

Escáner de Realidad Aumentada con OpenCV



Raúl Núñez García - UO265205
Asignatura de RAA, Universidad de Oviedo, diciembre 2020

Índice

- Idea inicial
- Primeros pasos
- Desarrollo del prototipo
- Dificultades y posibles ampliaciones
- Recursos utilizados

Idea inicial

Este proyecto surgió de la experimentación con la librería OpenCV para la realización de lo que en un principio iba a ser el proyecto individual (tratamiento de vídeo en streaming de dron con OpenCV), pero que finalmente, dadas las complicaciones y la poca viabilidad de un prototipo, fue descartado. Sin embargo, al trabajar con OpenCV y observar la comunidad que tiene detrás y la gran cantidad de recursos online para aprender a utilizar la librería, surgió idea de utilizar la cámara de dispositivos móviles como escáner, pero yendo más allá de una simple aplicación que aplique filtros de enfoque y tratamiento de colores para simular un escaneado real.

El proyecto se centra en un tratamiento enfocado a la corrección de exámenes tipo test y su visualización superpuesta al feed de vídeo existente, combinándose también con la librería de OCR [Tesseract](#) para el reconocimiento de texto y las posibilidades que surgen de ello.

Primeros pasos

En un principio la idea era realizar una aplicación web que hiciese uso de las cámaras presentes en smartphones y tablets. Para ello, la versión a utilizar de OpenCV debía ser la de Javascript. Después del trabajo con la librería hasta el punto de reconocimiento de opciones test, que ya fue de por sí más complejo de lo habitual, puesto que [OpenCV.js](#) no cuenta con las mismas opciones, documentación disponible ni rendimiento que las versiones presentes en otros lenguajes. El esquema existente en ese momento era el siguiente:

Sin embargo, aunque el flujo de datos era bastante simple y muy práctico su uso con los dispositivos móviles, los resultados fueron bastante mediocres, con fallos habituales de reconocimiento y dificultad para detectar contornos incluso en condiciones de buena iluminación.

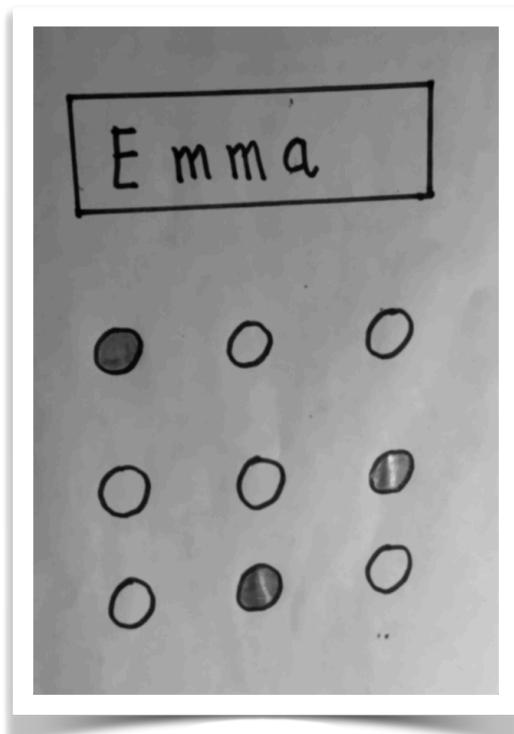
Por esa razón, la transición a la versión de Python de OpenCV se volvió necesaria (la versión de este lenguaje es, junto con la de C++, una de las más avanzadas y con mayor apoyo por parte de la comunidad). Al utilizar Python, el componente web y móvil se complicó (más detalles en la sección de Dudas y posibles ampliaciones), por lo tanto se prefirió realizar el trabajo en una versión consistente en un script a ejecutar desde equipos de escritorio con el siguiente flujo y tratamiento de datos:

Desarrollo del prototipo

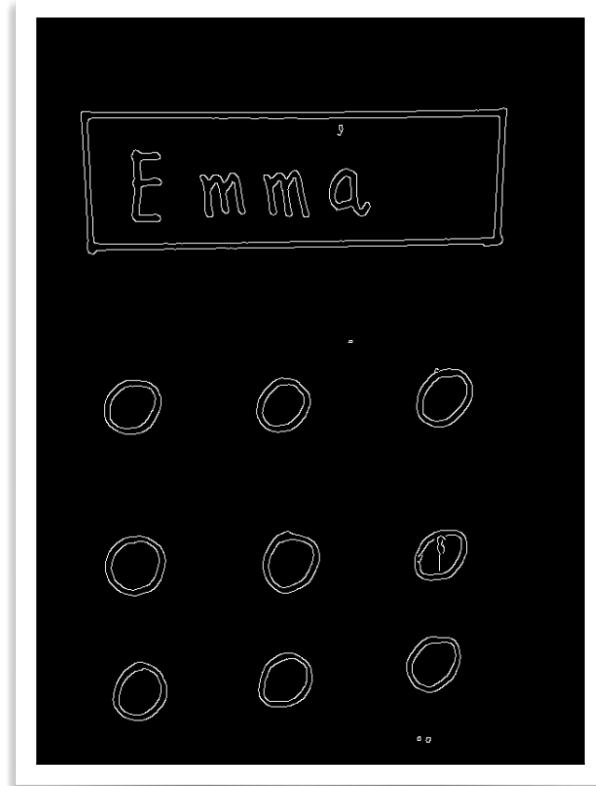
La parte principal del prototipo se trata de la distinción del examen tipo test, sus opciones y la parte del examen en la que se encuentra el nombre del examinado. Para obtener las partes a partir de cada frame proveniente del vídeo, se ha de llevar a cabo un tratamiento sobre la imagen bastante común en este tipo de aplicaciones, con el objetivo final de hacer más fácil para OpenCV la detección de contornos y su correcta delimitación.

Las acciones a realizar en este caso son las siguientes:

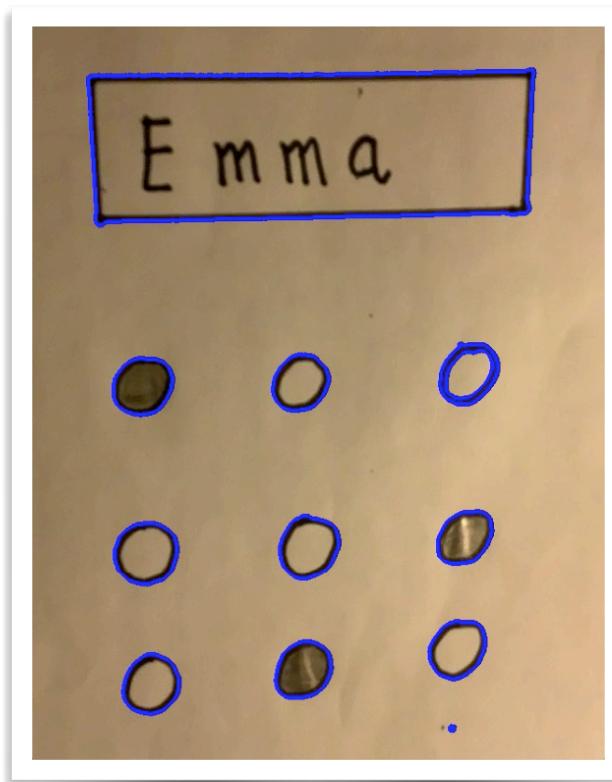
Paso a escala de grises y aplicación de blur Gaussiano (en este caso con una matriz de 5x5 como kernel) para reducir la cantidad de ruido de la imagen. Un kernel demasiado pequeño no elimina mucho ruido y a veces puede hacer difícil la detección posterior, mientras que demasiado blur puede eliminar algunos bordes, perdiendo por consiguiente los contornos que conformaban.



Detección de bordes 'Canny'



Búsqueda de contornos



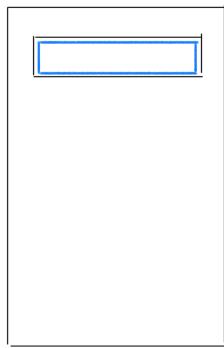
Los contornos devueltos por OpenCV suelen ser muchos, dependiendo especialmente del grosor de los trazos presentes en la imagen y las condiciones lumínicas. Para evitar trabajar con más contornos de los necesitados y aumentar la eficacia de la detección el primer paso a realizar es detectar el contorno del nombre del alumno por un lado, y el resto del examen por otro y aislarlos entre sí .

Para ello, realizamos una búsqueda de contornos con la utilidad de OpenCV y a continuación vamos iterando a través de los contornos encontrados (ordenados en función de su área) buscando uno en el que el polígono aproximado tenga 4 esquinas, el cual será el rectángulo del examen en el que el nombre del alumno está contenido.

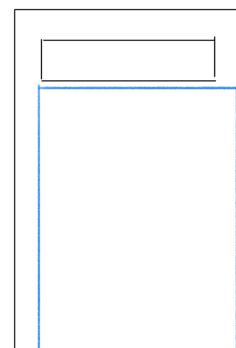
Una vez obtenido este contorno, procedemos a realizar un pequeño recorte para eliminar los bordes del contorno, ya que en el posterior OCR dichos bordes pueden afectar ligeramente el resultado final. Para realizar el recorte llevamos a cabo un slice del vector que representa la imagen.

Tras conocer las coordenadas del contorno del nombre, podemos obtener el resto del examen realizando un slice en el que partimos desde la esquina inferior izquierda del nombre hasta la esquina inferior derecha del frame completo.

Slice para obtener el nombre



Slice para obtener el resto del examen



Una vez obtenidos ambos contornos correctamente, procedemos en dos fases, el análisis del nombre y el análisis del resto del examen, así como su corrección.

Para la obtención del nombre, enviamos el recuadro del nombre a Tesseract que llevará a cabo el OCR. El reconocimiento no siempre es perfecto debido factores como las condiciones lumínicas o el propio trazo del nombre. Para eliminar caracteres que con total seguridad no formarán parte del nombre del alumnado, solo aceptamos aquellos que sean letras (haciendo uso del filtrado y la propiedad *isalpha* en Python).

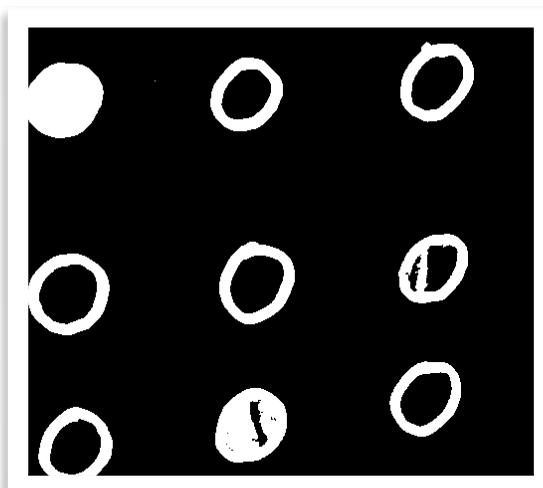
Una vez obtenido un nombre que puede pertenecer a uno de los estudiantes, procedemos a compararlo con los estudiantes obtenidos del archivo externo, y en el caso de coincidir, obtenemos los datos de dicho alumno para posteriormente integrarlos en la salida.

En el caso de que no coincida con ninguno de los estudiantes cargados, la operación de escaneado OCR se vuelve a realizar.

Por otro lado, con el contorno del resto del examen procedemos con el mismo orden que en la primera búsqueda de contornos, transformando a escala de grises, aplicando un threshold binario para mejorar la detección de bordes y llevando a cabo la detección de las burbujas tipo test en el método *find_test_circles*. En dicho método obtenemos todos los contornos del frame, y descartamos todos aquellos que no tengan unas dimensiones mínimas, así como aquellos cuyo ratio de aspecto sea diferente a 1 (con un margen de error para evitar ser demasiado estrictos). De esta manera, obtendremos finalmente solo aquellos contornos consistentes en las burbujas del examen, pues superan las dimensiones mínimas, y al ser círculos, cuentan con un ratio de aspecto dentro del rango buscado.

A continuación, si el número de burbujas encontrado es igual al número esperado para el examen cargado (cuya carga se lleva a cabo mediante el módulo *exam_retriever*), se procede a llevar a cabo la corrección.

Para ello necesitamos diferenciar aquellas opciones seleccionadas de aquellas que no lo están, aplicando en primer lugar un threshold binario para simplificar la detección, puesto que transforma todos los valores en 1 o 0 en función de su luminosidad.



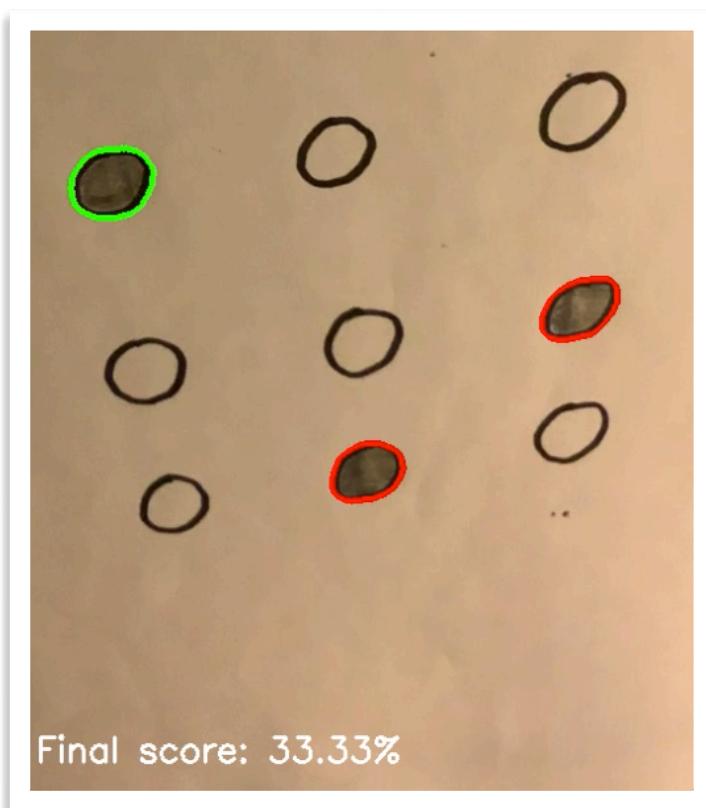
En este caso, podemos observar tras el threshold, que se han producido reflejos de la luz en dos de las opciones seleccionadas y por lo tanto no se observan tan llenas como la primera, una situación que puede ocurrir a veces dependiendo del ángulo con el que la cámara apunte al examen. No obstante, como la diferencia con las opciones no seleccionadas sigue siendo significante, utilizamos una máscara para determinarlo de manera definitiva y numérica, aplicando un “*bitwise AND*” y contando el número de unos en la máscara, que representarán la cantidad de relleno presente en la opción y permitirá distinguir las opciones seleccionadas incluso en condiciones subóptimas como la presentada aquí.

Al comparar con el resto de contornos de la misma fila, marcaremos el que mayor número de unos posee como la opción seleccionada, y en caso de que coincida con la respuesta correcta cargada del fichero externo, cambiaremos el color del contorno a verde, además de marcar la respuesta como correcta. En caso contrario, el contorno seleccionado será rojo, ya que la respuesta seleccionada no será la correcta.

Tras el dibujo de los contornos en el frame, procedemos a llamar a *show_exam_information* para poder mostrar adecuadamente toda la información recogida acerca del examen.

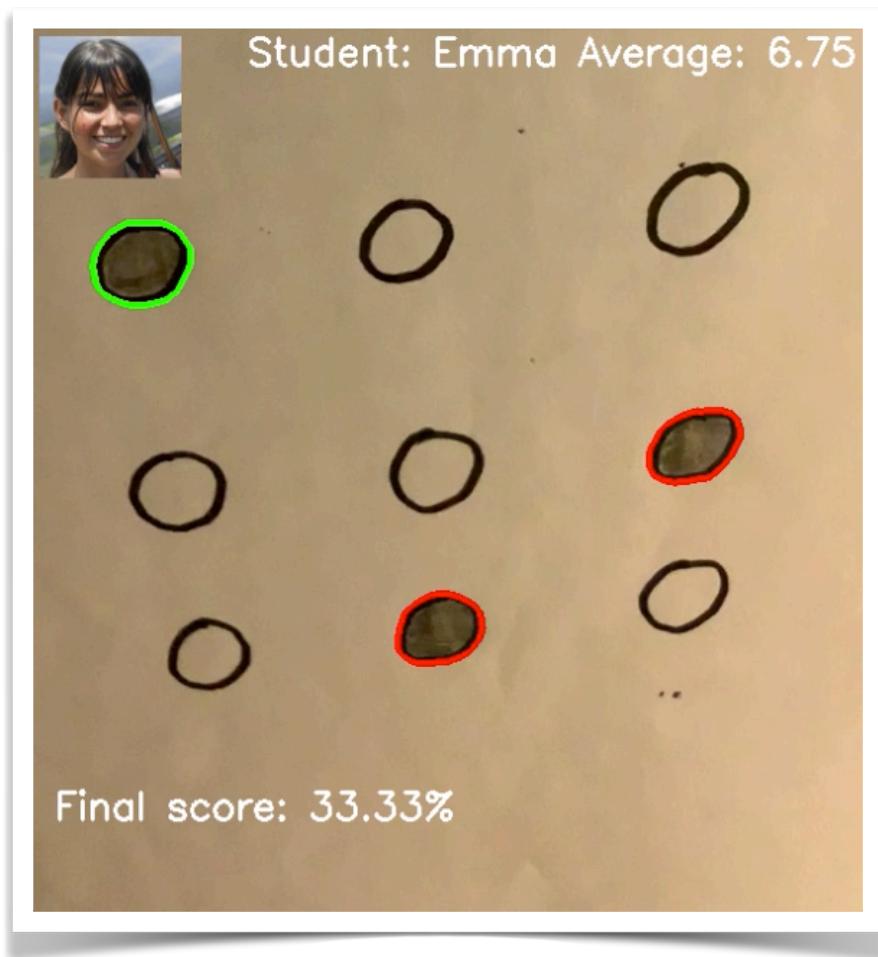
En esta última parte de la función *show_exam_information* calculamos la nota en función del número de respuestas correctas obtenido previamente, y procedemos a incluir el texto que indique la nota en el frame final. En el caso de haber obtenido una coincidencia en el nombre del alumno, lo añadiremos al frame, así como su nota media hasta el momento, y en el caso de contar también con una imagen, también será incluida en el resultado final.

Caso en el que el OCR aún no ha encontrado una coincidencia, y por lo tanto solo se muestra la nota del examen junto con su corrección



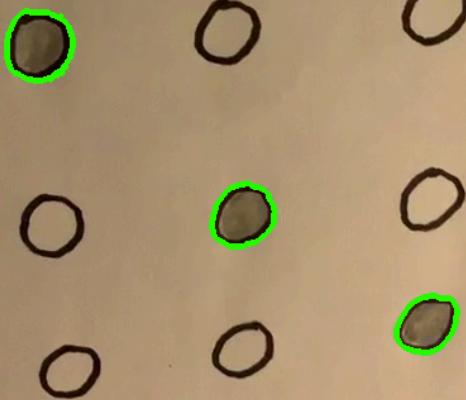
Si no ha habido una coincidencia hasta el momento con el nombre del alumno, mostraremos por pantalla únicamente el examen corregido con la nota final. Si posteriormente en un nuevo OCR encontramos una coincidencia, se llamará a *show_exam_information* una vez más para que añada la nueva información encontrada, utilizando el mismo frame que hasta el momento.

La salida final se puede comprobar en los siguientes ejemplos:





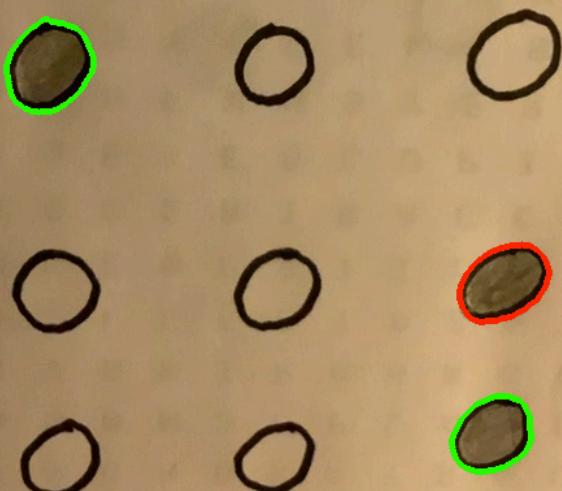
Student: Carlos Average: 8.75



Final score: 100.00%



Student: Juan Average: 5.25



Final score: 66.67%

Dificultades y posibles ampliaciones

El proyecto resultó ser de dificultad media, aunque mi inexperiencia con el tratamiento de imágenes hizo los comienzos más complicados, especialmente hasta conseguir una comprensión básica de los procedimientos a realizar. No obstante, la transición a Python hizo el trabajo más fácil, no solo por la naturaleza del lenguaje y su facilidad para la creación de un prototipo básico, sino también por la agilidad de OpenCV en Python comparado con JavaScript, y la gran diferencia de resultados.

Como posibles mejoras, la sustitución de la búsqueda de los contornos de las burbujas mediante *findContours* por la búsqueda de círculos mediante el método de gradiente Hough podría aumentar el rendimiento, aunque en los intentos iniciales de utilizar esta técnica (en la versión de JavaScript) los resultados fueron mediocres, y algunas opiniones comparando ambas técnicas remarcan que las irregularidades en los círculos (que ocurren a menudo en el relleno de las opciones de un examen) provocan muchos más fallos de detección en la opción de Hough, con lo cual esta sustitución podría ser una opción de futuro, pero no una prioridad.

Por otro lado, la portabilidad a una versión web para hacer uso del programa en dispositivos móviles sería una clara mejora, como mencionado anteriormente. Para llevar a cabo dicha transformación a un entorno web, podrían utilizarse herramientas como Heroku o Flask, que permitiesen la ejecución de un entorno Python con las librerías necesarias en un servidor web.

Recursos utilizados

Las librerías externas utilizadas en el entorno de Python (Anaconda) fueron las siguientes:

- OpenCV para el tratamiento de imágenes y la mayor parte del trabajo de detección de contornos
- Numpy para el trabajo con vectores y matrices
- Tesseract para llevar a cabo el OCR del texto del examen
- Imutils para obtener, ordenar y tratar los contornos obtenidos de una manera más fácil y ágil que la requerida por OpenCV

Otras referencias utilizadas en el proyecto:

- Fotografías de alumnos (generadas por IA):

<https://thispersondoesnotexist.com/>

- Documentación oficial de OpenCV para JavaScript:

https://docs.opencv.org/master/d5/d10/tutorial_js_root.html

- Documentación oficial de OpenCV para Python:

https://docs.opencv.org/3.1.0/d6/d00/tutorial_py_root.html

- Tutoriales de OpenCV para Python:

<https://www.tutorialkart.com/opencv/python/>

- Transformaciones convolucionales en imágenes:

<https://www.youtube.com/watch?v=8rrHTtUzyZA&list=LL&index=7&t=3s>

<https://www.pyimagesearch.com/2016/07/25/convolutions-with-opencv-and-python/>

- Escaneo de burbujas:

https://www.pyimagesearch.com/2016/10/03/bubble-sheet-multiple-choice-scanner-and-test-grader-using-omr-python-and-opencv/?_s=1yjf9rgm5g8vl4friemy

- Utilización de la librería Imutils con OpenCV:

<https://www.pyimagesearch.com/2015/02/02/just-open-sourced-personal-imutils-package-series-opencv-convenience-functions/>