Universidad de Córdoba

Escuela Politécnica Superior de Córdoba

Sistemas en Tiempo Real
Segundo Curso de Segundo Ciclo
Ingeniería Informática
Curso 2013/2014

Memoria de prácticas

Francisco Arjona López
Cristóbal Castro Villegas
José Antonio Espino Palomares
Antonio Osuna Caballero

Córdoba, 11 de febrero de 2014

Índice general

Índice de figuras

Capítulo 1

Introducción

En esta memoria se recoge el trabajo realizado para la solución de las diferentes prácticas de la asignatura, donde en cada capítulo se recoge la solución de cada práctica y sus diferentes apartados.

El primer paso para realización de las prácticas ha sido el montaje del robot, quedando como se muestra en la Figura 1.1. A medida que sean necesarios diferentes sensores para la realización de las practicas se irán añadiendo.

Figura 1.1: Lego Mindstorms

En las prácticas se ha utilizado el lenguaje de implementación de **OSEK** [1].

Capítulo 2

Práctica 1. Manejo básico de Lego Mindstorms NXT: Sensores y Actuadores

2.1. Objetivos

Tras la realización de esta práctica el alumno debería ser capaz de:

- Programar movimientos sincronizados del robot.
- Conocer los parámetros de funcionamiento básico de los motores.
- Utilización de los sensores básicos.

2.2. Ejercicio A

Calibrar la potencia relativa de los motores para realizar un movimiento lineal con un error menor a 1cm. de desvío por cada 1 m. de avance lineal. (Error menor al 1 %).

2.2.1. Estrategia

El objetivo de este apartado es que el robot avance de manera lineal, sin que se produzcan desviaciones en su trayectoria.

Para ello se decide establecer la velocidad de los motores al 50 %, y que este corrija mediante una tarea denominada *calibrar* la velocidad de cada motor, para conseguir un momento lineal sin desviaciones.

2.2.2. Tareas

A continuación se realiza una descripción de las tareas utilizadas para la resolución de este apartado y su prioridad.

- **Avanzar**
  - *Prioridad*: 1

- *Descripción*: Se ejecuta durante 6000 milisegundos, que es el tiempo que corresponde aproximadamente al avance de un metro. Los motores se activan a una velocidad del 50 %.

- **Calibrar**

  - *Prioridad*: 2
  - *Descripción*: Se encarga de calibrar los motores con el fin de que el robot avance recto, esta asociada a la alarma *alarm1*. Pueden ocurrir dos casos:

    1. El motor C vaya a más revoluciones que el motor B, entonces para ello se comprueba que la velocidad de este último sea menor que la velocidad establecida al principio (con un margen de más 5), si se cumple se aumenta la velocidad del motor B, en el caso contrario se disminuye la velocidad del motor C.
    2. El caso contrario, que el motor B vaya a más revoluciones que el motor C, entonces se comprueba la velocidad que la velocidad de éste sea menor que la establecida al principio (con un margen de más 5), si se cumple se aumenta la velocidad del motor C, en en caso contrario se disminuye la velocidad del motor B.

- **Final**

  - *Prioridad*: 3
  - *Descripción*: Esta tarea hace detener el robot.

En la Figura 2.1, se muestra un gráfico de su funcionamiento.

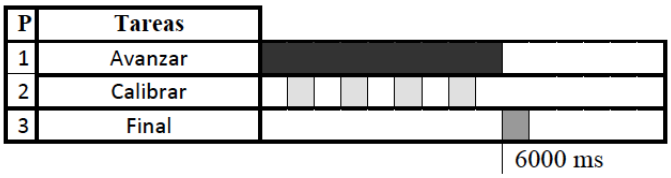

Figura 2.1: Funcionamiento práctica 1a.

2.2.3. Alarmas

Para la resolución de éste apartado ha sido necesaria la utilización de una alarma.

- **Alarm1**

  - *Periodo*: 500 ciclos
  - *Autoinicio*: Sí
  - *Descripción*: Esta alarma se encarga de ejecutar la tarea *calibrar* para que se produzca una corrección en los motores si fuese necesario.

2.2.4. Código del fichero practica1_a.c

```
/*-----------------------------------------------------------------------------
  Practica: 1.a

  Autores:
       - Francisco Arjona Lopez
       - Cristobal Castro Villegas
       - Jose Antonio Espino Palomares
       - Antonio Osuna Caballero
-----------------------------------------------------------------------------*/
#include "kernel.h"
#include "kernel_id.h"
#include "ecrobot_interface.h"

/*-----------------------------------------------------------------------------*/
/* OSEK declarations                                                           */
/*-----------------------------------------------------------------------------*/
DeclareTask(Avance);
DeclareTask(Final);
DeclareTask(Calibrar);
DeclareCounter(cnt1);
DeclareAlarm(alarm1);

/*-----------------------------------------------------------------------------*/
/* Definitions                                                                 */
/*-----------------------------------------------------------------------------*/
int OriginalSpeedB = 50, speedB;
int OriginalSpeedC = 50, speedC;

/*-----------------------------------------------------------------------------*/
/* Function to be invoked from a category 2 interrupt                          */
/*-----------------------------------------------------------------------------*/
void user_1ms_isr_type2(){
  SignalCounter(cnt1);
}

/*-----------------------------------------------------------------------------*/
// Task information:
// -----------------
// Name     : Avance
// Priority: 1
// Typ      : EXTENDED TASK
// Schedule: FULL
// Objective: Realizar el avance
/*-----------------------------------------------------------------------------*/
TASK(Avance)
{
    int time_out = systick_get_ms() + 6000 ;

  speedB = OriginalSpeedB;
  speedC = OriginalSpeedC;

    // Activa motores para comprobar si existe desvio en la navegacion
  nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
  nxt_motor_set_speed(NXT_PORT_C, speedC, 1);

  // Espera hasta que se agote el time_out
```

```
57    while(systick_get_ms() < time_out);

59    //CancelAlarm(alarm1);
      ActivateTask(Final);
61
        // Termina la tarea actual
63      TerminateTask();
}
65

67  /*-----------------------------------------------------------------------*/
    // Task information:
69  // ------------------
    // Name    : Final
71  // Priority: 3
    // Typ     : EXTENDED TASK
73  // Schedule: FULL
    // Objective: Parar los motores del robot
75  /*-----------------------------------------------------------------------*/
    TASK(Final)
77  {
        // Para los motores
79      nxt_motor_set_speed(NXT_PORT_B, 0, 1);
        nxt_motor_set_speed(NXT_PORT_C, 0, 1);
81
    // Cancela la alarma
83  CancelAlarm(alarm1);

85      // Termina la tarea actual
    TerminateTask();
87
    }
89  /*-----------------------------------------------------------------------*/
    // Task information:
91  // ------------------
    // Name    : Calibrar
93  // Priority: 2
    // Typ     : EXTENDED TASK
95  // Schedule: FULL
    // Objective: Se encargar de calibar las ruedas y que el
97  //        movimiento sea lineal
    /*-----------------------------------------------------------------------*/
99  TASK(Calibrar)
    {
101   int rpmB = nxt_motor_get_count(NXT_PORT_B);
      int rpmC = nxt_motor_get_count(NXT_PORT_C);
103
      // Muestra las velocidades de ambos motores por pantalla
105   display_goto_xy(0,0);
      display_int (rpmB, 3);
107   display_goto_xy(6,0);
      display_int (rpmC,3);
109   display_update();

111   //Correccion de velocidad en los motores para su calibracion
      if(rpmB < rpmC){
113     if(speedB<OriginalSpeedB+5) //Velocidad original (margen +5)
          speedB++;
```

```
115       else
            speedC--;
117       nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
          nxt_motor_set_speed(NXT_PORT_C, speedC, 1);
119     }

121     if(rpmC < rpmB){
          if(speedC<OriginalSpeedC+5) //Velocidad original (margen +5)
123         speedC++;
          else
125         speedB--;
          nxt_motor_set_speed(NXT_PORT_C, speedC, 1);
127       nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
        }
129
        nxt_motor_set_count(NXT_PORT_B,0);
131     nxt_motor_set_count(NXT_PORT_C,0);

133       // Termina la tarea actual
        TerminateTask();
135   }
```

2.2.5. Código del fichero practica1_a.oil

En este fichero se definen los valores por defecto con los que empieza la ejecución de cada tarea, prioridad, tipo de esquema, si debe iniciarse la tarea al encender el robot o no, etc.

```
1   /*-----------------------------------------------------------------
      Practica: 1.a
3
      Autores:
5         - Francisco Arjona Lopez
          - Cristobal Castro Villegas
7         - Jose Antonio Espino Palomares
          - Antonio Osuna Caballero
9   -----------------------------------------------------------------*/
    #include "implementation.oil"
11
    CPU ATMEL_AT91SAM7S256 // CPU del Lego NXT
13  {
      OS LEJOS_OSEK
15      {
          STATUS = EXTENDED;
17        STARTUPHOOK = FALSE;
          ERRORHOOK = FALSE;
19        SHUTDOWNHOOK = FALSE;
          PRETASKHOOK = FALSE;
21        POSTTASKHOOK = FALSE;
          USEGETSERVICEID = FALSE;
23        USEPARAMETERACCESS = FALSE;
          USERESSCHEDULER = FALSE;
25      };

27    APPMODE sample_appmode1{}; // Modo de aplicacion por defecto.

29      TASK Avance
```

```
    {
      AUTOSTART = TRUE
    {
        APPMODE = sample_appmode1;
      }; // Autoinicio en modo de aplicacion descrito
    // Si no se desea que se autoarranque: AUTOSTART = FALSE;

      PRIORITY  = 1;
      ACTIVATION  = 1;
      SCHEDULE  = FULL;
      STACKSIZE   = 512;
    };

    TASK Calibrar
  {
      AUTOSTART = FALSE;
      PRIORITY  = 2;
      ACTIVATION  = 1;
      SCHEDULE  = FULL;
      STACKSIZE   = 512;
    };

    TASK Final
    {
      AUTOSTART = FALSE;
        PRIORITY  = 3;
        ACTIVATION  = 1;
        SCHEDULE  = FULL;
        STACKSIZE   = 512;
    };

    COUNTER cnt1 {
    MINCYCLE = 1;
    MAXALLOWEDVALUE = 10000;
    TICKSPERBASE = 1;
  };

  ALARM alarm1 {
    COUNTER = cnt1;
    ACTION = ACTIVATETASK {
      TASK = Calibrar;
    };
    AUTOSTART = TRUE {
      ALARMTIME = 1;
      CYCLETIME = 500;
      APPMODE = sample_appmode1;
    };
  };
};
```

2.2.6. Resultados

En un principio se pensó realizar la práctica intentado poner a cada motor una potencia para que se produjera el movimiento lineal pero aún así se producía bastante desviación. Finalmente se decidió añadir una alarma que se encargara de calibrar los motores en el movimiento de avance, obteniendo con ello mejores resultados.

Vídeo: `http://www.youtube.com/watch?v=tBR9gWx_GH8`

2.3. Ejercicio B

Programar una maniobra de aparcamiento del robot sin tener en cuenta sensores. Se muestra una descripción del entorno de aparcamiento (Figura 2.2). El robot se posicionará con las ruedas motrices rozando la línea lateral y con las ruedas justo por delante de la línea de inicio.

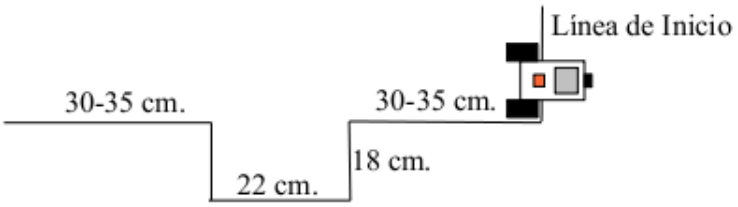

Figura 2.2: Diagrama del aparcamiento.

2.3.1. Estrategia

Como se describe en el enunciado se debe realizar la maniobra de aparcamiento sin tener en cuenta los sensores. Nuestra idea a sido simular el aparcamiento igual que sucede con los coches. Para ello se ha establecido el siguiente orden de movimiento:

1. Avanzar desde la linea de inicio hasta el punto 1, como se muestra en la Figura 2.3.

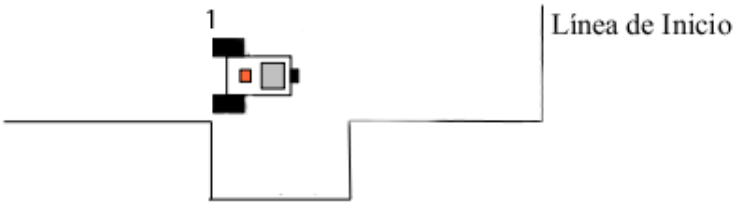

Figura 2.3: Diagrama del aparcamiento - Paso 1.

2. Girar a la derecha hacia atrás, hasta quedar en la posición que se muestra en la Figura 2.4.

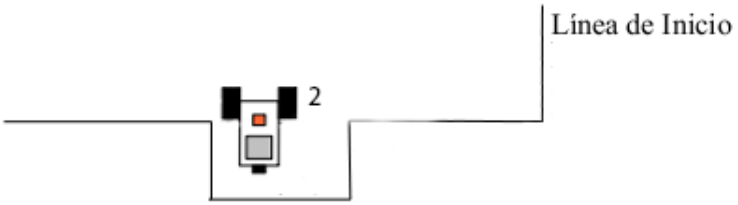

Figura 2.4: Diagrama del aparcamiento - Paso 2.

3. Girar a la izquierda hacia atrás, hasta quedar en la posición que se muestra en la Figura 2.5.

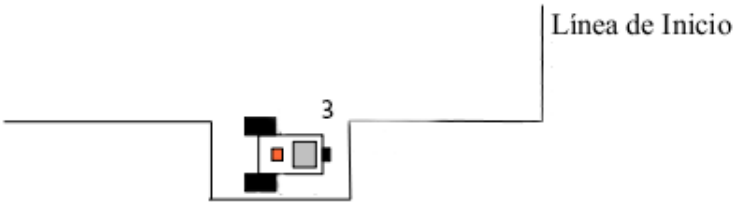

Figura 2.5: Diagrama del aparcamiento - Paso 3.

Para saber la distancia que se debe recorrer se calcula el numero de grados que se deben recorrer en cada momento, para ello se ha utilizado la siguiente fórmula:

$$g = \frac{360 \cdot d}{2 \cdot \pi \cdot r}$$

donde, g es el número de grados recorridos, d es la distancia que se quiere recorrer y r es el radio de las ruedas.

2.3.2. Tareas

A continuación se realiza una descripción de las tareas utilizadas para la resolución de este apartado y su prioridad.

- **Principal**
  - *Prioridad*: 1
  - *Descripción*: Se encarga de activar las tareas *Avance*, *GiroIzq*, *Retroceso*, *GiroIzq*.
- **GiroIzq**
  - *Prioridad*: 2
  - *Descripción*: Se encarga de realizar un giro hacia la izquierda hacia atrás, activando sólo para ello el motor C. Se ejecuta hasta que se realice el giro, ya que con la fórmula se ha calculado cuanto distancia debe durar.
- **Retroceso**
  - *Prioridad*: 3
  - *Descripción*: Se encarga de retroceder para ello se activan ambos motores con velocidad negativa. Se ejecuta hasta que se realice el retroceso, ya que con la fórmula se ha calculado.
- **GiroDer**
  - *Prioridad*: 4

- *Descripción*: Se encarga de realizar un giro hacia la derecha hacia atrás, activando sólo para ello el motor B. Se ejecuta hasta que se realice el giro, ya que con la fórmula se ha calculado cuanto distancia debe durar.

- **Avance**
  - *Prioridad*: 5
  - *Descripción*: Se ejecuta durante el tiempo que tarda en recorrer el la distancia que se ha calculado mediante la fórmula. Se encarga de realizar un avance lineal.

- **Calibrar**
  - *Prioridad*: 6
  - *Descripción*: Se encarga de calibrar los motores con el fin de que el robot avance recto, se activa mediante una alarma. Pueden ocurrir dos casos:
    1. El motor C vaya a más revoluciones que el motor B, entonces para ello se comprueba que la velocidad de este último sea menor que la velocidad establecida al principio (con un margen de más 5), si se cumple se aumenta la velocidad del motor B, en el caso contrario se disminuye la velocidad del motor C.
    2. El caso contrario, que el motor B vaya a más revoluciones que el motor C, entonces se comprueba la velocidad que la velocidad de éste sea menor que la establecida al principio (con un margen de más 5), si se cumple se aumenta la velocidad del motor C, en en caso contrario se disminuye la velocidad del motor B.

- **Final**
  - *Prioridad*: 7
  - *Descripción*: Esta tarea hace detener el robot.

En la siguiente Figura 2.6, se muestra un gráfico de su funcionamiento.

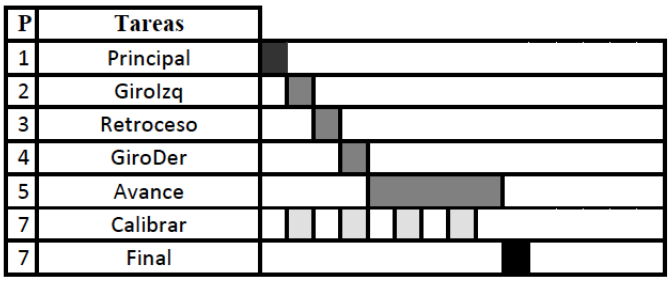

Figura 2.6: Funcionamiento práctica 1b.

2.3.3. Alarmas

Para la resolución de éste apartado ha sido necesaria la utilización de una alarma.

- **Alarma1**

- *Periodo*: 10 ciclos
- *Autoinicio*: Sí
- *Descripción*: Esta alarma se encarga de ejecutar la tarea *calibrar* para que se produzca una corrección en los motores si fuese necesario.

2.3.4. Código del fichero practica1_b.c

```
/*-----------------------------------------------------------------------
  Practica: 1.b

  Autores:
       - Francisco Arjona Lopez
       - Cristobal Castro Villegas
       - Jose Antonio Espino Palomares
       - Antonio Osuna Caballero
-----------------------------------------------------------------------*/
#include "kernel.h"
#include "kernel_id.h"
#include "ecrobot_interface.h"

/*-----------------------------------------------------------------------*/
/* OSEK declarations                                                     */
/*-----------------------------------------------------------------------*/
DeclareCounter(Contador);
DeclareTask(Retroceso);
DeclareTask(Avance);
DeclareTask(GiroIzq);
DeclareTask(GiroDer);
DeclareTask(Final);
DeclareTask(Calibrar);
DeclareTask(Principal);
DeclareAlarm(Alarma1);


/*-----------------------------------------------------------------------*/
/* Definitions                                                           */
/*-----------------------------------------------------------------------*/
int OriginalSpeedB = 50, speedB;
int OriginalSpeedC = 50, speedC;
int gradosMotorB = 0;
int gradosMotorC = 0;

/*-----------------------------------------------------------------------*/
/* Function to be invoked from a category 2 interrupt                    */
/*-----------------------------------------------------------------------*/
void user_1ms_isr_type2(){
  SignalCounter(Contador);
}

/*-----------------------------------------------------------------------*/
// Task information:
// -----------------
// Name     : Avance
// Priority: 5
// Typ      : EXTENDED TASK
// Schedule: FULL
```

```
// Objective: Realizar el avance
/*-----------------------------------------------------------------------------*/
TASK(Avance)
{
  speedB = OriginalSpeedB;
    speedC = OriginalSpeedC;

   // Activar servomotores para avanzar en linea recta
    nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
    nxt_motor_set_speed(NXT_PORT_C, speedC, 1);

  // grados de giro = (360 * recorrido) / (2 * PI * radio_ruedas)
  int gradosRecorrido = (360 * 53) / (2 * 3.1415 * 2.75) ;

  display_goto_xy(0,4);
  display_int (gradosRecorrido,5);
  display_update();

    // Espera hasta recorrer la distancia calculada
  while((gradosMotorB+nxt_motor_get_count(NXT_PORT_B)<gradosRecorrido) && (
      gradosMotorC+nxt_motor_get_count(NXT_PORT_C)<gradosRecorrido));


  // Cancela la alarma
  CancelAlarm(Alarma1);

    // Termina la tarea actual
    TerminateTask();

}

/*-----------------------------------------------------------------------------*/
// Task information:
// ------------------
// Name     : Retroceso
// Priority: 3
// Typ      : EXTENDED TASK
// Schedule: FULL
// Objective: Realizar el retroceso
/*-----------------------------------------------------------------------------*/
TASK(Retroceso)
{
  // Inicializar
  nxt_motor_set_count(NXT_PORT_C,0);
  nxt_motor_set_count(NXT_PORT_B,0);

    // Activar motores
    nxt_motor_set_speed(NXT_PORT_B, -speedB, 1);
    nxt_motor_set_speed(NXT_PORT_C, -speedC, 1);

  int gradosRecorrido = -((360 * 5) / (2 * 3.1415 * 2.75)) ;

    // Espera activa hasta hacer el retroceso
  while(nxt_motor_get_count(NXT_PORT_B)>gradosRecorrido &&
      nxt_motor_get_count(NXT_PORT_C)>gradosRecorrido);


    // Termina la tarea actual
```

```
    TerminateTask();

}

/*-----------------------------------------------------------------------------*/
// Task information:
// ------------------
// Name     : GiroIzq
// Priority: 2
// Typ      : EXTENDED TASK
// Schedule: FULL
// Objective: Realizar el giro a la izquierda.
/*-----------------------------------------------------------------------------*/
TASK(GiroIzq)
{
  nxt_motor_set_speed(NXT_PORT_C, 0, 1);
  nxt_motor_set_speed(NXT_PORT_B, 0, 1);

  // Inicializar
    nxt_motor_set_count(NXT_PORT_C,0);
  nxt_motor_set_count(NXT_PORT_B,0);

    // Activar servomotores para realizar un giro a la izquierda
    nxt_motor_set_speed(NXT_PORT_C, -speedC, 1);
  nxt_motor_set_speed(NXT_PORT_B, 0, 1);

  float recorridocm = (3.1415 * 10) / 2.0;
  // grados de giro = (360 * recorrido) / (2 * PI * radio_ruedas)
  int gradosRecorrido = -((360 * recorridocm) / (2 * 3.1415 * 2.75));

    // Espera hasta que se realice el giro a la izquierda
  while(nxt_motor_get_count(NXT_PORT_C)>gradosRecorrido);


   // Terminar la tarea actual
    TerminateTask();
}

/*-----------------------------------------------------------------------------*/
// Task information:
// ------------------
// Name     : GiroDer
// Priority: 4
// Typ      : EXTENDED TASK
// Schedule: FULL
// Objective: Realizar el giro a la derecha
/*-----------------------------------------------------------------------------*/
TASK(GiroDer)
{
  nxt_motor_set_speed(NXT_PORT_C, 0, 1);
  nxt_motor_set_speed(NXT_PORT_B, 0, 1);

  // Inicializar
    nxt_motor_set_count(NXT_PORT_C,0);
  nxt_motor_set_count(NXT_PORT_B,0);

    // Activar servomotores para realizar un giro a la derecha
    nxt_motor_set_speed(NXT_PORT_B, -speedC, 1);
```

```
  nxt_motor_set_speed(NXT_PORT_C, 0, 1);

  float recorridocm = (3.1415 * 11.5) / 2.0;
  // grados de giro = (360 * recorrido) / (2 * PI * radio_ruedas)
  int gradosRecorrido = -((360 * recorridocm) / (2 * 3.1415 * 2.75));

    // Espera hasta que se realice el giro a la derecha
  while(nxt_motor_get_count(NXT_PORT_B)>gradosRecorrido);

  // Despierta la tarea de parada de motores
    ActivateTask(Final);

   // Terminar la tarea actual
    TerminateTask();


}

/*--------------------------------------------------------------------------*/
// Task information:
// ------------------
// Name     : Final
// Priority: 7
// Typ      : EXTENDED TASK
// Schedule: FULL
// Objective: Parar los motores
/*--------------------------------------------------------------------------*/
TASK(Final)
{
    // Para los motores
    nxt_motor_set_speed(NXT_PORT_B, 0, 1);
    nxt_motor_set_speed(NXT_PORT_C, 0, 1);

    // Termina la tarea actual
  TerminateTask();

}

/*--------------------------------------------------------------------------*/
// Task information:
// ------------------
// Name     : Principal
// Priority: 1
// Typ      : EXTENDED TASK
// Schedule: FULL
// Objective: Activar las tareas
/*--------------------------------------------------------------------------*/
TASK(Principal)
{
    // Activar tareas
    ActivateTask(Avance);
    ActivateTask(GiroIzq);
  ActivateTask(Retroceso);
    ActivateTask(GiroDer);

    // Terminar la tarea actual
  TerminateTask();
}
```

```
223 /*-------------------------------------------------------------------------*/
    // Task information:
225 // -----------------
    // Name     : Calibrar
227 // Priority: 6
    // Typ      : EXTENDED TASK
229 // Schedule: FULL
    // Objective: Realizar la calibracion de los motores
231 /*-------------------------------------------------------------------------*/
TASK(Calibrar)
233 {
  int rpmB = nxt_motor_get_count(NXT_PORT_B);
235   int rpmC = nxt_motor_get_count(NXT_PORT_C);

237   // Muestra las velocidades de ambos motores por pantalla
  display_goto_xy(0,0);
239   display_int (rpmB, 3);
  display_goto_xy(6,0);
241   display_int (rpmC,3);
  display_update();
243
  //Correccion de velocidad en los motores para su calibracion
245   if(rpmB < rpmC){
    if(speedB<OriginalSpeedB+5) //Velocidad original (margen +5)
247       speedB++;
    else
249       speedC--;
    nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
251     nxt_motor_set_speed(NXT_PORT_C, speedC, 1);
  }
253
  if(rpmC < rpmB){
255     if(speedC<OriginalSpeedC+5) //Velocidad original (margen +5)
      speedC++;
257     else
      speedB--;
259     nxt_motor_set_speed(NXT_PORT_C, speedC, 1);
    nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
261   }

263   nxt_motor_set_count(NXT_PORT_B,0);
  nxt_motor_set_count(NXT_PORT_C,0);
265
    // Termina la tarea actual
267   TerminateTask();
}
```

2.3.5. Código del fichero practica1_b.oil

```
/*-------------------------------------------------------------------------
2   Practica: 1.b

4   Autores:
        - Francisco Arjona Lopez
6       - Cristobal Castro Villegas
        - Jose Antonio Espino Palomares
```

```
8         - Antonio Osuna Caballero
-------------------------------------------------------------------------------*/
10 #include "implementation.oil"

12 CPU ATMEL_AT91SAM7S256
   {
14   OS LEJOS_OSEK
       {
16       STATUS = EXTENDED;
         STARTUPHOOK = FALSE;
18       ERRORHOOK = FALSE;
         SHUTDOWNHOOK = FALSE;
20       PRETASKHOOK = FALSE;
         POSTTASKHOOK = FALSE;
22       USEGETSERVICEID = FALSE;
         USEPARAMETERACCESS = FALSE;
24       USERESSCHEDULER = FALSE;
       };
26
   APPMODE sample_appmode1{};
28
       COUNTER Contador
30     {
       MINCYCLE  = 1;
32     MAXALLOWEDVALUE = 10000;
       TICKSPERBASE  = 1;
34   };

36     ALARM Alarma1
       {
38     COUNTER   = Contador;
       ACTION    = ACTIVATETASK {
40       TASK  = Calibrar;
   };
42 AUTOSTART = TRUE {
        ALARMTIME  = 1;
44       CYCLETIME = 10;
         APPMODE = sample_appmode1;
46     };
   };
48
       TASK Principal
50     {
         AUTOSTART = TRUE
52     {
       APPMODE = sample_appmode1;
54        };
           PRIORITY  = 1;
56         ACTIVATION  = 1;
           SCHEDULE  = FULL;
58         STACKSIZE   = 512;
       };
60

62     TASK GiroIzq
   {
64         AUTOSTART = FALSE;
       PRIORITY  = 2;
```

```
66          ACTIVATION  = 1;
            SCHEDULE  = FULL;
68          STACKSIZE   = 512;
    };
70
     TASK GiroDer
72  {
              AUTOSTART = FALSE;
74          PRIORITY  = 4;
            ACTIVATION  = 1;
76          SCHEDULE  = FULL;
            STACKSIZE   = 512;
78  };

80  TASK Retroceso
    {
82
              AUTOSTART = FALSE;
84          PRIORITY  = 3;
            ACTIVATION  = 1;
86          SCHEDULE  = FULL;
            STACKSIZE   = 512;
88        };

90  TASK Avance
    {
92
              AUTOSTART = FALSE;
94          PRIORITY  = 5;
            ACTIVATION  = 1;
96          SCHEDULE  = FULL;
            STACKSIZE   = 512;
98        };

100     TASK Calibrar
    {
102
              AUTOSTART = FALSE;
104         PRIORITY  = 6;
            ACTIVATION  = 1;
106         SCHEDULE  = FULL;
            STACKSIZE   = 512;
108       };

110     TASK Final
      {
112         AUTOSTART = FALSE;
              PRIORITY  = 7;
114           ACTIVATION  = 1;
              SCHEDULE  = FULL;
116           STACKSIZE   = 512;
     };
118 };
```

2.3.6. Resultados

En el siguiente vídeo se puede observar como resuelve el problema de este apartado, destacar que se ha añadido también la alarma que tiene asociada la tarea calibrar para que el avance sea totalmente recto.

Video: `http://www.youtube.com/watch?v=N9PUJD5SiOQ`

2.4. Ejercicio C

Realizar un movimiento de avance del robot teniendo en cuenta los sensores de ultrasonidos y de pulsación, que estarán colocados en el frontal de avance del robot. El robot avanzará a plena potencia hasta que se encuentre a 1 m. de un obstáculo, que reducirá su potencia a la mitad. A 20 cm. de distancia del obstáculo, el robot reducirá su potencia de avance a un cuarto. El avance continuará hasta que el choque sea detectado por el sensor de pulsación, en cuyo caso, el robot retrocederá durante 1 seg. y girará 90º a la derecha, comenzando de nuevo el mismo procedimiento.

2.4.1. Estrategia

Como se indica en el enunciado, el robot debe avanzar a máxima velocidad, para ello hemos decidido definir la tarea avance la cual activará los motores del robot a máxima velocidad.

Mediante una alarma asociada a la tarea comprobar, como su nombre indica, se encargará de comprobar la distancia a la que se encuentran los obstáculos con el sensor de ultrasonidos, si esta distancia es menor de *1 m* se reducirá la potencia a la mitad, cuando la distancia sea inferior a *20 cm* se reducirá la potencia a un cuarto, en el momento que se produzca un choque, el cual se comprobará con el sensor de pulsación, se activará la tarea retroceder, esta tarea hará que el robot retroceda durante *1 seg.* y seguidamente se activará la tarea girar a la derecha. Una vez se produce el giro a la derecha se vuelve a activar la tarea avanzar de nuevo.

2.4.2. Tareas

A continuación se realiza una descripción de las tareas utilizadas para la resolución de este apartado y su prioridad.

- **Principal**
  - *Prioridad*: 1
  - *Descripción*: Se encarga de activar la tarea *Avance*.
- **Avance**
  - *Prioridad*: 2
  - *Descripción*: Se encarga de activar los motores a una velocidad del 100 %. Se ejecuta mientras no se active una tarea mayor prioridad.

- **Mitad_Potencia**
  - *Prioridad*: 4
  - *Descripción*: Se encarga de reducir los motores a la mitad de potencia.
- **Cuarto_Potencia**
  - *Prioridad*: 5
  - *Descripción*: Se encarga de reducir los motores a un cuarto de potencia.
- **Retroceso**
  - *Prioridad*: 6
  - *Descripción*: Se encarga de retroceder durante un segundo y activar la tarea *Giro_Dcha*.
- **Giro_Dcha**
  - *Prioridad*: 7
  - *Descripción*: Se encarga de realizar el giro a la derecha, para ello sólo el motor C.
- **Comprueba_Distancia**
  - *Prioridad*: 10
  - *Descripción*: Tarea asociada a la alarma *alarm1*, la cual se encarga de comprobar la distancia a la que se encuentra un obstáculo, para ello realiza seis medidas a través del sensor de ultrasonidos y calcula la media, en función de la distancia a la que se encuentre el obstáculo se activarán diferentes tareas.
    1. Si se encuentra entre una distancia menor de *1 metro* y superior a *20 cm*, se activa la tarea *Mitad_Potencia* reduciendo la velocidad del robot a la mitad.
    2. Si se encuentra a una distancia inferior a *20 cm*, se activa la tarea *Cuarto_Potencia* y la velocidad del robot se reduce a un cuarto.

    Si el cualquier momento se activa el sensor de pulsación, esto producirá que se active la tarea *retroceso*.

En la Figura 2.7, se muestra un ejemplo donde el robot se encuentra un obstáculo:

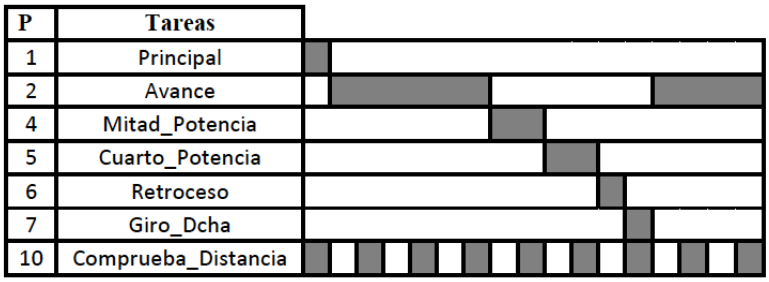

Figura 2.7: Funcionamiento práctica 1c.

2.4.3. Alarmas

Para la resolución de éste apartado ha sido necesaria la utilización de una alarma.

- **Alarm1**
  - *Periodo*: 200 ciclos
  - *Autoinicio*: Sí
  - *Descripción*: Esta alarma se encarga de ejecutar la tarea *Comprueba_Distancia* la cual se encarga de calcular la distancia a la que se encuentran los obstáculos y activar diferentes tareas en función de la distancia que se encuentre el obstáculo.

2.4.4. Código del fichero practica1_c.c

```
/*-----------------------------------------------------------------------
  Practica: 1.c

  Autores:
       - Francisco Arjona Lopez
       - Cristobal Castro Villegas
       - Jose Antonio Espino Palomares
       - Antonio Osuna Caballero
-----------------------------------------------------------------------*/
#include "kernel.h"
#include "kernel_id.h"
#include "ecrobot_interface.h"

#define SONAR_PORT NXT_PORT_S1
#define PULSADOR1_PORT NXT_PORT_S2


/*-----------------------------------------------------------------------*/
/* OSEK declarations                                                     */
/*-----------------------------------------------------------------------*/
DeclareTask(Principal);
DeclareTask(Avance);
DeclareTask(Final);
DeclareTask( Comprueba_Distancia );
DeclareTask( Mitad_Potencia );
DeclareTask( Cuarto_Potencia );
DeclareTask( Retroceso );
DeclareTask( Giro_Dcha );
DeclareCounter(ContadorDistancia);



void ecrobot_device_initialize()
{
    // Inicializar los sensores activos
    ecrobot_init_sonar_sensor( NXT_PORT_S4 );

}
void ecrobot_device_terminate()
{
    // Finalizar los sensores activos
    ecrobot_term_sonar_sensor( NXT_PORT_S4 );
```

```
43 }
45 /*--------------------------------------------------------------------------*/
   /* Definitions                                                              */
47 /*--------------------------------------------------------------------------*/
49 int OriginalSpeedB = 100, speedB;
   int OriginalSpeedC = 100, speedC;
51
   /*--------------------------------------------------------------------------*/
53 /* Function to be invoked from a category 2 interrupt                       */
   /*--------------------------------------------------------------------------*/
55 void user_1ms_isr_type2(){

57     //SignalCounter(Contador);
       SignalCounter(ContadorDistancia);
59 }

61 /*--------------------------------------------------------------------------*/
   // Task information:
63 // -----------------
   // Name     : Avance
65 // Priority: 2
   // Typ      : EXTENDED TASK
67 // Schedule: FULL
   // Objective: Realizar el avance
69 /*--------------------------------------------------------------------------*/
   TASK(Avance)
71 {
       nxt_motor_set_speed(NXT_PORT_B, OriginalSpeedB, 1);
73     nxt_motor_set_speed(NXT_PORT_C, OriginalSpeedC, 1);

75     // Terminar la tarea actual
       TerminateTask();
77
   }
79
   /*--------------------------------------------------------------------------*/
81 // Task information:
   // -----------------
83 // Name     : Mitad_Potencia
   // Priority: 4
85 // Typ      : EXTENDED TASK
   // Schedule: FULL
87 // Objective: Reducir la potencia a la mitad
   /*--------------------------------------------------------------------------*/
89 TASK( Mitad_Potencia )
   {
91     nxt_motor_set_speed(NXT_PORT_B, OriginalSpeedB/2, 1);
       nxt_motor_set_speed(NXT_PORT_C, OriginalSpeedC/2, 1);
93
       // Terminar la tarea actual
95     TerminateTask();

97 }

99 /*--------------------------------------------------------------------------*/
   // Task information:
```

```
// ------------------
// Name     : Cuarto_Potencia
// Priority: 5
// Typ      : EXTENDED TASK
// Schedule: FULL
// Objective: Reducir un cuarto la potencia
/*-----------------------------------------------------------------------*/
TASK( Cuarto_Potencia )
{
    nxt_motor_set_speed(NXT_PORT_B, OriginalSpeedB/4, 1);
    nxt_motor_set_speed(NXT_PORT_C, OriginalSpeedC/4, 1);

    // Terminar la tarea actual
    TerminateTask();

}

/*-----------------------------------------------------------------------*/
// Task information:
// ------------------
// Name     : Principal
// Priority: 1
// Typ      : EXTENDED TASK
// Schedule: FULL
// Objective: Activar la tarea avance
/*-----------------------------------------------------------------------*/
TASK(Principal)
{
     // Activar la tarea avance
    ActivateTask( Avance );

    // Terminar la tarea actual
    TerminateTask();

}

/*-----------------------------------------------------------------------*/
// Task information:
// ------------------
// Name     : Retroceso
// Priority: 6
// Typ      : EXTENDED TASK
// Schedule: FULL
// Objective: Realizar el retroceso durante 1 seg.
/*-----------------------------------------------------------------------*/
TASK( Retroceso )
{

    nxt_motor_set_speed(NXT_PORT_B, -100, 1);
    nxt_motor_set_speed(NXT_PORT_C, -100, 1);

    systick_wait_ms(1000);

    // Activar la tarea giro_dcha
    ActivateTask( Giro_Dcha );

    // Terminar la tarea actual
    TerminateTask();
```

```
159 }
161 /*-----------------------------------------------------------------------------*/
    // Task information:
163 // -------------------
    // Name     : Giro_Dcha
165 // Priority: 7
    // Typ      : EXTENDED TASK
167 // Schedule: FULL
    // Objective: Realizar el el giro a la derecha.
169 /*-----------------------------------------------------------------------------*/
    TASK( Giro_Dcha )
171 {
        nxt_motor_set_count(NXT_PORT_C, 0);
173
        // Activamos motores
175     nxt_motor_set_speed(NXT_PORT_C, 40, 1);
        nxt_motor_set_speed(NXT_PORT_B, 0, 1);
177
        // Espera activa hasta hacer el giro
179     while(nxt_motor_get_count(NXT_PORT_C)<350);

181     // Terminar la tarea actual
        TerminateTask();
183
    }
185 /*-----------------------------------------------------------------------------*/
    // Task information:
187 // -------------------
    // Name     : Comprueba_Distancia
189 // Priority: 10
    // Typ      : EXTENDED TASK
191 // Schedule: FULL
    // Objective: Comprobar la distancia a la que se encuentra los obstaculos.
193 /*-----------------------------------------------------------------------------*/
    TASK (Comprueba_Distancia)
195 {
        int contador = 1;
197
        int media = 0;
199
        int distancia;
201
        int sensorPresion;
203
      sensorPresion = ecrobot_get_touch_sensor( NXT_PORT_S1 );
205
      //Si se pulsa el sensor de presion
207   if( sensorPresion == 1 ){

209     // Activar la tarea retroceso
        ActivateTask( Retroceso );
211
        // Terminar la tarea actual
213         TerminateTask();
        return;
215      }
```

```
217   //Se realizan 6 mediciones y se calcula la media
      while( contador < 6){
219         distancia = ecrobot_get_sonar_sensor( NXT_PORT_S4 );

221         media += distancia;
            contador++;
223     }

225     distancia = media / contador;

227     if( distancia <= 100 && distancia > 20 ){

229     // Activar la tarea Mitad_Potencia
        ActivateTask( Mitad_Potencia );
231
        // Terminar la tarea actual
233       TerminateTask();

235     }

237     else if( distancia <= 20 ){

239         int sensorPresion;

241         sensorPresion = ecrobot_get_touch_sensor( NXT_PORT_S1 );

243         if( sensorPresion == 1 ){

245             // Activar la tarea retroceso
          ActivateTask( Retroceso );
247
          // Terminar la tarea actual
249           TerminateTask();
            }
251
            else{
253
                // Activar la tarea Cuarto_Potencia
255             ActivateTask( Cuarto_Potencia );

257         // Terminar la tarea actual
                TerminateTask();
259         }
        }
261
        else{
263       // Activar la tarea Cuarto_Potencia
          ActivateTask( Avance );
265
          // Terminar la tarea actual
267       TerminateTask();
        }
269
}
```

2.4.5. Código del fichero practica1_c.oil

```
/*-----------------------------------------------------------------------------
  Practica: 1.c

  Autores:
       - Francisco Arjona Lopez
       - Cristobal Castro Villegas
       - Jose Antonio Espino Palomares
       - Antonio Osuna Caballero
-----------------------------------------------------------------------------*/
#include "kernel.h"
#include "kernel_id.h"
#include "ecrobot_interface.h"

#define SONAR_PORT NXT_PORT_S1
#define PULSADOR1_PORT NXT_PORT_S2


/*-----------------------------------------------------------------------------*/
/* OSEK declarations                                                           */
/*-----------------------------------------------------------------------------*/
DeclareTask(Principal);
DeclareTask(Avance);
DeclareTask(Final);
DeclareTask( Comprueba_Distancia );
DeclareTask( Mitad_Potencia );
DeclareTask( Cuarto_Potencia );
DeclareTask( Retroceso );
DeclareTask( Giro_Dcha );
DeclareCounter(ContadorDistancia);



void ecrobot_device_initialize()
{
    // Inicializar los sensores activos
    ecrobot_init_sonar_sensor( NXT_PORT_S4 );

}
void ecrobot_device_terminate()
{
    // Finalizar los sensores activos
    ecrobot_term_sonar_sensor( NXT_PORT_S4 );
}

/*-----------------------------------------------------------------------------*/
/* Definitions                                                                 */
/*-----------------------------------------------------------------------------*/

int OriginalSpeedB = 100, speedB;
int OriginalSpeedC = 100, speedC;

/*-----------------------------------------------------------------------------*/
/* Function to be invoked from a category 2 interrupt                          */
/*-----------------------------------------------------------------------------*/
void user_1ms_isr_type2(){

    //SignalCounter(Contador);
```

```
58      SignalCounter(ContadorDistancia);
    }
60
    /*--------------------------------------------------------------------------*/
62  // Task information:
    // -----------------
64  // Name     : Avance
    // Priority: 2
66  // Typ      : EXTENDED TASK
    // Schedule: FULL
68  // Objective: Realizar el avance
    /*--------------------------------------------------------------------------*/
70  TASK(Avance)
    {
72      nxt_motor_set_speed(NXT_PORT_B, OriginalSpeedB, 1);
        nxt_motor_set_speed(NXT_PORT_C, OriginalSpeedC, 1);
74
        // Terminar la tarea actual
76      TerminateTask();

78  }

80  /*--------------------------------------------------------------------------*/
    // Task information:
82  // -----------------
    // Name     : Mitad_Potencia
84  // Priority: 4
    // Typ      : EXTENDED TASK
86  // Schedule: FULL
    // Objective: Reducir la potencia a la mitad
88  /*--------------------------------------------------------------------------*/
    TASK( Mitad_Potencia )
90  {
        nxt_motor_set_speed(NXT_PORT_B, OriginalSpeedB/2, 1);
92      nxt_motor_set_speed(NXT_PORT_C, OriginalSpeedC/2, 1);

94      // Terminar la tarea actual
        TerminateTask();
96
    }
98
    /*--------------------------------------------------------------------------*/
100 // Task information:
    // -----------------
102 // Name     : Cuarto_Potencia
    // Priority: 5
104 // Typ      : EXTENDED TASK
    // Schedule: FULL
106 // Objective: Reducir un cuarto la potencia
    /*--------------------------------------------------------------------------*/
108 TASK( Cuarto_Potencia )
    {
110     nxt_motor_set_speed(NXT_PORT_B, OriginalSpeedB/4, 1);
        nxt_motor_set_speed(NXT_PORT_C, OriginalSpeedC/4, 1);
112
        // Terminar la tarea actual
114     TerminateTask();
```

```
116 }
118 /*-----------------------------------------------------------------------*/
    // Task information:
120 // -------------------
    // Name     : Principal
122 // Priority: 1
    // Typ      : EXTENDED TASK
124 // Schedule: FULL
    // Objective: Activar la tarea avance
126 /*-----------------------------------------------------------------------*/
    TASK(Principal)
128 {
        // Activar la tarea avance
130     ActivateTask( Avance );

132     // Terminar la tarea actual
        TerminateTask();
134
    }
136
    /*-----------------------------------------------------------------------*/
138 // Task information:
    // -------------------
140 // Name     : Retroceso
    // Priority: 6
142 // Typ      : EXTENDED TASK
    // Schedule: FULL
144 // Objective: Realizar el retroceso durante 1 seg.
    /*-----------------------------------------------------------------------*/
146 TASK( Retroceso )
    {
148
        nxt_motor_set_speed(NXT_PORT_B, -100, 1);
150     nxt_motor_set_speed(NXT_PORT_C, -100, 1);

152     systick_wait_ms(1000);

154     // Activar la tarea giro_dcha
        ActivateTask( Giro_Dcha );
156
        // Terminar la tarea actual
158     TerminateTask();
    }
160
    /*-----------------------------------------------------------------------*/
162 // Task information:
    // -------------------
164 // Name     : Giro_Dcha
    // Priority: 7
166 // Typ      : EXTENDED TASK
    // Schedule: FULL
168 // Objective: Realizar el el giro a la derecha.
    /*-----------------------------------------------------------------------*/
170 TASK( Giro_Dcha )
    {
172     nxt_motor_set_count(NXT_PORT_C, 0);
```

```
174     // Activamos motores
        nxt_motor_set_speed(NXT_PORT_C, 40, 1);
176     nxt_motor_set_speed(NXT_PORT_B, 0, 1);

178     // Espera activa hasta hacer el giro
        while(nxt_motor_get_count(NXT_PORT_C)<350);
180
        // Terminar la tarea actual
182     TerminateTask();

184 }
    /*-----------------------------------------------------------------------*/
186 // Task information:
    // -----------------
188 // Name     : Comprueba_Distancia
    // Priority: 10
190 // Typ      : EXTENDED TASK
    // Schedule: FULL
192 // Objective: Comprobar la distancia a la que se encuentra los obstaculos.
    /*-----------------------------------------------------------------------*/
194 TASK (Comprueba_Distancia)
    {
196     int contador = 1;

198     int media = 0;

200     int distancia;

202     int sensorPresion;

204   sensorPresion = ecrobot_get_touch_sensor( NXT_PORT_S1 );

206   //Si se pulsa el sensor de presion
      if( sensorPresion == 1 ){
208
        // Activar la tarea retroceso
210     ActivateTask( Retroceso );

212     // Terminar la tarea actual
          TerminateTask();
214     return;
         }
216
      //Se realizan 6 mediciones y se calcula la media
218   while( contador < 6){
          distancia = ecrobot_get_sonar_sensor( NXT_PORT_S4 );
220
          media += distancia;
222       contador++;
      }
224
      distancia = media / contador;
226
      if( distancia <= 100 && distancia > 20 ){
228
      // Activar la tarea Mitad_Potencia
230   ActivateTask( Mitad_Potencia );
```

```
232     // Terminar la tarea actual
          TerminateTask();
234
        }
236
        else if( distancia <= 20 ){
238
            int sensorPresion;
240
            sensorPresion = ecrobot_get_touch_sensor( NXT_PORT_S1 );
242
            if( sensorPresion == 1 ){
244
                // Activar la tarea retroceso
246       ActivateTask( Retroceso );

248       // Terminar la tarea actual
              TerminateTask();
250         }

252         else{

254             // Activar la tarea Cuarto_Potencia
                ActivateTask( Cuarto_Potencia );
256
            // Terminar la tarea actual
258             TerminateTask();
            }
260     }

262     else{
          // Activar la tarea Cuarto_Potencia
264         ActivateTask( Avance );

266         // Terminar la tarea actual
            TerminateTask();
268     }

270 }
```

2.4.6. Resultados

En el siguiente vídeo se muestra el resultado de la práctica.

Vídeo: `http://www.youtube.com/watch?v=YRg_E0FvMig`

Capítulo 3

Práctica 2. Manejo básico de Lego Mindstorms NXT: Tareas y Comunicaciones

3.1. Objetivos

Tras la realización de esta práctica el alumno debería ser capaz de:

- Acceder a sensores de manera efectiva.
- Realizar programación concurrente con memoria compartida
- Realizar comunicaciones distribuidas mediante el modelo cliente-servidor.

3.2. Ejercicio A

Construir y programar un robot con un sensor de ultrasonidos que sea capaz de encontrar la salida de un "laberinto" (Dicho laberinto será una concatenación de pequeñas estancias rectangulares con una única entrada y una única salida, como puede observarse en la Figura 3.1). Se incorporará un sensor de pulsación en la parte superior para indicarle al robot que ha salido del laberinto.

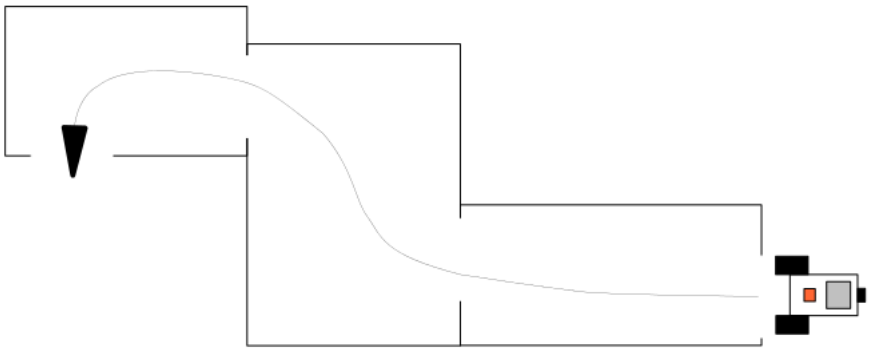

Figura 3.1: Diagrama del laberinto.

3.2.1. Estrategia

Para resolución de este apartado ha sido necesario añadir al robot el sensor de utrasonidos para encontrar la salida del laberinto y dos sensores de pulsación, uno delantero para detectar que se ha producido un choque con alguna de las paredes del laberinto y otro superior para indicar al robot que se ha salido del laberinto.

La solución propuesta para salir del laberinto ha sido la siguiente, una tarea principal la cual se encarga de avanzar mientras no se produzca choque o se pulse el pulsador de fin de laberinto, si se produce el choque con alguna pared del laberinto retrocederá y activará la tarea giro que se encargará de girar el robot 90 grados hacia la izquierda y derecha y en cada giro medir la distancia, el robot elegirá el camino en el que la distancia sea mayor.

También se tendrá una tarea pulsador asociada a una alarma, encargada de comprobar si se ha pulsado el sensor de fin de laberinto.

3.2.2. Tareas

A continuación se realiza una descripción de las tareas utilizadas para la resolución de este apartado y su prioridad.

- **Principal**
  - *Prioridad*: 1
  - *Descripción*: Tarea que se ejecuta mientra la bandera fin se encuentre desactivada, es decir mientras no se pulse el pulsador de fin de laberinto. Esta tarea se encargará de que el robot avance, en el momento que se produzca un choque con algunas de las paredes del laberinto el robot retrocederá y activará la tarea *giro*.
- **Giro**
  - *Prioridad*: 2
  - *Descripción*: La tarea *giro* se encargará de:
    1. Girar el robot *90 grados* hacia la derecha y realizar 10 mediciones con el sensor de ultrasonidos y calcular la distancia media.
    2. Girar el robot *90 grados* hacia la izquierda y realizar 10 mediciones con el sensor de ultrasonidos y calcular al distancia media.

    Una vez realizada las mediciones a ambos lados, el robot comprueba si la distancia a la derecha es mayor que a la izquierda, si es cierto, se vuelve a girar el robot hacia la derecha, en caso contrario, no se gira ya que se encuentra en la mejor dirección.
- **Pulsador**
  - *Prioridad*: 3
  - *Descripción*: Tarea asociada a la alarma *alarm1*, la cual se encargará de comprobar que se ha pulsado el sensor fin de laberinto y pondrá la bandera fin a 1 para que la tarea *principal* finalice.

En la Figura 3.2, se muestra un ejemplo donde el robot intenta salir del laberinto:

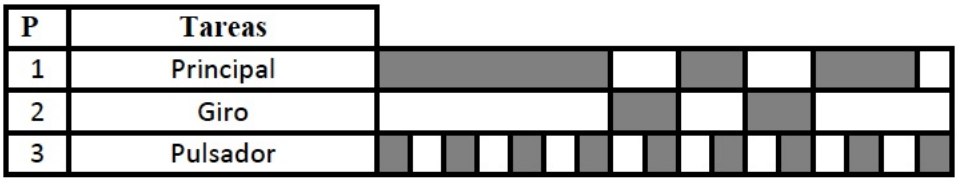

Figura 3.2: Funcionamiento práctica 2a.

3.2.3. Alarmas

- **Alarm1**
  - *Periodo*: 100 ciclos
  - *Autoinicio*: Sí
  - *Descripción*: Esta alarma se encarga de ejecutar la tarea *Pulsador* la cual se encarga de comprobar de detectar que se ha pulsado el pulsador fin de laberinto.

3.2.4. Código del fichero practica2_a.c

```
/*-----------------------------------------------------------------------
  Practica: 2.a

  Autores:
       - Francisco Arjona Lopez
       - Cristobal Castro Villegas
       - Jose Antonio Espino Palomares
       - Antonio Osuna Caballero
-----------------------------------------------------------------------*/
#include "kernel.h"
#include "kernel_id.h"
#include "ecrobot_interface.h"

#define PULSADOR_CHOQUE NXT_PORT_S1
#define PULSADOR_FIN NXT_PORT_S2
#define SENSOR_ULTRASONIDOS NXT_PORT_S4


/*-----------------------------------------------------------------------*/
/* OSEK declarations                                                     */
/*-----------------------------------------------------------------------*/
DeclareTask(Giro);
DeclareTask(Principal);
DeclareTask(Pulsador);
DeclareCounter(Contador);
DeclareAlarm(alarm1);


/*-----------------------------------------------------------------------*/
/* Definitions                                                           */
/*-----------------------------------------------------------------------*/
```

```
32 int fin = 0; //indica el fin de la ejecucion
   int pulsador_choque=0;
34 int OriginalSpeedB = 50, speedB = 50;
   int OriginalSpeedC = 50, speedC = 50;
36
   void ecrobot_device_initialize()
38 {
     // Inicializar los sensores activos
40   ecrobot_init_sonar_sensor(SENSOR_ULTRASONIDOS);
   }
42
   void ecrobot_device_terminate()
44 {
     // Finalizar los sensores activos
46   ecrobot_term_sonar_sensor(SENSOR_ULTRASONIDOS);
   }
48


50 /*-----------------------------------------------------------------------*/
   /* Function to be invoked from a category 2 interrupt                    */
52 /*-----------------------------------------------------------------------*/
   void user_1ms_isr_type2(){
54
     SignalCounter(Contador); //pone en marcha el contador
56 }

58 /*-----------------------------------------------------------------------*/
   // Task information:
60 // -----------------
   // Name     : Giro
62 // Priority: 2
   // Typ      : EXTENDED TASK
64 // Schedule: FULL
   // Objective: Girar a la derecha y a la izquierda para comprobar la
66 //         distancia y elegir la mejor opcion
   /*-----------------------------------------------------------------------*/
68 TASK(Giro)
   {
70
     //Inicializamos para medir la distancia a uno y otro lado
72   int distanciaDcha = 0, distanciaIzq = 0, media=0;
     int motorB = 1, motorC = 1;
74   int i;

76   /* Comprobamos qué distancia hay al girar a la derecha */

78   // Inicializamos contadores
     nxt_motor_set_count(NXT_PORT_B,0);
80   nxt_motor_set_count(NXT_PORT_C,0);

82   // Activamos motores para girar a la dcha
     nxt_motor_set_speed(NXT_PORT_B, -40, 1);
84   nxt_motor_set_speed(NXT_PORT_C, 40, 1);

86   while (motorB || motorC){
       if(nxt_motor_get_count(NXT_PORT_B) <= -180){
88       nxt_motor_set_speed(NXT_PORT_B, 0, 1);
         motorB =0;
```

```
90      }
92      if(nxt_motor_get_count(NXT_PORT_C) >= 180){
          nxt_motor_set_speed(NXT_PORT_C, 0, 1);
94        motorC =0;
        }
96
    }
98
    // Hacemos una pausa para comenzar a realizar las mediciones
100 systick_wait_ms(500);
102 // Obtenemos el valor de la distancia en la dirección derecha
    for(i=0; i<10;i++){
104   media += ecrobot_get_sonar_sensor(SENSOR_ULTRASONIDOS);
      systick_wait_ms(100);
106 }
    distanciaDcha = media/10;
108
110
112 /* Comprobamos ahora la distancia a la izquierda */
    nxt_motor_set_count(NXT_PORT_B, 0);
114 nxt_motor_set_count(NXT_PORT_C, 0);
116 // Activamos motores para girar a la izq
    nxt_motor_set_speed(NXT_PORT_B, 40, 1);
118 nxt_motor_set_speed(NXT_PORT_C, -40, 1);
120 motorB = motorC = 1;
    while (motorB || motorC){
122   if(nxt_motor_get_count(NXT_PORT_B) >= 360){
        nxt_motor_set_speed(NXT_PORT_B, 0, 1);
124     motorB=0;
      }
126
      if(nxt_motor_get_count(NXT_PORT_C) <= -360){
128     nxt_motor_set_speed(NXT_PORT_C, 0, 1);
        motorC=0;
130   }
    }
132
134 // Obtener el valor de la distancia en la dirección izquierda
136 media = 0;
    for(i =0; i<10;i++){
138   media += ecrobot_get_sonar_sensor(SENSOR_ULTRASONIDOS);
      systick_wait_ms(100);
140 }
    distanciaIzq = media/10;
142
144 /* Si distancia derecha es mayor que la izquierda:
        - Volvemos a girar el robot hacia la derecha
146     En caso contrario:
        - El robot no se gira porque la dirección a tomar es la izquierda
```

```
148     */
    if(distanciaDcha > distanciaIzq){
150     // Inicializamos contadores
      nxt_motor_set_count(NXT_PORT_B,0);
152     nxt_motor_set_count(NXT_PORT_C, 0);

154     // Activamos motores para girar a la dcha
      nxt_motor_set_speed(NXT_PORT_B, -40, 1);
156     nxt_motor_set_speed(NXT_PORT_C, 40, 1);

158     motorB = 1, motorC = 1;
      while (motorB || motorC){
160       if(nxt_motor_get_count(NXT_PORT_B) <= -360){
          nxt_motor_set_speed(NXT_PORT_B, 0, 1);
162         motorB =0;
        }
164
        if(nxt_motor_get_count(NXT_PORT_C) >= 360){
166       nxt_motor_set_speed(NXT_PORT_C, 0, 1);
          motorC =0;
168     }
      }
170   }

172   // Terminar la tarea
    TerminateTask();
174 }

176
/*-----------------------------------------------------------------------*/
178 // Task information:
// ------------------
180 // Name    : Principal
// Priority: 1
182 // Typ     : EXTENDED TASK
// Schedule: FULL
184 // Objective: Avanzar hasta que se produzca algun choque.
/*-----------------------------------------------------------------------*/
186 TASK (Principal)
{
188   //int pulsador_choque;

190   systick_wait_ms(10000);


192
    ecrobot_get_sonar_sensor(SENSOR_ULTRASONIDOS);
194
     /* La tarea Principal se ejecutará mientras la bandera de fin esté
        desactivada (igual a 0) */
196
    do{
198     // Avanza hasta que se toque alguno de los dos pulsadores delanteros
          nxt_motor_set_speed(NXT_PORT_B, OriginalSpeedB, 1);
200     nxt_motor_set_speed(NXT_PORT_C, OriginalSpeedC, 1);

202     nxt_motor_set_count(NXT_PORT_B,0);
      nxt_motor_set_count(NXT_PORT_C,0);
204
```

```
    do{
      if(pulsador_choque){
        nxt_motor_set_speed(NXT_PORT_B, 0, 1);
        nxt_motor_set_speed(NXT_PORT_C, 0, 1);
        break;
      }

      int rpmB = nxt_motor_get_count(NXT_PORT_B);
      int rpmC = nxt_motor_get_count(NXT_PORT_C);

      if(rpmB < rpmC){
        if(speedB<OriginalSpeedB)
          speedB++;
        else
          speedC--;
        nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
        nxt_motor_set_speed(NXT_PORT_C, speedC, 1);
      } else if(rpmC < rpmB){
        if(speedC<OriginalSpeedC)
          speedC++;
        else
          speedB--;
        nxt_motor_set_speed(NXT_PORT_C, speedC, 1);
        nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
      }

      if(pulsador_choque){
        nxt_motor_set_speed(NXT_PORT_B, 0, 1);
        nxt_motor_set_speed(NXT_PORT_C, 0, 1);
        break;
      }

      systick_wait_ms(300);

    }while(pulsador_choque!=1 && fin !=1);

     if(fin != 1){
      nxt_motor_set_speed(NXT_PORT_B, 0, 1);
      nxt_motor_set_speed(NXT_PORT_C, 0, 1);

      // Va hacia atrás un poco para empezar de nuevo el reconocimiento
      nxt_motor_set_speed(NXT_PORT_B, -50, 1);
      nxt_motor_set_speed(NXT_PORT_C, -50, 1);
      systick_wait_ms(600);

      ActivateTask(Giro);
    }

  }while(fin!=1);


  // Paramos los motores antes de finalizar la tarea
  nxt_motor_set_speed(NXT_PORT_B, 0, 1);
  nxt_motor_set_speed(NXT_PORT_C, 0, 1);

  // Terminar la tarea actual
  TerminateTask();
}
```

```
264 /*-------------------------------------------------------------------------*/
    // Task information:
266 // -----------------
    // Name     : Pulsador
268 // Priority: 3
    // Typ      : EXTENDED TASK
270 // Schedule: FULL
    // Objective: Comprobar si se ha pulsado el sensor de fin de laberinto
272 /*-------------------------------------------------------------------------*/
    TASK (Pulsador){
274   int pulsador_fin;

276   // Obtenemos el valor del sensor de pulsación para ver si ha sido activado
      pulsador_fin = ecrobot_get_touch_sensor(PULSADOR_FIN);
278   pulsador_choque = ecrobot_get_touch_sensor(PULSADOR_CHOQUE);


280
      /* Si ha sido activado:
282            - Paramos los motores
               - Establecemos la bandera de fin a 1 para indicar que el roboto
                  debe finalizar
284       Si no ha sido activado:
               - El programa continua normalmente
286    */
      if(pulsador_fin == 1){
288     fin = 1;
      }
290
      // Terminar la tarea actual
292   TerminateTask();
    }
```

3.2.5. Código del fichero practica2_a.oil

```
1  /*-------------------------------------------------------------------------
     Practica: 2.a
3
     Autores:
5         - Francisco Arjona Lopez
          - Cristobal Castro Villegas
7         - Jose Antonio Espino Palomares
          - Antonio Osuna Caballero
9  -------------------------------------------------------------------------*/
   #include "implementation.oil"
11
   CPU ATMEL_AT91SAM7S256
13 {
     OS LEJOS_OSEK
15     {
         STATUS = EXTENDED;
17       STARTUPHOOK = FALSE;
         ERRORHOOK = FALSE;
19       SHUTDOWNHOOK = FALSE;
         PRETASKHOOK = FALSE;
21       POSTTASKHOOK = FALSE;
         USEGETSERVICEID = FALSE;
```

```
23      USEPARAMETERACCESS = FALSE;
        USERESSCHEDULER = FALSE;
25    };

27  APPMODE sample_appmode1{};

29  COUNTER Contador
      {
31    MINCYCLE  = 1;
      MAXALLOWEDVALUE = 6000;
33    TICKSPERBASE  = 1;
      };
35
    //Viene lo que es en la pagina 5 del manual de referencia
37  ALARM alarm1
    {
39    COUNTER = Contador;
      ACTION = ACTIVATETASK {
41      TASK = Pulsador;
      };
43
      AUTOSTART = TRUE {
45      ALARMTIME = 100; //instante en el que se inicia la alarma
        CYCLETIME = 100; //cada 100 unidades del contador se activa
47      APPMODE = sample_appmode1;
      };
49  };

51  TASK Principal
      {
53      AUTOSTART = TRUE
      {
55        APPMODE = sample_appmode1;
        };
57
        PRIORITY  = 1;
59      ACTIVATION  = 1;
        SCHEDULE  = FULL;
61      STACKSIZE   = 512;
      };
63


65   TASK Giro
     {
67      AUTOSTART   = FALSE;
        PRIORITY  = 2;
69      ACTIVATION  = 1;
        SCHEDULE  = FULL;
71      STACKSIZE   = 512;
     };
73
      TASK Pulsador
75    {
        AUTOSTART   = FALSE;
77      PRIORITY  = 3;
        ACTIVATION  = 1;
79      SCHEDULE  = NON;
        STACKSIZE   = 512;
```

```
81      };

83  };
```

3.2.6. Resultados

3.3. Ejercicio B

Programar un segundo robot sin sensores (se podrá utilizar el robot básico de algún compañero), de tal forma que el robot con sensores le envíe mediante comunicaciones bluetooth las órdenes necesarias para que el robot sin sensores sea capaz de realizar el recorrido de salida del laberinto. Se recomienda que el robot con sensores almacene todos los datos necesarios y los envíe mediante un modelo cliente-servidor. A su vez, el robot sin sensores debería ejecutar una tras otra las acciones que le ha enviado el primer robot, con la cadencia de tiempo correcta (en función de la velocidad del robot).

3.3.1. Estrategia

Este ejercicio está basado en el ejercicio anterior, sólo que para la solución ha sido necesario añadir las primitivas de *Bluetooth* para poder conectarse a otro robot y enviar la información.

En nuestro caso se ha pensado que la mejor solución sería que el robot servidor almacene sus movimientos en un vector que posteriormente mandará al robot cliente y realice los mismo movimientos para salir del laberinto.

En este caso no comentaros cada una de las tareas y alarmas utilizadas, porque son las mismas que en el ejercicio anterior.

3.3.2. Código del fichero practica_2b_cliente.c

```
/*-----------------------------------------------------------------------
2   Practica: 2.b - Cliente

4   Autores:
        - Francisco Arjona Lopez
6       - Cristobal Castro Villegas
        - Jose Antonio Espino Palomares
8       - Antonio Osuna Caballero
-----------------------------------------------------------------------*/
10
#include <string.h>
12 #include "kernel.h"
#include "kernel_id.h"
14 #include "ecrobot_interface.h"

16 #define MAX_MOV 200

18 /*-----------------------------------------------------------------------*/
/* OSEK declarations                                                     */
20 /*-----------------------------------------------------------------------*/
```

```
DeclareTask(Principal);
DeclareTask(Avance);
DeclareTask(Retroceso);
DeclareTask(GiroIzq);
DeclareTask(GiroDer);
DeclareTask(Final);


void ecrobot_device_initialize()
{
  // Inicializamos el bluetooth
   ecrobot_init_bt_slave("STR");
}
void ecrobot_device_terminate()
{
  // Finalizamos el bluetooth
   ecrobot_term_bt_connection();
}

/*-----------------------------------------------------------------------*/
/* Function to be invoked from a category 2 interrupt                    */
/*-----------------------------------------------------------------------*/
void user_1ms_isr_type2() { }


/*-----------------------------------------------------------------------*/
/* Definitions                                                           */
/*-----------------------------------------------------------------------*/
int i = 0;
int movimientos[MAX_MOV];

int OriginalSpeedB = 50, speedB = 50;
int OriginalSpeedC = 47, speedC = 47;

TASK(Principal)
{
  int n=0;

  systick_wait_ms(10000);
  systick_wait_ms(3000);

  nxt_motor_set_speed(NXT_PORT_B, 50, 1);
  nxt_motor_set_speed(NXT_PORT_C, 50, 1);

  systick_wait_ms(1100);

  nxt_motor_set_speed(NXT_PORT_B, 0, 1);
  nxt_motor_set_speed(NXT_PORT_C, 0, 1);


  while(n == 0)
    ecrobot_read_bt_packet(&n, sizeof(int));

  while(i!=n && i < MAX_MOV){
    movimientos[i]=0;
    while(movimientos[i] == 0)
      ecrobot_read_bt_packet(&movimientos[i], sizeof(int));
    i++;
```

```
  }

  // Ejecutamos cada uno de los movimientos
  int parada = 0;
  for(i=0; i < n && !parada; i++)
  {
      if(movimientos[i] == (unsigned char) 1)
         // Giro hacia la izquierda
         ActivateTask(GiroIzq);
      else if(movimientos[i] == (unsigned char) 2)
         // Giro hacia la derecha
         ActivateTask(GiroDer);
      else if(movimientos[i] == (unsigned char) -1)
         // Fin de los movimientos
         parada = 1;
      else if(movimientos[i] > (unsigned char) 0)
         // Avance del robot
         ActivateTask(Avance);
  }

   // Activamos la tarea final
   ActivateTask(Final);

   // Terminar la tarea actual
   TerminateTask();
}

TASK(Avance)
{
  // Inicializamos contadores
  nxt_motor_set_count(NXT_PORT_B,0);
  nxt_motor_set_count(NXT_PORT_C,0);

  // Activamos los motores
  nxt_motor_set_speed(NXT_PORT_B, OriginalSpeedB, 1);
  nxt_motor_set_speed(NXT_PORT_C, OriginalSpeedC, 1);


   int motorB = 1, motorC = 1;
  while(motorB || motorC){
    systick_wait_ms(100);

    display_clear(1);
    display_goto_xy(0, 4);
    display_int(nxt_motor_get_count(NXT_PORT_B), 0);
    display_update();

    display_goto_xy(0, 5);
    display_int(nxt_motor_get_count(NXT_PORT_C), 0);
    display_update();

    display_goto_xy(0, 1);
    display_int(i, 0);
    display_update();

    if(nxt_motor_get_count(NXT_PORT_B)>= movimientos[i]){
      nxt_motor_set_speed(NXT_PORT_B, 0, 1);
      motorB = 0;
```

```
    }

    if(nxt_motor_get_count(NXT_PORT_C)>=movimientos[i]){
      nxt_motor_set_speed(NXT_PORT_C, 0, 1);
      motorC = 0;
    }


    if(motorB && motorC){
      int rpmB = nxt_motor_get_count(NXT_PORT_B);
      int rpmC = nxt_motor_get_count(NXT_PORT_C);

      if(rpmB < rpmC){
        if(speedB<OriginalSpeedB+5)
          speedB++;
        else
          speedC--;
        nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
        nxt_motor_set_speed(NXT_PORT_C, speedC, 1);
      }

      if(rpmC < rpmB){
        if(speedC<OriginalSpeedC+5)
          speedC++;
        else
          speedB--;
        nxt_motor_set_speed(NXT_PORT_C, speedC, 1);
        nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
      }
    }
  }

  // Paramos los motores
  nxt_motor_set_speed(NXT_PORT_B, 0, 1);
  nxt_motor_set_speed(NXT_PORT_C, 0, 1);

  //i++;
  systick_wait_ms(500);

  // Terminar la tarea actual
  TerminateTask();
}

TASK(GiroDer)
{
  // Inicializamos contadores
  nxt_motor_set_count(NXT_PORT_B,0);
  nxt_motor_set_count(NXT_PORT_C,0);

  // Activamos motores para girar a la dcha
  nxt_motor_set_speed(NXT_PORT_B, -40, 1);
  nxt_motor_set_speed(NXT_PORT_C, 40, 1);

  int motorB = 1, motorC = 1;
  while (motorB || motorC){
    if(nxt_motor_get_count(NXT_PORT_B) <= -180){
      nxt_motor_set_speed(NXT_PORT_B, 0, 1);
      motorB=0;
```

```
    }
196
    if(nxt_motor_get_count(NXT_PORT_C) >= 180){
198   nxt_motor_set_speed(NXT_PORT_C, 0, 1);
      motorC=0;
200 }
  }
202
  systick_wait_ms(500);
204
  // Terminar la tarea actual
206 TerminateTask();
}
208
TASK(GiroIzq)
210 {
  // Inicializamos contadores
212 nxt_motor_set_count(NXT_PORT_B,0);
  nxt_motor_set_count(NXT_PORT_C,0);
214
  // Activamos motores para girar a la dcha
216 nxt_motor_set_speed(NXT_PORT_B, 40, 1);
  nxt_motor_set_speed(NXT_PORT_C, -40, 1);
218
  int motorB = 1, motorC = 1;
220 while (motorB || motorC){
    if(nxt_motor_get_count(NXT_PORT_B) >= 180){
222   nxt_motor_set_speed(NXT_PORT_B, 0, 1);
      motorB=0;
224 }

226 if(nxt_motor_get_count(NXT_PORT_C) <= -180){
      nxt_motor_set_speed(NXT_PORT_C, 0, 1);
228   motorC=0;
    }
230 }

232 systick_wait_ms(500);
  // Terminar la tarea actual
234 TerminateTask();
}
236
TASK(Final)
238 {
   // Apagamos el sistema
240 ShutdownOS(0);

242 // Terminar la tarea actual
   TerminateTask();
244 }
```

3.3.3. Código del fichero practica_2b_cliente.oil

```
/*-----------------------------------------------------------------
2 Practica: 2.b - cliente

4 Autores:
```

```
        - Francisco Arjona Lopez
        - Cristobal Castro Villegas
        - Jose Antonio Espino Palomares
        - Antonio Osuna Caballero
-------------------------------------------------------------------------------*/

#include "implementation.oil"

CPU ATMEL_AT91SAM7S256
{
  OS LEJOS_OSEK
    {
      STATUS = EXTENDED;
      STARTUPHOOK = FALSE;
      ERRORHOOK = FALSE;
      SHUTDOWNHOOK = FALSE;
      PRETASKHOOK = FALSE;
      POSTTASKHOOK = FALSE;
      USEGETSERVICEID = FALSE;
      USEPARAMETERACCESS = FALSE;
      USERESSCHEDULER = FALSE;
    };

  APPMODE sample_appmode1{};

  TASK Principal
  {
      AUTOSTART = TRUE
    {
        APPMODE = sample_appmode1;
      };
      PRIORITY  = 1;
      ACTIVATION  = 1;
      SCHEDULE  = FULL;
      STACKSIZE   = 512;
  };

  TASK Avance
  {
      AUTOSTART   = FALSE;
      PRIORITY  = 6;
      ACTIVATION  = 1;
      SCHEDULE  = FULL;
      STACKSIZE   = 512;
  };

  TASK GiroIzq
  {
      AUTOSTART = FALSE;
      PRIORITY  = 6;
      ACTIVATION  = 1;
      SCHEDULE  = NON;
      STACKSIZE   = 512;
  };

  TASK GiroDer
  {
      AUTOSTART = FALSE;
```

```
        PRIORITY   = 6;
64      ACTIVATION   = 1;
        SCHEDULE   = NON;
66      STACKSIZE    = 512;
    };
68
    TASK Final
70  {
        AUTOSTART = FALSE;
72      PRIORITY   = 8;
        ACTIVATION   = 1;
74      SCHEDULE   = NON;
        STACKSIZE    = 512;
76  };
};
```

3.3.4. Código del fichero practica_2b_servidor.c

```
1  /*-----------------------------------------------------------------------
     Practica: 2.b - Servidor
3
     Autores:
5         - Francisco Arjona Lopez
          - Cristobal Castro Villegas
7         - Jose Antonio Espino Palomares
          - Antonio Osuna Caballero
9  -----------------------------------------------------------------------*/
   #include "kernel.h"
11 #include "kernel_id.h"
   #include "ecrobot_interface.h"
13
   #define PULSADOR_CHOQUE NXT_PORT_S1
15 #define PULSADOR_FIN NXT_PORT_S2
   #define SENSOR_ULTRASONIDOS NXT_PORT_S4
17
   #define MAX_MOV 200
19
21 /*-----------------------------------------------------------------------*/
   /* OSEK declarations                                                     */
23 /*-----------------------------------------------------------------------*/
   DeclareTask(Giro);
25 DeclareTask(Principal);
   DeclareTask(Pulsador);
27 DeclareCounter(Contador);
   DeclareAlarm(alarm1);
29
31 /*-----------------------------------------------------------------------*/
   /* Definitions                                                           */
33 /*-----------------------------------------------------------------------*/

35 //direccion del cliente
   const U8 bd_addr[7] = {0x00,0x16,0x53,0x01,0x82,0xD6,0x00};
37 int movimientos[MAX_MOV];
   int iterador = 0;
39
```

```
int fin = 0; //indica el fin de la ejecucion
int pulsador_choque=0;

int OriginalSpeedB = 50, speedB = 50;
int OriginalSpeedC = 50, speedC = 50;


void ecrobot_device_initialize()
{
  // Inicializar los sensores activos
  ecrobot_init_sonar_sensor(SENSOR_ULTRASONIDOS);

  ecrobot_init_bt_master(bd_addr,"STR"); //inicializa el esclavo
}

void ecrobot_device_terminate()
{
  // Finalizar los sensores activos
  ecrobot_term_sonar_sensor(SENSOR_ULTRASONIDOS);

  ecrobot_term_bt_connection(); //para la ejecucion del bluetooth
}


/*-----------------------------------------------------------------------*/
/* Function to be invoked from a category 2 interrupt                    */
/*-----------------------------------------------------------------------*/
void user_1ms_isr_type2(){

  SignalCounter(Contador); //pone en marcha el contador
}

/*-----------------------------------------------------------------------*/
// Task information:
// -----------------
// Name    : Giro
// Priority: 2
// Typ     : EXTENDED TASK
// Schedule: FULL
// Objective: Girar a la derecha y a la izquierda para comprobar la
//        distancia y elegir la mejor opcion
/*-----------------------------------------------------------------------*/
TASK(Giro)
{

  //Inicializamos para medir la distancia a uno y otro lado
  int distanciaDcha = 0, distanciaIzq = 0, media=0;
  int motorB = 1, motorC = 1;

  int i;

  /* Comprobamos que distancia hay al girar a la derecha */

  // Inicializamos contadores
  nxt_motor_set_count(NXT_PORT_B,0);
  nxt_motor_set_count(NXT_PORT_C,0);

  // Activamos motores para girar a la dcha
```

```
  nxt_motor_set_speed(NXT_PORT_B, -40, 1);
99  nxt_motor_set_speed(NXT_PORT_C, 40, 1);

101 while (motorB || motorC){
      if(nxt_motor_get_count(NXT_PORT_B) <= -180){
103     nxt_motor_set_speed(NXT_PORT_B, 0, 1);
        motorB =0;
105   }

107   if(nxt_motor_get_count(NXT_PORT_C) >= 180){
        nxt_motor_set_speed(NXT_PORT_C, 0, 1);
109     motorC =0;
      }
111
    }
113
    // Hacemos una pausa para comenzar a realizar las mediciones
115 systick_wait_ms(500);

117 // Obtenemos el valor de la distancia en la direccion derecha

119 for(i=0; i<10;i++){
      media += ecrobot_get_sonar_sensor(SENSOR_ULTRASONIDOS);
121   systick_wait_ms(100);
    }
123 distanciaDcha = media/10;

125



127
    /* Comprobamos ahora la distancia a la izquierda */
129
    nxt_motor_set_count(NXT_PORT_B, 0);
131 nxt_motor_set_count(NXT_PORT_C, 0);

133 // Activamos motores para girar a la izq
    nxt_motor_set_speed(NXT_PORT_B, 40, 1);
135 nxt_motor_set_speed(NXT_PORT_C, -40, 1);

137 motorB = motorC = 1;
    while (motorB || motorC){
139   if(nxt_motor_get_count(NXT_PORT_B) >= 360){
        nxt_motor_set_speed(NXT_PORT_B, 0, 1);
141     motorB=0;
      }
143
      if(nxt_motor_get_count(NXT_PORT_C) <= -360){
145     nxt_motor_set_speed(NXT_PORT_C, 0, 1);
          motorC=0;
147   }
    }
149


151 // Obtener el valor de la distancia en la direccion izquierda

153 media = 0;
    for(i =0; i<10;i++){
155   media += ecrobot_get_sonar_sensor(SENSOR_ULTRASONIDOS);
```

```
    systick_wait_ms(100);
157 }
  distanciaIzq = media/10;
159

161 /* Si distancia derecha es mayor que la izquierda:
      - Volvemos a girar el robot hacia la derecha
163    En caso contrario:
      - El robot no se gira porque la direccion a tomar es la izquierda
165  */

167

  if(distanciaDcha > distanciaIzq){
169   // Inicializamos contadores
    nxt_motor_set_count(NXT_PORT_B,0);
171   nxt_motor_set_count(NXT_PORT_C, 0);

173   // Activamos motores para girar a la dcha
    nxt_motor_set_speed(NXT_PORT_B, -40, 1);
175   nxt_motor_set_speed(NXT_PORT_C, 40, 1);

177   motorB = 1, motorC = 1;
    while (motorB || motorC){
179     if(nxt_motor_get_count(NXT_PORT_B) <= -360){
        nxt_motor_set_speed(NXT_PORT_B, 0, 1);
181       motorB =0;
      }
183

      if(nxt_motor_get_count(NXT_PORT_C) >= 360){
185       nxt_motor_set_speed(NXT_PORT_C, 0, 1);
        motorC =0;
187     }
    }
189

    //Si al final tiene que ir a la derecha almacenamos el valor
191   movimientos[iterador] = 2;
    iterador++;
193 } else{
    //En caso contrario almacenamos que gire a la izq
195   movimientos[iterador] = 1;
    iterador++;
197 }

199 // Terminamos la tarea
  TerminateTask();
201 }

203

/*-----------------------------------------------------------------------*/
205 // Task information:
// ------------------
207 // Name    : Principal
// Priority: 1
209 // Typ     : EXTENDED TASK
// Schedule: FULL
211 // Objective: Avanzar hasta que se produzca algun choque.
/*-----------------------------------------------------------------------*/
213 TASK (Principal)
```

```
{
  int pulsador;

  systick_wait_ms(10000);


  ecrobot_get_sonar_sensor(SENSOR_ULTRASONIDOS);

   /* La tarea Principal se ejecutara mientras la bandera de fin este
      desactivada (igual a 0) */

  do{
    // Avanza hasta que se toque alguno de los dos pulsadores delanteros
       nxt_motor_set_speed(NXT_PORT_B, OriginalSpeedB, 1);
    nxt_motor_set_speed(NXT_PORT_C, OriginalSpeedC, 1);

    nxt_motor_set_count(NXT_PORT_B,0);
    nxt_motor_set_count(NXT_PORT_C,0);

    do{
      if(pulsador_choque){
        nxt_motor_set_speed(NXT_PORT_B, 0, 1);
        nxt_motor_set_speed(NXT_PORT_C, 0, 1);
        break;
      }

      int rpmB = nxt_motor_get_count(NXT_PORT_B);
      int rpmC = nxt_motor_get_count(NXT_PORT_C);


      if(rpmB < rpmC){
        if(speedB<OriginalSpeedB)
          speedB++;
        else
          speedC--;
        nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
        nxt_motor_set_speed(NXT_PORT_C, speedC, 1);
      } else if(rpmC < rpmB){
        if(speedC<OriginalSpeedC)
          speedC++;
        else
          speedB--;
        nxt_motor_set_speed(NXT_PORT_C, speedC, 1);
        nxt_motor_set_speed(NXT_PORT_B, speedB, 1);
      }

      if(pulsador_choque){
        nxt_motor_set_speed(NXT_PORT_B, 0, 1);
        nxt_motor_set_speed(NXT_PORT_C, 0, 1);
        break;
      }

      systick_wait_ms(300);

    }while(pulsador_choque!=1 && fin !=1);

     if(fin != 1){
      nxt_motor_set_speed(NXT_PORT_B, 0, 1);
```

```
271         nxt_motor_set_speed(NXT_PORT_C, 0, 1);

273         // Va hacia atras un poco para empezar de nuevo el reconocimiento
            nxt_motor_set_speed(NXT_PORT_B, -50, 1);
275         nxt_motor_set_speed(NXT_PORT_C, -50, 1);
            systick_wait_ms(600);
277
            movimientos[iterador] = (nxt_motor_get_count(NXT_PORT_B) +
                nxt_motor_get_count(NXT_PORT_C))/2;
279         iterador++;

281         ActivateTask(Giro);
          }
283
        }while(fin!=1);
285
        movimientos[iterador] = (nxt_motor_get_count(NXT_PORT_B) +
            nxt_motor_get_count(NXT_PORT_C))/2;
287     iterador++;

289     // Paramos los motores antes de finalizar la tarea
        nxt_motor_set_speed(NXT_PORT_B, 0, 1);
291     nxt_motor_set_speed(NXT_PORT_C, 0, 1);

293     movimientos[iterador] = -1; //FIN
        iterador++;
295
        ecrobot_send_bt_packet(&iterador, sizeof(int));
297     systick_wait_ms(100);

299     for(int i = 0;i<iterador;i++){
          ecrobot_send_bt_packet(&movimientos[i], sizeof(int));
301       systick_wait_ms(100);
        }
303


305     // Apagamos el sistema
        ShutdownOS(0);
307
        // Terminar la tarea actual
309     TerminateTask();
    }
311


313 /*-----------------------------------------------------------------------*/
    // Task information:
315 // ------------------
    // Name     : Principal
317 // Priority: 1
    // Typ      : EXTENDED TASK
319 // Schedule: FULL
    // Objective: Avanzar hasta que se produzca algun choque.
321 /*-----------------------------------------------------------------------*/
    TASK (Pulsador){
323   int pulsador_fin;

325   // Obtenemos el valor del sensor de pulsacion para ver si ha sido activado
      pulsador_fin = ecrobot_get_touch_sensor(PULSADOR_FIN);
```

```
327   pulsador_choque = ecrobot_get_touch_sensor(PULSADOR_CHOQUE);

329
   /* Si ha sido activado:
331          - Paramos los motores
             - Establecemos la bandera de fin a 1 para indicar que el roboto
                debe finalizar
333      Si no ha sido activado:
             - El programa continua normalmente
335    */
   if(pulsador_fin == 1){
337    fin = 1;
   }
339
   // Terminar la tarea actual
341 TerminateTask();
}
```

3.3.5. Código del fichero practica_2b_servidor.oil

```
1  /*-----------------------------------------------------------------------
   Practica: 1.b - servidor
3
   Autores:
5       - Francisco Arjona Lopez
        - Cristobal Castro Villegas
7       - Jose Antonio Espino Palomares
        - Antonio Osuna Caballero
9  -----------------------------------------------------------------------*/
#include "implementation.oil"
11
CPU ATMEL_AT91SAM7S256
13 {
   OS LEJOS_OSEK
15    {
        STATUS = EXTENDED;
17      STARTUPHOOK = FALSE;
        ERRORHOOK = FALSE;
19      SHUTDOWNHOOK = FALSE;
        PRETASKHOOK = FALSE;
21      POSTTASKHOOK = FALSE;
        USEGETSERVICEID = FALSE;
23      USEPARAMETERACCESS = FALSE;
        USERESSCHEDULER = FALSE;
25    };

27    APPMODE sample_appmode1{};

29 COUNTER Contador
      {
31    MINCYCLE  = 1;
      MAXALLOWEDVALUE = 6000;
33    TICKSPERBASE  = 1;
      };
35
   ALARM alarm1
37 {
```

```
    COUNTER = Contador;
    ACTION = ACTIVATETASK {
      TASK = Pulsador;
    };

    AUTOSTART = TRUE {
      ALARMTIME = 100; //instante en el que se inicia la alarma
      CYCLETIME = 100; //cada 200 unidades del contador se activa
      APPMODE = sample_appmode1;
    };
  };

  TASK Principal
    {
      AUTOSTART = TRUE
    {
        APPMODE = sample_appmode1;
      };

      PRIORITY  = 1;
      ACTIVATION  = 1;
      SCHEDULE  = FULL;
      STACKSIZE   = 512;
    };


    TASK Giro
    {
      AUTOSTART   = FALSE;
      PRIORITY  = 2;
      ACTIVATION  = 1;
      SCHEDULE  = FULL;
      STACKSIZE   = 512;
    };

    TASK Pulsador
    {
      AUTOSTART   = FALSE;
      PRIORITY  = 3;
      ACTIVATION  = 1;
      SCHEDULE  = NON;
      STACKSIZE   = 512;
    };

};
```

3.3.6. Resultados

En el siguiente vídeo podemos observar el resultado del ejercicio de esta práctica.

Vídeo: `http://www.youtube.com/watch?v=_GzLX9yPcCM`

Capítulo 4

Práctica 3. Manejo avanzado de Lego Mindstorms NXT

4.1. Objetivos

Tras la realización de esta práctica el alumno debería ser capaz de:

- Acceder a sensores no estándar de manera efectiva.
- Determinar un acceso eficiente a los sensores y a los actuadores.
- Planificar las tareas siguiendo un análisis teórico.
- Realizar programación concurrente con memoria compartida.

4.2. Ejercicio A

Construir un robot que se apoye en el suelo únicamente sobre 2 ruedas y programarlo para que se mantenga estable en posición vertical. Para determinar si el robot se mantiene vertical, se le podrá incorporar diferentes tipos de sensores, por ejemplo:

- Sensor de giro (giróscopo).
- Sensor de aceleración (acelerómetro).
- Dos sensores de pulsación (uno delante y otro detrás del robot, para determinar si el robot se inclina más de un cierto ángulo hacia adelante o hacia atrás).
- Sensor de iluminación (el sensor emite luz que es leída por el propio sensor pudiendo determinar la distancia en función de la iluminación recibida. Ojo, este mecanismo es muy sensible a los cambios de iluminación ambiental y a la superficie en la que se ponga el robot).
- Cualquier otro sensor que esté disponible en el laboratorio.

El robot accionará los motores de las ruedas para contrarrestar la caída del robot hacia adelante o hacia atrás.

4.2.1. Estrategia

Para realización de este apartado, ha sido necesario modificar el robot, el resultado de esta modificación se mostrará en el vídeo en el apartado de resultados. También ha sido necesario añadir un sensor de luz, para controlar que el robot se mantenga en equilibrio.

La idea para solucionarlo ha sido definir dos niveles, los cuales en el momento que se superan el robot avanza o retrocede para mantenerse en equilibrio.

4.2.2. Tareas

A continuación se realiza una descripción de las tareas utilizadas para la resolución de este apartado y su prioridad.

- **Principal**
  - *Prioridad*: 1
  - *Descripción*: Se encarga de activar la tarea *equilibrio*.
- **Equilibrio**
  - *Prioridad*: 2
  - *Descripción*: Se encarga mediante el sensor de luz de comprobar que se encuentra en equilibrio el robot, para ello define dos niveles:
    1. Nivel 1: Si el nivel del robot es inferior a 640.
       - En el caso de que el nivel sea inferior a 620, el robot se moverá las ruedas hacia atrás a mayor velocidad
       - En caso contrario, el robot moverá las ruedas hacia atrás a una velocidad normal.
    2. Nivel 2: Si el nivel del robot es superior a 655.
       - En el caso de que el nivel sea superior a 685, el robot se moverá las ruedas hacia adelante a mayor velocidad
       - En caso contrario, el robot moverá las ruedas hacia adelante a una velocidad normal.

En la siguiente Figura 4.1, se muestra un gráfico de su funcionamiento.

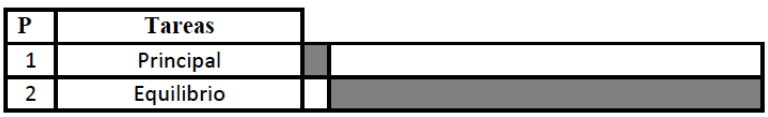

Figura 4.1: Funcionamiento práctica 3a.

4.2.3. Código del fichero practica3_a.c

```
/*-----------------------------------------------------------------------------
  Practica: 3.a

  Autores:
       - Francisco Arjona Lopez
       - Cristobal Castro Villegas
       - Jose Antonio Espino Palomares
       - Antonio Osuna Caballero
-----------------------------------------------------------------------------*/
#include "kernel.h"
#include "kernel_id.h"
#include "ecrobot_interface.h"

/*-----------------------------------------------------------------------------*/
/* OSEK declarations                                                           */
/*-----------------------------------------------------------------------------*/
DeclareTask(Principal);
DeclareTask(Equilibrio);


void ecrobot_device_initialize()
{
  // Inicializar los sensores activos
  ecrobot_set_light_sensor_active(NXT_PORT_S1);
}
void ecrobot_device_terminate()
{
  // Finalizar los sensores activos
  ecrobot_set_light_sensor_inactive(NXT_PORT_S1);
}


/*-----------------------------------------------------------------------------*/
/* Definitions                                                                 */
/*-----------------------------------------------------------------------------*/
int parada = 0;

/*-----------------------------------------------------------------------------*/
/* Function to be invoked from a category 2 interrupt                          */
/*-----------------------------------------------------------------------------*/
void user_1ms_isr_type2(){}

/*-----------------------------------------------------------------------------*/
// Task information:
// -----------------
// Name     : Equilibrio
// Priority: 2
// Typ      : EXTENDED TASK
// Schedule: FULL
// Objective: Se encarga de mantener el robot en equilibrio
/*-----------------------------------------------------------------------------*/
TASK(Equilibrio)
{

  nxt_motor_set_count(NXT_PORT_C,0);
  nxt_motor_set_count(NXT_PORT_B,0);

```

```
  while(1)
    {
      int nivel;

      nivel=ecrobot_get_light_sensor(NXT_PORT_S1);

      //Mostraremos por pantalla el valor del sensor
      display_clear(0);
      display_goto_xy(0,3);
      display_int(nivel,1);
      display_update();

      //Este nivel ha sido elegido segun las condiciones dadas durante la
          realizacion de la practica.
      if(nivel <640)
      {
      if(nivel<620){
        nxt_motor_set_speed(NXT_PORT_B, -77, 0);
        nxt_motor_set_speed(NXT_PORT_C, -77, 0);
      }
      else{
      nxt_motor_set_speed(NXT_PORT_B, -68, 0);
      nxt_motor_set_speed(NXT_PORT_C, -68, 0);
      }
      }
      else if(nivel >655)
      {
      if(nivel > 685){
        nxt_motor_set_speed(NXT_PORT_B, 75, 0);
        nxt_motor_set_speed(NXT_PORT_C, 75, 0);
      }
      else{
        nxt_motor_set_speed(NXT_PORT_B, 65, 0);
        nxt_motor_set_speed(NXT_PORT_C, 65, 0);
      }

      }
      else
      {
        nxt_motor_set_speed(NXT_PORT_B, 0, 0);
        nxt_motor_set_speed(NXT_PORT_C, 0, 0);
    }



    }
    // Terminar la tarea actual
    TerminateTask();
}

/*--------------------------------------------------------------------------*/
// Task information:
// -----------------
// Name     : Principal
// Priority: 1
// Typ      : EXTENDED TASK
// Schedule: FULL
// Objective: Activa la tarea equilibrio
```

```
115 /*-----------------------------------------------------------------------*/
    TASK(Principal)
117 {
      //
119   ActivateTask(Equilibrio);

121   // Terminar la tarea actual
      TerminateTask();
123 }
```

4.2.4. Código del fichero practica3_oil.c

```
1  /*-----------------------------------------------------------------------
     Practica: 3.a
3
     Autores:
5         - Francisco Arjona Lopez
          - Cristobal Castro Villegas
7         - Jose Antonio Espino Palomares
          - Antonio Osuna Caballero
9  -----------------------------------------------------------------------*/

11 #include "implementation.oil"

13 CPU ATMEL_AT91SAM7S256
   {
15   OS LEJOS_OSEK
       {
17       STATUS = EXTENDED;
         STARTUPHOOK = FALSE;
19       ERRORHOOK = FALSE;
         SHUTDOWNHOOK = FALSE;
21       PRETASKHOOK = FALSE;
         POSTTASKHOOK = FALSE;
23       USEGETSERVICEID = FALSE;
         USEPARAMETERACCESS = FALSE;
25       USERESSCHEDULER = FALSE;
       };
27
       APPMODE sample_appmode1{};
29
       TASK Equilibrio
31     {
         AUTOSTART    = FALSE;
33       PRIORITY   = 2;
         ACTIVATION   = 1;
35       SCHEDULE   = FULL;
         STACKSIZE    = 512;
37     };

39     TASK Principal
       {
41       AUTOSTART = TRUE
       {
43         APPMODE = sample_appmode1;
        };
45      PRIORITY   = 1;
```

```
        ACTIVATION   = 1;
47      SCHEDULE     = FULL;
        STACKSIZE    = 512;
49    };
};
```

4.2.5. Resultados

Los niveles definidos para mantener el robot en equilibro han sido definidos para la superficie, en la que nosotros hemos utilizado el robot, puede producirse que en otro superficie se necesiten establecer otros niveles para mantener el robot en equilibrio.

A pesar de que sea resuelto el problema con el sensor de luz, el cual es muy sensible a los cambios de iluminación y la superficie en la que se ponga el robot, los resultados han sido buenos y se ha logrado mantener el robot en equilibrio durante un buen periodo de tiempo como se puede observar en el siguiente vídeo.

Vídeo: `http://www.youtube.com/watch?v=OFuzqQBd6Do`

Bibliografía

[1] nxtosek/jsp, Última visita: Febrero 2014.
Url: `http://lejos-osek.sourceforge.net`