COGNITIVE ROBOTICS

BCSE427P

Name: Amartya Anayachala

Registration No: 22BRS1090

Slot: L31 + L32

Assessment: Lab 7

RPi Kit 08

Team: Amartya Anayachala – 22BRS1090

GitHub Repo: https://github.com/azrael-2704/cognitive-robotics-lab-7

Question: Gesture control from RPi to move the Smart Car Kit

Code:

```
// Movement.ino
#define ENA 5
#define ENB 6
#define IN1 7
#define IN2 8
#define IN3 9
#define IN4 11
int data1 = 0;
void setup() {
pinMode(IN1, OUTPUT);
pinMode(IN2, OUTPUT);
pinMode(IN3, OUTPUT);
pinMode(IN4, OUTPUT);
pinMode(ENA, OUTPUT);
pinMode(ENB, OUTPUT);
Serial.begin(115200);
// match Python baud }
void loop() {
if (Serial.available() > 0)
{
char c = Serial.read(); // read one byte
data1 = c - '0'; // convert char ('1','2','3') to int (1,2,3)
Serial.print("Received: ");
Serial.println(data1);
if (data1 == 1) {
// Forward
digitalWrite(ENA, HIGH);
digitalWrite(ENB, HIGH);
digitalWrite(IN1, HIGH);
digitalWrite(IN2, LOW);
digitalWrite(IN3, HIGH);
digitalWrite(IN4, LOW);
}
else if (data1 == 2) {
// Backward
digitalWrite(ENA, HIGH);
digitalWrite(ENB, HIGH);
digitalWrite(IN1, LOW);
digitalWrite(IN2, HIGH);
digitalWrite(IN3, LOW);
digitalWrite(IN4, HIGH);
}
else if (data1 == 3) {
// Stop
digitalWrite(ENA, LOW);
digitalWrite(ENB, LOW);
digitalWrite(IN1, LOW);
digitalWrite(IN2, LOW);
digitalWrite(IN3, LOW);
digitalWrite(IN4, LOW);
}
else { // Unknown input -> stop
digitalWrite(ENA, LOW);
digitalWrite(ENB, LOW);
digitalWrite(IN1, LOW);
digitalWrite(IN2, LOW);
digitalWrite(IN3, LOW);
digitalWrite(IN4, LOW);
}
}
}
```

```
// Gesture.py
#!/usr/bin/env python3

import argparse
import time
from collections import deque, Counter

import cv2
import mediapipe as mp
import numpy as np
import serial

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils


def fingers_from_landmarks(lm):
    tips = [4, 8, 12, 16, 20]
    fingers = []
    for i in range(1, 5):
        try:
            tip_y = lm[tips[i]].y
            pip_y = lm[tips[i] - 2].y
            fingers.append(tip_y < pip_y)
        except Exception:




            fingers.append(False)
    try:
        thumb_up = lm[4].x > lm[3].x
    except Exception:
        thumb_up = False
    return [thumb_up] + fingers



def classify_three(fingers):
    thumb, idx, mid, ring, pinky = fingers
    if idx and mid and ring and pinky:
        return 'forward'
    if idx and mid and not ring and not pinky:
        return 'backward'
    return 'stop'



class GestureSerialGUI:
    def _init_(self, stream_url, serial_port=None, baud=115200,
                 smoothing_buffer=7, stable_frames=4, show_debug=True,
resize_w=None):
        self.stream_url = stream_url
        self.serial_port = serial_port
        self.baud = baud
        self.smoothing_buffer = smoothing_buffer
        self.stable_frames = stable_frames
        self.show_debug = show_debug
        self.resize_w = resize_w
        self.ser = None
        if serial_port:
            try:
                self.ser = serial.Serial(serial_port, baudrate=baud,
timeout=1)
                print(f"[SERIAL] Opened {serial_port} @ {baud}")
                time.sleep(2.0)
            except Exception as e:
                print(f"[SERIAL] Could not open serial {serial_port}:
{e}")
```

```
                self.ser = None
        self.cap = None
        self.open_capture()
        self.hands = mp_hands.Hands(
            static_image_mode=False,
            max_num_hands=1,
            model_complexity=0,





            min_detection_confidence=0.5,
            min_tracking_confidence=0.5
        )
        self.buffer = deque(maxlen=self.smoothing_buffer)
        self.last_sent = None
        self.same_count = 0
        self.map_cmd = {'forward': b'1', 'backward': b'2', 'stop':
b'3'}
        self._last_time = time.time()
        self._fps = 0.0
        if self.show_debug:
            cv2.namedWindow('Gesture', cv2.WINDOW_NORMAL)

    def open_capture(self):
        if self.cap:
            try:
                self.cap.release()
            except Exception:
                pass
        print(f"[VIDEO] Opening stream: {self.stream_url}")
        self.cap = cv2.VideoCapture(self.stream_url)
        time.sleep(0.5)
        if not (self.cap and self.cap.isOpened()):
            print("[VIDEO] Warning: cannot open stream right now. Will
retry in
loop.")
            self.cap = None

    def send_serial(self, gesture):
        if self.ser is None:
            print(f"[SERIAL] (dry-run) -> {gesture} ->
{self.map_cmd.get(gesture)}")
            return
        cmd = self.map_cmd.get(gesture)
        if cmd is None:
            return
        try:
            self.ser.write(cmd)
            self.ser.flush()
            print(f"[SERIAL] Sent {cmd} for gesture {gesture}")
        except Exception as e:
            print(f"[SERIAL] Serial write failed: {e}")

    def _update_fps(self):
        now = time.time()





        dt = now - self._last_time
        if dt > 0:
            self._fps = 0.9 * self._fps + 0.1 * (1.0 / dt) if self._fps
else (1.0
/ dt)
        self._last_time = now

    def run(self):
        print("[RUN] Starting main loop. Press Ctrl+C or 'q' in window
to quit.")
        try:
            while True:
                if not self.cap or not self.cap.isOpened():
                    self.open_capture()
                    time.sleep(0.5)
                    continue
                ret, frame = self.cap.read()
                if not ret or frame is None:
                    self.cap.release()
                    self.cap = None
                    time.sleep(0.2)
                    continue
                if self.resize_w:
                    h, w = frame.shape[:2]
                    new_h = int(h * (self.resize_w / w))
                    frame = cv2.resize(frame, (self.resize_w, new_h))
                rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                results = self.hands.process(rgb)
                gesture = 'stop'
                if results.multi_hand_landmarks and
len(results.multi_hand_landmarks) > 0:
                    hand_lm = results.multi_hand_landmarks[0].landmark
                    fingers = fingers_from_landmarks(hand_lm)
                    gesture = classify_three(fingers)
                    mp_drawing.draw_landmarks(
                        frame,
                        results.multi_hand_landmarks[0],
                        mp_hands.HAND_CONNECTIONS
                    )
                self.buffer.append(gesture)
                most_common, count =
Counter(self.buffer).most_common(1)[0]
```

```
                if most_common == self.last_sent:
                    self.same_count += 1
                else:
                    self.same_count = 1





                if self.same_count >= self.stable_frames and
most_common !=
self.last_sent:
                    self.send_serial(most_common)
                    self.last_sent = most_common
                self._update_fps()
                cv2.putText(frame, f'Gesture: {gesture}', (10, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 255, 0),
2)
                cv2.putText(frame, f'FPS: {self._fps:.1f}', (10, 60),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 200,
0), 2)
                if self.show_debug:
                    cv2.imshow('Gesture', frame)
                    key = cv2.waitKey(1) & 0xFF
                    if key == ord('q'):
                        print("[RUN] Quit pressed")
                        break
        except KeyboardInterrupt:
            print("\n[RUN] Interrupted by user")
        finally:
            try:
                if self.cap:
                    self.cap.release()
            except Exception:
                pass
            try:
                cv2.destroyAllWindows()
            except Exception:
                pass
            try:
                if self.hands:
                    self.hands.close()
            except Exception:
                pass
            try:
                if self.ser:
                    self.ser.close()
            except Exception:
                pass
            print("[RUN] Clean exit")


if _name_ == '_main_':
    parser = argparse.ArgumentParser()
    parser.add_argument('--url', '-u', required=True)
    parser.add_argument('--serial', '-s', default=None)




    parser.add_argument('--baud', '-b', type=int, default=115200)
    parser.add_argument('--buffer', type=int, default=7)
    parser.add_argument('--stable', type=int, default=4)
    parser.add_argument('--no-gui', action='store_true')
    parser.add_argument('--resize', type=int, default=640)
    args = parser.parse_args()

    gs = GestureSerialGUI(
        stream_url=args.url,
        serial_port=args.serial,
        baud=args.baud,
        smoothing_buffer=args.buffer,
        stable_frames=args.stable,
        show_debug=not args.no_gui,
        resize_w=args.resize if args.resize > 0 else None
    )
    gs.run()
```

Screenshots: