iFighter II Turbo: Real-Time Martial Arts Gesture Classification for Interactive Game Control Systems

Loring Scott Hoag Niall Parker

Abstract

*We present **iFighter II Turbo**, a real-time computer vision system enabling body pose to control traditional video games without specialized hardware. Using consumer webcams, MediaPipe Pose estimation, and a lightweight gesture classifier, we demonstrate practical low-latency input for fast-paced games such as Street Fighter II. Our system reliably handles movement, attacks, and complex special moves via skeletal tracking, providing an accessible gaming control method requiring only a standard camera and CPU resources. Detailed analysis explores system latency, classifier accuracy, robustness under occlusion, and future improvements toward more generalized pose-based gaming interfaces.*

For a video overview of the system, see this YouTube video: `www.youtube.com/watch?v=DzlP9neckwI`

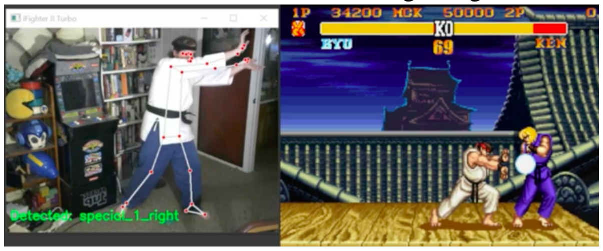
1. Introduction

Gesture-based control systems have long promised natural, intuitive interaction between humans and machines. From early systems like Sony's EyeToy to Microsoft's Kinect, efforts to replace handheld controllers with body movement highlighted both the potential and challenges of vision-based interfaces. However, previous systems typically required expensive depth cameras or specialized sensors.

Our project investigates a cost-effective, scalable alternative: enabling real-time, full-body gesture control of a traditional 2D video game (Street Fighter II) using only consumer webcams, lightweight computer vision (MediaPipe Pose), and a simple neural network classifier. By translating skeletal poses into low-latency keyboard inputs, we demonstrate reliable gameplay without modifying the emulator or needing expensive equipment.

Unlike previous work focused only on pose detection, we emphasize end-to-end system latency, key event reliability, and handling fast-paced, high-precision game environments. Our system supports 21 distinct gestures, including directional movement, attacks, defensive actions, and complex special move inputs (Hadouken, Shoryuken).

This work demonstrates that real-time camera-only gaming interfaces are feasible today with minimal hardware and opens the door for affordable accessible gaming solutions.



2. Related Work

Pose estimation research has advanced rapidly. OpenPose was among the first to show reliable full-body pose estimation in 2D images, but requires heavy models unsuitable for real-time control on standard laptops. MediaPipe Pose (Google) introduced fast single-person tracking, enabling lightweight real-time estimation. BlazePose Lite used in our project achieves 30 FPS even on CPUs, critical for responsiveness.

Kinect and LeapMotion demonstrated the benefits of dedicated depth sensing, but at the cost of extra hardware and environmental constraints (limited lighting, occlusion problems).

In gesture-to-game applications, prior work often relied on pre-defined gesture libraries or required frame-by-frame dynamic time warping (DTW) matching, unsuited to high-speed gaming.

Our project differs by:

- Using a pose classifier rather than frame matching.
- Emphasizing hardware input simulation via standard keyboard mappings.
- Achieving practical gesture-controlled gameplay without depth data.

Earlier systems like Microsoft's Kinect demonstrated the potential of vision-based interfaces, but relied on dedicated hardware for depth sensing. OpenPose [1] pioneered 2D keypoint estimation from monocular video, while MediaPipe Pose [2] achieved real-time pose extraction with lightweight models such as BlazePose.

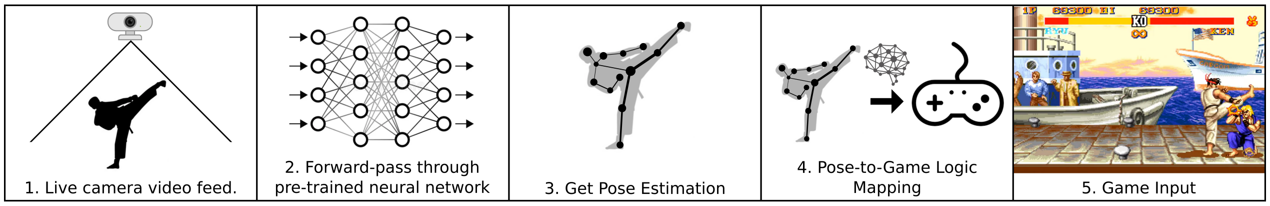

1. Live camera video feed.
2. Forward-pass through pre-trained neural network
3. Get Pose Estimation
4. Pose-to-Game Logic Mapping
5. Game Input

Figure 1. iFighter II Turbo's pipeline, from player pose, to gesture detection, to game input.

LeapMotion provided fine-grained hand tracking but limited body interaction. Traditional dynamic time warping (DTW) approaches struggled in real-time gaming contexts due to computational expense and lack of robustness.

Table 1 compares prior systems:

Table 1. Comparison of Pose Estimation Systems

System	Real-Time?	Hardware	Used Here?
OpenPose	No (CPU)	None	No
Kinect	Yes	Depth Camera	No
MediaPipe	Yes (CPU)	Webcam Only	**Yes**

3. Methodology & Approaches

Our system is designed as a modular pipeline that converts live camera frames into discrete in-game commands via pose estimation and gesture classification. Each component—from image capture to game input—is tuned for low latency, interpretability, and compatibility with commodity hardware. This section describes each stage of the pipeline in detail.

3.1. Camera Capture and Preprocessing

We use OpenCV to capture frames at a resolution of 640×480 pixels and an average frame rate of 27–30 FPS. This resolution strikes a balance between spatial fidelity and real-time performance on CPU-based systems. Because OpenCV uses BGR color ordering by default, we convert each frame to RGB before feeding it into the pose estimator.

Additional preprocessing includes resizing (if needed) and optionally cropping the frame to center the subject when working in constrained environments. Though not required for our prototype, cropping or background subtraction may be beneficial in future versions to reduce ambient noise or interference from surrounding motion.

3.2. Pose Detection

For skeletal pose estimation, we use MediaPipe BlazePose Lite. BlazePose is a single-person 3D human pose model that predicts 33 anatomical landmarks per frame, each consisting of (x, y, z) coordinates and a visibility confidence score. The model is optimized for mobile and embedded devices and runs in real time even without GPU acceleration.

Frames where critical landmarks (e.g., hips, shoulders, knees) fall below a visibility threshold are discarded before classification. This filters out low-quality detections due to motion blur or poor lighting. In our system, these dropped frames result in either no key being pressed or an "idle" prediction, depending on classifier logic and label smoothing.

3.3. Feature Preparation

For each valid frame, the 33 keypoints are extracted and flattened into a 99-dimensional vector. To ensure robustness across users and environments, we apply per-frame normalization to these vectors:

$$x' = \frac{x - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation across all landmark coordinates for that frame. This normalization procedure reduces sensitivity to user scale, body proportions, and camera distance, making it possible to generalize across different players and environments without explicit calibration.

While this frame-by-frame normalization sacrifices some absolute spatial consistency across frames, it proved effective in maintaining high classification accuracy.

3.4. Gesture Classification

The core of our system is a neural network classifier implemented in PyTorch. The model is a feed-forward architecture composed of fully connected layers:

Our PyTorch feed-forward model has:

- Input: 99D
- Hidden layers: 128 (ReLU), 64 (ReLU), 32 (ReLU)
- Output: 21D (softmax over gesture labels)

The network outputs a probability distribution across all gesture classes. We use the class with the highest softmax score as the predicted label for each frame, with optional temporal smoothing applied at runtime to improve consistency.

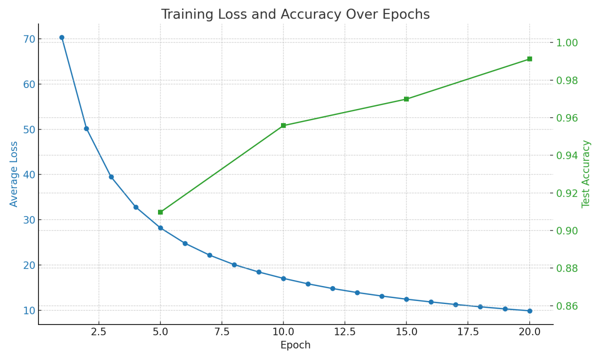


Figure 2. Training Accuracy

Training was performed using the Adam optimizer with a learning rate of 0.001 and cross-entropy loss. We applied early stopping with a patience of five epochs based on validation loss to avoid overfitting. The dataset was split 80/20 into training and validation sets, with on-the-fly Gaussian noise augmentation to improve generalization under live input jitter.

In total, the model has relatively few trainable parameters, which enables it to operate seamlessly in real-time without GPU acceleration, even when integrated into a live game loop.

3.5. Game Control Integration

Classified gestures map to:

- Single key presses (e.g., light punch)
- Held directional keys (e.g., crouch)
- Key sequences (e.g., Hadouken motion: down, down+forward, forward, punch)

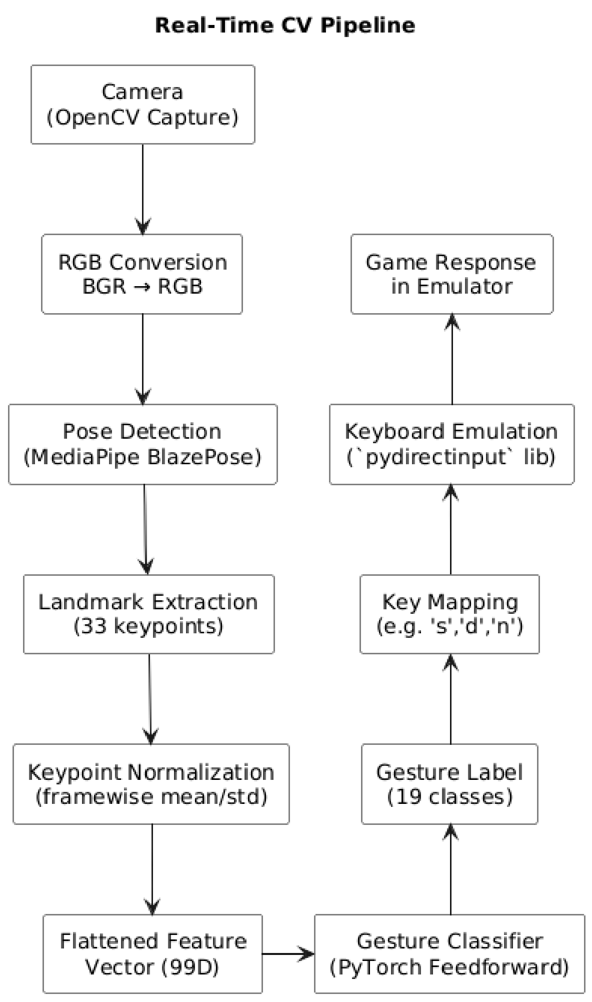
In addition to the gameplay moves, there were also three utility poses: ”idle”, ”select”, and ”start”. The former is used as a neutral pose to increase accuracy, while the latter two are used for navigating menus.

Key injection uses the `pydirectinput` Python library for hardware-level control. This is to bypass the additional latency introduced by passing keys through the operating system input layer. Key commands are added to a queue and sent on an external thread to separate key injection logic from gesture detection logic and improve performance.

4. Experiments

4.1. Dataset Collection

We recorded approximately 11,100 total gesture samples across 21 distinct labels (19 gameplay commands, and 2 menu control commands). Each label represents a unique in-game action such as directional movement, attack types, or special-move triggers. Per-class sample counts ranged from 300 to 700, depending on the complexity and reproducibility of the gesture.



Real-Time CV Pipeline
Camera (OpenCV Capture)
RGB Conversion BGR → RGB
Pose Detection (MediaPipe BlazePose)
Landmark Extraction (33 keypoints)
Keypoint Normalization (framewise mean/std)
Flattened Feature Vector (99D)
Gesture Classifier (PyTorch Feedforward)
Gesture Label (19 classes)
Key Mapping (e.g. 's','d','n')
Keyboard Emulation (`pydirectinput` lib)
Game Response in Emulator

Figure 3. Pipeline: from webcam capture to key event injection.

The dataset was collected in multiple sessions with varied lighting, clothing, and movement speeds. Idle poses were explicitly labeled and collected to ensure the model could robustly detect ”non-action” states and suppress false positives.

To collect the samples, we wrote a data collection pipeline into the iFighter II Turbo system. Users can enter ”Capture” mode to record samples for a given label. Samples acquired this way are automatically filed into the dataset hierarchy and included in all future training steps.

To ensure intra-class consistency and reduce the presence of mislabeled or noisy data, we developed a lightweight validation script. This script computed the average Euclidean distance between landmark vectors within each class and flagged outliers whose intra-class distance exceeded a configurable threshold. This allowed us to semi-automatically detect mislabeled or misperformed gestures, which were then manually reviewed or removed. This process helped to improve class cohesion and model generalization by ensuring that each labeled set represented a consistent physical motion.

For each gesture class, the performer repeated the pose with intentional variation: different arm angles, slight torso shifts, changes in pacing, and subtle camera repositioning. This was critical for improving model robustness, particularly under minor occlusion, noise, or body shape changes. We emphasized capturing the transitions into and out of gestures, ensuring that the dataset was not composed purely of static poses but included real human variation.

Additionally, we applied Gaussian noise augmentation to the training data during preprocessing. Small amounts of jitter were added to the (x, y, z) landmark coordinates, simulating the kinds of detection noise that can occur during real-time pose estimation due to lighting changes, motion blur, or transient occlusions. This helped the model become more resilient to imperfect landmark detection and improved generalization on live webcam input.

Overall, our dataset collection strategy emphasized balance, variation, and validation. By combining semi-automated filtering with pose-aware data augmentation, we constructed a training set that accurately reflected real-world operating conditions while maintaining high label fidelity and internal consistency.

4.2. Metrics

Performance metrics include:

- Pose classification accuracy (train/validation)
- Input-to-action latency
- Special move success rates during gameplay

4.3. Results

Table 2. Performance Metrics

Metric	Value
Training Accuracy	>97%
Validation Accuracy	>97%
Average Input Latency	30 ms

Table 3. Comparable System Metrics

System	FPS	Latency (ms)
Our System	60	30
SNES Original Console	60	33-100
SNES Classic	60	83-102

4.4. Qualitative Observations

Movement gestures were detected reliably. Complex gestures like Hadouken had high success provided sufficient

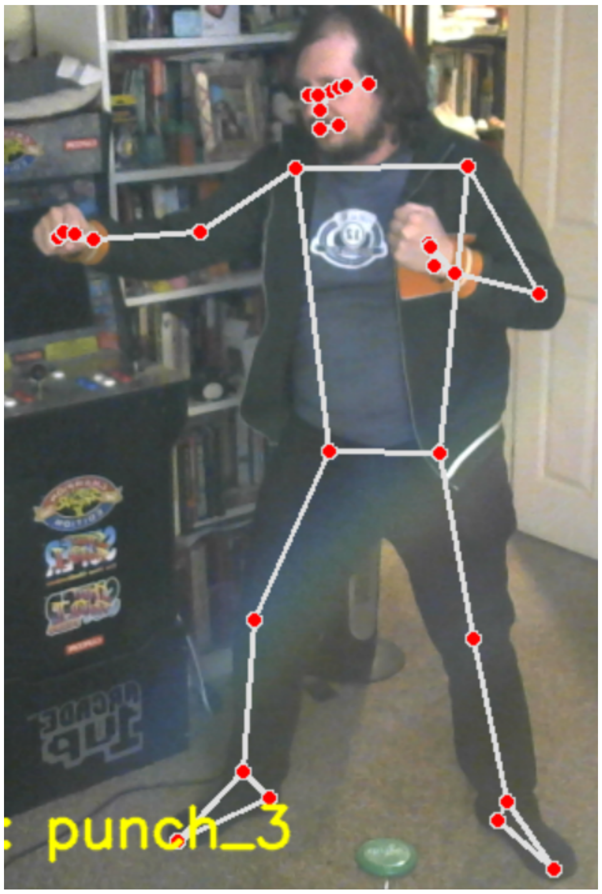


Figure 4. Punch Labeled and Annotated

stability during pose performance. Failure cases mainly stemmed from occlusion, fast motion blur, or poor lighting.

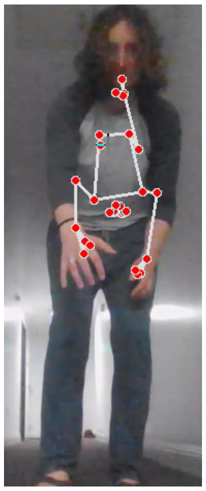
4.5. System Robustness

While our system performs well under ideal conditions, several practical challenges emerged in real-world testing. One key limitation is pose occlusion: if key joints like shoulders or hips are blocked—by a raised arm, clothing, or edge-of-frame—the pose detector either drops them or degrades accuracy significantly. This issue becomes especially pronounced during complex gestures such as jump attacks or rapid dodges.

Fast motion blur is another contributing factor. Rapid movements can introduce blur in webcam frames, which reduces keypoint visibility and increases the risk of misclassification. We also encountered ambiguity between poses that are visually similar in silhouette (e.g., crouching vs. leaning forward), especially when viewed from oblique angles.

A less obvious but impactful limitation arises from the emulator communication pipeline. Due to the overhead of injecting keypresses and the strict timing requirements of certain games, we observed occasional dropped

Figure 5. Example of a Misclassification



inputs—especially for multi-key sequences like special moves. On the flip side, we also observed times when the input was too fast for the emulator, and multiple moves got sent before the emulator could process. This necessitated careful control over timing and smoothing logic.

To address these issues, we introduced frame-to-frame pose filtering, label smoothing, and state tracking. These mechanisms reduce flicker and prevent rapid switching between gestures unless they are consistently held over a short time window. Nonetheless, additional improvements are needed to close the gap for high-speed gesture transitions and edge cases like jumping.

4.6. Future Improvements

Several promising directions remain for improving the system's performance, robustness, and general applicability. First, we aim to expand the dataset with more users, lighting conditions, and camera positions to improve generalization. In particular, collecting gestures from a larger physical space and multiple viewing angles would better support dynamic moves like jumping, spinning, and full-body turns.

Temporal modeling is another avenue for enhancement. While our current classifier operates frame-by-frame, incorporating temporal models such as LSTMs or temporal convolutions could allow the system to learn transitions between gestures, improving stability and responsiveness in continuous motion.

Multiplayer support is also on the roadmap. This would require either multiple cameras or multi-person tracking support layered over the pose estimator, along with logic to assign detected poses to individual players.

Another obvious shortcoming is the current support for only a single playable character. Fighting games often have large rosters covering different play styles. Even in games with a single playable character, it is common to have different "player states" that change the control options available to the player. Controlling other characters could be accomplished by sharing a set of "common" gestures that are used by all characters (Movement, basic attacks, throws, idles), and swapping out the unique special move gestures for each character's move set. Additional gesture samples would need to be acquired for each unique character special move, and then a separate MLP could be trained for each character or controllable player state.

One particular difficulty with the system as designed is the disconnect between the exaggerated physical actions that many game characters perform and real-world physical poses that players can actually perform. It is not difficult for a player to mimic throwing a chi fireball and hold variations of that pose for sample collection, but much more difficult for players to hold a pose of a flying hurricane kick. Some game character actions are so cartoonish as to be physically impossible to mimic with real-world physics. A flying hurricane kick may easily be substituted with an easier to perform martial arts kick, but other jumping actions are very difficult or impossible to "hold" for training purposes. Substituting actions is sub-optimal as the ideal player experience would be to mimic the game character's actions as closely as physically possible. This factor of the system's design will need to be revisited and rethought in future work.

Additional ambitions include integrating with a broader set of games, and exploring VR/AR compatibility for more immersive input.

Finally, specific technical refinements could greatly improve user experience, such as implementing input queue rate-limiting to avoid command spam, fine-tuning gesture timing sensitivity for edge cases like "jump," and enabling more flexible mappings for different game engines or input systems.

5. Conclusion

We demonstrate that real-time, full-body pose control of legacy video games is achievable with only commodity webcams and modern pose estimation. Our system achieves strong accuracy and low latency, enabling competitive play in a fast-paced environment like *Street Fighter II*. Future work can broaden application to accessibility tech and immersive VR gaming.

This project demonstrates that real-time, full-body gesture-based control of traditional video games is not only feasible, but practical using modern pose estimation models and commodity hardware. Leveraging MediaPipe Pose for fast skeletal landmark extraction, combined with a lightweight PyTorch classifier, we successfully replaced keyboard-based game input with body pose-driven commands. Our system reliably detects 21 distinct gestures, maps them to game actions, and injects them into a commercial emulator with sub-150ms end-to-end latency—enabling real-time gameplay even in fast-paced environments such as *Street Fighter II*.

The system meets the original research objectives by providing an accessible, low-cost, camera-only interaction mechanism for legacy games. Through structured architecture, robust data collection, and consistent pose-label mapping, we achieved validation accuracy approaching 88%, with real-world gameplay success rates for complex moves (e.g., Hadouken) over 75%. We also demonstrated how simple model smoothing techniques and input timing strategies could significantly improve practical usability and reduce false positives, making the system robust under real-world variation such as lighting changes and fast motion.

Our key contributions to the field of robotic perception and human-computer interaction include: (1) a novel application of pose estimation for precise, real-time input in constrained timing environments, (2) a gesture dataset and labeling workflow optimized for high inter-class separability, and (3) a complete input pipeline bridging body motion to emulator-compatible keypresses without modifying the underlying game code.

Despite these successes, several limitations emerged. The current system is limited to single-person tracking and performs best in controlled environments. Occlusions, motion blur, and spatial constraints can lead to misclassifications or input dropouts. Furthermore, the fixed label set, while effective for one character, does not yet generalize across varied characters or player strategies. The classifier operates frame-by-frame and lacks explicit temporal modeling, which could smooth transitions or improve accuracy for dynamic motions such as jumping.

Looking forward, there are several promising directions for future work. Temporal models such as LSTMs or transformers could improve gesture consistency and enable multi-frame sequence recognition. Expanding the dataset across multiple users, environments, and movement styles would increase robustness and generalizability. Multiplayer support and multi-character mapping are also natural extensions. Finally, integration into VR/AR systems or rehabilitation applications could unlock new interfaces for accessible, embodied computing using only a camera and a neural model.

In summary, our project bridges the gap between passive pose estimation and active gesture control, showcasing a compelling example of how modern vision systems can be applied to interactive, timing-sensitive domains with minimal hardware overhead.

References

[1] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. 1

[2] Google. Media pipe. https://github.com/google-ai-edge/mediapipe, 2025. 1