Universidad de Monterrey
Facultad: Ingeniería y Tecnologías
Visión Computacional

Detección de personas con foto y video usando Otsu

Alejandro Acosta Murillo #616929

“Doy mi palabra de que he realizado esta actividad con integridad académica”

Descripción del problema:

El objetivo de este proyecto es implementar un sistema de procesamiento de imágenes en tiempo real y estáticas, aplicando técnicas de segmentación mediante el método de Otsu y la detección de contornos. El proyecto se centra en dos escenarios: uno que captura video en vivo y otro que procesa imágenes estáticas. En ambos casos, se utilizan algoritmos de visión por computadora para detectar y resaltar objetos o formas relevantes, como los contornos.

La segmentación por el método de Otsu es clave en este proyecto, ya que permite binarizar las imágenes de manera automática al calcular un umbral óptimo, diferenciando las áreas de interés del fondo. Esto resulta útil en la detección de contornos, ya que facilita la identificación precisa de los límites de objetos en la escena, como personas u otras formas. En el caso de video en vivo, los contornos se detectan y resaltan en tiempo real, lo que es útil para aplicaciones de monitoreo o análisis dinámico.

Video en vivo:

El primer código captura video en tiempo real desde una cámara conectada al sistema. A través de un proceso en bucle, cada cuadro de video se convierte a escala de grises y se le aplica un filtro de **Gauss** para reducir el ruido presente en la imagen. Posteriormente, se utiliza el método de **Otsu** para binarizar la imagen, separando los píxeles en blanco y negro de manera automática. Finalmente, se detectan y dibujan los contornos de los objetos presentes en el cuadro, resaltándolos en verde sobre la imagen original.

Este enfoque es útil en aplicaciones donde es necesario resaltar los bordes de los objetos en una imagen o video en tiempo real, como la visión por computadora en la automatización industrial o la robótica.

Código:

```
# Importamos la librerías
import cv2
import numpy as np

# Se inicia la captura de imágenes
cap = cv2.VideoCapture(0)

# Ciclo infinito
while True:
```

```
    # Empezamos la captura por frame
    ret, frame = cap.read()

    # El frame se converte a escala de grises
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Se utiliza el filtro de Gauss para eliminar ruido
    blur = cv2.GaussianBlur(gray, (5, 5), 0)

    # Aplicar el umbral de Otsu
    ret, otsu_threshold = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

    # Detectar los contornos en la imagen filtrada con Otsu
    contours, hierarchy = cv2.findContours(otsu_threshold, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    # Dibujar los contornos en la imagen original
    cv2.drawContours(frame, contours, -1, (0, 255, 0), 2)  # Color verde para los contornos

    # Mostrar el frame original con los contornos detectados
    cv2.imshow('Video con Contornos Detectados', frame)

    # Mostrar la imagen binarizada usando el mé*todo de Otsu*
    cv2.imshow('Imagen aplicando el umbral de Otsu', otsu_threshold)

    # Presionar la barra espaciadora para salirse
    if cv2.waitKey(1) & 0xFF == ord(' '):
        break

# Liberar la cámara y cerrar todas las ventanas
cap.release()
cv2.destroyAllWindows()
```

Video en fotografía:

El segundo código carga una imagen estática y aplica diferentes técnicas de umbralado para segmentar la imagen en áreas claras y oscuras. Se exploran tres métodos:

- **Global Thresholding**, donde se aplica un umbral fijo a toda la imagen.
- **Otsu's Thresholding**, que ajusta automáticamente el umbral basado en la distribución de píxeles.
- **Otsu con Filtro Gaussiano**, donde se reduce el ruido antes de aplicar el umbral de Otsu, mejorando la segmentación en imágenes con más ruido.

Estos métodos se visualizan lado a lado utilizando **Matplotlib**, lo que permite comparar cómo el procesamiento afecta la calidad de la segmentación. Este enfoque es útil en la preparación de imágenes para análisis más complejos, como la detección de objetos o reconocimiento de patrones.

```
import cv2
from matplotlib import pyplot as plt

# Cargamos la imagen en escala de grises
img = cv2.imread('ejemplo1.jpeg')

# Se aplica el filtro de global thresholding
ret1, th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

# Otsu's thresholding
ret2, th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Otsu thresholding después de aplicar un filtro de gaus
blur = cv2.GaussianBlur(img,(5,5),0)
ret3, th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Graficamos los resultados
titles = ['Original Image','Global Thresholding (v=127)',"Otsu's Thresholding after Gaussian
Filtering"]
images = [img,th1,th2,th3]

for i in range(4):
   plt.subplot(2,2,i+1)
```

Así mismo, se agregó una versión diferente para mezclar estos dos códigos y darle la oportunidad al usuario de elegir cuál de los dos métodos quiere realizar.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Solicitar al usuario que seleccione una opción
opcion = input("Escriba 'i' para procesar una imagen o 'v' para procesar un video en vivo:
").lower()

# Opción 1: Proceso de detección de contornos en video en tiempo real
if opcion == 'v':
    print("Iniciando la detección de contornos en tiempo real...")

    # Se comienza la captura de video desde la cámara
    cap = cv2.VideoCapture(0)

    # Bucle que se ejecuta de forma continua
    while True:
        # Captura de fotogramas del video
        ret, frame = cap.read()

        # Convertimos el fotograma a escala de grises
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Aplicamos un filtro Gaussiano para reducir el ruido
        blur = cv2.GaussianBlur(gray, (5, 5), 0)

        # Se utiliza el método de Otsu para binarizar la imagen
        ret, otsu_threshold = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

        # Se detectan los contornos de la imagen binarizada
        contours, hierarchy = cv2.findContours(otsu_threshold, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

        # Se dibujan los contornos sobre la imagen original
        cv2.drawContours(frame, contours, -1, (0, 255, 0), 2)  # Color verde para los contornos

        # Mostrar el video original con los contornos resaltados
        cv2.imshow('Video con Contornos Detectados', frame)

        # Mostrar la imagen umbralizada con el método de Otsu
        cv2.imshow('Imagen binarizada usando umbral de Otsu', otsu_threshold)
```

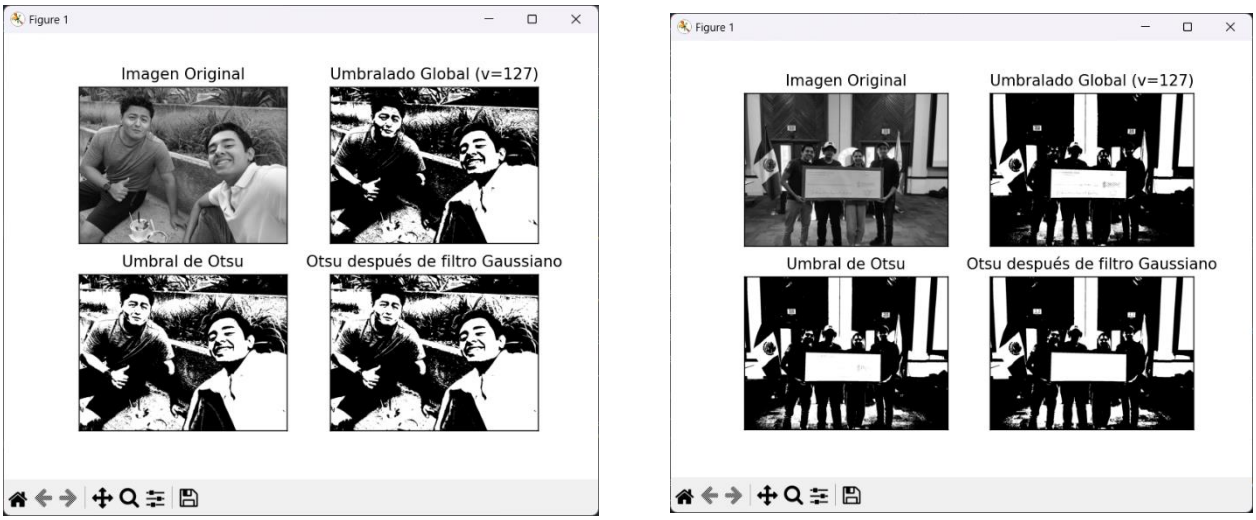
```
        # Presionar la barra espaciadora para salir del bucle
        if cv2.waitKey(1) & 0xFF == ord(' '):
            break

    # Se libera el acceso a la cámara y se cierran las ventanas de visualización
    cap.release()
    cv2.destroyAllWindows()

# Opción 2: Procesamiento de imagen estática aplicando diferentes técnicas de umbralado
elif opcion == 'i':
    print("Procesando una imagen estática...")

    # Cargamos la imagen en escala de grises desde el archivo
    img = cv2.imread('ejemplo1.jpeg', 0)  # Usamos 0 para cargar en escala de grises

    # Aplicamos umbralado global con un valor fijo
    ret1, th1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

    # Aplicamos el método de umbral de Otsu directamente
    ret2, th2 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    # Umbral de Otsu después de aplicar un filtro Gaussiano
    blur = cv2.GaussianBlur(img, (5, 5), 0)
    ret3, th3 = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    # Mostramos las imágenes resultantes con sus títulos correspondientes
    titles = ['Imagen Original', 'Umbralado Global (v=127)', "Umbral de Otsu", "Otsu después de
filtro Gaussiano"]
    images = [img, th1, th2, th3]

    for i in range(4):
        plt.subplot(2, 2, i + 1)
        plt.imshow(images[i], 'gray')
        plt.title(titles[i])
        plt.xticks([]), plt.yticks([])
    plt.show()

# Si el usuario ingresa una opción no válida
else:
    print("Opción no reconocida. Por favor, ingrese 'i' para video o 'v' para imagen.")
```

Resultados:

Primeramente, se utilizaron fotografías sin una persona, para revisar la aplicación de los filtros (*Figuras 1.1 y 1.2*):

Figuras 1.1. Fotografías con filtro de Otsu sin personas

Una vez que se hicieron esas pruebas se incluyeron fotografías con individuos.



Figuras 1.2. Fotografías con filtro de Otsu con personas

Resultados con video en tiempo real:

Para las siguientes pruebas se utilizó la opción de detección de contornos en tiempo real, utilizando la continua toma de fotografías por parte del programa, los resultados se muestran a continuación (*Figuras 1.3 y 1.4)*

Figuras 1.3. Detección de personas en tiempo real

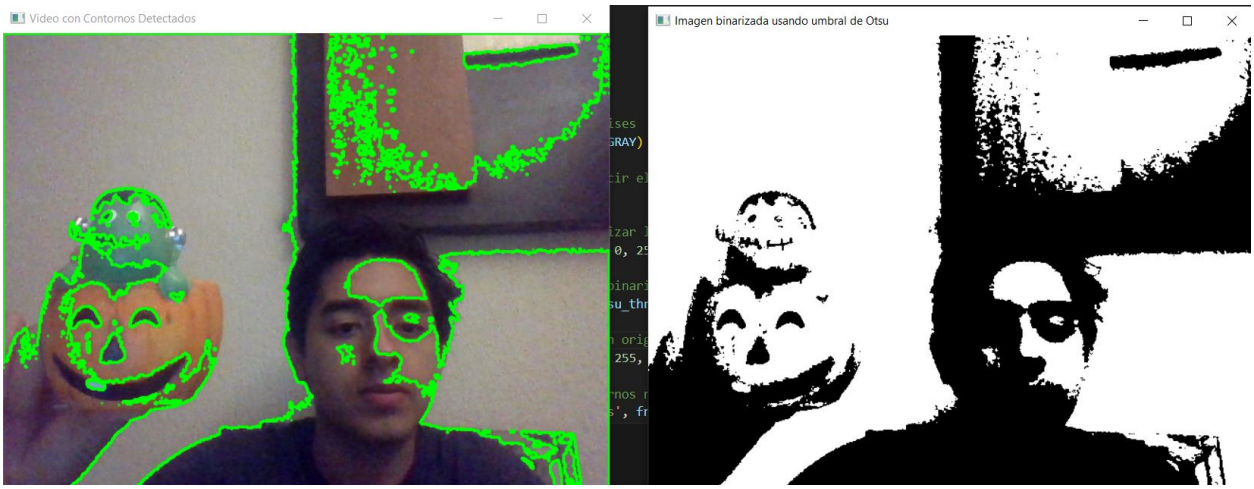
Figuras 1.3. Detección de personas en tiempo real binarizadas

Para el código de los resultados en tiempo real se puede observar claramente la segmentación entre contornos, así mismo al momento de correr el código este se comporta de manera fluida, sin pausas ni lags. Por lo que se puede concluir que este se encuentra optimizado.

Análisis de resultados:

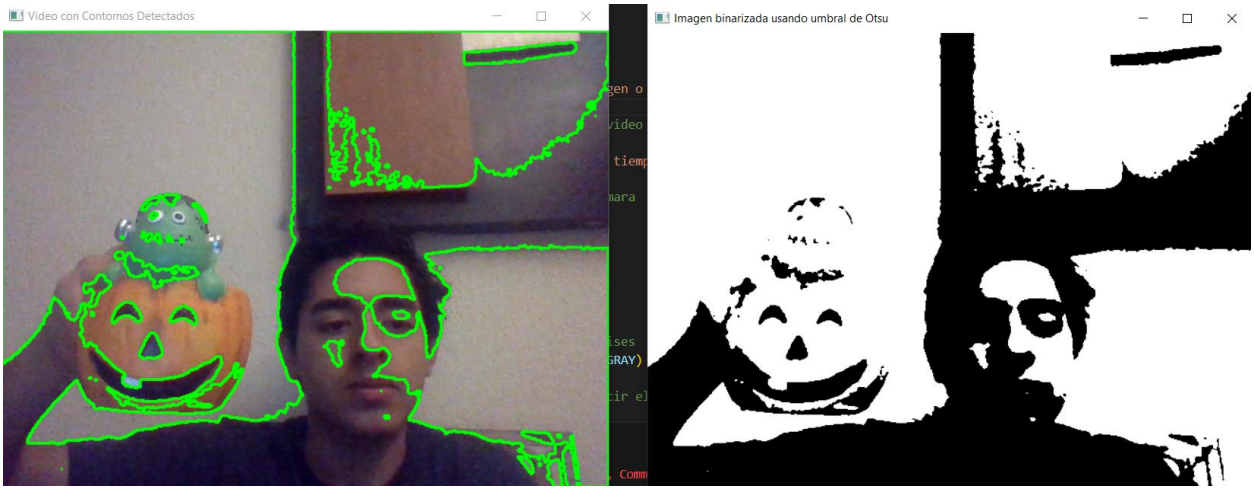
A continuación, se muestran los resultados de la detección sin el filtro Gaussiano:

```
    # Bucle que se ejecuta de forma continua
    while True:
        # Captura de fotogramas del video
        ret, frame = cap.read()

        # Convertimos el fotograma a escala de grises
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Aplicamos un filtro Gaussiano para reducir el ruido
        #blur = cv2.GaussianBlur(gray, (5, 5), 0)

        # Se utiliza el método de Otsu para binarizar la imagen
        ret, otsu_threshold = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

        # Se detectan los contornos de la imagen binarizada
        contours, hierarchy = cv2.findContours(otsu_threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

        # Se dibujan los contornos sobre la imagen original
        cv2.drawContours(frame, contours, -1, (0, 255, 0), 2)  # Color verde para los contornos
```



Con filtro gaussiano:

```python
# Bucle que se ejecuta de forma continua
while True:
    # Captura de fotogramas del video
    ret, frame = cap.read()

    # Convertimos el fotograma a escala de grises
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Aplicamos un filtro Gaussiano para reducir el ruido
    blur = cv2.GaussianBlur(gray, (5, 5), 0)

    # Se utiliza el método de Otsu para binarizar la imagen
    ret, otsu_threshold = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    # Se detectan los contornos de la imagen binarizada
    contours, hierarchy = cv2.findContours(otsu_threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    # Se dibujan los contornos sobre la imagen original
    cv2.drawContours(frame, contours, -1, (0, 255, 0), 2)  # Color verde para los contornos
```



Haciendo la comparación de ambos resultados, podemos notar la diferencia entre la detección de los contornos entre las imágenes con filtro gaussiano y sin él filtro. En la imagen con filtro Gaussiano encontramos una mejor definición de los detalles, no se encuentran tantas sombras y no se detectan tantos puntos al azar (más específicamente en el cuadro superior derecho).

En lo personal considero que el filtro Gaussiano es una buena opción para la detección de contornos.

Conclusión:

En conclusión, este proyecto implementa un sistema de procesamiento de imágenes para la detección de personas utilizando el método de Otsu para su segmentación.

En este caso se programó un código que te permite hacer la detección de personas y contorno desde imágenes estáticas así como con video en tiempo real. Para el video el sistema captura cuadros de una cámara, para posteriormente convertirlo a escala de grises y después aplicar un filtro Gaussiano para reducir el ruido antes de binarizar la imagen, esto ayuda a resaltar los contornos de las personas u objetos presentes en la imagen. Cabe recalcar que el procesamiento en video se realiza de manera fluida, sin retrasos, lo que indica la buena optimización del código.

Analizando los resultados con y sin filtro Gaussiano, podemos observar la importancia de este, y como si en caso de no usarse, se encuentran casos de falsa detección en los objetos que se están sosteniendo, así como en el cuadro en la esquina superior derecha.