

Act 3. Signal Detection

Ahedo Reyes Ana Lucía
Implementación de Robótica Inteligente
Tecnológico de Monterrey
Ciudad de México, México
A01661890@tec.mx

Mercandetti Hernández Zoe
Implementación de Robótica Inteligente
Tecnológico de Monterrey
Ciudad de México, México
A01656465@tec.mx

I. METODOLOGÍA

Este código implementa un sistema de detección y de reconocimiento de 7 señales de tráfico en tiempo real.

Lo primero que se hace es meter todas las señales en una carpeta. Con esto se podrán analizar varias señales en tiempo real. Conjunto a esto, se usa la biblioteca "os" para tener acceso y manipulación de las imágenes de la carpeta creada. La función "os.listdir(folder_path)" se emplea para listar todos los archivos presentes en el directorio especificado por folder_path, en este caso llamado "imagenes", devolviendo una lista con los nombres de estos archivos. Posteriormente, se utiliza "os.path.join(folder_path, file)" para construir rutas de archivo completas combinando el directorio base con cada nombre de archivo file. Además, se utiliza "os.path.splitext(file)[0]" para obtener los nombres de los archivos sin sus extensiones, donde os.path.splitext(file) devuelve una tupla con el nombre del archivo y su extensión, y [0] selecciona solo el nombre.

```
folder_path = 'imagenes'
image_paths = [os.path.join(folder_path, file) for file in os.listdir(folder_path)]
signal_names = [os.path.splitext(file)[0] for file in os.listdir(folder_path)] # Ob
```

Fig. 1. Os

Posteriormente, se crea un sustractor de fondo utilizando el algoritmo K-Nearest Neighbors (KNN) a través de la función "cv2.createBackgroundSubtractorKNN()", lo que permite identificar el primer plano en movimiento en el video. La captura de video se realiza utilizando la cámara web del sistema con "cv2.VideoCapture(0)". Se verifica si la cámara está activa con "cap.isOpened()" y en caso de no estarlo, el código manda un mensaje de error.

Posteriormente, se procede a cargar las imágenes de señales de tráfico desde un directorio especificado "imagenes". Cada imagen es procesada para extraer sus características distintivas utilizando el detector "ORB (cv2.ORB_create)", configurado para detectar hasta 1500 características por imagen. Estas características se utilizan como descriptores para comparar las imágenes capturadas por la cámara en tiempo real.

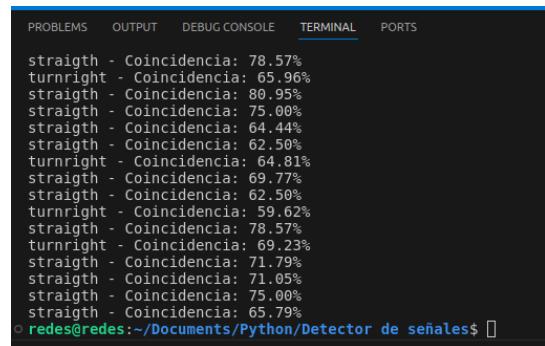
```
reference_images = []
reference_descriptors = []
orb = cv2.ORB_create(nfeatures=1500)
```

Fig. 2. Coincidencias

En el bucle principal, se captura cada cuadro de video de la cámara. Se aplica un proceso de substracción de fondo para resaltar el primer plano en movimiento. Luego, se detectan los puntos clave y descriptores en el cuadro actual utilizando ORB.

La comparación entre los descriptores del cuadro y los de las imágenes de referencia se realiza con "cv2.BFMatcher". Esto permite identificar coincidencias entre los descriptores de las señales de tráfico predefinidas y los elementos detectados en el cuadro. Las coincidencias que superen un umbral del 50% son consideradas válidas. Para estas coincidencias válidas, se verifica la correspondencia geométrica utilizando homografía, asegurando que la configuración espacial de los puntos clave coincida.

Si se confirma una coincidencia, se muestra el nombre de la señal detectada en el video y abre una ventana con la imagen correspondiente de la señal. Cada vez que se detecta una señal, también se despliegan los datos del porcentaje de coincidencia en la terminal, tal y como lo muestra la siguiente figura:



The terminal window shows a list of signal names and their corresponding coincidence percentages. The columns are labeled PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The data is as follows:

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
straigth			Coincidencia: 78.57%	
turnright			Coincidencia: 65.96%	
straigth			Coincidencia: 80.95%	
straigth			Coincidencia: 75.00%	
straigth			Coincidencia: 64.44%	
straigth			Coincidencia: 62.50%	
turnright			Coincidencia: 64.81%	
straigth			Coincidencia: 69.77%	
straigth			Coincidencia: 62.50%	
turnright			Coincidencia: 59.62%	
straigth			Coincidencia: 78.57%	
turnright			Coincidencia: 69.23%	
straigth			Coincidencia: 71.79%	
straigth			Coincidencia: 71.05%	
straigth			Coincidencia: 75.00%	
straigth			Coincidencia: 65.79%	

redes@redes:~/Documents/Python/Detector de señales\$

Fig. 3. Porcentaje de coincidencia en terminal

En caso de no detectar coincidencias significativas solamente se visualiza el video sin la detección de señales.

El ciclo de procesamiento se repite hasta que se finaliza el programa presionando Esc.

II. RESULTADOS

A continuación se presentan los resultados obtenidos para cada una de las señales:

1. Give Way: En la figura 4, se puede observar como la señal tiene en promedio un nivel de coincidencia del 75% con la señal detectada en el video.

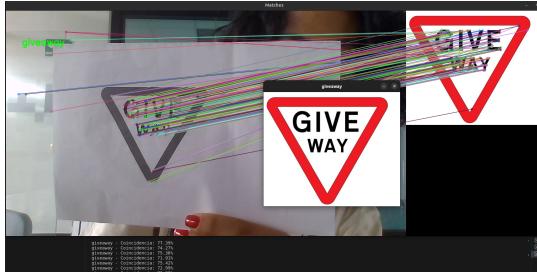


Fig. 4. Give Way

2. Turn Around: En la siguiente figura se puede notar como se identifica la señal de "return".

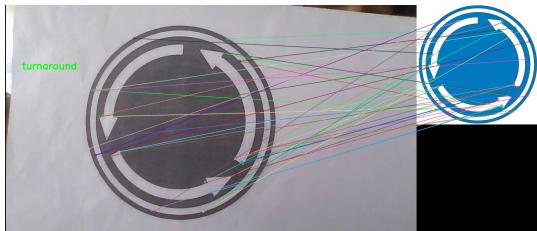


Fig. 5. Turn Around

3. Turn Right: La figura 6 ilustra como se identificó la vuelta a la derecha. En esta detección se tuvo una pequeña complicación ya que se confundía con la de "giro a la izquierda", sin embargo, se logró detectar con éxito.

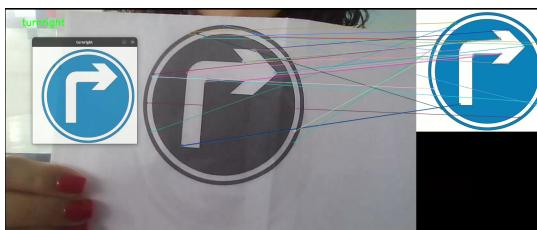


Fig. 6. Turn right

4. Turn Left: En esta se presentó el mismo problema que en el caso anterior, pero se logró diferenciar entre ambas señales. La figura 7 ilustra el resultado de identificación.

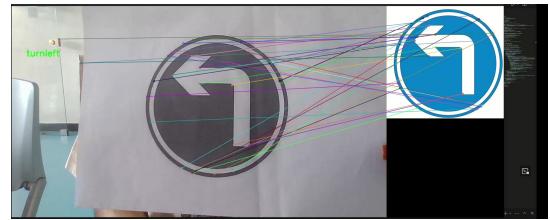


Fig. 7. Turn Left

5. Work in Progress: Tal y como lo muestra la figura 8, esta señal fue de las más sencillas de detectar. En la terminal se muestra que se tuvo, en promedio, un 80% de coincidencia entre la imagen vista de la cámara y la señal de referencia.

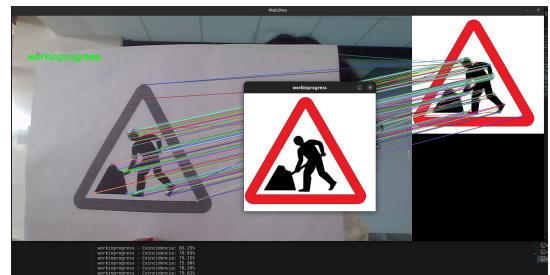


Fig. 8. Work in Progress

6. Stop: Esta señal, al igual que la anterior, fue de suma facilidad de identificar. En terminal mostraba un 70% de coincidencia.

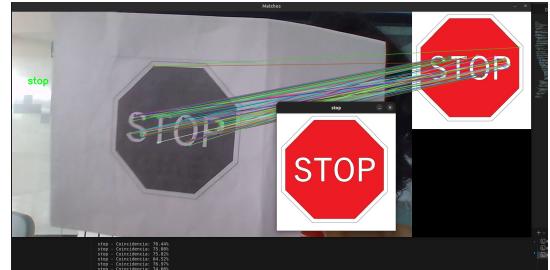


Fig. 9. Stop

7. Straight: En esta última señal, se tuvieron distintas complicaciones al momento del reconocimiento, ya que, o no la identificaba o se confundía con las otras dos flechas. Sin embargo, como lo muestra la figura 10, se logró la identificación.

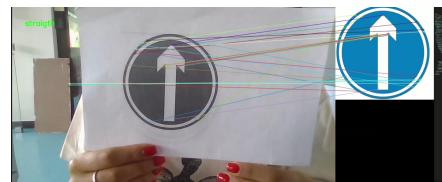


Fig. 10. Straight

III. VIDEO DE DEMOSTRACIÓN

<https://youtu.be/e40IrEKc8eQ>

IV. REPOSITORIO DEL CÓDIGO

https://github.com/analuciaahedo/Signal_detection