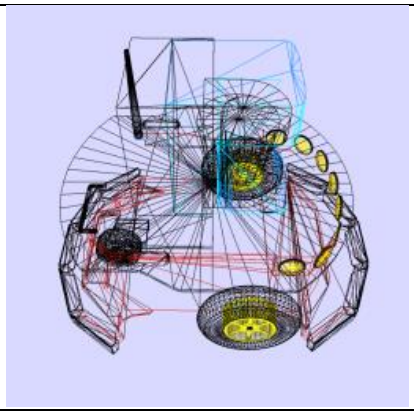
UNIVERSIDAD CARLOS III DE MADRID
PROGRAMACIÓN. GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES
CONVOCATORIA ORDINARIA MAYO 2014. PROBLEMAS

Apellidos:_____
Nombre:_____

Segunda parte. Problemas. Duración 2h 30 min.
Problema 1. (3,5 puntos del total del examen)

En la actualidad es común encontrar robots realizando tareas que era impensable que llevaran a cabo hace pocos años. Un ejemplo son los robots aspiradora que podemos encontrar en cualquier tienda de electrodomésticos. Una parte del éxito de la robótica se debe a los avances del hardware, y otra parte a los avances en el desarrollo de software, y en concreto a la capacidad de dotar a los robots de cierto grado de "inteligencia". Los simuladores son herramientas fundamentales el desarrollo de software para el control de robots, ya que nos permiten evaluar de forma rápida y segura su comportamiento ante distintas situaciones. Un simulador es un software que permite simular el comportamiento del robot y llevar a cabo pruebas que en el mundo real podrían no ser viables o seguras.
En este ejercicio se desarrollará parte de un programa que permita simular el comportamiento de varios robots, basado en los vehículos de *Braitenberg*. Este tipo de vehículos exhiben un comportamiento basado en la relación directa entre los diferentes sensores del robot (como sonar, láser, contacto, etc.) y sus diferentes actuadores (ruedas, brazos, etc.).
El ejercicio se basa en el robot Pioneer 3DX, una plataforma robótica para fines educativos y de investigación que permite la utilización de distintos sensores y actuadores. En la Figura 1 se muestra el P3DX simulado con sus dos ruedas, y la distribución de sus sensores de distancia (sonar).

(a)



(b)

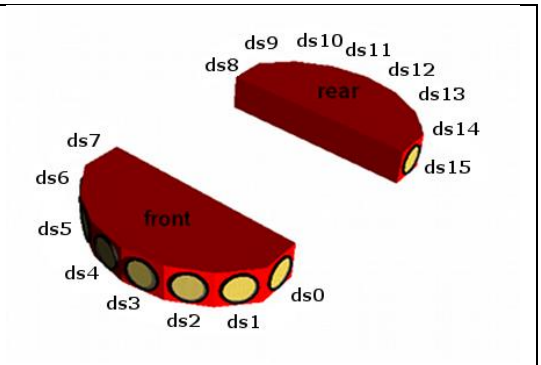


Fig. 1. Robot Pioneer 3DX simulado (a). Distribución de sensores sonar (b)

Para desarrollar el programa que simula el comportamiento de los robots es necesario completar los siguientes apartados.

Apartado 1 (2 puntos /10)
Defina un array de estructuras o registros (robots) que permita almacenar los datos de cada uno de los robots que forman parte de la simulación. Para cada robot se almacenará:

- Nombre
- Nombre de cada uno de los sensores de distancia (sónar) activos (que serán 8 o 16). Como se ve en la figura 1.b el nombre está formado por el número de sensor precedido del el prefijo "*ds*"
- Peso asignados a cada sensor para cada rueda (tipo `double`) . Estos valores servirán para calcular la velocidad de cada una de las dos ruedas motrices del robot P3DX. Cada sensor tendrá dos valores de peso diferentes, pues la medida de un sensor afectará de forma diferente al cálculo de la velocidad de cada rueda.
- Número de sensores sonar activos.

Apartado 2 (1 punto /10)
Escriba una función que inicialice la simulación. Esta función deberá obtener, pidiéndolos al usuario, el número de robots que van a participar en la simulación (máximo 10) y el número de veces que se va a repetir la simulación (número de ciclos de ejecución, máximo 10000). La función deberá controlar que los valores introducidos por el usuario son válidos.

Apartado 3 (2 puntos /10)
Escriba una función para inicializar un robot. La función deberá pedir al usuario el número de sensores sonar (de distancia) activos que tiene cada robot, que como se ha dicho para el P3DX puede ser 8 o 16, controlando que el valor introducido sea válido.
Después debe inicializar cada sensor, para lo que debe asignarle nombre de acuerdo con lo definido en el apartado uno, llamar a la función `inicializar_sensor` que activa el sensor para que luego pueda ser utilizado y pedir al usuario los pesos que desea asignar al sensor para cada una de las ruedas motrices.
Para inicializar los sensores se hará uso de la función (que se asume que existe ya, y no es necesario desarrollar) `inicializar_sensor(char nombre[])` de tipo `void`. Esta función recibe como parámetro el nombre de un sensor y se encarga de activarlo para que luego pueda ser utilizado

Nota: puede usar la función `sprintf` para concatenar el prefijo asignado al sensor (*"ds"*) y el número de sensor. La función `sprintf` funciona de manera similar a la función `printf` con la salvedad de que en vez de imprimir la cadena por pantalla, imprime en una cadena.

```
int sprintf(char *cadena, const char *formato, ...);
```

Por ejemplo, suponiendo que `nombre` contiene la cadena "`PEPITO`" y `num_s=8`, tras la llamada

```
sprintf(salida, "El Robot %s tiene %d sensores.", nombre, num_s);
```

la variable `salida` contendrá la cadena "`El Robot PEPITO tiene 8 sensores`".

Apartado 4 (3 puntos /10)
Escriba una función que mueva el robot para un paso (ciclo) de simulación. La función que leerá los valores de distancia medidos por los sensores, calculará la velocidad para cada rueda, y dará la orden de mover las ruedas.
Se asume que existe la función `sensor_get_value(char sensor[])` que recibe como parámetro el nombre del sensor y devuelve su lectura de distancia (`double`).
La velocidad de cada una de las dos ruedas motrices del robot será la suma de:

*peso_sensor * (1.0 – (valor_lectura_sensor /RANGO)*

donde el peso de cada sensor lo determina el usuario al inicio de la simulación y el RANGO es una constante definida en el código fuente.
Por otro lado, se asume que existe la función `wheels_set_speed(double S1,double S2)` de tipo `void` que lleva a cabo el movimiento del robot en un ciclo de simulación, y recibe como parámetro la velocidad de cada una de las dos ruedas.

Apartado 5 (2 puntos /10)
Implemente la función `main` que, basándose en las funciones anteriores, lleve a cabo la simulación. La función debe realizar las siguientes tareas:

- Inicializar la simulación y los robots.
- Ejecutar la simulación (para el número de ciclos determinado por el usuario) calculando en cada ciclo el movimiento de cada uno de los robots.

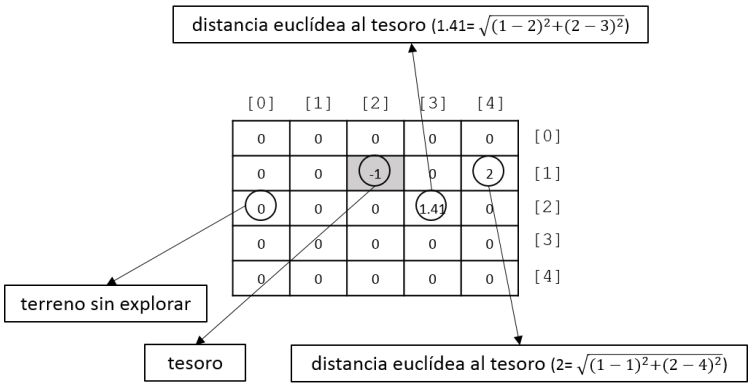
Problema 2. (3,5 puntos del total del examen)

Introducción

El juego *el tesoro escondido* consiste en lo siguiente: en un tablero de *N x M* casillas hay escondido un tesoro; el jugador tendrá que adivinar, tras una serie de intentos, cuáles son las coordenadas bajo las que se encuentra escondido. En cada turno el jugador indica las coordenadas donde cree que está enterrado el tesoro; se aplican las siguientes reglas:

- Si el jugador no acierta
    - Se le indica la distancia en línea recta a la que se encuentra el tesoro.
    - Si aún tiene la posibilidad, se le ofrece utilizar un radar de orientación, con cual podrá realizar dos tipos de exploraciones
        - Horizontal: comunica al usuario si el tesoro se encuentra al Este, al Oeste o en la misma longitud respecto de las últimas coordenadas de búsqueda introducidas.
        - Vertical: comunica al usuario si el tesoro se encuentra al Norte, al Sur o en la misma latitud respecto de las últimas coordenadas de búsqueda introducidas.
- Si el jugador acierta o se supera un número máximo de aciertos, se acaba la partida.

Para implementar este juego existirá una matriz *mapa* que contendrá un -1 en la casilla donde se encuentra el tesoro, un 0 las casillas que no hayan sido aún exploradas por el usuario y el valor de la distancia euclídea (en línea recta) al tesoro en las casillas que sí hayan sido exploradas.
Ejemplo:



Preguntas

1. Búsqueda (3/10)

Implemente una función llamada `buscar_tesoro` que calcule las coordenadas en las que se encuentra el tesoro (`fila_t, columna_t`). La cabecera de la función ha de ser la siguiente:

```
void buscar_tesoro(float mapa[N][M], int *fila_t, int *columna_t);
```

2. Cálculo de distancia (1/10)

Implemente una función llamada `distancia_tesoro` que actualice en la matriz *mapa* la distancia en línea recta desde las coordenadas de exploración (`fila_e, columna_e`) hasta el tesoro. La cabecera de la función ha de ser la siguiente:

```
void distancia_tesoro(float mapa[N][M], int fila_e, int columna_e);
```

Nota: Para calcular la distancia en línea recta hágase uso de la fórmula de la distancia euclídea $d = \sqrt{(f_t - f_e)^2 + (c_t - c_e)^2}$, en la que f_t y c_t se corresponden con la fila y la columna bajo la cual se encuentra escondido el tesoro y f_e y c_e con la fila y la columna de exploración. Puede hacer uso de las funciones potencia y raíz cuadrada, cuyas cabeceras son:

- `float pow(float base, float exp);`
- `float sqrt(float valor);`

3. Impresión del mapa (1/10)

Implemente una función llamada `imprimir_mapa` que muestre por pantalla la matriz *mapa* teniendo en cuenta que la casilla que contiene el tesoro tiene que ser mostrada como si no hubiera sido explorada. Las casillas no exploradas deben mostrarse con el valor 0 y las exploradas con el valor de la distancia calculada al tesoro. La cabecera de la función ha de ser la siguiente:

```
void imprimir_mapa(float mapa[N][M]);
```

4. Utilización del radar (2/10)

Implemente una función llamada `usar_radar` tal que, a partir de las coordenadas de exploración introducidas previamente por el usuario (`fila_e`, `columna_e`) y del tipo de radar a utilizar, devuelva el carácter 'N' si el tesoro se encuentra arriba de esas coordenadas de búsqueda, 'S' si se encuentra abajo (radar vertical), 'O' si se encuentra a la izquierda y 'E' si se encuentra a la derecha (radar horizontal). Si el tesoro se encuentra en la misma latitud o longitud, la función devolverá el carácter 'L'. La cabecera de la función ha de ser la siguiente:

```
char usar_radar(int tipo, float mapa[N][M], int fila_e, int columna_e);
```

5. Juego completo (3/10)

Implemente una función *main* que, utilizando las funciones anteriores, permita a un usuario jugar una partida a *el tesoro escondido*. Este programa principal deberá de:

1. Declarar las variables necesarias.
2. Inicializar la matriz *mapa*. Para esta inicialización se hará uso de una función que se asume ya implementada llamada `esconder_tesoro` que recibe como parámetros dicha matriz y no devuelve ningún valor. La matriz del mapa quedará con un -1 en una casilla aleatoria y con un 0 en el resto (terreno sin explorar).
3. Ir pidiendo coordenadas de búsqueda al jugador hasta que este gane o supere un número máximo de intentos dado por la expresión `(N + M) / 3`. Una vez que el usuario haya introducido las coordenadas, si no ha encontrado el tesoro, se imprimirá el mapa de búsqueda y se le ofrecerá la posibilidad de usar el radar si no lo ha utilizado previamente (se ha de preguntar al jugador el tipo de radar a usar). Si se utiliza el radar, se comunicará al usuario si el tesoro se encuentra al Norte, al Sur, al Este, al Oeste o si está en la misma latitud o longitud respecto de las coordenadas de exploración introducidas.
4. Terminada la partida, informar al usuario de si ha ganado o si por el contrario ha superado el número de intentos.