**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*](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:**

 