Práctica Final Visión Por Ordenador: Tres en Raya

Jorge Gómez Azor y Carlos Mazuecos Reíllo

Introducción:

Este trabajo de Visión por ordenador consistirá en implementar un sistema de visión por ordenador con el uso de una Raspberry Pi y una cámara conectada a la Raspberry Pi. El trabajo se divide en dos secciones, calibración de la cámara y después el programa con su correspondiente aplicación.

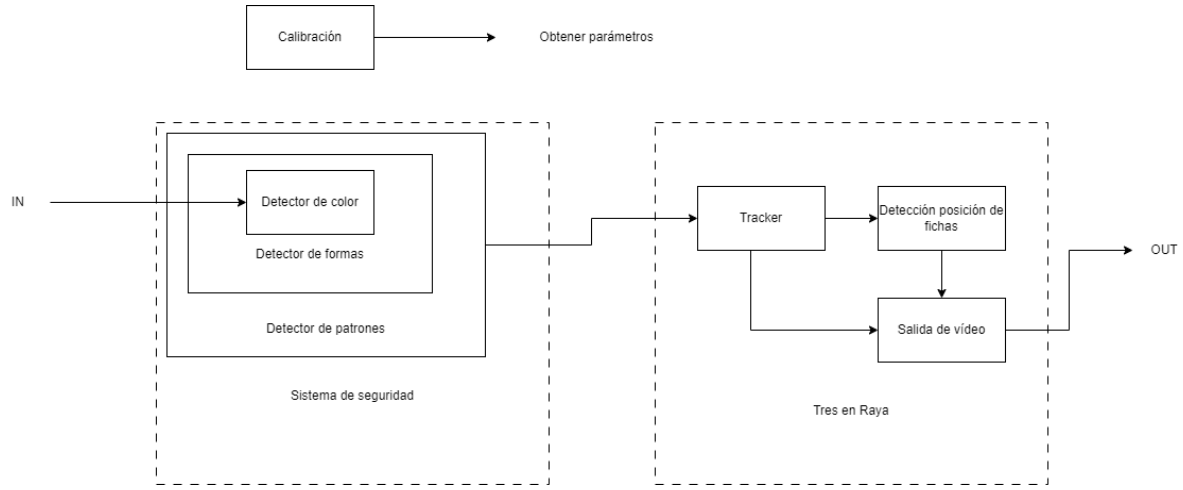
Módulos mínimos:

Calibración para obtener unos parámetros.

El software del programa se estructura en varios módulos que están interconectados entre sí y que trabajan de manera coordinada para garantizar el correcto funcionamiento del programa. El sistema de seguridad se basa en la detección de patrones. En la primera etapa se analizan los colores de la imagen y en la segunda se aborda la identificación especifica de patrones basada en las formas.

Una vez los cuatro patrones han sido identificados, el flujo se transfiere al módulo del Tres en Raya. Se utilizará el módulo del tracker y se visualiza a través de la salida del vídeo proporcionando una representación clara y actualizada de la disposición de las fichas.

En el sistema de seguridad, se detectarán los patrones, detectando primero el color y después la forma. Una vez se hayan detectado los 4 patrones se pasará al tracker. Este tracker se utilizará para detectar la posición de las fichas y mostrarlas por la salida de vídeo.



Calibración:

En el archivo 'camera_calibration.ipynb' se calibrará la cámara. Para ello, se toman unas 30 fotos con la cámara de la Raspberry Pi, desde diferentes ángulos de un tablero de ajedrez 12x11 con casillas de 14 milímetros de lado. Se cargarán todas las imágenes y con la función de cv2 *cv2.findChessboardCorners* se detectarán las esquinas interiores del tablero de ajedrez para cada imagen. A continuación, con *cv2.drawChessboardCorners* se pintarán las esquinas para cada imagen. Es importante aclarar que en caso de que no se vean todas las esquinas de un tablero en la imagen, el programa no dibujará ninguna.

Con una correcta calibración, se detectarán las esquinas interiores si se ven en la imagen todas las esquinas. En las figuras 2 y 3 sí se detectan correctamente a diferencia con la primera en la que, como no se ven todas sus esquinas inferiores izquierdas, no.

Al finalizar estos pasos de calibración, se obtienen unos parámetros de calibración de la cámara. Para ello, el programa filtrará de todas las imágenes, que esquinas son detectadas correctamente y qué imágenes

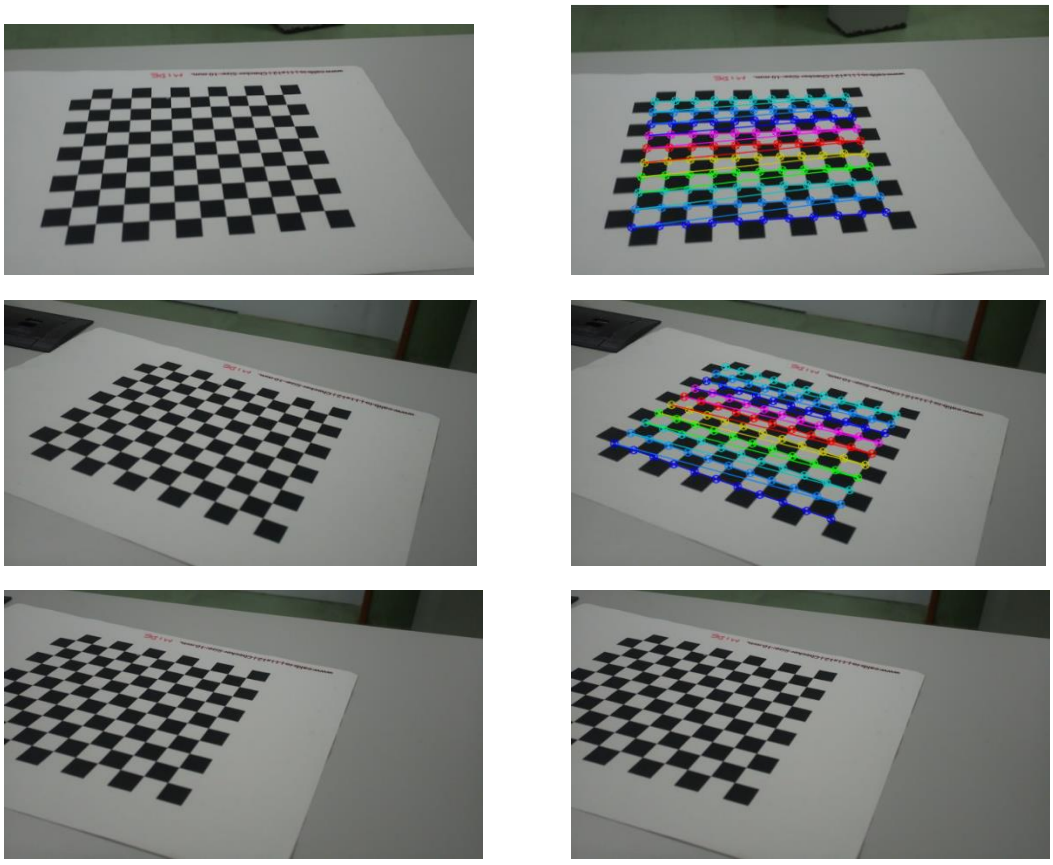
son calibradas correctamente. Por último, con la función *cv2.calibrateCamera* se obtendrán los diferentes parámetros de calibración de la cámara.

Es relevante destacar que se obtiene un RMSE de 0.69 aproximadamente lo cual indica que la cámara está calibrada correctamente. Estos son los parámetros obtenidos en la calibración de la cámara:

```
Corners standard intrinsics:
 [[643.5676128    0.         329.02828778]
 [  0.         683.83069295 300.43414557]
 [  0.           0.           1.        ]]
```

```
Corners standard dist_coefs:
 [[ 0.09265423 -0.79852718  0.00386719  0.00398983  2.47347966]]
root mean sqaure reprojection error:
 0.6897669186319814
```

Ejemplos de detección de esquinas del tablero para calibrar:



Programa:

El programa de Python estará dividido en diferentes secciones. En primer lugar, habrá un sistema de seguridad que habrá que superar y una vez se haya conseguido esto, habrá un videojuego al que jugar.

Sistema de seguridad:

El sistema de seguridad consistirá en la detección de una secuencia predefinida. Cada patrón de la secuencia será una figura de un color concreto. La secuencia concreta para seguir va a ser un triángulo amarillo, pentágono verde, cuadrado azul y triángulo amarillo (La secuencia a detectar se puede cambiar en la variable pattern del main). Cada 10 segundos, se procesará el frame que la cámara esté detectando en ese momento. Si el patrón dentro de la correcta es el correcto, se pasará al siguiente y en caso contrario, se reiniciará la detección de la secuencia.

Para esto, en primer lugar, se filtrará el frame con la función filter_image. El filtrado se realizará sobre por un color concreto, el cual dependerá de en qué patrón de la secuencia se encuentra. Por ejemplo, si se encuentra detectando el primer patrón, ya que hay que detectar un triángulo amarillo, se filtrarán los colores amarillos o si tiene que detectar un triángulo rojo, se filtrarán los colores rojos. Este enfoque consigue que, al aplicar primero el filtro de color, se detectará que la figura que se registra en el frame es del color que se requiere. En caso de que por ejemplo se pasase una figura verde cuando se requiere una roja, haría que después de aplicar los filtros, la imagen fuese entera negra y no se pudiese operar con ella.

Para esto, se pasa el frame a hsv, ya que este espacio de color suele ser más conveniente para el filtrado de colores, ya que separa la información de luminosidad y crominancia de la imagen. Se aplicará el filtro de colores a la imagen y la resultante máscara al frame. Con esto, ya tendremos la imagen filtrada por color.

A continuación, con la imagen resultante después de aplicar todos los filtros, se suavizará con operaciones de procesamiento como la erosión, el umbral adaptativo y la dilatación para resaltar contornos. A continuación, se encontrarán los bordes con la función 'cv2.findContours'. Después, se aproximarán los bordes a un polígono con 'cv2.approxPolyDP' para detectar la forma de los contornos detectados.

Con la función validate_pattern, comprobaremos si el número de lados finas del polígono, después de aplicar todos los filtros a la imagen es igual al del patrón. En caso de que lo sea, se pasará al siguiente y en caso contrario, se reiniciará la detección de la secuencia. Una vez se halla detectado correctamente toda la secuencia, empezará el juego. Ejemplo de detección triángulo rojo:

Frame normal

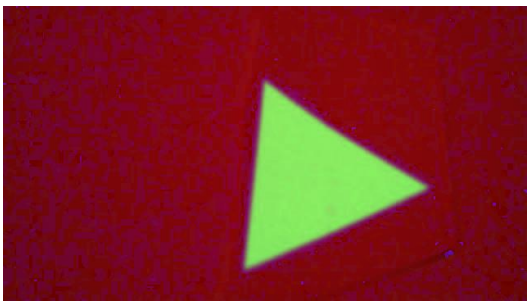
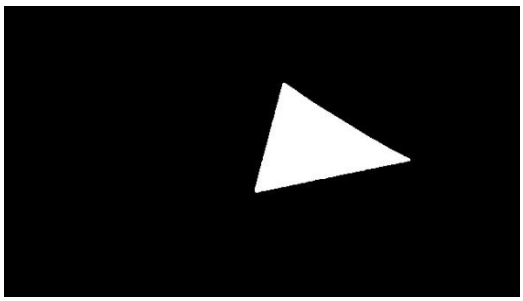
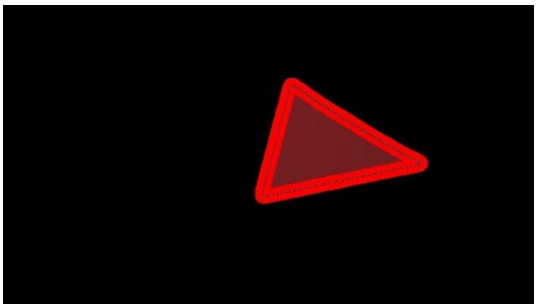


Imagen hsv



Máscara



Detección de la forma

Tracker:

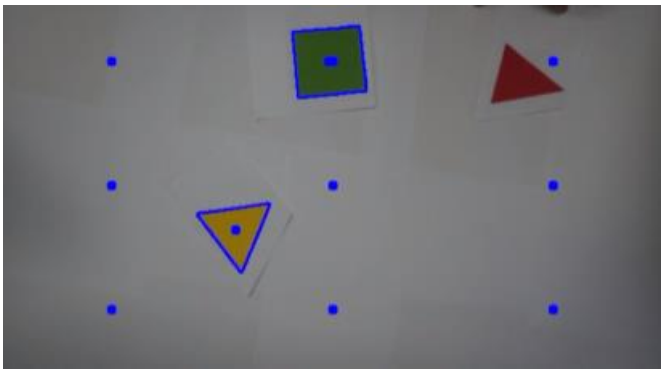
El tracker funcionará cuando se esté jugando a la partida. Habrá dos, uno que seguirá el patrón del jugador 1 (cuadrado verde) y otro para el jugador 2 (triángulo amarillo). Están ambos programados para que únicamente se siga a la figura del color específico.

El funcionamiento para la detección de la figura en el tracker tiene el mismo funcionamiento que el de la detección de patrones, primero se filtra por color y después por número de lados. La diferencia del tracker con la detección de patrones es que el tracker detectará figuras segundo a segundo. La cámara seguirá a las dos figuras que se pasen por el main y no seguirá, siguiendo con el ejemplo propuesto anteriormente,

a un cuadrado azul o un triángulo verde (Aclaración: si se quieren cambiar los patrones de las fichas, se puede cambiar fácilmente en la variable figures_track del main).

Como pueden haber varias fichas sobre el tablero, con la función imutils.grab_contours se detectarán todos ellos para después, con max(cnts, key=cv2.contourArea) calcular el área de todos los bordes y quedarnos solo con el mayor para ese patrón específico.

Con los patrones ya detectados, ahora queda mostrarlos por pantalla en caso de que aparezcan. Si se detectan los patrones, se adaptará el frame y se dibujarán sobre ellos. El tracker funcionará haciendo que se vea por pantalla el borde del patrón y en el centro del polígono formado por el borde, un punto que hará referencia al centro de la figura. Como se observa en la siguiente imagen, el tracker solo funciona para los patrones que se han pasado en el main del programa:

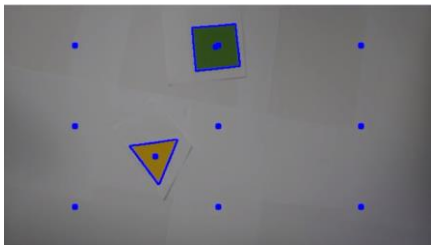


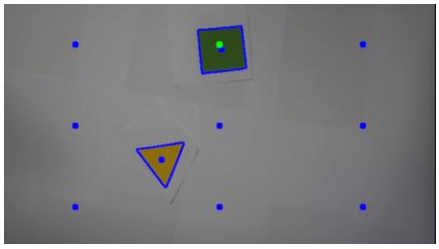
Juego:

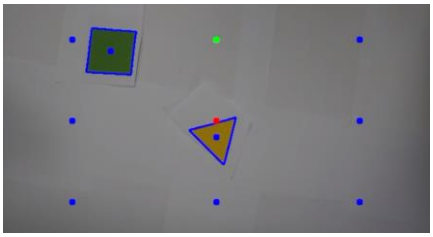
El funcionamiento del juego será similar al de un 'Tres en Raya'. La ficha del jugador uno será el cuadrado verde y la del segundo el triángulo amarillo. Los jugadores tendrán libertad para cambiar las fichas y cada 10 segundos, se evaluará el frame. Si es el turno del jugador uno, se evaluará su ficha, independientemente de la posición de la otra y viceversa.

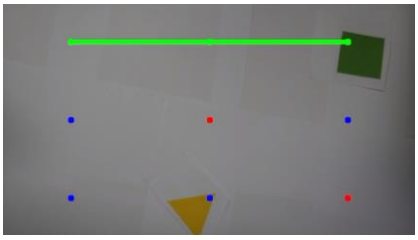
Para ello, cada frame de la cámara se dividirá en 9 casillas y sobre cada casilla se dibuja el centro de ella. Con el tracker, se detectan los centros de las figuras, como se ha explicado ya. Cuando se evalúe el frame, el centro de casilla del cual esté más próximo el tracker del jugador que le toca, se considerará que es donde se ha colocado la figura. El centro pasará a ser de color verde, si es la ficha del jugador 1 o roja si es la del jugador 2.

Ejemplo de cómo se cambian los puntos por pantalla dependiendo de donde estén las fichas y el final del juego, uniendo los puntos que hacen el tres en raya.









Futuros desarrollos

Para futuros desarrollos se propone que los puntos, se adapten al tablero. Es decir, en vez de hacerlo sobre un fondo blanco, que sea un tablero 3x3 de colores y que el programa detecte dónde se encuentran las esquinas. Con esas esquinas ubicar el centro de la casilla y emplear el mismo algoritmo de proximidad de la ficha a trackear con el centro de la casilla.

También se podría añadir una detección de círculos.

Los umbrales cuando se pasa la imagen de bgr a hsv son peores y dependen bastante más de la iluminación que los que se obtienen al pasar de rgb a hsv. Podría intentar examinar eso para arreglarlo y que funcione para cualquier iluminación.

Para más información sobre el código o un ejemplo del funcionamiento:

LINK DEL REPO:

https://github.com/jorgegomezazor/Computer_Vision_FP