Universidad De Alicante

Escuela Politécnica Superior

Práctica 3: Programación de tareas en robots móviles

Grado en Ingeniería Robótica

Robots móviles

Autor: Erica Tébar Máximo

11 de enero de 2025

1. Análisis de detección

Para el seguimiento se emplea una banda slap amarilla brillante y reflectante con letras color gris:

Imagen 1: Banda de seguimiento

La finalidad de este apartado es la de documentar las pruebas realizadas para obtener la mejor detección de esta.

Los ajustes de detección varían entre cámaras, por lo que es necesario realizarlos específicamente con la cámara del TurtleBot. Esto asegura que los parámetros sean compatibles con las características propias del dispositivo.

Los casos a probar en este análisis son los siguientes:

- **HSV**: En este espacio de color, **H** representa el matiz (hue), **S** la saturación y **V** el valor (brillo). Sus posibles valores, según OpenCV en Python, están normalizados de igual manera que en HSL
- **BGR**: Este espacio de color es utilizado directamente por OpenCV al cargar imágenes o video. Corresponde a los colores primarios en inglés (Blue, Green, Red), y cada componente varía entre [0, 255].
- **HSL**: El componente **H** es idéntico al de HSV, mientras que **S** (saturación) no necesariamente coincide debido a diferencias en la definición matemática del espacio de color. El componente **L** representa la luminosidad. Sus posibles valores en OpenCV están normalizados entre los rangos:

  - H: [0, 179]
  - S: [0, 255]
  - L: [0, 255]

- **Escala de grises**: En este caso, solo se toma en cuenta la intensidad de luz, lo que equivale a un único parámetro. Este enfoque limita la capacidad de detección específica, ya que no considera información sobre color.

En primer lugar, se realizó una prueba preliminar con una cámara externa al TurtleBot (la del ordenador), ya que no había tiempo ni disponibilidad para probar con el TurtleBot real. Para ello, se empleó el código `pixel_color.py`, que devuelve el valor del píxel donde se posiciona el ratón. Este proceso permitió obtener una aproximación inicial de los valores buscados, mediante ingeniería inversa. Un dato interesante observado fue que los valores del píxel varían constantemente, pero tienden a concentrarse en un rango definido, a menos que la banda reflectante se mueva y cause destellos de luz.

A continuación, se implementó un script denominado `color_space.py`, que alterna entre diferentes espacios de color, ajustando sus parámetros según los resultados obtenidos. Es importante destacar que, para utilizar este código, es necesario intercalar con el espacio BGR; es decir, no se permite una visualización directa entre diferentes espacios de color sin pasar previamente por BGR. Esta restricción se implementa con el fin de mantener la simplicidad de un código diseñado para pruebas rápidas. Tras probar cada técnica, se llegó a las siguientes conclusiones:

Escala de grises: Este método resultó ineficaz. Detecta reflejos y contornos, pero no permite aislar únicamente la banda reflectante, lo que imposibilita un resultado funcional. A continuación, se muestra una imagen ilustrativa de la detección fallida:

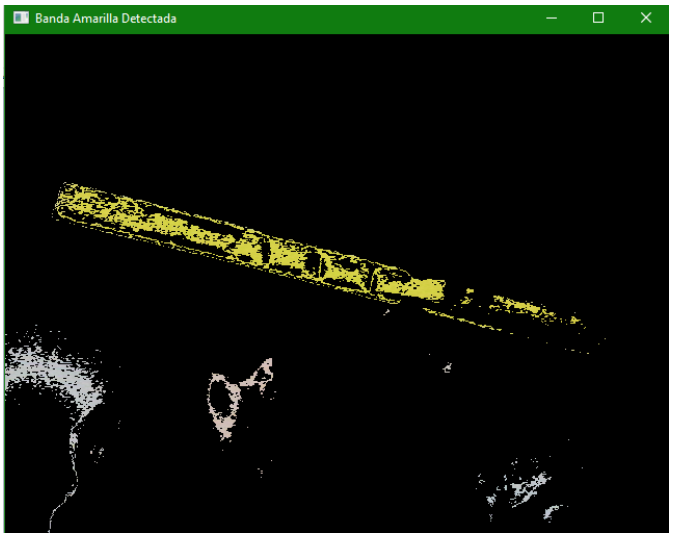

Imagen 2: Detección fallida en escala de grises

BGR: En este espacio de color, el amarillo es una combinación de azul y verde, lo que coincide con los valores obtenidos en `a.py`, donde se observaron valores altos para azul y verde, y un valor menor para rojo. Aunque se consigue detectar la banda reflectante, también se detectan objetos con tonalidades amarillas similares, lo que implica el requerimiento de combinar información de color y brillo. La detección obtenida se muestra en la siguiente figura:

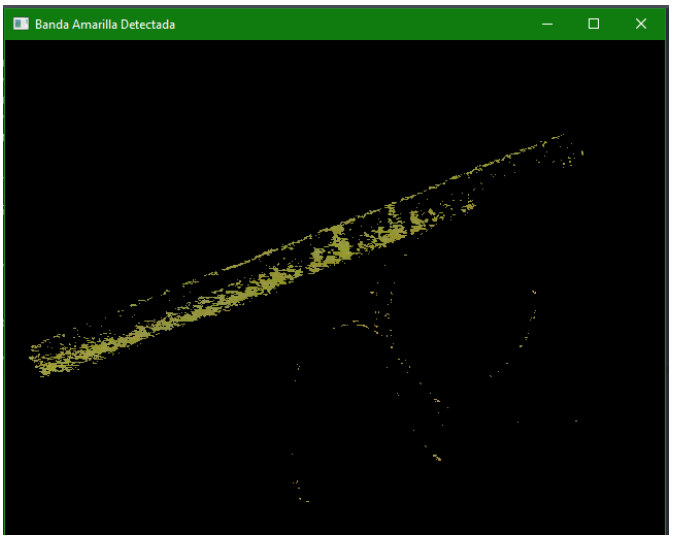

Imagen 3: Detección en espacio BGR

HSV y HSL: Como era de esperarse, estos espacios de color ofrecieron mejores resultados al considerar tanto el color (clave para identificar una banda amarilla intensa) como el brillo (para evitar falsas detecciones en objetos amarillos no reflectantes). Entre ambos:

- En **HSV**, se detectaron más puntos de la banda, pero, en ocasiones, no las letras impresas sobre ella.
- En **HSL**, se detectaron las letras, aunque la banda fue segmentada en partes con puntos más separados entre sí.

Ninguno de los 2 métodos funciona a la perfección, ambos dependen de las condiciones de reflectancia, luminosidad, tipo de luz y de características propias de la banda reflectante, como las letras en ella. La luminosidad es una de las principales fuentes de variación entre ambos sistemas de detección, mientras que hace funcionar mejor la detección por HSV, empeora la HSL; Sin embargo, descendiendo hasta cierto valor de luminosidad, son inviables todas las detecciones empleadas.

La siguiente figura muestra una comparación entre ambas detecciones:

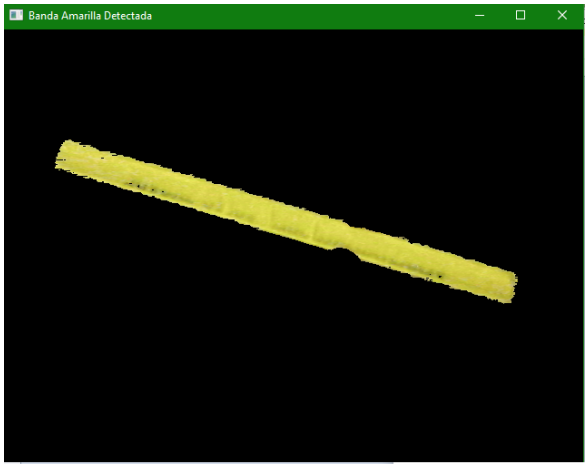

Imagen 4: Detección en HSV.

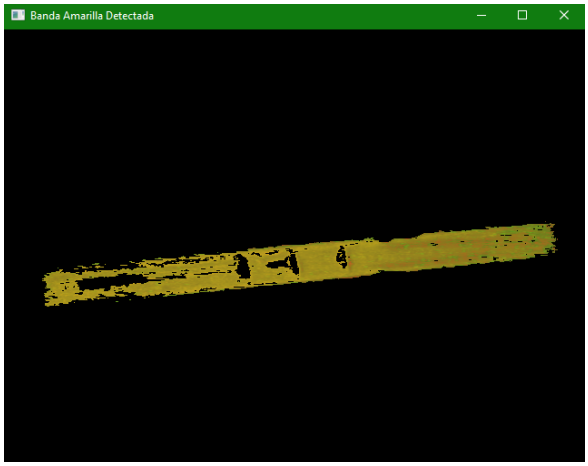

Imagen 5: Detección en HSL.

Finalmente se propone un sistema adaptativo en función de la luminosidad de la estancia empleando la detección HSV. HSV supone la mejor detección ya que es la que más puntos de la banda detecta, por lo que, en base a un valor calculado como media de los valores de intensidad de los píxeles de la imagen, se emplea un rango de detección u otro; esto queda implementado de manera muy sencilla en el script **HSV_adaptative.py**, ya que tener un canal que indique directamente la luminosidad hace muy directa la adaptación

a esta.

Además, se identificó una ambivalencia inherente a la banda reflectante: su naturaleza reflectiva dificulta la detección en ciertas partes debido a los reflejos, pero también reduce el riesgo de confusión con otros objetos amarillos. Como medida adicional, se propone implementar un filtro de forma rectangular para descartar falsos positivos y mejorar la precisión del sistema en el entorno de laboratorio (evitando problemas con luces, superficies metálicas pulidas, ropa reflectante, u otros objetos brillantes y amarillos).

Una vez detectada la banda por su color, eliminar falsos positivos por forma presenta un desafío demasiado grande, puesto que:

- La banda varía su forma según la distancia a la cámara.
- Los puntos del contorno no son uniformes, es difícil detectar correctamente las esquinas y el cuadrado o rectángulo formado por esas esquinas puede ser coincidente con casi cualquier otro objeto.

Por lo que se limita a escoger el contorno más grande obtenido, verificando que su área esté dentro de ciertos límites predefinidos, los cuales se determinaron mediante pruebas experimentales. Una vez identificado el contorno válido, se dibuja su bounding box (un rectángulo que delimita el contorno) y se calcula el punto medio de dicho rectángulo. Esto se implementa en el script **band_detection**.

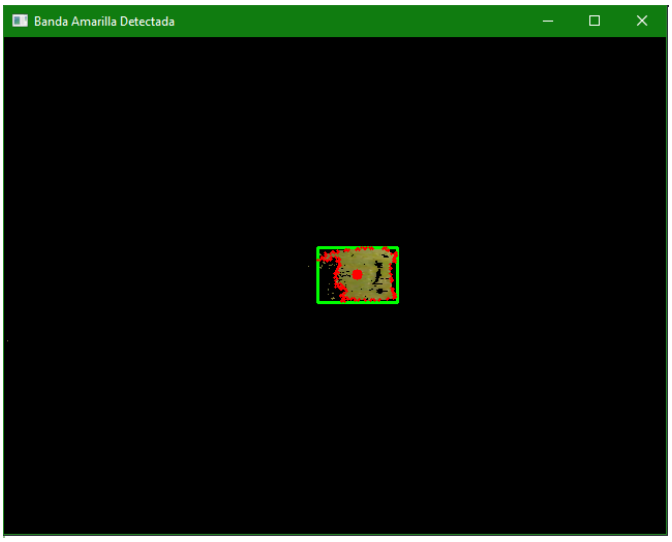

Imagen 6: bounding box y punto central del contorno

El diagrama de flujo es el siguiente:

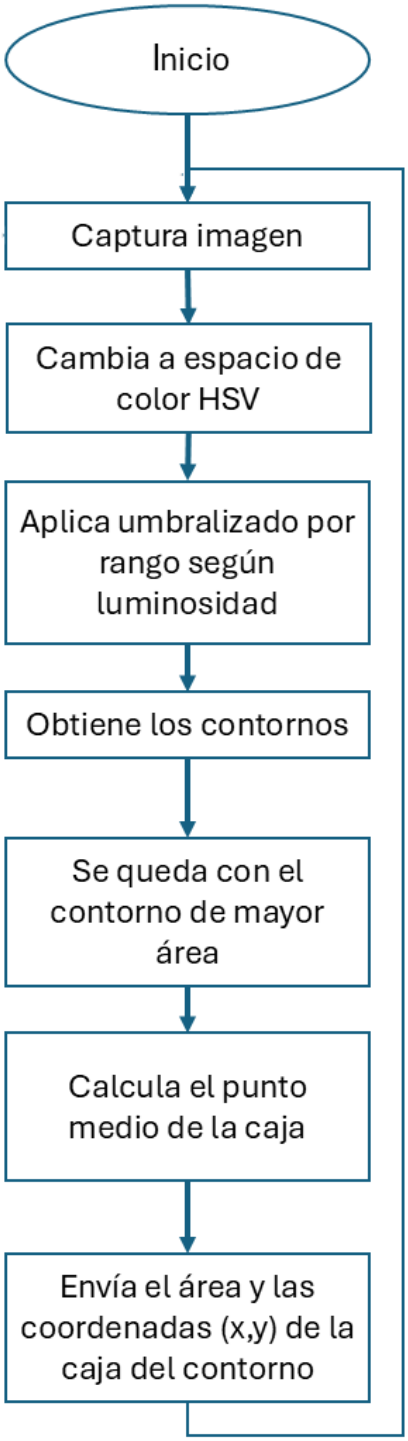

Imagen 7: Diagrama flujo

2. Detección en el turtlebot

Ha surgido la necesidad de realizar ciertas adaptaciones sobre el último script (**band_detection.py**) para que este se adapte al funcionamiento del turtlebot. Este script modificado y llamado **turtlebot_band_detection.py**, es la implementación del detector en ROS, para ello se ha empleado el script de la práctica 3 de base, añadiendo la parte de detección al callback que se ejecuta al obtener una lectura de imagen en el topic '/image'; también se publican las lecturas de detección en el topic '/color_detected', si se produce una detección. Para que la detección comience, se debe publicar en el topic /camera_onoff el string 'suscribe'; para desuscribirse de la lectura de cámara y ahorrar recursos se debe enviar por el mismo topic el string 'desuscribe'.

Dado que el tiempo de desarrollo con el robot real ha sido limitado, se ha optado por desarrollar un entorno de simulación. Para ello, se utilizó el simulador Stage, en el cual se creó un mundo que incluye un robot predefinido, su cámara y un obstáculo móvil de color amarillo que puede ser manipulado (arrastrado y soltado) con el ratón. Esta simulación, así como el script de detección, se ejecutan con: *roslaunch vision launcher.launch*

Adicionalmente, este programa ha sido testeado en el robot real para comprobar su correcto funcionamiento. Para ello, ha sido necesario inicializar la cámara ejecutando desde el turtlebot: *roslaunch astra_launch astra.launch*, el topic desde el que se lee la imagen se llama '/camera/rgb/image_raw'.

Al ejecutar **turtlebot_band_detection.py** en el turtlebot, es necesario ajustar los valores del umbral de detección, pues la cámara del turtlebot es muy diferente a la del ordenador y no es capaz de detectar la banda con los ajustes iniciales. Por la falta de tiempo, no se ha conseguido un ajuste óptimo, pero tras probar con varias iluminaciones de la habitación y la detección de objetos de similar color se ha llegado a un ajuste que detecta la banda en la mayoría de los casos y, a la vez, no detecta otros objetos de color similar. He aquí los valores ajustados:

- Para poca iluminación (menor de 40): V $\in$ [10, 120]
- Para iluminación media (40-70): V $\in$ [120, 240]
- Para iluminación alta (mayor de 70): V $\in$ [240, 255]

Para cualquiera de los tres umbrales de iluminación, los valores de H y S son idénticos:

- Umbral inferior: $H = 35$, $S = 100$
- Umbral superior: $H = 45$, $S = 255$

Además, se ha observado que la cámara del turtlebot es más sensible a la luz, por lo que la detección no funciona si no se consigue la suficiente iluminación. Esta nueva versión del programa con el ajuste para el robot real se implementa en el script **turtlebot_band_detection_real.py**