Development of a HIWIN Robotic Arm with Multimodal (Speech, Gesture, and Stereo Vision) Control

Chi-Chun Chang
Advisor: Prof. Yin-Tien Wang
Department of Mechanical and Electro-Mechanical Engineering
Tamkang University

Abstract

This study aims to explore how visual and speech technologies, including stereo vision for depth estimation, can be utilized to operate a HIWIN robotic arm. The combination of vision, stereo depth perception, and speech capabilities allows the robotic arm to better perceive its surroundings in 3D and adapt to different scenarios and applications. By mimicking human operation, the robotic arm becomes more flexible and precise. Furthermore, integrating these technologies enables the robot to better understand and respond to spoken and gestural commands, thereby facilitating more natural human-robot interaction. We investigate methods to fuse vision, stereo depth, and speech information to achieve comprehensive and accurate environmental perception and object recognition. We also examine how this information can be used to precisely control the robotic arm for performing fine and complex tasks. Additionally, we explore how the robot can continually learn and adapt to new environments and tasks via audiovisual perception to enable autonomous intelligent operations. Finally, we aim to develop intuitive, natural, and effective human-robot interfaces that allow the robotic arm to understand user intentions and execute commands accordingly. Through this research, we aim to enhance the robotic arm's perceptual and operational capabilities, ultimately developing a more advanced and user-friendly system with broad application potential.

1 Research Motivation and Problem Statement

In specialized tasks or disaster response scenarios, frequent tool switching and reliance on human assistance to remove obstacles are common. However, such human intervention is prone to misjudgment, inefficiency, and fatigue, which may lead to accidents. Therefore, we explore whether robotic arms controlled by speech and enhanced with stereo vision can replace manual handling of objects. In this project, we develop a HIWIN robotic arm system that integrates voice, gesture, and stereo vision recognition. By combining gesture, voice input, and depth estimation, the robot can move to the specified location and grasp the object for the user. This multimodal system enhances usability and flexibility. For example, pointing to the right and saying “20cm” enables the robotic arm to operate according to the user’s command with accurate depth perception. This study reduces the risk of human error and is particularly useful for repetitive, labor-intensive tasks. Experimental results will verify the feasibility of our proposed method and demonstrate the practical potential of voice-vision collaborative human-robot interaction.

2 Literature Review

Disaster prevention and mitigation rely heavily on technological advancements. A variety of “black tech” products have proven effective in emergency rescue operations. In March 2013, CCTV introduced a rescue-specific robotic arm capable of demolishing reinforced concrete and performing delicate operations. This dual-arm rescue robot, known as a “dual-power intelligent dual-arm rescue vehicle,” comes in different models (20-ton, 40-ton, and 60-ton variants). The 20-ton model can lift 8 tons per arm and reach up to 8 meters. Various end-effector tools can be installed to perform rotation, demolition, cutting, and separation tasks. The arms are modeled after human limbs, allowing them to work cooperatively to execute complex tasks. The system supports dual-power operation (fuel and electric) and dual-mode control (manual or remote), making it one of the few high-tonnage intelligent rescue systems in the world. Robotic arms are not limited to simple, repetitive tasks like picking objects from a known location. Traditionally, robots struggled to identify specific objects among many, determine precise positions with tolerances, or adapt grasping based on object orientation. Today, with advancements like Intel RealSense high-resolution depth cameras, powerful CPUs and GPUs, and AI tools like OpenVINO, robotic arms have gained perceptual intelligence. Enhanced with machine vision, stereo depth estimation, and AI, these robots can detect, classify, and manipulate surrounding objects with greater precision, stability, and safety. As machine vision, AI, and networking technologies evolve, robotic arms can now perceive, analyze, and react to their environments in real time. This also enables data feedback for facility and business management systems. One promising application is predictive maintenance, which reduces costs and increases uptime by performing data analysis at the edge or in the cloud. Mobility is a key trend for robotics in the 2020s. The integration of sensors, actuators, embedded vision, AI, and edge computing is making mobile robots more versatile. Mobile robotics is growing rapidly in fields such as manufacturing, healthcare, and home care. In factories, mobile platforms carrying robotic arms can optimize material transport and packaging while enhancing manufacturing flexibility.

3 Methodology

3.1 Gesture Recognition using MediaPipe

MediaPipe is an open-source, cross-platform framework developed by Google in 2019 to simplify visual processing tasks. It uses a modular pipeline architecture, with components for pose estimation, hand tracking, face detection, and gesture recognition. Modules can be used independently or combined for complete visual workflows. MediaPipe offers:

- Cross-platform support (desktop, mobile, embedded systems)
- High performance through GPU/TPU acceleration
- Easy-to-use APIs with rich documentation and examples
- Flexible modular design
- Open-source community support

3.2 Stereo Hand Depth Estimation

This subsystem demonstrates a 3D depth estimation system for hand joints using stereo vision and MediaPipe Hands. Two cameras (built-in + USB) are used to capture synchronized images of the user's hands. By computing pixel distances between joint pairs across left and right frames, the system calculates disparity and estimates depth (Z-axis) values for each finger. Stereo camera calibration is performed using OpenCV's `stereoCalibrate()` function. An ArUco chessboard pattern is used to extract corner correspondences across two views. The calibration process solves for intrinsic and extrinsic parameters, including the rotation matrix (R), translation vector (T), essential matrix (E), and fundamental matrix (F). The system detects hand landmarks using MediaPipe in both left and right camera frames. It focuses on computing the pixel distance between the 1st and 2nd joints of each finger (e.g., Thumb joint 2 $\rightarrow$ 3). After estimating disparity, a calibrated linear depth correction formula is applied:

$$\text{depth}_{actual} = 0.018 \cdot \left(\frac{f \cdot B}{\text{disparity}} \right) + 0.25$$

where f is focal length in pixels, B is the baseline (15 cm), and disparity is the difference in pixel distance between joint pairs from left and right views.

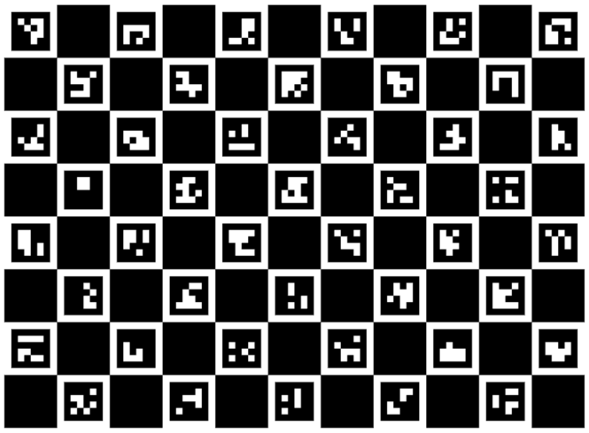


Figure 1: ArUco chessboard used for stereo calibration

3.3 Speech Recognition

We developed a language learning and translation tool using Python. The core functionality of converting speech to text was implemented using Python's SpeechRecognition library, with processing via Google's Cloud API. The system also supports text-to-speech output. Future implementations may incorporate NLP (Natural Language Processing) to handle more flexible sentence structures and improve understanding. NLP combines computer science, AI, and linguistics to allow computers to parse, understand, and generate human language.

3.4 HIWIN Robotic Arm

We used the HIWIN RA610-1476-GB six-axis robotic arm. The system uses Cartesian coordinate frames, including robot, base, and tool coordinates. The tool frame can be shifted and rotated based on the gripper. The maximum operational radius is 1476mm. We use `C++` to call the Python program and utilize `HRSDK_V2.1.7` to achieve control of the HIWIN robotic arm.

圖1-1 RA610-1476-GB機械手臂

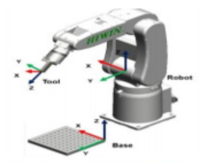


圖1-2 機械手臂相關坐標系定義

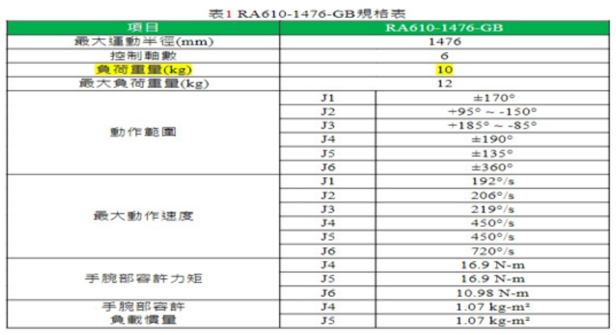


表1 RA610-1476-GB規格表

項目	RA610-1476-GB	
最大運動半徑(mm)	1476	
控制軸數	6	
負荷重量(kg)	10	
最大負荷重量(kg)	12	
動作範圍	J1	±170°
	J2	+95° ~ -150°
	J3	+185° ~ -85°
	J4	±190°
	J5	±135°
	J6	±360°
最大動作速度	J1	192°/s
	J2	206°/s
	J3	219°/s
	J4	450°/s
	J5	450°/s
	J6	720°/s
手腕部容許力矩	J4	16.9 N-m
	J5	16.9 N-m
	J6	10.98 N-m
手腕部容許負載慣量	J4	1.07 kg-m²
	J5	1.07 kg-m²

Figure 2: HRSDK_V2.1.7 is a development kit for the SSE Robot Arm, used for programmatic integration and development. It supports C++, C#, and VB, allowing users to write custom software to control the robot.

3.5 Machine Learning

Solving problems using machine learning typically involves:

1. Define the Problem
2. Build the Dataset
3. Train the Model
4. Evaluate the Model

5. Use the Model

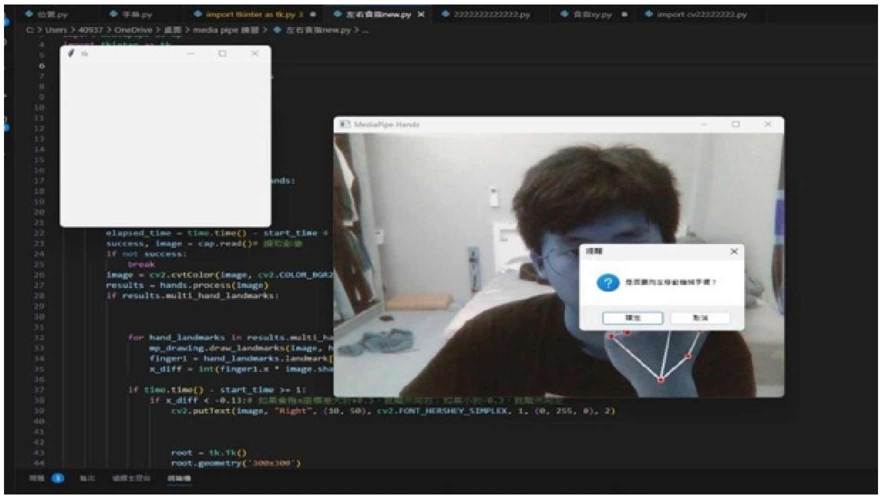
Each step includes:

- **Defining the problem:** Clarify the task and choose between supervised, unsupervised, or reinforcement learning.
- **Building the dataset:** Collect, inspect, and statistically analyze data to ensure quality.
- **Training the model:** Use training and testing sets to iteratively minimize the loss function.
- **Evaluating the model:** Tune hyperparameters with a validation set and monitor for overfitting.
- **Using the model:** Retrain on full data and evaluate with a final test set.

4 Results and Conclusion

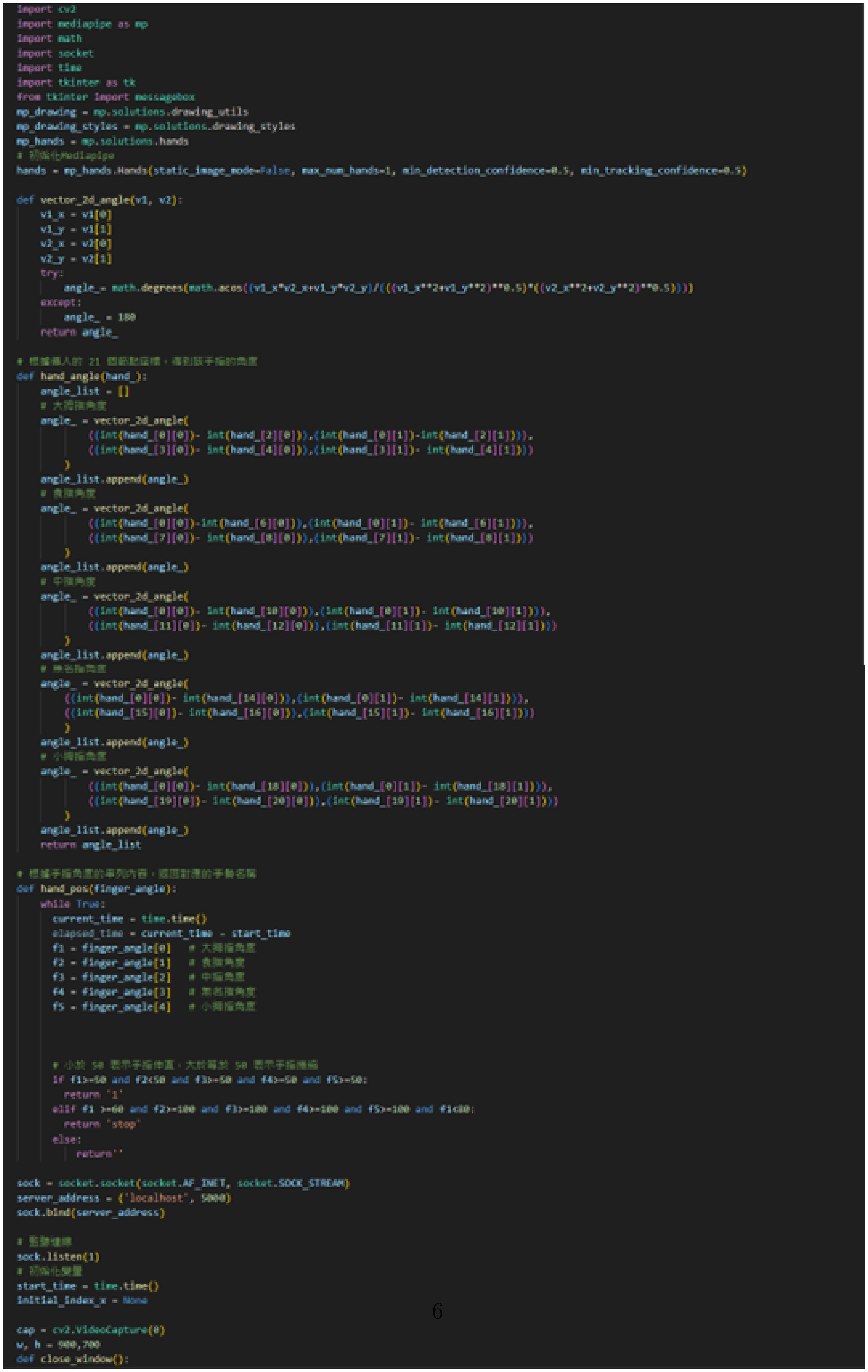
4.1 Gesture Recognition

We used MediaPipe Holistic for gesture recognition. It includes models for full-body pose, face landmarks, and hand tracking. In total, it detects 543 keypoints: 33 pose, 468 facial, and 21 per hand, all with x, y, z coordinates for 3D application. We calculated finger angles using `vector_2d_angle` and `hand_angle` functions. Gestures were determined based on joint positions and timing differences. A warning dialog was displayed before motion commands to ensure safety.

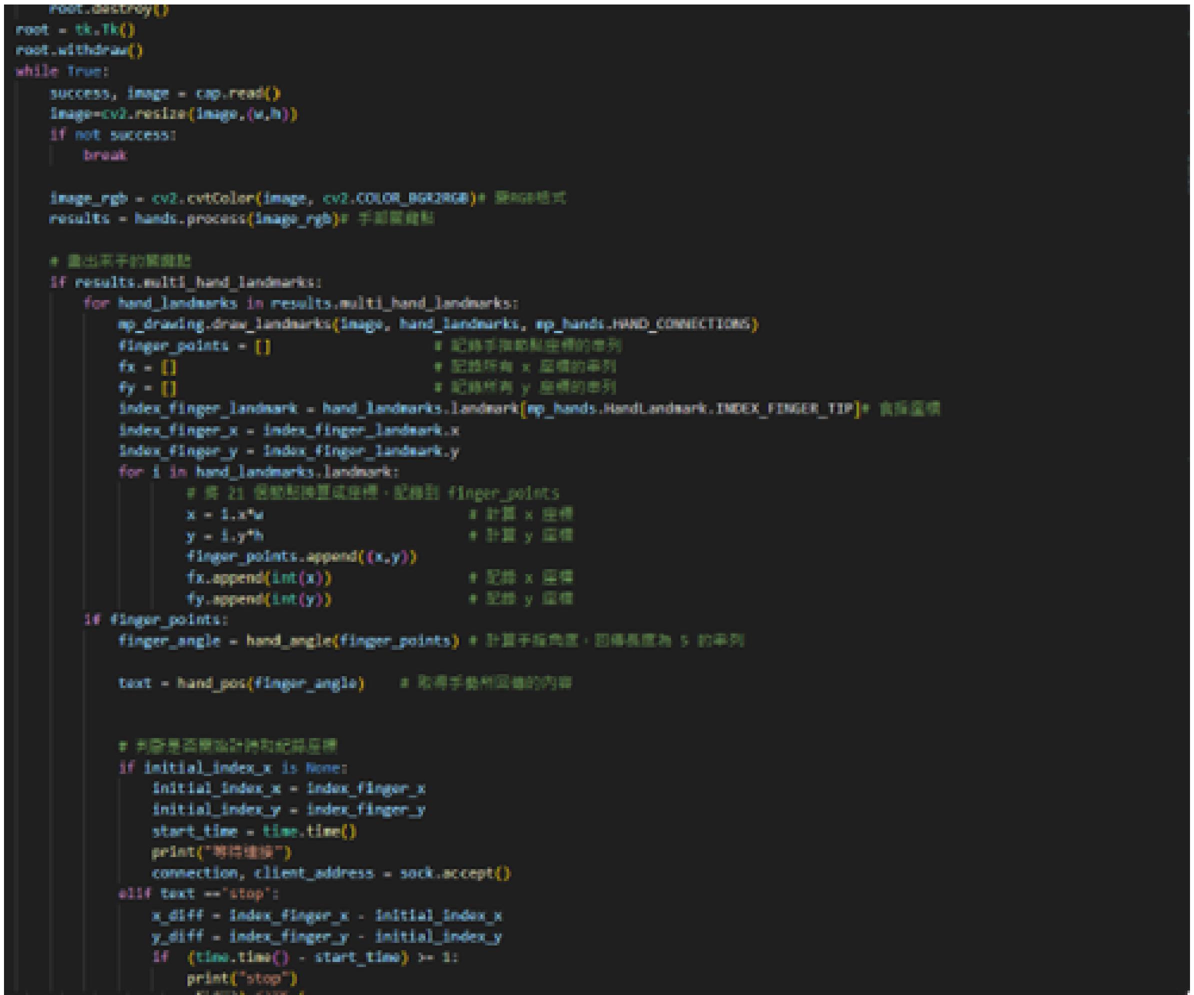


```
        elapsed_time = time.time() - start_time #
        success, image = cap.read()# [illegible]
        if not success:
            break
        image = cv2.cvtColor(image, cv2.COLOR_BGR2
        results = hands.process(image)
        if results.multi_hand_landmarks:


            for hand_landmarks in results.multi_ha
                mp_drawing.draw_landmarks(image, h
                finger1 = hand_landmarks.landmark[
                x_diff = int(finger1.x * image.sha

            if time.time() - start_time >= 1:
                if x_diff < -0.13:# [illegible]
                    cv2.putText(image, "Right", (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)


                    root = tk.Tk()
                    root.geometry('300x300')
```

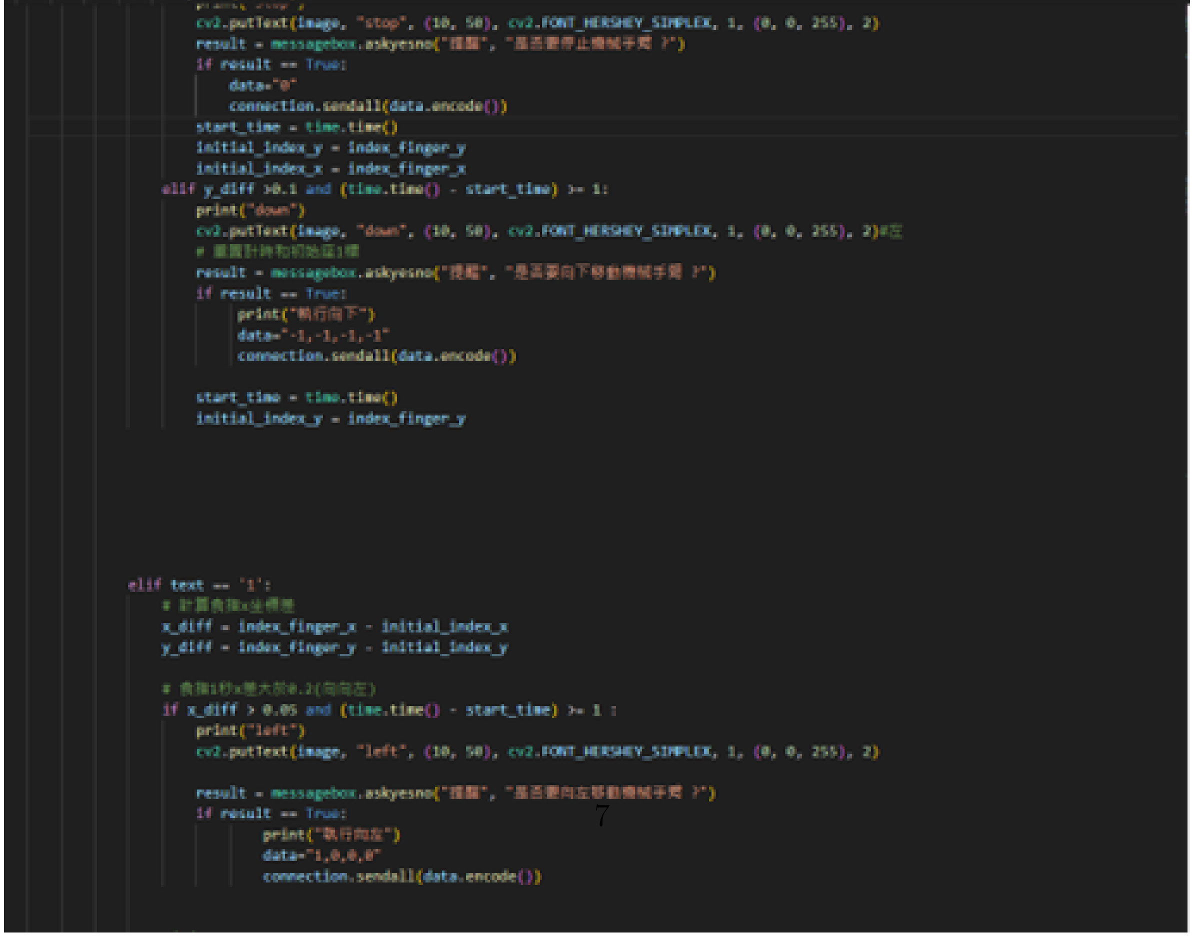
Figure 3: Warning



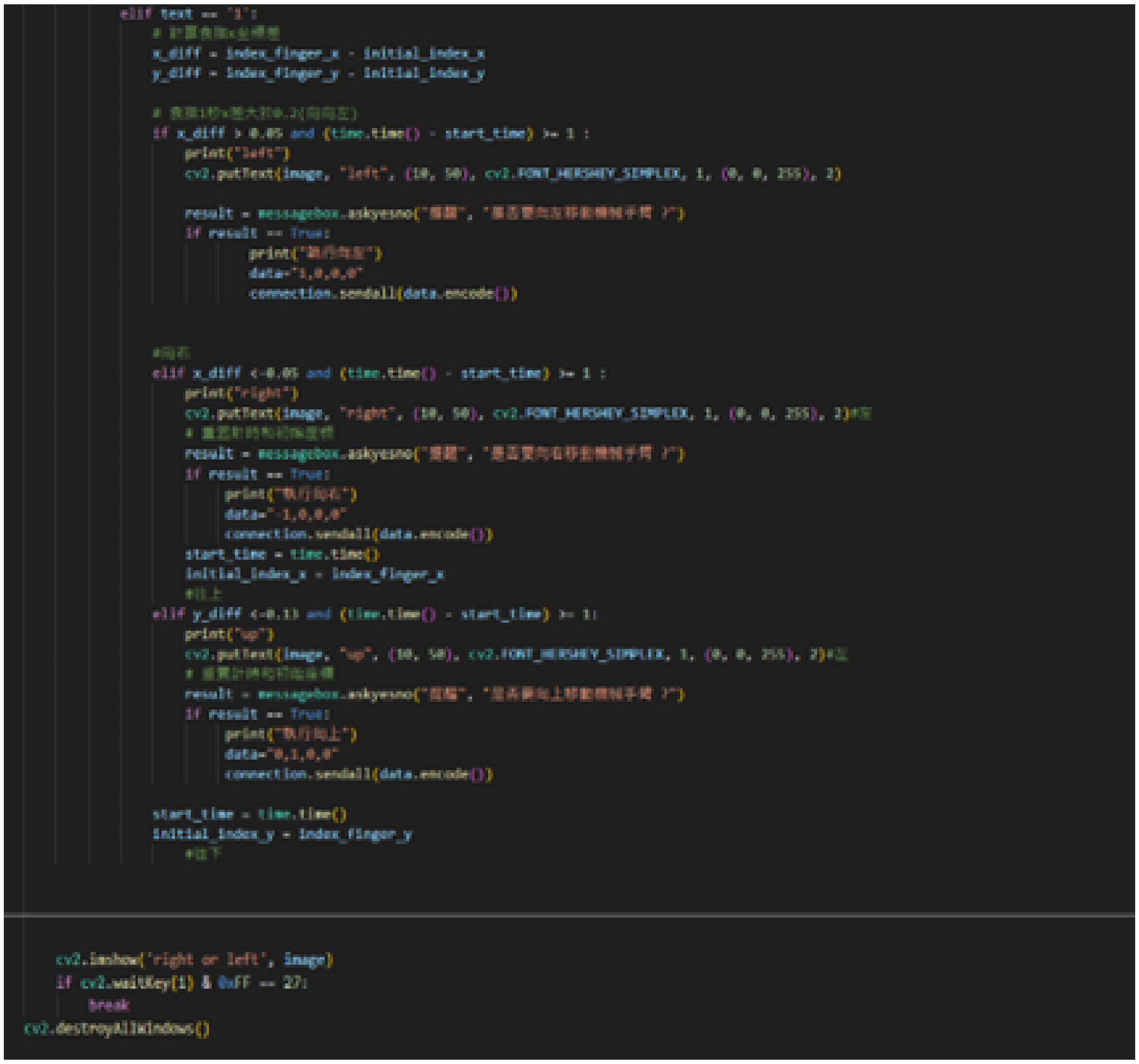
```python
import cv2
import mediapipe as mp
import math
import socket
import time
import tkinter as tk
from tkinter import messagebox
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands
# 初始化MediaPipe
hands = mp_hands.Hands(static_image_mode=False, max_num_hands=1, min_detection_confidence=0.5, min_tracking_confidence=0.5)

def vector_2d_angle(v1, v2):
    v1_x = v1[0]
    v1_y = v1[1]
    v2_x = v2[0]
    v2_y = v2[1]
    try:
        angle_= math.degrees(math.acos((v1_x*v2_x+v1_y*v2_y)/(((v1_x**2+v1_y**2)**0.5)*((v2_x**2+v2_y**2)**0.5))))
    except:
        angle_ = 180
    return angle_

# 根據傳入的 21 個節點座標，得到該手指的角度
def hand_angle(hand_):
    angle_list = []
    # 大拇指角度
    angle_ = vector_2d_angle(
            ((int(hand_[0][0])- int(hand_[2][0])),(int(hand_[0][1])-int(hand_[2][1]))),
            ((int(hand_[3][0])- int(hand_[4][0])),(int(hand_[3][1])- int(hand_[4][1])))
        )
    angle_list.append(angle_)
    # 食指角度
    angle_ = vector_2d_angle(
            ((int(hand_[0][0])-int(hand_[6][0])),(int(hand_[0][1])- int(hand_[6][1]))),
            ((int(hand_[7][0])- int(hand_[8][0])),(int(hand_[7][1])- int(hand_[8][1])))
        )
    angle_list.append(angle_)
    # 中指角度
    angle_ = vector_2d_angle(
            ((int(hand_[0][0])- int(hand_[10][0])),(int(hand_[0][1])- int(hand_[10][1]))),
            ((int(hand_[11][0])- int(hand_[12][0])),(int(hand_[11][1])- int(hand_[12][1])))
        )
    angle_list.append(angle_)
    # 無名指角度
    angle_ = vector_2d_angle(
        ((int(hand_[0][0])- int(hand_[14][0])),(int(hand_[0][1])- int(hand_[14][1]))),
        ((int(hand_[15][0])- int(hand_[16][0])),(int(hand_[15][1])- int(hand_[16][1])))
        )
    angle_list.append(angle_)
    # 小拇指角度
    angle_ = vector_2d_angle(
            ((int(hand_[0][0])- int(hand_[18][0])),(int(hand_[0][1])- int(hand_[18][1]))),
            ((int(hand_[19][0])- int(hand_[20][0])),(int(hand_[19][1])- int(hand_[20][1])))
        )
    angle_list.append(angle_)
    return angle_list

# 根據手指角度的串列內容，返回對應的手勢名稱
def hand_pos(finger_angle):
    while True:
      current_time = time.time()
      elapsed_time = current_time - start_time
      f1 = finger_angle[0]   # 大拇指角度
      f2 = finger_angle[1]   # 食指角度
      f3 = finger_angle[2]   # 中指角度
      f4 = finger_angle[3]   # 無名指角度
      f5 = finger_angle[4]   # 小拇指角度



      # 小於 50 表示手指伸直，大於等於 50 表示手指捲縮
      if f1>=50 and f2<50 and f3>=50 and f4>=50 and f5>=50:
        return '1'
      elif f1 >=60 and f2>=100 and f3>=100 and f4>=100 and f5>=100 and f1<80:
        return 'stop'
      else:
          return''

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 5000)
sock.bind(server_address)

# 監聽連線
sock.listen(1)
# 初始化變量
start_time = time.time()
initial_index_x = None

cap = cv2.VideoCapture(0)
w, h = 900,700
def close_window():
```



```
        root.destroy()
root = tk.Tk()
root.withdraw()
while True:
    success, image = cap.read()
    image=cv2.resize(image,(w,h))
    if not success:
        break

    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)# 轉RGB格式
    results = hands.process(image_rgb)# 手部關鍵點

    # 畫出來手的關鍵點
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(image, hand_landmarks, mp_hands.HAND_CONNECTIONS)
            finger_points = []                   # 記錄手指節點座標的串列
            fx = []                              # 記錄所有 x 座標的串列
            fy = []                              # 記錄所有 y 座標的串列
            index_finger_landmark = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]# 食指座標
            index_finger_x = index_finger_landmark.x
            index_finger_y = index_finger_landmark.y
            for i in hand_landmarks.landmark:
                    # 將 21 個節點換算成座標，記錄到 finger_points
                    x = i.x*w                        # 計算 x 座標
                    y = i.y*h                        # 計算 y 座標
                    finger_points.append((x,y))
                    fx.append(int(x))                # 記錄 x 座標
                    fy.append(int(y))                # 記錄 y 座標
        if finger_points:
            finger_angle = hand_angle(finger_points) # 計算手指角度，回傳長度為 5 的串列

            text = hand_pos(finger_angle)    # 取得手勢所回傳的內容


            # 判斷是否開始計時和紀錄座標
            if initial_index_x is None:
                initial_index_x = index_finger_x
                initial_index_y = index_finger_y
                start_time = time.time()
                print("等待連接")
                connection, client_address = sock.accept()
            elif text =='stop':
                x_diff = index_finger_x - initial_index_x
                y_diff = index_finger_y - initial_index_y
                if  (time.time() - start_time) >= 1:
                    print("stop")
```



```
                    cv2.putText(image, "stop", (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
                    result = messagebox.askyesno("提醒", "是否要停止機械手臂 ?")
                    if result == True:
                        data="0"
                        connection.sendall(data.encode())
                    start_time = time.time()
                    initial_index_y = index_finger_y
                    initial_index_x = index_finger_x
                elif y_diff >0.1 and (time.time() - start_time) >= 1:
                    print("down")
                    cv2.putText(image, "down", (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)#左
                    # 重置計時和初始座標
                    result = messagebox.askyesno("提醒", "是否要向下移動機械手臂 ?")
                    if result == True:
                         print("執行向下")
                         data="-1,-1,-1,-1"
                         connection.sendall(data.encode())

                    start_time = time.time()
                    initial_index_y = index_finger_y






            elif text == '1':
                # 計算食指x坐標差
                x_diff = index_finger_x - initial_index_x
                y_diff = index_finger_y - initial_index_y

                # 食指1秒x差大於0.2(向向左)
                if x_diff > 0.05 and (time.time() - start_time) >= 1 :
                    print("left")
                    cv2.putText(image, "left", (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

                    result = messagebox.askyesno("提醒", "是否要向左移動機械手臂 ?")
                    if result == True:
                            print("執行向左")
                            data="1,0,0,0"
                            connection.sendall(data.encode())
```

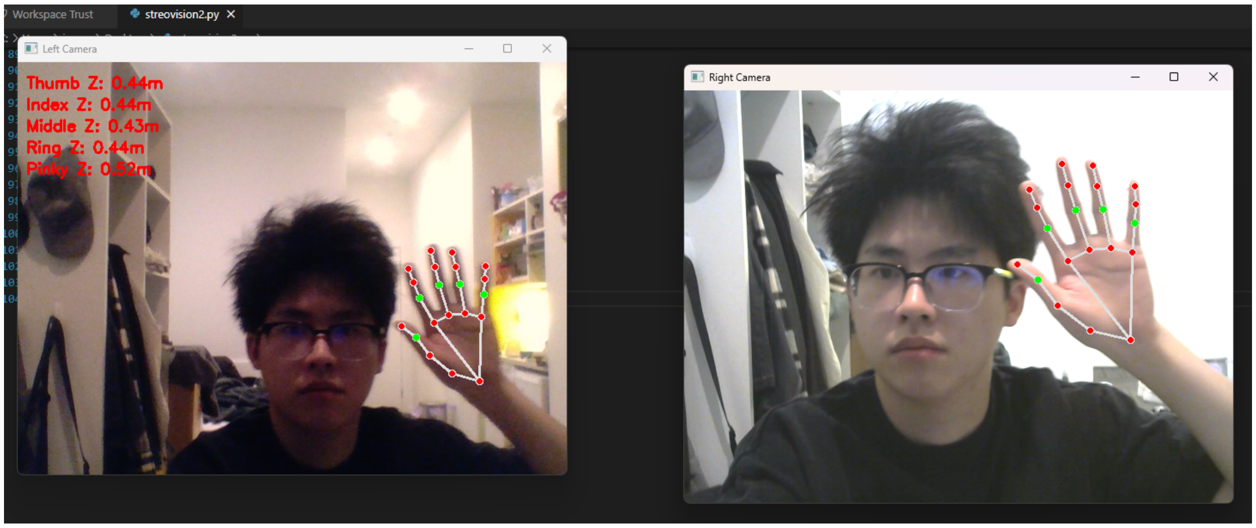


```
            elif text == '1':
                # 計算食指x坐標差
                x_diff = index_finger_x - initial_index_x
                y_diff = index_finger_y - initial_index_y

                # 食指1秒x差大於0.2(向向左)
                if x_diff > 0.05 and (time.time() - start_time) >= 1 :
                    print("left")
                    cv2.putText(image, "left", (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

                    result = messagebox.askyesno("提醒", "是否要向左移動機械手臂 ?")
                    if result == True:
                            print("執行向左")
                            data="1,0,0,0"
                            connection.sendall(data.encode())


                #向右
                elif x_diff <-0.05 and (time.time() - start_time) >= 1 :
                    print("right")
                    cv2.putText(image, "right", (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)#右
                    # 重置計時和初始座標
                    result = messagebox.askyesno("提醒", "是否要向右移動機械手臂 ?")
                    if result == True:
                        print("執行向右")
                        data="-1,0,0,0"
                        connection.sendall(data.encode())
                    start_time = time.time()
                    initial_index_x = index_finger_x
                    #往上
                elif y_diff <-0.13 and (time.time() - start_time) >= 1:
                    print("up")
                    cv2.putText(image, "up", (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)#上
                    # 重置計時和初始座標
                    result = messagebox.askyesno("提醒", "是否要向上移動機械手臂 ?")
                    if result == True:
                        print("執行向上")
                        data="0,1,0,0"
                        connection.sendall(data.encode())

                start_time = time.time()
                initial_index_y = index_finger_y
                    #往下


    cv2.imshow('right or left', image)
    if cv2.waitKey(1) & 0xFF == 27:
        break
cv2.destroyAllWindows()
```

4.2 Stereo Hand Depth Estimation Results

The system provides real-time hand depth visualization by detecting hand landmarks in both camera frames and computing disparities. Key results include:

- Depth range accuracy within 3–5cm after correction.
- Finger-by-finger Z-axis estimation shown in real-time.
- Fully vision-based system without any physical sensors.



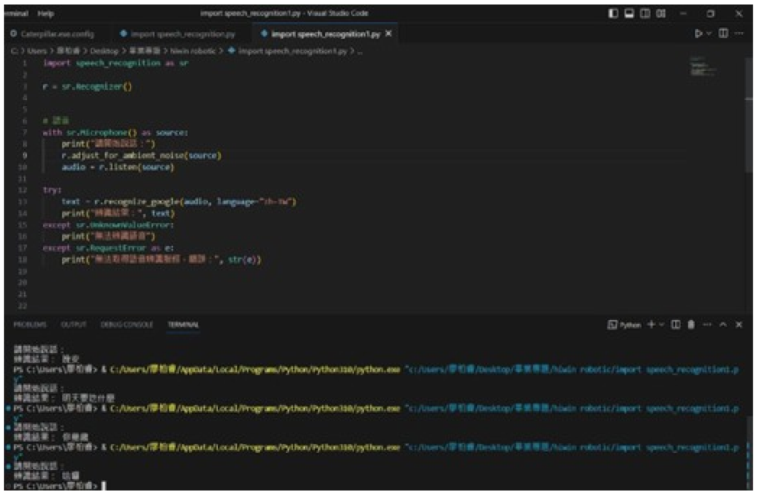
```
Thumb Z: 0.44m
Index Z: 0.44m
Middle Z: 0.43m
Ring Z: 0.44m
Pinky Z: 0.52m
```

Figure 4: Stereo vision output with estimated Z depth for each finger joint pair

4.3 Speech Recognition

We implemented a translation tool using Python and Google's Speech API. The `SpeechRecognition` library was used to transcribe input, and results were output as text or synthesized voice. For future expansion, NLP can be applied to enhance semantic understanding, enabling natural conversation with machines.

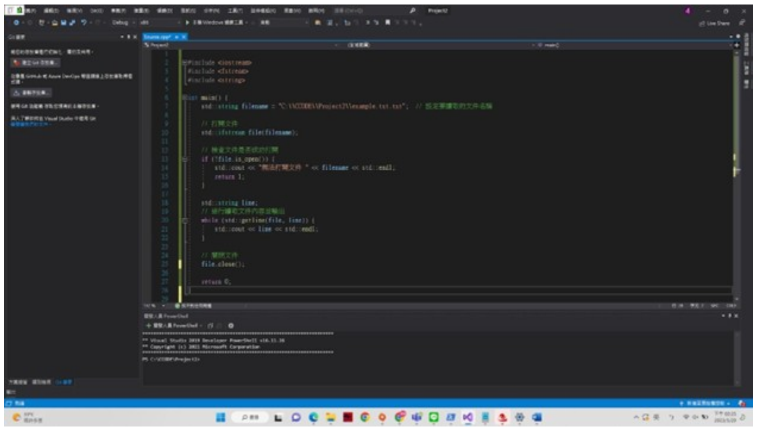
4.4 Interface Integration



```python
import speech_recognition as sr

r = sr.Recognizer()


# 語音
with sr.Microphone() as source:
    print("請開始說話：")
    r.adjust_for_ambient_noise(source)
    audio = r.listen(source)

try:
    text = r.recognize_google(audio, language="zh-TW")
    print("轉譯結果：", text)
except sr.UnknownValueError:
    print("無法轉譯語音")
except sr.RequestError as e:
    print("無法取得語音轉譯服務，錯誤：", str(e))
```

```
請開始說話：
轉譯結果： 晚安
PS C:\Users\[illegible]> & C:/Users/[illegible]/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/[illegible]/Desktop/[illegible]/hiwin robotic/import speech_recognition1.p
y"
請開始說話：
轉譯結果： 明天要吃什麼
PS C:\Users\[illegible]> & C:/Users/[illegible]/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/[illegible]/Desktop/[illegible]/hiwin robotic/import speech_recognition1.p
y"
請開始說話：
轉譯結果： [illegible]
PS C:\Users\[illegible]> & C:/Users/[illegible]/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/[illegible]/Desktop/[illegible]/hiwin robotic/import speech_recognition1.p
y"
請開始說話：
轉譯結果： [illegible]
PS C:\Users\[illegible]>
```

Figure 5: This program uses a microphone to receive speech, applies a recognition engine to transcribe spoken words, and then outputs the text. The accuracy exceeds 90%, although occasional errors may occur due to background noise, unclear articulation, or speaking too quickly.



```cpp
#include <iostream>
#include <fstream>
#include <string>

int main() {
    std::string filename = "C:\\CODE\\Project2\\example.txt.txt";  // 指定要讀取的文件名稱

    // 打開文件
    std::ifstream file(filename);

    // 檢查文件是否成功打開
    if (!file.is_open()) {
        std::cout << "無法打開文件 " << filename << std::endl;
        return 1;
    }

    std::string line;
    // 逐行讀取文件內容並輸出
    while (std::getline(file, line)) {
        std::cout << line << std::endl;
    }

    // 關閉文件
    file.close();

    return 0;
}
```

```
**********************************************************************
** Visual Studio 2019 Developer PowerShell v16.11.26
** Copyright (c) 2021 Microsoft Corporation
**********************************************************************
PS C:\CODE\Project2>
```

Figure 6: Using C++ TCP communication to transfer point coordinate data from Python, combined with MediaPipe, enables the robotic arm to operate smoothly as shown above.

4.5 Conclusion

In conclusion, this research successfully integrates speech recognition, gesture recognition via MediaPipe, and stereo vision depth estimation to control a HIWIN robotic arm, enabling more natural and precise human-robot interaction. The system demonstrates high accuracy in gesture and speech processing, with depth estimation achieving 3-5 cm precision, making it suitable for applications in disaster response, manufacturing, and assistive technologies. Experimental results validate the feasibility and effectiveness of the multimodal approach, reducing human error and enhancing operational flexibility. Future work could focus on incorporating advanced NLP for more complex commands, real-time adaptive learning, and expanding the system to mobile robotic platforms for broader applications.