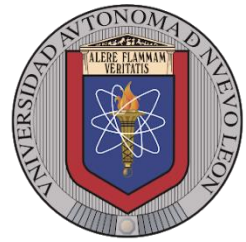


UANL

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

8.– Elipses

Materia: VISION COMPUTACIONAL LABORATORIO

Docente: RAYMUNDO SAID ZAMORA PEQUEÑO

Hora: N1-N2 **Días: Miércoles**

Fecha: 24/11/24

Objetivo

Identificar y clasificar patrones geométricos como líneas, círculos y elipses en una imagen y visualizarlos de forma separada para su análisis detallado.

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).

Método de Cuerda Tangente

El método de cuerda tangente es una técnica utilizada en geometría computacional y en el análisis de curvas para identificar propiedades de figuras geométricas, como circunferencias, y determinar puntos críticos de tangencia entre ellas. En este contexto, el método de cuerda tangente podría ser interpretado como un enfoque para identificar y verificar las características de los círculos detectados a partir de las líneas conectadas. Esto se puede realizar ajustando las "líneas tangentes" a las circunferencias aproximadas detectadas y verificando su concordancia con los bordes en la imagen.

Este método se basa en calcular puntos tangenciales a la curva del óvalo mediante el uso de líneas secantes (cuerdas) que se aproximan gradualmente hasta convertirse en tangentes. Esto se logra al evaluar puntos consecutivos en el contorno del óvalo y analizar el cambio en la pendiente entre ellos. A medida que la longitud de la cuerda disminuye, las líneas se aproximan a la dirección tangente de la curva en un punto específico. Esta información es crucial para describir la geometría y orientación del óvalo, permitiendo detectar características como ejes principales, puntos de inflexión o curvaturas significativas, que son útiles en aplicaciones como reconocimiento de patrones, visión artificial y procesamiento de imágenes.

Detección de Bordes, Líneas y Círculos

La detección de bordes es una técnica fundamental en el procesamiento de imágenes, ya que permite identificar las fronteras entre diferentes regiones de una imagen. Se utiliza bordes detectados para realizar un análisis más detallado. Uno de los algoritmos comunes para detectar bordes es el Detector de Canny, el cual identifica áreas de cambio abrupto en la intensidad de los píxeles, señalando bordes o límites de objetos en la imagen**.**

Se implementa la detección de líneas utilizando Hough Transform, un método comúnmente utilizado para encontrar líneas rectas en imágenes. Este enfoque transforma el espacio de coordenadas cartesianas en el espacio polar, permitiendo identificar líneas rectas incluso cuando los bordes no están completamente conectados.

El programa también implementa el algoritmo RANSAC, que es utilizado para detectar modelos geométricos, como líneas, en conjuntos de datos con ruido. En este contexto, RANSAC se emplea para ajustar líneas a los puntos de borde detectados en la imagen. Este enfoque es robusto frente a los outliers, permitiendo la detección de líneas más precisas y resistentes a ruido visual que pudiera afectar los bordes de la imagen.

Introducción

En el presente se muestra el procesamiento avanzado de imágenes utilizando técnicas de visión por computadora y aprendizaje automático. Este programa tiene como objetivo principal identificar elementos geométricos como líneas, círculos y elipses a partir de una imagen de entrada, mediante el uso de algoritmos como la transformada de Hough, la segmentación por vecindarios y el ajuste elíptico. El flujo general del programa incluye la detección de bordes en la imagen, la identificación de vecindarios de píxeles con valores similares, la extracción de líneas rectas, y la detección de círculos y elipses. Posteriormente, estos elementos se representan visualmente en imágenes procesadas y se guardan tanto en formatos gráficos como en matrices para su análisis posterior.

Desarrollo

Inicialmente, la imagen es convertida a escala de grises, tras lo cual se segmentan vecindarios mediante análisis de intensidad utilizando un rango definido. Esto permite identificar áreas homogéneas dentro de la imagen, a las cuales se les extraen bordes. Con los bordes detectados, se aplican algoritmos para identificar líneas rectas, eliminándolas progresivamente de los bordes restantes. Este

proceso permite aislar componentes significativos que posteriormente servirán para la detección de características más complejas.

Una vez eliminadas las líneas recta y a partir de estos bordes restantes, se realiza un análisis más detallado para localizar y delinear círculos mediante aproximaciones geométricas que consideran la conectividad y la disposición espacial de los píxeles, con los círculos detectados, se procesan los bordes restantes para encontrar contornos con suficiente complejidad como para ajustar elipses. Este paso implica analizar contornos específicos con al menos 160 puntos, permitiendo determinar las características de cada elipse mediante un ajuste geométrico preciso. Las elipses detectadas se superponen sobre la imagen original, completando un análisis exhaustivo de las formas presentes en la imagen. Cada etapa del proceso se valida visualmente mediante la generación de imágenes intermedias que resaltan los resultados parciales (vecindarios, líneas, círculos y elipses).

Cálculos y Resultados

P08 Ovalos.py

El código comienza con una función que utiliza una búsqueda en profundidad (DFS) para identificar líneas conectadas en una imagen binaria de bordes. Cada línea detectada se almacena como una imagen independiente, lo que facilita su análisis posterior. El enfoque consiste en recorrer los píxeles blancos (valor 255) y marcar los píxeles visitados mientras se extiende en las direcciones cardinales (arriba, abajo, izquierda y derecha).

```
def detectar_lineas_conectadas(bordes_restantes):
    # Crear una copia para ir eliminando líneas una por una
    bordes_temp = np.copy(bordes_restantes)
    lineas_detectadas = []

    # Dimensiones de la imagen
    rows, cols = bordes_temp.shape

    # Función para seguir una línea conectada usando una búsqueda en
profundidad (DFS)
    def seguir_linea(x, y):
        stack = [(x, y)]
        linea_actual = np.zeros_like(bordes_temp)  # Imagen para la línea
actual
        while stack:
            cx, cy = stack.pop()
```

```
            if 0 <= cx < cols and 0 <= cy < rows and bordes_temp[cy, cx] ==
255:
                # Marcar el píxel en la imagen de la línea actual y en
bordes_temp
                linea_actual[cy, cx] = 255
                bordes_temp[cy, cx] = 0
                # Agregar píxeles vecinos (4 direcciones) a la pila
                for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
                    nx, ny = cx + dx, cy + dy
                    stack.append((nx, ny))
        return linea_actual

    # Recorrer la imagen para encontrar cada línea
    for y in range(rows):
        for x in range(cols):
            if bordes_temp[y, x] == 255:
                linea_actual = seguir_linea(x, y)
                lineas_detectadas.append(linea_actual)

    return lineas_detectadas
```

Esta parte del código evalúa si un centro potencial de una forma cerrada cumple ciertos criterios geométricos. Para esto, traza líneas en 8 direcciones alrededor del centro candidato, verificando cuántas de ellas intersectan con bordes en la imagen. Si al menos tres líneas tocan un borde, el punto es considerado un candidato válido para el centro de un círculo.

```
def verificar_centro_candidato(linea_detectada, centro_x, centro_y):
    # Dimensiones de la imagen
    rows, cols = linea_detectada.shape

    # Definir las 8 direcciones de los píxeles
    direcciones = [
        (-1, 0), (1, 0), (0, -1), (0, 1),
        (-1, -1), (-1, 1), (1, -1), (1, 1)
    ]

    # Contador de líneas que tocan un borde
    contador_lineas_que_toquen_borde = 0

    # Crear una copia de la imagen para dibujar las líneas de prueba
    imagen_lineas = linea_detectada.copy()

    # Verificar cada dirección y dibujar las líneas de prueba
    for dx, dy in direcciones:
        end_x, end_y = centro_x, centro_y
```

```
        # Extender la línea en la dirección actual hasta encontrar un borde
o salir de la imagen
        while 0 <= end_x + dx < cols and 0 <= end_y + dy < rows:
            end_x += dx
            end_y += dy
            if linea_detectada[end_y, end_x] == 255:  # Si encontramos un
borde
                contador_lineas_que_toquen_borde += 1
                break

    # Comprobar si al menos 3 líneas tocan un borde
    return contador_lineas_que_toquen_borde >= 3
```

Este código utiliza un enfoque basado en un acumulador para identificar centros de círculos en las líneas detectadas. Se rastrean longitudes de líneas en distintas direcciones desde puntos de interés y se acumulan votos en puntos medios (centros potenciales). Una vez identificado un candidato, se verifica su validez y, si es aceptado, se mide el radio del círculo.

```
def encontrar_centro_con_rango_de_busqueda(linea_detectada, intervalo=10,
rango=20):
    acumulador = np.zeros_like(linea_detectada, dtype=np.int32)
    rows, cols = linea_detectada.shape

    # Definir las 8 direcciones de los píxeles
    direcciones = [
        (-1, 0), (1, 0), (0, -1), (0, 1),
        (-1, -1), (-1, 1), (1, -1), (1, 1)
    ]

    contador_pixeles = 0

    for y in range(rows):
        for x in range(cols):
            if linea_detectada[y, x] == 255:
                contador_pixeles += 1
                if contador_pixeles % intervalo == 0:
                    longitudes_lineas = []
                    puntos_finales = []

                    for dx, dy in direcciones:
                        end_x, end_y = x, y
                        longitud = 0
                        while 0 <= end_x + dx < cols and 0 <= end_y + dy <
rows:
```

```
                            end_x += dx
                            end_y += dy
                            longitud += 1
                            if linea_detectada[end_y, end_x] == 255:
                                break

                        longitudes_lineas.append(longitud)
                        puntos_finales.append((end_x, end_y))

                    max_longitud_idx = np.argmax(longitudes_lineas)
                    end_x, end_y = puntos_finales[max_longitud_idx]

                    mid_x = (x + end_x) // 2
                    mid_y = (y + end_y) // 2

                    acumulador[mid_y, mid_x] += 1

    contador_vecindario = np.zeros_like(acumulador)
    for y in range(rango, rows - rango):
        for x in range(rango, cols - rango):
            ventana = acumulador[y - rango:y + rango + 1, x - rango:x +
rango + 1]
            contador_vecindario[y, x] = np.sum(ventana)

    centro_y, centro_x = np.unravel_index(np.argmax(contador_vecindario),
contador_vecindario.shape)
    max_votos = contador_vecindario[centro_y, centro_x]

    # Crear una copia para la imagen de prueba de verificación
    if verificar_centro_candidato(linea_detectada, centro_x, centro_y):
        max_distancia = 0
        for dx, dy in direcciones:
            end_x, end_y = centro_x, centro_y
            while 0 <= end_x + dx < cols and 0 <= end_y + dy < rows:
                end_x += dx
                end_y += dy
                if linea_detectada[end_y, end_x] == 255:
                    distancia = np.sqrt((end_x - centro_x) ** 2 + (end_y -
centro_y) ** 2)
                    if distancia > max_distancia:
                        max_distancia = distancia
                    break

        if max_distancia > 0:
            return (centro_x, centro_y), max_distancia
```

```
    return None, None
```

Este código aplica las funciones anteriores para procesar cada línea detectada, identificar círculos y dibujarlos en imágenes de salida. También genera una imagen que conserva los bordes restantes tras eliminar los círculos detectados.

```
# Procesar todas las líneas en bordes_restantes y dibujar los círculos en la imagen final
def procesar_todas_las_lineas(bordes_restantes, imagen_original):
    lineas_detectadas = detectar_lineas_conectadas(bordes_restantes)
    imagen_resultado = imagen_original.copy()  # Imagen final para todos los círculos
    imagen_solo_circulos = np.zeros_like(bordes_restantes)  # Imagen final para solo los círculos
    bordes_restantes_2 = bordes_restantes.copy()  # Bordes restantes despues de los circulos

    for idx, linea in enumerate(lineas_detectadas):
        print(f"Procesando línea {idx + 1}")
        centro, radio = encontrar_centro_con_rango_de_busqueda(linea)
        if centro:
            print(f"Centro del círculo en línea {idx + 1}: {centro}")
            print(f"Diámetro del círculo en línea {idx + 1}: {2 * radio}")

            # Dibujar el círculo detectado en la imagen de resultado
            cv2.circle(imagen_resultado, centro, int(radio), (255, 0, 0), 2)
            cv2.circle(imagen_resultado, centro, 2, (0, 0, 255), 3)

            # Dibujar solo los círculos detectados en la imagen nueva
            cv2.circle(imagen_solo_circulos, centro, int(radio), (255), 2)

            # Borrar circulos de los bordes restantes
            cv2.circle(bordes_restantes_2, centro, int(radio), (0), 2)
        else:
            print(f"No se encontró un círculo válido en línea {idx + 1}.")

    return imagen_resultado, imagen_solo_circulos, bordes_restantes_2

# 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
```

Se segmenta la imagen original en regiones con valores de intensidad similares mediante una exploración BFS (búsqueda en anchura). Cada vecindario detectado se almacena junto con su borde, permitiendo análisis adicionales.

```
# Función para detectar vecindarios (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
```

Se identifican las líneas en una imagen binaria de bordes basándose en su representación en el espacio Hough. Las líneas detectadas se eliminan de la imagen para enfocar el análisis en formas curvas como círculos y elipses. Finalmente, los contornos restantes en la imagen se procesan para ajustar elipses. Estas elipses se dibujan tanto en la imagen original como en imágenes dedicadas, proporcionando una representación visual de los resultados.

```
# Cargar imagen en escala de grises
imagen_a_color = cv2.imread('imagen.png')
if imagen_a_color is None:
    print("Error: No se pudo cargar la imagen.")
else:
    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=75, minLineLength=3, maxLineGap=1)

    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), 2) #
Quita las lineas rectas

    # Funcion para obtener los circulos en la imagen ylos bordes que restan
    imagen_circulos, imagen_solo_circulos, bordes_restantes_2 =
procesar_todas_las_lineas(bordes_imagen, imagen_a_color)

    # Detectar contornos en la imagen para la elipse
    contornos, _ = cv2.findContours(bordes_restantes_2, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
    # Crear una copia para visualizar el resultado en la imagen original y
la sola
    imagen_elipses_sobrepuestos = imagen_circulos.copy()
    imagen_elipses = np.zeros_like(imagen)

    # Lista para guardar las elipses detectadas
    elipses_detectadas = []

    # Procesar cada contorno
    for contorno in contornos:
        if len(contorno) >= 160:  # Necesitamos al menos 10 puntos para
ajustar una elipse
            # Ajustar una elipse al contorno
            elipse = cv2.fitEllipse(contorno)
            elipses_detectadas.append(elipse)
            # Dibujar las elipses en la imagen original
            cv2.ellipse(imagen_elipses_sobrepuestos, elipse, (121, 84, 183),
2)
            # Dibujar las elipses una imagen nueva
            cv2.ellipse(imagen_elipses, elipse, (255), 2)
```

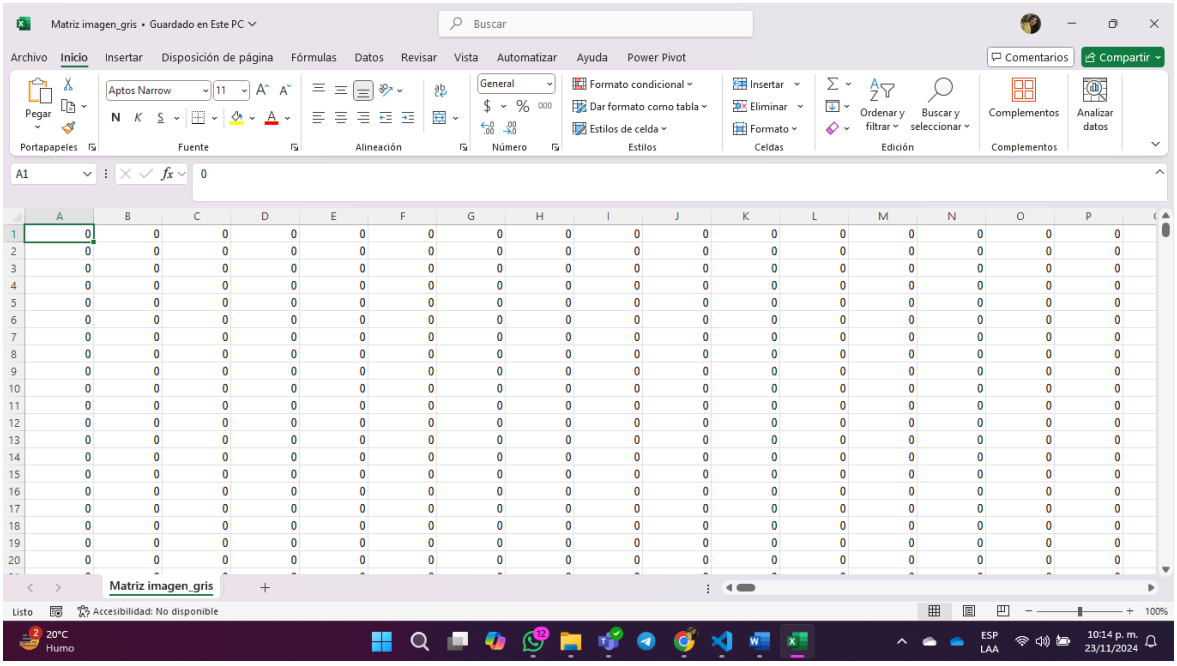
El código guarda las imágenes y matrices resultantes en archivos CSV y PNG, asegurando que los resultados sean accesibles para análisis futuros.

```
    # Mostrar las imagenes procesadas
    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.imshow("Circulos detectados", imagen_circulos)
    cv2.imshow("Solo circulos", imagen_solo_circulos)
    cv2.imshow("Bordes restnantes despues de circulos", bordes_restantes_2)
    cv2.imshow("Elipses", imagen_elipses_sobrepuestos)
    cv2.imshow("Solo elipses", imagen_elipses)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Guardar las imagenes procesadas
    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)
```

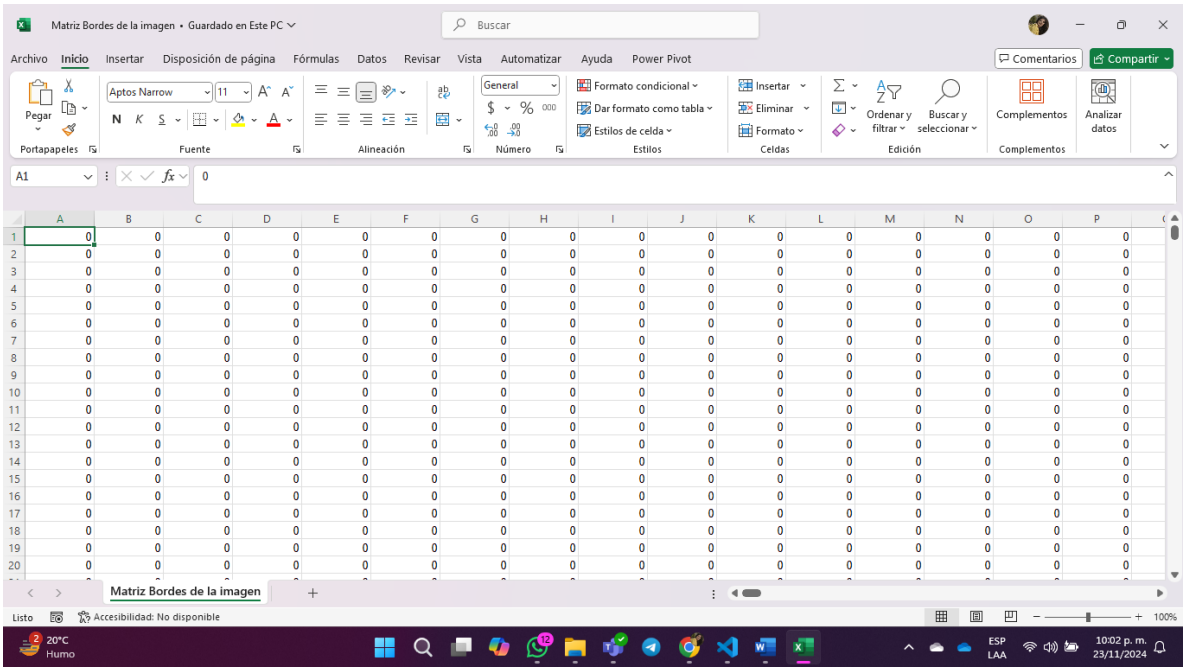
```
    cv2.imwrite("Imagen Circulos detectados.png", imagen_circulos)
    cv2.imwrite("Imagen Solo circulos.png", imagen_solo_circulos)
    cv2.imwrite("Imagen Bordes restantes despues de circulos.png",
bordes_restantes_2)
    cv2.imwrite("Imagen Elipses.png", imagen_elipses_sobrepuestos)
    cv2.imwrite("Imagen Solo elipses.png", imagen_elipses)

    # 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
'Matriz imagen_gris.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
'Matriz 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
'Matriz Bordes de la imagen.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
'Matriz Solo lineas rectas.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
'Matriz Bordes restantes de la imagen.csv'.")

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

```
    # Guardar la matriz de la imagen Bordes restantes de la imagen en un
archivo CSV
    np.savetxt('Matriz Bordes restantes despues de circulos.csv',
bordes_restantes_2, delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'Matriz Bordes restantes despues de circulos.csv'.")

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

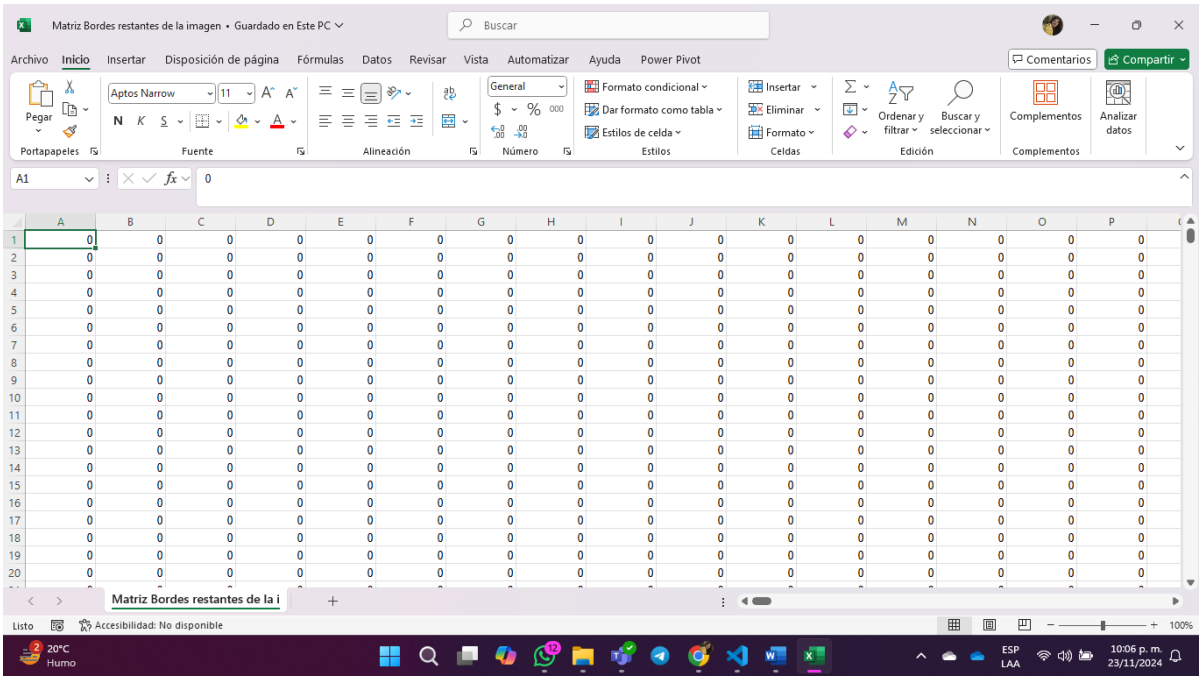
Matriz imagen_gris.csv



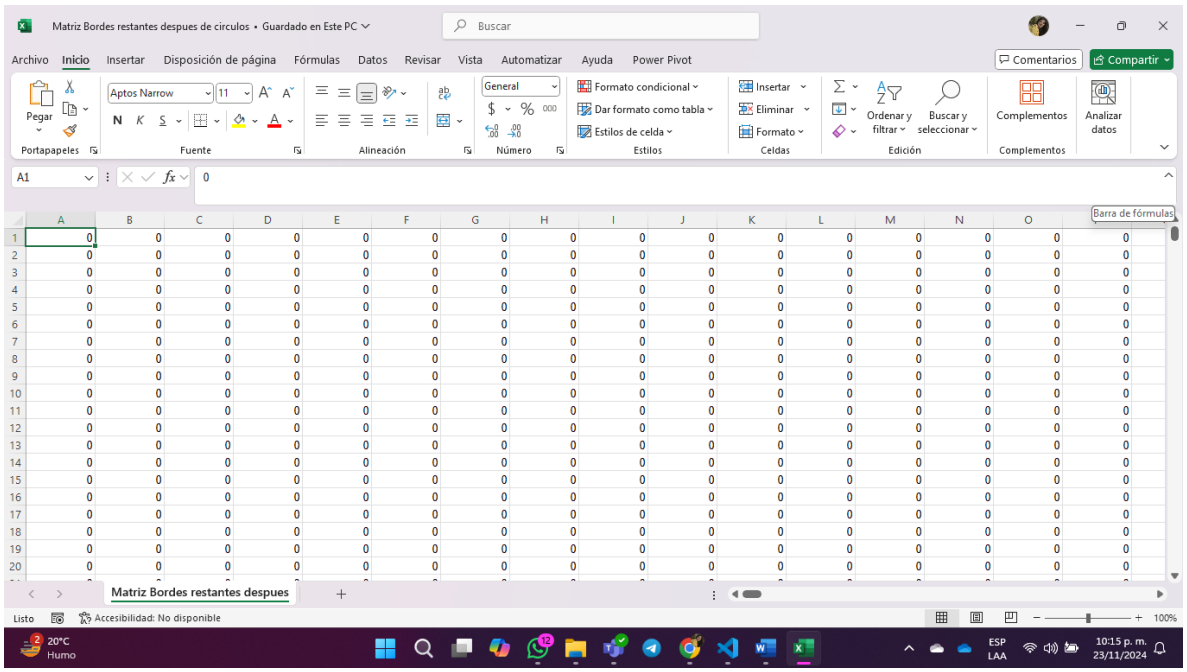
Matriz bordes de la imagen.csv



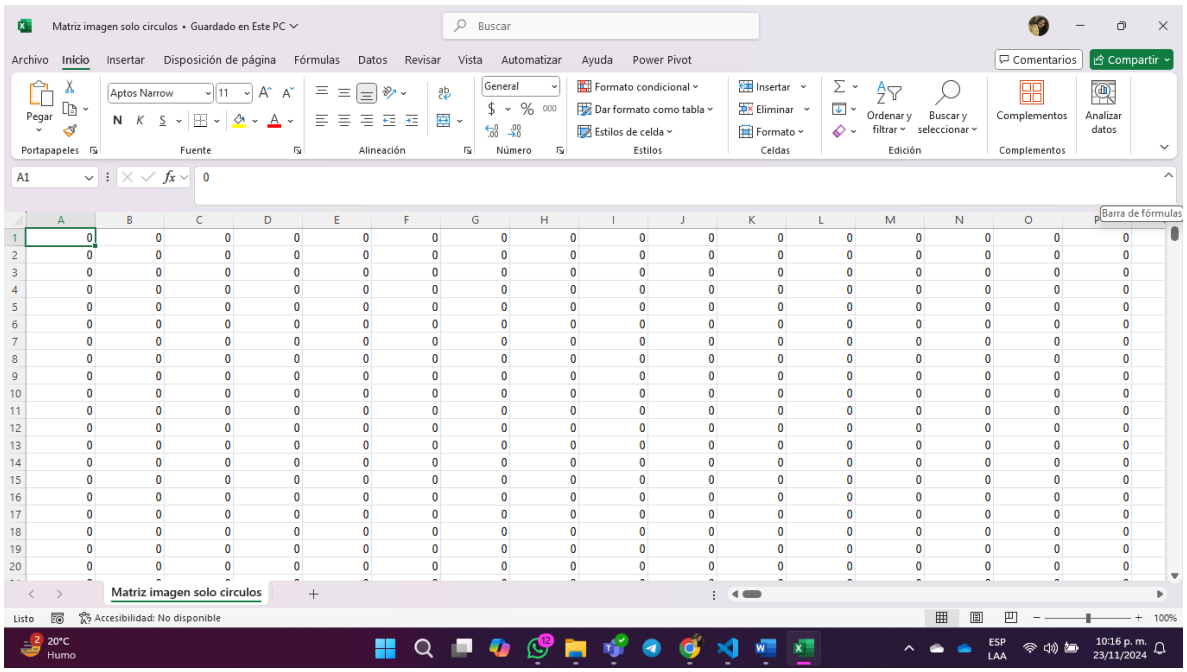
Matriz bordes restantes de la imagen.csv



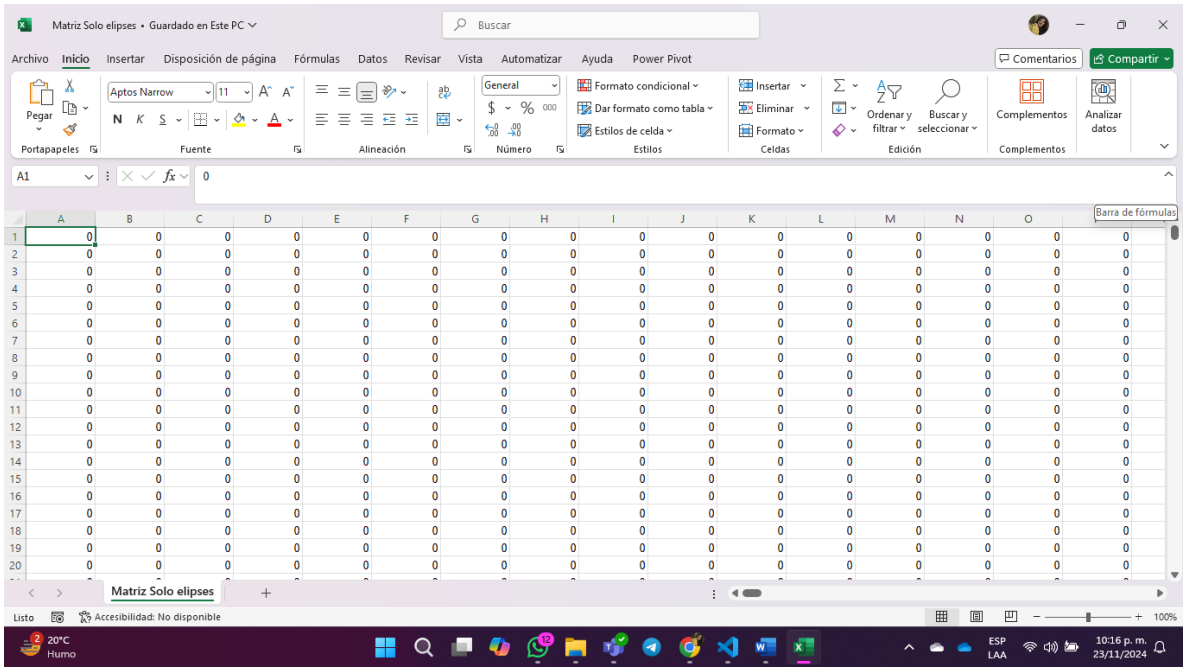
Matriz bordes restantes después de circulos.csv



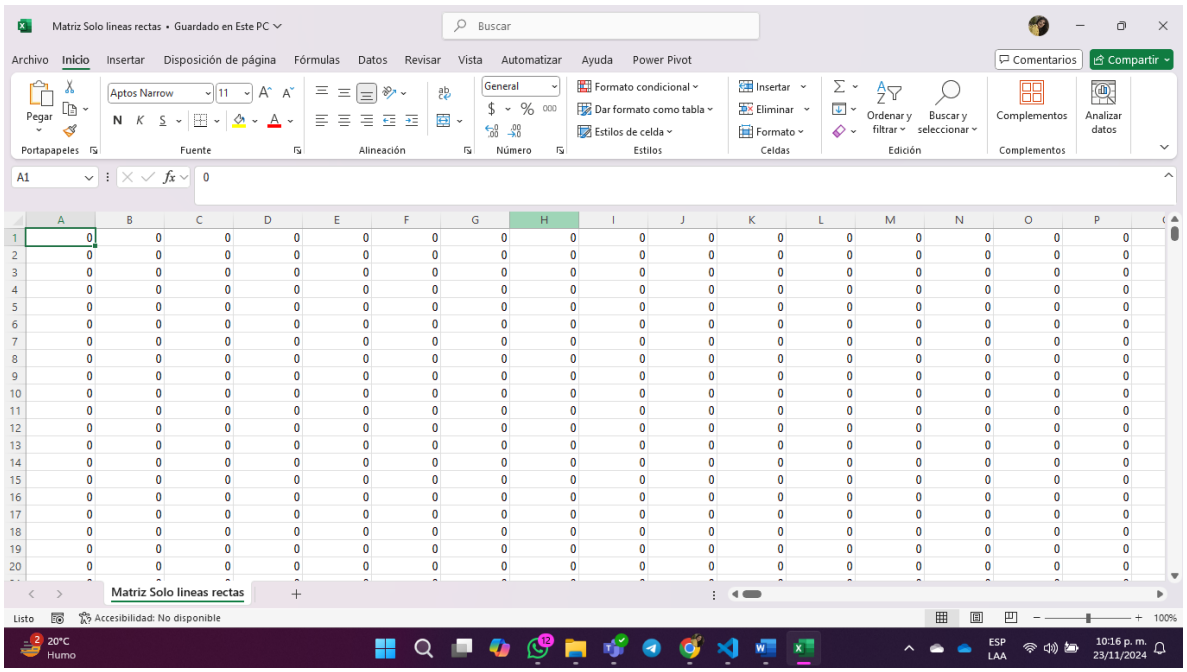
Matriz imagen solo circulos.csv



Matriz solo elipses.csv



Matriz solo líneas rectas.csv



Matriz vecindarios.csv

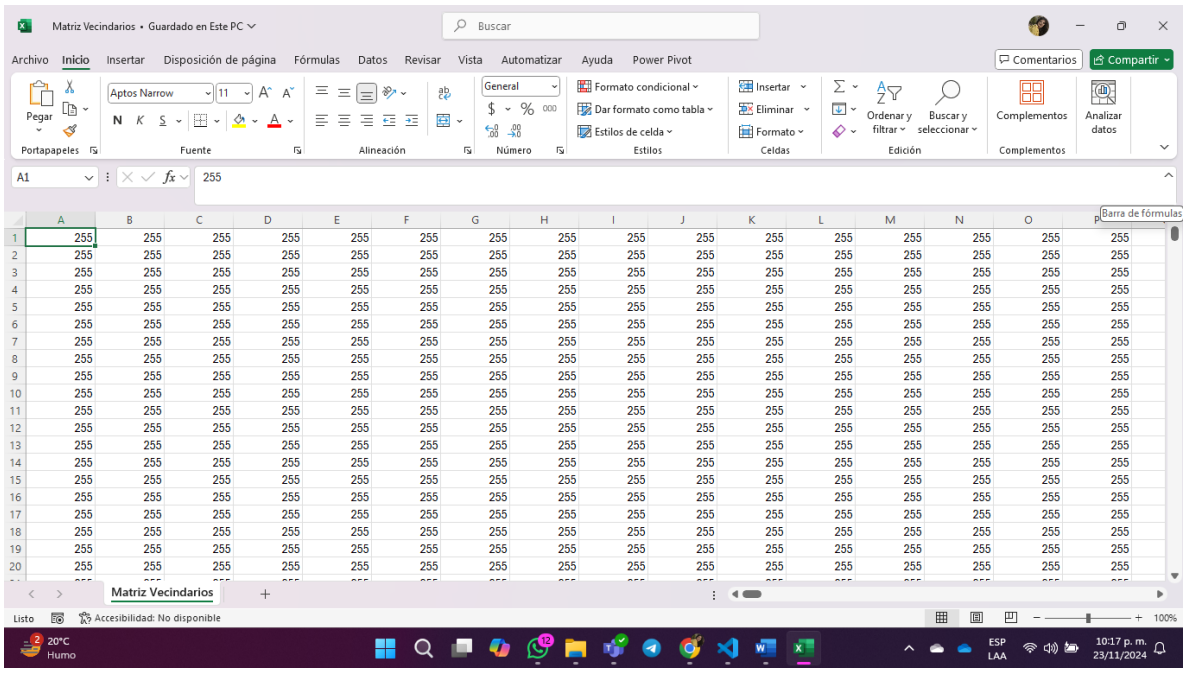


Imagen.png

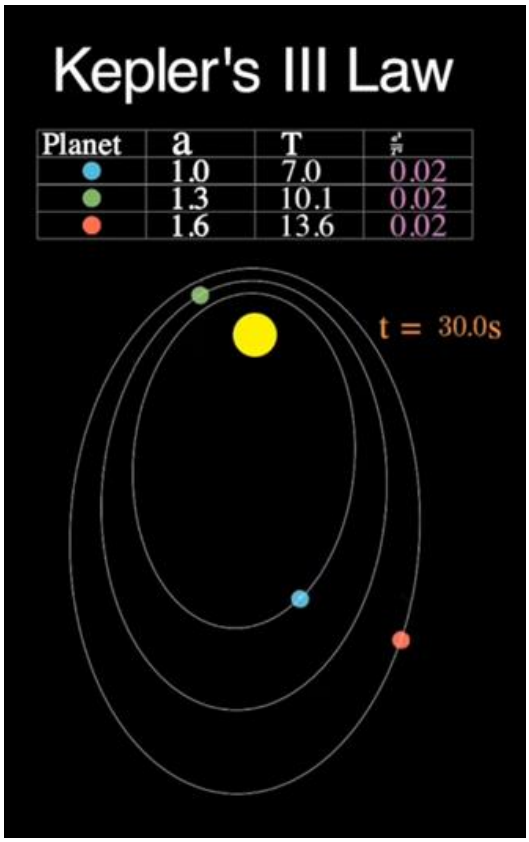


Imagen vecindarios.png

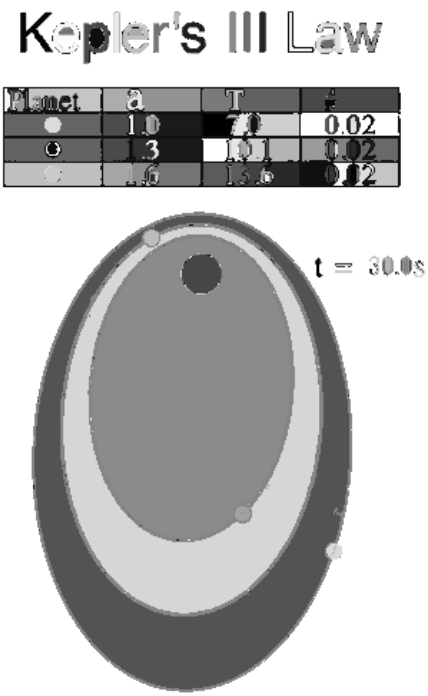


Imagen bordes de la imagen.png

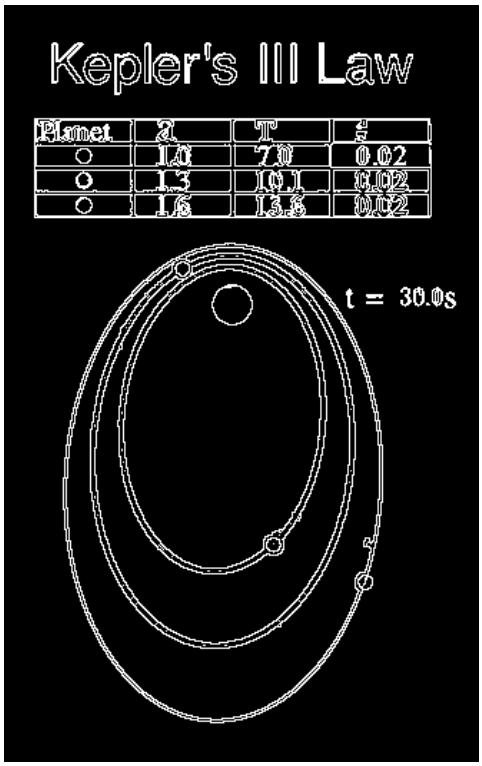


Imagen líneas rectas sobrepuestas.png

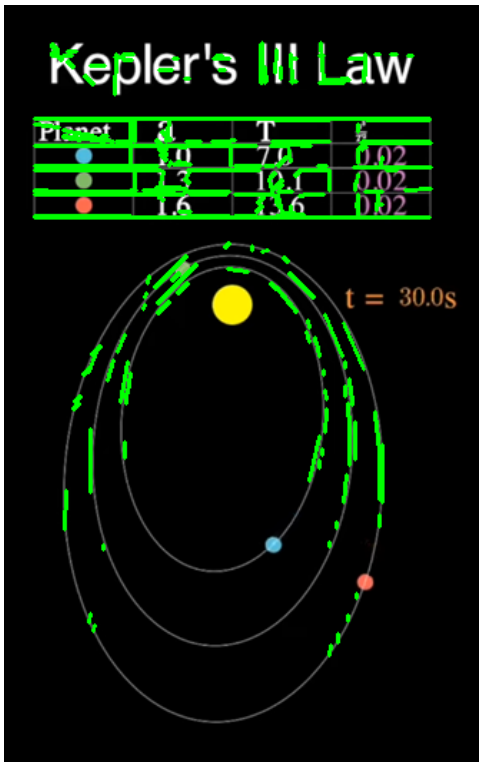


Imagen solo líneas rectas.png

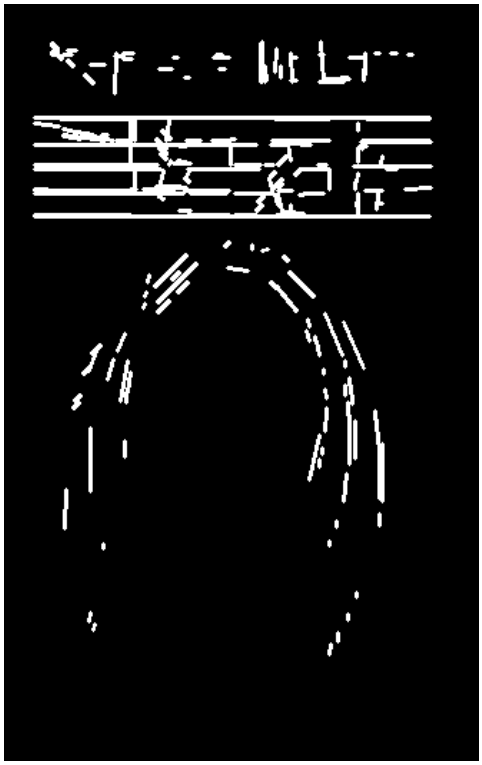


Imagen bordes restantes de la imagen.png

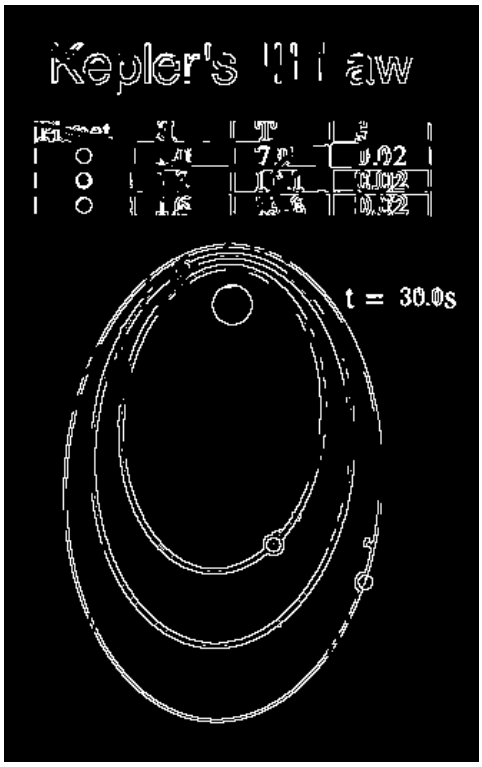


Imagen círculos detectados.png

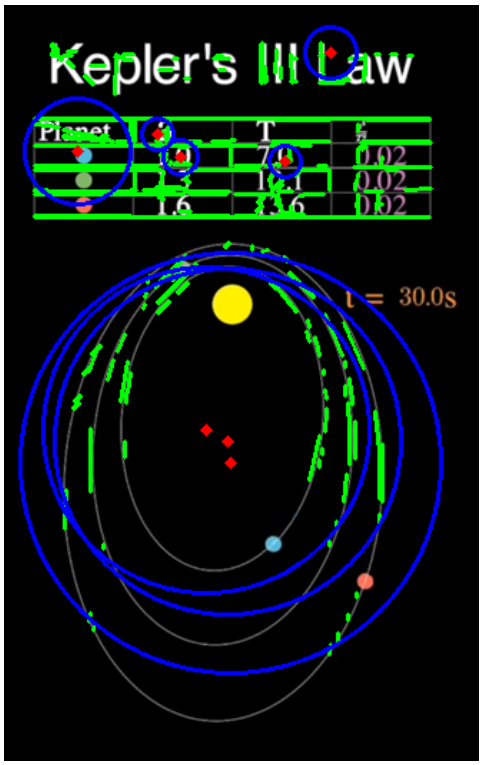


Imagen solo circulos.png

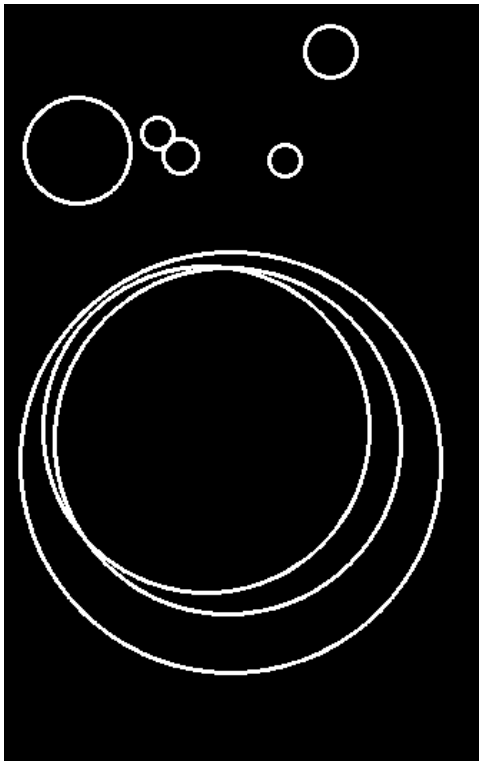


Imagen bordes restantes después de circulos.png

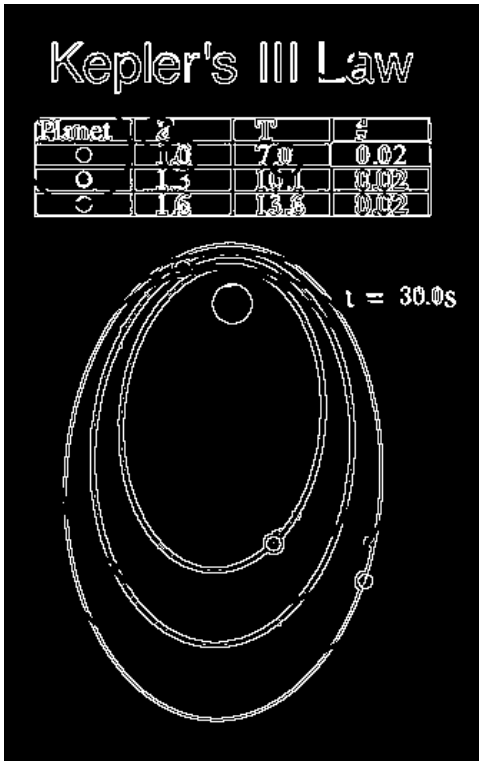


Imagen elipses.png

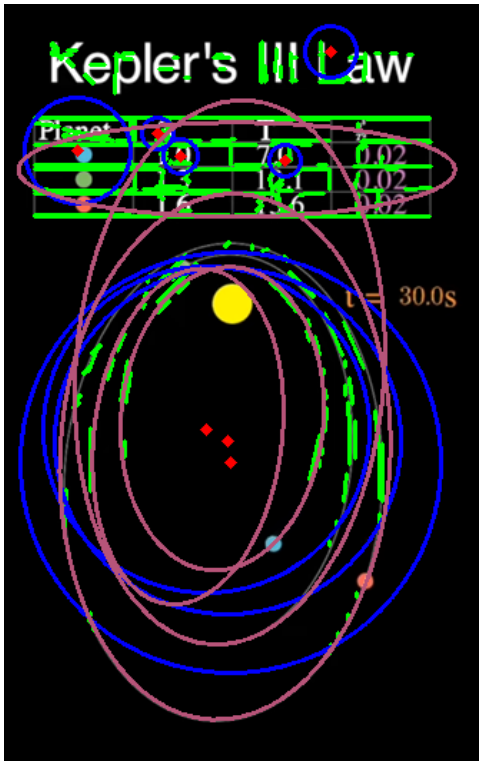
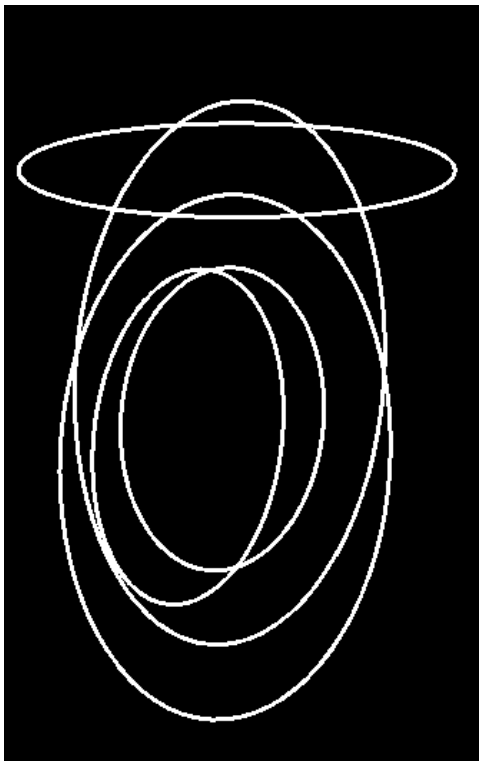


Imagen solo elipses.png



Conclusiones

El programa desarrollado implementa un flujo integral de procesamiento de imágenes diseñado para identificar características geométricas como líneas, círculos y elipses en imágenes en escala de grises. Al emplear técnicas avanzadas como la Transformada de Hough y algoritmos basados en vecindarios y conectividad, el programa es capaz de segmentar y analizar patrones visuales complejos de manera eficiente. Además, utiliza el algoritmo RANSAC para garantizar una detección más precisa de líneas incluso en escenarios donde el ruido podría dificultar la identificación de estas estructuras.

Una de las fortalezas del programa es su capacidad para superponer los elementos detectados sobre la imagen original, además de generar salidas separadas en formatos útiles, como imágenes procesadas y archivos CSV. Estas características facilitan el análisis detallado de los resultados y su integración en sistemas de procesamiento más amplios. La modularidad del código, combinada con el uso de librerías como OpenCV, lo hace adaptable a diferentes tipos de imágenes y objetivos específicos, incrementando su versatilidad.

Bibliografía

OpenCV modules. OpenCV. (n.d.). https://docs.opencv.org/4.x/index.html

NumPy Org. (2024, June 17). NumPy. https://numpy.org/

Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing, Global Edition*. Pearson Higher Ed.

Tangentes, cuerdas y arcos – Círculos y Pi. Mathigon. Retrieved November 24, 2024, from https://es.mathigon.org/course/circles/tangets-chords-arcs