Artur Gil Torres, a.gtorres@udc.es
Pelayo López Tomé, pelayo.lopez@udc.es

Controlador con Arquitectura Subsumida - Memoria

Para la realización de esta práctica se han implementado los 4 comportamientos establecidos en arquitectura subsumida, siguiendo la prioridad entre ellas, de mayor a menor: escapar, dirigirse a la luz, seguir paredes y acercarse a pared. Cada uno de estos comportamientos opera de manera concurrente como una tarea de RobotC estructurada en las siguientes fases generales ejecutadas de manera cíclica:

- Comprobación de estado inhibido. De estar inhibida se inhibirá la tarea inmediatamente inferior.
- Sensorización para determinar si se debe actuar. Se realizará en caso de que la tarea no esté inhibida, cada tarea comprobará su condición para decidir si puede actuar o no.
- Actuación. Se inhibirán las tareas inferiores en caso de tener que actuar, cuando se dejen de cumplir las condiciones de actuación se desinhibirá la siguiente tarea de nivel inferior.

Para coordinar la ejecución de las tareas se emplea una variable global inhibidas[3], un array booleano que indica con 1 si la tarea i está inhibida, 0 en caso contrario. Las posiciones de inhibidas[3] representan las tareas:

- inhibidas[0] -> Acercarse a pared
- inhibidas[1] -> Seguir paredes
- inhibidas[2] -> Dirigirse a la luz

Se ha decidido que el array inhibidas[3] no mantenga una posición para la tarea Escapar ya que ésta última de mayor prioridad nunca podrá estar inhibida.
Por ser una variable compartida se debe proteger de accesos concurrentes entre tareas de lecturas/escrituras, y para ello se emplea un semáforo, para hacer estas tareas de manera atómica.
Hemos definido 3 funciones en las que incluimos el semáforo para operar con la variable global de manera segura:

- inhibir_nivel_inferior(int nivel) : void -> Inhibirá la tarea con índice nivel-1 en inhibidas[3].
- desinhibir_nivel_inferior(int nivel) : void -> De manera homóloga a la anterior desinhibe la tarea de nivel inferior.
- esta_inhibida(int nivel) : int -> Comprueba si la tarea con índice nivel en inhibidas[3] está inhibida, simplemente retorna el valor de esa posición en el array compartido.

Artur Gil Torres, a.gtorres@udc.es
Pelayo López Tomé, pelayo.lopez@udc.es

ESCAPAR

La tarea de escape nunca podrá estar inhibida, ya que tiene la mayor prioridad. Esto significa que siempre estará evaluando si debe actuar. El robot deberá activarse si se cumple alguna de las siguientes condiciones:

- La distancia frontal del robot es menor a un umbral muy pequeño.
- El touch sensor se activa.
- Detecta que ha estado cerca de una pared durante un período prolongado.

Si alguna de estas condiciones se cumple, el robot inhibirá las tareas de menor prioridad y activará los motores para moverse hacia atrás. Una vez que el robot se haya alejado de la pared hasta alcanzar la distancia objetivo, continuará retrocediendo durante dos segundos adicionales. Posteriormente, desinhibirá las tareas inferiores y dejará de actuar.

SEGUIR LUZ

La tarea de seguir la luz estará continuamente verificando si ha sido inhibida por una tarea de mayor prioridad. En caso de estar inhibida, reiniciará constantemente su estado y su temporizador, además de establecer que no debe actuar.

Si no está inhibida, evaluará si debe activarse. Para ello, comprobará si la intensidad de la luz detectada se encuentra dentro de un rango específico: debe superar un umbral mínimo, pero no exceder un umbral máximo, ya que esto indicaría que está demasiado cerca de la fuente de luz.

El comportamiento de la tarea se ha dividido en tres estados, lo que permite que, si otra tarea la inhibe, el tiempo de interrupción sea mínimo:

1. Estado 1: El robot gira sobre sí mismo realizando una vuelta completa para detectar la dirección en la que se encuentra la mayor intensidad de luz. Una vez identificada, pasa al siguiente estado.
2. Estado 2: El robot se orienta en la dirección donde encontró la luz máxima. Luego, pasa al siguiente estado.
3. Estado 3: El robot avanza en línea recta durante un tiempo determinado. Después, regresa al Estado 1 para actualizar la ubicación de la fuente de luz.

Durante todo el proceso, el robot verificará constantemente si la luz detectada supera el umbral superior; si esto ocurre, dejará de actuar.

Artur Gil Torres, a.gtorres@udc.es
Pelayo López Tomé, pelayo.lopez@udc.es

Si la tarea no debe actuar, desinhibirá a las tareas de menor prioridad.

SEGUIR PAREDES

Como antes, sólo en caso de no estar inhibida se realizará la sensorización para determinar si se debe actuar o no. En este caso la condición se basa en la distancia detectada con el sensor de ultrasonidos. Si es inferior a un umbral preestablecido la tarea comenzará a actuar. De la misma manera separamos el comportamiento de la tarea en estados:

1. Estado 1: El robot gira sobre sí mismo hasta orientar 100º respecto a su posición original contra el muro.
2. Estado 2: El robot activa los motores, el de la derecha con mayor potencia que el de la izquierda y avanza realizando una media luna. Una vez se sitúa por debajo de un umbral de distancia mínima se volverá al Estado 1 para volver a repetir el proceso.

En todo momento se estará comprobando que la distancia contra un muro eventualmente caiga por debajo del umbral establecido. De no suceder esto en 10 segundos se cancelará la actuación de esta tarea, desinhibiendo la tarea inferior, ya que el robot habrá comenzado a girar en círculos sin encontrar ninguna pared.

ACERCARSE A PARED

La tarea de acercarse a una pared verificará constantemente si ha sido inhibida por otra tarea, ya que es la de menor prioridad. En caso de ser inhibida, reiniciará su estado a 0.

Si no está siendo inhibida, esta tarea siempre estará activa. Su comportamiento se ha dividido en tres estados, siguiendo la misma lógica aplicada en la tarea de seguir la luz, para garantizar una respuesta más eficiente en caso de interrupción.

1. Estado 1: El robot girará sobre sí mismo para detectar la dirección en la que se encuentra la menor distancia registrada por el sonar sensor. Una vez identificada, pasará al siguiente estado.
2. Estado 2: Se orientará hacia la dirección donde detectó la menor distancia. Luego, pasará al siguiente estado.
3. Estado 3: El robot avanzará en línea recta a una velocidad normal en la dirección identificada. Cuando se acerque a unos centímetros por encima de la distancia objetivo, reducirá ligeramente su velocidad para lograr un acercamiento más preciso.

Artur Gil Torres, a.gtorres@udc.es
Pelayo López Tomé, pelayo.lopez@udc.es

OBSERVACIONES

Uno de los problemas que detectamos y que no habíamos considerado inicialmente es que, al finalizar el comportamiento de una tarea, deteníamos todos los motores. El problema surge cuando una tarea de mayor prioridad necesita activarse: esta tarea configura los motores según sus necesidades e inhibe a las tareas de menor prioridad. Sin embargo, la tarea inferior, al detectar que ha sido inhibida, detiene los motores después de que la tarea superior los haya activado. Como resultado, el robot quedaba detenido permanentemente.

Además, para hacer el funcionamiento del programa más seguro en términos de concurrencia siempre antes de activar los motores se deberá comprobar que la tarea no fue inhibida. Esto se hace para minimizar los casos en los que una tarea de nivel superior justo inhiba la tarea inferior en el momento de activar los motores.

Se ha visto también un problema la gestión de los sensores de manera concurrente, ya que podría llegar a suceder en un caso límite que una tarea por ejemplo, resetee el valor del giroscopio mientras otra lo está leyendo, es por ésto que en las mediciones de ángulos se incluyen comprobaciones esta_inhibida() para minimizar los errores. Como este caso se han incluído numerosas comprobaciones esta_inhibida() en el modo actuar de cada tarea, así los errores de lectura/escritura en sensores son mínimos.

Esta implementación de arquitectura subsumida facilita enormemente la incorporación de nuevas tareas con diferentes funcionalidades. Al contar con un sistema de prioridades entre tareas, donde estas se inhiben o desinhiben en cascada, lo único que se necesita para agregar una nueva tarea es establecer su orden de prioridad. Para esto simplemente bastaría con añadir una posición nueva al array inhibidas[] e incorporar la nueva task, incluyendo en las operaciones inhibir_nivel_inferior(), desinhibir_nivel_inferior(), esta_inhibida() su nuevo índice correspondiente.