**DOCUMENTACIÓN TÉCNICA**

Detector de Señas LSC - Análisis de Sistema

**🤟 Detector de Señas LSC**

Lenguaje de Señas Colombiano

Versión: 1.0  
Fecha: 29/09/2025

# Tabla de Contenidos

1. 1. Arquitectura del Sistema
2. 2. Tecnologías Utilizadas
3. 3. Algoritmos de Detección
4. 4. Análisis de Funciones Principales
5. 5. Estructura de Datos
6. 6. Diagramas de Flujo
7. 7. Métricas de Rendimiento
8. 8. Análisis de Precisión
9. 9. Optimizaciones Implementadas
10. 10. Limitaciones y Consideraciones

# 1. Arquitectura del Sistema

El Detector de Señas LSC está diseñado con una arquitectura modular que separa la lógica   
de detección de la interfaz de usuario. Esta separación permite mantener múltiples versiones   
con diferentes interfaces mientras comparten el núcleo de detección.

## Componentes Principales:

|  |  |  |
| --- | --- | --- |
| Componente | Función | Tecnología |
| MediaPipe Integration | Detección de landmarks de mano | MediaPipe Hands |
| Computer Vision | Procesamiento de video | OpenCV |
| Sign Recognition Engine | Lógica de reconocimiento de señas | Algoritmos personalizados |
| GUI Framework | Interfaz gráfica | tkinter + PIL |
| Performance Monitor | Métricas de rendimiento | Threading + time |
| Word Formation System | Formación de palabras | Estado y eventos |

# 2. Tecnologías Utilizadas

## MediaPipe (v0.10.21)

Framework de Google para detección de landmarks corporales. Utilizamos específicamente:  
  
• MediaPipe Hands: Detecta 21 puntos de referencia por mano  
• Modelo de machine learning pre-entrenado  
• Procesamiento en tiempo real optimizado  
• Coordenadas normalizadas (0-1) para independencia de resolución

## OpenCV (v4.11.0.86)

Biblioteca de visión computacional para:  
  
• Captura de video desde cámaras web  
• Procesamiento de imágenes en tiempo real  
• Conversión entre formatos de color (BGR/RGB)  
• Renderizado de elementos gráficos sobre video

## tkinter + PIL/Pillow

Framework de interfaz gráfica nativo de Python:  
  
• Ventanas y widgets nativos del sistema operativo  
• Manejo de eventos de usuario en tiempo real  
• Integración con PIL para mostrar video  
• Threading para operaciones no bloqueantes

# 3. Algoritmos de Detección

## Análisis de Estados de Dedos

La función get\_finger\_states() implementa el algoritmo principal para determinar   
si cada dedo está extendido o doblado:  
  
Algoritmo:  
1. Obtener landmarks de MediaPipe (21 puntos por mano)  
2. Para cada dedo, comparar posiciones de articulaciones  
3. Calcular vectores entre articulaciones consecutivas  
4. Determinar extensión basada en posición relativa Y  
5. Aplicar correcciones específicas por dedo (pulgar usa coordenada X)  
  
Criterios de Detección:  
• Índice: tip\_y < pip\_y (punta menor que articulación proximal)   
• Medio: tip\_y < pip\_y  
• Anular: tip\_y < pip\_y  
• Meñique: tip\_y < pip\_y   
• Pulgar: tip\_x > pip\_x (diferente eje por orientación)

## Detección de Direcciones

La función get\_finger\_directions() extiende el análisis básico:  
  
Estados Posibles:  
• 'up': Dedo extendido hacia arriba  
• 'down': Dedo extendido hacia abajo   
• 'folded': Dedo completamente doblado  
  
Algoritmo Mejorado:  
1. Si el dedo está extendido (según get\_finger\_states())  
2. Comparar posición de punta vs base para determinar dirección  
3. Si no está extendido, clasificar como 'folded'  
4. Aplicar umbrales específicos por dedo

## Detección de Dedos Curvados

La función dedos\_curvados() detecta configuraciones complejas:  
  
Criterio de Curvatura:  
• Calcular distancia euclidiana entre punta y base del dedo  
• Comparar con umbral dinámico basado en tamaño de mano  
• Considerar posición de articulaciones intermedias  
• Clasificar como curvado si distancia < umbral  
  
Aplicaciones:  
• Seña 'C': Todos los dedos curvados  
• Seña 'E': Configuración de puño con curvatura  
• Seña 'M', 'N': Dedos específicos curvados sobre pulgar