Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Mecánica

Desarrollo de un programa en Python para seguimiento de imágenes de llamas esféricas (ANEXOS)

Autor:

López García, Beatriz Isabel

Tutores:

Reyes Serrano, Miriam
Ingeniería Energética y Fluidomecánica

Sastre Zamora, Rosaura
Ingeniería Energética y Fluidomecánica

Valladolid, junio de 2024.

ÍNDICE

1. PROGRAMA DE PYTHON

```python
# LIBRERÍAS
# Apertura de las librerías que vamos a usar en el desarrollo del programa
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os

#  LECTURA DEL VIDEO
# Abrir el archivo de video

nombre_video = "BC_E15.cine"
nombre_video_sin_extension = os.path.splitext(nombre_video)[0]
captura = cv2.VideoCapture(nombre_video)

# Crear el objeto BackgroundSubtractor
fgbg = cv2.createBackgroundSubtractorMOG2()

# Propiedades del video original
# Los fps van a depender del tipo de extensión de video que se este haciendo.
# Si es formato cine, corresponde exactamente con la que recoge la cámara, pero
# si es otro tipo de extension de video seguramente sea inferior, para evitar errores,
# con las variables que están definidads más abajo, lo que se ha hecho es que para otro tipo de extension,
# que no sea cine, se pone el valor de fps a lo que se recoge la cámara, si este valor se modifica en
# la cámara, se debe de modificar en la variable.

# Propiedades del video original
if nombre_video.endswith(".cine"):
    fps = int(captura.get(cv2.CAP_PROP_FPS))  #Fotogramas que tiene el video
else:
    fps = 3100

ancho = int(captura.get(cv2.CAP_PROP_FRAME_WIDTH))  #Ancho del video original
alto = int(captura.get(cv2.CAP_PROP_FRAME_HEIGHT))  #Alto del video original
#Número de fotogramas que tiene el video
n_fotogramas = int(captura.get(cv2.CAP_PROP_FRAME_COUNT))

# Redimension del video que se muestra por pantalla
nuevo_ancho = ancho // 2
nuevo_alto = alto // 2

# Variable para definir el botón de pausa
pausa = False

# Crear el video de salida con la informacion procesada con los datos recogidos en el bucle
out = cv2.VideoWriter('Resultado.cine', cv2.VideoWriter_fourcc(*'XVID'), fps, (ancho, alto))


#  RADIO CAMARA DE COMBUSTION
# Tratamiento del radio camara de combustion

ret, primer_fotograma = captura.read()

fotograma_inicial = 1
# Transformamos a escala de grises el fotograma almacenado
grises = cv2.cvtColor(primer_fotograma, cv2.COLOR_BGR2GRAY)

# Obtener la imagen umbralizada
umbral = cv2.threshold(grises, 100, 255, cv2.THRESH_BINARY)[1]

# Encontrar contornos
contornos, jerarquia = cv2.findContours(umbral, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

# Encontrar la región de mayor área
max_contorno = max(contornos, key=cv2.contourArea)

# Ajustar elipse a la región
elipse = cv2.fitEllipse(max_contorno)

# Obtener centro y radio de la elipse
centro, ejes, angulo = elipse
eje_mayor, eje_menor = ejes
radio = (eje_mayor + eje_menor) / 4
centro = (int(centro[0]), int(centro[1]))
print("El centro de la elipse es {}" .format(centro))
print("El radio de la camara de combustiones de {}" .format(int(radio)))

# Calculamos la variable calibracion que nos sirve de variable de control para el proceso del calculo del radio del frente de llama
calibracion = radio/5.7
print("La calibracion es de {}" .format(calibracion))

# Dibujar la circunferencia de la cámara de combustión
cv2.circle(primer_fotograma, (int(centro[0])+14, int(centro[1])+5), 2, (0, 0, 255), 2)

# Calculamos el centro que vamos a considerar real
centro_real = (int(centro[0]) + 14, int(centro[1]) + 5)
print("El centro de la camara de combustion es {}" .format(centro_real))
cv2.circle(primer_fotograma, centro_real, int(radio), (255, 0, 0), 2)

# Mostrar imagen resultante
cv2.imshow('Contorno de la cámara de combustión', primer_fotograma)
```

```python


#  RADIO FRENTE DE LLAMA
# Tratamiento de fotogramas

#Contador de fotogramas
contador_fotogramas = 1

# Listas que almacenan los datos generados del radio y el tiempo
x_tiempo_radio = []
y_radio = []
x_tiempo_celularidad = []
y_celularidad = []


while True:
  if not pausa:
    ret, fotograma = captura.read()


    # Salir del bucle cuando se haya procesado todo el video
    if not ret:
        break
    fotograma = cv2.resize(fotograma, (nuevo_ancho, nuevo_alto))
    copia = fotograma.copy()
    contador_fotogramas += 1


    # Aplicar el sustractor de fondo para obtener la imagen de los objetos en movimiento
    mascara = fgbg.apply(fotograma)

    # Aplicar un umbral para obtener solo los píxeles más brillantes en la imagen
    umbralizacion = cv2.threshold(mascara, 127, 255, cv2.THRESH_BINARY)[1]

    # Encontrar los contornos en la imagen binaria
    contornos, hierarchy = cv2.findContours(umbralizacion, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Ordenar los contornos por área de mayor a menor
    contornos = sorted(contornos, key=cv2.contourArea, reverse=True)

    # Tomar los dos primeros contornos (los de mayor área)
    modificacion_area = 4
    contornos_grandes = contornos[:modificacion_area]

    # Encontrar la elipse circunscrita
    if len(contornos_grandes) == modificacion_area:
        rectangulo = cv2.minAreaRect(np.vstack(contornos_grandes))
        centro_rectangulo = rectangulo[0]
        diametro_mayor = max(rectangulo[1])
        diametro_menor = min(rectangulo[1])
        angulo = rectangulo[2]
        radio = (diametro_mayor * diametro_menor) / (diametro_mayor + diametro_menor)
        radio_celularidad = radio * 0.8

        # Dibujar los contornos que está tomando como ejemplo
        cv2.circle(fotograma, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(radio), (0, 255, 0), 2)

        # Comparar radio con el de la camara de combustion y cálculo de la variable tiempo
        tiempo = (contador_fotogramas - fotograma_inicial) * (1000/fps)
        print ("El tiempo en este instante es {}" .format(tiempo))

        if int(radio)/calibracion <= 5.7:
             # Comparar radio con el de la camara de combustion
             radio_video = int(radio) / calibracion
             radio_dato = radio_video * 2
             print("El radio (con el radio medio) del frente de llama es de {}" .format(radio_dato))

             # Guardar los en cada instante de los datos recogidos por el programa
             x_tiempo_radio.append(tiempo)
             y_radio.append(radio_dato)

```

```
        plt.figure(1)
        plt.plot(x_tiempo_radio, y_radio, color='red')
        plt.title('Velocidad del frente de llama')
        plt.xlabel('Tiempo (ms)')
        plt.ylabel('Radio del frente de llama (cm)')

        grafico_radio = plt.gcf()
        grafico_radio.canvas.draw()
        grafico_radio =np.array(grafico_radio.canvas.renderer.buffer_rgba())

        # Mostrar la imagen del gráfico
        #cv2.imshow('Grafico Radio', grafico_radio)

        # CELULARIDAD
        # Definir la región de interés (ROI) centrada en el centro del rectángulo mínimo
        # Crear una máscara circular
        mascara = np.zeros(fotograma.shape[:2], dtype=np.uint8)
        # Esta fijado el centro con el obtenido previamente en la cámara de combustión
        cv2.circle(mascara, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(radio_celularidad), 255, -1)
        cv2.circle(fotograma, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(radio_celularidad), (0, 255, 255), 2)

        # Aplicar la máscara al fotograma copiado para obtener solo los píxeles dentro de la circunferencia
        roi = cv2.bitwise_and(copia, copia, mask=mascara)

        # Aplicar el sustractor de fondo y umbralización solo a la región de interés
        mascara_roi = fgbg.apply(roi)

        # Aplicar un umbral para obtener solo los píxeles más brillantes en la imagen
        umbralizacion_roi = cv2.threshold(mascara_roi, 127, 255, cv2.THRESH_BINARY)[1]

        # Aplicar la transformación de adelgazamiento y encontrar los contornos en la imagen binaria resultante
        thin_roi = cv2.ximgproc.thinning(umbralizacion_roi)
        contornos_roi, hierarchy = cv2.findContours(thin_roi, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        # Dibujar los contornos encontrados en la imagen copiada (copia)
        cv2.drawContours(fotograma, contornos_roi, -1, (0, 255, 0), 2)

        densidad_celular = len(contornos_roi) / (np.pi *((int(radio_celularidad)/calibracion)**2))
        densidad_celular = densidad_celular * 2
        contornos_color = cv2.cvtColor(thin_roi, cv2.COLOR_GRAY2BGR)
        print("Cantidad de elementos {}".format(densidad_celular))


        #cv2.imshow("Celularidad fotograma", fotograma)
        #cv2.imshow('Culularida en contornos', thin_roi)

        # Guardamos en unas listas los datos recogidos para cada fotograma
        x_tiempo_celularidad.append(tiempo)
        y_celularidad.append(densidad_celular)

        plt.figure(2)
        plt.plot(x_tiempo_celularidad, y_celularidad, color ='red')
        plt.title('Crecimiento de la celularidad')
        plt.xlabel('Tiempo (ms)')
        plt.ylabel('Densidad celularidad')

        grafico_celularidad = plt.gcf()
        grafico_celularidad.canvas.draw()
        grafico_celularidad = np.array(grafico_celularidad.canvas.renderer.buffer_rgba())

        #cv2.imshow('Grafico Celularidad', grafico_celularidad)


        # Unir los dos gráficos en una sola ventana y representarlos
        grafico_completo = cv2.hconcat([grafico_radio, grafico_celularidad])
        videos_representacion = cv2.hconcat([fotograma, contornos_color])

        cv2.imshow('Grafico completo', grafico_completo)
        cv2.imshow('Videos unidos', videos_representacion)
        #cv2.imshow('Pantalla completa', conjunto_ventanas)


        # Mostrar la imagen del gráfico
        #cv2.imshow('Grafico Radio', grafico_celularidad)
```

```
    # Mostrar la imagen del gráfico
    #cv2.imshow('Grafico Radio', grafico_radio)

    # Crear una imagen en blanco
    ancho, alto = 180, 470
    imagen = np.zeros((ancho, alto, 3), np.uint8)
    imagen.fill(255)

    # Posicion de la tabla y tamaño
    a, b = 30, 30

    cv2.putText(imagen, "FOTOGRAMA", (a+5, b+20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
    cv2.putText(imagen, "TIEMPO", (a+5, b+50), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
    cv2.putText(imagen, "RADIO", (a+5, b+80), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
    cv2.putText(imagen, "DENSIDAD CELULARIDAD", (a+5, b+110), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
    cv2.putText(imagen, str(contador_fotogramas), (a+200, b+20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
    cv2.putText(imagen, str(tiempo), (a+200, b+50), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
    cv2.putText(imagen, str(radio_dato), (a+200, b+80), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
    cv2.putText(imagen, str(densidad_celular), (a+200, b+110), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)

    cv2.imshow('Tabla de datos', imagen)

    # Escribir el fotograma procesado en el video de salida
    out.write(fotograma)
    print("Estoy en el fotograma {}" .format(contador_fotogramas))
    # Mostrar las figuras
    plt.show(block=False)

    # Mostrar el fotograma procesado en una ventana
    #cv2.imshow('Radio del frente de llama', fotograma)
    #cv2.imshow('Celularidad en contornos', thin_roi)

    # Guardar los datos en un archivo de excel
    # Primero generamos un dataframe
    df = pd.DataFrame({"TIEMPO RADIO": x_tiempo_radio , "RADIO FRENTE DE LLAMA": y_radio , "TIEMPO CELULARIDAD": x_tiempo_celularidad , "CELULARI
    # Salir del bucle cuando el usuario presione la tecla 'Esc'
    delay = int(1000/fps)
```

```
  # Detectar la pulsación de la barra espaciadora para pausar o reanudar el video
  key = cv2.waitKey(1)
  if key == ord(' '):
     pausa = not pausa
  if cv2.waitKey(30) & 0xff == 27:
     break

# Guardamos el dataframe en un archivo de excel
archivo_excel = nombre_video_sin_extension + "_completo.xlsx"
df.to_excel(archivo_excel, index = False)

# Cerrar los objetos VideoCapture, VideoWriter y todas las ventanas
captura.release()
out.release()
cv2.destroyAllWindows()
```

2. OPERACIONES DE CÁLCULO

2.1. Operaciones del radio

```python
# Encontrar la elipse circunscrita
if len(contornos_grandes) == modificacion_area:
    rectangulo = cv2.minAreaRect(np.vstack(contornos_grandes))
    centro_rectangulo = rectangulo[0]
    diametro_mayor = max(rectangulo[1])
    diametro_menor = min(rectangulo[1])
    angulo = rectangulo[2]
    radio = (diametro_mayor * diametro_menor) / (diametro_mayor + diametro_menor)

    # Calculamos los vertices del rectangulo
    vertices =cv2.boxPoints(rectangulo)
    vertices = np.int0(vertices)
    orden = np.argsort(vertices[:, 0])

    cv2.drawContours(fotograma, contornos_grandes, -1, (0, 255,0), 2)
    cv2.rectangle(fotograma, vertices[orden[0]], vertices[orden[2]], (0, 255, 0), 2) # Cuadrado color verde
    cv2.drawContours(fotograma, [vertices], 0, (0, 255, 0), 2)
    #cv2.drawContours(fotograma, contornos_grandes, -1, (255, 0, 0), 2)
    for cnt in contornos_grandes:
        if len(cnt) < 5.7:
            continue
        try:
            elipse = cv2.fitEllipse(cnt)
            centro_elipse, (diametro_mayor_elipse, diametro_menor_elipse), angulo_elipse = elipse
            distancia_mayor = np.linalg.norm(np.array(centro_elipse) - np.array(centro_rectangulo))
            distancia_menor = max(diametro_mayor_elipse, diametro_menor_elipse) / 2
            diametro_mayor = max(diametro_mayor, distancia_mayor + distancia_mayor)
            diametro_menor = min(diametro_menor, distancia_menor + distancia_menor)
            angulo = angulo_elipse
            #cv2.ellipse(fotograma, elipse, (255, 0, 0), 2, cv2.LINE_AA)
        except cv2.error:
            pass
    elipse_circunscrita = ((int(centro_rectangulo[0]), int(centro_rectangulo[1])), (int(diametro_mayor), int(diametro_menor)), angulo)
    cv2.ellipse(fotograma, elipse, (255, 0, 0), 2, cv2.LINE_AA)   # Elipse color azul
    cv2.circle(fotograma, (int(centro_rectangulo[0]), int(centro_rectangulo[1])) , int(diametro_mayor/2), (0, 255, 255), 2) # Circ. color amarill
    cv2.circle(fotograma, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(diametro_menor/2), (255, 0, 255), 2) # Circ. color rosa
    cv2.circle(fotograma, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(radio), (0, 255, 0), 2)  # Circ. color rojo

```

2.2. Operaciones de celularidad

```python
        # Aplicar la transformación de adelgazamiento y encontrar los contornos en la imagen binaria resultante
        thin_roi = cv2.ximgproc.thinning(umbralizacion_roi)
        canny = cv2.Canny(umbralizacion_roi, 50, 150)
        contornos_roi, hierarchy = cv2.findContours(thin_roi, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        # Comparar radio con el de la camara de combustion y cálculo de la variable tiempo
        tiempo = (contador_fotogramas - fotograma_inicial) * (1000/fps)
        print ("El tiempo en este instante es {}" .format(tiempo))
```