Escuela Politécnica Superior de Elche

SISTEMAS INFORMÁTICOS EN TIEMPO REAL

2º Ingeniería Industrial

PRÁCTICAS DE PROGRAMACIÓN DE MINIROBOTS

2. Manejo de Sensores y Actuadores 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 la primera práctica donde se presentaba el manejo básico del entorno IC y las funcionalidades del robot RugWarrior Pro.

Se desarrollarán diferentes aspectos de la programación en IC y del manejo de sensores y actuadores:

- Diferencias fundamentales de IC frente a ANSI C
- Uso de la librería estándar de manejo de sensores
- Librería extendida
- Librería de programación de comportamientos de alto nivel

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:

- El 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 PROGRAMACIÓN EN IC

Vamos a profundizar algo más en el lenguaje de programación IC. Como ya comentamos, está basado en el lenguaje C. Partiendo de la premisa de que el alumno conoce la programación en ANSI C de forma correcta, nos centraremos en las diferencias sustanciales que incorpora IC, de forma que podamos conocer de forma precisa qué podemos o no podemos hacer.

Estas diferencias tienen por objetivo hacer el desarrollo de aplicaciones más sencillo, compacto y seguro. Se resumen a continuación:

- **Tipos de datos reconocidos:**

  No se admiten todos los tipos de datos estándar de ANSI C. Por ejemplo los modificadores **unsigned** o **double** no están permitidos. Tampoco se puede definir constantes con el modificador **const**. Los tipos de datos reconocidos son:

  - Enteros de 16 bits (`int`)

  - Enteros de 32 bits (`long`)

- Coma flotante de 32 bits (**`float`**)

- Caracteres de 8 bits (**`int`**)

- Estructuras (versión 3.2) (**`struct`**)

- **Inicialización de Variables**:

  Las variables globales y locales pueden ser inicializadas al declararlas. Si no se indica un valor son inicializadas a cero. Las variables globales son inicializadas siempre que: *se resetea la tarjeta, se descarga un programa o se ejecuta la función main().*

  Existe un modificador especial para *variables globales* (**`persistent`**) que permite que sea inicializada una sola vez, evitando que la variable sea reinicializada en cada reset o encendido de la tarjeta. Su valor se mantiene mientras no se descarguen las baterías o se elimine la variable. Ejemplo:

  ```
  persitent int i=500;
  ```

- **El manejo de cadenas de cualquier tipo de datos no puede realizarse mediante punteros**:

  Si al implementar una función se necesita pasar como parámetro una cadena de caracteres deberá indicarse de forma explícita que es una cadena y no un puntero. Ejemplo: `strcpy( char c1[], char c2[])`

  Aunque se dispone de punteros, no se permite el acceso a las cadenas de caracteres mediante ellos. Esto es así para permitir un mayor control en la gestión de cadenas, incorporando un chequeo de límites, lo que permite evitar corromper zonas adyacentes de memoria por un descuido. La función **`_array_size()`** permite conocer el tamaño de un array.

- **Punteros:**

  Se pueden declarar punteros, pasarlos como parámetros a una función y devolverlos. No está permitida la aritmética de punteros ni la equivalencia con cadenas.

- **Control de flujo:**

  Se dispone de todas las estructuras básicas de control excepto la estructura (**`switch - case`**) que puede ser implementada mediante las sentencias más eficientes (**`if ... else if..`**). Tampoco está disponible la versión compacta de la sentencia **`if`** (**`? : ;`**)

- **Algunas funciones estándar son más sencillas**.

  El caso más destacado es la función **`printf()`** que solo permite manejar los formateadores de datos más comunes (%d, %x, %b, %c, %f, %s). De este modo el código es más pequeño y eficiente.

- **Todos las declaraciones de los ficheros que componen un programa tienen un alcance global.**

  Todas las macros, definiciones, variables globales y funciones tienen alcance global por lo que no es preciso realizar inclusiones (no existe la cláusula **`#include`**), ni tampoco se puede declarar prototipos de funciones (**está prohibido**).

- **La versión 3.2 de IC incluye un precompilador con las clausulas #define, #ifdef, #endif.**

  En versiones anteriores no existía el precompilador, no pudiéndose definir constantes ni macros.

- **La carga de programas formados por varios ficheros:**

  Si un programa está formado por varios ficheros se utiliza un fichero de texto con extensión **.lis** en el que se indica, en cada línea, cada uno de los ficheros que conforman la aplicación.

- **Se dispone de funciones para la creación y la planificación de procesos**.

  IC dispone de un planificador preemptivo (interrumpe al proceso en ejecución después de un intervalo de tiempo), el cual permite la ejecución concurrente de varias tareas.

- **Se dispone de funciones para el acceso a puertos y registros a bajo nivel**.

  Permite configurar los registros del 68HC11 y los manejadores de interrupciones (ejecución de tareas críticas en tiempo real).

Destacar que la versión 3.2 de IC incorpora algunas características nuevas no descritas en el documento original de Fred Martin (*icmain.pdf*). Estas características (recogidas en el fichero ic.hlp) son la posibilidad de utilizar las directiva *#define*, *#ifdef*, declarar arrays bidimensionales y la declaración de estructuras.

4 LIBRERÍA BÁSICA DE MANEJO DE SENSORES Y ACTUADORES

Las funciones de manejo de los sensores y actuadores dependen del robot. Recordemos que el entorno IC se utiliza en diferentes tarjetas por lo que deberemos tener una librería específica para cada caso (en el laboratorio se dispone también de tarjetas HandyBoard que utilizan IC pero difieren en los puertos de E/S).

Esta librería está formado por cuatro ficheros englobados en un fichero con extensión *.lis* denominado ***lib_rwp.lis*** (contiene el listado de los ficheros que deben ser cargados). Esta librería incorpora las funciones básicas de manejo de los sensores y actuadores disponibles en el robot, y se carga al arrancar el programa ICWin.

Los ficheros están disponibles en el subdirectorio IC/libs/ (o ICRW/libs/), y son los siguientes:

lib_ic.c
lib_rwp.c
speed.icb
shaft.c

No es preciso comprender el código pero puede ser muy interesante analizarlo si se desea manejar nuevos sensores y actuadores y familiarizarse con la programación de bajo nivel del 68HC11. (*Ver también el Apéndice A de la guía de montaje del RugWarrio Pro*)

Vamos a enumerar a continuación las funciones disponibles comentando su funcionamiento:

FICHERO: lib_ic.c	
VARIABLES GLOBALES	**DESCRIPCIÓN**
`persistent int` **`test_number`**`;`	variable global persistente, utilizada para ir ejecutando de forma secuencial varios programas después de cada reset.
FUNCIONES	**DESCRIPCIÓN**
`void` **`reset_system_time()`**	Pone a cero el reloj del sistema
`float` **`seconds()`** `long` **`mseconds()`**	Devuelve el tiempo actual (entre 0 a 32.767) en minutos o segundos
`void` **`sleep(float seconds)`** `void` **`msleep(long msec)`**	Suspende un proceso durante un numero de segundos o milisegundos(no libera la CPU)
`void` **`beep()`** `void` **`tone(`**`float frequency, float length`**`)`** `void` **`beeper_on()`** `void` **`beeper_off()`** `void` **`set_beeper_pitch(`**`float frequency`**`)`**	Funciones de manejo del altavoz: Pitido estándar, tono especificado, o funcionamiento continuo a una determinada frecuencia seleccionada con set_beeper_pitch()

FICHERO: lib_ic.c	

`long timer_create_mseconds(long timeout)`
`long timer_create_seconds(float timeout)`
`int timer_done(long timer)`
	Simulan el comportamiento de un temporizador:
	Calculan el tiempo final de expiración y chequea con el tiempo actual.

`int analog(int port)`	Lee un canal del puerto analógico
`void hog_processor()`	Da al proceso que lo llama 256 ticks más(entorno a 1/4 sec) para seguir en ejecución

FICHERO: lib_rwp.c	
VARIABLES GLOBALES	**DESCRIPCIÓN**

`int photo_right = 0;`
`int photo_left = 1;`
`int microphone = 2;`
	Canales del puerto analógico utilizados

FUNCIONES	**DESCRIPCIÓN**
`int digital(int port)`	Lee el canal (bit) indicado del puerto digital A: Los canales 1 y 2 (entrada) están disponibles Los canales 0 y 7 (entrada) se utilizan para los encoders Los canales 5 y 6 (salida) se utilizan para los motores El canal 4 se utiliza para la pantalla LCD El canal 3 se utiliza para el altavoz.
`int left_shaft()` `int right_shaft()`	Lee el estado (0,1) de los encoders bits 0 y 7 del puerto A
`void stop()`	Control de los motores: Para ambos motores
`void motor(int index, int vel)`	Selecciona la velocidad para cada motor 0: izquierdo 1: derecho vel: [-99,99]
`void drive(int trans_vel, int rot_vel)`	Selecciona la velocidad traslacional y rotacional (positivo izquierda) y calcula la velocidad para cada motor. Límites abs(tvel)+abs(rvel) < 99
`int bumper()`	Determina el estado de los bumpers codificados en los 3 primeros bits (1 significa cerrado)
`int ir_detect()`	Determina el estado de los sensores de infrarrojos 0b00 => no reflection, 0b01 => reflection on right, 0b10 => reflection on left, 0b11 => reflection on both sides

FICHEROS: saft.c, speed.icb	
VARIABLES GLOBALES	**DESCRIPCIÓN**
`int` **`left_clicks`**	Los ficheros ***icb*** son ficheros de código máquina generado con un ensamblador, que puede ser utilizado desde el código IC. En este caso implementa la rutina de interrupción para la lectura del encoder de la rueda izquierda y define una variable global para almacenar el número de pulsos.
FUNCIONES	**DESCRIPCIÓN**
`void` **`init_velocity();`**	Activa los contadores de pulsos en los canales 0 y 7 del puerto A. Debe ser llamada antes de utilizar las funciones siguientes:
`int` **`get_left_clicks();`** `int` **`get_right_clicks();`**	Obtiene el número de pulsos para cada rueda desde la última lectura (ponen el contador a cero)

5 LIBRERÍA EXTENDIDA

En el fichero de configuración ***lib_rwp.lis*** se pueden incorporar librerías adiciones. En nuestro caso incorporaremos dos nuevos ficheros que contienen funciones de uso habitual pero que no están incluidos en la librería estándar, así como un conjunto de constantes para el manejo cómodo de los parámetros de las funciones.

Estos ficheros están disponibles en el subdirectorio IC/libs/ y son los siguientes:

comun.c
defines.c

Vamos a enumerar a continuación las funciones disponibles comentando su funcionamiento:

FICHERO: comun.c	
FUNCIONES	**DESCRIPCIÓN**
`int abs(int arg);` `float fabs(float arg);`	Calcula el valor absoluto de un número entero/coma flotante.
`int min(int a,int b);` `float fmin(float a,float b);`	Calcula el mínimo de dos números enteros/coma flotante.
`int max(int x, int y);` `float fmax(float x, float y);`	Calcula el máximo de dos números enteros /coma flotante.
`int aprox(int val, int cmp, int tol);` `int faprox(float val, float cmp, float tol);`	Indica si dos números enteros /coma flotante, son aproximadamente iguales con una tolerancia
`int limit(int val, int low, int high);` `float flimit(float val, float low, float high);`	Limita el valor de un número entero/coma flotante.
`int strcpy(char s1[],char s2[]);` `int strcat(char s1[],char s2[]);`	Copia y concatenación de cadenas de caracteres
`void wait(int milli_seconds);`	Espera un tiempo especificando liberando el uso de la CPU.
`int write_port (int port, int value, int mask);`	Copia solamente los bits indicados en **mask** con el valor **val** en un puerto
`int random(int mod);`	Números aleatorios en el rango especificado en **mod**. (rango máximo 2 a 32767)

FICHERO: defines.c

VARIABLES GLOBALES

```
int TRUE = 1;
int FALSE = 0;

/* BUMPERS */
int BUMP_NONE = 0b000;
int BUMP_BACK = 0b100;
int BUMP_LEFT = 0b010;
int BUMP_RIGHT = 0b001;
int BUMP_BL = 0b110;
int BUMP_BR = 0b101;
int BUMP_LR = 0b011;
int BUMP_BLR = 0b111;

/* IR-Sensor */
int IR_NONE = 0b00;
int IR_LEFT = 0b10;
int IR_RIGHT = 0b01;
int IR_BOTH = 0b11;
```

```
/* Limites enteros */
int MAX_INT =  32767;
int MIN_INT = -32768;
long MAX_LONG =  2147483647L;
long MIN_LONG = -2147483648L;


/* Analog sensors */
int PHOTO_RIGHT = 0;
int PHOTO_LEFT  = 1;
int MICRO       = 2;
int BUMPER      = 3;

/* motors */
int MOTOR_LEFT = 0;
int MOTOR_RIGHT = 1;

/* I/O */
int IO_DIR = 0x4000;
```

6 LIBRERÍA lib_umh.lis

Esta librería, disponible en el subdiretorio *IC/umhlibs/*, incorpora un conjunto de funciones útiles, manejo de alto nivel de accionamientos y sensores y diferentes comportamientos. El objetivo es que esta librería sirva de base de cualquier proyecto y así mismo cualquier desarrollo se vaya incorporando a esta librería para que sirva de base en futuros proyectos, acumulado la experiencia en el manejo del robot.

Está formada por los siguientes ficheros:

regdefs.c	Constantes con los valores de todos los registros del 68HC11
pantalla.c	Funciones de manejo avanzado de la pantalla LCD
musica.c	Codificación de las frecuencias de varias canciones para uso del altavoz
play.c	Rutinas para codificación de canciones en cadenas de texto para el uso del altavoz
sonido.c	Rutinas de muestreo y reconocimiento de sonidos mediante el micrófono
motores.c	funciones de manejo de los motores mediante comandos de alto nivel. Incorpora también una rutina de accionamiento diferencial con corrección de desviación.
servovel.c	Funciones para implementar un control realimentado de velocidad de los motores
comport.c	Rutinas con varios comportamientos para el robot.

Se recomienda repasar el código de estas librerías, ya que puede ser una guía para la realización de aplicaciones en IC.

Se detalla a continuación el contenido del fichero motores.c

FICHERO: motores.c	
VARIABLES GLOBALES	**DESCRIPCIÓN**

`int STOP = 0;`
`int FORWARD = 1;`
`int BACKWARD = 2;`
`int TURN_LEFT = 3;`
`int TURN_RIGHT = 4;`
`int ARC_LEFT = 5;`
`int ARC_RIGHT = 6;`
`int SLOW_FORWARD = 7;`
`int SLOW_BACKWARD = 8;`
`int SLOW_TURN_LEFT = 9;`
`int SLOW_TURN_RIGHT = 10;`
	Comandos de movimiento para la función **move**()

`int DRIVE_BIAS = 0;`	Corrección de velocidad entre ambos motores

`int MSEC_PER_REV = 1071;`
`int MSEC_PER_RAD = 170;`
`int REV_2 = MSEC_PER_REV / 2;`
`int REV_4 = MSEC_PER_REV / 4;`
`int REV_8 = MSEC_PER_REV / 8;`
	Estimaciones de tiempo en milisegundos para diferentes giros de las ruedas.

FUNCIONES	**DESCRIPCIÓN**
`void driveb(int trans, int rot)`	Accionamiento diferencial de los motores (traslación y rotación) con corrección de velocidad de los motores.
`void move(int operation)`	Función de comandos de alto nivel para los motores.
`int track(int trans_vel, int rot_vel, int clicks)`	Acciona los motores a la velocidad especificada hasta que las ruedas giren un número determinado de pulsos del encoder.

7 TAREAS: Test de los sensores y actuadores.

Cargaremos en el robot el programa de prueba *rwp_test.lis*. Este fichero se puede encontrar en el subdirectorio **/libs** (en algunos equipos puede encontrase en el subdirectorio **/libs/test**).

Una vez cargado se debe pulsar sucesivamente el botón de reset para seleccionar el test adecuado (ver figuras 1 y 2 para la ubicación de los sensores y actuadores):

- Test de pantalla LCD
- Test del altavoz
- Test sensores de luz
- Test de detector de infrarrojos
- Test micrófono.
- Test detectores de colisión (*bumpers*)
- Test de encoders
- Test de motores
- Test puerto digital
- Test puerto analógico

7.1 Ajuste del Sensor de Infrarrojos:

Se deberá ajustar el alcance de detección de los sensores de infrarrojos (ajuste de potenciómetro azul). Para ello ubicaremos un obstáculo frente a los emisores y ajustaremos el potenciómetro en el umbral de detección (ver figura 1).

7.2 Ajuste de deriva de los motores:

Así mismo debe ajustarse la deriva entre los dos motores ajustando la variable **DRIVE_BIAS** en el fichero **motores.c** de forma que cuando se envía un comando de traslación pura siga perfectamente una línea recta.

Para realizarlo deberá realizar un programa que incluya los ficheros **motores.c** , **comun.c** , **defines.c** y un fichero con la función **main()**. Este fichero ejecutará un movimiento de avance a diferentes velocidades (función **move()**) De forma empírica ajustaremos la variable **DRIVE_BIAS** ubicada en el fichero motores.c hasta que el robot se mueva en línea recta.

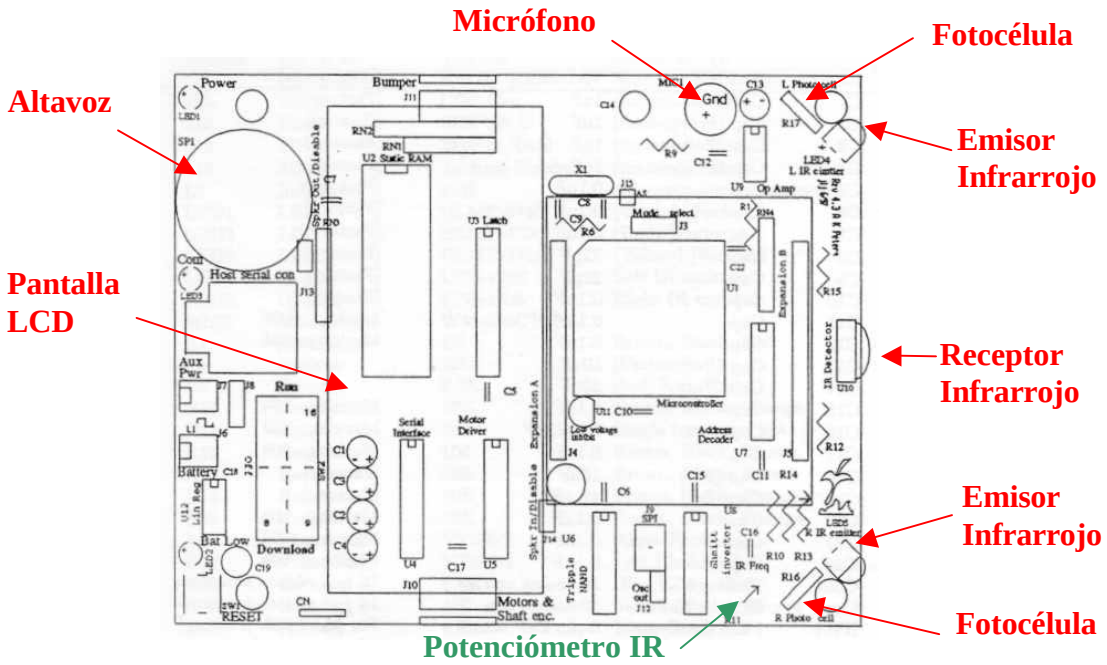

Figura 1 Distribución de los sensores de la tarjeta controladora

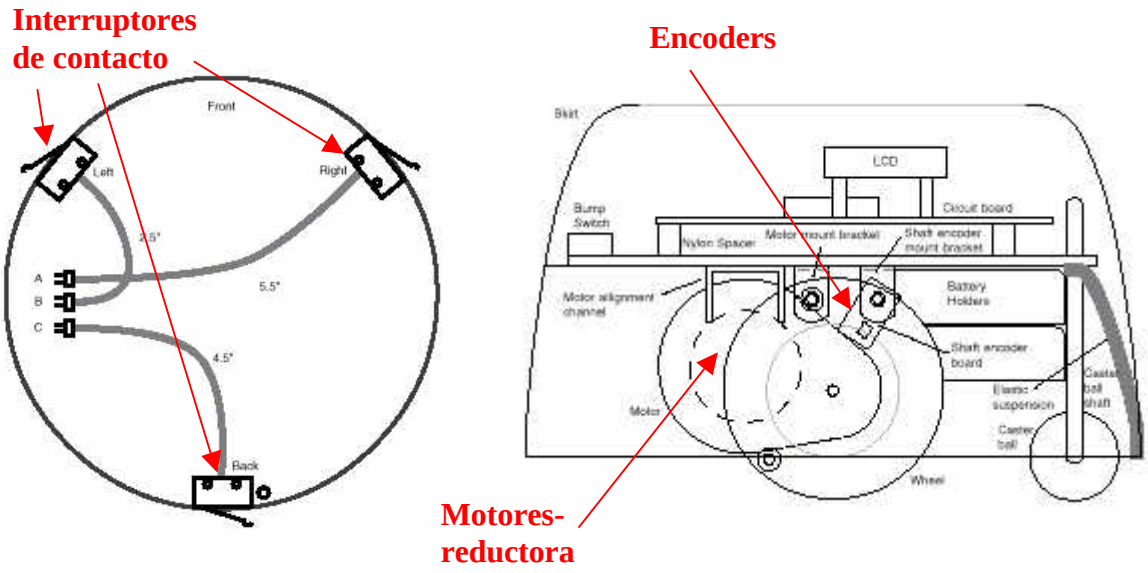

Figura 2 Distribución de los sensores y accionamientos en la estructura mecánica