Pakistan Institute of Engineering and Applied Sciences (PIEAS)

RTOS Based Human-Machine Interface (HMI) System with NLP, Manual Joystick Control and Camera Feedback

Course: Real Time Embedded Systems
Instructor: M. Aqil

Submitted by:

Anum Iqbal	BS-22-IB-108041
Alisha Hasnain	BS-22-IB-108145
Satwat Rahman	BS-22-FB-2104585

Date of Submission: December 11, 2025

Abstract

This report presents the design and implementation of a Human Machine Interface system developed as a Problem Based Learning project for the course Real Time Embedded Systems. The system enables real time interaction with an embedded controller through manual joystick input, offline voice commands, and live camera feedback.

The design emphasizes real time behavior, concurrent task execution, and predictable communication latency. An ESP32 WROOM DevKit V1 is used as the primary embedded controller and communication node. A Python based graphical user interface manages multiple tasks using multithreading to ensure non blocking and responsive operation.

Wireless communication between the host system and the ESP32 is implemented over Wi Fi, providing reliable performance suitable for soft real time systems. The results demonstrate that a multi modal HMI can be effectively implemented using accessible hardware while adhering to real time embedded system design principles.

Contents

Chapter 1

Introduction

In today's rapidly advancing technological landscape, the development of efficient and intuitive Human-Machine Interfaces (HMI) has become a crucial area of research and innovation. HMIs serve as the bridge between human operators and machines, enabling seamless interaction, monitoring, and control. This project focuses on designing and imple menting a multi-modal HMI system that integrates voice commands and joystick-based manual control to interact with a robotic system in real time.

The core objective of this project is to enable a user to control a robotic system through two distinct modes: Natural Language Process ing (NLP) and Manual Joystick Input, both of which operate concur rently or can be selected based on user preference. The voice command module utilizes an offline speech recognition engine (Vosk) to recognize keywords like "start" and "stop", ensuring control even in environments without internet access. The manual control mode captures joystick movements on the PC and transmits the data wirelessly to the robot using an ESP32 microcontroller over Wi-Fi. This data is then used to control the robot's behavior, such as starting or stopping a servo motor, based on joystick activity.

A Python-based Graphical User Interface (GUI) developed using Tk inter provides a user-friendly dashboard that displays real-time feed back from both the joystick and voice input. The GUI allows seamless switching between the NLP and Manual modes, ensuring a flexible and interactive experience. The objective of this project is to apply real time embedded design principles such as task separation, concurrency, and latency management to a practical HMI system.

Chapter 2

System Architecture

The proposed Human-Machine Interface (HMI) system is structured around a modular architecture that integrates input acquisition, processing, communication, and feedback mechanisms. The system is de signed to provide seamless interaction between the user and a robotic platform through two control modes that are Voice Command (NLP) and Manual Joystick Input, with continuous real-time feedback dis played via a GUI.

2.1 Input Layer

- **Joystick Module (Manual Mode):**
  - A USB joystick is connected to the PC and monitored using the Pygame library in Python.
  - The X and Y axis values are captured in real time and represent directional control or movement intent.
  - These values are sent over Wi-Fi to the ESP32 microcontroller using a socket-based client-server model.
- **Voice Command Module (NLP Mode):**
  - A microphone connected to the PC captures the user's speech input.
  - The Vosk offline speech recognition engine processes the audio and extracts commands such as "start" and "stop."
  - Recognized commands are printed and sent to the control logic to determine actions.

2.2 Control Layer

- **Graphical User Interface (GUI):**
  - Built using Tkinter, the GUI allows the user to:
    - Select between NLP and Manual modes.
    - View real-time joystick data in Manual mode.
    - Display the recognized voice commands in NLP mode.

  - The GUI runs both voice recognition and joystick monitoring in separate threads to ensure simultaneous input handling.

- **ESP32 Wi-Fi Communication:**
  - The ESP32 acts as a server listening for data from the PC.
  - In Manual mode, it receives joystick data over Wi-Fi and can forward it to other devices (e.g., STM32).
  - Data is parsed and interpreted for appropriate control actions (e.g., moving a servo motor when $X \neq 0$ or $Y \neq 0$).

2.3 Output Layer

- **Microcontroller Interface (STM32 or ESP32-controlled Device):**
  - The received joystick values are used to determine motor behavior.
  - For example, if both X and Y values are 0, the motor stops; otherwise, it runs.
  - Communication between ESP32 and STM32 is achieved using UART (wired serial connection).

- **Real-Time Feedback:**
  - All control signals (joystick values or NLP commands) are displayed on the GUI.
  - This ensures the user is informed of the system's current state and responses.

2.4 Communication Channels

- **Wi-Fi Communication:**
  - Socket programming is used to transmit joystick data from the PC to the ESP32.
  - Ensures low-latency, real-time data transfer without the limitation of a physical serial port.

Chapter 3

Global Configuration

This block sets up core resources such as IP addresses, sample rates, and queues.

```
ESP32_IP = "http://192.168.10.12"
model_path = "voice/vosk_model"
sample_rate = 16000
q = queue.Queue()
cap = None
joystick = None
```

Listing 3.1: Global Variables

3.1 Camera Mode

The system uses OpenCV to fetch the video stream from a mobile phone over Wi-Fi.

```
def start_camera_feed(text_area):
    url = 'http://192.168.10.8:8080/video'
    cap = cv2.VideoCapture(url)

    while True:
        ret, frame = cap.read()
        if not ret:
            print("Failed to grab frame")
            break
        cv2.imshow("Mobile Camera", frame)

        if cv2.waitKey(1) == ord('q'):
            break

        if text_area:
            text_area.delete(1.0, tk.END)
            text_area.insert(tk.END, "Camera Mode Active...\n")

    cap.release()
    cv2.destroyAllWindows()
```

Listing 3.2: Start Camera Feed

Explanation: The video stream from a mobile device is continuously displayed. The GUI is also updated to reflect the active mode.

3.2 Voice Mode (NLP)

Voice commands such as “start” or “stop” are processed offline using Vosk.

```
def callback(indata, frames, time, status):
    if status:
        print("Status:", status)
    q.put(bytes(indata))

def start_nlp(text_area):
    model = Model(model_path)
    recognizer = KaldiRecognizer(model, sample_rate)

    def listen():
        print(" Listening... Say 'start', 'stop', etc.")
        with sd.RawInputStream(samplerate=sample_rate, blocksize
           =8000,
                               dtype='int16', channels=1, callback
                                  =callback):
            while True:
                data = q.get()
                if recognizer.AcceptWaveform(data):
                    result = json.loads(recognizer.Result())
                    text = result.get("text", "")
                    if text:
                        print(f" You said: {text}")
                        text_area.delete(1.0, tk.END)
                        text_area.insert(tk.END, f"Voice Command:
                           {text}\n")

    listen()
```

Listing 3.3: Start NLP (Voice Recognition)

Explanation: This component continuously listens to microphone input and updates the GUI with recognized commands.

3.3 Manual Mode (Joystick)

Joystick readings are captured and sent to the ESP32 to control actuators.

```
def start_joystick_data(text_area):
    pygame.init()
    pygame.joystick.init()

    if pygame.joystick.get_count() == 0:
        print("No joystick detected.")
        exit()
```

```
8
9       joystick = pygame.joystick.Joystick(0)
10      joystick.init()
11      print("Joystick connected:", joystick.get_name())
12
13      while True:
14          pygame.event.pump()
15          x_axis = joystick.get_axis(0)
16          y_axis = joystick.get_axis(1)
17
18          try:
19              response = requests.get(f"{ESP32_IP}/joystick",
20                                      params={"x": round(x_axis, 2),
21                                              "y": round(y_axis, 2)
                                                 })
22              print(response.text)
23              if text_area:
24                  text_area.delete(1.0, tk.END)
25                  text_area.insert(tk.END,
26                      f"Joystick Data: X={round(x_axis, 2)} Y={round
                           (y_axis, 2)}\n")
27          except Exception as e:
28              print("Error:", e)
29
30          time.sleep(1)
```

Listing 3.4: Start Joystick Data Stream

Explanation: Joystick axis values are sampled, formatted, and sent over HTTP GET to the ESP32. These values control motor operations.

3.4 Graphical User Interface (GUI)

```
1   class App:
2       def __init__(self, root):
3           self.root = root
4           self.root.title("Robot HMI Interface")
5           self.root.geometry("800x600")
6           self.root.config(bg="#f0f0f0")
7
8           self.frame = ttk.Frame(self.root, padding=20)
9           self.frame.pack(fill='both', expand=True)
10
11          self.mode = tk.StringVar(value="Manual")
12
13          self.nlp_button = ttk.Button(self.frame, text="NLP Mode",
14                                       command=self.set_nlp_mode)
15          self.manual_button = ttk.Button(self.frame, text="Manual
                Mode",
16                                          command=self.
                                               set_manual_mode)
```

```
17          self.camera_button = ttk.Button(self.frame, text="Camera
               Mode",
18                                          command=self.
                                               set_camera_mode)
19
20          self.nlp_button.grid(row=0, column=0, padx=20, pady=10)
21          self.manual_button.grid(row=0, column=1, padx=20, pady=10)
22          self.camera_button.grid(row=0, column=2, padx=20, pady=10)
23
24          self.text_area = tk.Text(self.frame, height=10, width=80,
               bg="#e6e6e6")
25          self.text_area.grid(row=1, column=0, columnspan=3, padx
               =20, pady=20)
26
27          self.camera_thread = None
28          self.nlp_thread = None
29          self.joystick_thread = None
```

Listing 3.5: Tkinter GUI Application

3.5 Mode Switching and Threading

```
1  def set_nlp_mode(self):
2      self.terminate_previous_mode()
3      self.mode.set("NLP")
4      self.text_area.insert(tk.END, "NLP Mode Activated...\n")
5      self.nlp_thread = threading.Thread(target=start_nlp,
6                                         args=(self.text_area,),
                                              daemon=True)
7      self.nlp_thread.start()
8
9  def set_manual_mode(self):
10     self.terminate_previous_mode()
11     self.mode.set("Manual")
12     self.text_area.insert(tk.END, "Manual Mode Activated...\n")
13     self.joystick_thread = threading.Thread(target=
           start_joystick_data,
14                                             args=(self.text_area,)
                                                   , daemon=True)
15     self.joystick_thread.start()
16
17 def set_camera_mode(self):
18     self.terminate_previous_mode()
19     self.mode.set("Camera")
20     self.text_area.insert(tk.END, "Camera Mode Activated...\n")
21     self.camera_thread = threading.Thread(target=self.
           start_camera_feed_thread,
22                                           daemon=True)
23     self.camera_thread.start()
```

Listing 3.6: Mode Handlers

3.6 Main Execution

```
if __name__ == "__main__":
    root = tk.Tk()
    app = App(root)
    root.mainloop()
```

Listing 3.7: Main Function

Chapter 4

Hardware Implementation

4.1 Joystick Module (Input Device)

A USB joystick is connected to the PC, where it is interfaced using Python's Pygame library. The joystick serves as the manual control device and provides two-dimensional input (X and Y axes). The analog values from the joystick are interpreted in software and used to generate control signals.

4.2 ESP32 Microcontroller (Communication Node)

The ESP32 module functions as a Wi-Fi-enabled communication node. It operates as a server that continuously listens for incoming data from the PC over a local network socket. The ESP32 receives joystick data and, if needed, forwards it via UART to the STM32 board for further motor control.

4.3 Power Supply and Connections

- The servo motor is powered using a 5V external source (e.g., charger or regulated supply).
- ESP32 is powered via USB or 3.3V LDO regulator.
- STM32 is powered via micro-USB or VIN pin from a 5V source.
- All modules share a common ground to ensure proper communication.

4.4 Optional: Microphone for NLP (Voice Mode)

Although not part of the initial hardware test, a microphone is connected to the PC for voice command processing using the Vosk offline speech recognition library. Recognized commands such as "start" or "stop" are interpreted by the software for future integration into motor control logic.

Chapter 5

Hardware Results

This section presents the hardware-level validation of the proposed Human-Machine Interface (HMI) system, covering all three operational modes: Manual (Joystick), Voice Command (NLP), and Camera-based Monitoring. The interface was successfully implemented using Python, with GUI developed using Tkinter, real-time data transmission via Wi Fi to the ESP32.

5.1 Graphical User Interface (GUI)

The GUI serves as the central control panel for selecting between different operating modes. It was developed using Tkinter in Python and provides real-time feedback for joystick movement, voice commands, and camera monitoring. Mode selection buttons dynamically change the interface content. The GUI remained responsive under concurrent execution of all operational modes. Mode switching occurred without noticeable delay.

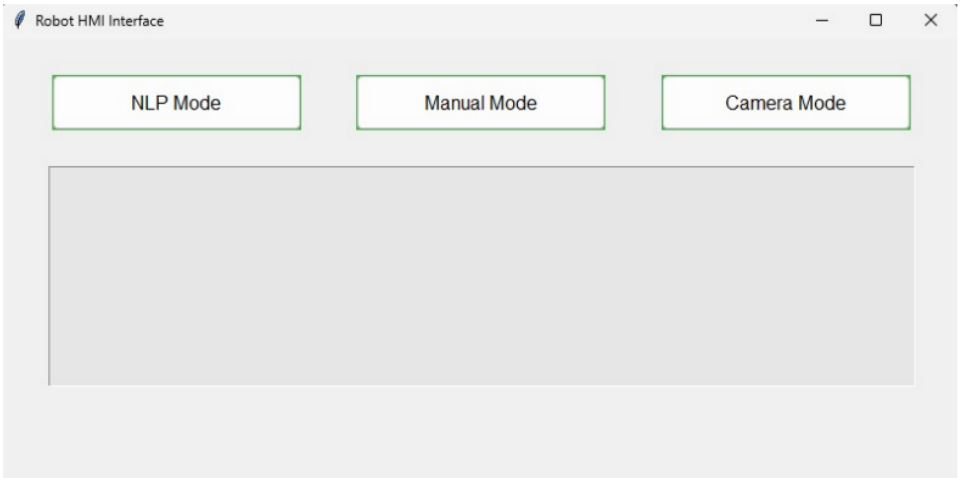

Figure 5.1: Main GUI interface with options for Manual, NLP, and Camera modes

5.2 Manual Mode (Joystick Control)

In Manual Mode, joystick values (X, Y) are read using the Pygame library and sent to the ESP32 over Wi-Fi. The ESP32 then transmits this data to the STM32 to control the servo motor based on joystick activity. When the joystick is idle (X = 0,Y = 0), the

motor stops; otherwise, it activates. Joystick data was transmitted to the ESP32 with minimal latency. The ESP32 processed commands consistently, demonstrating reliable soft real time behavior.

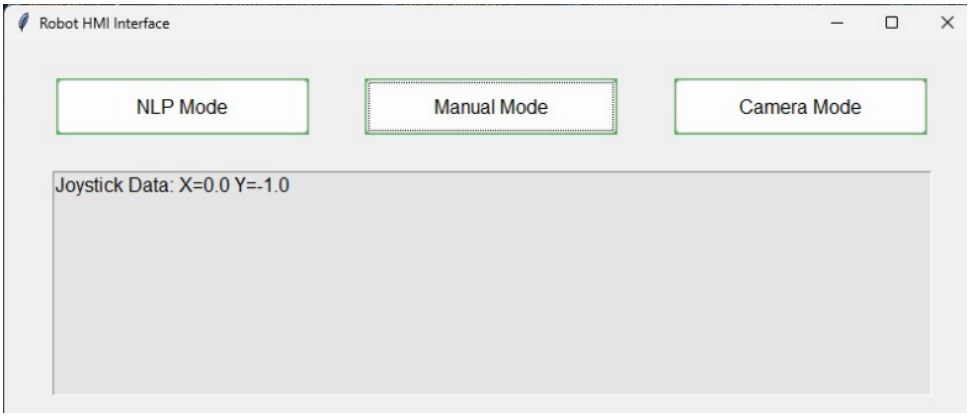

Figure 5.2: Manual Mode: Joystick values received from ESP32 and displayed on GUI

5.3 Voice Command Mode (NLP)

In Voice Mode, voice commands such as "start", "stop", "left", or "right" are processed using the offline Vosk speech recognition model. Recognized commands are displayed live in the GUI and are intended to control future robot actions. Voice commands were recognized and displayed with low response time. Offline processing ensured stable performance.

Figure 5.3: Voice Mode: NLP command recognized using Vosk and shown in GUI

5.4 Camera Mode (Live Monitoring)

Live video streaming operated continuously without affecting control tasks.The camera mode continuously receives video feed using a mobile phone camera via the IP Webcam app. OpenCV processes the feed in real time, which can be extended for future object or robot tracking appli cations. The feed is always active, regardless of mode selection. Figure 5.4: Camera Mode: Real-time video feed displayed alongside GUI

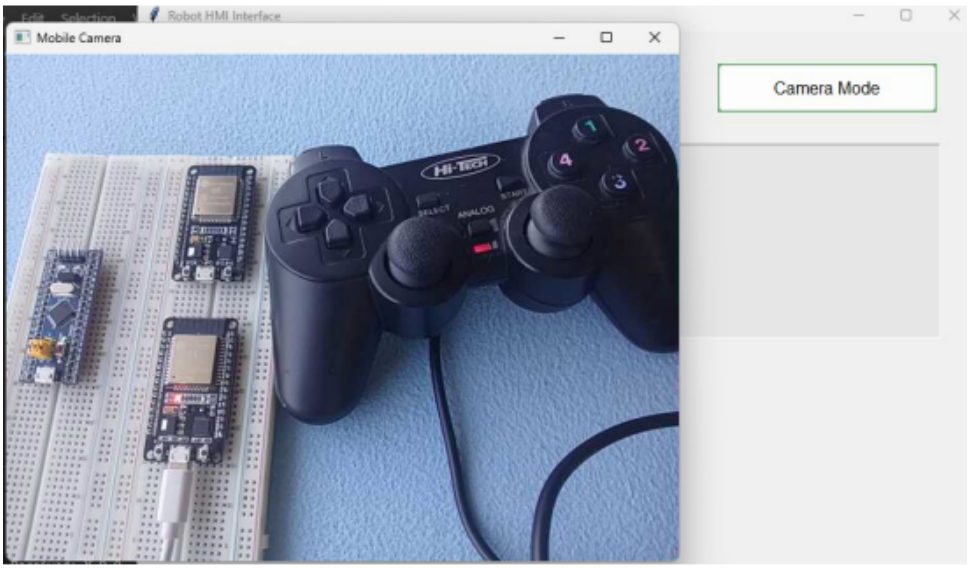

Figure 5.4: Camera Mode: Real-time video feed displayed alongside GUI

Chapter 6

Discussion

The project focused on designing a Human Machine Interface system that operates reliably under real time constraints. The system maintained responsiveness while handling multiple concurrent tasks. The implementation of a multi-modal Human-Machine Interface (HMI) demonstrated the feasibility and flexibility of integrating voice com mands, joystick control, and visual feedback into a single system. Each module functioned cohesively, forming an efficient communication bridge between the user and the robotic system.

Task separation using multithreading proved to be effective. Each input mode operated independently, preventing blocking behavior and improving system stability. The Manual Mode enabled real-time control using a joystick, with positional data transmitted via Wi-Fi from the PC to the ESP32, and then to the ESP32 for motor actuation for future use. This proves that real time manual operation can be achieved effectively over wireless communication, given careful handling of latency and data formatting.

Wireless communication introduced variable latency, but careful data handling ensured delays remained within acceptable limits for soft real time operation. In Voice Mode, the offline speech recognition engine (Vosk) accurately interpreted predefined commands such as “start”, “stop”, and “left”. These commands were captured with minimal delay and displayed immediately on the GUI. Although voice commands were not yet directly linked to actuator control, the underlying structure enables future extension with minimal changes. The choice of offline NLP en sured privacy and reliability even in the absence of internet connectivity. Offline voice processing improved reliability by removing network dependency. Commands were recognized promptly during testing.

The Camera Mode, based on a mobile phone’s IP webcam stream, remained continuously active in the background, offering real-time video surveillance. Although no image processing or object tracking was im plemented in this phase, the foundation has been laid for future inte gration of computer vision techniques such as ArUco marker tracking or obstacle detection. The central GUI served as a seamless interface, allowing users to switch between control modes and visualize incoming data effectively. The system’s modularity allowed each mode to operate independently while still coexisting in a single framework. The graphical user interface functioned as an effective supervisory layer, allowing smooth mode transitions and real time feedback.

Using the ESP32 as the sole controller simplified the system and reduced commu-

nication overhead. The controller consistently handled real time communication tasks. Despite successful implementation, certain limitations were identified:

- Servo motor actuation was not fully automated in Voice Mode during this phase.
- There is potential to optimize GUI responsiveness and background thread handling.

Overall, the project validated the use of accessible hardware and Python-based software to create a functional and extensible HMI platform capable of supporting multiple interaction paradigms and the project demonstrated practical application of real time embedded system principles.

6.1 Conclusion

conclusion This HMI system demonstrates a flexible, real-time robotic control mechanism supporting voice, joystick, and camera-based inputs. This report presented the design and implementation of a real time Human Machine Interface system using the ESP32 WROOM DevKit V1. The system demonstrated effective task scheduling, predictable communication, and reliable performance. The project validates the suitability of ESP32 based systems for real time HMI applications and provides a foundation for future enhancements involving tighter timing constraints or direct actuator control. It is scalable for future integration of machine learning and real-time feed back protocols such as MQTT

Bibliography

[1] Vosk: https://alphacephei.com/vosk

[2] OpenCV: https://docs.opencv.org/

[3] Pygame: https://www.pygame.org/

[4] Tkinter: https://docs.python.org/3/library/tkinter.html

[5] ESP32 Docs: https://docs.espressif.com/