Technologies for autonomous vehicles assignment

Laura Scigliano
s331082@studenti.polito.it

***Abstract*—This work presents a system for driver monitoring using computer vision, implemented in Python with MediaPipe. The system calculates the Eye Aspect Ratio (EAR), PERCLOS, and estimates the gaze direction and head orientation. An alarm is triggered in case of drowsiness or distraction, based on specific angular and temporal criteria.**

I. Introduction

This work describes a system for monitoring driver attention, developed in Python using the *MediaPipe* and *OpenCV* libraries. The approach is designed to detect two main alert signals: drowsiness and visual distraction.

The first part of the code focuses on eye blink analysis through the calculation of the *Eye Aspect Ratio* (EAR). To avoid arbitrary thresholds, the system performs an initial automatic calibration phase at startup: a maximum EAR value is acquired during an open-eye condition, which will then be used as a reference. If the EAR value remains over 80% of the maximum value for a 10 second interval, an alarm message is triggered, indicating a drowsy state.

Next, the PERCLOS 80 parameter is calculated, which measures the percentage of time the eyes are closed within a sliding time window.

The final part of the code is dedicated to estimating the head and gaze orientation using 3D-2D landmarks and Rodrigues rotations. The position of the nose and eyes is projected into space to determine if the direction of the face or gaze deviates more than $\pm 30°$ from the neutral orientation. If this occurs, a visual distraction alert is raised.

II. Calibration Phase

This phase involves a series of steps to set baseline values for the Eye Aspect Ratio (EAR) while the user interacts with the system. The process is initiated automatically upon startup, guiding the user to keep their eyes open and closed for specific intervals.

```
# CALIBRATION
if not calibration_done:
    cv2.putText(image, 'CALIBRATION PHASE',
        (20, 50), cv2.FONT_HERSHEY_SIMPLEX, 1,
         (0, 0, 0), 1)
    calibration_done = Calibration_timer(start
        )
    first_count_down_timer = count_down_timer
        (3, start, 'Keep your eyes OPENED for
        ')
    if first_count_down_timer == 'DONE' and
        second_count_down_timer != 'DONE':
        second_count_down_timer =
            count_down_timer(6, start, 'Keep
            your eyes CLOSED for ')
```

First of all, in this phase, it is verified that the calibration has not yet been completed through the variable *calibration_done*.

During the calibration phase, the system displays the message *CALIBRATION PHASE* on the screen to inform the user that the calibration process is in progress.

The process begins with the function *Calibration_timer()*, which is essentially a timer responsible for ending the calibration phase after 6 seconds from the opening of the window. If the time has not yet expired (i.e., 6 seconds have not yet passed), the function *EAR_calibration()* is called, which calculates the EAR for each eye and stores the results in two separate lists (one for each eye).

```
# First half of Calibration_timer function
end = time.time()
result = end - start
if result <= 6:
    # Save calibration samples
    EAR_calibration()
```

Finally, once the *Calibration_timer()* ends, the minimum and maximum EAR values are calculated by extracting the respective minimum and maximum values previously stored in the lists. At this stage, the 80% and 20% thresholds of the maximum EAR value are also determined. Finally, the variable *calibration_done* is set to `True`, indicating that the calibration has been successfully completed. At this point, the system has acquired the reference values necessary for the real-time monitoring of EAR, allowing it to detect situations of drowsiness or distraction while driving.

It is important to note that this calibration method is a simplified approach, which is not perfectly precise. It assumes that the driver correctly performs the task by not opening the eyes too wide or closing them too tightly in order to achieve more accurate minimum and maximum values.

III. EAR Calculation and Drowsiness Detection

The core of the drowsiness detection system is based on the calculation of the Eye Aspect Ratio (EAR). The EAR is calculated for both the left and right eyes, and if either of the values exceeds a predefined threshold of 80% for more than 10 seconds, the system triggers an alert indicating that the driver may be drowsy.

```
if (ear_left > treshold_left_80) or (
    ear_right > treshold_right_80):
    if start_closure_timer == True:
        start_closure_timer = False
        start_timer = time.time()
    end = time.time()
    totalTime = end - start_timer
    if (totalTime > 10):
```

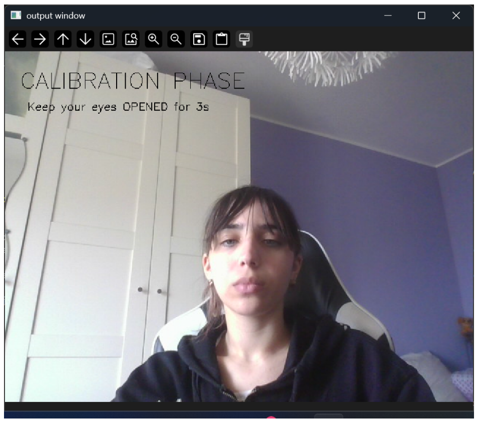

Fig. 1: Descrizione dell'immagine

```
            cv2.putText(image, 'DROWSY DRIVER'
                , (50, 200), cv2.
                FONT_HERSHEY_SIMPLEX, 1, (0,
                0, 255), 1)
    else:
        start_closure_timer = True
        start_timer = 0
```

This monitoring phase begins after the calibration is completed. Initially, the EAR is calculated for each eye and normalized using the *normalization()* function, which scales the EAR value into a range between 0 and 1, where 0 corresponds to the minimum value obtained during calibration, and 1 corresponds to the maximum value obtained during the calibration phase.

```
def normalization(current_value, side):
    if side == 'left':
        return (current_value - min_left_EAR)
            / (max_left_EAR - min_left_EAR)
    else:
        return (current_value - min_right_EAR)
             / (max_right_EAR - min_right_EAR)
```

The actual monitoring phase is managed by a conditional statement that checks whether the EAR values of both the right and left eyes exceed the 80% threshold. If the condition is met (i.e., the eyes are almost fully open), a timer is started to monitor how long the EAR remains above 80%. At the same time, *start_closure_timer* is set to `False` to prevent the timer from being repeatedly restarted.

```
    totalTime = end - start_timer
    if (totalTime > 10):
        cv2.putText(image, 'DROWSY DRIVER',
            (50, 200), cv2.
            FONT_HERSHEY_SIMPLEX, 1, (0, 0,
            255), 1)
```

If the EAR remains above 80% for more than 10 seconds, the system outputs a message indicating that the driver is drowsy.

When the EAR value drops below the threshold, the timer is reset and *start_closure_timer* is triggered to `True`, so that the timer can be restarted once the EAR exceeds the 80% threshold again.

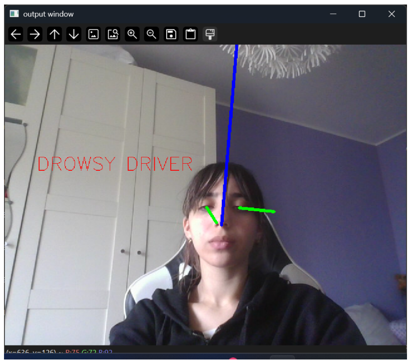

Fig. 2: Drowsy driver allert

IV. Calculation of PERCLOS 80

The PERCLOS computation method used in this system is based on tracking the eye state through distinct phases of a blink cycle. Eye state transitions are detected using predefined EAR thresholds (80% for open eyes and 20% for closed eyes). Each transition starts a timer specific to that state. A control variable, *blink_phase*, ensures that all phases occur in the correct order. The PERCLOS value is calculated. Once calculated only after completing the full blink cycle, then timers and states are reset for the next cycle.

```
# Phase 1 of PERCLOS calculation cycle
if blink_phase == 1 and ((
    previous_ear_left > ear_left and
    ear_left >= treshold_left_80) or (
    previous_ear_right > ear_right and
    ear_right >= treshold_right_80)):
    if start_timer_t1 == True:
        start_timer_t1 = False
        start_t1 = time.time()
    end = time.time()
    t[0] = end - start_t1
else:
    start_timer_t1 = True
    start_t1 = 0
    # Change phase t1 -> t2
    if blink_phase == 1 and ((
        previous_ear_left > ear_left and
        ear_left < treshold_left_80) or (
        previous_ear_right > ear_right and
         ear_right < treshold_right_80)):
        blink_phase = 2
```

In the first phase of the PERCLOS calculation cycle, the initial condition is checked. The variable *blink_phase* must be equal to 1, and the previous EAR value must be greater than the current one, while the current EAR must still be greater than the 80% threshold (for both eyes). This condition ensures that the eye is open and has started closing.

If this condition is satisfied and the timer for this phase (*start_t1*) has not yet been started, it is initialized by setting the variable *start_timer_t1* to *False*, preventing the timer from restarting at each frame.

During each frame, the duration of phase 1 (*t[0]*) is updated as the difference between the current time and the start time *start_t1*. Essentially, this accumulates the duration during which the eye is in the process of closing but still above the 80% threshold.

If the main `if` condition is not satisfied, the timer is reset to be ready for a new cycle. Subsequently, the transition condition to the next phase is checked. This condition ensures that the system is still in *blink_phase = 1* and that the EAR value has dropped below the 80% threshold while the previous EAR value was greater than the current one—indicating that the eye is continuing to close and is now entering the next phase.

```
# Passage block between two phases
if blink_phase == 1 and ((
    previous_ear_left > ear_left and
    ear_left < treshold_left_80) or (
    previous_ear_right > ear_right and
    ear_right < treshold_right_80)):
     blink_phase = 2
```

Each of the four phases for calculating the PERCLOS follows the same structure. For every phase, an initial condition is checked, which typically corresponds to:

- **Phase 1:** The eye is open above the 80% threshold and begins to close.

```
blink_phase == 1 and ((
    previous_ear_left > ear_left and
    ear_left >= treshold_left_80) or (
    previous_ear_right > ear_right and
    ear_right >= treshold_right_80))
```

- **Phase 2:** The eye opening is below the 80% threshold and continues to close.

```
blink_phase == 2 and ((
    previous_ear_left > ear_left and
    treshold_left_80 > ear_left >=
    treshold_left_20) or (
    previous_ear_right > ear_right and
    treshold_right_80 > ear_right >=
    treshold_right_20))
```

- **Phase 3:** The eye opening is below the 20% threshold.

```
blink_phase == 3 and ((ear_left <
    treshold_left_20) or (ear_right <
    treshold_right_20))
```

- **Phase 4:** This phase includes both the previous fully closed state and the final opening movement of the eye up to the 80% threshold.

```
(blink_phase == 4 or blink_phase == 3)
     and (((previous_ear_left <
    ear_left and treshold_left_80 >=
    ear_left >= treshold_left_20) or
    ear_left < treshold_left_20) or ((
    previous_ear_right < ear_right and
    treshold_right_80 >= ear_right >=
    treshold_right_20) or ear_right <
    treshold_right_20))
```

When the initial condition is satisfied, the timer related to the current phase is started; otherwise, the timer is reset and the conditions for transitioning to the next phase are checked.

When phase 4 is completed, the PERCLOS calculation is performed:

```
# Calculation of PERCLOS
if blink_phase == 5:
    perclos = calculate_PERCLOS_80(t)
    t = [0, 0, 0, 0]
    # Reset blink_phase if finished
    blink_phase = 1

def calculate_PERCLOS_80(t):
    P80 = (t[2] - t[1]) / (t[3] - t[0]) if
         t[3] != t[0] else 0
    return P80
```

The function *calculate_PERCLOS_80()* computes the PERCLOS value using the time durations obtained from the previous four phases. Finally, the time values and the *blink_phase* variable are reset in order to restart the cycle from the first phase.

V. Eyes and Head gaze positions

After estimating the rotations and translations required to map from the 3D world to the 2D image for the face and eyes, and after converting the rotation vectors into rotation matrices and then into Euler angles, the next step is to determine whether the driver is distracted. This is done by analyzing the *yaw* angle values of both the face and the eyes.

```
# Check if the driver is distracted
if np.abs(yaw) > 30 and np.abs(
    yaw_left_eye) > 30 and np.abs(
    yaw_right_eye) > 30:
    cv2.putText(image, 'DISTRACTED DRIVER'
        , (50, 300), cv2.
        FONT_HERSHEY_SIMPLEX, 1, (0, 0,
        255), 1)
```

If the absolute value of the *yaw* angles exceeds 30°, a warning message `DISTRACTED DRIVER` is displayed on the screen.

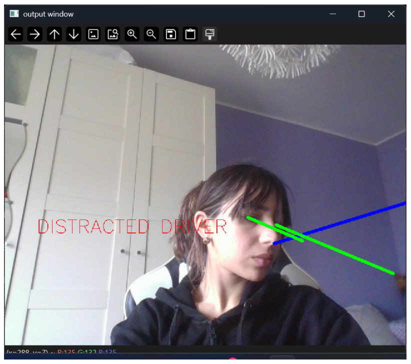

Fig. 3: Distracted driver allert