Informe
Proyecto Final
Visión por Ordenador

Joaquín Mir Macías

202110004@alu.comillas.edu

Manuel Rodríguez Villegas

manuelrodriguez@alu.comilllas.edu

Curso: 3º iMAT

Fecha de realización: 13/11/2023

Índice

1. Introducción

En este proyecto vamos a utilizar una cámara conectada a la Raspberry Pi para detectar un patrón de figuras geométricas y, si es correcto, abrir el paso para ejecutar un *tracking* de manos. El programa identificará una de las manos del sujeto y contará cuántos dedos están apuntando hacia arriba. Las posibles extensiones de este proyecto se explicarán detalladamente en su correspondiente apartado. Además, también se incluye la posibilidad de ejecutar un *tracker* de caras y un *tracker* de una pelota de tenis que tiene la particularidad de no utilizar algoritmos de *Deep Learning*. Finalmente, el proyecto incorpora la opción de calibrar la cámara utilizada mediante una serie de capturas de pantalla o incluso en tiempo real. A lo largo del informe se explicará cada apartado del proyecto junto a los resultados obtenidos.

2. Metodología

Así, para ejecutar todas las opciones lo primero sería calibrar la cámara mediante el fichero calibration.py. Si se desea, se pueden guardar los parámetros de calibración para utilizarlos en futuros proyectos (en este caso no se hace porque el alcance no lo requiere).

Para la calibración de la cámara conectada a la Raspberry Pi hemos implementado dos métodos:

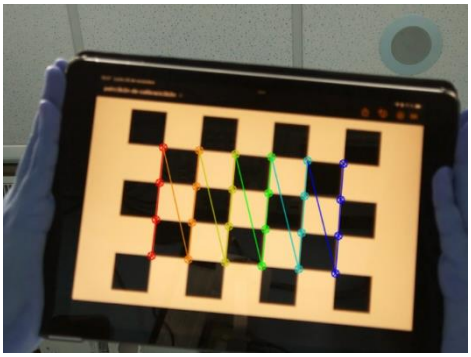
- Capturas de pantalla de las imágenes proporcionadas por la Raspberry PI.
- Medición en tiempo real calibrando a través de fotogramas (*frames*).

Para ello hemos escogido un tablero de cuadrados en blanco y negro de dimensiones (4, 6) y con una anchura por cuadrado de 23mm por cuadrado.

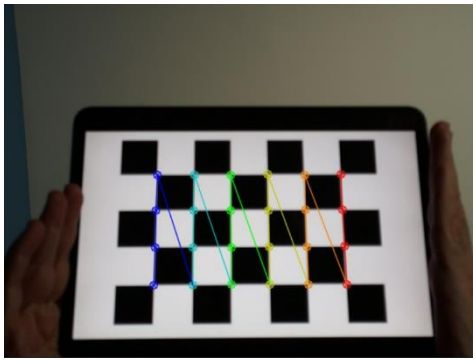
Para hacer la calibración utilizando la librería de “cv2” donde hemos seguido los siguientes pasos (OpenCV, s.f.):

1. Cambio de color de las imágenes: pasamos las imágenes de RGB a *gray* (gama de grises).
2. Búsqueda de los *corners* (esquinas) del tablero: en una primera instancia detectamos las esquinas del tablero usando "cv2.findChessboardCorners". Para una mejor calibración empleamos un método más refinado usando los parámetros CRITERIA de OpenCV.
3. Una vez obtenidas las esquinas del tablero, comparamos las posiciones detectadas con las posiciones teóricas, calculadas con el número de cuadros del tablero y el lado de cada cuadrado. Para calificar el error cometido por la cámara nos basamos en:
   - RMS (Root Mean Square Error): indica el error cuadrático medio entre las coordenadas proyectadas reales y las coordenadas proyectadas calculadas. Un valor bajo de RMS generalmente indica una buena calidad en la calibración.
   - Parámetros Intrínsecos (*intrinsics*): representa la matriz de la cámara (matriz de parámetros intrínsecos). Contiene información sobre la distancia focal, punto principal y otros parámetros intrínsecos de la cámara.
   - Coeficientes de Distorsión (dist_*coeffs*): contiene los coeficientes de distorsión radial y tangencial de la lente de la cámara. La distorsión radial es causada por la forma no ideal de la lente, mientras que la distorsión tangencial es causada por la posición no ideal de la lente con respecto al plano de la imagen.
   - Rotación de los Vectores (*rvecs*): obtenemos una lista de vectores de rotación para cada una de las imágenes de calibración. Cada vector de rotación representa la orientación de la cámara en términos de un vector que indica la rotación necesaria para transformar de la posición original a la posición actual.
   - Traslación de los vectores (*tvec*): obtenemos una lista de vectores de traslación para cada una de las imágenes de calibración. Cada vector de traslación representa la posición de la cámara en términos de las coordenadas de traslación en el espacio 3D.
   - Parámetros Extrínsecos: los utilizamos para calcular matrices de transformación extrínsecas que representan la posición y orientación de la cámara en el espacio tridimensional para cada imagen de calibración. Esto es esencial para relacionar los sistemas de coordenadas de la cámara y del mundo.

En este apartado nosotros hemos implementado todo el procesado que se realiza al fotograma para poder obtener las matrices correspondientes, así como el bucle para la calibración en tiempo real.



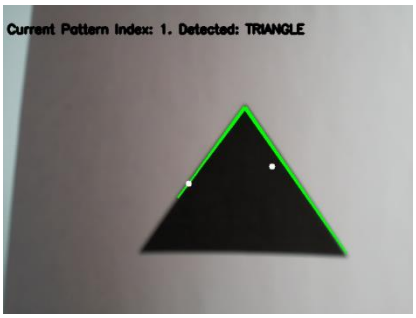
Calibración con Snapshots

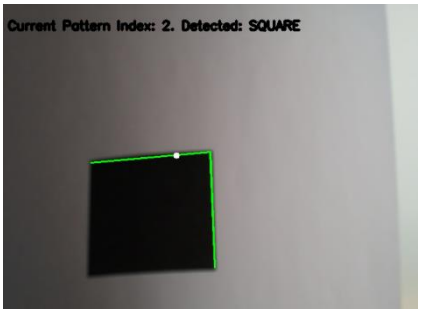


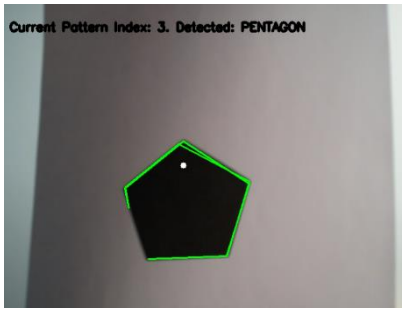
Calibración en Tiempo Real

El segundo fichero a ejecutar sería pattern_and_tracking.py. Aquí se empieza definiendo las variables necesarias para la detección de manos (utilizando la librería *mediapipe* (MediaPipe, 2023) y su solución *hands*), así como todas las funciones necesarias para el proceso. Así, la ejecución comienza identificando el patrón de figuras que desbloqueará el sistema de *tracking*. En nuestro caso, utilizamos tres figuras para el patrón: un triángulo, un cuadrado y un pentágono, siendo el patrón: TRIANGLE, SQUARE, PENTAGON, TRIANGLE. A continuación, se describe el funcionamiento del detector de patrones:

La parte fundamental del detector es la función que recibe una imagen y devuelve qué figura aparece en ella. Esta función comienza transformando la imagen a escala de grises y reduciendo el ruido para mejorar la precisión de la estimación. A continuación, se utiliza la función de openCV findContours para obtener los contornos detectados en el fotograma y pasárselos a la función cv2.approxPolyDP, que devuelve un polígono aproximado con ese contorno. Además del resultado de esta función, también calculamos el área del contorno detectado mediante contourArea para identificar la figura, ya que la función anterior no tiene una alta precisión para determinadas formas.



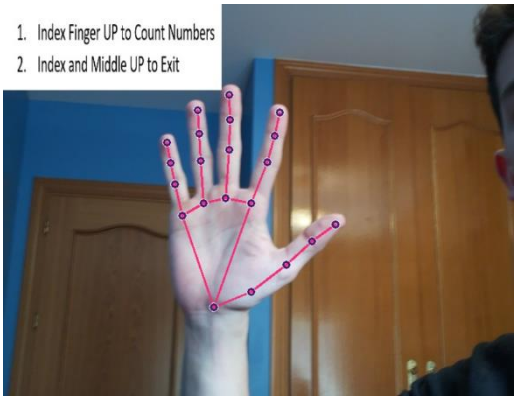


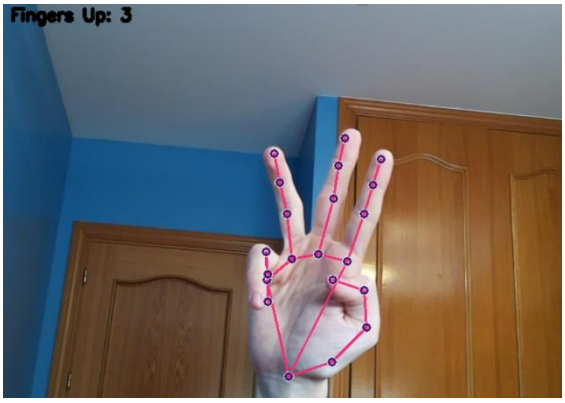


Sin embargo, es posible que la función tenga un error de vez en cuando, por ejemplo, al introducir o al sacar la figura del fotograma. Por este motivo, hemos creado una lista donde almacenamos las últimas figuras detectadas y seleccionamos la que más se repita, para añadir robustez al algoritmo a costa de perder algo de velocidad. Se puede entender como el transitorio de un sistema de control, siendo la referencia la nueva figura introducida. Así, en todo momento tenemos una respuesta robusta de qué figura se está identificando. De esta forma, simplemente tenemos que controlar que las figuras detectadas sigan el orden establecido en el patrón. Un problema que hemos tenido con esta parte ha sido que, al llegar al laboratorio, al ser el techo de cuadrados, el algoritmo no detectaba None cuando no había ninguna figura, como en los ensayos en casa. Por este motivo, hemos improvisado una presentación en la que vamos pasando diapositivas con figuras y conseguimos que la detección sea mucho más precisa, al ser las condiciones de testeado fácilmente repetibles.

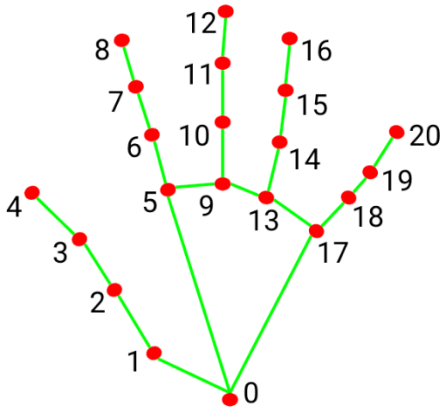
Una vez que hemos introducido el patrón correcto, se despliega el menú del *tracker* de manos. Así, si levantamos el índice hacia arriba entraremos en el modo de contar cuántos dedos están levantados. Si levantamos el índice y el corazón hacia a arriba, haciendo el símbolo de la V, terminaremos la ejecución del

programa. Para esta parte hemos empleado la librería mediapipe (MediaPipe, 2023), que utiliza *DeepLearning* para detectar la posición de los dedos en una imagen. Tiene la opción de incluir un gran número de manos en la detección, pero nosotros nos hemos limitado a una por motivos de rendimiento y porque para la aplicación tampoco hacían falta más. Esta librería nos da acceso a las coordenadas de 21 puntos en la mano, que podemos emplear para identificar qué gesto se está haciendo. En los cuatro de dedos de arriba comprobamos que la punta esté por encima del nudillo, mientras que para el pulgar hacemos un cálculo que relaciona la distancia de la punta a la mano con la diferencia en vertical de la punta con el nudillo. Este cambio ha tenido que hacerse ya que el movimiento del pulgar es muy diferente al del resto de dedos.





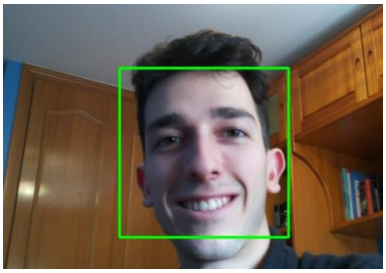
En este fichero las partes que no hemos implementado nosotros han sido las funciones de leer contornos y calcular áreas, además de la función de *mediapipe* que nos devuelve las coordenadas de los dedos como se muestra en la imagen. La información que recibimos de estas funciones es la base de la lógica que hemos desarrollado para que funcione correctamente, como la detección de una figura en particular, la lógica de los dedos comparando las coordenadas de las puntas con los nudillos o el extra de robustez utilizando una lista auxiliar.

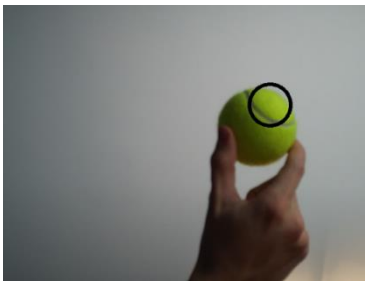


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

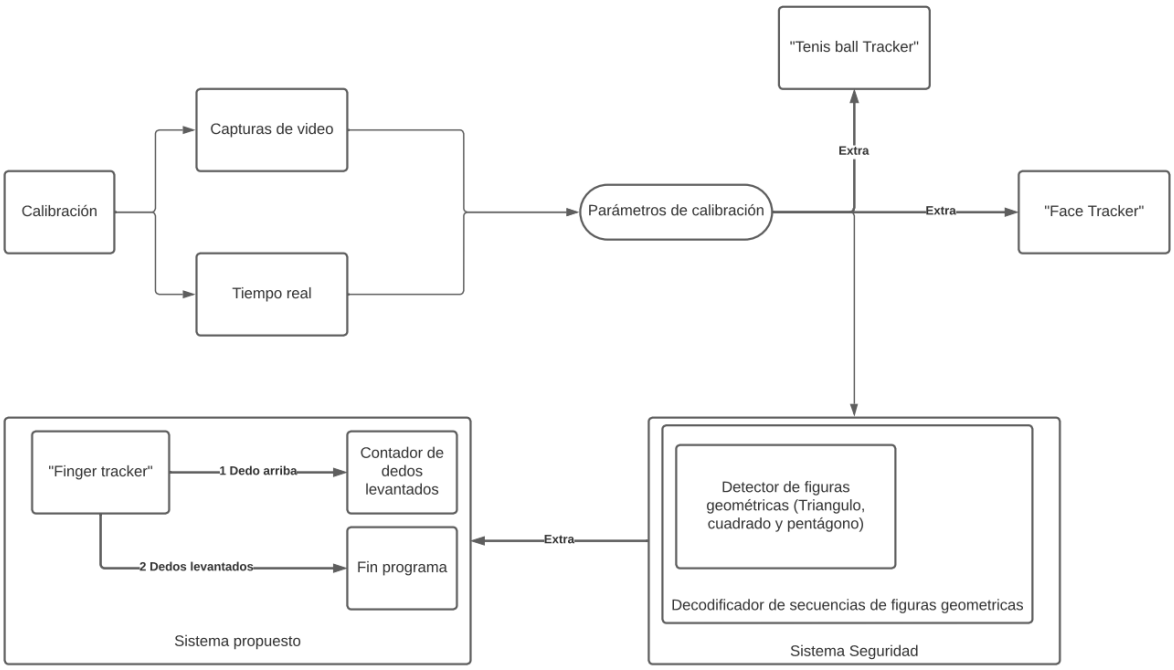
Por otro lado, el tracker de caras emplea la función de OpenCV CascadeClassifier (OpenCV, s.f.) para identificar las coordenadas de las caras que aparecen en un fotograma. Una vez que obtenemos las coordenadas, simplemente hay que dibujar un rectángulo para visualizarlas en pantalla, que es la parte que hemos implementado nosotros.

Finalmente, el *tracker* de la pelota de tenis utiliza una máscara de los tonos verdes de la imagen para identificarla. Accediendo al momento central de la imagen, es capaz de obtener el centroide de la pelota y dibujar un círculo alrededor de él. El principal problema con este algoritmo es que los tonos de verde dependen de la luz del ambiente y es posible, como ha sucedido en algunos ensayos, que por las condiciones de luz la pelota no sea detectada.





El esquema de módulos de software que hemos seguido en el proyecto es el siguiente:



3. Futuros Desarrollos

Al ser un proyecto sencillo dentro del amplio abanico de posibilidades de la Visión por Ordenador, son muchos los futuros desarrollos que se pueden aplicar sobre este proyecto.

- Calibración real de la cámara mediante los parámetros recibidos al comparar las coordenadas de las esquinas del tablero obtenidas con la cámara y la posición real de estas esquinas (error cuadrático medio, parámetros intrínsecos y extrínsecos y la rotación y traslación de los vectores). Aplicándolos sobre los fotogramas de la cámara, se podría hacer una calibración más profunda con el objetivo de obtener una representación más fidedigna de la realidad.
- El algoritmo de detección de patrones se puede ampliar de muchísimas formas: añadiendo más figuras, mejorando la precisión, incluyendo un tiempo límite, etc.
- El *tracker* de manos se puede aplicar para ejecutar una determinada acción según la posición de los dedos. Por ejemplo, bajar el volumen del ordenador, pasar de canción, empezar un temporizador o tomar una foto. También puede resultar tremendamente útil para aplicaciones de realidad aumentada.
- En el *tracker* de caras se puede realizar una triangulación del rostro para un sistema de reconocimiento facial o simplemente una App de filtros.
- El *tracker* de la pelota de tenis se puede utilizar para identificar sujetos desde una cámara de seguridad, para una cámara de control del tráfico o para un detector de señales, si se le añaden configuraciones más avanzadas.

4. Bibliografía

MediaPipe. (2023, Mayo 10). *MediaPipe*. Retrieved from Solutions: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

OpenCV. (n.d.). *OpenCV*. Retrieved from Camera Calibration: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

OpenCV. (n.d.). *OpenCV*. Retrieved from Cascade Classifier: https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html

Enlace al repositorio de GitHub con los vídeos de demostración: https://github.com/joaquinmaciias/CV2-Camera-Calibration-and-Figure-Detection