Design and Implementation of a Real-Time Gesture-Controlled Interactive Snake System Based on Jetson Nano

Zhu Jiarun
12210212@mail.sustech.edu.cn

Feng Junming
12311031@mail.sustech.edu.cn

Zhang Sihua
12410106@mail.sustech.edu.cn

Zhou Zixi
12312051@mail.sustech.edu.cn

Liu Yulin
12312054@mail.sustech.edu.cn

***Abstract*—Targeting the potential of edge computing devices in Natural User Interface (NUI) scenarios, this project implements a real-time gesture-controlled Snake system on the NVIDIA Jetson Nano platform. The system deploys the pre-trained MediaPipe Hands model for efficient keypoint extraction and utilizes a custom geometric constraint-based action mapping logic to convert 3D coordinate streams into game control commands. The study focuses on verifying the operating efficiency and system integration performance of off-the-shelf deep learning models on low-power devices. Results indicate that through GStreamer hardware acceleration and a hybrid rendering optimization strategy, the system achieves smooth operation at approximately 28 FPS (single-hand mode) while maintaining the native high precision of MediaPipe. This demonstrates the feasibility of building low-latency, high-interactivity AI applications directly on edge devices using Python environments.**

***Index Terms*—Computer Vision, MediaPipe, NVIDIA Jetson Nano, Gesture Control, Edge Computing**

I. Introduction

Traditional embedded gaming interactions typically rely on physical buttons or touch screens, which are prone to hardware wear and offer limited interaction dimensions. Gesture control based on computer vision provides a contactless, intuitive, and immersive interaction method. However, running deep learning inference, game physics engines, and high-definition UI rendering simultaneously on resource-constrained edge devices presents significant optimization challenges, as surveyed by Mittal [1].

A. Paper Objectives

In this project, we select the classic "Snake" game as a carrier to address the following core engineering challenges:

1) **Inference Efficiency**: How to achieve high-precision real-time hand tracking on a GPU with limited computing power (Maxwell architecture).
2) **Interaction Logic**: How to design low-computational-cost gesture mapping logic to reduce CPU load while ensuring robustness.
3) **Rendering Balance**: How to balance visual rendering quality (aesthetic UI) with system response speed.

This paper details the system's hardware-software architecture and key algorithm implementations, focusing on stress testing and bottleneck analysis of the system's real-time performance on the Jetson Nano.

II. System Architecture

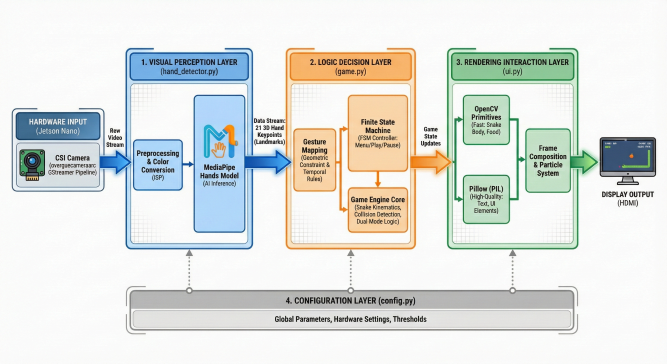


Fig. 1. System Architecture Diagram

The system follows the design principle of "High Cohesion and Low Coupling," adopting a layered architecture where data flow is driven by visual input, processed by logic, and output as rendered images.

1) **Visual Perception Layer (`hand_detector.py`)**: Acting as the system input, it is responsible for video stream acquisition and preprocessing. Its core function is invoking the MediaPipe Hands model [2] to extract 21 3D hand keypoints (Landmarks) from RGB images.
2) **Logic Decision Layer (`game.py`)**: The core controller of the system. It functions as a Finite State Machine (FSM) managing game states (Menu, Playing, Pause, Game Over). It handles snake kinematics, collision detection (AABB or distance-based), and complex interaction logic in two-player mode (e.g., the "Cut-and-Absorb" mechanic).
3) **Rendering Interaction Layer (`ui.py`)**: Responsible for visual output. This layer implements a **Hybrid Rendering Pipeline**: it utilizes `OpenCV` for high-speed geometric drawing (snake body, food) and `Pillow (PIL)` for high-quality anti-aliased text and UI rendering. Crucially, it integrates a physics-based **particle**

system for visual feedback and visualizes the "Control Ring" to guide user interaction.

4) **Configuration Layer (`config.py`)**: Unifies global parameters (resolution, color space, thresholds) and hardware-related settings (GStreamer pipelines).

III. Methodology

A. Visual Front-end and Hardware Acceleration

The system integrates the Google MediaPipe framework. Considering the resource limitations of the Jetson Nano, we focused on optimizing the input pipeline rather than model retraining.

Input Stream Optimization: A `nvarguscamerasrc` pipeline was constructed using GStreamer to leverage the onboard ISP (Image Signal Processor) for video stream acquisition and color space conversion. This provides efficient image frames to the inference engine, significantly reducing the CPU overhead compared to standard USB camera capture methods.

B. Rule-based Action Mapping Logic

To convert the 21 general hand keypoints into specific game commands, we designed a **Geometric & Temporal Mapping Strategy**:

1) **Static Gesture Classification**: We defined heuristic rules based on the relative coordinates of fingertips ($P_{\{\text{tip}\}}$) and knuckles ($P_{\{\text{pip/mcp}\}}$).
   - **Navigation**: "Index Up" (Single Player), "Victory" (Dual Player).
   - **Menu Control**: "Pinky Up" (Easy Mode), "Thumb Up" (Hard Mode).
   - **System Control**: "Fist" (Quit/Back), detected when four non-thumb fingers are folded.
2) **Temporal Confirmation Mechanism**: To prevent false triggering (e.g., passing gestures), critical actions like "Menu Selection" or "Quit" require the gesture to be held for a time window $T_{\{\text{hold}\}}ge3.0s$. This is implemented via a frame accumulator that resets instantly if the gesture confidence drops or changes.
3) **Trajectory Tracking (Dynamic Control)**: We map the **Index Finger Tip** (P_8) coordinates directly to the snake's head position using a smooth following algorithm:

$$P_{\{\text{head}\}}^{\{t\}} = P_{\{\text{head}\}}^{\{t-1\}} + \alpha \cdot \left(P_{\{\text{finger}\}}^{\{t\}} - P_{\{\text{head}\}}^{\{t-1\}} \right) \quad (1)$$

where α is a smoothing factor. This provides finer granularity for steering control than discrete grid-based movement.

C. Dynamic Rendering Optimization

In the rendering layer, the system adopts a frame-by-frame compositing strategy. To ensure UI aesthetics without sacrificing performance:

1) **Hybrid Approach**: Complex vector shapes (rounded rectangles, text) are drawn using PIL, while high-frequency updates (snake movement) use OpenCV primitives.
2) **Particle System**: A simplified physical model is used for the "eating" effect:

$$P_{\{\text{new}\}} = P_{\{\text{old}\}} + V \cdot \Delta t, \quad \text{A} = \text{A} - \text{Decay} \quad (2)$$

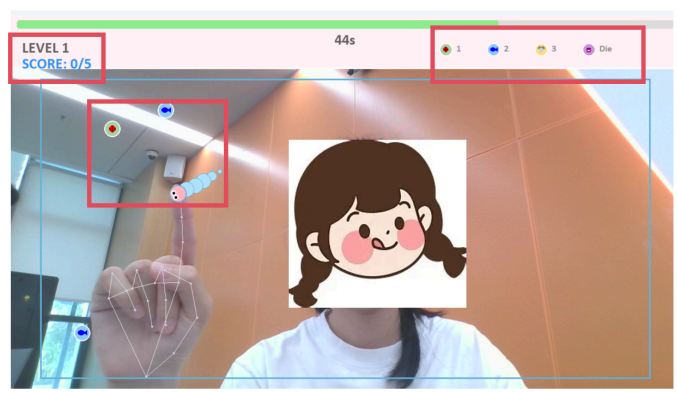
Particles are removed immediately after their lifecycle to prevent memory leaks.

D. Advanced Game Engine Logic

The logic layer implements sophisticated mechanics beyond simple collision:

1) **Dynamic Object Lifecycle (Hard Mode)**: Food objects are assigned a **Time-To-Live (TTL)** of 3 seconds. Expired food is automatically despawned and respawned, increasing cognitive load and difficulty.
2) **Topological Collision (Dual Mode)**: We implemented a "Cut-and-Absorb" mechanic. If Snake A collides with the body of Snake B, Snake B is **truncated** at the collision index. The severed segments are added to Snake A's length, creating a high-stakes competitive environment.

IV. Experimental Results & Analysis



We evaluated the system on the NVIDIA Jetson Nano (4GB RAM, Maxwell 128-core GPU). The environment is JetPack 4.6 with Python 3.6.

A. Real-time Assessment (Latency & FPS)

We tested the system's frame rate and processing latency in single-player and two-player modes.

TABLE I
SYSTEM PERFORMANCE METRICS

Scenario	FPS	Infer (ms)	Render (ms)	Proc. Latency
Single Hand	28.5	32.0	18.5	50.5ms
Two Hands	22.1	58.4	24.2	82.6ms

Analysis:

1) **Inference Bottleneck**: MediaPipe relies heavily on CPU (with TFLite XNNPACK) for pre/post-processing. Switching from single to dual-hand detection increases inference time by 82%, as the model performs regression on two independent ROIs.
2) **Rendering Efficiency**: Rendering time increases slightly in dual mode due to drawing two snakes and additional UI elements. However, it remains under 25ms, proving the hybrid rendering strategy is efficient enough for real-time applications.
3) **Playability**: The total processing latency stays within acceptable limits for a casual game. The system remains responsive even in dual-player mode.

B. Interaction Stability

Recognition Accuracy: Under standard lighting, the heuristic gesture recognition achieves a high success rate. The "Control Ring" mechanism (binding hands only within a specific radius of the snake head) effectively solves the ID switching problem common in multi-hand tracking.

Robustness: Rapid hand movements can cause tracking loss due to motion blur. We mitigated this by implementing a `HAND_LOSS_GRACE` period (2.0s), allowing the game to pause automatically rather than crash or lose control immediately.

C. Resource Overhead

Monitoring via `jtop` shows CPU load consistently above 70%, while GPU usage is around 35%. This reveals that the Python Global Interpreter Lock (GIL) and CPU-bound preprocessing are the main bottlenecks. The GPU accelerates the core inference, but the overall throughput is limited by the CPU's single-core performance.

V. CONCLUSION

This project verifies the feasibility of **directly applying** mature deep learning models to develop real-time interactive systems on edge devices.

1) **Summary**: Through efficient GStreamer pipeline design and geometric mapping logic, we successfully converted MediaPipe's perception capabilities into a smooth gaming experience.
2) **Contribution**: We designed a "Hybrid Rendering" pipeline and a "Geometric Mapping Strategy" that balances visual quality with performance on low-power devices.
3) **Future Work**: Future optimization could involve migrating core logic to C++ or using TensorRT to further offload CPU tasks to the GPU.

REFERENCES

[1] S. Mittal, "A survey on optimized implementation of deep learning models on the NVIDIA Jetson platform," *Journal of Systems Architecture*, vol. 97, pp. 428–442, 2019.

[2] F. Zhang *et al.*, "MediaPipe Hands: On-device real-time hand tracking," *arXiv preprint arXiv:2006.10214*, 2020.