UNIVERSIDAD DEL BÍO-BÍO

26 Tutorial Inspección Visual Automatizada: Coincidencia de Plantillas y Análisis de Manchas en Tiempo Real

Estudiantes Practicantes:
Luis Pereira

Profesor:
Luis Vera

Laboratorio CIMUBB

2023-2

Introducción:

El programa que se presenta a continuación se ha diseñado para desempeñar un papel esencial en un proceso de inspección visual automatizado, emulando de alguna manera el funcionamiento de una máquina de clasificación como las utilizadas en entornos industriales, como los supermercados.

Reconocimiento de Plantillas:

El programa utiliza el poderoso motor de coincidencia de plantillas de OpenCV para identificar y reconocer figuras específicas en las imágenes capturadas en tiempo real por una cámara. Estas plantillas actúan como referencias visuales, y cuando se encuentran coincidencias, se activan acciones correspondientes.

Análisis de Manchas:

Simultáneamente, el programa realiza un análisis detallado de las manchas presentes en las áreas de interés de las imágenes. Esta funcionalidad es crucial para evaluar la calidad y la integridad de las figuras identificadas. Se cuentan y cuantifican tanto las manchas blancas como las negras, proporcionando datos precisos sobre la limpieza y el estado general de cada figura.

Aplicación en un Proceso de Inspección:

Imaginemos un escenario donde las imágenes capturadas representan productos o componentes que pasan por una línea de producción. El programa, actuando como una "máquina de inspección visual", verifica la presencia y la calidad de las figuras deseadas, al mismo tiempo que evalúa la presencia de manchas que podrían indicar defectos o condiciones no deseadas.

Este enfoque integrado no solo agiliza el proceso de inspección, sino que también mejora la confiabilidad al combinar la identificación visual con métricas cuantitativas sobre la limpieza de las figuras. El resultado es un sistema robusto que contribuye a la calidad y la eficiencia en la producción.

Explicacion del codigo

1. Importar las bibliotecas necesarias:

```
import cv2
import os
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk
import numpy as np
```

2. Funciones utilizadas:

- **coincidencia_de_plantilla(ruta_imagen)**: Función para realizar coincidencia de plantillas en una imagen.

```
# Función para realizar coincidencia de plantillas
def coincidencia_de_plantilla(ruta_imagen):
    plantilla = cv2.imread(ruta_imagen, cv2.IMREAD_COLOR)
    plantilla_gris = cv2.cvtColor(plantilla, cv2.COLOR_BGR2GRAY)

    # Realizar coincidencia de plantilla en la imagen de la cámara (frame)
    resultado = cv2.matchTemplate(frame_gris, plantilla_gris, cv2.TM_CCOEFF_NORMED)
    _, _, _, max_loc = cv2.minMaxLoc(resultado)

    return resultado, max_loc
```

La función **coincidencia_de_plantillas(ruta_imagen)** tiene como objetivo buscar coincidencias de una plantilla específica en el fotograma actual capturado por la cámara. Aquí hay una explicación paso a paso de lo que hace:

1. **Cargar la Plantilla**: Lee la imagen de la plantilla desde la ruta especificada (**ruta_imagen**) utilizando OpenCV y la almacena en la variable plantilla.
2. Convertir a Escala de Grises: Convierte la plantilla a escala de grises utilizando **cv2.cvtColor()**, guardando el resultado en la variable plantilla_gris.
3. **Realizar Coincidencia de Plantilla:** Utiliza la función **cv2.matchTemplate()** para realizar una comparación entre la plantilla y la imagen actual del fotograma (almacenada en **frame_gris**). Esto genera un mapa de coincidencia, guardado en la variable resultado.

4. **Encontrar la Mejor Coincidencia:** Utiliza **cv2.minMaxLoc()** para encontrar las coordenadas del valor máximo en el mapa de coincidencia (**max_loc**). Este valor máximo indica la región donde la plantilla coincide mejor con la imagen.
5. **Devolver Resultados**: La función devuelve el mapa de coincidencia (**resultado**) y las coordenadas de la mejor coincidencia (**max_loc**), que se pueden utilizar para identificar la ubicación de la plantilla en la imagen.

- **procesar_frame()**: Función para procesar cada fotograma del flujo de la cámara.

```python
# Función para procesar el siguiente frame de la cámara
def procesar_frame():
    ret, frame = cap.read()

    if ret:
        global frame_gris, ImagenRecortada, th, bin_image, bin_image_aux, bin_image_aux2, ImagenRecortadaG
        frame_gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        _, th = cv2.threshold(frame_gris, 128, 240, cv2.THRESH_BINARY)
        contornos, _ = cv2.findContours(th, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
         # CONTANDO OBJETOS
        total = 0
        for c in contornos:
            area = cv2.contourArea(c)
            if area > 1700:
                peri = cv2.arcLength(c, True)
                approx = cv2.approxPolyDP(c, 0.02 * peri, True)
                if len(approx) == 4:
                    # Dibujar un contorno de un rectángulo alrededor del objeto
                    cv2.drawContours(frame, [approx], -1, (255, 0, 0), 2, cv2.LINE_AA)
                    total += 1

                    # Recortar el rectángulo
                    x, y, w, h = cv2.boundingRect(approx)
                    ImagenRecortada = frame[y:y + h, x:x + w]
                    # Mostrar la imagen recortada en el segundo label
                    ImagenRecortada = cv2.cvtColor(ImagenRecortada, cv2.COLOR_BGR2GRAY)
                    ImagenRecortadaG = ImagenRecortada
                    _, bin_image = cv2.threshold(ImagenRecortada, int(umbral_var.get()), 255, cv2.THRESH_BINARY)
                    bin_image_aux = bin_image
                    bin_image_aux2 = bin_image
                    bin_image = Image.fromarray(bin_image)
                    bin_image = ImageTk.PhotoImage(image=bin_image)
                    LImagen.configure(image=bin_image)
                    LImagen.image = bin_image

            if bin_image_aux is not None:
                contar_manchas()


        # Iterar sobre las imágenes en la carpeta y realizar coincidencia de plantillas
        for ruta_imagen in rutas_imagenes:
            resultado, max_loc = coincidencia_de_plantilla(ruta_imagen)

            # Si el valor máximo está por encima de un umbral, mostrar el resultado
            if resultado is not None and umbral_resultado.get() < resultado[max_loc[1], max_loc[0]]:
                texto_resultado.set(f"Coincidencia de plantilla con {os.path.basename(ruta_imagen)}")

        # Mostrar el frame en la interfaz gráfica
        mostrar_frame()

        # Llamar a esta función nuevamente después de un breve intervalo
        ventana.after(10, procesar_frame)
```

La función **procesar_frame()** realiza el procesamiento principal de cada fotograma capturado por la cámara. Aquí está una explicación paso a paso de lo que hace:

1. **Capturar Fotograma**: Utiliza **cap.read()** para capturar un fotograma de la cámara, y almacena el resultado en frame.

2. **Convertir a Escala de Grises**: Convierte el fotograma (frame) a escala de grises utilizando **cv2.cvtColor()**, y guarda el resultado en **frame_gris**.
3. **Binarizar la Imagen**: Utiliza **cv2.threshold()** para binarizar la imagen en blanco y negro (**th**). Este paso facilita la detección de contornos y objetos en la imagen.
4. **Encontrar Contornos**: Utiliza **cv2.findContours()** para identificar los contornos en la imagen binarizada. Los contornos se almacenan en la variable contornos.
5. **Contar Objetos:**
    - Itera sobre los contornos encontrados.
    - Filtra los contornos basándose en el área.
    - Aproxima cada contorno a un polígono.
    - Si el polígono tiene 4 vértices, lo considera un rectángulo.
    - Dibuja un contorno alrededor del objeto y lo recorta.
    - Muestra la imagen recortada en un segundo label (**LImagen**).
    - Llama a la función **contar_manchas()** si hay una imagen binarizada disponible.
6. **Coincidencia de Plantillas:**
    - Itera sobre las rutas de las imágenes de plantillas.
    - Llama a la función **coincidencia_de_plantilla()** para cada plantilla y verifica si hay coincidencias por encima de un umbral.
    - Muestra el resultado en la interfaz gráfica.
7. **Mostrar el Frame en la Interfaz Gráfica:** Utiliza **mostrar_frame()** para mostrar el fotograma actualizado en la interfaz gráfica.
8. **Llamada Recursiva**: Llama a la función nuevamente después de un breve intervalo **(ventana.after(10, procesar_frame))**, asegurando un procesamiento continuo y en tiempo real.

- **mostrar_frame()**: Función para mostrar el fotograma actual en la interfaz gráfica.

```
# Función para mostrar el frame en la interfaz gráfica
def mostrar_frame():
    if cap.isOpened():
        ret, frame = cap.read()
        if ret:
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = Image.fromarray(frame)
            img = ImageTk.PhotoImage(img)
            GImagenROI.img = img
            GImagenROI.config(image=img)
    else:
        GImagenROI.config(text="Error al abrir la cámara")
```

La función **mostrar_frame()** es esencial para visualizar el fotograma actual en la interfaz gráfica, permitiendo que la aplicación muestre el flujo en tiempo real proveniente de la cámara.

- **seleccionar_carpeta():** Función para seleccionar una carpeta de imágenes.

```
# Función para seleccionar la carpeta de imágenes
def seleccionar_carpeta():
    ruta_carpeta = filedialog.askdirectory()
    if ruta_carpeta:
        global rutas_imagenes
        rutas_imagenes = [os.path.join(ruta_carpeta, archivo) for archivo in os.listdir(ruta_carpeta) if archivo.endswith(('.png', '.jpg', '.jpeg'))]
```

La función **seleccionar_carpeta()** facilita al usuario la selección de una carpeta que contiene imágenes para ser utilizadas en el proceso de coincidencia de plantillas. Las rutas de estas imágenes se almacenan para su posterior procesamiento.

- **contar_manchas()**: Función para contar manchas blancas y negras en una imagen.

```
# Función para contar manchas blancas en la imagen
def contar_manchas():
    # Contar el número de pixeles con manchas blancas
    imagen_invertida = cv2.bitwise_not(bin_image_aux2)
    num_pixels_con_manchas_blancas = cv2.countNonZero(bin_image_aux)
    num_pixels_con_manchas_negras = cv2.countNonZero(imagen_invertida)
    # Calcular el porcentaje de manchas
    porcentaje_manchasB = (num_pixels_con_manchas_blancas / bin_image_aux.size) * 100
    porcentaje_manchasN = (num_pixels_con_manchas_negras / bin_image_aux.size) * 100

    # Contar manchas blancas
    contornos_blancos, _ = cv2.findContours(bin_image_aux, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cantidad_manchas_blancas = len(contornos_blancos)
    contornos_negros, _ = cv2.findContours(imagen_invertida, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cantidad_manchas_negras = len(contornos_negros)

    Cadena = f"Cantidad de manchas blancas: {cantidad_manchas_blancas}\nPorcentaje área con manchas: {round(porcentaje_manchasB, 2)}%"
    Cadena2 = f"Cantidad de manchas negras: {cantidad_manchas_negras}\nPorcentaje área con manchas: {round(porcentaje_manchasN, 2)}%"

    CajaTextoBlanco.configure(state='normal')
    CajaTextoBlanco.delete(1.0, tk.END)
    CajaTextoBlanco.insert(1.0, Cadena)
    CajaTextoBlanco.configure(state='disabled')
    CajaTextoNegro.configure(state='normal')
    CajaTextoNegro.delete(1.0, tk.END)
    CajaTextoNegro.insert(1.0, Cadena2)
    CajaTextoNegro.configure(state='disabled')
```

La función **contar_manchas()** realiza un análisis cuantitativo de las manchas en la imagen binarizada recortada y presenta los resultados en la interfaz gráfica. Esto proporciona información sobre la presencia y la extensión de manchas blancas y negras en la región de interés de la imagen.

- **iniciar_camara():** Función para iniciar la cámara y procesar los fotogramas.

```python
# Función para iniciar la cámara
def iniciar_camara():
    global cap
    cap = cv2.VideoCapture(puerto_camara)
    if not cap.isOpened():
        texto_resultado.set("Error al abrir la cámara")
    else:
        texto_resultado.set("Cámara iniciada")
        procesar_frame()
```

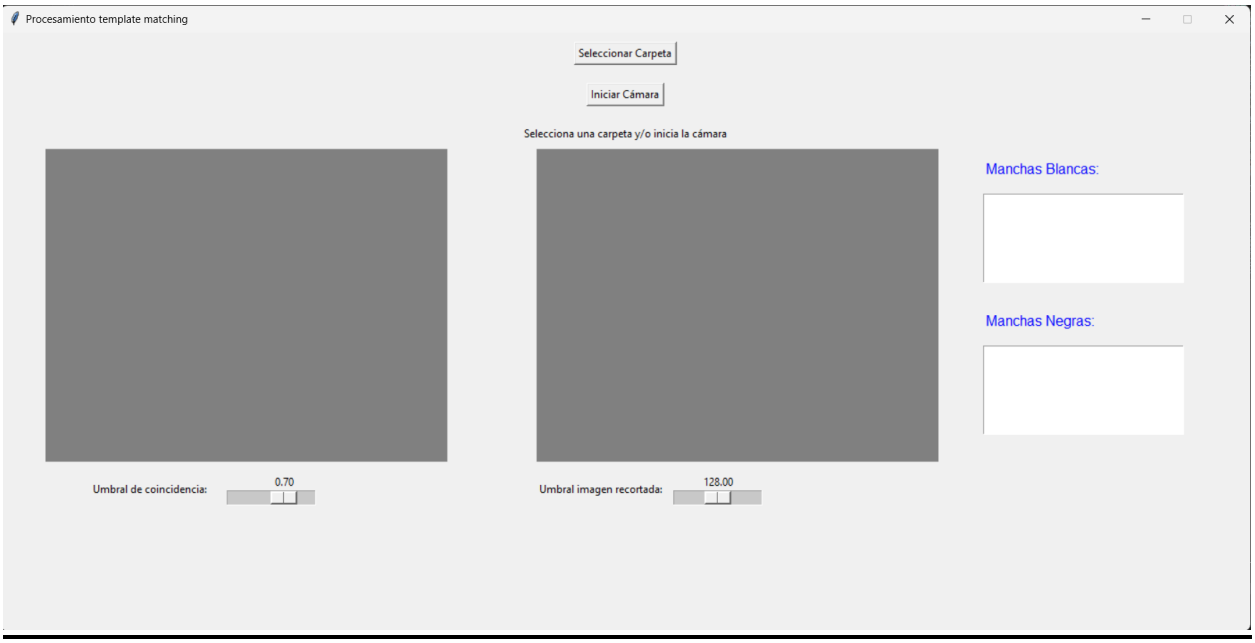
3. Crear la interfaz gráfica con Tkinter:

- **Botones de Acción:**
  - **Seleccionar Carpeta:** Permite al usuario elegir una carpeta que contiene imágenes de plantillas para el proceso de coincidencia.
  - **Iniciar Cámara**: Inicia el flujo de video desde la cámara en tiempo real para la inspección visual continua.
- **Etiqueta de Resultado:**
  - Muestra mensajes informativos sobre el estado del programa, como si la cámara se ha iniciado correctamente o si hay algún error.
- **Área de Visualización de Imágenes:**
  - Dos cuadros de imágenes (**GImagenROI** y **LImagen**) que presentan el flujo de video en tiempo real desde la cámara y la imagen binarizada recortada respectivamente. Estas áreas ofrecen una visualización en tiempo real de los objetos detectados y las manchas analizadas.
- **Cuadros de Texto:**
  - Dos cuadros de texto (**CajaTextoBlanco** y **CajaTextoNegro**) que proporcionan información detallada sobre la cantidad de manchas blancas y negras detectadas, así como el porcentaje de área ocupada por estas manchas.
- **Controles de Umbral:**
  - Dos controles deslizantes (**slider_umbral** y **slider_umbral2**) para ajustar dinámicamente los umbrales utilizados en el proceso de coincidencia de plantillas y en la binarización de la imagen recortada.
- **Información Adicional:**
  - Etiquetas informativas (**etiqueta_manchas_blancas** y **etiqueta_manchas_negras**) que describen la información presentada en los
  - cuadros de texto.

La interfaz gráfica se ha diseñado para proporcionar una experiencia fácil de usar y comprender, permitiendo al usuario controlar y monitorear la inspección visual automatizada en tiempo real. La combinación de elementos visuales y datos cuantitativos mejora la eficacia del proceso de análisis, facilitando la identificación y evaluación de objetos específicos y la presencia de manchas.

```
# Variables globales
cap = None
frame_gris = None
rutas_imagenes = []
texto_resultado = tk.StringVar()
texto_resultado.set("Selecciona una carpeta y/o inicia la cámara")
umbral_resultado = tk.DoubleVar()
umbral_resultado.set(0.7)
umbral_var = tk.DoubleVar()
umbral_var.set(128)
# Botones y etiquetas
boton_seleccionar_carpeta = tk.Button(ventana, text="Seleccionar Carpeta", command=seleccionar_carpeta)
boton_seleccionar_carpeta.pack(pady=10)

boton_iniciar_camara = tk.Button(ventana, text="Iniciar Cámara", command=iniciar_camara)
boton_iniciar_camara.pack(pady=10)

etiqueta_resultado = tk.Label(ventana, textvariable=texto_resultado)
etiqueta_resultado.pack(pady=10)

# Cuadros de Imagen grises
LImagen = tk.Label(ventana, background="gray")
LImagen.place(x=600, y=130, width=450 , height=350)
GImagenROI = tk.Label(ventana, background="gray")
GImagenROI.place(x=50, y=130, width=450, height=350)
# Cuadros de texto para mostrar resultados
CajaTextoBlanco = tk.Text(ventana, state="disabled")
CajaTextoBlanco.place(x=1100, y=180, width=225, height=100)

CajaTextoNegro = tk.Text(ventana, state="disabled")
CajaTextoNegro.place(x=1100, y=350, width=225, height=100)
# Etiqueta para manchas blancas
etiqueta_manchas_blancas = tk.Label(ventana, text="Manchas Blancas:", font=("Arial", 12), fg="blue")
etiqueta_manchas_blancas.place(x=1100, y=140)

# Etiqueta para manchas negras
etiqueta_manchas_negras = tk.Label(ventana, text="Manchas Negras:", font=("Arial", 12), fg="blue")
etiqueta_manchas_negras.place(x=1100, y=310)

etiqueta_umbral = tk.Label(ventana, text="Umbral de coincidencia:")
etiqueta_umbral.place(x=100, y=500)
slider_umbral = tk.Scale(ventana, from_=0, to=1, resolution=0.01, orient=tk.HORIZONTAL, variable=umbral_resultado)
slider_umbral.place(x=250, y=490)

etiqueta_umbral2 = tk.Label(ventana, text="Umbral imagen recortada:")
etiqueta_umbral2.place(x=600, y=500)
slider_umbral2 = tk.Scale(ventana, from_=0, to=255, resolution=0.01, orient=tk.HORIZONTAL, variable=umbral_var)
slider_umbral2.place(x=750, y=490)
```

4. Resultado final



Para garantizar un rendimiento óptimo del programa, es esencial seguir ciertos pasos clave. Aquí hay una guía que destaca las consideraciones importantes:

Selección de Carpeta con Plantillas:

- Antes de iniciar la inspección visual, asegúrate de seleccionar una carpeta que contenga las plantillas de manera clara y visible. Esto proporcionará al programa las referencias necesarias para realizar la coincidencia de patrones.

Uso del Programa Generador de Plantillas:

- Se recomienda utilizar el programa generador de plantillas adjunto. Este programa facilita el recorte de las imágenes de plantillas, lo que contribuye a una coincidencia más precisa y eficiente durante el proceso de inspección.

Ajuste del Umbral de Coincidencia:

- Selecciona cuidadosamente el umbral de coincidencia utilizando el control deslizante correspondiente. Ajustar este umbral de manera adecuada permitirá una identificación más precisa de las plantillas en el flujo de video de la cámara.

Umbral de la Imagen Recortada:

- El control deslizante destinado al umbral de la imagen recortada es fundamental para un análisis de manchas eficiente. Ajusta este umbral según sea necesario para obtener resultados precisos en la detección y cuantificación de manchas.

Estas consideraciones ayudarán a optimizar el rendimiento del programa y garantizarán una inspección visual eficaz. Recuerda que la selección adecuada de carpetas y umbrales es crucial para obtener resultados precisos y significativos durante la ejecución del programa.