Tecnológico de Monterrey

Campus Ciudad de México

Fundamentación de Robótica

TE3001B.160

Act 3. Signal Identification

Aylín Millán Cázares - A01655861

Olivia Navarrete Carrasco - A01654903

Profesor:

Diego López Bernal

Para la elaboración del algoritmo se trabajó con OpenCv para la identificación de señales de tránsito. Dentro del código abordó el funcionamiento de Sift así como de Contour Approximation. Los pasos para realizar el algoritmo fueron los siguientes.

1. Definición de la función para clasificar la figura según el número de vértices

- Se calcula el perímetro del contorno utilizando **cv2.arcLength**. para obtener la longitud del contorno.
- Se utilizó **cv2.approxPolyDP**, con esta se pretendió aproximar el contorno a una forma con menos vértices según un umbral especificado.
- Se clasificó el número de vértices basado en las siguientes condiciones:
    - Si son 3 vértices es un triángulo (diferencia entre triángulo normal e invertido).
    - Si son 8 vértices es un Octágono.
    - Si son más de 8 vértices es un Círculo.
    - En el caso de tener menos de 3 o entre 4 y 7 vértices es Indefinido.

```python
def detectar_forma(contorno):

    peri = cv2.arcLength(contorno, True)
    approx = cv2.approxPolyDP(contorno, 0.04 * peri, True)
    vertices = len(approx)

    # Clasificar la forma según el número de vértices
    if vertices == 3:
        sorted_vertices = sorted(approx, key=lambda x: x[0][1])
        if sorted_vertices[0][0][1] < sorted_vertices[1][0][1] and
sorted_vertices[0][0][1] < sorted_vertices[2][0][1]:
            return "Triángulo Normal"
        else:
            return "Triángulo Invertido"
    elif vertices == 8:
        return "Octágono"
    elif vertices > 8:
        return "Círculo"
    else:
        return "Indefinido"
```

2. Definición de la función para detectar señales de tránsito

- Se utilizó **cv2.cvtColor** para convertir la imagen a escala de grises.
- Se crearon puntos clave y descriptores en la imagen.
- Se definieron los parámetros para el emparejamiento de características.
- Se encontraron las coincidencias utilizando el método **KNN** y se filtraron todas las coincidencias positivas .
- Se seleccionó la referencia con las coincidencias positivas y se válido si cumple el umbral para considerarse una señal detectada.

Python

```
def detectar_senales(imagen, referencias):
    gris = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)

    # Detectar características SIFT en la imagen de la cámara
    sift = cv2.SIFT_create()
    kp, des = sift.detectAndCompute(gris, None)

    if des is None or len(des) == 0:
        return imagen, None

    # Configuración del match
    index_params = dict(algorithm=1, trees=5)
    search_params = dict(checks=50)
    flann = cv2.FlannBasedMatcher(index_params, search_params)
    num_coincidencias = []
```

3. Detección de coincidencias

- Se inicializa la detección de coincidencias
- Se utilizó el algoritmo **flann.knnMatch** para encontrar los dos mejores emparejamientos
- Se inicializa una lista (**good_matches**)para almacenar los emparejamientos positivismo que se encontraron
- Se utilizó **np.argmax** para encontrar el índice del mayor número de coincidencias buenas. Si el número máximo de buenas coincidencias es menor que 10, se establece como **mejor_indice** como ese índice; de lo contrario, se establece como **None.**
- Para la detección del contorno se utiliza un umbral en la imagen para pasarlo a escala de grises
- Para cada contorno encontrado, se determina su forma mediante la función **detectar_forma**(contorno).
    - Si la forma es una de las formas específicas ("Triángulo Normal", "Triángulo Invertido", "Octágono", "Círculo"), se dibuja el contorno en la imagen original (imagen).

Python

```
# Encontrar coincidencias entre la cámara y las imágenes de referencia

    if num_coincidencias[indice_max_coincidencias] >= 10:
        mejor_indice = indice_max_coincidencias
    else:
        mejor_indice = None
```

```
    # Detectar contornos
    _, umbral = cv2.threshold(gris, 240, 255, cv2.THRESH_BINARY)
    contornos, _ = cv2.findContours(umbral, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    for contorno in contornos:
        forma = detectar_forma(contorno)
        if forma in ["Triángulo Normal", "Triángulo Invertido", "Octágono",
"Círculo"]:
            cv2.drawContours(imagen, [contorno], -1, (0, 255, 0), 2)
            x, y, w, h = cv2.boundingRect(contorno)

            # Regresar imagen y mejor índice si se detecta una forma válida
            return imagen, mejor_indice

    # Si no se detecta ninguna forma válida
    return imagen, mejor_indice
```

4. Lectura de archivos de referencia

- Se definió una lista con los nombres de los archivos de referencias (**nombres_archivos**)
- Se utilizó un for para iterar sobre cada nombre en la lista
- Se construyó la ruta del archivo de la imagen de referencia
- Se utilizó **cv2.imread** para leer la imagen desde el archivo.
- Dentro del ciclo if se verificó si el **ref_img** es “None”, en caso de serlo indica que la imagen no se pudo cargar y se imprime un mensaje de error.

Python

```
nombres_archivos = ["stop", "giveaway", "straigth", "turnaround",
"turnleft", "turnright", "workinprogress"]


referencias = []

# Iterar sobre los nombres de archivos y cargar las imágenes de referencia
for nombre in nombres_archivos:

    ruta_imagen = f"{nombre}.jpg"
```

```
    ref_img = cv2.imread(ruta_imagen, 0)

    # Verificar si la imagen se ha leído correctamente
    if ref_img is None:
        print(f"No se pudo leer la imagen : {ruta_imagen}")
        continue
```

5. Características SIFT

- Usando **cv2.SIFT_create** se crea un objeto SIFT (Scale-Invariant Feature Transform), para detectar y describir características locales en las imágenes.
- Se utilizó **sift.detectAndCompute** para la detección de puntos clave y calcular los descriptores en la imagen de referencia
- Dentro del condicional se verificó si el descriptor (**ref_des**) es" None", en caso de serlo no se encontraron características SIFT en la imagen de referencia.
    - Si no se encuentran descriptores, se imprime un mensaje de error indicando que no se encontraron características en la imagen de referencia (**ruta_imagen)**

```
Python
   # Detectar características SIFT en la imagen de referencia
    sift = cv2.SIFT_create()
    ref_kp, ref_des = sift.detectAndCompute(ref_img, None)

    # Verificar si se detectaron características
    if ref_des is None:
        print(f"No se encontró referencia: {ruta_imagen}")
        continue

    # Almacenar los puntos clave
    referencias.append((ref_kp, ref_des))
```

6. Verificación de imágenes de referencia

- Se imprimió un mensaje para indicar si se han cargado correctamente las imágenes de referencia
- Se abre la transmisión de video desde la cámara utilizando OpenCV.

```
Python
# Verificar si se han cargado correctamente las imágenes de referencia
if len(referencias) == 0:
    print("No se han cargado correctamente las imágenes de referencia")
else:
    print("Se han cargado correctamente las imágenes de referencia")
```

```
cap = cv2.VideoCapture(0)
```

7. Procesamiento en Tiempo Real de Señales de Tráfico desde una Cámara

- Dentro del ciclo **while** se capturó de forma continua los fotogramas de la cámara, para detectar señales de tráfico utilizando la función **detectar_senales**, y mostrar el resultado en una ventana de visualización en tiempo real.
- Se capturó un fotograma (frame) de la cámara. La variable ret indica si la captura fue exitosa o no.
- Dentro del ciclo if si ret es False, significa que no se pudo capturar correctamente el fotograma.
    - Si **indice_senal_detectada** is not None significa que se detectó una señal de tráfico , se muestra el nombre de la señal en la esquina superior izquierda del fotograma utilizando **cv2.putText**.
- Se muestra el fotograma procesado en una ventana titulada "Detector de Señales de Tránsito".

Python

```
while True:

    ret, frame = cap.read()
    if not ret:
        break

    # Detectar la referencia en la imagen de la camara
    frame, indice_senal_detectada = detectar_senales(frame, referencias)

    # Si se detectó una señal mostrar el nombre
    if indice_senal_detectada is not None:
        nombres_senales = ["Stop", "Giveaway", "Straigth", "Turnaround",
"Turn Left", "Turn Right", "Work in Progress"]
        nombre_senal = nombres_senales[indice_senal_detectada]
        cv2.putText(frame, nombre_senal, (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 255, 0), 2)


    cv2.imshow('Detector de Señales de Tránsito', frame)


    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```