UNIVERSIDAD POLITÉCNICA DE CARTAGENA
Escuela Técnica Superior de Ingeniería Industrial

Estudio de las posibilidades didácticas en ingeniería de control del LEGO Mindstorms NXT

Titulación: Ingeniero Técnico Industrial, especialidad Electrónica Industrial
Intensificación: Automática
Alumno: Guillermo Nieves Molina
Directores: Julio José Ibarrola Lacalle
José Manuel Cano Izquierdo

Cartagena, a 15 de Mayo de 2008

Nota legal:

Todos los datos e imágenes contenidos en este documento son, o bien de elaboración propia, o han sido utilizados con permiso de sus respectivos propietarios:

Las imágenes 1.2, 1.3, 3.2 y 3.4 se usan por cortesía de LEGO, © The LEGO Group.

La imagen 1.1 ha sido obtenida de Wikimedia Commons, bajo Licencia GNU GPL:
(`http://commons.wikimedia.org/wiki/Image:LegoMindstormsRCX.jpg`)

Las imágenes 2.1 hasta 2.7 son © de Mindsensors, usadas con permiso expreso.

Las imágenes 2.8 hasta 2.11 son © de HiTechnic, usadas con permiso expreso.

Las imágenes 2.12 hasta 2.14 son © de Techno-Stuff, usadas con permiso expreso.

La imaten 2.15 es © de Vernier Inc., usada con permiso expreso.

Este trabajo no está respaldado ni relacionado de ninguna manera con The LEGO Group o sus subsidiarios.

LEGO, el logotipo de LEGO, LEGO Mindstorms y Mindstorms NXT son ® y ™ de The LEGO Group, usadas con buena fe, sin ánimo de lucro y según las directrices de The LEGO Group.

Índice general

Índice de tablas

Índice de imágenes

Capítulo 1

Introducción y objetivos

¡La próxima generación de robots que puedes construir y programar para que hagan lo que quieras!

Con estas palabras da la bienvenida al novato la web de LEGO Mindstorms NXT, el kit de robótica educativa objeto de estudio en este proyecto. A lo largo de estas páginas se desglosa un análisis de las posibilidades de este, en principio, juguete educativo. Juguete muy conocido y querido por la comunidad internacional de aficionados a la robótica, tanto en el ámbito personal como el didáctico, pero que no ha tenido en ningún momento ni lugar un acercamiento serio a la enseñanza en ingeniería de control. Aquí se pretende probar si, en efecto, este NXT puede ser útil en la docencia de dicha materia, partiendo de la base de que por un precio más que competitivo se adquiere un set completo y funcional con un universo a su alrededor mayor de lo que cabría esperar.

De una investigación previa sobre la escena NXT internacional, en todos los campos posibles, y pasando por el análisis detallado de sus componentes, la realización de pruebas y ensayos, y la construcción de dos ejemplos, se intenta dar aquí una idea de hasta dónde se puede llegar con un NXT en un laboratorio universitario.

Empezando por el principio: un poco de información sobre los orígenes de este LEGO Mindstorms NXT.

1.1. Historia de LEGO Mindstorms

LEGO Mindstorms NXT no ha surgido de la noche a la mañana, sino que ha derivado de la colaboración entre LEGO y el Instituto Tecnológico de Massachusetts (MIT), originaria de 1986, y materializada en un acuerdo según el cual, entre otros aspectos, LEGO financiaría investigaciones del grupo de epistemología y aprendizaje del MIT, y a cambio obtendría nuevas ideas para sus productos, que podría lanzar al mercado sin tener que pagar regalías.

A partir de la investigación y trabajo llevado a cabo en el MIT, que desembocaría en el *Programmable Brick*, para uso docente y adaptado al estudio del aprendizaje, LEGO desarrolló lo que sería la primera generación de Mindstorms, el *RCX*. La primera versión, de las tres que acabaría teniendo, salió al mercado en septiembre de 1998 a un precio de unos 200 dólares. Los resultados superaron toda expectativa, vendiéndose más de 80.000 unidades en tres meses, además de hacerse un hueco en la escena robótica internacional.

Imagen 1.1: Imagen del anterior Mindstorms RCX

Esta primera generación contaba con un *ladrillo inteligente* de potencia, lógicamente, inferior al del NXT (CPU Hitachi H8/3292 a 16MHz, 16 KB de memoria ROM y 32 KB de RAM), tres entradas para sensores y tres salidas a actuadores, además de un puerto de infrarrojos. La apariencia del RCX es mucho más cercana a la de los bloques tradicionales de LEGO; de hecho, tanto los sensores como el bloque programable conservan la forma de ladrillos característica de la casa.

A principios de 2004, debido a los malos resultados de LEGO del año anterior, con pérdidas de unos 188 millones de euros, se extendió el rumor de que abandonaría la línea Mindstorms y volvería a su mercado tradicional. Sin embargo, en enero de 2006, LEGO anunció el NXT, que empezó a comercializar en junio de ese mismo año.

La línea de productos Mindstorms no siempre ha sido bien vista por los aficionados a LEGO, que la acusaban de perder el rumbo, ni por los miembros del citado grupo de epistemología del MIT, dado que algunas de las decisiones comerciales de LEGO contradecían los resultados de sus investigaciones conjuntas. En cualquier caso, a día de hoy, LEGO Mindstorms tiene un hueco considerable entre los aficionados a la robótica de cualquier edad y cualquier parte del mundo, como se muestra en el siguiente capítulo.

1.2. Objetivos

El objetivo principal del presente proyecto es analizar y evaluar las posibilidades docentes del robot educativo LEGO Mindstorms NXT en el ámbito de la ingeniería de control.

Actualmente, en los laboratorios de control de las universidades, la enseñanza práctica se reduce normalmente a simulaciones por ordenador, dado que se suele disponer en general de unas pocas maquetas de procesos, y muy limitadas, debido sobre todo al alto precio de éstas. Motores de corriente continua con conexión a PC, plantas simples y procesos industriales a pequeña escala son un material muy específico y que repercute seriamente en el presupuesto del departamento a cargo.

Por contra, un NXT se puede adquirir desde unos 250 € (un precio razonablemente bajo comparado con las maquetas habituales), e incluye, además

de una CPU de una potencia a tener en cuenta, una pequeña colección de sensores y actuadores de buena calidad, conjuntamente a una generosa cantidad de piezas de LEGO Technic, que, con un poco de habilidad, permiten construir en principio un amplio abanico de escenas apropiadas para una clase práctica de ingeniería de control.

Así, desde este punto de partida, surge un interesante estudio de las posibilidades de este robot, pese a la desconfianza inicial que pudiera suscitar al estar hablando de un *juguete*. Pero, como se demuestra aquí, LEGO Mindstorms NXT es mucho más que un juguete.

1.2.1. Desarrollo de los objetivos

Para conseguir llevar a cabo los objetivos de la mejor manera posible, se ha dividido el plan de trabajo del proyecto en varias etapas, reflejadas aquí en capítulos de esta memoria.

1. Búsqueda de información a través de todas las fuentes posibles (principalmente internet, dado que es la que permite mayor difusión y autopublicación) sobre la situación actual de la escena LEGO Mindstorms, y resumen de ella en un estado del arte. Desde alternativas a la comercial para su programación hasta proyectos en otras universidades, pasando por libros editados, grupos de usuarios, etc.

2. Descripción completa y exhaustiva del sistema.

3. Análisis, calibrado y pruebas de laboratorio a sensores y actuadores.

4. Desarrollo de aplicaciones concretas donde se muestre la idoneidad de este robot para el uso docente.

5. Conclusión final

Imagen 1.2: Imagen del ladrillo inteligente con los sensores y motores conectados

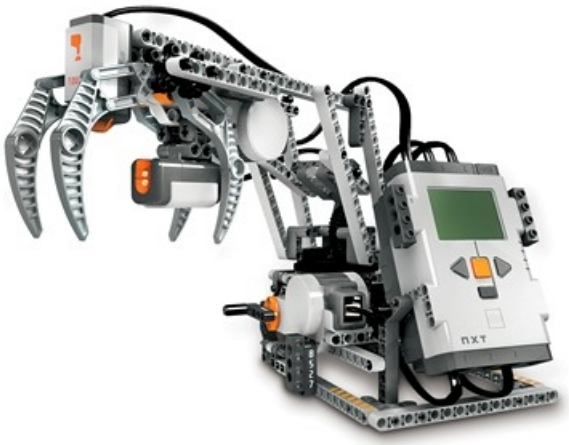


Imagen 1.3: Imagen promocional de un montaje del NXT

Capítulo 2

Estado del arte

A día de hoy, escribir en el famoso buscador de internet Google *Lego Mindstorms NXT* devuelve más de medio millón de resultados en diversos idiomas. Tras un rápido escrutinio, sin profundizar mucho en la materia, se despliegan ante los ojos del navegante decenas de páginas temáticas, proyectos, lenguajes y entornos de desarrollo, libros editados e incluso campeonatos mundiales de robótica usando el NXT. Diversos departamentos de universidades de todo el mundo han adaptado el robot que nos ocupa para la enseñanza en sus clases, y hay proyectos fin de carrera e incluso tesis doctorales sobre aplicaciones concretas y posibles.

Al estar liberados tanto las especificaciones de hardware como las de software, ha sido muy sencillo para la comunidad de usuarios elaborar tan amplio catálogo de posibilidades. Este proyecto no es, ni de lejos, el primero que profundiza en el uso docente del NXT, por lo que hay disponible una extensa bibliografía, una sólida comunidad de usiarios, y, en resumen, un auténtico *mundillo* alrededor de una idea en principio simple como es un robot educativo de LEGO.

Esquematizando, podemos organizar la escena actual en:

- Programación del NXT
  - Lenguajes de programación
  - Entornos integrados de desarrollo (IDEs)
  - Manuales y guías sobre programación
  - Aplicaciones de comunicación con el PC
- Hardware
- Proyectos en otras universidades
- Bibliografía y fuentes de información
- Otras utilidades

2.1. Programación del NXT

La presentación comercial de Lego Mindstorms NXT incluye un software de desarrollo de tipo gráfico, basado en NI Labview, relativamente sencillo de utilizar y funcional en Windows y Mac. Sin embargo, dado que se busca una mayor profundidad al programar y un acceso a nivel más bajo, se hace necesaria la existencia de un lenguaje de programación por entrada de texto, más funcional y orientado a un uso más serio que el propuesto por LEGO (que, no obstante, es bastante completo y útil para los primeros pasos en el mundo de la programación).

2.1.1. Lenguajes de programación

En la siguiente tabla se resumen las alternativas disponibles hoy día a la hora de programar el NXT.

Nombre	Tipo	Firmware	IDE	SO	Libre
NXT-G Incluido al adquirir el NXT	Gráfico	Original	Sí (basada en NI Labview)	Win/ Mac	No
RoboLAB Orientado al uso educativo (descontinuado, se usaba en las anteriores versiones de Lego Mindstorms)	Gráfico	Original	Sí	Win/ Mac	No
NQC (Not Quite C) (se usaba en anteriores versiones de Lego Mindstorms)	C/C++	Original	Sí	Win/ Mac/ Linux	Sí (MPL[a])
NBC (Next Byte Codes)	Ensamblador	Original[b]	Sí	Win/ Mac/ Linux	Sí (MPL)
NXC (Not eXactly C) Es el más usado a nivel mundial, el más documentado, y el que se usa-rá en este proyecto de manera mayoritaria	C/C++	Original	Sí	Win/ Mac/ Linux	Sí (MPL)
RobotC	C	Original	Sí	Win/ Mac	No
NI Labview Toolkit	Gráfico	Original	No (Usa NI Labview)	Win/ Mac	No
leJOS NXJ	Java	Custom	No (Se puede usar Eclipse o Netbeans)	Cualquiera[c]	Sí (MPL)
pbLua	Lua	Custom	No[d]	Win/ Mac/ Linux/ BSD	Freeware
leJOS OSEK	ANSI C/C++	Custom	No (Usa Eclipse)	Win/ Mac/ Linux	Sí (MPL)

Tabla 2.1: Resumen de lenguajes de programación para el NXT.

[a] Mozilla Public License (ver `http://www.mozilla.org/MPL/MPL-1.1.html`)
[b] Puede ser actualizado a un firmware mejorado
[c] Sólo es necesario instalar la máquina virtual Java
[d] Los programas se compilan en el mismo NXT

2.1.2. Entornos integrados de desarrollo (IDEs)

1. **NXT-G:** es el IDE original, basado en NI Labview, incluido con el NXT. Incorpora tutoriales, herramientas de actualización de firmware y otras características. Si bien tiene cierto valor educativo o de iniciación, no es útil para este proyecto.

2. **Robolab:** El anterior a NXT-G, hoy descontinuado.

3. **BricxCC:** Open source, y probablemente el más completo de todos, BricxCC es un entorno para NQC y NBC/NXC con múltiples opciones, desde herramientas de actualización de firmware, explorador de archivos, control del NXT desde el PC,... Es el utilizado en este proyecto, en el siguiente capítulo se tratará más en profundidad. Disponible libremente en la web del proyecto NBC/NXC ([1]).

4. **RobotC:** Entorno que posibilita el uso del lenguaje RobotC, de pago y muy popular entre los aficionados de alto nivel, sobre todo en campeonatos. Disponible en su página web (`http://www.robotc.net/`)

2.1.3. Manuales y guías sobre programación

Actualmente hay publicado un pequeño número de libros sobre Mindstorms NXT, la mayoría de ellos dedicados a montajes particulares, proyectos curiosos o programación avanzada en NXT-G, el lenguaje base. Sin embargo, existen unos pocos manuales sobre programación en el lenguaje que aquí nos ocupa, NBC/NXC.

1. **Programming LEGO NXT Robots using NXC (beta 30 or higher)**, por Daniele Benedettelli. ([2]) Manual introductorio de NXC, el lenguaje maś extendido, incluyendo los conceptos de tipado, acceso a sensores y motores, escritura a archivos, etc. Útil como primera lectura y si no se está muy familiarizado con el mundo de la programación, pero no profundiza en ningún aspecto más de lo necesario. Disponible gratuitamente en la web del proyecto NBC/NXC ([1]).

2. **Not eXactly C (NXC) Programmer's Guide**, por John Hansen ([3]). John Hansen (creador del lenguaje NXC), ha compendiado y documentado todas las funciones disponibles en NXC, con una breve des-

cripción de su utilidad, argumentos aceptados, y tipos de datos. Disponible gratuitamente en la web del proyecto NBC/NXC ([1]). Si bien las descripciones pueden llegar a ser excesivamente sucintas, la mayor utilidad de este documento (incluido como ayuda en el IDE BricxCC) es la de listar y tabular los tipos de argumento que acepta y devuelve cada función, los identificadores que se pueden utilizar y que vienen ya implementados de base, etc.

2.1.4. Aplicaciones de comunicación con el PC

Dado que la tarea de programación se realiza en el ordenador, es imprescindible contar con un buen interfaz de comunicaciones entre los dos elementos. El NXT viene preparado de fábrica para la conexión vía USB y Bluetooth.

1. **fantom driver:** Driver básico para la comunicación del NXT vía USB. Se instala con el entorno de desarrollo NXT-G, pero además el código está disponible liberado en la web de LEGO ([4]).

2. **NXTender:** Software desarrollado de manera independiente que permite al NXT controlar el ordenador, emulando un teclado o un ratón. Está disponible para descarga gratuitamente en la web del proyecto: `http://www.tau.ac.il/~stoledo/lego/NXTender/`

3. **RWTH - Mindstorms NXT Toolbox:** Toolbox de comunicación vía Bluetooth para MATLAB, desarrollada en el Instituto de Imagen y Visión por Ordenador RWTH Aachen. Según la propia web, además de funciones de alto nivel que permiten interacción directa con los sensores y motores del NXT, están disponibles comandos de control y sistema de la propia documentación del NXT. El toolbox está licenciado bajo GNU GPL, y dispone de un repositorio SVN con las últimas versiones, en la web del proyecto ([5]).

4. Asimismo, un gran número de *wrappers* y pequeñas aplicaciones de comunicación tanto vía USB como Bluetooth, la mayoría de ellas por consola, han sido programadas por la comunidad, en toda una variedad de lenguajes y formatos. Aunque el estudio pueda ser interesante, no es objeto en este proyecto.

2.2. Hardware

Así como la apertura de las especificaciones de software ha permitido que a día de hoy haya un gran abanico de posibilidades a la hora de programar el NXT, LEGO también ha publicado las especificaciones de hardware, con lo que varias empresas se han lanzado al mercado con productos destinados a su uso con LEGO Mindstorms NXT.

2.2.1. Mindsensors

Las descripciones e imágenes están tomadas de la página oficial, [6].

- **Subsistema de visión para el NXT:** Este completo sisema de visión, con lentes de focal ajustable, al conectarlo a un puerto estándar de sensor, sigue y rastrea hasta 8 objetos de distintos colores. Además, devuelve las coordenadas de cada objeto. Se conecta al ordenador a través de un puerto USB estándar, aunque no es necesario para el funcionamiento autónomo del NXT.

Imagen 2.1: Imagen del subsistema de visión de Mindsensors

- **Acelerómetro:** Este acelerómetro de 3 ejes opera en cuatro niveles de sensibilidad (2,5g, 3,3g, 6,7g y 10,0g), y mide orientación, velocidad,

y aceleración. Se comunica con el NXT mediante el protocolo I2C, el mismo que utiliza el sensor de ultrasonidos incluido en la caja, y soporta la arquitectura ADPA (Auto Detecting Parallel Architecture), que permite conectar varios sensores al mismo puerto sin necesidad de multiplexores adicionales.

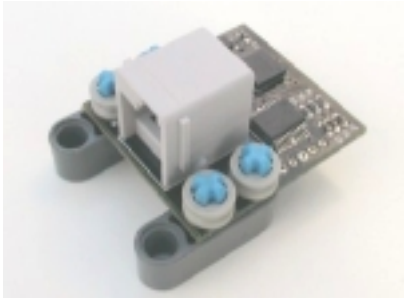


Imagen 2.2: Imagen del acelerómetro de Mindsensors

- **Detector de obstáculos por infrarrojos duales:** Dispone de dos LED infrarrojos orientados 45º hacia adelante, lo que le permite detectar casi cualquier obstáculo que se interponga en su camino, en un rango de 15 a 45 cm.

Imagen 2.3: Imagen del detector de obstáculos de Mindsensors

- **Brújula magnética:** Este sensor mide el campo magnético de la Tierra y devuelve la orientación del robot, con una resolución de hasta 0.1 grados. Su funcionamiento está basado en el sensor magnético de dos ejes ortogonales Honeywell HMC1052.

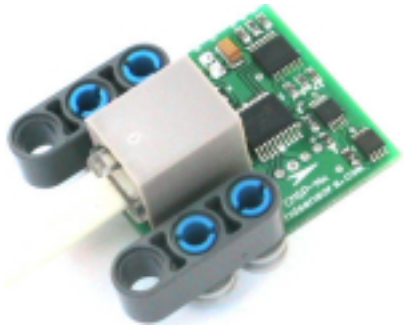


Imagen 2.4: Imagen de la brújula de Mindsensors

- **Sensor de presión neumática:** Mide la presión neumática sobre el sensor, desde 0 a 240 kg/cm^2. Está pensado para usarlo conjuntamente con LEGO Pneumatics, la gama de productos de neumática LEGO (valvulería, pistones, etc).

Imagen 2.5: Imagen del sensor de presión de Mindsensors

- **Reloj de tiempo real:** Es simplemente un reloj, con batería independiente que puede durar hasta 10 años. Compensa segundos intercalares hasta el año 2100.

Imagen 2.6: Imagen del reloj de Mindsensors

- **Sensores Infrarrojos de precisión:** Para corta (4 a 30 cm, sensor Sharp GP2D120), media (10 a 80 cm, sensor Sharp GP2Y0A21YK), y larga (20 a 150 cm, sensor Sharp GP2Y0A02YK) distancia. Devuelven en valor de la medida en milímetros, con más precisión que el resto de sensores. También se comunican mendiante el protocolo I2C.

Imagen 2.7: Imagen del sensor de precisión de Mindsensors

2.2.2. HiTechnic

Las descripciones e imágenes están tomadas de la página oficial, [7]. El hardware de HiTechnic está certificado por LEGO, con lo que se asegura una compatibilidad y fiabilidad absolutas.

- **Brújula magnética:** Al igual que el de Mindsensors, devuelve la orientación actual del robot. La salida, actualizada 100 veces por segundo,

es un valor entre 0 y 359, equivalente a grados sexagesimales. Puede ajustarse para prevenir interferencias por campos magnéticos externos.

Imagen 2.8: Imagen de la brújula de HiTechnic

- **Sensor de color:** Esta actualización del sensor óptico es capaz de reconocer el color de una superficie a 6 mm de distancia (distancia de funcionamiento recomendada por el fabricante). Funciona usando tres LEDs de distintos colores para iluminar la superficie y medir la intensidad de cada reflejo. En función de los valores, calcula el color de la superficie objetivo.

Imagen 2.9: Imagen del sensor de color de HiTechnic

- **Sensor de aceleración:** Su funcionamiento es similar al de Mindsensors. Mide en tres ejes, aceleraciones desde -2g hasta +2g.

Imagen 2.10: Imagen del acelerómetro de HiTechnic

- **Sensor de giro:** Este sensor giroscópico devuelve la velocidad de rotación del NXT como valor entre 0 y 359 grados por segundo, así como el sentido de giro.

Imagen 2.11: Imagen del giróscopo de HiTechnic

2.2.3. Techno-Stuff

Techno-Stuff también dispone de una pequeña gama de productos para el LEGO Mindstorms NXT. Las descripciones e imágenes están tomadas de la página oficial, [8].

- **Sensor de presión:** Mide la presión neumática sobr el sensor, en un rango de 0 a 170 KPa. Conserva la forma de ladrillo tradicional de LEGO.

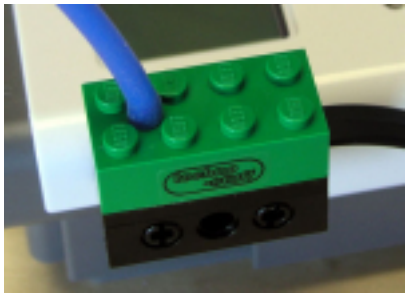


Imagen 2.12: Imagen del sensor de presión de Techno-Stuff

- **Sensor de movimiento por infrarrojos:** Utiliza un sensor PIR (*Passive IR*, infrarrojos pasivos), como el de muchos sistemas de alarma comerciales, para detectar movimiento en los alrededores (hasta 6 metros de rango). No interfiere con las señales de mandos a distancia o similares.

Imagen 2.13: Imagen del sensor de movimiento de Techno-Stuff

- **Sensor de fuentes de infrarrojos:** Este sensor es capaz de detectar fuentes de radiación infrarroja como bombillas incandescentes, velas, o incluso la luz del sol, con una sensibilidad máxima de 940 nm de longitud de onda.

Imagen 2.14: Imagen del sensor de infrarrojos de Techno-Stuff

2.2.4. Vernier

La gran novedad que ha incorporado Vernier al mercado es el Adaptador NXT, con el cual, según los datos extraídos de su web, [9], se pueden conectar directamente sensores de esta casa al NXT. El espectro de más de 30 sensores soportados cubre desde acelerómetros, barómetros, sensores químicos (de O_2, pH, etc.), sondas de corriente,... Prácticamente de todo tipo existente hoy día en la industria. En la propia web de Vernier hay desarrollados varios proyectos con el NXT: un compresor de aire, un comprobador de carga de baterías, y diversas aplicaciones más. Es conveniente destacar que no funciona con NBC/NXC, sino únicamente con NXT-G, LabVIEW o RoboLAB.

Imagen 2.15: Imagen del adaptador de sensores de Vernier

En resumidas cuentas, y a la luz de lo expuesto, hoy día en el mercado se puede encontrar casi cualquier tipo de sensor para utilizarlo con el NXT (y si no, siempre se puede fabricar uno casero o adaptar uno comercial, siguiendo las instrucciones, por ejemplo, de Extreme NXT [10]), lo cual viene a paliar en cierta medida la escasez de actuadores disponibles.

Mediante sensores como la brújula o los acelerómetros se consiguen realimentaciones externas de posición, orientación, etc., necesarias en robótica móvil; sensores de color y cámaras dotan a nuestros robots de *ojos* que cada vez ven más y mejor, y todo ello cimentado sobre la sencilla estructura de LEGO Technic, que permite construir robots de cada vez más complejidad sin tener que soldar un cable, una chapa, o necesitar un laboratorio de altas prestaciones.

2.2.5. Batería recargable

La necesidad de una batería recargable nace inevitablemente dado el alto gasto en pilas (6 unidades AA) del NXT, aunque el consumo en sí no sea desmesurado. La batería, comercializada por la propia LEGO, se compone de polímeros de ión de litio, y proporciona 1400 mAh[1] a 7.4 V, una cantidad ligeramente inferior a la que teóricamente se obtendría con seis pilas de 1.5 V (9 V), pero bastante aproximada a la que en realidad proporcionan dichas pilas.

La batería reemplaza la tapadera trasera del ladrillo y sobresale por debajo de éste la anchura de una barra agujereada de LEGO Technic, circunstancia a tener en cuenta a la hora del diseño, como se muestra en la imagen 2.16. El rendimiento en las pruebas es muy bueno, y en ningún momento da muestras de insuficiente voltaje o potencia. La duración es aproximadamente la misma que la de unas pilas alcalinas de buena calidad.

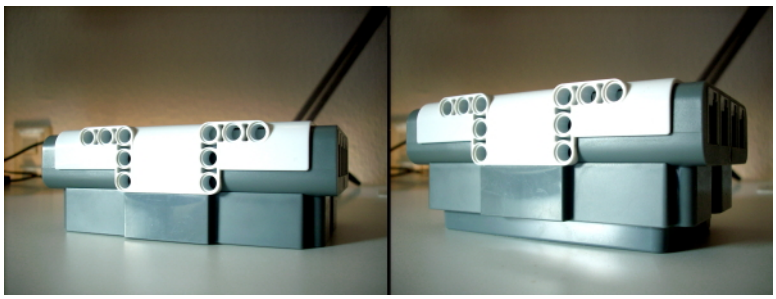


Imagen 2.16: Comparación entre el ladrillo con y sin la batería

[1]Miliamperio hora, medida de carga eléctrica equivalente a 3.6 Culombios. La carga eléctrica en mAh dividido por el consumo del dispositivo en mA da una medida en horas del tiempo estimado de descarga.

Imagen 2.17: Imagen de la batería recargable

2.3. Proyectos en otras universidades

1. **Tesis doctoral en Informática: "LEGO MINDSTORMS - based mobile robots team"**, por Daniele Benedettelli (Universitá degli Studi di Siena), sobre robótica en grupo cooperativa. Ha sido realizada usando la versión RCX de Mindstorms. No está en principio relacionada con este proyecto, sino enfocada desde el punto de vista de la informática aplicada a la robótica. Informe final disponible (en italiano) gratuitamente en su web personal, `http://daniele.benedettelli.com`

2. **APRIL (A PID Robot Implemented with LEGO):** Proyecto de Kevin McLeod (University of British Columbia) sobre la implementación de un sistema de control con un regulador PID en un Mindstorms NXT. Informe disponible gratuitamente en su web personal, `http://www.physics.ubc.ca/~kevinmcl/projects/lego/APRIL/`

3. **Matlab meets Mindstorms:** Proyecto del Instituto de Imagen y Visión por Ordenador RWTH Aachen, sobre el uso de Matlab en el control directo del NXT. Probablemente uno de los proyectos más importantes sobre la materia, incluyendo cursos con asistencia de más de 300 alumnos y el uso de 100 robots. Es responsable también del toolbox de comunicación vía bluetooth para Matlab **RWTH - Mindstorms NXT Toolbox**, disponible gratuitamente en la web del proyecto ([5]).

2.4. Bibliografía y fuentes de información

2.4.1. Internet

La mejor fuente de información sobre el NXT hoy día es internet. Utilizar un buscador o un foro, grupo de usuarios, etc permite llegar prácticamente a cualquier resultado, y es casi seguro que, cualquiera que sea el problema que nos ocupe, alguien lo habrá resuelto antes o se mostrará encantado de ayudarnos a resolverlo.

Aún así, conviene reseñar aquí las fuentes de referencia más completas y necesarias.

1. **NXTreme:**
   `http://mindstorms.lego.com/Overview/NXTreme.aspx`
   Como bien describe la propia LEGO al entrar a la página, aquí se recopilan *todas las herramientas para llevar a tu Mindstorms NXT hasta el extremo*. Las especificaciones al completo del NXT se encuentran disponibles en esta página, tanto de hardware como de software, así como un pequeño directorio de enlaces. LEGO ha confiado en que liberar todo sobre su robot a la comunidad redundaría en su propio beneficio al permitir así que los usuarios puedan exprimir todo su potencial.

2. **Mindstorms Community:**
   `http://mindstorms.lego.com/community/default.aspx`
   También dentro de la propia web de LEGO, esta sección es un intento por agrupar a todos los usuarios y permitir su comunicación, la transmisión y compartición de sus ideas y proyectos y, en definitiva, actuar como núcleo de toda la actividad Mindstorms en internet. Incluye enlaces a eventos, al Programa de Desarrolladores Mindstorms, y descargas diversas.

3. **NXTasy:** `http://nxtasy.org` Probablemente el blog más activo en cuanto a Mindstorms se refiere. Los grandes nombres de la escena NXT contribuyen habitualmente con proyectos, ideas, enlaces, etc. Cabe destacar además su enorme repertorio de enlaces a sitios relacionados con el NXT. Su foro (`http://forums.nxtasy.org`) es de los más completo de internet y el que centraliza la mayoría de los usuarios.

4. **The NXT step:** `http://thenxtstep.blogspot.com/` Otro blog bastante conocido en la comunidad NXT. En él se pueden encontrar montajes, consejos, vídeos, y un largo etcétera de artículos relacionados con LEGO Mindstorms.
5. **BricxCC:** `http://bricxcc.sourceforge.net` Aunque ya nombrada, la web del lenguaje de programación que se usará en este proyecto.

2.4.2. Libros

A pesar de que el uso de internet como fuente primaria de información sea casi dominante, existe no obstante un pequeño número de libros publicados sobre la temática que nos aborda. Principalmente se dividen en dos grupos:

- **Libros de programación:** Estos libros son los que en principio cabría imaginar más interesantes, al versar acerca de técnicas, ejemplos y buenas prácticas de programación en diversos lenguajes. Ejemplos son *LEGO MINDSTORMS NXT-G Programming Guide* (2007, Apress), por James Floyd Kelly, utilizando el entorno de programación original de LEGO NXT-G, o *LEGO Mindstorms NXT Power Programming: Robotics in C* (2007, Apress), por John Hansen, creador de NBC/NXC y muy recomendable. La portada se muestra en la imagen 2.18.
- **Libros de montajes:** Este tipo de libros, de interés inicial menor para lo que nos ocupa, exploran diversos montajes concretos, a veces temáticos, con sus correspondientes programaciones, buscando más la espectacularidad y el entretenimiento que la didáctica. Ejemplos son *The LEGO MINDSTORMS NXT Zoo!* (2008, No Starch Press), por Fay Rhodes, o *The Unofficial LEGO MINDSTORMS NXT Inventor's Guide* (2008, No Starch Press), por David J. Perdue. Con este tipo de libros aprenderemos a construir desde insectos hasta brazos robóticos, pasando por vehículos, etc.

2.5. Otras utilidades

En este apartado se incluyen los programas que, si bien son útiles al trabajar con LEGO Mindstorms NXT, no tienen cabida en ninguna otra

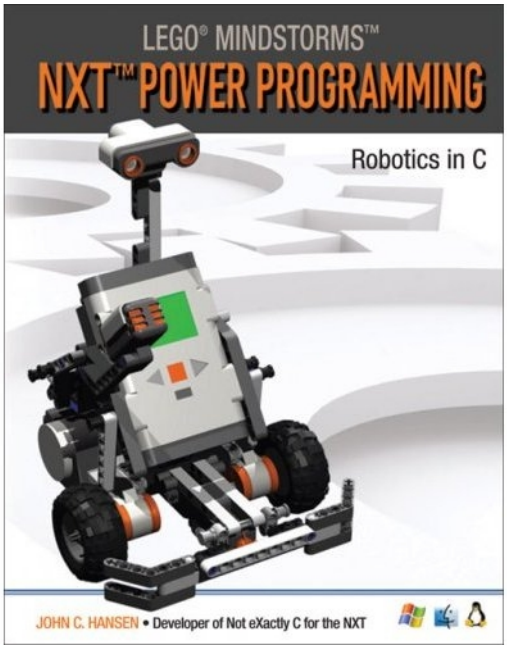


Imagen 2.18: Portada del libro escrito por John Hansen

sección al cubrir aspectos distintos de funcionamiento.

2.5.1. LEGO Digital Designer

El LEGO Digital Designer (o **LDD**), es un programa de CAD tridimensional para diseñar y reproducir montajes reales de LEGO. Una vasta paleta de piezas, que cubre desde los clásicos LEGO City hasta los Mindstorms NXT, permite construir en un entorno 3D como si de la realidad se tratase. Si bien existe una gama de, o bien programas de diseño de LEGO, o bien sets de piezas para usar en programas de diseño tradicionales, como LeoCAD o LDraw, este es el más completo y actualizado.

Entre sus opciones más útiles, está el calcular, en base a los precios de la tienda oficial LEGO, el coste de cada diseño; el comenzar con una determinada colección de piezas y trabajar con ellas; y, sobre todo, la generación automática de manuales de instrucciones. A partir del diseño final, y en base a unos algoritmos con parámetros ajustables (tales como número máximo de piezas por paso), el LDD crea rápidamente una guía de construcción paso a paso, lista para imprimir y seguir.

El funcionamiento es bastante intuitivo, aunque puede dar algún problema de piezas que no lleguen a conectar justo como se desea al estar en una posición difícil, o de conexiones que en la realidad sean posibles por tolerancias de fabricación o fuerce de piezas, pero que en el entorno de LDD no puedan tener lugar.

El software, funcional en Windows y Mac, se puede descargar gratuitamente de la web de LEGO (ver [11]).

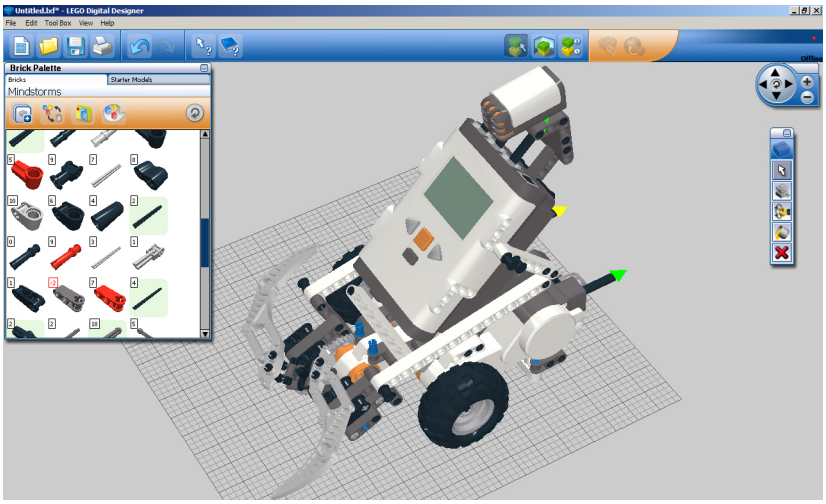


Imagen 2.19: Captura de pantalla del LEGO Digital Designer

A día de hoy, toda la bibliografía, así como la práctica totalidad de la información aprovechable existente en internet, se encuentra únicamente en inglés, dado el carácter internacional de la escena NXT.

Capítulo 3

El NXT en profundidad

La presentación comercial del NXT incluye una CPU (el *ladrillo inteligente*, o *the brick*, como se le conoce mayoritariamente en la escena anglosajona), tres motores y cuatro sensores, además de gran número de piezas, tanto tradicionales de Lego Technic como específicas del NXT. Una guía para principiantes permite tener el primer robot (un vehículo de tres ruedas) operativo en media hora, y se incluyen, además, instrucciones para construir otros tres montajes: una grúa de pinza, un robot humanoide, y un escorpión.

A continuación se resumen las principales características de los componentes:

3.1. El ladrillo

El ladrillo es el elemento fundamental del LEGO Mindstorms NXT. Incluye la CPU, el sistema de comunicaciones, los puertos de entrada y salida y la interfaz de usuario, todo en un bloque de plástico sólido de pequeños tamaño y peso como el que aparece en la imagen siguiente.

Imagen 3.1: Imagen del ladrillo

Como se aprecia en la tabla a continuación (extraída de la propia web de LEGO, [4]), a pesar de estar pensado en principio como un *juguete educativo* (para mayores de 12 años), la inclusión de un microprocesador de capacidad similar a la de un Intel 486, más los puertos de entrada y salida y la comunicación tanto con PC como con otros NXT, cableada o por bluetooth, lo convierten en un elemento muy interesante y de un potencial inicial muy elevado:

Procesador principal	Atmel 32-bit ARM (AT91SAM7S256) 256 KB de Flash 64 KB de RAM Reloj a 48 MHz
Procesador secundario	Atmel 8-bit AVR (ATmega48) 4 KB de Flash 512 B de RAM Reloj a 8 MHz
Procesador para comunicación por BlueTooth	CSR BlueCore 4 v2.0 (+Sistema EDR) Soporte de Serial Port Profile (SPP) 47 KB de RAM interna 8 MB de Flash externa Reloj a 26 Mhz
Comunicación USB 2.0	Puerto a 12 Mbit/s
Puertos de entrada	4, por cable de 6 líneas, con soporte digital y analógico siendo uno de ellos de alta velocidad (IEC 61158 Tipo 4/EN 50170)
Puertos de salida	3, por cable de 6 líneas. Soporte para entrada desde encoders
Display	LCD monocromo Resolución: 100x64 (26x40.6 mm)
Altavoz	Canal de 8 bit Ancho de banda: de 2 a 16 Khz
Interfaz de usuario	4 botones
Alimentación	6 pilas AA

Tabla 3.1: Resumen del Hardware del NXT

3.2. Los actuadores

Dada la naturaleza del NXT, el rango de actuadores se reduce básicamente a pequeños motores de corriente continua. Las líneas de transmisión difícilmente pueden entregar mucha más potencia, o regular elementos como, por ejemplo, un cilindro neumático. De cualquier manera, LEGO Mindstorms NXT no es un robot diseñado ni mucho menos para un uso industrial, con las evidentes limitaciones que esto implica: pequeñas potencias y rango de actuación reducido.

Así, puede surgir el problema de exigir más potencia al NXT. Una solución puede ser sustituir los motores por relés, que con una circuitería adecuada puedan conmutar dispositivos de mayor carga, o bien por electroválvulas si lo que se persigue es el uso en circuitos de aire comprimido.

En el mercado existe toda una gama de nuevos sensores desarrollados por terceros, pero hasta ahora no se han comercializado nuevos actuadores. La razón, además de lo anteriormente aclarado, es que mediante el uso de motores se puede emular casi cualquier otro tipo de actuador necesario. Por ejemplo: uno de los montajes propuestos por LEGO al comprar el NXT es una grúa tipo portuaria con pinza. Pues bien, la pinza no es un actuador *per se*, sino que el efecto se consigue con una astuta disposición de las piezas que transmiten el movimiento rotatorio como apertura y cierre de la pinza, como se muestra en la imagen 3.2

En cualquier caso, LEGO tiene disponible desde hace varios años su gama de productos Pneumatic, compuesta por actuadores lineales, valvulería, etc., que, si bien operan en rangos de actuación reducidos, pueden ser útiles para uso en laboratorio y simulación de procesos industriales. Sin embargo, nunca han calado hondo entre los aficionados, muchos de los cuales los encuentran innecesarios.

Por lo tanto, el estudio del uso del NXT parte de la premisa inicial de que los únicos actuadores disponibles son motores de corriente continua. Los que se suministran al comprar el NXT son tres motores de corriente continua y funcionamiento PWM[1]. El uso de un motor paso a paso pudiera parecer en

[1] **P**ulse **W**idth **M**odulation, modulación por anchura de pulso. La alimentación al motor no es constante, sino que se activa y desactiva muy rápidamente, del orden de 7800 veces por segundo. Variando la relación $\frac{Tiempoencendido}{Tiempoapagado}$ se consigue variar el valor medio de la tensión suministrada al motor y, por tanto, la velocidad

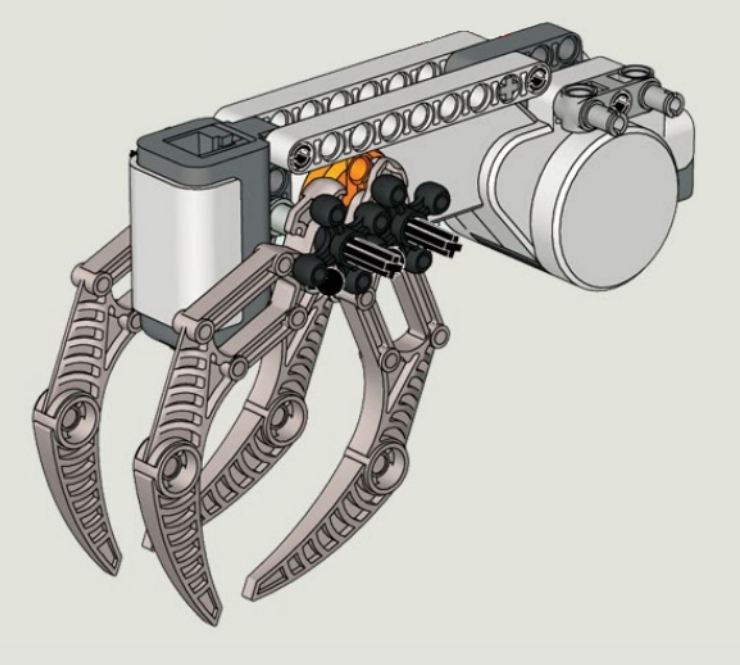


Imagen 3.2: Imagen del montaje de la pinza

principio más interesante, pero los suministrados con el NXT son tradicionales de corriente continua. Aún así, y a pesar de su reducido tamaño, esconden en su interior un complejo sistema de reducción por tren de engranajes y un sensor de rotación de tipo tacométrico. Como se aprecia en la imagen 3.3, disponen de diversos puntos de unión con piezas de LEGO.

Imagen 3.3: Imagen externa del motor

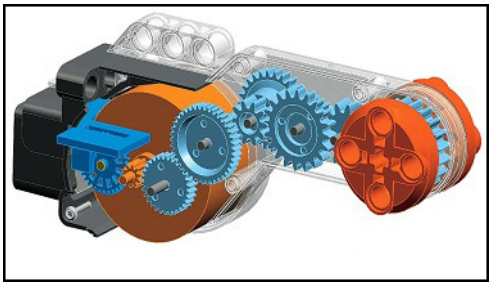


Imagen 3.4: Imagen interna del motor, donde se aprecia el complicado tren de engranajes y el tacómetro

Según las pruebas realizadas en [12], el motor alcanza sin carga una velocidad de 160 rpm, y hasta 120 rpm con una carga de 11.5 N·cm, alimentado a 9 V. Es capaz de entregar un par máximo de 25 N·cm a 60 rpm, aunque no es recomendable exceder de los 15 N·cm en periodos prolongados (puede producir fatiga al actuador).

3.3. Los sensores

3.3.1. El sensor óptico

El sensor óptico incluye un fototransistor para la medida de iluminación, junto a un diodo LED que puede activarse o no, dependiendo de si la medición se está efectuando en un ambiente oscuro o iluminado. Según [10], el rango de medida es más amplio que el espectro visible por el ojo humano, lo que puede inducir a confusiones con fuentes de luz que no se consideren en un principio, como pilotos infrarrojos. El pico de sensibilidad se obtiene para las longitudes de onda de 900 nm, mientras que en el espectro de los 400 a los 700 oscila entre un 30 y un 60 % de sensibilidad.

El margen de medición, de nuevo según las pruebas realizadas en [10], comprende desde los 0 hasta los 1000 lux, con una curva de respuesta bastante lineal.

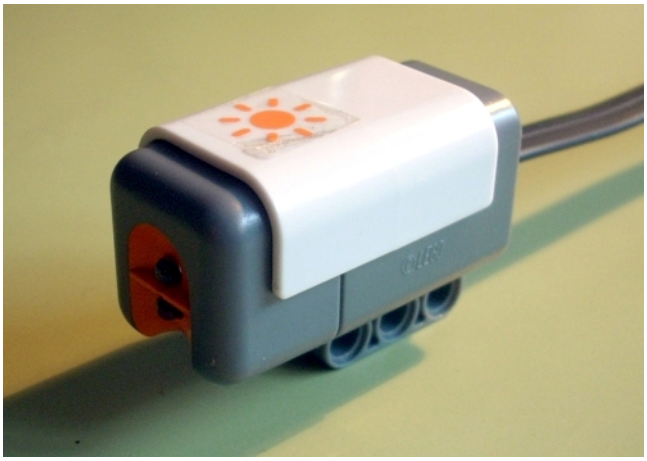


Imagen 3.5: Imagen del sensor óptico con el cable de conexión

3.3.2. El sensor acústico

El sensor acústico puede configurarse para devolver los valores de medida en decibelios (**dB**) o decibelios ajustados (**dBA**), estando esta última unidad de medida ponderada a los niveles audibles por el oído humano.

La sensibilidad máxima se encuentra en los 90 dB (aproximadamente el mismo nivel sonoro que una calle ruidosa con mucho tráfico). La respuesta a intensidades en decibelios crecientes es aproximadamente exponencial.

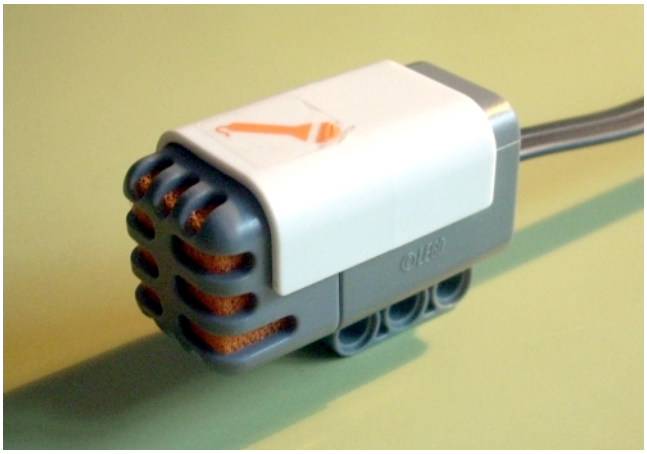


Imagen 3.6: Imagen del sensor acústico con el cable de conexión

3.3.3. El sensor de distancia

El sensor de distancia por ultrasonidos puede por sí mismo merecer un capítulo aparte en esta documentación. Su complejo funcionamiento le hace requerir su propio microprocesador para la interpretación de la señal, y dispone de un protocolo de comunicaciones desarrollado en exclusiva, por supuesto documentado por LEGO en su web (ver [4]).

Al contrario que el resto de sensores, no devuelve los valores en ninguna escala ni porcentaje, sino en unidades reales, bien centímetros, bien pulgadas. El alcance de medición comprende desde 3 cm hasta 255 cm (tiene una zona muerta en distancias muy cortas), y su funcionamiento por ultrasonidos a 40 KHz lo hace altamente efectivo para escenas relativamente simples y con objetos bien definidos, mientras que si aparecen elementos pequeños o en grandes cantidades, sus mediciones pueden no ser tan buenas. En el capítulo 6 se profundizará más sobre el funcionamiento de este importante componente.

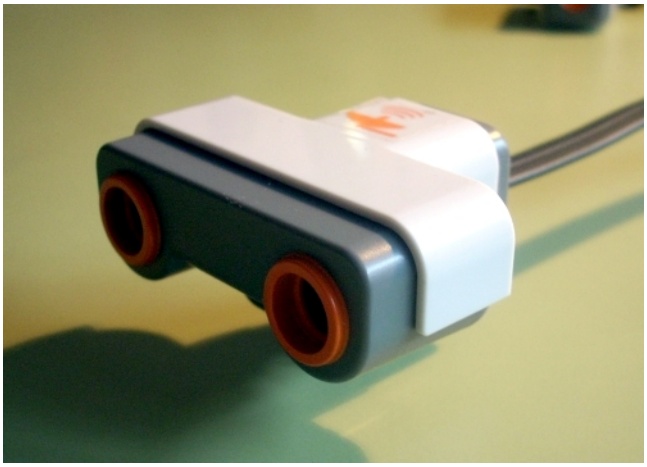


Imagen 3.7: Imagen del sensor de distancia con el cable de conexión

3.3.4. El sensor de presión

El sensor de presión simplemente devuelve una señal digital binaria cuando se activa y desactiva. Normalmente se usa en conjunción con más piezas de LEGO Technic para aumentar el alcance táctil del robot.

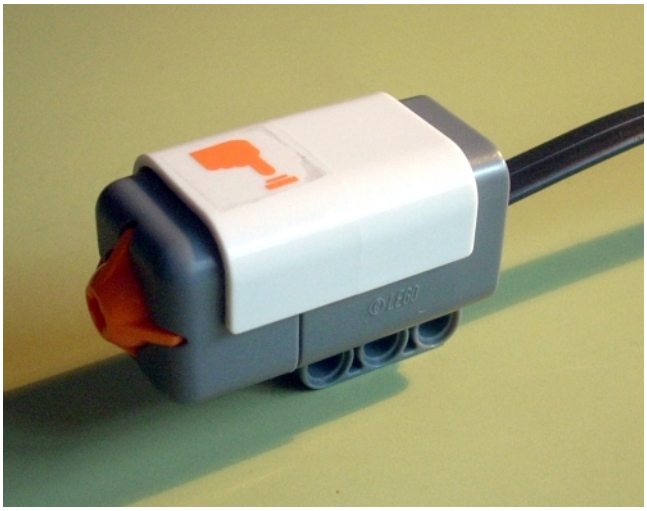


Imagen 3.8: Imagen del sensor de presión con el cable de conexión

3.3.5. Sensores HiTechnic

Para este proyecto tenemos disponibles tres sensores de la casa HiTechnic: la brújula, el acelerómetro, y el giróscopo. Los dos primeros necesitan tener instalado en el ladrillo, al menos, la versión 1.03 del firmware de LEGO.

La brújula HiTechnic

La brújula no presenta inicialmente ningún problema, salvo que hay que tratarla como un sensor LowSpeed (se comunica mediante el protocolo I2C). Al trabajar con ella, devuelve directamente un entero de 0 a 360; el valor en grados de diferencia al norte magnético. Según las instrucciones del propio fabricante, es conveniente montarlo alejado de los motores y el ladrillo inteligente (al menos 10 cm de este último y 15 de aquéllos), para evitar interferencias, y mantenerlo sujeto firme y horizontalmente, si bien, como se muestra en las pruebas del apartado 6.2, esas recomendaciones pueden quedarse un poco escasas.

HiTechnic, en toda su gama de productos, proporciona bloques de NXT-G totalmente funcionales, y además publica los mapas de direcciones de memoria a las que acceden sus sensores, para poder implementar su uso en

otros lenguajes de programación. NBC/NXC incluye ya de serie una función, `SensorHTCompass()`, que devuelve el valor numérico antes citado. Con él se puede programar un robot para que siga un rumbo, mantenga una orientación, etc. El bloque para NXT-G incluye de serie distintas funciones como rango (indica si se mantiene la lectura dentro de un rango determinado), lectura relativa (diferencia entre el valor deseado y el valor actual) y calibración (que normalmente no se usará, pero puede ser necesario en caso de que el sensor se vea expuesto a fuertes interferencias magnéticas). Estas funciones, aunque no se encuentran en NXC, pueden emularse directamente escribiendo un código que haga lo mismo.

Una imagen de la brújula se puede encontrar en el apartado 2.2.2.

El acelerómetro HiTechnic

Este sensor incorpora un acelerómetro de tres ejes, distribuidos como se muestra en la imagen 3.9, de considerable precisión: las lecturas, actualizadas cada 10 milisegundos,devuelven valores entre -400 y + 400, equivalentes aproximadamente a -2g y +2g. Opera también mediante el protocolo I2C.

La sensibilidad es muy grande comparada con el rango de actuación en que se suele mover el resto de sensores del NXT, lo que, para aplicaciones de precisión es muy útil, pero, sin embargo, puede llevar a problemas de *exceso de precisión* en aplicaciones más limitadas, que interpreten pequeñas variaciones que no se correspondan con cambios reales. Puede hacerse necesario el uso de un factor de escala que desprecie los cambios más pequeños.

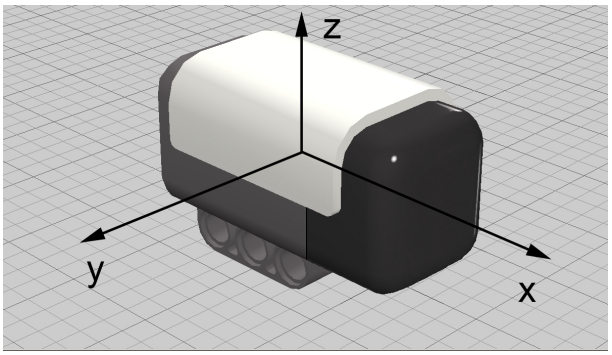


Imagen 3.9: Disposición de los tres ejes del acelerómetro de HiTechnic

HiTechnic tiene disponible, como para el resto de su línea de sensores, bloques de NXT-G para el acelerómetro y las direcciones de memoria a las que acceden los sensores (para implementaciones en otros lenguajes y sistemas), pero también está soportado en RobotC y NXC. Al programar en NXC, se utiliza la función `ReadSensorHTAccel(puerto, valorx, valory, valorz)`, que no devuelve los valores de dichas medidas en x, y y z, sino un valor tipo byte indicando si las lecturas han sido correctas o no. *valorx*, *valory* y *valorz* son enteros con signo que deben haber sido declarados antes y que pasan como argumento a la función.

Capítulo 4

Dos aplicaciones ya desarrolladas en el campo de la ingeniería de control

Aquí presentamos dos aplicaciones que ya han sido desarrolladas para LEGO Mindstorms NXT relacionadas con la ingeniería de control: NXTWay (Segway de LEGO) y APRIL (A Pid Robot Implemented with LEGO).

La mayoría del uso académico que le ha sido dado al NXT ha sido en el campo de la robótica; no en vano se trata de un autómata muy completo por un precio bastante bajo, con lo que resulta atractivo para un laboratorio de robótica al uso. Sin embargo, aquí se persigue el uso para laboratorios de control, así que para evaluar su utilidad una aplicación debe cumplir al menos los dos objetivos básicos de la ingeniería de control: el seguimiento de consigna y el rechazo a perturbaciones (o, cuanto menos, el seguimiento de consigna en lazo realimentado). Así, estos son los proyectos que más relacionados están con este campo.

4.1. APRIL: A Pid Robot Implemented with LEGO

APRIL es proyecto de Kevin MCL, estudiante de la **University of British Columbia**. Citando sus propias palabras, *APRIL es un sencillo robot de 4 ruedas programado con un controlador PID*. En la parte frontal del montaje, un sensor de distancia (Kevin usó originalmente el sensor infrarrojos de Mindsensors, ver sección 2.2.1 para más información, pero también hay una versión adaptada al estándar de ultrasonidos) mide la consigna al inicio del programa y, a partir de ahí, el robot intentará mantener esa distancia frente a las posibles variaciones que pueda sufrir. Esto se consigue con la implementación en código NXC de un lazo de control realimentado con un regulador PID.

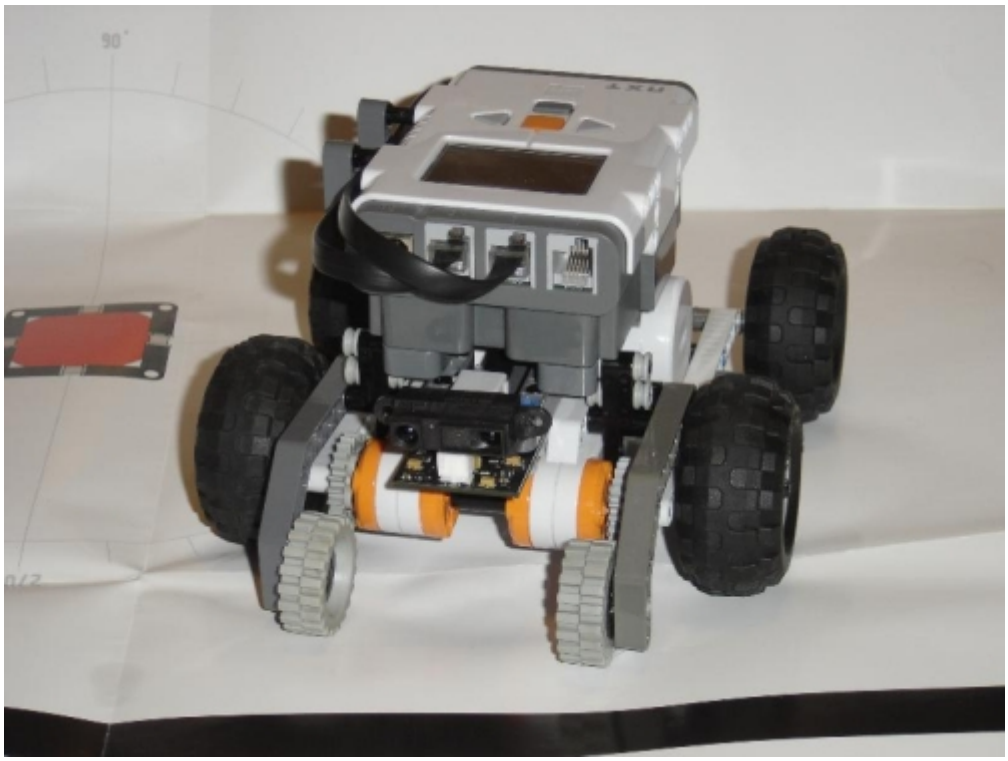


Imagen 4.1: Imagen del robot propuesto por Kevin. Nótese que el sensor de distancia no es el estándar de ultrasonidos

El punto de interés de este proyecto es la inclusión en el lazo de control de un regulador PID. La variación de sus parámetros determina cómo de

ajustada será la salida del sistema respecto a la ideal teórica. Recordemos que la ecuación que rige la ley de control de un regulador PID es de la forma:

$$U(t) = U_p(t) + U_i(t) + U_d(t) = k_p e(t) + k_i \int e(t)dt + k_d \frac{de(t)}{dt}$$

siendo $U(t)$ la salida al actuador (en este caso, los motores) y $e(t)$ la señal de error (la consigna menos la entrada el sensor, convenientemente dimensionada). El programa original incorporaba los valores de las constantes del regulador PID, K_i, K_p y K_d que mejor resultado habían dado al autor; una posterior modificación permite especificar al inicio del programa los valores de estas constantes, para comprobar cómo cambia la respuesta según se utilicen unos parámetros u otros.

El código original, descargado de la propia web del proyecto, junto con las modificaciones realizadas se puede encontrar en el **Apéndice A**.

4.2. NXTWay

Este montaje ha sido diseñado por Philippe E. Hurbain, coautor del libro Extreme NXT ([10]) y otro pilar de la escena NXT internacional. El objetivo es mantener al robot estable sobre dos ruedas, usando para ello un sensor óptico para medir las distancias y los motores para, mediante pequeños impulsos a las ruedas, estabilizarlo. En las pruebas realizadas por el propio Philippe, el NXTway se mantiene estable durante casi dos minutos, una cifra considerable habida cuenta de las limitaciones del montaje y del propio NXT.

Este NXTWay es la evolución natural del LegWay, construido en su día por Steve Hassenplug usando la anterior versión de Mindstorms, RCX (ver [13]). A raíz del éxito obtenido por ambos, diversas versiones modificadas de este balancín han sido construidas por aficionados, desde versiones con dos sensores de luz a otras con giróscopos, acelerómetros, etc. Llama poderosamente la atención, por su enfoque serio y científico, el NXTWay-GS, construido por Ryo Watanabe (ver [14]), que mediante control basado en un modelo obtenido en espacio de estados, consigue un control casi perfecto. Ryo ha modelado el sistema basándose en las ecuaciones de un péndulo invertido

para poder introducirlas en MATLAB y simularlas. Su NXTWay no sólo se mantiene estable con oscilación mínima, sino que gira, se desplaza, y sube rampas.

El código fuente del NXTWay, programado en NBC, y descargado de la web personal de Philippe (http://www.philohome.com), así como más información del NXTWay-GS, se puede encontrar en el **Apéndice B**.

Imagen 4.2: Imagen del robot propuesto por Philippe E. Hurbain

Capítulo 5

Selección de entorno y lenguaje

Dada la amplia documentación disponible, así como la sencillez (por su gran parecido con C), se ha utilizado el lenguaje de programación **NXC** y el entorno de desarrollo **BricxCC** sobre Microsoft Windows XP. Si bien existen alternativas de programación sobre sistemas operativos libres, al disponer en el laboratorio de la universidad únicamente de ordenadores con este sistema se ha decidido trabajar con él.

A continuación se describen las características fundamentales de ambos, lenguaje y entorno.

5.1. El lenguaje de programación NXC

NXC significa *Not eXactly C* (No eXactamente C), y es un lenguaje de programación de alto nivel basado en C. Está construido sobre el compilador **NBC** (anterior a NXC, de tipo ensamblador), y es la evolución natural del anterior lenguaje de programación **NQC** (Not Quite C), usado en las primeras versiones de LEGO Mindstorms, los RCX. Su creador es John Hansen, ingeniero de software y ampliamente conocido en la escena Mindstorms internacional, además de miembro del Mindstorms Developers Program (LEGO MDP, ver [15]).

El clásico programa *Hello, World!* quedaría así en NXC:

```
task main()
{
  //Muestra el texto en la pantalla LCD
  TextOut(0, LCD_LINE1, "Hello, World!");

  //Espera 1000 milisegundos
  Wait(1000);
}
```

De un rápido vistazo a la bibliografía disponible gratuitamente en internet se comprueba que, en efecto, NXC es *casi* C. Las características propias más destacadas son:

- Cada programa consta de un número determinado de tareas (denominadas task), al menos una de las cuales deberá ser la principal (task main()), de la misma manera que en C. Un programa estándar soporta hasta 256 tareas, según [3].

- NXC dispone de su propia biblioteca de funciones adaptadas al NXT. En el ejemplo *Hello, world!*, la función `TextOut` escribe texto en la pantalla LCD del ladrillo, aceptando argumentos como la posición horizontal, vertical, y el propio texto. Nótese que, además de valores absolutos, (el primer argumento es un `0`, que indica que el texto debe aparecer en la posición X inicial, a la izquierda), se permiten una serie de alias (el segundo argumento, `LCD_LINE_1`, indica que el texto debe aparecer en la primera línea, sin especificar el píxel concreto donde empieza). En [3] se indican todos los argumentos que acepta cada función, así como los alias disponibles.

- Para evitar conflictos en la ejecución simultánea de diversas tareas, existe una variable llamada **mutex**, que con las funciones **acquire** y **release** (adquirir y liberar), aseguran que el fragmento de código entre las dos se va a ejecutar con prioridad frente al resto, al estilo de la programación concurrente (por ejemplo, si una tarea ordenara a un motor girar en un sentido, y otra en el contrario, la tarea que tuviera el mutex adquirido sería la única que tuviera efecto).

El resto es tremendamente accesible si se maneja C con un mínimo de fluidez. Se pueden declarar subrutinas para ser llamadas luego, se pueden redefinir las llamadas de bajo nivel al NXT, e incluso se pueden incorporar fragmentos de NBC dentro de un código NXC. Es, como vemos, un lenguaje muy potente a la par que asimilable con unos conocimientos mínimos de programación.

Para aprovechar al máximo la capacidad de NXC existe un firmware mejorado, disponible de nuevo gratuitamente en la web del proyecto, que añade posibilidades como el uso de matrices multidimensionales o el acceso al cuarto botón del ladrillo mientras se está ejecutando un programa[1].

5.2. El entorno integrado de desarrollo BricxCC

Habiendo elegido NXC como el lenguaje para progamar el NXT, el siguiente paso lógico es hacerlo usando el IDE BricxCC, tremendamente completo y cómodo de utilizar.

BricxCC también es una creación de John Hansen, evolución de la anterior RcxCC (para el RCX). El espacio de escritura dispone de resaltado de sintaxis, autocompletado, ayudas al indentado, marcado y numerado de líneas, etc., que ayudan a mejorar la comprensión del código mientras se va escribiendo. Asimismo, el software incluye un explorador de código que muestra la estructura del programa, muy útil en proyectos complejos; y una guía con todas las funciones y estructuras disponibles en NXC (que es, en realidad, la Guía del Programador escrita por el mismo John Hansen, [3]). Para facilitar la labor al programador que se está iniciando en NXC, cada función indica el tipo de dato que acepta como argumento al iluminarla con el cursor, y en la ayuda se puede encontrar una completa guía de referencia.

Además de las facilidades de programación descritas, BricxCC incorpora otras utilidades, algunas tan útiles e interesantes como:

- Manejador de macros, para usar cómodamente expresiones regulares

[1] Con el firmware original, pulsar el botón de salida durante la ejecución de un programa implica su terminación instantánea. Utilizando el firmware mejorado, es posible utilizar este botón como cualquier otro, siendo necesaria una pulsación prolongada para detener la ejecución

- Ventanas de diagnóstico del estado del NXT
- Un piano, con teclado en pantalla, que utiliza el altavoz integrado del ladrillo para reproducir los sonidos
- Una suerte de *joystick en pantalla*, con una serie de controles que permiten desplazar e interactuar con el robot a voluntad
- Un explorador de la memoria del ladrillo, que permite acceder a su sistema de archivos y operar con él como si se tratara de una carpeta más de nuestro equipo, con posibilidades de borrado, carga y descarga de archivos, desfragmentación de la memoria, etc.
- Un visor de la pantalla del ladrillo
- Visores y organizadores de la memoria del ladrillo.
- Y todavía más...

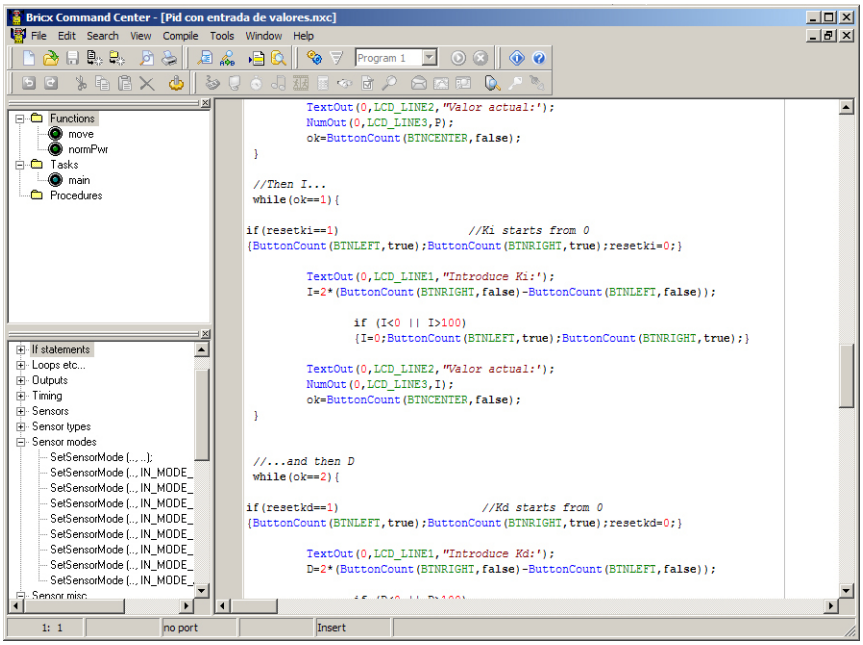


Imagen 5.1: Captura de pantalla del IDE BricxCC

Capítulo 6

Ensayos de laboratorio

Dado que el objetivo principal de este proyecto es analizar la idoneidad del robot educativo LEGO Mindstorms NXT para el uso docente, se desprende como lógica consecuencia el análisis de los elementos que lo conforman y de sus propiedades. En este capítulo se abordan diferentes ensayos y pruebas de laboratorio realizados a los sensores y actuadores incluidos en la presentación comercial del NXT.

6.1. El sensor de ultrasonidos

En robótica móvil, autónoma o semi autónoma, sin duda la información de entrada desde el mundo exterior más importante es la detección de objetos y la medida de distancias. En el NXT, esta función se realiza principalmente con los sensores óptico y de ultrasonidos, siendo este último mucho más complejo, preciso, y de mayor alcance, y por tanto merecedor de un estudio más pormenorizado.

El principio de funcionamiento de los ultrasonidos es bien sencillo y está presente en la naturaleza, en especies animales como murciélagos o delfines. Electrónicamente hablando, un sensor de ultrasonidos como el que nos ocupa está compuesto de las siguientes partes: emisor, receptor, y unidad de control que procesa los resultados. El emisor produce una onda sonora de alta frecuencia (como mínimo 20 KHz) que reflecta en los objetos que se en-

cuentren a su paso. La medida del tiempo transcurrido entre que la onda es emitida y su reflejada es detectada por el receptor da una idea de la distancia a la que se encuentra el objeto en cuestión. En el sensor de ultrasonidos incluido al comprar el NXT, el emisor y el receptor son claramente visibles como dos *ojos* orientados en la misma dirección (de hecho, la intención de los diseñadores de LEGO es hacerlo parecer una cabeza al estilo de los clásicos robots de películas como *Cortocircuito*), separados unos milímetros. Encapsulado se encuentra también el microprocesador y el sistema de transmisión responsable de hacer llegar las medidas a la CPU principal.

El sensor de ultrasonidos puede, pues, funcionar para dos objetivos: la detección de objetos y la medida de distancia a éstos. Por la propia naturaleza de los ultrasonidos, emitidos como una onda, la medida del sensor es sensible a los obstáculos que se encuentren en las inmediaciones del objeto cuya distancia hasta el emisor se desea medir, ya que, como veremos en las pruebas, pueden reflejar la onda y producir resultados falseados. Por contra, este problema se vuelve beneficioso a la hora de detectar objetos, ya que permite un rango mayor de sensibilidad.

6.1.1. Medida de distancia

A continuación se exponen los resultados de las pruebas realizadas para medida de distancia usando el sensor de ultrasonidos en un entorno controlado, con objetos adecuados (sólidos, no reflectantes, y suficientemente poco porosos). Todas las mediciones de este apartado han sido obtenidas utilizando el programa `pruebaultrasonidos.nxc`, reflejado también en el **Apéndice C**.

Según las propias especificaciones de LEGO, disponibles en su web ([4]), el sensor de ultrasonidos tiene un alcance de 0 a 255 cm, con una precisión en la medida de ± 3 cm. Como veremos, para distancias cortas esta tolerancia es casi nula, mientras que para distancias más largas se incrementa.

La primera prueba consiste en la medición de distancia a un objeto de 20 cm de ancho por 5 de alto, en un rango de 0 a 100 cm, por pasos de 1 cm. La altura del sensor sobre el plano es de 5 cm. Como se aprecia en la imagen 6.1, a excepción de casos puntuales, la desviación de la medida respecto a la distancia real no sobrepasa de ordinario los ± 2 cm. Alrededor de los 80 cm, sin embargo, se produce una fuerte imprecisión en la medida, que en

posteriores respeticiones del experimento no se ha producido.

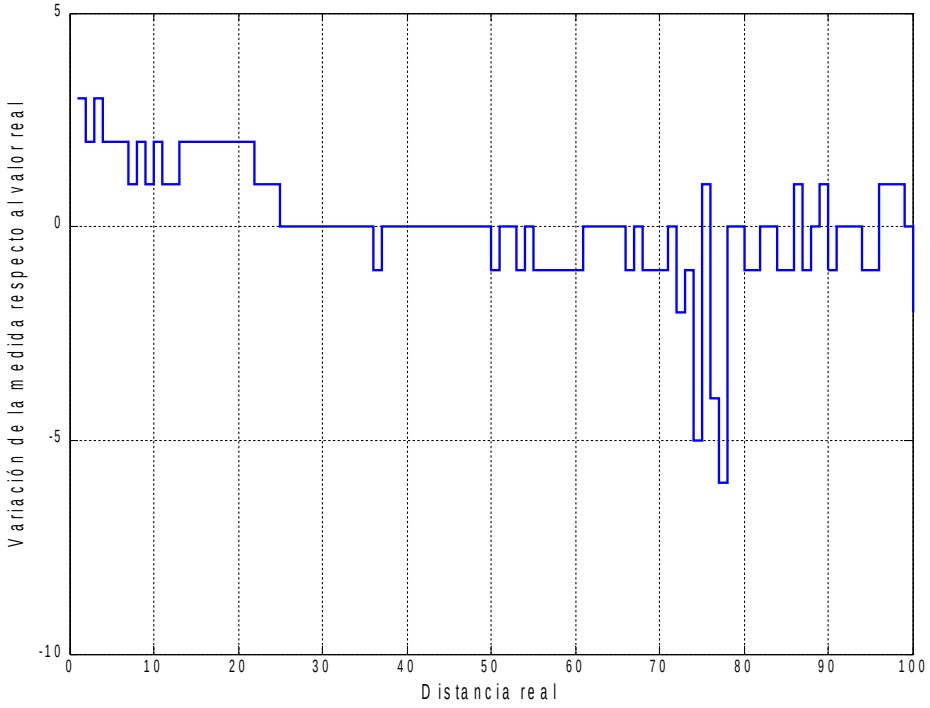


Imagen 6.1: Gráfico que muestra la variación de la distancia medida frente a la real en los primeros 100 cm

A partir de los 100 cm de medición el reducido tamaño del objeto frente a otros de su entorno hace necesario, o bien eliminar todo obstáculo, o bien aumentar, proporcionalmente, el volumen de aquél. Los resultados mostrados en la gráfica 6.2 han sido obtenidos en un entorno libre de obstáculos próximos utilizando un objeto de 40 cm de ancho por 20 de alto, manteniendo la altura del sensor sobre el plano de 5 cm. La medida cubre desde los 100 cm hasta los 260 cm, por pasos de 10 cm, y como se comprueba, en las distancias más grandes el sensor pierde fidelidad. De nuevo se producen fuertes imprecisiones en la medida al final del rango de medición. Para mejorar esta respuesta es necesario utilizar un objeto a medir más grande.

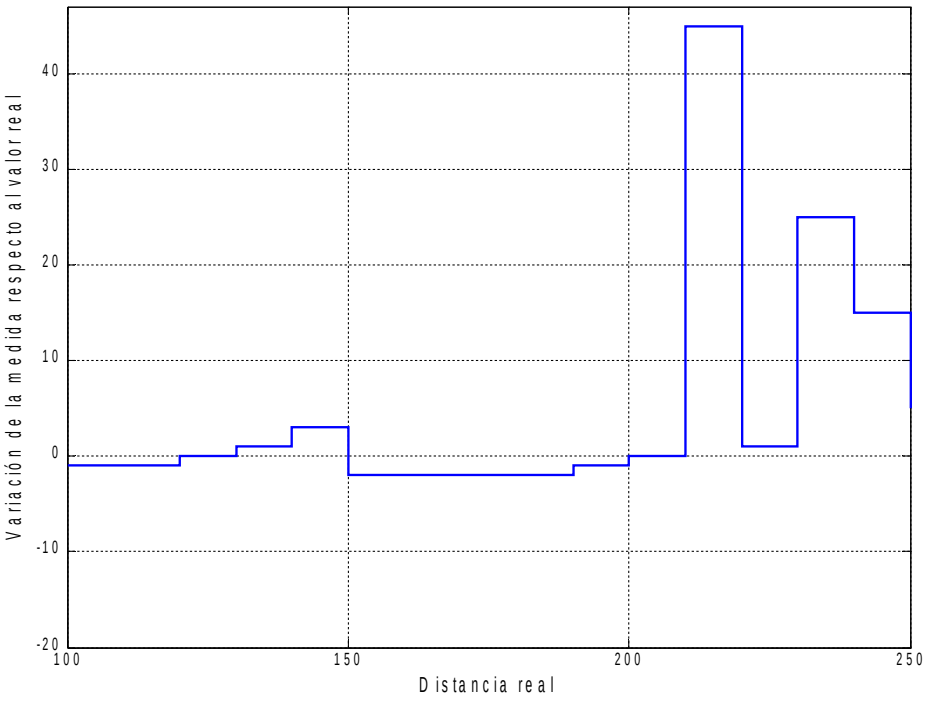


Imagen 6.2: Gráfico que muestra la variación de la distancia medida frente a la real desde los 100 hasta los 250 cm

6.1.2. Fidelidad de la medida con obstáculos

Como se ha comentado anteriormente, la presencia de obstáculos en los laterales del rango de medición del sensor puede entorpecer la adecuada propagación de la onda y falsearse así la medida. Un obstáculo demasiado próximo al sensor podría reflejar prematuramente la señal de ultrasonidos y devolver, así, valores equivocados.

En la gráfica 6.3 se muestra la diferencia de valores que se obtienen al alejar, desde el contacto directo con el sensor hasta los 25 cm de distancia, por pasos de 5 cm, un obstáculo liso y continuo en el lateral de la zona de medición (asimilable a medir distancias cerca de una pared).

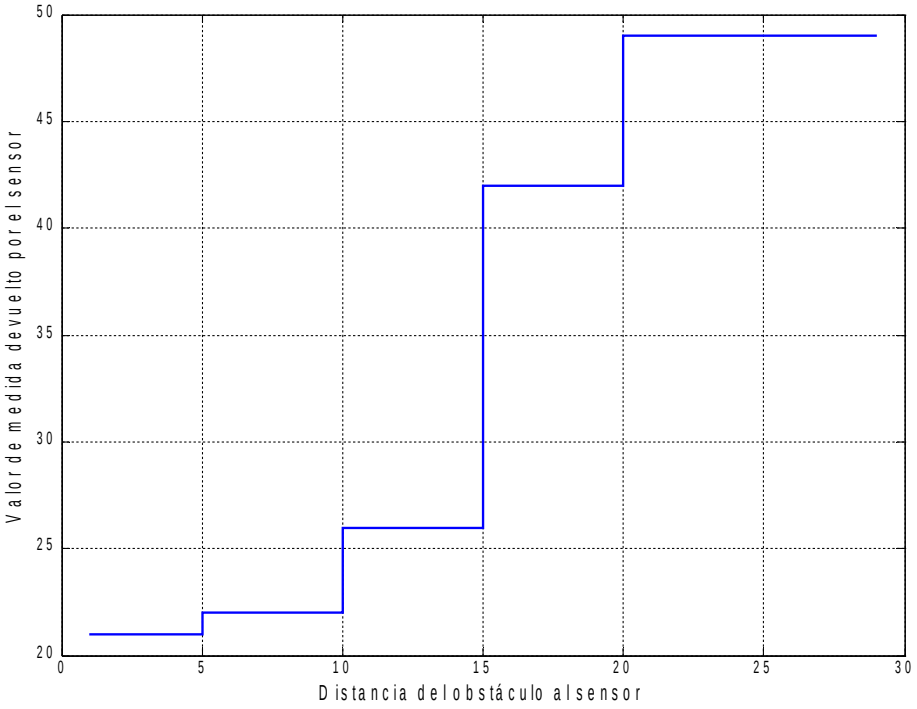


Imagen 6.3: Gráfico que muestra la distancia devuelta frente a la distancia del obstáculo al sensor

La distancia real a la que se encontraba el objeto es de 50 cm. Como se aprecia, si el obstáculo está suficientemente lejos del sensor, la medida no se ve apenas alterada, mientras que, conforme se acerca, el valor devuelto es cada vez menor, a consecuencia del efecto intrusivo del obstáculo. Esto ha de ser tenido en cuenta al diseñar utilizando el sensor de ultrasonidos, ya que mantener siempre el robot en un espacio libre de elementos ajenos al propósito original no siempre es fácil, máxime si estamos hablando de robótica móvil autónoma.

6.2. La brújula de HiTechnic

La brújula de HiTechnic, anteriormente descrita, tiene la forma básica del sensor NXT, dado que el hardware de esta casa está certificado por la propia LEGO. Debe operar horizontalmente, como las brújulas tradicionales, y mantenerse alejado de todo tipo de perturbación magnética (imanes, dispositivos electrónicos mínimamente complejos, motores, etc.). En las primeras medidas recién salido de la caja se obtienen resultados satisfactorios en cuanto a precisión, repetibilidad y veracidad, siempre y cuando se cumplan los requisitos recomendados por HiTechnic.

Con objeto de comprobar hasta qué punto interfieren elementos externos en las medidas devueltas por la brújula, se ha estudiado la desviación de la respuesta en función de la proximidad a dos elementos que podrían ser conflictivos: el ladrillo inteligente y un motor del NXT. Otros elementos externos no se han tenido en cuenta porque son dependientes del montaje y la escena que se haya escogido, y porque tanto un dispositivo electrónico como un motor son los dos principales objetos que podrían ser un problema en cualquier situación, dado que su uso es inevitable en el ámbito que nos ocupa. Todas las mediciones de este apartado han sido obtenidas utilizando el programa `medidabrujula.nxc`, reflejado también en el **Apéndice C**.

Las pruebas se realizaron orientando el sensor hasta que marcara 0^o (orientación Norte), en un entorno libre de cualquier obstáculo en un radio esférico de 30 cm, y con la brújula en posición horizontal. Al ir acercando el ladrillo y tomando medidas cada centímetro, en la imagen 6.4 se aprecia que a partir de los 16 cm de distancia la medida comienza a distorsionarse añadiendo entre 1 y 29 grados. Al situar el ladrillo directamente bajo el sensor, la medida se altera hasta los 340 grados, o lo que es lo mismo, 20 grados menos que en la primera medida.

Es conveniente añadir que HiTechnic recomienda operar con el sensor alejado al menos 10 cm del ladrillo, pero según las pruebas aquí realizadas la distorsión puede afectar hasta a 16 cm. En cambio, la recomendación de mantener el sensor al menos a 15 cm de los motores en movimiento del NXT sí es correcta, dado que, como se muestra en la figura 6.5, se producen distorsiones hasta esa distancia.

En el caso del motor en movimiento las perturbaciones son en general más acusadas y de signo negativo, con lo que la medida se atenúa entre 1

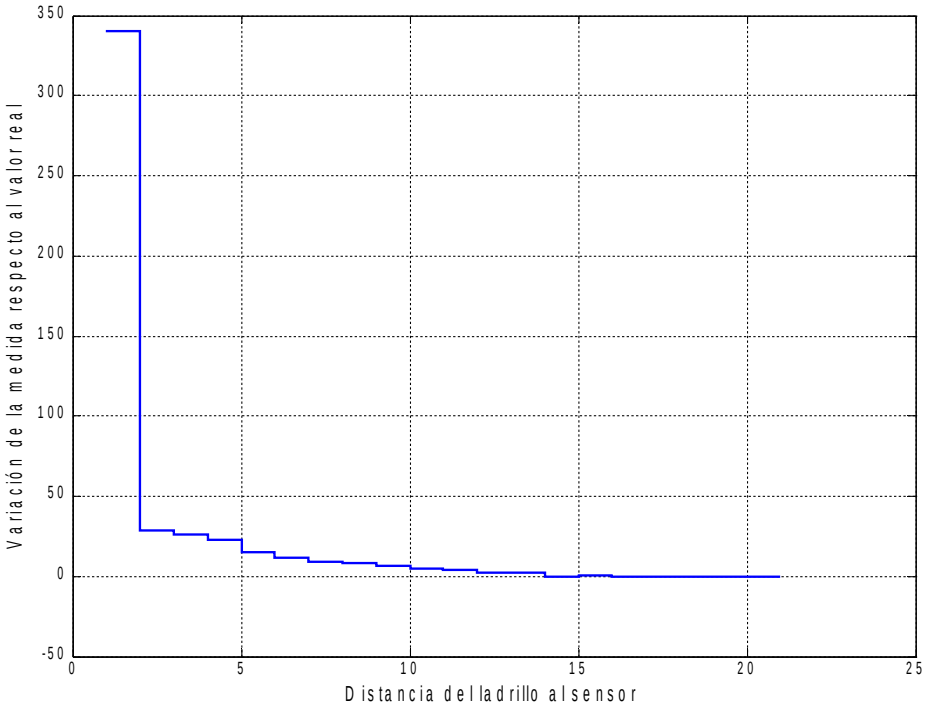


Imagen 6.4: Gráfico que muestra la variación de la distancia medida frente a la distancia del ladrillo inteligente al sensor

y 42 grados, esta última al situar el motor directamente bajo el sensor. La distorsión al situar el motor en contacto con el sensor no es tan abrupta como en el caso del ladrillo.

Después de efectuar toda esta tanda de mediciones con interferencias externas, e incluso varias veces, las medidas devueltas por la brújula al situarla de nuevo en un ambiente adecuado son las mismas que antes del experimento, por lo que podemos asumir que las perturbaciones originadas por los elementos estudiados no afectan permanentemente al sensor ni desvirtúan su funcionamiento normal.

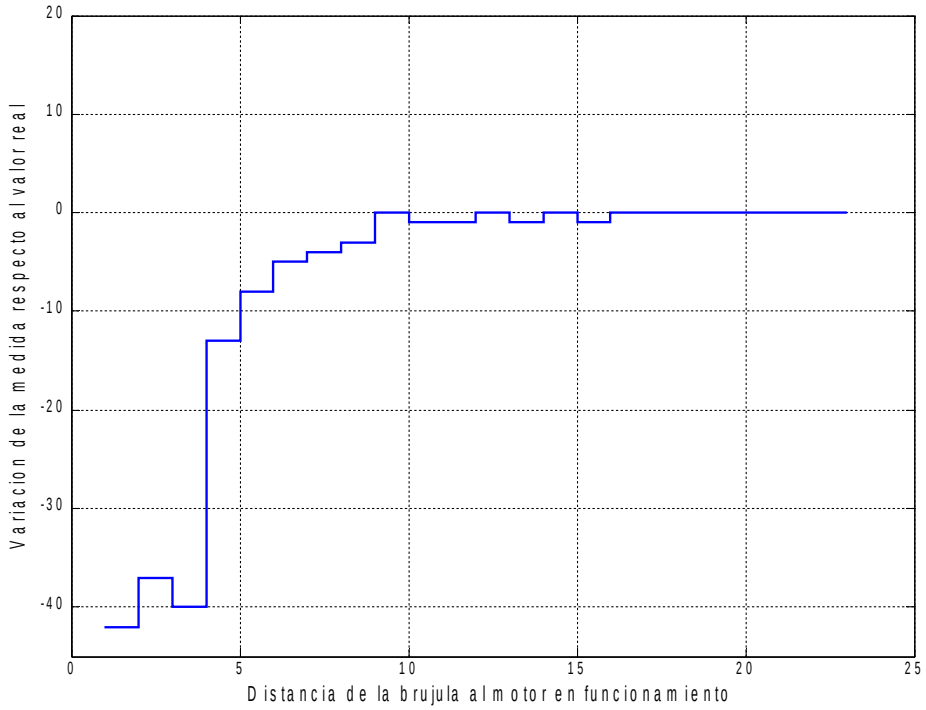


Imagen 6.5: Gráfico que muestra la variación de la distancia medida frente a la distancia del motor al sensor

6.3. Escritura a archivos

Estrictamente hablando, la escritura a archivos no ha precisado de pruebas de laboratorio como tales, sino de trabajo de programación pura y dura, pero por su importancia y por la necesidad de una guía de referencia a la hora de crear registros, habida cuenta de los problemas que surgen, se ha incluido en esta sección.

En cualquier aplicación destinada a ingeniería, docencia, laboratorios, etc., es fundamental un registro de datos de cualquier experimento, para posteriormente trabajar con él. Así, crear archivos con mediciones, velocidades, y otros parámetros, mientras se ejecuta un programa en el NXT ha sido prioridad desde el principio de este proyecto. De hecho, todas las gráficas aparecidas aquí provienen de datos registrados durante la ejecución de

diversos programas, y recuperados tras concluir ésta.

La API[1] del NXT permite crear, renombrar, buscar, y eliminar archivos de su sistema; archivos que son almacenados en la memoria Flash con el único límite del espacio disponible en ésta (que variará en función del número de archivos). NXC provee de diversas funciones con este fin; una mirada rápida al tutorial de este lenguaje ([2]) basta para hacerse una idea de cuáles pueden ser los primeros pasos para crear, escribir, y cerrar un archivo, pero para exprimir al máximo el potencial del sistema de archivos del NXT es necesario ir un poco más allá.

El primer paso para crear y editar un archivo de texto en NXC es conocer sus requisitos: un nombre, un tamaño, y una etiqueta (o *manejador*).

- El nombre, junto con su extensión (opcional, ya que todos los archivos son de texto plano, y así son tratados por el NXT), es lo que veremos al navegar por el ladrillo desde nuestro ordenador, y no debe ser más largo que 19 caracteres (extensión incluida).
- El tamaño debe ser especificado, porque el NXT es incapaz de manejar archivos de tamaño variable; de hecho, si el archivo se llena y un programa ordena seguir escribiendo en él, se producirá un error. Más adelante se contemplan maneras de evitar esto.
- El manejador (*handle*, en inglés), es la etiqueta que usaremos para referirnos al archivo asociado durante la programación. Debe ser una variable.

Tomando estos requisitos y condensándolos en una función, escribimos en nuestro código: `CreateFile(‘‘nombre.ext’’, tamaño, etiqueta)`. Puede ser conveniente dar orden de borrar primero todos los archivos con el mismo nombre, por si existieran, dado que si ya existe no se creará. Incluir un `DeleteFile(‘‘nombre.ext’’)` al principio del código no es mala práctica, aunque siempre es mejor comprobarlo manualmente. Para terminar, y antes de realizar cualquier operación con el archivo que no sea escribir en él, hay que cerrarlo, evidentemente con un `CloseFile(etiqueta)`.

Todas las funciones del sistema de archivos devuelven el resultado de la

[1] **A**pplication **P**rogramming **I**nterface, Interfaz de programación de aplicaciones.

operación: exitoso, o los diversos errores (no hay espacio, archivo no encontrado, etc.). La lista completa de los errores se haya en [3].

En principio, podemos considerar que el único error que merece la pena tener en cuenta es el de falta de espacio en el archivo, dado que es el que más problemas puede dar. Si se diera el caso de tener que almacenar una gran cantidad de datos, podría ser difícil estimar el tamaño del archivo a contenerlos. Utilizando un código como el siguiente, nos garantizamos que ningún dato se perderá y que no malgastaremos espacio.

El código también se incluye en el archivo `codigoarchivo.nxc`

```
/*
 *   Autor: Guillermo Nieves Molina
 *   Proyecto fin de carrera: Estudio de las posibilidades didacticas en
 *   ingenieria de control del LEGO Mindstorms NXT
 *   Año 2008
 *
 *   Este programa es una solucion al problema del espacio al escribir a
 *   archivos en el NXT: cuando el espacio del archivo se acaba, crea uno
 *   consecutivo y sigue escribiendo en el.
 *
 *   Este codigo se distribuye libremente bajo licencia GNU GPL; una copia
 *   completa de dicha licencia se puede encontrar en
 *   http://www.gnu.org/licenses/gpl.txt
 */


//Variables globales
int Contador=1;

string NumeroAsignado;

/*Funcion que devuelve un string con el nombre del archivo en forma de numero,
empezando por el 1

Los archivos no tienen extension porque el NXT los trata igualmente como
archivos de texto plano.*/

string Asignador()
{
  NumeroAsignado=NumToStr(Contador);
  Contador++;
  return NumeroAsignado;
}



task main(){

//Variable usada para almacenar el nombre del archivo, dado que la funcion
CreateFile no acepta otras funciones como argumento en el campo del nombre
string Nombre;

//Variables usadas para manejar los archivos
byte Etiqueta;
short Resultado;

//Variable usada en la eliminacion de archivos
string tmp;

//Variable que almacena el codigo del error
int Error=0;

/*Borramos los archivos previamente creados con este programa Esto se ha
incluido al ser un ejemplo, pero es conveniente comprobar manualmente la
existencia de archivos antes de ejecutar ningun código*/

for(int i=1; i<=6; i++)
{
 tmp=NumToStr(i);
 DeleteFile(tmp);
}
```

```
//Creamos 5 archivos
while(Contador<=5){

  //Creamos el archivo llamando a la funcion para el nombre
  Nombre=Asignador();
  CreateFile(Nombre,500,Etiqueta);

      //Escribimos la secuencia "esto es una prueba" hasta que se acabe el
espacio
      until(Error!=0)
      {
      Error = WriteLnString(Etiqueta, "esto es una prueba" , Resultado);
      }

  CloseFile(Etiqueta);

  /*En principio solo se tiene en cuenta la existencia de error, no el tipo de
  este. Se podria añadir una funcion que identificara el error segun el codigo
  y actuara en consecuencia*/

  //Reiniciamos la variable que almacena el cogido del error
  Error=0;

  }


}
```

El funcionamiento del código es bien sencillo: Se crea un archivo con un nombre que es simplemente un número natural, para simplificar, y se escribe en él *esto es una prueba* hasta que devuelve un código de error distinto de cero, señal de que se ha acabado el espacio. A continuación, se crea otro archivo de nombre el número consecutivo y se continúa escribiendo *esto es una prueba*, así hasta cinco veces.

Algunas consideraciones al programa:

- La función que asigna nombres a los archivos, `string Asignador()`, tiene que devolver un valor de tipo **string**, para que al utilizarlo como nombre de archivo, éste se corresponda al valor deseado. Si utilizáramos una función numérica, los nombres de archivo serían símbolos ilegibles.
- El programa sólo contempla la posibilidad de que el error sea distinto de cero, pero no distingue entre los diferentes errores posibles. A discreción del lector queda completar el programa con, por ejemplo, una estructura `switch - case`, que reconozca los diferentes códigos. Un punto importante a tener en cuenta es que los códigos de error son devueltos por el programa en decimal, mientras que en la tabla de [3] están en hexadecimal, con lo que habrá que convertirlos. Por ejemplo, el error por falta de espacio está tipificado como 0x8400, mientras que el programa devuelve -31744, es decir, el mismo valor, en notación decimal.
- En el programa sólo se incluye una etiqueta para manejar los archivos, dado que en él nunca habrá abierto más de uno a la vez. Opcionalmente se puede trabajar con diversos archivos a la vez (hasta cuatro), pero utilizando para cada uno su correspondiente etiqueta. Si esto se hace, no se debe olvidar cerrarlos todos al final de la ejecución.

6.4. Los motores

El estudio de los motores es de gran importancia dado que son virtualmente los únicos actuadores de los que se dispone al trabajar con el NXT. Como ya se ha descrito anteriormente, cada uno de estos motores incluye un prodigioso mecanismo de reducción de velocidad por engranajes, así como un tacómetro incorporado, imprescindible para obtener realimentación. A continuación veremos las diferentes maneras de operar con ellos.

6.4.1. Manejo de motores

El lenguaje NXC incluye una colección de funciones que cubren prácticamente cualquier posibilidad a la hora de utilizar los motores, así como una serie de alias para identificarlos. Los motores pueden ser accionados individualmente, por parejas, o todos a la vez, y se puede ordenar su giro de distintas formas:

- Las funciones `OnFwd(puerto, potencia)` y `OnRev(puerto, potencia)`, que toman como argumento tanto el/los puertos a los que están conectados los motores como la potencia suministrada (un entero de 1 a 100, equivalente al porcentaje de alimentación), simplemente encienden los motores y los actúan hasta que otra función les indique lo contrario: `Off(puerto)` (parada en seco) o `Float(puerto)` (corte de alimentación). Al usar estas funciones, es necesario añadir después un tiempo de espera o un condicional para especificar el tiempo que van a estar girando; si el código simplemente continúa a los motores no les dará tiempo a actuar.

  Existen, además, variantes de estas funciones (todas ellas documentadas en [3]) para hacer que los motores giren sincronizados, reseteen la cuenta del tacómetro tras el giro, etc.

- Las funciones `RotateMotor(puerto,potencia,ángulo)` y `RotateMotorPID (puerto,potencia,ángulo,p,i,d)` giran el eje del motor un número específico de grados, la primera usando el regulador PID que el ladrillo aplica por defecto; y la segunda dando opción a especificar los valores de los parámetros de dicho regulador. Esta última función resulta bastante interesante dado que la sintonía de fábrica que trae el PID en cuestión no es excesivamente buena, con lo que, bien para mejorar la respuesta de los motores, bien para ilustrar el funcionamiento de un regulador PID, es útil poder configurar manualmente los valores de sus parámetros.

6.4.2. Comportamiento de los motores según el regulador

A continuación se muestran una serie de gráficas que recogen la respuesta del motor ante un giro de 270 grados, tanto en sentido horario como anti-

horario, en función de los diferentes valores de regulación. En el eje Y de las gráficas se consignan los grados de giro del eje, mientras que en el X se cuantifican las muestras de encoder óptico. Los tiempos de giro no se corresponden a muestras de dicho tacómetro, dado que éste actúa según el giro del eje sin tener en cuenta la velocidad, pero dichos tiempos pueden calcularse con precisión de un milisegundo usando el reloj interno del NXT: al tomar una lectura del tiempo justo antes de ordenar el giro y compararla con la lectura inmediatamente posterior, obtendremos el tiempo necesario para dicho giro (o para cualquier situación en que se quiera utilizar). Todas las mediciones de este apartado han sido obtenidas utilizando el programa `pruebamotores.nxc`, reflejado también en el **Apéndice C**.

Como se comprueba en la gráfica siguiente, los valores de los parámetros P, I y D incluidos por defecto por LEGO no son los más adecuados que cabría esperar: la salida presenta mucha oscilación antes de estabilizarse, y cuando lo hace, normalmente tiene un apreciable error estacionario. Estos valores son: 40 para el término proporcional, 20 para el término integral, y 100 para el término derivativo. El pequeño término integral puede ser responsable de ese permanente error estacionario, y el, proporcionalmente, enorme valor del término derivativo puede contribuir a la excesiva oscilación de la salida.

Tras una considerable batería de pruebas, los valores óptimos que aseguren un error estacionario nulo o casi nulo (como mucho un grado), así como la menor oscilación y sobrepico posibles, se han fijado en 50 para el término proporcional, 75 para el integral, y 80 para el derivativo. Pequeñas variaciones de estos parámetros dan buenos resultados también, aunque sin mejoras apreciables. Mantener un valor elevado en el término derivativo es necesario para aumentar la rapidez de respuesta, aun a costa de mantener un sobrepico relativamente grande (en torno al 10 %), y el mayor peso del integrador garantiza esa precisión antes comentada.

A continuación se incluyen también una serie de gráficas con las respuestas de los motores a diferentes potencias al configurar distintos juegos de parámetros en el regulador PID. El ángulo de giro deseado es también 270 grados.

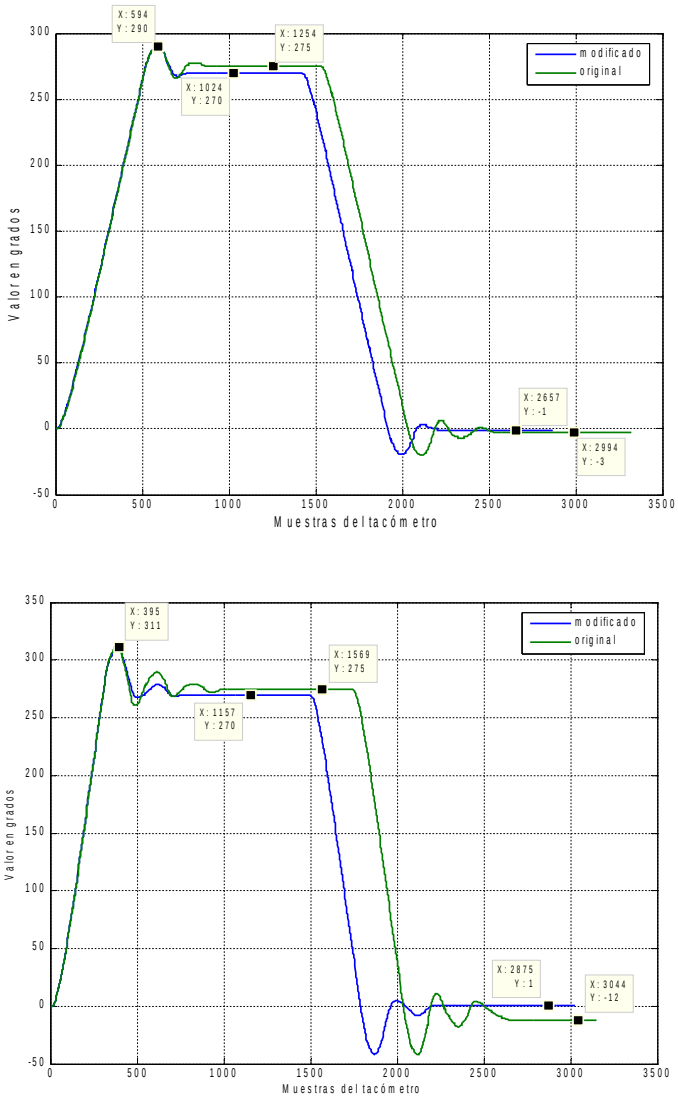
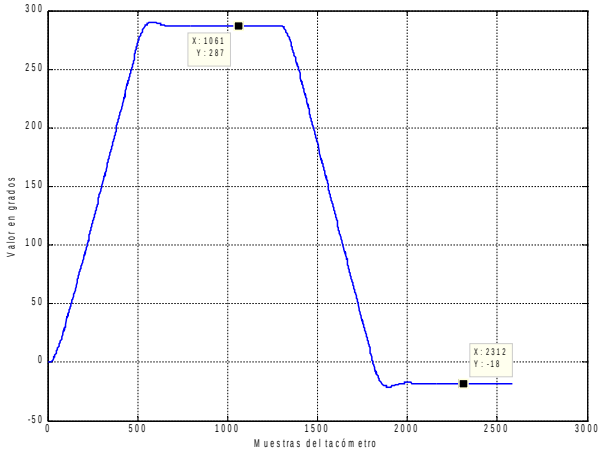
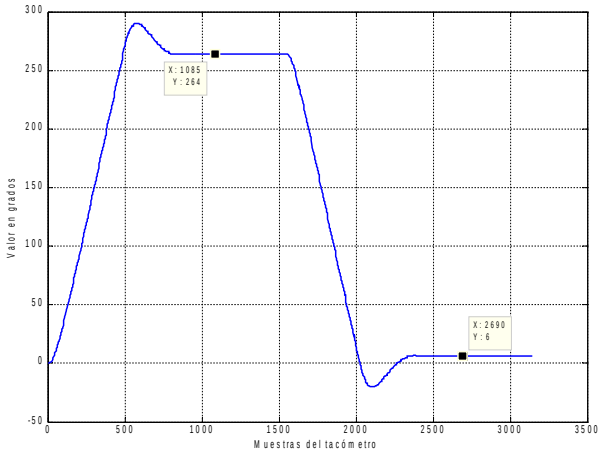


Imagen 6.6: Gráficos con los mejores resultados.

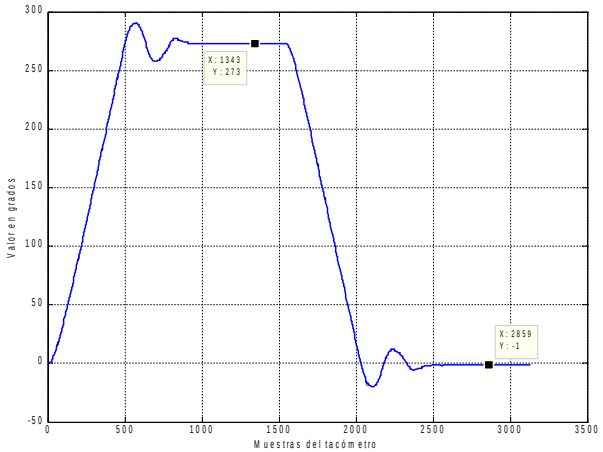
En verde, la curva de respuesta con los parámetros originales. En azul, los modificados. La primera imagen corresponde al motor al 50 % de potencia, con unos parámetros de: 50 (P), 75 (I) y 80 (D). La segunda corresponde al motor al 90 % de potencia, con los parámetros 45 (P), 60 (I), y 95 (D).
En ambos se consiguen buenos resultados.



Motor al 50 % de potencia, con unos parámetros de: 50 (P), 35 (I) y 5 (D)

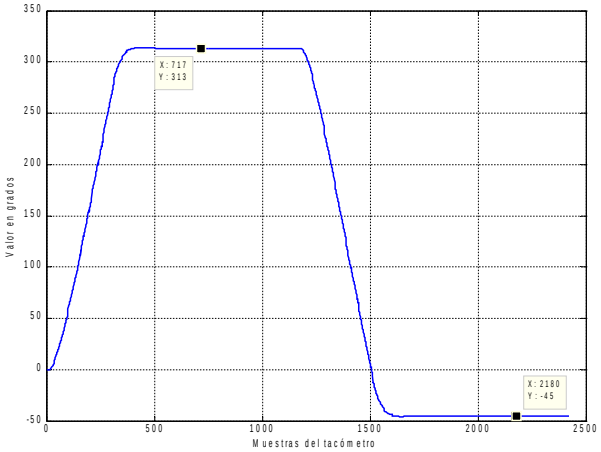


Motor al 50 % de potencia, con unos parámetros de: 60 (P), 70 (I) y 5 (D)

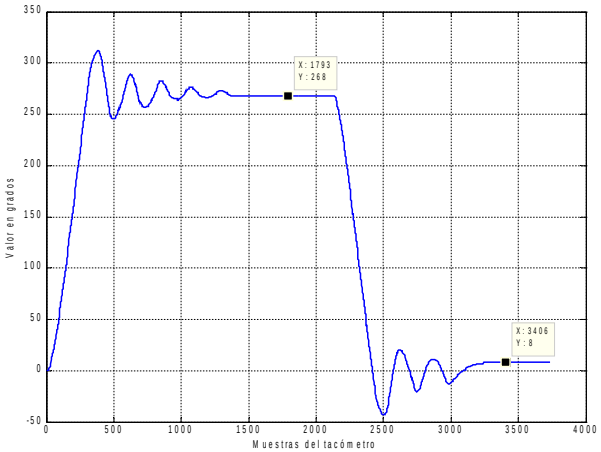


Motor al 50 % de potencia, con unos parámetros de: 100 (P), 65 (I) y 30 (D)

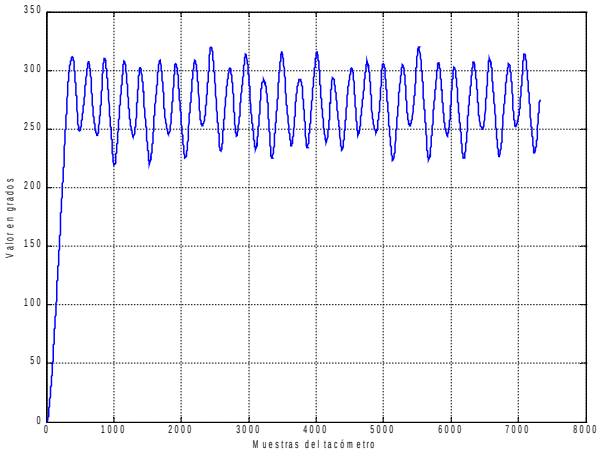
Imagen 6.7: Gráficos de las respuestas con otros parámetros al 50 % de potencia



Motor al 90 % de potencia, con unos parámetros de: 20 (P), 20 (I) y 5 (D)



Motor al 90 % de potencia, con unos parámetros de: 120 (P), 90 (I) y 85 (D)



Motor al 90 % de potencia, con unos parámetros de: 500 (P), 650 (I) y 875 (D)

Imagen 6.8: Gráficos de las respuestas con otros parámetros al 90 % de potencia

Capítulo 7

Aplicaciones

En este capítulo se desarrollan dos aplicaciones que pueden resultar de utilidad en los términos que aquí nos ocupan. Evidentemente, aquí se plantea un acercamiento que consideramos el más apropiado, pero es sólo un ejemplo de cómo se podría llevar a la práctica en la docencia de control el uso de estos robots. Desde simples lazos realimentados a sistemas con reguladores PID, como los que se plantean aquí, las posibilidasdes se extienden a complejos sistemas de múltiples entradas y salidas, o a modelados por ordenador implementados directamente en el NXT.

7.1. Revisión del NXTWay

El NXTWay, como ya se ha comentado en el capítulo 4 y ampliado en el **Apéndice B**, consiste en un montaje capaz de mantenerse estable sobre dos ruedas, usando para ello una acción de control. La información sobre el estado del robot puede obtenerse de un buen número de formas, desde sensores de distancia o luz a giróscopos, proporcionando estos últimos normalmente mejores resultados.

Sin entrar en mayores complicaciones como el modelado del sistema o su simulación por ordenador, lo que se deja a discreción del lector, aquí se tratará inicialmente de mantener el sistema en equilibrio sobre dos ruedas. Utilizando el sensor de luz y el código propuesto por Phillipe Hurbain, en [12],

se alcanzan los mejores resultados con unos valores para el regulador PID de: término proporcional = 40, término integral = 35 y término derivativo = 20. Aunque el equilibrio no es perfecto, sí que está a las puertas de serlo.

A partir de ahí, comienza el desarrollo de un montaje y una programación que sí garanticen el mejor resultado posible, utilizando bien otros sensores, bien otros lenguajes, o ambos.

7.1.1. Problemas encontrados

Desde las primeras pruebas que fueron llevadas a cabo, el código propuesto por Phillipe Hurbain distaba de proporcionar los resultados esperados (pese a que, al repetir el experimento varias veces y encontrar el punto de equilibrio justo, el NXTWay acababa manteniéndose estable por periodos relativamente grandes de tiempo), ya fuera por las limitaciones del propio trabajo de programación, o por el montaje elegido.

Además, en el enfoque planteado aquí, como ya se ha comentado, el lazo de control se ejecuta cada 100 milisegundos, un periodo muy grande si lo comparamos con la capacidad de procesamiento del NXT. Pero es necesario dejar este tiempo de ejecución porque, de no estar presente, la señal de activación sería enviada a los motores, pero el programa saltaría a la instrucción siguiente y éstos no tendrían tiempo material para girar. Este requerimiento del proceso puede suponer una limitación muy grande, al verse la frecuencia de actuación reducida a 10 Hz, pese a que los cálculos sólo suponen una fracción del tiempo de lazo.

Por lo tanto, la primera aproximación es mejorar, dentro de lo posible, la programación, manteniendo el mismo montaje.

7.1.2. Revisión del código original

Para operar con más soltura, se ha reescrito el código en NXC (recordemos que el original propuesto por Phillipe estaba en NBC, de más bajo nivel), obteniéndose como resultado el incluido en el **Apéndice C**. El funcionamiento es básicamente el mismo: al iniciar el programa, se toma la primera lectura del sensor y se almacena como consigna (por lo que es importante que, antes

de ejecutar el código, se coloque el NXT en posición totalmente vertical). A partir de ahí, el bucle de control actúa cada 100 milisegundos: toma una lectura del sensor, compara con la consigna para obtener la señal de error, la pasa por el regulador PID y la envía a los motores para su funcionamiento.

Este código se puede encontrar en el **Apéndice C**.

Observaciones

La medida de distancia se toma utilizando la función `SensorNormalized()`, más precisa que la estándar `Sensor()`, pero que devuelve unos valores mucho mayores que ésta, al no estar reescalados. En consecuencia, para que al aplicar el algoritmo PID las potencias de salida no sean astronómicamente grandes, e igualmente para aumentar la precisión en el cálculo dada la ausencia de decimales en operaciones con variables, se incluye un factor de escala en el programa. La medida normalizada se compara con la consigna, se pasa por el regulador PID, y el resultado se divide por este factor, lo que devuelve un número que normalmente está entre -150 y 150. Este número es, en realidad, la potencia a la que deben funcionar los motores, y que se les pasará como parámetro.

Como la potencia a la que funcionan estos motores ha de ser especificada en términos de porcentaje, y como los valores demasiado pequeños son incapaces de mover el balancín, antes de girar los motores la potencia es saturada entre 20 y 100, en valor absoluto. El sentido de giro se determinará después, según el error actual sea positivo o negativo.

Los resultados obtenidos con esta configuración, incluso afinando al máximo la sintonía del regulador, son evidentemente similares a los conseguidos usando el programa original de Phillipe Hurbain; el sensor de luz no es muy preciso y la estabilidad alcanzada dista mucho de ser perfecta. Para conseguir un mejor control, habría que recurrir a métodos analíticos basados en modelo, como ya se ha hecho (ver [14]), pero eso queda fuera del alcance de este proyecto. Utilizando ese tipo de herramientas, el control obtenido es casi perfecto, siendo capaz el sistema de aguantar y rechazar perturbaciones, avanzar, girar sobre sí mismo, y subir rampas sin ningún problema. Sin llegar a esos extremos, se hace inevitable alterar el diseño del balancín, en este caso un giróscopo por sus, en principio, interesantes posibilidades.

7.1.3. Implementación con giróscopo

Al utilizar el giróscopo de HiTechnic, ya que es del que se dispone, el trabajo se puede plantear de dos maneras: como la salida de dicho sensor es su velocidad de giro, y por tanto, la del cuerpo a que está unido (el NXTWay), la consigna a introducir puede ser simplemente velocidad de giro cero (estabilidad absoluta), y tomar el error como cualquier diferencia a este valor.

Otra posibilidad es integrar, en tiempo discreto, la velocidad, para hallar el ángulo girado, partiendo de las lecturas devueltas por el giróscopo y sumándolas. En este caso, habrá que utilizar un lazo de integración paralelo al de la acción de control, de frecuencia mucho mayor que la de éste, para aprovechar la gran precisión del sensor.

Con la primera aproximación los resultados obtenidos son similares a los del sensor de luz. Los códigos utilizados están incluidos en el **Apéndice C**.

7.2. El Autoparker

Uno de los ejemplos más vistosos de las capacidades del NXT es el **Autoparker**: un coche autónomo de LEGO, capaz de avanzar entre una línea de *coches* buscando un hueco en el que quepa y, una vez encontrado, aparcar en él. Se utilizan sensores de distancia en cada fase del proceso para asegurar que nuestro coche aparque con la mayor soltura posible y sin chocar. Una foto del vehículo en cuestión se puede observar a continuación.

El desarrollo de esta aplicación no ha estado exento de problemas, desde el propio montaje del coche hasta la programación.

7.2.1. Problemas encontrados

Inicialmente se consideró llevar a cabo esta aplicación utilizando una modificación del montaje Tribot, propuesto por LEGO, consistente en un carrito con dos ruedas motrices, cada una con su motor, y una rueda loca en la parte trasera. Posteriormente, al construir dicho robot, se comprobó que la rueda

Imagen 7.1: Imagen del vehículo Autoparker

loca, pese a estar calibrada su posición y descentrada su masa, se comportaba perfectamente cuando iba con las ruedas motrices por delante, pero se desviaba enormemente si era la que tenía que abrir la marcha. Circular con la rueda loca al frente provocaba que al encontrarse con pequeñas irregularidades del suelo, o al darse cualquier fallo de sincronía entre los dos motores, por pequeño que fuera, el carrito inevitablemente se desviara.

Tras considerar diversas soluciones, todas ellas fallidas, como poner una rueda fija en lugar de una loca, se sustituyó desde cero el montaje, cambiándolo por un coche completo, de cuatro ruedas, con su sistema de dirección. Un motor iría dedicado a dicha dirección y otro a la cabeza sensora, por lo que restaba únicamente un motor para dirigir las ruedas, lo que desembocó inevitablemente en el mismo problema en que se encontraron los pioneros de la automoción moderna: al girar, las ruedas recorren diferentes distancias, pero el giro transmitido a las dos es el mismo, lo que causa enormes problemas. Se hizo necesario incorporar un mecanismo diferencial, para asegurar que los giros se efectuaran de la mejor manera posible. Afortunadamente, un diseño sencillo, funcional, y, principalmente, construido únicamente con piezas de la caja del NXT se halla disponible en internet (ver [16]).

7.2.2. Programación del Autoparker

El primer programa desarrollado para este montaje es simplemente un aparcamiento en batería: el robot avanza hasta que detecta que el espacio es suficiente, frena dejando un margen para poder pivotar, gira, y recula hasta entrar en el hueco, utilizando el sensor de ultrasonidos para las tareas de medición y el de luz para no chocar con la hipotética pared del garaje.

El mayor problema presentado es el de realimentar la orientación externa del coche, para saber en qué momento se halla perfectamente perpendicular a su trayectoria original y puede retroceder sin problemas. Una solución puede ser utilizar el tacómetro interno de los motores para contar las vueltas al girar, y utilizando la siguiente relación grados de giro - longitud, estimar la distancia recorrida por la rueda:

$$Grados\ de\ giro = \frac{distancia\ en\ mm * 360}{\pi * diametro}$$

Con diámetro = 56 mm (rueda incluida en la caja del NXT)

En principio parece una buena solución, pero la presencia del diferencial añade un componente a la ecuación que implicaría dificultosos cálculos y, probablemente, de imprecisos resultados. Además, hay que tener en cuenta que las ruedas pueden resbalar una pequeña cantidad, dependiendo del tipo de suelo, y que, por la propia configuración del coche, es inevitable cometer algún error. Por lo tanto, se optó por la solución de usar la brújula de HiTechnic para conocer en todo momento la orientación del vehículo, y así asegurar que pueda entrar al hueco sin mayores problemas.

La brújula, por su parte, también presenta un detalle a tener en cuenta: los valores devueltos por ella, como se probó en el capítulo 6, se alejan de la realidad si el encapsulado se encuentra próximo a motores en funcionamiento o elementos electrónicos. Así, para evitar esto, se puede situar la brújula separada del coche (en un mástil de al menos 15 cm de altura), o bien compensar este desvío en la medida experimentalmente.

El código utilizado se encuentra en el **Apéndice C**.

Capítulo 8

Discusión final y conclusiones

8.1. Conclusiones

El objetivo fundamental de este proyecto era recabar toda la información posible y estudiar y evaluar las capacidades del LEGO Mindstorms NXT para su uso docente en ingeniería de control. Tras concluir esta tarea, se ha encontrado que:

- Un set completo de NXT, con su Ladrillo inteligente, sensores, motores, y piezas de LEGO Technic, es enormemente barato en comparación con el equipamiento habitual de un laboratorio de control, y mucho más versátil.

- Existe una gran comunidad de LEGO Mindstorms NXT por todo el mundo, organizada en foros y grupos de usuarios, que no vacilan en ayudar y aconsejar ante cualquier duda que se les plantee. Ingenieros de software, expertos en robótica y automática, profesores de universidad, y un largo etcétera de voces con grandes conocimientos colaboran día a día en la escena NXT, tanto en los mentados foros y grupos como en sus propias páginas y blogs personales.

- LEGO, consciente del interés de los aficionados a la robótica por profundizar lo máximo posible en sus dispositivos, ha liberado todas las especificaciones, tanto de hardware como de software, del NXT, y lo

ha preparado para el *homebrew*[1] y la experimentación. Esto ha contribuido sin duda a su rápida popularización frente a otros modelos propietarios.

- Consecuencia de lo anterior es el mercado de piezas y nuevos sensores por terceras empresas, la gran diversidad en cuanto a lenguajes y entornos de programación, y la facilidad de acceso a ellos, siendo los más utilizados también libres.
- El NXT ya ha sido usado en varias universidades como parte de la enseñanza práctica (El Instituto RWTH Aachen, el MIT, la UBC, etc.), principalmente en materias de robótica. Sin embargo, excepto el NXTWay-GS de Ryo Watanabe (ver [14]), no ha sido usado seriamente en ingeniería de control.
- La programación del NXT no requiere aprender nuevos lenguajes ni procedimientos exclusivos, sino que puede realizarse con pequeñas variantes de los lenguajes de programación más conocidos y estudiados, como C/C++, Java, etc., y a través de entornos intuitivos y gráficos, sin tener que preocuparse por nada excepto por el propio código.

Nos encontramos ante una herramienta completa, madura y enormemente versátil, de una constitución física sólida y resistente (avalada por la gran calidad característica de LEGO), y capaz de comunicarse inalámbricamente con un ordenador, registrar datos, controlar sistemas, y del que se sabe absolutamente todo porque el propio fabricante se ha encargado de ello. La impresión no podría ser mejor.

8.2. Aspectos negativos

A pesar de lo anterior, no todas las conclusiones son tan positivas; algunas puntualizaciones han de ser consideradas antes de plantearse el trabajar con un NXT, en cualquier campo y ámbito.

- En determinadas situaciones, la precisión de los sensores y motores, así como la potencia de éstos, puede no llegar a ser suficiente. Es im-

[1] Término con el que se designa todo software o firmware casero, no oficial, y obra normalmente de aficionados con alto nivel de conocimientos.

prescindible comprobar, en base tanto a pruebas y estudios mostrados aquí como realizados por terceros, la idoneidad del LEGO Mindstorms NXT para cada aplicación concreta.

- La propia naturaleza del NXT, construido con piezas de LEGO, puede provocar que se pierda más tiempo del previsto en el diseño del robot a utilizar, si no se aprovecha algún montaje ya existente, como ha ocurrido en el desarrollo de este proyecto.

8.3. Trabajos futuros

LEGO Mindstorms NXT puede ser usado en docencia de ingeniería de control a un nivel mucho más alto que el presentado aquí, en aplicaciones apoyadas por un ordenador, en sistemas más complejos de órdenes superiores, etc., lo que podría cubrir todo el espectro universitario actual de la enseñanza en estas materias, desde los primeros cursos hasta los más avanzados. A continuación se detallan algunas ideas que han surgido a lo largo del desarrollo de este proyecto, pero han debido mantenerse al margen para no desviarlo de su propósito original.

- Las dos aplicaciones incluidas aquí en el capítulo 7 están planteadas y brevemente resueltas, pero un estudio más profundo se revela muy interesante: El NXTWay puede ser diseñado para, además de mantenerse en equilibrio, avanzar, pivotar o esquivar obstáculos; el Autoparker puede ser modificado para su aparcamiento en línea en lugar de batería, etcétera.

  En este proyecto, dichas aplicaciones aparecen como ejemplos prácticos, pero no están, ni mucho menos, todo lo desarrolladas que podrían; una profundización en la materia puede permitir utilizar el NXT para casos más completos y avanzados.

- La capacidad de comunicar los NXT tanto con un ordenador como entre ellos vía BlueTooth no ha sido trabajada aquí, por cuanto es algo merecedor de un estudio propio independiente. Un enorme abanico de posibilidades se extiende al considerar la comunicación inalámbrica entre unidades (cada NXT puede ser conectado simultáneamente a otros tres, a una distancia de hasta 10 metros), en campos como el control distribuido.

- La flexibilidad y comodidad de trabajo de las piezas de LEGO Technic, junto a la versatilidad y capacidad de trabajo conjunto de los Ladrillos NXT abre un espectro de grandes montajes y configuraciones que pueden desembocar en proyectos mucho más ambiciosos que los presentados aquí: un brazo robótico de varios grados de libertad, un robot autónomo totalmente capaz,... La lista de posibilidades es muy larga.

Apéndice A

Código original y modificaciones al APRIL

A continuación se incluye el código original, escrito en NXC, para el robot APRIL (A PID Robot Implemented with LEGO), proyecto de Kevin MCL, de la **University of British Columbia**. Ha sido descargado de la web original del proyecto,
`http://www.physics.ubc.ca/~kevinmcl/projects/lego/APRIL/`

El programa implementa una aproximación digital de un regulador PID estándar, siempre teniendo en cuenta las características y limitaciones del hardware del NXT, con la que se consiguen unos resultados bastante buenos, aunque mejorables. El objetivo básico del proyecto es el de seguimiento de consigna: El NXT acepta como entrada los valores de distancia que le llegan desde el sensor, e intenta, mediante unas ruedas motorizadas, mantener esa distancia lo más próxima posible a la consigna dada al inicio de la ejecución.

El código se reproduce a continuación con el consentimiento expreso del autor, incluido además en los propios comentarios del archivo fuente. También se halla disponible en el archivo `april.nxc`

```
/*
 *   Author: Kevin McLeod
 *   Affiliation: UBC Department of
 *   Physics and Astronomy (Physics 210)
 *   Date: December 5th 2007
 *
 *   This code is uses a PID control algorithm
 *   to maintain a relative distance
 *   from an object
 *   It was written for a simple 4 wheeled robot
 *   using the standard Ultrasonic Sensor
 *
 *   Please see
 *   http://www.physics.ubc.ca/~kevinmcl/projects/lego/APRIL
 *   for complete project details
 *
 *   And the BricxCC IDE, http://bricxcc.sourceforge.net
 *
 *   This code is, of course, free to be used and/or
 *   modified by anyone for any reason they see fit
 *   If you do use this code in your own project please
 *   contact me (see the above website) because I
 *   would love to see what you're working on!
 *   Also feel free to contact me with any questions
 *   or comments.
 */

/*
Corrects an output power value so that it is between
20% and 100%
The minimum value of 20% is used because power levels
below 20% are incapable of moving the device
*/
int normPwr(int power)
{
   int min = 20;

   if(abs(power)>100)
      return 100;
   else if(abs(power)<min)
      return min;
   else
      return abs(power);
}

/*
Turns on motors A and B
The given error determines whether motors should be in
forward or reverse direction
The direction of the motors will depend on the
orientation and gearing of the device
*/
int move(int currentError, int pwr)
{
   if(currentError<0)
   //Runs motors A and B in Reverse and synchronizes them
   OnRevReg(OUT_BC, normPwr(pwr), OUT_REGMODE_SYNC);
   else
   //Runs the motors as above, in reverse
   OnFwdReg(OUT_BC, normPwr(pwr), OUT_REGMODE_SYNC);
}
```

```
/*
Main method of the program
*/
task main()
{
   //Initializes the US sensor
   SetSensorLowspeed(IN_1);

   //Tolerance of the Distance Sensor
   int tolerance = 10;

   //Proportional gain constant
   int P = 25;

   //Integral gain constant
   int I = 3;

   //Derivative gain constant
   int D = 25;

   //The scaling factor of the gain constants
   int scale = 10;

   //The Period of the control loop (in ms)
   int loopTime = 150;

   //Set Point (the relative distance
   //we are trying to maintain)
   int setPoint = 0;

   //Displays starting message
   TextOut(0, LCD_LINE1, "Press button to", true);
   TextOut(0, LCD_LINE2, "begin ...", false);
   //Plays a notification
   PlayToneEx(1000, 100, 4, false);

   //Waits for button to be pressed
   //(the main orange button)
   until(ButtonPressed(BTN4,true));

   //Reads in the starting distance
   setPoint = SensorUS(IN_1);

   //Displays the "Running" message and plays a tone
   TextOut(0, LCD_LINE1, "Running...", true);
   //Plays a notification
   PlayToneEx(2000, 300, 4, false);

   //The following variables are used in the control loop

   //The current error in position
   int error=0;

   //The error from the previous loop run
   int prevError=0;

   //A sum of errors used in the integral term
   int sumError=0;

   //The power outputs for the Proportional,
   //Derivative and Integral terms respectively
   int pOut=0;
```

```
    int iOut=0;
    int dOut=0;

    //The sum of the 3 individual outputs
    int totalOut=0;

    //The main loop that handles the position correction
    while(true)
    {
       //Stores the previous error
       prevError = error;

       //Reads the current distance (see Mindensors Code)
       int distance = SensorUS(IN_1);

       //Calculates the current error
       error = setPoint-distance;

       //Generates the proportional output
       pOut = error*P;

       //Adds the current error to the sum
       sumError += error;

       //Calculates the integral output
       iOut = sumError*I;

       //Calculates the derivative output
       dOut = (error-prevError)*D;

       //Adds the 3 terms together and divides by the
       //scale term (to account for the lack
       //of floating numbers)
       //This number may be above 100%, and/or negative
       totalOut = (pOut+iOut+dOut)/scale;

       //Checks if the current position
       //is outside the set tolerance
       if(abs(error)>tolerance)
          //Turns on the motors
          move(error, totalOut);

       else
          //Resets the sum
          sumError=0;

       //Allows time for movement
       Wait(loopTime);

       //Cuts power to the motors, but does
       //not apply a stopping force
       Float(OUT_BC);
  }
}
```

A.1. Comentarios al código y descripción

Kevin ha efectuado un gran trabajo de documentación en los comentarios del código, que de por sí ocupan casi tanto espacio como las propias líneas de programación, pero aún así, a continuación se da la descripción de funcionamiento del programa.

- La primera subfunción, `int normPwr(int power)`, devuelve un entero que pasará como argumento de potencia a la rotación de los motores. Su función es simplemente establecer una saturación a la potencia de los motores: valores inferiores al 20 % de potencia son elevados a esa cantidad (ya que con menos potencia resulta casi imposible desplazar al robot), y valores mayores al 100 % son recortados a ese tope (ya que de otra manera podrían producirse problemas, y el código no llegaría a compilar).

- A continuación, la función `int move(int currentError, int pwr)` determina si el error actual es por proximidad o por lejanía, para así desplazar al robot hacia adelante o hacia atrás, tomando como argumentos el error actual (currentError) y la potencia saturada de la función anterior (pwr).

- La tarea principal (main) del código comienza con una declaración y establecimiento de todas las variables que se usarán en el programa, desde la tolerancia del sensor hasta el periodo de ejecución, pasando por los valores de los parámetros P, I y D del regulador PID. Posteriormente se detiene la ejecución del programa hasta que se pulse el botón central, momento en el cual la distancia medida se toma como consigna y el código entra en el bucle de control.

- El bucle de control propiamente dicho, de ejecución continuada hasta que se le detenga (por el `while(true)` que lo precede), se comporta según un regulador PID real:

  - Primero, almacena el error actual como previo.
  - Después, lee la distancia y calcula si existe error con respecto a la consigna.
  - A continuación, calcula los términos proporcional, integral, y derivativo: el proporcional simplemente multiplicando el error por el parámetro P, el integral añadiendo el error actual al error previo y multiplicando el

resultado por el parámetro I, y el derivativo multiplicando el parámetro D por la diferencia entre el error actual y el previo. Estas aproximaciones, si bien no totalmente precisas, son suficientes en principio.

- La salida a los motores se calcula sumando los tres términos y dividiendo por un factor de escala, para compensar la carencia de números decimales al operar con variables. Si el valor resultante es superior a la tolerancia, se llama a la función de movimiento anteriormente descrita. Si no, se resetea el error.
- El bucle finaliza con una espera para los motores, definida al principio. Si no se espera aunque sea un mínimo, los motores no tienen virtualmente tiempo para circular.

A.2. Modificaciones posteriores

El mismo Kevin reconoce en su web que el controlador PID está *lejos de perfectamente afinado*, así que se ha introducido una modificación en el código que permite variar los valores de los parámetros del regulador antes de la ejecución del programa, y no estar directamente incluidos en el código, lo que hace necesario tener que recompilar cada vez que se quieran variar.

Para esta ampliación se hace necesario utilizar el firmware modificado para NXC, disponible en la web del proyecto, ya que en ella se hace uso del botón de detención. El firmware original de LEGO detiene la ejecución de cualquier programa al pulsar dicho botón, pero con el modificado es posible incluir una instrucción (`SetLongAbort(true)`), que no detiene el programa a no ser que el botón se mantenga pulsado un breve periodo de tiempo. Esta inclusión resulta, como se verá, muy útil en muchos programas, al disponer de cuatro botones en lugar de tres.

El código modificado es el siguiente, con las nuevas líneas de código marcadas en negritas (sólo se ha incluido la tarea principal). También se incluye en el archivo `pidmodificado.nxc`

```
/*
Main method of the program
*/

task main()
{

   //Initializes the US sensor
   SetSensorLowspeed(IN_1);

   //Permite acceder al cuarto botón
   SetLongAbort(true);

   //Tolerance of the Distance Sensor
   int tolerance = 10;

   //Variables para los parámetros P, I y D
   int P, I, D;

   //The scaling factor of the gain constants
   int scale = 10;

   //The Period of the control loop (in ms)
   int loopTime = 150;

   //Variable for enter values
   short ok=0;

   //Set Point (the relative distance we are trying to maintain)
   int setPoint = 0;

   //Displays starting message
   //TextOut(0, LCD_LINE1, "Pulsa el boton", true);
   //TextOut(0, LCD_LINE2, "para empezar ...", false);
   //Plays a notification
   PlayToneEx(1000, 100, 4, false);

   //Primero, reseteamos contadores y limpiamos la pantalla
   ButtonCount(BTNEXIT,true);
   ButtonCount(BTNCENTER,true);
   ButtonCount(BTNLEFT,true);
   ButtonCount(BTNRIGHT,true);
   ResetScreen();

   //Espera a que el botón sea pulsado
        while(ok==0){

   //Saca por pantalla los valores
           TextOut(0,LCD_LINE1,"Kp:");
           NumOut(0,LCD_LINE2, P);
           TextOut(0,LCD_LINE3,"Ki:");
           NumOut(0,LCD_LINE4, I);
           TextOut(0,LCD_LINE5,"Kd:");
           NumOut(0,LCD_LINE6, D);

   //Cada pulsación sobre los diferentes botones aumenta el valor de los
parámetros P, I, y D. El botón de la izquierda aumenta P
           P=ButtonCount(BTNLEFT,false);

   //El botón de salida aumenta I (si se mantiene pulsado cancela la ejecución
del programa
           I=ButtonCount(BTNEXIT,false);
```

```
  //Y el botón de la derecha aumenta D
          D=ButtonCount(BTNRIGHT,false);

  //Los valores se terminan de establecer al pulsar el botón central.
          ok=ButtonCount(BTNCENTER,false);
          }

while(ok==1)
{

  //Reads in the starting distance
  setPoint = SensorUS(IN_1);

  //Displays the "Running" message and plays a tone
  TextOut(0, LCD_LINE1, "Ejecutando...", true);

  //Plays a notification
  PlayToneEx(2000, 300, 4, false);

  //The following variables are used in the subsequent control loop
  //The current error in position
  int error=0;

  //The error from the previous loop run
  int prevError=0;

  //A sum of errors used in the integral term
  int sumError=0;

  //The power outputs for the Proportional, Derivative and Integral terms
respectively
  int pOut=0;
  int iOut=0;
  int dOut=0;

  //The sum of the 3 individual outputs
  int totalOut=0;

  //The main loop that handles the position correction
  while(true)
  {

     //Stores the previous error
     prevError = error;

     //Reads the current distance (see Mindensors Code)
     int distance = SensorUS(IN_1);

     //Calculates the current error
     error = setPoint-distance;

     //Generates the proportional output
     pOut = error*P;

     //Adds the current error to the sum
     sumError += error;

     //Calculates the integral output
     iOut = sumError*I;

     //Calculates the derivative output
```

```
      dOut = (error-prevError)*D;

      //Adds the 3 terms together and divides by the scale term (to account for
the lack of floating numbers)
      //This number may be above 100%, and/or negative
      totalOut = (pOut+iOut+dOut)/scale;

      //Checks if the current position is outside the set tolerance
      if(abs(error)>tolerance)

         //Turns on the motors
         move(error, totalOut);

      else

         //Resets the sum
         sumError=0;

      //Allows time for movement
      Wait(loopTime);

      //Cuts power to the motors, but does not apply a stopping force
      Float(OUT_BC);

  } //End of main while loop

} //Fin del while ok==1

} //End of main
```

Apéndice B

Más sobre el NXTWay

Según la popular enciclopedia online Wikipedia (ver [17]), el Segway es un *vehículo de transporte ligero giroscópico eléctrico de dos ruedas, con autobalanceo, controlado por ordenador*, muy popular entre los aficionados a la electrónica y robótica. El NXTWay es una versión particular de este invento que, si bien evidentemente no es capaz de transportar a nadie, sí consigue mantenerse estable sobre dos ruedas.

B.1. Historia y versiones del NXTWay

La primera aproximación al NXTWay fue el LegWay, construido en RCX por Steve Hassenplug, muy conocido en la escena LEGO Mindstorms. El LegWay, pese a sus limitaciones, conseguía mantenerse estable, seguir líneas (con un sensor especial que ya no se fabrica) e incluso girar. Este montaje ha llegado a ser enormemente popular entre los aficionados a Mindstorms, y la web de su creador registra decenas de miles de visitas.

Con la llegada al mercado del Mindstorms NXT, era de esperar que, tarde o temprano alguien actualizara el LegWay en el nuevo set. Philippe E. Hurbain, otro *estudioso* de LEGO Mindstorms, fue el primero en hacerlo, creando este NXTWay. Mucho más simple (sólo utiliza un sensor óptico para mantenerse estable), y limitado (no gira ni se desplaza, simplemente se mantiene sobre dos ruedas), está programado en NBC, el lenguaje precursor

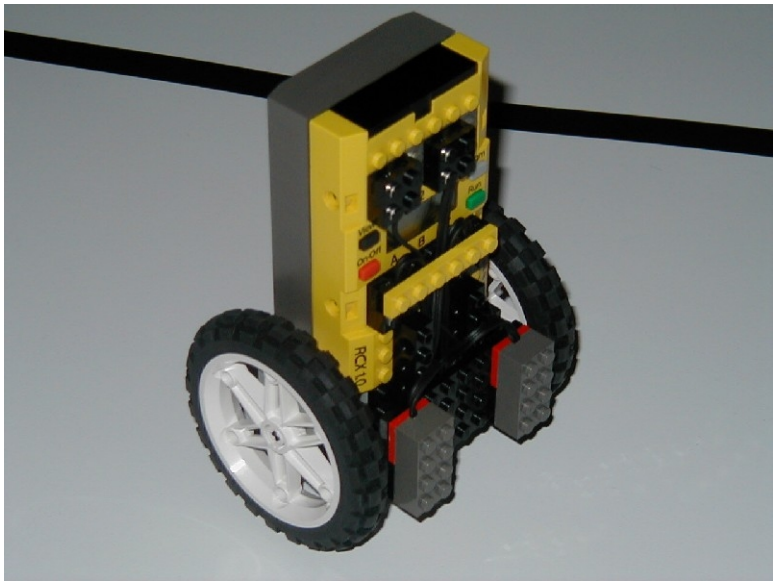


Imagen B.1: El LegWay original de Steve

de NXC, y el código original incluye un lazo de control tradicional con un regulador PID (al estilo del del APRIL).

A partir del NXTWay, y con la popularización del Mindstorms NXT, ha surgido un buen número de proyectos inspirados en este concepto de balancín sobre dos ruedas, aunque sin duda, el más digno de mención, tanto por su enfoque como por sus resultados, es el NXTWay-GS de Ryo Watanabe (Ver [14]).

El desarrollo del NXTWay-GS es en sí mismo todo un problema de control: desde un modelado con ecuaciones diferenciales del montaje, linealizado y pasado a espacio de estados, a una simulación por ordenador en MATLAB, y finalmente la aplicación al montaje real. Ryo ha usado para la simulación el **Embedded Coder Robot NXT**, un completísimo entorno de desarrollo basado en Simulink que permite simular montajes enteros del NXT (con representación tridimensional), y posibilita el control basado en modelo. Una demo gratuita del producto se puede descargar de la web de LEJOS OSEK, `http://lejos-osek.sourceforge.net`.

Dada la complejidad y el uso tanto de otros lenguajes de programación como de herramientas informáticas ajenas a las presentadas en este proyecto, no se ha profundizado más en este, por otra parte, denso proyecto, quedando

esta posibilidad a discreción del lector.

B.2. Código original del NXTWay

En la siguiente página se reproduce el código del NXTWay anteriormente citado, descargado de la web del propio Philippe E. Hurbain.También se halla disponible en el archivo `nxtwayoriginal.nbc`

```
//-----------------------------------------------
// NXTway - Philo - www.philohome.com - 6/5/2006
//-----------------------------------------------

dseg segment

// Sensor values
NVal word
offset word
err sdword
errold sdword
errdiff sdword
errint sdword

// Motor values
theUF byte
thePower sbyte
theOM byte OUT_MODE_MOTORON+OUT_MODE_BRAKE
theRM byte OUT_REGMODE_IDLE
theRS byte OUT_RUNSTATE_RUNNING
thePorts byte[] OUT_B, OUT_C  // motors B and C

// pid coeffs
kp sdword 40
kd sdword 10
ki sdword 35
scale sdword 45

// pid value
pid sdword

//temp var
temp  sdword

// timer vars
thenTick dword
nowTick dword

dseg ends

thread main

        setin   IN_TYPE_LIGHT_ACTIVE, IN_3, Type

  // initialize motors
  set thePower, 0
        set theUF, UF_UPDATE_SPEED+UF_UPDATE_MODE
        setout thePorts, OutputMode, theOM, RegMode, theRM, RunState, theRS,
UpdateFlags, theUF, Power, thePower
        set theUF, UF_UPDATE_SPEED

  // wait a bit to let sensor stabilize
        gettick nowTick
        add     thenTick, nowTick, 100 // wait 100 ms

Waiting:
        gettick nowTick
        brcmp LT, Waiting, nowTick, thenTick
```

```
  // reads center value. NXTway must be balanced.
  getin NVal, IN_3, NormalizedValue // More precise than PercentValue
  mov offset, NVal

Forever:
        getin   NVal, IN_3, NormalizedValue // read sensor values

  sub err, NVal, offset // Substract center value
  brtst GT, ErrPos, err // Linearize value
  mul err, err, 16      // (less variation if far from surface)
  div err, err, 10

ErrPos:
  sub errdiff, err, errold  // Compute differential error
  mov errold, err

  add errint, errint, err   // Compute integral error
  mul errint, errint, 2     // (with fast damping)
  div errint, errint, 3

  mul pid, kd, errdiff      // Differential component
  mul temp, kp, err         // Proportionnnal component

  add pid, pid, temp        // Add Diff+Prop
  mul temp, ki, errint      // Integral component
  add pid, pid, temp        // Add int
  div pid, pid, scale       // Scale PID value

  // saturate over 100 and under -100
  brcmp LT, under100, pid, 100
  mov pid, 100

under100:
  brcmp GT, overMin100, pid, -100
  mov pid, -100

overMin100:
  mov thePower, pid          // Update motor power
        setout thePorts, UpdateFlags, theUF, Power, thePower

  jmp Forever

  exit

endt
```

Apéndice C

Códigos utilizados en el proyecto

A continuación se detallan los códigos de programación usados en este proyecto, tanto en los ensayos de laboratorio como en las aplicaciones desarrolladas. Los archivos fuente también van incluidos en el cd, listos para ser compilados y ejecutados

C.1. Prueba de medida con el sensor de ultrasonidos

Correspondiente al archivo `pruebaultrasonidos.nxc`. El funcionamiento del código es bien sencillo: toma una lectura del sensor de ultrasonidos cada cierto periodo de tiempo (por defecto 3 segundos), emitiendo un sonido para avisar, y almacena el valor de dicha medida en un archivo de texto (*Prueba.txt*). El sensor debe conectarse en el puerto 1 para utilizar este código tal y como aquí se ofrece; si bien puede modificarse para utilizar cualquier otro.

La escritura de texto necesita de tres elementos: una variable tipo byte (en este caso `archivo`), que es el llamado *manejador* del archivo; una variable tipo short (en este caso `bytesWritten`), que devuelve la cuenta de los bytes escritos, aunque normalmente no se use; y una variable tipo string (en este caso `tmp`), que se usará para almacenar el valor numérico como texto antes

de escribirlo en el archivo. Es necesario disponer de suficiente espacio en la memoria para el archivo; de lo contrario no se creará.

```
/*
 *   Autor: Guillermo Nieves Molina
 *   Proyecto fin de carrera: Estudio de las posibilidades didacticas en
 *   ingenieria de control del LEGO Mindstorms NXT
 *   Año 2008
 *
 *   Este programa toma cada 3 segundos una lectura del sensor de ultrasonidos
 *   y la almacena en un archivo de texto editable.
 *
 *   Esta escrito en NXC, y necesita tener instalado en el NXT, al menos, el
 *   firmware 1.03.
 *
 *   Este codigo se distribuye libremente bajo licencia GNU GPL; una copia
 *   completa de dicha licencia se puede encontrar en
 *   http://www.gnu.org/licenses/gpl.txt
 */


  task main()
  {

  //Tiempo de espera entre medidas, en milisegundos
  int tiempobucle=3000;

  //Numero de lecturas que se tomaran
  int numerolecturas=50;

  //Variable que almacena el valor de la lectura
  int lectura;

  //Contador
  int i;

  //Variables necesarias para manejar los archivos de texto
  byte archivo;
  short bytesWritten;
  string tmp;

  /*Creamos un archivo con un tamaño adecuado al numero
  de lecturas que se vayan a tomar*/
  DeleteFile("Prueba.txt");
  CreateFile("Prueba.txt", 8000, archivo);
  WriteLnString(archivo,"Prueba de medida con el sensor de ultrasonidos" ,
bytesWritten);

  //Iniciamos el sensor en el puerto 1
  SetSensorLowspeed (IN_1);

  //Comienza el bucle de medida
  for (i=0;i<=numerolecturas;i++)
  {
       //Mide cada 3 segundos
       Wait(tiempobucle);

       //Reproduce un sonido para avisar
       PlayToneEx(1000, 100, 4, false);

       //Lectura de la distancia
       lectura = SensorUS(IN_1);

       //Pasa a caracter y escribe
       tmp = NumToStr(lectura);
```

C.2. Prueba de distorsiones en la medida de la brújula HiTechnic

Correspondiente al archivo `medidabrujula.nxc`. Este código toma una lectura de la brújula cada vez que se pulsa el botón central (naranja) del ladrillo, y almacena el valor de la medida en un archivo de texto (*Prueba.txt*). La brújula debe conectarse en el puerto 1 para utilizar este código, y es necesario disponer, al menos, del firmware 1.03.

El mecanismo de escritura a archivo es exactamente el mismo que en el caso anterior: pasar el valor numérico a texto y escribirlo en el archivo. La función `WriteLnString` escribe dicho valor e inserta un salto de línea al final, de manera que el archivo resultante, como en el caso anterior, es una lista de números precedida por un título.

Es conveniente remarcar que el periodo de espera de 500 milisegundos es necesario porque, si no se incluye, se pueden producir múltiples lecturas iguales al estar el botón pulsado aunque sea una mínima cantidad de tiempo (recordemos que el procesador tiene un reloj de 48 MHz, con lo que en el breve lapso de pulsar y soltar un botón el programa puede ejecutarse muchas veces).

```
/*
 *   Autor: Guillermo Nieves Molina
 *   Proyecto fin de carrera: Estudio de las posibilidades didacticas en
 *   ingenieria de control del LEGO Mindstorms NXT
 *   Año 2008
 *
 *   Este programa toma una lectura de la brujula HiTechnic cada vez que se
 *   pulse el boton central del NXT y la almacena en un archivo de texto.
 *
 *   Esta escrito en NXC, y necesita tener instalado en el NXT, al menos, el
 *   firmware 1.03.
 *
 *   Este codigo se distribuye libremente bajo licencia GNU GPL; una copia
 *   completa de dicha licencia se puede encontrar en
 *   http://www.gnu.org/licenses/gpl.txt
 */


  task main()
  {

  //Tiempo de espera entre medidas, en milisegundos
  int tiempoespera=500;

  //Variable que almacena el valor de la lectura
  int lectura;

  //Variables necesarias para manejar los archivos de texto
  byte archivo;
  short bytesWritten;
  string tmp;

  /*Creamos un archivo con un tamaño adecuado al numero de lecturas*/
  DeleteFile("Prueba.txt");
  CreateFile("Prueba.txt", 8000, archivo);
  WriteLnString(archivo, "Prueba de medida con la brujula HiTechnic",
bytesWritten);

  //Iniciamos el sensor en el puerto 1
  SetSensorLowspeed (IN_1);

  //Comienza el bucle de medida
  while(true)
  {
       //Espera a que el boton sea pulsado para leer
       until(ButtonPressed(BTNCENTER,true));

       //Reproduce un sonido para avisar
       PlayToneEx(1000, 100, 4, false);

       lectura = SensorHTCompass(IN_1);

       //Pasa a caracter y escribe
       tmp = NumToStr(lectura);
       WriteLnString(archivo,tmp, bytesWritten);

       //Espera para dar tiempo a soltar el boton
       Wait(tiempoespera);
  }
  CloseFile(archivo);

}
```

C.3. Prueba de respuesta de los motores en función de los parámetros del regulador

Correspondiente al archivo `pruebamotores.nxc`. Este código ordena al motor un giro de 270 grados, tanto en sentido horario como antihorario, y escribe a un archivo (rotacion.txt) los valores devueltos por el tacómetro interno del propio motor. Al descargar dicho archivo y graficarlo, se obtiene una representación visual de la respuesta del motor, en la que se puede apreciar el sobrepico, las oscilaciones, etc.

De nuevo es conveniente remarcar que en el eje X no se representa el tiempo, sino las muestras devueltas por el tacómetro.

El programa incluye tres variables globales, `p`, `i` y `d`, que serán después tomadas como argumento por la función de giro como los parámetros P, I y D del regulador interno del motor. Por defecto, estos valores son configurados por el NXT a 40, 20 y 100, pero la respuesta así conseguida puede no ser lo suficientemente buena. Así, los valores aquí recomendados, en torno a 45, 60 y 95, proporcionan una salida más rápida, con menos oscilación, y error estacionario casi nulo.

En este código se hace uso de la instrucción precedes, que llama a las dos funciones secundarias del código: `escribe()` y `mueve()`. Escribir así el programa permite ejecutar las dos simultáneamente, y que mientras el motor está girando, la rutina de escritura complete el archivo de texto. Ejecutar varias tareas simultáneamente no supone ningún problema siempre y cuando no haya conflictos entre ellas por acceso a hardware o instrucciones contradictorias.

```
/*
 *   Autor: Guillermo Nieves Molina
 *   Proyecto fin de carrera: Estudio de las posibilidades didacticas en
 *   ingenieria de control del LEGO Mindstorms NXT
 *   Año 2008
 *
 *   Este programa gira un motor un angulo determinado dando la posibilidad de
 *   variar los parametros P, I y D del regulador y almacena los valores del
 *   tacometro de dicho motor en un archivo.
 *
 *   Este codigo se distribuye libremente bajo licencia GNU GPL; una copia
 *   completa de dicha licencia se puede encontrar en
 *   http://www.gnu.org/licenses/gpl.txt
 */


//Constantes globales
#define potencia 50         //Potencia del motor en %
#define puertomotor OUT_B   //Puerto al que esta conectado el motor

#define p 45
#define i 60  //Valores de los parametros p, i y d (por defecto: 40, 20 y 100)
#define d 95

int angulo=-270;    //Angulo de giro en grados
short contador=0;

byte etiqueta;



/**********************/
task escribe()
{

string tmp;
short resultado;

//Escribe el valor del tacometro mientras los motores giran
  while(contador<2)
    {
    tmp=NumToStr(MotorRotationCount(puertomotor));
    WriteLnString(etiqueta, tmp, resultado);
    }

CloseFile(etiqueta);

}



/*************************/
task mueve()
{

  while(contador<2)
    {

    //Gira en los dos sentidos
    angulo=-angulo;
```

```
    RotateMotorPID(puertomotor,potencia,angulo,p,i,d);

    //Para utilizar los parametros por defecto del regulador PID, comentar la
    funcion anterior y utilizar la siguiente:
    //RotateMotor(puertomotor,potencia,angulo);

    //Espera un segundo para separar claramente las dos curvas
    Wait(1000);

    contador++;
    }

}

/*************************/

task main()
{

Precedes(escribe,mueve);

DeleteFile("rotacion.txt");
CreateFile("rotacion.txt",80000,etiqueta);

}
```

C.4. Revisión del código original del NXT-Way

Correspondiente al archivo `nxtway.nxc`. Este programa es simplemente la traducción a NXC del código escrito por Phillipe Hurbain en NBC para su NXTWay, obtenido de su web, [12].

El funcionamiento, como ya se ha comentado, implica la toma de una medida inicial, que será la consigna a mantener, y la ejecución continua de un lazo de control realimentado. La medida inicial, por tanto, debe tomarse con el balancín situado de la forma más equilibrada posible, porque ésa será la posición que intente alcanzar.

Los parámetros del regulador PID, así como el factor de escala, son distintos en este programa que en el original, por el distinto tratamiento que hacen los lenguajes a las lecturas de los sensores. Los resultados apreciables, en cambio, son bastante similares: el equilibrio alcanzado es casi perfecto pero continúa sin ser total.

```
/*
 *   Autor: Guillermo Nieves Molina
 *   Proyecto fin de carrera: Estudio de las posibilidades didacticas en
 *   ingenieria de control del LEGO Mindstorms NXT
 *   Año 2008
 *
 *   Este programa es una revision en NXC del escrito originalmente por
 *   Phillipe Hurbain para el control de su NXTWay.
 *
 *   El codigo incluye un lazo de control con un algoritmo PID; al iniciarlo,
 *   toma la primera lectura como consigna (punto de equilibrio).
 *
 *   Este codigo se distribuye libremente bajo licencia GNU GPL; una copia
 *   completa de dicha licencia se puede encontrar en
 *   http://www.gnu.org/licenses/gpl.txt
 */


  #define Puerto IN_2 //Puerto del sensor de luz
  #define Espera 80 //milisegundos

  task main(){

  //Variable que almacena la consigna
  int SetPoint;

  //Variable que almacena las lecturas
  int Lectura;

  //Variables para el algoritmo PID
  int P=90,I=15,D=20;
  int TerminoP,TerminoI,TerminoD;
  int ErrorIntegral,ErrorPrevio;

  //Factor de escala para compensar la ausencia de decimales al trabajar con
variables
  int FactorEscala=100;

  //Variables usadas en los calculos
  int Error, Potencia;

  //Variables usadas en la escritura
  byte Etiqueta;
  short Comprobante;
  string Tmp;

  DeleteFile("nxtway.txt");
  CreateFile("nxtway.txt", 50000, Etiqueta);

  //Iniciamos el sensor
  SetSensorLight(Puerto);

  //Estabilidad
  Wait(100);

  //Leemos la posicion inicial y la tomamos como set point
  SetPoint=SensorNormalized(Puerto);
```

```
  while(true)
  {
       //Almacena el error anterior (para el PID)
       ErrorPrevio=Error;

       //Leemos
       Lectura=SensorNormalized(Puerto);

       //Guardamos posicion antes de actuar

       //Diferencia de medidas
       Error=(Lectura-SetPoint);

       //Aplica el PID:

       //Termino proporcional
       TerminoP=P*Error;

       //Termino integral
       ErrorIntegral+= Error;

       TerminoI=I*(ErrorIntegral);

       //Termino derivativo (aproximacion)
       TerminoD=D*(Error-ErrorPrevio);

       //Suma de terminos
       Potencia=(TerminoP+TerminoI+TerminoD)/FactorEscala;

       Tmp = NumToStr(Potencia);
       WriteLnString(Etiqueta,Tmp, Comprobante);

       //Satura y pasa a positiva la potencia a entregar a los motores
       if(abs(Potencia)>100)
            Potencia=100;
       else if(abs(Potencia)<20)
            Potencia=20;
       else Potencia=abs(Potencia);

       //Gira en un sentido o en otro
       if(Error<0)
            OnRevReg(OUT_BC, Potencia, OUT_REGMODE_SYNC);
       else
            OnFwdReg(OUT_BC, Potencia, OUT_REGMODE_SYNC);

       //Resetea el termino de error integral
       ErrorIntegral=0;

       //Da tiempo a los motores para actuar
       Wait(Espera);

       //Corta la alimentacion a los motores
       Float(OUT_BC);

  }

  CloseFile(Etiqueta);

}
```

C.5. Implementación del NXTWay con giróscopo

Correspondiente al archivo `nxtwaygyro.nxc`. El código es una modificación del anterior, la revisión del original en NBC, pero utilizando en esta ocasión un giróscopo como sensor para la realimentación; el resto es prácticamente igual.

Al utilizar el giróscopo hay que tener en cuenta que, para que sus medidas sean veraces, hay que situarlo lo más próximo posible al eje de giro, lo más centrado posible, y bien orientado (recordemos que el giróscopo sólo capta sus lecturas en un eje).

El funcionamiento es ligeramente distinto al anterior programa; aquí la consigna no es una posición concreta, sino una velocidad angular nula (correspondiente a una ideal situación de equilibrio). Para ello, no se toma lectura alguna al inicio del programa, sino que directamente se iguala a cero y se trata cualquier medida devuelta por el giróscopo como error.

```
/*
 *   Autor: Guillermo Nieves Molina
 *   Proyecto fin de carrera: Estudio de las posibilidades didacticas en
 *   ingenieria de control del LEGO Mindstorms NXT
 *   Año 2008
 *
 *   Este programa utiliza un giroscopo para mantener estable el balancin
 *   NXTWay, tomando como consigna velocidad de giro nula.
 *
 *   El codigo incluye un lazo de control con un regulador PID.
 *
 *   Este codigo se distribuye libremente bajo licencia GNU GPL; una copia
 *   completa de dicha licencia se puede encontrar en
 *   http://www.gnu.org/licenses/gpl.txt
 */


  #define Puerto IN_1 //  Puerto del giroscopo
  #define Espera 70   //  milisegundos
  #define P 85      //
  #define I 30     //  Parametros del regulador
  #define D 80    //
  #define FactorEscala 100 //Factor de escala para compensar la salida

  task main(){

  //Variable que almacena la consigna
  int SetPoint;

  //Variable que almacena las lecturas
  int Lectura;

  //Variables para el algoritmo PID
  int TerminoP,TerminoI,TerminoD;
  int ErrorIntegral,ErrorPrevio;

  //Variables usadas en los calculos
  int Error, Potencia;

  //Variables usadas en la escritura
  byte Etiqueta;
  short Comprobante;
  string Tmp;

  DeleteFile("nxtway.txt");
  CreateFile("nxtway.txt", 50000, Etiqueta);

  //Iniciamos el sensor
  SetSensorHTGyro(Puerto);

  //La consigna es un movimiento de 0 grados por segundo.
  SetPoint=0;


  while(true)
  {
       //Almacena el error anterior (para el PID)
       ErrorPrevio=Error;

       //Leemos, teniendo en cuenta el offset
       Lectura=SensorHTGyro(Puerto,-4);
```

```
        //Diferencia de medidas
        Error=Lectura;

        //Aplica el PID:

        //Termino proporcional
        TerminoP=P*Error;

        //Termino integral
        ErrorIntegral+= Error;

        TerminoI=I*(ErrorIntegral);

        //Termino derivativo (aproximacion)
        TerminoD=D*(Error-ErrorPrevio);

        //Suma de terminos
        Potencia=(TerminoP+TerminoI+TerminoD)/FactorEscala;

        //Guardamos posicion antes de actuar
        Tmp = NumToStr(Potencia);
        WriteLnString(Etiqueta,Tmp, Comprobante);

        //Satura y pasa a positiva la potencia a entregar a los motores
        if(abs(Potencia)>100)
             Potencia=100;
        else if(abs(Potencia)<20)
             Potencia=20;
        else Potencia=abs(Potencia);

        //Gira en un sentido o en otro
        if(Error<0)
             OnFwdReg(OUT_BC, Potencia, OUT_REGMODE_SYNC);
        else
             OnRevReg(OUT_BC, Potencia, OUT_REGMODE_SYNC);

        //Resetea el termino de error integral
        ErrorIntegral=0;

        //Da tiempo a los motores para actuar
        Wait(Espera);

        Float(OUT_BC);

    }

    CloseFile(Etiqueta);

}
```

C.6. Estacionamiento en batería del Autoparker

Correspondiente al archivo `autoparkerbateria.nxc`. El programa aquí presentado es una simple y vistosa demostración de las capacidades del NXT, no estando específicamente relacionado con la ingeniería de control. Para su correcto funcionamiento, el coche debe situarse paralelo a la línea de *vehículos estacionados*, con el sensor de distancia apuntando a ésta.

Al inicio del programa, el sensor mide la distancia que le separa de la línea de coches y va avanzando hasta que detecta un aumento sustancial en esta medida. Cuando esto ocurre, comienza a medir el espacio disponible y, si es el suficiente, se detiene y comienza la rutina de aparcamiento, consistente en un giro de 90 grados y un recule hasta quedar cerca de la pared.

Para que los parámetros aquí especificados sean efectivos, se recomienda utilizar el montaje del coche propuesto aquí en el capítulo 7. Es necesario el empleo de la brújula de HiTechnic para las labores de orientación del vehículo.

```
/*
 *   Autor: Guillermo Nieves Molina
 *   Proyecto fin de carrera: Estudio de las posibilidades didacticas en
 *   ingenieria de control del LEGO Mindstorms NXT
 *   Año 2008
 *
 *   Este programa realiza un proceso de aparcamiento en bateria completo,
 *   teniendo en cuenta el tamaño del espacio para aparcar, la distancia al
 *   resto de los "coches" y la posicion final del vehiculo.
 *
 *   Este codigo se distribuye libremente bajo licencia GNU GPL; una copia
 *   completa de dicha licencia se puede encontrar en
 *   http://www.gnu.org/licenses/gpl.txt
 */


//Puertos de motores
#define MotorTraccion OUT_B
#define MotorDireccion OUT_C

//Puertos de sensores
#define SensorDistancia IN_3
#define SensorBrujula IN_1
#define SensorLuz IN_2

/*Espacio que necesita el coche para aparcar. La formula que relaciona este
numero
con distancia en mm se encuentra en la memoria del proyecto. */
#define EspacioCoche 400

//Variables globales
int DistanciaInicial=0;
int MedidaDistancia=0;
int TamanoHueco=0;
int Orientacion=0;


/*---Rutina de medicion de espacio---*/
int MideHueco()
  {
  //Resetea la cuenta del tacometro para medir el hueco
  ResetRotationCount(MotorTraccion);
  Wait(10);

  //Comienza la medida del hueco
  while(SensorUS(SensorDistancia)>2*DistanciaInicial)
    {
    OnFwd(MotorTraccion, 60);
    }

  //MotorRotationCount funciona mejor que las otras funciones, no resetea
  TamanoHueco = MotorRotationCount(MotorTraccion);

  return TamanoHueco;
  }
```

```
/*---Rutina de aparcamiento---*/
void Aparca()
  {
  //Medida de la orientacion del coche
  Orientacion=SensorHTCompass(SensorBrujula);

  //Giramos las ruedas; 70 grados es un buen angulo
  RotateMotor(MotorDireccion,50,70);

  Wait(100);

  //Obligamos a girar 90º, teniendo en cuenta que la brujula va de 0 a 359

  if(Orientacion < 91)
  {
    until((SensorHTCompass(SensorBrujula)) == Orientacion + 270)
      {
      OnRev(MotorTraccion,60);
      }
  }

  else
  {
    until(SensorHTCompass(SensorBrujula) == (Orientacion - 90))
      {
      OnRev(MotorTraccion,60);
      }
  }

  Off(MotorTraccion);

  //Y hacia atras
  RotateMotor(MotorDireccion,50,-70);

  //Iniciamos el sensor de luz
  SetSensorLight(SensorLuz);

  //El coche retrocede
  until(Sensor(SensorLuz)>35)
    {
    OnRev(MotorTraccion,60);
    }

  //Fin
  Off(MotorTraccion);
  }



/*---Main del programa---*/
task main(){

  /*Iniciamos el sensor de distancia. El de luz tambien podemos iniciarlo aqui
  si queremos una bonita luz trasera*/
  SetSensorLowspeed(SensorDistancia);
  SetSensorLowspeed(SensorBrujula);

  /*En principio suponemos que el coche va perfectamente alineado.
  Mide la distancia y comienza la marcha*/
  DistanciaInicial = SensorUS(SensorDistancia);
```

```
  do
  {
    until(SensorUS(SensorDistancia)>2*DistanciaInicial)
      {
      OnFwd(MotorTraccion,80);
      }

  //Llama a la rutina de medida
  MideHueco();

  }while(TamanoHueco<EspacioCoche);

  Off(MotorTraccion);

  //Llama a la rutina de aparcamiento
  Aparca();

}
```

Bibliografía

[1] http://bricxcc.sourceforge.net/, web del proyecto NBC/NXC Consulta de 2008.

[2] Daniele Benedettelli. *Programming LEGO NXT Robots using NXC (beta 30 or higher).* Autopublicado en internet, 2007.

[3] John Hansen. *Not eXactly C (NXC) Programmer's Guide.* Autopublicado en internet, 2007.

[4] http://mindstorms.lego.com/Overview/NXTreme.aspx, sección en la web oficial de LEGO que recoge el código fuente del firmware, documentación técnica sobre el NXT y drivers, además de enlaces diversos. Altamente recomendable para aplicaciones avanzadas Consulta de 2008.

[5] http://www.lfb.rwth-aachen.de/en/education/ws07/mindstorms.html, web del proyecto Matlab meets Mindstorms Consulta de 2008.

[6] http://www.mindsensors.com, web de Mindsensors, empresa fabricante de hardware para el NXT Consulta de 2008.

[7] http://www.hitechnic.com, web de HiTechnic, empresa fabricante de hardware para el NXT Consulta de 2008.

[8] http://www.techno-stuff.com, web de Techno-Stuff, empresa fabricante de hardware para el RCX y el NXT Consulta de 2008.

[9] http://www.vernier.com, web de Vernier, empresa de instrumentación que fabrica hardware para el NXT Consulta de 2008.

[10] Michael Gasperi; Phillipe Hurbain. *Extreme NXT.* Apress, 2007.

[11] http://ldd.lego.com/, Sitio de descarga del LEGO Digital Designer Consulta de 2008.

[12] http://www.philohome.com/nxt.htm, web de Phillipe Hurbain, coautor del libro Extreme NXT Consulta de 2008.

[13] http://www.teamhassenplug.org/robots/legway/, web de Steve Hassenplug, creador del LegWay, entre otros montajes Consulta de 2008.

[14] http://web.mac.com/ryo_watanabe/, web de Ryo Watanabe, creador del NXTWay-GS Consulta de 2008.

[15] http://mindstorms.lego.com/MeetMDP/default.aspx, web del Programa de Desarrolladores de Mindstorms (MDP) Consulta de 2008.

[16] http://nxtasy.org/category/nxt-repository/projects, repositorio de proyectos de NXTasy.org Consulta de 2008.

[17] http://es.wikipedia.org/wiki/Segway, Entrada sobre el Segway en Wikipedia (ES) Consulta de 2008.