|  |
| --- |
| UNIVERSIDAD DE CÓRDOBA |
| Memoria de prácticas |
| Sistemas en Tiempo Real |
|  |
| **Manuel Pedrero Luque**  **Raúl Pérula Martínez** |
| **08/02/2011** |

|  |
| --- |
|  |

[88x31.png](http://creativecommons.org/licenses/by-sa/4.0/)

This work by [Raúl Pérula-Martínez](http://educatech.sytes.net/raul/) is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-sa/4.0/).

Índice

[Introducción 1](#_Toc286838442)

[Práctica 1: Manejo básico de Lego Mindstorms NXT: Sensores y Actuadores 3](#_Toc286838443)

[Apartado A 3](#_Toc286838444)

[Apartado B 5](#_Toc286838445)

[Apartado C 8](#_Toc286838446)

[Práctica 2: Manejo básico de Lego Mindstorms NXT: Tareas y Comunicaciones 12](#_Toc286838447)

[Apartado A 12](#_Toc286838448)

[Apartado B 15](#_Toc286838449)

[Práctica 3: Manejo avanzado de Lego Mindstorms NXT (opcional) 17](#_Toc286838450)

[Apartado A 17](#_Toc286838451)

[Apartado B 22](#_Toc286838452)

[Apartado C 22](#_Toc286838453)

[Bibliografía 23](#_Toc286838454)

[Anexo A. Ficheros fuente 24](#_Toc286838455)

[Práctica 1 24](#_Toc286838456)

[Apartado A 24](#_Toc286838457)

[Apartado B 27](#_Toc286838458)

[Apartado C 33](#_Toc286838459)

[Práctica 2 43](#_Toc286838460)

[Apartado A 43](#_Toc286838461)

[Apartado B 53](#_Toc286838462)

[Práctica 3 71](#_Toc286838463)

[Apartado A 71](#_Toc286838464)

# Introducción

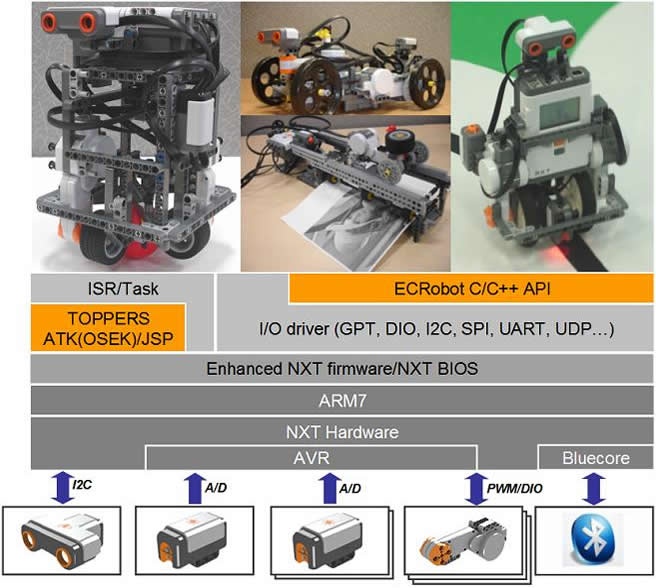
En relación con la asignatura Sistemas en Tiempo Real (STR) de Ingeniería Informática, se ha pedido una serie de prácticas a elaborar mediante robots LEGO Mindstorms para la aplicación y aprendizaje de la metodología seguida en la teoría de la asignatura.

Para la realización de las mismas se ha hecho uso del paquete educativo de robots LEGO Mindstorms y del software proporcionado por nxtOSEK [[1]](#_Bibliografía), drivers, firmware y programación con OIL y C.

nxtOSEKes un Sistema Operativo en Tiempo Real (SOTR) para las controladoras programables NXT de Lego Mindstorms. nxtOSEK consta del driver de E/S parte de de leJOS NXJ, TOPPERS OSEK RTOS y el código necesario para ello. nxtOSEK da la posibilidad de programar el NXT de los robots Lego Mindstorms con ANSI-C/C++.

nxtOSEK provee una API C/C++ que se llama ECRobot. La API de C ECRobot fue diseñada originalmente para desarrollar un entorno de Diseño Basado en Modelos para MATLAB&Simulink (llamado Embedded Coder Robot) para el NXT de LEGO MINDSTORMS. Aunque la API de C no se creó con un muy buen diseño, la nueva API en C++ si está específicamente diseñada para programadores en la materia, así, la estructura y los nombres de la API es más comprensiva y fácil de usar.

A continuación se muestra una imagen en la que se muestra de manera visual la estructura que sigue ECRobot para el SOTR nxtOSEK:



Para la documentación de las prácticas se ha dividido en diferentes secciones para poder documentarla. La primera será esta misma, en la que se detallará cómo se ha realizado la práctica. La segunda será material visual (imágenes y vídeos) para la comprobación de la estructura del robot en cada práctica y del funcionamiento del mismo. Y la tercera será los ficheros correspondientes y necesarios para la ejecución de cada práctica, donde los ficheros fuente estarán correctamente comentados para su posterior visualización y comprensión.

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

## Apartado A

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

Para resolverlo se han hecho diversas pruebas en el calibrado de ambas ruedas no obteniendo un resultado fiable del todo, ya que el material utilizado se encuentra con imperfecciones por el uso continuo de otros años, pero que permite circular más o menos recto.

Para la implementación han hecho falta dos tareas, una de avance y otra para finalizar. Será la tarea de avance la que se modificará con los valores de porcentaje de velocidad en la función *nxt\_motor\_set\_speed* para obtener unos resultados medianamente aceptables y que así el robot se mueva frontalmente y lo más recto posible.

El montaje del robot usado para calibrar las ruedas y potencia del mismo para andar lo más recto posible ha sido el siguiente:

***Imágenes de robot 360º***



Ilustración . Robot LEGO Mindstorms perfil



Ilustración . Robot LEGO Mindstorms planta



Ilustración . Robot LEGO Mindstorms frontal



Ilustración . Robot LEGO Mindstorms trasera

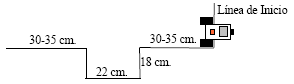
Para justificar el calibrado realizado se muestra el siguiente vídeo en el que se puede apreciar como los valores escogidos proporcionan una solución que más o menos puede ser factible:

***Vídeo de robot circulando frontalmente para la calibración.***

Éstos videos se podrán encontrar en la carpeta Elem.Multimedia → Práctica 1 → Vídeos.

## Apartado B

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



Según el esquema de trabajo a seguir y la descripción del entorno de aparcamiento existen diferentes soluciones al problema. Como no se cuenta con sensores para la realización de la práctica se ha hecho uso de las funciones proporcionadas por nxtOSEK para controlar las distancias andadas y así poder controlar el funcionamiento para aparcar.

Para ello, se ha han creado dos funciones propias, una sirve para mover un motor, el izquierdo o el derecho, una distancia 'y' a una velocidad 'z'. La otra función realiza algo parecido pero con las dos ruedas. Para ello hay que conocer que la longitud de las ruedas es de 18 cm y con ello se puede calcular los grados para un movimiento de, por ejemplo, 90º o 180º.

El giro para realizar el aparcamiento se ha implementado de forma que se realice con ambas ruedas y continuamente hacia atrás como en el aparcamiento normal de un coche.

Para todo esto, se han implementado tres tareas, una para avanzar, otra para girar y otra para parar. Adicionalmente se ha agregado una tarea para poder controlar por pantalla los valores proporcionados por la función *ecrobot\_status\_monitor* y así ver en todo momento los valores de los servomotores en revoluciones.

El montaje del robot usado para realizar el aparcamiento ha sido el siguiente:

***Imágenes de robot 360º***



Ilustración . Robot LEGO Mindstorms frontal



Ilustración . Robot LEGO Mindstorms perfil



Ilustración . Robot LEGO Mindstorms planta

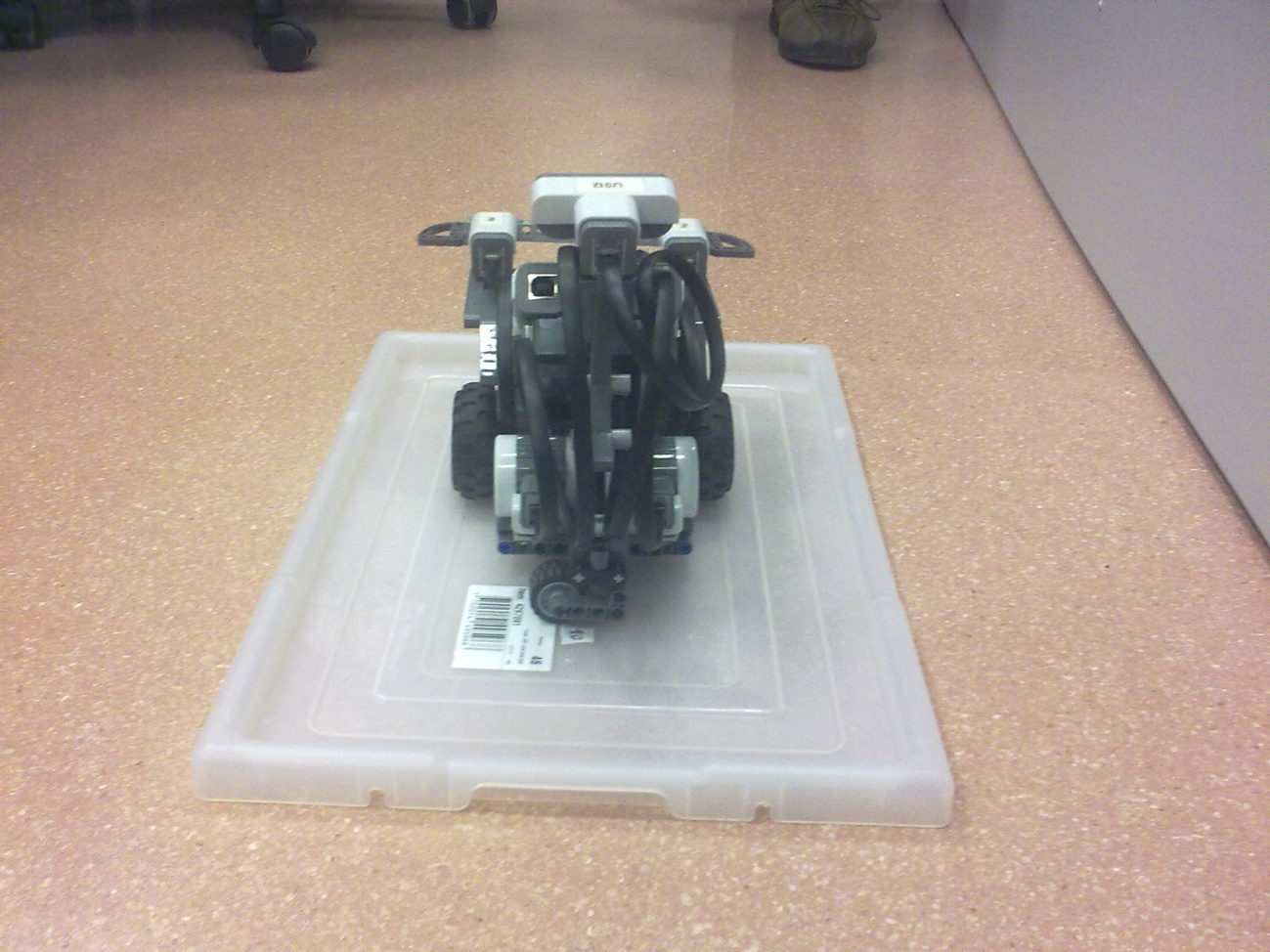


Ilustración . Robot LEGO Mindstorms trasera

Para justificar la realización del aparcamiento se ha realizado un vídeo demostrativo del mismo:

***Vídeo de robot realizando un aparcamiento.***

Éstos videos se podrán encontrar en la carpeta Elem.Multimedia → Práctica 1 → Vídeos.

## Apartado 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 s y girará 90º a la derecha, comenzando de nuevo el mismo procedimiento.**

En este caso, ya que se cuenta con sensores para facilitar y realizar otro tipo de tareas diferentes. Siguiendo la filosofía vista en clase y explicada para las prácticas de realizar varias tareas pequeñas que se puedan controlar mejor y no pocas y con mucho código, se han implementado las siguientes tareas. Tres tareas para controlar la velocidad de los motores, siendo la primera la tarea para la máxima velocidad, la segundo la mitad de la velocidad de la anterior y la tercera la mitad de la velocidad de la anterior o un cuarto de la primera. Una tarea para retroceder, una tarea para parar los motores y una tarea para girar, en este caso concreto, 90º. Por último, se han realizado una serie de tareas periódicas que sirven para captar el valor del sensor de sonar, otra para, con dicho valor, controlar la distancia capturada y en base a ello realizar una acción, y otra para controlar sensores de pulsación. Además de estas tareas se encuentra también una tarea adicional que muestra los datos de estado del NXT por la pantalla LCD.

Como aspectos a tener en cuenta, hay que inicializar y finalizar el sensor de sonar para que el sistema del NXT pueda reconocer el puerto del sensor; además hay que tener en cuenta que para captar el valor del sensor de sonar hay que hacer una espera de 20 ms para que el valor sea correcto.

Se ha hecho uso de la función del apartado anterior para mover un motor una distancia especificada y con ello se puede realizar el giro de 90º.

Otro de los aspectos a tener en cuenta es que hemos descubierto que cuando se encuentra en el momento de retroceder o de girar y se presiona algunos de los pulsadores, el sistema no capta bien esto y hacía que el sistema se colgase no proporcionando el funcionamiento deseado. Para arreglar dicho bug se ha tomado la decisión de crear un flag, llamado “atras” el cual soluciona dicho problema.

El montaje del robot usado para realizar este funcionamiento ha sido el siguiente:

***Imágenes de robot 360º***



Ilustración . Robot LEGO Mindstorms lateral



Ilustración . Robot LEGO Mindstorms planta



Ilustración . Robot LEGO Mindstorms frontal



Ilustración . Robot LEGO Mindstorms trasera

Para justificar la realización del avance junto con el cambio de velocidades y, cuando existe una pulsación, el retroceso y giro de 90º, se ha realizado un vídeo demostrativo del mismo:

***Vídeo de robot reduciendo la velocidad, chocando, retrocediendo y girando 90º.***

Éstos videos se podrán encontrar en la carpeta Elem.Multimedia → Práctica 1 → Vídeos.

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

## Apartado 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). Se incorporará un sensor de pulsación en la parte superior para indicarle al robot que ha salido del laberinto.**

Para la realización de esta práctica, además de haber modificado la estructura del robot que posteriormente se mostrará, la complejidad del problema ha aumentado considerablemente. Se han introducido nuevos sensores como el sonar y las tareas han adquirido una composición más compleja.

Se ha hecho uso de variables compartidas globales para los controles de los valores obtenidos por el sensor de sonar, para el control de las comparaciones o chequeos, para la detección de los posibles choques y para la parada del robot.

También se han implementado diversas funciones o módulos para detener el robot parando sus servomotores. Para realizar un muestreo eficiente, es decir, se han realizado tres obtenciones de los valores del sonar y posteriormente se ha realizado la media de ellos, con ello podremos asegurar que si existe un valor muy diferente no lo tendrá totalmente en cuenta. Y por último, se ha seguido haciendo uso de las funciones para mover los motores que se han especificado en los dos últimos apartados de la práctica anterior.

Entrando en la especificación de las tareas podremos decir que existen dos tipos de tareas implementadas, las tareas no periódicas y las tareas periódicas. De las tareas periódicas, la primera es la usada para leer de los pulsadores, esta tarea comprobará si se ha pulsado o no un pulsador y en caso afirmativo se efectuará la acción de parada del robot. La siguiente tarea se dedica a leer los valores del sonar constantemente de la misma manera que la función implementada con la salvedad de que ésta siempre se está ejecutando.

De las tareas no periódicas, la primera que aparecerá será la encargada de comparar a qué distancia se encuentra de un obstáculo para así parar y empezar el chequeo o avanzar según ésta. Dentro de esta tarea se llamará a la tarea mirar que hará justamente lo necesario para que la anterior pueda funcionar correctamente, es decir, girará 90º a la izquierda y a partir de ese punto comenzará a chequear en un rango de 180º, en intervalos de 10º aproximadamente obteniendo el valor máximo por el que deberá guiarse para salir del laberinto. Por último, se encuentra la tarea encargada del avance en el que simplemente se pondrán los motores en funcionamiento continuo a una velocidad determinada.

Excepcionalmente y como tarea periódica pero no de las principales, se encuentra la tarea para mostrar por pantalla los valores de estado del robot. Ésta tareas ya se ha definido anteriormente y no se hará más hincapié en ella.

El montaje del robot usado para realizar este funcionamiento ha sido el siguiente:

***Imágenes de robot 360º***

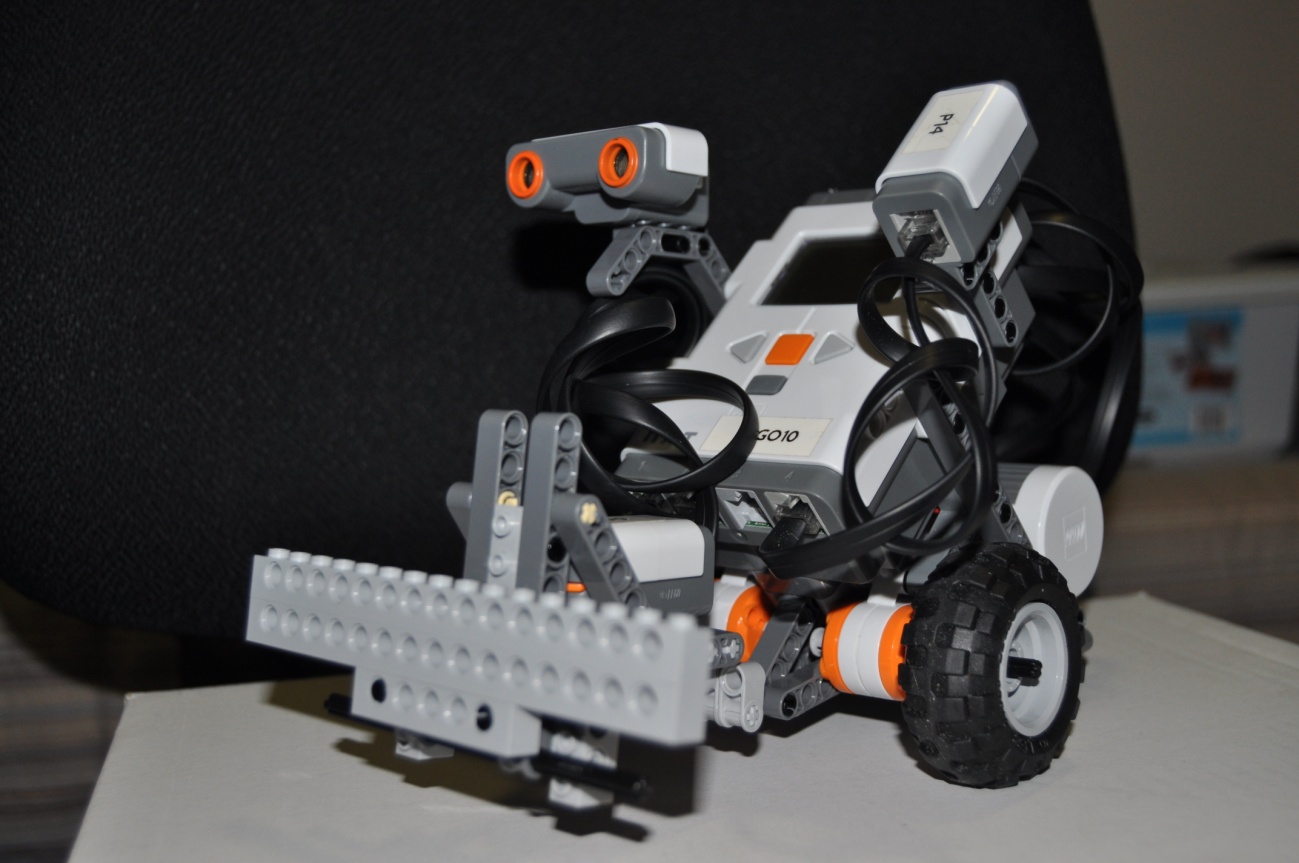


Ilustración . Robot LEGO Mindstorms principal

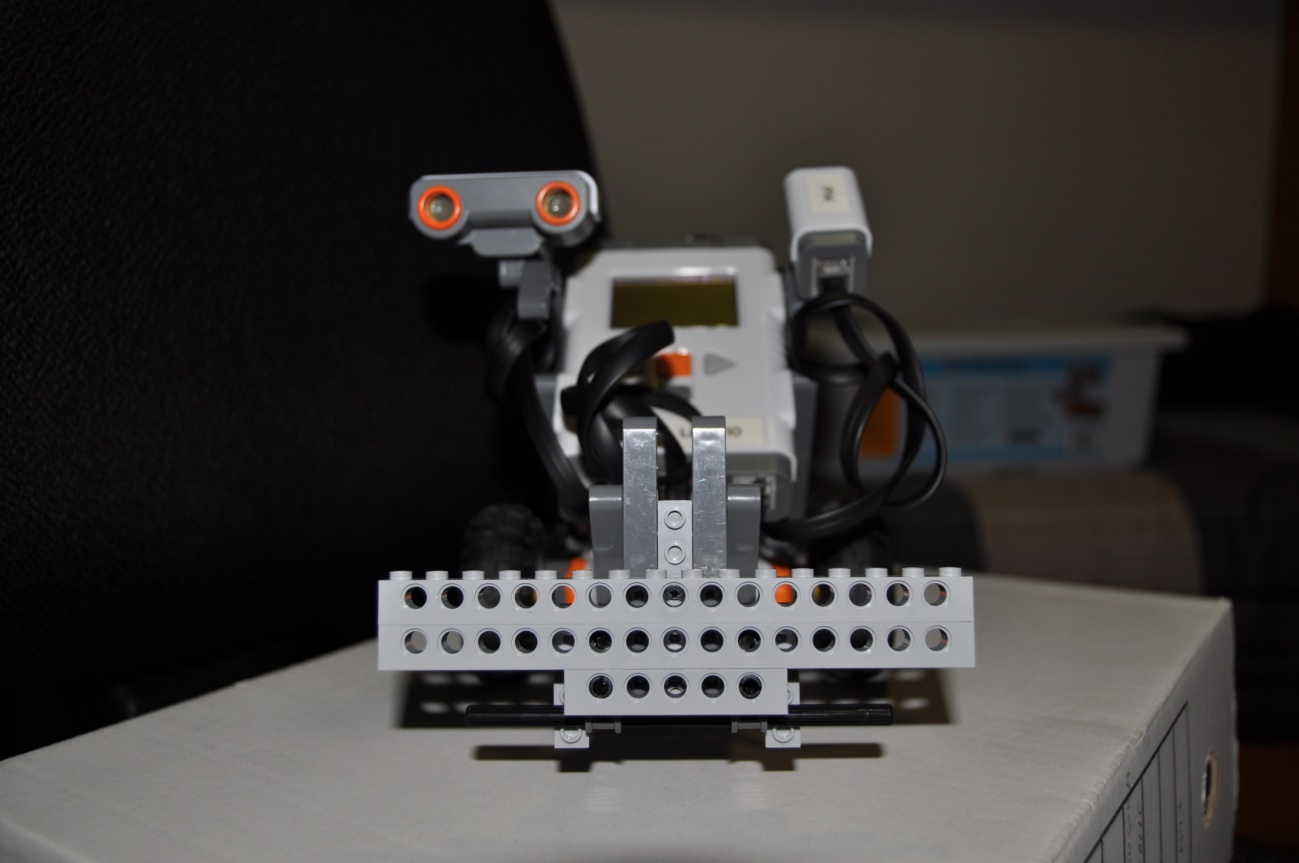


Ilustración . Robot LEGO Mindstorms frontal

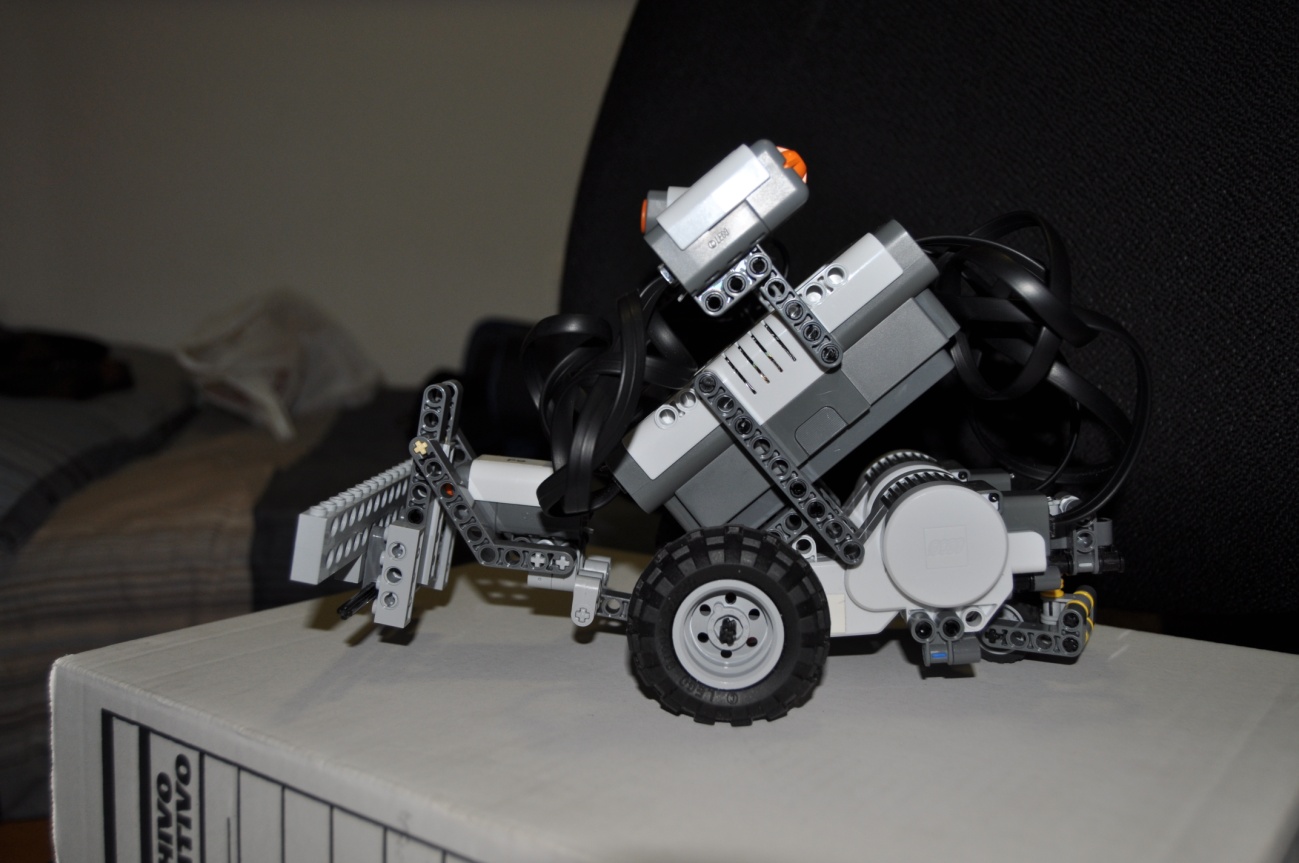


Ilustración . Robot LEGO Mindstorms perfil



Ilustración . Robot LEGO Mindstorms trasera

Para justificar la realización de la resolución del laberinto mediante detección por sonar, se ha realizado un vídeo demostrativo del mismo:

***Vídeos de robot saliendo del laberinto.***

Éstos videos se podrán encontrar en la carpeta Elem.Multimedia → Práctica 1 → Vídeos.

## Apartado 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).**

En esta práctica, implementada en base a la arquitectura cliente/servidor se han tenido que realizar dos códigos, uno para el cliente y otro para el servidor. Para cada uno de ellos se comentarán las tareas que se han implementado con base del apartado anterior, ya que es prácticamente añadir y establecer un protocolo de comunicación y almacenamiento de los datos.

Como se ha comentado, en el caso del servidor se ha hecho uso de la mayoría de las tareas del apartado anterior modificando las partes que se necesitaban para el establecimiento de la comunicación entre robots. Para ello, al principio hay que especificar la dirección MAC del robot cliente para poder establecer la comunicación mediante bluetooth. Para almacenar los movimientos que posteriormente se enviarán al cliente, se usará una matriz en la que las filas representarán el valor del movimiento y las columnas el movimiento en sí, es decir, 0 para identificar el avance y 1 para identificar los giros.

Para la transmisión se ha implementado una función de envío por bluetooth la cual envía de cinco en cinco los movimientos realizados. En el primer envío se mandará el número de pasos que habrá en total. En la tarea de leer pulsadores, cuando el pulsador se encuentre activo se rellenará con un avance cero, es decir, parado. En la tarea de comparar, en función de si el robot se encuentra por debajo del umbral de cercanía a un obstáculo, momento en el que se parará y chequeará para un posterior avance en la dirección correcta, o si ha chocado. En la tarea de mirar se almacenará el evento de girar y los grados a girar para un posicionamiento en la dirección correcta para que el cliente pueda recorrer el camino seguido por el servidor de una manera lo más similar posible. Para el cálculo de los grados a mandar al cliente se cogerá el índice, que es la posición regulada cada diez grados, esta cantidad será la que se le quite a los 180º de la vuelta para poder posicionarse en el lugar correcto, y posteriormente será esta cantidad resultante la que se reste a 90 para, de esta forma, controlar si lo que se ha hecho es un giro a la derecha o a la izquierda de la posición en la que se comenzó el chequeo.

Para la parte del cliente se ha creado una tarea principal de avance que será la que haga la llamada a una función para la recepción desde el servidor de los datos. Esta función hará de manera análoga a la del envío de datos, una recepción de cinco datos. Ya con los datos recibidos se recorrerá la matriz de acciones para elegir y ordenar al robot el avanzar o girar.

Adicionalmente se seguirá manteniendo la tarea para mostrar por pantalla los datos del estado del robot para controlar en todo momento las revoluciones del mismo. Aunque posiblemente podrá encontrarse desactivada para poder mostrar los datos recibidos y así poder comparar con los datos enviados desde el servidor.

El montaje de los robots usados para realizar este funcionamiento ha sido, el mismo usado por el [apartado anterior](#_Apartado_A_2) para el robot servidor y el montaje realizado en la [práctica 1](#_Práctica_1:_Manejo), [apartado a](#_Apartado_A), para el robot cliente.

Para justificar la realización de la resolución del laberinto se ha realizado un vídeo demostrativo del mismo en el cual se muestra al robot cliente saliendo del laberinto:

***Vídeo de robot cliente saliendo de un laberinto resuelto por el servidor.***

Éstos videos se podrán encontrar en la carpeta Elem.Multimedia → Práctica 2 → Vídeos.

# Práctica 3: Manejo avanzado de Lego Mindstorms NXT (opcional)

## Apartado 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.**

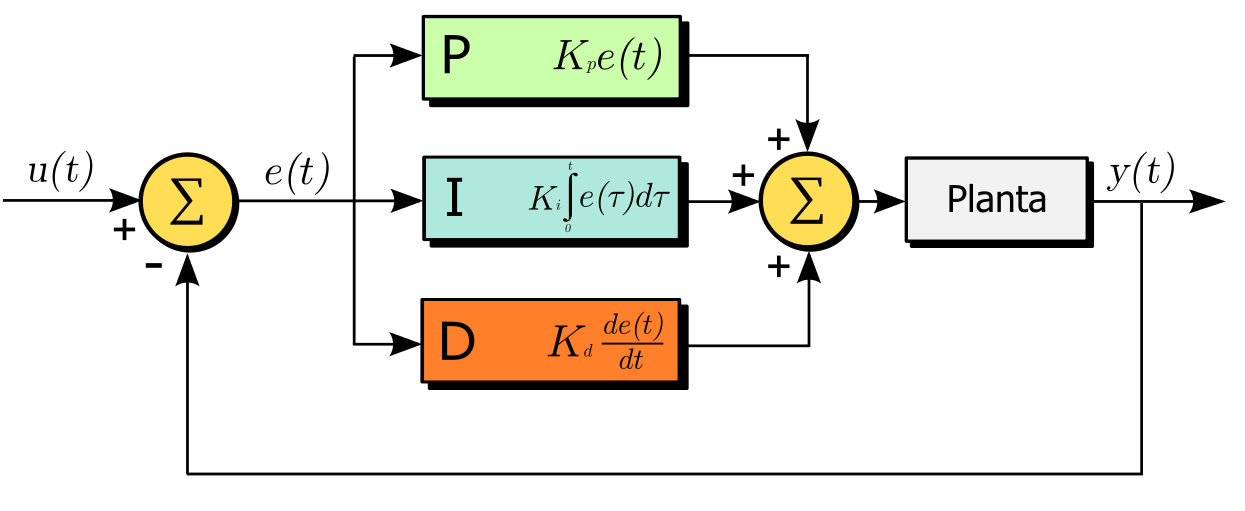
Para la realización de esta práctica se empezó con un enfoque sencillo, se equilibró el robot y se usó el valor de luminosidad para determinar la iluminación en el centro de gravedad aproximado. Se determinó que la luminosidad aumentaba si se volcaba el robot hacia atrás, y que disminuía si se volcaba hacia adelante, por lo que se optó por definir tres intervalos de modo que la tarea simplemente recogía el nivel de luminosidad, determinaba en qué intervalo se encontraba y activaba los motores a tres velocidades fijas (según el intervalo) para contrarrestar el desvío del robot.

Esta idea resultó ineficiente por dos razones: la luminosidad en el punto de equilibrio variaba según la luminosidad del ambiente y de la superficie donde se apoyaba el robot, por lo que al variar la superficie el robot utilizaba un centro de gravedad erróneo. Por otra parte, el uso de velocidades fijas era muy impreciso y el robot no llegaba a equilibrarse.

Un segundo enfoque fue el uso de un esquema proporcional. Para ello se estableció un límite de desviación a partir del cual se considera que el robot está en un estado irrecuperable. Luego se estableció una función de proporcionalidad que daba el máximo de potencia a los motores cuando la luminosidad alcanzaba el valor límite que habíamos establecido e iba disminuyendo su potencia conforme el robot se acercaba al punto de equilibrio.

Con este enfoque conseguimos que el robot se mantuviese oscilando 2 ó 3 segundos hasta que la oscilación era demasiado grande y el robot caía.

Con la idea mejorar el resultado, se buscó información sobre mecanismos de control, y se optó por un mecanismo PID (Proporcional, Integral, Derivativo) [[12]](#_Bibliografía) que gracias a su posibilidad de retroalimentación fuese capaz de corregir las desviaciones del robot. En concreto, se utilizó la web [[13]](#_Bibliografía) que explica los fundamentos del PID aplicado a un robot que sigue una línea negra.



En dicho mecanismo se calculan tres mediciones y se obtiene una suma ponderada de las mismas cuyo resultado se aplica a los motores del robot. Cada una de estas medidas busca corregir un aspecto de las desviaciones del robot:

* La función proporcional realiza una corrección mayor cuanto mayor sea la diferencia entre el valor de luminosidad actual y el valor de luminosidad en el punto de equilibrio (error de desviación).
* La función integral va sumando los errores de desviación de cada iteración de manera acumulativa, de modo que si un error se mantiene durante un periodo de tiempo prolongado, el resultado de este sumatorio aumenta y por tanto aumenta la potencia de los motores para corregirlo. Se basa por tanto en el comportamiento anterior del sistema para corregir la desviación.
* La función derivativa se basa en el error actual del robot y en el error anterior para determinar cuál será el error siguiente. Es decir, predice el comportamiento del robot y actúa en consecuencia. Esta función resulta especialmente útil cuando el robot ha corregido un desvío hacia un lado y por inercia tiende a desviarse para el otro. Gracias a esta función, los motores reducen su potencia antes de llegar al punto de equilibrio evitando que las oscilaciones sean cada vez mayores.

El sistema recoge el valor actual de luminosidad y calcula las tres funciones. A continuación pondera las mismas para dar más peso a una que a otra y suma los tres resultados, obteniendo así la potencia que se ha de aplicar a los servomotores.

A la hora de ponderar se usó un procedimiento similar al descrito en [[13]](#_Bibliografía). Se aumentaba el peso de la función proporcional eliminando las demás hasta que el robot oscila respecto al punto de equilibrio y a partir de ahí, se aumenta el peso de la función integral y por último el de la función derivativa hasta conseguir una respuesta adecuada. En nuestro caso, después de muchas pruebas, el comportamiento óptimo se consiguió con unos pesos de 5 para la función proporcional, 2 para la función integral y 1 para la función derivativa. Los resultados han sido bastante buenos, como se puede ver en el vídeo adjunto.

A la hora de implementar este procedimiento en OSEK se han considerado tres tareas:

* La tarea Calibrar, se encarga de recoger el valor de luminosidad base que se utilizará en el resto del programa. Para ello se equilibra el robot manualmente lo mejor que se pueda y se pulsa el botón de calibración, momento en el que se recoge la medida de luminosidad y se almacena.
* La tarea Equilibrar, se encarga de la implementación del esquema PID y del equilibrio del robot. Para ello recoge el valor de luminosidad de un instante, calcula las funciones Proporcional, Integral y Derivativa y obtiene la potencia que ha de suministrar a los motores. Puede ser interrumpida en cualquier momento por la tarea Calibrar.
* La tarea PantallaLCD, se encarga de mostrar por pantalla información relevante como los pesos de las funciones, los valores de las mismas en bruto y ponderados, la potencia que se le aplica a los motores, el valor de luminosidad actual y el valor de luminosidad base.

También se han parametrizado, para calibrar el robot en diferentes situaciones, los pesos de las funciones (Kp, Ki y Kd) y dos coeficientes que sirven para controlar la función integral:

* Kai: es la constante de amortiguación de la integral. Se puede obviar estableciéndola a 1 o aplicar si se usa un valor menor, como 2/3. Esta constante multiplica en cada iteración al valor de la integral, reduciendo su efecto en casos en los que se haya obtenido un error de desviación en un periodo relativamente largo de tiempo.
* Kri: es la constante de reseteo de la integral. Esta constante divide el valor de la integral cuando se obtiene una desviación del robot igual a 0. Sirve para limitar el impacto de la función integral en casos en los que el robot está prácticamente equilibrado.

En la práctica se ha establecido ambas variables a 1 (sin incidencia) pero se incluyen para que se pueda recalibrar en distintas situaciones.

El montaje del robot usado para realizar este funcionamiento ha sido el siguiente:

***Imágenes de robot 360º***



Ilustración . Robot LEGO Mindstorms frontal

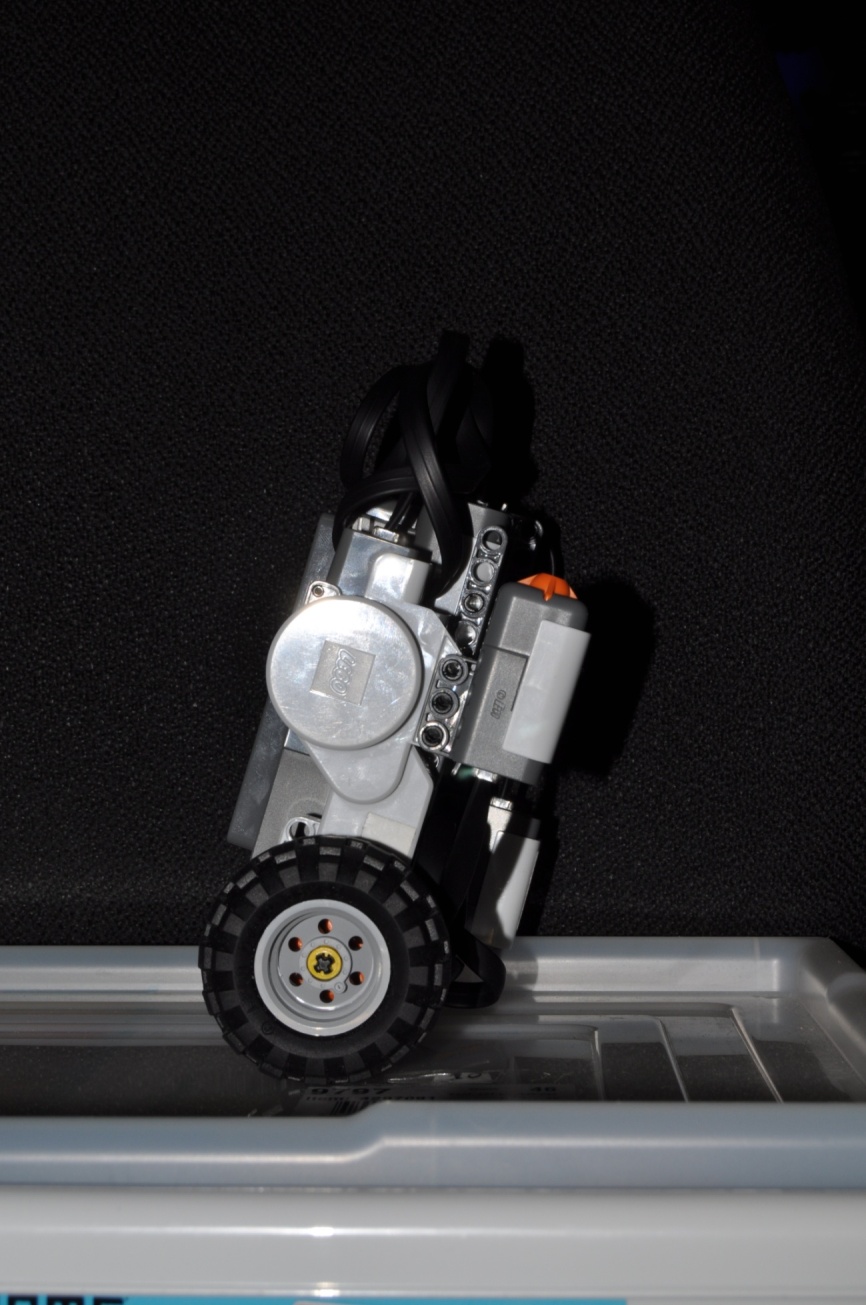


Ilustración . Robot LEGO Mindstorms perfil izquierdo

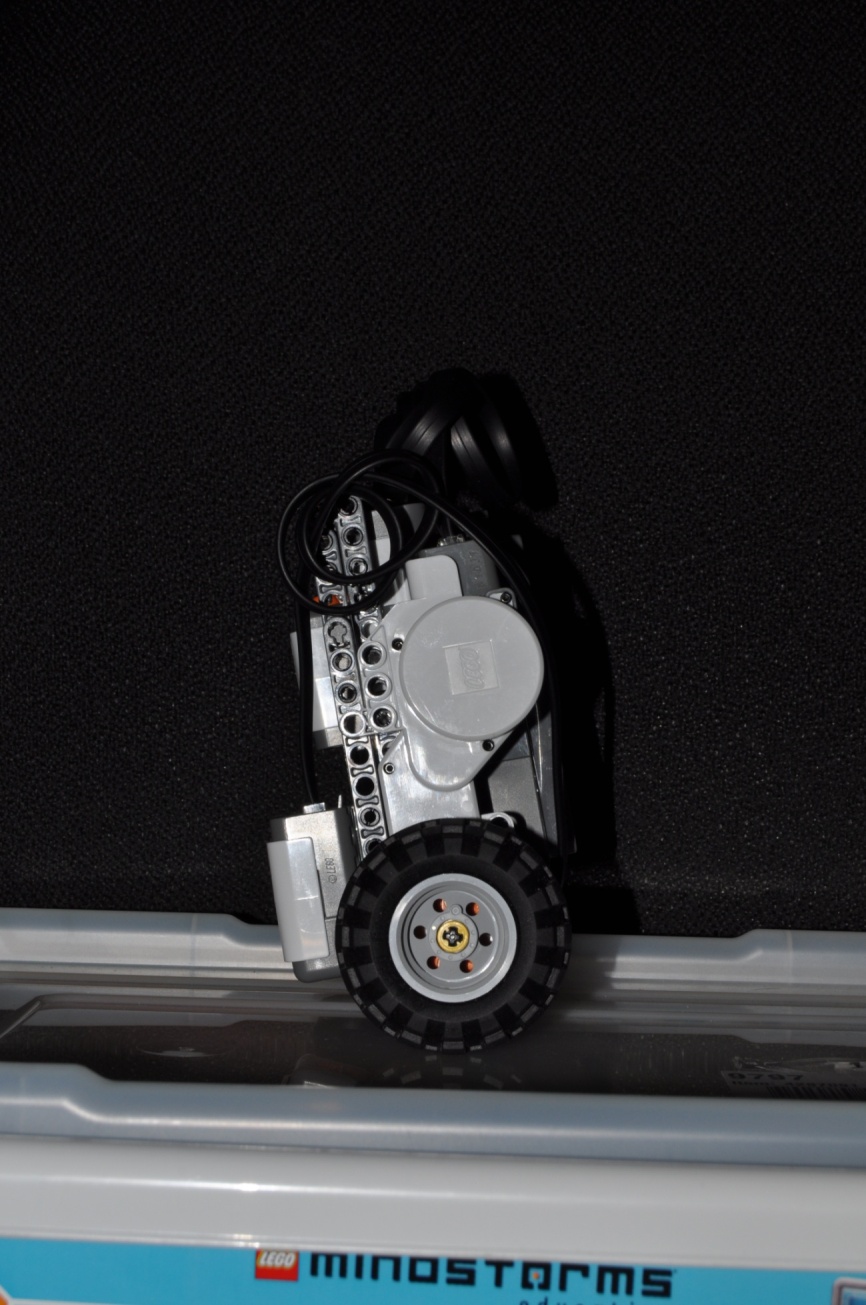


Ilustración . Robot LEGO Mindstorms perfil derecho



Ilustración . Robot LEGO Mindstorms calibrando



Ilustración . Robot LEGO Mindstorms en uso

Para justificar la realización de un mecanismo de equilibro automático se ha realizado un vídeo demostrativo del mismo:

***Vídeo de robot manteniendo el equilibrio.***

Éstos videos se podrán encontrar en la carpeta Elem.Multimedia → Práctica 3 → Vídeos.

## Apartado B

**Incorporar a dicho robot un sensor de ultrasonidos y programarlo para que sea capaz de avanzar sin caerse y retroceder en caso de detectar un obstáculo delante de él.**

No se ha podido realizar.

## Apartado C

**Programar dicho robot para que avance cuando no detecte ningún obstáculo y cuando detecte un obstáculo sea capaz de seguirlo, avanzando, retrocediendo o realizando giros hacia la derecha o hacia la izquierda, según movamos dicho obstáculo (por ejemplo, poniendo la mano delante del sensor de ultrasonidos).**

No se ha podido realizar.

# Bibliografía

1. nxtOSEK/JSP (ANSI C/C++ with OSEK/μITRON RTOS for LEGO MINDSTORMS NXT). Última visita: 20-02-2011.

Enlace: <http://lejos-osek.sourceforge.net/>

1. Instalación de nxtOSEK. Última visita: 20-02-2011.

Enlace: <http://lejos-osek.sourceforge.net/installation.htm>

1. ECRobot C/C++ API for nxtOSEK. Última visita: 20-02-2011.

Enlace: <http://lejos-osek.sourceforge.net/api.htm>

1. FAQ nxtOSEK. Última visita 20-02-2011.

Enlace: <http://lejos-osek.sourceforge.net/faq.htm>

1. Manual de Referencia OIL. UCO Moodle. Última visita: 20-02-2011.

Enlace: <http://www3.uco.es/moodle/>

1. Manual de Referencia nxtOSEK. UCO Moodle. Última visita: 20-02-2011.

Enlace: <http://www3.uco.es/moodle/>

1. Portal OSEK VDX. Última visita: 20-02-2011.

Enlace: <http://portal.osek-vdx.org/index.php?option=com_frontpage&Itemid=1>

1. OSEK Implementation Language (OIL) (Version 2.5). Última visita: 20-02-2011.

Enlace: <http://portal.osek-vdx.org/files/pdf/specs/oil25.pdf>

1. OSEK OS (Version 2.2.3). Última visita: 20-02-2011.

Enlace: <http://portal.osek-vdx.org/files/pdf/specs/os223.pdf>

1. LEGO Mindstorms. Última visita: 20-02-2011.

Enlace: <http://mindstorms.lego.com/en-us/default.aspx>

1. UCO Moodle. Última visita: 20-02-2011.

Enlace: <http://www3.uco.es/moodle/>

1. Wikipedia: Proporcional integral derivativo.Última visita: 02-03-2011.

Enlace: <http://es.wikipedia.org/wiki/Proporcional_integral_derivativo>

1. PID controller for NXT Robots

Enlace: [http://www.inpharmix.com/jps/ PID\_Controller\_For\_Lego\_Mindstorms\_Robots.html](http://www.inpharmix.com/jps/%20PID_Controller_For_Lego_Mindstorms_Robots.html)

# Anexo A. Ficheros fuente

## Práctica 1

### Apartado A

#### Fichero OIL

|  |
| --- |
| #include "implementation.oil"  CPU ATMEL\_AT91SAM7S256 // CPU del Lego NXT{ OS LEJOS\_OSEK { STATUS = EXTENDED; STARTUPHOOK = FALSE; ERRORHOOK = FALSE; SHUTDOWNHOOK = FALSE; PRETASKHOOK = FALSE; POSTTASKHOOK = FALSE; USEGETSERVICEID = FALSE; USEPARAMETERACCESS = FALSE; USERESSCHEDULER = FALSE; };  APPMODE sample\_appmode1{}; // Modo de aplicacion por defecto.  //##############################################################################  TASK Avance { AUTOSTART = TRUE { APPMODE = sample\_appmode1; }; // Autoinicio en modo de aplicacion descrito  // Si no se desea que se autoarranque: AUTOSTART = FALSE; PRIORITY = 1; //Tarea de menor prioridad ACTIVATION = 1; SCHEDULE = FULL; STACKSIZE = 512; };  TASK Final { AUTOSTART = TRUE { APPMODE = sample\_appmode1;  }; PRIORITY = 2; ACTIVATION = 1; SCHEDULE = FULL; STACKSIZE = 512; };  }; |

#### Fichero C

|  |
| --- |
| #include "kernel.h"#include "kernel\_id.h"#include "ecrobot\_interface.h"  /\*--------------------------------------------------------------------------\*//\* OSEK declarations \*//\*--------------------------------------------------------------------------\*/  DeclareTask(Avance);DeclareTask(Final);  /\*--------------------------------------------------------------------------\*//\* Definitions \*//\*--------------------------------------------------------------------------\*/  /\*--------------------------------------------------------------------------\*//\* Function to be invoked from a category 2 interrupt \*//\*--------------------------------------------------------------------------\*/void user\_1ms\_isr\_type2(){}  void ecrobot\_device\_terminate(){ ecrobot\_set\_motor\_speed(NXT\_PORT\_A, 0); ecrobot\_set\_motor\_speed(NXT\_PORT\_C, 0);}  //##############################################################################  /\*--------------------------------------------------------------------------\*//\* Task information: \*//\* ----------------- \*//\* Name : Avance \*//\* Priority: 1 \*//\* Typ : EXTENDED TASK \*//\* Schedule: FULL \*//\* Objective: Funcion para avanzar y a la vez calibrar los servomotores \*//\*--------------------------------------------------------------------------\*/TASK(Avance){ //int time\_out = systick\_get\_ms()+5000; // Se calcula 5 segundos  // Activa motores para comprobar si existe desvio en la navegacion nxt\_motor\_set\_speed(NXT\_PORT\_A, 49, 1); nxt\_motor\_set\_speed(NXT\_PORT\_C, 50, 1);  // Espera hasta que se agote el time\_out systick\_wait\_ms(5000);  // Despierta la tarea de parada de motores ActivateTask(Final);  // Termina la tarea actual TerminateTask();}  /\*--------------------------------------------------------------------------\*//\* Task information: \*//\* ----------------- \*//\* Name : Final \*//\* Priority: 2 \*//\* Typ : EXTENDED TASK \*//\* Schedule: FULL \*//\* Objective: Funcion para parar los servomotores \*//\*--------------------------------------------------------------------------\*/TASK(Final){ // Parar los motores nxt\_motor\_set\_speed(NXT\_PORT\_A, 0, 1); nxt\_motor\_set\_speed(NXT\_PORT\_C, 0, 1);  // Termina la tarea actual TerminateTask();  } |

### Apartado B

#### Fichero OIL

|  |
| --- |
| #include "implementation.oil"  CPU ATMEL\_AT91SAM7S256 // CPU del Lego NXT{ OS LEJOS\_OSEK { STATUS = EXTENDED; STARTUPHOOK = FALSE; ERRORHOOK = FALSE; SHUTDOWNHOOK = FALSE; PRETASKHOOK = FALSE; POSTTASKHOOK = FALSE; USEGETSERVICEID = FALSE; USEPARAMETERACCESS = FALSE; USERESSCHEDULER = FALSE; }; APPMODE sample\_appmode1{}; // Modo de aplicacion por defecto.  //##############################################################################  TASK Avance { AUTOSTART = TRUE { APPMODE = sample\_appmode1; }; // Autoinicio en modo de aplicacion descrito  // Si no se desea que se autoarranque: AUTOSTART = FALSE; PRIORITY = 10; ACTIVATION = 1; SCHEDULE = FULL; STACKSIZE = 512; };  TASK Giro { AUTOSTART = FALSE; PRIORITY = 10; ACTIVATION = 1; SCHEDULE = FULL; STACKSIZE = 512; };  TASK Final { AUTOSTART = FALSE; PRIORITY = 11; ACTIVATION = 1; SCHEDULE = FULL; STACKSIZE = 512; };  //##############################################################################  /\* Definicion de contador de alarma \*/ COUNTER SysTimerCnt { MINCYCLE = 1; MAXALLOWEDVALUE = 10000; TICKSPERBASE = 1; /\* Un tick es igual a 1 ms \*/ };  /\* Definicion del tiempo de ejecución de la tarea para la LCD \*/ ALARM AlarmaLCD { COUNTER = SysTimerCnt; ACTION = ACTIVATETASK { TASK = PantallaLCD; }; AUTOSTART = TRUE { ALARMTIME = 10; CYCLETIME = 500; /\* se ejecuta cada 500 ms \*/ APPMODE = sample\_appmode1; }; };  /\* Definicion de a tarea en background para la LCD \*/ TASK PantallaLCD { AUTOSTART = TRUE { APPMODE = sample\_appmode1; }; PRIORITY = 15; /\* lowest priority \*/ ACTIVATION = 1; SCHEDULE = FULL; STACKSIZE = 512; };  }; |

#### Fichero C

|  |
| --- |
| #include "kernel.h"#include "kernel\_id.h"#include "ecrobot\_interface.h"  #define MOT\_I NXT\_PORT\_C#define MOT\_D NXT\_PORT\_A#define MED\_IZD 50#define MED\_DCH 49  //##############################################################################  // Funcion para mover el motor izq. o der. una distancia 'y' a una velocidad 'z'void mover\_motor(U32 motor, int distancia, int velocidad){ int grados = (360\*distancia)/18; if(velocidad < 0){ grados = grados\*(-1); }  int ref = nxt\_motor\_get\_count(motor)+grados; nxt\_motor\_set\_speed(motor, velocidad, 1);  if(velocidad < 0){ while(nxt\_motor\_get\_count(motor) > ref); } else{ while(nxt\_motor\_get\_count(motor) < ref); }  nxt\_motor\_set\_speed(motor, 0, 1);}  // Funcion para mover los dos motores una distancia 'y' a una velocidad 'z'void mover\_motores(U32 motor1, U32 motor2, int dist1, int dist2, int vel1, int vel2){ int llega1 = 0; int llega2 = 0; int dir1 = 1; //1 -> adelante, -1 -> atras, 0 -> parado int dir2 = 1;  // Las ruedas motrices tienen 18 cm de circunferencia int grados1 = (360\*dist1)/18; int grados2 = (360\*dist2)/18;  if(vel1 < 0){ grados1 = -grados1; dir1 = -dir1; } if(vel2 < 0){ grados2 = -grados2; dir2 = -dir2; }  int ref1 = nxt\_motor\_get\_count(motor1)+grados1; int ref2 = nxt\_motor\_get\_count(motor2)+grados2;  nxt\_motor\_set\_speed(motor1, vel1, 1); nxt\_motor\_set\_speed(motor2, vel2, 1);  while((llega1 != 1) || (llega2 != 1)){ if(dir1 < 0){ if(nxt\_motor\_get\_count(motor1) <= ref1){ llega1 = 1; nxt\_motor\_set\_speed(motor1, 0, 1); } } else{ if(nxt\_motor\_get\_count(motor1) >= ref1){ llega1 = 1; nxt\_motor\_set\_speed(motor1, 0, 1); } } if(dir2 < 0){ if(nxt\_motor\_get\_count(motor2) <= ref2){ llega2 = 1; nxt\_motor\_set\_speed(motor2, 0, 1); } } else{ if(nxt\_motor\_get\_count(motor2) >= ref2){ llega2 = 1; nxt\_motor\_set\_speed(motor2, 0, 1); } } }}  //##############################################################################  /\*--------------------------------------------------------------------------\*//\* OSEK declarations \*//\*--------------------------------------------------------------------------\*/DeclareTask(Avance);DeclareTask(Giro);DeclareTask(Final);DeclareTask(PantallaLCD);DeclareCounter(SysTimerCnt);  /\*--------------------------------------------------------------------------\*//\* Definitions \*//\*--------------------------------------------------------------------------\*/  /\*--------------------------------------------------------------------------\*//\* Function to be invoked from a category 2 interrupt \*//\*--------------------------------------------------------------------------\*/void user\_1ms\_isr\_type2(){ StatusType ercd;  ercd = SignalCounter(SysTimerCnt); /\* Incrementa el contador de alarma \*/ if (ercd != E\_OK) { ShutdownOS(ercd); }}  void ecrobot\_device\_terminate(){ ecrobot\_set\_motor\_speed(MOT\_I, 0); ecrobot\_set\_motor\_speed(MOT\_D, 0);}  //##############################################################################  /\*--------------------------------------------------------------------------\*//\* Task information: \*//\* ----------------- \*//\* Name : Avance \*//\* Priority: 10 \*//\* Typ : EXTENDED TASK \*//\* Schedule: FULL \*//\* Objective: Tarea para avanzar moviendo los servomotores \*//\*--------------------------------------------------------------------------\*/TASK(Avance){ // Esperar 5 segundos para empezar el programa systick\_wait\_ms(5000); // Mover los motores hacia delante mover\_motores(MOT\_I, MOT\_D, 40, 40, 50, 49);  // Activar la tarea de giro ActivateTask(Giro);  // Termina la tarea actual TerminateTask();}  /\*--------------------------------------------------------------------------\*//\* Task information: \*//\* ----------------- \*//\* Name : Giro \*//\* Priority: 10 \*//\* Type : EXTENDED TASK \*//\* Schedule: FULL \*//\* Objective: Tarea para girar haciendo uso de los servomotores \*//\*--------------------------------------------------------------------------\*/TASK(Giro){ // Activa motores para comprobar si existe desvio en la navegacion mover\_motor(MOT\_I, 19, 50); mover\_motores(MOT\_I, MOT\_D, 5, 5, 50, 49); mover\_motor(MOT\_D, 19, 49);  // Termina la tarea actual TerminateTask();}  /\*--------------------------------------------------------------------------\*//\* Task information: \*//\* ----------------- \*//\* Name : Final \*//\* Priority: 11 \*//\* Typ : EXTENDED TASK \*//\* Schedule: FULL \*//\* Objective: Tarea para parar los servomotores \*//\*--------------------------------------------------------------------------\*/TASK(Final){ // Parar los motores nxt\_motor\_set\_speed(MOT\_D, 0, 1); nxt\_motor\_set\_speed(MOT\_I, 0, 1);  // Termina la tarea actual TerminateTask();}  /\*--------------------------------------------------------------------------\*//\* Task information: \*//\* ----------------- \*//\* Name : PantallaLCD \*//\* Priority: 11 \*//\* Typ : EXTENDED TASK \*//\* Schedule: FULL \*//\* Objective: Tarea que muestra los valores de estado por la LCD \*//\*--------------------------------------------------------------------------\*/TASK(PantallaLCD)  {  // Muestra por la pantalla del nxt los datos de los servomotores ecrobot\_status\_monitor("Aparcamiento");  // Termina la tarea actual  TerminateTask();  } |

### Apartado C

#### Fichero OIL

|  |
| --- |
| #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 VelBaja  {  AUTOSTART = FALSE;  PRIORITY = 4;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK VelMedia  {  AUTOSTART = FALSE;  PRIORITY = 3;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK VelAlta  {  AUTOSTART = FALSE;  PRIORITY = 2;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };  TASK Retroceso  {  AUTOSTART = FALSE;  PRIORITY = 7;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };  TASK CambioDir  {  AUTOSTART = FALSE;  PRIORITY = 6;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };  TASK PararMotor  {  AUTOSTART = FALSE;  PRIORITY = 5;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };  //##############################################################################  // Tareas periódicas    TASK ChequeoDistancia  {  AUTOSTART = TRUE  {  APPMODE = sample\_appmode1;  };  PRIORITY = 10;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK ChequeoSonar  {  AUTOSTART = TRUE  {  APPMODE = sample\_appmode1;  };  PRIORITY = 10;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK ChequeoPulsadores  {  AUTOSTART = TRUE  {  APPMODE = sample\_appmode1;  };  PRIORITY = 11;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };  //##############################################################################    TASK PantallaLCD  {  AUTOSTART = TRUE  {  APPMODE = sample\_appmode1;  };  PRIORITY = 1;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };  COUNTER SysTimerCnt  {  MINCYCLE = 1;  MAXALLOWEDVALUE = 10000;  TICKSPERBASE = 1;  };  ALARM AlarmaLCD  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = PantallaLCD;  };  AUTOSTART = TRUE  {  ALARMTIME = 10;  CYCLETIME = 500;  APPMODE = sample\_appmode1;  };  };  //##############################################################################  // Alarmas de las tareas periódicas  ALARM AlarmaSonar  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = ChequeoSonar;  };  AUTOSTART = TRUE  {  ALARMTIME = 10;  CYCLETIME = 50;  APPMODE = sample\_appmode1;  };  };    ALARM AlarmaDistancia  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = ChequeoDistancia;  };  AUTOSTART = TRUE  {  ALARMTIME = 10;  CYCLETIME = 75;  APPMODE = sample\_appmode1;  };  };    ALARM AlarmaPulsadores  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = ChequeoPulsadores;  };  AUTOSTART = TRUE  {  ALARMTIME = 10;  CYCLETIME = 10;  APPMODE = sample\_appmode1;  };  };  }; |

#### Fichero C

|  |
| --- |
| #include "kernel.h"  #include "kernel\_id.h"  #include "ecrobot\_interface.h"  // Sensores  #define SONAR\_PORT NXT\_PORT\_S2  #define PULSADOR1\_PORT NXT\_PORT\_S1  #define PULSADOR2\_PORT NXT\_PORT\_S4  // Actuadores  #define MOT\_I NXT\_PORT\_C  #define MOT\_D NXT\_PORT\_A  /\*--------------------------------------------------------------------------\*/  /\* OSEK declarations \*/  /\*--------------------------------------------------------------------------\*/  DeclareTask(VelBaja);  DeclareTask(VelMedia);  DeclareTask(VelAlta);  DeclareTask(Retroceso);  DeclareTask(CambioDir);  DeclareTask(PararMotor);  DeclareTask(ChequeoDistancia);  DeclareTask(ChequeoSonar);  DeclareTask(ChequeoPulsadores);  DeclareTask(PantallaLCD);  DeclareCounter(SysTimerCnt);  // Zona de variables globales  int sensor = 0;  int pulsado = 0;  int atras = 0;  void ecrobot\_device\_initialize()  {  // Inicializar los sensores activos  ecrobot\_init\_sonar\_sensor(SONAR\_PORT);  }  void ecrobot\_device\_terminate()  {  // Finalizar los sensores activos  ecrobot\_term\_sonar\_sensor(SONAR\_PORT);  }  /\*--------------------------------------------------------------------------\*/  /\* Definitions \*/  /\*--------------------------------------------------------------------------\*/  // Funcion para mover el motor izq. o der. una distancia 'y' a una velocidad 'z'  void mover\_motor(U32 motor, int distancia, int velocidad)  {  int grados = (360\*distancia)/18;  if(velocidad < 0){  grados = grados\*(-1);  }  int ref = nxt\_motor\_get\_count(motor) + grados;  nxt\_motor\_set\_speed(motor, velocidad, 1);  if(velocidad < 0){  while(nxt\_motor\_get\_count(motor) > ref);  }  else{  while(nxt\_motor\_get\_count(motor) < ref);  }  nxt\_motor\_set\_speed(motor, 0, 1);  }  /\*--------------------------------------------------------------------------\*/  /\* Function to be invoked from a category 2 interrupt \*/  /\*--------------------------------------------------------------------------\*/  void user\_1ms\_isr\_type2(){  StatusType ercd;  ercd = SignalCounter(SysTimerCnt);  if(ercd != E\_OK){  ShutdownOS(ercd);  }  }  //##############################################################################  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : VelBaja \*/  /\* Priority: 4 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que pone los servomotores a un cuarto de velocidad. \*/  /\*--------------------------------------------------------------------------\*/  TASK(VelBaja)  {  // Pone los motores a un cuarto de velocidad  nxt\_motor\_set\_speed(MOT\_I, 25, 1);  nxt\_motor\_set\_speed(MOT\_D, 22, 1);  // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : VelMedia \*/  /\* Priority: 3 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que pone los servomotores a la mitad de velocidad. \*/  /\*--------------------------------------------------------------------------\*/  TASK(VelMedia)  {  // Pone los motores a la mitad de velocidad  nxt\_motor\_set\_speed(MOT\_I, 50, 1);  nxt\_motor\_set\_speed(MOT\_D, 48, 1);  // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : VelAlta \*/  /\* Priority: 2 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que pone los servomotores a maxima velocidad. \*/  /\*--------------------------------------------------------------------------\*/  TASK(VelAlta)  {  atras = 0;  // Pone los motores a maxima velocidad  nxt\_motor\_set\_speed(MOT\_I, 100, 1);  nxt\_motor\_set\_speed(MOT\_D, 98, 1);  // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : Retroceso \*/  /\* Priority: 7 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que mueve los servomotores hacia atras. \*/  /\*--------------------------------------------------------------------------\*/  TASK(Retroceso)  {  // Poner los motores para retroceder  nxt\_motor\_set\_speed(MOT\_I, -25, 1);  nxt\_motor\_set\_speed(MOT\_D, -24, 1);  // Esperar un segundo  systick\_wait\_ms(1000);  // Parar los motores  nxt\_motor\_set\_speed(MOT\_I, 0, 1);  nxt\_motor\_set\_speed(MOT\_D, 0, 1);  // Cambio de tarea para realizar el giro  ChainTask(CambioDir);  // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : CambioDir \*/  /\* Priority: 6 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: 1 \*/  /\* Objective: Tarea que realiza un giro de 90 grados. \*/  /\*--------------------------------------------------------------------------\*/  TASK(CambioDir)  {  // Realiza un giro de 90 grados  mover\_motor(MOT\_I, 19, 50);  atras = 0;  // Cambia de tarea a maxima velocidad  ChainTask(VelAlta);  // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : PararMotor \*/  /\* Priority: 5 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que para los servomotores. \*/  /\*--------------------------------------------------------------------------\*/  TASK(PararMotor)  {  // Para los motores  nxt\_motor\_set\_speed(MOT\_I, 0, 1);  nxt\_motor\_set\_speed(MOT\_D, 0, 1);  // Termina la tarea actual  TerminateTask();  }  //##############################################################################  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : ChequeoDistancia \*/  /\* Priority: 10 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea periodica que chequea a que distancia se encuentra. \*/  /\*--------------------------------------------------------------------------\*/  TASK(ChequeoDistancia)  {  if(sensor == -1){  ActivateTask(PararMotor);  }  // Si la distancia es mayor a 120 cm los motores se encuentran a maxima velocidad  else if(sensor > 120){  ActivateTask(VelAlta);  }  // Si la distancia se encuentra entre 120 cm y 40 cm se reduce la potencia de los motores a la mitad  else if((sensor <= 120) && (sensor > 40)){  ActivateTask(VelMedia);  }  // Si la distancia es menor que 40 cm se reduce la potencia de los motores a la mitad  else if(sensor <= 40){  ActivateTask(VelBaja);  }  // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : ChequeoSonar \*/  /\* Priority: 10 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea periodica que captura el valor del sonar. \*/  /\*--------------------------------------------------------------------------\*/  TASK(ChequeoSonar)  {  // Se coge el valor que capta el sonar  sensor = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);  // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : ChequeoPulsadores \*/  /\* Priority: 11 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que chequea si los pulsadores estan activos. \*/  /\*--------------------------------------------------------------------------\*/  TASK(ChequeoPulsadores)  {  // Si alguno de los dos pulsadores se encuentra activo y no se encuentra en  // el momento de retroceder, se empieza a retroceder  if((ecrobot\_get\_touch\_sensor(PULSADOR1\_PORT) == 1 || ecrobot\_get\_touch\_sensor(PULSADOR2\_PORT) == 1) && (atras == 0)){  if(!pulsado){  pulsado = 1;  atras = 1;  ActivateTask(Retroceso);  }  }  else{  pulsado = 0;  }  // Termina la tarea actual  TerminateTask();  }  //##############################################################################  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : PantallaLCD \*/  /\* Priority: 1 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: 1 \*/  /\* Objective: Tarea que muestra los valores de estado por la LCD \*/  /\*--------------------------------------------------------------------------\*/  TASK(PantallaLCD)  {  // Muestra por la pantalla del nxt los datos de los servomotores  ecrobot\_status\_monitor("");  // Termina la tarea actual  TerminateTask();  } |

## Práctica 2

### Apartado A

#### Fichero OIL

|  |
| --- |
| #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 LeerPulsadores  {  AUTOSTART = FALSE;  PRIORITY = 10;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK LeerSonar  {  AUTOSTART = FALSE;  PRIORITY = 9;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK Comparar  {  AUTOSTART = FALSE;  PRIORITY = 6;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK Mirar  {  AUTOSTART = FALSE;  PRIORITY = 7;  ACTIVATION = 1;  SCHEDULE = NON;  STACKSIZE = 512;  };    TASK Avanzar  {  AUTOSTART = FALSE;  PRIORITY = 5;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK PantallaLCD  {  AUTOSTART = FALSE;  PRIORITY = 15;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };  COUNTER SysTimerCnt  {  MINCYCLE = 1;  MAXALLOWEDVALUE = 10000;  TICKSPERBASE = 1; /\* One tick is equal to 1msec \*/  };  ALARM AlarmaLCD  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = PantallaLCD;  };  AUTOSTART = TRUE  {  ALARMTIME = 10;  CYCLETIME = 100; /\* executed every 500msec \*/  APPMODE = sample\_appmode1;  };  };  ALARM AlarmaSonar  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = LeerSonar;  };  AUTOSTART = TRUE  {  ALARMTIME = 1;  CYCLETIME = 100;  APPMODE = sample\_appmode1;  };  };    ALARM AlarmaPulsador  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = LeerPulsadores;  };  AUTOSTART = TRUE  {  ALARMTIME = 1;  CYCLETIME = 50;  APPMODE = sample\_appmode1;  };  };  }; |

#### Fichero C

|  |
| --- |
| #include "kernel.h"  #include "kernel\_id.h"  #include "ecrobot\_interface.h"  // Sensores  #define SONAR\_PORT NXT\_PORT\_S1  #define PUL\_PARO\_PORT NXT\_PORT\_S4  #define PUL\_CHOQUE\_PORT NXT\_PORT\_S2  // Actuadores  #define MOT\_I NXT\_PORT\_C  #define MOT\_D NXT\_PORT\_A  // Otros  #define VEL\_GIRO 50  #define VEL\_DESP 75  /\*--------------------------------------------------------------------------\*/  /\* OSEK declarations \*/  /\*--------------------------------------------------------------------------\*/  DeclareTask(LeerPulsadores);  DeclareTask(LeerSonar);  DeclareTask(Comparar);  DeclareTask(Mirar);  DeclareTask(Avanzar);  DeclareTask(PantallaLCD);  DeclareCounter(SysTimerCnt);  // Zona de variables globales  int sensor = 0; // almacena la medida en bruto del sonar  int desact\_comparar = 0; // flag que desactiva temporalmente la tarea de comparacion  int choque = 0; // flag que almacena si el pulsador de choque esta activado o no  int parada = 0; // flag que almecena si el pulsador de parada esta activado o no  // Funciones del sistema  void ecrobot\_device\_initialize()  {  // Inicializar los sensores activos  ecrobot\_init\_sonar\_sensor(SONAR\_PORT);  }  void ecrobot\_device\_terminate()  {  // Finalizar los sensores activos  ecrobot\_term\_sonar\_sensor(SONAR\_PORT);  }  /\*--------------------------------------------------------------------------\*/  /\* Function to be invoked from a category 2 interrupt \*/  /\*--------------------------------------------------------------------------\*/  void user\_1ms\_isr\_type2(){  StatusType ercd;  ercd = SignalCounter(SysTimerCnt);    if(ercd != E\_OK){  ShutdownOS(ercd);  }  }  /\*--------------------------------------------------------------------------\*/  /\* Definitions \*/  /\*--------------------------------------------------------------------------\*/  // Esta funcion detiene ambos servos para parar el robot  void parar\_motor()  {  // parar motor  nxt\_motor\_set\_speed(MOT\_I, 0, 1);  nxt\_motor\_set\_speed(MOT\_D, 0, 1);  }  // Esta funcion toma tres muestras del sonar y se queda con la media  int obtener\_distancia()  {  int sensor1, sensor2, sensor3, sensor;  sensor1 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);  sensor2 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);  sensor3 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);    sensor = (sensor1+sensor2+sensor3)/3;  // if(sensor == 255) sensor = 0;  return (sensor);  }  // Funcion para mover el motor x la distancia en mm y a la velocidad z  void mover\_motor(U32 motor, int distancia, int velocidad)  {  int grados = (360\*distancia)/188;  if(velocidad < 0){  grados = grados\*(-1);  }  int ref = nxt\_motor\_get\_count(motor)+grados;  nxt\_motor\_set\_speed(motor, velocidad, 1);    if(velocidad < 0){  while(nxt\_motor\_get\_count(motor) > ref);  }  else{  while(nxt\_motor\_get\_count(motor) < ref);  }    nxt\_motor\_set\_speed(motor, 0, 1);  }  // Funcion para mover dos motores x la distancia y a la velocidad z  void mover\_motores(U32 motor1, U32 motor2, int dist1, int dist2, int vel1, int vel2)  {  int llega1 = 0; // flag para determinar el final del movimiento del motor 1  int llega2 = 0; // flag para determinar el final del movimiento del motor 1  int dir1 = 1; //1 -> adelante, -1 -> atras, 0 -> parado  int dir2 = 1; //1 -> adelante, -1 -> atras, 0 -> parado    // Obtenemos los grados que tienen que girar las ruedas  int grados1 = (360\*dist1)/188;  int grados2 = (360\*dist2)/188;    // Esta conversion permite especificar velocidades negativas y positivas  if(vel1 < 0){  grados1 = -grados1;  dir1 = -dir1;  }  if(vel2 < 0){  grados2 = -grados2;  dir2 = -dir2;  }    // Calculamos angulos de referencia  int ref1 = nxt\_motor\_get\_count(motor1) + grados1;  int ref2 = nxt\_motor\_get\_count(motor2) + grados2;    // Iniciamos los motores  nxt\_motor\_set\_speed(motor1, vel1, 1);  nxt\_motor\_set\_speed(motor2, vel2, 1);    // Comprobamos si hemos llegado al angulo de referencia y si es asi  // paramos los motores  while((llega1 != 1)||(llega2 != 1)){  if(dir1 < 0){  if(nxt\_motor\_get\_count(motor1) <= ref1){  llega1 = 1;  nxt\_motor\_set\_speed(motor1,0,1);  }  }  else{  if(nxt\_motor\_get\_count(motor1) >= ref1){  llega1 = 1;  nxt\_motor\_set\_speed(motor1,0,1);  }  }  if(dir2 < 0){  if(nxt\_motor\_get\_count(motor2) <= ref2){  llega2 = 1;  nxt\_motor\_set\_speed(motor2,0,1);  }  }  else{  if(nxt\_motor\_get\_count(motor2) >= ref2){  llega2 = 1;  nxt\_motor\_set\_speed(motor2,0,1);  }  }  }  }  //##############################################################################  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : LeerPulsadores \*/  /\* Priority: 10 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que lee ambos pulsadores (choque y parada y actua en \*/  /\* consecuencia. \*/  /\*--------------------------------------------------------------------------\*/  TASK(LeerPulsadores)  {  // Cuando detecte el sensor de parada se activa el flag parada y se paran los motores  if(ecrobot\_get\_touch\_sensor(PUL\_PARO\_PORT) == 1 || parada == 1){  desact\_comparar = 1;  parada = 1;  parar\_motor();  }  // Comprobamos si ha habido un choque y asignamos el valor correspondiente al flag  if(ecrobot\_get\_touch\_sensor(PUL\_CHOQUE\_PORT) == 1){  choque = 1;  }  else{  choque = 0;  }    // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : LeerSonar \*/  /\* Priority: 9 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que toma tres muestras del sonar y se queda con la \*/  /\* media. \*/  /\*--------------------------------------------------------------------------\*/  TASK(LeerSonar)  {  int sensor1, sensor2, sensor3;  sensor1 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);  sensor2 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);  sensor3 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);    sensor = (sensor1+sensor2+sensor3)/3;    // Activa la comparacion  ChainTask(Comparar);  // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : Comparar \*/  /\* Priority: 6 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que realiza las tareas pertinentes segun la distancia \*/  /\* del robot a la pared. \*/  /\*--------------------------------------------------------------------------\*/  TASK(Comparar)  {  // Comprobamos que no se este mirando la distancia maxima  if(desact\_comparar == 0){  if(sensor == -1){  // Aqui podriamos actuar en caso de error del sonar  }  // Si estamos cerca de un obstaculo paramos y miramos una nueva ruta  else if(sensor <= 25){  desact\_comparar = 1;  parar\_motor();  ActivateTask(Mirar);  }  // Si nos hemos chocado nos movemos hacia atras y miramos una nueva ruta  else if(choque == 1){  desact\_comparar = 1;  parar\_motor();  mover\_motores(MOT\_I,MOT\_D,75,75,-VEL\_GIRO,-VEL\_GIRO);    // Buscamos nueva ruta  ActivateTask(Mirar);  }  // Si no hay ningun problema seguimos avanzando  else{  ActivateTask(Avanzar);  }  }    // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : Mirar \*/  /\* Priority: 7 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: NON \*/  /\* Objective: Tarea que busca la distancia maxima en un radio de 180º \*/  /\* delante del robot. \*/  /\*--------------------------------------------------------------------------\*/  TASK(Mirar)  {  int dist = 0;  int nuevadist = 0;  int max\_dist = 0;  int ind\_giro = 0;  int i;    // se gira 90 grados a la izquierda  mover\_motores(MOT\_I,MOT\_D,90,90,-VEL\_GIRO,VEL\_GIRO);    // chequear la distancia maxima  for(i=0;i<18;i++){  nuevadist = obtener\_distancia();  if(nuevadist != 255){  dist = nuevadist;  }  display\_clear(0);  display\_goto\_xy(0,0);  display\_int(dist,3);  display\_update();    if(dist >= max\_dist){  max\_dist = dist;  ind\_giro = i;  }  mover\_motores(MOT\_I,MOT\_D,10,10,VEL\_GIRO,-VEL\_GIRO);  }    // volver a la posicion donde se encontraba la distancia maxima  mover\_motores(MOT\_I,MOT\_D,190-(10\*ind\_giro),190-(10\*ind\_giro),-VEL\_GIRO,VEL\_GIRO);    // se activa la comprobacion otra vez  desact\_comparar = 0;  // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : Avanzar \*/  /\* Priority: 5 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que pone en marcha los motores del robot \*/  /\*--------------------------------------------------------------------------\*/  TASK(Avanzar)  {  nxt\_motor\_set\_speed(MOT\_I, VEL\_DESP, 1);  nxt\_motor\_set\_speed(MOT\_D, VEL\_DESP-1, 1);    // Termina la tarea actual  TerminateTask();  }  //##############################################################################  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : PantallaLCD \*/  /\* Priority: 1 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que muestra los valores de estado por la LCD \*/  /\*--------------------------------------------------------------------------\*/  TASK(PantallaLCD)  {  ecrobot\_status\_monitor("");    // Termina la tarea actual  TerminateTask();  } |

### Apartado B

#### Fichero OIL (Servidor)

|  |
| --- |
| #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 LeerPulsadores  {  AUTOSTART = FALSE;  PRIORITY = 10;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK LeerSonar  {  AUTOSTART = FALSE;  PRIORITY = 9;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };  TASK Comparar  {  AUTOSTART = FALSE;  PRIORITY = 6;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK Mirar  {  AUTOSTART = FALSE;  PRIORITY = 7;  ACTIVATION = 1;  SCHEDULE = NON;  STACKSIZE = 512;  };    TASK Avanzar  {  AUTOSTART = FALSE;  PRIORITY = 5;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK PantallaLCD  {  AUTOSTART = FALSE;  PRIORITY = 15;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };  COUNTER SysTimerCnt  {  MINCYCLE = 1;  MAXALLOWEDVALUE = 10000;  TICKSPERBASE = 1; /\* One tick is equal to 1msec \*/  };  ALARM AlarmaLCD  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = PantallaLCD;  };  AUTOSTART = TRUE  {  ALARMTIME = 10;  CYCLETIME = 100; /\* executed every 500msec \*/  APPMODE = sample\_appmode1;  };  };  ALARM AlarmaSonar  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = LeerSonar;  };  AUTOSTART = TRUE  {  ALARMTIME = 1;  CYCLETIME = 100;  APPMODE = sample\_appmode1;  };  };    ALARM AlarmaPulsador  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = LeerPulsadores;  };  AUTOSTART = TRUE  {  ALARMTIME = 1;  CYCLETIME = 50;  APPMODE = sample\_appmode1;  };  };  }; |

#### Fichero C (Servidor)

|  |
| --- |
| #include "kernel.h"  #include "kernel\_id.h"  #include "ecrobot\_interface.h"  // Sensores  #define SONAR\_PORT NXT\_PORT\_S1  #define PUL\_PARO\_PORT NXT\_PORT\_S4  #define PUL\_CHOQUE\_PORT NXT\_PORT\_S2  // Actuadores  #define MOT\_I NXT\_PORT\_C  #define MOT\_D NXT\_PORT\_B  // Otros  #define VEL\_GIRO 50  #define VEL\_DESP 50  /\*--------------------------------------------------------------------------\*/  /\* OSEK declarations \*/  /\*--------------------------------------------------------------------------\*/  DeclareTask(LeerPulsadores);  DeclareTask(LeerSonar);  DeclareTask(Comparar);  DeclareTask(Mirar);  DeclareTask(Avanzar);  DeclareTask(PantallaLCD);  DeclareCounter(SysTimerCnt);  // Dirección MAC del robot (cambiar segun el cliente)  const U8 bd\_addr[7] = {0x00,0x16,0x53,0x01,0x86,0x58,0x00}; // carlos  // const U8 bd\_addr[7] = {0x00,0x16,0x53,0x01,0x83,0x78,0x00}; // miguel  // Zona de variables globales  int sensor = 0; // almacena la medida en bruto del sonar  int desact\_comparar = 0; // flag que desactiva temporalmente la tarea de comparacion  int giro = 0;  int giro\_ant = 0;  int distancia = 0;  int choque = 0; // flag que almacena si el pulsador de choque esta activado o no  int parada = 0; // flag que almecena si el pulsador de parada esta activado o no  // Globales para log de movimientos  int pulsado = 0; // flag que controla la pulsacion para enviar los movimientos al cliente  int paso = 0; // variable que almacena el numero de pasos que ha dado el servidor para salir del laberinto  int listamovimientos[500][2]; // matriz que almacena el tipo de  // movimiento (0 -> adelante, 1-> giro  // y los grado de movimiento o giro  void ecrobot\_device\_initialize()  {  // Inicializar los sensores activos  ecrobot\_init\_sonar\_sensor(SONAR\_PORT);  // Inicializar modulo bluetooth  ecrobot\_init\_bt\_master(bd\_addr,"STR");  }  void ecrobot\_device\_terminate()  {  // Finalizar los sensores activos  ecrobot\_term\_sonar\_sensor(SONAR\_PORT);  // Finalizar conexion bluetooth  ecrobot\_term\_bt\_connection();  }  /\*--------------------------------------------------------------------------\*/  /\* Function to be invoked from a category 2 interrupt \*/  /\*--------------------------------------------------------------------------\*/  void user\_1ms\_isr\_type2(){  StatusType ercd;  ercd = SignalCounter(SysTimerCnt);    if(ercd != E\_OK){  ShutdownOS(ercd);  }  }  /\*--------------------------------------------------------------------------\*/  /\* Definitions \*/  /\*--------------------------------------------------------------------------\*/  // Esta funcion detiene ambos servos para parar el robot  void parar\_motor()  {  // parar motor  nxt\_motor\_set\_speed(MOT\_I, 0, 1);  nxt\_motor\_set\_speed(MOT\_D, 0, 1);  }  // Esta funcion toma tres muestras del sonar y se queda con la media  int obtener\_distancia()  {  int sensor1, sensor2, sensor3, sensor;  // Cuando detecte el sensor de sonar que hay una distancia  // de 1m se reduce la potencia de los motores a la mitad  sensor1 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);  sensor2 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);  sensor3 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);    sensor = (sensor1+sensor2+sensor3)/3;  // if(sensor == 255) sensor = 0;  return (sensor);  }  // Funcion para mover el motor x la distancia en mm y a la velocidad z  void mover\_motor(U32 motor, int distancia, int velocidad)  {  int grados = (360\*distancia)/188;  if(velocidad < 0){  grados = grados\*(-1);  }  int ref = nxt\_motor\_get\_count(motor)+grados;  nxt\_motor\_set\_speed(motor, velocidad, 1);    if(velocidad < 0){  while(nxt\_motor\_get\_count(motor) > ref);  }  else{  while(nxt\_motor\_get\_count(motor) < ref);  }    nxt\_motor\_set\_speed(motor, 0, 1);  }  // Funcion para mover dos motores x la distancia y a la velocidad z  void mover\_motores(U32 motor1, U32 motor2, int dist1, int dist2, int vel1, int vel2)  {  int llega1 = 0; // flag para determinar el final del movimiento del motor 1  int llega2 = 0; // flag para determinar el final del movimiento del motor 1  int dir1 = 1; //1 -> adelante, -1 -> atras, 0 -> parado  int dir2 = 1; //1 -> adelante, -1 -> atras, 0 -> parado    // Obtenemos los grados que tienen que girar las ruedas  int grados1 = (360\*dist1)/188;  int grados2 = (360\*dist2)/188;    // Esta conversion permite especificar velocidades negativas y positivas  if(vel1 < 0){  grados1 = -grados1;  dir1 = -dir1;  }  if(vel2 < 0){  grados2 = -grados2;  dir2 = -dir2;  }    // Calculamos angulos de referencia  int ref1 = nxt\_motor\_get\_count(motor1)+grados1;  int ref2 = nxt\_motor\_get\_count(motor2)+grados2;    // Iniciamos los motores  nxt\_motor\_set\_speed(motor1, vel1, 1);  nxt\_motor\_set\_speed(motor2, vel2, 1);    // Comprobamos si hemos llegado al angulo de referencia y si es asi  // paramos los motores  while((llega1 != 1) || (llega2 != 1)){  if(dir1 < 0){  if(nxt\_motor\_get\_count(motor1) <= ref1){  llega1 = 1;  nxt\_motor\_set\_speed(motor1,0,1);  }  }  else{  if(nxt\_motor\_get\_count(motor1) >= ref1){  llega1 = 1;  nxt\_motor\_set\_speed(motor1,0,1);  }  }  if(dir2 < 0){  if(nxt\_motor\_get\_count(motor2) <= ref2){  llega2 = 1;  nxt\_motor\_set\_speed(motor2,0,1);  }  }  else{  if(nxt\_motor\_get\_count(motor2) >= ref2){  llega2 = 1;  nxt\_motor\_set\_speed(motor2,0,1);  }  }  }  }  // Funcion que envia la matriz de movimientos por filas  void enviar\_movimientos\_bt(void)  {  int i,j;  U8 nmovs[1], mov[5];  nmovs[0] = (U8) paso;  ecrobot\_send\_bt\_packet(nmovs, 1);  for(i=0;i<paso;i++){  mov[0] = (U8) listamovimientos[i][0];  for(j=4;j>0;j--){  mov[j] = listamovimientos[i][1]%10;  listamovimientos[i][1] = listamovimientos[i][1]/10;  }  while(ecrobot\_send\_bt\_packet(mov,5) != 5);  }  }  //##############################################################################  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : LeerPulsadores \*/  /\* Priority: 10 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que lee ambos pulsadores (choque y parada) y actua en \*/  /\* consecuencia. \*/  /\*--------------------------------------------------------------------------\*/  TASK(LeerPulsadores)  {  int i;    // Cuando detecte el sensor de pulsacion de paro, almacena el ultimo  // movimiento y muestra los movimientos por pantalla  if(ecrobot\_get\_touch\_sensor(PUL\_PARO\_PORT) == 1 || parada == 1){  desact\_comparar = 1;  parada = 1;  parar\_motor();    if(pulsado == 0){  pulsado = 1;  // Almacena el ultimo movimiento  listamovimientos[paso][0] = 0;  listamovimientos[paso][1] = nxt\_motor\_get\_count(MOT\_I);  paso++;    // Muestra la lista de movimientos a enviar  display\_clear(0);  for(i=0;i<paso;i++){  display\_goto\_xy(0,i);  display\_string("PASO ");  display\_int(i,1);  display\_string(": ");  display\_int(listamovimientos[i][0],1);  display\_string(" ");  display\_int(listamovimientos[i][1],3);  }  display\_update();    // envia los movimientos por bluetooth al cliente  enviar\_movimientos\_bt();  }  }  // Comprobamos si ha habido un choque y asignamos el valor correspondiente al flag  if(ecrobot\_get\_touch\_sensor(PUL\_CHOQUE\_PORT) == 1){  choque = 1;  }  else{  choque = 0;  }    // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : LeerSonar \*/  /\* Priority: 9 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que toma tres muestras del sonar y se queda con la media\*/  /\*--------------------------------------------------------------------------\*/  TASK(LeerSonar)  {  int sensor1, sensor2, sensor3;    // Cuando detecte el sensor de sonar que hay una distancia  // de 1m se reduce la potencia de los motores a la mitad  sensor1 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);  sensor2 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);  sensor3 = ecrobot\_get\_sonar\_sensor(SONAR\_PORT);  systick\_wait\_ms(20);    sensor = (sensor1+sensor2+sensor3)/3;    // Activa la comparacion  ChainTask(Comparar);  // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : Comparar \*/  /\* Priority: 6 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que realiza las tareas pertinentes segun la distancia \*/  /\* del robot a la pared. \*/  /\*--------------------------------------------------------------------------\*/  TASK(Comparar)  {  // Comprobamos que no se este mirando la distancia maxima  if(desact\_comparar == 0){  // Si estamos cerca de un obstaculo paramos y miramos una nueva ruta  if(sensor <= 25){  // Almacenamos el paso anterior  listamovimientos[paso][0] = 0;  listamovimientos[paso][1] = nxt\_motor\_get\_count(MOT\_I);  paso++;  desact\_comparar = 1;  parar\_motor();    // Buscamos nueva ruta  ActivateTask(Mirar);  nxt\_motor\_set\_count(MOT\_I,0);  nxt\_motor\_set\_count(MOT\_D,0);  }  // Si nos hemos chocado nos movemos hacia atras y miramos una nueva ruta  else if(choque == 1){  desact\_comparar = 1;  parar\_motor();  mover\_motores(MOT\_I,MOT\_D,75,75,-VEL\_GIRO,-VEL\_GIRO);    // Almacenamos el movimiento  listamovimientos[paso][0] = 0;  listamovimientos[paso][1] = nxt\_motor\_get\_count(MOT\_I);  nxt\_motor\_set\_count(MOT\_I,0);  nxt\_motor\_set\_count(MOT\_D,0);  paso++;    // Buscamos nueva ruta  ActivateTask(Mirar);  }  else{  ActivateTask(Avanzar);  }  }    // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : Mirar \*/  /\* Priority: 7 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: NON \*/  /\* Objective: Tarea que busca la distancia maxima en un radio de 180º \*/  /\* delante del robot. \*/  /\*--------------------------------------------------------------------------\*/  TASK(Mirar)  {  int dist = 0;  int nuevadist = 0;  int max\_dist = 0;  int ind\_giro = 0;  int i;    // se gira 90 grados a la izquierda  mover\_motores(MOT\_I,MOT\_D,90,90,-VEL\_GIRO,VEL\_GIRO);    // chequear la distancia maxima  for(i=0;i<17;i++){  nuevadist = obtener\_distancia();  if(nuevadist != 255){  dist = nuevadist;  }  display\_clear(0);  display\_goto\_xy(0,0);  display\_int(dist,3);  display\_update();    if(dist > max\_dist){  max\_dist = dist;  ind\_giro = i;  }  mover\_motores(MOT\_I,MOT\_D,10,10,VEL\_GIRO,-VEL\_GIRO);  }    // volver a la posicion donde se encontraba la distancia maxima  mover\_motores(MOT\_I,MOT\_D,190-(10\*ind\_giro),190-(10\*ind\_giro),-VEL\_GIRO,VEL\_GIRO);    listamovimientos[paso][0] = 1;  // Se le suma 90 para poder pasar el número positivo por bluetooth  listamovimientos[paso][1] = (90-(180-(10\*ind\_giro)))+90;  paso++;    // se activa la comprobacion otra vez  desact\_comparar = 0;  // Termina la tarea actual  TerminateTask();  }  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : Avanzar \*/  /\* Priority: 5 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que pone en marcha los motores del robot. \*/  /\*--------------------------------------------------------------------------\*/  TASK(Avanzar)  {  nxt\_motor\_set\_speed(MOT\_I, VEL\_DESP, 1);  nxt\_motor\_set\_speed(MOT\_D, VEL\_DESP-1, 1);    // Termina la tarea actual  TerminateTask();  }  //##############################################################################  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : PantallaLCD \*/  /\* Priority: 1 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que muestra los valores de estado por la LCD. \*/  /\*--------------------------------------------------------------------------\*/  TASK(PantallaLCD)  {  // Tarea que muestra las medidas del robot por pantalla  ecrobot\_status\_monitor("");    // Termina la tarea actual  TerminateTask();  } |

#### Fichero OIL (Cliente)

|  |
| --- |
| #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 Avanzar  {  AUTOSTART = TRUE  {  APPMODE = sample\_appmode1;  };  PRIORITY = 5;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };    TASK PantallaLCD  {  AUTOSTART = FALSE;  PRIORITY = 15;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };  COUNTER SysTimerCnt  {  MINCYCLE = 1;  MAXALLOWEDVALUE = 10000;  TICKSPERBASE = 1; /\* One tick is equal to 1msec \*/  };  ALARM AlarmaLCD  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = PantallaLCD;  };  AUTOSTART = TRUE  {  ALARMTIME = 10;  CYCLETIME = 100; /\* executed every 500msec \*/  APPMODE = sample\_appmode1;  };  };  }; |

#### Fichero C (Cliente)

|  |
| --- |
| #include "kernel.h"  #include "kernel\_id.h"  #include "ecrobot\_interface.h"  // Sensores  #define SONAR\_PORT NXT\_PORT\_S1  #define PUL\_PARO\_PORT NXT\_PORT\_S4  #define PUL\_CHOQUE\_PORT NXT\_PORT\_S2  // Actuadores  #define MOT\_I NXT\_PORT\_C  #define MOT\_D NXT\_PORT\_B  // Otros  #define VEL\_GIRO 50  #define VEL\_DESP 50  /\*--------------------------------------------------------------------------\*/  /\* OSEK declarations \*/  /\*--------------------------------------------------------------------------\*/  DeclareTask(Avanzar);  DeclareTask(PantallaLCD);  DeclareCounter(SysTimerCnt);  // Dirección MAC del robot  const U8 bd\_addr[7] = {0x00,0x16,0x53,0x01,0x86,0x58,0x00};  // Zona de variables globales  int sensor = 0;  int desact\_comparar = 0;  int giro = 0;  int giro\_ant = 0;  int distancia = 0;  int choque = 0;  int parada = 0;  // Globales para log de movimientos  int pulsado = 0;  int paso = 0;  int inicio\_movimiento = 0;  int rev\_inicio;  int rev\_fin;  int listamovimientos[500][2];  void ecrobot\_device\_initialize()  {  // Inicializar modulo bluetooth en modo slave  ecrobot\_init\_bt\_slave("STR");  }  void ecrobot\_device\_terminate()  {  // Finalizar conexion bluetooth  ecrobot\_term\_bt\_connection();  }  /\*--------------------------------------------------------------------------\*/  /\* Function to be invoked from a category 2 interrupt \*/  /\*--------------------------------------------------------------------------\*/  void user\_1ms\_isr\_type2(){  StatusType ercd;  ercd = SignalCounter(SysTimerCnt);    if(ercd != E\_OK){  ShutdownOS(ercd);  }  }  /\*--------------------------------------------------------------------------\*/  /\* Definitions \*/  /\*--------------------------------------------------------------------------\*/  // funcion para detener los servomotores del robot  void parar\_motor()  {  // parar motor  nxt\_motor\_set\_speed(MOT\_I, 0, 1);  nxt\_motor\_set\_speed(MOT\_D, 0, 1);  }  // Funcion para mover el motor x la distancia en mm y a la velocidad z  void mover\_motor(U32 motor, int distancia, int velocidad)  {  int grados = (360\*distancia)/188;  if(velocidad < 0){  grados = grados\*(-1);  }  int ref = nxt\_motor\_get\_count(motor)+grados;  nxt\_motor\_set\_speed(motor, velocidad, 1);    if(velocidad < 0){  while(nxt\_motor\_get\_count(motor) > ref);  }  else{  while(nxt\_motor\_get\_count(motor) < ref);  }    nxt\_motor\_set\_speed(motor, 0, 1);  }  // Funcion para mover dos motores x la distancia y a la velocidad z  void mover\_motores(U32 motor1, U32 motor2, int dist1, int dist2, int vel1, int vel2)  {  int llega1 = 0;  int llega2 = 0;  int dir1 = 1; //1 -> adelante, -1 -> atras, 0 -> parado  int dir2 = 1;    int grados1 = (360\*dist1)/188;  int grados2 = (360\*dist2)/188;    if(vel1 < 0){  grados1 = -grados1;  dir1 = -dir1;  }  if(vel2 < 0){  grados2 = -grados2;  dir2 = -dir2;  }    int ref1 = nxt\_motor\_get\_count(motor1) + grados1;  int ref2 = nxt\_motor\_get\_count(motor2) + grados2;    nxt\_motor\_set\_speed(motor1, vel1, 1);  nxt\_motor\_set\_speed(motor2, vel2, 1);    while((llega1 != 1)||(llega2 != 1)){  if(dir1 < 0){  if(nxt\_motor\_get\_count(motor1) <= ref1){  llega1 = 1;  nxt\_motor\_set\_speed(motor1,0,1);  }  }  else{  if(nxt\_motor\_get\_count(motor1) >= ref1){  llega1 = 1;  nxt\_motor\_set\_speed(motor1,0,1);  }  }  if(dir2 < 0){  if(nxt\_motor\_get\_count(motor2) <= ref2){  llega2 = 1;  nxt\_motor\_set\_speed(motor2,0,1);  }  }  else{  if(nxt\_motor\_get\_count(motor2) >= ref2){  llega2 = 1;  nxt\_motor\_set\_speed(motor2,0,1);  }  }  }  }  // Funcion para recibir la matriz de movimientos del servidor mediante bluetooth  void recibir\_movimientos\_bt(void)  {  int i;  U8 nmovs[1], mov[5];    nmovs[0] = 0;    // Leemos hasta que nos llegue el dato correcto  while(nmovs[0] == 0){  ecrobot\_read\_bt\_packet(nmovs,1);  };  paso = (int)nmovs[0];    display\_clear(0);  display\_goto\_xy(0,0);  display\_int(paso,2);  display\_update();  // Leemos la matriz por columnas y transformamos los 4 octetos  // en un entero de 32 bits  for(i=0;i<paso;i++){  while(ecrobot\_read\_bt\_packet(mov, 5) != 5);  listamovimientos[i][0] = mov[0];  listamovimientos[i][1] = mov[4] + mov[3]\*10+ mov[2]\*100 + mov[1]\*1000;  if(mov[0]==1){  listamovimientos[i][1] -= 90;  }  }  }  //##############################################################################  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : Avanzar \*/  /\* Priority: 5 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que recibe los movimientos del servidor y sigue dichos \*/  /\* movimientos. \*/  /\*--------------------------------------------------------------------------\*/  TASK(Avanzar)  {  int i = 0;  int opcion = 2;    // recibe los movimientos por bluetooth del servidor  recibir\_movimientos\_bt();  // Muestra la lista de movimientos recibidos  display\_clear(0);  for(i=0;i<paso;i++){  display\_goto\_xy(0,i);  display\_string("PASO ");  display\_int(i,1);  display\_string(": ");  display\_int(listamovimientos[i][0],1);  display\_string(" ");  display\_int(listamovimientos[i][1],3);  }  display\_update();  // Segun el tipo de movimiento se hace la accion correspondiente  for(i=0;i<paso;i++){  opcion = listamovimientos[i][0];  switch(opcion)  {  // Movimiento recto  case 0:  mover\_motores(MOT\_I,MOT\_D,listamovimientos[i][1]/2,listamovimientos[i][1]/2,VEL\_DESP,VEL\_DESP);  break;    // Giro  case 1:  if(listamovimientos[i][1] < 0){  mover\_motores(MOT\_I,MOT\_D,-listamovimientos[i][1],-listamovimientos[i][1],-VEL\_DESP,VEL\_DESP);  }  else{  mover\_motores(MOT\_I,MOT\_D,listamovimientos[i][1]\*2,listamovimientos[i][1]\*2,VEL\_DESP,-VEL\_DESP);  }  break;  }  }    // Termina la tarea actual  TerminateTask();  }  //##############################################################################  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : PantallaLCD \*/  /\* Priority: 1 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: FULL \*/  /\* Objective: Tarea que muestra los valores de estado por la LCD. \*/  /\*--------------------------------------------------------------------------\*/  TASK(PantallaLCD)  {  ecrobot\_status\_monitor("");    // Termina la tarea actual  TerminateTask();  } |

## Práctica 3

### Apartado A

#### Fichero OIL

|  |
| --- |
| #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 Calibrar  {  AUTOSTART = TRUE  {  APPMODE = sample\_appmode1;  };  PRIORITY = 15;  ACTIVATION = 1;  SCHEDULE = NON;  STACKSIZE = 512;  };    TASK Equilibrar  {  AUTOSTART = FALSE;  PRIORITY = 10;  ACTIVATION = 1;  SCHEDULE = NON;  STACKSIZE = 512;  };    TASK PantallaLCD  {  AUTOSTART = FALSE;  PRIORITY = 5;  ACTIVATION = 1;  SCHEDULE = FULL;  STACKSIZE = 512;  };  COUNTER SysTimerCnt  {  MINCYCLE = 1;  MAXALLOWEDVALUE = 10000;  TICKSPERBASE = 1; /\* One tick is equal to 1msec \*/  };  ALARM AlarmaLCD  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = PantallaLCD;  };  AUTOSTART = TRUE  {  ALARMTIME = 10;  CYCLETIME = 100; /\* executed every 500msec \*/  APPMODE = sample\_appmode1;  };  };  ALARM AlarmaEquilibrio  {  COUNTER = SysTimerCnt;  ACTION = ACTIVATETASK  {  TASK = Equilibrar;  };  AUTOSTART = TRUE  {  ALARMTIME = 1;  CYCLETIME = 25;  APPMODE = sample\_appmode1;  };  };  }; |

#### Fichero C

|  |
| --- |
| #include "kernel.h"  #include "kernel\_id.h"  #include "ecrobot\_interface.h"  // Sensores  #define LIGTH\_PORT NXT\_PORT\_S4  // Actuadores  #define MOT\_I NXT\_PORT\_C  #define MOT\_D NXT\_PORT\_B  /\*--------------------------------------------------------------------------\*/  /\* OSEK declarations \*/  /\*--------------------------------------------------------------------------\*/  DeclareTask(Calibrar);  DeclareTask(Equilibrar);  DeclareTask(PantallaLCD);  DeclareCounter(SysTimerCnt);  // Zona de variables globales  int luz\_act = 0;  int dif\_act = 0;  int dif\_ant = 0;  int calibrado = 0;  int equilibrio = 0;  int integral = 0;  /\* Errores que se calculan en cada ejecucion de la tarea equilibrar \*/  int Ep = 0; // Error proporcional  int Ed = 0; // Error derivativo  int Ei = 0; // Error integral  /\* Las siguientes constantes se pueden modificar para calibrar el robot \*/  int Kp = 5; // Constante proporcional (valores recomendados de 0 a 10)  int Ki = 1; // Constante integral (valores recomendados de 0 a 5)  int Kd = 2; // Constante derivativa (valores recomendados de 0 a 3)  int Kai = 1; // Constante de amortiguacion de la integral (valores de 0 a 1)  int Kri = 1; // Constante de reseteado de la integral (valores de 1 a 10)  void ecrobot\_device\_initialize()  {  // Inicializar los sensores activos  ecrobot\_set\_light\_sensor\_active(LIGTH\_PORT);  }  void ecrobot\_device\_terminate()  {  // Finalizar los sensores activos  ecrobot\_set\_light\_sensor\_inactive(LIGTH\_PORT);  }  /\*--------------------------------------------------------------------------\*/  /\* Definitions \*/  /\*--------------------------------------------------------------------------\*/  // Funcion que para y frena ambos motores del robot  void parar\_motor()  {  // parar motor  nxt\_motor\_set\_speed(MOT\_I, 0, 1);  nxt\_motor\_set\_speed(MOT\_D, 0, 1);  }  /\* Funcion que calcula el error proporcional. Este error depende  exclusivamente de la desviacion respecto al punto de equilibrio.  Actua sobre la potencia de los motores de manera directamente  proporcional a la desviacion que sufra el robot en cada instante.  \*/  void calcular\_error\_proporcional()  {  // Ponemos la desviacion actual en la anterior antes de actualizar  dif\_ant = dif\_act;  // Actualizamos la desviacion actual  dif\_act = (luz\_act - equilibrio);    Ep = dif\_act;  }  /\* Funcion que calcula el error derivativo. Este error se basa en el  comportamiento actual y el anterior del robot e intenta predecir  para donde se desviara el robot en el siguiente instante. Permite  adelantarse a desequilibrios del robot y actuar sobre los motores  antes de que sea tarde  \*/  void calcular\_error\_derivativo()  {  Ed = dif\_act - dif\_ant;  }  /\* Funcion que calcula el error integral. Este error se basa en la  historia de errores anteriores a lo largo del tiempo e intenta  predecir la tendencia actual del error. Permite bajar la potencia  a los motores antes de pasarse del punto de equilibrio por inercia  y aumentar la potencia de los mismos si el error no se corrige con  la potencia actual  \*/  void calcular\_error\_integral()  {  integral = Kai\*integral + dif\_act;  if(dif\_act == 0) integral = integral/Kri;  Ei = integral;  }  /\*--------------------------------------------------------------------------\*/  /\* Function to be invoked from a category 2 interrupt \*/  /\*--------------------------------------------------------------------------\*/  void user\_1ms\_isr\_type2(){  StatusType ercd;  ercd = SignalCounter(SysTimerCnt);    if(ercd != E\_OK){  ShutdownOS(ercd);  }  }  //##############################################################################  /\* Esta tarea permite recalibrar el robot. Para ello se intentara dejar el robot  sobre las dos ruedas de la manera mas equilibrada posible y se pulsara el boton  situado en la parte superior izquierda del mismo, tras lo cual el robot comenzara  el proceso de equilibrado automatico. Si se desea recalibrar el robot basta con  pulsar de nuevo el boton para parar el robot y comenzar el proceso de nuevo  \*/  TASK(Calibrar)  {  // Paramos motores  parar\_motor();    // Esperamos a que el usuario suelte el boton  while(ecrobot\_get\_touch\_sensor(NXT\_PORT\_S3) == 1);    // Reseteamos flag calibrado  calibrado = 0;    // Reseteamos errores y contador de integral  Ei = 0;  Ep = 0;  Ed = 0;  integral = 0;    // Mostramos mensaje de advertencia al usuario  display\_clear(0);  display\_goto\_xy(0,0);  display\_string("Calibrando");  display\_goto\_xy(0,1);  display\_string("equilibrar y ");  display\_goto\_xy(0,2);  display\_string("pulsar.");  display\_update();    // Recogemos valor de equilibrio  while(calibrado == 0){  while(ecrobot\_get\_touch\_sensor(NXT\_PORT\_S3) == 1){  calibrado = 1;  equilibrio = ecrobot\_get\_light\_sensor(LIGTH\_PORT);  }  }  TerminateTask();  }  /\* En esta tarea se equilibra el aparato. Para ello se usara un modelo PID  (proporcional, integrador, derivativo) para intentar predecir el  comportamiento del robot y actuar en consecuencia. Los coeficientes  Kp, Ki y Kd permiten ponderar los errores proporcional, integral y  derivativo respectivamente. Actualmente, estos coeficientes estan  calibrados, pero puede que sea necesario recalibrarlos para su uso en  superficies distintas a la usada como prueba.  \*/  TASK(Equilibrar)  {      // Tomamos la medida actual  luz\_act = ecrobot\_get\_light\_sensor(LIGTH\_PORT);    // Calculamos el error respecto al punto de equilibrio  calcular\_error\_proporcional();    // Calculamos el error integral  calcular\_error\_integral();    // Actualizamos la tasa de desviacion  calcular\_error\_derivativo();    // Movemos motores  nxt\_motor\_set\_speed(MOT\_I, Kp\*Ep+Ki\*Ei+Kd\*Ed, 1);  nxt\_motor\_set\_speed(MOT\_D, Kp\*Ep+Ki\*Ei+Kd\*Ed, 1);    // Comprobamos pulsador de calibrado  if(ecrobot\_get\_touch\_sensor(NXT\_PORT\_S3) == 1){  ActivateTask(Calibrar);  }    TerminateTask();  }  //##############################################################################  /\*--------------------------------------------------------------------------\*/  /\* Task information: \*/  /\* ----------------- \*/  /\* Name : PantallaLCD \*/  /\* Priority: 5 \*/  /\* Typ : EXTENDED TASK \*/  /\* Schedule: 1 \*/  /\* Objective: Tarea que muestra los valores de estado por la LCD \*/  /\*--------------------------------------------------------------------------\*/  TASK(PantallaLCD)  {  // Limpiamos pantalla  display\_clear(0);    // Mostramos constantes  display\_goto\_xy(0,0);  display\_string("Kp:");  display\_int(Kp,1);  display\_string(" Ki:");  display\_int(Ki,1);  display\_string(" Kd:");  display\_int(Kd,1);    // Mostramos valores de luz y de equilibrio  display\_goto\_xy(0,1);  display\_string("Luz:");  display\_int(luz\_act,3);  display\_string(" Equ:");  display\_int(equilibrio,3);    // Mostramos errores absolutos y ponderados  display\_goto\_xy(0,2);  display\_string("Ep:");  display\_int(Ep,3);  display\_string(" (");  display\_int(Kp\*Ep,3);  display\_string(")");  display\_goto\_xy(0,3);  display\_string("Ei:");  display\_int(Ei,3);  display\_string(" (");  display\_int(Ki\*Ei,3);  display\_string(")");  display\_goto\_xy(0,4);  display\_string("Ed:");  display\_int(Ed,3);  display\_string(" (");  display\_int(Kd\*Ed,3);  display\_string(")");    // Mostramos movimiento actual de los motores  display\_goto\_xy(0,6);  display\_string("MOT:");  display\_int(Kp\*Ep+Ki\*Ei+Kd\*Ed,4);    // Actualizamos pantalla y terminamos  display\_update();  TerminateTask();  } |