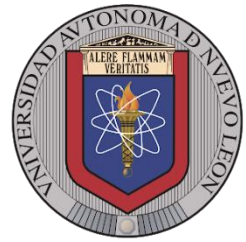


UANL

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

FIME

Nombre: Roberto Erick Aguilar Morales

Matricula: 1871004

Carrera: Ingeniero en Tecnologías del Software

6.- Líneas Rectas

Materia: VISION COMPUTACIONAL LABORATORIO

Docente: RAYMUNDO SAID ZAMORA PEQUEÑO

Hora: N1-N2 **Días: Miércoles**

Fecha: 24/11/2024

Objetivo

Identificar las líneas rectas en una imagen utilizando el método RANSAC y la transformada de Hough

Marco teórico

Procesamiento de Imágenes

El procesamiento de imágenes es una disciplina que se ocupa de la manipulación y análisis de imágenes digitales a través de computadoras. Se emplea en diversas áreas, desde la medicina y la astronomía, hasta la inteligencia artificial y el control de calidad industrial. Dentro de esta área, el propósito principal es extraer información útil de las imágenes o transformarlas en representaciones más adecuadas para su análisis.

El primer paso en el procesamiento de imágenes digitales suele ser la conversión de la imagen a escala de grises, lo que simplifica el análisis y permite un tratamiento más eficiente de los píxeles. Las imágenes en escala de grises son representaciones bidimensionales donde cada píxel tiene un valor de intensidad que oscila entre 0 (negro) y 255 (blanco).

Segmentación de Imágenes

La segmentación busca dividir una imagen en regiones homogéneas. En este caso:

- **Vecindarios homogéneos:** Son áreas donde los píxeles tienen valores de intensidad similares, definidos por un rango (±50 en el programa).
- **Bordes:** Son las fronteras entre diferentes vecindarios, donde ocurre un cambio significativo en la intensidad.

Métodos de Detección de Bordes

Los bordes son transiciones rápidas en la intensidad de los píxeles. En este programa, los bordes se utilizan como base para detectar líneas rectas. Los bordes detectados son procesados mediante técnicas geométricas.

Transformada de Hough Probabilística

La Transformada de Hough es una técnica popular para detectar líneas rectas en una imagen. Funciona transformando puntos en una imagen al espacio paramétrico donde se representan líneas posibles.

- Ventajas: Robusta ante ruido y discontinuidades en las líneas.
- Versión probabilística (HoughLinesP): Optimiza la técnica clásica considerando solo un subconjunto de puntos para mejorar la eficiencia.

Parámetros clave:

- rho: Resolución en píxeles para la distancia radial.
- theta: Resolución angular (en radianes).
- threshold: Mínimo número de puntos que deben estar alineados para considerar una línea.
- minLineLength: Longitud mínima para que se detecte una línea.
- maxLineGap: Máxima distancia entre dos puntos alineados.

RANSAC (Random Sample Consensus)

RANSAC es un algoritmo iterativo robusto para ajustar modelos a datos con ruido o con valores atípicos (outliers).

Proceso básico:

1. Selección aleatoria de un subconjunto de puntos (muestra).
2. Ajuste de un modelo (por ejemplo, una línea) con la muestra seleccionada.
3. Evaluación del modelo en el conjunto completo, clasificando puntos como inliers (se ajustan al modelo) o outliers.
4. Repetición hasta encontrar el modelo con mayor número de inliers.

En este programa, RANSAC es usado para detectar varias líneas en los puntos detectados, eliminando las líneas ajustadas tras cada iteración.

Introducción

El presente trabajo describe un programa que combina técnicas avanzadas de procesamiento de imágenes para identificar vecindarios homogéneos en una imagen y detectar líneas rectas presentes en ella. Para lograrlo, se emplean algoritmos robustos como RANSAC (Random Sample Consensus) y la Transformada de Hough Probabilística, herramientas ampliamente reconocidas por su capacidad para trabajar con datos ruidosos y estructuras geométricas.

Este código detecta múltiples líneas en un conjunto de puntos usando el algoritmo RANSAC iterativamente. Cada iteración identifica una línea ajustada a un subconjunto de puntos (inliers) mientras ignora los valores atípicos (outliers). Los inliers detectados se eliminan, y el proceso continúa hasta que haya pocos puntos o se alcance el límite de iteraciones. Las líneas encontradas se almacenan y se devuelven como resultado.

```
# Función para aplicar RANSAC en múltiples iteraciones
def detectar_varias_lineas_ransac(puntos_X, puntos_Y, min_puntos=5,
max_iteraciones=50):
    lineas_ransac = []
    iteracion = 0

    while len(puntos_X) > min_puntos and iteracion < max_iteraciones:
        # Aplicar RANSAC para detectar una línea
        ransac = RANSACRegressor()
        ransac.fit(puntos_X, puntos_Y)

        # Obtener los inliers, es decir, los puntos que se ajustan bien a la
línea
        inlier_mask = ransac.inlier_mask_

        # Extraer los puntos que corresponden a la línea detectada
        puntos_inliers_X = puntos_X[inlier_mask]
        puntos_inliers_Y = puntos_Y[inlier_mask]

        # Guardar los puntos de la línea detectada
        lineas_ransac.append((puntos_inliers_X, puntos_inliers_Y))

        # Eliminar los inliers de la lista de puntos (quitar la línea ya
detectada)
        puntos_X = puntos_X[~inlier_mask]
        puntos_Y = puntos_Y[~inlier_mask]

        iteracion += 1
```

```
    return lineas_ransac
```

Este código identifica vecindarios de píxeles en una imagen basándose en un rango de valores. Los vecindarios agrupan píxeles cercanos cuyo valor está dentro de un rango definido alrededor de un valor pivote. También detecta los bordes de cada vecindario, es decir, píxeles adyacentes que no cumplen con el criterio del rango. El proceso usa una cola para explorar píxeles vecinos y marca los visitados para evitar procesarlos más de una vez. Al final, devuelve la lista de vecindarios y sus respectivos bordes.

```
# Función para detectar vecindarios en rango de +50 y -50 (ya existente)
def detectar_vecindarios(image, rango=50):
    rows, cols = image.shape
    visitado = np.zeros((rows, cols), dtype=bool)
    vecindarios = []
    bordes_vecindarios = []

    def obtener_vecindario(r, c, pivote_valor, rango):
        vecindario = [(r, c)]
        borde = set()
        cola = [(r, c)]
        visitado[r, c] = True
        pivote_valor = int(pivote_valor)

        while cola:
            x, y = cola.pop(0)
            for dx in [-1, 0, 1]:
                for dy in [-1, 0, 1]:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < rows and 0 <= ny < cols and not
visitado[nx, ny]:
                        if pivote_valor - rango <= int(image[nx, ny]) <=
pivote_valor + rango:
                            visitado[nx, ny] = True
                            vecindario.append((nx, ny))
                            cola.append((nx, ny))
                        else:
                            borde.add((nx, ny))
        return vecindario, borde

    for r in range(rows):
        for c in range(cols):
            if not visitado[r, c]:
```

```
                pivote_valor = image[r, c]
                vecindario, borde = obtener_vecindario(r, c, pivote_valor,
rango)
                vecindarios.append(vecindario)
                bordes_vecindarios.append(borde)

    return vecindarios, bordes_vecindarios
```

Este código identifica vecindarios de píxeles en una imagen basándose en un rango de valores. Los vecindarios agrupan píxeles cercanos cuyo valor está dentro de un rango definido alrededor de un valor pivote. También detecta los bordes de cada vecindario, es decir, píxeles adyacentes que no cumplen con el criterio del rango. El proceso usa una cola para explorar píxeles vecinos y marca los visitados para evitar procesarlos más de una vez. Al final, devuelve la lista de vecindarios y sus respectivos bordes.

```
# Función para detectar vecindarios en rango de +50 y -50 (ya existente)
def detectar_vecindarios(image, rango=50):
    rows, cols = image.shape
    visitado = np.zeros((rows, cols), dtype=bool)
    vecindarios = []
    bordes_vecindarios = []

    def obtener_vecindario(r, c, pivote_valor, rango):
        vecindario = [(r, c)]
        borde = set()
        cola = [(r, c)]
        visitado[r, c] = True
        pivote_valor = int(pivote_valor)

        while cola:
            x, y = cola.pop(0)
            for dx in [-1, 0, 1]:
                for dy in [-1, 0, 1]:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < rows and 0 <= ny < cols and not
visitado[nx, ny]:
                        if pivote_valor - rango <= int(image[nx, ny]) <=
pivote_valor + rango:
                            visitado[nx, ny] = True
                            vecindario.append((nx, ny))
                            cola.append((nx, ny))
                        else:
                            borde.add((nx, ny))
        return vecindario, borde
```

```
    for r in range(rows):
        for c in range(cols):
            if not visitado[r, c]:
                pivote_valor = image[r, c]
                vecindario, borde = obtener_vecindario(r, c, pivote_valor,
rango)
                vecindarios.append(vecindario)
                bordes_vecindarios.append(borde)

    return vecindarios, bordes_vecindarios
```

Este código carga una imagen, la convierte a escala de grises y luego utiliza la función detectar_vecindarios para identificar grupos de píxeles similares (vecindarios) y los bordes entre ellos. A continuación, se muestra cuántos vecindarios se encontraron. Para visualizar los resultados, colorea cada vecindario de manera diferente en una imagen de salida. También genera una imagen binaria donde los bordes detectados se marcan en blanco.

```
# Cargar imagen
imagen_a_color = cv2.imread('imagen.png')
if imagen_a_color is None:
    print("Error: No se pudo cargar la imagen.")
else:
    # Convertirla a escala de grises
    imagen = cv2.cvtColor(imagen_a_color, cv2.COLOR_BGR2GRAY)

    # Detectar vecindarios y bordes
    vecindarios, bordes_vecindarios = detectar_vecindarios(imagen)

    # Mostrar cuántos vecindarios se encontraron
    print(f"Número de vecindarios encontrados: {len(vecindarios)}")

    # Para visualizar los vecindarios en la imagen (opcional)
    output_image = np.zeros_like(imagen)
    for i, vecindario in enumerate(vecindarios):
        color_value = (255 - (i * 25)) % 256  # Asegurar que los valores
estén entre 0 y 255
        for (r, c) in vecindario:
            output_image[r, c] = color_value  # Colorear cada vecindario de
manera diferente
    # Crear imagen binaria para los bordes
    bordes_imagen = np.zeros_like(imagen)

    # Marcar los píxeles de borde en la imagen binaria
```

```
    for borde in bordes_vecindarios:
        for (r, c) in borde:
            bordes_imagen[r, c] = 255  # Borde en blanco
```

Este código aplica la Transformada de Hough sobre una imagen binarizada (de bordes) para detectar líneas rectas. Se especifican parámetros como el umbral de votos, la longitud mínima de línea y la separación máxima entre puntos de una línea. Luego, dibuja las líneas detectadas en la imagen original y crea imágenes separadas para mostrar solo las líneas, los bordes de la imagen y los bordes restantes. Finalmente, guarda las imágenes procesadas y las matrices correspondientes en archivos CSV.

```
# Aplicar la Transformada de Hough directamente sobre la imagen binarizada
de bordes
    # threshold: Decide que el minimo de votos que debe tener para
detectarse como linea recta
    # minLineLength: La minima cantidad de pixeles para poder considerarse
linea recta
    # masLineGap: maxima separacion permitida entre dos puntos de una linea
    lineas = cv2.HoughLinesP(bordes_imagen, rho=1, theta=np.pi / 180,
threshold=50, minLineLength=2, maxLineGap=10)

    solo_lineas = np.zeros_like(bordes_imagen)
    bordes_restantes = np.copy(bordes_imagen)

    if lineas is not None:
        for linea in lineas:
            for x1, y1, x2, y2 in linea:
                cv2.line(imagen_a_color, (x1, y1), (x2, y2), (0, 255, 0),
2)  # Dibuja cada línea con color verde
                cv2.line(solo_lineas, (x1, y1), (x2, y2), (255), 2)  #
Dibuja cada línea con color verde
                cv2.line(bordes_restantes, (x1, y1), (x2, y2), (0, 255, 0),
2)  # Quita las lineas rectas

    # Mostrar la imagen con las líneas detectadas
    cv2.imshow('Vecindarios', output_image)
    cv2.imshow("Bordes de la imagen", bordes_imagen)
    cv2.imshow('Lineas rectas sobrepuestas', imagen_a_color)
    cv2.imshow('Solo lineas rectas', solo_lineas)
    cv2.imshow("Bordes restantes de la imagen", bordes_restantes)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Guardar las imagenes
```

```
    cv2.imwrite('Imagen Vecindarios.png', output_image)
    cv2.imwrite("Imagen Bordes de la imagen.png", bordes_imagen)
    cv2.imwrite('Imagen Lineas rectas sobrepuestas.png', imagen_a_color)
    cv2.imwrite('Imagen Solo lineas rectas.png', solo_lineas)
    cv2.imwrite("Imagen Bordes restantes de la imagen.png",
bordes_restantes)

    # Guardar la matriz de la imagen en escala de grises en un archivo CSV
    np.savetxt('Matriz imagen_gris.csv', imagen, delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen.csv'.")

    # Guardar la matriz de la imagen vecindarios en un archivo CSV
    np.savetxt('Matriz Vecindarios.csv', output_image, delimiter=',',
fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen_vecindarios.csv'.")

    # Guardar la matriz de la imagen Bordes de la imagen en un archivo CSV
    np.savetxt('Matriz Bordes de la imagen.csv', bordes_imagen,
delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen_vecindarios.csv'.")

    # Guardar la matriz de la imagen Solo lineas rectas en un archivo CSV
    np.savetxt('Matriz Solo lineas rectas.csv', solo_lineas, delimiter=',',
fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen_vecindarios.csv'.")

    # Guardar la matriz de la imagen Bordes restantes de la imagen en un
archivo CSV
    np.savetxt('Matriz Bordes restantes de la imagen.csv', bordes_restantes,
delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen_vecindarios.csv'.")
```

Cálculos y Resultados

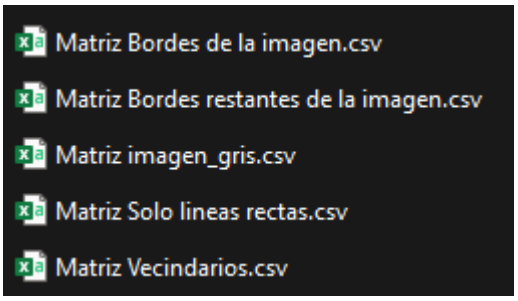
P06 Lineas rectas.py

```
import cv2
import numpy as np
from sklearn.linear_model import RANSACRegressor

# Función para aplicar RANSAC en múltiples iteraciones
def detectar_varias_lineas_ransac(puntos_X, puntos_Y, min_puntos=5,
max_iteraciones=50):
    lineas_ransac = []
    iteracion = 0

    while len(puntos_X) > min_puntos and iteracion < max_iteraciones:
        # Aplicar RANSAC para detectar una línea
        ransac = RANSACRegressor()
        ransac.fit(puntos_X, puntos_Y)

        # Obtener los inliers, es decir, los puntos que se ajustan bien a la
línea
        inlier_mask = ransac.inlier_mask_

        # Extraer los puntos que corresponden a la línea detectada
        puntos_inliers_X = puntos_X[inlier_mask]
        puntos_inliers_Y = puntos_Y[inlier_mask]

        # Guardar los puntos de la línea detectada
        lineas_ransac.append((puntos_inliers_X, puntos_inliers_Y))

        # Eliminar los inliers de la lista de puntos (quitar la línea ya
detectada)
        puntos_X = puntos_X[~inlier_mask]
        puntos_Y = puntos_Y[~inlier_mask]

        iteracion += 1

    return lineas_ransac

# Función para detectar vecindarios en rango de +50 y -50 (ya existente)
def detectar_vecindarios(image, rango=50):
    rows, cols = image.shape
    visitado = np.zeros((rows, cols), dtype=bool)
    vecindarios = []
    bordes_vecindarios = []
```

```
    def obtener_vecindario(r, c, pivote_valor, rango):
        vecindario = [(r, c)]
        borde = set()
        cola = [(r, c)]
        visitado[r, c] = True
        pivote_valor = int(pivote_valor)

        while cola:
            x, y = cola.pop(0)
            for dx in [-1, 0, 1]:
                for dy in [-1, 0, 1]:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < rows and 0 <= ny < cols and not
visitado[nx, ny]:
                        if pivote_valor - rango <= int(image[nx, ny]) <=
pivote_valor + rango:
                            visitado[nx, ny] = True
                            vecindario.append((nx, ny))
                            cola.append((nx, ny))
                        else:
                            borde.add((nx, ny))
        return vecindario, borde

    for r in range(rows):
        for c in range(cols):
            if not visitado[r, c]:
                pivote_valor = image[r, c]
                vecindario, borde = obtener_vecindario(r, c, pivote_valor,
rango)
                vecindarios.append(vecindario)
                bordes_vecindarios.append(borde)

    return vecindarios, bordes_vecindarios

# Cargar imagen
imagen_a_color = cv2.imread('imagen.png')
if imagen_a_color is None:
    print("Error: No se pudo cargar la imagen.")
else:
    # Convertirla a escala de grises
    imagen = cv2.cvtColor(imagen_a_color, cv2.COLOR_BGR2GRAY)

    # Detectar vecindarios y bordes
    vecindarios, bordes_vecindarios = detectar_vecindarios(imagen)
```

```
    # Mostrar cuántos vecindarios se encontraron
    print(f"Número de vecindarios encontrados: {len(vecindarios)}")

    # Para visualizar los vecindarios en la imagen (opcional)
    output_image = np.zeros_like(imagen)
    for i, vecindario in enumerate(vecindarios):
        color_value = (255 - (i * 25)) % 256  # Asegurar que los valores
estén entre 0 y 255
        for (r, c) in vecindario:
            output_image[r, c] = color_value  # Colorear cada vecindario de
manera diferente
    # Crear imagen binaria para los bordes
    bordes_imagen = np.zeros_like(imagen)

    # Marcar los píxeles de borde en la imagen binaria
    for borde in bordes_vecindarios:
        for (r, c) in borde:
            bordes_imagen[r, c] = 255  # Borde en blanco

    # Aplicar la Transformada de Hough directamente sobre la imagen
binarizada de bordes
    # threshold: Decide que el minimo de votos que debe tener para
detectarse como linea recta
    # minLineLength: La minima cantidad de pixeles para poder considerarse
linea recta
    # masLineGap: maxima separacion permitida entre dos puntos de una linea
    lineas = cv2.HoughLinesP(bordes_imagen, rho=1, theta=np.pi / 180,
threshold=50, minLineLength=2, maxLineGap=10)

    solo_lineas = np.zeros_like(bordes_imagen)
    bordes_restantes = np.copy(bordes_imagen)

    if lineas is not None:
        for linea in lineas:
            for x1, y1, x2, y2 in linea:
                cv2.line(imagen_a_color, (x1, y1), (x2, y2), (0, 255, 0),
2)  # Dibuja cada línea con color verde
                cv2.line(solo_lineas, (x1, y1), (x2, y2), (255), 2)  #
Dibuja cada línea con color verde
                cv2.line(bordes_restantes, (x1, y1), (x2, y2), (0, 255, 0),
2)  # Quita las lineas rectas

    # Mostrar la imagen con las líneas detectadas
    cv2.imshow('Vecindarios', output_image)
    cv2.imshow("Bordes de la imagen", bordes_imagen)
```

```
    cv2.imshow('Lineas rectas sobrepuestas', imagen_a_color)
    cv2.imshow('Solo lineas rectas', solo_lineas)
    cv2.imshow("Bordes restantes de la imagen", bordes_restantes)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Guardar las imagenes
    cv2.imwrite('Imagen Vecindarios.png', output_image)
    cv2.imwrite("Imagen Bordes de la imagen.png", bordes_imagen)
    cv2.imwrite('Imagen Lineas rectas sobrepuestas.png', imagen_a_color)
    cv2.imwrite('Imagen Solo lineas rectas.png', solo_lineas)
    cv2.imwrite("Imagen Bordes restantes de la imagen.png",
bordes_restantes)

    # Guardar la matriz de la imagen en escala de grises en un archivo CSV
    np.savetxt('Matriz imagen_gris.csv', imagen, delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen.csv'.")

    # Guardar la matriz de la imagen vecindarios en un archivo CSV
    np.savetxt('Matriz Vecindarios.csv', output_image, delimiter=',',
fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen_vecindarios.csv'.")

    # Guardar la matriz de la imagen Bordes de la imagen en un archivo CSV
    np.savetxt('Matriz Bordes de la imagen.csv', bordes_imagen,
delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen_vecindarios.csv'.")

    # Guardar la matriz de la imagen Solo lineas rectas en un archivo CSV
    np.savetxt('Matriz Solo lineas rectas.csv', solo_lineas, delimiter=',',
fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen_vecindarios.csv'.")

    # Guardar la matriz de la imagen Bordes restantes de la imagen en un
archivo CSV
    np.savetxt('Matriz Bordes restantes de la imagen.csv', bordes_restantes,
delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen_vecindarios.csv'.")
```



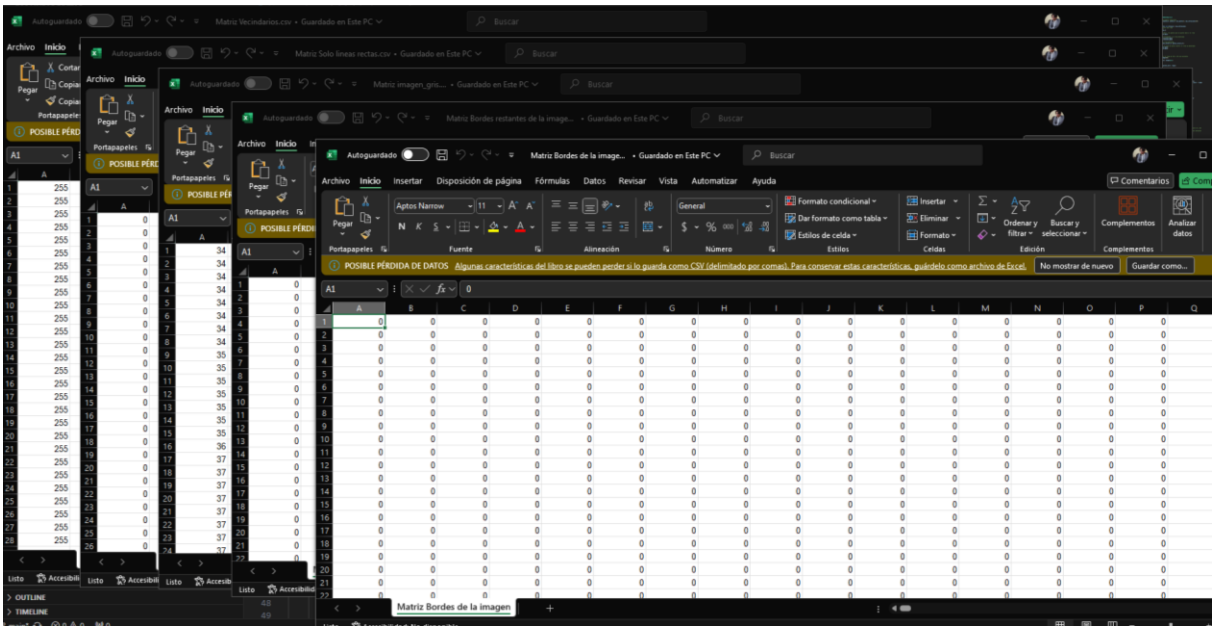


Imagen original

Imagen vecindarios.png

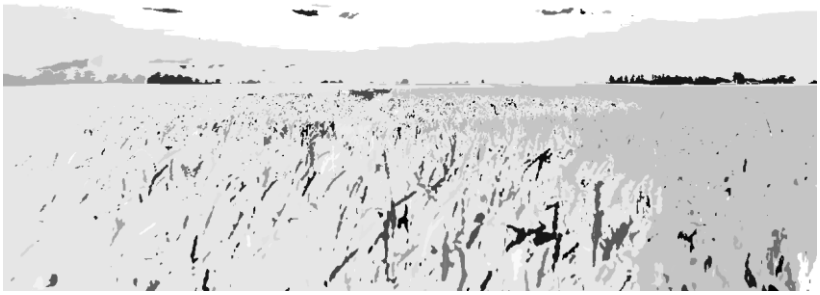


Imagen bordes de la imagen.jpg

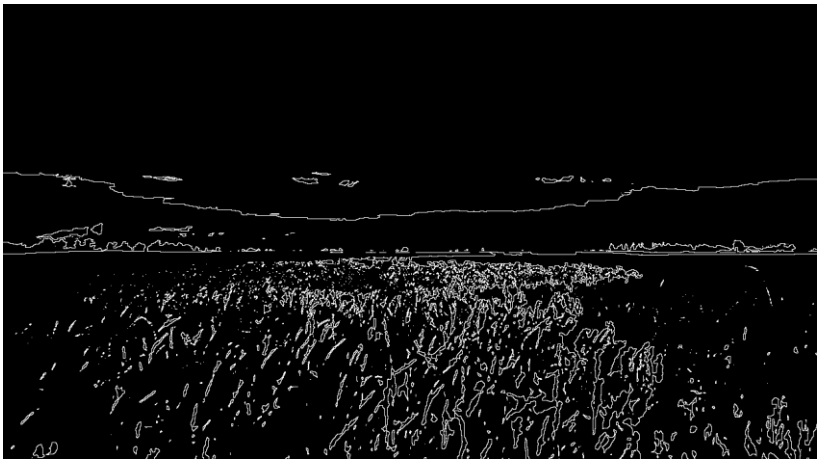


Imagen líneas rectas sobrepuestas.jpg

Imagen solo líneas rectas.jpg

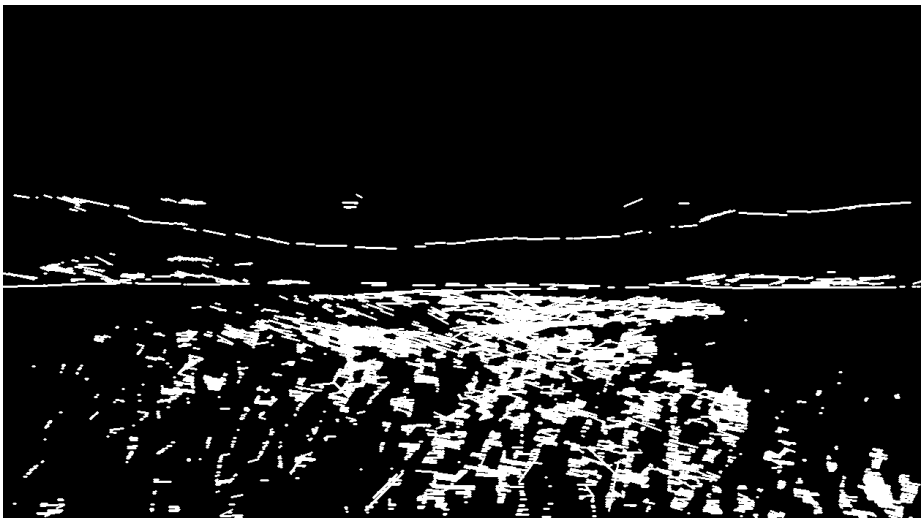


Imagen bordes restantes de la imagen.jpg

Conclusiones

El programa desarrollado ilustra cómo integrar técnicas avanzadas de procesamiento de imágenes para segmentar regiones homogéneas y detectar líneas rectas de manera precisa en imágenes digitales. Al emplear algoritmos robustos como la Transformada de Hough Probabilística y RANSAC, se logró identificar patrones geométricos con alta efectividad, incluso frente a desafíos como el ruido en la imagen o pequeñas discontinuidades en los datos.

Además, el enfoque implementado no solo permite la detección de líneas rectas, sino que también garantiza que estas cumplan con criterios de continuidad, descartando aquellas interrumpidas o incompletas. Esta capacidad es particularmente útil en aplicaciones donde la precisión en la identificación de estructuras geométricas es crítica, como en visión computacional, análisis de patrones, diseño asistido por computadora (CAD) y sistemas de navegación.

Bibliografía

- Gonzalez, R. C., & Woods, R. E. (2002). *Digital image processing* (2nd ed.). Pearson Prentice Hall.
- Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media.
- Meyer, F. (2001). Topographic distance and watershed lines. *Signal Processing*, 38(1), 113-125. https://doi.org/10.1016/S0165-1684(01)00064-4
- OpenCV Documentation. (n.d.). *OpenCV documentation*. OpenCV. Retrieved from https://docs.opencv.org/