



Tecnológico de Monterrey

Instituto Tecnológico y de Estudios Superiores de Monterrey

Proyecto final

Reporte

Periodo Semestre Febrero - Junio 2023

Visión para Robots

Profesor:

Dr. César Torres Huitzil

Integrantes:

Rodrigo Emmanuel Oliveros Vazquez | A01329696

Daniel Sebastián de la Torre Chapell | A01733400

Sergio Ivan Villegas Arenas | A01625055

Fecha de entrega | 1 de junio de 2023

Índice

1. Introducción
 - 1.1. Objetivo
 - 1.2. Motivación
 - 1.3. Diseño, Implementación y validación general
2. Metodología
3. Análisis de resultados experimentales
 - 3.1. Software utilizado
 - 3.2. Hardware utilizado
 - 3.3. Resultados
4. Conclusiones
5. Referencias
6. Apéndice

1. Introducción

1.1 Objetivo

Desarrollo de un programa que sea capaz de facilitar el acceso a la información capturada por una imagen, con el propósito de apoyar a personas con ceguera a desenvolverse con mayor facilidad.

El programa recibirá una imagen como entrada y se encargará de analizarla para obtener una descripción en tiempo real de la misma, además, si en la imagen se vislumbra texto este será interpretado igualmente.

Tanto la descripción general de la imagen como, en caso de requerirse, la interpretación del texto se guardarán en archivos .txt, posteriormente, a partir de estos se generará un archivo .mp3 para cada uno, el cual será reproducido para que la persona utilizando el programa pueda darse una idea del contenido de la imagen capturada a través de audio.

1.2 Motivación

En esta propuesta de proyecto nosotros abordaremos un problema social que afecta a nuestro país que es la deficiencia visual y ceguera, ya que en

México la discapacidad visual ocupa el segundo lugar por frecuencia de las discapacidades donde se calcula que existen 13 millones de personas con debilidad visual y cerca de medio millón de personas con ceguera, además de esto cabe mencionar que México se encuentra entre los primeros 20 países con mayor número de personas afectadas por las discapacidades antes mencionadas.

La importancia del sistema Braille está dado por facilitar el acceso a la lectura a personas ciegas y/o con discapacidad visual. Según la Comisión Nacional de los Derechos Humanos (CNDH), “siguen existiendo omisiones como la inexistencia de prescripciones y consentimientos informados en Braille que fomentan la discriminación, vulnerabilidad familiar, social y laboral”.

1.3 Diseño, Implementación y validación general

Nuestro programa fue diseñado originalmente para ser implementado en el notebook de Google, Google Colab. Sin embargo por el uso de la cámara se decidió implementar la ejecución de manera local, ya que en Colab el procesamiento de la imagen se buscaba llevar a cabo antes de que la imagen terminara de ser obtenida.

El código del proyecto se dividió en tres partes principales; una parte enfocada en el uso de la cámara; otra en el reconocimiento de texto en imágenes y otra enfocada en obtener la descripción de una imagen.

2. Metodología

Como mencionamos, la creación del código del proyecto se dividió en tres partes principales; una parte estaría enfocada en poder usar la cámara, que en nuestro caso sería la webcam de nuestra computadora; otra parte enfocada en el reconocimiento de texto en imágenes para después convertirlos a braille y la otra en obtener la descripción de una imagen y convertir dicha descripción en audio.

En nuestro código se utilizó la librería cv2 de Opencv; para poder acceder a la cámara, realizar algunas operaciones de filtros sobre la imagen obtenida y mostrar resultados. También se utilizó la librería Numpy para poder usar arrays que permiten representar colecciones de datos de un mismo tipo en varias dimensiones.

Se decidió empezar programando la parte de reconocimiento de texto en imágenes, en el cual se buscará que se extraiga el texto de una imagen, se convierta el texto en un archivo de audio para posteriormente traducir el texto a braille y escribir el texto en braille en otro archivo de texto.

Para esto utilizamos el módulo de python Easy OCR, el cual permite extraer el texto que se encuentra en una imagen; la librería *pybraille*, la cual se encarga de convertir texto a braille; y la librería *gTTS*, ambas de python; la cual nos permite leer archivos de texto y generar un archivo de audio tipo MP3/WAV.

Esta parte se implementó en Google Colab y se probaron diferentes carteles con indicaciones, ya que fue uno de los enfoques iniciales del proyecto; obteniendo una correcta obtención de texto, traducción de braille y archivo de audio. Para corroborar la traducción de braille se utilizó la pag web <https://www.brailletranslator.org/es.html>

Para esta parte primero creamos un lector de texto en español con *easyocr.Reader*; y usamos su función *readtext*; es importante que pongamos el parámetro Paragraph en True para que el texto encontrado lo agrupe en párrafos.

Readtext te da como resultado un conjunto de strings; en cada string te da dos componentes; la ubicación de un párrafo de texto y el texto encontrado en ese párrafo. Por lo que se hace un ciclo for para que el texto encontrado en cada párrafo sea escrito en un archivo de texto; dicho texto se convierte a braille usando la función *converText* de Pybraille y se escriba en otro archivo de texto; procesamos la imagen para que el texto de la imagen esté dentro de un rectángulo; y se dibuja un rectángulo arriba de dicho texto para escribir el texto que obtuvo *readtext*. Posteriormente se creó una función que lea el texto en español en el primer archivo de texto y lo convierte en un archivo de audio.

Para el ciclo *for* se utilizó la librería Re la cual maneja expresiones regulares (Patrones de coincidencia de texto descritos con una sintaxis formal); y de esta manera se hace una iteración por cada expresión regular de nuestro conjunto de strings. De igual manera con esa librería se usó la función *sub* la cual nos retorna el string de texto reemplazando escapes como \n o \r en un carácter de una sola línea;

esto se hace porque dichos escapes generan error a la hora de querer convertir en braille.

La segunda parte en programar fue el uso de la cámara; donde buscamos usar nuestra cámara web como entrada de imagen y está utilizarla en nuestro reconocedor de texto.

Para esto se usó como base el notebook obtenido en <https://colab.research.google.com/drive/1QnC7IV7oVFk5OZCm75fqbLAfD9qBy9bw?usp=sharing#scrollTo=ghUIAJzKSjFT> el cual nos permitió usar nuestra cámara web en Colab; sin embargo a la hora de querer juntarlo con la detección de texto previamente programada; esto no era posible por que se mencionaba el resultado de la detección antes de que el modelo pudiera obtener el resultado, ya que en la misma celda de colab donde se obtiene la imagen se tiene que implementar la detección de texto.

Es por esto que se decidió realizar la ejecución de manera local; usando como base el código de visión del Dr. César Torres Huitzil el cual maneja diferentes capas para la obtención de la imagen; iniciando con la percepción de la cámara obteniendo la imagen de nuestra cámara web; después pasa por una capa de comunicación para obtener un delay en la obtención de imagen usando la librería *time*, y finalmente atraviesa por una capa de procesamiento de imagen.

De esta manera nuestro código nos permite tomar un instante del video y procesarlo para obtener los dos archivos de texto y el archivo de audio; con el texto y audio del texto reconocido en la imagen y otro con el texto en braille previamente mencionado.

La última parte a programar fue la parte de descripción de imagen; para esto nos basamos en el notebook 01-computer-vision-captioning del Dr. César Torres Huitzil; el donde usa LAVIS (*Language-VISion*); librería *open-source* que utiliza *deep-learning* para el procesamiento de lenguaje y visión computacional. Utilizamos el modelo BLIP, el cual nos permite asociar procesadores de la imagen a modelos previamente entrenados, generando una descripción de la imagen.

El resultado de esta parte del código es obtener la descripción de una imagen; se traducirá la descripción; ya que esta está en inglés; usando la librería de python Google_Trans la cual implementa la API de Google Translate. Una vez obtenida la traducción se escribirá en un archivo de texto para después pasarl a audio.

Por último se juntaron los 3 códigos hechos en uno solo; esto para que el usuario usando su cámara pueda tomar un instante de la imagen para obtener la descripción de la imagen y si existe algún texto en la imagen se pueda escuchar en archivos de audio.

Para una mejor obtención de texto; se agregó un procesamiento de imagen en el cual se convertirá la imagen en blanco y negro y se aplicara un suavizado guassiano. Esta imagen procesada es la que usará solo para obtener el texto.

Para obtener la descripción de la imagen, exportamos la interfaz array de la imagen y la convertimos a RGB. Para después pasarla al modelo.

Como resultado de nuestro código; de un instante de la imagen obtendremos 3 archivos de texto; uno con la descripción de la imagen; si se encuentra un texto en la imagen tendremos dos archivos uno con el texto en español y otro en braille; y si no hay texto sólo se obtendrá un archivo con la descripción que no hay un texto en la escena. Dos archivos de audio; uno con la descripción de la imagen y de igual manera; otro donde sí hay texto tendremos un audio con el texto y si no uno que describa qué no lo hay. Y si existe texto obtendremos también la imagen con el texto encontrado en un cuadro junto con el texto obtenido impreso en la parte de arriba.

3. Análisis de resultados experimentales

3.1 Software utilizado

Para desarrollar este proyecto nos planteamos el uso de diferentes herramientas en el lenguaje de python las cuales son:

[OpenCV](#)

Es una poderosa librería de procesamiento de imágenes y visión por computadora. Su funcionamiento consiste en cargar imágenes y videos desde archivos o fuentes de tiempo real, como cámaras, este utiliza algoritmos optimizados para leer y decodificar diferentes formatos de imagen y video. Una vez que la imagen está cargada, OpenCV ofrece una amplia variedad de funciones para manipular y procesar imágenes, esto incluye operaciones básicas como cambiar el tamaño, recortar, rotar, cambiar el contraste y la saturación, entre muchos otros algoritmos de procesamiento de imágenes. Posteriormente, OpenCV proporciona algoritmos y técnicas para la detección de y seguimiento de objetos en imágenes y videos, esto incluye detección de rostros, detección de características como esquinas y bordes, detección y seguimiento de movimiento, reconocimiento de objetos, entre otras aplicaciones de visión por computadora. Esta librería está optimizada para el procesamiento de imágenes en tiempo real, lo que significa que puede procesar imágenes y video en tiempo real, a medida que se capturan o se reciben desde una fuente en tiempo real, esto se logra mediante el uso de técnicas de optimización, como el uso de algoritmos eficientes y la paralelización de tareas utilizando múltiples núcleos de CPU o incluso GPU. Además de esto, se puede integrar con otras bibliotecas y frameworks, esto permite una fácil manipulación y visualización de datos de imagen y flujo de trabajo suave con otras herramientas bibliotecas de procesamiento de datos.

Numpy

Está librería es fundamental para el procesamiento numérico y científico de datos. El objeto principal de Numpy es el arreglo multidimensional, también conocido como ndarray. Estos son estructuras de datos eficientes y optimizadas que permiten almacenar y manipular grandes conjuntos de datos de manera uniforme, internamente, los arreglos de esta librería están representados como bloques de memoria contigua y utilizan punteros y desplazamientos para acceder a los elementos. También proporciona una amplia gama de tipos de datos numéricos, como enteros, flotantes, complejos, entre otros. Puede identificar el tamaño y forma del arreglo, que determina la cantidad de elementos y su distribución en el espacio de memoria. Una característica clave de Numpy es la capacidad de realizar operaciones matemáticas y lógicas de forma vectorizada, esto significa que las

operaciones se aplican a todo el arreglo, evitando así la necesidad de bucles explícitos. Internamente está librería utiliza optimizaciones de bajo nivel, como el uso de instrucciones SIMD (Single Instruction, Multiple Data) y la delegación de cálculos a bibliotecas de álgebra lineal, para mejorar el rendimiento de estas operaciones. Además de esto, ofrece una gran variedad de métodos para acceder y manipular elementos individuales o conjuntos de elementos de un arreglo. Puede utilizar índices enteros, rebanadas y máscaras booleanas para seleccionar elementos de un arreglo, lo que permite realizar operaciones selectivas y transformaciones de datos. Otras funciones importantes de esta librería son la amplia gama de funciones matemáticas y estadísticas que operan en arreglos, como sumas, multiplicaciones, exponenciales, promedios, trigonometría, estadística, etc. Estas funciones están implementadas de manera eficiente y optimizadas para trabajar grandes conjuntos de datos numéricos.

EasyOCR

Librería de python que utiliza técnicas de aprendizaje profundo y visión por computadora para realizar reconocimiento de caracteres(OCR). Primero, realiza un preprocesamiento en la imagen de entrada para mejorar la calidad y facilitar la detección del texto. Después utiliza el modelo de detección de objetos para identificar regiones de la imagen que contienen texto (este modelo puede ser una red neuronal convolucional entrenada específicamente para la detección de texto. Una vez que se detectaron las regiones de texto, se recortan y se preparan para el siguiente paso de reconocimiento, estas imágenes recortadas se convierten en imágenes individuales que contienen una sola palabra, línea o bloque de texto. Posteriormente para realizar el reconocimiento de texto propiamente dicho, la librería utiliza un modelo de reconocimiento de texto basado en atención óptica, este modelo utiliza una red neuronal recurrente (RNN) para leer y comprender el texto presente en cada una de las regiones recortadas. Por último, una vez que se ha reconocido el texto, se realiza un post procesamiento para mejorar la precisión y la legibilidad, esto puede incluir la corrección de ortografía, la eliminación de caracteres no deseados o la aplicación de reglas gramaticales específicas.

Pybraille

Librería que nos permite traducir un texto a lenguaje braille, cabe mencionar que traduce el texto a un patrón de 6 puntos de braille grado 1, lo que significa que este patrón de puntos tiene una relación de uno a uno con las letras del alfabeto. Esta librería cuenta con algunas funciones que nos permiten la traducción a este lenguaje casi de forma automática, por ejemplo la función de convertText que recibe como parámetro un string y lo traduce directamente a lenguaje Braille. Por otro lado está la función convertFile que nos permite convertir archivos de texto completos a lenguaje Braille.

gTTS

Librería que nos permite convertir texto a un archivo de audio mp3 o wav. Está comienza su funcionamiento con el texto que se desea convertir en voz, este texto puede ser una cadena simple o incluso un archivo de texto completo, ya que la librería se encarga de preparar y procesar el texto para después proceder a convertirlo. Una vez preparado el texto gTTS envía una solicitud a la API Text-To-Speech de google para solicitar la síntesis de voz, esta incluye el texto a convertir y también puede contener opciones adicionales, por ejemplo el idioma de destino y la velocidad de habla. Posteriormente la API procesa la solicitud recibida y utiliza modelos y algoritmos de síntesis de voz avanzados para generar el audio correspondiente al texto proporcionado, estos modelos pueden ser redes neuronales u otros enfoques basados en inteligencia artificial. Después de que el audio de voz es generado en el servidor de google, gTTS descarga el archivo de audio resultante en tu sistema local (la librería utiliza una conexión a internet para obtener el archivo de audio generado y luego lo guarda en el dispositivo.). Finalmente, gTTS utiliza biblioteca de reproducción de audio de python, para reproducir el archivo descargado.

LAVIS

Librería *open-source* que utiliza *deep-learning* para el procesamiento de lenguaje y visión computacional. De esta librería utilizaremos el modelo BLIP dentro de *load_model_and_preprocess* junto con *checkpoints* afinados dentro del *dataset* MSCOCO, lo anterior nos permite generar un texto descriptivo de la imagen tomada al asociar cada modelo pre-entrenado con los procesadores asociados a la misma.

Cabe mencionar que BLIP (*Bootstrapping Language Image Pretraining*) funciona de la siguiente forma: para pre-entrenar un modelo unificado visión-lenguaje, tanto con entendimiento como capacidad de generación, BLIP introduce una mezcla multimodal de codificador-decodificador, un modelo *multi-task* que puede operar en una de tres funcionalidades: codificadores unimodales, codificadores de texto basado en imágenes, decodificadores de texto basado en imágenes.

time

Nos proporciona varias funciones relacionadas con el tiempo. Se puede utilizar la función time() para obtener la hora actual en segundos desde el 1 de enero de 1970, está utiliza funciones del sistema operativo subyacente para obtener la hora actual del reloj del sistema. Además de esto, también proporciona funciones para convertir el tiempo en diferentes formatos.

re

Librería para usar expresiones regulares que nos permitirá manejar los resultados obtenidos de la detección de texto de easyOcr. Al utilizar esta librería normalmente comienza compilando un patrón de expresión regular utilizando la función compile(), esta función toma el patrón de expresión regular como argumento y devuelve un objeto de patrón que representa dicho patrón. Ya que tiene un objeto patrón, puede utilizar diferentes métodos para buscar coincidencias en el texto, además de buscar estas coincidencias la librería también permite realizar manipulaciones en el texto utilizando expresiones regulares. Por último, utiliza algoritmos eficientes, como el algoritmo de expresiones de Thompson o el algoritmo de expresiones regulares de búsqueda hacia adelante, para procesar y evaluar los patrones de expresión regular. Cabe destacar que estos algoritmos están diseñados para realizar búsquedas y manipulaciones de texto de manera rápida y eficiente.

PIL

Python Imaging Library (PIL) es una librería gratuita que permite la edición de imágenes directamente desde Python, soporta una variedad de formatos, incluidos los más utilizados como GIF, JPEG y PNG. Al utilizar esta biblioteca se comienza

cargando una imagen desde un archivo o desde la memoria, PIL utiliza diferentes métodos para hacer la carga de las imágenes en función del formato de archivo, estos métodos analizan la estructura del archivo y extrae los datos necesarios para trabajar con la imagen. Una vez cargada la imagen, puedes aplicar una amplia gama de operaciones para manipularla. Estas operaciones incluyen cambiar el tamaño, recortar, rotar, ajustar el contraste, cambiar la saturación, aplicar filtros, agregar texto, entre otros. Posteriormente, para acceder y manipular los píxeles individuales de una imagen, PIL proporciona métodos que permiten leer y escribir valores de píxeles, esto incluye obtener el valor del píxel en una ubicación específica, modificar el valor del mismo y obtener información sobre el tamaño y color de la imagen. Por último, una vez que se realizaron las operaciones de manipulación y edición en la imagen, PIL te permite guardar la imagen resultante en diferentes formatos de archivo, este utiliza los datos modificados de la imagen y los escribe en un nuevo archivo de imagen con el formato especificado.

Google trans

Es una librería de python gratuita e ilimitada que implementa la API de Google Translate, utiliza la API Ajax de Google Translate para hacer llamadas mediante Ajax para detectar idioma y traducir textos. Una vez que la librería hace la solicitud a la API, ésta realiza el procesamiento y la traducción del texto. Este servicio de google utiliza algoritmos avanzados de procesamiento de lenguaje natural y modelos de traducción para generar la traducción del texto solicitado. Después de que se realizó la traducción, la librería recopila la respuesta de la API, que incluye el texto traducido y otros metadatos asociados. Por último, se procesan y devuelven los resultados obtenidos de la API al usuario, esto puede incluir el texto traducido, el idioma detectado automáticamente, la confiabilidad automática, entre otros.

3.2 Hardware utilizado

Primeramente se requiere que el dispositivo en el cual se corra el programa tenga acceso tanto a una cámara como a una bocina, pues de no tener el primer requisito no se podrían tomar fotos, y de faltar el segundo no se podría reproducir el audio descriptivo. En segundo lugar, se deben tener permisos para generar archivos

de texto y audio, de lo contrario las descripciones no se generarían, y por ende tampoco los archivos de audio que “leen” dichas descripciones.

También es necesario que el dispositivo que utilice el programa pueda correr Python, pues todo el código está desarrollado en este lenguaje de programación.

Por último, si bien no es un impedimento para el funcionamiento del programa, es importante notar que se recomienda que el dispositivo a utilizar contenga una GPU para obtener un mejor rendimiento, sin embargo, de no ser el caso todo el procesamiento caerá en la CPU; funcionando correctamente pero generando un mayor retraso en la velocidad de ejecución.

Los resultados presentados a continuación se llevaron a cabo en una laptop ASUS VivoBook, con un procesador Intel Core i5-8250U, que alcanza hasta 3.4 GHz, con una memoria RAM de 8GB y sistema operativo Windows 10; y en una laptop Lenovo HC7L51PN con un procesador Intel(R) Core(TM) i5-7200U. Las únicas diferencia en cuanto resultados entre computadoras fueron producto de la diferencia entre cámaras que están tienen integradas.

3.3 Resultados

Problemas encontrados durante el desarrollo y su solución.

Al desarrollar este proyecto tuvimos que enfrentarnos a distintas problemáticas en el camino, estas principalmente de software, pues al probar las distintas librerías que nos eran de utilidad para el mismo, nos dimos cuenta de que algunas tenían problemas de compatibilidad entre sí.

En un principio este proyecto estaba diseñado para ejecutarse en el entorno de google colab, pero no tuvimos éxito al combinar los algoritmos de detección de texto y descripción de una imagen con el algoritmo del funcionamiento de la cámara, por lo que optamos por probar el programa de forma local, haciendo la instalación de las librerías antes mencionadas y validando la correcta instalación de las mismas para poder ejecutar el programa adecuadamente.

Además, para la interpretación del texto generado a partir de las imágenes capturadas tuvimos que especificar una codificación basada en el estándar de

codificación utf-8, pues Windows tenía problemas para interpretar algunos caracteres.

Otro problema surgió al instalar LAVIS, pues esta instalación desinstala un paquete de opencv llamado opencv-headers, la solución fue desinstalar por completo opencv después de la instalación de LAVIS y posteriormente volver a instalarlo por completo.

Reconocedor de texto

Para obtener un buen resultado al reconocer el texto, la cámara debe ser capaz de poder enfocarlo correctamente, mostrando de manera clara la distinción entre las letras mostradas; la distancia a la que estas se encuentren de la cámara impactará directamente en el resultado obtenido. Durante las pruebas obtuvimos la relación tamaño de letra-distancia para obtener resultados correctos de la interpretación del texto, siendo esta de 1/60 cm, indicándonos que el programa es capaz de interpretar correctamente hasta un máximo de distancia de 60 veces el tamaño de la letra que se está buscando interpretar.

También es importante tener una buena iluminación, cuidando que el reflejo de la luz no se encuentre sobre las letras pues esto produce resultados incorrectos, y mantener el texto a leer preferentemente en paralelo a la cámara, pues al tener ángulos de 45° en adelante se producen resultados incorrectos, y en ángulos menores a 45° los resultados varían en precisión, por lo que no se recomienda tener ángulos.

Otro aspecto a tener en cuenta es la dificultad que se tiene al reconocer caracteres especiales como signos de puntuación, diéresis, tilde etc.

Por último, por obvias razones, la resolución de la cámara juega un papel de vital importancia.

Las pruebas pueden encontrarse en el Apéndice, en “Pruebas de interpretación de texto”

Descriptor de imagen

En cuanto al descriptor de imagen, una vez más los resultados se verán directamente afectados por la resolución de la cámara y la iluminación.

Las descripciones dadas por LAVIS suelen ser más generales que particulares, aunque pueden ser entrenadas para tener una mayor precisión en ambientes específicos, nosotros buscamos la descripción de casos completamente arbitrarios.

Dicho lo anterior, la descripción será generalizada y por este motivo como por los posibles errores de interpretación que se puedan generar, se realizan 5 descripciones de la imagen, las cuales darán una idea aproximada de lo que se está visualizando.

Las pruebas pueden encontrarse en el Apéndice, en “Pruebas de descripción de imagen”

Generación de audio

La función encargada de la generar el audio se encarga de leer el texto que las dos funciones previamente mencionadas tienen y guardan en archivos de texto generar un archivo de audio que se puede reproducir desde el ordenador, lo que permite que las personas puedan escuchar lo que los algoritmos obtuvieron como resultado de las imágenes.

Análisis de resultados

Para obtener resultados satisfactorios es necesario buscar que no exista una fuente de luz que le dé directamente a la cámara; como se puede observar en las imágenes 1.6 ; la detección de texto manda un texto ilegible a pesar de que en la no aparenta haber diferencia entre la imágenes vistas desde la cámara. Además; la descripción dada sin la fuente de luz dando a la cámara es mucho más precisa; mientras que con la fuente hay descripciones que no son correctas.

También mientras menos texto se encuentre en la imagen mejor se obtiene el texto; ya que en largas cantidades de texto; como se muestra en la figura 1.8 y 1.9; existe mucho texto no detectado correctamente en donde existe una gran cantidad de texto.

A pesar que especificamos que se detecte texto en español, también se puede detectar texto en Ingles como se muestra en la figura 1.10

Como se muestra en la imagen 1.11 ; podemos observar que la diferencia entre tener como entrada el archivo de imagen a obtenerla desde la cámara web; es que aunque se obtiene el mismo texto; en la cámara web como entrada no se extrae bien cuales son mayúsculas y cuales minúsculas; lo cual genera un audio menos entendible. En cuanto a las descripciones dadas en la imagen desde la cámara web son más acertadas mientras que en la otra las descripciones son erróneas.

Cabe destacar que en escenarios controlados se pueden obtener resultados satisfactorios con textos bien detectados, audios entendibles y descripciones que dan una idea general de la escena. Sin embargo, en escenarios más naturales como se muestran en la imagen 1.7 y 1.5 ; la detección de textos no es acertada; ya que se esperaría que se detecte correctamente el texto del cartel y detectar el texto del edificio; sin embargo las descripciones siguen siendo utilizables; mas no exactas.

En cuanto a la generación del texto en braille; se complica dado que en el archivo de texto aparece muy pequeño; sin embargo comparando con el texto generado por la pagina web <https://www.brailletranslator.org/es.html>; se puede observar en la imagen 1.13 que la mayoría de los caracteres coinciden con la página web.

4. Conclusiones

En conclusión obtuvimos un prototipo funcional para la problemática que decidimos abordar para este proyecto, no obstante es importante mencionar que este funciona bajo ciertas condiciones específicas y parámetros que obtuvimos mediante la validación del mismo. Por lo que es necesario recalcar los puntos importantes en cuanto al correcto funcionamiento del proyecto, pues depende en cuanto a hardware se refiere de una buena cámara y buenas capacidades del ordenador para obtener mejores resultados de reconocimiento de texto y menor tiempo de ejecución, de no ser así y al sumar esto a otros factores como la iluminación del lugar, en ángulo en que se tomó la foto, si el cartel o texto está muy pequeño, el tipo de letra, entre otros, puede afectar gravemente los resultados obtenidos.

Es importante mencionar que el lenguaje español cuenta con letras mayúsculas, tildes, puntos, etc. que nos llevan a las limitaciones de software, en estos casos el reconocedor de texto puede llegar a fallar al no distinguir bien estos símbolos, lo que conlleva a que el archivo de audio y el archivo en braille contengan algunos errores.

Por otro lado en cuanto al sistema de descripción es importante señalar que el texto obtenido puede ser ambiguo en ciertos casos de prueba pues el modelo que utilizamos ya está previamente entrenado en casos más generales. Ya que para obtener resultados más contundentes en los escenarios que manejamos para el proyecto deberíamos entrenar este modelo en dichos escenarios.

También es importante mencionar que el programa genera un archivo de audio que contiene los resultados de los modelos anteriores, pero es necesario reproducirlo manualmente para escuchar su contenido.

Sin embargo, el desarrollo de este prototipo nos da una buena primera aproximación en la solución del problema que escogimos, pues si bien es cierto que aún tenemos un largo camino por recorrer nos da una idea de los posibles pasos a seguir y los puntos a mejorar en el desarrollo de este sistema que tiene la finalidad de mejorar la calidad de vida de las personas que tienen ceguera o alguna discapacidad visual, pues en la actualidad consideramos que sería de mucha ayuda para mejorar su estilo de vida y hacer que la sociedad se vuelva más inclusiva.

5. Referencias

CONADIS (2018). Día Mundial del Braille. Obtenido 20 de abril de 2022. Sitio Web: <https://www.gob.mx/conadis/articulos/dia-mundial-del-braille-89348?idiom=es>

PatrickFarley. (2023, March 18). *OCR: reconocimiento óptico de caracteres - Azure Cognitive Services*. Microsoft.com. <https://learn.microsoft.com/es-es/azure/cognitive-services/computer-vision/overview-ocr>

¿Qué es una red neuronal convolucional? | 3 cosas que debe saber. (2023). Mathworks.com.

<https://la.mathworks.com/discovery/convolutional-neural-network-matlab.html>

Arana, C. (2021). *UNIVERSIDAD DEL CEMA Buenos Aires Argentina Serie DOCUMENTOS DE TRABAJO.*

<https://ucema.edu.ar/publicaciones/download/documentos/797.pdf>

What is Visual Question Answering? - Hugging Face. (2022, August 2).

Huggingface.co. <https://huggingface.co/tasks/visual-question-answering>

Robert de Luna. (2020, February 20). *A Tesseract-based Optical Character Recognition for a Text-to-Braille Code Conversion.* ResearchGate; Insight Society.

https://www.researchgate.net/publication/339822122_A_Tesseract-based_Optical_Character_Recognition_for_a_Text-to-Braille_Code_Conversion

salesforce. (2023, May 25). *GitHub - salesforce/LAVIS: LAVIS - A One-stop Library for Language-Vision Intelligence.* GitHub.

<https://github.com/salesforce/LAVIS#visual-question-answering-vqa>

OpenCV: OpenCV modules. (2023). Opencv.org. <https://docs.opencv.org/4.x/re — Regular expression operations>. (2023). Python Documentation.

<https://docs.python.org/3/library/re.html>

JaidedAI. (2023, May 25). *GitHub - JaidedAI/EasyOCR: Ready-to-use OCR with 80+ supported languages and all popular writing scripts including Latin, Chinese, Arabic, Devanagari, Cyrillic and etc.* GitHub. <https://github.com/JaidedAI/EasyOCR>

Pillow. (2023). Pillow (PIL Fork). <https://pillow.readthedocs.io/en/stable/time — Time access and conversions>. (2023). Python Documentation.

<https://docs.python.org/3/library/time.html>

googletrans. (2020, June 14). PyPI. <https://pypi.org/project/googletrans/>

AuditT. (2022, June 29). GitHub - AuditT/braille: A Python module that supports conversion between text, image, speech, and braille. GitHub.

<https://github.com/AuditT/braille>

6. Apéndice

```
[3] from pybraille import convertText  
  
[4] print(convertText("hola mundo"))  
hola mundo
```

Imagen 0.0: Prueba de la librería de pybraille.



Imagen 0.1: Prueba de la librería de EasyOCR.

Funciones desarrolladas

```
#funciones cámara
def init_camera():
    video_capture = cv2.VideoCapture(0)
    ret = video_capture.set(3, IMG_COL_RES)
    ret = video_capture.set(4, IMG_ROW_RES)
    return video_capture

def acquire_image(video_capture):
    ret, frame = video_capture.read()
    scaled_rgb_frame = cv2.resize(frame, (0,0), fx=0.25, fy=0.25)
    scaled_rgb_frame = scaled_rgb_frame[:, :, ::-1]
    return frame, scaled_rgb_frame

def show_frame(name, frame):
    cv2.imshow(name, frame)
```

Imagen 0.2: Funciones encargadas de la cámara.

```
#funcion audio
def tts(text_file, lang, name_file):
    with open(text_file, "r") as file:
        text = file.read()
    file = gTTS(text = text, lang = lang)
    filename = name_file
    file.save(filename)
```

Imagen 0.3: Función encargada del audio.

```
#funcion traductor
def translate_text(texto):
    translator = Translator(service_urls=['translate.google.com'])
    translated = translator.translate(texto, dest='es')
    return translated.text
```

Imagen 0.4: Función encargada de la traducción.

Variables globales

```

47 #Modelo para detección de textos
48 reader = easyocr.Reader(["es"])
49
50 #Modelo para descripción de imágenes
51 device = "cpu"
52 model, vis_processors, _ = load_model_and_preprocess(
53     name="blip_caption", model_type="large_coco", is_eval=True, device=device)
54
55 #tiempos para comunicación
56 lastPublication = 0.0
57 PUBLISH_TIME = 10
58
59 #iniciando percepción
60 video_capture = init_camera()

```

Imagen 0.5: Variables globales.

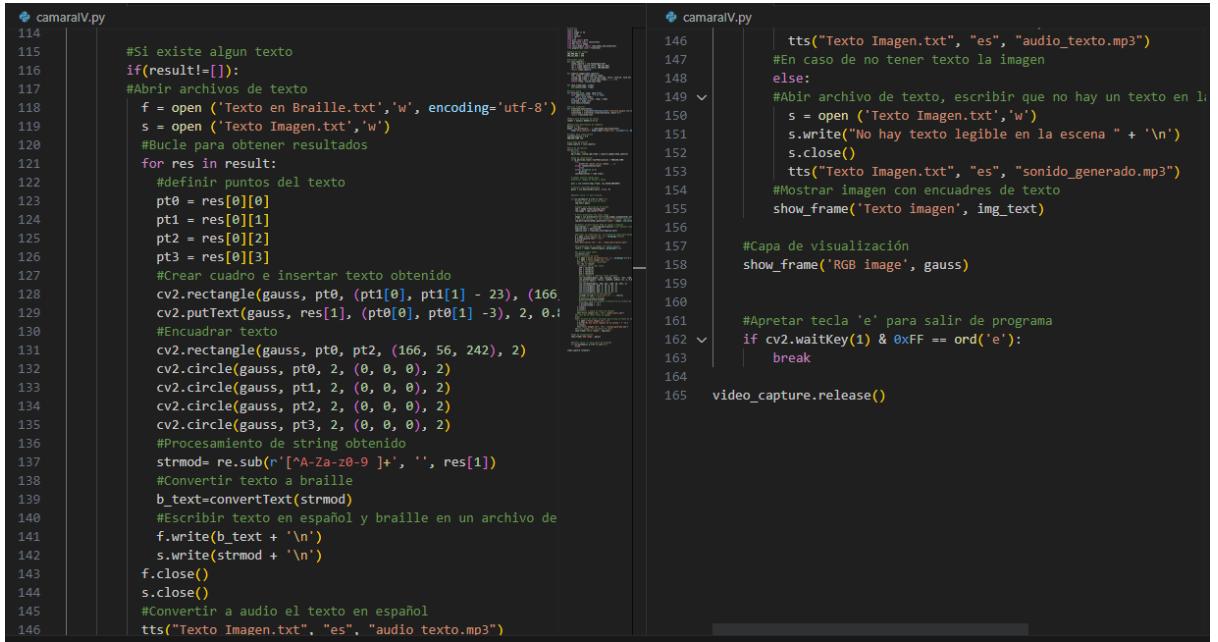
```

camaraIV.py
62 #bucle de percepción
63 while(True):
64     #Capa de senseo
65     bgr_frame, scaled_rgb_frame = acquire_image(video_capture)
66
67     #Capa de comunicación
68     if np.abs(time.time()-lastPublication) > PUBLISH_TIME:
69         try:
70             print("No remote action needed ...")
71         except KeyboardInterrupt:
72             break
73         except Exception as e:
74             print(e)
75         lastPublication = time.time()
76
77     # Añadir nuestro código aquí
78     #Convertir imagen en blanco y negro.
79
80     gris = cv2.cvtColor(bgr_frame, cv2.COLOR_BGR2GRAY)
81
82     # Aplicar suavizado Gaussiano
83     gauss = cv2.GaussianBlur(gris, (5,5), 0)
84
85
86     #Apretar tecla 'f' para funcion.
87
88     if cv2.waitKey(1) & 0xFF == ord('f'):

camaraIV.py
88     if cv2.waitKey(1) & 0xFF == ord('f'):
89         #Imagen para detección de texto
90         img_text= gauss
91
92         #Imagen para descripción de escena
93         img = Image.fromarray(img_text)
94         raw_image = img.convert("RGB")
95
96         # First preprocess the input image
97         image = vis_processors["eval"] (raw_image).unsqueeze(0).to
98         # Generate caption, for this example 3 different captions
99         img_description=model.generate({"image": image}, use_nucle
100
101         #Traducir la descripción dada en inglés a español
102         description = ' '.join(img_description) # por default viene
103         english_text = description
104         spanish_text = translate_text(english_text)
105
106         #Escribir la traducción en un archivo de texto para despi
107         d = open ('Descripción.txt','w', encoding='utf-8')
108         d.write(spanish_text + '\n')
109         d.close()
110         tts("Descripción.txt", "es", "audio_descripción.mp3")
111
112         #Procesamiento de la imagen con modelo easyocr
113         result = reader.readtext(gauss, paragraph=True)
114

```

Imagen 0.6: Ciclo while, parte 1



```

camaraIV.py
114
115     #Si existe algun texto
116     if(result!=[]):
117         #Abrir archivos de texto
118         f = open ('Texto en Braille.txt','w', encoding='utf-8')
119         s = open ('Texto Imagen.txt','w')
120         #Bucle para obtener resultados
121         for res in result:
122             #definir puntos del texto
123             pt0 = res[0][0]
124             pt1 = res[0][1]
125             pt2 = res[0][2]
126             pt3 = res[0][3]
127             #Crear cuadro e insertar texto obtenido
128             cv2.rectangle(gauss, pt0, (pt1[0], pt1[1] - 23), (166, 56, 242), 2)
129             cv2.putText(gauss, res[1], (pt0[0], pt0[1] - 3), 2, 0.1)
130             #Encuadrar texto
131             cv2.rectangle(gauss, pt0, pt2, (166, 56, 242), 2)
132             cv2.circle(gauss, pt0, 2, (0, 0, 0), 2)
133             cv2.circle(gauss, pt1, 2, (0, 0, 0), 2)
134             cv2.circle(gauss, pt2, 2, (0, 0, 0), 2)
135             cv2.circle(gauss, pt3, 2, (0, 0, 0), 2)
136             #Procesamiento de string obtenido
137             strmod= re.sub('[A-Za-zA-Zñ ]+', ' ', res[1])
138             #Convertir texto a braille
139             b_text=convertText(strmod)
140             #Escribir texto en español y braille en un archivo de
141             f.write(b_text + '\n')
142             s.write(strmod + '\n')
143             f.close()
144             s.close()
145             #Convertir a audio el texto en español
146             tts("Texto Imagen.txt", "es", "audio_texto.mp3")

```

```

camaraIV.py
146     tts("Texto Imagen.txt", "es", "audio_texto.mp3")
147     #En caso de no tener texto la imagen
148     else:
149         #Abrir archivo de texto, escribir que no hay un texto en la
150         s = open ('Texto Imagen.txt','w')
151         s.write("No hay texto legible en la escena " + '\n')
152         s.close()
153         tts("Texto Imagen.txt", "es", "sonido_generado.mp3")
154         #Mostrar imagen con encuadres de texto
155         show_frame('TextoImagen', img_text)
156
157         #Capa de visualización
158         show_frame('RGB image', gauss)
159
160
161         #Apretar tecla 'e' para salir de programa
162         if cv2.waitKey(1) & 0xFF == ord('e'):
163             break
164
165         video_capture.release()

```

Imagen 0.7: Ciclo while, parte 2

Pruebas de interpretación de texto (VGA Camera)

Los resultados se dividen en correcto e incorrecto



Imagen 0.8: Ángulo 0°, resultado: correcto

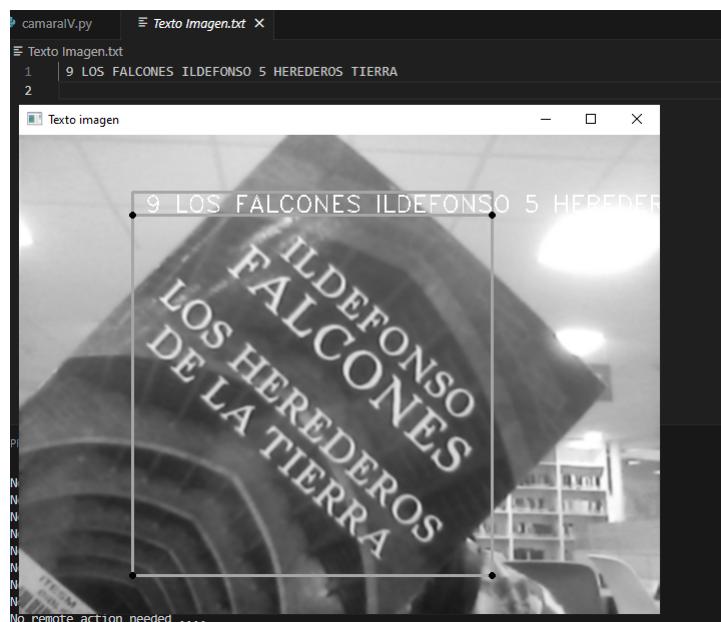


Imagen 0.9: Ángulo horizontal 45°, resultado: incorrecto

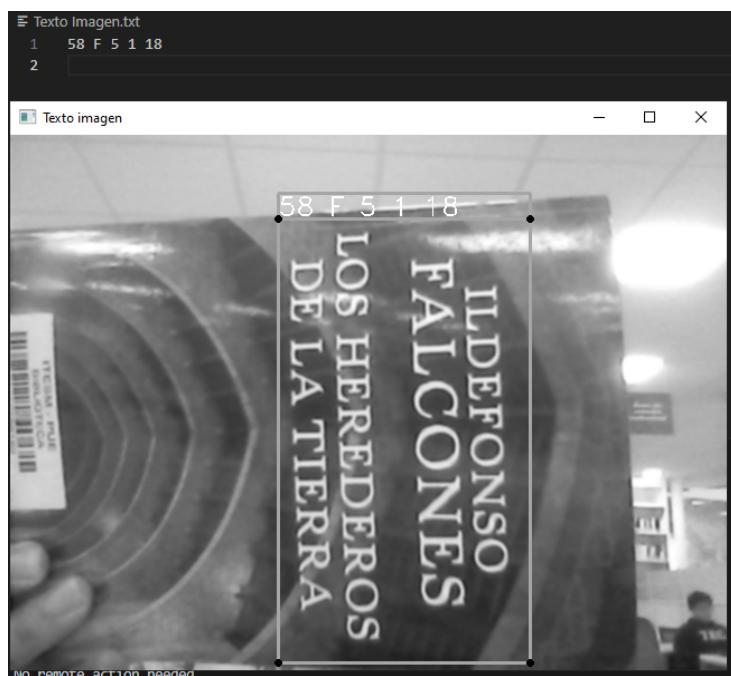


Imagen 0.10: Ángulo horizontal 90°, resultado: incorrecto

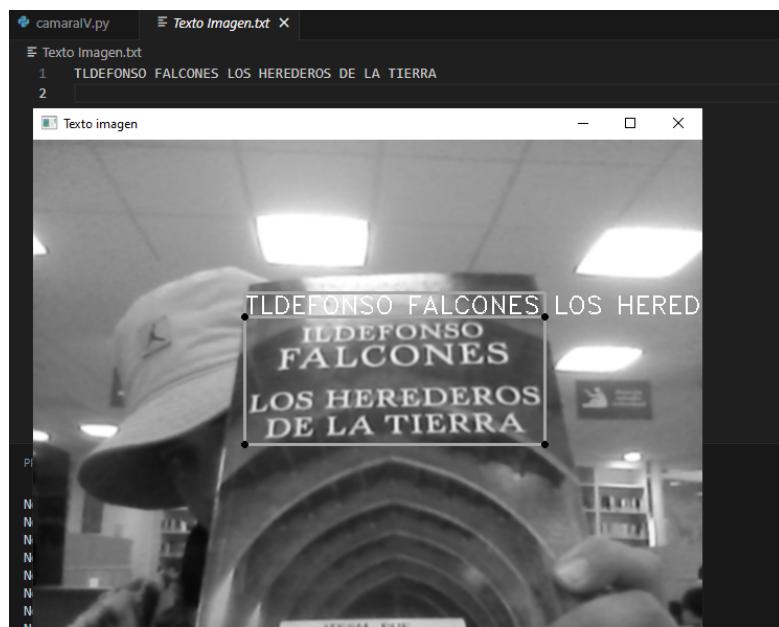


Imagen 0.11: Ángulo horizontal 20°, resultado: correcto

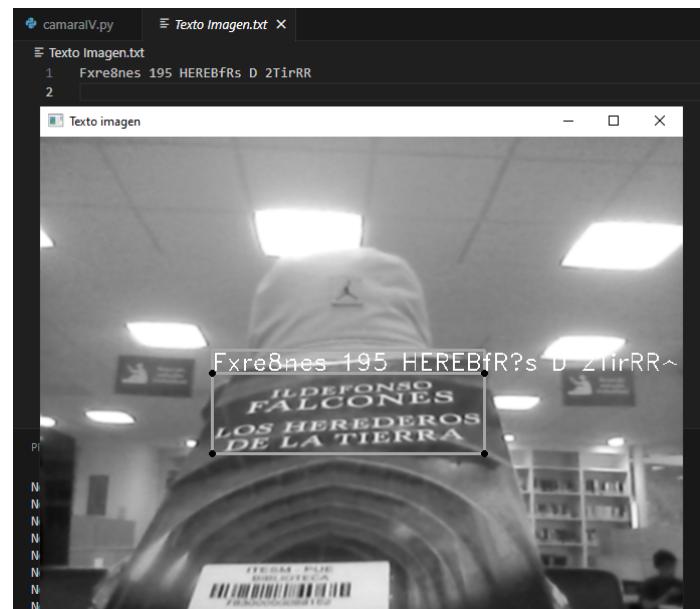


Imagen 0.12: Ángulo profundidad 45°, resultado: incorrecto

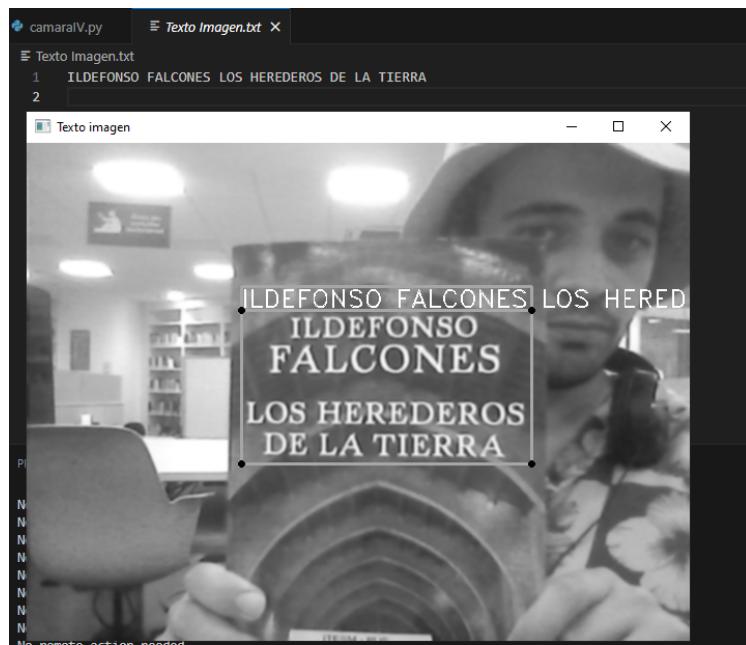


Imagen 0.13: Tamaño de letra 1cm, distancia de cámara 30cm, resultado: correcto

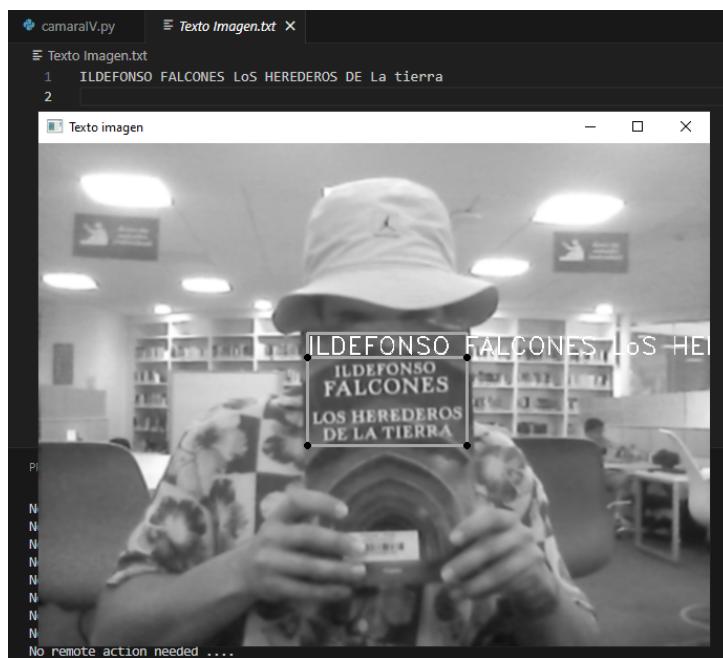


Imagen 0.14: Tamaño de letra 1cm, distancia de cámara 60cm, resultado: correcto

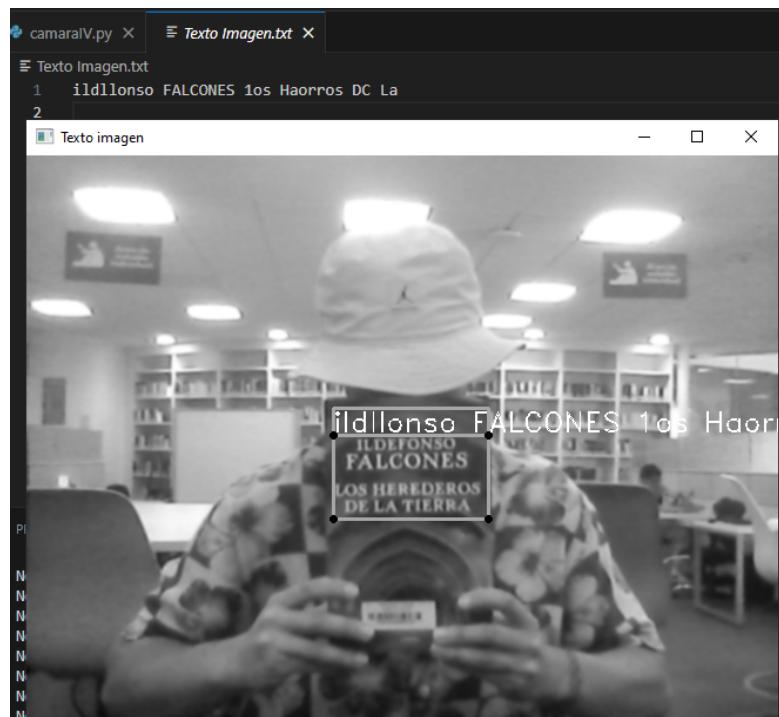


Imagen 0.15: Tamaño de letra 1cm, distancia de cámara 75cm, resultado: incorrecto

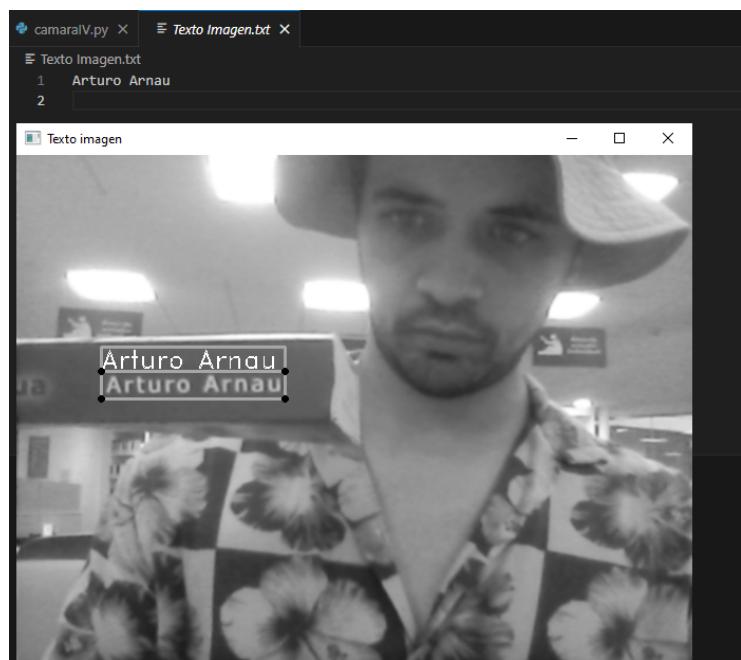


Imagen 0.16: Tamaño de letra 0.5cm, distancia de cámara 30cm, resultado: correcto

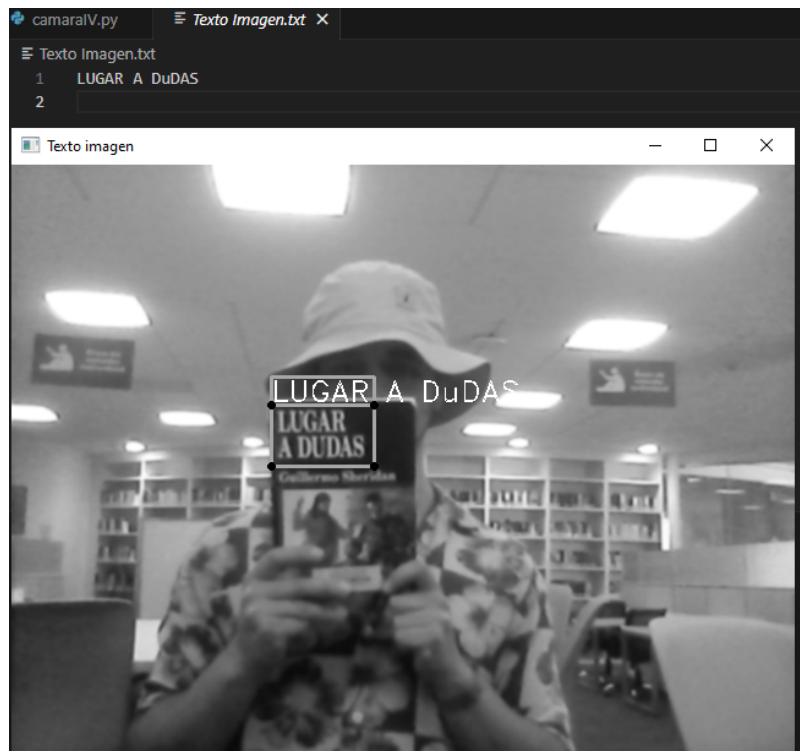


Imagen 0.17: Tamaño de letra 2cm, distancia de cámara 120cm, resultado: correcto

Pruebas de descripción de imagen (VGA camera)

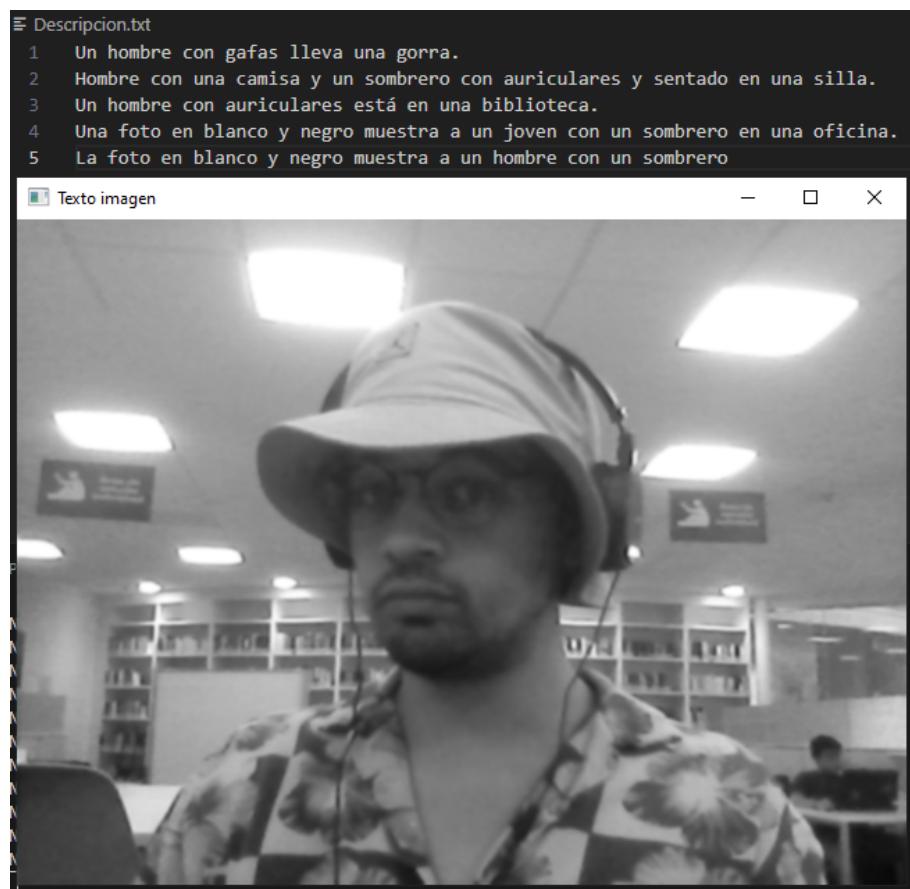


Imagen 0.18: Prueba de descripción de imagen, resultado: describe de manera bastante precisa el contenido de la misma.



Imagen 0.19: Segunda prueba de descripción de imagen, resultado: describe de manera bastante precisa el contenido de la misma.



Imagen 0.20: Tercera prueba de descripción de imagen, resultado: describe bien el contenido general, pero no es tan preciso al identificar los objetos individuales.

Prueba de traductor de español a braille

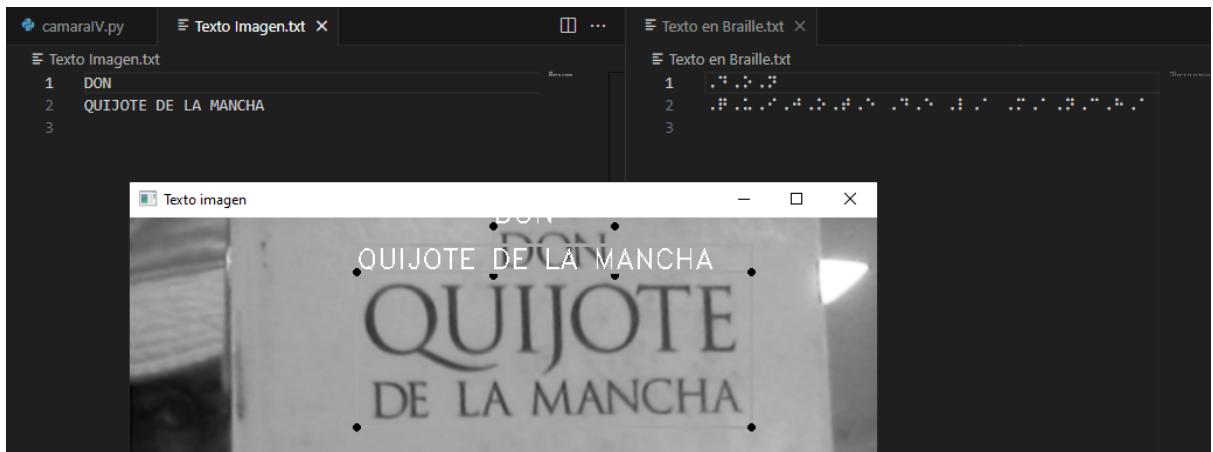


Imagen 0.21: Reconocimiento de texto y traducción a lenguaje braille.

audio_descripcion	31/05/2023 10:58 p. m.	Archivo MP3	106 KB
audio_texto	31/05/2023 10:58 p. m.	Archivo MP3	7 KB

Imagen 0.22. Archivos de audio generados.

Pruebas con cámara HD HP TrueVision



Imagen 1.1: Descripción y texto obtenido de libro con un ángulo 0° a una distancia ~30 cm.

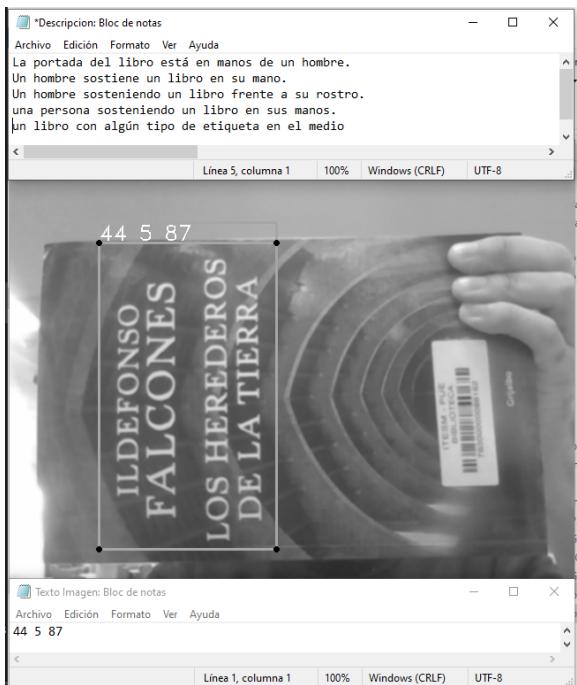


Imagen 1.2: Descripción y texto obtenido de libro con un ángulo 90° a una distancia ~30 cm;90°

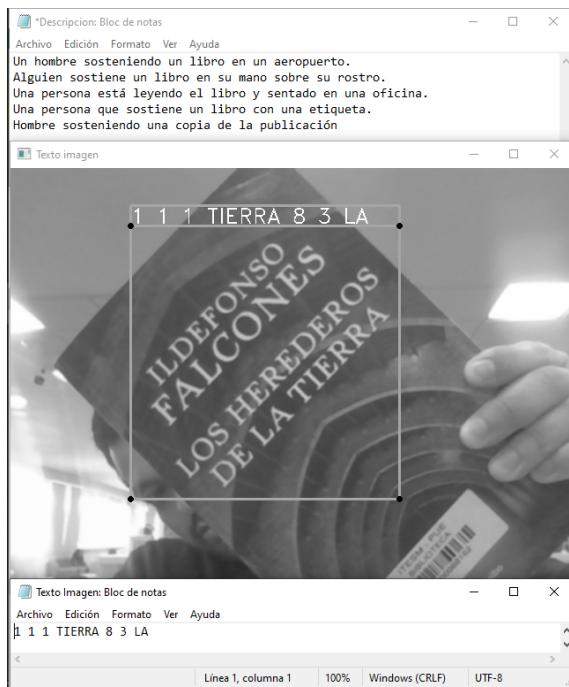


Imagen 1.3: Descripción y texto obtenido de libro con un ángulo 45º a una distancia ~30 cm; primer intento.

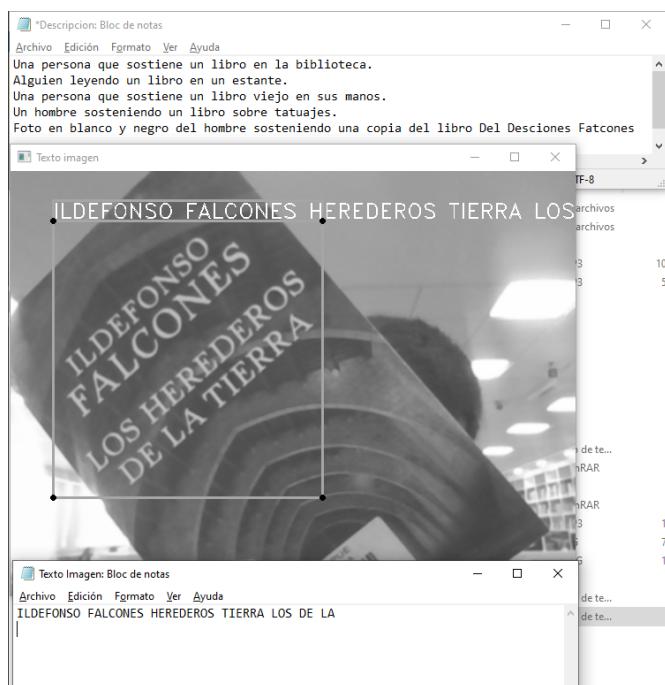


Imagen 1.4: Descripción y texto obtenido de libro con un ángulo 45º a una distancia ~30 cm; segundo intento.

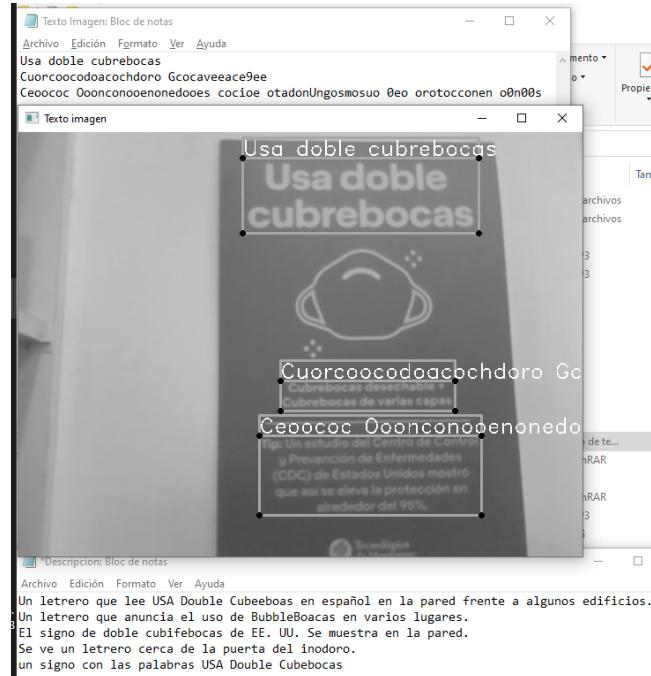


Imagen 1.5: Descripción y texto obtenido de un cartel en una biblioteca a una distancia ~90 cm.

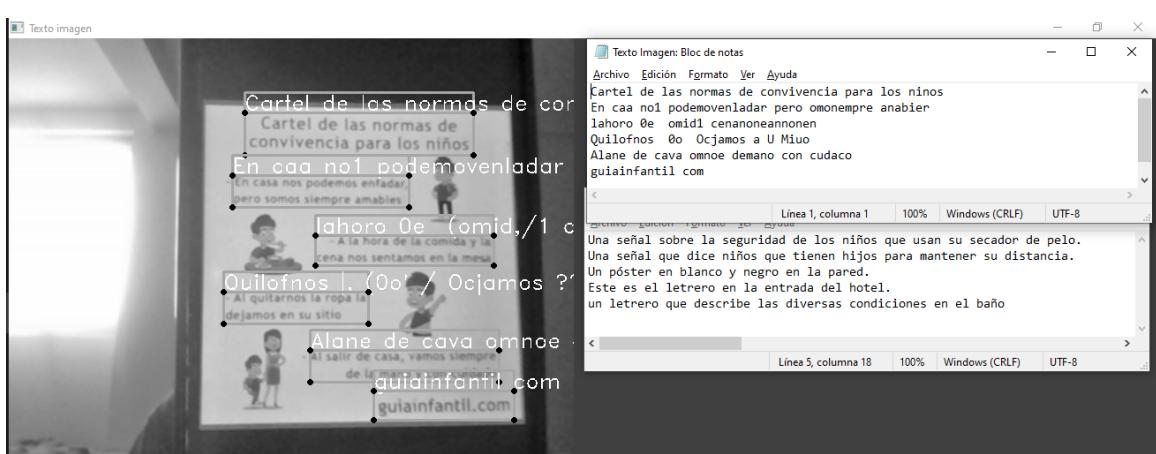
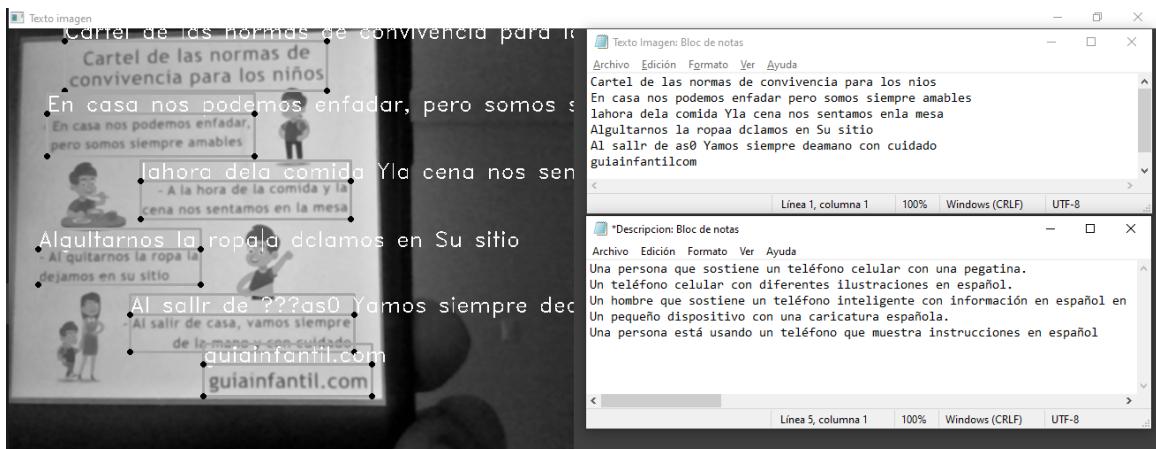


Imagen 1.6: Comparación de analizar la imagen en el celular con y sin luz en el fondo.

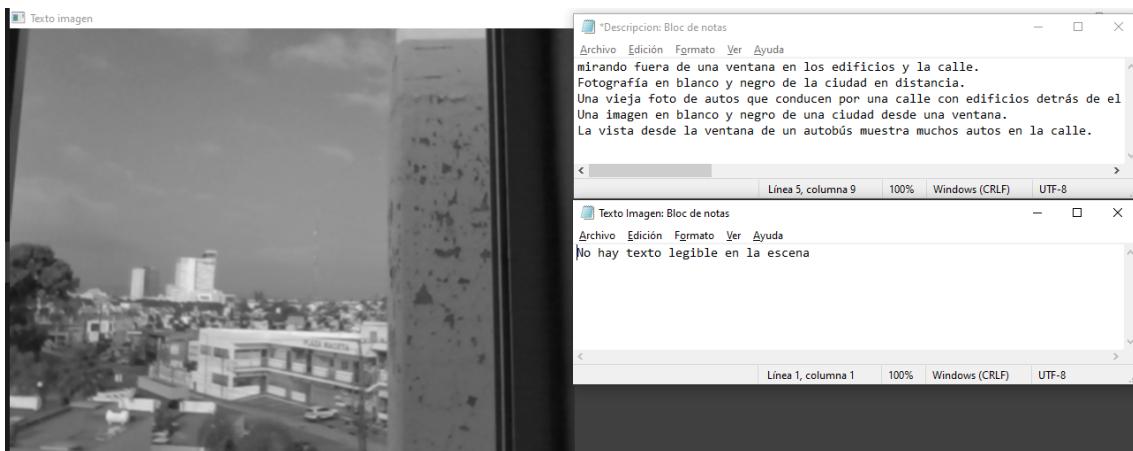


Imagen 1.7: Texto encontrado y descripción de paisaje.

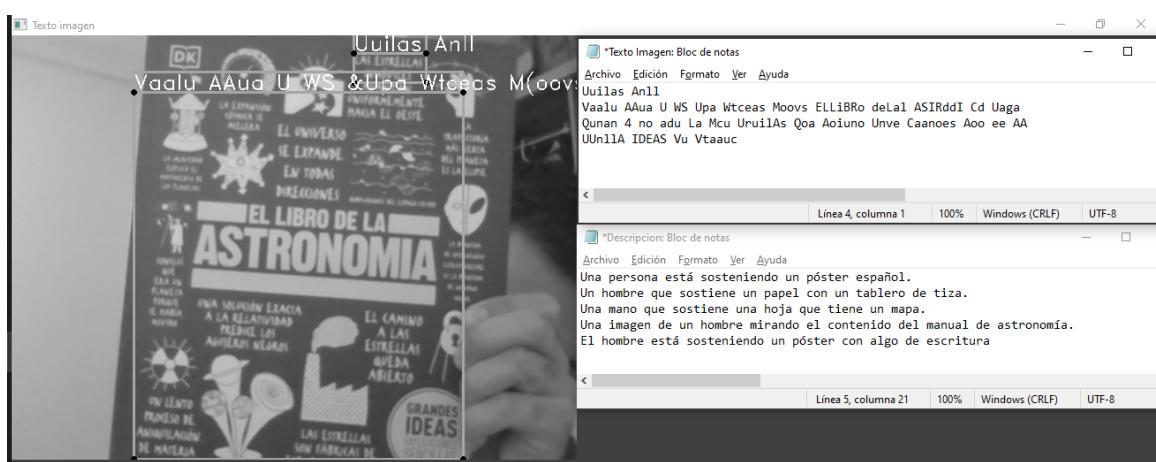
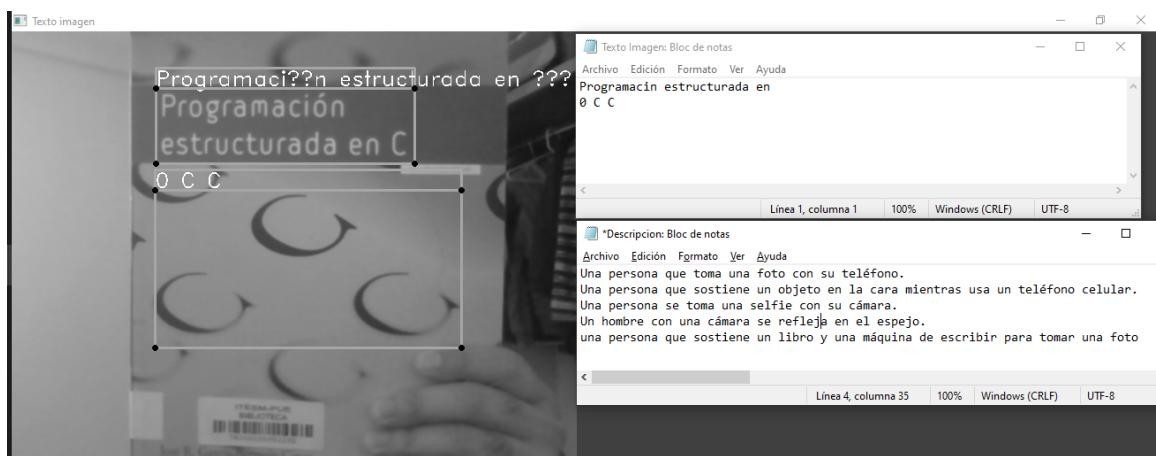


Imagen 1.8: Comparación entre imágenes con poco y mucho texto.

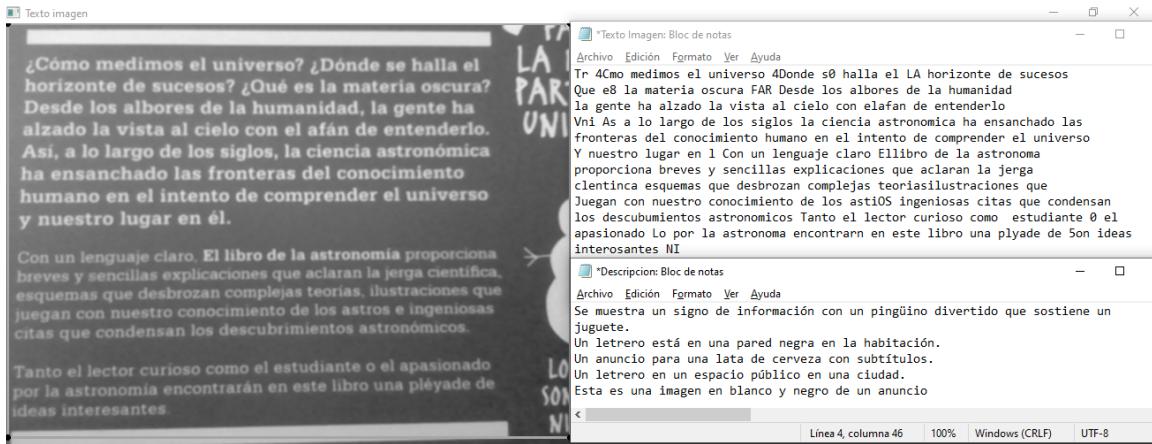


Imagen 1.9: Texto encontrado y descripción de contraportada de libro.

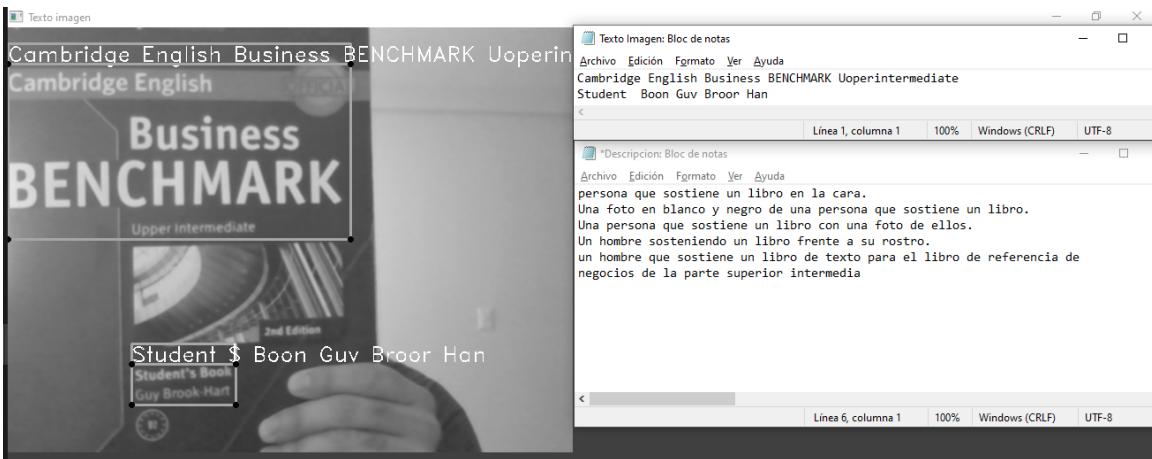


Imagen 1.10: Texto encontrado y descripción de portada de libro en inglés.

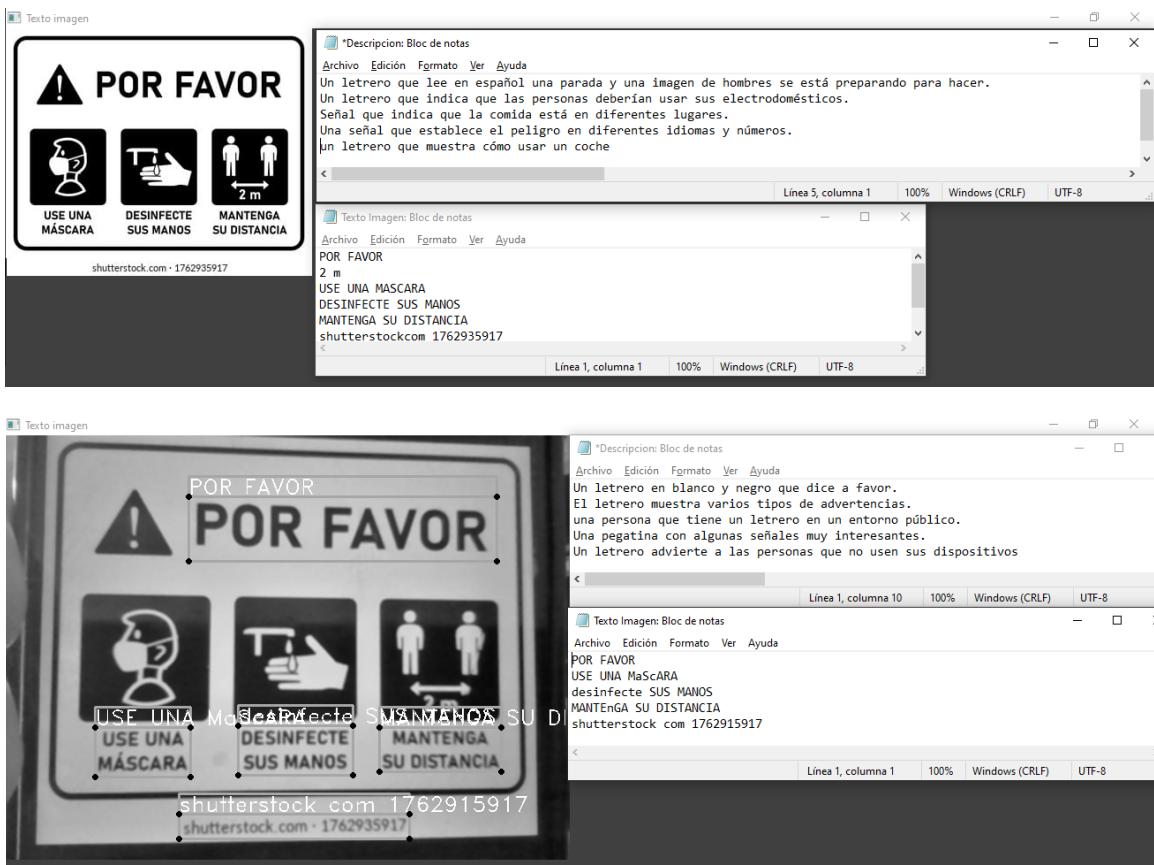
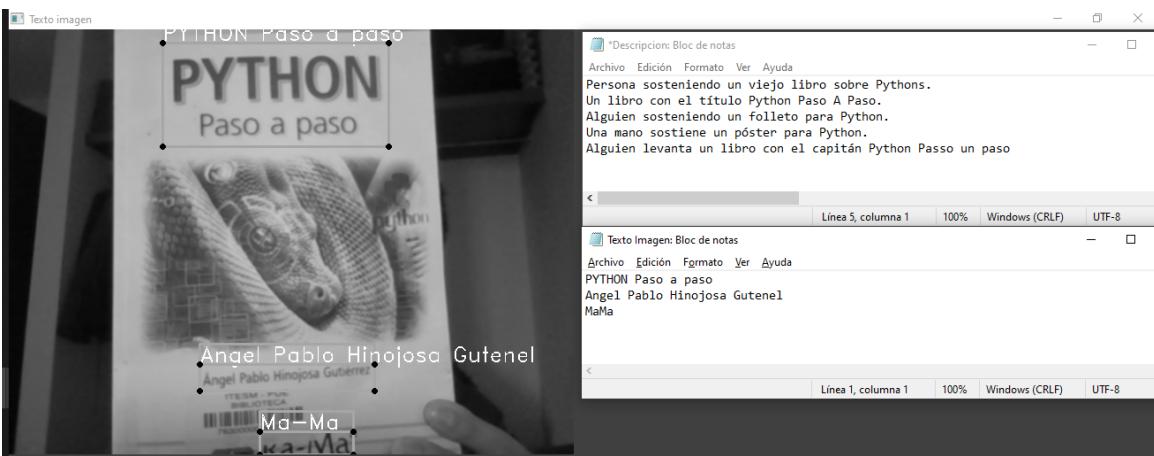


Imagen 1.11: Comparación de descripción y texto encontrado entre entrada de imagen desde el celular y entrada de la imagen como archivo.



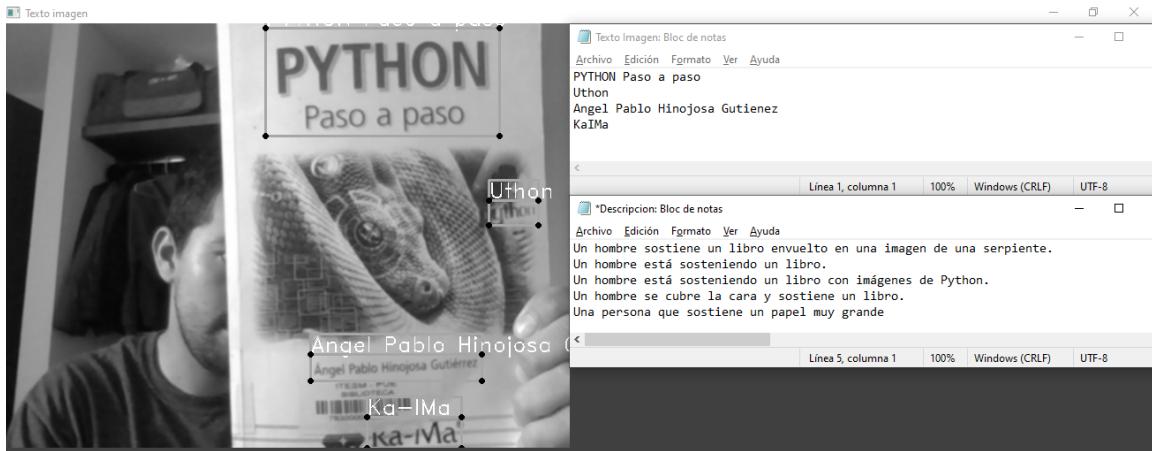


Imagen 1.12: Comparación de descripción y texto encontrado en portada de libro con(imagen de arriba) y sin (imagen de abajo) luz a la espalda de la cámara.

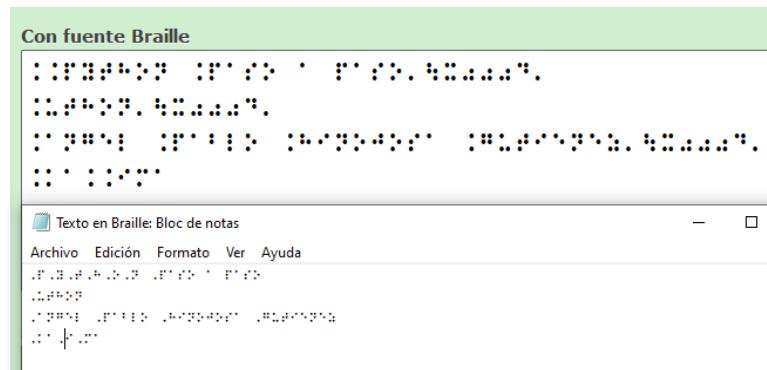


Imagen 1.13: Comparación de texto en braille generado desde una página web y el generado por nuestro proyecto.