Proyecto Final

Smart Wake-Up System

Juan García Moraga, Javier Andrés Bernárdez (Grupo A3-3)

1. Introducción

El objetivo de este proyecto es el desarrollo de un sistema llamado *"Smart Wake-Up System"* el cual realiza la función de un despertador automático implementando técnicas de visión por ordenador.

El sistema se divide en dos bloques principales, un primer bloque para la seguridad del sistema, y un segundo para el funcionamiento del despertador y el "trackeado" del usuario. Para el primer bloque se ha implementado la detección de una secuencia de distintas figuras geométricas, y para el segundo bloque se ha implementado un sistema de seguimiento del usuario obligando a este a desplazarse una distancia mínima para desactivar la alarma.

Repositorio del proyecto: https://github.com/Jandres321/ProyectoCV.git

2. Metodología

2.1.Calibración de la cámara

Para garantizar la precisión geométrica de la detección y el seguimiento, se ha implementado un proceso de calibración offline. El objetivo es eliminar la distorsión de la lente y obtener la matriz de parámetros intrínsecos de la cámara.

Se utilizó un patrón de calibración tipo tablero de ajedrez (chessboard) con dimensiones internas de 8x6 esquinas, capturándose múltiples imágenes del patrón en diversas orientaciones y distancias.

Detección aproximada de esquinas mediante `cv2.findChessboardCorners`. Refinamiento de la posición de las esquinas a nivel subpíxel utilizando `cv2.cornerSubPix`, mejorando la precisión de las coordenadas.

Finalmente se realiza el cálculo de la matriz de la cámara y coeficientes de distorsión mediante `cv2.calibrateCamera`.

Los parámetros intrínsecos y los coeficientes de distancia se guardan en un archivo llamado **calibration_parameters.npz** para luego ser usados por el programa principal

Los parámetros obtenidos se cargan al inicio de la ejecución principal. Se genera un mapa de rectificación con `cv2.initUndistortRectifyMap` y se aplica a cada frame capturado utilizando `cv2.remap` antes de cualquier procesamiento posterior.

Los resultados de calibración obtenidos son:

- **RMS** = 1.8

- **Matriz de parámetros intrínsecos (K)**:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2272.51 & 0 & 1279.01 \\ 0 & 2281.07 & 947.43 \\ 0 & 0 & 1 \end{bmatrix}$$

2.2.Arquitectura del sistema

El programa opera como una máquina de estados con los siguientes modos: BLOQUEADO, COUNTDOWN, TRACKING, UNLOCKED.

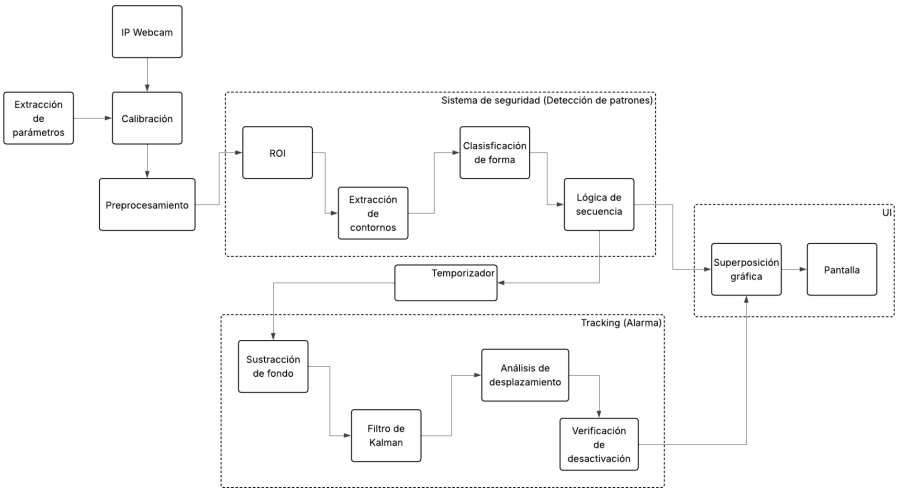


Diagrama de flujo del sistema

2.3.Secuencia de transformación de la imagen (ROI)

Para la detección de patrones, nos centraremos directamente en la parte central de la imagen de 450*450 píxeles, ignorando el resto. La cadena de procesamiento implementada en la función `compute_contours` es:

1. **Imagen ROI:** Selección parcial de la imagen
2. **Escala de Grises:** Simplificación de canales.
3. **Suavizado Gaussiano (`cv2.GaussianBlur`):** Eliminación de ruido de alta frecuencia.
4. **Umbralización Adaptativa (`cv2.adaptiveThreshold`):** Permite detectar figuras incluso con iluminación no uniforme, generando una imagen binaria.
5. **Apertura Morfológica (`cv2.morphologyEx`):** Elimina el ruido residual (puntos blancos pequeños) y separa objetos conectados.

2.4.Sistema de seguridad: detección

El módulo de seguridad valida la identidad analizando los contornos detectados en la ROI.

Algoritmo de Clasificación: Se utiliza `cv2.approxPolyDP` con una tolerancia del 2% del perímetro para simplificar el contorno. La clasificación se basa en la geometría:

- **Círculo:** más de 7 vértices, convexo y circularidad > 0.7.
- **Cuadrado:** 4 vértices, convexo.
- **Flecha:** 4 vértices, no convexo.
- **Montaña:** 5 vértices, no convexo, 1 defecto de convexidad.
- **Pico:** 5 vértices, no convexo, 2 defectos de convexidad.

Usar `cv2.convexityDefects` permite distinguir entre la "Montaña" y el "Pico", ya que ambos son pentágonos cóncavos.

Lógica de Acceso: La secuencia de desbloqueo programada es, (aunque esta puede ser cambiada): **MONTAÑA, PICO, FLECHA, CÍRCULO**. Si el usuario muestra una figura incorrecta el sistema reinicia la secuencia (`detected_sequence = []`), obligando a empezar de nuevo. Para permitir que el usuario pueda cambiar de figura sin que se produzcan detecciones erróneas, se emplea un temporizador de seguridad de 2 segundos siguientes a una detección válida, donde ninguna figura que se pueda detectar erróneamente afectará a la contraseña.

2.5.Sistema propuesto: Tracker y Desactivación

Una vez superada la seguridad, comienza una cuenta atrás y se activa el sistema de tracking.

- **Detección de Movimiento:** Se emplea `cv2.createBackgroundSubtractorMOG2` para aislar al usuario del fondo estático.
- **Filtrado de Kalman:** Se implementó la clase `AutoKalmanTracker`. Este filtro predice la posición futura del usuario basándose en su velocidad y corrige la estimación con la nueva medición del centroide. Esto suaviza la trayectoria y mantiene el seguimiento ante fallos momentáneos de detección.
- **Condición de Desactivación:** El sistema calcula la distancia euclídea entre la posición inicial (P_0) y la actual (P_t).

$$D = \sqrt{(x_t - x_0)^2 + (y_t - y_0)^2}$$

La alarma solo se desactiva si $D > 700$ píxeles, visualizado mediante una barra de progreso verde sobre la interfaz.

3. Resultados

Las pruebas funcionales arrojaron los siguientes resultados:

- **Robustez geométrica:** La combinación de aproximación poligonal y defectos de convexidad permite una tasa de acierto alta en la distinción de figuras complejas (Montaña vs Pico) manteniendo un rendimiento más que aceptable.

- **Rendimiento:** El sistema opera en tiempo real mostrando los FPS en pantalla. Pese a correcciones de distorsión y el cálculo del filtro de kalman, el sistema se muestra estable alrededor de los 30 fps.

- **Experiencia de usuario:** La barra de progreso, mensajes en pantalla y contadores visibles proporcionan retroalimentación inmediata.

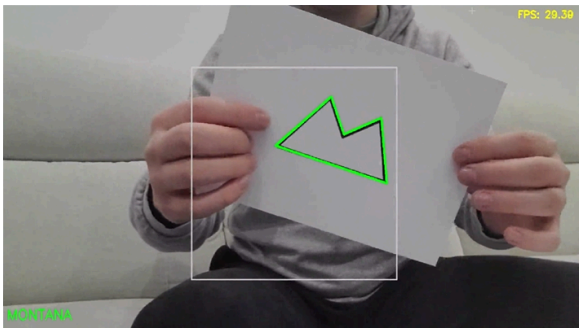


Figura 3: Detección exitosa en el módulo de seguridad.

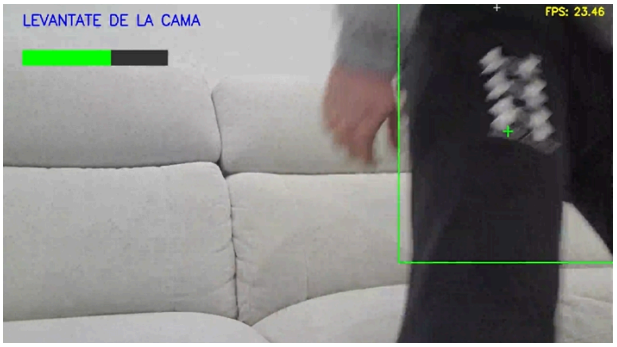


Figura 4: Interfaz de seguimiento durante la actividad física.

4. Futuros desarrollos

Para iteraciones futuras del proyecto se proponen las siguientes mejoras:

1. **Integración de audio:** Sincronizar el estado **TRACKING** con una alarma sonora real.
2. **Reconocimiento facial:** Permitir detectar la cara del usuario con detección facial (Haar Cascades o DL) para mejorar el tracking de este.
3. **Adaptabilidad lumínica:** Implementar algoritmos de balance de blancos automático para mejorar la segmentación en habitaciones oscuras.

5. Bibliografía

- *OpenCV Documentation*. https://docs.opencv.org/4.x/
- Materiales de clase

6. Uso de IA generativa

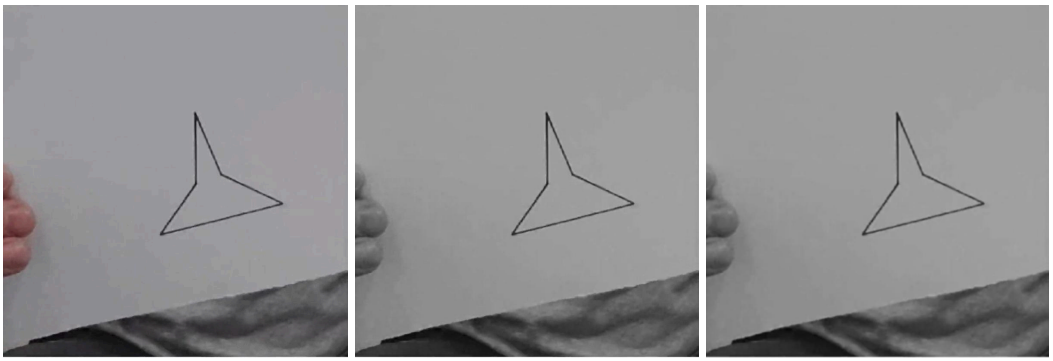
Se ha hecho uso de GeminiPro y el autocompletado de copilot para:

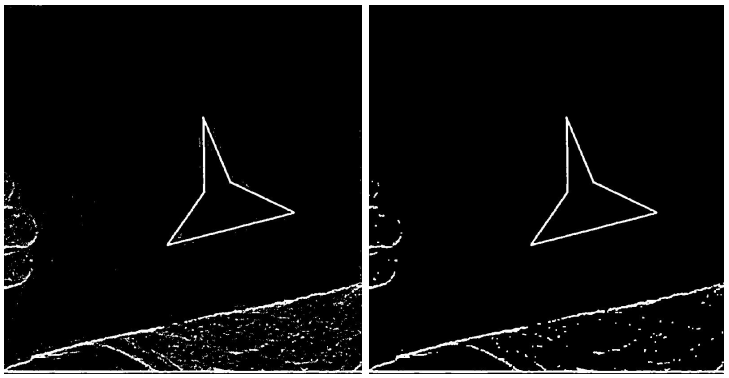
- Conversión de filtro de Kalman en una clase.
- Creación de barra de progreso.
- Optimización de salida de video.
- Solución de errores.
- Redacción del README

7. Anexos

7.1. Anexo 1: Chessboards

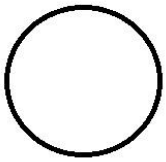
7.2. Anexo 2: Pasos de transformación de la imágen





Pasos de transformación de la imagen 1-5

7.3. Anexo 3: Posibles figuras geométricas

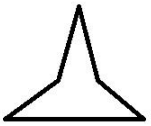


Círculo

Cuadrado

Flecha

Montaña



Pico