Laboratorio 7

Jonathan David Diaz,David Felipe Diaz, David Santiago Lopez

Noviembre 2025

Resumen

Este documento describe el laboratorio: diseño, implementación y explicación del **detector de emociones por cámara** que identifica tres estados (Feliz, Enojado, Triste) usando MediaPipe Face Mesh y técnicas de programación concurrente en Python (hilos, mutex y semáforos). Se incluye el código completo y una explicación detallada del mismo. También este informe describe, de forma concisa, las actividades realizadas para el desarrollo de la propuesta *AgroDigital IA* y los artefactos entregados. Cada apartado contiene únicamente la descripción de lo ejecutado.

1 Objetivo

Implementar un sistema que, usando la cámara del computador:

- Detecte la cara y entregue **landmarks** faciales mediante MediaPipe Face Mesh.
- Clasifique la emoción en tres clases: **Feliz**, **Enojado** o **Triste**.
- Utilice **hilos** para separar captura y procesamiento, **mutex** para proteger datos compartidos y **semaforos** para controlar el acceso a secciones críticas.

2 Descripción de las imágenes de referencia

De izquierda a derecha:

1. Cara amarilla con ojos café y sonrisa → **Feliz**.
2. Cara roja con cejas fruncidas y expresión de enojo → **Enojado**.
3. Cara azul con lágrima y boca triste → **Triste**.

3 Fundamento técnico

3.1 MediaPipe Face Mesh

MediaPipe Face Mesh detecta hasta 468 puntos faciales (landmarks) normalizados en el rango [0,1] en cada eje (x,y,z relativo al tamaño del frame). Cada landmark tiene propiedades como `.x`, `.y`, `.z`. En este proyecto usamos algunos índices representativos:

- **13,14** — puntos alrededor de la boca superior / inferior (útiles para medir apertura de boca).
- **145,159** — puntos del párpado / ojo izquierdo (útiles para medir apertura del ojo izquierdo).
- **374,386** — puntos del ojo derecho (análogos al izquierdo).
- **66,105** — puntos cerca de la ceja izquierda (para detectar fruncimiento).

3.2 Cómo se hace la clasificación (idea)

Usamos diferencias en coordenadas `y` (verticales normalizadas) para estimar:

- **Apertura de boca** = distancia vertical entre 13 y 14.
- **Apertura de ojo** = distancia vertical entre párpados (ej. 159-145).
- **Inclinación / elevación de ceja** = diferencia entre puntos de ceja (ej. 66 vs 105).

Con reglas simples basadas en umbrales:

- Boca bastante abierta → **Feliz** (o sorprendido según criterio).
- Ojos muy cerrados + ceja fruncida → **Enojado**.
- Si no cumple otras condiciones → **Triste** (o neutral según afinación).

3.3 Programación concurrente: por qué usar hilos, mutex y semáforo

- **Hilos (threads)**: permiten que la captura (I/O) y el procesamiento pesado (detección de face mesh + clasificación) no bloqueen la interfaz / visualización.
- **Mutex (locks)**: protegen variables compartidas (por ejemplo la emoción detectada) frente a condiciones de carrera.
- **Semáforo**: controla el acceso a una sección crítica que solo debe ejecutarse por un hilo a la vez (por ejemplo, evitar que varios threads inicien procesamiento simultáneamente y saturen la CPU).

4 Código: explicaciones y comentarios

A continuación se muestra el código real empleado y, después, una explicación por bloques y línea clave.

4.1 `emotion_detector.py`

import threading import cv2 import mediapipe as mp import numpy as np import time

———————— Semáforo para controlar el acceso al procesamiento ———————— process$_s emaphore = threading.Semaphore(1)$

———————— Mutex para proteger los datos compartidos ———— ———————— data$_m utex = threading.Lock()$

Variable compartida shared$_e motion =$ "$Nodetectada$"

Inicializar MediaPipe mp$_f ace = mp.solutions.face_m eshmp_d raw = mp.solutions.drawing_u tils$

def classify$_e motion(landmarks) : Muybásico : usaladistanciaentrepuntosparaaproximaremoción.Sepuede$

mouth$_u p = landmarks$[13]$.ymouth_d own = landmarks$[14]$.yeye_l eft_u p = landmarks$[159]$.yeye_l eft_d own = landmarks$[145]$.y$

mouth$_o pen = abs(mouth_d own - mouth_u p) eye_o pen = abs(eye_l eft_d own - eye_l eft_u p)$

if mouth$_o pen > 0.04 : return$"$Sorprendido$"$elif eye_o pen < 0.01 : return$"$Enojado$"$else : return$"$Neutral$"$

def emotion$_p rocessing(frame, results) : global shared_e motion$

Entrar al semáforo with process$_s emaphore$:

for face$_l andmarksinresults.multi_f ace_l andmarks : emotion = classify_e motion(face_l andmarks.landmark$

Proteger con mutex with data$_m utex : shared_e motion = emotion$

time.sleep(0.02) evitar sobrecarga

def emotion$_t hread(frame, results) : t = threading.Thread(target = emotion_p rocessing, args = (frame, results))t.start()$

Explicación por partes (`emotion_detector.py`)

- process$_s emaphore = threading.Semaphore(1) : semáforoconvalor1|actúacomounbloqueoquepermitesó$
- data$_m utex = threading.Lock() : mutexparaprotegeraccesoa$`shared_emotion`.
- `shared_emotion: variable global donde escribimos la emoción detectada para que otros hilos (p. ej. hilo de UI) la lean sin inconsistencia.`
- mp$_f ace = mp.solutions.face_m eshymp_d raw : inicializacionesdeMediaPipe(utilessidibujaslandmarks)$.
- `classify_emotion(landmarks): función que recibe la lista de landmarks de una cara y calcula:`
  - `mouth_open = diferencia vertical entre puntos 14 y 13.`

- `eye_open = apertura del ojo (puntos 145 y 159).`
- `Umbrales: 0.04 para boca abierta (puede ajustarse), 0.01 para ojo cerrado.`
- `Devuelve "Sorprendido", "Enojado" o "Neutral" según reglas simples.`

- `emotion_processing(frame, results): función que hace:`
  1. `with process_semaphore: entra en sección controlada por semáforo (asegura que sólo un thread procese landmarks a la vez).`
  2. `Itera for face_landmarks in results.multi_face_landmarks y clasifica.`
  3. `with data_mutex: protege la asignación a shared_emotion.`
  4. `time.sleep(0.02): pequeña pausa para evitar usar 100% CPU (ajustable).`
- `emotion_thread(frame, results): función auxiliar que lanza emotion_processing en un nuevo hilo para no bloquear el hilo que capturó el frame.`

4.2 `main.py`

import cv2 import mediapipe as mp from threading import Thread, Lock, Semaphore

Inicializar MediaPipe Face Mesh mp$_face_mesh = mp.solutions.face_mesh$

class EmotionDetector: def $_init_(self) : self.face_mesh = mp_face_mesh.FaceMesh(max_num_faces = 1, refine_landmarks = True, min_detection_confidence = 0.5, min_tracking_confidence = 0.5)self.lock = Lock()Mutexself.semaphore = Semaphore(1)Semáforoself.emotion = "Desconocida"$

def detectar$_emocion(self, landmarks) : """DetectaFelicidad, EnojooTristezausandolabios, ojosycejas"""$

— Ojos — ojo$_izq = landmarks[159].y−landmarks[145].yojo_der = landmarks[386].y−landmarks[374].y$

— Boca — boca$_sup = landmarks[13]boca_inf = landmarks[14]boca_dist = boca_inf.y − boca_sup.y$

— Cejas (inclinación de la ceja izquierda) — ceja$_izq_sup = landmarks[105]ceja_izq_inf = landmarks[66]ceja_inclinacion = ceja_izq_inf.y − ceja_izq_sup.y$

— Lógica simple — FELIZ: boca abierta o leve sonrisa (ceja normal) if boca$_dist > 0.025andceja_inclinacion > 0 : return"Feliz"$

ENOJADO: ojos cerrados y ceja fruncida if ojo$_izq < 0.004andojo_der < 0.004andceja_inclinacion < −0.003 : return"Enojado"$

TRISTE: boca cerrada y leve caída de cejas return " Triste"

def detect(self, frame): with self.lock: rgb = cv2.cvtColor(frame, cv2.COLOR$_BGR2RGB)results = self.face_mesh.process(rgb)ifresults.multi_face_landmarks : face = results.multi_face_landmarks[0]self.emot self.detectar_emocion(face.landmark)else : self.emotion = "Nodetectada"returnself.emotion$

def camera$_thread(detector) : detector.semaphore.acquire()cap = cv2.VideoCapture(0)$

if not cap.isOpened(): print("No se pudo abrir la cámara") detector.semaphore.release() return

while True: ret, frame = cap.read() if not ret: break
emotion = detector.detect(frame)
cv2.putText(frame, f"Emoción: emotion", (50,50), cv2.FONT$_H ERSHEY_S IMPLEX, 1, (0, 255, 0), 2)
cv2.imshow("Emotion Detector", frame)
if cv2.waitKey(1) 0xFF == 27: ESC para salir break
cap.release() cv2.destroyAllWindows() detector.semaphore.release()
if $_n ame_= = "_m ain"$: $detector = EmotionDetector()t1 = Thread(target = camera_t hread, args = (detector,))t1.start()t1.join()$

Explicación por partes (`main.py`)

- `mp_face_mesh = mp.solutions.face_mesh`: referencia al módulo Face Mesh (se crea instancia en la clase).
- **Clase** `EmotionDetector`:
  - Constructor: crea `FaceMesh` con parámetros:
    - `max_num_faces=1`: solo una cara (simplifica).
    - `refine_landmarks=True`: activa detalles finos en ojos y labios.
    - `min_detection_confidence` y `min_tracking_confidence`: umbrales para detección y seguimiento.
  - `self.lock`: mutex para proteger el método `detect` que llama a MediaPipe (evita condiciones de carrera si varios hilos llaman).
  - `detectar_emocion(landmarks)`: función que implementa la lógica de clasificación usando landmarks:
    - Calcula apertura de ojos (`ojo_izq`, `ojo_der`).
    - Calcula distancia entre labios (`boca_dist`).
    - Calcula inclinación de la ceja izquierda (`ceja_inclinacion`).
    - Aplica reglas con umbrales (0.025, 0.004, -0.003 — valores iniciales que pueden ajustarse según dataset/iluminación).
  - `detect(frame)`: convierte a RGB, procesa con `face_mesh.process`, y si hay landmarks, actualiza `self.emotion` usando `detectar_emocion`. Todo protegido por `self.lock`.
- `camera_thread(detector)`:
  - `detector.semaphore.acquire()`: adquiere semáforo antes de abrir la cámara (evita que múltiples instancias intenten abrirla simultáneamente).
  - Bucle de captura: lee frame, llama `detector.detect(frame)`, dibuja texto y muestra ventana.
  - Salida con `ESC` (código 27), libera recursos y semáforo.

5 Cómo ejecutar

1. Crear entorno virtual (recomendado):

```
python -m venv venv
source venv/bin/activate   # Linux / macOS
venv\Scripts\activate      # Windows
```

2. Instalar dependencias:

```
pip install -r requirements.txt
```

3. Ejecutar:

```
python main.py
```

4. Presiona `ESC` para salir.

6 Archivo `requirements.txt`

mediapipe==0.10.20 opencv-python==4.7.0.72 numpy==1.23.5

7 Ajustes, mejoras y recomendaciones

- **Ajustar umbrales:** los valores (0.04, 0.025, 0.004, -0.003) dependen de la distancia de la cámara y la resolución. Calíbralos con ejemplos reales.
- **Normalizar por distancia cara-cámara:** usar distancia entre dos landmarks fijos (por ejemplo, ancho entre mejillas) para escalar umbrales.
- **Filtrado temporal:** aplicar un filtro tipo *rolling window* o suavizado para evitar fluctuaciones entre frames.
- **Dataset y ML:** para mayor robustez, recolecta ejemplos etiquetados y entrena un clasificador (SVM, RandomForest o red pequeña) sobre features extraídas de landmarks.
- **Manejo de múltiples caras:** ampliar `max_num_faces` y elegir la cara más cercana o la primaria.
- **Manejo de errores:** capturar excepciones de MediaPipe y OpenCV; loguear si `results` es `None`.

8 Limitaciones conocidas

- Clasificación basada en reglas simples — no robusta a variaciones de rostro, etnia, edad o iluminación.
- Umbrales fijos: sensibles a la distancia entre cara y cámara.
- MediaPipe requiere CPU razonable; en máquinas lentas puede ser lento; considera usar GPU si está disponible.

9 Punto 3) Propuesta del Proyecto

La primera actividad consistió en desarrollar la propuesta conceptual y técnica del sistema llamado **AgroDigital IA**. El enfoque del proyecto se basa en resolver problemáticas del sector agrícola colombiano mediante tecnologías modernas alineadas con la convocatoria "Colombia Inteligente – Infraestructura" del Ministerio de Ciencia.

9.1 Descripción General del Proyecto

El proyecto plantea una plataforma integral capaz de:

- Monitorear variables agrícolas a través de redes de sensores IoT distribuidos.
- Procesar información de manera local en un nodo Edge capaz de ejecutar inteligencia artificial en contenedores.
- Crear y actualizar un **gemelo digital** de los cultivos, permitiendo simulación, visualización y predicción.
- Activar actuadores autónomos como riego inteligente, drones o robots agrícolas simulados.
- Permitir a los usuarios visualizar y gestionar el sistema desde un panel web.

9.2 Problema y Justificación

En Colombia, los agricultores enfrentan retos derivados de:

- Ineficiencia en el uso de agua.
- Falta de información en tiempo real.
- Costos asociados a plagas y estrés hídrico.
- Variabilidad climática.

La plataforma AgroDigital IA se justifica como una herramienta para optimizar recursos, mejorar la productividad y facilitar la toma de decisiones basada en datos.

9.3 Metodología del Diseño

Para estructurar la propuesta, se siguieron las siguientes etapas:

1. Identificación de problemas agrícolas comunes.
2. Selección de sensores adecuados y análisis de protocolos de comunicación.

3. Definición de un nodo Edge compatible con contenedores Docker.
4. Diseño preliminar del gemelo digital y de los modelos de inteligencia artificial.
5. Elaboración del flujo completo del sistema.

10 Punto b) Desarrollo del README y Arquitectura Técnica

En este punto se elaboró un documento técnico extenso en formato Markdown, destinado a servir como README principal del repositorio del proyecto.

10.1 Contenido del README Desarrollado

El README contiene:

- Introducción conceptual del proyecto.
- Definición del problema agrícola y motivación de la solución.
- Explicación del flujo ETL creado.
- Arquitectura general del ecosistema IoT–Edge–IA–Gemelo Digital.
- Diagramas de flujo representativos.
- Descripción técnica de los contenedores empleados.
- Descripción del dashboard desarrollado en Streamlit.
- Impactos técnicos, sociales y ambientales.

10.2 Diagramas Elaborados

Se diseñó un diagrama de flujo general que muestra:

1. Captura de datos en sensores IoT.
2. Envío de datos al nodo Edge.
3. Procesamiento local mediante IA en contenedores.
4. Actualización del gemelo digital.
5. Generación de recomendaciones.
6. Activación de actuadores.
7. Visualización en el dashboard.

10.3 Construcción del ETL

Durante la elaboración técnica se desarrolló un ETL completo compuesto por:

- **Extract**: Lectura de un archivo Excel con 17 hojas, filtrado de filas vacías y valores repetidos.
- **Transform**: Limpieza de datos, eliminación de duplicados, normalización de columnas.
- **Load**: Inserción en una base de datos SQLite.

El módulo `extract.py` fue modificado para ignorar datos inconsistentes y garantizar la integridad de la información procesada.

10.4 Dashboard en Streamlit

El dashboard desarrollado permite:

- Visualizar los datos cargados desde SQLite.
- Explorar estadísticas por hoja o por variable.
- Preparar terreno para integrar la información del gemelo digital.

11 Punto c) Recomendaciones de Tecnologías Futuras

Como parte del trabajo, se realizaron recomendaciones basadas en tendencias actuales y tecnologías abordadas en Digitales III.

11.1 Tecnologías Futuras para Agricultura Inteligente

Las tecnologías sugeridas incluyen:

11.1.1 Robótica y Automatización

- Robots agrícolas autónomos con ROS2.
- Vehículos aéreos no tripulados con cámaras multiespectrales.
- Manipuladores robóticos para cosecha asistida.

11.1.2 Inteligencia Artificial

- Aprendizaje federado para fincas descentralizadas.
- Redes neuronales ligeras optimizadas para Edge.
- IA generativa para simulación de cultivos.

11.1.3 Sensores y Comunicaciones

- Sensores biodegradables con nanotecnología.
- Redes LoRaWAN de largo alcance.
- Mallas inalámbricas resilientes para ambientes rurales.

11.1.4 Infraestructura Digital

- Contenedores avanzados con Kubernetes en mini clusters.
- Integración de Digital Twins mediante estándares DTDL.

12 Actividades Complementarias Realizadas

Además de los puntos principales, se desarrollaron múltiples componentes técnicos adicionales:

12.1 Scripts Programados

- Código para lectura de 17 hojas de Excel con validación automática.
- Módulo de filtrado para eliminar filas vacías y duplicadas.
- Pipeline ETL completo ejecutable desde consola.
- Ejemplo funcional de aplicación en Streamlit.

12.2 Validación de Datos

Se implementaron funciones para:

- Detectar errores en hojas de Excel.
- Identificar columnas faltantes.
- Asegurar coherencia entre hojas.

12.3 Estructura del Proyecto

Se creó una estructura modular con tres componentes:

- Módulo de extracción.
- Módulo de transformación.
- Módulo de carga.

13 Conclusiones Generales

El desarrollo del proyecto permitió integrar diversas tecnologías modernas aplicables a la agricultura colombiana. El trabajo realizado cumple con los requerimientos académicos y técnicos, y sienta las bases para una implementación real en campo o un prototipo avanzado para presentación ante MinCiencias.

El sistema AgroDigital IA representa un paso importante hacia la digitalización rural, aprovechando conceptos clave de Digitales III como contenedores, simulación robótica y analítica de datos.