

UNIVERSIDAD NACIONAL DE PIURA

*Facultad de Ciencias*

*Escuela Profesional de Ingeniería Electrónica y Telecomunicaciones*

INFORME DE TRABAJO FINAL

*"Implementación de Reconocimiento Óptico de Caracteres (OCR)*

*utilizando Python, OpenCV y Tesseract con exportación de resultados a PDF y Word"*

Curso: Procesamiento digital de señales

Docente: Mag.Ing. Yonatan Aguirre

Alumnos:

● Bailón Fiorentini Jean Pier Igor

● Martinez Yovera Jose Luis

● Goicochea Lalupú Joseph Aldahir

Ciclo: 2025-I

Fecha: [01/08/2025]

## Índice

1. **Portada**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
2. **Indice**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
3. Introducción. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..3
4. Justificación. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..3
5. Objetivos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4  
   5.1 Objetivo general. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..4  
   5.2 Objetivos específicos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4
6. Marco Teórico. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5  
   6.1 Reconocimiento Óptico de Caracteres (OCR) . . . . . . . . . . . . . . . . . . . . . . . . 5  
   6.2 Preprocesamiento de imágenes. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5  
   6.3 Tesseract OCR. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5  
   6.4 Exportación de resultados. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

6.5 Google Colab. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .6

1. Metodología. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7

7.1 Diagrama de Flujo. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7  
7.2 Descripción de los bloques de código. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7   
- Instalación de dependencias. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

- Configuración de librerías. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .8  
- Subida y visualización de imagen. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .9  
- Preprocesamiento de la imagen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9   
- Detección de contornos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10  
- Función para ordenar puntos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11  
- Transformación de perspectiva y OCR. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .12  
- Generación de PDF. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .13

- Diferencias principales entre PDF+ WORD. . . . . . . . . . . . . . . . . . . . . . . . . . .15

1. Ventajas del método en Google colab frente al ESP32-CAM. . . . . . . . . . . . . . .17
2. Resultados. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .18  
   9.1 Imagen original . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .19  
   9.2 Imagen en escala de grises . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19  
   9.3 Detección de bordes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20  
   9.4 Contornos detectados . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21  
   9.5 Puntos ordenados. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .21  
   9.6 Imagen transformada para OCR . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .22  
   9.7 Texto OCR detectado. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23  
   9.8 Generación de PDF. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
3. Conclusiones. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
4. Anexos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .24

11.1 Código completo del **OCR en PDF**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

11.2 Código completo del **OCR en PDF** y Word. . . . . . . . . . . . . . . . . . . . . . . . . . . 27

## Introducción

El reconocimiento óptico de caracteres (OCR) es una tecnología que permite extraer texto de imágenes de forma automática. Su aplicación es fundamental en áreas como la digitalización de documentos, la automatización de procesos administrativos y la integración de sistemas de visión artificial.

En este trabajo se implementa un sistema de OCR utilizando Python como lenguaje principal debido a su amplia disponibilidad de librerías científicas y de procesamiento de imágenes. Se emplea OpenCV para el preprocesamiento de imágenes, lo que incluye la conversión a escala de grises, detección de bordes y corrección de perspectiva, con el fin de mejorar la legibilidad del texto antes de su reconocimiento. Para la extracción de caracteres, se utiliza Tesseract OCR, una herramienta de código abierto ampliamente validada en entornos industriales y académicos.

Además, el desarrollo se realiza en Google Colab, aprovechando su entorno en la nube para ejecutar el código sin necesidad de configuraciones locales complejas. Finalmente, los resultados obtenidos se integran en informes generados automáticamente en formato PDF y Word, incorporando tanto las imágenes de cada etapa de procesamiento como el texto detectado.

Este enfoque demuestra una solución práctica, reproducible y extensible que puede adaptarse fácilmente a diferentes fuentes de captura de imágenes, incluyendo cámaras integradas como la ESP32-CAM, sin modificar el flujo principal de procesamiento y reconocimiento.

## Justificación

El reconocimiento óptico de caracteres (OCR) es una herramienta clave en la automatización de tareas que requieren la digitalización y análisis de información impresa. En el contexto académico y profesional, comprender los fundamentos de OCR y su implementación práctica permite sentar las bases para desarrollar soluciones más avanzadas, como sistemas de visión artificial o aplicaciones integradas con microcontroladores.

En este trabajo se eligió Python debido a su versatilidad, simplicidad y la amplia disponibilidad de bibliotecas de procesamiento de imágenes y generación de documentos. La combinación de OpenCV para el preprocesamiento de imágenes y Tesseract OCR para la lectura de texto constituye una solución robusta y ampliamente utilizada en entornos reales.

El uso de Google Colab se justifica por su entorno basado en la nube, el cual elimina barreras de instalación y facilita la ejecución del proyecto en cualquier equipo con conexión a Internet. Asimismo, se implementó la exportación automática de los resultados a PDF y Word, garantizando un informe profesional y directamente utilizable.

Aunque el presente trabajo no emplea directamente la captura de imágenes desde una ESP32-CAM, la metodología desarrollada es totalmente compatible con dicha integración. Esto demuestra que la solución propuesta no solo cumple los objetivos académicos, sino que también es escalable y adaptable a aplicaciones más complejas en el ámbito de la ingeniería electrónica y de sistemas embebidos.

En síntesis, la elección de esta metodología permite trabajar con herramientas estandarizadas y demostrar un flujo de trabajo reproducible, eficiente y de utilidad práctica para futuras implementaciones.

## Objetivos

### Objetivo general

Desarrollar un sistema de reconocimiento óptico de caracteres (OCR) basado en Python, OpenCV y Tesseract que permita procesar imágenes, aplicar transformaciones de preprocesamiento y generar informes automáticos de resultados en formato PDF y Word, demostrando la integración de librerías de visión por computadora y herramientas de automatización de documentos.

### Objetivos específicos

Analizar el funcionamiento de Tesseract OCR, para el reconocimiento de texto en imágenes y su integración con Python.

Implementar técnicas de procesamiento de imágenes con OpenCV, incluyendo:

* Conversión a escala de grises
* Detección de bordes
* Identificación de contornos
* Ordenamiento de puntos y corrección de perspectiva

Realizar el reconocimiento de texto (OCR), sobre imágenes procesadas y optimizadas para maximizar la precisión de lectura.

Diseñar un flujo de trabajo reproducible en Google Colab, que permita ejecutar todo el proceso sin necesidad de instalación compleja en el sistema local.

Automatizar la generación de informes en formato PDF y Word, integrando imágenes intermedias del procesamiento y el texto OCR reconocido.

Evaluar la escalabilidad de la solución, justificando cómo el mismo flujo podría adaptarse para recibir imágenes desde otros dispositivos, como una cámara ESP32-CAM.

Documentar y explicar cada bloque de código, para garantizar la comprensión detallada de cada etapa del proceso.

## Marco Teórico

### 1. Reconocimiento Óptico de Caracteres (OCR)

El OCR (Optical Character Recognition) es una técnica de visión artificial que permite convertir texto presente en imágenes o documentos escaneados en información digital editable. Para lograrlo, se aplican algoritmos de segmentación, detección de caracteres y reconocimiento de patrones.  
Su uso es común en sistemas de gestión documental, lectura automática de facturas, procesamiento de formularios y aplicaciones de accesibilidad.  
En este proyecto, el motor OCR seleccionado es Tesseract, una herramienta de código abierto desarrollada inicialmente por HP y actualmente mantenida por Google, la cual ofrece soporte multilenguaje y alta precisión.

### 2. Preprocesamiento de imágenes

Antes de aplicar OCR, es necesario realizar una serie de transformaciones que mejoren la calidad de la imagen y reduzcan el ruido visual:

Conversión a escala de grises: simplifica la imagen eliminando la información de color, conservando únicamente la intensidad de los píxeles.

Detección de bordes: permite identificar las zonas de transición de intensidad mediante algoritmos como Canny.

Detección de contornos: facilita aislar las regiones de interés que podrían contener texto.

Transformación de perspectiva: corrige distorsiones de la imagen para mejorar la alineación de los caracteres y aumentar la precisión del OCR.

Estas operaciones se implementan con OpenCV, una biblioteca de visión por computadora ampliamente utilizada en investigación y aplicaciones industriales.

### 3. Tesseract OCR

Tesseract es un motor OCR que soporta múltiples lenguajes, incluyendo el español, y utiliza redes neuronales LSTM (Long Short-Term Memory) para el reconocimiento de texto. Su integración con Python se realiza mediante la librería pytesseract, la cual permite enviar imágenes a Tesseract y obtener el texto reconocido como una cadena de caracteres.  
Tesseract también admite configuraciones adicionales, como:

Selección de idioma.

Definición de áreas específicas de análisis.

Aplicación de filtros de precisión.

### 4. Exportación de resultados

La presentación de los resultados en un formato estructurado es fundamental para su interpretación y validación. En este proyecto se emplean dos métodos de exportación:

4.1 Generación de PDF:

Se utiliza la librería FPDF para crear informes en formato PDF que incluyen:

* Imágenes de cada etapa del procesamiento.
* Texto OCR reconocido.
* Formato ordenado y compatible con cualquier visor PDF.

4.2 Generación de documentos Word

Para permitir la edición y personalización del informe, se implementa la exportación a Word utilizando python-docx. Esta librería permite:

* Insertar imágenes en el documento.
* Añadir encabezados y texto.
* Guardar el informe en formato .docx para futuras modificaciones.

### 5. Google Colab

Google Colab es un entorno de desarrollo basado en la nube que permite ejecutar código Python sin necesidad de configuración local. Además, ofrece:

Integración directa con librerías como OpenCV, pytesseract, FPDF y python-docx.

Soporte para ejecución gratuita con recursos de GPU y CPU.

Posibilidad de cargar y descargar archivos fácilmente.

En este proyecto, Colab se utiliza como plataforma principal debido a su facilidad de uso y a su compatibilidad con el flujo de trabajo requerido.

## Metodología

### Diagrama de Flujo

[Inicio]

↓

[Instalación de dependencias]

↓

[Subida de imagen]

↓

[Preprocesamiento de imagen]

↓

[Detección de contornos]

↓

[Ordenamiento de puntos y transformación de perspectiva]

↓

[OCR con Tesseract]

↓

[Generación de informe PDF]

↓

[Fin]

### Descripción de los bloques de código

### Instalación de dependencias

!apt-get install -y tesseract-ocr

!pip install pytesseract opencv-python-headless numpy matplotlib

!apt-get install -y tesseract-ocr-spa

!pip install fpdf

Este bloque garantiza que el entorno de Google Colab cuente con todas las herramientas necesarias:

| Dependencia | Función principal |
| --- | --- |
| tesseract-ocr | Instala el motor OCR que realizará el reconocimiento de texto. |
| tesseract-ocr-spa | Añade el paquete de idioma español, necesario para interpretar correctamente caracteres como "ñ" o acentos. |
| pytesseract | Librería de Python que sirve de puente entre el script y Tesseract. |
| opencv-python-headless | Permite realizar procesamiento de imágenes sin necesidad de interfaz gráfica. |
| numpy | Facilita operaciones matemáticas y de matrices, útiles para la manipulación de imágenes. |
| matplotlib | Se usa para mostrar imágenes procesadas paso a paso. |
| fpdf | Genera archivos PDF con imágenes y texto OCR para documentar el proceso. |

En Colab no es necesario configurar rutas de instalación manualmente; sin embargo, es indispensable instalar el paquete de idioma para que Tesseract pueda reconocer texto en español.

### Configuración de librerías

import cv2

import numpy as np

import pytesseract

import matplotlib.pyplot as plt

from fpdf import FPDF

pytesseract.pytesseract.tesseract\_cmd = r'/usr/bin/tesseract'

Se importan las librerías requeridas y se configura la ruta de Tesseract para Google Colab:

* cv2 (OpenCV): biblioteca para manipular imágenes y realizar operaciones como filtrado, detección de bordes y transformaciones geométricas.
* numpy: proporciona soporte para arrays multidimensionales, necesarios en el tratamiento de imágenes.
* pytesseract: facilita la conexión entre el script y el motor OCR de Tesseract.
* matplotlib: se utiliza para visualizar las imágenes en cada etapa del procesamiento.
* fpdf: prepara la generación automática de informes PDF con resultados intermedios.
* Además, pytesseract.pytesseract.tesseract\_cmd establece explícitamente la ruta de ejecución de Tesseract dentro de Colab (/usr/bin/tesseract), garantizando que el motor OCR funcione correctamente.

### Subida y visualización de imagen

uploaded = files.upload()

image = cv2.imread(list(uploaded.keys())[0])

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

plt.imshow(image\_rgb)

plt.title("Imagen original")

plt.axis('off')

plt.show()

Se sube una imagen desde el equipo local y se convierte de BGR a RGB para mostrarla correctamente en Colab:

* Carga de imagen: files.upload() permite al usuario subir imágenes desde su PC al entorno de Colab.
* Lectura de imagen: cv2.imread() carga la imagen en formato BGR, que es el estándar en OpenCV.
* Conversión de color: cv2.cvtColor(..., cv2.COLOR\_BGR2RGB) transforma la imagen a RGB para que se muestre correctamente con matplotlib.
* Visualización: plt.imshow() muestra la imagen sin bordes ni ejes (plt.axis('off')), facilitando la inspección visual.

***Visualizar la imagen original es crucial para verificar que la carga y el formato son correctos antes de iniciar el procesamiento***

### Preprocesamiento de la imagen

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

plt.imshow(gray, cmap='gray')

plt.title("Escala de grises")

plt.axis('off')

plt.show()

canny = cv2.Canny(gray, 10, 150)

canny = cv2.dilate(canny, None, iterations=1)

plt.imshow(canny, cmap='gray')

plt.title("Detección de bordes (Canny)")

plt.axis('off')

plt.show()

Se reduce la complejidad de la imagen eliminando el color y se detectan los bordes usando el algoritmo Canny.

El preprocesamiento prepara la imagen para el OCR:

* Escala de grises:

Se reduce la complejidad de la imagen eliminando la información de color.

La imagen pasa de 3 canales (RGB) a 1 solo canal (intensidad de luz).

Esto mejora la eficiencia y facilita la detección de bordes.

* Detección de bordes (Canny):

cv2.Canny(gray, 10, 150) aplica el algoritmo de Canny para detectar bordes relevantes.

Los valores 10 y 150 son los umbrales inferior y superior para definir los bordes.

Esta operación segmenta los elementos importantes de la imagen (como los bordes de un documento o caracteres).

* Dilatación:

cv2.dilate() expande los bordes detectados para conectarlos y facilitar la detección de contornos.

Es útil cuando los bordes son discontinuos o demasiado delgados.

* Visualización:

***Se muestran tanto la imagen en grises como el resultado de la detección de bordes, lo que permite verificar que los parámetros elegidos son adecuados.***

### Detección de contornos

cnts = cv2.findContours(canny, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)[0]

cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:1]

image\_contours = image.copy()

cv2.drawContours(image\_contours, cnts, -1, (0, 255, 0), 2)

plt.imshow(cv2.cvtColor(image\_contours, cv2.COLOR\_BGR2RGB))

plt.title("Contornos detectados")

plt.axis('off')

plt.show()

* Se identifican los contornos más grandes para localizar la región de interés, y se dibujan sobre la imagen.

cv2.findContours(): Localiza los bordes cerrados presentes en la imagen de bordes.

cv2.RETR\_EXTERNAL: Busca solo los contornos externos (evita detectar bordes internos irrelevantes).

cv2.CHAIN\_APPROX\_SIMPLE: Almacena solo los puntos esenciales de cada contorno para optimizar memoria.

* Ordenamiento de contornos:

sorted(cnts, key=cv2.contourArea, reverse=True)[:1]: selecciona el contorno más grande (normalmente el documento o la región principal de texto).

* Dibujo de contornos:

cv2.drawContours() marca los contornos detectados en verde sobre la imagen original.

Esto permite verificar visualmente si el contorno detectado coincide con el área de interés.

***Este paso es fundamental para aislar el documento o la región de texto y descartar información irrelevante de la imagen.***

### Función para ordenar puntos

n\_puntos = np.concatenate([puntos[0], puntos[1], puntos[2], puntos[3]]).tolist()

y\_order = sorted(n\_puntos, key=lambda n\_puntos: n\_puntos[1])

x1\_order = sorted(y\_order[:2], key=lambda x: x[0])

x2\_order = sorted(y\_order[2:4], key=lambda x: x[0])

return [x1\_order[0], x1\_order[1], x2\_order[0], x2\_order[1]]

Ordena los 4 puntos detectados de un contorno en una secuencia consistente (esquina superior izquierda, superior derecha, inferior izquierda, inferior derecha).

* Pasos internos:

Se concatenan los 4 puntos detectados.

Se ordenan por la coordenada Y para separar los puntos superiores de los inferiores.

Dentro de cada par (superior e inferior), se ordenan por X para diferenciar izquierda y derecha.

***Con esto se garantiza una base sólida para realizar una transformación de perspectiva sin errores geométricos.***

### Transformación de perspectiva y OCR

for c in cnts:

epsilon = 0.01 \* cv2.arcLength(c, True)

approx = cv2.approxPolyDP(c, epsilon, True)

if len(approx) == 4:

puntos = ordenar\_puntos(approx)

image\_points = image.copy()

colores = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0)]

for i, punto in enumerate(puntos):

cv2.circle(image\_points, tuple(punto), 7, colores[i], 2)

plt.imshow(cv2.cvtColor(image\_points, cv2.COLOR\_BGR2RGB))

plt.title("Puntos ordenados")

plt.axis('off')

plt.show()

pts1 = np.float32(puntos)

pts2 = np.float32([[0, 0], [270, 0], [0, 310], [270, 310]])

M = cv2.getPerspectiveTransform(pts1, pts2)

dst = cv2.warpPerspective(gray, M, (270, 310))

plt.imshow(dst, cmap='gray')

plt.title("Imagen transformada para OCR")

plt.axis('off')

plt.show()

texto = pytesseract.image\_to\_string(dst, lang='spa')

print("Texto detectado:", texto)

* Aproximación de contornos:

cv2.approxPolyDP() simplifica el contorno para identificar polígonos.

Si el resultado tiene 4 puntos, se asume que el documento es rectangular.

* Dibujo de puntos:

Cada esquina detectada se marca con un color diferente para verificar su posición.

* Transformación de perspectiva:

cv2.getPerspectiveTransform() genera una matriz de transformación que "endereza" el documento.

cv2.warpPerspective() aplica la transformación para corregir inclinaciones o deformaciones.

OCR: pytesseract.image\_to\_string() reconoce el texto dentro de la imagen transformada.

Se usa el parámetro lang='spa' para asegurar que el reconocimiento sea en español.

***Se obtiene una imagen perfectamente alineada para una lectura OCR precisa.***

### Generación de PDF

pdf = FPDF()

pdf.set\_auto\_page\_break(auto=True, margin=15)

pdf.add\_page()

pdf.set\_font("Arial", size=12)

pdf.cell(200, 10, "Imagen original", ln=True)

plt.imsave("original.png", image\_rgb)

pdf.image("original.png", x=10, y=30, w=100)

pdf.add\_page()

pdf.cell(200, 10, "Escala de grises", ln=True)

plt.imsave("grises.png", gray, cmap='gray')

pdf.image("grises.png", x=10, y=30, w=100)

pdf.add\_page()

pdf.cell(200, 10, "Detección de bordes", ln=True)

plt.imsave("bordes.png", canny, cmap='gray')

pdf.image("bordes.png", x=10, y=30, w=100)

pdf.add\_page()

pdf.cell(200, 10, "Contornos detectados", ln=True)

plt.imsave("contornos.png", cv2.cvtColor(image\_contours, cv2.COLOR\_BGR2RGB))

pdf.image("contornos.png", x=10, y=30, w=100)

pdf.add\_page()

pdf.cell(200, 10, "Texto OCR detectado", ln=True)

pdf.multi\_cell(190, 10, texto)

pdf.output("resultado\_OCR.pdf")

files.download("resultado\_OCR.pdf")

* Inicialización del PDF:

FPDF() crea un documento vacío.

set\_auto\_page\_break() activa saltos de página automáticos.

* Inserción de imágenes y texto:

Se añaden páginas para cada paso del proceso.

plt.imsave() guarda imágenes temporales de cada etapa.

pdf.image() inserta esas imágenes en el documento PDF.

* Resultado final:

Se escribe el texto OCR en una página independiente.

El PDF se guarda y descarga automáticamente en Colab.

***Se genera un informe en PDF con cada paso del proceso y el texto OCR reconocido.***

## Diferencias principales entre (PDF + Word)

### Instalación de librerías adicionales

!apt-get install -y tesseract-ocr tesseract-ocr-spa

!pip install pytesseract opencv-python-headless numpy matplotlib fpdf python-docx

Se añade python-docx para generar documentos Word (.docx).

Tesseract sigue igual, solo se instala el idioma español (tesseract-ocr-spa).

### Procesamiento de imágenes

El procesamiento es idéntico al de la versión PDF (subida, conversión a grises, bordes, contornos, puntos y OCR).

Se guardan imágenes temporales (original.png, grises.png, bordes.png, etc.) para reutilizarlas tanto en PDF como en Word.

### Generación de PDF

pdf = FPDF()

pdf.set\_auto\_page\_break(auto=True, margin=15)

for archivo, titulo in imagenes:

if os.path.exists(archivo):

pdf.add\_page()

pdf.set\_font("Arial", size=12)

pdf.cell(200, 10, titulo, ln=True)

pdf.image(archivo, x=10, y=30, w=100)

pdf.add\_page()

pdf.set\_font("Arial", size=12)

pdf.cell(200, 10, "Texto OCR detectado", ln=True)

pdf.multi\_cell(190, 10, texto)

pdf.output("resultado\_OCR.pdf")

files.download("resultado\_OCR.pdf")

Igual al método anterior, pero ahora la lista de imágenes imagenes permite automatizar la inclusión de todos los pasos.

### Generación de Word

document = Document()

document.add\_heading('Resultados de OCR paso a paso', level=1)

for archivo, titulo in imagenes:

if os.path.exists(archivo):

document.add\_heading(titulo, level=2)

document.add\_picture(archivo, width=Inches(4))

texto\_limpio = ''.join(ch for ch in texto if ch.isprintable())

document.add\_heading("Texto OCR detectado", level=2)

document.add\_paragraph(texto\_limpio)

document.save("resultado\_OCR.docx")

files.download("resultado\_OCR.docx")

* Nuevo bloque exclusivo:

Se crea un documento Word con python-docx.

Se añaden las imágenes igual que en PDF, pero con títulos jerárquicos (add\_heading).

Se limpia el texto OCR para evitar caracteres no imprimibles.

Se agrega el texto reconocido como párrafo.

Finalmente, se descarga el documento Word.

### Ventajas de esta segunda versión

* **Automatización total:** Un solo flujo genera PDF y Word sin necesidad de código duplicado.
* **Formato profesional:** Word permite edición posterior del documento sin necesidad de regenerarlo.
* **Integración escalable:** Si mañana agregas más pasos al OCR, solo se añaden a la lista (imágenes).
* **Compatibilidad:** Word es más accesible para entornos académicos que solo aceptan archivos editables.

## Ventajas del método en Google Colab frente al ESP32-CAM

| Aspecto | Método en Google Colab (PC) | Método con ESP32-CAM |
| --- | --- | --- |
| Potencia de procesamiento | Aprovecha la CPU (y en algunos casos GPU de Colab), lo que permite un análisis más rápido y preciso de imágenes complejas. | Limitado al microcontrolador ESP32, que tiene recursos muy reducidos. |
| Precisión del OCR | Usa Tesseract con soporte de múltiples idiomas, configuraciones avanzadas y librerías de preprocesamiento (OpenCV, NumPy). | El ESP32 no puede ejecutar Tesseract directamente, normalmente solo captura imágenes y las envía a un servidor. |
| Facilidad de desarrollo | Se desarrolla y prueba directamente en Python con un entorno flexible y muchas librerías. | Requiere programación en C++/Arduino y librerías limitadas. |
| Preprocesamiento avanzado | Se pueden aplicar filtros, transformaciones de perspectiva, mejoras de contraste y limpieza de ruido. | El ESP32 no tiene recursos suficientes para realizar preprocesamiento avanzado. |
| Integración con informes | Permite generar PDFs y documentos Word automáticamente con imágenes procesadas y resultados OCR. | El ESP32 solo puede enviar datos, no generar informes. |
| Escalabilidad | Puede procesar múltiples imágenes de alta resolución sin limitaciones de memoria. | Memoria muy limitada, imágenes de baja resolución para evitar saturación. |
| Idioma y configuraciones OCR | Compatible con tesseract-ocr-spa y otros idiomas fácilmente. | Dependería de un servidor externo, ya que el ESP32 solo captura. |

### Conclusiones de las ventajas

* Mayor calidad de resultados: El procesamiento en PC/Colab ofrece un OCR más preciso, especialmente en documentos complejos.
* Menor dependencia de hardware: No se necesita un microcontrolador ni módulos de cámara físicos.
* Documentación automática: La integración directa con generación de PDF/Word es imposible de implementar directamente en ESP32.
* Flexibilidad y escalabilidad: Puedes aplicar nuevas técnicas de visión artificial sin restricciones de memoria o potencia.

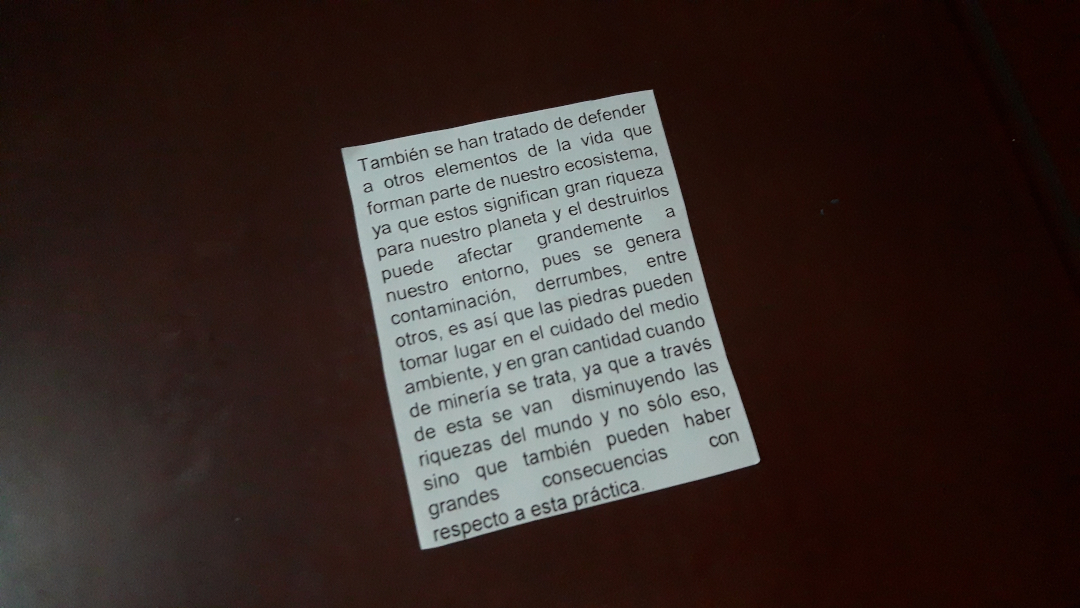
## Resultados

En esta sección se presentan los resultados obtenidos en cada etapa del procesamiento de imágenes y generación de documentos PDF y Word. Para cada bloque de código ejecutado, se incluye la imagen correspondiente y el texto reconocido por el OCR.

### 1️. Imagen original

Carga y visualización de la imagen original.

Resultado visual:

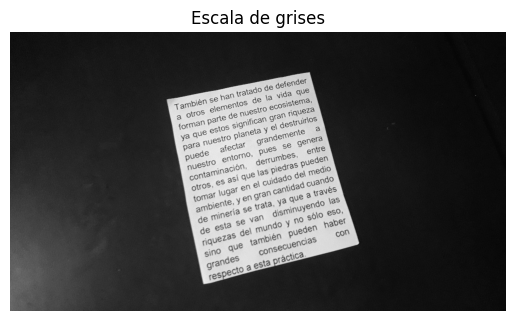


Se observa la imagen inicial cargada desde el sistema. Esta es la base sobre la cual se aplicarán los pasos de preprocesamiento y análisis.

### 2️. Escala de grises

Conversión de la imagen a escala de grises.

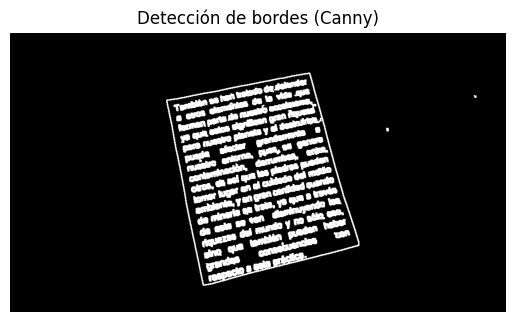
Resultado visual:

  
El paso a escala de grises elimina información de color innecesaria, optimizando el rendimiento de los algoritmos de detección de bordes y contornos.

### 3️. Detección de bordes

Aplicación del algoritmo Canny y dilatación de bordes.

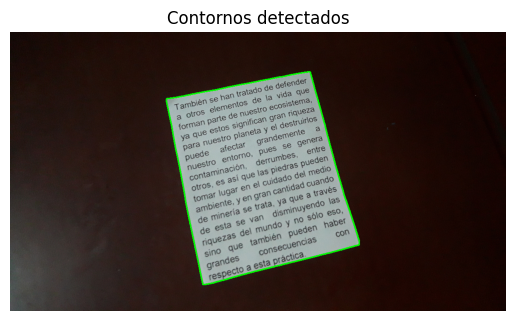
Resultado visual:

  
Se resaltan los bordes principales de la imagen, lo que permite aislar las regiones de interés para la detección de contornos.

### 4️. Contornos detectados

Localización y dibujo de contornos en la imagen.

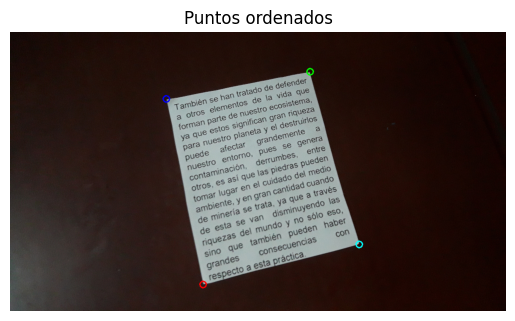
Resultado visual:

  
Los contornos permiten identificar la región del documento sobre la cual se realizará la transformación de perspectiva.

### 5️. Puntos ordenados

Ordenamiento de puntos de los vértices del documento.

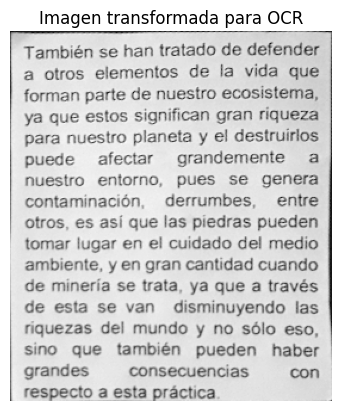
Resultado visual:

  
Se muestran los puntos clave que delimitan el área del documento, facilitando la posterior corrección de perspectiva.

### 6️. Imagen transformada para OCR

Transformación de perspectiva y recorte.

Resultado visual:

  
El documento queda alineado, lo que mejora significativamente la precisión del OCR.

### 7️. Texto OCR detectado

Ejecución de Tesseract OCR sobre la imagen transformada.

Resultado obtenido:

También se han tratado de defender

a otros elementos de la vida que

forman parte de nuestro ecosistema,

ya que estos significan gran riqueza

para nuestro planeta y el destruirlos

puede afectar grandemente a

nuestro entorno, pues se genera

contaminación, derrumbes, entre

otros, es así que las piedras pueden

tomar lugar en el cuidado del medio

ambiente, y en gran cantidad cuando

de minería se trata, ya que a través

de esta se van disminuyendo las

riquezas del mundo y no sólo eso,

sino que también puede haber

grandes consecuencias con

## respecto a esta práctica.

### 8. PDF obtenido

### Se obtiene el pdf al ejecutar el bloque final.



## Conclusiones

***Respecto al objetivo de implementar un sistema de OCR funcional en Google Colab:***  
Se logró desarrollar un flujo completo de procesamiento de imágenes y reconocimiento de texto utilizando Python, OpenCV y Tesseract, demostrando que es posible extraer información textual de documentos de manera precisa sin necesidad de hardware especializado como el ESP32-CAM.

***Respecto al objetivo de generar documentos PDF y Word automáticamente:***  
Se implementó exitosamente un sistema que, además de mostrar los resultados en pantalla, exporta el análisis completo en formatos PDF y Word, incluyendo imágenes de cada etapa del proceso y el texto detectado. Esto mejora la documentación y facilita la entrega de resultados de forma profesional.

***Respecto al objetivo de analizar cada etapa del procesamiento de imágenes:***  
Cada fase (conversión a escala de grises, detección de bordes, contornos, puntos de transformación y OCR) fue probada y documentada visualmente, lo que permite validar el funcionamiento del sistema y su eficiencia para mejorar la calidad de la lectura de caracteres.

***Respecto al objetivo de comparar con métodos alternativos (ESP32-CAM):***  
Se determinó que, aunque el ESP32-CAM es útil para capturar imágenes, su capacidad de procesamiento es limitada para realizar OCR localmente. Sin embargo, puede integrarse como un módulo de captura para enviar imágenes a un servidor con el flujo OCR desarrollado.

***Respecto a la escalabilidad y aplicabilidad del sistema:***  
El método implementado en Google Colab ofrece un entorno flexible, escalable y con herramientas potentes para tareas de OCR, lo que lo hace adecuado no solo para fines académicos, sino también para aplicaciones prácticas como la digitalización de documentos.

## Anexos

### Códigos fuente

Enlace para crear pdf (Google colab): ***https://colab.research.google.com/drive/16C6XYb4UqtZK79AekLFjmME5yLa7S-oC?usp=sharing***

Código completo del **OCR en PDF:**

!apt-get install -y tesseract-ocr

!pip install pytesseract opencv-python-headless numpy matplotlib

!apt-get install -y tesseract-ocr-spa

!pip install fpdf

# Subir imagen desde la PC

uploaded = files.upload()

# Cargar imagen

image = cv2.imread(list(uploaded.keys())[0])

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# Mostrar imagen original

plt.imshow(image\_rgb)

plt.title("Imagen original")

plt.axis('off')

plt.show()

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

plt.imshow(gray, cmap='gray')

plt.title("Escala de grises")

plt.axis('off')

plt.show()

canny = cv2.Canny(gray, 10, 150)

canny = cv2.dilate(canny, None, iterations=1)

plt.imshow(canny, cmap='gray')

plt.title("Detección de bordes (Canny)")

plt.axis('off')

plt.show()

cnts = cv2.findContours(canny, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)[0]

cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:1]

image\_contours = image.copy()

cv2.drawContours(image\_contours, cnts, -1, (0, 255, 0), 2)

plt.imshow(cv2.cvtColor(image\_contours, cv2.COLOR\_BGR2RGB))

plt.title("Contornos detectados")

plt.axis('off')

plt.show()

def ordenar\_puntos(puntos):

    n\_puntos = np.concatenate([puntos[0], puntos[1], puntos[2], puntos[3]]).tolist()

    y\_order = sorted(n\_puntos, key=lambda n\_puntos: n\_puntos[1])

    x1\_order = sorted(y\_order[:2], key=lambda x: x[0])

    x2\_order = sorted(y\_order[2:4], key=lambda x: x[0])

    return [x1\_order[0], x1\_order[1], x2\_order[0], x2\_order[1]]

for c in cnts:

    epsilon = 0.01 \* cv2.arcLength(c, True)

    approx = cv2.approxPolyDP(c, epsilon, True)

    if len(approx) == 4:

        puntos = ordenar\_puntos(approx)

        # Dibujar puntos

        image\_points = image.copy()

        colores = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0)]

        for i, punto in enumerate(puntos):

            cv2.circle(image\_points, tuple(punto), 7, colores[i], 2)

        plt.imshow(cv2.cvtColor(image\_points, cv2.COLOR\_BGR2RGB))

        plt.title("Puntos ordenados")

        plt.axis('off')

        plt.show()

        # Transformación de perspectiva

        pts1 = np.float32(puntos)

        pts2 = np.float32([[0, 0], [270, 0], [0, 310], [270, 310]])

        M = cv2.getPerspectiveTransform(pts1, pts2)

        dst = cv2.warpPerspective(gray, M, (270, 310))

        plt.imshow(dst, cmap='gray')

        plt.title("Imagen transformada para OCR")

        plt.axis('off')

        plt.show()

        # OCR

        texto = pytesseract.image\_to\_string(dst, lang='spa')

        print("Texto detectado:", texto)

pdf = FPDF()

pdf.set\_auto\_page\_break(auto=True, margin=15)

# Imagen original

pdf.add\_page()

pdf.set\_font("Arial", size=12)

pdf.cell(200, 10, "Imagen original", ln=True)

plt.imsave("original.png", image\_rgb)

pdf.image("original.png", x=10, y=30, w=100)

# Imagen en grises

pdf.add\_page()

pdf.cell(200, 10, "Escala de grises", ln=True)

plt.imsave("grises.png", gray, cmap='gray')

pdf.image("grises.png", x=10, y=30, w=100)

# Bordes

pdf.add\_page()

pdf.cell(200, 10, "Detección de bordes", ln=True)

plt.imsave("bordes.png", canny, cmap='gray')

pdf.image("bordes.png", x=10, y=30, w=100)

# Contornos

pdf.add\_page()

pdf.cell(200, 10, "Contornos detectados", ln=True)

plt.imsave("contornos.png", cv2.cvtColor(image\_contours, cv2.COLOR\_BGR2RGB))

pdf.image("contornos.png", x=10, y=30, w=100)

# Texto OCR

pdf.add\_page()

pdf.cell(200, 10, "Texto OCR detectado", ln=True)

pdf.multi\_cell(190, 10, texto)

pdf.output("resultado\_OCR.pdf")

files.download("resultado\_OCR.pdf")

Enlace para crear pdf y word (Google colab):

***https://colab.research.google.com/drive/1SlrURBveAjiwb5zDNHhXGywKKXa\_brYi?usp=sharing***

Código completo del **OCR en PDF y Word:**

!apt-get install -y tesseract-ocr tesseract-ocr-spa

!pip install pytesseract opencv-python-headless numpy matplotlib fpdf python-docx

import cv2

import numpy as np

import pytesseract

import matplotlib.pyplot as plt

from google.colab import files

from fpdf import FPDF

from docx import Document

from docx.shared import Inches

import os

pytesseract.pytesseract.tesseract\_cmd = r'/usr/bin/tesseract'

# Subir imagen

uploaded = files.upload()

image\_path = list(uploaded.keys())[0]

image = cv2.imread(image\_path)

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# Guardar imagen original

plt.imsave("original.png", image\_rgb)

# Escala de grises

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

plt.imsave("grises.png", gray, cmap='gray')

# Bordes

canny = cv2.Canny(gray, 10, 150)

canny = cv2.dilate(canny, None, iterations=1)

plt.imsave("bordes.png", canny, cmap='gray')

# Contornos

cnts = cv2.findContours(canny, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)[0]

cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:1]

image\_contours = image.copy()

cv2.drawContours(image\_contours, cnts, -1, (0, 255, 0), 2)

plt.imsave("contornos.png", cv2.cvtColor(image\_contours, cv2.COLOR\_BGR2RGB))

# Función para ordenar puntos

def ordenar\_puntos(puntos):

    n\_puntos = np.concatenate([puntos[0], puntos[1], puntos[2], puntos[3]]).tolist()

    y\_order = sorted(n\_puntos, key=lambda n: n[1])

    x1\_order = sorted(y\_order[:2], key=lambda x: x[0])

    x2\_order = sorted(y\_order[2:4], key=lambda x: x[0])

    return [x1\_order[0], x1\_order[1], x2\_order[0], x2\_order[1]]

# Puntos y OCR

texto = "[No se detectó texto]"

for c in cnts:

    epsilon = 0.01 \* cv2.arcLength(c, True)

    approx = cv2.approxPolyDP(c, epsilon, True)

    if len(approx) == 4:

        puntos = ordenar\_puntos(approx)

        # Dibujar puntos

        image\_points = image.copy()

        colores = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0)]

        for i, punto in enumerate(puntos):

            cv2.circle(image\_points, tuple(punto), 7, colores[i], 2)

        plt.imsave("puntos.png", cv2.cvtColor(image\_points, cv2.COLOR\_BGR2RGB))

        # Transformación de perspectiva

        pts1 = np.float32(puntos)

        pts2 = np.float32([[0, 0], [270, 0], [0, 310], [270, 310]])

        M = cv2.getPerspectiveTransform(pts1, pts2)

        dst = cv2.warpPerspective(gray, M, (270, 310))

        plt.imsave("transformada.png", dst, cmap='gray')

        # OCR

        texto = pytesseract.image\_to\_string(dst, lang='spa')

# Lista de imágenes y títulos

imagenes = [

    ("original.png", "Imagen original"),

    ("grises.png", "Escala de grises"),

    ("bordes.png", "Detección de bordes"),

    ("contornos.png", "Contornos detectados"),

    ("puntos.png", "Puntos ordenados"),

    ("transformada.png", "Imagen transformada para OCR")

]

# === PDF ===

pdf = FPDF()

pdf.set\_auto\_page\_break(auto=True, margin=15)

for archivo, titulo in imagenes:

    if os.path.exists(archivo):

        pdf.add\_page()

        pdf.set\_font("Arial", size=12)

        pdf.cell(200, 10, titulo, ln=True)

        pdf.image(archivo, x=10, y=30, w=100)

pdf.add\_page()

pdf.set\_font("Arial", size=12)

pdf.cell(200, 10, "Texto OCR detectado", ln=True)

pdf.multi\_cell(190, 10, texto)

pdf.output("resultado\_OCR.pdf")

files.download("resultado\_OCR.pdf")

# === WORD ===

document = Document()

document.add\_heading('Resultados de OCR paso a paso', level=1)

for archivo, titulo in imagenes:

    if os.path.exists(archivo):

        document.add\_heading(titulo, level=2)

        document.add\_picture(archivo, width=Inches(4))

# Limpiar el texto OCR para evitar caracteres inválidos

texto\_limpio = ''.join(ch for ch in texto if ch.isprintable())

document.add\_heading("Texto OCR detectado", level=2)

document.add\_paragraph(texto\_limpio)

document.save("resultado\_OCR.docx")

files.download("resultado\_OCR.docx")