

Proyecto Final: Laboratorio Computer Vision

**Miguel Ángel Huamani Salinas y Jorge
Ibinarriaga Robles**

Grupo B

**Visión por Ordenador I
3º Grado en Ingeniería Matemática e Inteligencia Artificial**

Índice

Introducción	3
Metodología.....	3
Calibración de la cámara.....	3
Diagrama de bloques del Sistema	4
Secuencia de transformación de la imagen	4
Sistema de seguridad: Detección de patrones.....	5
Sistema de seguridad: Extracción de información	6
Sistema propuesto: ampliaciones y salida de vídeo	7
Resultados	7
Futuros Desarrollos	7

Link al github: <https://github.com/miguelangel-huamani/cv-final-project>

Introducción

Este proyecto tiene como objetivo el desarrollar un sistema capaz de permitir al usuario introducir la contraseña de su ordenador usando únicamente los ojos, más concretamente, en la detección y seguimiento de círculos oscuros (que es equivalente al conjunto iris-pupila, o a la pupila únicamente si el usuario tiene los ojos muy claros), a través de la cámara de la Raspberry Pi. El sistema utiliza técnicas de procesamiento de imágenes y detección de características para identificar los patrones de parpadeo o desaparición de los ojos, que posteriormente se traducen a una contraseña numérica basada en código Morse, y se compara con una contraseña real establecida previamente.

El propósito de este proyecto es explorar métodos alternativos y accesibles para la entrada de información en un dispositivo con cámara, como bien puede ser un ordenador. El enfoque del proyecto tiene aplicaciones potenciales en la seguridad informática, ya que permitiría al usuario introducir la contraseña de su ordenador sin la necesidad de teclear nada (reduciendo el riesgo de ser observado en el teclado), pero sobre todo para aquellas personas con discapacidades físicas que no puedan introducir la contraseña de otra forma. El único requisito de este sistema sería que el usuario supiese la codificación Morse de los números.

NOTA IMPORTANTE: Siempre que se refiera a iris en este informe se referirá a los círculos oscuros detectados (iris-pupila o pupila). Además, por incompatibilidad de Mac con la visualización de la cámara, las pruebas en este informe se han hecho, en vez de en tiempo real, sobre videos grabados desde la Raspberry Pi y luego enviados al ordenador. Por lo tanto, aunque el trabajo este orientado a un video grabado, se puede adaptar fácilmente a tiempo real, cambiando la entrada de vídeo a la cámara de la Raspberry Pi directamente.

Metodología

Calibración de la cámara

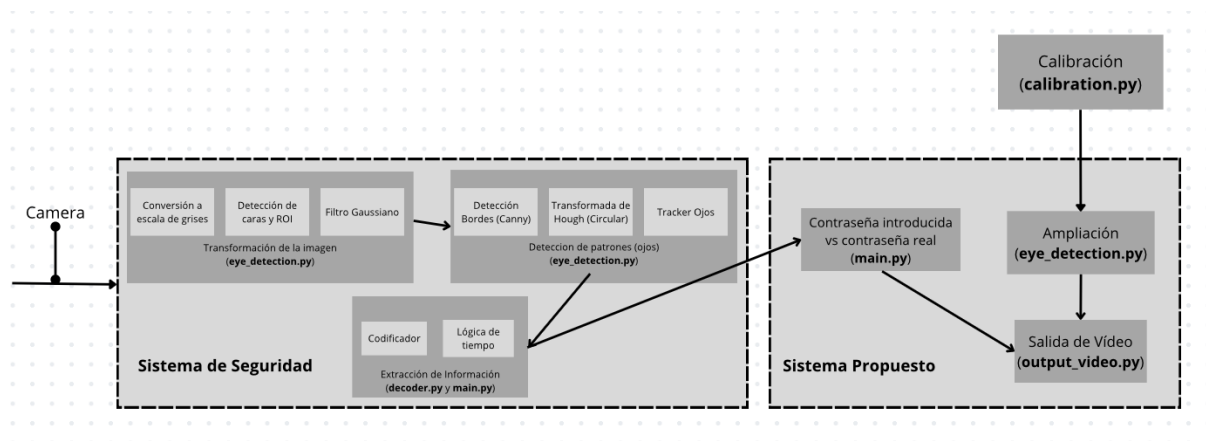
El primer paso ha consistido en la calibración de la cámara con el uso de un patrón de ajedrez (de dimensiones 6x9), impreso en papel y realizado tomando multiples fotos desde distintos ángulos (el total de fotos ha sido 29, y estas se han almacenado en la carpeta **calibration_images**). Tras realizar el proceso de detección de coordenadas 3D y 2D, y la funcion cv2.calibrateCamera de OpenCV, obtenimos los siguientes resultados:

- Matriz intrínseca (K):
$$\begin{bmatrix} 925.35 & 0 & 295.24 \\ 0 & 938.04 & 274.86 \\ 0 & 0 & 1 \end{bmatrix}$$
- Coefficientes de distorsión: [1.11e-01 , -2.11e+08 , 2.99e-03 , -1.43e-02 , 1.02e+01]
- Error RMS de calibración: 0.4012

Un error aproximado de 0.4 indica una calibración precisa, por lo que utilizaremos los parámetros obtenidos para los siguientes apartados, ya que estos nos ayudarán tanto en la eliminación de distorsión en las imágenes como en la reconstrucción 3D de las escenas.

Diagrama de bloques del Sistema

La siguiente figura muestra el diagrama de bloques del sistema donde se observa la arquitectura principal del sistema:



Los bloques grises oscuros indican las operaciones o conjunto de operaciones realizadas sobre la imagen obtenida de la cámara, junto con el nombre del módulo donde se han realizado.

Secuencia de transformación de la imagen

Se han realizado la siguiente serie de transformaciones sobre los frames capturados por la cámara de la Raspberry Pi para extraer información relevante y procesar los datos necesarios para la detección de iris:

1. Conversión a escala de grises

El primer paso consiste en pasar cada imagen a escala de grises con el fin de simplificar el procesamiento de la imagen y ayudar a la precisión de los algoritmos de detección.

2. Detección de caras (Algoritmo de Viola-Jones)

A forma de delimitar la búsqueda de iris en los frames, se ha seleccionado una primera región de interés que contiene el rostro detectado por la cámara. Para esto se ha utilizado el algoritmo de Viola-Jones, implementado en OpenCV mediante cascadas Haar (haarcascade_frontalface_default.xml).

3. Definición de la región de interés (ROI)

Tras detectar la cara hemos decidido delimitar aún más dentro de la región del rostro, donde se encuentran los ojos y el iris, reduciendo el área de búsqueda y mejorando la precisión del sistema.

4. Aplicación de filtro Guassiano

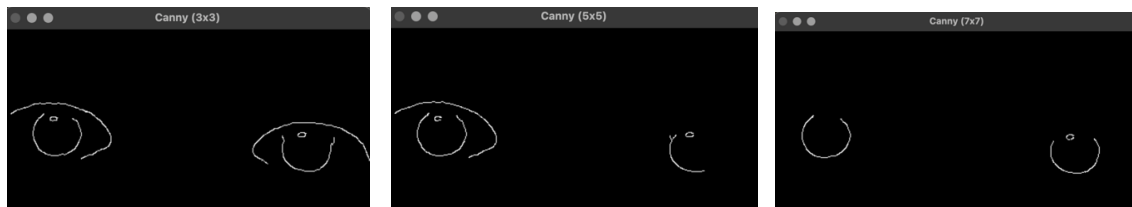
Para eliminar el ruido y resaltar las características relevantes, como los bordes del iris, se ha aplicado un filtro Gaussiano con kernel de tamaño 7x7 y desviación típica 2 (`cv2.GaussianBlur(roi_gray, (7, 7), 2)`), esto sobre la región de interés en escala de grises.

Sistema de seguridad: Detección de patrones

El siguiente paso consiste en reconocer los patrones del iris, es decir, dos círculos negros que se encuentran en la ROI. Esto se ha logrado implementando lo siguiente:

1. Detección de Bordes (Canny)

Para localizar los bordes del iris, hacemos uso del algoritmo de detección de bordes de Canny (`cv2.Canny(blurred, 50, 150)`). Las siguientes imágenes muestran los bordes detectados para distintos tamaños de kernel usados en el previo filtro gaussiano.



Como se puede observar en las imágenes, a menor filtro Gaussiano, más bordes se detectan alrededor del iris (en este caso el borde del ojo, donde se encuentran las pestañas). Dependiendo del frame, los bordes afectan a la detección del iris, por eso elegimos el tamaño de kernel 7x7, que como se ve, devuelve los bordes centrados en los iris y con la forma más redonda y completa de todos.

2. Transformada de Hough (Circular)

A modo de contrastar que los bordes detectados son en efecto iris redondos, hemos hecho uso de la Transformada de Hough de Círculos, que buscará las formas circulares correspondientes a los iris.

El método utilizado es `cv2.HoughCircles` con el algoritmo `cv2.HOUGH_GRADIENT`, que identifica círculos basándose en gradientes de intensidad. Entre los parámetros, `param2=15` es el más crucial, ya que determina la sensibilidad para validar círculos, ajustado tras pruebas para evitar falsos negativos cuando los ojos se ven afectados ligeramente por las pestañas. Además, `minDist=30` establece la distancia mínima entre círculos detectados, mientras que `minRadius=8` y `maxRadius=20` restringen el tamaño de los círculos, acorde al rango esperado del iris dada la proximidad del usuario a la cámara (en caso de que no se detecte el iris en personas con ojos claros, se puede reducir el radio mínimo para centrarse en la pupila en vez del conjunto iris-pupila).

3. Tracking de la pupila

Hemos decidido incluir el apartado de tracking en esta sección ya que forma parte de la detección de iris, pero estará relacionado con la sección de extracción de información de forma directa mediante el uso de un booleano llamado: **eyes_detected_now**.

Tras detectar círculos con la transformada de Hough, se filtran los no deseados calculando la intensidad media de los píxeles dentro de cada círculo mediante una máscara binaria. Si la intensidad media supera un umbral de 60, el círculo se descarta, evitando falsos positivos y mejorando la precisión en la detección de iris.

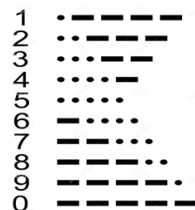
Las coordenadas globales del centro del círculo se obtienen sumando las coordenadas relativas de la ROI a las del círculo detectado. Con estas, se dibuja un círculo verde en la imagen original para indicar un círculo válido y un pequeño círculo rojo en el centro para destacar la ubicación exacta de la pupila.

Sistema de seguridad: Extracción de información

La obtención de la secuencia de caracteres correspondiente a una contraseña numérica se realiza mediante un proceso de codificación, basado en la presencia y ausencia de los iris durante un intervalo de tiempo determinado.

Codificador: Lógica de Detección y Decodificación

Está basado en una secuencia binaria que se genera observando la presencia o ausencia de los iris, ya que cada vez que se detectan, se asigna un “1” a la secuencia binaria, mientras que cuando no se detectan, se asigna un “0”. Tras registrarse 5 bits, se procesa la secuencia en una función “traductor Morse” que busca el número correspondiente o devuelve “NUMERO INTRODUCIDO NO VALIDO” si este no se encuentra. Este usa la siguiente tabla (en un diccionario) dónde las rayas equivalen a “1” y los puntos a “0”.



1	· ---
2	·· --
3	··· -
4	····
5	--- ·
6	--- ··
7	--- ···
8	--- ····
9	--- ···· ·
0	--- ·····

Tras obtener el número encriptado, este se añade a la contraseña de prueba hasta que esta tenga 4 números (que es la longitud de la contraseña real). Al final del programa se comprobará que ambas contraseñas sean iguales para desbloquear el ordenador.

Lógica Relacionada con el Tiempo

A modo de saber cuando los iris son detectados hemos hecho uso del booleano **eyes_detected_now**, sin embargo, un problema surgía cuando, a pesar de estar los ojos abiertos, en algunos frames no se detectaban, cambiando el valor del booleano y añadiendo falsos “0” a la contraseña. Por esta razón el **main.py** incluye una lógica basada en un intervalo de tolerancia de 0.5 segundos. Si se detectan ojos cerrados se comprueba que la última vez que estuvieron abiertos fue hace más de 0.5 segundos, ya que sino es así, es porque de verdad sí estaban abiertos.

Si la diferencia de tiempo entre dos detecciones consecutivas es mayor que un umbral (**time_threshold**), se considera una nueva detección válida y se añade el “1” o “0” correspondiente a la secuencia binaria (**password**).

Sistema propuesto: ampliaciones y salida de vídeo

Los parámetros de calibración se guardaron en un archivo **calibration_params.xml**, que se cargaron en el modulo **eye_detection.py** para corregir cada frame posiblemente distorsionado. Por otro lado, el módulo de salida de video se encarga de mostrar en tiempo real los resultados del procesamiento del video. En la pantalla se visualizan los frames del video con los iris detectados resaltados mediante círculos, lo que facilita la identificación de las detecciones realizadas. Además, se muestra la contraseña conforme se recopila, utilizando texto en color verde en la esquina superior izquierda. Al finalizar la recopilación de la contraseña, se muestra un mensaje final indicando si esta es correcta o incorrecta, con texto en verde o rojo según el resultado, manteniendo el video visible durante unos segundos para que el usuario pueda leerlo antes de cerrarse automáticamente.

Resultados

El umbral de tiempo de 1 segundo, y por ser corto, a veces ha resultado en algún ligero problema ya que si el usuario se retrasa en abrir o cerrar los ojos ligeramente es posible que el sistema registre un “0” o “1” a destiempo. Es por esto por lo que se puede cambiar el umbral para que el usuario pueda llevar mejor la cuenta del tiempo.

Como se explicó en la introducción, los resultados obtenidos y que se ven en el vídeo han sido probados sobre un vídeo grabado desde la Raspberry y no en tiempo real. A pesar de no poder probado el sistema en tiempo real, el resultado es equivalente, por lo que concluimos en que si el usuario lleva bien la cuenta del tiempo y se encuentra a una distancia aceptable de la cámara de la Raspberry, entonces el patrón introducido será bien interpretado, y si es el correcto, desbloqueará con éxito el ordenador.

Futuros Desarrollos

Entre las posibles mejores y ampliaciones de este proyecto se incluye:

- Optimización de la detección: En las condiciones en las que se ha probado el sistema, el ojo del usuario estaba tanto a una distancia, como a una iluminación aceptable. Es posible que si estos parámetros varían, el sistema no funcione también como se muestra, y es por esto que se debería mejorar el algoritmo de detección para múltiples condiciones
- Experiencia del usuario: A pesar de mencionar el “desbloqueo” del ordenador, esto se ha hecho verificando que se cumplía: **password == real_password**. Con el fin de aplicar esto a un sistema real, debería unirse el programa con el del ordenador para poder desbloquear un ordenador haciendo una comprobación parecida.