Escuela Politécnica Superior de Elche

SISTEMAS INFORMÁTICOS EN TIEMPO REAL

2º Ingeniería Industrial

PRÁCTICAS DE PROGRAMACIÓN DE MINIROBOTS

3. Programación de Comportamientos en (IC)

CURSO 00/01

Luis Miguel Jiménez

Rafael Puerto

Departamento de Ingeniería
Área de Ingeniería de Sistemas y Automática

1 OBJETIVO

El objetivo esta práctica es profundizar en la programación de Robots mediante IC. Se recomienda repasar y tener a mano la documentación de las dos primeras prácticas donde se presentaba el manejo básico del entorno IC , las funcionalidades del robot RugWarrior Pro, y las librerías de manejos de sensores y actuadores.

Se desarrollarán diferentes aspectos del diseño de aplicaciones de tiempo real:

- Conceptos de multiprogramación
- Implementación de especificaciones temporales
- Gestión de dispositivos físicos mediante sensores y actuadores
- Técnicas de diseño
- Conceptos de inteligencia artificial

2 MATERIAL EMPLEADO

La práctica se realizará en grupos de tres personas, disponiendo de un PC con S.O. Windows 95/98, el entorno de desarrollo ICWin, y un Robot RugWarrior Pro (compartido entre varios grupos). La documentación necesaria se puede encontrar en el servidor http://lorca.umh.es/isa/es/asignaturas/sitr/minirobots. A continuación se resume la documentación recomendada:

- Capítulo 9 del libro "*Mobile Robots: Inspiration to Implementation*"
- Manual de IC: **icmain.pdf** o **ic.hlp** (versión más actualizada), disponible en el servidor web (documentación) y en cada PC del laboratorio
- Manual de montaje del RugWarrior Pro. (libro rojo disponible en el laboratorio y en el servidor web en formato pdf)

3 CONTROL REACTIVO DE ROBOTS. PROGRAMACIÓN DE COMPORTAMIENTOS.

Vamos a profundizar en el diseño de aplicaciones de control de robots mediante IC. Comenzaremos presentando una visión general de las estrategias utilizadas en el diseño de sistemas de control de robots. Posteriormente nos centraremos en el esquema de control propuesto para el control de pequeños robots basado en la ***fusión de comportamientos*** mostrando su implementación en IC.

Dentro del campo de la robótica existen multitud de algoritmos y estrategias utilizadas en el control del robots. La mayoría de ellas suelen ser específicas de la aplicación final del sistema, es decir están orientadas al ***propósito*** del sistema, constituido tanto por el robot como por el resto de elementos de su entorno.

Las diferentes propuestas pueden clasificarse en dos tipos de estrategias de control denominadas:

- **Control Planificado** (control basado en modelos)
- **Control Reactivo** (control basado en la fusión de comportamientos)

El **Control Planificado** tiene su origen en las primeras aplicaciones de la robótica y está basado en la obtención de un **modelo** preciso del entorno en el que se debe desenvolver el robot. A partir de este modelo del mundo real se realiza una planificación que determina las acciones que tiene que ejecutar el robot en cada instante para alcanzar su objetivo (figura 1).

Este tipo de esquema (*modelado-planificación*) tiene la ventaja de obtener un comportamiento óptimo. En cambio puede presentar problemas en ciertas aplicaciones debido a la gran tiempo de procesamiento y capacidad de almacenamiento necesaria.

En primer lugar exigen un **modelo preciso del mundo,** lo que requiere normalmente una fusión sensorial muy exigente en tiempo de cálculo y propensa a errores debida al ruido en las medidas, por lo que varios sensores pueden generar medidas conflictivas. El mundo real es demasiado complejo y tiene demasiados detalles como para obtener modelos exactos de mismo. Exige asimismo una elevada capacidad de almacenamiento para guardar el modelo del mundo. En muchos casos se reduce esta complejidad considerando el entorno estático lo cual reduce la capacidad de adaptarse cambios no previstos.

Una vez obtenido este modelo se debe proceder a la planificación. Se suelen utilizar algoritmos de optimización con tiempos de respuesta elevados y que exigen una gran capacidad de procesamiento.

Este tipo de estrategia es utilizada de forma generalizada en el control de robots para células de fabricación y en grandes robots móviles, donde los problemas comentados pueden ser soslayados. Como se puede observar, se trata de un proceso secuencial en el cual no es preciso ningún paralelismo.

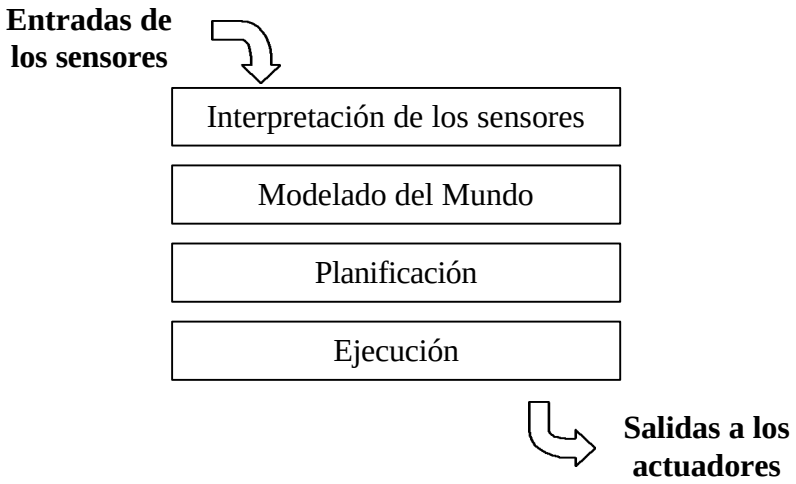

Figura 1. Esquema de control planificado

El **control reactivo** en cambio no utiliza un modelo del mundo para ejecutar las acciones del robot. En este caso cada sensor actúa de forma de forma ***refleja*** generando uno o varios comportamientos. La ***fusión de los comportamientos*** generados por cada sensor determina la acción de control sobre los actuadores y por tanto el comportamiento final del robot.

Este tipo de estrategia permite una reacciona ante los estímulos procedentes de los sensores en tiempo real. No exige un procesamiento complejo de la información sensorial ya que la

fusión se produce en este caso en el nivel de los *comportamientos* y no de los sensores. En este nivel la información es de mayor nivel y por tanto más reducida exigiendo por tanto menor capacidad de cálculo y de almacenamiento. No existe en este caso un modelo del mundo.

Esta estrategia de control se adapta perfectamente a la programación de minirobots ya que son equipos que disponen de una limitada capacidad de procesamiento y almacenamiento. Además, en muchos casos no es preciso un modelo del mundo para actuar en él con un comportamiento inteligente.

Cada uno de los comportamientos activados por los sensores operan temporalmente en paralelo por lo que precisamos capacidad para la ejecución multitarea. El problema de los conflictos entre los datos aportados por diferentes sensores se maneja como un problema de conflicto entre comportamientos por lo que debemos establecer un mecanismo de fusión de la ***red*** de comportamientos. La figura 2 muestra un esquema de este tipo de control. En la figura 3 se concreta en un ejemplo:

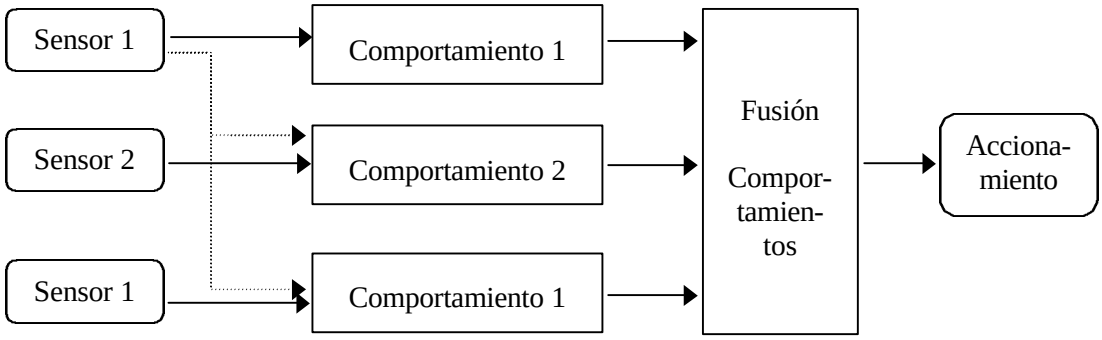

Figura 2 Esquema de Control Reactivo

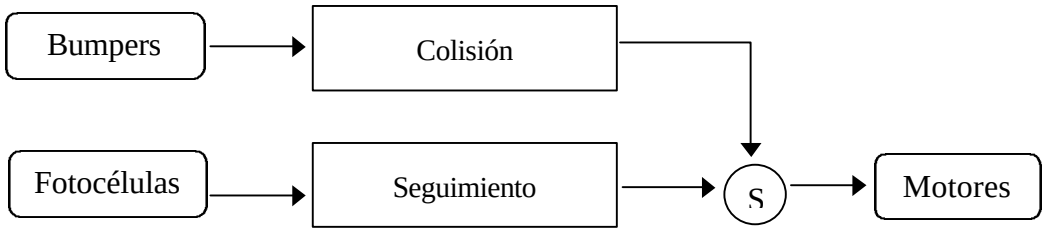

Figura 3. Ejemplo de control reactivo basado en comportamientos

En el ejemplo disponemos de dos conjuntos de sensores. Varias fotocélulas que generan un comportamiento de **seguimiento** de la luz. Un conjunto de interruptores (*bumpers*) que indican si se ha producido una colisión que generan un comportamiento de **colisión**. La combinación de ambos comportamientos se controla mediante un **supresor** que establece una mayor prioridad al comportamiento de colisión en la generación de comandos a los motores. De este modo normalmente el robot estará siguiendo la luz hasta que se produzca una colisión, en ese momento el comportamiento de colisión toma el control retrocediendo

hasta que se evite la colisión. Una vez que se no se detecta la colisión se inhibe dicho comportamiento volviendo a actuar en comportamiento de seguimiento.

Como se puede observar este esquema de control establece un conjunto de ***capas de comportamiento*** y un mecanismo de ***prioridades*** en el acceso a los accionamientos, el cual regula la fusión de los comportamientos. Se trata de una arquitectura inherentemente paralela que puede ser implementada mediante sistemas multitarea (mono o multiprocesador). No precisa de un modelo del mundo por lo que el consumo de memoria es pequeño. El código de cada comportamiento puede ser sencillo y la arquitectura altamente modular, pudiendo ser extendida fácilmente para incorporar nuevos comportamientos y sensores.

4 IMPLEMENTACIÓN DEL CONTROL REACTIVO EN EL RUGWARRIOR MEDIANTE IC

En la prácticas anteriores se ha mostrado la funcionalidad del robot RugWarrior Pro. Resumimos a continuación los sensores y actuadores disponibles:

Sensores	Actuadores
3 Sensores de Contacto 2 Fotocélulas 2 Sensores Infrarrojos 2 Encoders Incrementales 1 Micrófono	2 Motores Pantalla LCD Altavoz

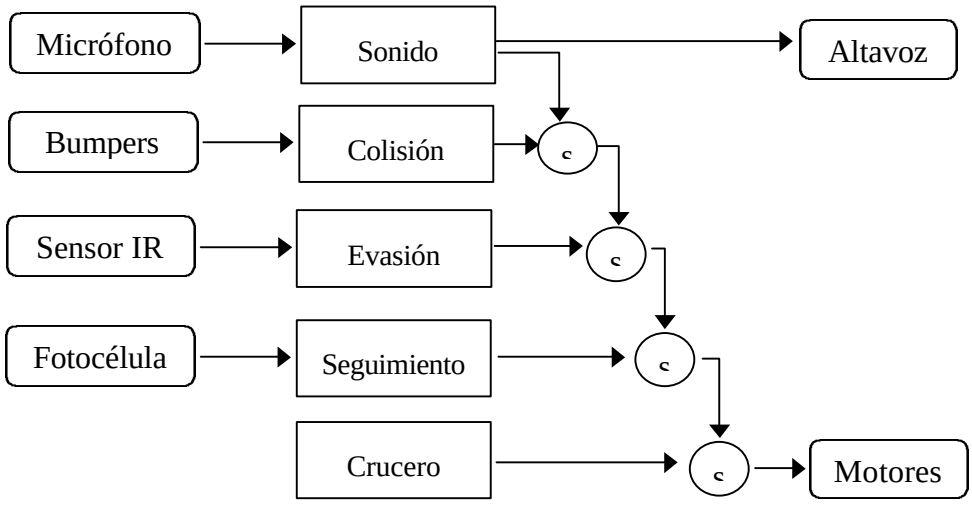

Figura 4. Ejemplo de arquitectura de control reactivo para el RugWarrior Pro.

En la figura 4 se muestra un ejemplo del esquema de control reactivo propuesto para el robot RugWarrior Pro. En el disponemos de cinco comportamientos asociados a diferentes sensores.

- Comportamiento **Crucero**: el propósito de este comportamiento es que el robot haga siempre algo interesante, es decir, moverse. No está disparado por ningún sensor y tiene la menor prioridad.

- Comportamiento **Seguimiento**: monitoriza continuamente las fotocélulas y determina la corrección de trayectoria par dirigirse al punto de máxima brillantez. Tiene mayor prioridad que el comportamiento *Crucero*.

- Comportamiento **Evasión**: determina, en función de los sensores de infrarrojos, la proximidad y orientación de un obstáculo. Genera la corrección de trayectoria para evitarlo (girar durante un tiempo). Tiene mayor prioridad que los comportamientos *Crucero* y *Seguimiento*.

- Comportamiento **Colisión**: chequea continuamente el estado de los interruptores de contacto para detectar un posible choque. En caso de activarse, para los motores y ejecuta una maniobra de evasión del obstáculo (retroceder durante un tiempo y girar). Este comportamiento tiene prioridad sobre los anteriores.

- Comportamiento **Sonido**: detecta patrones específicos de sonido (secuencias de palmadas y pausas) para determinar la activación o desactivación de los motores, generando así mismo una melodía indicando el reconocimiento del comando. Este comportamiento inhibe a todos los demás.

La fusión de todos los comportamientos genera un comportamiento global complejo, de forma que el robot busca la luz rodeando o evitando obstáculos, activándose o desactivándose con palmadas.

4.1. Implementación en IC

Como comentamos anteriormente el control reactivo es intrínsecamente paralelo. Para lograr este paralelismo utilizaremos el planificador preemptivo disponible en IC. Cada uno de los bloques con los comportamientos y la fusión de comportamientos descritas en la figura 4 serán implementadas mediante **procesos** que se ejecutarán concurrentemente.

Tomemos como ejemplo el comportamiento asociado al *seguimiento* con las fotocélulas. El código podría ser el mostrado a continuación. Analicémoslo:

1. Esta constituido por una función asociada a un proceso. Esta función tiene un bucle infinito (**while**(1){}) que ejecuta continuamente la lectura de los sensores y actúa en consecuencia.

2. La lectura de los sensores se realiza al principio del bucle. Leemos los valores de las dos fotocélulas que nos proporcionan un valor analógico entre 0 y 255. cuanto menor es el valor más brillante (menor voltaje).

(**Nota**: La variable `photo_cal` nos permite realizar un ajuste entre las lecturas de ambas fotocélulas. El valor lo podemos obtener dirigiendo el robot hacia una fuente de luz difusa. Con el programa de test haremos una lectura de los valores de cada fotocélula, obteniendo `photo_cal` como la diferencia entre el valor leído entre la fotocélula derecha y la izquierda).

3. Se calcula la diferencia entre ambas lecturas y se selecciona entre tres posibilidades:
    a. No activar el comportamiento si la diferencia es pequeña (variable `photo_dead_zone`)
    b. Girar a la izquierda
    c. Girar a la derecha

4. Para comunicarse con el proceso encargado de coordinar los comportamientos en el manejo de los motores se utilizan dos variables globales:
    - `photo_command`: variable que indica el movimiento a realizar.
    - `photo_active` : variable lógica que determina el estado, activado o desactivado, del comportamiento

5. Finalmente se libera el uso de la CPU (**defer()**). Esto no es estrictamente necesario ya que el planificador es de tipo preemptivo y va a expulsarlo cuando acabe su tiempo. La liberación voluntaria mejora el comportamiento temporal ya que la próxima iteración empezará siempre al principio del *while*. De este modo (si asignamos suficiente tiempo al proceso en su creación) disponemos de un ***cuantum*** igual al tiempo que precisa en cada momento una iteración completa del código del comportamiento, y además éste se ejecutará sin ser interrumpido a la mitad.

El código asociado al proceso que coordina todos los comportamientos en su acceso a los motores `motor_control()`, está constituido por un conjunto de sentencias **if .. else if .... else if** ubicadas en un bucle infinito (**while**(1)). Especifica, de este modo, un mecanismo de prioridad, de forma las primeras condiciones tienen prioridad sobre las siguientes. Además solo se ejecutará una de las opciones.

Se comprueba si está activado cada comportamiento, cuando se llega a uno que está activado se ejecuta su comando y se libera la CPU. Esta iteración se repite indefinidamente alternándose con el resto de procesos del sistema.

Nota: En negrita se han remarcado las sentencias necesarias para probar el comportamiento *seguimiento()*. El resto de sentencias, corresponden al resto de comportamientos que deberán ser añadidos (no las introduciremos hasta que no implementemos tales comportamientos).

Pro último la función `main()` debe asociar un proceso a cada una de las dos tareas:

`seguimiento()`	comportamiento
`motor_control()`	Fusionador de comportamientos

```
/* Seguimiento de la zona más brillante => (menor valor) */
int photo_dead_zone = 2;      /* umbral de detección... */
int photo_cal = 0;      /* calibración de las fotocélulas */

int photo_command = STOP;
int photo_active = FALSE;

int seguimiento()
{
    int lpc, rpc, delta;      /* variables */

    while (1) {
       lpc = analog(1) + photo_cal; /* lectura fotocélula izqda */
       rpc = analog(0);             /* lectura fotocélula dcha */
       delta = (rpc - lpc);         /* diferencia */
       /* + => Izqda. ve más brillante, - => Dcha. más brillante */

       if ( abs(delta) > photo_dead_zone )
       {
            if (delta > 0)
                  photo_command = ARC_LEFT;
                  /* Izqda. más brillante => girar izquierda */
            else
                  photo_command = ARC_RIGHT;
                  /* Dcha. más brillante => girar derecha */
            photo_active = TRUE;    /* Activar cuando se detecta */
       }
       else
          photo_active = FALSE; /* Desactivar cuando no se detecta */

       defer();
            /* Libera CPU: una iteración por tick del planificador */
     }
}

void motor_control()
{
   while (1)
   {
      if (sound_active)
         move(sound_command);
      else if (bump_active)
         move(bump_command);
      else if (ir_active)
         move(ir_command);
      else if (photo_active)
         move(photo_command);
      else if (cruise_active)
           move(cruise_command);
      else
            move(STOP); /* No hay comandos => STOP */
      defer();
            /* Libera CPU: una iteración por tick del planificador */
   }
}

void main()
{
      printf("Práctica 3:");
      start_process(seguimiento());
      start_process(motor_control());
}
```

5 Tareas:

- Implementar el comportamiento `seguimiento()` descrito en el apartado anterior. Recordar que parte del código de `motor_control()` no debe ser introducido en esta etapa ya que incluye variables todavía no definidas (ver nota en apartado anterior)

- Implementar el código de los diferentes comportamientos descritos en los apartados anteriores (se pueden utilizar las referencias bibliográficas y los ejemplos, pero se valorarán más positivamente las aportaciones originales)

- Crear nuevos comportamientos opuestos a los anteriores:
  - Comportamiento de seguimiento de objetos mediante infrarrojos.
  - Comportamiento de fotobobía (evasión de la luz)

- Insertar un mecanismo de aprendizaje, mediante ***premios*** y ***castigos***, que coordine los comportamientos opuestos. Un comportamiento baja su prioridad si se ***reprende*** al robot al ejecutarlo.