Final Project Report - Hand Gesture Recognition System Using Raspberry Pi Camera and STM32

R13522802 陳冠呈

B12901130 劉劭毅

I. Introduction

This project aims to design and implement a real-time, embedded vision-based recognition system to enable touchless human–computer interaction. The system utilizes a Raspberry Pi equipped with a camera module to capture images of hand gestures, which are then transmitted to an STM32 microcontroller, where a dedicated convolutional neural network (CNN) performs gesture classification. Lastly, to demonstrate the practical utility, we developed a Python-based GUI that emulates a gesture-controlled elevator system. The integration of image processing, embedded systems, and serial communication explores a robust approach to modern IoT and smart home applications.

II. Motivation

While powerful single-board computers such as Raspberry Pi can easily handle small computer vision tasks, implementing these on microcontrollers requires a careful integration of image processing, real-time communication, and machine learning.

By designing an architecture consisting of a Raspberry Pi and an STM32 microcontroller, we aim to explore the separation of tasks:

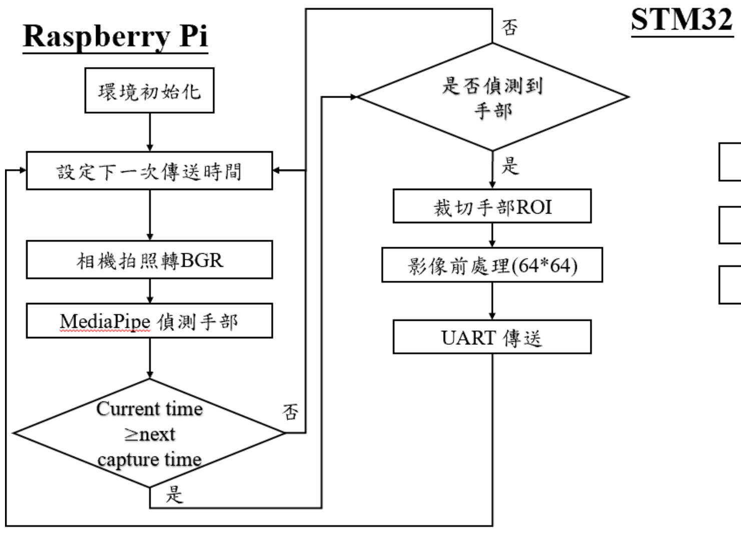
- High-level processing: Handling the heavy lifting of image capture and complex landmark extraction.
- Real-time execution: Running a neural network on a resource-constrained device to prove the feasibility.

Ultimately, this project serves as a practical exploration of how modern software tools can be bridged with industrial-standard microcontrollers to create a seamless user experience.

III. Method

The system is a multi-stage pipeline that transitions from raw optical data to a classified gesture output. Three parts of the workflow will be discussed:
1. image preprocessing
2. image transmission
3. model inference

Diagram: The flowchart of the system



List of tools used

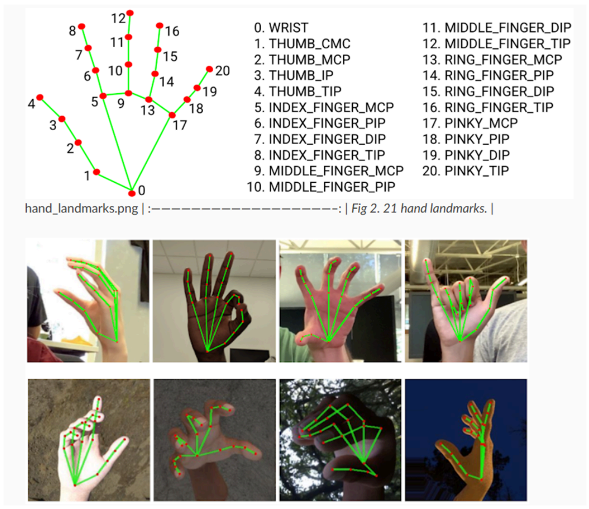
Hardware	Software
● STM32 board (B-L4S5I-IOT01A) ● Raspberry Pi 4 ● Raspberry Pi Camera V2	● MediaPipe (Raspberry Pi) ● OpenCV (Raspberry Pi) ● TensorFlow Lite (STM32) ● STM32Cube.AI (STM32) ● PySide6 (Laptop)

1. Image Preprocessing and Feature Extraction (Raspberry Pi)

The Raspberry Pi captures images of 640x480 pixels every 10 seconds. However, feeding a raw, high-resolution image directly into a microcontroller's neural network is computationally demanding and thus undesirable due to memory limits. To solve this, we implement Google's MediaPipe Hand Landmarker to crop out the hand gesture in the image.

The multi-stage image preprocessing pipeline:

1. **MediaPipe:** It first identifies the bounding box of at most one hand within the larger image frame. It operates with a pre-trained machine learning model to locate 21 hand-knuckles. The bounding box defines the region of interest for our model. If no hand is detected, then the following steps are skipped, and the program waits for the next image.



Source: MediaPipe

2. **Cropping:** the bounding box is cropped out for further processing. As a result, at a later stage, the CNN will only receive the image of the hand as its input.
3. **Grayscale conversion:** The image is converted into grayscale (uint8) and is compressed into an image of 64x64 pixels.
4. **Data normalization:** The uint8 (0~255) data type is converted to float32 (0, 1) for improving the performance of the CNN. The image is now ready to be sent to STM32.

2. Data Transmission

We utilize UART for communication between the Raspberry Pi and STM32, and also between STM32 and the computer. This choice provides a stable, point-to-point connection that is easy to debug and sufficient for transmitting the image data required for gesture recognition.

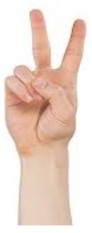
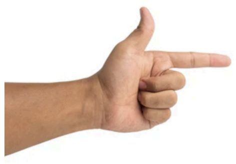
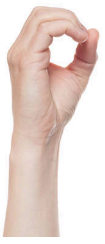
3. Neural Network Inference

The STM32 runs a lightweight CNN optimized for embedded deployment. By receiving the compressed grayscale image rather than the raw pixels, the input layer of the neural network is significantly reduced in size. We have observed that the model performance remains when the input is in grayscale.

The model architecture consists of:
• Convolutional layers with stride and pooling
• Global Average Pooling
• Fully connected (dense) output layer with Softmax

Layer	Type	Explanation
Input	Input	64x64 grayscale
1	Conv2D + Pool	8 filters, stride=2
2	Conv2D + Pool	16 filters
3	Conv2D + Pool	32 filters
Head	GAP + Dense	3-Class output

Due to the limited memory of the STM32 board, we have constrained the number of the outputs of the CNN to three gesture classes, which we label: K, L, O. The CNN will output the likelihood of the input image belonging to each class.

K	L	O
		

The model was trained using the TensorFlow framework and Kaggle dataset. The model is then transformed into an TensorFlow Lite format, which optimizes the memory saving and compute efficiency of the neural network by pruning, weight clustering, and inter-layer buffering, at the cost of compromising the performance.

TensorFlow Lite conversion is compatible with STM32Cube.AI, a toolset for converting a pre-trained ML model into optimized C code for STM microcontrollers.

IV. Results

The performance of the gesture recognition system was evaluated based on its accuracy and real-time responsiveness. While the integration was successful, we identified two primary bottlenecks: the extraction accuracy of the MediaPipe framework and the classification accuracy of the CNN hosted on the STM32.

We observed that thermal management is essential. Without active cooling, thermal throttling severely impairs the camera's frame rate and the detection ability of MediaPipe software. However, under stable thermal conditions, MediaPipe almost never fails in identifying the 21 hand-knuckles, providing a reliable data stream for the subsequent neural network inference.

The CNN implemented on the STM32 board was tested empirically to determine its effectiveness in a real-world setting.

- **Classification Accuracy:** The model achieves an approximate accuracy of 90% (9 out of 10 correct classifications), with most cases achieving 99% likelihood of the correct class, demonstrating that the simplified architecture is sufficient for distinguishing between distinct hand gestures.
- **System Latency:** The real-time latency of the entire workflow (image capture, MediaPipe preprocessing, UART transmission, and CNN inference) is at most 1 second.

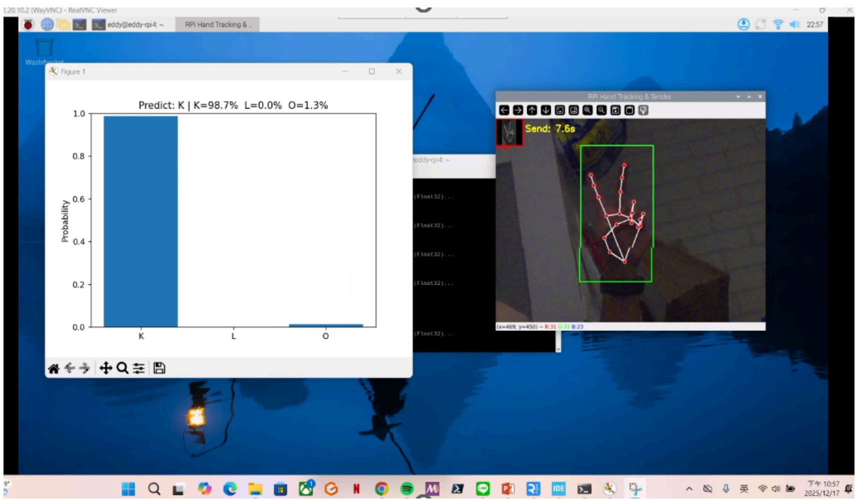


Figure 1
Predict: K | K=98.7% L=0.0% O=1.3%
Probability
1.0
0.8
0.6
0.4
0.2
0.0
K
L
O
RPi Hand Tracking & Sender
Send: 7.6s

Image: captured image & CNN output

References

1. What is VNC? https://deskin.io/zh/resource/blog/what-is-vnc
2. MediaPipe example https://mediapipe.readthedocs.io/en/latest/solutions/hands.html
3. Mediapipe 手掌偵測 https://ithelp.ithome.com.tw/m/articles/10299964
4. Hand landmarks detection guide
https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker?hl=zh-tw
5. Install MediaPipe on a Raspberry Pi – Example Gesture Recognition
https://randomnerdtutorials.com/install-mediapipe-raspberry-pi/
6. GitHub Repository for Sign Language to Speech: Unvoiced
https://www.kaggle.com/datasets/grassknoted/asl-alphabet

Appendices

Software Versions

Component / Package	Version	Notes
Raspberry Pi OS	2025/11/24 (Libcamera-based)	Operating system used on Raspberry Pi
Python	3.11.2	Main programming language
Picamera2	0.3.3	CSI camera driver library (Libcamera-based)
MediaPipe	0.10.18	Core hand detection and landmark library
OpenCV-Python	4.12.0.88	Image processing and visualization
NumPy	1.25.0	Numerical computation and array processing
PySerial	Latest stable	UART communication with STM32
Protobuf	4.25.3	Required dependency for MediaPipe
absl-py	Latest stable	Core dependency for MediaPipe
matplotlib	Latest stable	Used for visualization and debugging
attrs	Latest stable	Dependency for MediaPipe object handling

Issues and Solutions

問題階段	核心錯誤訊息	根本原因	最終解決方案
I. 影像讀取失敗	cv2.VideoCapture(0) 無法開啟	Raspberry Pi OS 從 V4L2 轉換到 Libcamera，導致 OpenCV 無法使用舊接口。	採用 picamera2 函式庫，直接使用原生 Libcamera 驅動。
II. 驅動程式啟用	rpicam-still 或 libcamera-still 找不到	較新 RPi OS 採用新命名，但需確認底層硬體是否正常。	確認 rpicam-still 可成功拍照，證明 CSI 攝影機硬體正常。
III. Libcamera 導入	ModuleNotFoundError: No module named 'libcamera'	虛擬環境 (myenv) 的 Python 3.11 無法找到系統 apt 安裝的 python3-libcamera 模組。	手動複製 系統 /usr/lib/python3/dist-packages/libcamera 資料夾到虛擬環境的 site-packages 中。
IV. 顯示環境衝突	ModuleNotFoundError: No module named 'pykms' / qt.qpa.xcb: could not connect to display	SSH 連線缺乏圖形化介面，picamera2 預設嘗試使用高效能的 DRM 預覽。	1. 修改 picamera2 原始碼，移除對 DrmPreview 的導入和引用。 2. 在 RPi 實體螢幕上運行 程式碼（或在 SSH 中註釋 cv2.imshow）。
V. 版本兼容性	numpy<2 vs numpy 2.3.5 衝突	mediapipe 和 opencv 的舊版本與新版 numpy 2.x 不兼容。	將 Numpy 版本降級 到 1.25.0 滿足所有庫的要求。

PySide6 GUI App

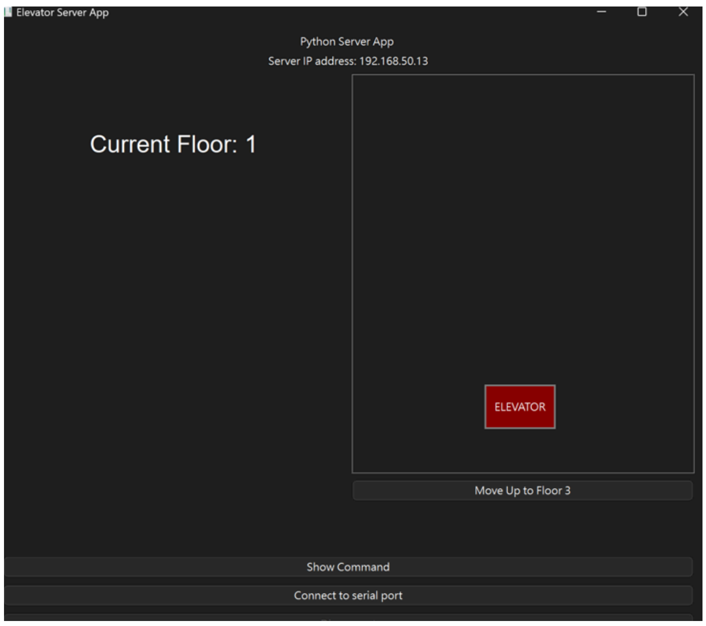


Image: PySide6 GUI