PROJECT REPORT

Title : Real-Time Face Detection and Graphics Overlay System Using OpenCV and Streamlit

Objective

The objective of this project is to develop a real-time computer vision application capable of detecting human faces from multiple input sources such as webcam, images, and video files. The system applies graphical overlays such as bounding boxes, emoji filters, and blur effects to detected faces using OpenCV image processing techniques.
The project demonstrates file handling, real-time frame processing, drawing dynamic shapes, and overlaying graphical content on detected facial regions. It also integrates Streamlit to provide an interactive and user-friendly web interface. The application aims to showcase the practical implementation of computer vision techniques in augmented reality filters, privacy protection, and multimedia enhancement.

Theoretical Background

Face Detection

Face detection is a computer vision technique used to locate human faces in digital images or video streams. It is commonly used in surveillance systems, biometric authentication, augmented reality applications, and multimedia processing.
This project uses the Haar Cascade Classifier, which is a machine learning-based approach introduced by Viola and Jones. The classifier detects faces by identifying characteristic facial features such as edges and intensity differences.

Image and Video Processing

Image processing involves analyzing and modifying digital images using algorithms. Video processing extends this concept by analyzing frames extracted from video streams or camera feeds.
The application processes:

- Static image files
- Video files
- Real-time webcam streams

Graphics Overlay Techniques

Graphics overlay involves drawing shapes, text, or images on detected objects. The project applies overlays such as:

- Bounding boxes for visualization
- Emoji filters for augmented reality effects
- Face blurring for privacy protection

Streamlit Integration

Streamlit is used to build a web-based interface that allows users to upload media files, select filters, and interact with the application dynamically.

Technical Requirements

Software Requirements

- Python 3.x
- OpenCV
- Streamlit
- NumPy
- Pillow

Hardware Requirements

- Webcam or camera-enabled device
- Minimum 4 GB RAM recommended

System Architecture

Input Layer

Accepts data from:

- Camera frames
- Video files
- Image files

Processing Layer

- Face detection using Haar Cascade classifier
- Frame-by-frame processing
- Filter and overlay application

Output Layer

- Display processed frames
- Save snapshots automatically
- Display face count

Code

```
import streamlit as st
import cv2
import numpy as np
from PIL import Image

# ---------------- Load Face Detector ----------------
face_cascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
)
# ---------------- UI ----------------
st.title("🎭 Real-Time Face Detection with Filters")
st.sidebar.header("Settings")
mode = st.sidebar.selectbox(
    "Select Input Source",
    ["Webcam", "Upload Video", "Upload Image"]
)
filter_option = st.sidebar.selectbox(
    "Select Filter",
    ["Bounding Box", "Emoji", "Blur Face"]
)
# Emoji selection
emoji_map = {
    "😈": "smiling_imp.png",
    "😁": "1f601.png",
    "🥸": "disguised_face.png",
    "🤬": "face_with_symbols_on_mouth.png",
    "😍": "heart_eyes.png",
    "😡": "rage.png"
}
if filter_option == "Emoji":
    emoji_choice = st.sidebar.selectbox(
        "Choose Emoji",
        list(emoji_map.keys())
    )
    emoji_img = cv2.imread(
        emoji_map[emoji_choice],
        cv2.IMREAD_UNCHANGED
    )
snapshot = st.sidebar.checkbox("Auto Snapshot")
FRAME_WINDOW = st.image([])
snapshot_counter = 0
# ---------------- Emoji Overlay Function ----------------
def overlay_emoji(frame, emoji_img, x, y, w, h):
    # Prevent overflow outside frame
    h_frame, w_frame = frame.shape[:2]
    if y + h > h_frame or x + w > w_frame:
        return frame
    emoji_resized = cv2.resize(emoji_img, (w, h))
    if emoji_resized.shape[2] == 4:  # Transparent PNG
        alpha = emoji_resized[:, :, 3] / 255.0
        for c in range(3):
            frame[y:y+h, x:x+w, c] = (
                alpha * emoji_resized[:, :, c] +
                (1 - alpha) * frame[y:y+h, x:x+w, c]
            )
    else:
        frame[y:y+h, x:x+w] = emoji_resized
    return frame
# ---------------- Filter Function ----------------
def apply_filter(frame, faces):
    global snapshot_counter
    for (x, y, w, h) in faces:
        if filter_option == "Bounding Box":
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,0), 2)
            cv2.putText(frame, "Face", (x, y-10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,255,0), 2)
        elif filter_option == "Emoji":
            frame = overlay_emoji(frame, emoji_img, x, y, w, h)
        elif filter_option == "Blur Face":
            face_region = frame[y:y+h, x:x+w]
            blur = cv2.GaussianBlur(face_region, (99, 99), 30)
            frame[y:y+h, x:x+w] = blur
        # Snapshot
        if snapshot:
            filename = f"snapshot_{snapshot_counter}.jpg"
            cv2.imwrite(filename, frame)
            snapshot_counter += 1
    # Face count display
    cv2.putText(frame, f"Faces: {len(faces)}",
                (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
                1, (255, 0, 0), 2)
    return frame
# ---------------- Webcam Mode ----------------
if mode == "Webcam":
    run = st.checkbox("Start Webcam")
    if run:
        cap = cv2.VideoCapture(0)
        while run:
            ret, frame = cap.read()
            if not ret:
                break
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray, 1.3, 5)
            frame = apply_filter(frame, faces)
            FRAME_WINDOW.image(
                cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            )
        cap.release()
# ---------------- Video Upload Mode ----------------
```

```
elif mode == "Upload Video":
    video_file = st.file_uploader(
        "Upload Video",
        type=["mp4", "avi", "mov"]
    )
    if video_file is not None:
        with open("temp_video.mp4", "wb") as f:
            f.write(video_file.read())
        cap = cv2.VideoCapture("temp_video.mp4")
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray, 1.3, 5)
            frame = apply_filter(frame, faces)
            FRAME_WINDOW.image(
                cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            )
        cap.release()
# ---------------- Image Upload Mode ----------------
elif mode == "Upload Image":
    img_file = st.file_uploader(
        "Upload Image",
        type=["jpg", "png", "jpeg"]
    )
    if img_file is not None:
        img = Image.open(img_file)
        frame = np.array(img)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)
        frame = apply_filter(frame, faces)
        st.image(frame)
```

Output

The application generates processed visual outputs showing:

1. Real-time face detection
2. Graphics overlay effects
3. Snapshot generation when faces are detected
4. Processed images or videos with filters applied

Advantages

- Real-time processing capability
- Multi-source input support
- User-friendly interface
- Customizable graphical filters
- Lightweight and efficient implementation

Limitations

- Haar Cascade detection accuracy may reduce in low lighting
- Limited facial landmark detection compared to deep learning models
- Performance depends on hardware capability

Future Enhancements

The system can be enhanced by:

- Implementing deep learning face detection models
- Adding emotion recognition filters
- Using MediaPipe face mesh for precise filter alignment
- Adding face recognition authentication
- Allowing processed video download
- Integrating cloud storage for snapshots

Conclusion

This project successfully demonstrates the implementation of real-time face detection combined with graphical overlays using OpenCV and Streamlit. The system effectively processes live video streams and uploaded media files to detect faces and apply dynamic filters.

The application highlights the importance of computer vision techniques in modern multimedia systems and augmented reality technologies. It also demonstrates the integration of machine learning algorithms with interactive web-based interfaces, making the solution accessible and user-friendly.