PRACTICA 4 - ARQUITECTURA SUBSUMIDA: ROBOBO BUSCADOR DE LUZ

Pablo Chantada Saborido & Jose Romero Conde

---

1. Introducción y Objetivos

Este proyecto implementa una arquitectura subsumida en un robot Robobo que debe buscar fuentes de luz y empujar objetos de color verde hacia ellas, mientras evita obstáculos y caídas. Utilizando el enfoque de arquitectura subsumida, hemos desarrollado comportamientos independientes con diferentes prioridades que trabajan juntos para lograr un comportamiento global complejo.

La misión específica del robot es:

- Explorar el entorno buscando objetos verdes
- Cuando encuentra un objeto verde, empujarlo hacia la fuente de luz
- Evitar colisiones con obstáculos durante la navegación
- Prevenir caídas por bordes

2. Diseño de la Arquitectura

2.1. Jerarquía de Comportamientos

Hemos implementado cuatro comportamientos principales, organizados en orden descendente de prioridad:

1. **AvoidFall**: Detiene cualquier acción y ejecuta una maniobra de retroceso cuando detecta un posible borde mediante los sensores IR frontales.
2. **AvoidObstacle**: Evita colisiones con objetos del entorno, excepto cuando el objeto detectado es el blob verde que el robot intenta empujar.
3. **PushColor**: Se activa cuando un objeto verde está centrado en la cámara y lo suficientemente cerca para ser empujado. Después de tres empujones, activa el modo "push-to-light"para dirigir el objeto hacia la luz.
4. **FindColor**: Busca y centra objetos verdes girando el robot hasta que el blob quede en el centro de la imagen.

Aunque no existe un comportamiento específico FindLight, la función de buscar luz está integrada en el modo "push-to-light"del comportamiento PushColor, que utiliza el método `scan_for_light()` de la clase base para identificar la dirección de la fuente de luz.

3. Comunicación entre Comportamientos

Los comportamientos comparten información a través de un diccionario `params` que contiene variables cruciales como:

- `stop`: Flag para terminar todos los comportamientos
- `blob_detected`: Indica si se ha detectado un blob de color

- `blob_centered`: Indica si el blob está centrado en la cámara
- `pushing_to_light`: Modo activado después de 3 empujones
- `was_pushing`: Indica si estábamos empujando un objeto
- `push_count`: Contador de empujones realizados
- `light_threshold`: Umbral de brillo para considerar misión completa

4. Implementación de los Comportamientos

4.1. Base: Behaviour

Todos los comportamientos heredan de una clase base `Behaviour` que implementa:

- Gestión de hilos para ejecución paralela
- Mecanismo de supresión
- Métodos abstractos `take_control()` y `action()`
- Método `scan_for_light()` para detectar direcciones de luz

4.2. AvoidFall

Este comportamiento tiene la máxima prioridad y se activa cuando los sensores IR frontales detectan valores muy bajos, indicando un posible borde. Sus características principales son:

- Se activa cuando los sensores IR frontales izquierdo o derecho tienen lecturas por debajo del umbral de seguridad
- Ejecuta una maniobra de retroceso seguida de un giro en dirección opuesta al borde detectado
- Tiene lógica especial para caso de estar empujando un objeto cerca del borde, girando hacia la luz en lugar de alejarse simplemente del borde

4.3. AvoidObstacle

Este comportamiento evita colisiones mientras navega, con la excepción del objeto verde que se quiere empujar:

- Monitoriza todos los sensores IR (frontales y traseros)
- Ignora objetos cuando el blob verde está centrado y el IR central lo detecta (para poder empujarlo)
- Implementa diferentes estrategias de evitación según la ubicación del obstáculo
- Mantiene memoria de la última dirección de giro para alternar entre direcciones

4.4. PushColor

Este comportamiento es el núcleo de la tarea, encargándose de empujar objetos verdes y dirigirlos hacia la luz:

- Toma el control cuando un blob verde está centrado y a distancia adecuada
- Refina la alineación con el blob antes de empujar
- Mantiene la cuenta de empujones realizados al mismo objeto
- Después del tercer empujón, activa el modo "push-to-light"
- Implementa un controlador PID para calcular velocidades de movimiento suaves hacia la luz
- Escanea periódicamente la dirección de mayor brillo para orientar el empuje

4.5. FindColor

Busca y centra objetos verdes en el campo visual:

- Busca blobs verdes usando la cámara y los alinea en el centro de la imagen
- Implementa un patrón de búsqueda cuando no detecta ningún blob
- Utiliza velocidad proporcional al error de posición para centrar el blob suavemente
- Detecta situaciones donde el blob puede estar fuera del campo visual pero detectable por IR

5. Desafíos y Soluciones

Durante la implementación, nos enfrentamos a varios desafíos:

- **Detección cuando el blob está fuera del campo visual**: Implementamos una lógica para detectar un objeto mediante el sensor IR frontal incluso cuando no es visible en la cámara.
- **Coordinación entre comportamientos**: Utilizamos el diccionario compartido `params` para mantener el estado y permitir que los comportamientos colaboren.
- **Evitar obstáculos sin evitar el objetivo**: Modificamos el comportamiento AvoidObstacle para ignorar objetos que son el blob objetivo, comprobando tanto la lectura del sensor central como la bandera de blob centrado.
- **Recuperación de situaciones atascadas**: Implementamos una detección de inactividad en el bucle principal que realiza movimientos aleatorios y reinicia el estado cuando el robot está atascado.

6. Conclusiones

La arquitectura subsumida ha demostrado ser efectiva para esta tarea al permitir que cada comportamiento se especialice en un aspecto específico de la misión. El diseño modular facilita la prueba individual de comportamientos y su integración en un sistema completo.

La jerarquía de comportamientos garantiza que las acciones de seguridad (evitar caídas y obstáculos) siempre tienen prioridad sobre los objetivos de la misión, mientras que los comportamientos de nivel inferior (buscar y empujar) pueden ejecutarse cuando es seguro hacerlo.

El sistema resultante es robusto y capaz de operar en entornos dinámicos con diferentes posiciones iniciales y configuraciones de objetos.