PROYECTO FINAL

1. Introducción

Este proyecto simula la identificación de objetos en una cinta de fabricación, y consta de dos partes principales: la capa de seguridad, y la identificación, seguimiento y conteo de objetos. Nuestra fábrica produce tres productos distintos que son: Nutella, Mermelada de Cereza y Mantequilla de Cacahuete. El sistema propuesto realiza un conteo de la cantidad de cada producto tras reconocerlos, para que la fábrica lleve un registro activo de cuánto se produce.

2. Metodología

a. Diseño del software

Nuestro programa está dividido en cinco módulos:

- conectar_camara_fotos.py
- calibración.py
- seguridad.py
- contador.py
- Controlador.py

El primer módulo lo hemos usado para obtener las imágenes del patrón de calibración. Estas las hemos usado para después calibrar la cámara. En este script de Python, se muestra por pantalla en una ventana el directo de la cámara para poder ver el contenido de este antes de hacer una foto. Para hacer la foto es necesario pulsar la tecla *h*. La imagen se guardará en un archivo JPG con el nombre seleccionado seguido del número de la foto realizada. Este nombre hay que modificarlo según el contexto de la foto realizada.

El segundo módulo realiza la calibración de la cámara y guarda los parámetros obtenidos en el archivo "*parametros_calibracion.npz*". Para hacer la calibración, hicimos un gran número de fotos de nuestro patrón de calibración, un tablero de 4x5 círculos negros. Los centros de dichos círculos están separados por 5 centímetros. De todas esas fotos, seleccionamos 20 de ellas de tal manera que se viese el tablero de círculos desde el máximo número de perspectivas diferentes. Una vez obtenidas las fotos que usaremos para la calibración, las hemos guardado en la carpeta de "*ImagenesCirculos*" bajo el nombre de "*Imagen_circulos{n}.jpg*" siendo n el número de la imagen. Luego, para obtener los parámetros intrínsecos y los coeficientes de distorsión de la cámara, usamos las funciones de cv2 "*findCirclesGrid*" para encontrar los centros de los círculos y "calibrateCamera" para asociar la posición real de dichos centros con la teórica. La matriz de intrínsecos obtenida es la siguiente:

$$\begin{bmatrix} 576.40054054 & 0 & 236.34185308 \\ 0 & 578.20370771 & 140.39325346 \\ 0 & 0 & 1 \end{bmatrix}$$

Y los coeficientes de distorsión obtenidos se muestran a continuación:

$$[-0.175074774 \quad 2.06032668 \quad -0.00616407088 \quad -0.00375020339 \quad -6.93391208]$$

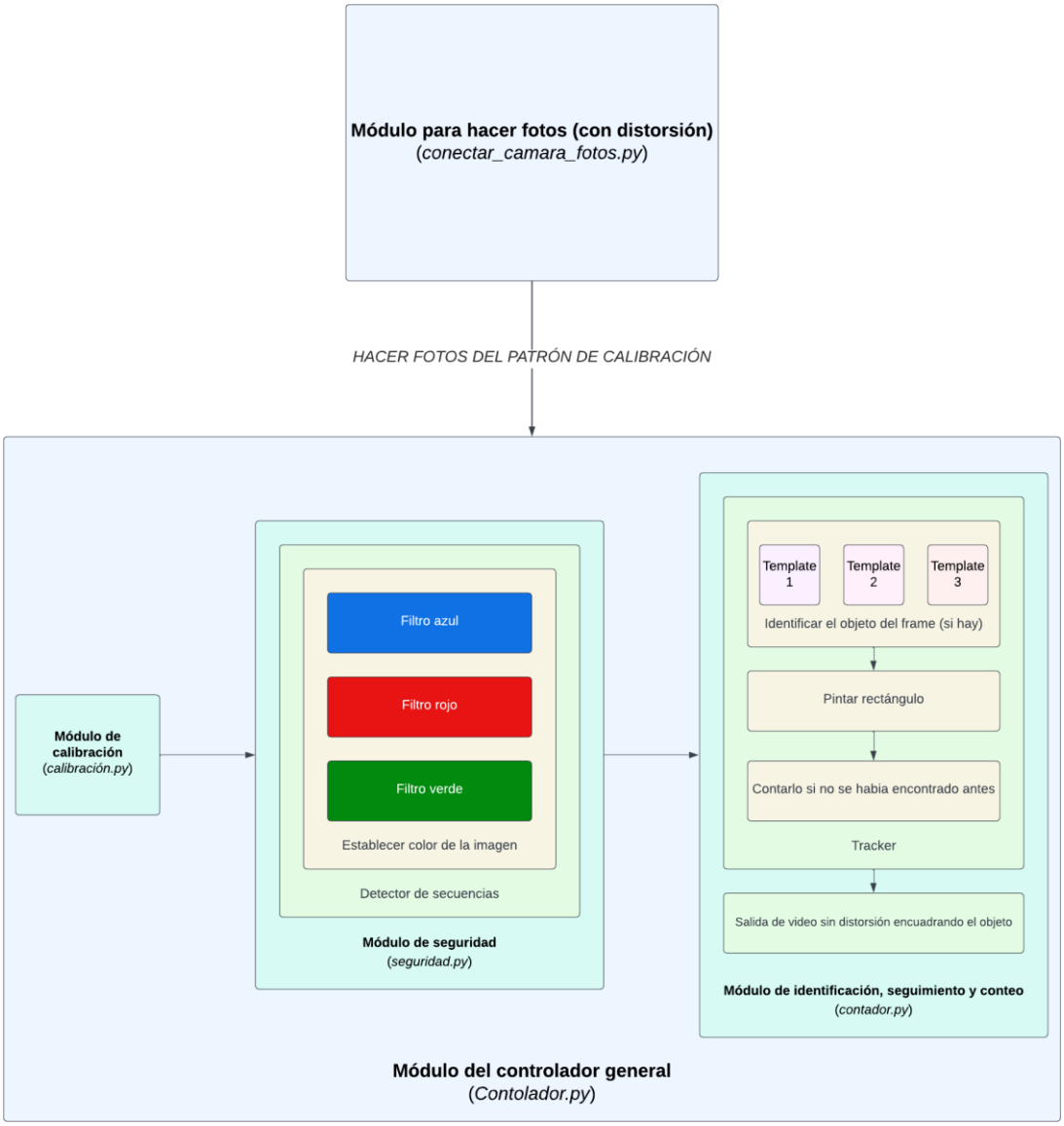
Y una vez hacemos eso, el RMS obtenido es de 0.3945338586801945.

El tercer módulo, *seguridad.py*, se encarga de la capa de seguridad y está implementada con un código de colores, el cual el que proporciona un acceso exitoso es: azul, rojo, rojo, verde, azul; en este orden. Los colores se posicionan de uno en uno delante de la cámara y se presiona la tecla SpaceBar para realizar una captura del color y verificarlo. Para esta verificación hemos creado filtros de los colores mencionados, para poder filtrar la imagen capturada por la cámara. Esta imagen la pasamos por todos los filtros, y contamos en cada uno el número de píxeles que contienen este color, para así devolver el nombre del color cuyo número de píxeles es máximo. A medida que se identifican los colores, estos se imprimen por pantalla, y cuando se cumple la longitud de la clave requerida, imprimirá: 'Combinación incorrecta. Vuelva a intentarlo' o 'Combinación correcta', dependiendo del resultado.

El cuarto archivo llamado *contador.py* es el más importante de este proyecto. Se encarga no sólo de la identificación y el seguimiento de cada producto, sino de los respectivos conteos también. En primer lugar, para la identificación de los productos, hemos cargado en el programa fotos de cada producto sacadas de internet como 'templates' y también las hemos impreso en papel. Entonces, hemos desarrollado un código el cual identifica los keypoints entre el directo de la cámara y la imagen cargada en la memoria de la RaspberryPi, compara lo que ve la cámara con los tres templates y para cada comparación crea una lista con los keypoints encontrados. Por lo tanto, la lista más larga será la que más coincidencias haya encontrado entre el objeto posicionado delante de la cámara y el template, así que lo identificará como éste último. Por si en la imagen de la cámara no hubiese ningún objeto, hemos puesto un mínimo de keypoints necesarios para hacer un match y contar y seguir dicho objeto. Cabe destacar que, para hacer el tracking, no basta con distinguir simplemente que objeto es, hay que reconocer que, si en dos frames seguidos aparece el mismo objeto, no debe contarlo dos veces ya que se trata del mismo producto. Para ello hemos incluido en el código una lista de buleanos (uno para cada objeto que queramos reconocer, en este caso 3) que usaremos como flancos. Si un objeto se ha detectado en el frame actual, el flanco asociado a dicho objeto será true, y mientras este sea verdadero, no se volverá a contabilizar el objeto. Una vez que este valor es true, se pasará a false si no encuentra ningún producto durante dos frames seguidos, para darle al tracker cierto margen por si el frame actual se ve peor o borroso y no lo reconoce, o si aparece otro objeto que no es el que estaba siguiendo el tracker. A continuación, para verificar el seguimiento de los objetos reconocidos, hemos implementado un código que produce un rectángulo alrededor del objeto y se mueve alrededor de él en la escena. Para ello, el primer paso es obtener las coordenadas de los keypoints en el frame y en el template. Con estas coordenadas podemos calcular la matriz de homografía. Esta matriz nos permite convertir coordenadas del template en coordenadas de la imagen de la cámara. Esto es necesario ya que, para pintar el rectángulo, cogemos las cuatro esquinas de la imagen del template, y con la matriz de homografía las transformamos a coordenadas del frame para obtener los vértices del rectángulo. Como resultado, se dibujará el rectángulo alrededor del objeto detectado en el frame.

Finalmente, el archivo *Controlador.py* es el responsable del orden de ejecución del proyecto. Primero de todo, ejecutará la calibración de la cámara con el módulo de *calibración.py*, seguido del programa de la capa de seguridad, *seguridad.py*, y por último el archivo de seguimiento, identificación y conteo de productos de nuestra fábrica, *contador.py*.

b. Diagrama de bloques del sistema



3. Resultados

Como se muestra en el vídeo de la demostración adjunto, el resultado del proyecto propuesto ha sido mayoritariamente exitoso. Se muestran a continuación imágenes mostrando los centros identificados durante la calibración, el correcto reconocimiento de colores, el correcto reconocimiento de un objeto y finalmente los matches identificados con una imagen de la RaspberryPi y el template original. Además de estas imágenes, el funcionamiento general del programa se puede ver en el video *VideoDemo.mkv*.

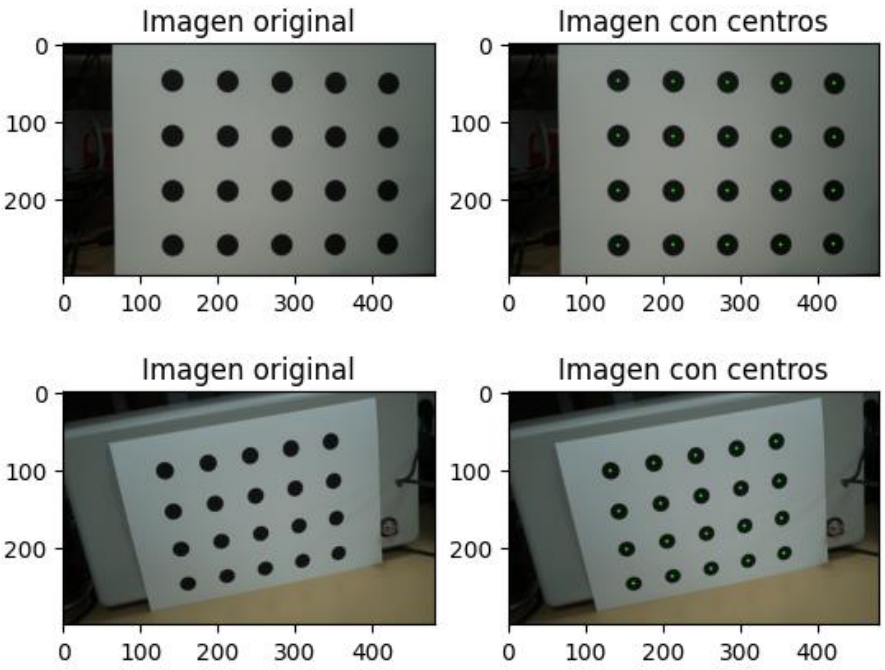


Ilustración 1: Centros identificados durante la calibración

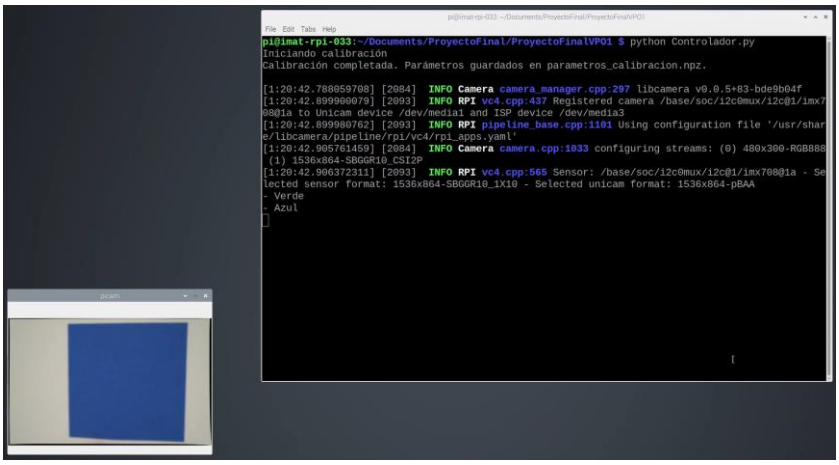


Ilustración 2: Identificación correcta de un color.

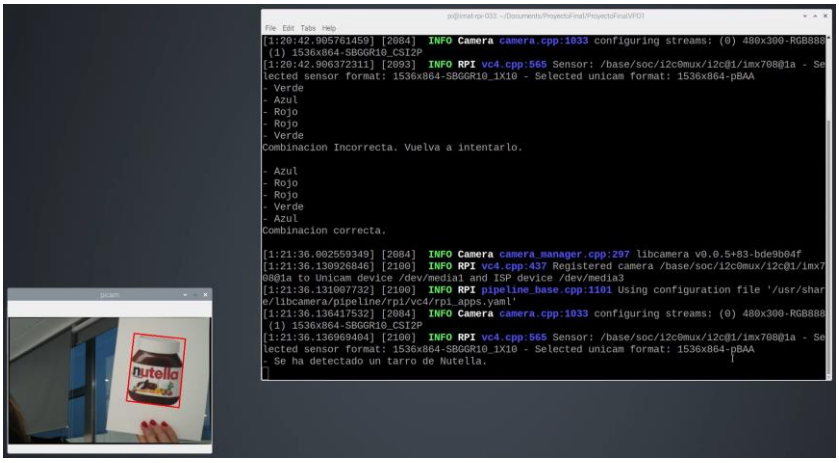


Ilustración 3: Identificación y tracking de un producto.

Ilustración 4: Matches identificados de una imagen y de un template.

4. Futuros desarrollos

La mejora principal de nuestro proyecto sería incrementar la velocidad de seguimiento, ya que, aunque se realiza correctamente, se puede observar un cierto retardo. Como solución a este problema, se podría optimizar el código para que sea lo más eficiente posible y, además, se podría conectar la RaspberryPi directamente por cable HDMI a un monitor para su eficiencia máxima, en vez de usar el control remoto a través de nuestro portátil. Otra mejora que podríamos incluir es que nuestro programa, si hay más de un objeto en la escena, detecte a ambos, y no solo el que más keypoints tenga.

5. GitHub

Se muestra a continuación el enlace al repositorio de GitHub conteniendo el proyecto desarrollado.

https://github.com/emmareysanchez/ProyectoFinalVPO1.git