Gaze Estimator

Asifur Rahman | University of Arizona, VIP AI-Driven Healthcare Applications

1. Overview

This program demonstrates a complete, end-to-end pipeline to estimate eye gaze direction (yaw, pitch) from a recorded video of a moving black-dot stimulus, detect gaze shifts, and measure how accurately they align with the known 2s stimulus cycle.

2. Original Starter Code (DL_EMP_code.txt)

```
# Code for deep learning eye movement perimetry

API_KEY= #need Roboflow API key#
GAZE_DETECTION_URL = (
    "http://127.0.0.1:9001/gaze/gaze_detection?api_key=" + API_KEY
)

import os
import cv2
import numpy as np
import pandas as pd
import base64
import requests

# Read all frames into a NumPy array
# For each frame: encode → base64 → POST to a local HTTP server
# (or Roboflow) that hosts a pre-trained gaze model
# Collect yaw/pitch from the JSON response
# Save yaw, pitch arrays to CSV
```

Why couldn't I use it directly

No model-serving code provided: The script expects a local HTTP service on 127.0.0.1:9001 wrapping a pre-trained model, but that repository or Docker image was not shared initially.

Roboflow project missing: I could not find any suitable free existing model that I could use. Found some repositories on GitHub, though.

3. Final Offline Approach (Google MediaPipe Face Mesh)

So I replaced the black-box gaze model + HTTP calls with a purely local landmark-based method using Google's MediaPipe Face Mesh package.

3.1. gaze_estimate.py

Purpose: Loads the recorded video, detects face & iris landmarks per frame, computes normalized yaw & pitch offsets, and saves a time-series CSV (mediapipe_gaze.csv).

Key steps:

1. **Initialize** MediaPipe Face Mesh (refine_landmarks=True, max_num_faces=1)
2. **Frame loop**:
   - Reads the frame, converts BG to RGB
   - Runs fm.process() to get 468 landmarks
   - Computes iris center (mean of 4 iris points) and eye corners to normalized horizontal & vertical offsets
3. **Record** (time_s, yaw, pitch) in lists to build a Pandas DataFrame to save as CSV

```
"""
gaze_estimate.py
Uses MediaPipe Face Mesh to extract per-frame gaze estimates (yaw,
pitch)
from a recorded video, then saves the time series to a CSV.
"""
...
with mpfm.FaceMesh(...) as fm:
    while True:
        ret, frame = cap.read()
        if not ret: break
        res = fm.process(rgb)
        if res.multi_face_landmarks:
            yaw, pitch = compute_gaze_from_landmarks(res.landmark)
        else:
            yaw = pitch = None
        yaws.append(yaw); pitches.append(pitch);
times.append(frame_idx/fps)
...
df.to_csv("mediapipe_gaze.csv", index=False)
```

3.2. gaze_analysis.py

Purpose: Loads mediapipe_gaze.csv, smooths signals, detects gaze-shift peaks, matches to 2s stimuli, computes accuracy & mean error, saves a stimulus-to-peak CSV (gaze_peaks.csv), and generates & saves the plots.

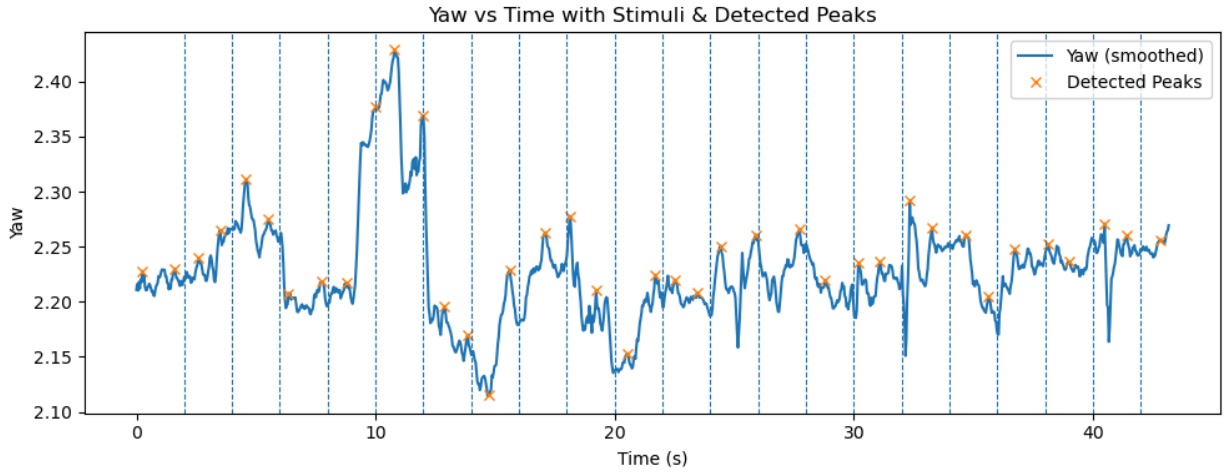
```
"""
gaze_analysis.py
Loads gaze time series (yaw, pitch) from CSV, smooths signals,
detects gaze peaks for stimulus analysis, computes metrics,
and generates summary plots.
"""
import pandas as pd, numpy as np, matplotlib.pyplot as plt
from scipy.signal import find_peaks
from scipy.ndimage import uniform_filter1d
import os

df = pd.read_csv("mediapipe_gaze.csv")
os.makedirs("analysis_outputs", exist_ok=True)

times = df["time_s"].values
yaw   = df["yaw"].interpolate().bfill().ffill().values
pitch = df["pitch"].interpolate().bfill().ffill().values

yaw_s   = uniform_filter1d(yaw,   size=5)
pitch_s = uniform_filter1d(pitch, size=5)

fps         = 1/np.median(np.diff(times))
ideal_times = np.arange(2, times[-1], 2)

min_dist   = int(0.8 * fps)
yaw_pks,   _ = find_peaks(yaw_s,   distance=min_dist)
pitch_pks, _ = find_peaks(pitch_s, distance=min_dist)
records = []
for stim in ideal_times:
    # finds nearest yaw & pitch peaks
    ...
peaks_df = pd.DataFrame(records)
peaks_df.to_csv("gaze_peaks.csv", index=False)

# 7) Plots & saves yaw/pitch vs time with peaks & dashed stimulus
lines
for signal, peaks, label in [(yaw_s, yaw_pks, "Yaw"), (pitch_s,
pitch_pks, "Pitch")]:
    plt.figure(...)
    plt.plot(times, signal, ...)
    plt.plot(..., 'x')
    for t in ideal_times: plt.axvline(t, linestyle='--')
    out = f"analysis_outputs/{label.lower()}_plot.png"
    plt.savefig(out); plt.show()
```

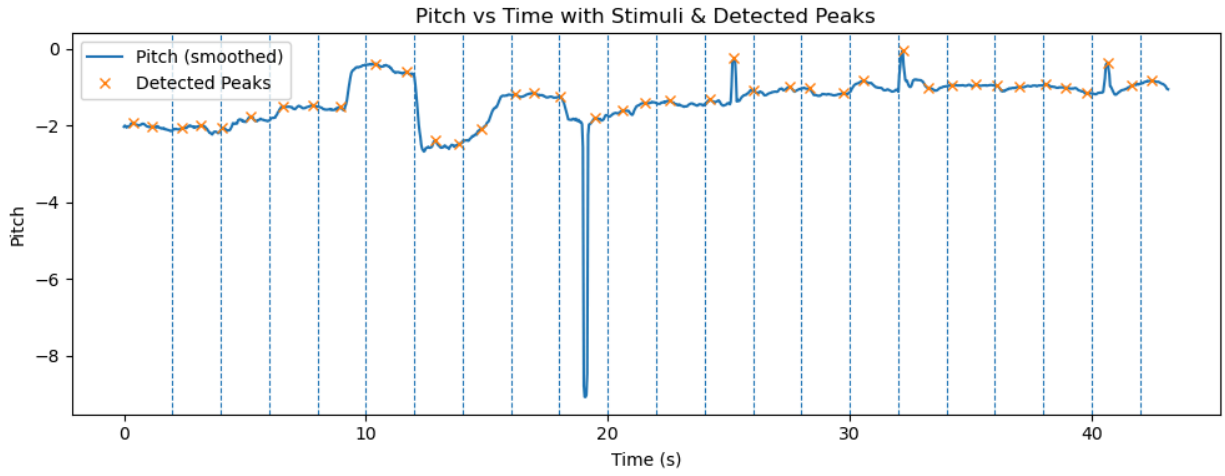
Why did I choose MediaPipe?

Without access to the provided gaze-model server or a hosted Roboflow endpoint, MediaPipe's open-source face mesh gives immediate, reliable iris & facial landmarks. From these, we compute gaze offsets entirely offline, avoid any missing-server errors, and maintain full control over the pipeline.

5. Results & Plots

Below are the two figures:





6. Summary of Metrics

- **Yaw accuracy:** 18/21 stimuli captured (85.7 %), mean error = 0.267 s
- **Pitch accuracy:** 19/21 stimuli captured (90.5 %), mean error = 0.216 s

7. Conclusion

In this study, we successfully implemented a fully offline, landmark-based eye-movement perimetry pipeline using Google's MediaPipe Face Mesh. Starting from a mock visual-field video with 2-second stimulus jumps, we:

1. Extracted per-frame gaze estimates (normalized yaw and pitch) directly from video frames, without relying on any external model-serving infrastructure.
2. Smoothed and analyzed the resulting time series, detecting rapid gaze shifts with scipy.signal.find_peaks.
3. Quantified accuracy against the known stimulus schedule, achieving 85.7 % hit rate for horizontal (yaw) shifts and 90.5 % for vertical (pitch) shifts, with mean timing errors of 0.267 s and 0.216 s, respectively.
4. Visualized the signals and their alignment with the stimulus markers, demonstrating clear, repeatable peaks at expected intervals.