Utkarsh pandey - 22BRS1111

Hrijumana Dubey – 22BRS1187

Ankit kumar – 22BRS1130

Lab-7

**Gesture control from RPi to move the Smart Car Kit**

**Aim:**  
To design and implement a gesture-controlled smart car using a Raspberry Pi, where hand gestures are captured, processed, and transmitted to control the movement of the car in real time, ensuring that the control functionality is enabled only for authorized users through face recognition.

1.Arduino Code

#define ENA 5

#define ENB 6

#define IN1 7

#define IN2 8

#define IN3 9

#define IN4 11

void setup() {

pinMode(IN1, OUTPUT);

pinMode(IN2, OUTPUT);

pinMode(IN3, OUTPUT);

pinMode(IN4, OUTPUT);

pinMode(ENA, OUTPUT);

pinMode(ENB, OUTPUT);

Serial.begin(9600);

}

void moveForward() {

digitalWrite(IN1, HIGH);

digitalWrite(IN2, LOW);

digitalWrite(IN3, LOW);

digitalWrite(IN4, HIGH);

analogWrite(ENA, 200);

analogWrite(ENB, 200);

}

void moveBackward() {

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

digitalWrite(IN3, HIGH);

digitalWrite(IN4, LOW);

analogWrite(ENA, 200);

analogWrite(ENB, 200);

}

void moveLeft() {

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

digitalWrite(IN3, LOW);

digitalWrite(IN4, HIGH);

analogWrite(ENA, 200);

analogWrite(ENB, 200);

}

void moveRight() {

digitalWrite(IN1, HIGH);

digitalWrite(IN2, LOW);

digitalWrite(IN3, HIGH);

digitalWrite(IN4, LOW);

analogWrite(ENA, 200);

analogWrite(ENB, 200);

}

void stopCar() {

digitalWrite(IN1, LOW);

digitalWrite(IN2, LOW);

digitalWrite(IN3, LOW);

digitalWrite(IN4, LOW);

}

void loop() {

if (Serial.available() > 0) {

char command = Serial.read();

switch (command) {

case 'F':

moveForward();

break;

case 'B':

moveBackward();

break;

case 'L':

moveLeft();

break;

case 'R':

moveRight();

break;

case 'S':

stopCar();

break;

}

}

}

2.Raspberry pi code

import cv2

import numpy as np

import serial

import time

import face\_recognition

import os

ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1) # Update port if needed

time.sleep(2)

known\_encodings = []

known\_names = []

# Load authorized faces from "known\_faces" directory

for filename in os.listdir("known\_faces"):

if filename.endswith((".jpg", ".png")):

img = face\_recognition.load\_image\_file(f"known\_faces/{filename}")

enc = face\_recognition.face\_encodings(img)[0]

known\_encodings.append(enc)

known\_names.append(os.path.splitext(filename)[0])

print("[INFO] Loaded authorized faces:", known\_names)

def count\_fingers(thresh, drawing):

contours, \_ = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

if not contours:

return 0

max\_contour = max(contours, key=cv2.contourArea)

if cv2.contourArea(max\_contour) < 1000:

return 0

hull = cv2.convexHull(max\_contour, returnPoints=False)

if hull is None or len(hull) < 3:

return 0

defects = cv2.convexityDefects(max\_contour, hull)

if defects is None:

return 0

finger\_count = 0

for i in range(defects.shape[0]):

s, e, f, d = defects[i, 0]

start = tuple(max\_contour[s][0])

end = tuple(max\_contour[e][0])

far = tuple(max\_contour[f][0])

a = np.linalg.norm(np.array(end) - np.array(start))

b = np.linalg.norm(np.array(far) - np.array(start))

c = np.linalg.norm(np.array(end) - np.array(far))

# FIX: corrected formula (used \*\* instead of \*2 typo)

angle = np.arccos((b\*\*2 + c\*\*2 - a\*\*2) / (2 \* b \* c))

if angle <= np.pi / 2 and d > 10000:

finger\_count += 1

cv2.circle(drawing, far, 8, [255, 0, 0], -1)

return finger\_count + 1

def skin\_mask(frame):

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

lower = np.array([0, 30, 60], dtype=np.uint8)

upper = np.array([20, 150, 255], dtype=np.uint8)

mask = cv2.inRange(hsv, lower, upper)

kernel = np.ones((3, 3), np.uint8)

mask = cv2.dilate(mask, kernel, iterations=4)

mask = cv2.GaussianBlur(mask, (5, 5), 0)

return mask

def main():

ip\_url = 'http://192.168.1.100:8080/video' # Change to your IP cam

cap = cv2.VideoCapture(ip\_url)

if not cap.isOpened():

print("Error: Cannot open IP webcam stream")

return

last\_command = ''

last\_time = time.time()

command\_delay = 1.0

authorized = False

while True:

ret, frame = cap.read()

if not ret:

break

frame = cv2.flip(frame, 1)

small\_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

rgb\_small = cv2.cvtColor(small\_frame, cv2.COLOR\_BGR2RGB)

face\_locations = face\_recognition.face\_locations(rgb\_small)

face\_encodings = face\_recognition.face\_encodings(rgb\_small, face\_locations)

authorized = False

for enc in face\_encodings:

matches = face\_recognition.compare\_faces(known\_encodings, enc, tolerance=0.5)

if True in matches:

authorized = True

break

if authorized:

cv2.putText(frame, "AUTHORIZED", (50, 50),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)

roi = frame[100:400, 100:400]

cv2.rectangle(frame, (100, 100), (400, 400), (0, 255, 0), 2)

mask = skin\_mask(roi)

\_, thresh = cv2.threshold(mask, 127, 255, cv2.THRESH\_BINARY)

drawing = np.zeros(roi.shape, np.uint8)

fingers = count\_fingers(thresh, drawing)

if fingers == 1:

command = 'F'

elif fingers == 2:

command = 'B'

elif fingers == 3:

command = 'L'

elif fingers == 4:

command = 'R'

else:

command = 'S'

current\_time = time.time()

if command != last\_command and (current\_time - last\_time) > command\_delay:

ser.write(command.encode())

print(f"Fingers: {fingers} -> Command sent: {command}")

last\_command = command

last\_time = current\_time

cv2.imshow('Threshold', thresh)

cv2.imshow('Drawing', drawing)

else:

cv2.putText(frame, "NOT AUTHORIZED", (50, 50),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)

ser.write(b'S') # force STOP if unauthorized

cv2.imshow('Frame', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

ser.close()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Github link for the implementation : [varshini197/COG-lab](https://github.com/varshini197/COG-lab)