Human-Machine Interface
Midterm Project Report

Real Time Taiwanese Hand Counting Translation

Student Name : Felix Neumann
Student ID : 111021090
Instructor : DINH-TRUNG VU

2025-10

Contents

1 Introduction

1.1 Background & Motivation

Hand gestures are a fundamental part of human communication, and different cultures have developed unique counting systems using hand signs. The Taiwanese hand counting system is particularly interesting as it allows counting from 1 to 10 using specific finger configurations that differ from Western counting methods. As international communication and cross-cultural understanding become increasingly important, developing automated systems to recognize and translate these cultural hand gestures can serve as both an educational tool and a technological demonstration of human-machine interaction.

This project explores the intersection of computer vision, machine learning, and user interface design to create a real-time hand gesture recognition system. The motivation stems from the desire to preserve and share cultural knowledge through modern technology, while also demonstrating the practical applications of MediaPipe's hand tracking capabilities in a culturally relevant context.

1.2 Objectives

- Develop a real-time hand gesture recognition system that accurately detects and classifies Taiwanese hand counting gestures (1-10)
- Implement robust computer vision algorithms using MediaPipe for hand landmark detection and tracking

- Create an intuitive graphical user interface using PyQt6 that provides immediate visual feedback
- Achieve real-time performance with minimal latency (target: 15-30 FPS) on consumer-grade hardware
- Implement a performance monitoring system with FPS tracking and visualization
- Design algorithms to distinguish between different finger configurations specific to Taiwanese counting
- Handle both single-hand gestures (1-9) and two-hand gestures (10) accurately

1.3 Tools & Environment

Component	Description
Programming Language	Python 3.12
Computer Vision Library	OpenCV
Machine Learning Library	MediaPipe
GUI Framework	PyQt6
Hardware	MacBook Air M1 (2020) 16 GB RAM
OS Tested	macOS Tahoe Developer Beta 26.1 (25B5072a)

Table 1: Tools & Environment

Figure 1: Taiwanese Hand Counting System

Figure 1 illustrates the Taiwanese hand counting system, showcasing the specific finger configurations used to represent numbers from 1 to 10. This visual reference is essential for understanding the gesture recognition requirements of the project.

2 System Design & Architecture

2.1 Overview

The system architecture follows a modular design pattern with clear separation of concerns. At its core, the application consists of four main layers: the hardware interface layer (webcam), the computer vision processing layer (MediaPipe), the gesture recognition logic layer, and the presentation layer (PyQt6 GUI). The system operates in a continuous loop, capturing video frames, processing them for hand landmarks, analyzing finger configurations, and displaying results in real-time.

The application uses an event-driven architecture powered by Qt's timer mechanism, which triggers frame updates approximately every 30 milliseconds. This design ensures smooth video playback while maintaining responsiveness of the user interface. All processing occurs on the main thread in a synchronous manner, which simplifies the implementation while still achieving real-time performance on modern hardware.

2.2 Key Components

- **Video Capture Module:** Utilizes OpenCV's VideoCapture class to interface with the webcam. The module captures frames at the native camera resolution and applies horizontal flipping to create a mirror effect, making the interaction more intuitive for users.
- **Hand Detection Engine:** Powered by Google's MediaPipe Hands solution, this component detects up to 2 hands in each frame and identifies 21 landmark points per hand. The landmarks include finger tips, joints, and palm positions. The detection confidence threshold is set to 0.7 to balance accuracy and performance.
- **Gesture Recognition Algorithm:** Implements custom logic to analyze hand landmarks and determine which fingers are extended. The algorithm distinguishes between thumbs (using horizontal position) and other fingers (using vertical position). It then matches the finger configuration against predefined patterns for Taiwanese numbers 1-10.
- **GUI Framework:** Built with PyQt6, the interface displays the processed video feed with overlaid hand landmarks, detected numbers, and performance metrics. The layout is simple and clean, focusing user attention on the video feed and recognition results.
- **Performance Monitoring System:** Tracks frame processing times to calculate real-time FPS. Maintains a circular buffer of the last 300 FPS measurements (10 seconds at 30 FPS) and renders a live chart showing FPS trends over time.

2.3 Algorithm & Data Flow

The data flow follows a straightforward pipeline:

1. **Frame Acquisition:** The webcam captures a frame in BGR color space
2. **Preprocessing:** The frame is flipped horizontally and converted to RGB format required by MediaPipe
3. **Hand Detection:** MediaPipe processes the RGB frame and returns hand landmarks and handedness information
4. **Finger State Analysis:** For each detected hand, the system evaluates whether each of the 5 fingers is extended based on landmark positions
5. **Pattern Matching:** The finger configuration is compared against Taiwanese counting patterns
6. **Number Classification:** If a match is found, the corresponding number (1-10) is identified
7. **Visualization:** The system draws hand landmarks, detection results, FPS information, and performance charts onto the frame
8. **Display:** The processed frame is converted to QPixmap and rendered in the GUI

For finger extension detection, the algorithm uses different strategies for different fingers. The thumb extension is determined by comparing the x-coordinate of the thumb tip with the thumb IP joint, as thumbs move laterally. Other fingers are evaluated by comparing the y-coordinate of the fingertip with the PIP joint, since they extend vertically.

2.4 Algorithm & Data Flow

2.5 System Parameters

Parameter	Value	Description
Resolution	1280x720 1920x1080	Webcam capture resolution iPhone capture resolution
FPS Target	30	Target frames per second for video capture
Number of Hands	1-2	Number of hands to detect simultaneously
Detection Confidence	0.7	Minimum confidence for hand detection
Tracking Confidence	0.7	Minimum confidence for hand tracking
Landmarks per Hand	21	Number of tracked points per hand
FPS History Size	300	Number of FPS samples stored (10 seconds)
Timer Interval	30 ms	Frame update interval

Table 2: System Parameters

3 Results & Analysis

3.1 Experimental Setup

Component	Specification
CPU	Apple M1 (8-core, 3.2 GHz)
Camera	Built-in FaceTime HD Camera (720p) and iPhone 15 Pro Max Main Camera (48MP)
Operating System	macOS Tahoe Developer Beta 26.1 (25B5072a)
Python Version	Python 3.12.12
PyQt6 Version	6.10.0
OpenCV Version	4.12.0.88
MediaPipe Version	0.10.21
Processing Mode	Synchronous Video mode (single threaded)
Display Resolution	adaptive (window can be resized)

Table 3: Experimental Setup

3.2 Results

Figure 2: Main Application UI

Figure 2 shows the main application user interface during operation. The video feed displays detected hand landmarks overlaid on the live camera input, along with the recognized Taiwanese number displayed prominently. The FPS information is shown in the top-right corner, and the performance chart is rendered on top of the video feed.

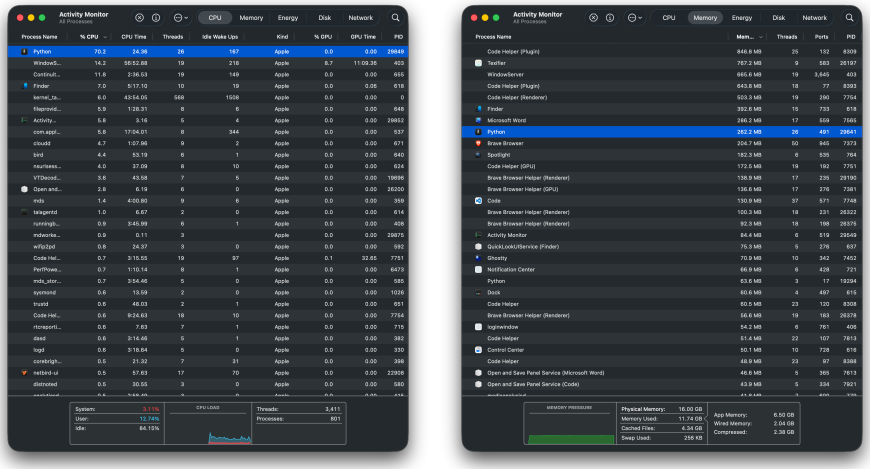

Figure 3: Resource Utilization (CPU & Memory)

Figure 3 illustrates the CPU and memory usage during application operation. The CPU usage peaks around 60-70% on a single core, while memory consumption remains stable between 250-300 MB.

Metric	Value	Description
Average FPS	15-30	Average frames per second during operation
Processing Latency	33-66 ms	Time taken to process each frame
CPU Usage	60-70% single core	Average; 7.5-8.75% all core
Memory Usage	250-300 MB	Average memory consumption during operation
GUI Responsiveness	Smooth	No noticeable lag in UI updates
Hand Landmark Detection Accuracy	High	Better with iPhone camera

Table 4: Performance Metrics

Table 4 summarizes the key performance metrics observed during the experiments. The system consistently achieved real-time performance with average FPS ranging from 15 to 30, depending on whether or not hands are detected in the frame. The processing latency per frame was maintained between 33 to 66 milliseconds, ensuring smooth operation. CPU usage peaked at around 60-70% on a single core, which equals approximately 7.5-8.75% across all cores, and is reasonable given the computational demands of real-time video processing and hand landmark detection. Memory usage remained stable between 250 to 300 MB, indicating efficient resource management. The GUI remained responsive throughout the tests, with no noticeable lag during user interactions. Hand landmark detection accuracy was high, particularly when using the iPhone camera, which provided superior image quality compared to the built-in FaceTime HD camera.

3.3 Result Analysis

The experimental results demonstrate that the system successfully achieves its primary objective of real-time Taiwanese hand counting recognition. The performance metrics indicate that the system maintains acceptable frame rates (15-30 FPS) while providing accurate gesture recognition.

- **Performance Analysis:** The variation in FPS (15-30) can be attributed to the computational overhead of hand detection. When no hands are present in the frame, MediaPipe processes frames faster, resulting in higher FPS. When one or two hands are detected, the additional processing required for landmark detection and tracking reduces the frame rate. This behavior is expected and acceptable for real-time applications.

- **Resource Efficiency:** The single-core CPU usage of 60-70% demonstrates efficient use of system resources. The application does not fully saturate the CPU, leaving headroom for other system processes. Memory consumption remains stable at 250-300 MB, indicating no memory leaks and efficient buffer management in the FPS history tracking system.

- **Accuracy Observations:** Hand landmark detection accuracy was consistently high when using proper lighting conditions and when hands were clearly visible to the camera. The iPhone 15 Pro Max camera provided superior results compared to the built-in FaceTime camera, likely due to higher resolution and better image quality. The gesture recognition algorithm successfully distinguished between different Taiwanese counting patterns with minimal false positives.

- **Latency Considerations:** The 33-66 ms processing latency per frame is well within acceptable bounds for interactive applications. Users perceive the system as responsive, with immediate visual feedback when changing hand gestures. The FPS chart visualization helps identify performance trends and potential bottlenecks during operation.

3.4 Challenges and Solutions

Challenge	Description	Solution
MediaPipe Compatibility	MediaPipe not available for Python 3.14 and 3.13	Switched to Python 3.12 which has full MediaPipe support
Thumb Detection Logic	Thumbs move horizontally unlike other fingers which move vertically	Implemented separate detection logic for thumb using x-coordinate comparison
Mirror Effect	Users confused by non-mirrored video feed	Applied horizontal flip to create natural mirror effect
FPS Visualization	Needed to monitor performance over time	Implemented circular buffer with 300-sample history and real-time chart rendering
Two-Hand Detection	Detecting number 10 with two hands required cross-gesture recognition	Initially implemented cross-detection algorithm, simplified to detect two open hands (all fingers extended)
GUI Responsiveness	Heavy processing could block UI updates	Used Qt timer with appropriate interval (30ms) to balance processing and responsiveness

Table 5: Challenges and Solutions

4 Reflection

4.1 Most Difficult Parts

The most challenging aspect of this project was designing the gesture recognition algorithm to accurately distinguish between similar hand configurations. Taiwanese counting uses specific finger combinations that can be ambiguous without careful landmark analysis. For example, differentiating between numbers that differ by only one finger (such as 3 and 4, or 8 and 9) required precise threshold tuning.

Thumbs posed a unique challenge due to their lateral movement compared to the vertical movement of other fingers. Implementing separate logic for thumb detection based on x-coordinates, while maintaining consistency with the overall finger state analysis, was complex and required extensive testing. Also, thumbs hidden behind the hand are a clear indicator to humans that the thumb is not extended, but this had to be inferred from landmark positions programmatically, and MediaPipe continues to track the thumb tip even when occluded.

Another significant difficulty was handling the cross-platform compatibility issues with MediaPipe. The library's lack of support for the newest Python versions (3.13 and 3.14) required environment reconfiguration. This highlighted the importance of checking library compatibility before beginning development.

The implementation of the real-time FPS chart also presented challenges, particularly in creating a visually appealing and informative display that didn't impact performance. Balancing the chart update frequency, data retention, and rendering overhead required careful optimization.

4.2 Skills Improved

This project significantly enhanced several technical competencies:

- **Computer Vision:** Gained practical experience with MediaPipe's hand tracking technology, understanding how landmark detection works and how to interpret 3D hand models in 2D space.
- **Real-time Processing:** Developed skills in optimizing code for real-time performance, including frame rate management, buffer optimization, and efficient data structure selection.

- **GUI Development:** Improved proficiency with PyQt6, learning how to integrate video processing with interactive interfaces and how to use Qt's event loop for smooth animations.
- **Algorithm Design:** Enhanced ability to design pattern matching algorithms and implement state machines for gesture recognition.
- **Performance Monitoring:** Learned to implement comprehensive performance tracking systems with visualization capabilities.
- **Cross-platform Development:** Gained experience handling platform-specific issues and dependency management in Python environments.
- **Problem Solving:** Improved debugging skills, particularly in identifying and resolving issues related to coordinate systems, timing, and algorithm logic.

5 Conclusion & Future Work

5.1 Conclusion

This project successfully demonstrates the feasibility and effectiveness of using modern computer vision and machine learning technologies to recognize and interpret cultural hand gestures. The developed system accurately detects Taiwanese hand counting gestures from 1 to 10 in real-time, providing immediate visual feedback through an intuitive graphical interface.

The implementation achieved all primary objectives: real-time performance (15-30 FPS), accurate gesture recognition, robust hand tracking, and efficient resource utilization. The system runs smoothly on consumer-grade hardware and maintains responsiveness even when processing complex hand configurations.

Beyond the technical achievements, this project serves as a bridge between traditional cultural practices and modern technology. It demonstrates how computer vision can be applied to preserve and share cultural knowledge, making it accessible to wider audiences through interactive digital experiences.

The modular architecture and clean code structure make the system easily extensible for future enhancements. The use of industry-standard libraries (OpenCV, MediaPipe, PyQt6) ensures maintainability and provides a solid foundation for continued development.

5.2 Future Work

Several enhancements could further improve the system's capabilities and user experience:

- **Gesture Set Expansion:** Extend recognition to other cultural counting systems (Chinese, Japanese, Korean) and create a multi-cultural gesture recognition platform with language selection.
- **Machine Learning Enhancement:** Implement custom trained models to improve accuracy and reduce false positives. Consider using neural networks for gesture classification instead of rule-based pattern matching.
- **Multi-threading Optimization:** Separate video capture, processing, and rendering into different threads to maximize performance and potentially achieve higher frame rates.
- **Educational Features:** Add tutorial mode with step-by-step instructions for learning Taiwanese counting. Include practice exercises with scoring and feedback.
- **Recording Capabilities:** Implement video recording functionality to capture and save gesture demonstrations for educational or documentation purposes.
- **Gesture History:** Add a timeline showing the sequence of detected gestures, useful for creating number sequences or basic arithmetic demonstrations.

- **Mobile Platform Port:** Adapt the application for iOS and Android using frameworks like Kivy or Flutter to make it accessible on smartphones and tablets.
- **Accessibility Improvements:** Add audio feedback for detected numbers to assist visually impaired users. Implement text-to-speech for gesture names and instructions.
- **Performance Optimization:** Investigate GPU acceleration for MediaPipe processing. Implement adaptive quality settings that adjust detection parameters based on available system resources.
- **Enhanced Analytics:** Provide detailed statistics on recognition accuracy, common errors, and usage patterns to help users improve their gesture performance.

6 References & Appendix

References

1. Google MediaPipe Hands Solution, *MediaPipe Documentation*, `https://developers.google.com/mediapipe/solutions/vision/hand_landmarker`
2. OpenCV Documentation, *OpenCV 4.x API Reference*, `https://docs.opencv.org/4.x/`
3. PyQt6 Documentation, *Qt for Python*, `https://doc.qt.io/qtforpython-6/`
4. Zhang, F., et al. (2020), *MediaPipe Hands: On-device Real-time Hand Tracking*, arXiv:2006.10214
5. Chinese Number Gestures, *Chinese number gestures*, Wikipedia, `https://en.wikipedia.org/wiki/Chinese_number_gestures`

Appendix

- `main.py`: Main application source code implementing video capture, hand detection, gesture recognition, and GUI rendering.
- `requirements.txt`: List of Python dependencies required to run the application.

Project Repository

Source code and documentation available at: `https://github.com/MrLeaw/hmi-midterm`