuring the accuracy and stability of 3D reconstruction. $A = U\Sigma V^T$

For the homogeneous matrix equation $AL = 0$ in Equation (3-11), the least-squares solution is obtained via SVD of matrix A:

$$A = U\Sigma V^{T} X_{fixed}^{(i)} = \alpha X_{cam}^{(i)} + \beta X_{kinect_aligned}^{(i)}$$

where $U \in R^{6\times6}$ and $V \in R^{4\times4}$ are orthogonal matrices, and $\Sigma \in R^{6\times4}$ is a diagonal matrix with singular values arranged in descending order: $\sigma_1^2 > \sigma_2^2 > \ldots > \sigma_4^2 > 0$. The least-squares solution for the parameter vector L corresponds to the right singular vector associated with the smallest singular value:

$$L = V[:, -1]$$

Here, the last column of V (corresponding to σ_4) is selected. Due to the scale ambiguity inherent in homogeneous equations, LL is normalized by its homogeneous coordinate component:

$$L = \frac{L}{L_4}$$

Where L_4 is the homogeneous component.

The initial 3D point coordinates are then $X_{init} = (L_1, L_2, L_3)$.

In the DLT function, the SVD-based workflow proceeds as follows:

First is constructing the overdetermined matrix. Using the projection matrices P_1, P_2, P_3^c and corresponding 2D points $(u_1, v_1), (u_2, v_2), (u_3, v_3)$, build the matrix $A \in R^{6\times4}$. Each row is derived from the linear combination of projection equations. For example:

$$\begin{aligned} A[0,:] &= v_1 P_1^{(2)} - P_1^{(1)}, \\ A[1,:] &= P_1^{(0)} - v_1 P_1^{(2)}, \\ A[2,:] &= v_2 P_2^{(2)} - P_2^{(1)}, \\ &\vdots \\ A[5,:] &= P_3^{(0)} - v_3 P_3^{(2)} \end{aligned}$$

Where $P_i^{(k)}$ denotes the k row of the i projection matrix.

And then, is SVD decomposition and initial solution. Perform SVD on A, extract the last column of V^T (associated with the smallest singular value), and normalize the homogeneous coordinates to obtain the initial 3D point X_{init}.

Finally, is nonlinear refinement: Using X_{init} as the initial guess, apply the Levenberg-Marquardt algorithm to minimize the reprojection error:

$$\min_X = \sum_{i=1}^{3} \| \pi(P_i X) - (u_i, v_i) \|^2$$

where $\pi(\cdot)$ represents the homogeneous coordinate normalization. The refined 3D point $X_{refined}$ is obtained after optimization.

In addition, in multi-camera systems, the direct output 3D key point sequences may exhibit jitter due to rapid hand motion, object movement and per-frame detection noise[2]. To address this, the system uses temporal smoothing algorithm to post-process the 3D data, enhancing motion capture smoothness and continuity.

Let the 3D coordinate of the J joint at frame t be $X_j^{(t)} \in R^3$. The smoothed coordinate $\tilde{X}_j^{(t)}$ is computed via a sliding-window weighted average:

$$\tilde{X}_j^{(t)} = \frac{1}{W} \Sigma k = t - W + 1^t w_k X_j^{(k)}$$

where W is the window size (set to W=5 in this system), and w_k are weight coefficients (defaulting to uniform weights w_k =1). This algorithm acts as a low-pass filter, suppressing high-frequency noise while preserving the dominant motion frequencies. In Implementation, this paper also uses some method to improve the performance of temporal smoothing algorithm.

Dynamic Window Adjustment: For initial frames ($t < W$), the window size is adjusted $t + 1$ to avoid truncation.

Invalid value handling: If invalid detections (e.g., $X_j^{(i)} = [-1, -1, -1]$)

exist within the window, they are skipped, and the denominator is adjusted to reflect valid frames.

Real-time optimization: A recursive formula updates the window sum to minimize redundant computations:

$$S^{(t)} = S^{(t-1)} + X_j^{(t)} - X_j^{(t-W)} \quad \text{yielding } \tilde{X}_j^{(t)} = S^{(t)} / W .$$

3.2 State-aware YOLO Model Group

3.2.1 Introduction to YOLO for Process State Recognition

The You Only Look Once (YOLO) family is foundational to modern real-time object detection, employing a single-pass convolutional neural network (CNN) to predict bounding boxes and class probabilities directly from entire images. This approach contrasts with region proposal methods and enables the high inference speeds essential for real-time tasks. The YOLO architecture has evolved considerably through various versions, improving performance and design.

YOLOv8 represents a significant iteration in this development trajectory. Although not the absolute latest version available (as of April 13, 2025), YOLOv8 offers substantial advantages in terms of maturity, ease of use, and extensive community support, making it a robust choice for research and application. It provides a spectrum of model sizes (n, s, m, l, x) that balance accuracy, speed, and resource requirements.

In this study, we utilize the YOLOv8s model. This variant was chosen for its well-regarded balance between computational efficiency and detection accuracy, which aligns well with the scope of our dataset and processing capabilities. Therefore, YOLOv8s functions as the base model for our experimental framework. It is also pertinent to note that YOLO's holistic processing allows its principles to be adapted for classifying distinct procedural stages as unique classes, showcasing its versatility beyond standard object detection.

3.2.2 Challenges in Vision-Based Assembly State Identification

Monitoring intricate industrial assembly processes via computer vision presents unique challenges. The multiplicity of components, frequent transient occlusions during manipulation, and the potential for significant visual similarity between temporally adjacent assembly stages can confound traditional object detection or classification approaches. Exhaustive identification of every individual component often yields limited operational value compared to discerning the current phase of the overall assembly sequence. Furthermore, tasking a single monolithic classification model (such as a standard YOLO network) with discriminating between a large number of visually similar assembly states can lead to degraded performance and increased classification ambiguity, particularly between consecutive steps in the sequence.

3.2.3 Proposed Architecture: Multi-View Sliding YOLO Group

To address the aforementioned challenges, this report proposes a state-aware architecture employing a group of specialized YOLO models operating in a coordinated, sliding manner, leveraging inputs from the synchronized multi-camera system (detailed in Section 3.1). This Multi-View Sliding YOLO Group architecture is designed to enhance the accuracy and robustness of assembly state identification.

The core principle involves performing state detection inference independently on the synchronized image frames captured by each of the three calibrated cameras (two RGB, one Kinect RGB stream). For each time step t, each camera view $v \in \{1,2,3\}$ provides an input frame to the currently active YOLO model.

The model group consists of several distinct YOLO networks, $\{Y_1, Y_2, \ldots, Y_N\}$, where N is the total number of core assembly stages. Each individual model, Y_k, is specifically trained and optimized to recognize only

three sequential assembly states: the state immediately preceding the central state ($S_k - 1$, Previous), the central state itself (S_k, Current), and the state immediately following ($S_k + 1$, Subsequent). This focused responsibility leverages the inherent temporal continuity of the assembly process, where the current state is contextually bound by its immediate predecessor and successor.

Figure 6 Various stages of the experimental assembly

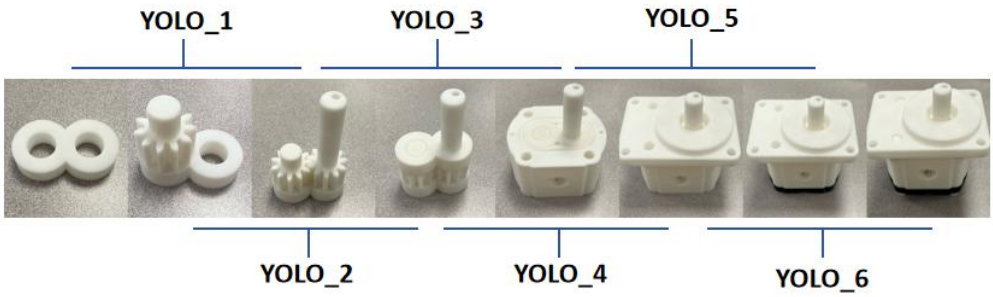


Figure 7 Dataset partitioning for the various units of the YOLO group.

3.2.4 Multi-View State Transition Logic

The system maintains an 'active' model, Y_k, corresponding to the currently anticipated assembly stage S_k. The transition between active models within the group is governed by a robust multi-view consensus mechanism combined with temporal stability criteria:

Multi-View Inference: At each synchronized time step t, the active model Y_k processes the input frames from all three camera views, yielding potentially three independent state classifications:$D_{1,t}, D_{2,t}, D_{3,t}$.

Consensus State Determination: A unified 'Consensus Detected State', D_t, is established only if the classification results agree across all three views:$D_t = S\ if\ D_{1,t} = D_{2,t} = D_{3,t}$. If there is any disagreement between the views, a consensus state is not achieved for that time step, and no transition evaluation

occurs.

Temporal Stability Filter: A state transition is initiated only if a non-central consensus state ($D_t = S_{k-1}$ or $D_t = S_{k+1}$) is consistently detected for a predefined duration, τ. This temporal filtering mitigates spurious transitions caused by momentary detection fluctuations. Based on the system's frame rate of 30 fps (as per Section 3.1.3), we set $\tau = 10$ consecutive frames, corresponding to approximately 0.33 seconds.

Model Group Sliding: If the consensus state is consistently for frames, the system transitions backward, activating the preceding model . This transition logic corresponds to the progression of the assembly sequence in our defined workflow. If the consensus state is consistently for frames, the system transitions forward, activating the subsequent model . This transition logic corresponds to the disassembly sequence or reversal of steps. If the consensus states, or if no consensus is reached, or if a non-central consensus state does not persist for frames, the current model remains active. Appropriate boundary conditions are applied for the first (Y_1) and last (Y_n) models in the group.

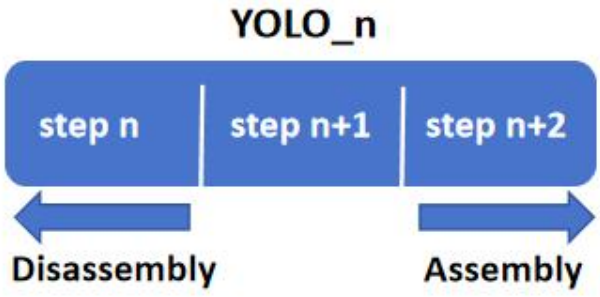


Figure 8 YOLO group unit switching schematic diagram

3.3 Similarity Evaluation Model

3.3.1 Introduction and Motivation

Subsequent to the acquisition and preprocessing of hand motion data into time series of 3D coordinates (shape: (T, 63)), a robust mechanism is required for the quantitative evaluation of these motion segments. This evaluation is crucial

for tasks such as verifying procedural adherence in assembly workflows, assessing skill acquisition, or identifying deviations from standard operating procedures. However, developing distinct classifiers for every possible granular action within complex procedures necessitates prohibitively large and meticulously annotated datasets, rendering exhaustive action-by-action classification often impractical due to data acquisition costs and scalability limitations. Therefore, rather than attempting individual action recognition, this project adopts a more scalable and data-efficient paradigm: similarity-based evaluation. This approach focuses on quantifying the resemblance between a captured motion sequence and a corresponding reference or standard motion sequence. Crucially, this method only requires establishing representative templates for correct execution, rather than modeling all possible action classes and their variations. To this end, we propose a data-driven similarity evaluation model specifically designed to perform this comparative assessment.

The core of this evaluation framework is a deep learning model based on the Transformer architecture. Transformers, originally introduced for natural language processing,[20] have demonstrated remarkable efficacy in modeling sequential data due to their self-attention mechanism. Specifically, self-attention operates by projecting the input sequence representations (of dimension (D_{model})) into Query (Q), Key (K), and Value (V) matrices using learned linear transformations.The attention output is then computed using scaled dot-product attention according to the seminal formulation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

In this equation, the matrix product (QK^T) computes the dot products between all query vectors and key vectors, yielding raw attention scores that quantify pairwise element compatibility. These scores are scaled by the inverse square root of the key dimensionality, ($\sqrt{d_k}$), a factor crucial for stabilizing gradients during training, particularly for larger dimensions. The softmax

function subsequently normalizes these scaled scores row-wise, producing probability distributions representing the attention weights. Finally, these weights are multiplied by the Value matrix V to yield an output representation where each position is a weighted sum of value vectors from the entire sequence, effectively aggregating global context based on learned relevance. This mechanism adeptly captures long-range temporal dependencies without the constraints imposed by the sequential processing inherent in recurrent architectures. Furthermore, Transformers typically employ Multi-Head Attention, executing the scaled dot-product attention function multiple times in parallel with distinct, learned linear projections for Q, K, and V. This allows the model to jointly attend to information from different representation subspaces at various positions. Leveraging this capability, our model aims to learn discriminative representations of hand motion dynamics to accurately predict the similarity between pairs of action sequences.

3.3.2 Model Architecture

The proposed similarity evaluation model employs a Siamese network structure built upon Transformer encoders, directly corresponding to the Transformer-based Similarity Model implementation presented previously. This architecture is specifically designed to process pairs of input motion sequences and output a scalar value indicating their similarity.

1. **Input Layer:** The model accepts pairs of motion sequences, $(\mathrm{M_A}, \mathrm{M_B})$, as input. Each sequence $\mathrm{M} \in \mathbb{R}^{\mathrm{T}\times\mathrm{D}}$,where T is the sequence length (potentially standardized via padding or truncation) and $\mathrm{D} = 63$ represents the dimensionality of the normalized, centered 3D coordinates for the 21 hand keypoints (excluding the reference wrist point).

2. **Shared Feature Extractor (Transformer-based Similarity Model):** A single, weight-shared Transformer-based feature extractor processes both input

sequences and . Sharing weights ensures that both sequences are mapped into the same representation space, facilitating meaningful comparison. This extractor comprises:

Input Embedding: A linear layer projects the input dimension D to the model's hidden dimension model_dim. An appropriate scaling factor ($\sqrt{model\_dim}$) is applied to post-embedding.

Positional Encoding: Standard sinusoidal positional encodings are added to the embeddings to provide the model with information about the temporal order of the frames.

Transformer Encoder Stack: A stack of L identical Transformer encoder layers processes the position-aware embeddings. Each layer contains a multi-head self-attention (MHSA) mechanism followed by a position-wise feed-forward network (FFN). MHSA allows the model to jointly attend to information from different representation subspaces at various positions, capturing complex spatio-temporal dependencies, while the FFN applies a non-linear transformation. Layer normalization and dropout are applied within each layer for stabilization and regularization. The output of the feature extractor for a sequence M is a feature sequence $F \in R^{T \times model\_dim}$.

3. **Temporal Pooling:** To obtain a fixed-size representation of the entire sequence, irrespective of its length T, a temporal pooling operation is applied to the output feature sequence F. Mean pooling aggregates the features across the time dimension, resulting in a single feature vector $f \in R^{T \times model\_dim}$ for each input sequence (f_A for M_A, f_B for M_B).

4. Similarity Feature Aggregation: The pooled feature vectors f_A and f_B are compared and combined to create a comprehensive feature vector encoding their relationship. Following the implementation strategy, this involves concatenating the individual feature vectors, their element-wise absolute difference, and their element-wise product:

$$v_{sim} = [f_A; f_B; |f_A - f_B|; f_A \odot f_B]$$

where ; denotes concatenation and $\odot$ represents element-wise multiplication. The resulting vector $v_{sim} \in \mathbb{R}^{4\times model\_dim}$.

5. Normalization Layer: The aggregated similarity feature vector f_agg is then passed through a normalization layer (e.g., Batch Normalization or Layer Normalization). This step helps stabilize training and can improve model generalization by normalizing the inputs to the subsequent classifier layer. Let the normalized vector be f_norm

$$f_{norm} = Normalize(f_{agg})$$

6. **MLP Classifier:** The normalized aggregated similarity feature vector f_{norm} is passed through a Multi-Layer Perceptron (MLP) classifier. This typically consists of one or more linear layers interspersed with non-linear activation functions (ReLU) and dropout layers. The final linear layer reduces the dimension to a single scalar output (logit).

7. **Output:** The model outputs a single logit value. This logit represents the model's prediction of the similarity between the input sequences M_A and M_B. Applying a sigmoid function to this logit would yield a probability score between 0 and 1.

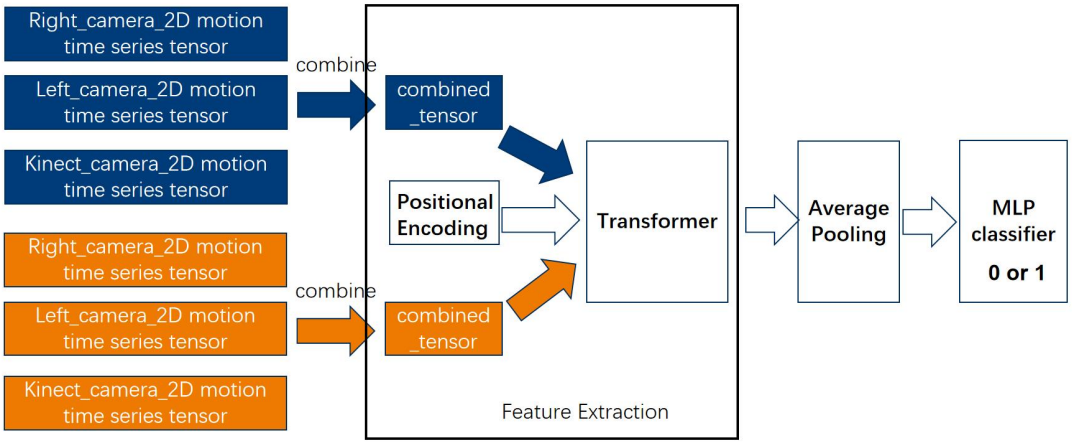


Figure 9 Action similarity model architecture.

3.3.3 Dataset and Training Protocol

1. **Dataset:** The model is trained using data derived from the First-Person Hand Action Benchmark. This benchmark provides diverse examples of manipulative hand actions, suitable for learning representative features of functional hand movements relevant to assembly tasks. The 3D time series data, preprocessed according to the methods described in Section 3.1.3, forms the basis for training sequence pairs.

2. **Pairwise Training Strategy:** A pairwise learning approach is adopted to train the similarity model. Training examples are constructed as pairs of sequences (MA,MB) with associated binary labels:

   Positive Pairs (Label = 1): M_A and M_B are two distinct instances randomly sampled from the same action category within the benchmark dataset. This teaches the model to recognize intra-class variations as similar.

   Negative Pairs (Label = 0): M_A and M_B are randomly sampled from different action categories. This teaches the model to distinguish between functionally distinct motions. Care is taken during pair generation to ensure balanced sampling across classes and potentially employ strategies like hard negative mining if necessary.

3. **Training Objective:** The network is trained end-to-end by minimizing a

binary cross-entropy objective function, specifically utilizing BCEWithLogitsLoss, which combines a sigmoid activation with the binary cross-entropy calculation for numerical stability. The loss compares the model's output logits against the ground truth pair labels (0 or 1). Optimization is performed using standard stochastic gradient descent methods, such as the Adam optimizer [Cite Kingma & Ba, 2014], with appropriate learning rate scheduling and weight decay for regularization.

4. Experiments and Results

This section details the experimental validation of the proposed methodologies for assembly state recognition and hand motion similarity evaluation. We conducted two primary experiments: the first evaluates the performance of the state-aware Multi-View Sliding YOLO Group against a baseline single YOLO model for process state identification, and the second assesses the effectiveness of the Transformer-based similarity model in quantifying hand motion resemblance.

4.1 Experiment 1

4.1.1 YOLO Model Training

To rigorously evaluate the influence of model architecture on recognition accuracy, identical datasets were employed for training both the Single-YOLO and YOLO Group models. This dataset was custom-built, captured utilizing a Kinect V2 camera. For data distribution, each distinct assembly stage comprised 150 images. Within the YOLO Group configuration, each constituent sub-module was trained using 450 images. In contrast, the standalone Single-YOLO model was trained using a total of 750 images. The training hyperparameters were configured as follows:

Parameters	Value
Epochs	200
Input Image Size	640x640 pixels
Training Environment	NVIDIA RTX A4000
Batch Size	64
Base Model	yolov8s.

Table 5 The parameters of YOLO model training

The resulting performance metrics are presented subsequently. Given that the

primary objective of this experiment is to assess the efficacy of the YOLO models in determining assembly states, Precision and Recall are considered the most pertinent metrics as they directly reflect the accuracy of state classification. In contrast, mAP50 and mAP50-95 primarily measure the localization accuracy, specifically the overlap between predicted and ground truth bounding boxes. While important, localization accuracy is deemed secondary to classification correctness for this study's objective. Consequently, Precision and Recall will be utilized to evaluate and compare the performance of the single-YOLO and group-YOLO models in accurately identifying assembly states.

4.1.2 Model Training and Convergence

The training dynamics for both the Single-YOLO and YOLO Group architectures were initially assessed. Performance was monitored using key metrics including classification loss (cls_loss), bounding box regression loss (box_loss), distribution focal loss (dfl_loss), alongside bounding box precision (P(B)) and recall (R(B)). As illustrated by the learning curves presented in table ?, both model configurations demonstrated stable convergence over the training epochs. Crucially, the close correspondence observed between the training and validation metric trajectories indicated that significant overfitting was successfully mitigated, confirming that the models generalized effectively beyond the training data. Given that the F1 curve illustrates the variation of the F1 score as a function of the confidence threshold, the observed peak occurring at a threshold value of approximately 0.9 signifies the point where an optimal balance between Precision and Recall is achieved. Consequently, this theoretically indicates that setting the confidence threshold to circa 0.9 should maximize the model's recognition performance, particularly when effectiveness is evaluated using the F1 metric, which represents the harmonic mean of Precision and Recall.

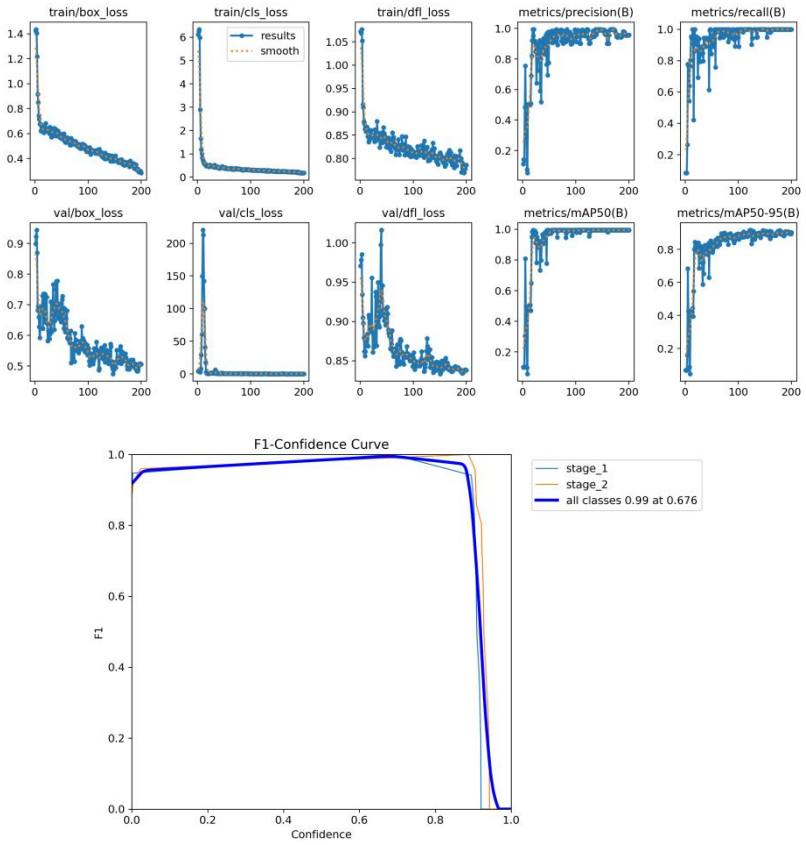


Figure 10 Training results of yolo_group_cell_1

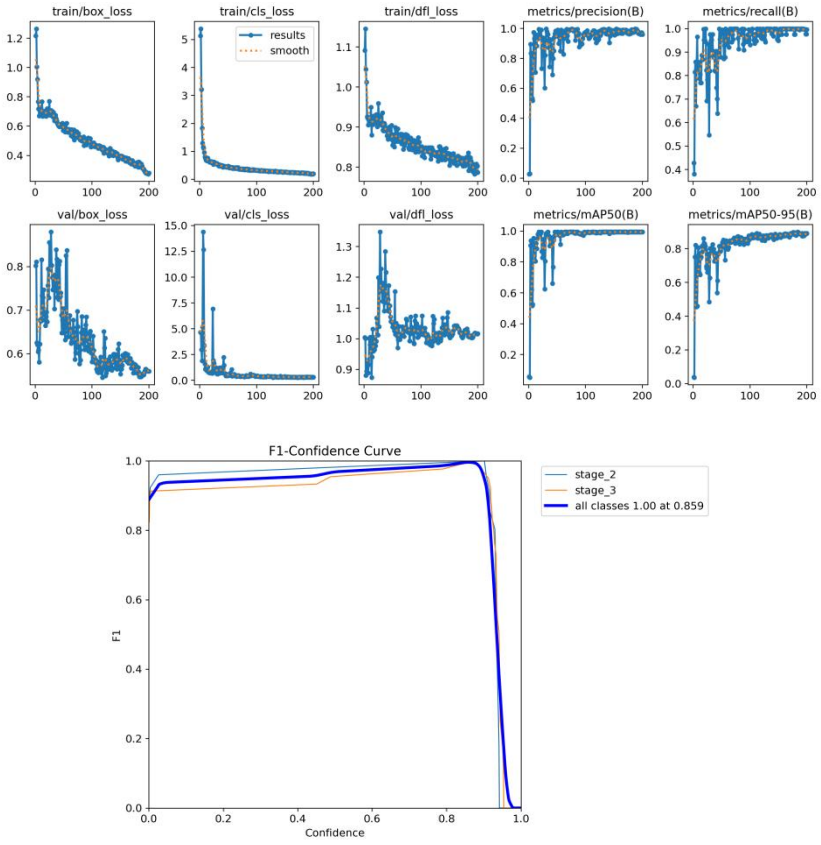
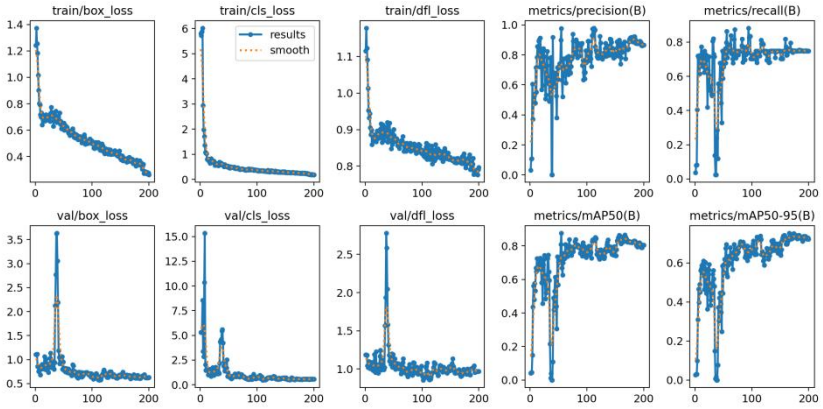


Figure 11 Training results of yolo_group_cell_2



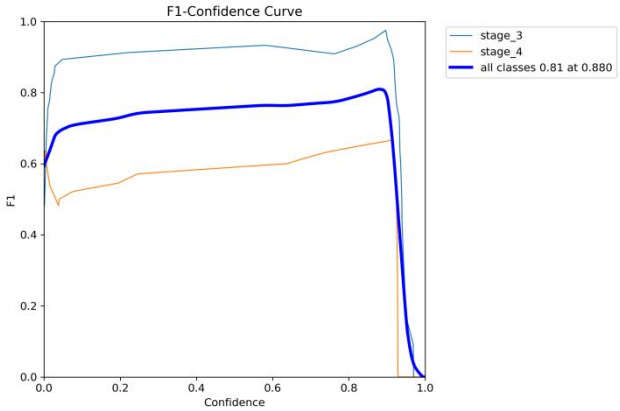
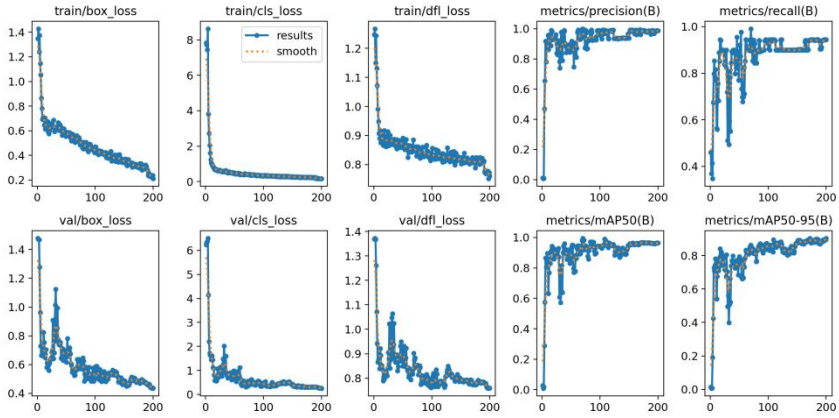


Figure 12 Training results of yolo_group_cell_3



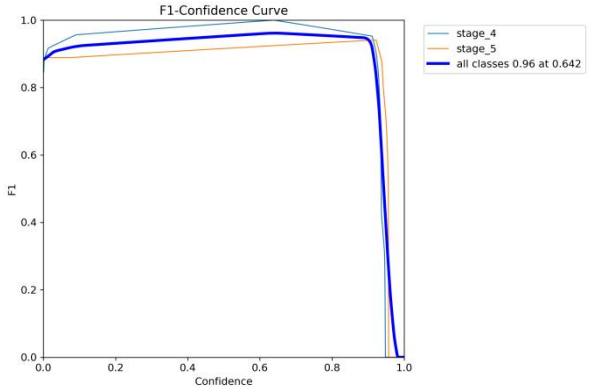


Figure 13 Training results of yolo_group_cell_4

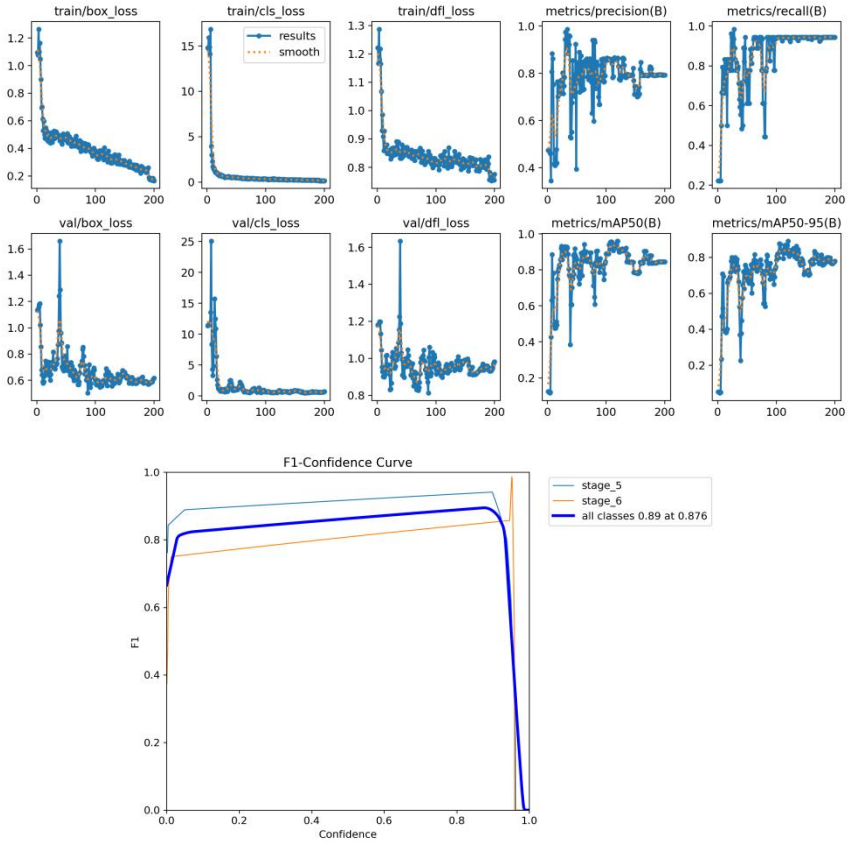


Figure 14 Training results of yolo_group_cell_5

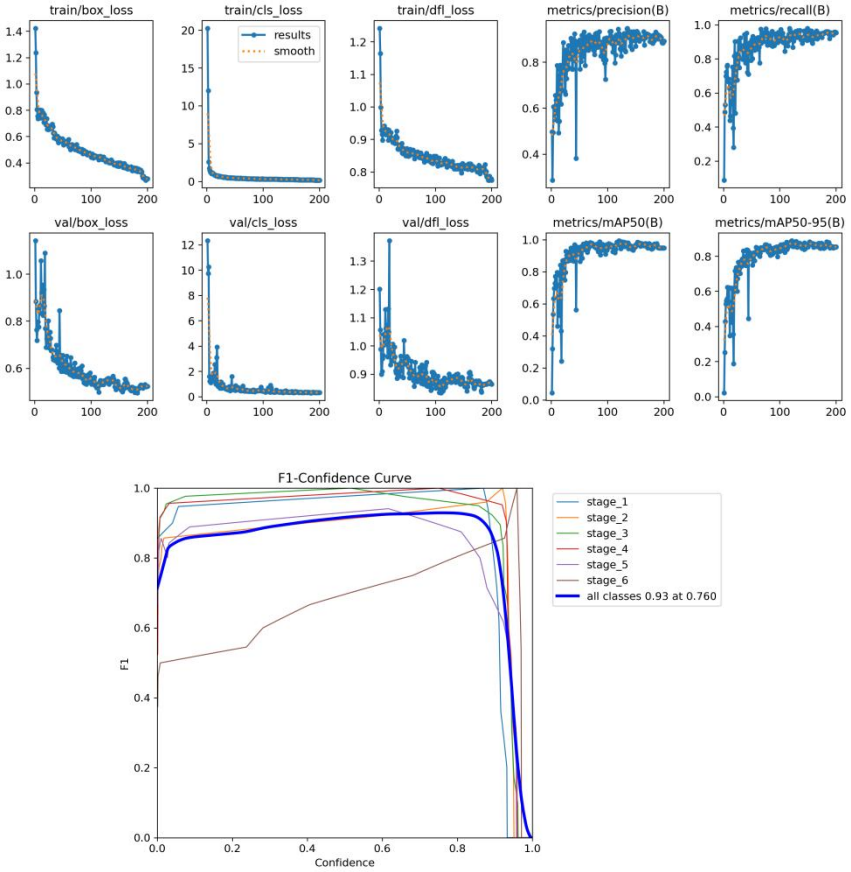


Figure 15 Training results of yolo_single

Subsequent evaluation based on test dataset focused on the primary objective: assembly state classification accuracy. Model performance was quantified using Precision (P) and Recall (R), with detailed per-stage results summarized in Figure 17 and visualized in Figure 18.

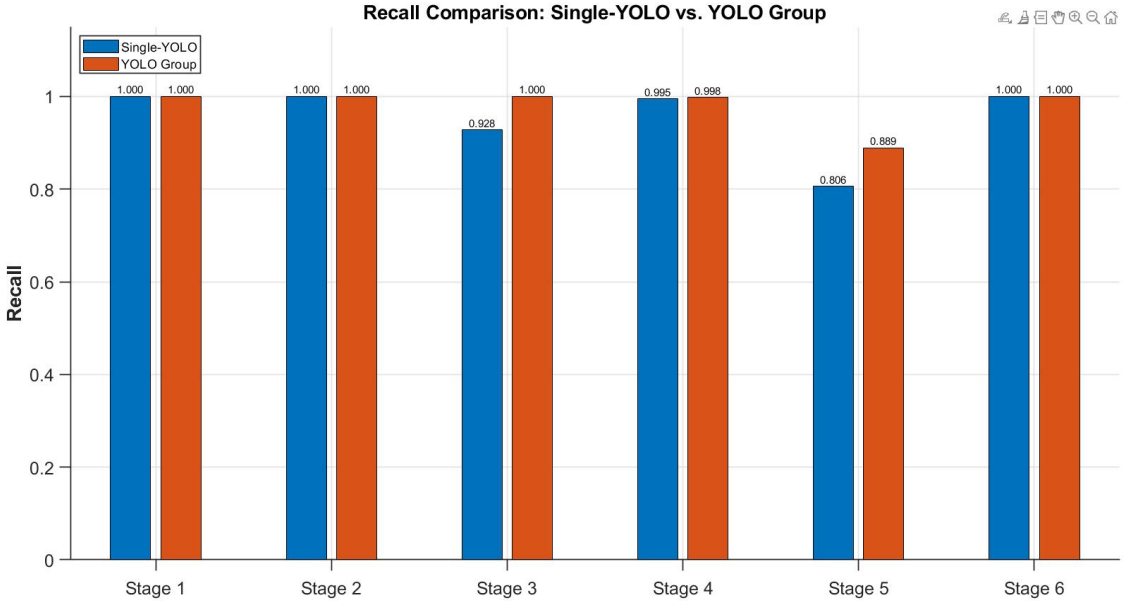


Figure 17 Recall comparison: Single YOLO vs Yolo Group

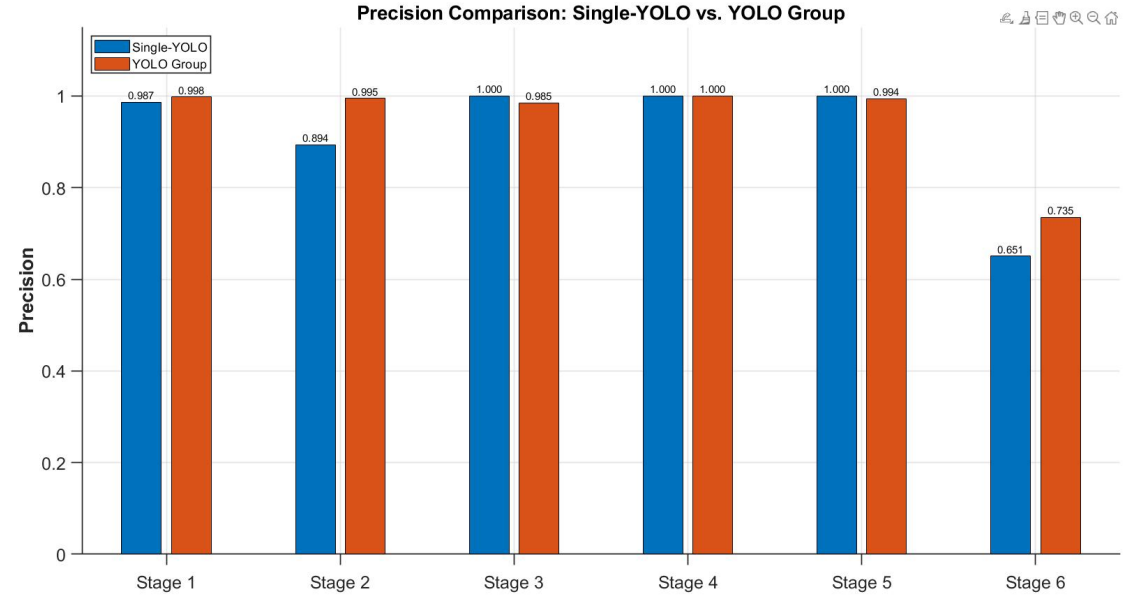


Figure 18 Precision comparison: Single YOLO vs Yolo Group

The integrated Single-YOLO model achieved respectable overall performance (P=0.922, R=0.955) but exhibited notable weaknesses, including reduced Precision for Stage 2 (0.894) and Stage 6 (0.651), and lower Recall for Stage 5 (0.806).

In contrast, the YOLO Group approach, utilizing specialized models for consecutive stage pairs, generally demonstrated strong Precision and Recall

within their respective domains. A comparative analysis reveals that the Group configuration often matched or outperformed the Single-YOLO model on a per-stage basis. More importantly, the Group models significantly mitigated the Single-YOLO's weaknesses, substantially improving Precision for Stage 2 (e.g., 0.995 vs 0.894) and Stage 6 (0.735 vs 0.651), and markedly boosting Recall for Stage 5 (0.889 vs 0.806). An exception was the lower performance observed for Group 3-4 on Stage 4.

These results suggest that specializing models on smaller, related subsets of assembly stages enhance discriminative capability, particularly between adjacent states. While mAP scores (localization accuracy) were generally high for both methods, the tangible improvements in Precision and Recall affirm the effectiveness of the YOLO Group architecture for the primary objective of accurate state determination, justifying its increased complexity despite the noted anomaly in Group 3-4.

4.2 Experimental 2

4.2.1 Dataset Preparation and Pre-processing for Similarity Learning

The dataset utilized for the action time-series similarity prediction task is the "First-Person Hand Action Benchmark with RGB-D Videos and 3D Hand Pose Annotations". Specifically, we leveraged the Hand_pose_annotation_v1 component, which provides annotated 3D hand pose kinematic data over time. This subset encompasses 1331 distinct action sequences across 45 action classes, with each sequence represented as a time series of 3D coordinates for 21 hand joints (resulting in 63 features per frame) captured at varying frame rates.

To facilitate the training and evaluation of a similarity judgment model, a supervised learning approach was adopted by generating paired samples. A total of 2000 pairs were randomly constructed: 1000 positive pairs, where both sequences in the pair originated from the same action class, and 1000 negative

pairs, where the sequences belonged to different action classes. This pool of 2000 generated pairs was subsequently partitioned into a training set (80%, comprising 1600 pairs: 800 positive and 800 negative) and a test set (20%, comprising 400 pairs: 200 positive and 200 negative) to evaluate model generalization.

A critical pre-processing step involved normalizing the temporal dimension of the sequences to enable consistent input for the model. Initially, a fixed length of 100 frames was targeted. Sequences longer than 100 frames were truncated by discarding the excess frames from the end, while sequences shorter than 100 frames were zero-padded to reach the target length. For training, the data was organized into batches with a batch size of 64. Each training instance thus consisted of two input tensors, each with the shape (64, 100, 63), representing the pair of sequences, and a corresponding binary label (1 for a positive pair, 0 for a negative pair). The same normalization procedure (truncation/padding to 100 frames) was applied to the sequences within the test set pairs prior to evaluation.

Recognizing that significant truncation or extensive zero-padding might introduce artifacts or dilute informative features, a second experimental setup was prepared. In this refined approach, sequences (from the original 1331) with frame counts less than 60 or greater than 120 were excluded before the pair generation and train/test split process described above. This filtering step aimed to create a more temporally homogeneous dataset for both training and testing.

Experiments were conducted using both the initially processed dataset (all sequences included before pairing/splitting) and the filtered dataset, resulting in two distinct sets of loss and accuracy curves for comparative analysis based on their respective train and test set performances.

4.2.2 Model Performance Evaluation and Impact of Sequence Length Handling

1. Baseline Performance Evaluation on Original Data

The performance of the proposed Transformer-based model was initially evaluated using the original, unprocessed dataset under the specified hyperparameter configuration. The primary metrics assessed were training and testing accuracy, alongside the corresponding loss values, to establish a baseline understanding of the model's capability in discerning action similarity.

The Binary Cross-Entropy with Logits loss function was selected for model training. This loss function is particularly suitable for binary classification tasks, such as the action similarity judgment problem addressed here, where the model outputs a raw score (logit) indicating the likelihood of similarity. BCEWithLogitsLoss combines a Sigmoid activation and the standard Binary Cross-Entropy (BCE) loss into a single, numerically stable computation. For a single output logit (x) and a binary target label(0 or 1), the BCE loss is defined as:

$$L(x, y) = -\left[y \cdot \log\left(\sigma(x)\right) + (1 - y) \cdot \log\left(1 - \sigma(x)\right)\right]$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. The BCEWithLogitsLoss implementation provides a more numerically stable equivalent calculation, often represented as:$L_{BCEWithLogits}(x, y) = x(1 - y) + \log(1 + e^{-x})$

As illustrated [], the evaluation yielded distinct performance characteristics on the training and testing datasets. The training accuracy converged and stabilized at approximately 87%, with the corresponding training loss settling around 0.3. When evaluated on the unseen test set, the model achieved an accuracy of approximately 80%, with an associated test loss of approximately 0.45.

These baseline results indicate that the model possesses a discernible capability for identifying similarities between skeletal action sequences using the raw data. However, the performance gap between training and testing, and the absolute accuracy achieved on the test set (≈80%), suggest that while the model has learned relevant features, its current level of precision and generalization may be insufficient for deployment in demanding industrial applications requiring

high-reliability action judgment or classification.

2. Evaluation on Frame-Filtered Data: Investigating Sequence Length Effects

Observing the baseline performance, it was hypothesized that a significant factor limiting accuracy for time-series analysis could be the common preprocessing step of forcing variable-length sequences into a fixed-length representation (e.g., 100 frames in the initial setup) via truncation or padding. Such operations can potentially discard crucial temporal information (truncation) or introduce non-informative padding tokens (padding), potentially hindering the model's ability to learn fine-grained temporal dependencies characteristic of human actions.

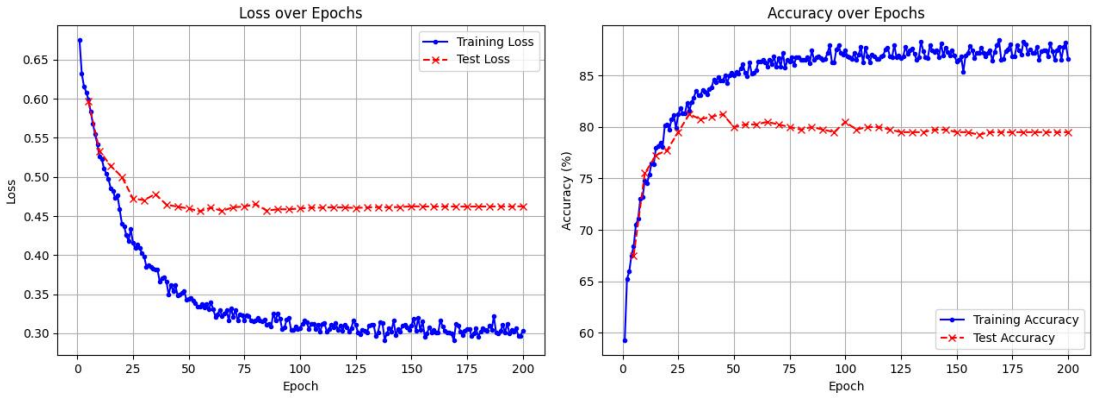


Figure 19 Epoch-wise training/testing metrics

To investigate this hypothesis, a second experiment was conducted. A subset of the dataset was curated by selecting only action sequences with original frame counts falling within the range of 70 to 130 frames. This filtering aimed to create a dataset with more homogeneous sequence lengths, thus minimizing the extent of truncation or padding required and preserving more of the original action dynamics. The Transformer-based similarity model was subsequently retrained on this frame-filtered dataset using the identical hyperparameter configuration and

BCEWithLogitsLoss function to ensure a direct comparison with the baseline.

The training and testing curves for this experiment are presented Figure 2. The results demonstrated a notable improvement compared to the baseline. On the filtered training set, accuracy exceeded 90%, while the training loss decreased to approximately 0.27. Critically, performance on the corresponding filtered test set also improved significantly, reaching an accuracy of approximately 84% with a test loss reduced to around 0.40.

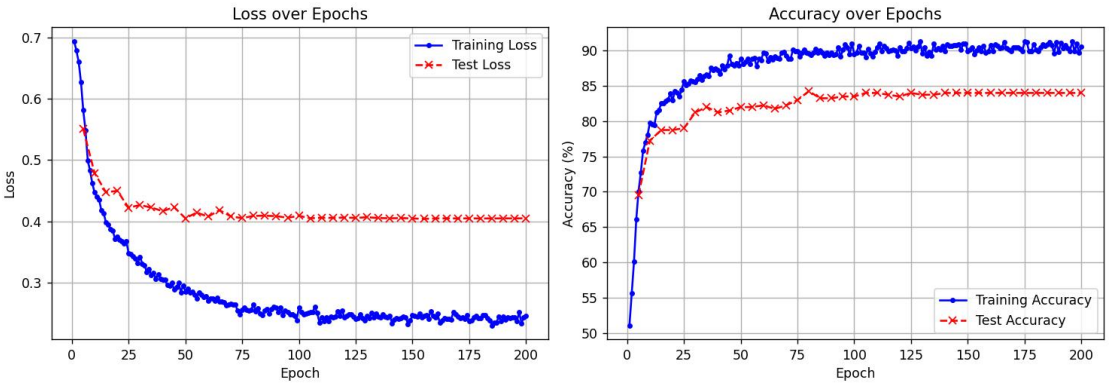


Figure 20 Epoch-wise training/testing metrics on the modified dataset

5. User Interface Development using Qt and PyQt

5.1 Introduction to Qt

Qt is a comprehensive, cross-platform application development framework widely utilized for creating graphical user interfaces (GUIs) as well as non-GUI applications. Written primarily in C++, Qt provides a rich set of libraries and tools that enable developers to build high-performance applications deployable across desktop, mobile, and embedded operating systems from a single codebase. Key features include an extensive collection of UI components, robust internationalization support, database and network connectivity modules, and a powerful signal-slot mechanism for inter-object communication, facilitating the development of complex, event-driven applications [24].

5.2 Introduction to PyQt

PyQt serves as a set of Python bindings for the Qt framework, developed by Riverbank Computing. It allows developers to harness the full potential of Qt within the Python programming environment. By integrating Qt's mature GUI capabilities and extensive libraries with Python's simplicity, extensive standard library, and rich ecosystem of third-party packages, PyQt enables rapid development of sophisticated cross-platform applications. It provides Python developers with access to the complete Qt API, making it a popular choice for creating desktop applications with complex user interfaces in Python.

5.3 Project-Specific PyQt Interface Implementation

In this project, a supervisory graphical user interface (GUI) was developed using the PyQt framework to facilitate operator guidance, process monitoring, and validation for the target assembly task, specifically focusing on hand detection and assembly verification. The interface serves as the primary interaction point for the operator and orchestrates the data acquisition and

analysis workflow.

The operational logic implemented within the PyQt interface provides real-time instructions to guide the operator through the assembly sequence. Initially, the interface prompts the operator, for instance, "Please proceed to Area 1". Upon detecting the operator's hand entering this predefined operational zone (monitored via connected sensors/cameras), the interface automatically updates its status to indicate "Recording initiated". This event triggers the activation of three cameras, which commence capturing synchronized RGB video streams and associated depth map data.

The system continuously monitors the assembly progress. When the YOLO object detection model identifies a predetermined change in the assembly state—signifying the successful completion of a task step (e.g., joining a part)—the interface acknowledges this. It signals the end of the current action's data recording phase and prompts the operator with the next instruction, such as "Action recording complete, please proceed to Area 2". This cycle of guidance, action execution, monitoring, and state-change detection repeats iteratively until the entire assembly sequence is concluded.

Upon completion of the full assembly process, the data captured by the three cameras is processed to generate five distinct time-series matrices, representing quantitative descriptions of the performed assembly actions (potentially derived from hand keypoint trajectories or other relevant features). These matrices are subsequently input into the previously described action similarity model. This model compares the extracted temporal data against normative data corresponding to standard, correctly executed procedures.

The graphical user interface (GUI) implemented in PyQt framework incorporates three synchronized display panels corresponding to the three camera viewpoints capturing the assembly process. The control panel contains three functional buttons: "Start", "Process", and "Load Videos". Upon activating the "Load Videos" button, the system initializes by displaying pre-recorded standard

assembly actions across all three display panels to serve as visual references for the operator.

When the "Start" button is engaged, the system transitions into active monitoring mode, simultaneously initiating real-time data acquisition from the camera array and converting the button label to "End". The program terminates automatically upon successful completion of the assembly process verified by the dual validation system, though manual termination via the "End" button remains available for operator override.

The "Process" button becomes operable post-termination, triggering the quantitative evaluation module. This module employs the action similarity model to generate normalized validation scores (0-1 scale) for each of the five constituent assembly actions, while the YOLO-based stage identification component verifies sequential progression through the six predefined assembly stages. The integrated validation system ensures that only processes satisfying both criteria - achieving unitary similarity scores across all action segments and demonstrating correct stage transitions from Stage 1 to Stage 6 - receive quality approval. Any deviations detected in either action execution patterns or stage progression sequence are automatically flagged through the supervisory interface, providing precise feedback for quality control interventions.

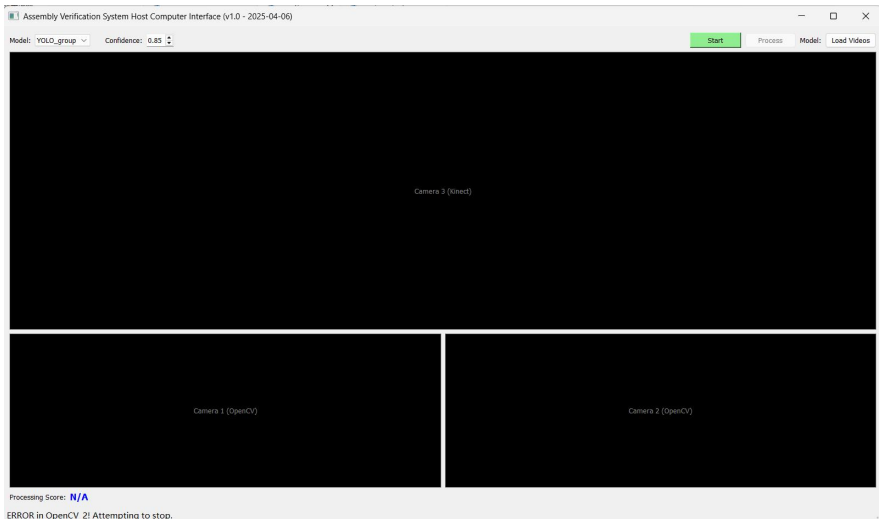


Figure 21 Qt GUI display screen/page

6. Conclusion

This research addressed the persistent challenge of ensuring procedural accuracy and quality in manual assembly operations within flexible manufacturing environments, where conventional oversight methods often lack objectivity and robustness. We presented an integrated, vision-based framework designed for automated assembly process verification, synergistically combining multi-sensor data acquisition, state-aware process stage identification, and quantitative hand motion similarity assessment.

The principal contributions and findings of this work are threefold:

1. **Optimized Multi-Sensor System for 3D Hand Tracking:** We successfully designed, implemented, and calibrated a hybrid three-camera framework integrating RGB-D (Kinect V2) and stereo RGB (Logitech C920) sensors. By employing robust calibration techniques and a 3D reconstruction algorithm enhanced with Singular Value Decomposition (SVD) for stability and optimized via least-squares refinement, the system effectively overcomes the visual field limitations inherent in single-view depth sensing and achieves high-fidelity capture of 3D hand keypoint kinematics.

2. **Enhanced Assembly State Recognition via YOLO Group:** Recognizing the limitations of monolithic models in distinguishing visually similar assembly stages, we proposed and validated a novel Multi-View Sliding YOLO Group architecture. Experimental results conclusively demonstrated that this approach, which utilizes specialized YOLOv8s models focused on sequential stage triplets, significantly improves state classification accuracy, particularly in discriminating between adjacent assembly steps, compared to a standard single YOLO model trained on all stages. This confirms the capability of YOLO architectures for state determination and the benefit of the proposed sliding strategy.

3. **Scalable Motion Verification using Transformer-based Similarity:** Addressing the prohibitive cost and lack of real-time feasibility associated with creating exhaustive datasets for every assembly action, we developed a

Transformer-based Siamese network for action similarity evaluation. This approach allows for efficient verification by comparing performed actions against a single recorded standard template. The model demonstrated strong performance, achieving approximately 84% accuracy on frame-filtered data in distinguishing between compliant and non-compliant motion sequences. This confirms the viability of similarity assessment for process verification, offering significant advantages in scalability and deployment effort. Furthermore, the results indicated potential for future optimization, such as developing strategies to better handle variations in action duration like frame length normalization or adaptive models.

In summary, this study contributes a comprehensive, multi-faceted system for automated assembly verification. By integrating advanced computer vision techniques for data acquisition, state recognition, and motion analysis, this framework provides a robust solution for enhancing quality assurance, facilitating operator guidance, and improving process control in human-centric manufacturing settings. The open-sourced nature of the developed code further encourages future research and application in related domains. Future work could focus on refining the temporal modeling within the similarity network and extending the system's adaptability to more diverse assembly tasks and environments.

7. Future Plan and Current Limitation

7.1 Bimanual Hand Tracking and Analysis

A notable limitation of the current implementation is its focus on single-hand analysis; both the 3D keypoint reconstruction and the subsequent motion similarity evaluation utilize data pertaining to only one hand. However, numerous industrial assembly operations inherently involve bimanual coordination and interaction. Consequently, a significant avenue for future research is the extension of the methodology to incorporate keypoints from both hands. This would necessitate developing robust mechanisms for simultaneous two-hand 3D tracking and establishing methods for constructing and analyzing bimanual 3D motion time-series matrices for similarity assessment, thereby enabling a more holistic representation and verification of operator actions in complex tasks.

7.2 Optimization of Fixed Parameters and Thresholds

The current system employs several parameters and thresholds set empirically or maintained at fixed values throughout the experiments. These include the depth fusion weight (lambda_k = 0.2), the temporal smoothing window size (W=5), the YOLO Group state transition stability threshold (N_stable = 10 frames), and the MediaPipe detection confidence threshold (0.9). While functional within the reported experimental context, these predetermined values may not represent optimal settings across diverse tasks, operator characteristics (e.g., execution speed), or varying environmental conditions. Future work should therefore investigate methods for the quantitative optimization or adaptive determination of these parameters. Techniques such as sensitivity analysis, automated hyperparameter optimization (e.g., Bayesian optimization, grid search or optuna), or developing adaptive algorithms that dynamically adjust these values based on real-time context could significantly enhance the system's overall robustness, accuracy, and adaptability.

8. References

[1] D. Moutinho, L. F. Rocha, C. M. Costa, L. F. Teixeira, and G. Veiga, "Deep learning-based human action recognition to leverage context awareness in collaborative assembly," Robotics and Computer-Integrated Manufacturing, vol. 80, Art. no. 102449, 2023, doi: 10.1016/j.rcim.2022.102449.

[2] K. Zheng and P. Liu, "Visual control systems in Industry 4.0: Enhancing assembly quality through machine learning," Computers in Industry, vol. 126, Art. no. 103485, Mar. 2021, doi: 10.1016/j.compind.2021.103485.

[3] Ö. Ay and E. Emel, "Real-Time Assembly Task Validation Using Deep Learning-Based Object Detection and Operator's Hand-Joints Trajectory Classification," IEEE Access, vol. 13, pp. 57009-57029, 2025, doi: 10.1109/ACCESS.2025.3554263.

[4] J.-J. Hu, C.-N. Huang, H.-W. Wang, P.-H. Shieh, and J.-S. Hu, "Safety-based human-robot collaboration in cellular manufacturing: A case study of power protector assembly," in Proc. Int. Conf. Adv. Robot. Intell. Syst. (ARIS), Tainan, Taiwan, 2013, pp. 28–31, doi: 10.1109/ARIS.2013.6573529.

[5] Md. Al-Amin et al., "Action Recognition in Manufacturing Assembly Using Multimodal Sensor Fusion," Procedia Manufacturing, vol. 39, pp. 158–167, 2019, doi: 10.1016/j.promfg.2020.01.288.

[6] Z. Wang and J. Yan, "Deep learning based assembly process action recognition and progress prediction facing human-centric intelligent manufacturing," Computers & Industrial Engineering, vol. 196, Art. no. 110527, 2024, doi: 10.1016/j.cie.2024.110527.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Las Vegas, NV, USA, Jun. 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[8] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Honolulu, HI, USA, Jul. 2017, pp. 6517–6525, doi: 10.1109/CVPR.2017.690.

[9] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv preprint arXiv:1804.02767, Apr. 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1804.02767.

[10] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," arXiv preprint arXiv:2004.10934, Apr. 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2004.10934.

[11] R. Khanam and M. Hussain, "What is YOLOv5: A deep look into the internal features of the popular object detector," arXiv preprint arXiv:2407.2 1 0892, Jul. 2024. Available: https://doi.org/10.48550/arXiv.2407.20892.

[12] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," arXiv preprint arXiv:2207.02696, Jul. 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2207.02696.

[13] R. Girshick, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Columbus, OH, USA, Jun. 2014, pp. 580–587, doi: 10.1109/CVPR.2014.81.

[14] M. Eisenbach, H. Franke, E. Franze, M. Köhler, D. Aganian, D. Seichter, and H.-M. Gross, "Detection of Novel Objects without Fine-Tuning in Assembly Scenarios by Class-Agnostic Object Detection and Object Re-Identification," Automation, vol. 5, no. 3, pp. 373–406, 2024, doi: 10.3390/automation5030023.

[15] H. Liu, Y. Zhang, and X. Sun, "Multi-camera tracking: A comprehensive review," Neurocomputing, vol. 526, pp. 92–106, Mar. 2023, doi: 10.1016/j.neucom.2023.126558.

[16] J. Oyekan, "Utilising Low-Cost RGB-D Cameras to Track the Real-Time Progress of a Manual Assembly Sequence," Assembly Automation, vol. 40, no. 6, pp. 925–939, 2020, doi: 10.1108/AA-06-2018-078.

[17] P. Bringmann, H. Zimmermann, and G. Fuchs, "A comprehensive review of movement comparison measures," International Journal of Geographical Information Science, vol. 28, no. 2, pp. 239–267, Feb. 2014, doi: 10.1080/15230406.2014.890071.

[18] Y. Chen and T. Li, "Dynamic updates for temporal sequence analysis: Optimized DTW algorithms," arXiv preprint arXiv:2310.18128, Oct. 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.18128.

[19] Y. Zhang and L. Nie, "Human motion similarity evaluation based on deep metric learning," Sci Rep (Scientific Reports), vol. 14, Art. no. 30908, 2024, doi: 10.1038/s41598-024-81762-8.

[20] A. Vaswani et al., "Attention is all you need," in Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NIPS'17), Long Beach, CA, USA, 2017, pp. 6000–6010.

[21] Z. Wang and X. Zhao, "MotionAGFormer: Transformer-based 3D action recognition with GCN integration," arXiv preprint arXiv:2310.16288, Oct. 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.16288

[22] M. Ibh, S. Grasshof, D. Witzner, and P. Madeleine, "TemPose: A new skeleton-based transformer model designed for fine-grained motion recognition in badminton," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW), Vancouver, BC, Canada, 2023, pp. 5199–5208, doi: 10.1109/CVPRW59228.2023.00548.

[23] G. Garcia-Hernando, S. Yuan, S. Baek, and T.-K. Kim, "First-Person Hand Action Benchmark with RGB-D Videos and 3D Hand Pose Annotations," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Salt Lake City, UT, USA, Jun. 2018, pp. 409–419, doi: 10.1109/CVPR.2018.00050.

[24] The Qt Company, "Qt | Tools for Each Stage of Software Development Lifecycle," Qt. Available: https://www.qt.io/product/development-tools.

[25] Google AI for Developers. MediaPipe Solutions Guide. Google AI Edge | Google AI for Developers. Retrieved April 5, 2025.4.5.

[26] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks." arXiv:1609.02907v4 [cs.LG] (2017). Published at ICLR 2017.