Event-Based 3D Hand Gesture Recognition and Pose Estimation

Shamma Alblooshi
Computer and Electrical Engineering
Khalifa University
100045387@ku.ac.ae / shamma.alblooshi@tii.ae

I. Introduction

Hand gestures are a natural and intuitive form of communication, playing a crucial role in human interaction and expression. With the rapid advancements in computer vision and artificial intelligence, the ability to recognize and interpret hand gestures has become a key component of applications in robotics, augmented reality (AR), virtual reality (VR), and human-computer interaction. From controlling robotic systems to navigating virtual environments or enabling touchless interaction with devices, accurate hand gesture recognition offers immense potential for enhancing user experiences across a wide range of domains.

Traditional hand gesture recognition systems primarily rely on RGB or RGB-D cameras to capture visual data for processing. These cameras excel at providing rich spatial and texture information, which can be leveraged to identify hand poses and movements. However, their performance often degrades under challenging conditions such as rapid hand motion, sudden changes in lighting, or environments with high dynamic range. Motion blur, overexposure, and limitations in temporal resolution hinder their ability to accurately capture and interpret fast or subtle gestures.

Event cameras, a novel type of imaging sensor, offer a compelling solution to these challenges. Unlike traditional cameras that capture frames at fixed intervals, event cameras record changes in brightness at each pixel asynchronously, producing a stream of events rather than full images. This unique imaging mechanism enables event cameras to achieve unparalleled temporal resolution (up to microseconds) and handle high dynamic range scenes effortlessly, all while consuming minimal power. These attributes make event cameras particularly suited for capturing the intricate dynamics of hand gestures in challenging scenarios.

Despite their advantages, the asynchronous and sparse nature of event data presents new challenges for hand gesture recognition. Capturing complex hand movements requires effective representations that encode both spatial and temporal information. Moreover, static hand poses generate minimal events, leading to motion ambiguity that complicates gesture interpretation. Additionally, the absence of dense annotations in event streams makes it difficult to train supervised models, as conventional annotation methods designed for frame-based data are not directly applicable.

To address these challenges, this report focuses on implementing and evaluating an event-based hand gesture recognition system. The system leverages edge and motion representations to capture the nuanced dynamics of hand gestures. By integrating temporal information from event streams, the method aims to resolve motion ambiguity and ensure accurate gesture recognition even in scenarios involving rapid hand movements or extreme lighting conditions. A weakly-supervised framework is employed to effectively learn from sparsely annotated data, incorporating loss functions such as contrast maximization and edge alignment to enhance performance.

As part of this work, a custom dataset featuring a diverse range of hand gestures was created using event cameras. This dataset includes scenarios with varying lighting conditions, fast and complex gestures, and diverse hand poses, providing a robust foundation for evaluating the system's performance. The analysis focuses on recognizing both predefined static gestures and dynamic sequences, demonstrating the system's capability to generalize across different scenarios.

The primary goal of this work is to highlight the transformative potential of event cameras in hand gesture recognition. Through rigorous testing and analysis, the report demonstrates the system's ability to achieve accurate and stable gesture recognition, outperforming traditional methods in challenging environments. Moreover, it provides valuable insights into the strengths and limitations of event-based approaches, paving the way for further research in applications such as touchless control interfaces, sign language interpretation, and interactive robotics.

By addressing the challenges associated with event-based gesture recognition and showcasing its practical applicability, this work contributes to advancing the field of human-computer interaction, enabling more seamless and efficient communication between humans and machines.

II. Related Work

A. Hand Gesture Recognition Using Conventional Cameras

Hand gesture recognition and pose estimation using RGB cameras have been a longstanding area of research. Most existing methods for hand gesture recognition and pose estimation rely on traditional RGB or depth cameras. Single-hand

estimation is a well-explored area, with models leveraging large-scale annotated datasets for supervised learning [1]–[4].

Recognition of two interacting hands using RGB cameras is inherently more complex due to the challenges of occlusion and the high degree of freedom in hand motions. Several approaches attempt to decompose the problem into intermediate tasks, such as hand segmentation and pose estimation, which are then combined to infer the overall gesture. Techniques leveraging attention mechanisms or probabilistic modeling have been shown to improve the plausibility of estimated hand interactions [5]–[7]. For example, segmenting hands from the background can simplify subsequent pose estimation, but accurate segmentation remains challenging in cluttered or dynamic backgrounds.

Despite their widespread use, RGB cameras face inherent limitations for hand gesture recognition in real-world settings. They suffer from low temporal resolution as RGB cameras typically operate at fixed frame rates (e.g., 30 fps), which can fail to capture rapid hand movements accurately. In addition, it can suffer from motion blur since fast hand gestures often result in blurry images, reducing the accuracy of pose estimation. Furthermore, sudden changes in lighting conditions or high dynamic range scenes can degrade the quality of the captured images. Finally, interacting hands or overlapping objects in the scene can obscure critical features needed for accurate gesture recognition [8].

B. Hand Gesture Recognition using Event-Based Cameras

Event cameras offer significant advantages for dynamic gesture recognition, such as high temporal resolution and resilience to motion blur. Traditional image-based techniques cannot be directly applied to event streams due to their sparse and asynchronous nature. To adapt, early approaches aggregated events into frame-like structures using fixed time intervals, enabling the application of conventional convolutional neural networks (CNNs) [9], [10]. For example, the Locally Normalized Event Surface (LNES) representation encodes events into 2D images, combining spatial and temporal information. However, such methods struggle with the sparsity of event data and are computationally expensive when processing dense event streams.

More recent efforts have proposed using Event Point Cloud (EPC) representations, which preserve the sparsity and temporal structure of event data [11], [12]. EPCs have been successfully applied to gesture recognition using models like PointNet. Extensions such as Rasterized Event Point Cloud (REPC) introduce re-sampling strategies to manage varying densities in event streams. However, these methods have primarily focused on simple gestures and single-hand scenarios, leaving more complex interactions underexplored.

One of the core challenges in event-based hand gesture recognition is handling entangled events from multiple moving hands or objects, which can lead to ambiguities in interpreting the data. Additionally, motion ambiguity arises when static hand parts generate no events, complicating gesture recognition tasks. Sparse annotation is another major obstacle. Event streams, by nature, lack the dense labels typical of traditional datasets, making fully supervised training infeasible. Semi-supervised and weakly-supervised approaches are increasingly being adopted, integrating synthetic data and real-world event streams to address the annotation gap.

Building upon these prior works, [13] introduces a method tailored to event-based gesture recognition that leverages an Event Point Cloud representation. By incorporating a feature-wise attention mechanism and semi-supervised training, the proposed approach addresses the sparsity and annotation challenges inherent in event data. Unlike previous methods that process event streams as dense frames, this work maintains the sparsity and temporal richness of events, enabling robust recognition of complex gestures. Additionally, the integration of synthetic event segmentation and real-world data bridges the domain gap, ensuring generalizability across diverse scenarios.

III. Methodology

The proposed system leverages event-based vision for robust and high-temporal-resolution hand gesture recognition. The overall framework consists of several stages, beginning with the conversion of asynchronous event streams into a structured event cloud representation and culminating in the accurate prediction of hand gestures and poses.

Figure 1 shows the overall architecture of the method.

A. Input Event

The system starts by processing the input from an event camera, which captures asynchronous changes in brightness. Each event is represented as:

$$e_i = (x_i, y_i, t_i, p_i)$$

where (x_i, y_i) denotes the pixel location, t_i is the timestamp, and $p_i \in \{-1, +1\}$ indicates the polarity of the brightness change. Unlike traditional frame-based inputs, this sparse and high-temporal-resolution data captures rapid and complex hand motions.

B. Event Cloud Representation

To process the asynchronous data, the event stream is converted into an *Event Point Cloud (EPC)* representation. The EPC aggregates the spatial (x, y) and temporal (t) components of the events into a structured format. This representation preserves the sparsity and temporal richness of the event data, enabling the model to capture fine-grained motion patterns critical for hand gesture recognition.

To form the Event Cloud, the system collects events within a fixed temporal window $[t_{\text{start}}, t_{\text{end}}]$. The set of events within this window is defined as:

$$E = \{e_i \mid t_{\text{start}} \leq t_i \leq t_{\text{end}}\}.$$

This representation retains the spatiotemporal structure of the events, resulting in a sparse, high-dimensional point cloud. The final Event Cloud can be written as:

$$E = \{(x_i, y_i, t_i, p_i)\}_{i=1}^{M},$$

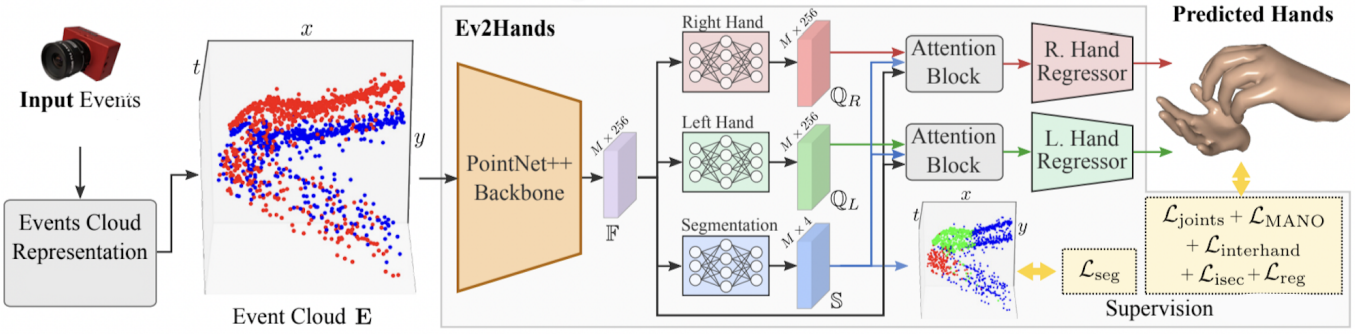


Fig. 1. System Architecture

where M is the number of events within the temporal window.

The Event Cloud is subsequently used as input to the PointNet++ model for feature extraction, preserving the spatial and temporal relationships among the events.

C. PointNet++ Backbone

PointNet++ is an advanced deep learning architecture designed to process point cloud data, which is inherently irregular, unordered, and sparse. Unlike traditional convolutional neural networks (CNNs) that work on structured grid data like images, PointNet++ operates directly on point clouds, leveraging hierarchical feature learning to capture both local and global patterns. In our context of event-based gesture recognition, the Event Cloud Representation serves as a point cloud, where each event is represented by spatial coordinates (x,y), temporal information (t), and polarity (p). PointNet++ is particularly suited for this scenario as it preserves the spatio-temporal relationships in the data, allowing the model to learn fine-grained motion details critical for hand gestures while also understanding the broader context of hand movement.

The hierarchical structure of PointNet++ is ideal for our use case as it first captures local features, such as finger motions and their interactions, and then aggregates them into global features representing the overall hand gesture. This is particularly important when dealing with dynamic scenarios, where events are generated asynchronously, and gestures involve complex spatial and temporal patterns. Moreover, PointNet++ can handle the sparse nature of event data effectively, avoiding the need to convert the data into dense frames, which might lose valuable temporal information. By using PointNet++, the system is enabled to process the Event Cloud Representation efficiently and accurately, ensuring robust feature extraction that supports the downstream tasks of hand pose estimation, segmentation, and gesture recognition.

The output of the PointNet++ backbone is a feature representation $F \in \mathbb{R}^{M \times 256}$, where:

- M is the number of points in the input Event Cloud E (number of events).
- 256 represents the feature dimension of each point, capturing spatial, temporal, and contextual information about the events.

This feature representation F serves as the input to the subsequent branches of the network (right hand branch, left hand branch, and segmentation branch.

D. Three Branches

The **right-hand branch** takes F as input and produces a latent feature representation for the right hand:

$$Q_R \in \mathbb{R}^{M \times 256}.$$

Similarly, the output of the **left-hand branch** is a latent feature representation:

$$Q_L \in \mathbb{R}^{M \times 256},$$

where:

- M is the number of points in the input Event Cloud.
- 256 is the feature dimension capturing the spatial, temporal, and contextual characteristics of the left/right hand

The **segmentation branch** processes the feature representation $F \in \mathbb{R}^{M \times 256}$ to classify each point in the Event Cloud into one of four categories: left hand, right hand, background, or no class. The output of this branch is:

$$S \in \mathbb{R}^{M \times 4},$$

where each row represents the probabilities of a point belonging to the four categories:

$$S_i = [P_{\text{left}}, P_{\text{right}}, P_{\text{background}}, P_{\text{no class}}],$$

with $P_{\text{left}}, P_{\text{right}}, P_{\text{background}}, P_{\text{no class}} \geq 0$ and $P_{\text{left}} + P_{\text{right}} + P_{\text{background}} + P_{\text{no class}} = 1$.

- **Left Hand Label (P_{left}):** Indicates that the event was directly produced by the left hand.
- **Right Hand Label (P_{right}):** Indicates that the event was directly produced by the right hand.
- **Background Label ($P_{\text{background}}$):** Covers events triggered by non-hand objects, such as torso or arm movements, or changes in the background.
- **No Class Label ($P_{\text{no class}}$):** Assigned when an event point combines multiple sources in the same temporal window. For example, if a left-hand event and a background event are triggered at the same pixel within the same time window, the event point is labeled as ”no class.”

The segmentation branch is crucial for isolating events relevant to the hands, ensuring that points corresponding to non-hand objects or ambiguous regions are appropriately labeled. This improves the clarity and accuracy of subsequent hand pose predictions.

To enhance the model's ability to focus on features relevant to each hand independently, we incorporate a feature-wise attention mechanism. This module dynamically assigns weights to features based on their importance for specific tasks, enabling the model to extract hand-specific information effectively.

E. Attention Block and Hand Regressor

The feature-wise attention block is defined as:

$$\text{Attention}(Q_{(\cdot)}, S, F) = F \left(\text{Softmax} \left(\frac{Q_{(\cdot)}^T S}{\sqrt{d_s}} \right) \right),$$

where:

- $Q_{(\cdot)}$ represents the hand-specific features (Q_L for the left hand and Q_R for the right hand).
- S is the key value matrix derived from the event segmentation predictions.
- F represents the event features from the PointNet++ backbone.
- d_s is the dimension of S, used for scaling in the attention computation.

The attention mechanism operates by combining the hand-specific features $Q_{(\cdot)}$ with the segmentation key values S using a scaled dot product, followed by a softmax operation. This produces attention weights that are applied to the event features F to generate refined attention features:

$$H_L, H_R \in \mathbb{R}^{M \times d_s},$$

where H_L and H_R are the refined features for the left and right hands, respectively.

This mechanism is applied independently to each hand, ensuring that the features for the left and right hands are disentangled while also responding to the segmentation predictions from S. By leveraging both hand-specific features and event segmentation, the attention block identifies the most relevant features for each hand under different interaction conditions.

Additionally, the feature-wise attention mechanism improves the segmentation predictions (S) by refining them during training, making them more accurate. This refinement is especially useful when training on synthetic data and fine-tuning on real-world datasets, enhancing the model's generalization capabilities across diverse scenarios.

The refined representations Q'_L and Q'_R are then passed to the left and right hand regressors, respectively. Each regressor uses the MANO model to parametrize the 3D pose and shape:

$$\text{Hand Pose} = \mathcal{P}(\theta, \beta),$$

where:

- θ represents the pose parameters (joint angles).
- β represents the shape parameters (hand-specific variations).

The global transformations R and T are predicted alongside the pose and shape parameters.

F. Loss Functions

To ensure accurate and physically plausible predictions, the proposed framework employs a combination of loss functions. These losses are applied to supervise the outputs from the left hand branch, right hand branch, and segmentation branch. Each loss term addresses a specific aspect of the model's predictions, including joint accuracy, hand interactions, and segmentation precision.

1) Joint Loss (L_{joints}): The joint loss penalizes discrepancies between the predicted and ground-truth joint positions. It ensures that the predicted 3D hand poses closely align with the actual joint locations:

$$L_{\text{joints}} = \frac{1}{N} \sum_{i=1}^{N} \|J_i - \hat{J}_i\|^2,$$

where:

- J_i is the ground-truth 3D position of the i-th joint.
- $\hat{J}_i$ is the predicted 3D position of the i-th joint.
- N is the total number of joints.

2) MANO Loss (L_{MANO}): The MANO loss penalizes deviations between the predicted and reference hand parameters. This includes pose parameters (θ), shape coefficients (β), global rotation (R), and rigid translation (t).

For pose and shape parameters, the loss uses the ℓ_2-norm to measure the squared differences between predicted and reference values. For the global translation vector (t), the loss employs the ℓ_1-norm to capture absolute deviations. The total MANO loss is defined as:

$$L_{\text{MANO}} = \|\hat{\theta} - \theta\|^2 + \|\hat{\beta} - \beta\|^2 + \|\hat{R} - R\|^2 + \|\hat{t} - t\|_1,$$

where:

- $\hat{\theta}$, $\hat{\beta}$, $\hat{R}$, and $\hat{t}$ are the predicted pose, shape, global rotation, and translation parameters, respectively.
- θ, β, R, and t are the reference (ground-truth) pose, shape, global rotation, and translation parameters.

3) Inter-Hand Loss ($L_{interhand}$): The inter-hand loss is designed to penalize deviations between the shape parameters of the left and right hands and the relative positions between them. By considering both hands simultaneously, this loss ensures realistic articulation and spatial relationships during interactions.

The inter-hand loss is defined as:

$$L_{\text{interhand}} = \|\beta_{\text{left}} - \beta_{\text{right}}\|^2 + \mathcal{I}_J + \mathcal{I}_T,$$

where:

- β_{left} and β_{right} are the shape parameters of the left and right hands, respectively.
- $\mathcal{I}_J$ accounts for relative articulation errors.
- $\mathcal{I}_T$ accounts for relative translation errors.

The term $\mathcal{I}_J$ measures the deviation in relative joint positions between the predicted and ground-truth values:

$$\mathcal{I}_J = \frac{1}{N_J} \sum_{i=1}^{N_J} \|(\hat{J}_{\text{left},i} - \hat{J}_{\text{right},i}) - (J_{\text{left},i} - J_{\text{right},i})\|^2,$$

where:

- N_J is the total number of joints.
- $\hat{J}_{\text{left},i}$ and $\hat{J}_{\text{right},i}$ are the predicted 3D positions of the i-th joint for the left and right hands, respectively.
- $J_{\text{left},i}$ and $J_{\text{right},i}$ are the corresponding ground-truth joint positions.

The term $\mathcal{I}_T$ measures the deviation in relative translations between the predicted and ground-truth global translations:

$$\mathcal{I}_T = \|(\hat{t}_{\text{left}} - \hat{t}_{\text{right}}) - (t_{\text{left}} - t_{\text{right}})\|^2,$$

where:

- $\hat{t}_{\text{left}}$ and $\hat{t}_{\text{right}}$ are the predicted global translations for the left and right hands, respectively.
- t_{left} and t_{right} are the corresponding ground-truth translations.

By penalizing differences in shape parameters and ensuring accurate relative positions of joints and translations, the inter-hand loss encourages realistic hand interactions. This is particularly important for scenarios where both hands are interacting or are in close proximity, ensuring plausible spatial and temporal relationships.

4) Segmentation Loss (L_{seg}): The segmentation loss supervises the segmentation branch to classify each event in the Event Cloud into one of the defined classes: left hand, right hand, or background. This loss is primarily used during training on synthetic data, where ground-truth labels for segmentation are available. When training on real event data, the segmentation branch is indirectly supervised through gradients propagated from the hand parameter supervision.

The segmentation loss is defined as:

$$L_{\text{seg}} = \text{CrossEntropy}(\text{Softmax}(S), c),$$

where:

- $S \in \mathbb{R}^{M \times 3}$ is the predicted segmentation output for M event points, with probabilities for each of the three classes: left hand, right hand, and background.
- $c \in \mathbb{R}^M$ is the ground-truth class label for each event point.
- $\text{Softmax}(S)$ converts the logits S into probabilities for each class.
- CrossEntropy measures the discrepancy between the predicted probabilities and the ground-truth labels.

5) Regularization Loss (L_{reg}): The regularization loss is used to enforce smoothness and stability in the predicted hand poses:

$$L_{\text{reg}} = \lambda_\theta \|\theta\|^2 + \lambda_\beta \|\beta\|^2,$$

where:

- θ and β are the predicted pose and shape parameters, respectively.
- λ_θ and λ_β are regularization coefficients.

6) Total Loss (L_{total}): The total loss function combines all the individual losses, weighted by their respective coefficients:

$$L_{\text{total}} = \alpha_{\text{joints}} L_{\text{joints}} + \alpha_{\text{MANO}} L_{\text{MANO}} + \alpha_{\text{interhand}} L_{\text{interhand}} + \alpha_{\text{seg}} L_{\text{seg}} + \alpha_{\text{reg}} + \alpha_{\text{isec}} L_{\text{isec}} + L_{\text{reg}},$$

where $\alpha_{\text{joints}}, \alpha_{\text{MANO}}, \alpha_{\text{interhand}}, \alpha_{\text{seg}}, \alpha_{\text{reg}}$ are the weights for the respective loss terms.

This comprehensive loss formulation ensures that the model learns accurate hand poses, plausible hand interactions, precise segmentations, and stable predictions, contributing to robust event-based hand gesture recognition.

IV. Experimental Setup & Code Explanation

Figure 2 shows the experimental setup for collecting a new dataset. It consists of an event camera mounted securely on a tripod with a suction base for stability. The camera is connected to a laptop via a USB interface, facilitating data capture and processing in real-time.

The system is implemented using Pytorch and using Adam optimiser. For page limit concerns I will not explain the whole code, but will explain functions that are important for our implementation.

The script (Code 1 in Appendix A) starts by importing essential libraries for data handling, model processing, and visualization. Custom modules such as *Ev2HandRDataset* for dataset handling and *TEHNetWrapper* for the model are included. The paths and settings, like output dimensions (OUTPUT_HEIGHT, OUTPUT_WIDTH) and the main camera settings (MAIN_CAMERA), are also imported from the settings module. The script checks the availability of a GPU and assigns the device (either CUDA or CPU) accordingly.

The demo function (Code 2 in Appendix A) handles the core task of running inference on a batch of event-based data using the pre-trained model and preparing the output for visualization. The model is set to evaluation mode (net.eval()), and the input event data is transferred to the selected device. The inference process is performed within a torch.no_grad() block to optimize memory usage. The inference results include class logits (to classify hand parts) and 3D hand pose data (vertices and joints) for both left and right hands. For visualization, a segmentation mask is generated by iterating over the coordinates of the event data and assigning colors based on the predicted class labels. Each frame, consisting of segmentation masks, 3D vertices, and joints, is appended to a list of frames to be visualized later.

The main function(Code 3 in Appendix A) orchestrates the entire workflow. It begins by setting up argument parsing (arg_parser.demo()) and creating an output directory (outputs). The pre-trained model is loaded using torch.load() and its weights are restored using net.load_state_dict(). The dataset is initialized using the Ev2HandRDataset class, which prepares the event-based data for inference. A renderer is created to render the 3D hand meshes. A scene is constructed with ambient lighting and directional lights to ensure clear visualization. The renderer and scene are set up to visualize the predicted

Fig. 2. Experimental setup

3D hand pose data. The dataset is divided into batches, and the demo function is called for each batch. For each frame in the batch, the script processes event frames and segmentation masks. It uses the predictions (vertices and faces) to create 3D meshes for both hands using Trimesh. These meshes are transformed and rendered into RGB images using Pyrender. The rendered images are combined with the event frames and segmentation masks into a stacked image (img_stack). The stacked images are displayed in real-time using OpenCV (cv2.imshow()) and simultaneously written to a video file (outputs/video.mp4). Users can interrupt the visualization by pressing the q key, which releases the video writer and exits the program. A critical aspect of this script is the generation of 3D hand meshes. For each hand, the predicted vertices and faces are combined into a 3D mesh object. The meshes are visualized using a virtual camera (MAIN_CAMERA) in a Pyrender scene. The rendered images provide a clear depiction of the model's predictions. The program loops through all batches in the dataset, continuously visualizing and saving the predictions. Once all batches are processed, the video writer is released, ensuring the generated video file is saved properly

V. Results

The results presented in table I- V provide a comprehensive visualization of the system's progression from raw event data to final 3D hand pose predictions. Starting with the first column, the Event Data provides a direct representation of the asynchronous changes in pixel intensity captured by the event camera. These images clearly showcase the advantages of event cameras, which include their ability to capture fine-grained motion details with high temporal resolution. The richness of the event data reflects the dynamic nature of the hand gestures being performed, with high clarity and minimal noise, ensuring that the foundational input to the pipeline is robust and reliable. The event data effectively encapsulates both spatial and temporal variations, making it an ideal starting point for downstream processes like feature extraction and gesture modeling. The visual quality of these event frames highlights the system's ability to preprocess and represent event streams in a manner conducive to accurate analysis.

In the second column, the Event Point Cloud representation demonstrates the translation of the raw event data into 3D spatiotemporal representations. This step provides a richer understanding of the motion dynamics by mapping the data across temporal and spatial axes. The event clouds show distinct clustering of points that correspond to the movements and shapes of the hand during the gestures, with each frame revealing a clear depiction of motion trajectories. The inclusion of the temporal dimension allows the system to model the progression of motion over time, which is critical for interpreting dynamic gestures. The event clouds reflect the effectiveness of the feature extraction process in capturing both the shape and motion patterns of the hand, enabling a more nuanced analysis than traditional 2D representations. By visualizing the depth and structure of the gestures, the event clouds reinforce the importance of preserving temporal information, which is crucial for accurate 3D pose estimation.

The third column, featuring the segmentation maps, provides insight into the system's ability to isolate relevant hand regions from the background. These results showcase precise segmentation of the hand, with distinct boundaries that separate the hand from other elements in the scene. The segmentation process is crucial for ensuring that only the relevant data is passed to the pose estimation model, minimizing interference from extraneous elements. The color-coded maps emphasize different regions of the hand, such as the palm and fingers, further enhancing the clarity and utility of the segmented output. The consistency across the segmentation results, even for diverse poses and orientations, demonstrates the robustness of the segmentation algorithm in identifying the key features of the hand. This step lays a strong foundation for the subsequent prediction process by ensuring that the input to the pose estimation model is focused solely on the hand and its key structures.

Finally, the fourth column highlights the system's predicted 3D hand gesture poses , representing the culmination of the processing pipeline. The predicted poses are highly detailed, capturing intricate aspects of hand structure and motion. Each pose shows a clear articulation of fingers and joints, with accurate reconstructions of gestures ranging from simple hand shapes to more complex configurations. The system's ability to produce such realistic and precise 3D representations is a testament to the effectiveness of the model in leveraging both spatial and temporal information from the earlier stages. The clear alignment between the segmented regions and the predicted hand poses further reinforces the robustness of the pipeline, demonstrating a seamless transition from raw input to meaningful output.

Overall, the results in the table reflect a well-executed methodology for event-based 3D hand pose estimation. The clarity of the event data, the depth provided by the event clouds, the precision of the segmentation, and the realism of the 3D predictions all contribute to showcasing the system's effectiveness. These results underscore the capability of event

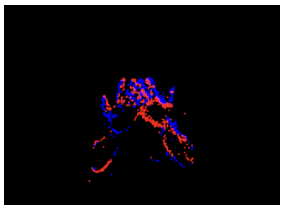
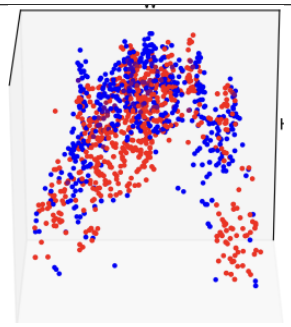
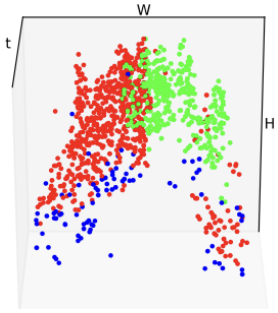
Event	Event Cloud	Segmentation	Prediction
			

TABLE I
COMPARISON OF EVENT DATA, EVENT CLOUD, SEGMENTATION, AND PREDICTION FOR EVENT 1.

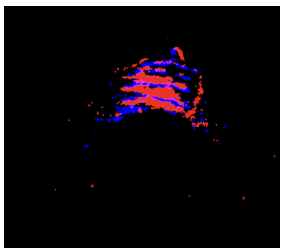
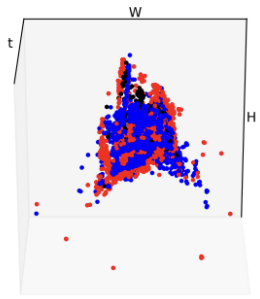
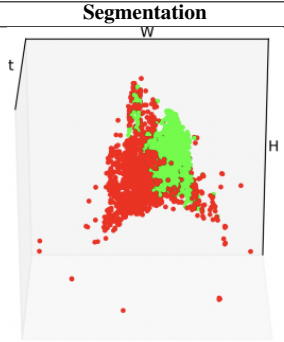
Event	Event Cloud	Segmentation	Prediction
			

TABLE II
COMPARISON OF EVENT DATA, EVENT CLOUD, SEGMENTATION, AND PREDICTION FOR EVENT 2.

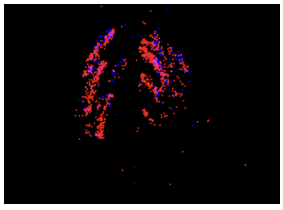
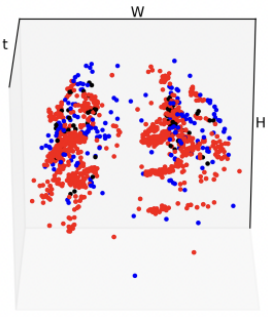
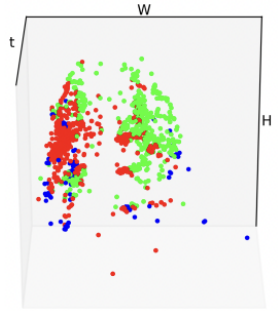
Event	Event Cloud	Segmentation	Prediction
			

TABLE III
COMPARISON OF EVENT DATA, EVENT CLOUD, SEGMENTATION, AND PREDICTION FOR EVENT 3.

cameras and advanced computational models to handle dynamic and complex gestures, paving the way for innovative applications in real-world scenarios. This pipeline exemplifies the strength of combining high-resolution event data with sophisticated processing techniques to achieve accurate and reliable outcomes.

In addition, the design of the system demonstrates exceptional robustness in challenging lighting conditions, as evident in Figure 3. Traditional RGB-based methods often struggle in low-light environments due to their reliance on capturing high-quality texture and color information, which becomes degraded in such scenarios. However, this system leverages the unique capabilities of event cameras, which are inherently well-suited for these conditions.

As shown in the RGB stream, the input captured in a dimly lit environment contains limited visual information, making it difficult for conventional approaches to extract meaningful features. In contrast, the event stream successfully captures motion-related changes with high temporal resolution, unaffected by the lack of ambient light. This ensures that the system continues to deliver precise and reliable performance in low-light settings. The predicted 3D hand pose further validates this capability, as the system accurately reconstructs the hand structure despite the challenges posed by poor lighting.

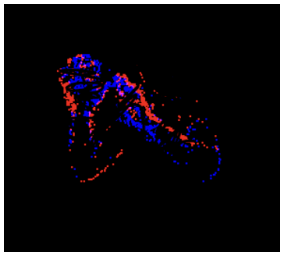
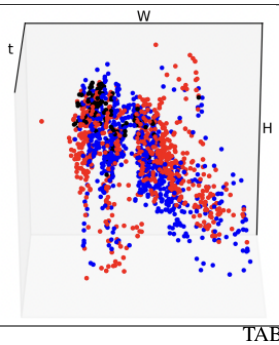
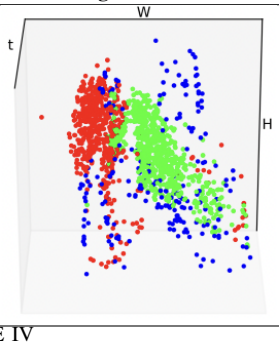
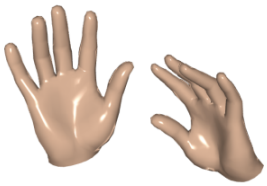
Event	Event Cloud	Segmentation	Prediction
			

TABLE IV
COMPARISON OF EVENT DATA, EVENT CLOUD, SEGMENTATION, AND PREDICTION FOR EVENT 4.

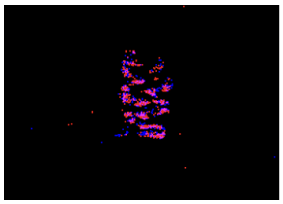
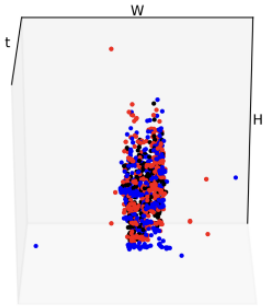
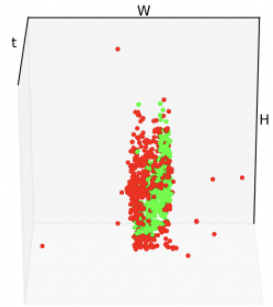
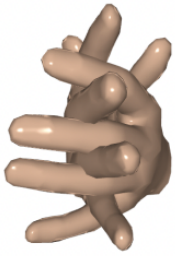
Event	Event Cloud	Segmentation	Prediction
			

TABLE V
COMPARISON OF EVENT DATA, EVENT CLOUD, SEGMENTATION, AND PREDICTION FOR EVENT 5.

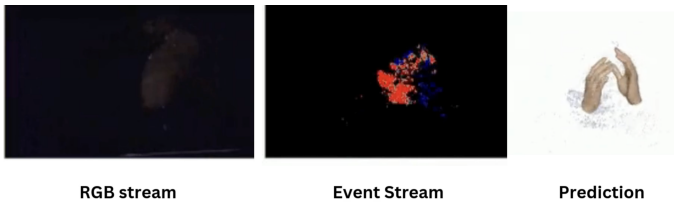


Fig. 3. Light Condition

VI. LIMITATIONS AND FUTURE WORK

While the results presented in this work demonstrate the robustness and accuracy of the proposed pipeline, certain limitations provide opportunities for future exploration and enhancement. One of the primary assumptions in this system is the use of a fixed camera. Although this assumption aligns with traditional RGB-based methods, it introduces challenges when considering a portable or moving event camera. A moving event camera generates a substantial amount of background clutter due to its sensitivity to changes in pixel intensity across the entire frame. This additional noise could interfere with the system's ability to isolate and focus on the object of interest, such as the hand, thereby reducing the accuracy of the segmentation and pose estimation processes. Future work could address this limitation by developing techniques to extract events specifically caused by the object of interest while filtering out irrelevant background noise. Such advancements would extend the applicability of this system to more dynamic scenarios, such as wearable devices or mobile applications.

Another promising direction for future research lies in the integration of RGB data with event streams. The current approach leverages only event-based data, which excels in capturing motion dynamics and operates with low latency. However, it lacks the rich texture and color information provided by RGB cameras. Combining these two modalities could enhance the visual fidelity of the data, allowing the system to leverage the strengths of both. For instance, RGB data could provide detailed textures and color information, while the event stream ensures responsiveness and accuracy in capturing rapid hand movements. This fusion could enable the system to produce high-quality textured 3D reconstructions of fast-moving hands, significantly expanding its potential applications in areas like virtual reality, animation, and advanced human-computer interaction.

Overall, addressing the limitations of fixed-camera setups and exploring multi-modal data fusion could significantly enhance the robustness, versatility, and visual quality of the system. These directions not only build upon the current strengths of the approach but also open up exciting possibilities for extending its capabilities to more complex and dynamic real-world scenarios.

APPENDIX

In the Appendix, we provide the code used in our experiments and we will refer to them in section IV

```
import sys; sys.path.append('..') # add
    settings.
import os
import torch
import cv2
import time
import pyrender
import numpy as np
import trimesh

import arg_parser

from torch.utils.data import DataLoader
from dataset import Ev2HandRDataset
from model import TEHNetWrapper
from settings import OUTPUT_HEIGHT,
    OUTPUT_WIDTH, MAIN_CAMERA,
    REAL_TEST_DATA_PATH
```

Code 1. Libraries and Settings required for experiments

```
def demo(net, device, batch):
    net.eval()

    events = batch['events']
    events = events.to(device=device, dtype=
        torch.float32)

    start_time = time.time()
    with torch.no_grad():
        outputs = net(events)

    # Waits for everything to finish running
    torch.cuda.synchronize()
    end_time = time.time()

    N = events.shape[0]
    print(end_time - start_time)

    outputs['class_logits'] = outputs['
        class_logits'].softmax(1).argmax(1).
        int().cpu()

    frames = list()
    for idx in range(N):
        hands = dict()

        hands['left'] = {
            'vertices': outputs['left']['
                vertices'][idx].cpu(),
            'j3d': outputs['left']['j3d'][idx
                ].cpu(),
        }

        hands['right'] = {
            'vertices': outputs['right']['
                vertices'][idx].cpu(),
            'j3d': outputs['right']['j3d'][idx
                ].cpu(),
        }

        coordinates = batch['coordinates'][idx
            ]
        seg_mask = np.zeros((OUTPUT_HEIGHT,
            OUTPUT_WIDTH, 3), dtype=np.uint8)
        for edx, (y, x) in enumerate(
            coordinates):
            y, x = y.int(), x.int()

            cid = outputs['class_logits'][idx
                ][edx]

            if cid == 3:
                seg_mask[y, x] = 255
            else:
                seg_mask[y, x, cid] = 255

        hands['seg_mask'] = seg_mask

        frames.append(hands)

    return frames
```

Code 2. Support Function for Running Inference On The Data

```
def main():
    arg_parser.demo()
    os.makedirs('outputs', exist_ok=True)

    device = torch.device('cuda' if torch.cuda
        .is_available() else 'cpu')

    net = TEHNetWrapper(device=device)

    save_path = os.environ['CHECKPOINT_PATH']
    batch_size = int(os.environ['BATCH_SIZE'])

    checkpoint = torch.load(save_path,
        map_location=device)
    net.load_state_dict(checkpoint['state_dict
        '], strict=True)

    renderer = pyrender.OffscreenRenderer(
        viewport_width=OUTPUT_WIDTH,
        viewport_height=OUTPUT_HEIGHT)

    scene = pyrender.Scene(ambient_light=(0.3,
         0.3, 0.3))
    light = pyrender.DirectionalLight(color
        =[1.0, 1.0, 1.0], intensity=0.8)
    light_pose = np.eye(4)
    light_pose[:3, 3] = np.array([0, -1, 1])
    scene.add(light, pose=light_pose)
    light_pose[:3, 3] = np.array([0, 1, 1])
    scene.add(light, pose=light_pose)
    light_pose[:3, 3] = np.array([1, 1, 2])
    scene.add(light, pose=light_pose)

    rot = trimesh.transformations.
        rotation_matrix(np.radians(180), [1,
        0, 0])

    mano_hands = net.hands

    if os.path.exists(REAL_TEST_DATA_PATH) and
         len(os.listdir(REAL_TEST_DATA_PATH))
        > 0:
        data = REAL_TEST_DATA_PATH
    else:
```

```
    data = '../data/demo/example.pickle'

validation_dataset = Ev2HandRDataset(data,
     demo=True, augment=False)
validation_loader = DataLoader(
    validation_dataset, batch_size=
    batch_size, shuffle=True, num_workers
    =8, pin_memory=True)

video_fps = 25
video = cv2.VideoWriter('outputs/video.mp4
    ', cv2.VideoWriter_fourcc(*'mp4v'),
    video_fps, (3 * OUTPUT_WIDTH,
    OUTPUT_HEIGHT))

for bdx, batch in enumerate(
    validation_loader):
    frames = demo(net=net, device=device,
        batch=batch)

    for idx, frame in enumerate(frames):
        event_frame = batch['event_frame'
            ][idx].cpu().numpy().astype(
            dtype=np.uint8)
        seg_mask = frame['seg_mask']

        pred_meshes = list()
        for hand_type in ['left', 'right'
            ]:
            faces = mano_hands[hand_type].
                faces

            pred_mesh = trimesh.Trimesh(
                frame[hand_type]['vertices
                '].cpu().numpy() * 1000,
                faces)
            pred_mesh.visual.vertex_colors
                 = [255, 0, 0]
            pred_meshes.append(pred_mesh)

        pred_meshes = trimesh.util.
            concatenate(pred_meshes)
        pred_meshes.apply_transform(rot)

        camera = MAIN_CAMERA

        nc = pyrender.Node(camera=camera,
            matrix=np.eye(4))
        scene.add_node(nc)

        mesh_node = pyrender.Node(mesh=
            pyrender.Mesh.from_trimesh(
            pred_meshes))
        scene.add_node(mesh_node)
        pred_rgb, depth = renderer.render(
            scene)
        scene.remove_node(mesh_node)
        scene.remove_node(nc)

        pred_rgb = cv2.cvtColor(pred_rgb,
            cv2.COLOR_RGB2BGR)
        pred_rgb[pred_rgb == 255] = 0

        img_stack = np.hstack([event_frame
            , seg_mask, pred_rgb])
        video.write(img_stack)
        cv2.imshow('image', img_stack)
        c = cv2.waitKey(1)

        if c == ord('q'):
            video.release()
            exit(0)

video.release()
```

Code 3. The Main Code

REFERENCES

[1] Seungryul Baek, Kwang In Kim, and Tae-Kyun Kim. Pushing the envelope for rgb-based dense 3d hand pose estimation via neural rendering, 2019.

[2] Adnane Boukhayma, Rodrigo de Bem, and Philip H. S. Torr. 3d hand shape and pose from images in the wild, 2019.

[3] Liuhao Ge, Zhou Ren, Yuncheng Li, Zehao Xue, Yingying Wang, Jianfei Cai, and Junsong Yuan. 3d hand shape and pose estimation from a single rgb image, 2019.

[4] Franziska Mueller, Florian Bernard, Oleksandr Sotnychenko, Dushyant Mehta, Srinath Sridhar, Dan Casas, and Christian Theobalt. Ganerated hands for real-time 3d hand tracking from monocular rgb, 2017.

[5] Zicong Fan, Adrian Spurr, Muhammed Kocabas, Siyu Tang, Michael J. Black, and Otmar Hilliges. Learning to disambiguate strongly interacting hands via probabilistic per-pixel part segmentation, 2021.

[6] Shile Li and Dongheui Lee. Point-to-pose voting based hand pose estimation using residual permutation equivariant layer, 2018.

[7] Franziska Mueller, Micah Davis, Florian Bernard, Oleksandr Sotnychenko, Mickeal Verschoor, Miguel A. Otaduy, Dan Casas, and Christian Theobalt. Real-time pose and shape reconstruction of two interacting hands with a single depth camera. *ACM Trans. Graph.*, 38(4), July 2019.

[8] Andrea Ceccarelli and Francesco Secci. Rgb cameras failures and their effects in autonomous driving applications. *IEEE Transactions on Dependable and Secure Computing*, 20(4):2731–2745, 2023.

[9] Lan Xu, Weipeng Xu, Vladislav Golyanik, Marc Habermann, Lu Fang, and Christian Theobalt. Eventcap: Monocular 3d capture of high-speed human motions using an event camera, 2019.

[10] Viktor Rudnev, Vladislav Golyanik, Jiayi Wang, Hans-Peter Seidel, Franziska Mueller, Mohamed Elgharib, and Christian Theobalt. Eventhands: Real-time neural 3d hand pose estimation from an event stream, 2021.

[11] Qinyi Wang, Yexin Zhang, Junsong Yuan, and Yilong Lu. Space-time event clouds for gesture recognition: From rgb cameras to event cameras. *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1826–1835, 2019.

[12] Jiaan Chen, Hao Shi, Yaozu Ye, Kailun Yang, Lei Sun, and Kaiwei Wang. Efficient human pose estimation via 3d event point cloud. In *2022 International Conference on 3D Vision (3DV)*, pages 1–10, 2022.

[13] Christen Millerdurai, Diogo Luvizon, Viktor Rudnev, André Jonas, Jiayi Wang, Christian Theobalt, and Vladislav Golyanik. 3d pose estimation of two interacting hands from a monocular event camera, 2023.