UNIVERSIDAD DEL BÍO-BÍO

20 Seguimiento de Puntos con OpenCV y Tkinter

Tutorial Procesamiento de Imagen con webcam

Estudiantes Practicantes:
Luis Pereira

Profesor:
Luis Vera

Laboratorio CIMUBB

2023-2

Introducción al Programa de Seguimiento de Puntos con OpenCV y Tkinter:

En la vanguardia de la visión por computadora, el seguimiento de objetos y la detección de características son cruciales. Presentamos un programa que fusiona las potentes bibliotecas OpenCV y Tkinter en Python para crear una aplicación interactiva. Esta aplicación permite el seguimiento en tiempo real de un punto específico en la imagen capturada por la cámara de un dispositivo.

OpenCV, una biblioteca de código abierto, se destaca en el procesamiento de imágenes y visión por computadora. En nuestro programa, aprovechamos sus funciones para la detección de bordes, el seguimiento óptico de puntos mediante el método de Lucas-Kanade, y la detección de líneas utilizando la transformada de Hough.

El método de Lucas-Kanade se basa en el flujo óptico, modelando cómo los píxeles se desplazan entre fotogramas consecutivos. La función cv2.calcOpticalFlowPyrLK implementa este algoritmo, rastreando el movimiento del punto seleccionado.

La transformada de Hough se emplea para la detección de líneas, específicamente a través de cv2.HoughLinesP. Este método transforma información de píxeles en el espacio de parámetros de líneas, donde las líneas se representan como puntos. Al encontrar picos en este espacio de parámetros, identificamos las líneas en la imagen.

Además, el programa busca intersecciones entre las líneas detectadas, resaltándolas visualmente en la imagen. Esta aplicación no solo demuestra la versatilidad de OpenCV y Tkinter, sino que también sirve como base para aplicaciones más avanzadas en el campo de la visión por computadora.

Objetivo del Tutorial:

El objetivo principal de este programa es proporcionar una herramienta versátil para el seguimiento de puntos y la detección de líneas e intersecciones en la imagen de la cámara en tiempo real. La interfaz gráfica creada con Tkinter facilita la interacción del usuario, permitiendo la selección manual de un punto de interés.

Funcionalidades Destacadas:

1) **Seguimiento Óptico de Puntos:**

   Utiliza el método de Lucas-Kanade para rastrear un punto seleccionado en la imagen, proporcionando una experiencia interactiva.

2) **Detección de Líneas:**

   Aplica el operador Canny para detectar bordes y utiliza la transformada de Hough para encontrar y visualizar líneas en la imagen.

3) **Detección de Intersecciones:**

   Identifica los puntos de intersección entre las líneas detectadas, destacando áreas de convergencia en la escena.

4) **Interfaz Gráfica Amigable:**

   Utiliza Tkinter para crear una interfaz gráfica de usuario (GUI) que permite ajustar el umbral de detección de bordes y muestra las coordenadas del punto seleccionado.

Cómo Utilizar el Programa:

1) **Inicio**:

Al ejecutar el programa, la interfaz gráfica se abrirá, y la cámara comenzará a capturar imágenes en tiempo real.

2) **Selección de Puntos:**

Haz clic en la imagen para seleccionar manualmente un punto de interés. Este punto será rastreado en fotogramas sucesivos cada 1 segundo.

3) **Ajuste de Parámetros:**

Utiliza la barra deslizante para ajustar el umbral de detección de bordes (Canny), adaptando la sensibilidad a las condiciones específicas de la escena.

4) **Visualización en Tiempo Real:**

Observa cómo el programa realiza el seguimiento del punto seleccionado y detecta líneas e intersecciones en tiempo real.

5) **Coordenadas y Registro:**

Las coordenadas del punto seleccionado se muestran en la interfaz, y se registran automáticamente en un archivo de texto para su posterior análisis.

Explicación de las funciones:

mostrar_imagen(imagen):

```
def mostrar_imagen(imagen):
    imagen_rgb = cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)
    img = Image.fromarray(imagen_rgb)
    img = ImageTk.PhotoImage(img)

    panel_imagen.img = img
    panel_imagen.config(image=img)
```

1) Convierte la imagen de formato BGR a RGB utilizando OpenCV y PIL.

2) Crea un objeto ImageTk.PhotoImage a partir de la imagen convertida.
3) Configura el widget panel_imagen para mostrar la imagen actualizada.

actualizar_coordenadas(x, y):

```
def actualizar_coordenadas(x, y):
    texto_coordenadas.set("Coordenadas seleccionadas: ({}, {})".format(x, y))
    guardar_coordenadas(x, y)
```

1) Actualiza el texto del widget label_coordenadas con las coordenadas (x, y).
2) Llama a la función guardar_coordenadas(x, y) para guardar las coordenadas en un archivo.

guardar_coordenadas(x, y):

```
def guardar_coordenadas(x, y):
    with open("coordenadas.txt", "a") as archivo:
        archivo.write("Coordenadas: ({}, {})\n".format(x, y))
```

Abre el archivo "coordenadas.txt" en modo de añadir ("a") y escribe las coordenadas en formato de cadena

procesar_imagen(): Se explicara el código de la función en 2 imágenes ya que la función es robusta.

```
def procesar_imagen():
    global gris_anterior, punto_seleccionado, punto_interseccion  # Declarar como global

    # Obtener el valor actual del slidebar
    umbral_canny = slidebar_canny.get()

    # Capturar un fotograma de la cámara
    ret, frame = cap.read()
    if not ret:
        print("Error al capturar la imagen de la cámara")
        return

    # Convertir a escala de grises
    gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    desenfoque = cv2.GaussianBlur(gris, (5, 5), 0)
    bordes = cv2.Canny(desenfoque, umbral_canny, 2 * umbral_canny)

    # Rastrear el punto solo si se ha seleccionado uno
    if punto_seleccionado is not None:
        # Rastrear el punto usando el método de Lucas-Kanade
        p0 = np.array([punto_seleccionado], dtype=np.float32)
        p1, st, err = cv2.calcOpticalFlowPyrLK(gris_anterior, gris, p0, None, **lk_params)

        # Actualizar la posición del punto seleccionado si el seguimiento es exitoso
        if st is not None and st[0][0] == 1:
            punto_seleccionado = (int(p1[0][0]), int(p1[0][1]))

```

Hasta aquí la función hace lo siguiente:

1) **Declaración de Variables Globales:**
    - Se declara que las variables gris_anterior, punto_seleccionado, y punto_interseccion son globales. Esto es necesario para poder modificar estas variables dentro de la función.

2) **Obtención del Valor del Slider:**
    - Obtiene el valor actual del slider (slidebar_canny) que se utiliza como umbral para el detector de bordes Canny.

3) **Captura de Fotograma:**
    - Lee un fotograma de la cámara utilizando cap.read().
    - Verifica si la operación de lectura (ret) fue exitosa. Si no, imprime un mensaje de error y sale de la función.

4) **Preprocesamiento de la Imagen:**
    - Convierte el fotograma a escala de grises (gris).
    - Aplica un desenfoque gaussiano (desenfoque) para reducir el ruido.
    - Utiliza el operador Canny para detectar bordes en la imagen (bordes).

5) **Rastreo del Punto Seleccionado:**
    - Comprueba si hay un punto seleccionado (punto_seleccionado).
      Si hay un punto seleccionado, utiliza el método de Lucas-Kanade para rastrear ese punto en el fotograma actual. Actualiza la posición del punto seleccionado si el seguimiento es exitoso.

```python
# Dibujar el punto en la imagen
if punto_seleccionado is not None:
    cv2.circle(frame, punto_seleccionado, 5, (0, 255, 0), -1)

# Detectar líneas en la imagen usando HoughLinesP
lineas = cv2.HoughLinesP(bordes, 1, np.pi / 180, threshold=50, minLineLength=50, maxLineGap=10)

# Dibujar las líneas en la imagen
if lineas is not None:
    for linea in lineas:
        x1, y1, x2, y2 = linea[0]
        cv2.line(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)

# Buscar intersección de las líneas
if lineas is not None:
    puntos_interseccion = encontrar_puntos_interseccion(lineas)
else:
    puntos_interseccion = []

# Mostrar las intersecciones en la imagen
for punto_interseccion in puntos_interseccion:
    cv2.circle(frame, punto_interseccion, 5, (255, 0, 0), -1)

# Mostrar la imagen con el punto rastreado y las líneas
mostrar_imagen(frame)

# Actualizar la imagen anterior
gris_anterior = gris.copy()

# Actualizar las coordenadas
if punto_seleccionado is not None:
    actualizar_coordenadas(punto_seleccionado[0], punto_seleccionado[1])

# Llamar recursivamente para procesar el siguiente fotograma después de 1 segundo
ventana.after(1000, procesar_imagen)
```

Esta parte del código se encarga de:

6) **Dibujar Punto Seleccionado:**
   - Si hay un punto seleccionado, dibuja un círculo verde en la imagen en esa posición.

7) **Detectar y Dibujar Líneas:**
   - Utiliza la transformada de Hough para detectar líneas en la imagen.
   - Dibuja las líneas detectadas en rojo.

8) **Buscar e Imprimir Intersecciones:**
   - Encuentra y dibuja los puntos de intersección entre las líneas en azul.

9) **Actualizar Imagen y Coordenadas:**
   - Muestra la imagen actualizada con el punto, líneas e intersecciones.
   - Actualiza la imagen anterior y las coordenadas del punto seleccionado.
   - Llamada Recursiva:
   - Programa una llamada recursiva para procesar el siguiente fotograma después de 1 segundo.

encontrar_puntos_interseccion(lineas):

```python
# Función para encontrar todos los puntos de intersección de líneas
def encontrar_puntos_interseccion(lineas):
    puntos_interseccion = []

    for i in range(len(lineas)):
        for j in range(i + 1, len(lineas)):
            x1, y1, x2, y2 = lineas[i][0]
            x3, y3, x4, y4 = lineas[j][0]

            # Calcular la intersección de las líneas
            det = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4)
            if det != 0:
                px = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4 - y3 * x4)) / det
                py = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4 - y3 * x4)) / det
                puntos_interseccion.append((int(px), int(py)))

    return puntos_interseccion
```

La función encontrar_puntos_interseccion calcula los puntos de intersección entre todas las combinaciones de líneas proporcionadas. Para cada par de líneas, determina si son no paralelas y, en caso afirmativo, calcula las coordenadas de intersección. Luego, devuelve la lista de todos los puntos de intersección encontrados.

ventana:

```python
# Crear la ventana principal
ventana = tk.Tk()
ventana.title("Seguimiento de Punto")

# Iniciar la cámara
cap = cv2.VideoCapture(0)  # 0 para la cámara predeterminada

# Parámetros para el seguimiento óptico de Lucas-Kanade
lk_params = {
    'winSize': (15, 15),
    'maxLevel': 2,
    'criteria': (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03),
}

# Panel para mostrar la imagen
panel_imagen = tk.Label(ventana)
panel_imagen.pack()

# Slidebar para ajustar el umbral de Canny
slidebar_canny = tk.Scale(ventana, from_=0, to=200, orient=tk.HORIZONTAL, label="Umbral Canny", length=300, resolution=1)
slidebar_canny.set(50)  # Valor inicial
slidebar_canny.pack(pady=10)

# Label para mostrar las coordenadas
texto_coordenadas = tk.StringVar()
label_coordenadas = tk.Label(ventana, textvariable=texto_coordenadas, font=("Arial", 12))
label_coordenadas.pack(pady=10)

# Inicializar el punto seleccionado (None al principio)
punto_seleccionado = None
```

```python
# Inicializar la imagen anterior para el seguimiento óptico
ret, frame = cap.read()
gris_anterior = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Función para cerrar la cámara cuando se cierra la ventana
def cerrar_ventana():
    cap.release()
    ventana.destroy()

ventana.protocol("WM_DELETE_WINDOW", cerrar_ventana)

# Manejar clic en la imagen
def clic_en_imagen(event):
    global punto_seleccionado
    x, y = event.x, event.y
    punto_seleccionado = (x, y)
    actualizar_coordenadas(x, y)

panel_imagen.bind("<Button-1>", clic_en_imagen)

# Iniciar el procesamiento de la imagen
procesar_imagen()

ventana.mainloop()
```

La sección 'ventana' del código realiza las siguientes acciones: Crea la ventana principal de la interfaz gráfica de usuario con Tkinter, inicia la captura de video (cámara) utilizando OpenCV, configura widgets como panel_imagen, slidebar_canny, y label_coordenadas, inicializa el punto seleccionado y la imagen anterior, define funciones para cerrar la ventana y manejar clics en la imagen, inicia el procesamiento de imágenes llamando a procesar_imagen(), y lanza el bucle principal de la interfaz gráfica con ventana.mainloop().