FACE DETECTION AND RECOGNISATION HUMAN TRACKING ROBOT

Coding Part

ABSTRACT

A robotics project integrating face recognition and human tracking. Using OpenCV and deep learning models (e.g., CNNs, FaceNet), it detects and tracks faces in real-time. The system adapts to dynamic environments, ideal for security, surveillance, and interactive robotics applications.

Amar Sunadholi

Code:

```
import cv2
import numpy as np
import os
import torch
import dlib
from openface import align_dlib
from openface import TorchNeuralNet
from sklearn.preprocessing import Normalizer

# Paths for OpenFace
OPENFACE_MODEL_DIR = 'path/to/openface/models'  # Update with your OpenFace model path
PREDICTOR_PATH = 'shape_predictor_68_face_landmarks.dat'  # Dlib shape predictor file

# Initialize OpenFace model and aligner
aligner = align_dlib.AlignDlib(PREDICTOR_PATH)
net = TorchNeuralNet(os.path.join(OPENFACE_MODEL_DIR, 'nn4.small2.v1.t7'), 96)  # The model used for face recognition

# Initialize face detector (Haar Cascade + Dlib face detector)
video = cv2.VideoCapture(0)
facedetect = cv2.CascadeClassifier('data/haarcascade_frontalface_default.xml')

# Check if the classifier loaded properly
if facedetect.empty():
   print("Error loading face cascade!")
   exit()

# Create a folder to store the face images
```

```
if not os.path.exists('captured_faces'):
    os.makedirs('captured_faces')

faces_data = []
capture_interval = 10  # Capture one face every 10 frames
i = 0

def get_face_embedding(face_image):
    # Align the face using dlib aligner
    aligned_face = aligner.align(96, face_image, landmarkIndices=align_dlib.AlignDlib.OUTER_EYES_AND_NOSE)
    if aligned_face is not None:
        # Extract the embedding using OpenFace neural net
        face_embedding = net.forward(aligned_face)
        return face_embedding
    else:
        return None

def recognize_face(face_embedding, known_faces):
    # Compare the embeddings to known faces (simple Euclidean distance for now)
    min_dist = float("inf")
    identity = None
    for name, known_embedding in known_faces.items():
        dist = np.linalg.norm(face_embedding - known_embedding)
        if dist < min_dist:
            min_dist = dist
            identity = name
    return identity

known_faces = {}  # Dictionary to store known faces and their embeddings
```

```
while True:
    ret, frame = video.read()

    # If no frame is captured, exit
    if not ret:
        print("Failed to grab frame")
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces
    faces = facedetect.detectMultiScale(gray, 1.3, 5)

    # For each detected face, process and store it
    for (x, y, w, h) in faces:
        crop_img = frame[y:y+h, x:x+w]  # Crop the face
        resized_img = cv2.resize(crop_img, (100, 100))  # Resize the cropped face

        if len(faces_data) < 100 and i % capture_interval == 0:  # Limit to 100 faces
            # Get the face embedding for recognition
            face_embedding = get_face_embedding(crop_img)

            if face_embedding is not None:
                # Store the face embedding (For example, we can store the first 100 faces for later recognition)
                identity = f'Face_{len(faces_data)}'
                known_faces[identity] = face_embedding
                print(f"Captured {identity} with embedding.")

                # Save the captured face to the disk with a unique filename
                face_filename = f'captured_faces/face_{len(faces_data)}.jpg'
                cv2.imwrite(face_filename, resized_img)
                faces_data.append(resized_img)

        # Draw rectangle around the face and display the count
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, f"Faces Detected: {len(faces_data)}", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

        # Perform face recognition (if embeddings are available)
        if len(known_faces) > 0:
            face_embedding = get_face_embedding(crop_img)
            if face_embedding is not None:
                identity = recognize_face(face_embedding, known_faces)
                if identity:
                    cv2.putText(frame, f"Recognized: {identity}", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 255), 2)

    # Show the frame with faces detected
    cv2.imshow("Frame", frame)

    # Break on 'q' key press
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    i += 1

# Release the video capture and destroy windows
video.release()
cv2.destroyAllWindows()
```

```
print(f"Captured {len(faces_data)} faces.")
```