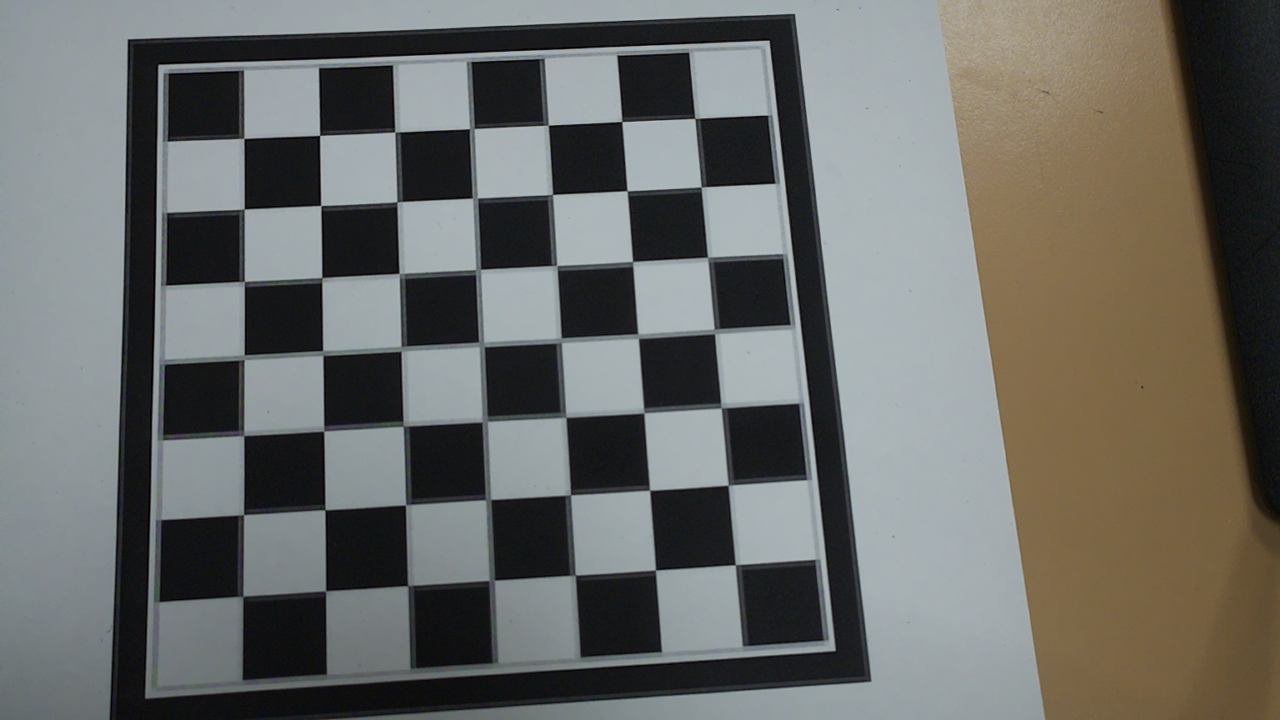
# Introducción

El propósito de este proyecto es idear e implementar un sistema de visión por ordenador utilizando los conocimientos adquiridos en la asignatura. El proyecto constará de dos bloques: un bloque de seguridad, en el que el usuario tiene que identificarse mediante la decodificación de patrones visuales; y un bloque de videojuego. El juego utilizado como base para este proyecto es el “Just Dance”, en el que el jugador tiene que imitar poses de baile para ganar puntos. El proyecto se realizará en una Raspberry Pi, utilizando una cámara como entrada de información del sistema.

Puesto que la idea del proyecto es utilizar nuestros conocimientos de visión por ordenador, incluyendo calibración, procesamiento de imágenes, extracción de características y detección de objetos; el proyecto no será un juego completo sino una demo que demuestre las capacidades del sistema ideado y el potencial que tiene.

# Calibración de cámara

Para calibrar hemos usado el mismo sistema empleado en las practicas. Para comenzar usamos una cuadrícula similar a un tablero de ajedrez y le hicimos fotos con la cámara desde distintos ángulos, distancias y doblando el papel.



Tras esto, en el script de calibrate.py recoge todas estas imágenes y usa el siguiente algoritmo:

1. Obtener las esquinas con findChessboardCorners y tomar aquellas que sean válidas.
2. Obtener las imágenes en blanco y negro.
3. Usando cornerSubPix obtenemos mejores esquinas.
4. Dibuja las esquinas en la imagen y la guarda.
5. Se obtienen los parámetros con calibrateCamera.

Los resultados que hemos obtenido están en Calibrate.txt y son:

Intrinsics:

[[1.52503668e+03 0.00000000e+00 7.86953498e+02]

[0.00000000e+00 1.48317685e+03 1.62634457e+02]

[0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Distortion coefficients:

[[-0.16657612 0.07946734 -0.0005522 0.00902144 0.20447945]]

Root mean squared reprojection error:

2.36587963519781

Pensamos en usar estos para el juego y sistema pero resultaba muy incómodo y vemos que el error no es demasiado grande

# Sistema de Seguridad

Para el sistema de seguridad hemos usado un simple sistema de contraseña con formas geométricas simples.

Antes de comenzar hay que crear la contraseña. Como cosa curiosa hemos añadido un simple [cifrado cesar](https://en.wikipedia.org/wiki/Caesar_cipher) con un step.

La creación de la contraseña tiene lugar en créate\_passwor.py y la crea con el siguiente algoritmo:

1. Toma la contraseña de un input

Valida que cumpla las siguientes normas:

* 1. La longitud no sea impar
  2. Si es de longitud de 2 tiene que tener – delante de t o r
  3. Si es más larga, va letra a letra verificando que tenga la forma: - letra siempre.

1. Cifra la contraseña con el cifrado cesar
2. Guarda la contraseña cifrada en password.txt.

Completada la parte de la creación de la contraseña, se puede iniciar el programa SecuritySystem.py.

Primero hay que insertar el paso del cfrado para la contraseña, que es traducida en una lista de formas.

Este consta de una máquina de estados que va verificando si la forma mostrada en la pantalla es la esperada, la actual o una incorrecta.

Para detectar que forma se está mostrando hemos usado un algoritmo geométrico que usa las esquinas de la imagen para saber que forma es.

Para obtener las esquinas hemos usado el detector de [shi tomasi](https://es.wikipedia.org/wiki/Detector_de_esquinas#El_algoritmo_Harris_&_Stephens_/_Plessey_/_el_Shi-Tomasi_detector_de_esquinas). Este lo sacamos de la función goodFeaturesToTrack. Hemos optado por los siguientes parámetros:  
max\_corners = 7

Quality = 0.2

min\_distance = 7

corner\_color = (255, 0, 255)

radius = 5

Que nos dieron buenos resultados.

La función de deteción también dibuja las esquinas para que al presentarse en pantalla se vean.

Para el ciclo principal, el objetivo es introducir correctamente la contraseña. Con el fin de demostrar el funcionamiento, el programa dice si es correcta o no la figura introducida, pero en un caso de uso, esta funcionalidad sería opcional o eliminada.

Además, el programa te pone en la esquina superior derecha cual fue la última figura introducida.

En un principio pensamos en hacer que fuera sin pulsar ninguna tecla, pero resultó prácticamente imposible dado que al moverse la cámara tanto, nos imposibilitaba que se centrase.

La máquina de estados básicamente aplica:  
Si figura = actual -> Mantener estado.

Si figura = siguiente -> Siguiente estado.

Si no -> Reinicia al primer estado.

# Demo del juego (game.py)

El objetivo de la demo es ser capaz de detectar mandos con un color específico (en este caso hemos utilizado sobres de correos); crear un tracker que siga a ambos mandos, guardando su posición y dibujando una caja que siga al objeto; y comparar la posición del mando con posiciones de referencia para comprobar si el jugador está en la postura correcta.

Además, con el propósito de mostrar este proyecto, se mostrará retroalimentación visual en tiempo real sobre la exactitud de las posiciones detectadas en relación con las ideales, así como la tasa de fotogramas por segundo. Para dibujar las posiciones ideales en pantalla podemos utilizar la función circle de la librería cv2, y para mostrar texto en pantalla utilizaremos la función putText. Para calcular la tasa de fotogramas por segundo mediremos el tiempo que transcurre entre cada iteración de nuestro programa.

Para desarrollar el tracker que sigue a los mandos, se ha utilizado en primer lugar segmentación de color. Los fotogramas se convierten en primer lugar al espacio de color HSV, donde la información sobre el tono y la intensidad están separados. Esto nos permite segmentar el frame por colores, creando una máscara que identifique qué píxeles se encuentran dentro del rango de color definido para los mandos.

La máscara creada será una imagen binaria en la que sólo tendremos los mandos y algún pixel de la escena que se encuentre dentro del rango de color por casualidad. Podemos utilizar la función findContours de la librería cv2 para encontrar el contorno de todos los objetos de la escena. Para mantener solo la información de los mandos, guardaremos solo los contornos que delimiten áreas con un valor superior a un umbral. Esto nos permitirá dibujar las cajas alrededor de los mandos utilizando la función rectangle de cv2.

Además, podemos estimar el centro del área delimitada por el contorno a partir del ancho y la altura de este. Esta información nos permite comparar la posición del mando con la posición del punto en el que queremos situarlo. Si la distancia entre ambos es inferior a un umbral, daremos la posición como válida y la mostraremos en pantalla.

Con esta demo, tenemos todas las bases para diseñar un juego de “Just Dance”. Si vamos generando nuevas posiciones ideales (en este caso solo se ha generado una para poder mostrar el proyecto) tenemos un juego en el que podemos dar puntos al jugador cada vez que alcance la nueva posición ideal, si consigue hacerlo antes de que se genere una posición nueva.