

TECNOLÓGICO DE MONTERREY CAMPUS GUADALAJARA



Sistemas de visión por computadora

Profesor: Rodolfo Rubén Álvarez González e Iván Alejandro García Ramírez

Proyecto final: reconocimiento de texto manuscrito



Ismael Lizárraga González	A01630333
Rubén Barajas Curiel	A01630323
Victor Daniel Green Silva	A01229717

Lunes 24 de noviembre del 2018

INTRODUCCIÓN.

El núcleo de la detección de texto es el diseño de features que permitan distinguir texto del fondo en el que se encuentra. Un enfoque tradicional a esto ha sido el diseñar manualmente features que capturen las propiedades del texto, al mismo tiempo que con métodos basados en Deep Learning se pueda aprender y entrenar. Mas sin embargo, existen muchas cuestiones que deben tomarse en cuenta cuando se desean implementar sistemas de este tipo y los principales son:

- Imagen / ruido del sensor: el ruido del sensor de una cámara de mano suele ser mayor que el de un escáner tradicional. Además, las cámaras de bajo precio normalmente interpolan los píxeles de los sensores en bruto para producir colores reales.
- Ángulos de visión: el texto de escena natural puede tener ángulos de visión que no son paralelos al texto, lo que hace que el texto sea más difícil de reconocer.
- Difuminado: los entornos no controlados tienden a aparecer borrosos, especialmente si el usuario final utiliza un teléfono inteligente que no tiene algún tipo de estabilización.
- Condiciones de iluminación: las fotografías tomadas pueden estar cerca de la oscuridad, el flash de la cámara puede estar encendido o el sol puede estar brillando intensamente, saturando toda la imagen.
- Resolución: no todas las cámaras son iguales, se puede estar tratando con cámaras con una resolución inferior o superior a la media.
- Objetos que no son papel: la mayoría, pero no todo, el papel no es reflectivo (al menos en el contexto del papel que está intentando escanear). El texto en escenas naturales puede ser reflexivo, incluidos logotipos, signos, etc.
- Objetos no planos: considerar lo que sucede cuando se envuelve el texto alrededor de una botella: el texto de la superficie se distorsiona y se deforma. Si bien los humanos aún pueden ser capaces de "detectar" y leer fácilmente el texto, los algoritmos de Deep Learning tendrán dificultades.
- Diseño desconocido: no se puede usar ninguna información a priori para dar a los algoritmos pistas sobre dónde reside el texto.

Algunos de los mencionados problemas serán solucionados con un preprocesamiento de la imagen pero estamos a merced de cualquier situación que pudiera presentarse. En el siguiente proyecto se intentará reconocer e interpretar el texto proveniente de una imagen de entrada.

DESARROLLO.

Como la extensión del proyecto es muy grande comparado con cualquier práctica que se desarrollaron a lo largo del semestre, éste se dividió en distintas etapas cada una independiente de la otra para que cada una de ellas pudiera trabajarse en paralelo por cada uno de nosotros; una metodología que mejoró nuestro rendimiento y nos permitió trabajar eficientemente.

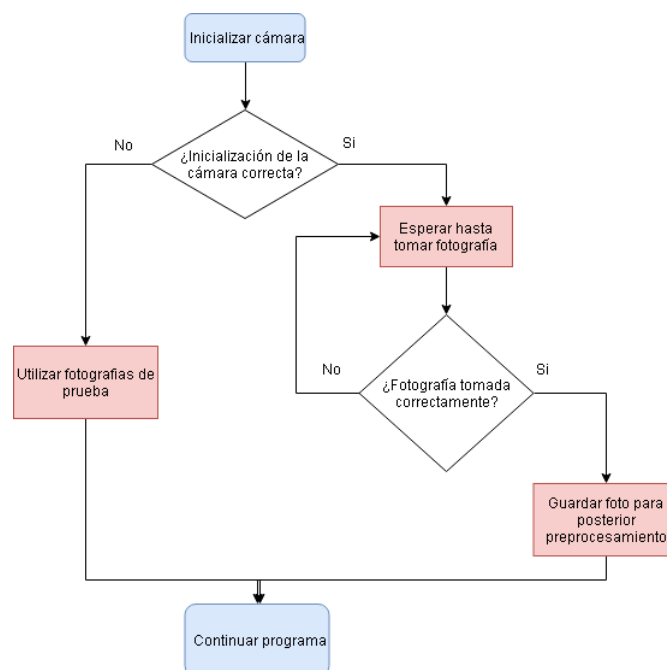
El diagrama a bloques que refleja el proyecto completo es el de a continuación:



Cabe destacar que si bien las etapas mostradas anteriormente fueron planificadas previo a arrancar el proyecto la única excepción fue la sección del procesamiento de palabras que tuvo lugar debido al gran error mostrado en la detección por parte del Tesseract.

Etapas de la cámara.

Esta etapa es posiblemente la más sencilla de realizar pero no por ello es inútil; su tarea consiste en capturar la imagen de entrada al programa mediante la cámara de la computadora y en caso de cualquier error relacionado con la misma, el programa seleccionará una imagen prueba precargada en la carpeta de compilación. La secuencia de pasos son los siguientes:



Tal como muestra el diagrama de flujo, cualquier eventualidad a causa de la cámara el programa seleccionará una imagen de entrada que en su momento fue capturada. La imagen de entrada ante tal situación es:

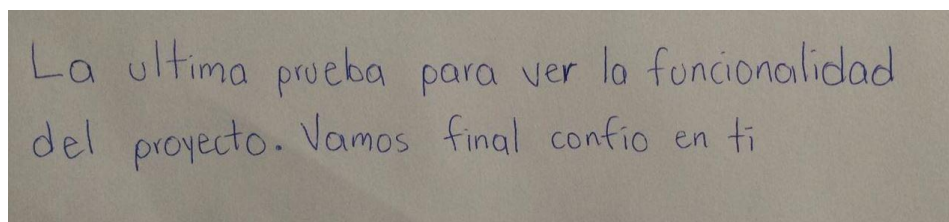
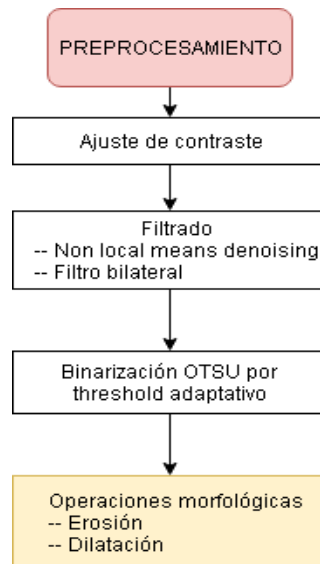


Imagen original de entrada al programa.

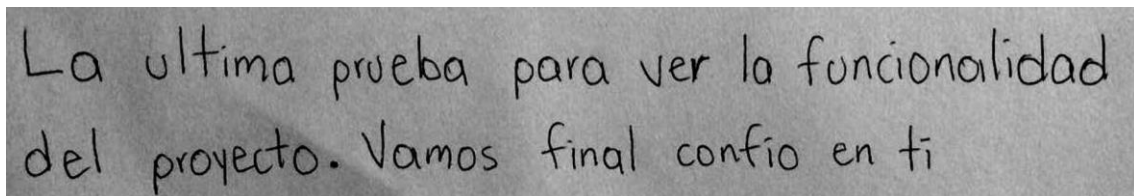
Esta imagen es la que será utilizada como referencia para mostrar los distintos cambios en la misma a lo largo del proceso.

Etapas de preprocesamiento.

Esta etapa es muy importante para el desarrollo de las posteriores; su principal función es dejar la imagen de entrada binarizada, con el mejor ruido posible y con la mejor calidad de las palabras impresas en el texto. Para ello lograr tal resultado será muy importante seguir una serie de procedimientos mostrados en la siguiente figura:



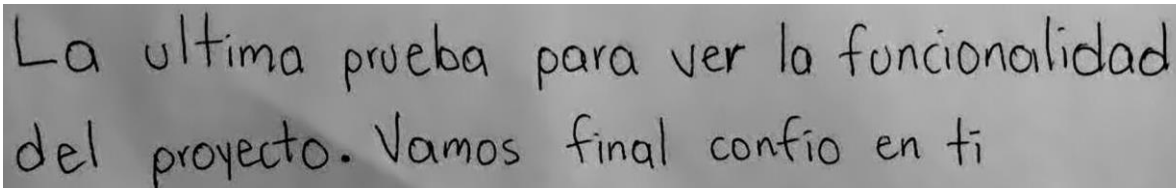
Ajustar el contraste en nuestro proyecto significó incrementar al máximo. El contraste puede definirse de manera sencilla como el cambio de luminosidad entre las zonas más oscuras o más claras de una fotografía, simulando a la vez, un mejor enfoque y claridad de imagen.



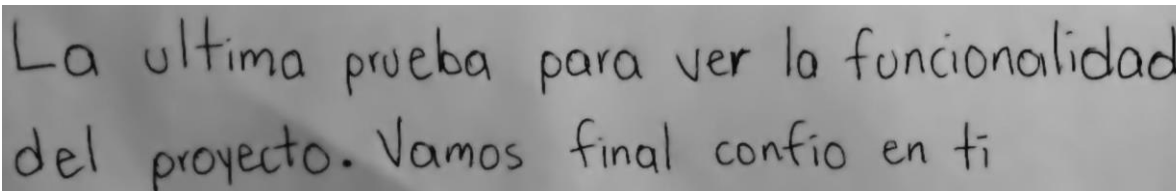
Aplicación y resultado de incrementar el contraste.

Una vez realizado el aumento del contraste, es necesario aplicar un filtrado a la imagen de entrada para eliminar el ruido blanco que se genera al tomar la fotografía original.

- El filtro “Non Local Means Denoising” es un algoritmo de procesamiento de imágenes que toma una media de todos los píxeles de la imagen y ponderada por la similitud de estos píxeles. Esto da como resultado una mayor claridad y una menor pérdida de detalle en la imagen en comparación con los algoritmos.
- El filtro bilateral es un filtro no lineal, preservador de bordes y de reducción de ruido y suavizado de imágenes. El valor de intensidad en cada pixel de la imagen es reemplazado por una media ponderada de los valores de intensidad de los píxeles cercanos. Este peso se puede basar en una distribución de Gauss. Fundamentalmente, los pesos dependen no sólo de la distancia euclidiana de píxeles, sino también en las diferencias radiométricas.

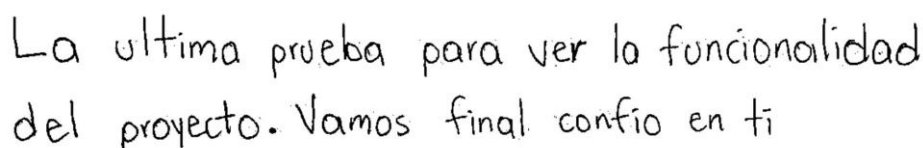


Aplicación del filtro Non Local Means Denoising a la imagen con alto contraste.



Aplicación del filtro bilateral a la imagen ya filtrada por Non Local Means Denoising.

Una vez teniendo filtrada la imagen, la binarización Otsu por threshold adaptativo será un paso más fácil de cumplir para llegar a las expectativas. Básicamente la binarización consiste en analizar pixel a pixel de la imagen; si se detecta que excede cierto nivel, se le asignará negro o blanco y de lo contrario, el color opuesto al seleccionado. Utilizar un threshold adaptativo significa que primero se barre la imagen en la búsqueda de ese punto central que brinde mejor segmentación.



Aplicación de la binarización Otsu a la imagen resultado del filtro bilateral.

Por último, la parte de las operaciones morfológicas es opcional y se le deja al usuario final como una opción. La aplicación o no de dichas operaciones dependerá directamente del tamaño de la letra y su grosor principalmente. En nuestro caso de análisis, el tamaño de la letra es el correcto definitivamente pero el grosor de la letra es muy angosto como para aplicar las operaciones y mejorar aún más la forma de las letras.

- Erosión: la idea básica de la erosión es como la erosión del suelo, es decir, erosiona los límites del objeto de primer plano a través de un kernel que se desliza por toda la imagen. Un píxel en la imagen original (1 o 0) se considerará 1 solo si todos los píxeles debajo del kernel son 1, de lo contrario se erosionará (se pondrá a cero).
- Dilatación: es justo lo contrario de la erosión. Aquí, un elemento de píxel es 1 si al menos un píxel debajo del núcleo es 1. Por lo tanto, aumenta el tamaño del objeto en primer plano. Normalmente, en casos como la eliminación de ruido, la erosión es seguida por la dilatación porque elimina el ruido blanco, pero también encoge el objeto.

Prueba para OpenCV. Espero que esto funcione.

Prueba para OpenCV. Espero que esto funcione.

Otra imagen de entrada como prueba. Hasta este punto el resultado es la binarización Otsu.

Prueba para OpenCV. Espero que esto funcione.

Prueba para OpenCV. Espero que esto funcione.

Erosión a la otra imagen de prueba después de la binarización Otsu.

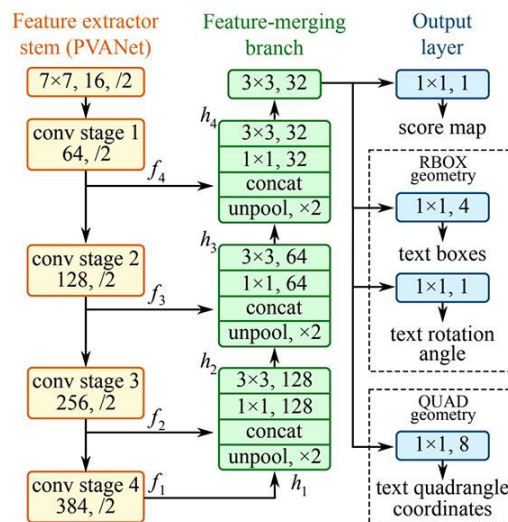
Prueba para OpenCV. Espero que esto funcione.

Prueba para OpenCV. Espero que esto funcione.

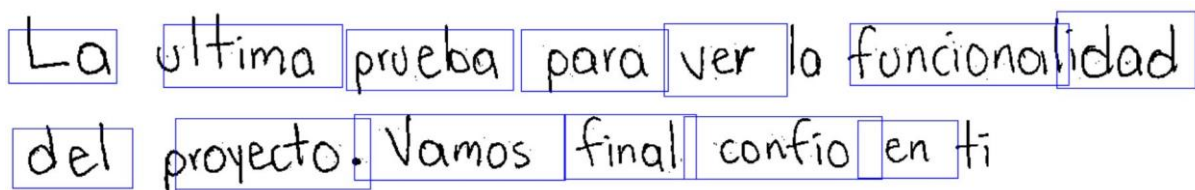
Dilatación aplicada después de erosionar la imagen anterior.

Etapas del East Detector.

El detector de texto EAST (Efficient and Accurate Scene Text) es un modelo de Deep Learning basado en una arquitectura y un patrón de entrenamiento novedosos. Es capaz de ejecutarse casi en tiempo real a 13 FPS en imágenes de 720p y obtiene una precisión de detección de texto de vanguardia. La implementación del detector de texto de EAST es bastante robusta, capaz de localizar texto incluso cuando la imagen está borrosa, reflexiva o parcialmente oculta.



La estructura de la red de detección de texto EAST Fully-Convolutional.

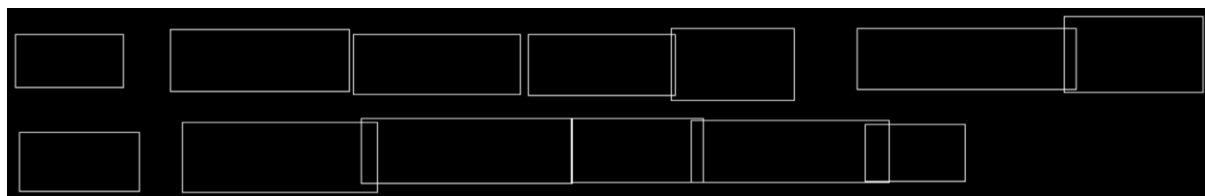


Resultado de aplicar EAST a la imagen que viene de la etapa anterior.

Etapas de procesamiento de las palabras.

Una vez detectadas las palabras mediante EAST, el Tesseract exige una imagen muy clara de la palabra que va a detectar y es por ello que se procede a cortar palabra por palabra u oraciones para facilitarle la tarea a la siguiente etapa.

Primeramente, la imagen resultada de EAST tiene encerrada en cuadrados azules la localización de la palabra lo cual es un punto de partida muy importante. Será clave aplicar un detector de color (en este caso azul) para segmentar dichos rectángulos y aplicar un detector de bordes a dicha imagen segmentada para encontrar la localización de los puntos clave sobre los cuales, la imagen será cortada.



Localización de los cuadros donde la imagen será cortada (segmentación por color azul).

Hasta este entonces las coordenadas de los puntos clave ya fueron encontradas para cada recuadro, ahora es turno de cortar la imagen original a partir de los puntos dados. Hay que destacar que si los recuadros aparecen traslapados, estos se tomarán como uno solo gracias al propio detector que toma los puntos más alejados del origen.

La última
prueba para ver
funcionalidad del
proyecto. Vamos final confio en .

Cada uno de los recortes creados.

Ya teniendo como resultado final de la etapa las imágenes cortadas, se espera que el Tesseract tenga menores dificultades al momento de encontrar el texto. Como parte importante del resultado de esta prueba se destaca el que dos palabras no fueron detectadas como texto (la y ti), además de que la palabra proyecto no fue completamente segmentada puesto que la letra p no se alcanza a visualizar completamente. Mas sin embargo, las otras palabras si fueron correctamente segmentadas y agrupadas para formar oraciones pequeñas evitando cualquier salto de línea.

Etapa del Tesseract.

Tesseract es un motor OCR(Optical Character Recognition) libre; está considerado como uno de los motores OCR libres con mayor precisión disponibles actualmente. Tesseract permite el formateo de texto de salida, la información posicional hOCR y el análisis de diseño de página. El soporte para una serie de formatos de imagen se agregó usando la biblioteca Leptónica. Tesseract puede detectar si el texto es monoespaciado o espaciado proporcionalmente.

Las versiones iniciales de Tesseract solo podían reconocer texto en inglés. Versiones posteriores a la inicial agregaron seis idiomas occidentales adicionales (francés, italiano, alemán, español, portugués brasileño, holandés). El soporte de idioma extendido incluye idiomas ideográficos (chino y japonés) y de derecha a izquierda (árabe, hebreo), así como muchos más scripts. Para el 2015 Tesseract es capaz de reconocer texto en más de 100 idiomas y tiene la posibilidad de ser entrenado para detectar otros lenguajes.

Lo único que debe tomarse en cuenta al momento de utilizar la librería en Python es instalar correctamente las librerías y mandar llamar la función que es capaz de realizar dicha traducción. En automático se detecta el lenguaje del texto en disposición y el resultado viene dado en un string que puede ser procesado posteriormente para por ejemplo guardarlo en un archivo txt o traducir el texto a voz, etc.

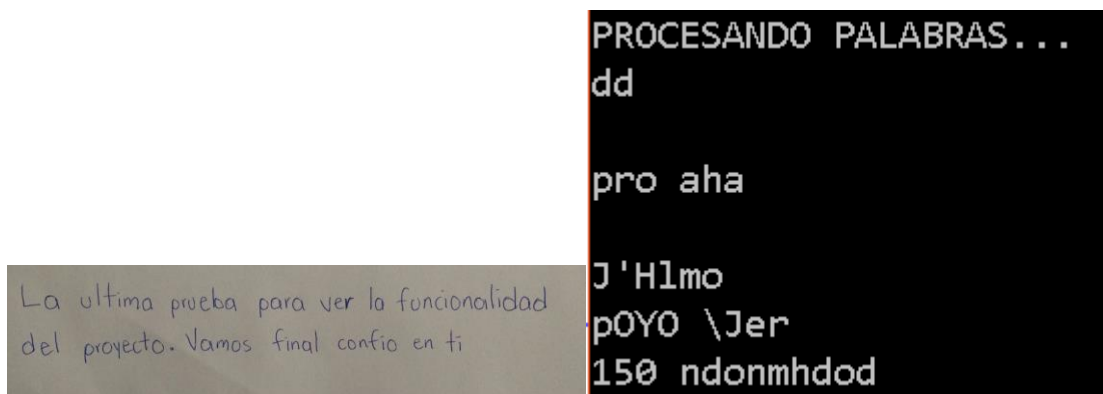


Imagen de entrada al programa. Salida en texto detectado.

Prueba para OpenCV. Espero que esto funcione.

Prueba para OpenCV. Espero que esto funcione.

PROCESANDO PALABRAS...
Prueba para OpenCV. Espero que esto funcione.
Prueba para OpenCV. Espero que esto funcione.

Imagen de entrada al programa. Salida en texto detectado.

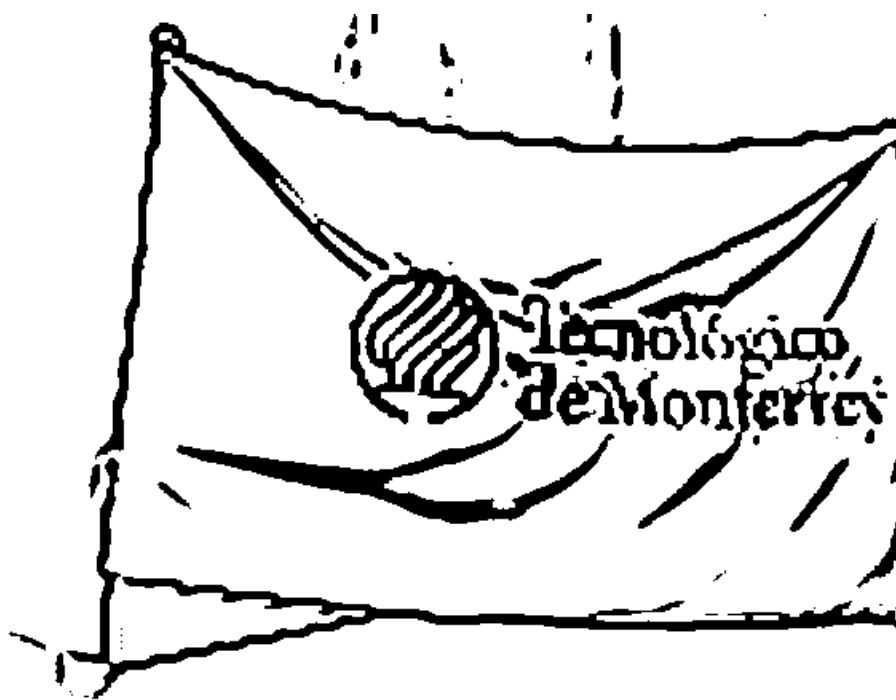
Como los resultados lo demuestran, Tesseract funciona mejor cuando hay una muy clara segmentación del texto de primer plano desde el fondo. En la práctica, puede ser extremadamente difícil garantizar este tipo de segmentaciones. Es muy importante destacar que texto escrito a mano imposibilita aún más la traducción del texto debido a la imperfección de las letras comparada con las que son escritas por la computadora.

RESULTADOS DE DEMOSTRACIÓN.

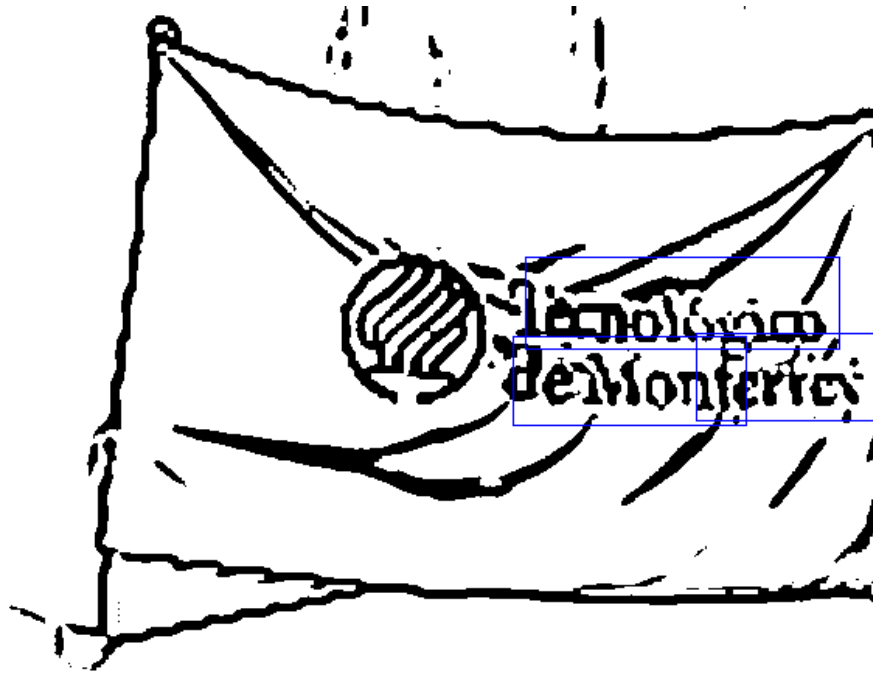
Una vez explicada la funcionalidad del proyecto en su totalidad, la prueba realizada ante los ojos del espectador en la presentación final fue la siguiente.



Fotografía tomada al momento de la presentación del proyecto.



Resultado de la binarización Otsu con previo filtrado y aumento de contraste. Las operaciones morfológicas no fueron implementadas.



Algoritmo EAST detectando el texto respectivo.



Recorte de la sección de texto encontrada por EAST.

Como es fácilmente deducible, Tesseract no es capaz de traducir el texto de la imagen anterior, así que solo dejamos como demostración final el gran rendimiento de EAST detector.

Para más detalles acerca de la instalación y ejecución del programa, ir la siguiente liga del proyecto en la que se especifican los pasos a tener en cuenta.

Link del proyecto: <https://github.com/Barajas95/OpenCV/tree/master/ProyectoFinal>

PRINCIPALES LIMITANTES Y PUNTOS DE MEJORA

Algunas de las principales limitantes que deben tenerse en cuenta a la hora de ejecutar el proyecto son:

- La letra debe ser de molde, es decir, no se aceptará manuscrita o cualquier tipo de tipografía adornada.
- El tamaño de la letra debe ser grande y con un grosor apreciable a la vista de la cámara.

- Cuanto mas parecida la letra sea a tipografía Arial o Times New Roman, mejores resultados visibles entregará el algoritmo.
- La fotografía de entrada al algoritmo debe ser plana evitando cualquier arruga o ángulo de visión que imposibilite la letra legible.
- Tamaño de espaciado entre letras y renglones suficiente, no muy pegadas las palabras o renglones.

Quizás el punto de mejora más importante del proyecto se encuentre en la etapa del Tesseract puesto que la salida del algoritmo no es capaz en muchas ocasiones de detectar el texto impreso en la imagen. alguna de las posibles vías de solución a tal problemática es crear nuestro propio clasificador de letras, palabras y oraciones con Tensoflow. Para tal tarea será necesario cambiar la versión de Python a Python3 lo cual significa migrar el proyecto completo, instalar las librerías respectivas y probar nuevamente la funcionalidad del mismo.

CONCLUSIONES.

Rubén Barajas Curiel: una de las etapas que más dificultad nos dio al momento de completar el proyecto fue la etapa de Tesseract. Mediante prueba y error nos dimos cuenta de que si la imagen de entrada al Tesseract le llegaba directamente de la binarización Otsu, encontrar palabras en el texto completo sería una tarea imposible. Es por ello que decidimos primero implementar un detector de texto y posteriormente cortar dichas palabras detectadas y procesarlas por separado. Dicha tarea resultó funcionar al momento de la detección de palabras aunque la detección en muchas ocasiones no fue la correcta, ya existía la posibilidad de traducir las palabras. Quizás el mayor error que cometimos fue el no diseñar nuestro propio traductor de imagen a texto mediante un entrenador con tensores; de haberlo implementado nuestras posibilidades pudieran haberse incrementado pero es algo que no conoceremos, al menos, no por ahora. Yo destaco la gran labor desarrollada en el preprocesamiento de la imagen (contraste, filtrado y binarización) y el detector de texto EAST porque los resultados lo revelan, la salida de ambas etapas es muy buena teniendo en consideración la imagen de entrada al sistema.

Víctor Daniel Green Silva: este proyecto final representó bastantes complicaciones y retos, principalmente a la hora de traducir el texto con la herramienta tesseract, el texto era traducido incorrectamente la mayoría del tiempo, sin embargo con cierto tipo de letra y texto en inglés la traducción podía ser bastante precisa. Una posible solución para este error era el hacer un traductor propio desde cero. Algo que pudo haber mejorado de forma notoria el proyecto fue hacer entrenamientos, estos entrenamientos con distintos escenarios/imágenes nos hubiera proporcionado un avance valioso y hubiera hecho el proyecto funcionara de forma más precisa, sin embargo los entrenamientos duraban alrededor de 18 horas con una computadora de alto poder y por esto mismo se nos complicó este entrenamiento mencionado, del lado positivo el detector EAST que utilizamos funciona de una manera perfecta, como se ve en los resultados, funciona más del 90% del tiempo a la hora de detección, y funcionó con la bandera del Tec en la presentación final lo cual demuestra que el algoritmo es sumamente complejo.

Ismael Lizárraga González: al implementar un detector de texto como el que realizamos nos encontramos con dos retos principales: identificar el texto e interpretar este texto ya en términos de palabras. Para facilitar la ‘traducción’ de imagen a palabras, lo que se hizo fue identificar segmentos de texto individualmente y procesarlos uno por uno, en lugar de intentar procesar todo el texto de la

imagen en una pasada. Esto nos permitió tener una mejor detección y traducción del texto en este proyecto. A la hora de aplicar el contenido de la clase en este proyecto, nos sorprendieron las grandes diferencias que el procesamiento de la imagen puede tener en los resultados finales, por ejemplo, con la binarización de Otsu no se podían identificar palabras por lo que se decidió ajustar el contraste y realizar filtrado antes de binarizar la imagen. El apoyarnos en el algoritmo EAST nos permitió ver un algoritmo entrenado a través de redes neuronales y los resultados que tiene, lo cual nos permitió lograr buenos resultados de detección de texto en este proyecto a pesar de las dificultades que se tuvieron con tesseract.

REFERENCIAS.

- Argman (2016). A tensorflow implementation of EAST text detector [GitHub]. Recuperado de <https://github.com/argman/EAST>
- PyImageSearch, Deep Learning. (2018). OpenCV Text Detection (EAST text detector). Recuperado de <https://www.pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/>
- PyImageSearch, Deep Learning. (2017). Using Tesseract OCR with Python. Recuperado de <https://www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/>
- Towards Data Science, Computer Vision. (2015). Build a Handwritten Text Recognition System using TensorFlow. Recuperado de <https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5>
- Weil, S. (2018). Tesseract at UB Mannheim [GitHub]. Recuperado de <https://github.com/UB-Mannheim/tesseract/wiki>
- Zhou X., Yao C., Wen H., Wang Y., Zhou S., He W., y Liang J. (2015). EAST: An Efficient and Accurate Scene Text Detector [PDF]. Recuperado de http://openaccess.thecvf.com/content_cvpr_2017/papers/Zhou_EAST_An_Efficient_CVPR_2017_paper.pdf