John de Vere Potential Intern Onboarding Project: Real-Time Eye Tracking System

Overview of Project

Your project is to build a real-time eye tracking system that detects and classifies eye states (open/closed) using computer vision. This project will introduce you to the core technologies we use in our swimming analysis platform: **OpenCV** for video processing and **MediaPipe** for landmark detection.

This project will run through building a small real-time computer vision system. It's less about detecting eyes specifically and more on whether you can take a video feed, run a model, extract landmarks, do geometry on them, and classify an event (eye blink). These are the same skills used in our swim stroke classification pipeline that Abhi is working on. The goal is to evaluate your ability to read documentation, structure code, and debug end-to-end CV systems

---

Due Date: Wednesday, Feb 25, 11:59pm. Submission details here

Difficulty: Beginner to Intermediate

Technologies: Python, OpenCV, MediaPipe, NumPy

General questions to **Eugene**, Coding questions to **Abhi - abhimokkapati@berkeley.edu**

Learning Objectives

By completing this project, you will:

- Understand video processing and frame manipulation with OpenCV
- Learn facial landmark detection using MediaPipe
- Implement geometric calculations for feature extraction
- Build a rule-based classification system
- Handle real-time video streams from webcam
- Debug and optimize computer vision pipelines

Technical Requirements

Core Functionality

Your eye tracking system must:

1. **Live footage**
    a. Access webcam feed and process video in real-time at 30fps
2. **Landmarks**
    a. Detect facial landmarks using MediaPipe Face Mesh
    b. Extract eye landmarks for both left and right eyes
3. **Causal relationships**
    a. Calculate Eye Aspect Ratio (EAR) using geometric relationships
4. **Classification**
    a. Classify eye state as OPEN or CLOSED based on EAR threshold
5. **Interface**
    a. Display results with visual overlay on the video feed
6. **Other**
    a. Handle edge cases
        i. No face detected
        ii. Partial occlusion
        iii. Etc.

Technical Specifications

Input

- **Video Source**
  - Webcam
- **Frame Rate**
  - 30 fps min.
- **Resolution**
  - 640x480 or higher

Output

- Real-time video display
  - Eye contour visualization (drawn landmarks)
  - Eye state text label (OPEN/CLOSED)
  - Current EAR value
  - Frame counter
- Console output
- Show detection status

Algorithm Requirements

Eye Aspect Ratio (EAR) Formula:

$$EAR = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2 \cdot ||p_1 - p_4||}$$

Where,
p_1, p_4 = horizontal eye corners (left, right),
p_2, p_3, p_5, p_6 = vertical eye landmarks (top and bottom),
|| || represents Euclidean distance.

Classification Threshold:

- EAR $< 0.21 \rightarrow$ Eyes CLOSED
- EAR $\geq 0.21 \rightarrow$ Eyes OPEN

MediaPipe Face Mesh Landmarks:

- Left Eye: Indices [362, 385, 387, 263, 373, 380]
- Right Eye: Indices [33, 160, 158, 133, 153, 144]

Project Structure

```
eye_tracking_project/
├── eye_tracker.py          # Main implementation
├── requirements.txt        # Python dependencies
├── README.md               # Project documentation
└── tests/                  # Optional: unit tests
    └── test_ear.py         # Test EAR calculation
```

Implementation Phases

1st – Environment Setup

Tasks:

- Set up Python virtual environment
- Install required packages: opencv-python, mediapipe, numpy, scipy
- Test webcam access with basic OpenCV
- Verify MediaPipe installation

Deliverable: Working webcam display using OpenCV

2nd – Face Detection

Tasks:

- Initialize MediaPipe Face Mesh
- Process webcam frames and detect face
- Extract and visualize all facial landmarks
- Handle "no face detected" gracefully

Deliverable: Real-time face landmark detection

3rd – Eye Landmark Extraction

Tasks:

- Identify correct eye landmark indices from MediaPipe documentation
- Extract 6 landmarks per eye (12 total)
- Convert normalized coordinates to pixel coordinates
- Draw eye contours on video feed

Deliverable: Visual eye contour overlay

4th – EAR Calculation

Tasks:

- Implement Euclidean distance function
- Calculate EAR for both eyes
- Compute average EAR
- Display EAR value on screen

Deliverable: Real-time EAR computation

5th – Eye State Classification

Tasks:

- Implement threshold-based classification
- Add state labels (OPEN/CLOSED)
- Color-code display (green=open, red=closed)
- Test with deliberate blinking

Deliverable: Working eye state classifier

6th – Polish & Edge Cases

Tasks:

- Handle multiple faces (process only one)
- Add error handling for camera failures
- Implement clean exit (press 'q' to quit)
- Optimize performance if needed
- Write documentation

Deliverable: Production-ready code with documentation

Required Dependencies

opencv-python>=4.8.0
mediapipe>=0.10.0
numpy>=1.24.0
scipy>=1.10.0

Install with:

pip install opencv-python mediapipe numpy scipy

Code Architecture

Your implementation should follow this structure:

```
class EyeTracker:
    def __init__(self):
        # Initialize MediaPipe Face Mesh
        # Set EAR threshold

    def calculate_ear(self, eye_landmarks):
        # Compute Eye Aspect Ratio
        # Return float value

    def get_eye_landmarks(self, landmarks, indices, frame_w, frame_h):
        # Extract specific eye landmarks
        # Return list of (x, y) coordinates

    def process_frame(self, frame):
        # Process single frame
        # Return annotated frame

    def run(self):
        # Main loop: capture, process, display
        # Handle keyboard input
```

Testing Checklist

Before submission, verify:

- ❖ Webcam initializes successfully
- ❖ Face detection works in various lighting conditions
- ❖ Eye state correctly identified when deliberately blinking
- ❖ System handles "no face detected" without crashing
- ❖ 'q' key exits program cleanly
- ❖ No memory leaks (program runs >5 minutes without issues)
- ❖ Console shows clear status messages
- ❖ Video display updates smoothly
- ❖ EAR values are reasonable (0.15-0.35 range typically)

Common Challenges & Hints

1st – MediaPipe Coordinate System

Problem: MediaPipe returns normalized coordinates (0.0-1.0)

Solution: Multiply by frame width/height to get pixel coordinates

2nd – Wrong Landmarks

Problem: Eye contours are seeming incorrect

Solution: Double-check landmark indices against MediaPipe documentation

3rd – Threshold Too Sensitive

Problem: False positives/negatives for eye state

Solution: Test threshold values between 0.18-0.25, find what works best

4th – Performance Issues

Problem: Video feed shows signs of lagging

Solution: Process every 2nd or 3rd frame, or reduce video resolution

5th – No Face Detected

Problem: MediaPipe cannot find any face

Solution: Ensure good lighting, face camera directly, check camera permissions

Optional – Stretch Goals

If you finish early, try these enhancements:

Level 1: Enhanced Features

- Track blink count over time
- Calculate blink frequency (blinks per minute)
- Add blink duration measurement
- Support both eyes independently (detect winking)
- Save session data to CSV (timestamp, EAR, state)
- Add configurable threshold via keyboard input
- Implement temporal smoothing (reduce jitter)
- Support video file input (not just webcam)

Level 2: Advanced Functionality

Level 3: Research Extensions

- Implement drowsiness detection (extended closed eyes)
- Add gaze direction estimation
- Compare EAR vs. other eye state algorithms
- Write technical report on findings

Resources

Documentation

[OpenCV Python Tutorials] [MediaPipe Face Mesh Guide] [NumPy Documentation]

Research Papers

- Soukupová & Čech (2016): "Real-Time Eye Blink Detection using Facial Landmarks"
- Original EAR algorithm paper (provided separately)

Submission Guidelines

Deliverables

1. **Source Code**
    - Submit via GitHub repository
    - Include all required files
    - Add .gitignore for virtual environments
2. **Documentation**
    - README.md with:
        - Project description
        - Installation instructions
        - Usage examples
        - Known limitations
    - Inline code comments
3. **Demo Video** (1-3 minutes)
    - Screen recording showing:
        - Program startup
        - Eye detection working
        - Deliberate blinking demonstration
        - Edge case handling (no face, looking away)
    - Verbal explanation of approach
    - Submit in Google Drive and share with Eugene
4. **Reflection Document** (1 page)
    - What you learned
    - Challenges faced and solutions
    - How this relates to our swimming platform
    - Ideas for improvement

Format: Please email all deliverables directly to Eugene and Zac (zachary@scott-murphy.com). The email should include the GitHub repo link, the Google Drive containing the demo video and reflection doc, and any notes/context you'd like to add.

Connection to Our Platform

This eye tracking project introduces you to the same foundational techniques we use for swimming analysis:

Eye Tracking	Swimming Analysis
Face landmark detection	Body pose detection
Ear Calculation	Joint angle calculation
Open/Closed Classification	Stroke classification
Threshold-based rules	Technique evaluation
Real-time video processing	Live pool feedback
Webcam input	Pool camera input

Understanding these concepts prepares you for contributing to our core swimming stroke analysis system.