

# **Biocibernética Computacional**

## **Práctica 1**

# NIUTINBOT



Eduardo Jorge Carrasco Hernández  
Jorge Hernández Ramírez  
Emilio Macías Conde  
Enrique Ismael Mendoza Robaina  
Javier Antonio Monzón Santana

# Índice

<b>1. HISTORIA</b> .....	página 3
Introducción .....	página 3
Colaboración con el MIT .....	página 3
Antecedentes y desarrollo .....	página 3
Decisiones de diseño .....	página 4
Lanzamiento del producto .....	página 4
<b>2. DESARROLLO TEÓRICO</b> .....	página 5
2.1. Objetivos .....	página 5
2.2. Conociendo las herramientas .....	página 5
2.2.1. NXT .....	página 6
2.2.2. Sensores .....	página 6
2.2.3. Motores .....	página 7
2.2.4. Lenguaje NXC .....	página 7
<b>3. DESARROLLO PRÁCTICO</b> .....	página 7
3.1. Evolución .....	página 7
3.2. Problemas en la construcción .....	página 11
3.3. Optimizando el algoritmo .....	página 12
<b>4. ¿QUÉ HACE ESPECIAL A NIUTINBOT?</b> .....	página 15
<b>5. RESUMEN Y CONCLUSIONES</b> .....	página 15
<b>6. BIBLIOGRAFÍA Y WEBS DE INTERÉS</b> .....	página 17
<b>7. CÓDIGO FUENTE</b> .....	página 18

## 1. HISTORIA

Legó Mindstorms posee elementos básicos de las teorías robóticas, como la unión de piezas y la programación de acciones, de forma interactiva. Este robot fue comercializado por primera vez en septiembre de 1998.

Comercialmente se publicita como Robotic Invention System, en español Sistema de Invención Robotizado (RIS). También se vende como herramienta educativa, lo que originalmente se pensó en una sociedad entre Legó y el MIT (Instituto Tecnológico de Massachusetts). Esta asociación se emplea como ejemplo de relación entre la industria y la investigación académica, que resulta muy beneficiosa para ambos socios. La versión educativa se llama *Legó Mindstorms for Schools* y viene con un software de programación basado en la GUI de Robolab.

Legó Mindstorms puede ser usado para construir un modelo de sistema integrado con partes electromecánicas controladas por computador. Prácticamente todo puede ser representado con las piezas tal como en la vida real, como un elevador o robots industriales.

### Colaboración con el MIT

La línea Legó Mindstorms nació en una época difícil para Legó, a partir de un acuerdo entre Legó y el MIT. Según este trato, Legó financiaría investigaciones del grupo de epistemología y aprendizaje del MIT sobre cómo aprenden los niños y, a cambio, obtendría nuevas ideas para sus productos, que podría lanzar al mercado sin tener que pagar regalías al MIT. Un fruto de esta colaboración fue el desarrollo del *MIT Programmable Brick (Ladrillo programable)*.

El mentor del grupo, Seymour Papert, era un matemático interesado desde la década de 1960 por la relación entre la ciencia, la adquisición del conocimiento y el desarrollo de la mente infantil. De hecho, el nombre del producto, *Mindstorms*, proviene del título de un libro suyo, llamado *MindStorms: Children, Computers, and Powerful Ideas*, en el que describe sus ideas respecto al empleo de las computadoras como impulsoras del aprendizaje. Papert, uno de los creadores de lenguaje de programación Logo, ampliamente empleado como herramienta para enseñar programación, toma de Jean Piaget la concepción de niño como “constructor de sus propias estructuras mentales”. La lectura de su libro fue lo que impulsó al presidente de Legó a contactar en 1985 con el MIT, pues le hizo pensar que ambos grupos tenían ideas similares sobre el aprendizaje infantil.

Se pensaba que en lugar de instruir al estudiante proporcionándole fórmulas y técnicas, es mejor potenciar el aprendizaje creando un entorno en el que los estudiantes puedan desempeñar actividades propias de ingenieros o inventores como vía para acceder a los principios fundamentales de la ciencia y la técnica; pues de esta manera es como se desarrolla la forma de pensar propia de los científicos, los estudiantes se interesan realmente en su trabajo y *motu proprio* tratan de informarse para resolver los problemas que van encontrando. Así que se concentraron, en palabras de Resnick, en “diseñar cosas que permitan a los estudiantes diseñar cosas”.

### Antecedentes y desarrollo del bloque programable

La línea Mindstorms no fue el primer fruto de la relación entre Legó y el MIT, aunque sí el más exitoso. Con anterioridad, Legó se había interesado por el Lenguaje de programación Logo. Fruto de este interés nació en 1986 Legó TC Logo, creado por Resnick y Steve Ocko, un sistema en el que se programaba en una computadora que estaba conectada por un cable a una construcción Legó que contaba con motores, luces y sensores.

Esta línea de desarrollo continuaría en 1993 con el lanzamiento de Control Lab, de software mejorado.

El paso de programar una computadora que se conectaba a una construcción Lego a programar un bloque de esa construcción era una idea natural que se estudió durante largo tiempo. Desde principios de los años 90 se empezó a investigar esta posibilidad. Sin embargo, el proyecto tuvo que esperar a que el mercado fuera propicio. Por una parte, el coste de la tecnología era demasiado alto en un principio. Por otra, el bloque se programaría desde un computador, y por esas fechas los computadores no estaban tan extendidos como lo estarían ocho años más tarde, lo que afectaría negativamente a la demanda. Hubo que esperar un lustro hasta que las condiciones eran las apropiadas y decidieran empezar seriamente el desarrollo de lo que acabaría siendo el bloque RCX, un bloque de Lego que contaba con un microcontrolador, y que constituye el corazón del producto Mindstorms.

De esta forma las construcciones Lego pasaban de ser estructuras estáticas a máquinas dinámicas que interactúan con el mundo. Por otra parte, mientras que en muchos casos los productos Lego proporcionaban las piezas necesarias para construir algo con un objetivo fijo, como un tren o un puente, lo que permite “aprender haciendo”, en el desarrollo del nuevo bloque se siguió en cambio la filosofía de Papert y Resnick de fomentar el “aprender diseñando”, y tratar de dejar más abiertas las posibilidades.

### **Decisiones de diseño**

El sistema del MIT estaba enfocado a la investigación del proceso de aprendizaje de los niños, lo que hacía que el sistema pudiera ser más caro de producir, pues no se fabricarían muchas unidades, y más frágil, pues siempre habría alguien para arreglarlo en un laboratorio lleno de ingenieros. En cambio, el bloque de Lego estaba destinado a la venta a preadolescentes a gran escala, así que debía ser más asequible y robusto.

Otras de las pautas de diseño fueron:

- El sistema debía ser sencillo para el nuevo usuario y a la vez debía permitir realizar diseños sofisticados para el iniciado.
- El juguete debía poderse emplear de muchas formas diferentes.
- Simplicidad.
- Elección cuidadosa de las cajas negras.
- Poner énfasis en el aprendizaje de la programación. Se desarrolló una versión gráfica, LogoBlocks, en la que las instrucciones estaban representadas por bloques y diagramas, que, en cambio, dificultaba la realización de software complejo. La decisión final del MIT fue implementar una versión híbrida: los programas podrían realizarse de forma gráfica o en código escrito. Sin embargo, Lego sólo incorporaría en un principio a RCX la versión gráfica del entorno de desarrollo y en lugar de Logo desarrollaría un nuevo lenguaje de programación: RCX Code.
- El bloque debía tener un número suficiente de puertos de entrada/salida que pudieran conectarse con diferentes tipos de sensores: de temperatura, de amplitud del sonido o de luz.

### **Lanzamiento del producto**

Este proyecto inicial se ramificaría en varias direcciones: el MIT 6.270 robotics competition kit, los *Crickets* del departamento de epistemología y aprendizaje del MIT y el bloque RCX de Lego. Posteriormente, uno de los desarrolladores del MIT involucrados lanzó al mercado su propio producto, fruto de la experiencia en esta investigación, llamado Handy Cricket.

La primera versión salió al mercado con un precio de \$200 dólares. Incluía 717 componentes, entre ellos el *bloque RCX*. Tras su lanzamiento se vendieron 80.000 unidades en tres meses. Además, la comunidad de aficionados a la robótica, un público adulto, acogió con interés este nuevo producto. Este interés imprevisto del público adulto hizo que las ventas triplicaran las expectativas así como numerosas páginas web de intercambio de ideas.

Además del bloque RCX, existieron otros bloques programables, los cuales gradualmente se fueron desarrollando hasta lograr la versión definitiva de la versión NXT. A partir de 1998 se comercializó el inicio de la línea con el robot *Cybermaster*. En enero de 2006 Lego anunció la versión Mindstorms NXT, de última generación, que empezó a comercializarse en junio de ese mismo año.

## 2. DESARROLLO TEÓRICO

### 2.1. Objetivos

El gran objetivo de la práctica consiste en construir un robot a partir de los Lego Mindstorms que sea capaz de seguir una línea negra y a su vez rebasar cualquier obstáculo que pudiese encontrar sobre dicha trayectoria.

Otros objetivos de esta práctica se enumeran a continuación:

- Conocer el entorno de desarrollo Bricx Comand Center, que permitirá programar el NXT en el lenguaje NXC (Not eXactly C)
- Familiarizarse con el ladrillo NXT como herramienta para construir y diseñar robots a partir de Lego, así como el uso de los sensores y motores de los que se dispone
- Conocer el lenguaje de programación NXC como herramienta para programar de forma concurrente al NXT
- Conocer el uso de los sensores al incorporarlos en nuestros robots así como sus limitaciones
- Desarrollar estrategias que nos permitan resolver el problema inicial
- Optimizar el diseño y programación del robot con el fin de que este pueda recorrer cualquier circuito en el menor tiempo posible

### 2.2. Conociendo las herramientas



### 2.2.1. NXT

El bloque NXT es una versión mejorada a partir de Lego Mindstorms RCX, que generalmente se considera la predecesora y precursora de los bloques programables de Lego.

NXT es el cerebro de los robots Mindstorms. Se trata de un bloque programable que permite dar un cierto nivel de inteligencia a los robots construidos.

Este mini ordenador está constituido por un microcontrolador ARM7 de 32 bits, que incluye 256 Kb de memoria Flash y 64 Kb de RAM externa. Además, dispone de conexión Bluetooth y USB así como capacidad para conectar distintos sensores.

### 2.2.2. Sensores

#### Sensor de tacto

Este sensor aporta a nuestro robot el sentido del tacto. Su funcionamiento consiste en detectar cuándo éste está siendo pulsado y cuándo no.



#### Sensor de sonido



En este caso, el sensor de sonido aporta la capacidad a nuestro robot de detectar ruido.

#### Sensor de luz

El sensor de luz detecta diferentes niveles de grises que le brindan al robot la capacidad ver. Este sensor es muy sensible a los cambios de luminosidad.



This is what your eyes see



This is what your robot will see, using the light sensor.

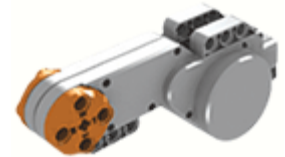
### Sensor de ultrasonido



El sensor de ultrasonido también aporta a nuestros robots la capacidad de ver, pero en este caso su uso está orientado a la detección de objetos que puedan interponerse en el camino de la máquina.

### 2.2.3. Motores

Todo kit Mindstorms incluye 3 motores que servirán para que nuestro robot realice acciones motoras. Además, cada motor dispone de un sensor de rotación interno.



### 2.2.4. Lenguaje: Not eXactly C (NXC)

Existen diversos lenguajes en los que podemos programar el NXT. Uno de ellos es el NXC. NXC es un lenguaje de alto nivel, similar a C, construido sobre el compilador del NBC que es el lenguaje ensamblador de la máquina.

Manual de NXC

[http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC\\_Guide.pdf](http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf)

## 3. DESARROLLO PRÁCTICO

### 3.1. Evolución

Como guía de nuestro trabajo diario detallamos a continuación los avances realizados en las sesiones prácticas.

#### Día 1



hemos instalado han sido el de ultrasonido, luz y sonido, además de los motores, a fin de conocer un poco más su funcionamiento.

Comenzamos a familiarizarnos con el Lego MindStorms y el NXT con el fin de encontrar un diseño inicial para nuestro robot. El diseño original fue obtenido del manual MindStorms Education, por lo que se puede decir que nuestro robot tuvo un diseño estándar.

Para familiarizarnos con el lenguaje de programación NXC (Not eXactly C) empezamos con pequeños programas de prueba en los que comprobábamos el funcionamiento de los sensores. Los que

Además, comenzamos la creación de un blog (<http://niutinbot.blogspot.com>) con el fin de tener un diario con todos los pasos del desarrollo del robot y también para poder compartir nuestra experiencia con la comunidad de usuarios de Lego Mindstorms.

## Día 2

Comenzamos con la primera parte de la práctica, que se basaba en seguir la línea negra, sabemos a priori que ésta será el camino que deberá seguir nuestro robot para completar el circuito y que no habrá curvas con un ángulo superior a  $90^\circ$ .

Para ello tuvimos que tener un buen conocimiento del uso de los sensores y los valores que nos proporcionaban. Al leer negro, el valor obtenido variaba entre 20 y 40, debido a que la luz ambiental afectaba en gran medida. Por lo tanto, el rango seleccionado ha sido lo suficientemente grande (de 15 a 45) como para barrer todas las posibles lecturas del valor negro en cualquier condición.

Lo importante era distinguir cuándo se estaba fuera de la línea negra y nos dimos cuenta de que al leer blanco, correspondiente al suelo, los valores que detectaba el sensor eran mayores de 45, por lo que no habría problemas para distinguir ambos colores.

Nuestro primer programa de prueba consistía en seguir una línea negra sin salirse de la trayectoria, al hacerlo de la forma más genérica posible nos aseguramos de que el robot fuera capaz de realizar curvas no muy pronunciadas, hasta que nos encontramos con el problema de realizar una curva de  $90^\circ$ .

## Día 3

El algoritmo hace pequeñas rectificaciones para corregir la trayectoria, pero estas rectificaciones no son suficientes cuando encontramos un giro de  $90^\circ$ . Por ello, añadimos una manera mediante la cual, el giro se fuera incrementando cada vez que hacía la búsqueda de la línea. De esta manera, se ampliaba un poco más el giro hasta encontrar la línea. Para terminar de optimizar el algoritmo, ajustamos las velocidades tanto en línea recta como en las rectificaciones para conseguir un equilibrio velocidad - fiabilidad.

También empezamos a utilizar el sensor de ultrasonido y realizamos pequeñas pruebas para comprobar que el robot se detenía cuando encontraba un objeto frente a él, para ello, necesitábamos conocer los valores que obteníamos del sensor y cómo debíamos configurarlo.

También nos planteamos la manera en la que esquivaríamos el objeto, y decidimos añadir un nuevo sensor de ultrasonido en la parte izquierda del robot y esquivar el objeto utilizándolo.

## Día 4

Nuestro primer intento de esquivar un obstáculo consistía en bordear el mismo manteniendo una distancia fija.

Para ello existen diferentes casos, si estamos muy cerca del objeto nos alejamos y si estamos muy lejos nos acercamos, pero existía una situación que no conseguimos solucionar: normalmente, cuando el robot pasaba de la mitad del objeto hacia delante, empezaba a girar a la izquierda y llegaba un momento en el que creía que estaba lejos del

objeto, y, en realidad, era que la colocación del sensor engañaba al robot y éste, con el afán de acercarse, continuaba girando sobre si mismo como vemos en la siguiente figura:



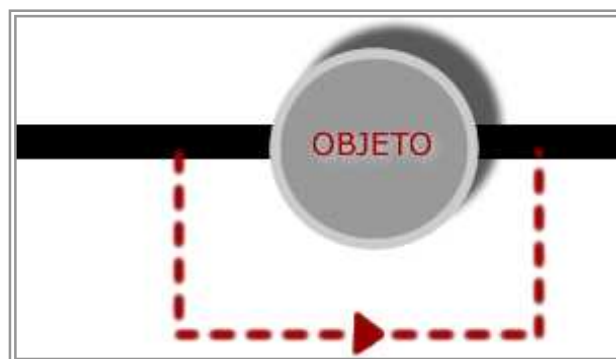
### Día 5

Seguimos intentando solucionar los problemas que nos surgieron el último día, y conseguimos detectar el caso en el que el robot pasaba de la mitad del objeto hacia delante. Para que éste no se perdiera y comenzara a dar vueltas, hicimos que avanzara de manera que nos acercáramos más al objeto, para que el sensor leyera un valor correcto.

Nuestra sorpresa fue que en ocasiones se acercaba demasiado y el sensor de ultrasonido no era tan eficaz como para leer valores tan bajos y colisionaba.

### Día 6

Después de varias pruebas con la solución propuesta en los días anteriores vimos que ésta daba más problemas de los que solucionaba, así que optamos por un cambio de estrategia.



Decidimos bordear el obstáculo utilizando rectas, de tal manera que, al detectarlo, girásemos 90 grados a la derecha y nos colocásemos en paralelo al mismo. A partir de aquí, el robot comienza a esquivarlo y avanzamos en línea recta mientras detectamos el obstáculo.

Al dejar de detectarlo, avanzamos un poco más para dar espacio suficiente al robot para que pueda girar sin tropezar con el objeto, girará 90° a la izquierda y avanzará hacia

delante hasta volver a encontrar el obstáculo. A partir de este instante estamos en la situación inicial y volverá a actuar de la misma forma.

Siempre realiza este algoritmo a la vez que comprueba si ha vuelto a la línea del circuito, es decir, avanzará o girará siempre y cuando no haya esquivado el obstáculo.

Esta solución minimizó los problemas, aunque tuvimos algunos otros, sobretodo en el momento en el que perdíamos el objeto. Como ya hemos comentado, era necesario avanzar un poco más para dejar espacio al girar, pero si no controlábamos el avance, el robot podría saltarse la línea, por lo que debemos controlar en todo momento si encontramos una línea negra que nos indique que hemos esquivado el objeto.

Con esta solución, los problemas que teníamos que controlar eran fácilmente resolubles en comparación con la idea anterior. De este modo conseguimos rebasar los objetos de manera más fiable.

### **Día 7**

De nuevo, una vez que hemos conseguido el objetivo, empezamos a perfilar el código para conseguir mayor velocidad, manteniendo la fiabilidad para esquivar el objeto.

Vimos que podíamos aumentar la velocidad en los giros y en las rectas que utilizábamos en el momento de esquivar, pero hasta un cierto punto, ya que si íbamos muy rápido se daba el caso en que los sensores no tenían tiempo de detectar el objeto y se obtenían valores erróneos.

Por ello, al aumentar la velocidad, fue necesario utilizar una función para leer valores del sensor varias veces seguidas y así realizar lecturas correctas sin que el robot cometiera errores. Es decir, que nos encontramos con muchos problemas típicos presentes en sistemas en los que se deben dar respuestas en tiempo real.

### **Días 8 y 9**

Por último, los días 8 y 9 los dedicamos casi exclusivamente a probar el robot con diferentes obstáculos y ponerlo a prueba en casos extremos y circuitos diferentes.

Nos dimos cuentas de nuevos problemas cuando empezamos a cronometrar el robot para conseguir minimizar el tiempo, vimos que en las líneas rectas nuestro robot perdía mucho tiempo y para mejorarlo hicimos los últimos retoques en el hardware y el software.

Hicimos una modificación en la construcción, que fue subir el sensor de luz de tal manera que los valores que obtenía no eran tan estrictos, sino que si leía medio negro medio blanco se consideraba negro, y seguía avanzando hasta que leyera completamente blanco que es cuando realmente ha salido. Con el sensor más bajo detectaba el negro y blanco con mayor precisión y si el robot detectaba medio negro, medio blanco consideraba que ya se había salido cosa que vimos que era menos eficiente.

En cuanto a la programación mejoramos el algoritmo que sigue la línea negra haciendo que cada vez que corrigiera la trayectoria parara los motores, para conseguir que el robot se centrara sobre la línea y así no se desviara debido a la inercia.

## **3.2. Problemas en la construcción.**

### **Construcción inicial**

El diseño del robot fue obtenido del manual Lego MindStorms Education. Dicho diseño disponía de dos motores y dos sensores. Uno de ultrasonido, para detectar obstáculos, y otro de luz, para seguir la línea recta.

### **Introducción del segundo sensor de ultrasonido**

Una vez que hicimos la construcción inicial del robot, la siguiente decisión relevante que tuvimos que tomar en el diseño, fue la de cómo íbamos a resolver el problema de rebasar el obstáculo.

Había dos alternativas. La primera era introducir un motor que controlase los movimientos del sensor de ultrasonido, ya que cuando rebasamos un obstáculo éste debe seguir en todo momento al objeto. Esto posibilitaba que el programador pueda decidir rebasar el obstáculo por la izquierda o derecha, dependiendo de la forma de éste.

La segunda opción consistía en instalar otro sensor de ultrasonido en uno de los lados, de tal forma que el primero de ellos sirviese para detectar el obstáculo cuando estuviésemos siguiendo la línea, mientras que el nuevo serviría para detectarlo mientras lo estuviese rebasando. Esta posibilidad hace que el robot tenga que rebasarlo siempre por la derecha. Sin embargo, la elección final fue la última, debido a que podíamos disponer de otro sensor de ultrasonido, y la instalación de un sensor y no un motor era más simple.

### **Introducción del sensor de sonido como contrapeso**

Durante la realización de la práctica tuvimos el problema de que en ocasiones, el robot no seguía una línea perfecta, sino que se desviaba hacia un lado. En primer lugar pensamos que se podía deber a un problema de descompensación en el robot, ya que teníamos más peso por la izquierda que por la derecha. Esto se debía, naturalmente, a que el segundo sensor de ultrasonido fue instalado por dicho lado. Para permitir una mejor compensación en el peso del robot, decidimos instalar por el lado derecho el sensor de sonido, con el fin de que el robot tuviese un reparto de pesos equitativo.

Esto parecía funcionar, sin embargo el problema real era que las baterías, al estar descargadas, no proporcionan una fiabilidad del 100% de que ambos motores tengan la misma potencia, por lo que posteriormente fue retirado.



### Recolocación del sensor de luz.

Tras terminar las dos partes de la práctica, decidimos optimizar el diseño y programación del robot para que pudiese realizar el circuito en el menor tiempo posible. En primer lugar, nos dimos cuenta de que el robot en las rectas se salía continuamente, por lo que perdía mucho tiempo. El problema estaba en que el sensor de luz estaba muy bajo, así que cuando el robot se salía de la trayectoria lo más mínimo debía corregir su trayectoria, ya que el valor leído por éste era blanco. La solución estaba en subir el sensor de tal forma que si nuestro robot se salía de la recta, pudiese leerla, ya que en este caso el sensor abarcaba una mayor área del suelo.



### 3.3. Optimizando el algoritmo

#### Giro Progresivo

Los circuitos que el robot debía superar no tenían curvas de más de 90 grados, por lo que al salirse de la trayectoria, el robot debía comprobar el ángulo recto, como máximo, a la izquierda y la derecha con el fin de encontrar el camino.

Esta opción tenía el problema de que en línea recta el robot perdería mucho tiempo, ya que debería comprobar 90 grados a sus lados para encontrarla y sólo bastaría con un pequeño giro a ambos lados para lograr el objetivo.

Para solucionar este problema decidimos que cada vez que el robot leyera blanco, hiciese un giro inicial pequeño a izquierda y derecha, con el fin de encontrar la línea rápidamente. En caso de que no la encontrase aumentaría el número de grados en el giro hasta recuperarla y volvería a mirar a sus lados.

### **Memoria de Giro**

Decidimos incorporar al robot la opción de que recordase el sentido del último giro que había realizado, de tal forma que no se perdiese mucho tiempo a la hora de coger una curva, ya que al superar el giro inicial siempre será el mismo.

### **Rebasar el obstáculo a base de rectas**

Para rebasar el obstáculo existen dos alternativas posibles.

- Superar el obstáculo siguiendo el contorno del objeto.
- Superar el obstáculo a base de rectas.

Inicialmente el grupo trabajó en la primera solución sin éxito. El algoritmo se basaba en mantener un radio constante con el objeto, pero ante la exigencia de pararnos muy cerca del objeto, debido a la poca precisión de los sensores frontal y lateral, decidimos optar por la segunda opción. Esta decisión se tomó porque aportaba mejoras sustanciales respecto a la anterior.

- ✓ Con la segunda opción se puede rebasar cualquier objeto a base de rectas, por lo que el robot puede ir a una velocidad elevada en ellas. Esto no es posible en la primera solución.
- ✓ Es capaz de rebasar cualquier tipo de objeto, grandes, pequeños, anchos, estrechos, redondos, cuadrados... La primera opción estaba orientada a rebasar objetos redondos, sin embargo, objetos que tuviesen un complejión cuadrada serían mas difíciles de sortear.

La primera complicación en la 'solución de las rectas' fue situar el robot a 90 grados exactos de la posición con la que encontraba el objeto. El problema radicaba en que no siempre encontraría el objeto de frente. Por lo que la solución tomada fue girar mientras el sensor lateral no leyese un valor muy pequeño. En este instante el sensor lateral estaría apuntando al objeto, mientras que el robot en su totalidad estaría a 90 grados.

A continuación éste debería ir hacia delante mientras el sensor lateral detectase el objeto. Una vez lo ha rebasado debería continuar un poco más para que pudiese girar todo su cuerpo. Debido a que los objetos podrían tener diferente tamaño, decidimos hacer una medición y alejarnos de ellos en función a ese valor.

Una vez realizado esto, para continuar esquivando el objeto tendríamos que repetir tantas veces como hiciera falta el siguiente algoritmo:

- ✓ Girar 90°
- ✓ Avanzar mientras no se vea el objeto
- ✓ Avanzar mientras se detecte el objeto
- ✓ Avanzar un poco más (en función del tamaño del objeto)

Sin embargo, tuvimos serios problemas debido a la poca fiabilidad de los sensores de ultrasonido que fueron solucionados mediante la implementación de dos nuevas funciones `valmin()` y `valmax()`.

### **Valmin() y Valmax()**

Realizamos una pequeña prueba que consistió en medir varias veces el sensor lateral sin que hubiese ningún objeto delante suya. El resultado fue sorprendente, ya que la mayoría de las veces las lecturas daban 255, que es el valor equivalente a infinito para el sensor. No obstante, inexplicablemente, daba alguna medida muy baja (20, 30...). Así pues, con el objetivo de garantizar una fiabilidad alta del sensor y que funcionara como deseábamos, decidimos crear las funciones `valmin()` y `valmax()` que se encargan de realizar “n” medidas del sensor (en nuestro caso decidimos que 5 mediciones eran suficientes) y se encargaban de dar el valor mínimo y máximo de las medidas obtenidas. Obviamente `valmin()` se utilizará cuando estemos detectando el objeto y `valmax()` cuando no lo estemos detectando.

### **Espera()**

En muchas ocasiones nos vimos sorprendidos por un comportamiento indeseado del robot, ya que al rebasar el obstáculo, éste no encontraba la línea negra. Esto se debía a que hacíamos uso de la función `wait()` para los avances y las esperas. Esta función lo único que hace es esperar un número determinado de microsegundos por lo que si se encontraba ejecutándola mientras pasaba sobre la línea negra, el robot no tenía oportunidad de detectar este hecho. Por ello, creamos una nueva función cuya utilidad es similar a la que implementa NXC, es decir, se encarga de esperar un determinado tiempo, pero, además, evaluamos en cada iteración si se lee negro o no.

### **Off al salir de las rectas**

Debido a los cambios de contexto durante la ejecución concurrente de nuestra aplicación, al salirse de la línea negra y volver a recuperar el camino, el robot avanzaba un poco más de lo necesario. El problema de estos cambios de contexto radicaba en que una vez encontrada la línea negra, se debían producir los cambios de contextos apropiados para que se ejecutase la función para caminar hacia adelante. Entonces, la solución adoptaba fue parar los motores mientras se producían los cambios de contexto. Se comprobó experimentalmente que con la solución propuesta el robot se salía menos en las rectas.

### **Elección de la velocidad correcta**

Los días antes de la competición, después de tener el diseño y la programación del robot perfectamente calibrado, decidimos encontrar las velocidades óptimas para obtener el mejor rendimiento de nuestro robot. Las velocidades tenían que ser elegidas de tal forma, que se mantuviera un equilibrio entre una alta velocidad al seguir el circuito, pocas salidas en recta y que el robot no cometiera errores.

## 4. ¿QUÉ HACE ESPECIAL A NIUTINBOT?

Nuestro robot, llamado NiutinBot, destaca en varios aspectos importantes que lo hacen más rápido y fiable.

### Flexibilidad

NiutinBot es capaz de esquivar una gran variedad de objetos: cajas de diferentes tamaños, botellas, obstáculos con poca anchura pero profundos y viceversa. Esta flexibilidad permite a NiutinBot ser genérico y conseguir esquivar objetos sin restringirse a un tamaño específico.

También es importante destacar que nuestro robot tiene algunos problemas para detectar obstáculos colocados en forma de rombo que se encuentre en la trayectoria, esto es debido a la colocación del sensor y la estrategia planteada para esquivar, que nos ofrece otras mejoras en el resto de los casos.

### Innovación

NiutinBot ofrece un diseño innovador, partiendo de una construcción estándar que nos asegura una buena estabilidad y un buen reparto de pesos, lo que nos permite añadir un nuevo sensor de ultrasonido para esquivar objetos sin que se vea afectada su estabilidad.

Podemos afirmar que el diseño de nuestro robot permite a los sensores funcionar a pleno rendimiento sin perder fiabilidad.

### Fiabilidad

NiutinBot es capaz de esquivar objetos con total seguridad pasando suficientemente próximo al objeto sin tocarlo, esquivando el obstáculo a una velocidad alta y constante. El robot se mueve con mucha suavidad, naturalidad y elegancia sin realizar correcciones excesivamente bruscas.

### Adaptabilidad

NiutinBot está programado de tal forma que pueda adaptarse a cualquier tipo entorno gracias a la definición de sus macros, que permiten cambiar los parámetros rápidamente tales como la velocidad al esquivar, girar o seguir una línea negra; el rango de valores de los sensores, etc.

Sin necesidad de realizar muchos cambios NiutinBot puede completar cualquier tipo circuito como hemos podido observar en la defensa de la práctica.

## 5. RESUMEN Y CONCLUSIONES

Con la realización de esta primera práctica de la asignatura hemos podido aplicar nuestros conocimientos de programación a un nuevo elemento físico y enfrentarnos al diseño de pequeños robots con la ayuda de Lego Mindstorms.

A lo largo de la realización de nuestro primer proyecto nos hemos enfrentado a diversos problemas e impedimentos, con los que incluso no contábamos en un primer momento. Ejemplo de

ello es el gran efecto que producen tanto el medio externo como otras complicaciones en el funcionamiento del robot. Debido a ello hemos tenido que separar el problema principal en pequeños subapartados a los que enfrentarnos uno a uno: construcción del robot, calibración de los sensores, lecturas, velocidades, etc.

Los principales problemas por los que nos hemos visto frenados en varias ocasiones han sido, por ejemplo, los efectos que tiene el medio externo sobre el robot:

- El nivel de la batería del NXT influye en gran medida al rendimiento de los motores y sensores del robot. Debido a esto, los motores podían realizar movimientos indeseados dependiendo del nivel de batería del NXT y los sensores llegaban incluso a obtener lecturas diferentes para el mismo código e iguales condiciones.

- Otro de los problemas que tuvimos que solventar fueron las lecturas incorrectas de los sensores, ya que éstas suelen oscilar, por lo que observamos que la decisión correcta sería evaluarlas en un rango aceptable y no de manera atómica, utilizando una función que realice varias lecturas consecutivas.

- Un ejemplo bastante sorprendente fue el que encontramos al intentar esquivar una papelerera de color blanco metalizado situada en el circuito. Debido a la luz externa, la papelerera reflejaba claridad sobre el circuito, por lo que nuestro robot llegaba a confundir el color negro de la línea con el resto del suelo y no reconocía la trayectoria a seguir. En varias ocasiones realizamos el circuito con la luz del laboratorio apagada y observamos que el rendimiento era muy superior.

Por otro lado, se incluyeron mejoras y variaciones en el código realizado inicialmente para solucionar los diversos problemas con los que nos íbamos encontrando a lo largo de las pruebas realizadas en situaciones excepcionales como las siguientes:

- En muchas ocasiones hacíamos uso del procedimiento *Wait*, disponible en el lenguaje, pero como este no permitía evaluar condiciones de salida para el mismo, decidimos crear una nueva función llamada *Espera* que realiza la misma función y además evalúa condiciones de salida. En nuestro caso, para que el robot dejara de ejecutar una determinada función si encontraba la línea negra o llegaba a un número determinado de ciclos.

- En lo que al sensor de luz se refiere, si éste lo situábamos más arriba, no identificaba tan rápidamente la pérdida de la línea negra, por lo que no realiza tantas correcciones de trayectoria y el robot la sigue de una manera mucho más eficiente y regular.

- Para esquivar cada objeto encontrado, primero intentábamos superarlo bordeando su contorno lo más pegado posible, pero esa idea fue desechada debido a que el sensor en ocasiones perdía el objeto obteniendo lecturas irregulares y eso provocaba un giro sobre sí mismo indefinidamente. Por lo tanto, se optó por esquivarlo mediante rectas alrededor del mismo.

- Por último, otro de los inconvenientes a tener en cuenta, fue la posición en la que quedaba el robot cuando éste encontraba un objeto mediante el sensor frontal. Puede ocurrir que el robot no esté situado correctamente sobre la línea negra, es decir, que esté ligeramente desplazado a los lados o incluso situado en un borde de la misma. Debido a ello, hemos decidido girar el robot inicialmente hasta que se sitúe justamente en el borde derecho de la línea, para, a continuación, girar los 90° aproximados a la derecha y poder realizar la primera "recta de bordeo".

A pesar de los problemas e impedimentos presentados anteriormente y de las diferentes dificultades sufridas en la elaboración de nuestro programa, hemos quedado satisfechos con el trabajo realizado, ya que se han cumplido los objetivos establecidos previamente, además de

añadir varias mejoras y novedades, que creemos han sido beneficiosas para nuestro proyecto. Gracias a ello hemos podido tener un primer contacto con un pequeño robot construido por nosotros mismos. Además lo hemos dotado de un pequeño nivel de inteligencia para que sea capaz de realizar cualquier circuito y sortear objetos encontrados en el mismo, siempre que cumpla una serie de características generales.

## 6. BIBLIOGRAFÍA Y WEBS DE INTERÉS

### Bibliografía utilizada

- Guión de la Práctica 1: “Introducción Programación Robots Lego. Sigue la línea”
- Guía del Lenguaje NXC: [http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC\\_Guide.pdf](http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf)
- Web de la Asignatura: <http://www.educandorobots.es>
- Wikipedia: <http://es.wikipedia.org>

### Webs de Interés

- Blog de la práctica realizada: <http://niutinbot.blogspot.com>

## 7. CÓDIGO FUENTE

```

#define negro 30 // Valor de negro según el sensor de luz
#define G90 50 // Valor del giro inicial
#define velocidad 55 // Velocidad de los motores cuando avanzamos hacia adelante
#define velgiro 30 // Velocidad del giro que se hace al corregir trayectoria
#define velgiroesq 20 // Velocidad de giro cuando esquivamos un objeto
#define rango 15 // Rango de error del sensor de luz
#define DER 0 // Macro para identificar el sentido del giro, 0 es Derecha
#define IZQ 1 // Macro para identificar el sentido del giro, 1 es Izquierda

int Control=0; // Control global de las Tareas
int Direccion=1; // Variable para Direccion (Der: 0; Izq: 1)
int Giros=G90; // 90º inicialmente
int FactorGiro = 1.2; // Factor para Aumentar Giro
int USLateral=S2; // Sensor UltraSonido Lateral
int USFrontal=S4; // Sensor UltraSonido Frontal
int avance=0; // Indica cuanto ha avanzado. Sirve para calcular el espera cuando ya no hay objeto
mutex Cerrojo;

// Función que hace 4 lecturas del sensor de luz y devuelve la de menor valor
int valluz(){
    int min=1023; // Inicializamos la variable min al valor máximo que puede dar el sensor
    for (int i=0; i<4; i++){ // Hacemos las 4 lecturas
        int valsensor=SENSOR_3;
        if(valsensor<min)
            min=valsensor;
    }
    return min;
}

// Función que hace 5 lecturas del sensor de ultrasonido lateral y devuelve la de mayor valor
int valmax(){
    int max=0;
    for (int i=0; i<5; i++){ // 10
        int valsensor=SensorUS(USLateral);
        if(valsensor>max)
            max=valsensor;
    }
    return max;
}

// Función que hace 5 lecturas del sensor de ultrasonido lateral y devuelve la de menor valor
int valmin(){
    int min=256;
    for (int i=0; i<5; i++){ // 10
        int valsensor=SensorUS(USLateral);
        if(valsensor<min)
            min=valsensor;
    }
    return min;
}

```

```

}

// Procedimiento de funcionamiento similar al wait pero evaluamos una condición de salida del mismo que
// en este caso es que encuentre negro
void espera(int tiempo){
    int i;
    i=0;
    // Mientras no encuentre negro y tampoco se haya cumplido el tiempo, seguimos en el bucle
    while ( ((SENSOR_3>negro+rango) || (SENSOR_3<negro-rango) ) && (i<tiempo*10) ){
        i++;
    }
}

// Procedimiento que hace girar 90 grados al robot en cualquier sentido: derecha o izquierda.
void gira90(int Dir){
    if (Dir==0) { // Gira a la Derecha
        OnRev(OUT_B,velgiroesq); //Giramos sobre el mismo
        OnFwd(OUT_C,velgiroesq);
        // Mientras No 90º
        while (!(SENSOR_3>negro+rango) || (SENSOR_3<negro-rango)) {
        }
        Off(OUT_BC);
        OnRev(OUT_B,velgiroesq*2); //Giramos sobre el mismo
        OnFwd(OUT_C,velgiroesq*2);
        // Mientras el sensor lateral lea más de 20, significará que no ha terminado de girar
        while (SensorUS(USLateral)>20) {}
        Off(OUT_BC);
    }
    else
    { // Gira a la Izquierda
        OnRev(OUT_C,0); //Giramos sobre el mismo
        OnFwd(OUT_B,velgiroesq*4);
        // Seguimos girando durante t=90 (nos asegura que giramos los 90 grados)
        espera(90);
        Off(OUT_BC);
    }
    // Hacemos que no sean 90º exactos para que no se pege mucho al objeto
    // debido al mal funcionamiento de los motores.
}

// Tarea que se encarga de leer los sensores de ultrasonidos y dependiendo de su valor, efectúa las
// operaciones necesarias para esquivar los objetos que se encuentre.
task lee_ultrasonido()
{
    while(true) {
        Acquire(Cerrojo); // Adquirimos la cerradura
        if (SensorUS(USFrontal)<26) { // Si el sensor frontal hace una lectura de menos de 26, significa que está
            // muy cerca de un objeto
            Control=1; // Ponemos control a 1 para que no vaya hacia adelante
        }
    }
}

```

```

// Giramos a la derecha cuando encuentre la línea mientras rebasa el obstáculo
Direccion=0;
Giros=1000; // Aumentamos el giro para asegurarnos que el camino que toma es la derecha

Off(OUT_BC); // Paramos los motores
gira90(DER); // Giramos 90 grados ala derecha
while ((valluz()>negro+rango) || (valluz()<negro-rango)) { // Mientras blanco (no negro)
  avance=0; // Inicializamos avance a 0
  // Mientras no encuentre negro y esté detectando al objeto
  while (((valluz()>negro+rango) || (valluz()<negro-rango)) && (valmin()<=30)){
    OnFwdReg(OUT_BC, 55,OUT_REGMODE_SYNC); // Hacia adelante.
    avance++; // Incrementamos avance
  }
  OnFwdReg(OUT_BC, 55,OUT_REGMODE_SYNC); // Hacia adelante
  espera(12+avance); // Es un poco heurístico (ensayo-error) Lo que hacemos es calcular el avance
  // mediante el tamaño del objeto que conocemos por el tiempo que el robot lo ha visualizado.
  Off(OUT_BC);
  // Si encuentra blanco giramos 90 grados a la izquierda
  if ((valluz()>negro+rango) || (valluz()<negro-rango))
    gira90(IZQ);
  Off(OUT_BC);
  // Mientras no encuentre el objeto avanzamos hacia adelante
  while (((valluz()>negro+rango) || (valluz()<negro-rango))&&(valmax()>30)){
    OnFwdReg(OUT_BC, 55,OUT_REGMODE_SYNC);
  }
}
// Paramos los motores
Off(OUT_BC);

}
// Soltamos al cerradura
Release(Cerrojo);
}
}

// Tarea que se encarga de avanzar hacia delante
task hacia_adelante()
{
  while(true){
    // Adquirimos el cerrojo para asegurar la exclusión mútua de la sección crítica
    Acquire(Cerrojo);
    if(Control==0) { // Si hay que moverse
      TextOut(0, LCD_LINE1, "Voy a seguir");
      OnFwdReg(OUT_BC, velocidad,OUT_REGMODE_SYNC); // Hacia adelante
    }
    // Abrimos al cerradura
    Release(Cerrojo);
  }
}
}

// Procedimiento que se encarga de corregir la trayectoria cuando se sale de la línea negra

```

```

void corregir_trayectoria()
{
  // Adquirimos el cerrojo
  Acquire(Cerrojo);
  TextOut(0, LCD_LINE2, "Voy a corregir"); // Texto por Pantalla
  if (Direccion==1) { // Gira a la Izquierda
    ResetRotationCount(OUT_B); // Contador de Giros a 0
    OnRev(OUT_C,velgiro); //Giramos sobre el mismo
    OnFwd(OUT_B,velgiro);
    // Mientras Blanco y No 90º
    while (((SENSOR_3>negro+rango) || (SENSOR_3<negro-rango)) &&
(MotorRotationCount(OUT_B)<Giros)) {}
    if (MotorRotationCount(OUT_B)>=Giros) { // Si llego a 90º
      Giros=Giros+(Giros*FactorGiro); // G180 Giramos hasta el otro extremo
      Direccion=1-Direccion; // Cambios de dirección
    }
    else {
      Off(OUT_BC);
      Giros=G90; // Inicializamos los giros
      Control=0; // Arranca Hacia Adelante
    }
  }
  else { // Gira a la Derecha
    ResetRotationCount(OUT_C); // Contador de Giros a 0
    OnFwd(OUT_C,velgiro); //Giramos sobre el mismo
    OnRev(OUT_B,velgiro);
    // Mientras Blanco y No 90º
    while (((SENSOR_3>negro+rango) || (SENSOR_3<negro-rango)) &&
(MotorRotationCount(OUT_C)<Giros)) {}
    if (MotorRotationCount(OUT_C)>=Giros) { // Si llego a 90º
      Giros=Giros+(Giros*FactorGiro); // Giramos hasta el otro extremo
      Direccion=1-Direccion; // Cambios de dirección
    }
    else {
      Off(OUT_BC);
      Giros=G90; // Inicializamos los giros
      Control=0; // Arranca Hacia Adelante
    }
  }
  // Abrimos el cerrojo
  Release(Cerrojo);
}

// Tarea encargada de leer el sensor de luz
task leer_sensor()
{
  while(true){
    Acquire(Cerrojo);
    if((SENSOR_3>negro+rango) || (SENSOR_3<negro-rango)) { // Si no lee Negro
      if(SENSOR_3>67){ // Si lee color blanco, paramos el programa
        StopAllTasks();
      }
    }
  }
}

```

```
}
Control=1; // Para no ejecutar Hacia Adelante

NumOut(0, LCD_LINE3, SENSOR_3); // Imprimimos el Valor
Release(Cerrojo);
corregir_trayectoria(); // Llamamos al procedimiento corregir_trayectoria
Acquire(Cerrojo);
}
else
{
Control=0; // Arranca Hacia Adelante

}
Release(Cerrojo);
}
}

// Programa principal
task main()
{
SetSensorLight(S3); // Configuramos el sensor de luz (colocado en el puerto 3)
ResetSensor(USLateral); // Reseteamos ambos sensores de ultrasonido
ResetSensor(USFrontal);
SetSensorLowspeed(USLateral); // Configuramos los dos sensores de ultrasonido
SetSensorLowspeed(USFrontal);
// Lanzamos concurrentemente 3 tareas
start hacia_adelante; // Lanzamos la tarea hacia_adelante
start leer_sensor; // Lanzamos la tarea leer_sensor
start lee_ultrasonido; // Lanzamos la tarea lee_ultrasonido
}
```