Avance de Sprints - Modelo basado en IA para detección de somnolencia

**Fecha:**24/04/2025

**Autor: Tirza Buendia**

**Versión:** 1.0

**1. Introducción**

En este entregable se hará la documentación del avance del Modelo basado en IA para la detección de somnolencia. El avance será documentado según las tareas planteadas en cada sprint.

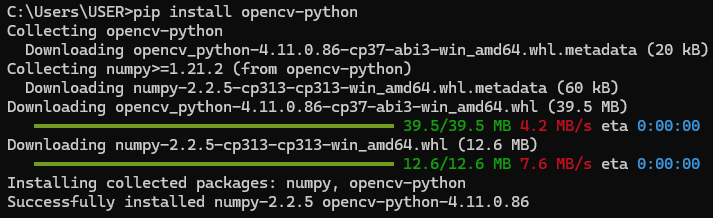
**2. Sprint 1**

## 2.1. Tarea 1.1.1: Configurar Cámara y flujo de video en tiempo real

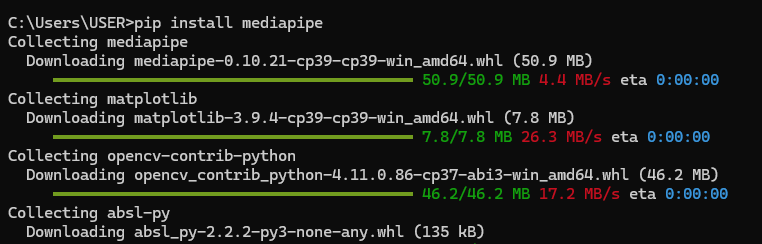
Módulos:

* captura\_video.py: Es el módulo encargado de capturar el video en tiempo real con la cámara que se usa en la configuración (en este caso se usa la cámara de la laptop con la que se desarrolla el modelo)

Instalación de librería cv2 desde la cmd (para importar biblioteca OpenCV para visión por computador)



Instalación de MediaPipe para detección facial con Python v 3.9.13



#Sprint "Configurar cámara y flujo de video en tiempo real"

from PIL import Image, ImageTk

import cv2

#para capturar video

captura = cv2.VideoCapture(0) # 0 = cámara web de laptop

def captura\_frames():

ret, frame = captura.read()

if ret: #si se lee bien el frame

print("capturando frames...")

return frame #devuelve el frame leído

else:

print("Error en captura")

return None

## 2.2. Tarea 1.1.2: Implementar detección facial con OpenCV MediaPipe

Módulos:

* deteccion\_facial.py: El módulo recibe frames del video en tiempo real desde captura\_video.py para luego aplicar detección facial con la librería MediaPipe y los resultados se muestran en el módulo alarma\_led.py que es el módulo que simula la alarma visual led.

#Sprint 1 "Implementar detección facial con MediaPipe"

import cv2

import mediapipe as mp

from captura\_video import captura\_frames # Importar la función del otro módulo

# Inicializar MediaPipe Face Detection

mp\_deteccionFacial = mp.solutions.face\_detection #modulo de detección facial

deteccionFacial = mp\_deteccionFacial.FaceDetection(model\_selection=0, min\_detection\_confidence=0.5) #modelo de selección rápida con un valor mínimo de confianza del 50%

#Función para procesar frames

def procesar\_frame():

frame = captura\_frames() #Se lee frame

if frame is None:

return None

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB) #se cambia frame a formato rgb para mediapipe

resultados = deteccionFacial.process(frame\_rgb)

#si detección facial es true se genera un cuadrado al rededor de la cara

if resultados.detections:

for detection in resultados.detections:

bboxC = detection.location\_data.relative\_bounding\_box

ih, iw, \_ = frame.shape

bbox = int(bboxC.xmin \* iw), int(bboxC.ymin \* ih), \

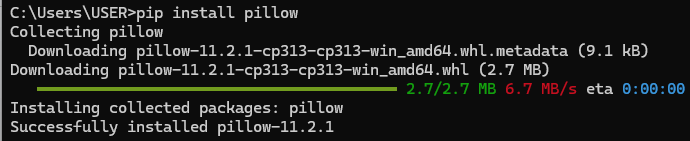
int(bboxC.width \* iw), int(bboxC.height \* ih)

cv2.rectangle(frame, bbox, (0, 255, 0), 2)

return frame

* alarma\_led.py: muestra el flujo del video con la detección facial y simula la alarma led.

Instalación de librería pillow desde la cmd (para el uso y procesamiento de imágenes)



import tkinter as tk

from PIL import Image, ImageTk

from deteccion\_facial import procesar\_frame

from captura\_video import captura

import cv2

ventana = tk.Tk() #crea la ventana principal de la interfaz

ventana.configure(bg="#FFF5EE") #configura color de fondo de la ventana

ventana.title("Alarma led") #titulo de ventana

ventana.geometry("800x600") #dimensiones de ventana wxh

#label para mostrar flujo de video

video\_label = tk.Label(ventana,bg="#1C1C1C")

video\_label.place( #configura posición y dimensiones

relx=0.065, # 0% desde la izquierda

rely=0.21, # 0% desde arriba

relwidth=0.43, # 43% del ancho total

relheight=0.58 # 58% del alto total

)

#label para mostrar alarma

fuente =("calibri, 24")

alarma\_label = tk.Label(ventana,text="hola",bg="#000000",fg="#FFFFFF", font=fuente)

alarma\_label.place(

relx=0.56, # 0% desde la izquierda

rely=0.21, # 0% desde arriba

relwidth=0.375, # 43% del ancho total

relheight=0.58 # 58% del alto total

)

#función para mostrar video (se muestra en frames)

def mostrar\_video():

frame = procesar\_frame()

if frame is not None: #si se manda la frame correctamente

frame = cv2.cvtColor(frame,cv2.COLOR\_BGR2RGB) #convierte el frame de formato bgr a rgb

img = Image.fromarray(frame) #convierte el array de pixeles NumPy a objeto Image de Pillow

imgtk = ImageTk.PhotoImage(image=img) #convierte la img en una imagen que tkinter muestra dentro de un label

video\_label.imgtk = imgtk

video\_label.configure(image=imgtk) #muestra imagen en pantallla (secuencias de imagen hacen video)

video\_label.after(10, mostrar\_video) #despues de 10 ms vuelve a llamar a la función

mostrar\_video() #inicia función

ventana.mainloop() #bucle principal para que se muestre programa

captura.release() #deja de capturar desde la cámara al cerrar app

## 2.3. Tarea 1.1.3: Optimizar latencia (<1 segundo por frame)

Con el fin de optimizar la latencia del programa se realizaron 2 cambios en el código:

* Reducción del tamaño del frame: Inicialmente, la captura de frames se realizaba con dimensiones automáticas. Posteriormente, en el módulo captura\_video.py, se ajustaron las dimensiones a 320x240 píxeles. Esta optimización reduce significativamente la latencia, ya que un frame de menor tamaño implica un procesamiento más rápido en el módulo deteccion\_facial.py (al disminuir el peso de los datos).

captura.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 320) #reducción de tamaño de frame para menor latencia

captura.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 240)

* Código aplicando multithreading: En lugar de ejecutar las tareas de forma completamente secuencial (1° capturar video, 2° procesar frame, 3° mostrar resultado), se implementó un modelo multithreading para optimizar el rendimiento del sistema. Se creó un hilo dedicado exclusivamente a la captura continua de frames desde la cámara, permitiendo que esta tarea ocurra en segundo plano de manera independiente. Mientras tanto, el hilo principal se encarga de procesar y mostrar el último frame disponible, sin necesidad de esperar a que se capture uno nuevo en cada ciclo. Este enfoque reduce significativamente la latencia, ya que evita cuellos de botella al desacoplar la captura de video del procesamiento y visualización. En lugar de procesar cada frame inmediatamente después de capturarlo, el sistema accede siempre al frame más reciente obtenido por la función obtener\_frame del módulo captura\_video.py, manteniendo así el flujo de video fluido y en tiempo real.

# Lanza un hilo que siempre está capturando frames

def iniciar\_hilo\_captura():

hilo = threading.Thread(target=capturar\_frames\_continuamente, daemon=True)

hilo.start()

### 2.3.1. Código completo

* captura\_video.py

#Sprint "Configurar cámara y flujo de video en tiempo real"

import cv2

import threading

#para capturar video

captura = cv2.VideoCapture(0) # 0 = cámara web de laptop

captura.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 320) #reducción de tamaño de frame para menor latencia

captura.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 240)

# Variable global para el último frame capturado

frame\_actual = None

lock = threading.Lock() # Para controlar acceso al frame entre hilos

def capturar\_frames\_continuamente():

global frame\_actual

while True:

ret, frame = captura.read()

if ret:

with lock:

frame\_actual = frame.copy() #copia el último frame disponible

# Lanza un hilo que siempre está capturando frames

def iniciar\_hilo\_captura():

hilo = threading.Thread(target=capturar\_frames\_continuamente, daemon=True)

hilo.start()

def obtener\_frame():

with lock:

if frame\_actual is not None:

return frame\_actual.copy()

else:

return None

* deteccion\_facial.py

#Sprint 1 "Implementar detección facial con MediaPipe"

import cv2

import mediapipe as mp

from captura\_video import obtener\_frame # Importar la función del otro módulo

# Inicializar MediaPipe Face Detection

mp\_deteccionFacial = mp.solutions.face\_detection #modulo de detección facial

deteccionFacial = mp\_deteccionFacial.FaceDetection(model\_selection=0, min\_detection\_confidence=0.5) #modelo de selección rápida con un valor mínimo de confianza del 50%

#Función para procesar frames

def procesar\_frame():

frame = obtener\_frame() #Se lee frame

if frame is None:

return None

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB) #se cambia frame a formato rgb para mediapipe

resultados = deteccionFacial.process(frame\_rgb)

#si detección facial es true se genera un cuadrado al rededor de la cara

if resultados.detections:

for detection in resultados.detections:

bboxC = detection.location\_data.relative\_bounding\_box

ih, iw, \_ = frame.shape

bbox = int(bboxC.xmin \* iw), int(bboxC.ymin \* ih), \

int(bboxC.width \* iw), int(bboxC.height \* ih)

cv2.rectangle(frame, bbox, (0, 255, 0), 2)

return frame

* alarma\_led.py

import tkinter as tk

from PIL import Image, ImageTk

from deteccion\_facial import procesar\_frame

from captura\_video import iniciar\_hilo\_captura, captura

import cv2

ventana = tk.Tk() #crea la ventana principal de la interfaz

ventana.configure(bg="#FFF5EE") #configura color de fondo de la ventana

ventana.title("Alarma led") #titulo de ventana

ventana.geometry("800x600") #dimensiones de ventana wxh

#label para mostrar flujo de video

video\_label = tk.Label(ventana,bg="#1C1C1C")

video\_label.place( #configura posición y dimensiones

relx=0.065, # 0% desde la izquierda

rely=0.21, # 0% desde arriba

relwidth=0.43, # 43% del ancho total

relheight=0.58 # 58% del alto total

)

#label para mostrar alarma

fuente =("calibri, 24")

alarma\_label = tk.Label(ventana,text="hola",bg="#000000",fg="#FFFFFF", font=fuente)

alarma\_label.place(

relx=0.56, # 0% desde la izquierda

rely=0.21, # 0% desde arriba

relwidth=0.375, # 43% del ancho total

relheight=0.58 # 58% del alto total

)

#función para mostrar video (se muestra en frames)

def mostrar\_video():

frame = procesar\_frame()

if frame is not None: #si se manda la frame correctamente

frame = cv2.cvtColor(frame,cv2.COLOR\_BGR2RGB) #convierte el frame de formato bgr a rgb

img = Image.fromarray(frame) #convierte el array de pixeles NumPy a objeto Image de Pillow

imgtk = ImageTk.PhotoImage(image=img) #convierte la img en una imagen que tkinter muestra dentro de un label

video\_label.imgtk = imgtk

video\_label.configure(image=imgtk) #muestra imagen en pantallla (secuencias de imagen hacen video)

video\_label.after(10, mostrar\_video) #despues de 10 ms vuelve a llamar a la función

iniciar\_hilo\_captura()

mostrar\_video() #inicia función

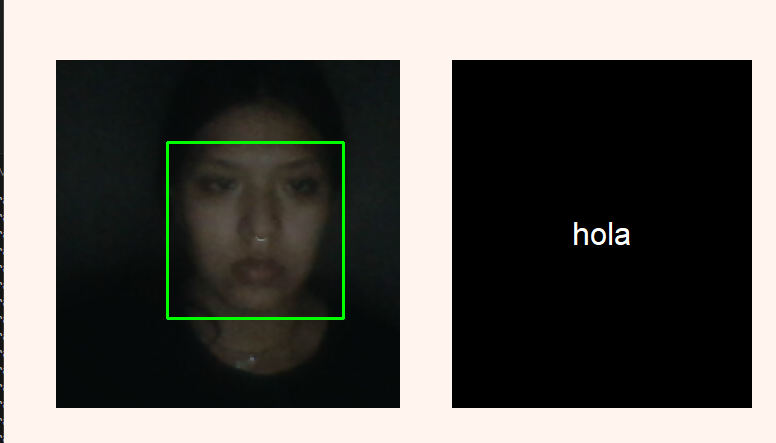
ventana.mainloop() #bucle principal para que se muestre programa

captura.release() #deja de capturar desde la cámara al cerrar app

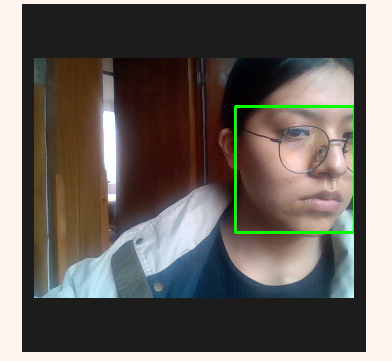
## 2.4. Tarea 1.1.4: Pruebas con diferentes ángulos y condiciones de luz

La configuración de detección facial con MediaPipe en el código permite seguir el rostro una vez identificado, lo que genera respuestas bastante consistentes. Sin embargo, cuando no hay un rostro claramente enfocado frente a la cámara, se observaron algunos falsos positivos con elementos del fondo del frame.

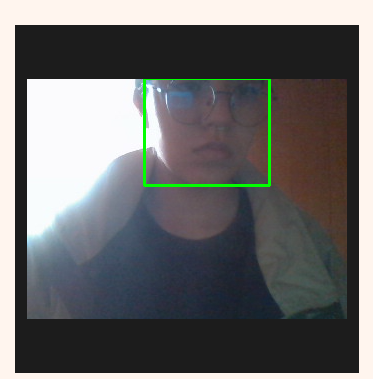
* Baja iluminación en el entorno pero con un poco de brillo de la pantalla.



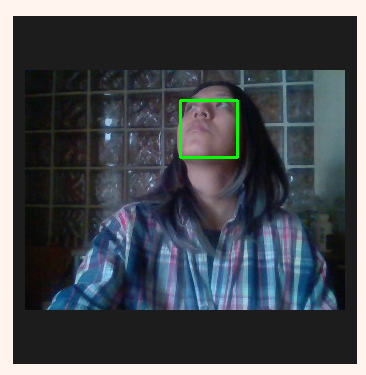
* Iluminación natural sin enfoque en el rostro.



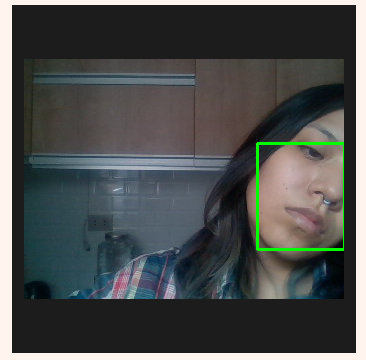
* Mala iluminación natural sin enfoque en el rostro



* Sin enfoque en el rostro



* Sin enfoque en el rostro, con buena iluminación



## 2.5. Tarea 1.1.5: Documentar configuración y requisitos de HW

### 2.5.1. Configuración del Entorno

* Sistema operativo: Windows 11 Pro
* Lenguaje de programación: Python 3.9.13 (compatible con librería MediaPipe)
* Bibliotecas utilizadas:OpenCV (cv2 v 4.11.0.86), MediaPipe (v 0.10.21), PIL / Pillow (11.2.1), Tkinter (incluido con Python), threading (módulo estándar de Python).
* IDE / Editor de código**:** Visual Studio Code
* Estructura de módulos actuales:
  + captura\_video.py: Captura continua de frames desde la cámara.
  + deteccion\_facial.py: Procesamiento de frames con MediaPipe.
  + alarma\_led.py: Interfaz gráfica y visualización del video procesado.
* **Lógica multithreading aplicada:** Se ha implementado un hilo independiente para la captura continua de frames, lo cual permite reducir la latencia del procesamiento al separar esta tarea del flujo principal de la interfaz y la detección.

### 2.5.2. Requisitos de HW

* Procesador: Intel i7
* Memoria RAM: 12 GB
* Cámara web: Cámara web HD nativa de laptop. Resolución: 1280x720.

## 2.6. Avance de tareas

| ID | Tarea | Responsable | Estimación (Horas) | Estado |
| --- | --- | --- | --- | --- |
| 1.1.1 | Configurar cámara y flujo de video en tiempo real | Backend Dev | 18 | Hecho |
| 1.1.2 | Implementar detección facial con OpenCV/MediaPipe | Computer Vision Engineer | 24 | Hecho |
| 1.1.3 | Optimizar latencia (<1 segundo por frame) | Backend Dev | 18 | Hecho |
| 1.1.4 | Pruebas con diferentes ángulos y condiciones de luz | QA Engineer | 10 | Hecho |
| 1.1.5 | Documentar configuración y requisitos de hw | Technical Writer | 6 | Hecho |
| TOTAL DE HORAS ESTIMADO | | | **76** | |
| TOTAL DE HORAS NETAS | | | **56** | |

## 