DÉTECTION DE FATIGUE PAR ANALYSE DE CYCLES PHYSIOLOGIQUES

Thème : Cycles et Boucles

Réalisé par : AMINE Ahmed
Encadré par : FENKOUCH Ahmed
Code cnc : MH081T

SOMMAIRE :

I. INTRODUCTION

+ La somnolence au volant représente un facteur majeur d'accidents

+ Concevoir un dispositif de prévention de la somnolence.

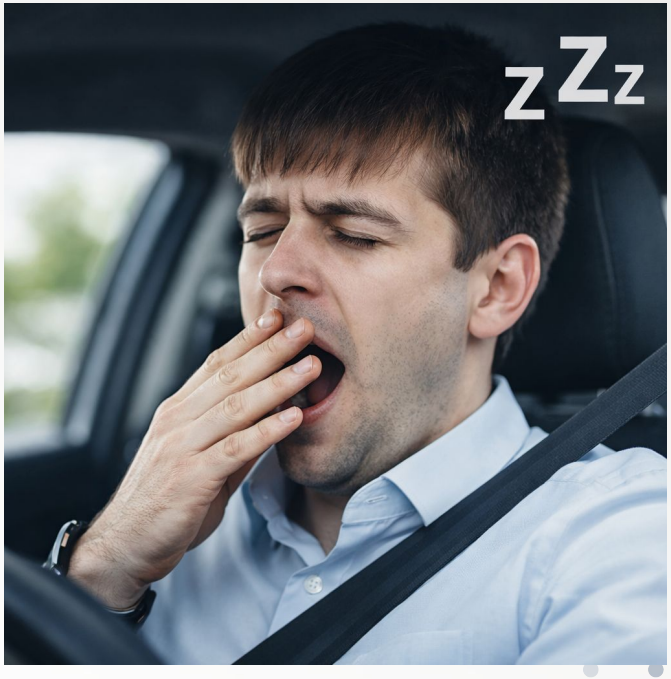


Figure 01: conducteur fatigué au volant

II. Principe de fonctionnement

- Surveillance en continu du conducteur.
- Détection des signes de somnolence.
- Un système qui agit avant l'accident.

Figure 02: Interaction entre le conducteur et le système de détection de la fatigue

II. PRINCIPE DE FONCTIONNEMENT

Une boucle de surveillance continue garantit la détection en temps réel

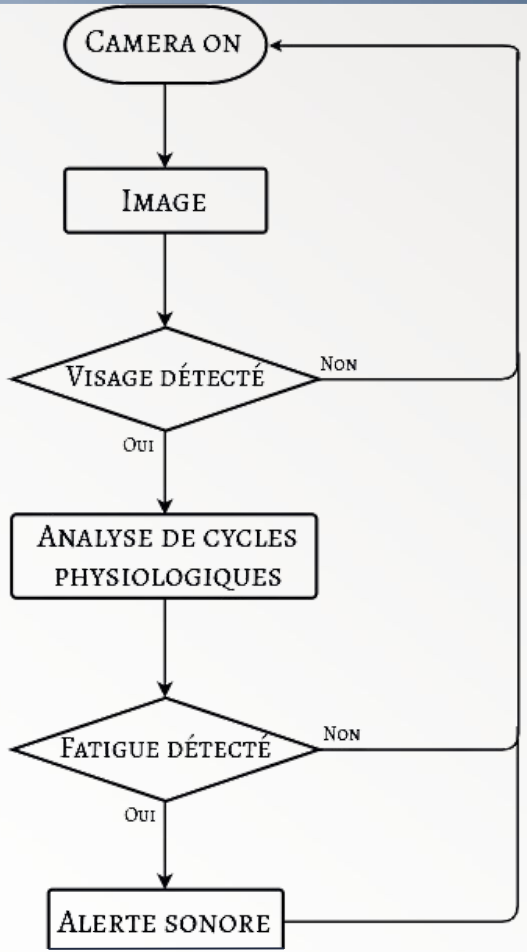


Figure 03: Architecture fonctionnelle du système

II. PRINCIPE DE FONCTIONNEMENT

"DIAGRAMME DE CAS D'UTILISATION"

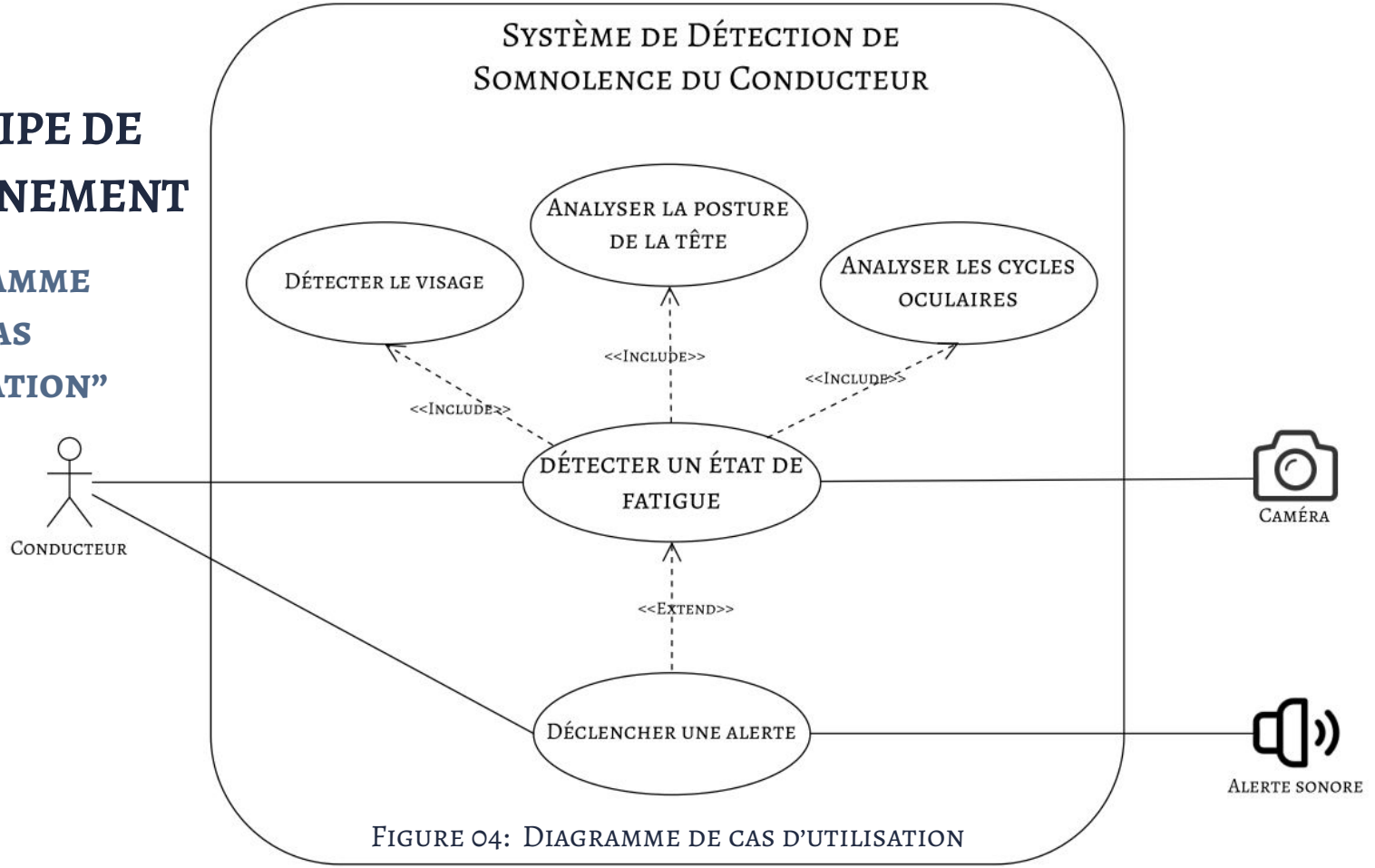


FIGURE 04: DIAGRAMME DE CAS D'UTILISATION

II. PRINCIPE DE FONCTIONNEMENT

"DIAGRAMME DES EXIGENCES"

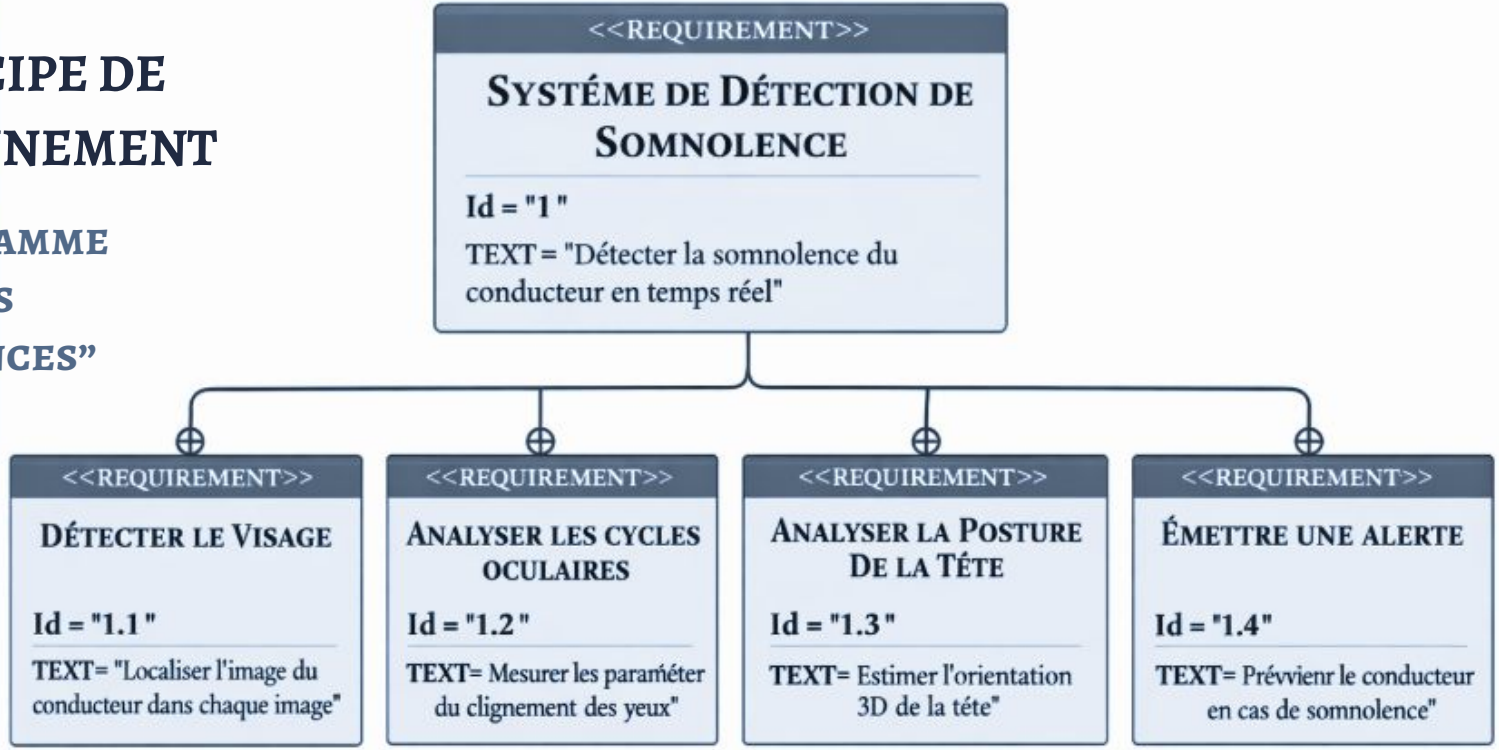


FIGURE 05: DIAGRAMME DES EXIGENCES

III

PROBLEMATIQUE

Comment analyser les perturbations de cycles physiologiques (oculaires et posturaux) dans une boucle de traitement vidéo temps réel pour détecter la somnolence au volant?

IV. OBJECTIFS

- **Objectif 1 :** Détecter le visage et analyser :
  - Sous-objectif 1.1 : Cycles oculaires
  - Sous-objectif 1.2 : Posture de la tête
- **Objectif 2 :** Système d'avertissement
  - Logique de décision
- **Objectif 3 :** Validation expérimentale
  - Tests + résultats

IV. 1. Objectif 01 :

⤮ « Présentation d'objectif »

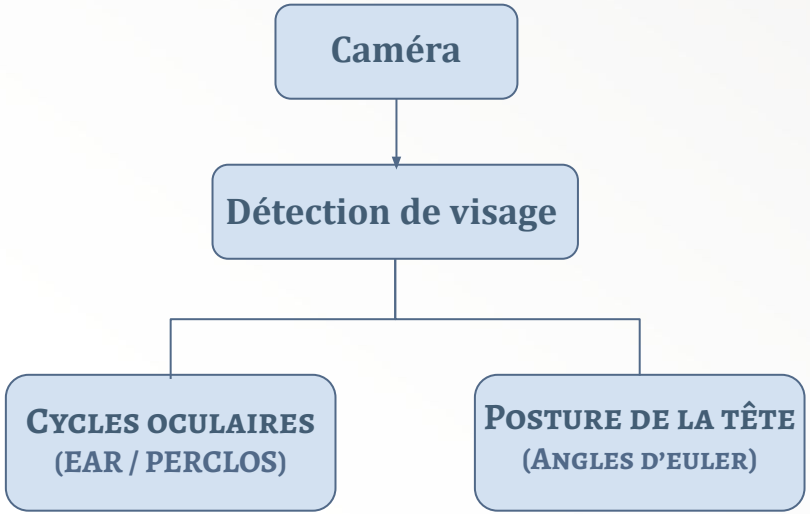


Figure 06: Schéma bloc d'analyse des cycles physiologiques

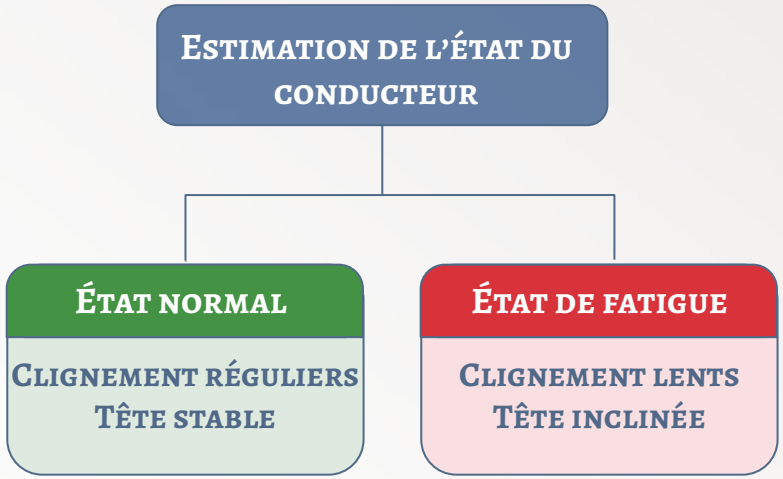


Figure 07: Classification des états du conducteur

IV. 1. Objectif 01 :

⤮ « Outils utilisés »

	**PYTHON** Langage de développement principal
	**OPENCV** Capture et traitement vidéo temps réel
	**DLIB** Détection du visage et landmarks
 	**SCIPY / NUMPY** Calculs mathématiques et distances euclidiennes

Figure 08: Tableau des outils logiciels et leurs logos

IV. 1. Objectif 01 :

🔀 « Détection du visage »

🚨 Principe général :

➔ Localiser le visage dans chaque image du flux vidéo.

➔ Prétraiter les images pour faciliter la détection.

🚨 Capture du flux vidéo :

Capture du flux vidéo avec OpenCV

- Classe `VideoCapture` → ouverture de la caméra.
- Méthode `read()` → lecture image par image.
- Conversion en niveaux de gris `cv2.cvtColor` → simplification du traitement.

IV. 1. Objectif 01 :

⤮ « Détection du visage »

Pourquoi Dlib ?

- OpenCV propose des détecteurs basiques

- Dlib offre une détection plus robuste avec une localisation précise du visage et des performances supérieures.

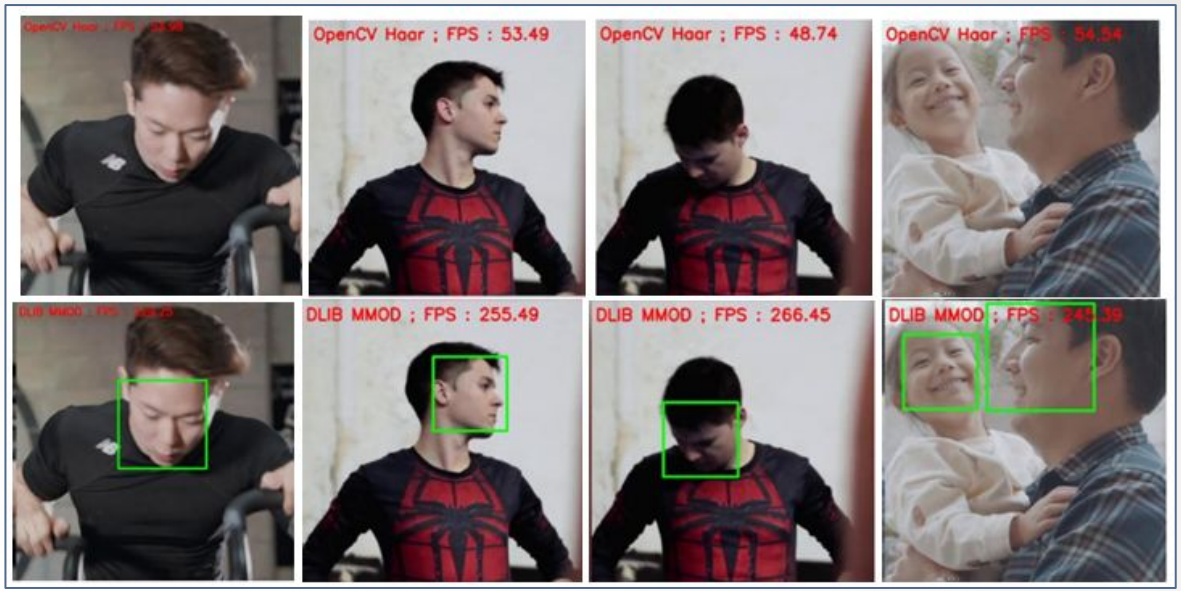


Figure 09: Exemples de détections avec OpenCV et Dlib

IV. 1. Objectif 01 :

« Détection du visage »

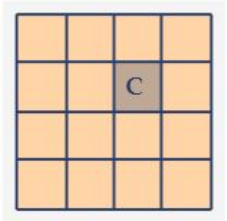
Méthode HOG (Dlib) :

HOG = Histogram of Oriented Gradients

Capturer l'image

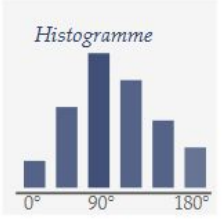
Conversion en gris

Division en cellules 8×8 px



Calcul des gradients

Comparaison au modèle entraîné



Visage détécté

Capture de flux par **OpenCV**

```
detector = dlib.get_frontal_face_detector()
faces = detector(gray, 0)
```

Figure 10: Étapes de la détection du visage par HOG

IV. 1. Objectif 01 :

🔀 « Détection du visage »

🚨 Méthode HOG (Dlib) :

❖ Qu'est-ce qu'un gradient ?

- la variation de luminosité entre deux pixels voisins
- Mesure la magnitude (force de contraste) et la direction
- Caractéristiques des gradients au visage :
  - Zone des yeux → Gradients forts horizontaux
  - Nez → Gradients diagonaux
  - Contour visage → Gradients forts verticaux
  - Front → Peu de gradients (zone uniforme)

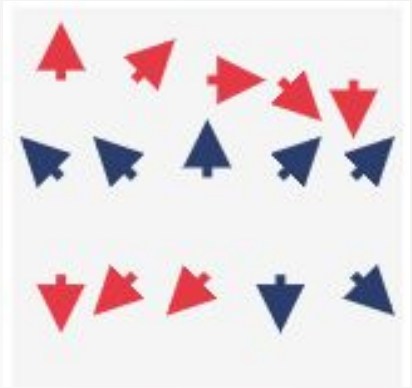


Figure 11: Illustration des gradients d'intensité

IV. 1. Objectif 01 :

« Détection du visage »

Les 68 LANDMARKS :

→ Sur chaque visage détecté

→ 68 points caractéristiques localisés

→ La classe `shape_predictor`

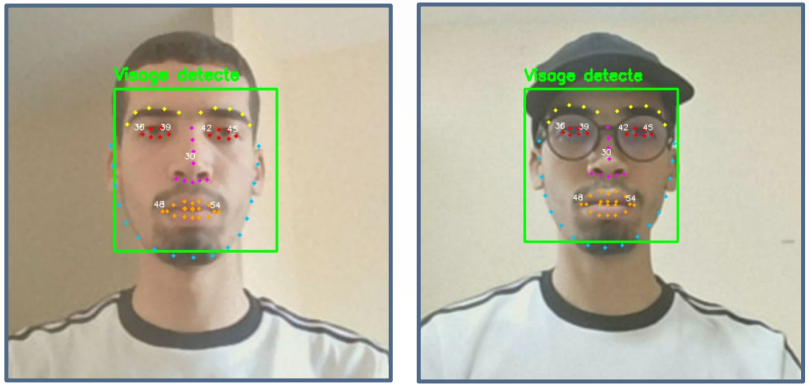


Figure 12: Localisation des 68 landmarks faciaux dans différentes conditions

Figure 13: Localisation des 68 landmarks faciaux

IV. 1. Objectif 01 : sous-objectif 01 :

🔀 “ Analyse de cycles oculaires ”

🚨 Principe général :

- ➔ Suivre l’ouverture et la fermeture des yeux pour détecter la somnolence.
- ➔ Des indicateurs fiables de vigilance.
- ➔ Les cycles oculaires : clignements, fermeture prolongée

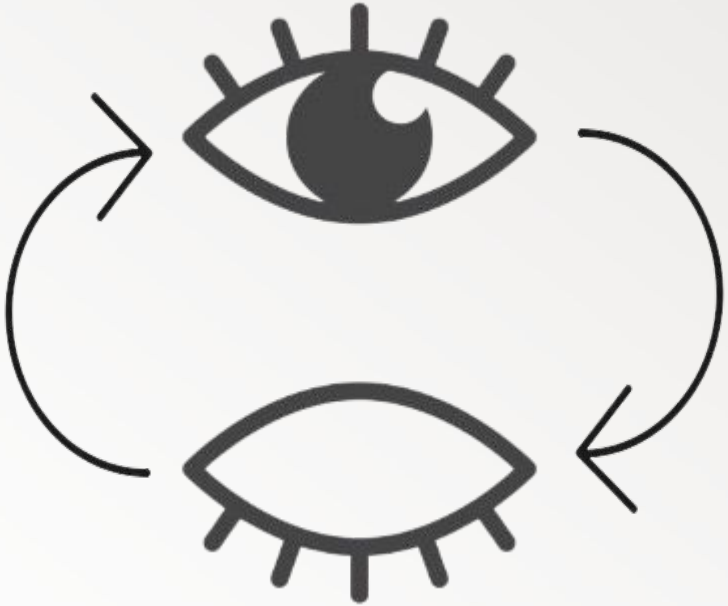


Figure 14: cycles oculaires (œil ouvert/fermé)

IV. 1. Objectif 01 : sous-objectif 01 :

⤮ “ Analyse de cycles oculaires ”

Méthode EAR : (Eye Aspect Ratio)

➔ Mesure l’ouverture de l’œil à partir de 6 points caractéristiques.

➔ Oeil Gauche : points de 37 à 42

➔ Oeil Droit : points de 43 à 48

➔ Diminue lorsque l’œil se ferme

Formule :

$$EAR = \frac{||P2-P6|| + ||P3-P5||}{2 \cdot ||P1-P4||}$$

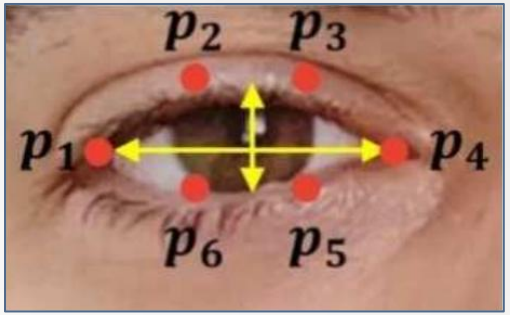


Figure 15: EAR pour œil ouvert

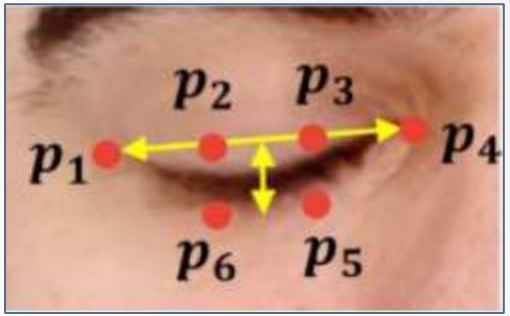


Figure 16: EAR pour œil fermé

IV. 1. Objectif 01 : sous-objectif 01 :

🔀 “ Analyse de cycles oculaires ”

🚨 Méthode EAR :

➔ Valeur finale = moyenne des deux EAR.

$$EAR_{final} = \frac{EAR_{gauche} + EAR_{droite}}{2}$$

➔ Œil **ouvert** : EAR > 0,25

➔ Œil **fermé** : EAR < 0,25

➔ **Clignement** : EAR chute brutalement puis remonte

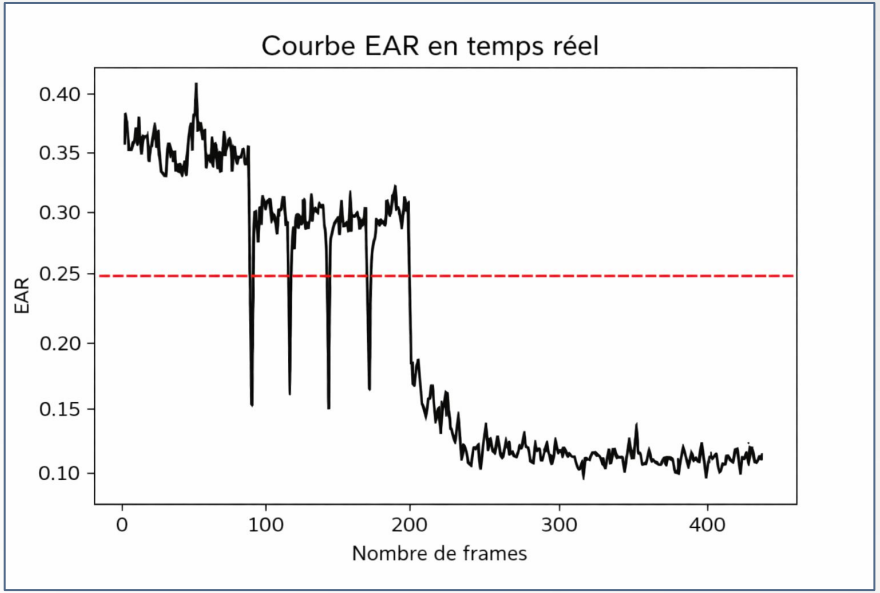


Figure 17: Courbe EAR en temps réel

IV. 1. Objectif 01 : sous-objectif 01 :

🔀 " Analyse de cycles oculaires "

🚨 Résultat de EAR :

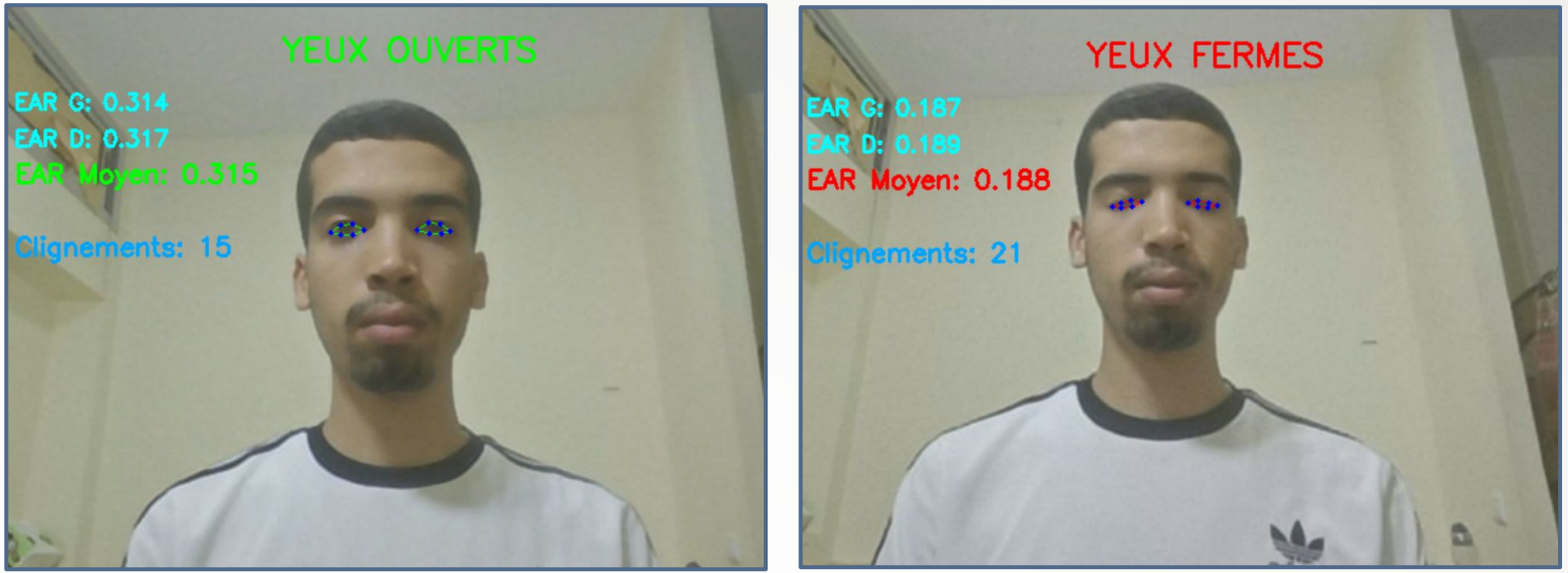


Figure 18: Variation de l'EAR selon l'état des yeux

IV. 1. Objectif 01 : sous-objectif 01 :

🔀 " Analyse de cycles oculaires "

🚨 Méthode PERCLOS : Percentage of Eye Closure

→ Mesure le pourcentage du temps durant lequel les yeux sont fermés (EAR < 0,25) dans une fenêtre de 60 secondes

$$\text{PERCLOS} = \frac{\text{Nombre de frames où EAR} < 0{,}25 \text{ (sur une fenêtre)}}{\text{Nombre total de frames (sur une fenêtre)}} \times 100$$

→ Qu'est-ce qu'un frame ?

→ Qu'est-ce qu'une fenêtre glissante ?

IV. 1. Objectif 01 : sous-objectif 01 :

⤮ « Analyse de cycles oculaires »

Méthode PERCLOS :

» Qu'est-ce qu'une frame ?

→ Une frame = une image capturée par la caméra
→ La caméra capture N frames par seconde (FPS)

→ Durée d'une frame = $\frac{1}{FPS}$

→ La mesure du temps écoulé entre deux instants grâce à la fonction `time.time()`

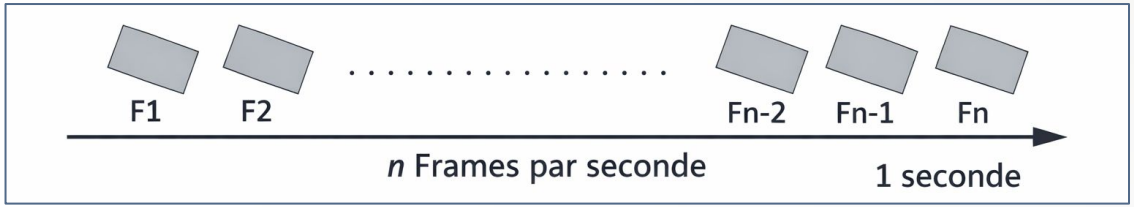


Figure 19: Principe de frames par seconde

IV. 1. Objectif 01 : sous-objectif 01 :

« Analyse de cycles oculaires »

Méthode PERCLOS :

Qu'est-ce qu'une fenêtre glissante ?

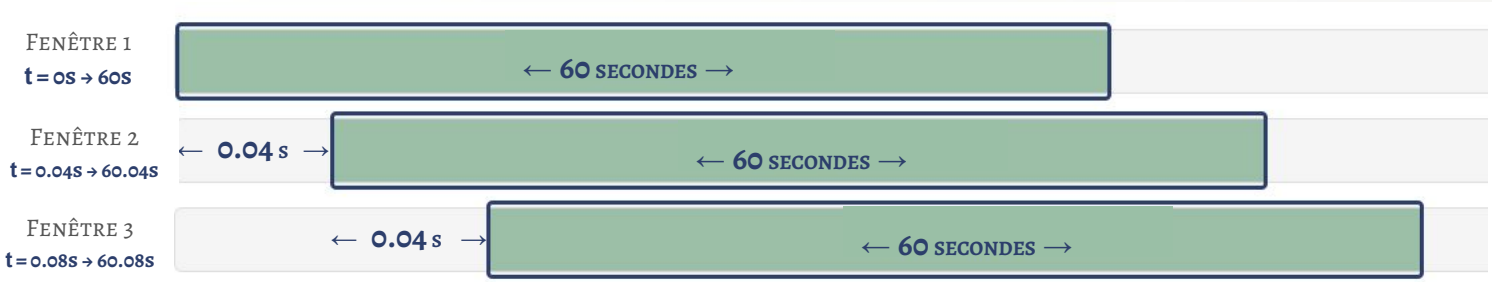
→ Recalculé à chaque frame

→ Fenêtre de 60s qui avance

→ Détection immédiate sans attente

→ ajoute la nouvelle frame avec son état (yeux ouverts/fermés)

→ Une boucle `while` supprime les frames plus de 60 s



Exemple pour :
FPS = 25
1 frame = 1/25= 0.04s
En 1 min :
60 × 25 = 1500 frames analysées

Figure 20: Principe de la fenêtre glissante pour FPS = 25

IV. 1. Objectif 01 : sous-objectif 01 :

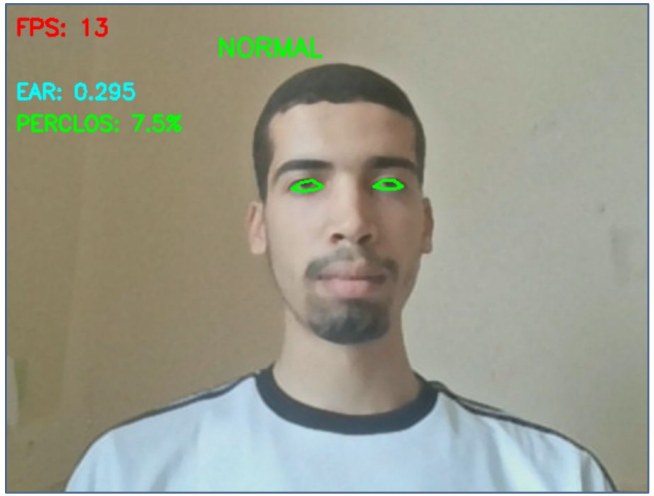
🔀 " Analyse de cycles oculaires "

🚨 Résultat de PERCLOS :

Simulation de l'état:

Normal :

PERCLOS < 15%



Fatigué :

PERCLOS > 15%

Figure 21: Comparaison PERCLOS : état normal vs état fatigué

IV. 1. Objectif 01 : sous-objectif 02 :

🔀 “ Analyse de posture de la tête ”

🚨 Présentation de sous-objectif :

➔ Observer l’orientation de la tête à partir des images capturées.

➔ Décrire les mouvements principaux (inclinaison, rotation, basculement).

➔ Mettre en évidence les variations de posture visibles sur les captures.

Figure 22: Conducteur endormi par posture de la tête

IV. 1. Objectif 01 : sous-objectif 02 :

🔀 " Analyse de posture de la tête "

🚨 Méthode PnP (Perspective-n-Point) :

→ Méthode mathématique qui estime l'orientation 3D d'un objet à partir de points 2D

→ Entrées :

- Points 2D : landmarks du visage (détectés par dlib)
- Points 3D : modèle 3D du visage (coordonnées réelles)

→ Sortie :

- Vecteur de rotation
- Vecteur de translation

→ Code utilisé:

- `cv2.solvePnP` : fonction de OpenCV qui résout le problème PnP
- Elle retourne la pose (position + orientation)

IV. 1. Objectif 01 : sous-objectif 02 :

⤮ « Analyse de posture de la tête »

Méthode PnP :

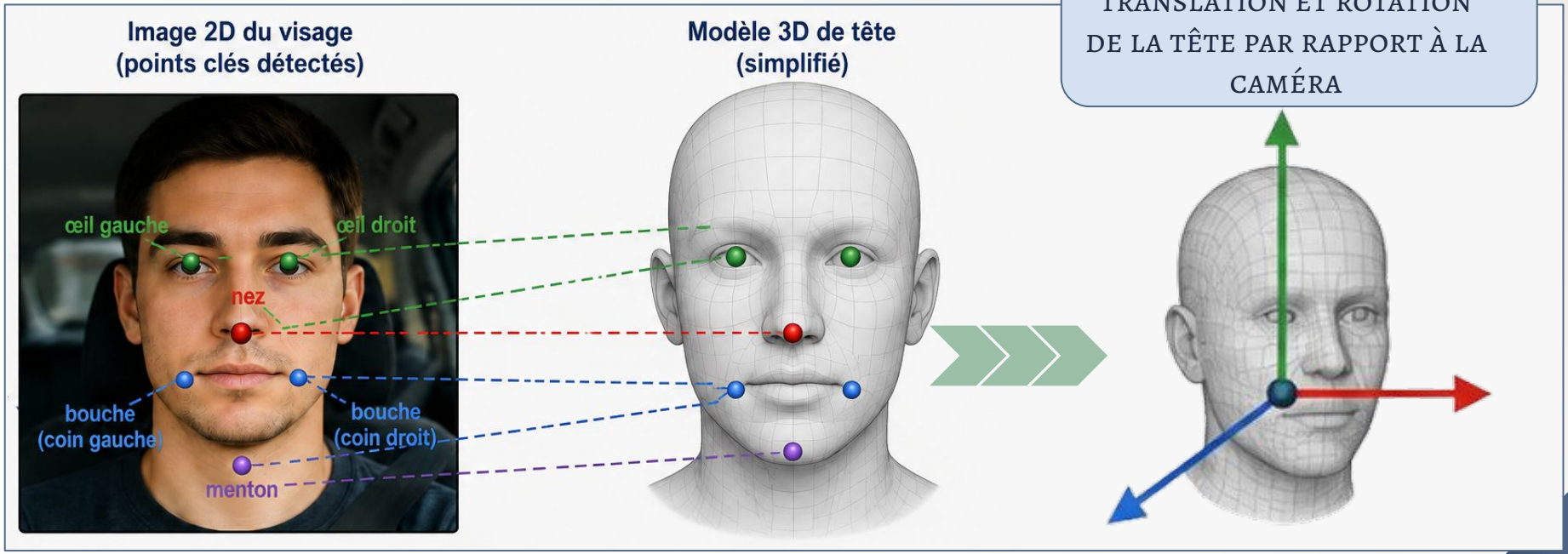


Figure 23: Illustration de la méthode PnP pour la pose du visage

IV. 1. Objectif 01 : sous-objectif 02 :

🔀 " Analyse de posture de la tête "

🚨 Angle d'euler :

→ 3 angles décrivant l'orientation de la tête dans l'espace 3D

→ Extraits du vecteur de rotation retourné par `cv2.solvePnP`

→ Calculés avec la fonction `np.arctan2`

→ À certain valeurs de chaque angle la tête tombe , tourne ou se penche

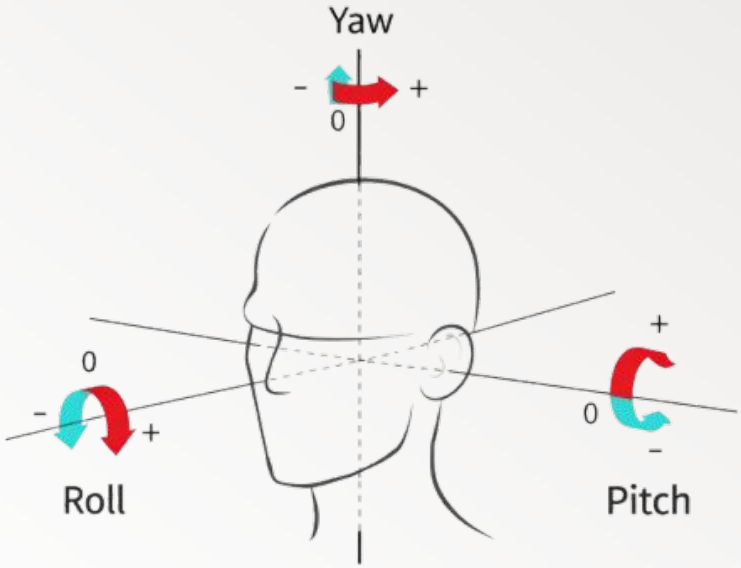


Figure 24: Posture de la tête avec angle d'euler

IV. 1. Objectif 01 : sous-objectif 02 :

🔀 “ Analyse de posture de la tête ”

🚨 Résultats :

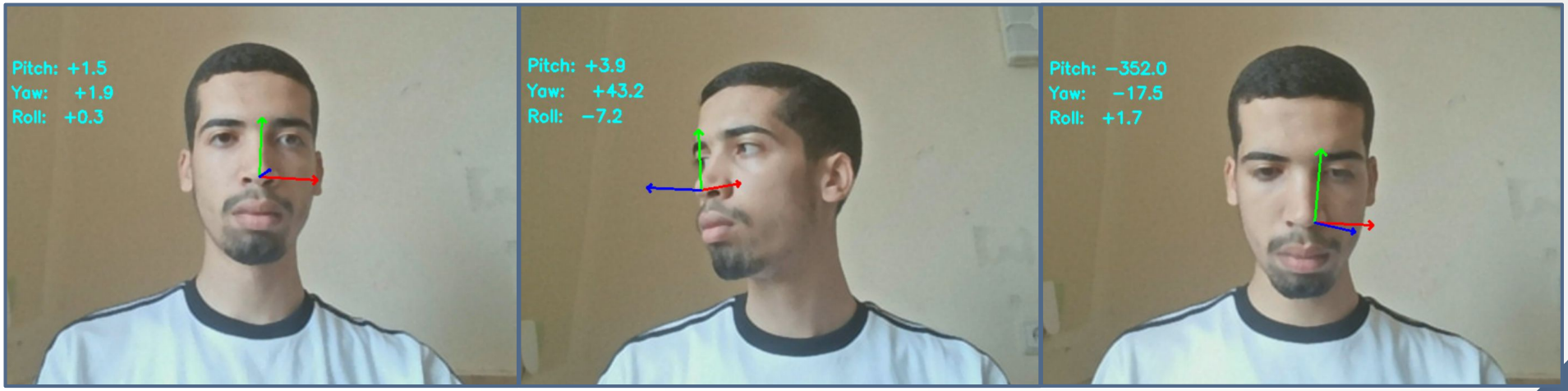


Figure 25: Angle d’euler pour différentes positions de la tête

IV. 2. Objectif 02 :

🔀 “ Systéme d’avertissement ”

🚨 Présentation :

➔ Prévenir les risques d’accident

➔ Analyser les données issues des sous-objectifs précédents (cycles oculaires, posture de la tête)

➔ Implémenter la logique de décision en temps réel

Figure 26 : Illustration du système d’avertissement

IV. 2. Objectif 02 :

🔀 « Systéme d'avertissement »

🚨 Conditions de déclenchement :

Indicateur	Condition	Signification
EAR	EAR < 0,25 pendant > 2s	Yeux fermés
PERCLOS	PERCLOS > 15%	Somnolence modérée
PERCLOS	PERCLOS > 30%	Somnolence sévère
Pitch	|Pitch| > 20°	Tête tombante
Yaw	|Yaw| > 35°	Tête tournée
Roll	|Roll| > 20°	Tête penchée

Figure 27 : Conditions de déclenchement du système d'avertissement

IV. 2. Objectif 02 :

⤮ « Systéme d'avertissement »

(Logique de décision) :

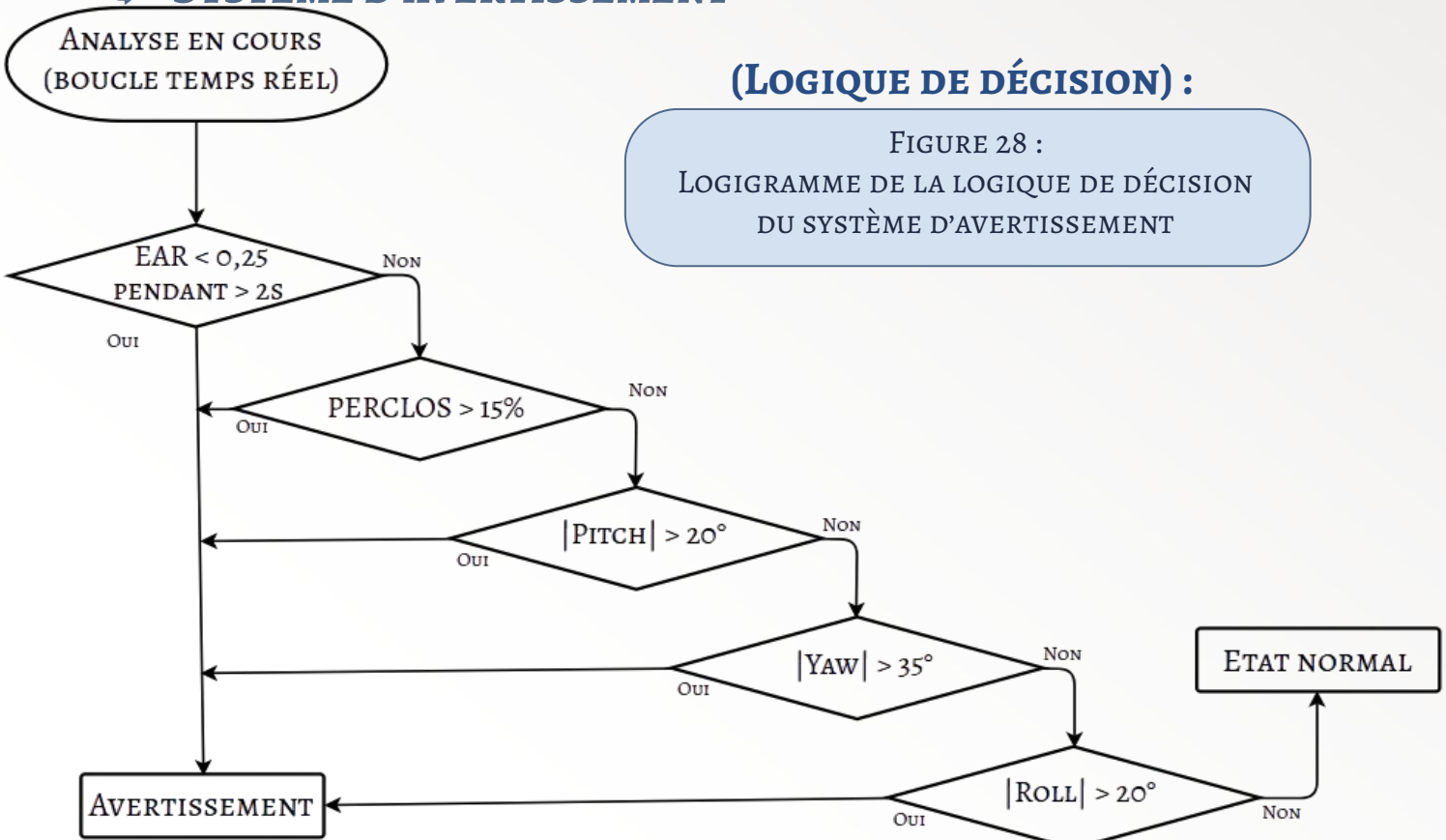


Figure 28 :
Logigramme de la logique de décision
du système d'avertissement

IV. 2. Objectif 02 :

“ Systéme d’avertissement ”

Résultats :

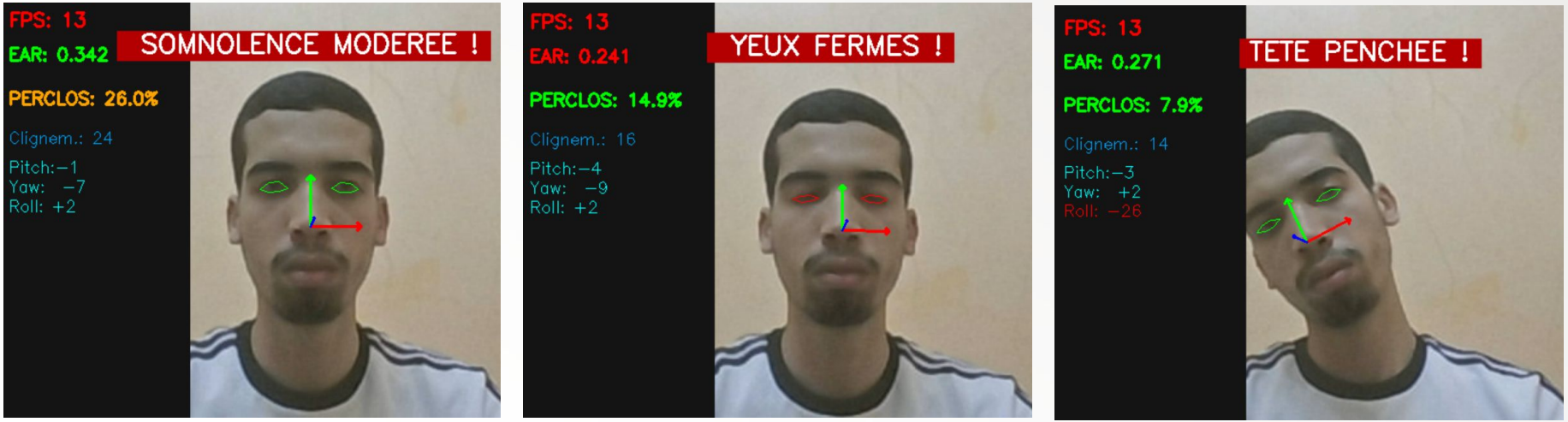


Figure 29 : Exemples de détection de somnelance

IV. 3. Objectif 03 :

🔀 “ Réalisation de prototype ”

🚨 Présentation :

- Construire un prototype fonctionnel
- Utiliser Raspberry Pi, caméra et buzzer
- Intégrer les algorithmes de détection et d’alerte
- Tester en conditions réelles

IV. 3. Objectif 03 :

« Réalisation de prototype »

Matériel utilisé :

Figure 30 : Raspberry pi 3B+

IV. 3. Objectif 03 :

« Réalisation de prototype »

Matériel utilisé :

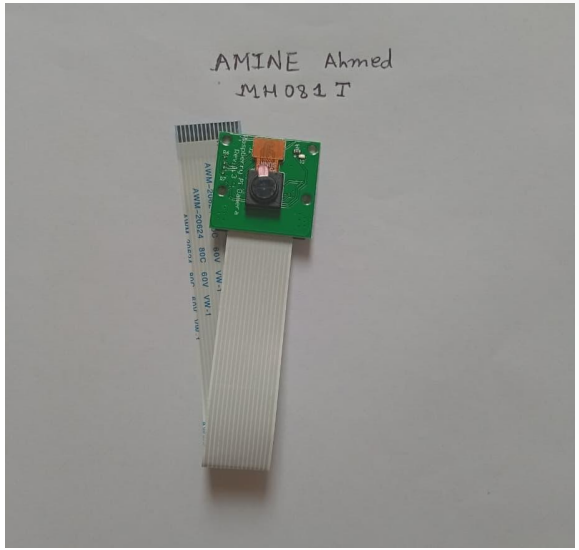


Figure 31 : Caméra pi

Figure 32 : Buzzer

IV. 3. Objectif 03 :

⤮ « Réalisation de prototype »

Architecture :

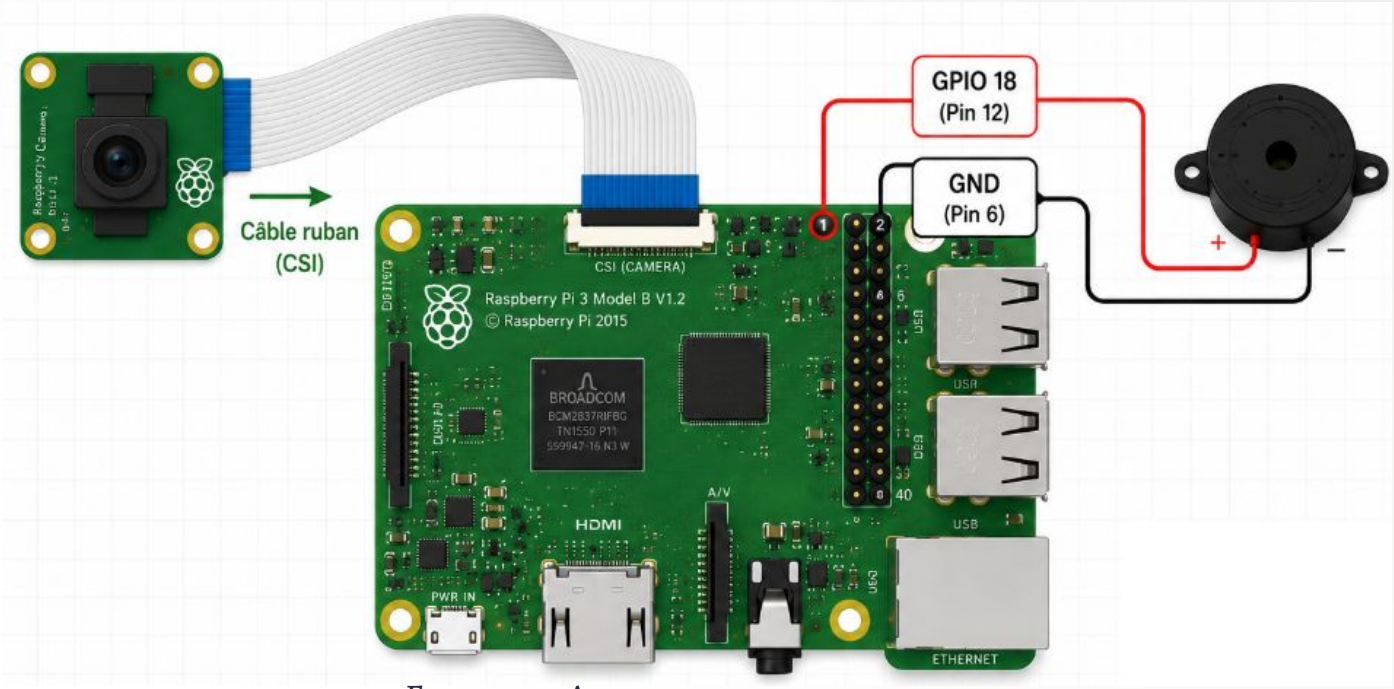


Figure 33 : Architecture de prototype

IV. 3. Objectif 03 :

“ Réalisation de prototype ”

Réalisation:

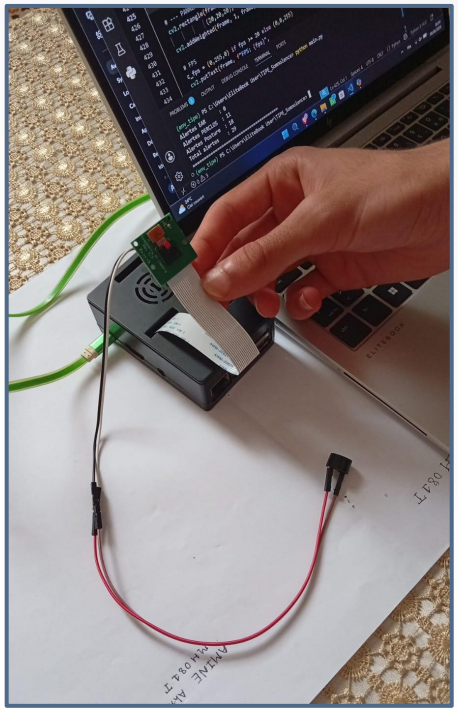
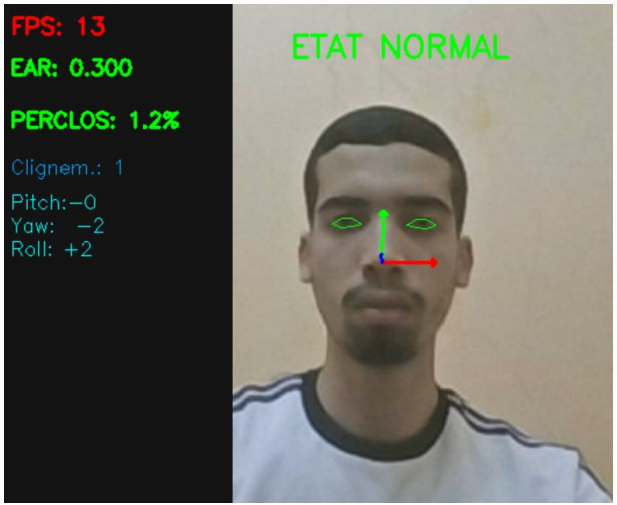


Figure 34 : Résultat de prototype

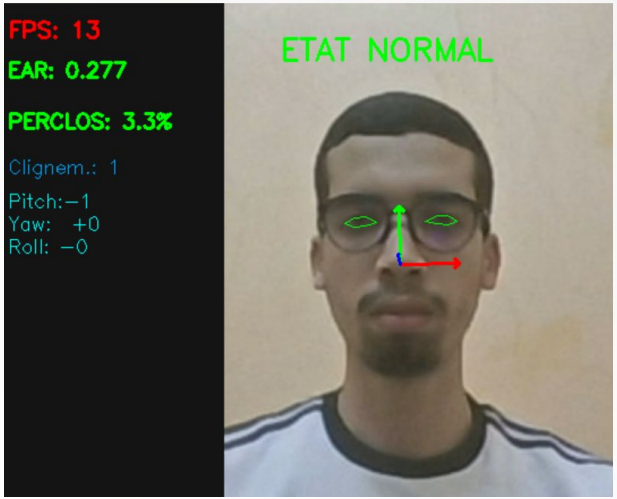
IV. 3. Objectif 03 :

⤮ « Réalisation de prototype »

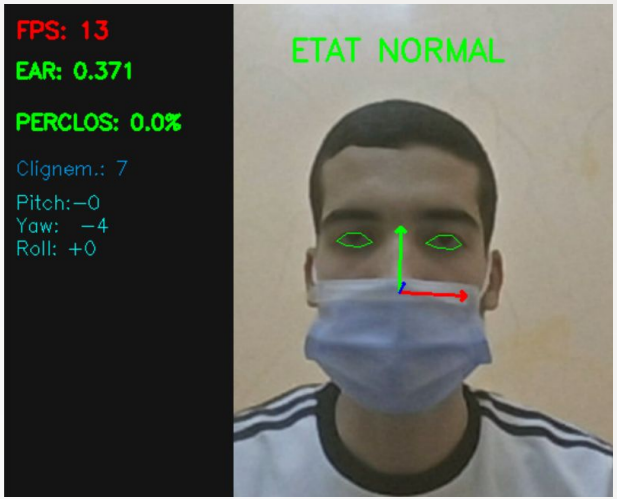
Test :



Visage Nu



Avecc lunette



Avec Bavette

Figure 35 : Différentes conditions de Test

IV. 3. Objectif 03 :

⤮ « Réalisation de prototype » Test :

Type de test	Visage nu	Avec lunettes	Avec Bavette
Nombre de test	10	10	10
Test par cycles oculaires réussit	10	8	4
Test par posture de tête réussit	10	10	3
Fausse alerte	0	2	1
Pas de détection	0	0	6

Figure 36 : résultat de test sous différent condition

V. Conclusion et perspectives

🔀 “ Réponse à la problématique ”

→Le système développé détecte efficacement la somnolence

→ Les cycles oculaires sont analysés via EAR et PERCLOS en temps réel

→ Les cycles posturaux sont estimés via les angles d'Euler (PnP)

→ La boucle de traitement vidéo fonctionne en temps réel

V. Conclusion et perspectives

🔀 « Limites du système »

🚨 Limites techniques :

→ Sensible aux conditions d'éclairage

→ Incompatible avec lunettes de soleil

→ FPS réduit sur Raspberry Pi (~15 FPS) les performances ne sont pas satisfaisantes

🚨 Limites méthodologiques :

→ Tests sur fatigue simulée (pas sur vraie somnolence)

V. Conclusion et perspectives

“ Perspectives ”

→ Une raspberry pi plus performant

→ Caméra infrarouge (nuit)

→ Ajout d'autres indicateurs

→ Intégration dans un véhicule réel

→ Connexion smartphone/tableau de bord

V

CONCLUSION

MERCI POUR VOTRE ATTENTION

ANNEXE

```python
import cv2
import dlib
import imutils
import numpy as np
import time, threading
import winsound
from scipy.spatial import distance as dist
from collections import deque

EAR_SEUIL        = 0.25
DUREE_ALERTE_EAR = 2.0
FRAMES_SEUIL     = 3

FENETRE_PERCLOS  = 60
PERCLOS_MODERE   = 15
PERCLOS_SEVERE   = 30

PITCH_SEUIL = 20
YAW_SEUIL   = 35
ROLL_SEUIL  = 20

LEFT_EYE_START  = 36
LEFT_EYE_END    = 42
RIGHT_EYE_START = 42
RIGHT_EYE_END   = 48
INDICES_PNP     = [30, 8, 36, 45, 48, 54]

detector  = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)
```

Annexe 01 : Initialisation du système

```python
compteur_frame += 1
if compteur_frame % 2 == 0:
    faces_cache = detector(gray, 0)
faces = faces_cache

for face in faces:
    landmarks = predictor(gray, face)
    left_eye  = [(landmarks.part(n).x, landmarks.part(n).y) for n in range(36, 42)]
    right_eye = [(landmarks.part(n).x, landmarks.part(n).y) for n in range(42, 48)]
```

Annexe 02 : Détection du visage (HOG) et landmarks

```python
def calculer_EAR(eye):

    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])
    C = dist.euclidean(eye[0], eye[3])
    return (A + B) / (2.0 * C)

ear_g     = calculer_EAR(left_eye)
ear_d     = calculer_EAR(right_eye)
ear_moyen = (ear_g + ear_d) / 2.0

couleur = (0,255,0) if ear_moyen >= EAR_SEUIL else (0,0,255)
cv2.polylines(frame, [np.array(left_eye, dtype=np.int32)], True, couleur, 1)
cv2.polylines(frame, [np.array(right_eye, dtype=np.int32)], True, couleur, 1)
```

Annexe 03 : Calcul et traçage de l'EAR

```python
etat_yeux = 1 if ear_moyen < EAR_SEUIL else 0

if etat_yeux == 1:
    compteur_ferme += 1

historique_perclos.append((temps_actuel, etat_yeux))

while (historique_perclos and temps_actuel - historique_perclos[0][0] >
FENETRE_PERCLOS):
    if historique_perclos[0][1] == 1:
        compteur_ferme -= 1
    historique_perclos.popleft()

if len(historique_perclos) > 0:
    perclos = (compteur_ferme / len(historique_perclos)) * 100
else:
    perclos = 0
```

Annexe 04 : Calcul du PERCLOS (fenêtre glissante)

```python
MODEL_3D = np.array([
    ( 0.0,    0.0,    0.0),
    ( 0.0,  -63.6, -12.5),
    (-43.3,   32.7, -26.0),
    ( 43.3,   32.7, -26.0),
    (-28.9, -28.9, -24.1),
    ( 28.9, -28.9, -24.1)
], dtype=np.float64)

points_2d = np.array([(landmarks.part(i).x, landmarks.part(i).y) for i in
INDICES_PNP], dtype=np.float64)

h, w = frame.shape[:2]
mat_cam = np.array([
    [w, 0, w/2],
    [0, w, h/2],
    [0, 0, 1]
], dtype=np.float64)

succes, vrot, vtrans = cv2.solvePnP(MODEL_3D, points_2d, mat_cam, np.zeros((4,1)),
flags=cv2.SOLVEPNP_ITERATIVE)
```

Annexe 05 : Estimation de la posture de la tête

```python
matrice_rot, _ = cv2.Rodrigues(vrot)

pitch = np.degrees(np.arctan2(matrice_rot[2][1], matrice_rot[2][2]))

yaw = np.degrees(np.arctan2(-matrice_rot[2][0], np.sqrt(matrice_rot[2][1]**2 +
matrice_rot[2][2]**2)))

roll = np.degrees(np.arctan2(matrice_rot[1][0], matrice_rot[0][0]))

if calibre:
    pitch_rel = pitch - pitch_ref
    yaw_rel   = yaw   - yaw_ref
    roll_rel  = roll  - roll_ref
```

Annexe 06 : Calcul des angles d'Euler

```python
if ear_moyen < EAR_SEUIL:
    if temps_yeux_fermes is None:
        temps_yeux_fermes = temps_actuel
    elif (temps_actuel - temps_yeux_fermes > DUREE_ALERTE_EAR):
        alerte_ear = True
else:
    temps_yeux_fermes = None

if perclos >= PERCLOS_MODERE:
    alerte_perclos = True

if   abs(pitch_rel) > PITCH_SEUIL:
    alerte_posture = True; raison = "TETE TOMBANTE"
elif abs(yaw_rel)   > YAW_SEUIL:
    alerte_posture = True; raison = "TETE TOURNEE"
elif abs(roll_rel)  > ROLL_SEUIL:
    alerte_posture = True; raison = "TETE PENCHEE"

alerte_globale = alerte_ear or alerte_perclos or alerte_posture

import RPI.GPIO as GPIO
BUZZER_PIN = 18
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER_PIN, GPIO.OUT)
GPIO.output(BUZZER_PIN, GPIO.LOW)

def jouer_alerte():
    GPIO.output(BUZZER_PIN, GPIO.HIGH)
    time.sleep(0.5)
    GPIO.output(BUZZER_PIN, GPIO.LOW)

if alerte_globale :
    jouer_alerte()
```

Annexe 07 : Décision et avertissement