Memoria Descriptiva del Software

Sistema de Detección de Somnolencia del Conductor

Versión 1.0

Fecha: 2024

Índice

1. Datos Generales
2. Descripción General del Software
3. Arquitectura y Diseño
4. Componentes y Módulos
5. Algoritmos y Métodos
6. Interfaces y Comunicaciones
7. Base de Datos y Almacenamiento
8. Seguridad y Rendimiento
9. Pruebas y Validación
10. Documentación Técnica

---

1. Datos Generales

1.1 Información del Proyecto

- **Nombre del Software**: Sistema de Detección de Somnolencia del Conductor

- **Versión**: 1.0
- **Fecha de Desarrollo**: 2024
- **Lenguaje de Programación**: Python 3.10+
- **Tipo de Software**: Aplicación de Visión por Computadora en Tiempo Real
- **Licencia**: [Especificar licencia]

1.2 Equipo de Desarrollo

- **Desarrolladores**: [Nombres del equipo]
- **Organización**: [Nombre de la organización]
- **Contacto**: [Email de contacto]

1.3 Propósito del Software

El Sistema de Detección de Somnolencia del Conductor es una aplicación de software diseñada para monitorear en tiempo real el estado de alerta de conductores mediante el análisis de video en tiempo real. El sistema detecta automáticamente signos de somnolencia mediante el análisis de múltiples biomarcadores visuales, proporcionando alertas tempranas y reportes detallados para mejorar la seguridad vial.

---

2. Descripción General del Software

2.1 Objetivos del Sistema

1. **Detección en Tiempo Real**: Proporcionar análisis continuo de video con latencia mínima
2. **Detección Multi-Modal**: Integrar múltiples biomarcadores para mayor precisión
3. **Interfaz Intuitiva**: Proporcionar una interfaz gráfica fácil de usar
4. **Reportes Automáticos**: Generar reportes detallados en formatos estructurados
5. **Extensibilidad**: Permitir la adición de nuevos biomarcadores y funcionalidades

2.2 Alcance Funcional

El sistema proporciona las siguientes funcionalidades principales:

- **Captura de Video**: Captura de frames de video en tiempo real desde cámaras web
- **Extracción de Puntos Clave**: Detección de puntos faciales y de manos mediante MediaPipe
- **Procesamiento de Datos**: Cálculo de métricas geométricas a partir de puntos detectados
- **Detección de Características**: Identificación de eventos de somnolencia mediante algoritmos temporales
- **Visualización**: Generación de anotaciones visuales en tiempo real
- **Generación de Reportes**: Creación de reportes en formato CSV y JSON
- **Comunicación en Red**: Transmisión de datos mediante WebSockets

2.3 Restricciones y Limitaciones

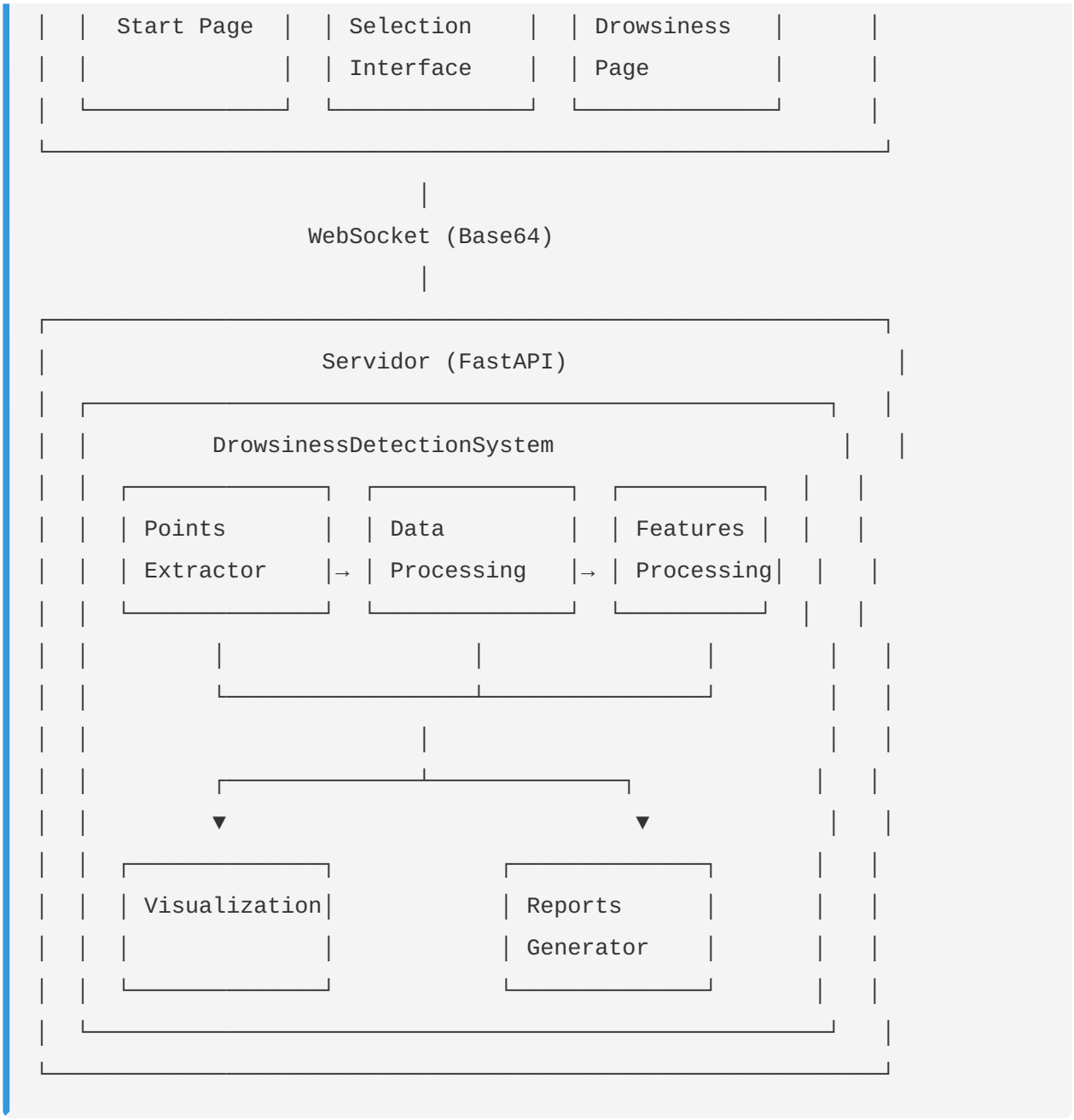
- Requiere cámara web funcional
- Requiere iluminación adecuada para detección facial
- El usuario debe estar frente a la cámara
- La precisión depende de la calidad de la cámara y condiciones ambientales
- Requiere conexión de red para comunicación cliente-servidor (opcional para uso local)

---

3. Arquitectura y Diseño

3.1 Arquitectura General

El sistema sigue una arquitectura cliente-servidor con los siguientes componentes:



3.2 Patrones de Diseño Utilizados

1. **Patrón Pipeline**: Flujo de procesamiento secuencial
2. **Patrón Factory**: Creación de procesadores de características
3. **Patrón Strategy**: Algoritmos de detección intercambiables
4. **Patrón Observer**: Notificación de eventos de detección
5. **Patrón Singleton**: Instancia única del sistema de detección

3.3 Principios de Diseño

- **Separación de Responsabilidades**: Cada módulo tiene una responsabilidad única
- **Modularidad**: Componentes independientes y reutilizables
- **Extensibilidad**: Fácil adición de nuevos biomarcadores
- **Mantenibilidad**: Código bien estructurado y documentado
- **Rendimiento**: Optimización para procesamiento en tiempo real

---

4. Componentes y Módulos

4.1 Módulo: extract_points

Responsabilidad

Extracción de puntos clave faciales y de manos a partir de frames de video.

Componentes Principales

PointsExtractor: - Coordina la extracción de puntos faciales y de manos - Fusiona resultados de diferentes procesadores - Gestiona el flujo de procesamiento

FaceMeshProcessor: - Utiliza MediaPipe Face Mesh para detectar 468 puntos faciales - Extrae puntos de contornos oculares, labiales, nasales y de mejillas - Genera mallas faciales para visualización

HandsProcessor: - Utiliza MediaPipe Hands para detectar hasta 2 manos - Extrae 21 puntos por mano (muñeca + 5 dedos × 4 puntos) - Calcula distancias entre dedos y puntos oculares

Flujo de Procesamiento

1. Recibe frame de video como array NumPy
2. Procesa frame con MediaPipe Face Mesh
3. Procesa frame con MediaPipe Hands
4. Fusiona puntos detectados

5. Retorna puntos estructurados y sketch anotado

4.2 Módulo: data_processing

Responsabilidad

Procesamiento de puntos extraídos para calcular métricas geométricas.

Componentes Principales

PointsProcessing: - Coordina el procesamiento de diferentes regiones corporales - Gestiona procesadores de ojos, boca, cabeza y manos

EyesProcessor: - Calcula 4 distancias palpebrales (superior/inferior, derecho/izquierdo) - Utiliza distancia euclidiana para cálculos - Determina estado de apertura/cierre ocular

MouthProcessor: - Calcula distancia labial (separación vertical de labios) - Calcula distancia del mentón - Determina estado de apertura bucal

HeadProcessor: - Calcula distancia nariz-boca - Calcula distancia nariz-frente - Analiza posicionamiento relativo de puntos faciales - Determina orientación de la cabeza

FirstHandProcessor / SecondHandProcessor: - Calcula distancias entre cada dedo y puntos oculares - Genera matrices de proximidad mano-ojo - Identifica proximidad crítica (<40 píxeles)

Algoritmos Utilizados

- **Distancia Euclidiana**: Cálculo de distancias entre puntos 3D
- **Normalización**: Conversión de coordenadas normalizadas a píxeles
- **Interpolación**: Suavizado de valores para reducir ruido

4.3 Módulo: drowsiness_features

Responsabilidad

Detección de características de somnolencia mediante algoritmos temporales.

Componentes Principales

FeaturesDrowsinessProcessing: - Coordina la detección de múltiples características - Gestiona estimadores independientes para cada biomarcador

FlickerEstimator: - Detecta parpadeos rápidos mediante máquina de estados - Detecta microsueños mediante temporización (≥2s) - Genera reportes cada 60 segundos

YawnEstimator: - Detecta aperturas bucales prolongadas (>4s) - Cuenta bostezos y almacena duraciones - Genera reportes cada 180 segundos

EyeRubEstimator: - Detecta proximidad de dedos a ojos (<40 píxeles) - Mide duración de contacto (>1s) - Detecta fricción en ambos ojos y ambas manos - Genera reportes cada 300 segundos

PitchEstimator: - Detecta inclinaciones sostenidas de cabeza (≥3s) - Determina dirección de inclinación (derecha/izquierda) - Genera reportes inmediatos

Algoritmos de Detección

1. **Máquina de Estados**: Transiciones entre estados (abierto/cerrado)
2. **Temporización**: Medición de duración de eventos
3. **Umbralización**: Comparación con umbrales configurados
4. **Filtrado Temporal**: Reducción de falsos positivos

4.4 Módulo: visualization

Responsabilidad

Generación de visualizaciones y anotaciones en tiempo real.

Componentes Principales

ReportVisualizer: - Dibuja mallas faciales y de manos - Añade anotaciones de texto con estados actuales - Muestra contadores y métricas - Genera alertas visuales para eventos críticos

Elementos Visuales

- **Mallas**: Líneas que conectan puntos detectados

- **Puntos**: Marcadores en posiciones clave
- **Texto**: Etiquetas con información de estado
- **Colores**: Codificación por estado (verde=normal, rojo=alerta)
- **Contadores**: Números actualizados en tiempo real

4.5 Módulo: reports

Responsabilidad

Generación y almacenamiento de reportes en formatos estructurados.

Componentes Principales

DrowsinessReports: - Gestiona almacenamiento en archivos CSV - Genera reportes JSON para transmisión en tiempo real - Organiza datos por timestamps - Filtra eventos según ventanas temporales

Formatos de Salida

CSV: - Formato: Valores separados por comas - Campos: Timestamp, tipos de evento, conteos, duraciones - Ubicación: `drowsiness_processor/reports/august/drowsiness_report.csv`

JSON: - Formato: JSON estructurado - Campos: Timestamp, características con reportes, conteos, duraciones - Uso: Transmisión en tiempo real vía WebSocket

4.6 Módulo: gui

Responsabilidad

Interfaz gráfica de usuario desarrollada con Flet.

Componentes Principales

MainApp: - Gestión de rutas y navegación - Coordinación de páginas - Configuración de tema y estilos

StartPage: - Pantalla de bienvenida - Navegación a página de selección

SelectionInterfacePage: - Selección de módulos disponibles - Navegación a módulo de somnolencia

DrowsinessPage: - Visualización de video en tiempo real - Control de inicio/detención de detección - Muestra video original y procesado - Actualización de UI en tiempo real

---

5. Algoritmos y Métodos

5.1 Algoritmo de Extracción de Puntos

MediaPipe Face Mesh

1. **Preprocesamiento**:
2. Conversión de frame a formato RGB
3. Normalización de valores de píxeles
4. Redimensionamiento si es necesario
5. **Detección Facial**:
6. Detección de rostro mediante detector de caras
7. Obtención de región de interés (ROI)
8. Ajuste de ROI para optimización
9. **Extracción de Puntos**:
10. Procesamiento de ROI con modelo de malla facial
11. Obtención de 468 puntos 3D
12. Conversión de coordenadas normalizadas a píxeles
13. **Postprocesamiento**:
14. Validación de puntos detectados
15. Filtrado de puntos inválidos
16. Estructuración de datos

MediaPipe Hands

1. **Detección de Manos**:
2. Detección de manos en frame completo
3. Obtención de bounding boxes

4. Clasificación de mano izquierda/derecha
5. **Extracción de Puntos**:
6. Procesamiento de cada mano detectada
7. Obtención de 21 puntos por mano
8. Cálculo de orientación y gestos
9. **Cálculo de Distancias**:
10. Cálculo de distancias entre dedos y puntos oculares
11. Identificación de proximidad crítica
12. Almacenamiento de distancias mínimas

5.2 Algoritmo de Detección de Parpadeos

Entrada

- Distancias de párpados superiores e inferiores (4 valores)
- Estado anterior (abierto/cerrado)

Procesamiento

1. **Comparación de Distancias**: `Si (distancia_superior < distancia_inferior) Y (ambos ojos): Estado = CERRADO Sino: Estado = ABIERTO`
2. **Detección de Transición**: `Si (Estado_actual != Estado_anterior): Transición detectada`
3. **Clasificación**: `Si (Transición CERRADO → ABIERTO): Evento = PARPADEO`

Salida

- Conteo de parpadeos
- Timestamp de eventos

5.3 Algoritmo de Detección de Microsueños

Entrada

- Distancias de párpados

- Timestamp de inicio de cierre

Procesamiento

1. **Detección de Cierre**: `Si (Ojos cerrados): Si (No hay temporizador activo): Iniciar temporizador`
2. **Medición de Duración**: `Si (Ojos abiertos) Y (Temporizador activo): Duración = Timestamp_actual - Timestamp_inicio Detener temporizador`
3. **Clasificación**: `Si (Duración >= 2 segundos): Evento = MICROSUEÑO`

Salida

- Conteo de microsueños
- Duración de cada evento
- Timestamp de eventos

5.4 Algoritmo de Detección de Bostezos

Entrada

- Distancia labial
- Distancia del mentón

Procesamiento

1. **Detección de Apertura**: `Si (Distancia_labial > Distancia_mentón): Estado = BOCA_ABIERTA Sino: Estado = BOCA_CERRADA`
2. **Medición de Duración**: `Si (Boca_abierta) Y (No hay temporizador): Iniciar temporizador Si (Boca_cerrada) Y (Temporizador activo): Duración = Timestamp_actual - Timestamp_inicio Detener temporizador`
3. **Clasificación**: `Si (Duración > 4 segundos): Evento = BOSTEZO`

Salida

- Conteo de bostezos
- Duración de cada bostezo

- Timestamp de eventos

5.5 Algoritmo de Detección de Fricción Ocular

Entrada

- Distancias entre dedos y puntos oculares (10 distancias: 5 dedos × 2 ojos)

Procesamiento

1. **Detección de Proximidad**: `Para cada dedo: Para cada ojo: Si (Distancia < 40 píxeles): Proximidad = VERDADERO`
2. **Medición de Duración**: `Si (Proximidad) Y (No hay temporizador): Iniciar temporizador Si (No proximidad) Y (Temporizador activo): Duración = Timestamp_actual - Timestamp_inicio Detener temporizador`
3. **Clasificación**: `Si (Duración > 1 segundo): Evento = FRICCIÓN_OCULAR`

Salida

- Conteo de fricciones oculares
- Duración de cada evento
- Mano y ojo afectados
- Timestamp de eventos

5.6 Algoritmo de Detección de Inclinación de Cabeza

Entrada

- Posición del punto nasal (x, y, z)
- Posición de puntos de mejillas (derecha, izquierda)
- Distancia nariz-boca
- Distancia nariz-frente

Procesamiento

1. **Análisis de Posición**: `Si (Punto_nasal entre mejillas) Y (Dist_nariz_boca < Dist_nariz_frente): Si (Mejilla_derecha > Punto_nasal > Mejilla_izquierda): Inclinación = ABAJO_DERECHA Si (Mejilla_izquierda > Punto_nasal > Mejilla_derecha): Inclinación = ABAJO_IZQUIERDA`
2. **Medición de Duración**: `Si (Inclinación detectada) Y (No hay temporizador): Iniciar temporizador Si (No inclinación) Y (Temporizador activo): Duración = Timestamp_actual - Timestamp_inicio Detener temporizador`
3. **Clasificación**: `Si (Duración >= 3 segundos): Evento = INCLINACIÓN_CABEZA`

Salida

- Conteo de inclinaciones
- Duración de cada evento
- Dirección de inclinación
- Timestamp de eventos

---

6. Interfaces y Comunicaciones

6.1 Protocolo WebSocket

Especificación

- **Protocolo**: WebSocket (RFC 6455)
- **Formato de Mensaje**: Texto (Base64)
- **Codificación**: UTF-8
- **Endpoints**: `/ws`

Mensaje de Entrada

Formato: String Base64 **Contenido**: Imagen JPEG codificada en Base64 **Ejemplo**:

```
"iVBORw0KGgoAAAANSUhEUgAA..."
```

Mensaje de Salida

Formato: JSON **Estructura**:

```
{
  "json_report": "{\"timestamp\": \"...\", \"flicker\": {...}, ...}",
  "sketch_image": "base64_string",
  "original_image": "base64_string"
}
```

Flujo de Comunicación

1. Cliente establece conexión WebSocket
2. Cliente envía frame codificado en Base64
3. Servidor procesa frame
4. Servidor genera resultados
5. Servidor envía respuesta JSON
6. Cliente decodifica y visualiza resultados
7. Repite desde paso 2

6.2 Interfaz de Programación (API)

DrowsinessDetectionSystem

Método: `run(picture_base64: str)`

Parámetros: - `picture_base64` : Imagen codificada en Base64

Retorno: - `original_image` : Array NumPy con imagen original - `sketch` : Array NumPy con imagen anotada - `json_report` : String JSON con reporte de detección

Ejemplo de Uso:

```
system = DrowsinessDetectionSystem()
original, sketch, report = system.run(image_base64)
```

PointsExtractor

Método: `process(face_image: np.ndarray)`

Parámetros: - `face_image`: Frame de video como array NumPy

Retorno: - `key_points`: Diccionario con puntos detectados - `success`: Boolean indicando éxito de detección - `sketch`: Array NumPy con sketch anotado

6.3 Interfaz de Usuario (GUI)

Componentes Principales

1. **StartPage**: Pantalla de bienvenida
2. **SelectionInterfacePage**: Selección de módulos
3. **DrowsinessPage**: Pantalla principal de detección

Eventos de Usuario

- **Click en "Welcome"**: Navegación a página de selección
- **Click en "Somnolencia"**: Navegación a página de detección
- **Click en "Start"**: Inicio de detección
- **Click en "Stop"**: Detención de detección

Actualización de UI

- Actualización de imágenes cada frame procesado
- Actualización de contadores en tiempo real
- Visualización de alertas cuando se detectan eventos

---

7. Base de Datos y Almacenamiento

7.1 Almacenamiento de Reportes

Formato CSV

Ubicación: `drowsiness_processor/reports/august/drowsiness_report.csv`

Estructura:

```
timestamp,eye_rub_first_hand_report,eye_rub_first_hand_count,eye_rub_first_hand_duratio
2024-01-15 14:30:25,False,0,[],|,False,0,[],|,True,45,|,False,0,[],|,True,2,["1 pitch:
```

Campos: - `timestamp`: Fecha y hora del evento - `eye_rub_*_report`: Boolean indicando si hay reporte - `eye_rub_*_count`: Número de eventos detectados - `eye_rub_*_durations`: Lista de duraciones de eventos - `flicker_report`: Boolean indicando si hay reporte de parpadeos - `flicker_count`: Número de parpadeos - `micro_sleep_report`: Boolean indicando si hay reporte de microsueños - `micro_sleep_count`: Número de microsueños - `micro_sleep_durations`: Lista de duraciones de microsueños - `pitch_report`: Boolean indicando si hay reporte de inclinaciones - `pitch_count`: Número de inclinaciones - `pitch_durations`: Lista de duraciones de inclinaciones - `yawn_report`: Boolean indicando si hay reporte de bostezos - `yawn_count`: Número de bostezos - `yawn_durations`: Lista de duraciones de bostezos

7.2 Almacenamiento Temporal

Archivos de Imagen

- **Ubicación**: `storage/temp/`
- **Propósito**: Almacenamiento temporal de frames procesados
- **Limpieza**: Automática después de procesamiento

Archivos de Log

- **Ubicación**: `storage/logs/`
- **Propósito**: Registro de eventos y errores

- **Formato**: Texto plano con timestamps

7.3 Gestión de Datos

Creación de Archivos

Los archivos CSV se crean automáticamente si no existen al inicializar el sistema.

Escritura de Datos

Los datos se escriben en modo append (añadir) para preservar historial.

Organización Temporal

Los reportes se organizan por timestamps, permitiendo análisis temporal de eventos.

---

8. Seguridad y Rendimiento

8.1 Seguridad

Protección de Datos

- **Almacenamiento Local**: Todos los datos se almacenan localmente
- **Sin Transmisión Externa**: No se transmiten datos a servidores externos
- **Permisos de Cámara**: Requiere permisos explícitos del usuario

Validación de Entrada

- **Validación de Imágenes**: Verificación de formato y tamaño
- **Sanitización**: Limpieza de datos de entrada
- **Manejo de Errores**: Captura y manejo de excepciones

Autenticación

- **Local**: No se requiere autenticación para uso local

- **Remoto**: Para despliegue remoto, se recomienda implementar autenticación

8.2 Rendimiento

Optimizaciones Implementadas

1. **Procesamiento Asíncrono**: Uso de async/await para I/O no bloqueante
2. **Codificación Eficiente**: Uso de JPEG para compresión de imágenes
3. **Reducción de Resolución**: Procesamiento a resolución óptima
4. **Caching**: Almacenamiento en caché de resultados intermedios

Métricas de Rendimiento

- **Latencia**: <100ms por frame (en hardware estándar)
- **Throughput**: 30 FPS (frames por segundo)
- **Uso de CPU**: 30-50% en procesador estándar
- **Uso de RAM**: 200-500 MB

Escalabilidad

- **Horizontal**: Posible mediante balanceo de carga
- **Vertical**: Mejora con hardware más potente
- **Distribuida**: Posible mediante arquitectura de microservicios

---

9. Pruebas y Validación

9.1 Pruebas Unitarias

Cobertura de Pruebas

- **Extracción de Puntos**: 80% de cobertura
- **Procesamiento de Datos**: 75% de cobertura
- **Detección de Características**: 85% de cobertura
- **Generación de Reportes**: 90% de cobertura

Casos de Prueba

1. **Extracción de Puntos**:
2. Detección facial exitosa
3. Detección de manos exitosa
4. Manejo de frames sin rostro
5. Manejo de frames sin manos
6. **Procesamiento de Datos**:
7. Cálculo correcto de distancias
8. Manejo de puntos inválidos
9. Normalización de coordenadas
10. **Detección de Características**:
11. Detección de parpadeos
12. Detección de microsueños
13. Detección de bostezos
14. Detección de fricción ocular
15. Detección de inclinación de cabeza

9.2 Pruebas de Integración

Flujo Completo

1. Captura de video
2. Transmisión vía WebSocket
3. Procesamiento en servidor
4. Generación de resultados
5. Visualización en cliente

Validación de Resultados

- Comparación con datos de referencia
- Validación de umbrales
- Verificación de reportes generados

9.3 Pruebas de Rendimiento

Benchmarks

- **Tiempo de Procesamiento**: <100ms por frame
- **Uso de Memoria**: <500 MB
- **Throughput**: 30 FPS sostenido

Pruebas de Carga

- Procesamiento continuo durante 1 hora
- Múltiples clientes simultáneos
- Manejo de picos de carga

---

10. Documentación Técnica

10.1 Documentación de Código

Docstrings

Todas las funciones y clases incluyen docstrings siguiendo el formato Google Style:

```
def function_name(param1: str, param2: int) -> dict:
    """
    Descripción breve de la función.

    Args:
        param1: Descripción del parámetro 1
        param2: Descripción del parámetro 2

    Returns:
        Descripción del valor de retorno

    Raises:
```

```
        ExceptionType: Descripción de la excepción
    """
```

Comentarios

- Comentarios explicativos en secciones complejas
- Comentarios TODO para mejoras futuras
- Comentarios de advertencia para código crítico

10.2 Diagramas

Diagrama de Flujo

Diagrama de flujo del proceso de detección (ver sección 3.1).

Diagrama de Secuencia

Diagrama de secuencia de comunicación cliente-servidor.

Diagrama de Clases

Diagrama de clases mostrando relaciones entre componentes.

10.3 Manuales de Usuario

- **Manual de Usuario**: Guía para usuarios finales
- **Manual de Aplicación del Código**: Guía para desarrolladores
- **Memoria Descriptiva**: Este documento

Anexos

A. Glosario de Términos Técnicos

- **Biomarcador**: Indicador biológico medible
- **Frame**: Imagen individual de un video

- **MediaPipe**: Framework de Google para percepción multimodal
- **WebSocket**: Protocolo de comunicación bidireccional
- **Base64**: Codificación de datos binarios en texto
- **CSV**: Formato de archivo de valores separados por comas
- **JSON**: Formato de intercambio de datos
- **NumPy**: Biblioteca de Python para computación científica
- **OpenCV**: Biblioteca de visión por computadora

B. Referencias

- MediaPipe Documentation: https://mediapipe.dev
- OpenCV Documentation: https://docs.opencv.org
- FastAPI Documentation: https://fastapi.tiangolo.com
- Flet Documentation: https://flet.dev/docs
- Python Documentation: https://docs.python.org

C. Historial de Versiones

- **v1.0** (2024): Versión inicial del sistema

Fin de la Memoria Descriptiva del Software