Tecnológico de Monterrey

TE3001B

Implementación de Robótica Inteligente

Grupo 561

Act 3: Signal Identification

Victoria Rodríguez de León	A01656328	A01656328@tec.mx
Juan Antonio Mancera Velasco	A01662634	A01662634@tec.mx

Diego López Bernal

20 de mayo de 2023

Campus Ciudad de México

Instituto Tecnológico y de Estudios Superiores de Monterrey

Escuela de Ingeniería y Ciencias

Instrucciones

- Basado en los temas revisados en clase, identificar las señales necesarias utilizando cualquier algoritmo de visión por computadora. Se pueden usar técnicas como máscaras, contornos, SIFT, ORB, MSE, RMSE, entre otras.
- Describir detalladamente la metodología seguida.

Metodología

- Con base en las instrucciones, se buscó generar un algoritmo capaz de detectar una serie de señales de tránsito utilizando visión computacional.
- Se decidió usar SIFT después de probar con otros algoritmos de detección de imágenes por su capacidad de detectar las mismas características en una imagen aunque esté escalada o rotada, su resistencia a cambios en las condiciones de iluminación y en general la correspondencia precisa que surge de la detección de los puntos por parte del algoritmo.

I. Importación de librerías

Se importaron las librerías cv2, numpy y os. “cv2” es para acceder a las funciones de OpenCV para las funciones utilizadas en aplicaciones de visión por computadora, “numpy” se utiliza para realizar operaciones matemáticas y “os” sirve para manipulación de rutas de archivos.

```python
import cv2
import numpy as np
import os
```

II. Configuración de rutas y nombres de archivos

Se creó una variable llamada “template_folder”, que contiene la ruta a la carpeta donde se almacenaron las señales de tránsito de referencia. Asimismo, se creó una variable “template_names” con los nombres de las imágenes de las plantillas.

```python
template_names = [ 'giveway.jpg', 'workinprogress.jpg', 'turnright.jpg',
'turnleft.jpg', 'turnaround.jpg', 'straigth.jpg', 'stop.jpg' ]
```

III. Almacenamiento de plantillas

Se inicializó un diccionario vacío llamado “templates”. Después de creó un bucle for que itera sobre el diccionario que contiene los nombres de las imágenes, construyendo la ruta completa del archivo combinando la carpeta de las plantillas y el nombre del archivo. Después se hace una lectura de la imagen en escala de grises usando la

función *imread()* de OpenCV, se verifica la carga correcta de la imagen y se realiza el almacenamiento de la plantilla en el diccionario creado al principio. Finalmente, se imprime en terminal si la imagen se cargó correctamente o no.

```python
templates = {}
for name in template_names:
    path = os.path.join(template_folder, name)
    template = cv2.imread(path, 0)     if template is not None:
        templates[name.split('.')[0]] = template
        print(f"Imagen {name} cargada correctamente.")
    else:
        print(f"Error al cargar la imagen {name}.")
```

IV. Inicialización de SIFT

Se crea un objeto SIFT para detectar puntos clave y descriptores. Se acotó el número máximo de características clave a 500. Se decidió limitar el número de características para mejorar la eficiencia del procesamiento, pensando en la aplicación en la Jetson Nano del Puzzlebot.

```python
sift = cv2.SIFT_create(nfeatures=500)
```

V. Detección de keypoints y generación de descriptores

Se realizó una detección de puntos clave y descriptores en cada imagen de la plantilla de las señales de tránsito. Para ello, se inicializó un diccionario vacío llamado “template_keypoints_descriptors” para almacenar los puntos clave y los descriptores de cada plantilla. Después se generó un bucle *for* que itera sobre el diccionario “templates”, que contiene las plantillas de señales de tránsito. “name” es el nombre de la señal y “template” es el valor (la imagen de la plantilla). Sobre eso, se realiza la detección de los puntos clave y el cálculo de los descriptores usando la función “sift.detectAndCompute()”.

Con los puntos clave se obtienen las características distintivas de la imagen que son invariables a escala y rotación. Con esto se genera un descriptor para cada punto clave, que es un vector que describe la apariencia del punto clave basado en el patrón de gradientes de intensidad en una vecindad alrededor del punto clave. Finalmente, se almacenan los valores en el diccionario creado y se imprime un mensaje de validación.

```python
template_keypoints_descriptors = {}
for name, template in templates.items():
    keypoints, descriptors = sift.detectAndCompute(template, None)
    template_keypoints_descriptors[name] = (keypoints, descriptors)
print(f"Características detectadas en la plantilla {name}: {len(keypoints)}
puntos clave.")
```

VI. Inicialización de la cámara

Con la función "VideoCapture" de CV2, se inicializa la cámara. Se colocó 0 como parámetro para indicar que es la cámara predeterminada. Después se redujo el tamaño de la resolución a 320x240 para reducir la cantidad de datos que el sistema necesita procesar, mejorando la fluidez del video y la capacidad de respuesta. Esto se hizo pensando en la aplicación en la Jetson Nano del Puzzlebot.

```python
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
```

VII. Función para emparejar características SIFT en cada cuadro.

Se desarrolló la función match_sift para detectar la mejor coincidencia de una señal de tránsito en un cuadro de video, comparando las características clave detectadas en el cuadro con las características clave de varias plantillas de señales de tránsito.
El algoritmo se puede describir de manera más concreta en los siguientes pasos:

1. Convierte el cuadro capturado a escala de grises.
    - Usando la función de openCV "cv2.cvtColor()". Este paso es necesario porque el algoritmo SIFT opera en imágenes en escala de grises para detectar características clave.
2. Detecta puntos clave y descriptores en el cuadro.
    - Usando la función de openCV "sift.detectAndCompute()".
3. Itera sobre las plantillas y sus descriptores correspondientes.
    - Para cada plantilla, se recuperan los puntos clave y los descriptores almacenados previamente.
4. Encuentra coincidencias utilizando el emparejador de características.
    - Usando la función bf.knnMatch(descriptors_template, descriptors_frame, k=2) para encontrar las dos mejores coincidencias para cada descriptor de la plantilla en los descriptores del cuadro.
5. Filtra las buenas coincidencias basándose en la distancia.
    - Esto lo hace aplicando el criterio de Lowe. Se adaptó para que seleccionara las coincidencias donde la distancia de la mejor coincidencia es menor al 75% de

la distancia de la segunda mejor coincidencia, eliminando coincidencias ambiguas.

6. Si encuentra suficientes buenas coincidencias, utiliza la homografía para determinar la transformación geométrica entre los puntos clave de la plantilla y el cuadro.
7. Devuelve el nombre de la plantilla que tiene el mayor número de buenas coincidencias.

```python
def match_sift(frame, template_keypoints_descriptors):
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    keypoints_frame, descriptors_frame = sift.detectAndCompute(gray_frame,
None)

    best_match = None
    best_matches_count = 0
    min_good_matches = 15
    for name, (keypoints_template, descriptors_template) in
template_keypoints_descriptors.items():

        if descriptors_template is not None and descriptors_frame is not
None:
            matches = bf.knnMatch(descriptors_template, descriptors_frame,
k=2)
            good_matches = [m for m, n in matches if m.distance < 0.75 *
n.distance]

            if len(good_matches) > min_good_matches:
                src_pts = np.float32([keypoints_template[m.queryIdx].pt for
m in good_matches]).reshape(-1, 1, 2)
                dst_pts = np.float32([keypoints_frame[m.trainIdx].pt for m
in good_matches]).reshape(-1, 1, 2)

                M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,
5.0)
                if M is not None:
                    matches_mask = mask.ravel().tolist()
                    if sum(matches_mask) > best_matches_count:
                        best_matches_count = sum(matches_mask)
                        best_match = name

    return best_match if best_matches_count >= min_good_matches else None
```

VIII. Etapa Final.

Para la etapa final, se capturaron los cuadros de video en tiempo real de la cámara que se inicializó previamente, procesando cada cuadro para detectar las señales de tránsito utilizando la función “match_sift” y mostrando el resultado en una ventana de visualización. Se especificó un intervalo de procesamiento para que fuera cada 5 cuadros.

Para tener una validación del método, se utilizó la función “cv2.putText()” para que, si se encuentra una coincidencia, se dibuje el nombre de la señal de tráansito en la imagen del cuadro. El texto se dibuja en la posición (50,50) con una fuente específica y de color verde.

```python
frame_count = 0
process_frame_interval = 5

while True:
    ret, frame = cap.read()
    match = match_sift(frame, template_keypoints_descriptors)  #

    if match is not None:
        cv2.putText(frame, match, (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 0), 2)
        cv2.imshow('Detected Traffic Sign', frame)

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

cap.release()
cv2.destroyAllWindows()
```