Manual de Aplicación del Código Fuente

Sistema de Detección de Fatiga del Conductor

Versión 1.0

Fecha: 2025

Índice

---

1. Introducción

Este manual proporciona información detallada sobre la estructura, implementación y aplicación del código fuente del Sistema de Detección de Fatiga del Conductor.

Está dirigido a desarrolladores, ingenieros de software y técnicos que deseen entender, modificar o extender el sistema.

1.1 Tecnologías Utilizadas

- **Python 3.10+**: Lenguaje de programación principal
- **FastAPI**: Framework web para el servidor
- **Flet**: Framework para la interfaz gráfica
- **MediaPipe**: Biblioteca para detección de puntos faciales y de manos
- **OpenCV**: Procesamiento de imágenes y video
- **WebSockets**: Comunicación en tiempo real
- **NumPy**: Operaciones matemáticas y arrays
- **Docker**: Contenedorización del servidor

1.2 Requisitos del Desarrollador

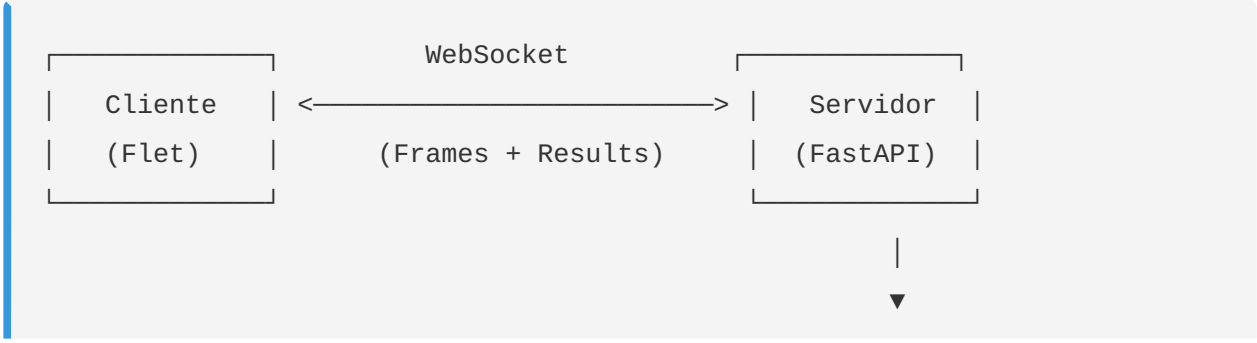
- Conocimiento de Python (intermedio-avanzado)
- Familiaridad con frameworks web (FastAPI)
- Conocimiento básico de visión por computadora
- Entendimiento de arquitectura cliente-servidor
- Conocimiento de Git para control de versiones

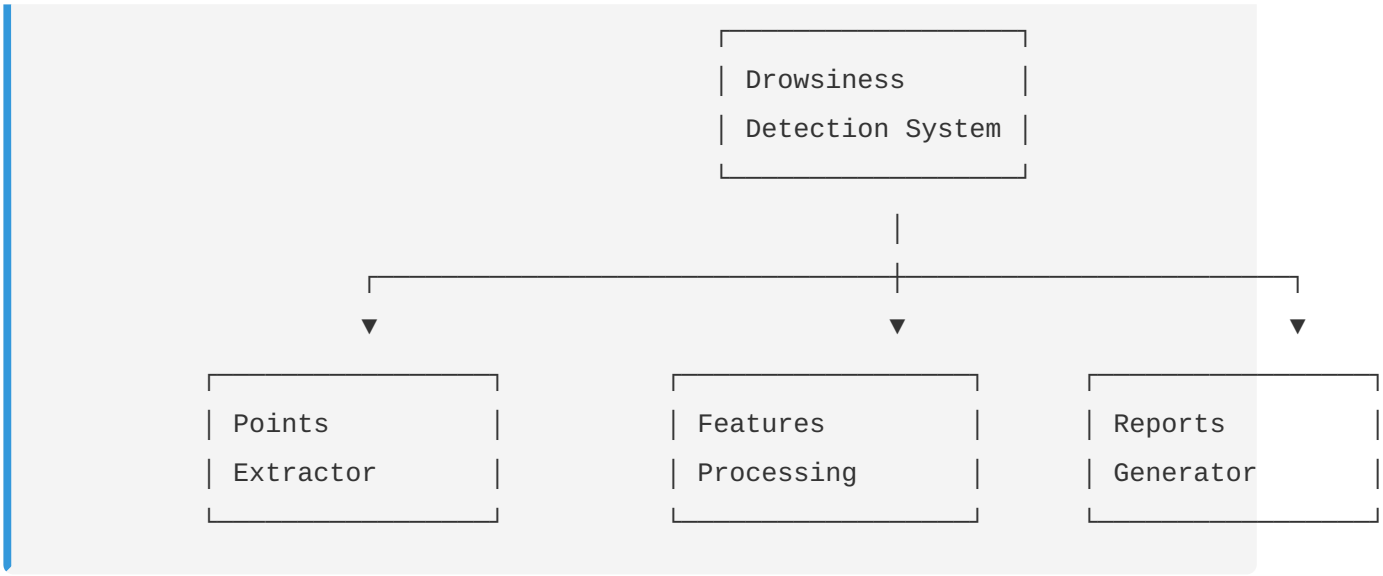
---

2. Arquitectura del Sistema

2.1 Arquitectura General

El sistema sigue una arquitectura cliente-servidor con comunicación WebSocket:





2.2 Flujo de Procesamiento

1. **Captura de Video**: El cliente captura frames de la cámara
2. **Codificación**: Los frames se codifican en JPEG y Base64
3. **Transmisión**: Los frames se envían al servidor vía WebSocket
4. **Extracción de Puntos**: El servidor extrae puntos faciales y de manos
5. **Procesamiento**: Se calculan distancias y se detectan características
6. **Detección**: Se aplican algoritmos temporales para detectar eventos
7. **Visualización**: Se generan anotaciones visuales
8. **Reportes**: Se generan reportes CSV y JSON
9. **Respuesta**: Los resultados se envían de vuelta al cliente
10. **Visualización Cliente**: El cliente muestra los resultados

---

3. Estructura del Proyecto

3.1 Estructura de Directorios

```
driver_fatigue_detection/
├── app.py                          # Servidor FastAPI
├── client.py                       # Cliente de ejemplo
├── main.py                         # Punto de entrada de la GUI
```

```
├── requirements_client.txt         # Dependencias del cliente
├── requirements_server.txt         # Dependencias del servidor
├── Dockerfile                      # Configuración Docker
├── drowsiness_processor/          # Módulo principal de procesamiento
│   ├── __init__.py
│   ├── main.py                    # Sistema principal de detección
│   ├── extract_points/            # Extracción de puntos clave
│   │   ├── point_extractor.py
│   │   ├── face_mesh/
│   │   │   └── face_mesh_processor.py
│   │   └── hands/
│   │       └── hands_processor.py
│   ├── data_processing/           # Procesamiento de puntos
│   │   ├── main.py
│   │   ├── eyes/
│   │   │   ├── eyes_processor.py
│   │   │   └── eyes_processing.py
│   │   ├── mouth/
│   │   │   ├── mouth_processor.py
│   │   │   └── mouth_processing.py
│   │   ├── head/
│   │   │   ├── head_processor.py
│   │   │   └── head_processing.py
│   │   ├── hands/
│   │   │   ├── first_hand/
│   │   │   └── second_hand/
│   │   └── processors/
│   │       ├── face_processor.py
│   │       └── hands_processor.py
│   ├── drowsiness_features/       # Detección de características
│   │   ├── processing.py
│   │   ├── processor.py
│   │   ├── flicker_and_microsleep/
│   │   ├── yawn/
│   │   ├── eye_rub/
│   │   └── pitch/
│   ├── visualization/             # Visualización
│   │   └── main.py
│   └── reports/                   # Generación de reportes
```

```
│       ├── main.py
│       └── august/
│           └── drowsiness_report.csv
├── gui/                          # Interfaz gráfica
│   ├── pages/
│   │   ├── start_page.py
│   │   ├── selection_interface_page.py
│   │   └── drowsiness_page.py
│   └── resources/
│       ├── fonts/
│       └── images/
├── examples/                     # Ejemplos de uso
│   ├── camera.py
│   └── video_stream.py
├── tests/                        # Pruebas
│   └── videos/
├── docs/                         # Documentación
│   └── descripcion_invento.md
└── storage/                      # Almacenamiento temporal
```

3.2 Archivos Principales

app.py

Servidor FastAPI que maneja las conexiones WebSocket y procesa las solicitudes.

Componentes principales: - Endpoint WebSocket `/ws` - Instancia de `DrowsinessDetectionSystem` - Codificación/decodificación de imágenes Base64

main.py (GUI)

Punto de entrada de la interfaz gráfica desarrollada con Flet.

Componentes principales: - Clase `MainApp`: Gestión de rutas y páginas - Páginas: Start, SelectionInterface, Drowsiness

drowsiness_processor/main.py

Sistema central de detección de fatiga.

Componentes principales: - `DrowsinessDetectionSystem`: Orquesta todos los módulos - Método `run()`: Procesa frames Base64 - Método `frame_processing()`: Procesa frames de video

---

4. Instalación y Configuración

4.1 Configuración del Entorno de Desarrollo

Paso 1: Obtener el Código Fuente

El código fuente se proporciona en un archivo ZIP o mediante un enlace de descarga.

Opción A: Archivo ZIP 1. Descomprima el archivo ZIP recibido 2. Navegue hasta la carpeta descomprimida

Opción B: Descarga desde Mediafire 1. Acceda al enlace de descarga proporcionado (Mediafire) 2. Descargue el archivo ZIP 3. Descomprima el archivo ZIP 4. Navegue hasta la carpeta descomprimida

Paso 2: Crear Entorno Virtual

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

Paso 3: Instalar Dependencias

```
# Instalar dependencias del servidor
pip install -r requirements_server.txt

# Instalar dependencias del cliente
pip install -r requirements_client.txt
```

4.2 Configuración del Servidor

Archivo: app.py

```
# Configuración del servidor
app = FastAPI()

# Endpoint WebSocket
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    # Configuración del sistema de detección
    drowsiness_detection_system = DrowsinessDetectionSystem()
    # ...
```

Variables de Entorno

Cree un archivo `.env` (opcional):

```
SERVER_HOST=0.0.0.0
SERVER_PORT=8000
REPORT_PATH=drowsiness_processor/reports/august/drowsiness_report.csv
```

4.3 Configuración del Cliente

Archivo: main.py

```
# Configuración de la GUI
self.page.window.width = 1920
self.page.window.height = 1080
self.page.bgcolor = "#fffffe"
```

Configuración de la Cámara

En `drowsiness_page.py`, puede modificar el índice de la cámara:

```
cap = cv2.VideoCapture(0)  # 0 = primera cámara, 1 = segunda cámara
```

---

5. Descripción de Módulos

5.1 Módulo: extract_points

point_extractor.py

Responsabilidad: Coordinar la extracción de puntos faciales y de manos.

Clase principal: `PointsExtractor`

Métodos: - `process(face_image)`: Procesa una imagen y extrae puntos clave - `merge_points(face_points, hands_points)`: Fusiona puntos faciales y de manos

Uso:

```
extractor = PointsExtractor()
key_points, success, sketch = extractor.process(image)
```

face_mesh_processor.py

Responsabilidad: Extraer puntos faciales mediante MediaPipe Face Mesh.

Clase principal: `FaceMeshProcessor`

Características: - Extrae 468 puntos faciales - Detecta contornos oculares, labiales, nasales - Genera mallas faciales para visualización

hands_processor.py

Responsabilidad: Extraer puntos de las manos mediante MediaPipe Hands.

Clase principal: `HandsProcessor`

Características: - Detecta hasta 2 manos simultáneamente - Extrae 21 puntos por mano - Calcula distancias entre dedos y puntos oculares

5.2 Módulo: data_processing

main.py

Responsabilidad: Coordinar el procesamiento de puntos extraídos.

Clase principal: `PointsProcessing`

Procesadores: - `EyesProcessor`: Procesa puntos oculares - `MouthProcessor`: Procesa puntos labiales - `HeadProcessor`: Procesa puntos de cabeza - `FirstHandProcessor` / `SecondHandProcessor`: Procesan puntos de manos

eyes_processing.py

Responsabilidad: Calcular distancias entre párpados.

Clase principal: `EyesPointsProcessing`

Métricas calculadas: - `right_upper_eyelid_distance`: Distancia del párpado superior derecho - `left_upper_eyelid_distance`: Distancia del párpado superior izquierdo - `right_lower_eyelid_distance`: Distancia del párpado inferior derecho - `left_lower_eyelid_distance`: Distancia del párpado inferior izquierdo

Uso:

```
processor = EyesPointsProcessing(EuclideanDistanceCalculator())
distances = processor.main(eyes_points)
```

5.3 Módulo: drowsiness_features

processing.py

Responsabilidad: Coordinar la detección de características de fatiga.

Clase principal: `FeaturesDrowsinessProcessing`

Estimadores: - `FlickerEstimator`: Detecta parpadeos y microsueños - `YawnEstimator`: Detecta bostezos - `EyeRubEstimator`: Detecta fricción ocular - `PitchEstimator`: Detecta inclinación de cabeza

flicker_and_microsleep/processing.py

Responsabilidad: Detectar parpadeos y microsueños.

Clases principales: - `FlickerDetection`: Detecta parpadeos rápidos - `MicroSleepDetection`: Detecta cierres oculares prolongados (≥2s)

Algoritmo: 1. Compara distancias de párpados superiores e inferiores 2. Detecta transiciones de estado (abierto/cerrado) 3. Mide duración de cierres oculares 4. Clasifica como parpadeo (<2s) o microsueño (≥2s)

yawn/processing.py

Responsabilidad: Detectar bostezos.

Clases principales: - `YawnDetection`: Detecta aperturas bucales prolongadas - `YawnCounter`: Cuenta bostezos y almacena duraciones

Algoritmo: 1. Calcula distancia labial y distancia del mentón 2. Detecta cuando distancia labial > distancia del mentón 3. Mide duración de la apertura 4. Clasifica como bostezo si duración >4s

Umbrales: - Duración mínima: 4 segundos - Ventana de reporte: 180 segundos (3 minutos)

eye_rub/processing.py

Responsabilidad: Detectar fricción ocular.

Clases principales: - `EyeRubDetection`: Detecta proximidad de dedos a ojos - `EyeRubCounter`: Cuenta fricciones oculares

Algoritmo: 1. Calcula distancias entre dedos y puntos oculares 2. Detecta cuando distancia < 40 píxeles 3. Mide duración de la proximidad 4. Clasifica como fricción si duración >1s

Umbrales: - Distancia máxima: 40 píxeles - Duración mínima: 1 segundo - Ventana de reporte: 300 segundos (5 minutos)

pitch/processing.py

Responsabilidad: Detectar inclinación de cabeza.

Clases principales: - `PitchDetection`: Detecta inclinaciones sostenidas - `PitchCounter`: Cuenta inclinaciones y almacena duraciones

Algoritmo: 1. Compara posición del punto nasal con puntos de mejillas 2. Compara distancia nariz-boca con distancia nariz-frente 3. Detecta inclinación hacia abajo 4. Mide duración de la inclinación 5. Clasifica como pitch si duración ≥3s

Umbrales: - Duración mínima: 3 segundos - Reporte: Inmediato cuando se detecta

5.4 Módulo: visualization

main.py

Responsabilidad: Generar visualizaciones en tiempo real.

Clase principal: `ReportVisualizer`

Funciones: - `visualize_all_reports(sketch, features)`: Visualiza todas las características - Dibuja mallas faciales y de manos - Añade anotaciones de texto - Muestra contadores y alertas

5.5 Módulo: reports

main.py

Responsabilidad: Generar y almacenar reportes.

Clase principal: `DrowsinessReports`

Funciones: - `main(report_data)`: Almacena reportes en CSV - `generate_json_report(report_data)`: Genera reporte JSON - `create_csv_file()`: Crea archivo CSV con encabezados

Formato CSV:

```
timestamp,eye_rub_first_hand_report,eye_rub_first_hand_count,...
2024-01-15 14:30:25,False,0,...
```

Formato JSON:

```
{
  "timestamp": "2024-01-15 14:30:25",
  "eye_rub_first_hand": {...},
  "flicker": {...},
```

```
  ...
}
```

---

6. API y Interfaces

6.1 API WebSocket

Endpoint: `/ws`

Método: WebSocket

Protocolo: Texto (Base64)

Mensaje de Entrada: - Formato: String Base64 (imagen JPEG codificada) - Ejemplo: `"iVBORw0KGgoAAAANSUhEUgAA..."`

Mensaje de Salida: - Formato: JSON

```
{
  "json_report": "{...}",
  "sketch_image": "base64_string",
  "original_image": "base64_string"
}
```

Ejemplo de Uso

```
import websockets
import json
import base64
import cv2

async def send_frame():
    uri = "ws://localhost:8000/ws"
    async with websockets.connect(uri) as websocket:
        # Capturar frame
        cap = cv2.VideoCapture(0)
        ret, frame = cap.read()
```

```
        # Codificar frame
        _, buffer = cv2.imencode('.jpg', frame)
        frame_base64 = base64.b64encode(buffer).decode('utf-8')

        # Enviar frame
        await websocket.send(frame_base64)

        # Recibir respuesta
        response = await websocket.recv()
        data = json.loads(response)

        # Procesar respuesta
        report = json.loads(data['json_report'])
        print(report)
```

6.2 Interfaces de Clase

DrowsinessDetectionSystem

```
class DrowsinessDetectionSystem:
    def __init__(self):
        """Inicializa el sistema de detección."""

    def run(self, picture_base64: str) -> Tuple[np.ndarray, np.ndarray, dict]:
        """
        Procesa una imagen codificada en Base64.

        Args:
            picture_base64: Imagen codificada en Base64

        Returns:
            Tuple de (imagen_original, sketch_anotado, reporte_json)
        """

    def frame_processing(self, face_image: np.ndarray) -> Tuple[np.ndarray, np.ndarray,
        """
        Procesa un frame de video.
```

```
        Args:
            face_image: Frame de video como array NumPy

        Returns:
            Tuple de (imagen_original, sketch_anotado, reporte_json)
        """
```

PointsExtractor

```
class PointsExtractor:
    def process(self, face_image: np.ndarray) -> Tuple[dict, bool, np.ndarray]:
        """
        Extrae puntos clave de una imagen.

        Args:
            face_image: Imagen de entrada

        Returns:
            Tuple de (puntos_extraídos, éxito, sketch)
        """
```

---

7. Configuración y Personalización

7.1 Modificar Umbrales de Detección

Parpadeos y Microsueños

Archivo: `drowsiness_processor/drowsiness_features/flicker_and_microsleep/processing.py`

```
# Modificar duración mínima de microsueño (línea 58)
if flicker_duration >= 2:  # Cambiar a valor deseado (en segundos)
    return True, flicker_duration
```

Bostezos

Archivo: `drowsiness_processor/drowsiness_features/yawn/processing.py`

```
# Modificar duración mínima de bostezo (línea 38)
if yawn_duration > 4:  # Cambiar a valor deseado (en segundos)
    return True, yawn_duration

# Modificar ventana de reporte (línea 98)
if elapsed_time >= 180:  # Cambiar a valor deseado (en segundos)
```

Fricción Ocular

Archivo: `drowsiness_processor/drowsiness_features/eye_rub/processing.py`

```
# Modificar distancia máxima (línea 23)
self.eye_rub = any(distance < 40 for distance in distances)  # Cambiar 40 a valor desea

# Modificar duración mínima (línea 34)
if eye_rub_duration > 1:  # Cambiar a valor deseado (en segundos)
```

Inclinación de Cabeza

Archivo: `drowsiness_processor/drowsiness_features/pitch/processing.py`

```
# Modificar duración mínima (línea 47)
if pitch_duration >= 3.0:  # Cambiar a valor deseado (en segundos)
```

7.2 Modificar Ruta de Reportes

Archivo: `drowsiness_processor/main.py`

```
# Modificar ruta del archivo CSV (línea 18)
self.reports = DrowsinessReports('ruta/personalizada/report.csv')
```

7.3 Modificar Configuración del Servidor

Archivo: `app.py`

```
# Modificar host y puerto
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    # ...
```

Para cambiar el puerto, modifique el comando de inicio:

```
uvicorn app:app --host 0.0.0.0 --port 8080  # Cambiar puerto
```

7.4 Modificar Resolución de Video

Archivo: `gui/pages/drowsiness_page.py`

```
# Modificar tamaño de imágenes (líneas 30-42)
self.original_image_control = Image(
    width=1280,  # Cambiar ancho
    height=720,  # Cambiar alto
    # ...
)
```

---

8. Extensión del Sistema

8.1 Agregar Nuevo Biomarcador

Paso 1: Crear Procesador de Puntos

Crear archivo: `drowsiness_processor/data_processing/nuevo_biomarcador/nuevo_processing.py`

```
class NuevoBiomarcadorProcessing:
    def __init__(self, distance_calculator: DistanceCalculator):
        self.distance_calculator = distance_calculator

    def main(self, points: dict):
        # Calcular métricas necesarias
        return metrics
```

Paso 2: Crear Estimador de Características

Crear archivo: `drowsiness_processor/drowsiness_features/nuevo_biomarcador/processing.py`

```
class NuevoBiomarcadorEstimator(DrowsinessProcessor):
    def __init__(self):
        self.detector = NuevoBiomarcadorDetection()

    def process(self, metrics: dict):
        # Implementar lógica de detección
        return report
```

Paso 3: Integrar en el Sistema Principal

Modificar: `drowsiness_processor/data_processing/main.py`

```
self.processed_points['nuevo_biomarcador'] = self.nuevo_processor.process(points)
```

Modificar: `drowsiness_processor/drowsiness_features/processing.py`

```
self.features_drowsiness['nuevo_biomarcador'] = NuevoBiomarcadorEstimator()
self.processed_feature['nuevo_biomarcador'] = self.features_drowsiness['nuevo_biomarcad
```

Paso 4: Agregar a Reportes

Modificar: `drowsiness_processor/reports/main.py`

```
self.fields = [..., 'nuevo_biomarcador_report', 'nuevo_biomarcador_count', ...]
```

8.2 Agregar Nueva Fuente de Video

Modificar: `gui/pages/drowsiness_page.py`

```
def run_detection(self):
    uri = "ws://localhost:8000/ws"
    # Cambiar fuente de video
    cap = cv2.VideoCapture("ruta/al/video.mp4")  # Para archivo de video
    # o
    cap = cv2.VideoCapture("rtsp://direccion_ip/stream")  # Para stream RTSP
    # ...
```

8.3 Agregar Notificaciones

Crear módulo: `drowsiness_processor/notifications/`

```
class NotificationManager:
    def send_alert(self, event_type: str, severity: str):
        # Implementar lógica de notificación
        # Email, SMS, sonido, etc.
        pass
```

Integrar en: `drowsiness_processor/main.py`

```
if micro_sleep_detected:
    notification_manager.send_alert("micro_sleep", "critical")
```

---

9. Testing y Debugging

9.1 Pruebas Unitarias

Crear archivo: `tests/test_eyes_processing.py`

```
import unittest
from drowsiness_processor.data_processing.eyes.eyes_processing import EyesPointsProcess

class TestEyesProcessing(unittest.TestCase):
    def test_calculate_distances(self):
        # Implementar prueba
        pass
```

Ejecutar pruebas:

```
python -m pytest tests/
```

9.2 Pruebas de Integración

Crear archivo: `tests/test_integration.py`

```
import unittest
from drowsiness_processor.main import DrowsinessDetectionSystem
import cv2

class TestIntegration(unittest.TestCase):
    def test_full_pipeline(self):
        system = DrowsinessDetectionSystem()
        image = cv2.imread("tests/test_image.jpg")
        result = system.frame_processing(image)
        self.assertIsNotNone(result)
```

9.3 Debugging

Habilitar Logs

Modificar: `drowsiness_processor/extract_points/point_extractor.py`

```
import logging
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)
```

```
# Agregar logs
logger.debug(f"Points extracted: {key_points}")
```

Visualización de Puntos

Modificar: `drowsiness_processor/extract_points/face_mesh/face_mesh_processor.py`

```
# Guardar imagen de debug
cv2.imwrite("debug_face_mesh.jpg", draw_sketch)
```

9.4 Profiling

```
import cProfile
import pstats

profiler = cProfile.Profile()
profiler.enable()

# Ejecutar código
system.run(image_base64)

profiler.disable()
stats = pstats.Stats(profiler)
stats.sort_stats('cumulative')
stats.print_stats(10)
```

---

10. Despliegue

10.1 Despliegue Local

Ejecución Directa

```
# Terminal 1: Servidor
uvicorn app:app --host 0.0.0.0 --port 8000

# Terminal 2: Cliente
flet run main.py
```

Ejecución con Docker

```
# Construir imagen
docker build -t drowsiness-server .

# Ejecutar contenedor
docker run -d -p 8000:8000 --name drowsiness-server drowsiness-server

# Ejecutar cliente
flet run main.py
```

10.2 Despliegue en Producción

Opción 1: Servidor Dedicado

1. Instalar dependencias en servidor
2. Configurar servicio systemd para el servidor
3. Configurar proxy reverso (nginx)
4. Configurar SSL/TLS

Opción 2: Cloud (AWS, Azure, GCP)

1. Crear instancia de VM
2. Instalar Docker
3. Desplegar contenedor
4. Configurar load balancer
5. Configurar auto-scaling

Opción 3: Platform as a Service (Heroku, Railway)

1. Configurar Procfile
2. Configurar variables de entorno
3. Desplegar mediante Git

10.3 Optimización de Rendimiento

Optimización de Código

```
# Usar NumPy vectorizado
distances = np.linalg.norm(points1 - points2, axis=1)

# Reducir resolución de video
frame = cv2.resize(frame, (640, 480))

# Usar threading para I/O
import threading
thread = threading.Thread(target=process_frame)
```

Optimización de Hardware

- Usar GPU para procesamiento (CUDA)
- Aumentar RAM
- Usar SSD para almacenamiento
- Optimizar configuración de cámara

Anexos

A. Referencias de Código

- MediaPipe: https://github.com/google/mediapipe
- OpenCV: https://docs.opencv.org/
- FastAPI: https://fastapi.tiangolo.com/
- Flet: https://flet.dev/docs/

B. Estándares de Código

- PEP 8: Guía de estilo de Python
- Type Hints: Anotaciones de tipo
- Docstrings: Documentación de funciones

C. Herramientas de Desarrollo

- **IDE**: VS Code, PyCharm
- **Linter**: pylint, flake8
- **Formatter**: black, autopep8
- **Testing**: pytest, unittest
- **Version Control**: Git

Fin del Manual de Aplicación del Código Fuente