Universidad de La Coruña

Facultad de Informática

Robótica Basada en Comportamiento:

Controlador con Arquitectura Subsumida

Xian Priego Martín
Samuel Otero Agraso

12 de marzo de 2024

1 Introducción

En el campo de la robótica, la creación de sistemas autónomos capaces de adaptarse a entornos dinámicos es un objetivo fundamental. Una estrategia efectiva para alcanzar este objetivo es mediante el uso de la arquitectura subsumida. Esta arquitectura permite la integración de comportamientos complejos a través de múltiples capas de control, lo que facilita que un robot realice tareas variadas y complejas de manera autónoma.

El objetivo de esta práctica es implementar un controlador para un robot móvil que pueda realizar cuatro tareas independientes de manera autónoma. Estas tareas incluyen seguir una fuente de luz y una pared, buscar una pared y escapar de una situación de atasco. Para lograr esto, se utilizará una arquitectura subsumida, que permitirá la combinación de diferentes comportamientos en un único controlador, priorizando y ejecutando las tareas según sea necesario.

En este documento, se describirá el proceso de diseño e implementación del controlador basado en arquitectura subsumida. Se discutirán las decisiones de diseño tomadas durante el desarrollo del controlador, así como los problemas y limitaciones encontrados durante su implementación.

2 Diseño

Durante el diseño del controlador basado en arquitectura subsumida para el robot móvil, se han tomado varias decisiones importantes que afectan el funcionamiento y la eficacia del sistema. A continuación, se detallan las principales decisiones de diseño adoptadas:

1. **Control de Tareas:** Para gestionar las tareas y su interacción en el sistema, se ha implementado un array global de booleanos compartido. Este array indica el estado de inhibición de cada tarea, donde un valor `True` indica que la tarea está inhibida. Este array está protegido por un semáforo para garantizar un acceso seguro y evitar condiciones de carrera.

2. **Inhibición de Tareas:** Se ha optado por utilizar la propagación de la inhibición entre las tareas. Es decir, cuando una tarea se inhibe, también inhibe a la siguiente tarea en la jerarquía. Esto garantiza un control claro y consistente sobre el flujo de ejecución de las tareas.

3. **Tarea de Escapar:** La tarea de escapar se activa únicamente cuando el robot está atascado, detectado mediante el sensor táctil (`touchSensor`). Consiste en hacer retroceder al robot durante un breve periodo de tiempo para alejarlo del obstáculo con el que ha chocado.

4. **Tarea de Dirigirse a la Luz:** Esta tarea comienza orientando el robot hacia la fuente de luz detectada por el sensor de color (`colorSensor`). Utiliza un giro de 360 grados para registrar los valores de luminosidad y luego se reorienta hacia el punto con la luminosidad máxima. Posteriormente, se aproxima a la luz utilizando la estrategia zig-zag.

5. **Tarea de Seguir Pared:** Implementada utilizando la estrategia de medialunas, la tarea sigue la pared girando en semicírculos hasta que se detecta que está muy cerca de la misma. En ese momento, se gira nuevamente para mantener una distancia constante.

6. **Tarea de Buscar Pared:** Avanza en línea recta hasta que mediante el sensor ultrasónico se detecta una pared cercana. Cuando se encuentra cerca de una pared, la tarea se detiene.

7. Cabe destacar que en la explicación de las dos tareas anteriores se menciona la palabra `hasta`, esto no se refleja en el código ya que de ser así se tendría un bucle while dentro de los módulos (además del propio de cada tarea). Esto es indeseado ya que acapararía demasiado tiempo la CPU y no permitiría dirigirse a una posible luz detectada o escapar si colisiona durante esos procesos. En vez de un while internamente se gestiona con un if-else que en cada iteración del bucle principal hace una cosa u otra.

8. **Control de Actuación de Buscar Pared y Seguir Pared:** Para evitar la pérdida de enfoque durante la ejecución de la tarea seguir pared, no basta con ver si hay o no pared, por lo que se ha implementado una variable de control con un temporizador. Esta variable verifica que el robot no pase más de 8 segundos sin detectar una pared durante el seguimiento de la pared. Si esto ocurre, se permite que la tarea de buscar pared tome el control.

3 Problemas y Limitaciones

Se van a enumerar una serie de problemas o limitaciones que tiene la implementación planteada:

3.1 Detección Básica de Atoramientos

La detección de atoramientos actualmente se basa únicamente en el touchsensor, lo que limita su efectividad en entornos complejos como laberintos, donde el robot puede quedar atrapado sin necesariamente chocar con el sensor. Una mejora potencial sería integrar el gyrosensor y un temporizador para detectar si el robot está intentando girar durante un período prolongado sin éxito, lo que indicaría un posible atoramiento.

3.2 Escape Básico

El comportamiento de escape actual consiste simplemente en retroceder el robot, lo que puede llevar a situaciones en las que el robot quede bloqueado con la parte trasera hacia un obstáculo. Para mejorar esta función, se podría modificar la lógica de escape para probar diferentes estrategias en función de la situación de atoramiento, aumentando así la probabilidad de un escape exitoso y seguro.

3.3 Velocidad en el Seguimiento de Luz

Aunque el comportamiento para dirigirse a una fuente de luz es preciso, su velocidad se ve afectada por la fase de orientación, donde el robot realiza un giro completo de 360 grados. Para mejorar la eficiencia de este comportamiento, se podría explorar la posibilidad de reducir la fase de orientación o implementar métodos de orientación más rápidos y precisos.

3.4 Seguimiento de Pared

La estrategia actual de seguimiento de pared mediante semicírculos es adecuada dadas las limitaciones de sensores del robot. Sin embargo, esta estrategia puede resultar tediosa y lenta, ya que el robot necesita corregir constantemente su dirección mientras se desplaza en línea recta. Una mejora potencial sería incorporar sensores laterales para permitir correcciones más efectivas durante el seguimiento de la pared, mejorando así la eficiencia y la precisión del movimiento.

3.5 Gestión de Tareas Prioritarias

La tarea de dirigirse hacia una fuente de luz acapara durante bastante tiempo la CPU. A diferencia de las tareas de búsqueda y seguimiento de pared, donde se evita conscientemente esto no utilizando, en la tarea de seguimiento de luz se ha optado por una ejecución continua. Esto se debe a que no tuvimos la posibilidad de probar el código adaptado para ir cediendo la CPU físicamente y nos parecía bastante arriesgado modificar el código sin probarlo. Sin embargo, somos conscientes de que esto no es lo ideal y que la solución sería semejante a la implementada en las otras dos tareas. No obstante, tampoco supone un enorme problema ya que la tarea de dirigirse hacia la luz es la tercera más prioritaria. Aunque si que es cierto que si el robot se quedase atorado mientras busca la luz, no se ejecutaría la tarea de escape, además a la hora de añadir otras tareas con mayor prioridad esto se haría más relevante aún.