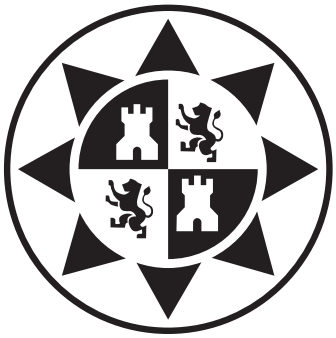
UNIVERSIDAD POLITÉCNICA DE CARTAGENA



Escuela Técnica Superior de Ingeniería Industrial

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Diseño y control de un prototipo carro-péndulo basado en LEGO Mindstorms NXT

Titulación	Ingeniería Técnica Industrial, especialidad en Electrónica Industrial
Intensificación	Automática
Autor	Ismael Sánchez Mendoza
Directores	Julio José Ibarrola Lacalle Jose Manuel Cano Izquierdo

Cartagena, Marzo de 2009

ÍNDICE

5. Sensorización

6. Identificación del sistema

7. Experimentos de control

ÍNDICE DE FIGURAS

CAPÍTULO 1

INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

LEGO Mindstorms es un kit de robótica educativa desarrollado por LEGO. En concreto, el modelo NXT (ver Figura 1.1) es el sucesor de otro más antiguo, el RCX. Éste, que fue el primer modelo comercializado, nació de los acuerdos de colaboración entre LEGO y el MIT (Instituto Tecnológico de Massachusetts). Su éxito hizo que rápidamente se extendiera su uso a gran cantidad de aficionados a la robótica.

No son pocos los proyectos y trabajos que hay ya realizados usando LEGO Mindstorms. Incluso existen algunos en ingeniería de control avanzada. Basta con una rápida búsqueda en Internet para darse cuenta de la cantidad de material disponible, tanto para expertos como para quienes no lo son tanto, sobre este famoso "*juguete*" de LEGO.

Y es que aunque la comercialización del NXT se ha enfocado hacia el mercado recreativo, la flexibilidad y capacidad de éste lo han llevado a ser aceptado ampliamente como una herramienta de docencia a tener en cuenta.

Figura 1.1. Imagen del *brick o* ladrillo programable del NXT, la base de su funcionamiento

Gran parte de la información básica y características del LEGO Mindstorms NXT y de sus distintos componentes, necesarias para la elaboración de este proyecto, se han obtenido de diversas webs y comunidades de Internet, pero sobre todo, este proyecto se fundamenta en el trabajo anterior "*Estudio de las posibilidades didácticas en ingeniería de control del LEGO Mindstorms NXT*" (2008) [1], de Guillermo Nieves Molina, compañero de universidad. Sin duda este trabajo me ha sido tremendamente útil para el acercamiento inicial a este robot, y lo recomiendo al lector si desea encontrar una muy buena recopilación de información sobre todo lo relacionado con el NXT, y una valiosa ayuda para sentar las bases antes de comenzar con trabajos más complicados.

Siguiendo este trabajo se plantea la idea de desarrollar una aplicación concreta, que muestre el potencial del NXT para la ingeniería de control y su posible uso docente a nivel universitario. Como punto fuerte, se tiene su relativamente bajo precio y su gran versatilidad (ver Figura 1.2), en comparación con algunas maquetas de laboratorio comerciales actuales.

Figura 1.2. Imagen de distintos montajes comerciales del NXT, con sus sensores y actuadores

1.2. Objetivos

El objetivo fundamental del presente proyecto es el diseño, construcción e implementación de un sistema de control automático de la oscilación de un péndulo montado sobre el chasis de un vehículo o plataforma móvil. Todo ello construido con elementos del Lego Mindstorms NXT, sus sensores y motores, y las piezas de LEGO correspondientes. La tarea básica del sistema de control será minimizar el balanceo del péndulo en diversas circunstancias, aunque más adelante se explicará esto con mayor detenimiento.

En principio se pueden considerar tres experimentos individualmente:

- Controlar la posición de la plataforma
- Controlar la oscilación del péndulo
- Controlar simultáneamente la posición de la plataforma y la oscilación del péndulo.

Este proyecto se ha centrado principalmente en la creación de una maqueta de control, por lo que tan sólo se comentarán las características del NXT relacionadas con esta aplicación y que son necesarias para su comprensión. Así pues, no se dará la explicación más detallada de las especificaciones o la descripción del propio NXT, del que se puede encontrar una amplia información en la bibliografía proporcionada y especialmente en el trabajo antes mencionado [1].

El desarrollo completo de esta aplicación constará de las siguientes tareas: la propia construcción, la elección de sensores, las pruebas, el modelado e identificación del sistema, la programación e implementación del controlador, y la búsqueda de usos o montajes alternativos, entre otras.

Sin embargo no se puede olvidar un objetivo adicional, como es el mostrar un ejemplo concreto de las capacidades y posibilidades del NXT para su uso docente en el área de la automática.

CAPÍTULO 2

EL SISTEMA COCHE-PÉNDULO

En este capítulo se comenzará el análisis general de la maqueta que nos ocupa. Así mismo, se describirán distintos proyectos similares ya realizados y que han servido de ayuda.

2.1. Antecedentes y trabajos previos

Como se ha comentado, ésta no es de las primeras aplicaciones desarrolladas con el NXT, que abarcan desde los automatismos más simples (Figura 2.1) hasta proyectos más ambiciosos. Como ejemplo, el robot estable sobre dos ruedas NXTWay-G de Ryo Watanabe [5], que usa hábilmente los sensores proporcionados con el NXT (Figura 2.2).

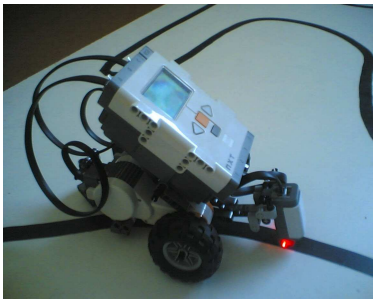


Figura 2.1. Imagen de una sencilla aplicación para el seguimiento de líneas

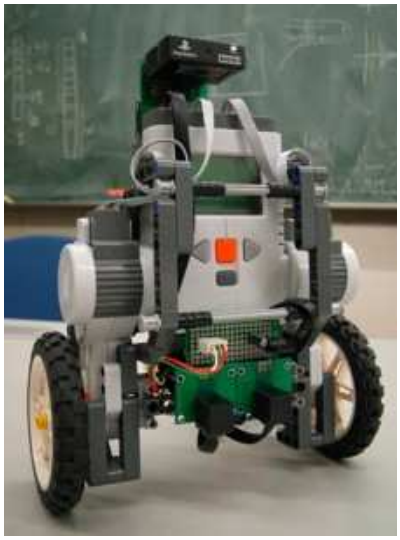


Figura 2.2. El NXTWay-G de Ryo Watanabe, para el desplazamiento a dos ruedas

Como se ve, el potencial de procesamiento del NXT hace que las posibilidades sean inmensas. De hecho, ya hay también realizado un experimento de control con un sistema coche-péndulo muy similar al expuesto en este proyecto. Se puede decir que ha sido en gran medida el responsable del surgimiento de nuestra idea. Veámoslo en el siguiente apartado.

2.1.1. *A LEGO-Based Control Experiment*

Con ese nombre se publicó un artículo en la revista de automática y control *IEEE Control Systems Magazine,* a finales del año 2004 [4]. Sus autores eran Peter Gawthrop, miembro y profesor del Departamento de Ingeniería Mecánica de la Universidad de Glasgow, y Euan McGookin, también profesor y miembro del Departamento de Ingeniería Eléctrica y Electrónica de la misma universidad.

En él se expone el experimento de control que ambos desarrollaron, basado en el modelo antiguo de LEGO Mindstorms, el RCX (Figura 2.3). El objetivo principal era, como en nuestro caso, el control del ángulo de oscilación de un péndulo mediante el desplazamiento del carro sobre el que va montado. La idea consistía en que si el péndulo comenzaba a oscilar, debido por ejemplo a un golpe con la mano, el coche debería moverse en consecuencia para anular esa oscilación de la manera más efectiva posible.

Como se ve, la idea de este proyecto no es nueva, pero aun así es muy interesante crear y analizar profundamente algo similar que muestre también las capacidades del nuevo NXT.

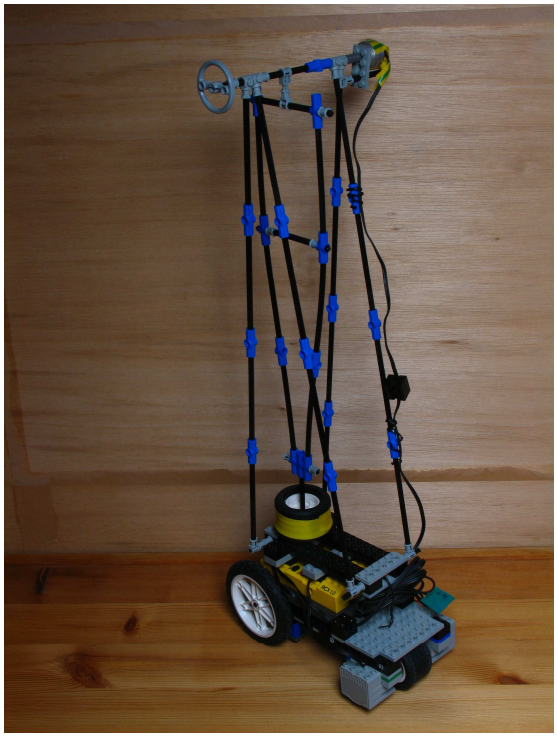


Figura 2.3. Imagen de la versión del *cart-pendulum system de 2006* desarrollado por Peter y Euan sobre el RCX

El documento completo [4] se puede encontrar y descargar gratuitamente en la página web dedicada a sus experimentos diversos con LEGO [3], junto a varios ejemplos más de aplicaciones (entre ellas una igual pero con el péndulo invertido). También se pueden encontrar vídeos de funcionamiento del aparato, algunos segmentos del código utilizado e imágenes detalladas de su estructura.

Si hacemos un repaso rápido y sin entrar en muchos detalles de esta construcción, encontramos algunos aspectos destacables. Como ya se ha dicho, el modelo de Euan y Peter se construyó en torno al predecesor del NXT, el RCX, puesto que el primero todavía no había salido a la luz. Esto supuso algunas complicaciones:

- Se necesitaba algún tipo de sensor de ángulo acoplado al eje de rotación del péndulo para medir el ángulo girado por éste. Sin embargo, el sensor de rotación oficial del RCX tan sólo constaba de 16 posiciones por vuelta (según [6]), lo que significa la muy baja resolución de 22.5 grados ($360^0/16$). Esto resultaba insuficiente para un péndulo que sólo oscilaría del orden de unos

pocos grados. Para solucionarlo, se desechó el uso de este sensor y en su lugar se utilizó un potenciómetro rotativo comercial de 10kΩ nominales, conectado directamente a una de las entradas del RCX mediante su cable oficial de sensores. El potenciómetro se puede observar en la Figura 2.4.

- Como los motores de corriente continua del RCX no disponían de tacómetros internos, no podían devolver ni la posición ni la velocidad del coche, variables que eran totalmente necesarias para los posteriores cálculos del software.

  - Para solventar el problema de la velocidad, se tuvo en cuenta el hecho de que un motor de corriente continua a circuito abierto genera una tensión proporcional a su velocidad angular. Sabiendo esto, se cortó uno de los cables de un motor de RCX, y se interpuso una elevada resistencia entre sus terminales. Este motor se unió mediante un tren de engranajes (con una relación 24:40) a uno de los motores motrices del coche. Así, ya estaba listo para proporcionar una tensión medible e indicadora de la velocidad angular de las ruedas. Una imagen de este tren de engranajes se puede ver en la Figura 2.6

  - En cuanto a la medida de la posición, se usó el sensor de rotación del RCX (ver Figura 2.4) conectado al eje de las ruedas mediante otro engranaje (en una relación 24:15), para aumentar su ya mencionada baja resolución.

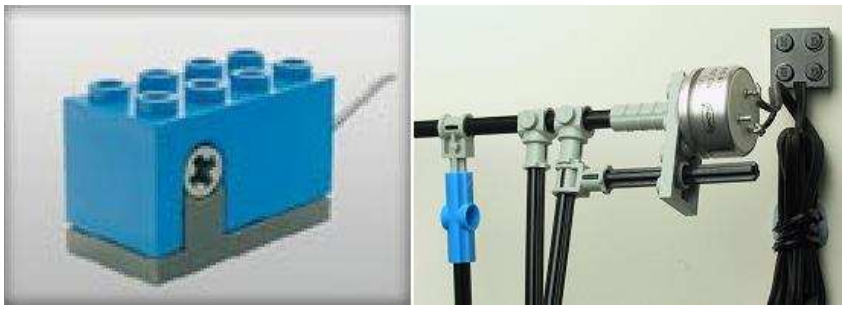


Figura 2.4. A la izquierda, imagen del sensor de rotación del RCX. A la derecha, detalle del montaje del potenciómetro sobre el eje de rotación del péndulo, junto a su cable.

Figura 2.5. Imagen de un motor DC del RCX modificado para funcionar como sensor de velocidad angular.

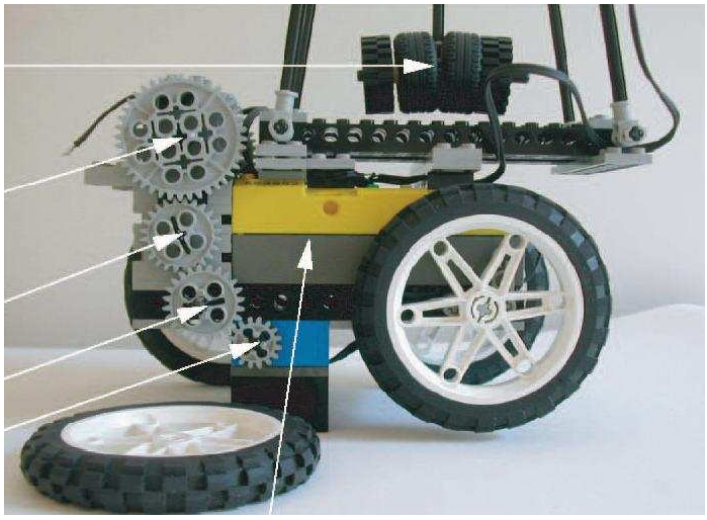


Figura 2.6. Detalle lateral de la idea de Euan y Peter. Se aprecia el tren de engranajes para transferir el movimiento al sensor de velocidad (en la parte superior) y al sensor de rotación (parte inferior)

A pesar de estos pequeños impedimentos, consiguieron que la maqueta funcionara perfectamente, tal y como se puede ver en sus grabaciones. Identificaron el sistema y también obtuvieron las ecuaciones necesarias para implementar correctamente el programa en el RCX. Esto lo hicieron en lenguaje de programación C y trabajando sobre el sistema operativo para el RCX *brickOS*. En este proyecto se creará algo similar, pero con la salvedad de que se utilizará el más moderno NXT, lo que implica el ahorro de los problemas antes vistos. Dejemos pues apartado de momento este modelo y centrémonos en el nuestro.

2.2. Aproximación al sistema.

Aquí se hará un esbozo de las características globales, tanto físicas como de funcionamiento, que fueron la base para comenzar a desarrollar los objetivos.

2.2.1. Aspecto físico general

Una vez llegado a este punto ya se debe tener una idea cercana a lo que debería ser la forma física nuestro aparato. A grandes rasgos, lo que se debía construir con las piezas de LEGO Technic incluidas con el NXT era lo siguiente:

- Un carro/vehículo, que pudiera sostener o alojar en su interior al propio ladrillo programable, su elemento fundamental. Quedó rápidamente desechada la posibilidad de hacerlo con tres ruedas (tipo triciclo), ya que una rueda sin pareja podría inducir algunas desviaciones en el recorrido del vehículo. Este movimiento debería ser en ambos sentidos, pero en una sola dirección (adelante y atrás). Por lo tanto, el montaje no necesitaría de ruedas directrices (que seguramente implicarían el construir un diferencial, tal y como se describe en [15]), y los ejes serían fijos. Se concluyó entonces que el vehículo dispondría de cuatro ruedas, como cualquier coche típico.

- Una estructura rígida sobre el coche que sirviera para la colocación de un péndulo, que debería tener un sólo plano de oscilación que cortara al coche longitudinalmente. Dicho de otra forma, el péndulo no debería oscilar a izquierda y derecha, sólo adelante y atrás. Además, la estructura debería ser lo suficientemente alta como para que la oscilación del péndulo fuese apreciable y no demasiado rápida. También se habría de colocar una masa adecuada en su extremo.

2.2.2. Idea del funcionamiento

Parecía bastante clara la forma de actuar del robot**:** cuando el péndulo se balanceara a causa de una acción externa, como por ejemplo un impulso con la mano o un movimiento brusco del coche, la respuesta debería ser, dicho de una forma burda, mover el coche adelante y atrás para conseguir detener esa oscilación. Esto debería hacerse de la forma más eficiente y rápida posible, devolviendo el sistema al estado de reposo inicial.

Además se pueden dar (y así se hará) distintas variantes un poco más complejas a nuestro problema, como por ejemplo, hacer llegar el coche a un punto

determinado, o que se mueva durante un tiempo concreto, intentando en todos los casos que el péndulo oscile lo mínimo posible.

Si hacemos una pequeña comparación de este problema con algún sistema real, encontramos gran similitud con el movimiento de cargas a lo largo de la típica grúa usada para la construcción. La carga oscilará colgada del cable debido a los movimientos de la maquinaria cuando el operario lo indique, pero también podría oscilar más descontroladamente debido a la acción del viento. La respuesta en ambos casos podría ser una acción de control para minimizar ese balanceo, sin que el trabajador tuviera que estar presente ni actuar manualmente.

Algo parecido, aunque obviamente a escala mucho menor, es lo que se intentará mostrar con la maqueta. Para ello se necesitará algún tipo de sensor que devuelva el estado del péndulo, y programar al NXT para que opere con estos datos y genere la respuesta de control adecuada. Pero de momento ya hay unas bases sobre las que empezar a trabajar.

CAPÍTULO 3

CONSTRUCCIÓN DE LA MAQUETA

En este capítulo se explicarán los pasos dados durante el montaje, los problemas surgidos durante el mismo, y las soluciones adoptadas.

3.1. Montaje

Con lo visto hasta ahora ya se tenía lo suficiente para hacernos una idea aproximada de cómo abordar este montaje.

En cuanto al kit utilizado, se comenzó con el *LEGO Mindstorms Education Base Set* [7]. Como su nombre indica, éste es un kit de base, más orientado a la iniciación con el NXT en las aulas, y con una pequeña guía para el montaje de un único y simple robot. Aunque disponía de todos los elementos electrónicos necesarios como el ladrillo, tres motores, diversos sensores, cables y batería, no incluía el software ni el cargador, que se tuvieron que adquirir aparte. Además, rápidamente fue patente que no contaba con suficientes piezas (sobre todo de elementos largos, como ejes o vigas), que eran básicos para dar altura a la estructura. Por este motivo se tuvo que recurrir a otra caja, el *LEGO Education Resource Set* [7] (ver Figura 3.1). Se trata de un kit de apoyo al de base sin elementos electrónicos (lo que reduce en gran medida su precio) pero con todas las piezas de LEGO Technic necesarias para montajes más complicados.

Figura 3.1. Imagen del kit básico LEGO Mindstorms Education

Hay que destacar que debido a la gran cantidad de posibilidades que ofrecen las piezas plásticas de LEGO, la construcción consiste más que nada en “echarle imaginación”, surgiendo cada poco tiempo los pequeños objetivos de “mantener fija tal sección”, “permitir flexibilidad en este punto”, o “intentar unir con aquello”, y todas estas cuestiones que pueden crear algún que otro quebradero de cabeza, por lo que en algunos momentos puede convertirse verdaderamente en una tarea tediosa.

Por ello el aspecto final variará en gran medida de un diseñador a otro. El diseño presentado aquí seguramente no sea ni mucho menos el más adecuado de todos, ni el más eficiente si hablamos del número de piezas utilizadas, pero cubre las expectativas de funcionamiento buscadas.

3.1.1. Desarrollo y problemas surgidos

Dividiremos este apartado en dos partes que se podrían considerar prácticamente independientes: el montaje del vehículo, y el montaje de la estructura del péndulo sobre el vehículo.

- **Montaje del vehículo:**

Partiendo de que había que construir un coche, lo primero que se decidió era referente al movimiento. Teniendo en cuenta que el peso que debería moverse podría ser bastante apreciable, se decidió usar dos motores para asegurar la tracción necesaria a las ruedas delanteras.

Estos motores se unieron a las ruedas delanteras mediante sendos ejes, uno distinto para cada una, permitiendo movimientos independientes para cada rueda y dejando la puerta abierta a futuros giros del vehículo, cosa que sería imposible si las ruedas compartieran el mismo eje. Además, no se consideró necesario utilizar ningún tipo de engranaje, por lo que a las ruedas se transmitiría la fuerza del motor íntegramente y sin modificar, como se puede ver en la Figura 3.2.

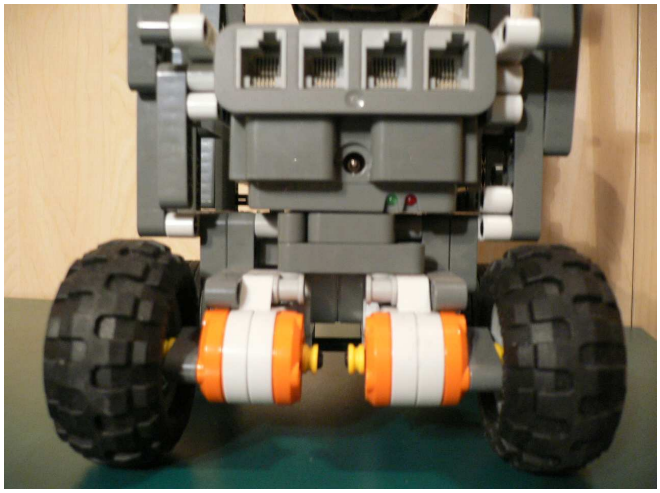


Figura 3.2. Imagen anterior del vehículo, donde se aprecian los dos ejes delanteros

Las ruedas libres, las traseras, girarían unidas mediante un único eje, formado por la unión de dos ejes más cortos (Figura 3.3).

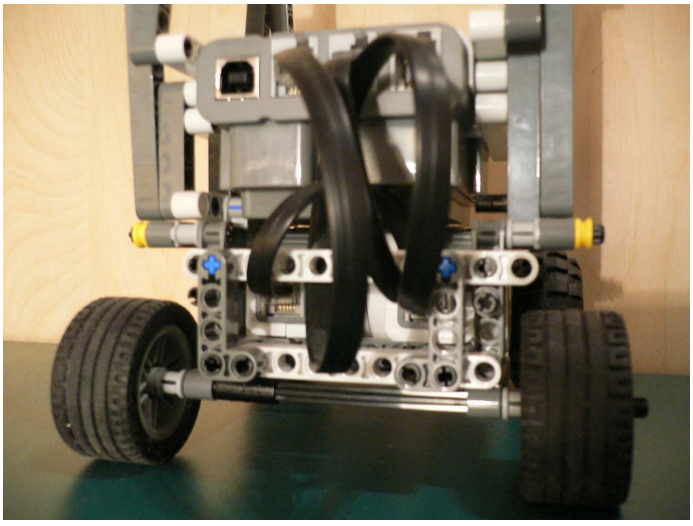


Figura 3.3. Imagen posterior del vehículo y su eje trasero

En cuanto a la elección del tipo de ruedas, en un principio se usaron en todas el modelo de neumático *Ballon* de *56x26 mm* (con referencia de LEGO

4297209). Posteriormente se observó que el ladrillo, montado sobre los motores e incrustado en el coche, quedaría inclinado debido a la forma de los motores, y esa inclinación podría haber afectado más tarde a la construcción de la estructura de sostén del péndulo.

Para compensar esta desviación, se cambiaron los neumáticos traseros por un modelo más pequeño, el *Normal 43,22x22 mm (ref. 4184286)*, que ya dejarían el coche "nivelado", como se aprecia en la Figura 3.4.

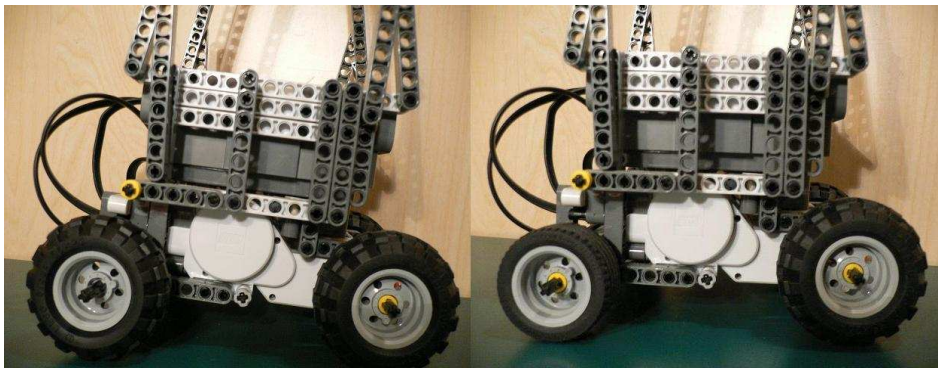


Figura 3.4. Comparación de inclinación del coche usando distintos tipos de neumático. A la izquierda el montaje inicial y a la derecha el vehículo ya nivelado.

Una de las mejoras de los motores del NXT respecto a los del RCX, es que éstos ya incluyen un sensor de rotación tacométrico en su interior, por lo que no haría falta ningún sensor de velocidad ni de posición externo tal y como hicieron Peter y Euan en su trabajo, pues los motores del NXT nos devuelven sin problemas estas variables. Así pues, de momento el coche no tendría ningún otro elemento electrónico más.

De las tareas más costosas, en lo que atañe al montaje, fue el encontrar la manera de sujetar el ladrillo programable encima del coche, de la manera más centrada posible, sin inclinación, ajustándolo a la forma de los motores y sobre todo, sin excederse demasiado en las medidas del vehículo. Se consiguió "atándolo" a la parte inferior con gran cantidad de piezas. Es importante tener en cuenta que este ladrillo se montó con la batería recargable, lo que hace que su parte posterior sobresalga un poco más de lo que lo haría si estuviésemos usando pilas (lo que afectó a la creación de la guía de montaje, como se verá en el apartado 3.3). El resultado final se ve en la Figura 3.5.

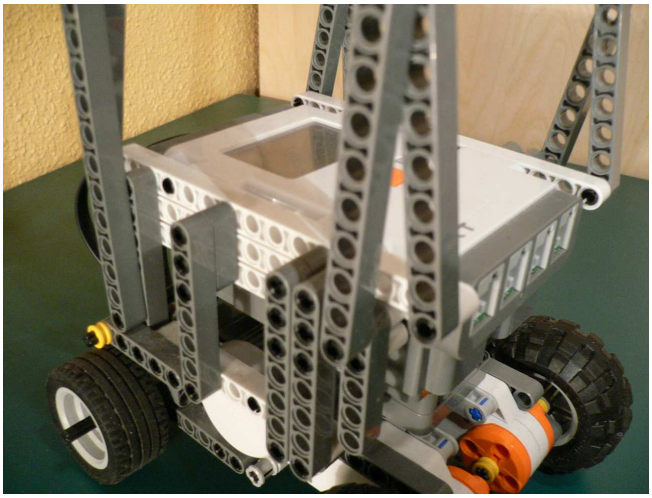


Figura 3.5. Imagen del *ladrillo* programable sobre el coche

Se puede ver en la vista inferior del vehículo (Figura 3.6), que los motores se colocaron en contacto el uno con el otro, para evitar excederse en la anchura final y a la misma vez obtener una estructura más compacta.

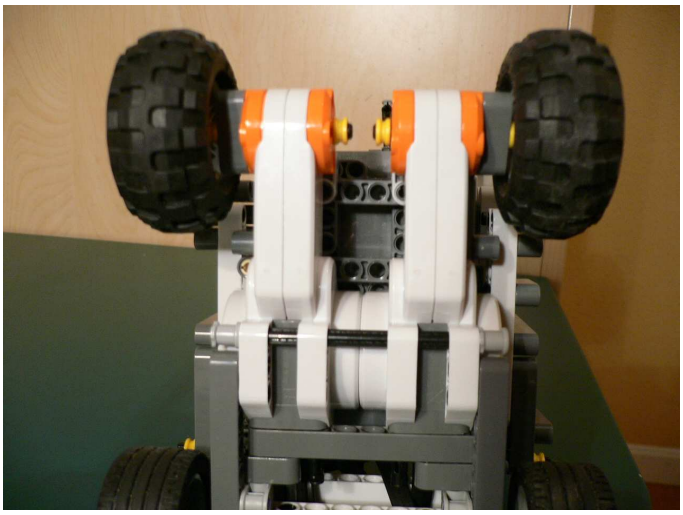


Figura 3.6. Imagen inferior del vehículo, donde se ve la disposición de los motores

- **Montaje de la estructura del péndulo:**

Una vez se tenía el coche construido, el siguiente paso fue el montaje del péndulo y de la estructura que lo sostendría.

Para la estructura se usaron básicamente piezas de tipo viga, para darle una forma triangular y una altura considerable. Por la base, se sujetó convenientemente a los laterales del ladrillo, mediante más piezas viga, y aproximadamente a la mitad de altura, se colocaron vigas de apoyo, para mejorar la inmovilización del conjunto. Esta estructura se ve en la Figura 3.7.

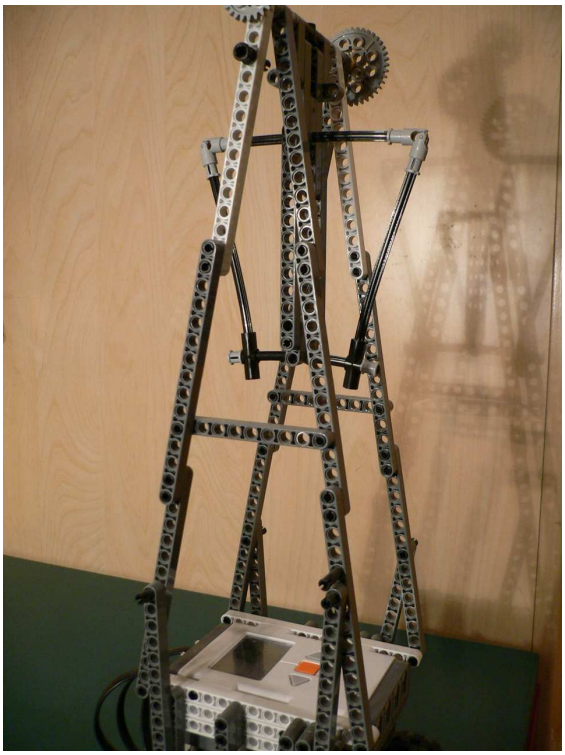


Figura 3.7. Imagen de la estructura para sostener el péndulo, con éste desmontado.

El eje para sostener el péndulo (ver Figura 3.8) se insertaría en la parte superior de la estructura, y estaría formado por dos ejes más pequeños unidos por su extremo mediante la pieza correspondiente. Así se le daría la longitud suficiente. Obviamente se le dejó exento de fricción para permitirle un movimiento libre.

Figura 3.8. Imagen del eje del péndulo y la unión del péndulo al eje en dos puntos

El péndulo se unió al eje en dos puntos (ver Figura 3.8), que junto al uso de las piezas de apoyo antes comentadas, evitarían en la medida de lo posible la oscilación o vibración indeseada del péndulo a derecha-izquierda. El cuerpo del péndulo se construyó sobre todo en torno a piezas tipo eje y con la máxima longitud. Además, con la idea de usar en su totalidad sólo piezas de LEGO, la masa que se colocó en su extremo fueron tres ruedas idénticas a las de los ejes delanteros. Este peso proporcionaría la inercia necesaria para el balanceo continuo del péndulo. En las figuras 3.9 y 3.10 se aprecia el aspecto final del péndulo.

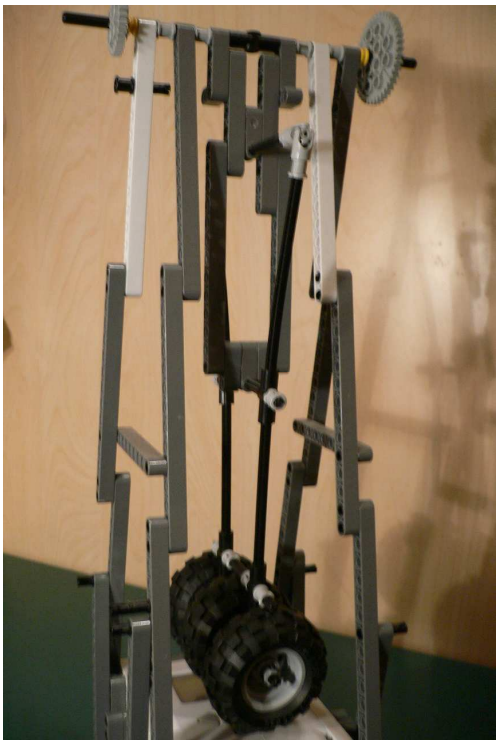


Figura 3.9. Imagen final del conjunto estructura-péndulo

Figura 3.10. Imagen del peso colocado en el extremo del péndulo

Un dato importante es que la parte inferior del péndulo (Figura 3.10) es fácilmente desmontable. Se recomienda desmontarlo cuando se prevean periodos largos sin uso del aparato. Esto evitará posibles deformaciones en el eje superior (recordemos que es de plástico), por una prolongada exposición al peso que sustenta, y que a la larga podrían producir problemas para el balanceo del péndulo.

Ya sólo quedaba la elección de un sensor de rotación adecuado y su acoplamiento al eje del péndulo. Éste fue el mayor problema que surgió con respecto al montaje completo de la maqueta, y por su complejidad merece un capítulo aparte (Capítulo 5).

3.2. Dimensiones finales

Las dimensiones finales aproximadas de la maqueta se exponen a continuación:

ANCHURA:

Longitud del eje delantero: 13 cm

Longitud del eje trasero: 14 cm

Anchura máxima: 14 cm (correspondientes al eje trasero, sin tener en cuenta el eje del péndulo)

LONGITUD:

Distancia entre ejes: 11.3 cm

Longitud máxima: 16 cm (extremos de las ruedas)

ALTURA:

Altura del coche (hasta el frontal del ladrillo): 11.2 cm

Altura total (hasta el eje del péndulo): 42.8 cm

PÉNDULO:

Longitud del eje de rotación del péndulo: 14.3 cm (sin sensor acoplado)

Longitud del péndulo (sin peso en su extremo): 28.5 cm

Masa de la maqueta sin el péndulo (incluye carro y estructura) : 0.7427 kg

Masa del péndulo completo: 0.0881 kg

3.3. Creación de una guía de montaje

Se planteó crear algún documento con las instrucciones para montar la maqueta, con el fin de evitarnos un gran problema en el caso de desmontarla y tener que construirla de nuevo.

Después de estudiar distintos programas de CAD para crear esta guía de montaje, como el genérico Google Sketchup, o el más antiguo LDraw, se decidió que lo más idóneo era usar el Lego Digital Designer (LDD), software creado expresamente para representar en 3D construcciones de LEGO, y creación de las respectivas instrucciones paso a paso del ensamblaje de las piezas. El programa, gratuito, se puede encontrar en su página web [12].

Este software es muy sencillo de utilizar por su simple interfaz, y sobre todo, por la facilidad a la hora de escoger las piezas, que pueden ser ordenadas según distintas características (tipo, color, caja, etc.). Ver figura 3.11.

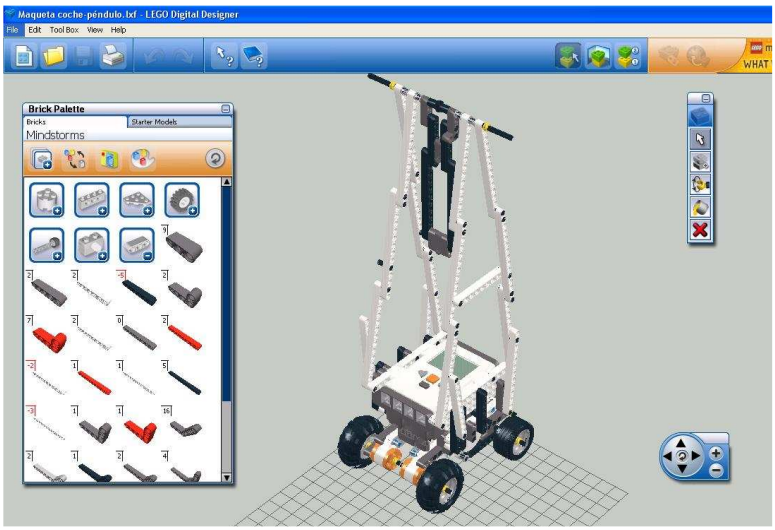


Figura 3.11. Imagen de la interfaz de LEGO Digital Designer

El montaje en el entorno LDD transcurrió en general con normalidad, aunque también surgieron algunos inconvenientes o consideraciones a tener en cuenta:

- En la construcción real, el ladrillo se montó con su batería insertada. Sin embargo, en LDD, el ladrillo no incluye la batería y se montó sin ésta. Como se comentó anteriormente, esto crea una diferencia de tamaño, y el resultado es que en LDD hay un hueco justo entre el ladrillo y el coche, que es el espacio que debería llenarse con la batería.

- LDD tiene el problema de que en algunos momentos no permite la conexión entre piezas, debido a que las considera forzadas, mientras que en la realidad estas uniones si serían posibles, tal vez debido a tolerancias en la misma fabricación. Esto ocurrió únicamente en dos puntos de la estructura del péndulo, aproximadamente a mitad de altura, y sólo en el lateral derecho. Aunque no se ha conseguido la unión completa en LDD, se deja indicada y es muy fácil completarla.

- En la realidad, parte del péndulo se montó en torno a piezas tipo eje, que se flexionaron para conseguir la forma adecuada. Esta torsión en LDD no es posible, por lo que no se representa el péndulo completo. Sin embargo la parte que falta es muy sencilla de construir y se adjuntan imágenes como aclaración.

- Obviamente tampoco es posible representar en LDD el montaje que se haría luego del potenciómetro, y tampoco de sus piezas de apoyo, debido a que presentó algunas uniones forzadas. Como en el caso anterior, se incluyen imágenes que aclaran este montaje.

En el Anexo B se puede encontrar toda la información necesaria para visualizar este montaje en LDD, que se incluye en el CD adjuntado con este proyecto.

CAPÍTULO 4

LENGUAJE DE PROGRAMACIÓN ROBOTC

En este capítulo se describirá el software que se ha utilizado para la creación de todos los programas de este proyecto, desde las pruebas para la obtención de datos hasta los programas de control finales.

4.1. Elección del lenguaje

RobotC es un lenguaje de programación desarrollado por la *Carnegie Mellon Robotic Academy,* como una alternativa al lenguaje gráfico NXT-G incluido por LEGO en su NXT. Como su nombre indica, está basado en C, y es bastante similar al lenguaje NXC (*Not eXactly C*), también basado en C. RobotC es bastante intuitivo, y suple las carencias derivadas del uso de un lenguaje de tan alto nivel como es NXT-G, permitiendo crear aplicaciones bastante más complejas. También es ampliamente usado en campeonatos de robots. RobotC sólo es funcional en sistemas operativos Windows.

NXC y RobotC son parecidos. El entorno para trabajar con NXC es gratuito, y además éste ya había sido usado para otros trabajos dentro del departamento encargado de este proyecto. A pesar de ello, hubo varios motivos que nos llevaron a decantarnos por RobotC. El primero es que a diferencia de NXC, RobotC es el único lenguaje que dispone de *debugger* paso a paso en tiempo real, característica que es de agradecer, sobre todo cuando los programas que desarrollamos son largos, y requieren comprobaciones cada cierto tiempo. Otra ventaja es que RobotC puede trabajar con datos tipo *float,* o decimales, que también nos facilitaría mucho la

tarea. Además, RobotC no había sido utilizado aún en anteriores trabajos en nuestro departamento, y no era mal momento para comenzar a probarlo.

Por todo ello se decidió finalmente adquirir una licencia para su uso, aunque una versión de prueba de treinta días puede conseguirse gratuitamente en su página web [8]. RobotC incluye asistencia técnica (importante si estamos hablando de algo que no es gratis) y en su web se pueden encontrar guías y tutoriales, junto con programas de ejemplo. También desde su web se puede acceder al foro dedicado exclusivamente a este lenguaje, visitado por una gran cantidad de usuarios que comparten dudas, opiniones y consejos.

Para el funcionamiento de RobotC en el NXT, se incluye un firmware que es necesario descargar al ladrillo, reemplazando al original de LEGO.

4.2. Entorno de trabajo

RobotC incluye su propio IDE (Entorno integrado de desarrollo), con el mismo nombre que el lenguaje.

Éste se caracteriza sobre todo por su gran sencillez y comodidad. Destaca la ventana adicional con las plantillas de todas las funciones disponibles para RobotC, ordenadas por categorías y explicación inmediata de su funcionamiento al hacer doble clic en alguna de ellas. Esto acelera en gran medida el trabajo. A su vez, la instalación suministra una carpeta con gran cantidad de programas de ejemplo, ideales para familiarizarse con el lenguaje. En el menú superior encontramos opciones muy útiles, como la de iniciar el debugger. También dispone de una aplicación para explorar los archivos que tengamos en el NXT (para trabajar con ellos, moverlos, subirlos al PC, etc.), y la opción de descargar fácilmente el firmware de RobotC al NXT. Estas dos últimas aplicaciones necesitan obviamente de tener conectado el NXT a nuestro ordenador, vía USB o Bluetooth.

Entre otros aspectos destacables del lenguaje, está el hecho de que no dispone de demasiadas funciones para configurar los sensores conectados al NXT. Esto es así porque el propio IDE incorpora una ventana para precisar qué sensores estarán conectados a qué puertos, y de qué modo.

En cuanto al espacio de trabajo, se nos facilita la escritura por la numeración de líneas, el uso de distintos colores en función del tipo de palabra, y alguna que otra característica más, como el uso de marcadores. También resulta muy útil la opción de autocompletado mientras se escribe el programa. Para hacerse una idea, la Figura 4.1 muestra una captura de pantalla.

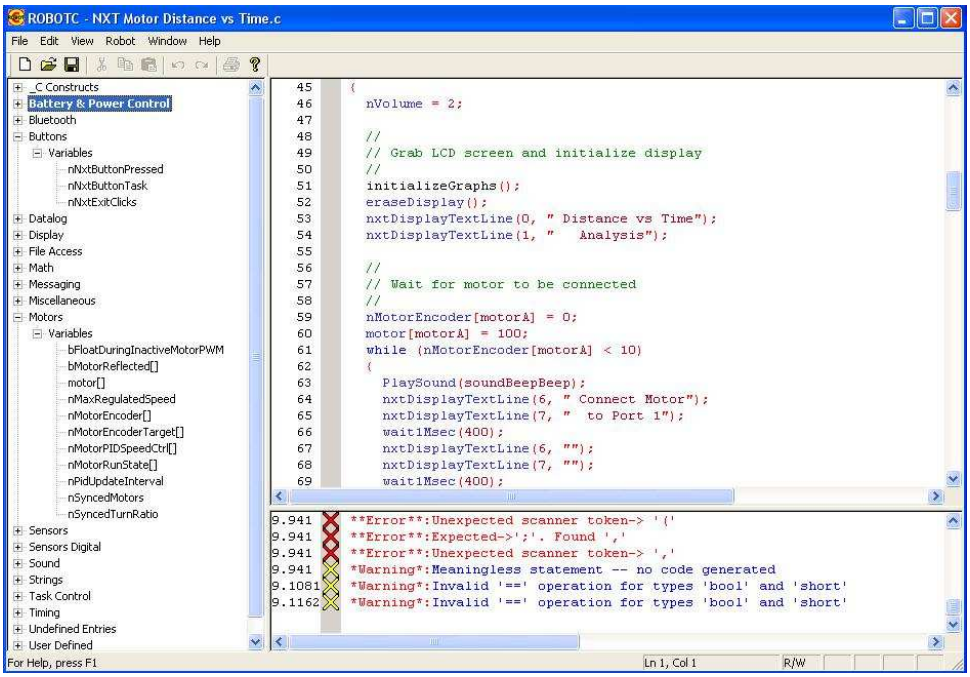


Figura 4.1. Captura de pantalla del IDE RobotC

4.3. El lenguaje RobotC

RobotC comparte la mayor parte de las características del lenguaje C, por lo que es bastante asequible para cualquiera que este anteriormente familiarizado con éste.

Dispone de todas las funciones especiales necesarias para usar en el NXT, relacionadas con los motores, la lectura de sensores, los botones, y las importantes funciones de creación y trabajo con archivos en el NXT. Además, como se ha dicho, estas funciones son fácilmente accesibles y movibles a nuestro programa, e incluyen una explicación detallada de los argumentos que utiliza cada una de ellas.

Por lo demás, todo es similar a C. Los programas están divididos en tareas (*tasks)*, habiendo siempre una principal. Se usan subrutinas, llamadas a funciones, tipos de variables similares a las de C, y las estructuras de control son las mismas. Incluso la sintaxis es prácticamente igual, aunque un pequeño inconveniente es que no admite caracteres que no sean anglosajones, como la letra *ñ* o caracteres con tilde. Otro problema encontrado es la ausencia de manuales oficiales o descripciones más profundas para exprimir todas las características del lenguaje, limitándose únicamente el fabricante a proporcionarnos la lista de funciones con

sus usos, algunos programas de ejemplo, y un escueto documento para la iniciación en las aulas con el robot. A pesar de todo esto, que puede causar problemas por falta de información para aplicaciones muy complejas, contamos en general con material suficiente para trabajar con la aplicación.

Por lo dicho, estamos ante un lenguaje de programación bastante recomendable para aquel que quiera exprimir un poco más las capacidades del NXT, con la ventaja de su simplicidad, y sobre todo, su parecido a C, lenguaje bastante usado y conocido a nivel docente.

CAPÍTULO 5

SENSORIZACIÓN

Una vez terminada la construcción de la maqueta, un único elemento quedaba por añadir al conjunto: algún dispositivo o sensor que midiese el ángulo del péndulo a cada momento, para que el sistema de control actuara de la manera más correcta. Como se verá aquí, esta elección no fue para nada trivial y se tuvieron que tener en cuenta muchos factores.

Toda las especificaciones técnicas del NXT mostradas en este capítulo han sido extraídas del libro *Extreme NXT* [2], de Michael Gasperi y Philippe Hurbain, y del documento *Hardware Developer Kit* (kit de desarrolladores de hardware), disponible abiertamente en la página web oficial de LEGO [10]. En el libro se hace una recopilación de la información necesaria para exprimir al máximo las capacidades del NXT junto con gran cantidad de aplicaciones de ejemplo. En el documento *Hardware Developer Kit,* LEGO proporciona abiertamente las características técnicas del hardware del NXT.

5.1. Las Entradas del NXT

Conviene analizar detenidamente el funcionamiento de los puertos de entrada del NXT y su manera de tratar la información proveniente del exterior, para barajar las distintas posibilidades de que dispondremos a la hora de escoger el sensor.

5.1.1. Descripción de los pines y señales de entrada

El NXT dispone de cuatro puertos de entrada para sensores, todos idénticos. Si miramos el extremo de un cable del NXT, se puede ver que está formado por seis pines o cables más pequeños de distintos colores. Cuando conectamos el cable a un puerto de entrada cualquiera, la función de cada pin dependerá del tipo de sensor al que esté conectado. La numeración y disposición de los seis pines se muestra en la siguiente Figura 5.1.

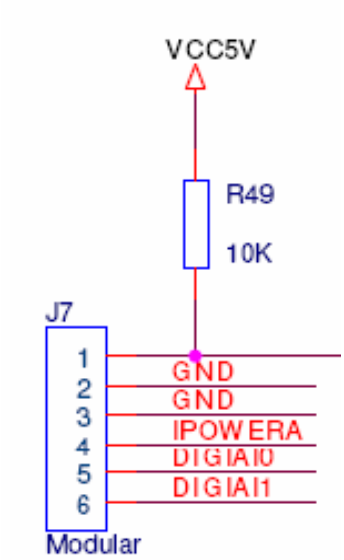


Figura 5.1. Esquemático de los pines de uno de los puertos de entrada

Es importante saber que los sensores y motores del RCX también se pueden conectar al NXT. Todos los sensores del RCX usan únicamente los pines 1 y 2 (AN y GND). Para realizar la conexión, el NXT cuenta con cables adaptadores, que no transforman la señal en modo alguno, sino que simplemente conectan el sensor del RCX a los dos pines correspondientes.

Figura 5.2. Imagen del cable adaptador para usar sensores del RCX con el NXT.

La descripción de cada uno de los pines se muestra a continuación:

- **Pin 1(Blanco): AN**

Este pin puede tener dos usos: como entrada analógica, o como fuente de energía para algunos sensores del antiguo RCX.

Si se usa este pin como entrada analógica, que será lo más común, la señal es conectada a un convertidor analógico-digital de 10 bits, incluido dentro del procesador del NXT. Esto se explicará con mayor detenimiento en el siguiente apartado.

Si, en cambio, se usa con algunos sensores del RCX (también llamados sensores activos), este pin suministra una tensión teórica de 9V, correspondiente al de las pilas (7.2V en el caso de estar funcionando con la batería de NiMH). Para este tipo de sensores, el NXT da tensión durante 3ms y lee la entrada durante 0.1ms, repitiendo el ciclo indefinidamente. La frecuencia de estas lecturas es de 333Hz (ver Figura 5.3), y aporta una corriente aproximada de 18mA.

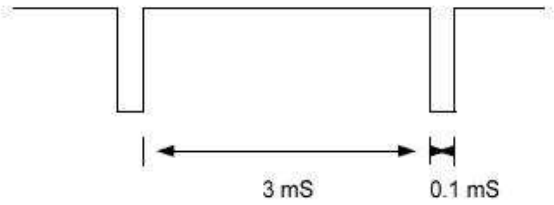


Figura 5.3. Gráfica donde se distinguen los periodos de suministro de tensión y de lectura, para sensores activos.

- **Pines 2 y 3 (Negro y rojo): GND**

Son los pines de tierra, que están conectados el uno con el otro dentro del NXT y dentro de los sensores, por lo que realmente es indistinto usar sólo un pin o ambos. Todas las señales son medidas tomando estos pines de masa como referencia.

- **Pin 4 (Verde): IPOWERA**

Proporciona la corriente necesaria a todos los sensores del NXT, y a los encoders de los motores. Está conectado internamente a los siete puertos de

entrada y salida del brick y tiene un límite de corriente de 180mA. Eso significa que cada puerto dispone de aproximadamente de unos 25mA, aunque se puede consumir más si otro consume menos. Si la carga total supera el límite máximo, la tensión de 4.3V de que dispone cae rápidamente.

- **Pines 5 y 6 (Amarillo y azul): DIGIAI0 y DIGIAI1**

Son pines de entrada/salida usados para el protocolo de comunicación digital I^2C, tal y como la que usa el sensor de ultrasonidos. Más información se puede encontrar en el libro *Extreme NXT.* No serán necesarios para la elección de nuestro sensor.

5.1.2. El convertidor A/D del NXT

Como se ha dicho, una de las especificaciones técnicas del NXT, indica que dispone internamente de un convertidor analógico-digital entre los pines 1 y 2 de los puertos de entrada. El esquemático correspondiente se puede ver en la Figura 5.4.

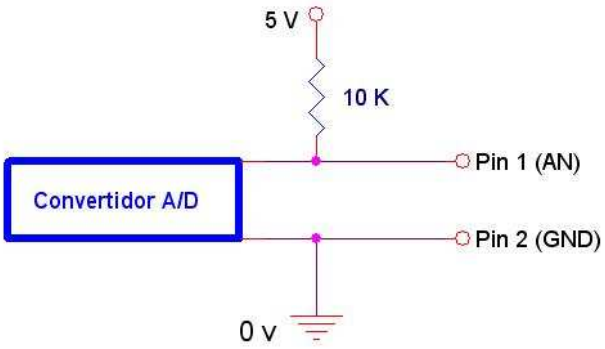


Figura 5.4. Esquemático del convertidor, conectado a los pines 1 y 2.

Dentro del NXT, una resistencia de 10kΩ (resistencia de *pull-up)* está permanentemente conectada entre el pin 1 y una tensión de 5V. El pin 2, como se dijo antes, corresponde a la toma de tierra (0V). Ambos pines están conectados a un convertidor A/D que, al ser de 10 bits, convertirá la tensión de entrada en un valor digital entre 0 y 1023 (lo que llamaremos el valor *raw* ó *en bruto*).

Por lo tanto, el valor raw variará entre 0 para una tensión de 0V entre los dos pines (circuito cerrado), y 1023 para una tensión de 5V (circuito abierto). Entonces la ecuación que nos define el Raw en función de la tensión de entrada será:

$$Raw = \frac{1023}{5} \cdot V$$

Si conectamos una resistencia externa entre los dos terminales, tal y como ocurriría al utilizar cualquier sensor resistivo, variaremos la tensión de entrada. La resistencia conectada, junto con la de pull-up de 10kΩ, actuarían como un simple divisor de tensión, siendo el valor máximo los 5V que proporciona el NXT. Por tanto la ecuación que describe la tensión dependiendo de la resistencia externa que coloquemos, es la siguiente:

$$V = \frac{R}{10000 + R} \cdot 5$$

Sustituyendo esta última ecuación en la del Raw, podemos deducir que el valor Raw en función de la resistencia externa variará según la relación

$$Raw = \frac{1023}{\frac{10000}{R} + 1},$$

donde R es la resistencia externa que coloquemos.

Se puede apreciar que no hay una relación lineal entre la resistencia y el valor Raw que nos da el convertidor, lo que afectará posteriormente a la elección del sensor.

Concluimos que podemos medir cualquier variable física que implique el variar una resistencia. Esto nos abre las puertas para usar cualquier tipo de sensor resistivo típico en electrónica con el NXT, como de temperatura (termistores) o de luz (LDR), pero sobre todo, nos hace pensar en la posibilidad de usar un potenciómetro rotativo para medir el ángulo que oscila nuestro péndulo.

5.2. Elección del sensor de rotación

Quizá podríamos decir que esta fue la parte del proyecto más decisiva. Esto no es de extrañar teniendo en cuenta que de esta elección dependería no sólo el montaje en el péndulo, sino el tratamiento de la señal y la posterior programación del controlador.

5.2.1. Alternativas

Finalmente nos decidimos por usar un potenciómetro rotativo comercial para obtener una señal que indicara la oscilación del péndulo. Sin embargo, hubo otras opciones anteriormente que merecen ser comentadas:

- Usar el sensor de rotación del RCX:

  Al igual que en el modelo de Euan y Peter, se pensó en utilizar este sensor. Aunque su fricción casi nula habría sido ideal para nuestro trabajo, su baja resolución (22.5^0) era insuficiente para nuestra aplicación. Sin embargo, se habría podido aumentar esta resolución de forma externa con un pequeño tren de engranajes, pero esta solución seguro que habría influido negativamente en la fricción del péndulo. Este problema, junto con el alto precio del sensor, hicieron desechar esta idea.

- Usar el giróscopo de Hitechnic:

  Hitechnic dispone del único sensor de giro expresamente fabricado para el NXT. Nos devuelve el número de grados rotados por segundo, a la vez que indica la dirección de esa rotación, pudiendo llegar a una frecuencia de 300 lecturas por segundo, y funcionando de 0^0 a 360^0. Todas las características se encuentran en la página web del fabricante [11]. Al tratarse de un sensor incremental, podían surgir problemas a la hora de utilizarlo en el proyecto, como la acumulación de errores o dificultades para convertir su salida a ángulo absoluto. Finalmente, el contar con gran variedad de potenciómetros distintos a nuestro alcance, nos hizo inclinarnos por esta última opción.

La decisión final fue adquirir un potenciómetro rotativo para usar con el NXT. Las premisas eran que con un poco de esfuerzo se conseguiría el mismo resultado que con el giróscopo de HiTechnic, con la diferencia de los pocos euros que puede costar uno de estos potenciómetros. Además, la aplicación final sería un pequeño ejemplo de “homebrew” del NXT, demostrando lo que se puede llegar a hacer sin usar hardware oficial.

Para aquel que quiera profundizar más en modificaciones caseras del hardware del NXT, es muy recomendable el libro *Extreme NXT* [2], en el que hay varios capítulos dedicados al tema.

5.2.2. Funcionamiento de un potenciómetro rotativo en el NXT.

Una página web donde se comenta el uso de potenciómetros como sensor de rotación, es la de Laurent Kneip [13], estudiante en Luxemburgo que ha desarrollado algunas interesantes aplicaciones en el ámbito de la robótica usando el RCX, como algunos brazos robóticos con varios grados de libertad.

Un potenciómetro se puede considerar como un sensor de tipo pasivo para el NXT que debe ir conectado a los pines 1 y 2 de los puertos de entrada, tal y como se observa en la Figura 5.5.

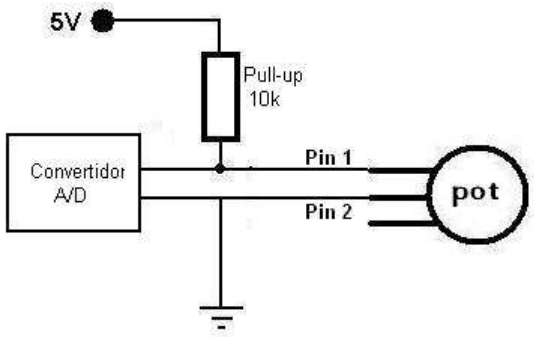


Figura 5.5. Esquemático de la conexión del potenciómetro al NXT

Lo normal y lo más adecuado es que la resistencia del potenciómetro rotativo varíe linealmente con el ángulo girado. Podemos escribir la ecuación para la resistencia instantánea donde A es el ángulo girado en grados, A_T es el ángulo máximo del potenciómetro y R_T es su resistencia máxima.

$$R = \frac{A}{A_T} \cdot R_T$$

Conectado a los pines AN y GND como se ha dicho, funcionará como un divisor de tensión (ver apartado 5.1.2.), con resistencia variable. Si sustituimos la fórmula anterior en la que ya hemos visto para los valores Raw (apartado 5.1.2.), obtenemos la ecuación más importante: la que nos define el valor Raw que obtendremos para cualquier ángulo:

$$Raw = \frac{1023}{\left(\frac{A_T \cdot 10000}{A \cdot R_T}\right) + 1}$$

donde A_T y R_T son las características del potenciómetro que escojamos.

Cómo era de suponer por las ecuaciones del apartado anterior, la relación entre el ángulo girado y el valor Raw no es lineal. Esto se ve claramente en la Figura 5.6, donde se representa el valor Raw de salida en función del ángulo girado en grados, para distintos valores de potenciómetro y suponiendo en todos un ángulo total (A_T) de 270^0.

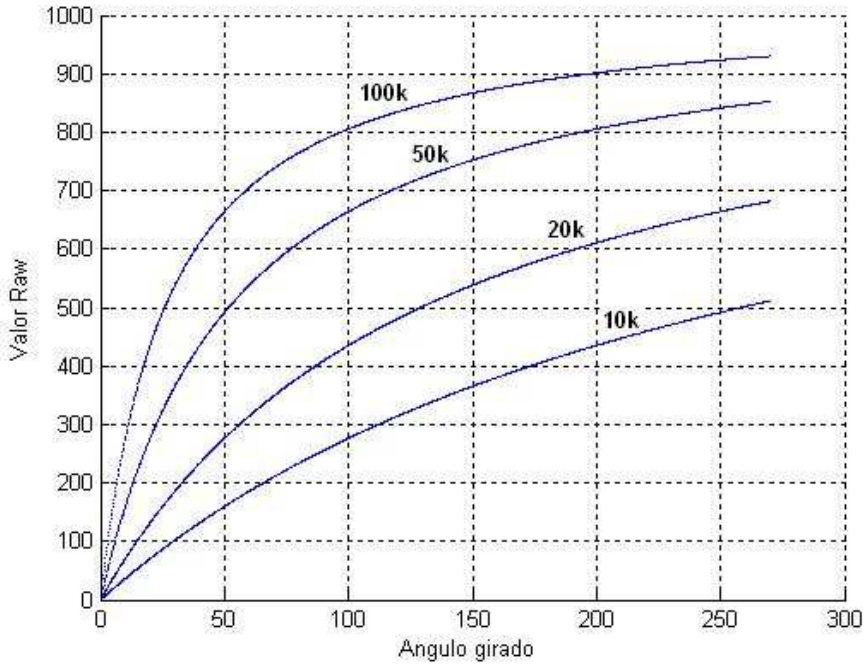


Figura 5.6. Gráfica del valor Raw en función del ángulo para distintos potenciómetros

Debido a que no hay una relación lineal entre Raw y ángulo, la sensibilidad del sensor depende del ángulo girado, como se ve en el gráfico anterior. La precisión del sensor será mayor en ángulos pequeños (alta pendiente), e irá disminuyendo conforme aumentemos el ángulo. Esta poca linealidad se puede solucionar seleccionando una pareja de valores de resistencia de pull-up y de potenciómetro. Sin embargo, la resistencia de pull-up es difícil cambiarla debido a que está fija en 10kΩ en el interior del NXT, de modo que la única opción que nos queda es elegir el potenciómetro. Para hacernos una idea, los valores más adecuados de potenciómetro para conseguir una salida más lineal son los que se acercan al valor de la resistencia de pull-up, como por ejemplo 10kΩ (Figura 5.6.).

El problema de mejorar la linealidad es que, como se ve, conlleva perder resolución o sensibilidad. Esto se podría solucionar para obtener un resultado aún mejor, elevando la tensión de referencia de 5V en el interior del NXT, o cambiando la resistencia de pull-up. Por su complejidad, esto se encuentra fuera de los objetivos de este proyecto, aunque una explicación algo más detallada de estos métodos se puede encontrar en [13].

El uso que le vayamos a dar al potenciómetro también influirá en la elección del mismo. Si por ejemplo vamos a trabajar con ángulos pequeños (como podría pasar en nuestra maqueta) tal vez vendría bien escoger un potenciómetro grande (100kΩ). Como se puede ver en la Figura 5.6, éste nos da una salida que podemos considerar casi lineal entre 0 y 50 grados de giro . Si por el contrario vamos a usar todo el rango de giro del potenciómetro, seguramente este potenciómetro sería inadecuado, ya que no se mantiene lineal, y deberíamos buscar uno más pequeño a costa de perder algo de sensibilidad.

5.3. Pruebas a los potenciómetros y elección final

Dado que se suponía que el péndulo no trabajaría con rotaciones muy grandes (ya que oscilando libremente no suele superar los 30 o 40 grados de giro hacia cada lado), la primera opción que surgió fue utilizar un potenciómetro de resistencia elevada, y trabajar con él en la región lineal de funcionamiento. Se hicieron las pruebas necesarias, y se encontraron algunos problemas, con lo que no hubo más remedio que cambiar a un potenciómetro menor y con salida más lineal. Éste sería finalmente el que se utilizaría en el proyecto.

El proceso seguido y las características más en detalle de esta decisión se describen a continuación.

5.3.1. Potenciómetro de 220kΩ

En la Figura 5.7 se muestra el potenciómetro con el que en un principio se iba a montar la maqueta. Era un potenciómetro rotativo habitual de 220kΩ nominales, aunque se confirmó con el multímetro que realmente eran 230kΩ, y así se consideró en los cálculos. El ángulo máximo de giro era de 270 grados. Un aspecto a favor era su bajo precio, de alrededor de 3 euros.

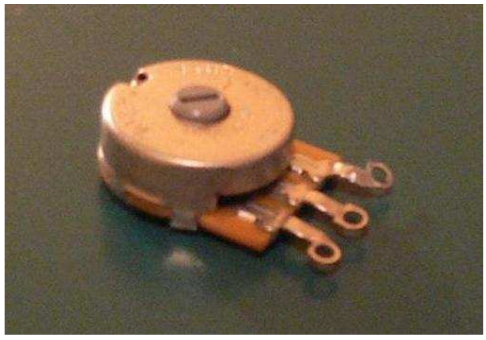


Figura 5.7. Imagen del potenciómetro inicial de 230kΩ

El mayor problema que mostraba a simple vista o, más correctamente, nada más tocarlo, era que tenía demasiada fricción. Obviamente esto era un gran problema, pues influiría negativamente al frenar él mismo el péndulo cuando estuviera conectado a éste. Afortunadamente, era fácilmente desmontable, por lo que lo primero que se hizo fue proceder a hacerlo para intentar disminuir de alguna forma su rozamiento interno (algo similar a lo mostrado en la web de Laurent Kneip [13]).

Figura 5.8. Imagen del potenciómetro desmontado. Las pletinas y la pista resistiva aparecen unidas.

Usando un destornillador se separaron todas las piezas que lo formaban (Figura 5.8), y se descubrió que prácticamente toda la fricción la proporcionaba una sustancia pastosa en la zona de unión donde se insertaba el eje del potenciómetro. Tras limpiarlo debidamente con alcohol y volverlo a montar, la fricción ya era considerablemente menor, pero aun así seguía siendo un problema. Se decidió volverlo a desmontar para intentar disminuir el poco rozamiento que quedaba: el de las propias pletinas metálicas al moverse por la pista resistiva. Para conseguirlo se separaron unos milímetros las dos partes, lo que disminuiría la presión que ejercían esas pletinas sobre la pista. Tras volverlo a montar, la fricción era aún menor, pero sin embargo quedaba alguna fricción residual e inevitable, pues es el propio funcionamiento del potenciómetro requiere del roce entre estos dos elementos.

En cuanto a la curva de funcionamiento, se creó un código para almacenar los valores Raw en el NXT al ir girando el potenciómetro. Los resultados se compararían con los obtenidos teóricamente y nos darían una idea del rango de valores en el que deberíamos trabajar con él. A priori sabíamos que al tratarse de un potenciómetro de valor tan alto, sería lineal en una región muy pequeña, pero que tal vez sería posible usar en nuestra aplicación. El potenciómetro se conectó al NXT mediante el cable adaptador para sensores del RCX.

El código utilizado es “GuardaRawPulsando.c”, que se puede encontrar en el Anexo A de este proyecto. Se recogió el valor Raw cada 30 grados de rotación, desde 0 hasta el máximo de 270 grados. Si graficamos los puntos resultantes, obtenemos la curva de la Figura 5.9.

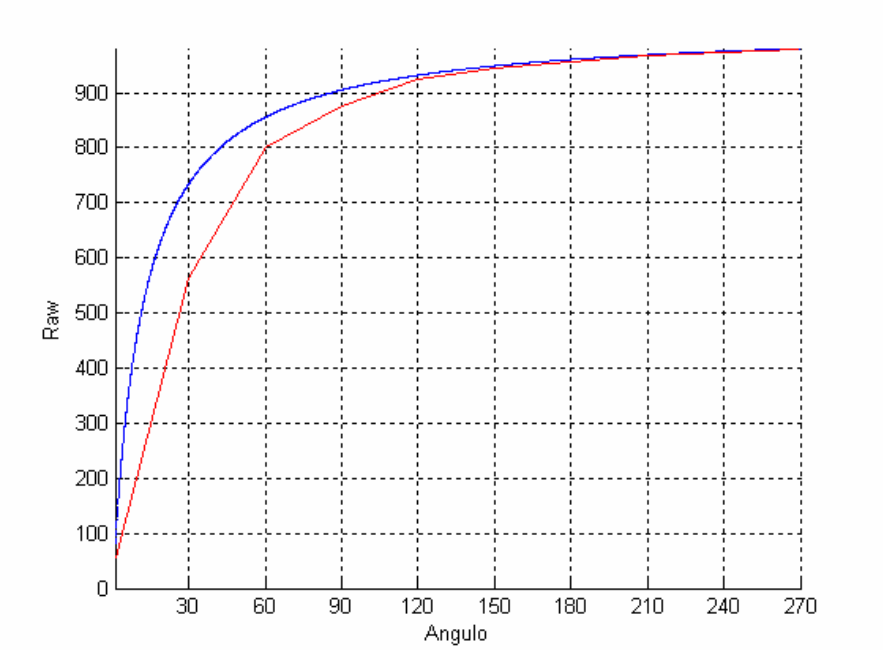


Figura 5.9. Comparación de la gráfica teórica (en azul) y los datos obtenidos experimentalmente (en rojo) para el potenciómetro de 230kΩ

Como se ve, los datos obtenidos experimentalmente coinciden bastante con los esperados. Sin embargo se aprecia una diferencia en ángulos pequeños, como por ejemplo para 30 grados, dónde experimentalmente obtenemos un Raw aproximado de 560 cuando en teoría deberíamos obtener más de 700. Esto se puede explicar con la extrema sensibilidad del potenciómetro en esta región, ya que en la práctica se hace difícil conseguir un ángulo exacto, y la más mínima imperfección hace variar ampliamente la salida obtenida.

Por lo demás, parece que podríamos considerar lineal la gráfica hasta aproximadamente unos 45 o 60 grados, que sería la zona de trabajo adecuada. En un principio se podría pensar que es un potenciómetro asequible para utilizar en nuestro péndulo, siempre y cuando el balanceo no supere esa magnitud de giro. Sin embargo, parecía difícil que el péndulo no oscilara alguna vez más de 30 grados a cada lado, y no podíamos limitarnos sólo a ángulos tan pequeños.

Éste problema, junto con la pequeña fricción que aún mantenía el potenciómetro, hicieron que fuera preferible buscar otra alternativa. Así pues, lo que se hizo fue obtener otro potenciómetro con las características buscadas.

5.3.2. Potenciómetro de 10kΩ

Finalmente se decidió conseguir un potenciómetro de más calidad. Concretamente se adquirió el modelo de alta precisión y de baja fricción *radiospares #173-580,* de 10kΩ nominales (ver figura 5.10), y cuyo precio es unos 50 euros. En efecto, se pudo comprobar fácilmente que la fricción era prácticamente nula, con lo que nos ahorrábamos un gran problema. Aunque la tolerancia indicada era del 20%, se comprobó con el multímetro que la resistencia máxima se mantenía en unos 10.7kΩ. Otra de las bondades es que el fabricante aseguraba un error de linealidad menor del 0.5%, valor realmente bajo. Aunque era de tipo multivuelta, el cambio de resistencia de 0 a 10.7kΩ abarcaba unos 345 grados en total, sobrando unos 15 grados de ángulo muerto.

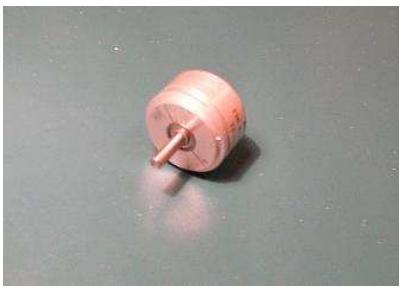


Figura 5.10. Imagen del nuevo potenciómetro de 10.7kΩ

Ya teníamos suficientes datos para calcular teóricamente la curva de funcionamiento aplicando la fórmula ya conocida. Además se hizo también de nuevo la prueba experimental con el mismo código en el NXT, obteniendo el Raw a intervalos de 30 grados pero esta vez llegando hasta 330 grados. En la Figura 5.11 se hace la comparación con los datos teóricos.

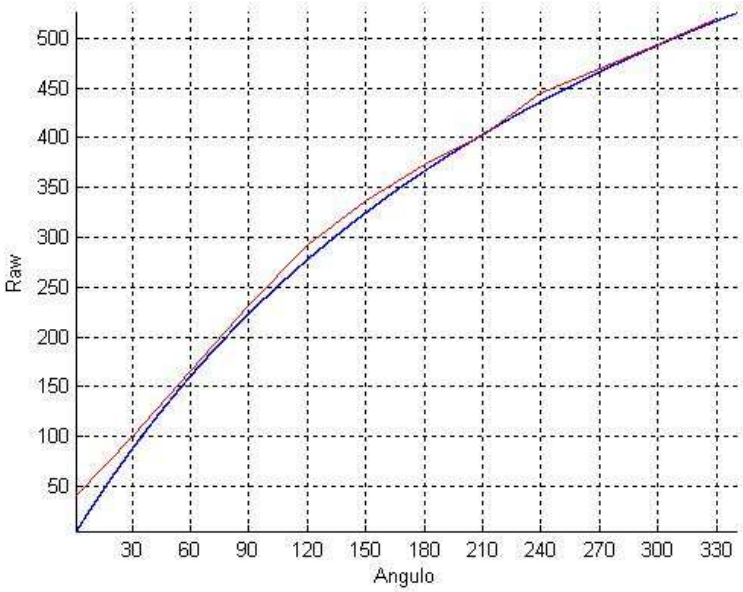


Figura 5.11. Comparación de la gráfica teórica (en azul) y los datos obtenidos experimentalmente (en rojo) para el potenciómetro de 10.7kΩ

En este caso se obtuvieron en la práctica unos resultados casi idénticos a los que predecía la teoría. Las dos curvas se superponen y se confirma la calidad del sensor. Se puede ver que con este potenciómetro el máximo Raw que se alcanza es de 520 aproximadamente, pero a cambio podemos considerar la salida en función del ángulo casi lineal en todo el rango de valores, lo que nos deja una zona de trabajo amplísima.

Este motivo, sumado al hecho de que el potenciómetro no presenta prácticamente fricción alguna, llevó a la decisión de darle uso en el proyecto. Ya sólo quedaba pensar cómo acoplarlo al péndulo, decidir cuál sería el punto de reposo o punto neutro, y proceder a su montaje.

5.4. Montaje del potenciómetro

La conexión del potenciómetro al eje del péndulo no fue una tarea fácil. La forma y superficie casi lisa de la carcasa no dejaba demasiadas opciones para unirlo a piezas de LEGO. Tras varios intentos fallidos para conseguirlo, se encontró una manera bastante aceptable.

Dio la casualidad de que el diámetro del potenciómetro encajaba casi perfectamente en una pieza poco común de transmisión de LEGO (Figura 5.13). Apretando un poco, se conseguía encajar e inmovilizar el potenciómetro en el agujero, sin necesidad de adhesivos. Al eje del potenciómetro ya encajado, se le insertó un pequeño bloque formado por cuatro piezas, que más tarde serviría para acoplarlo al eje del péndulo. Estas piezas se pueden ver en la figura 5.12.

Figura 5.12. Imagen de las cuatro piezas de apoyo para el potenciómetro

Figura 5.13. Imagen del potenciómetro encajado en la pieza circular y con el pequeño bloque enganchado a su eje

Sólo restaba acoplar este conjunto a la estructura. El eje del potenciómetro se unió al del péndulo mediante las piezas antes comentadas, y la pieza que albergaba al potenciómetro se unió al lateral de la estructura utilizando otras piezas de la caja. Con esto se consiguió finalmente independizar el movimiento del sensor y su eje. Además, el montaje se realizó de tal modo que la posición de reposo del péndulo mantuviera al potenciómetro aproximadamente a la mitad de su recorrido, cerca de los 4 o 5kΩ ($Raw \approx 300$). Esto se hizo para tener el mismo rango de trabajo a cada lado y mejorar un poco la sensibilidad del sensor. El resultado final se expone en la Figura 5.14.

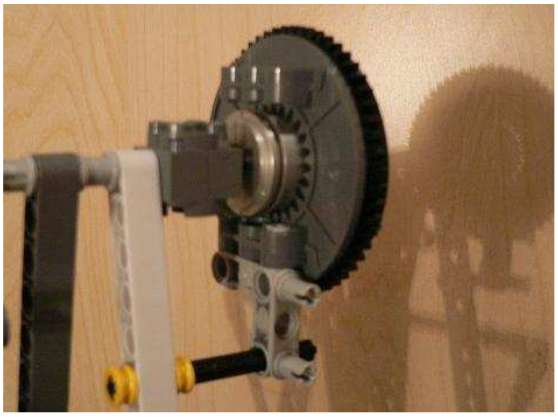


Figura 5.14. Imagen del potenciómetro montado en la maqueta

Por último, se utilizó un cable adaptador de sensores del RCX al NXT, ideal para conectar directamente nuestro potenciómetro a los pines 1 y 2 del puerto de entrada. Sin embargo, debido a la dificultad que suponía conectar el potenciómetro al extremo del cable (construido con la forma de un bloque de LEGO y casi sin partes metálicas), se cortó este extremo y los dos hilos resultantes se pegaron directamente a los pines del sensor. Hacer esto no supuso una gran pérdida, pues la caja incluye varios cables más y además no se preveía darles mucho uso. El resultado se ve en la Figura 5.15.

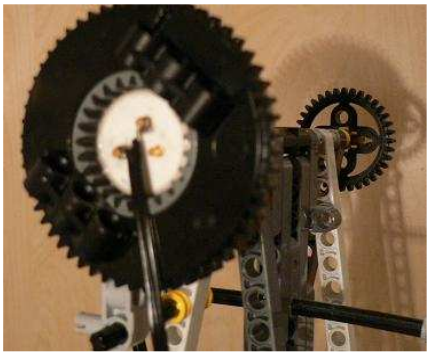


Figura 5.15. Imagen trasera del potenciómetro con el cable atado

En la guía de montaje adjuntada en el CD del proyecto, se incluyen las imágenes aclaratorias necesarias para realizar este montaje (ver Anexo B). Con el acople del potenciómetro a la estructura, se concluyó la tarea de construcción. En la Figura 5.16 se aprecia el aspecto definitivo de la maqueta, con el potenciómetro montado.

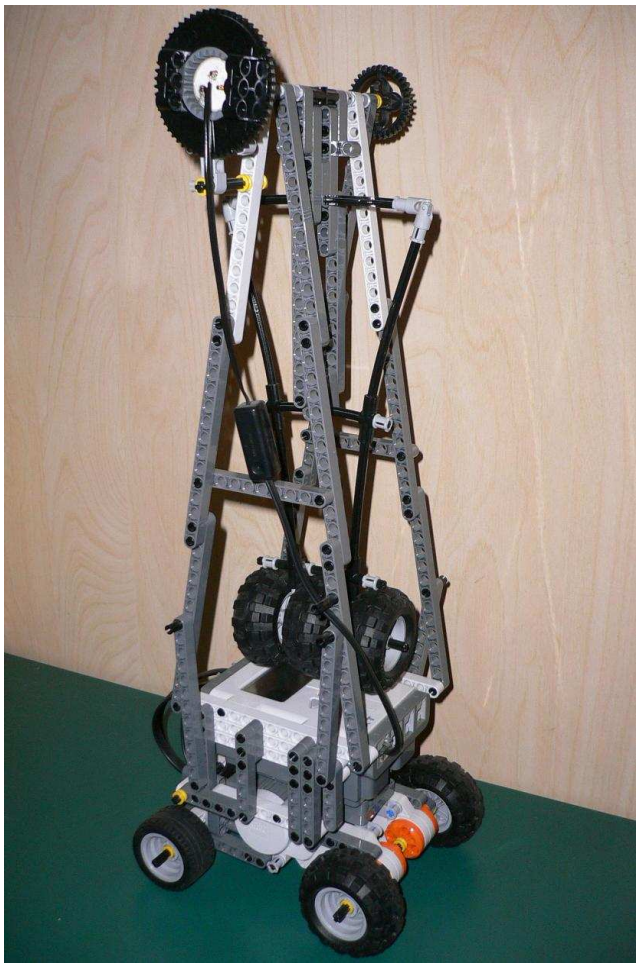


Figura 5.16. Imagen del aspecto final de la maqueta

CAPÍTULO 6

IDENTIFICACIÓN DEL SISTEMA

Uno de los pasos en toda implementación de un sistema de control, es la realización de pruebas u obtención de los datos que representen lo mejor posible la dinámica del sistema. Así se podrá actuar en consecuencia a la hora de definir adecuadamente el controlador o cualquier otro elemento para modificar el comportamiento del proceso.

Este sistema no fue una excepción. En principio se tenían dos caminos a seguir para obtener una idea de este comportamiento. Modelar matemáticamente este comportamiento físico, o identificarlo experimentalmente. Para el primero de los casos, se hace necesario un mayor conocimiento de la física del sistema con el que estamos actuando y un mayor manejo de la matemática que lo rodea. Aunque tratándose de un péndulo esta tarea podría no ser muy complicada, se consideró más factible identificar experimentalmente el sistema debido a su mayor sencillez, pero manteniendo la capacidad de reflejar adecuadamente los aspectos que nos interesan. Así no habría necesidad de tener en cuenta todas las leyes físicas que rigen su comportamiento.

La identificación se basa únicamente en datos experimentales. De ese modo, para llevarla a cabo se deben hacer una serie de pruebas que arrojen datos suficientes para definir el sistema con claridad. Se aplicarán distintas entradas a nuestro sistema, y se recogerán las salidas correspondientes (más adelante se explicarán cuales son estas entradas y salidas). A partir de ahí, se elije el modelo teórico que más se adecue a este comportamiento. Aunque no requiere tanto conocimiento a priori de las variables físicas que están interviniendo en nuestro

sistema, sí que se requiere saber qué se está haciendo y tener una idea de lo que se quiere conseguir con cada experimento.

Concretamente, para la maqueta que nos ocupa se diseñaron varias pruebas que nos aproximaran a las relaciones entre distintas variables. Por ejemplo, podríamos considerar el sistema cuya entrada sería tensión a los motores y cuya salida podría ser la oscilación del péndulo. Tras una serie de pruebas y recogida de esas parejas de valores entrada-salida, se puede determinar con precisión la dinámica del sistema.

La descripción más exhaustiva de todas las pruebas realizadas se expone en los siguientes apartados. Se recalca que todos los experimentos siguientes están destinados a conocer el comportamiento del sistema **en lazo abierto.**

6.1. Péndulo libre

La primera tarea que se planteó fue analizar el balanceo del péndulo de forma independiente, oscilando libremente.

6.1.1. Experimento y obtención de datos

Si utilizamos la ecuación del apartado 5.2.2, que usábamos para obtener el valor Raw a partir del ángulo, esta vez se puede hacer lo contrario y despejar el ángulo girado para obtener éste en función del valor Raw que indique el NXT. Para el potenciómetro utilizado en particular, después de simplificar queda lo siguiente

$$A = \frac{34500}{\frac{109461}{Raw} - 107}$$

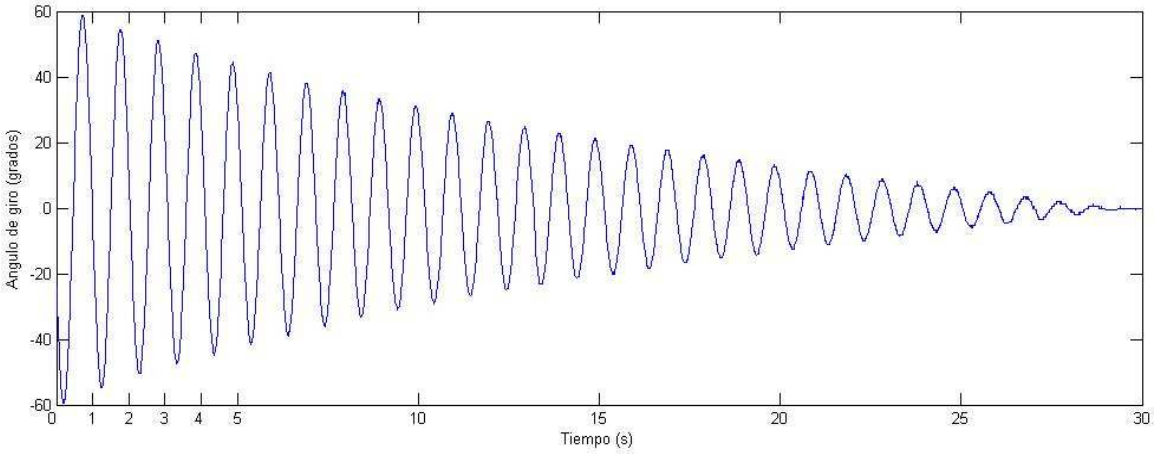
donde *A* será el ángulo actual de giro en grados del potenciómetro.

A continuación se creó un código en RobotC que, utilizando la ecuación anterior, almacenara el ángulo de reposo o de equilibrio inicial del potenciómetro, y al balancear el péndulo calculara el error respecto a ese ángulo inicial. Este error se guardaría en un archivo. Se tomó un tiempo de muestreo de 10 ms, desde el ángulo de comienzo del balanceo que se eligió de aproximadamente 60 grados, hasta la llegada al estado estacionario. El código usado es *"PruebaOscilacion.c"*, descrito en el Anexo A.

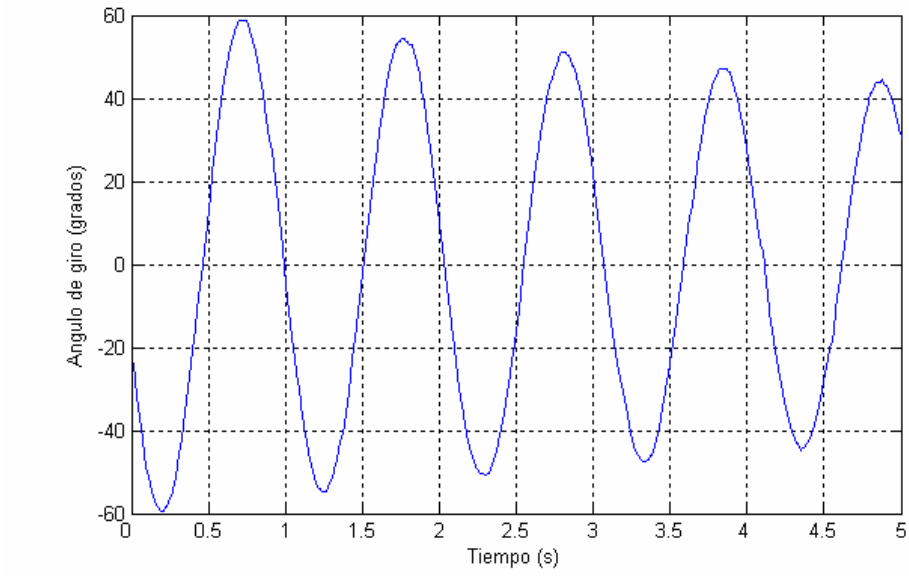
Nota:

El punto de equilibrio del péndulo corresponde a un valor Raw aproximado de 327, o lo que es lo mismo, un ángulo de 151.486 grados, del total de 340 grados de giro. La disposición del potenciómetro de esta manera no fue aleatoria, sino que se eligió así para trabajar aproximadamente en la zona media del recorrido de giro (ver Apartado 5.3.2.). De esta forma se tendrían zonas de trabajo de tamaños aproximadamente simétricos a cada lado.

Si graficamos todos los puntos resultantes del experimento anterior, obtenemos las curvas representadas en la Figura 6.1.



(a)



(b)

Figura 6.1. Ángulo de oscilación del péndulo en función del tiempo, desde 60 grados hasta el estacionamiento (a), y los cinco primeros segundos (b).

La primera información que salta a la vista es la bajísima amortiguación que presenta el movimiento del péndulo. Esto es debido a la fricción casi despreciable que implica que el péndulo tarde unos 30 segundos en volver a su estado de reposo. También se puede ver que el periodo de oscilación es algo mayor de 1 segundo. La ecuación física que nos define el periodo de oscilación de un péndulo cualquier es:

$$T = 2\pi\sqrt{\frac{l}{g}}$$

donde l es la longitud del péndulo en metros y g es la fuerza que ejerce la gravedad.

Para este caso concreto, utilizando los parámetros del péndulo:

$$T = 2\pi\sqrt{\frac{0.285}{9.81}} \approx 1.07s \quad ,$$

que coincide bastante con el dato que nos muestra la gráfica. Se concluye que la frecuencia natural del péndulo es aproximadamente de 1Hz, o lo que es lo mismo, 2π rad/s (6.28318 rad/s).

6.1.2. Obtención de la función de transferencia y validación

Podemos utilizar los datos conseguidos experimentalmente para obtener un modelo que nos relacione el ángulo de salida (la oscilación), con el ángulo de inicio del balanceo. Éste ángulo inicial está asociado a una fuerza inicial, por lo que la función de transferencia a obtener será

$$\frac{\theta(s)}{F(s)} = G(s)$$

donde *F(s)* será la fuerza de entrada aplicada al péndulo y $\theta(s)$ será el ángulo de salida. Si estamos sosteniendo el péndulo en un ángulo determinado, estamos ejerciendo una fuerza sobre él para mantenerlo en equilibrio en esa posición. Si a continuación lo soltamos (esa fuerza cae a cero), el péndulo comenzará a oscilar. Entonces podremos considerar la entrada en ese momento como un escalón de valor inicial igual a la fuerza necesaria para sostener el péndulo, y de valor final cero. La forma de hallar esta fuerza se explica un poco más adelante.

Si consideramos el sistema como un sistema de segundo orden, sabemos que su función de transferencia deberá ser de la forma:

$$G(s) = \frac{k \cdot \omega_n^{\,2}}{s^2 + 2\xi\omega_n s + \omega_n^{\,2}}$$

Entonces habrá que hallar el valor de la ganancia k, del coeficiente de amortiguamiento ξ, y de la frecuencia ω_n, partiendo de los datos experimentales. Pare ello se repitió la prueba de oscilación desde distintos ángulos, tanto positivos como negativos (60, 45, 20 y 10 grados), y se obtuvieron de las gráficas los datos necesarios.

Para cada uno de los experimentos, los valores se obtuvieron de la siguiente manera:

La *frecuencia natural* no amortiguada se calcula como

$$\omega_n = \frac{1}{T} \cdot 2\pi \quad ,$$

es decir, la inversa del periodo multiplicado por 2π para pasarlo a radianes por segundo.

El *coeficiente de amortiguamiento*, adimensional, se obtiene despejándolo de la ecuación

$$M_p = e^{\frac{-\pi\xi}{\sqrt{1-\xi^2}}} \quad ,$$

donde M_p es, en valor porcentual, el máximo sobrepico del sistema.

Por último, la *ganancia estática* k para cada experimento será la relación entre la salida y la entrada:

$$k = \frac{\theta}{F}$$

La fuerza que produce el movimiento del péndulo en un instante determinado, y que será por tanto la fuerza de entrada que se busca, se representa en el diagrama de la Figura 6.2.

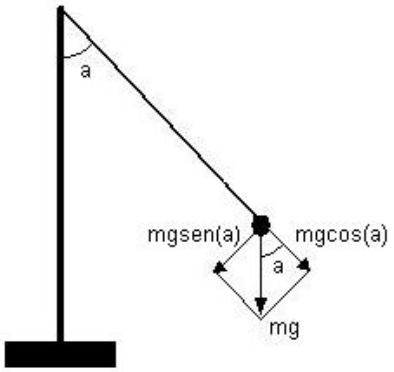


Figura 6.2. Diagrama de fuerzas que actúan en un péndulo

La fuerza que estamos buscando es:

$$F = m \cdot g \cdot sen(\theta)$$

donde m es la masa del péndulo, g es la aceleración debido a la gravedad, y θ es el ángulo del péndulo. Debido a que esta relación no es lineal (al incluir un término seno), tampoco lo será la relación ángulo-fuerza, por lo que la ganancia estática k variará de un experimento a otro en función del ángulo inicial. Por consiguiente también variará la función de transferencia. Por esta razón se determinaron el valor de k y de la función de transferencia de la manera mencionada para los distintos ángulos antes comentados (10, 20, 45 y 60), y se intentó encontrar alguna que se aproximara aceptablemente al comportamiento real del sistema.

La siguiente tabla muestra los valores encontrados para cada uno de los experimentos:

Experimento (ángulo inicial)	**K**	**W_n**	ξ
10 grados	66.632307	6.283185 rad/s	0.0183661
20 grados	67.660218	6.283185 rad/s	0.0169312
45 grados	73.63479	6.220975 rad/s	0.0139200
60 grados	80.163321	6.159984 rad/s	0.0149089

Como se ve en la tabla, la frecuencia W_n coincide con el valor obtenido de forma teórica a partir de los parámetros del péndulo (Apartado 6.1.1).

Las correspondientes funciones de transferencia son:

Para 10 grados:

$$\frac{2630.538042}{s^2 + 0.230795s + 39.478417}$$

Para 45 grados:

$$\frac{2849.70585}{s^2 + 0.1731919s + 38.700536}$$

Para 20 grados:

$$\frac{2671.118335}{s^2 + 0.2127637s + 39.478417}$$

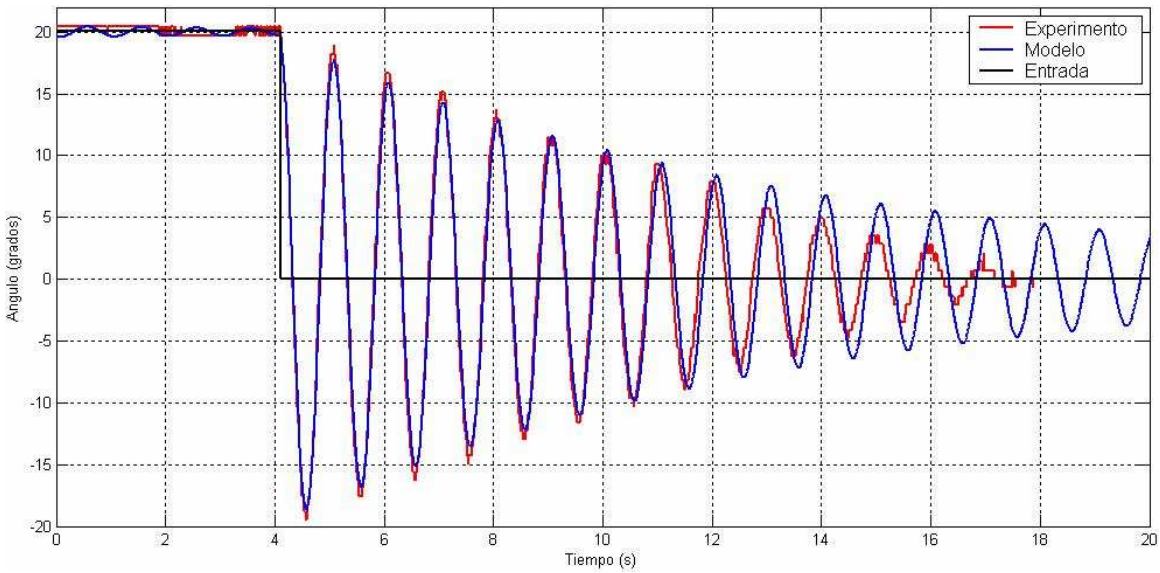
Para 60 grados:

$$\frac{3041.830511}{s^2 + 0.183677s + 37.945415}$$

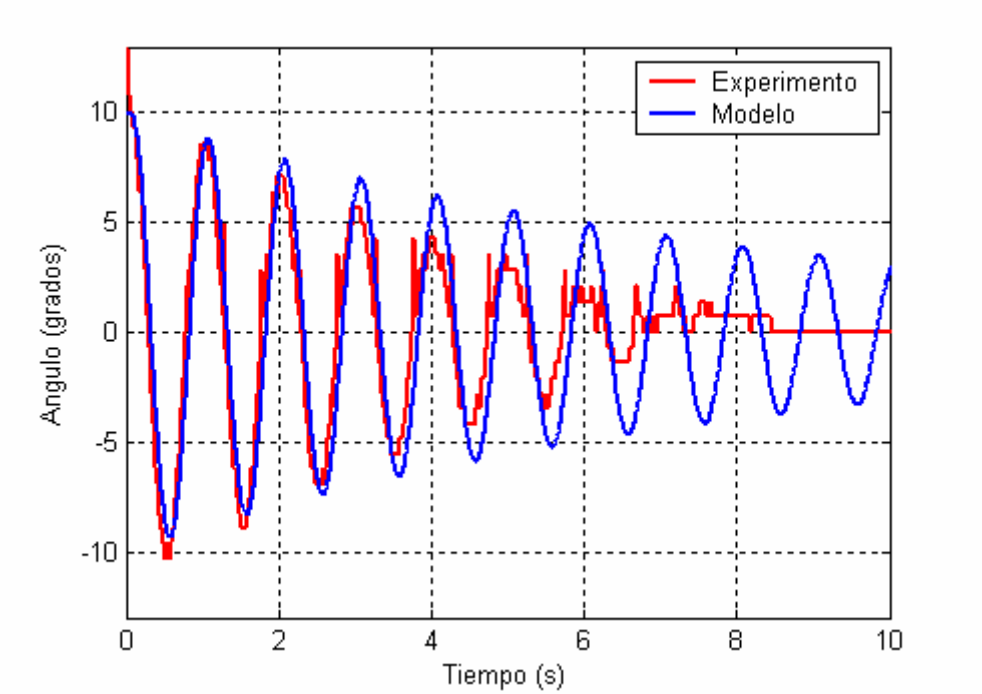
Estas funciones de transferencia se implementaron en Simulink para darle distintas entradas escalón, que como se ha dicho estarán dadas en unidades de fuerza. El diagrama de bloques está representado en la Figura 6.3.

Figura 6.3. Diagrama de bloques del sistema $\frac{\theta(s)}{F(s)}$

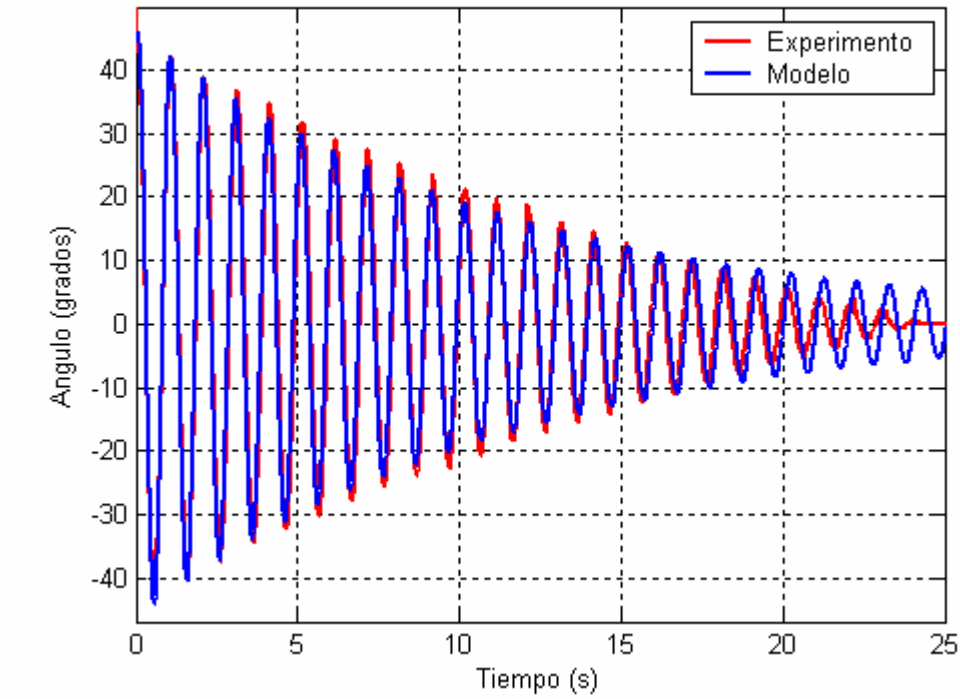
A cada función de transferencia se le aplicó su escalón correspondiente. Por ejemplo, usando la función de transferencia de 20 grados, se le aplicó una fuerza inicial de 0.295594 N (equivalente a 20 grados iniciales). Lo mismo se hizo para las otras tres funciones. Los resultados obtenidos se exponen en la Figura 6.4 (el escalón de entrada se ha convertido a ángulo para una mejor visualización).



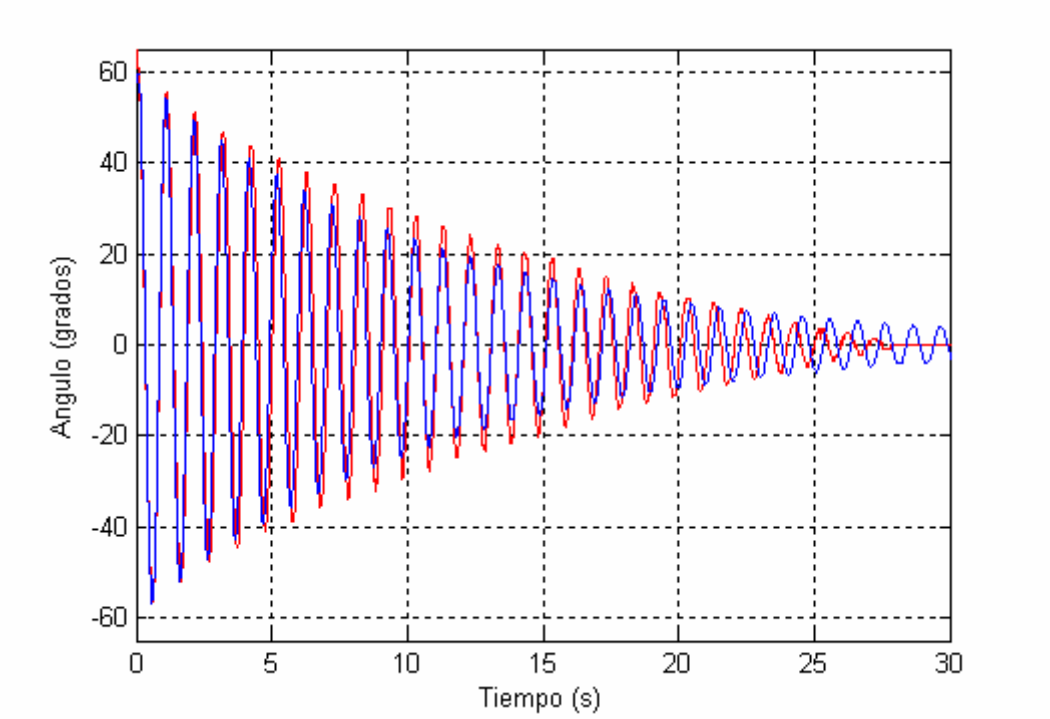
(a) Resultados para la f.d.t. de 20 grados



(b) Resultados para la f.d.t. de 10 grados



(c) Resultados para la f.d.t. de 45 grados



(d) Resultados para la f.d.t. de 60 grados

Figura 6.4. Comparación de los datos experimentales con los previstos por los modelos fuerza-ángulo.

Como se ve, es una buena aproximación. En este caso el valor previsto se superpone con los datos obtenidos experimentalmente en las cuatro pruebas. Sin embargo, cuanto más se aleje el ángulo inicial del ángulo para el que fue identificada cada función, peor será la predicción del modelo.

La única apreciación que hay que hacer a todas estas simulaciones es que a partir de cierto ángulo la predicción se aleja de los datos reales. Esto ocurre porque a partir de oscilaciones muy pequeñas (de alrededor de 5 grados hacia cada lado), la fricción real ya no puede considerarse lineal y surgen determinadas componentes que producen el paro total del péndulo, mientras que en el modelo no se contempla este comportamiento. Por su complejidad esto no se incluyó en las funciones de transferencia indicadas, que de por sí ya nos proporcionan una salida bastante acorde con la realidad. El efecto comentado se aprecia muy bien en la simulación desde 10 grados.

Se concluye que la función de transferencia que elijamos para caracterizar el sistema dependerá de la entrada que vayamos a aplicar, debido a la no linealidad de la ganancia estática.

6.2. Relación carro-péndulo

En este caso, es interesante conocer cómo se comportará el péndulo al aplicar una tensión determinada a los motores.

6.2.1. Obtención del modelo

Una vez hallada la relación entre la fuerza aplicada y el ángulo resultante, se puede ir más allá e intentar obtener la relación entre la tensión aplicada a los motores y el ángulo. Al fin y al cabo, un movimiento del coche debe producir una fuerza en el péndulo. Para ello se completó el diagrama de bloques del apartado anterior, obteniendo un posible modelo para definir esta relación. El nuevo diagrama de bloques se expone en la Figura 6.5. Es importante tener en cuenta que la fuerza representada es la paralela al movimiento del carro, mientras que en realidad deberíamos trabajar con la componente perpendicular al eje del péndulo (ver Figura 6.2). A pesar de ello, no se considera una mala aproximación, sobre todo trabajando en ángulos pequeños.

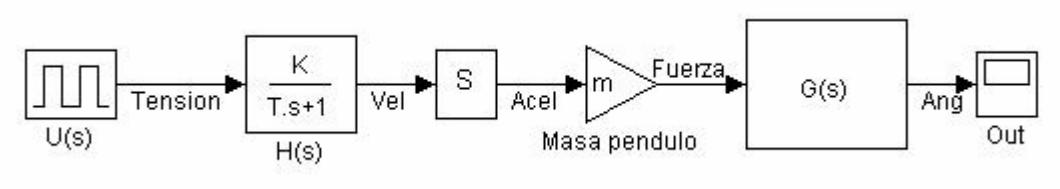


Figura 6.5. Diagrama de bloques genérico del sistema tensión-ángulo

El bloque H(s) corresponde a una función de transferencia de primer orden, que nos da la relación entre la tensión a los motores y la velocidad del carro. Una vez obtenida la velocidad, al multiplicar por S (derivar), obtenemos la aceleración. La aceleración multiplicada por la masa del péndulo nos da la fuerza resultante aplicada a éste (de la conocida ecuación F=ma). Y finalmente mediante la función de transferencia G(s) que se elija del apartado anterior se obtiene el ángulo final.

La masa del péndulo es conocida y de valor 0.881 kg. Sólo queda por hallar los parámetros de la función de transferencia de primer orden que nos relaciona la tensión a los motores y la velocidad del carro

$$\frac{V(s)}{U(s)} = H(s) = \frac{K}{Ts+1}$$

donde K es la ganancia estática (relación entre la salida en régimen estacionario y la entrada) y T es la constante de tiempo, o el tiempo que tarda el sistema en alcanzar el 63% de su valor final. Irremediablemente tenían que

obtenerse estos parámetros de forma gráfica, para lo que tuvo que diseñarse un experimento de velocidad. El código de este experimento, incluido en el Anexo A, es "*PruebaVelocidad.c*". Se dio al coche un escalón de tensión al 100% de potencia, y se graficó la velocidad resultante con un tiempo de muestreo de 30 ms. La forma de hacerlo fue comparando cada muestra del tacómetro con la muestra inmediatamente anterior. Una explicación más completa del código se encuentra en dicho anexo. Los resultados obtenidos se muestran en la figura 6.6.

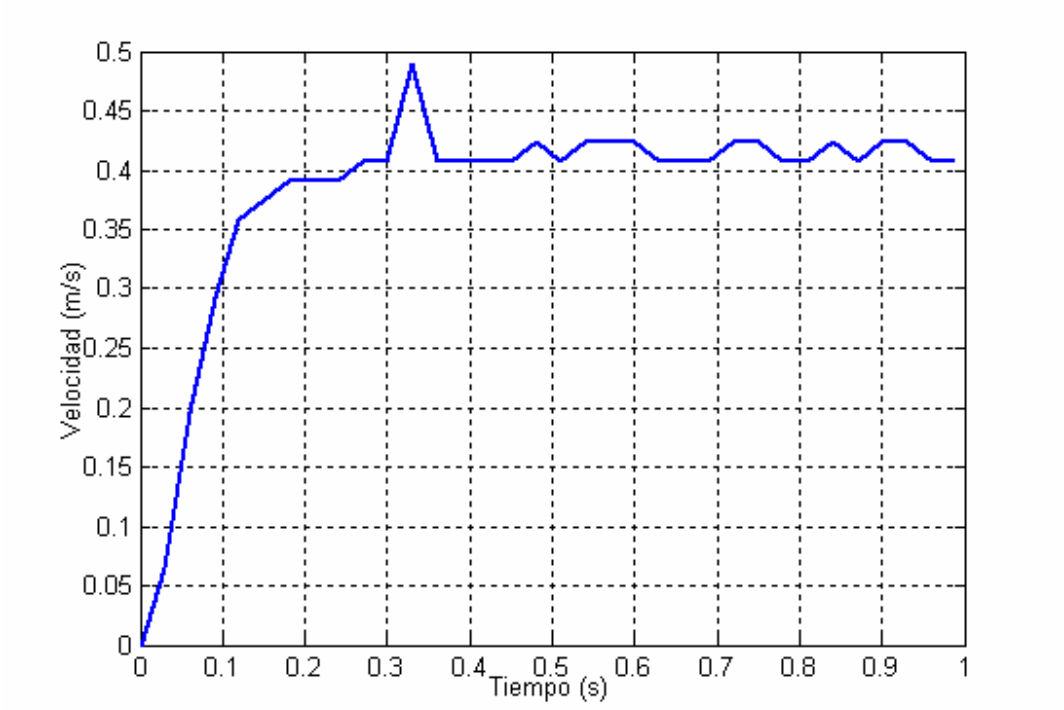


Figura 6.6. Grafica de resultados del experimento de velocidad, al 100% de potencia.

Los parámetros obtenidos son K=0.0040725, y T=0.0788 segundos. Otra información que sacamos de la gráfica es que la maqueta alcanza una velocidad máxima de algo más de 0.4 m/s, y que alcanza este valor en un tiempo muy pequeño (unos 0.2 segundos). La prueba se hizo al 100 % de potencia ya que valores menores acortaban mucho el tiempo de subida, por lo que hacían difícil muestrear la salida correctamente. Aun así hay algunos picos en la gráfica correspondientes a pequeñas imperfecciones, aunque la forma de la curva de salida es bastante clara.

Ya con los valores de los parámetros buscados, se puede formar la función de transferencia correspondiente:

$$H(s) = \frac{0.0040725}{0.0788s + 1}$$

Aplicándole distintas entradas escalón a esta función de transferencia, se comprobó que la respuesta concuerda bien con la del experimento. En concreto, para un escalón de 100% de potencia, se obtiene el resultado de la Figura 6.7.

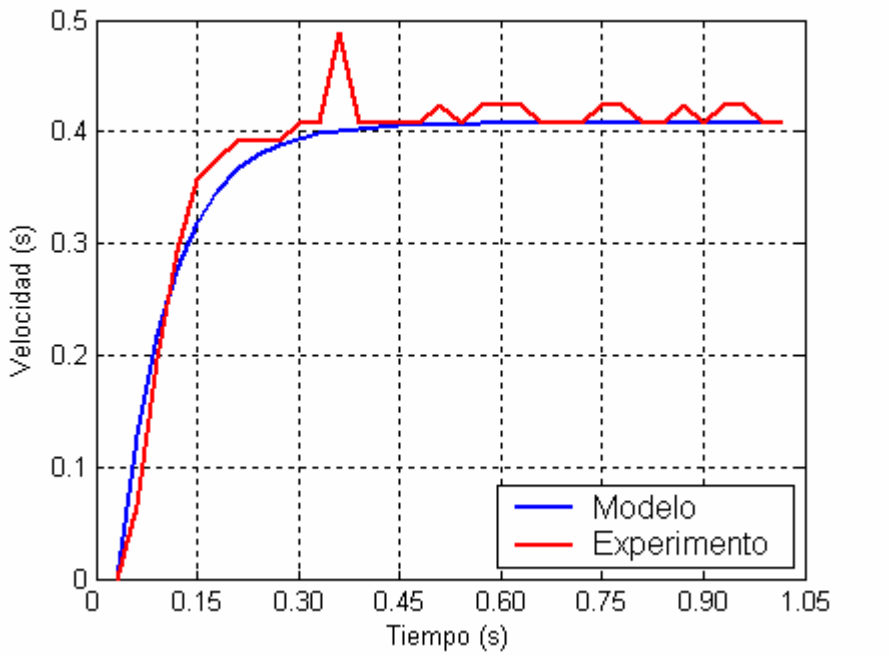


Figura 6.7. Grafica comparativa de la salida experimental y la salida del modelo para un escalón de 100%, en el experimento de velocidad.

Una vez obtenida H(s), ya se tienen todos los elementos para el modelo que se estaba buscando (Figura 6.8). Sólo falta por determinar qué función G(s) se usará.

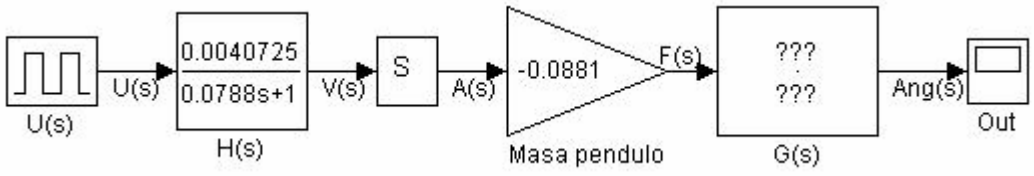


Figura 6.8. Diagrama de bloques final que relaciona ángulo de salida y tensión de entrada

Obsérvese que la masa del péndulo se ha introducido con signo negativo. Esto es debido a que realmente el sentido de la fuerza será contrario al del movimiento del vehículo, a causa de la inercia. La elección de la función G(s) se estudiará en el apartado siguiente.

6.2.2. Experimentos y validación

Para elegir la función G(s) más adecuada, se simularon distintas entradas de tensión en el modelo y se fue probando con las distintas G(s). Los resultados se compararon con los obtenidos experimentalmente. Estos experimentos consistieron también en aplicar una tensión al coche y recoger el ángulo de salida.

En estas pruebas, en lugar de dar la tensión de entrada durante un tiempo determinado, se optó por aplicar la señal de entrada hasta que el coche recorriera una distancia determinada. De modo que no es seleccionable el tiempo de impulso, sino la distancia que se quiere recorrer. Para ello es necesario relacionar la señal que nos proporcionan los encoders de los motores con la distancia recorrida. La ecuación correspondiente para realizarlo es la siguiente. Obtenemos el valor de encoder ó giro en grados que tendremos que elegir para recorrer la distancia buscada.

$$Giro = \frac{d \cdot 360}{56\pi}$$

Donde *d* es la distancia que queremos expresada mm. El 56 corresponde en nuestro caso al diámetro del neumático de LEGO que se ha utilizado, también en mm.

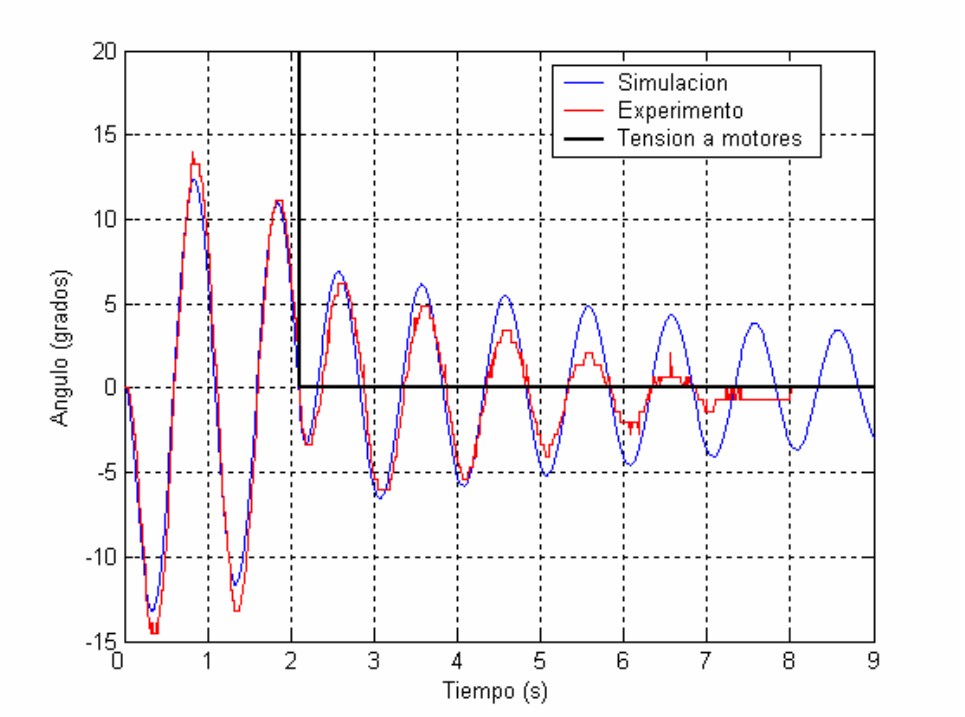
Es importante saber que esta ecuación no es del todo exacta. Debido a pequeños deslizamientos dependiendo de la superficie de trabajo, y a las imperfecciones en el tamaño y construcción de las ruedas o ejes, normalmente la distancia recorrida en la realidad será algo menor que la que predice la fórmula anterior. Sin embargo, se puede considerar una aproximación suficiente de momento.

Esta ecuación se usa en el código “PruebaImpulso.c” (ver Anexo A), que calcula y guarda en un archivo de texto el ángulo del péndulo con un periodo de muestreo de 10ms, y ante distintas entradas de tensión seleccionables. Esta entrada también se almacena con el mismo tiempo de muestreo en otro archivo. Obtenemos entonces las listas de valores de tensión aplicada a los motores y de ángulo de salida. Se recuerda que se aplica la entrada hasta que se recorra una distancia determinada. De todas formas, es fácilmente calculable el tiempo que se aplica esta entrada, comprobando el momento en que se da el último valor de tensión.

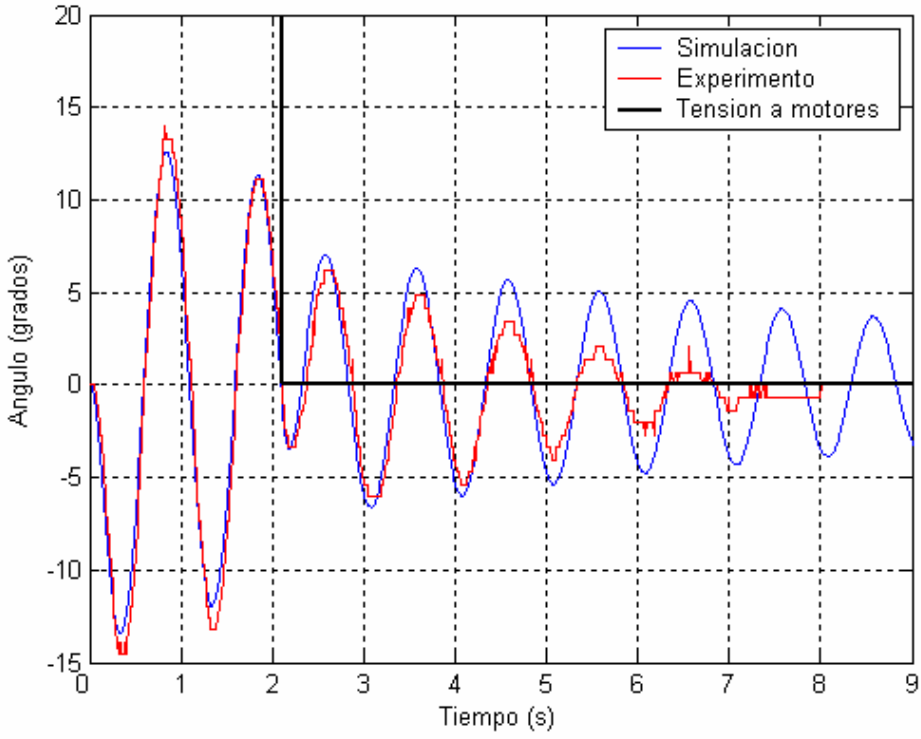
Algunas gráficas para comparar la oscilación obtenida en los experimentos con la obtenida en simulación, se muestran en las figuras 6.9 y 6.10. En todos los casos la potencia aplicada fue del 100%, pues potencias menores no mostraban movimientos considerables del péndulo. Nótese que la señal de entrada en las

gráficas se ha escalado para que no se saliera de la representación, por lo que su valor no corresponde con el indicado en el eje de ordenadas.

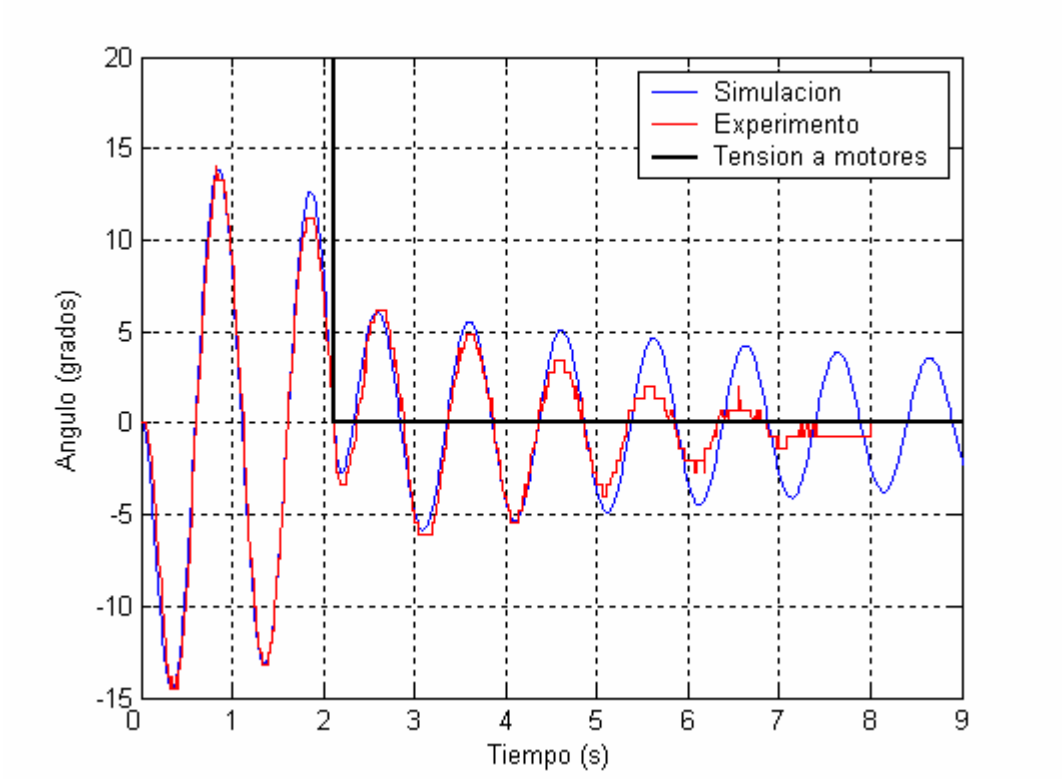
Para desplazamiento de 90 centímetros (Figura 6.9):



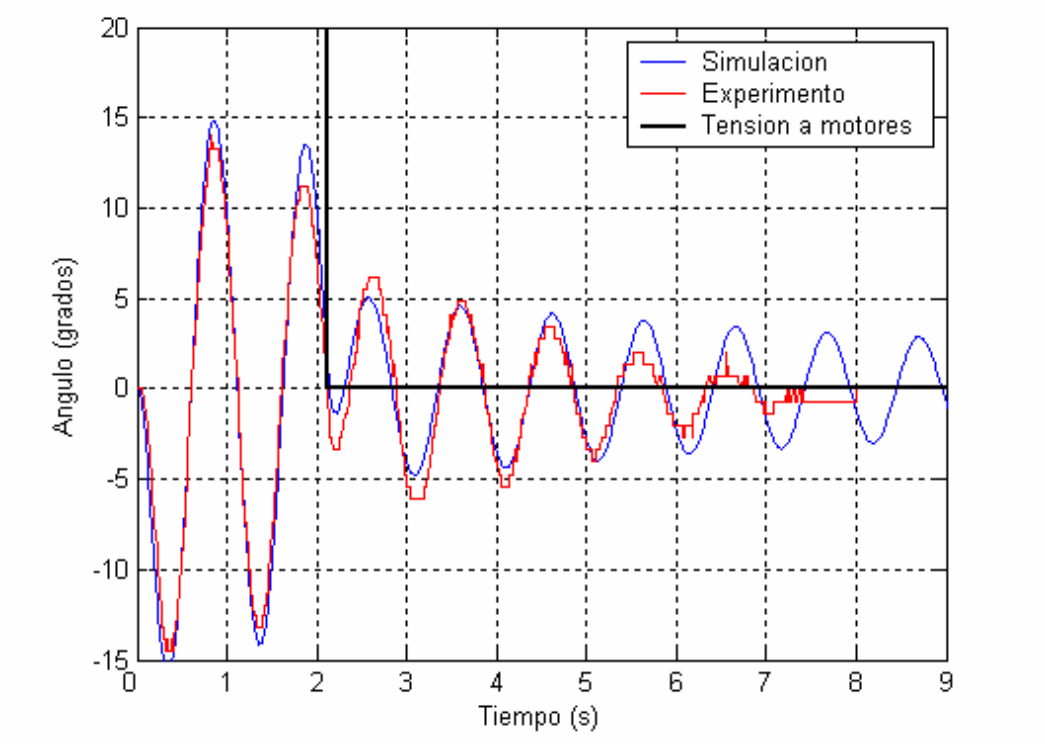
(a) Con G(s) identificada para 10 grados



(b) Con G(s) identificada para 20 grados



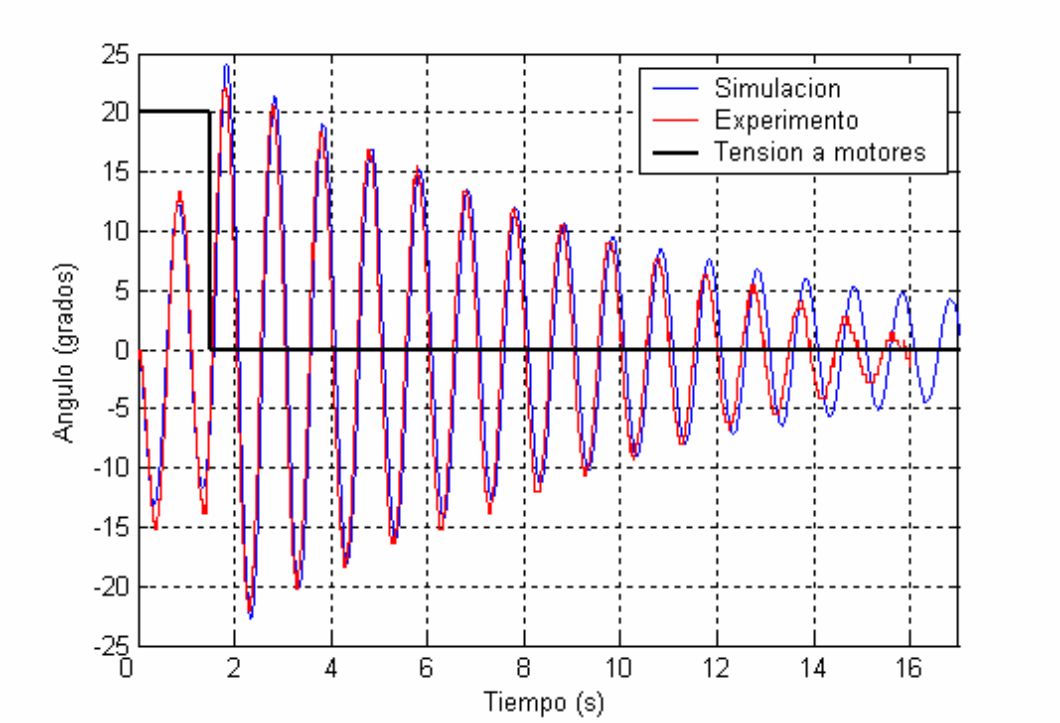
(c) Con G(s) identificada para 45 grados



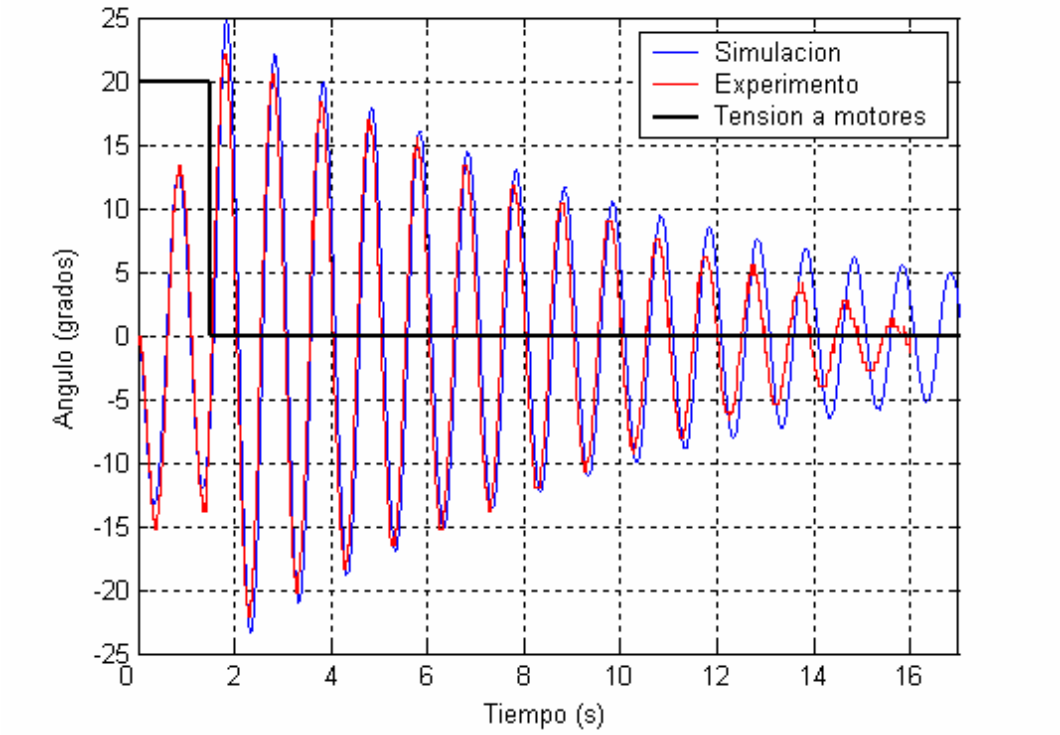
(d) Con G(s) identificada para 60 grados

Figura 6.9. Gráficas comparativas de la predicción de los modelos para una distancia de 90 cm

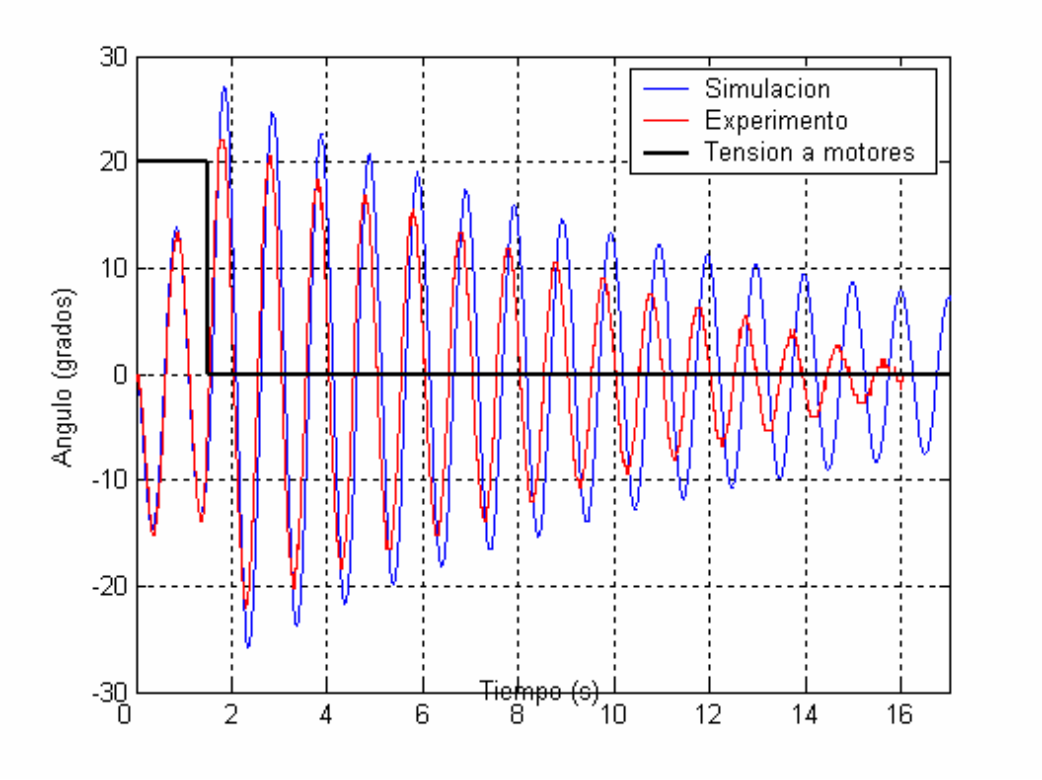
Para desplazamiento de 60 centímetros (Figura 6.10):



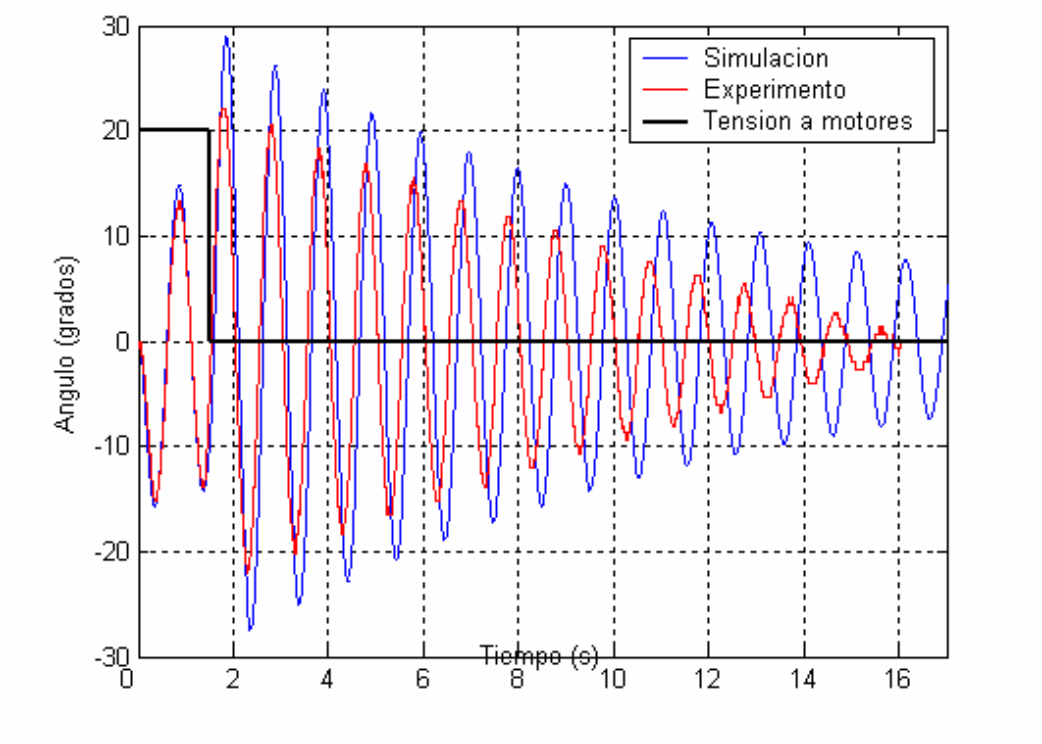
(a) Con G(s) identificada para 10 grados



(b) Con G(s) identificada para 20 grados



(c) Con G(s) identificada para 45 grados



(d) Con G(s) identificada para 60 grados

Figura 6.10. Gráficas comparativas de la predicción de los modelos para una distancia de 60 cm

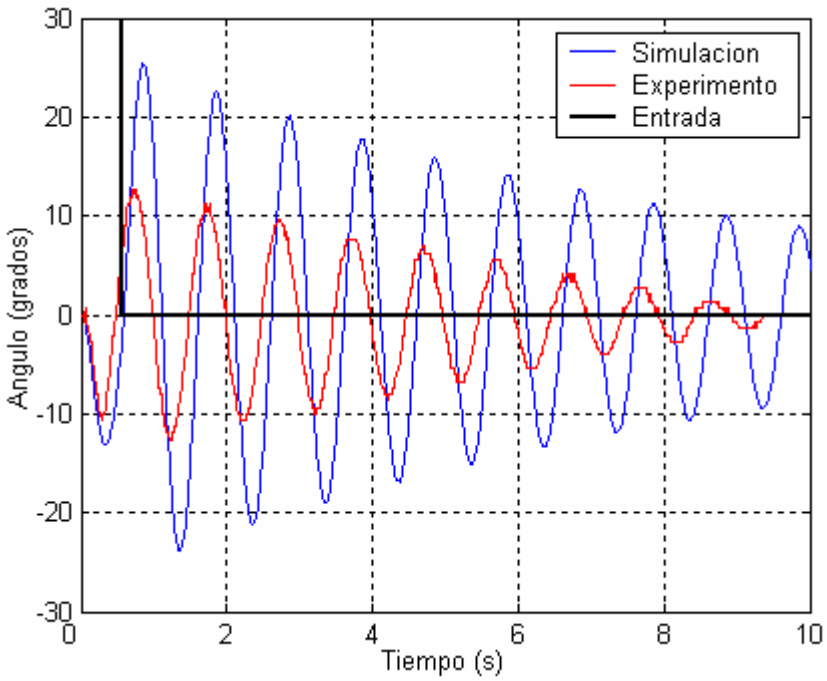
Se comprueba que para ambas distancias, el modelo define con gran exactitud la dinámica de la oscilación. Con todas las G(s) se consiguen buenas simulaciones. Destacan especialmente los resultados obtenidos con las G(s) identificadas para 10 y 20 grados, ya que las oscilaciones habituales del péndulo rondan estos valores. Con las G(s) correspondientes a 45 y 60 grados, se obtienen salidas algo mayores a las reales, como era de esperar.

Además de la comparación entre la simulación y los datos experimentales, también se obtienen algunas conclusiones más de la relación entre carro y péndulo. La primera oscilación, en función de la entrada utilizada, está en torno a los 15 grados negativos (resultado del movimiento del coche hacia delante). Este valor no varía apreciablemente entre una entrada y otra, pues la aceleración del coche es la misma en todos los casos. Una vez se alcanza esta oscilación máxima, comienza la atenuación del balanceo. Sin embargo, se aprecia un fenómeno al parar los motores. El coche, en su movimiento de frenada, produce o bien una amplificación o bien una atenuación del movimiento del péndulo, dependiendo del sentido que lleve éste en el momento de la frenada.

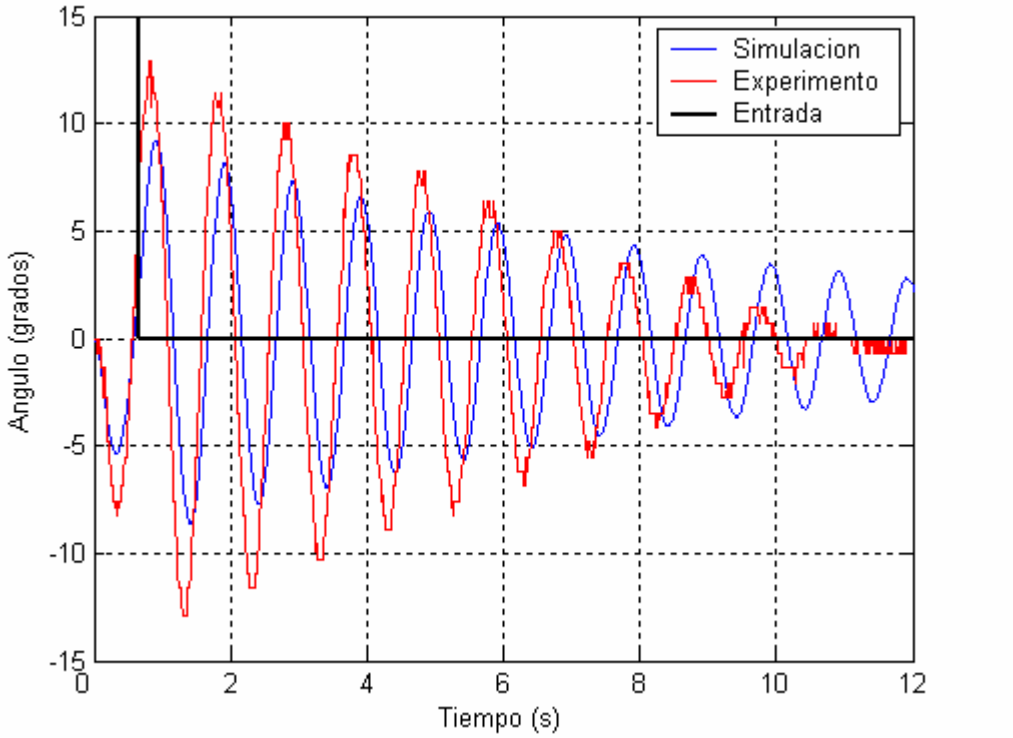
Este hecho se aprecia bien en las figuras 6.9 y 6.10. En el recorrido de 60 cm, el momento en que se produce la frenada coincide con un balanceo hacia adelante del péndulo (la gráfica de salida está creciendo) por lo que la inercia impulsa aún más el péndulo hasta llegar a unos 23 grados. El caso contrario ocurre en el recorrido de 90 cm (Figura 6.9), en la que la frenada atenúa la oscilación.

Cuando se recorren distancias menores (tiempo de impulso pequeño), como por ejemplo 10 centímetros, el experimento y la simulación no se superponen tan bien como debería. Esto es debido a que el modelo no contempla el derrape inicial de las ruedas que realmente se produce al activar los motores al 100 % de potencia. Debido a este derrape, el carro “cree” que ha recorrido una distancia cuando en realidad no lo ha conseguido, por lo que no se aplica la fuerza correspondiente al péndulo. Éste efecto solo es apreciable en recorridos cortos, en los que el vehículo no llega a alcanzar la velocidad que debería.

Para disminuir este derrape, y por consiguiente poder comparar el modelo con los datos experimentales, se activaron los motores al 40 % en lugar de al 100%. Los resultados del experimento para una distancia de 10 centímetros se representan en la Figura 6.11.



(a) 100 % de velocidad, 10 centímetros, simulación con G(s) identificada para 20 grados.



(b) 40 % de velocidad, 10 centímetros, simulación con G(s) identificada para 20 grados

Figura 6.11. Gráficas comparativas de la respuesta con derrape y sin derrape

Aunque no tan bien como en distancias mayores, se ve que la predicción mejora bastante al aplicar una tensión al 40%. Al no derrapar el carro, incluso la oscilación del péndulo es mayor que al iniciar los motores al 100%.

En base a todas las pruebas realizadas, se concluye que se puede dar por válido el modelo. La función de transferencia G(s) que al final se eligió fue la identificada para 20 grados, dado que genera buenas aproximaciones y que las oscilaciones típicas del péndulo ante movimientos del coche son cercanas a ese valor.

$$G(s)=\frac{\theta(s)}{F(s)}=\frac{2671.118335}{s^2+0.212763s+31.478417}$$

Y el diagrama de bloques final del sistema se muestra en la Figura 6.12.

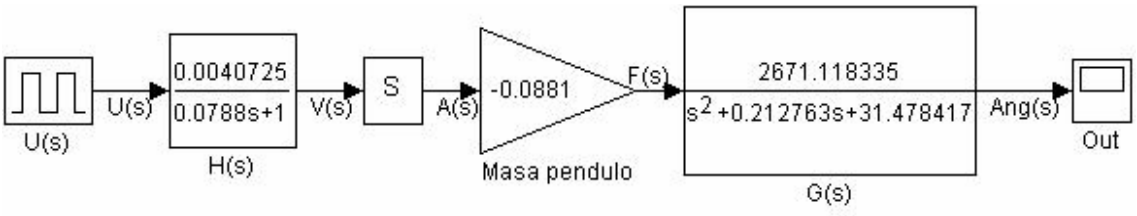


Figura 6.12. Diagrama de bloques final del sistema con la G(s) elegida.

CAPÍTULO 7

EXPERIMENTOS DE CONTROL

El objetivo final de este proyecto es llegar a diseñar tres experimentos de control utilizando el sistema coche-péndulo (control de posición, control de balanceo de la carga, y control simultáneo de las dos variables anteriores). El control de estos sistemas en lazo cerrado girará en torno a reguladores PID, uno para cada experimento, habiendo también para cada uno de ellos un objetivo a conseguir distinto.

Para llevar todo esto a cabo se cuenta con la capacidad de procesamiento del NXT y la de sus motores. Se verá que las especificaciones del conjunto, aun no siendo excesivamente potentes en comparación con las de equipos industriales, cumplen más que de sobra lo buscado. Más aún si se tiene en cuenta que estamos hablando de un producto destinado al entretenimiento. Por supuesto, de este último hecho surgen algunas limitaciones que se verán en su momento.

En este capítulo se explica el método de sintonización y la obtención de los parámetros k_p, k_i y k_d del regulador, para cada uno de los tres experimentos mencionados anteriormente. Además, se presentarán los resultados obtenidos. Obviamente para obtener estos resultados, anteriormente se han tenido que implementar estos controladores en RobotC. El código utilizado no es relevante ahora mismo, por lo que las tareas de programación no se comentarán en este capítulo, sino que se dejarán para el siguiente.

A continuación se describen más a fondo los resultados obtenidos para cada una de las pruebas, tanto en simulación como experimentalmente.

7.1. Control de la posición del carro

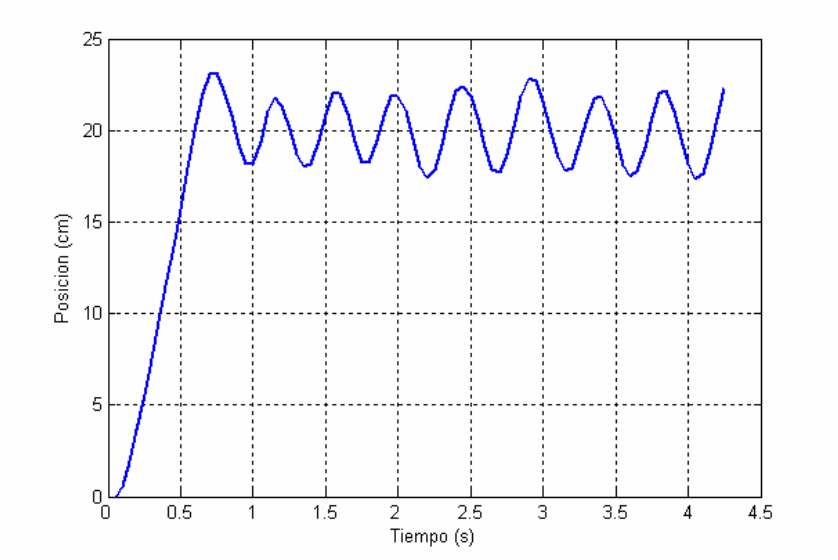
Este es el experimento más sencillo que se realizará, pues sólo involucra al NXT y sus motores, olvidándonos de momento del péndulo.

7.1.1. Descripción del sistema

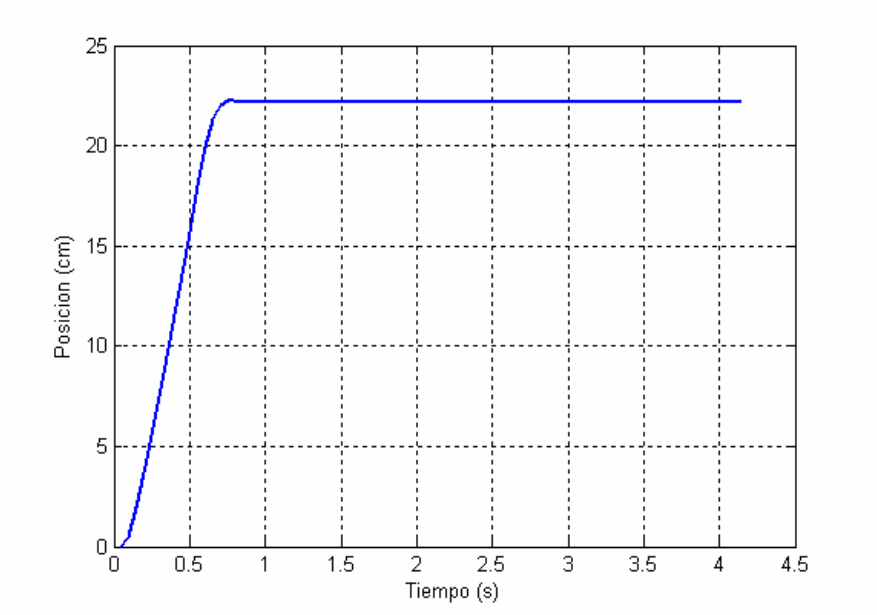
El primer experimento consistirá en el control de posición del coche. Por consiguiente, el vehículo deberá desplazarse hasta la posición que se le indique. Esta posición de consigna podrá especificarse positiva o negativa, lo que correspondería con un desplazamiento hacia adelante o hacia atrás. Todas las pruebas de este apartado suponen el coche desplazándose con toda la estructura y el peso del péndulo.

Surge un experimento previo a la implementación del PID. Si damos una referencia (posición) al sistema, se trata de usar un controlador *"todo-nada"* para aplicar la señal de control más sencilla a los motores. Esta señal valdrá +100 en el caso de que la señal de error sea positiva, y -100 en el caso contrario. Para hacerlo se hizo una pequeña modificación al código *"ControlPosicion.c"*, incluido en el Anexo A.

Se estableció la posición de consigna en 20 centímetros. Con un tiempo de muestreo de 50ms se obtuvieron los resultados mostrados en la Figura 7.1.



(a)



(b)

Figura 7.1. Gráficas de resultados con controlador on-off, para (a) tolerancia de 1cm y (b) tolerancia de 3cm

En ambos casos la consigna es 20 centímetros. Sin embargo, vemos que introduciendo una tolerancia de 1 centímetro en la posición (muy poca) el sistema se vuelve inestable debido a la brusquedad de la señal de control (recordemos que oscila entre -100 y +100). Si en cambio dejamos una holgura para el estacionamiento de 3 centímetros, el vehículo alcanza la posición buscada pero con un error de aproximadamente 2.4 centímetros debido a la frenada al llegar a los 20 centímetros.

Por supuesto estos resultados son mejorables, y es lo que se intenta conseguir introduciendo el PID. Aunque RobotC incluye de por sí una función para el control de velocidad de los motores mediante un PID (*nMotorPIDSpeedCtrl*), se decidió implementar uno manualmente para mantener una homogeneidad en todos los experimentos.

7.1.2. Sintonía del PID en simulación

El sistema con el que contamos, una vez incluida la realimentación y representado en forma de diagrama de bloques, se muestra en la Figura 7.2.

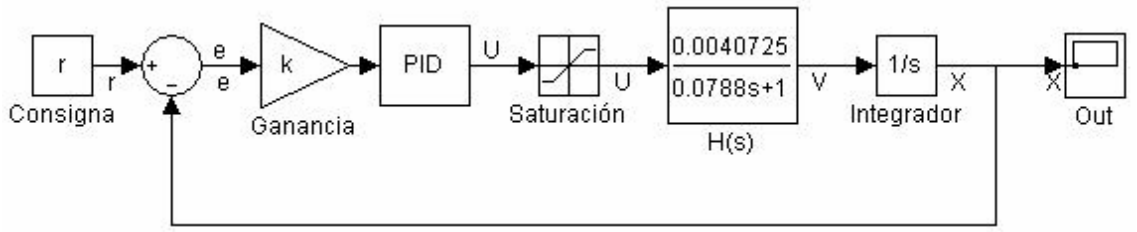


Figura 7.2. Diagrama de bloques del experimento de control de posición

Podemos considerar la entrada *r* como una entrada escalón, que será la posición de consigna. La salida *x* será la posición a la que llegamos. La señal de error *e* se multiplicará por una ganancia para adaptarla a la entrada del controlador PID, que a su vez nos dará la señal de control final a los motores. La función H(s) es la ya vista para relacionar la tensión y la velocidad, e integrando la velocidad (bloque 1/s) obtenemos la posición. La saturación representa el límite máximo y mínimo de tensión que puede entrar a los motores.

La respuesta de este sistema no es muy complicada y no presenta casi oscilación, pues se asemeja a un primer orden. Se pensó que ante esta situación sería suficiente el usar un controlador proporcional, dejando a un lado el uso del integral y el derivativo. Esto obviamente habría que comprobarlo.

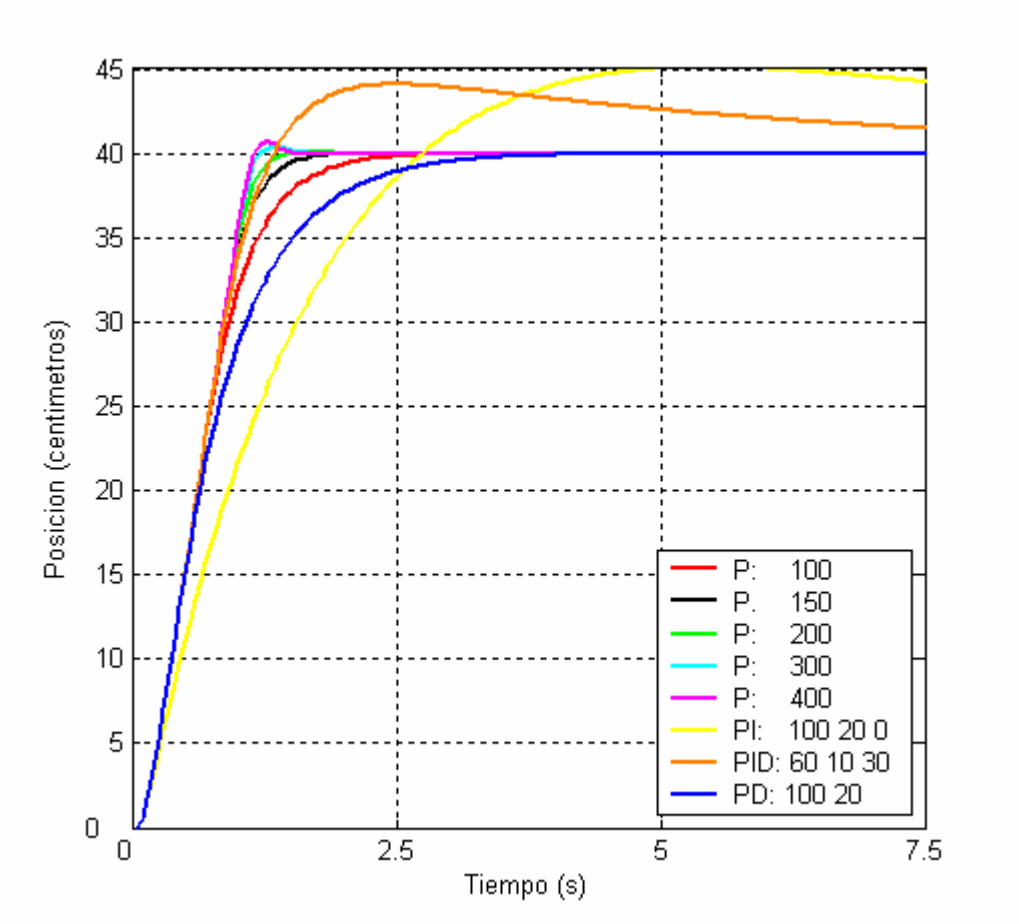
Una consideración previa a tener en cuenta es la magnitud de las distintas señales. En primer lugar, tendremos una entrada a motores, que variará entre -100 y +100. También sabemos que el error de posición será de unos cuantos centímetros, que luego serán multiplicados por los respectivos términos del PID (suponiendo que estén presentes los tres) y sumados entre sí, lo que puede hacer que la señal de control final sea mucho mayor que 100, y lleve a los motores a saturación muy rápido. Para evitarlo, se concluye que la ganancia previa al controlador deberá ser menor que la unidad para disminuir el valor de este error antes de entrar al controlador. Esto nos aumentará el rango de valores a elegir para los términos P, I y D de este regulador.

Se adelanta que este planteamiento será el mismo para todos los experimentos posteriores: será necesario incluir una ganancia menor que la unidad en todos. Se decidió en principio que esta ganancia sería ***k=0.05***, y si fuera necesario aumentar la señal posteriormente, hacerlo mayorando los términos del regulador.

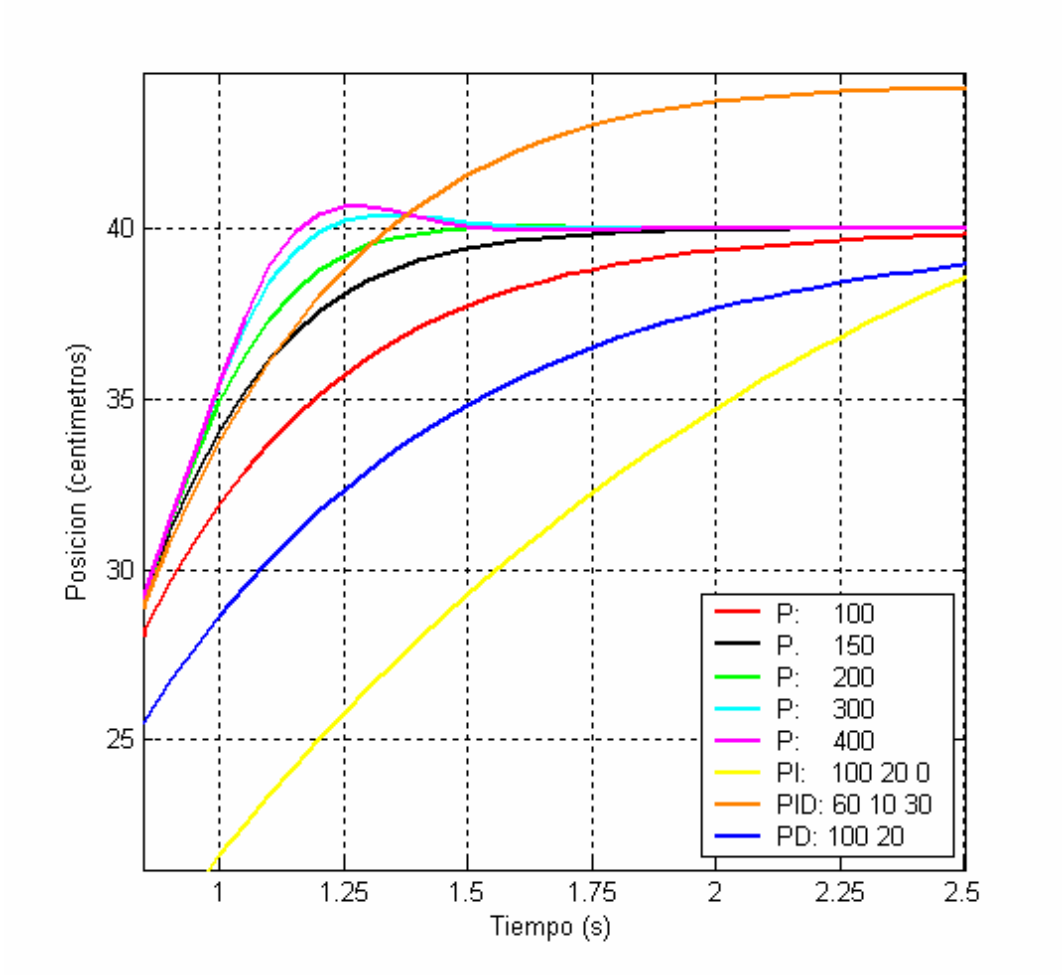
Hagamos una aproximación muy burda y a ojo de los valores adecuados que podrían encajar en el controlador. Supongamos que el valor de consigna es 20 centímetros. Entonces el error inicial será 20 centímetros, que multiplicado por *k=0.05* nos da una señal al controlador unitaria. Por supuesto, al no tratarse de una distancia pequeña, queremos que en este primer instante la señal a los motores

esté cerca de la máxima (100). Lo que significa que el término proporcional deberá estar cercano a este valor. En principio el error estacionario será tan pequeño que no se necesitará término integral, y la dinámica será tan sencilla que tampoco parece necesario un término derivativo.

Se decidió entonces fijar el valor de P en 100, y a partir de ahí, mediante distintas simulaciones, ir sintonizando o afinando adecuadamente este valor, o añadir alguno de los otros dos términos si fuese necesario. Con K=0.05 y con una posición de consigna de 40 centímetros, se hicieron en Simulink distintas simulaciones con distintos PID. Algunos resultados se muestran en la Figura 7.3.



(a) Resultados



(b) Resultados ampliados

Figura 7.3. Gráfica de resultados de la simulación del control de posición con distintos PID (a), y ampliación de los sobrepicos (b)

Efectivamente, se obtienen muy buenos resultados con controladores proporcionales de valor entre 150 y 400. Se aprecia que aunque varíe este valor gran cantidad (350 unidades), las salidas siguen siendo muy parecidas. Aun así, parece que el modelo predice que un P=400 sería el más adecuado. También se ve que al introducir un pequeño término integral, la señal de salida presenta demasiado sobrepico.

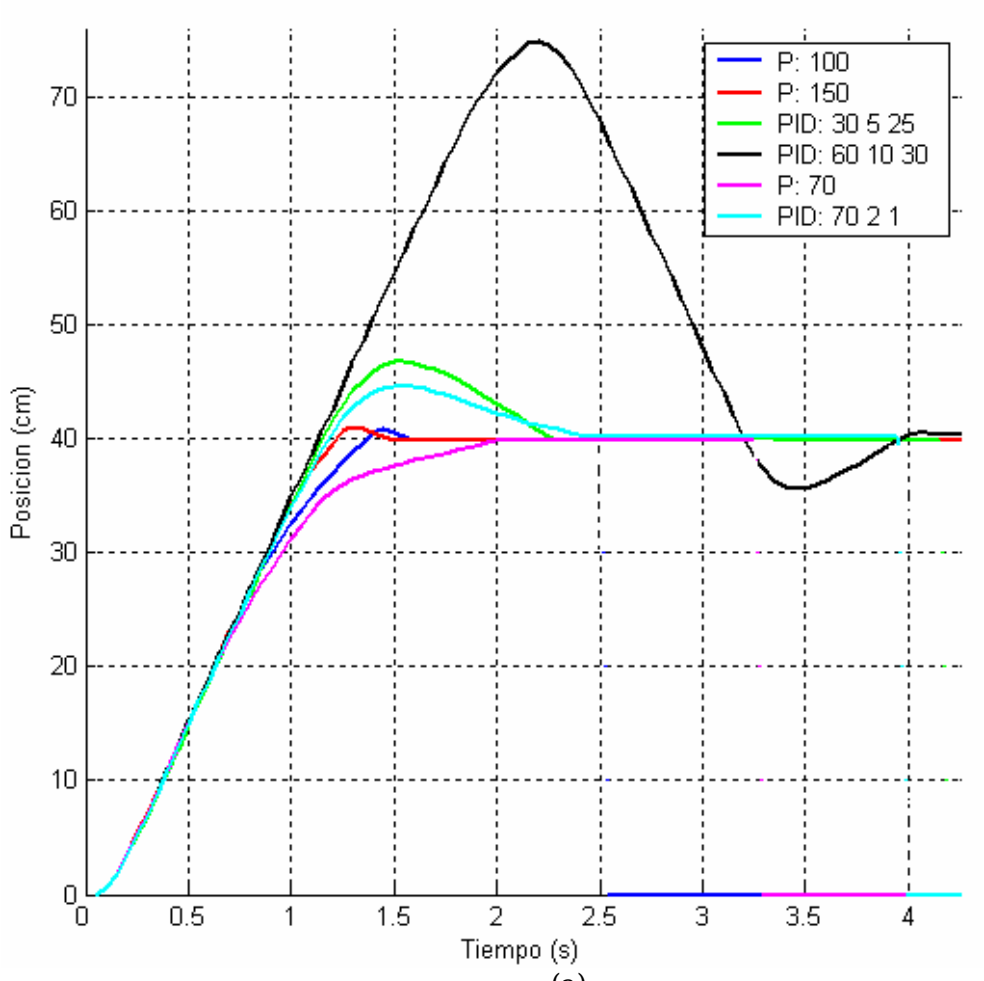
Ahora que ya está simulada la dinámica del sistema tenemos una idea de los valores que puede tomar el PID, pero para hallar el valor exacto habrá que pasar a hacer pruebas experimentales.

7.1.3. Ajuste del PID experimentalmente

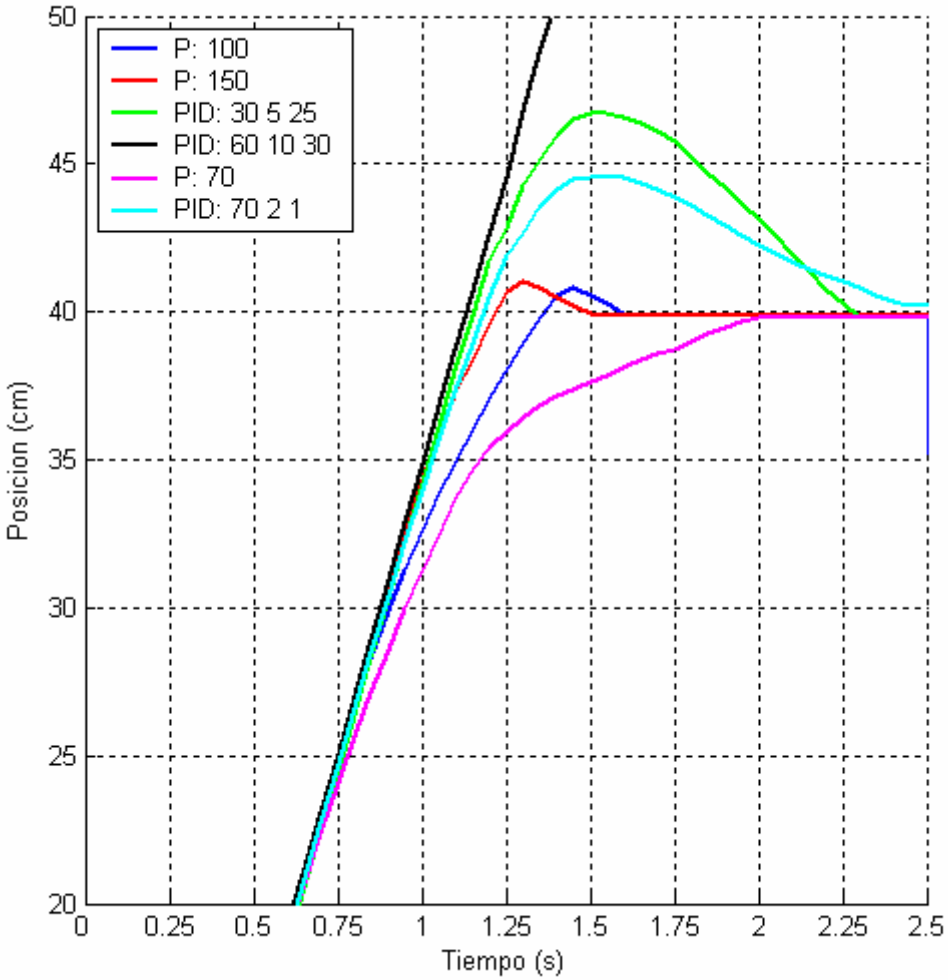
Una vez obtenidos los valores del PID en simulación, se pasó a implementar en RobotC el código correspondiente. Basándonos en los resultados obtenidos, y en que los resultados experimentales no deberían ser muy diferentes, se comenzó a dar valores al PID comenzando por un P=100. Algunos resultados se muestran en la Figura 7.4. En todos los casos la consigna de posición se fijó en 40 centímetros, y el tiempo para cada bucle de realimentación es de 50 milisegundos, lo suficiente para permitir el movimiento del coche.

Nota:

Se recuerda que la distancia real recorrida no es exactamente de 40 centímetros. Esto es debido sobre todo al derrape de las ruedas dependiendo de la superficie de trabajo, que hace que la distancia final sea de unos 2 ó 3 centímetros menos, disminuyendo este error como es lógico para distancias menores. De todas formas se consideró una buena aproximación a lo esperado, y no se realizó ningún ajuste adicional.



(a)



(b)

Figura 7.4. Gráfica de resultados experimentales para el control de posición (a), y ampliación de los sobrepicos (b)

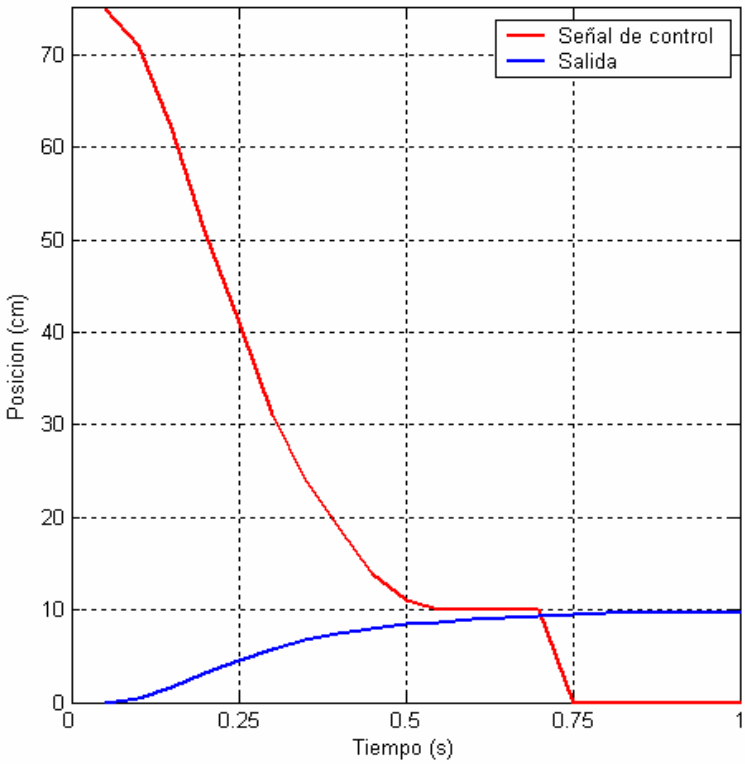
En este caso la mejor respuesta se ha obtenido usando un controlador proporcional de valor 150, menor que el obtenido en simulación. Es la respuesta con menor sobrepico y menor error estacionario, siendo este último despreciable y de aproximadamente 0.2 centímetros.

La razón de que los valores obtenidos para el PID experimentalmente difieran de los simulados (recuérdese que se obtuvo un P = 400 en simulación) es que el modelo teórico no es del todo exacto, siendo su principal problema que no contempla el derrape de las ruedas, que se produce sobre todo en la puesta en marcha y en el cambio de sentido que efectúa el carro en el momento del sobrepico. De todas formas, en simulación un P = 200 también produce muy buenos

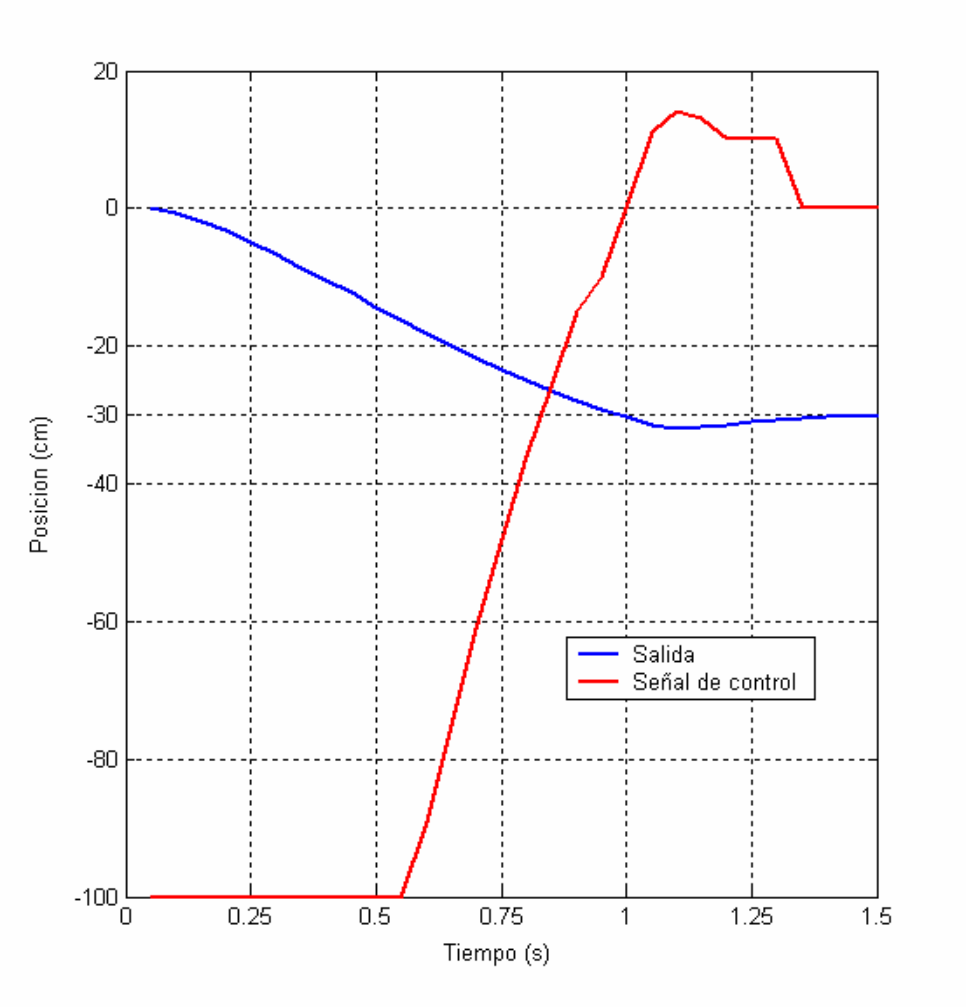
resultados, y se asemeja mucho a nuestro P = 150. La predicción que hizo el modelo es bastante buena, pues el sobrepico y el tiempo de establecimiento real (de 1.5 segundos) prácticamente coinciden.

En estos experimentos, nótese que aunque se reduzca el valor del término proporcional a más de la mitad, si se introducen unos valores bajísimos para I y D la salida se dispara produciendo un sobrepico inaceptable, por la suma recurrente de errores que produce el integral (curvas negra y verde) y que lleva a los actuadores a saturación muy rápido. Otra característica es que la pendiente inicial es muy similar para todos los controladores, y por tanto el tiempo de subida también. Este tiempo prácticamente no se puede disminuir, por la incapacidad de los motores de trabajar a más potencia.

Por tanto se eligió definitivamente un controlador proporcional de valor 150, por su sencillez y cumplimiento con la respuesta buscada. Ahora que se tiene la respuesta del sistema con este controlador para una distancia de 40 centímetros, pero no está de más poner a prueba el regulador con algunas consignas de distancia distintas. Como se verá, los resultados obtenidos de estas pruebas son satisfactorios, por lo que se da por bueno este controlador. En la Figura 7.5 se muestran los datos recogidos de dos de estos experimentos. Se muestra tanto la salida como la señal de control a motores.



(a) Consigna de 10 centímetros



(b) Consigna de -30 centímetros

Figura 7.5. Respuesta del control de posición con P = 150, para distintas entradas

7.2. Control de la oscilación del péndulo

En este caso el vehículo intentará, ante una oscilación descontrolada del péndulo, desplazarse para devolver a éste lo antes posible al punto de reposo.

7.2.1. Descripción del sistema

Ya se hizo una aproximación a la dinámica del balanceo del péndulo en el capítulo anterior. El paso siguiente es cerrar el lazo y construir el sistema realimentado.

Intuitivamente la forma de actuar será la siguiente: si el péndulo está desplazado hacia delante, el vehículo se moverá a una cierta velocidad en esa dirección, para amortiguar la oscilación. Cuando el péndulo se balancee hacia el otro lado, el coche se moverá en la otra dirección, y así ocurrirá hasta conseguir la frenada completa del péndulo. Un poco más adelante se verá que no es exactamente el signo de la señal de error el que define el sentido en el que debe desplazarse el carro, sino que lo importante será el signo de la señal de control, que puede no ser el mismo que el del error debido al uso de un término derivativo. En definitiva, lo importante será definir la forma de estos desplazamientos.

Hay una peculiaridad destacable en el sistema que estamos analizando, si lo comparamos con los sistemas habituales. Aunque se anule la señal de control, o aunque se produzca cualquier perturbación, el péndulo seguirá tendiendo él solo a alcanzar el valor de consigna con error estacionario cero (aunque en un tiempo demasiado grande), cosa que no es habitual en otros procesos. El único cometido del sistema de control será por lo tanto *acelerar* la llegada a este estado estacionario.

7.2.2. Sintonía del PID en simulación

El diagrama de bloques usado para simular el sistema se aprecia en la Figura 7.6.

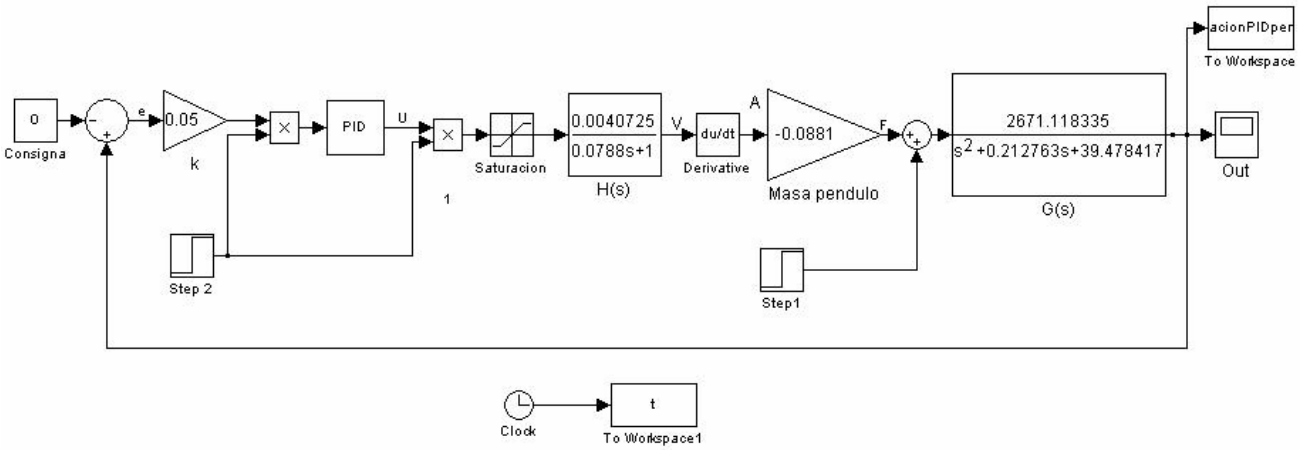


Figura 7.6. Diagrama de bloques del experimento de control del péndulo

La entrada de consigna en este caso será constante e igual al ángulo de reposo de nuestro péndulo, y la salida será el ángulo actual. Para simular correctamente el PID se han añadido unos bloques adicionales. La entrada escalón Step 1 es una fuerza de 0.748471 N, que se ha añadido para conseguir un estado estacionario inicial de 60 grados. Una vez que se llega a este estado estacionario, este escalón (fuerza) cae a cero, y la salida comienza a oscilar (tal y como se vio en el Apartado 6.1.2). El bloque Step 2 es el encargado de activar el PID, un momento después de que actúe el Step 1. De esta forma estaremos simulando un controlador

que comienza a actuar al soltar el péndulo desde una posición de 60 grados. Sin embargo, dado que estamos usando la G(s) que se identificó para 20 grados, el ángulo que conseguiremos (desde el cual comenzará la simulación del PID) es algo menor. El ángulo inicial es de unos 50 grados. Aun así no es mala aproximación.

El proceso es el mismo que en el caso anterior: se calcula el error, se multiplica por una ganancia, se introduce a los distintos términos del PID y se obtiene la señal final de tensión proporcionada a los motores. El resto de bloques son los mismos que se vieron en el capítulo anterior. El bloque de saturación acota la señal de control. Una característica de este sistema es que el error coincide con la señal de salida, debido a que la consigna será siempre el ángulo de referencia o ángulo "cero".

Como en el caso anterior, y para hacernos una idea de los valores que puede tomar el controlador, podemos comenzar simulando el sistema para distintos PID. Recordemos que la ganancia k la hemos fijado en 0.05. Algunos resultados se muestran en la Figura 7.7.

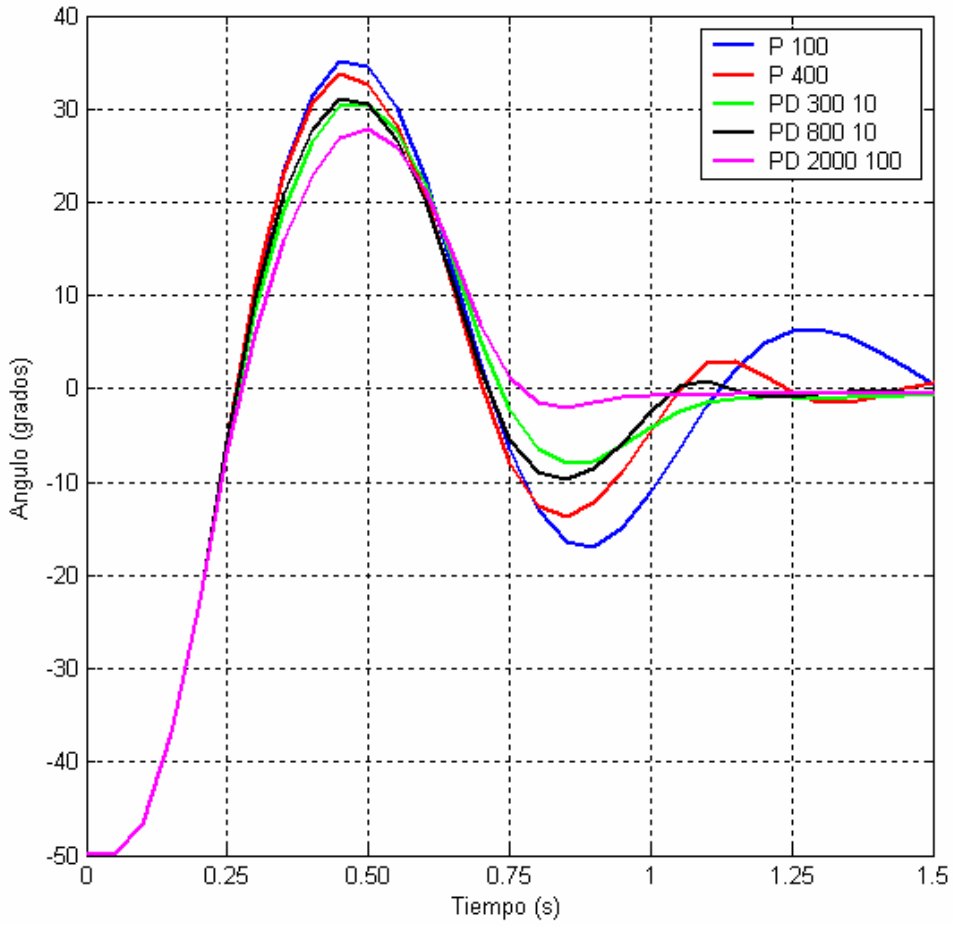


Figura 7.7. Gráfica de la simulación del control del péndulo con distintos valores para el PID

Si se usa un controlador proporcional, el sistema responde adecuadamente, pero hay un rango muy grande de ganancia para los que no se aprecia prácticamente diferencia en la salida (se observa en P = 150 y P = 400). Esto es porque para todos esos valores se produce saturación en los motores. En cuanto a la inclusión de un término integral prácticamente no tiene efecto ni es necesaria (se explica en el apartado siguiente), por lo que no ha sido representado en las gráficas.

En cambio si incluimos el término derivativo mejora la respuesta considerablemente, y permite aumentar el término proporcional. El controlador elegido es un PD de valores 2000 y 100. A priori este valor parece demasiado grande, aunque también parece que podría aumentarse incluso más. Se aprecia que con un PD mucho menor (300, 10), la salida no difiere demasiado, y que por tanto hay un rango amplísimo de valores que producen salidas parecidas. De todos modos, para el controlador final el sobrepico es de unos 28 grados y el tiempo de establecimiento está en torno a 1 segundo. Por supuesto, esta simulación sólo nos da una idea de los valores de PID a probar, y la idoneidad de estos valores habrá que comprobarla experimentalmente.

7.2.3. Ajuste del PID experimentalmente

Vistos los resultados de la simulación, y antes de comenzar a probar valores en el sistema real, se puede hacer un pequeño análisis preliminar de cuáles podrían ser valores adecuados para el controlador, analizando el peso que debería tener cada término.

La ganancia proporcional siempre debe estar presente y en base a ella se elegirán los otros dos términos. En cuanto a la acción integral, ésta es más adecuada generalmente en dinámicas lentas en las que se necesite eliminar el error estacionario. Como se ha comentado, ninguna de estas dos cosas se da en el movimiento del péndulo. Además, tiende a empeorar el transitorio del sistema, por lo que el término integral, en caso de que lo utilicemos, tendrá un valor muy bajo y de apoyo a la ganancia proporcional.

El caso del derivativo es muy distinto. Éste actúa sobre la variación del error en el tiempo (sobre la derivada) mejorando la parte transitoria en sistemas muy oscilatorios, “adelantándose” por decirlo de alguna forma, a la respuesta de nuestro sistema. Una forma de verlo más intuitivamente es la siguiente:

Supongamos que el péndulo, en un instante de su oscilación, está en una posición adelantada, pero sin embargo, está en un movimiento “de regreso” hacia el coche. La señal proporcional indicará al vehículo que debe avanzar, debido al signo

positivo del error. Sin embargo, lo que realmente debería hacer el vehículo es moverse hacia atrás, ya que el péndulo ya lo está haciendo, y nuestro cometido es "amortiguar" ese movimiento. La forma de conseguirlo es con el término derivativo, cuya señal indicará a los motores moverse hacia atrás correspondiendo con el signo de la "derivada" del error (este término no parece muy correcto si estamos hablando en tiempo discreto) en ese momento.

Por lo dicho, usar un término derivativo encaja perfectamente para controlar un péndulo cuya respuesta es completamente oscilatoria, y seguramente este término sea el de más peso en el controlador. Un posible inconveniente de su uso sería que tiende a amplificar cambios bruscos en el error (golpes o frenazos al péndulo), ya sea debido a ruido o a cambios en la consigna, pero de nuevo su uso cumple, pues en el péndulo no se preveía que se dieran este tipo de casos.

En principio parece que con un controlador PD podría ser suficiente, como en simulación. Se probó de todas formas con algunos PID, partiendo de valores bajos del orden de P = 80, I = 20, D = 120. Rápidamente se encontró que estos valores son inadecuados para el controlador, pues la salida se hace muy oscilatoria debido al término integral (incluso disminuyendo más este valor), por lo que finalmente se decidió no usar término integral en este experimento y quedarnos con el controlador PD.

Dado que los valores obtenidos para el PD en simulación parecían muy elevados, se decidió comenzar con valores menores y a partir de ahí, incrementar o disminuir P y D hasta encontrar la respuesta más adecuada. Se partió de valores para P y D en torno a 100. Para comparar con los resultados simulados, las pruebas se han realizado colocando manualmente el péndulo lo más cercano posible a la posición inicial de menos 60 grados. Esto se hizo a ojo, pero es una buena aproximación y produce gráficas similares en todos los experimentos.

En la Figura 7.8 se expone una muestra de los resultados obtenidos, algunos de los cuales son candidatos a ser valores del PD final. El periodo de muestreo ha sido de 50 milisegundos y se representa el ángulo de salida para distintos controladores PD, en función del tiempo.

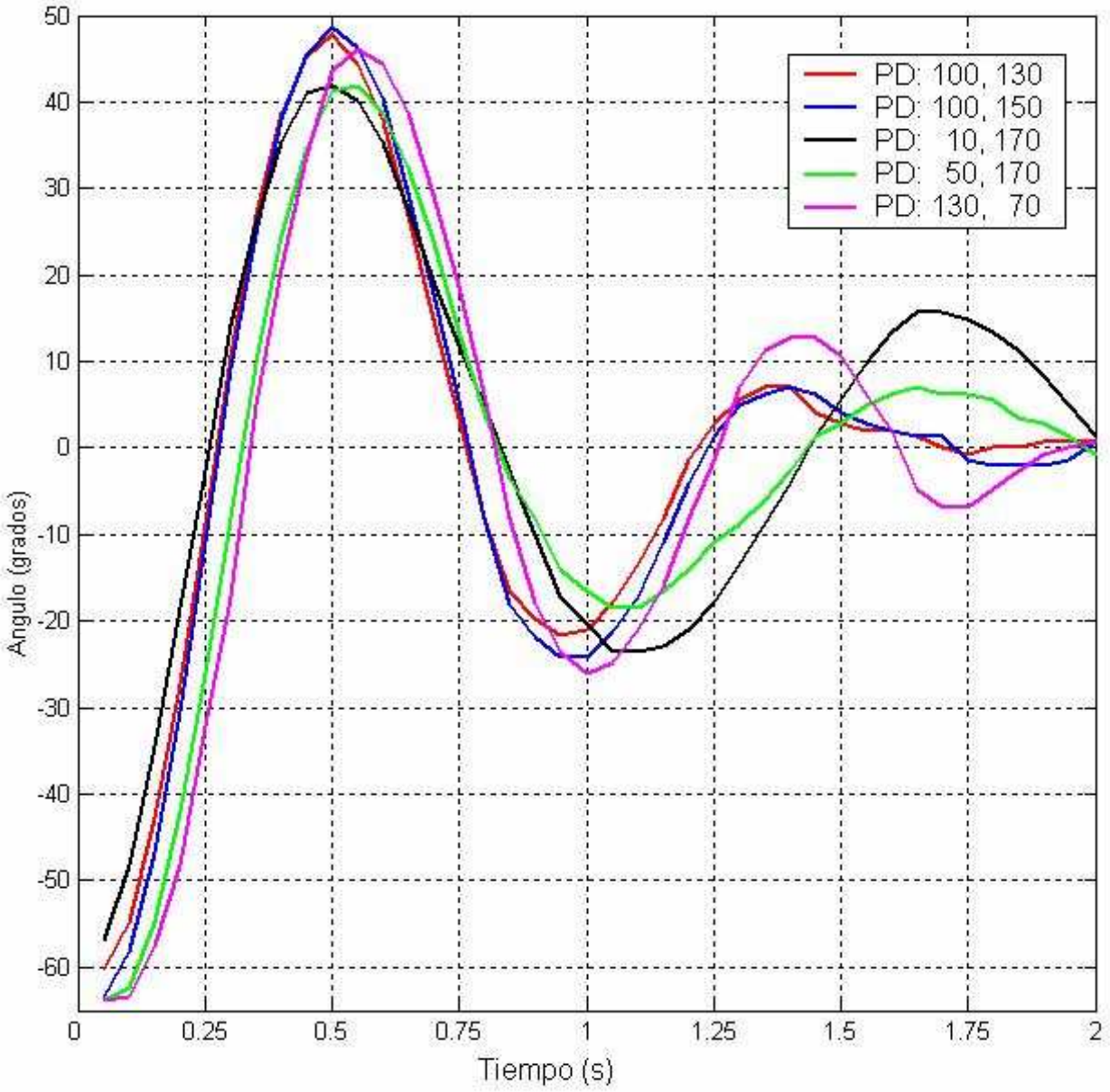


Figura 7.8. Gráfica de resultados experimentales para el control del péndulo

Como se dijo, en este experimento no es importante el error estacionario, ya que sabemos que irá a cero automáticamente. Nos fijaremos por tanto sólo en la parte transitoria de las curvas y el tiempo en alcanzar el valor de consigna. La gráfica sólo representa un conjunto del total de pruebas que se hicieron, dejando a un lado los resultados más desfavorables. Aquí se muestran algunas de las curvas más cercanas a la decisión final, que se adelanta que corresponde al PD con **P = 100 y D = 130**. Para este controlador se produce la menor sobreoscilación y el menor tiempo de establecimiento.

Ya se habrá notado que el controlador obtenido difiere mucho respecto al que se obtuvo en simulación (P=2000, D=100). Estas diferencias en los PID respecto a los valores simulados se deben de nuevo a dos motivos ya mencionados:

- La simulación no contempla el derrape de las ruedas dependiendo de la superficie, sobre todo en los cambios de sentido bruscos que debe realizar el carro.

- La simulación no contempla tampoco las componentes elevadas de fricción del péndulo que se originan al oscilar éste en ángulos pequeños (Capítulo 6).

Aunque el controlador obtenido en simulación no coincide con el real, nos da una idea de la curva de salida y sirve para orientarnos al elegir los PID de los experimentos.

Volviendo a estos PID probados, se ve que hay un rango muy grande de valores posibles a escoger. Esto es porque pequeñas diferencias en estos valores no producían cambios sustanciales en la respuesta del sistema. Debido a ello los distintos términos se fueron eligiendo y probando generalmente dando pasos de diez en diez.

Tras gran cantidad de pruebas se deduce que, en general, los valores candidatos al controlador deben cumplir:

- Valor del término integral muy pequeño, o nulo como en este caso, ya que no es necesario y sólo consigue acercar el sistema a la inestabilidad, pudiéndose suplir su falta incrementando el término proporcional.

- Término proporcional entre 80 y 130 aproximadamente (suponiendo que se use junto a un término derivativo), pues valores menores tienen a alargar el tiempo de establecimiento y mayores tienden a inestabilizar.

- Término derivativo muy elevado, entre 100 y 170, para mejorar el efecto de amortiguación comentado anteriormente.

Algunos ejemplos de respuestas mejorables, bien debido a valores inadecuados para P y D, o a la inclusión de un término integral, se muestran en la Figura 7.6.

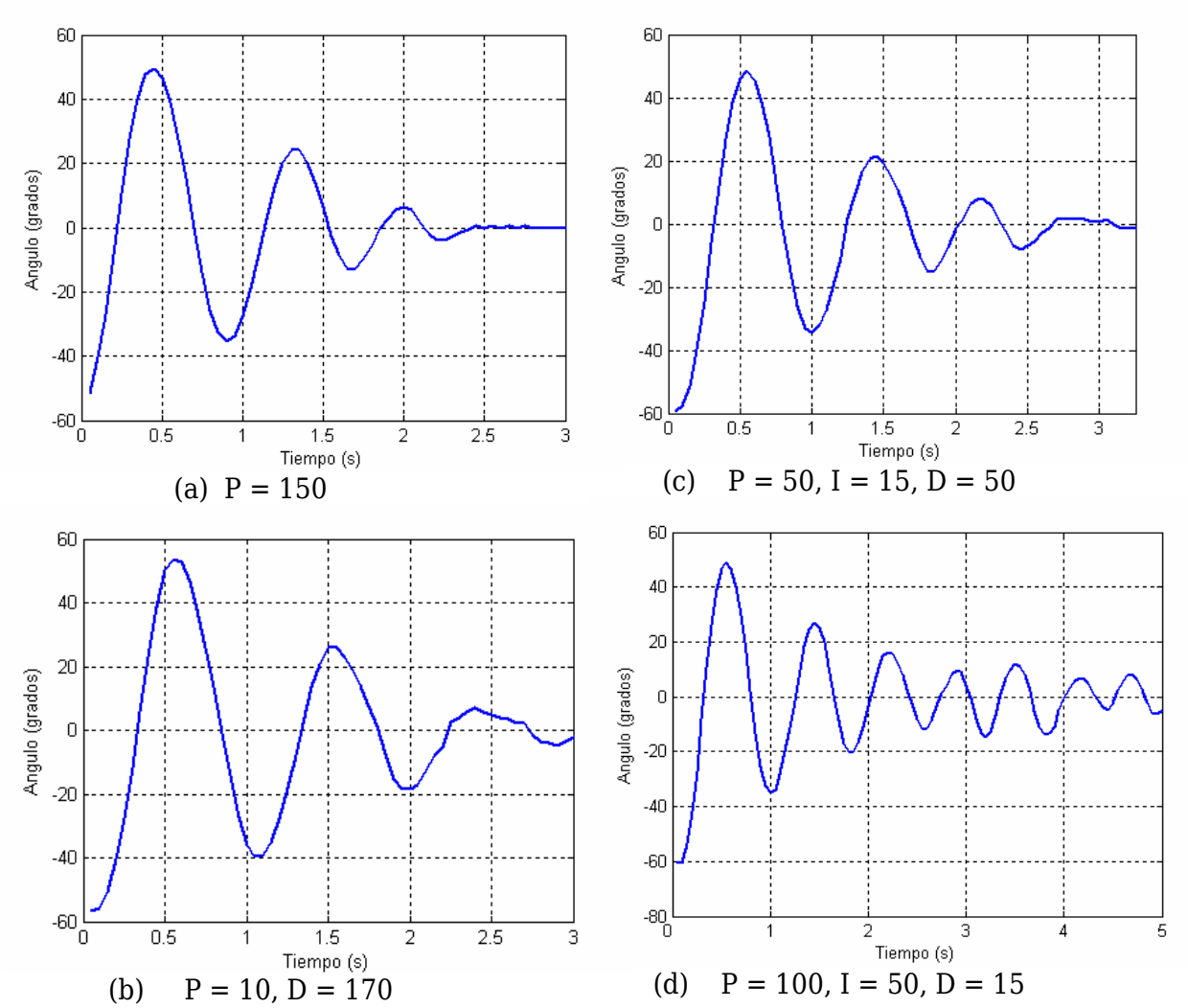


Figura 7.9. Gráficas de respuestas mejorables en el control del péndulo, con distintos valores para el PID

Volviendo a la gráfica de valores candidatos, la mejor respuesta obtenida fija los valores del controlador en P = 100, D = 130. De todas formas, pequeños cambios a este valor no se modifica casi nada la respuesta del sistema, como ocurría en simulación.

Se observa para esta curva un tiempo de establecimiento aproximado de 1.5 segundos, valor que no está nada mal si tenemos en cuenta que el péndulo sin controlar tardaba 30 segundos en detenerse por completo. Las otras tres curvas muestran valores para proporcional y derivativo, o bien demasiado elevados o bien demasiado pequeños, caracterizadas por excesivo tiempo de establecimiento y poca atenuación de la sobreoscilación.

Una vez elegido el PD, surgen dos pruebas más para cerciorarnos de que estos valores para el regulador son los adecuados. Consisten respectivamente en controlar el péndulo una vez está oscilando libremente (Figura 7.7), y en darle un impulso al péndulo una vez quieto para ver la respuesta del coche (Figura 7.8). En estos casos se representa la señal de control y la respuesta del sistema.

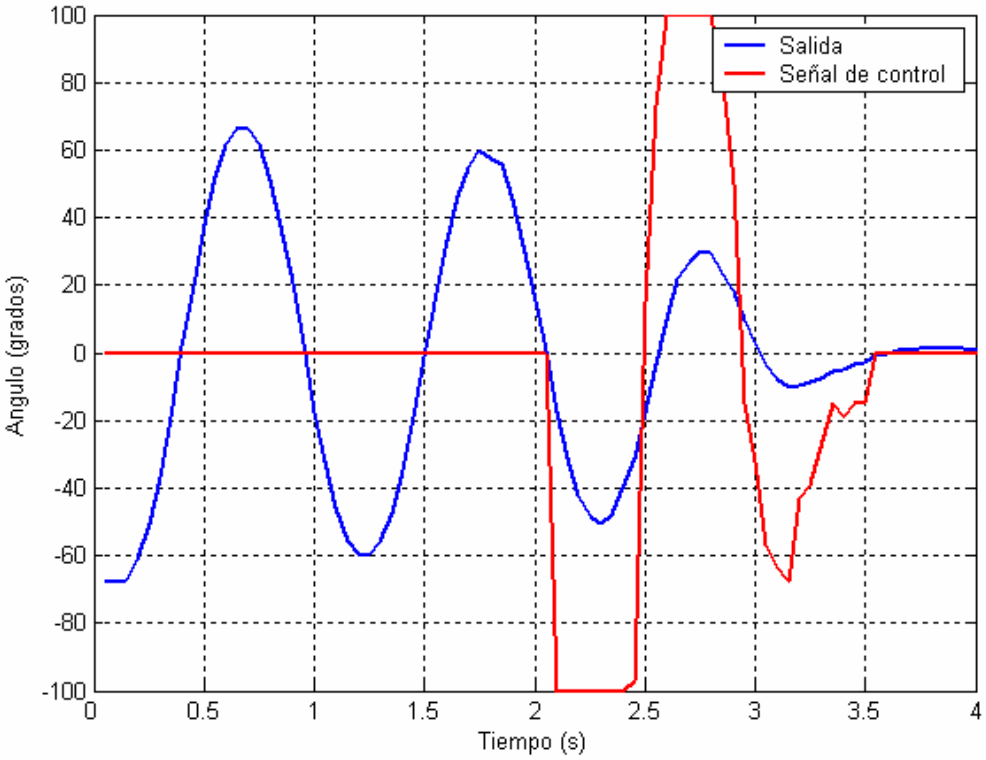


Figura 7.10. Gráfica de la señal de control y la respuesta del sistema para un balanceo arbitrario previo.

En la Figura 7.7, el péndulo oscila libremente hasta los 2 segundos, momento en que comienza el bucle de control. La respuesta a partir de ese momento es casi la misma que la vista anteriormente, con un tiempo aproximado de establecimiento de 1.5 segundos. Los pequeños picos en la señal de control se deben al uso del término derivativo, momentos en que la salida de éste término aumenta o disminuye rápidamente al variar la pendiente de la curva de salida. Para este sistema, estos picos no afectan casi nada la respuesta, aunque en otras aplicaciones habría que tener precauciones ante estos pequeños cambios bruscos en la señal de control.

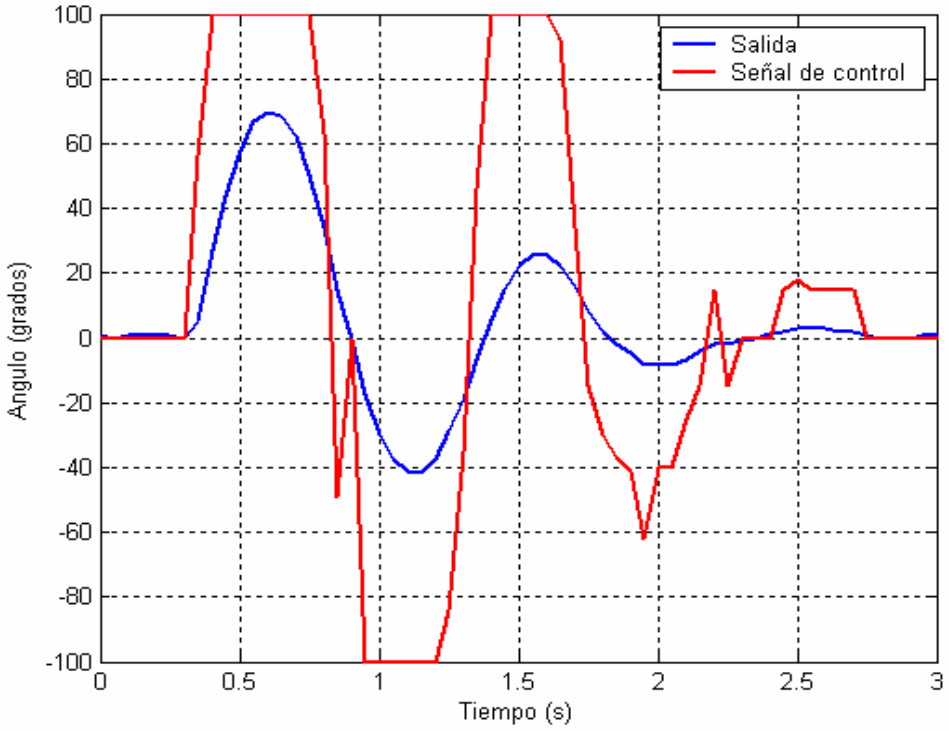
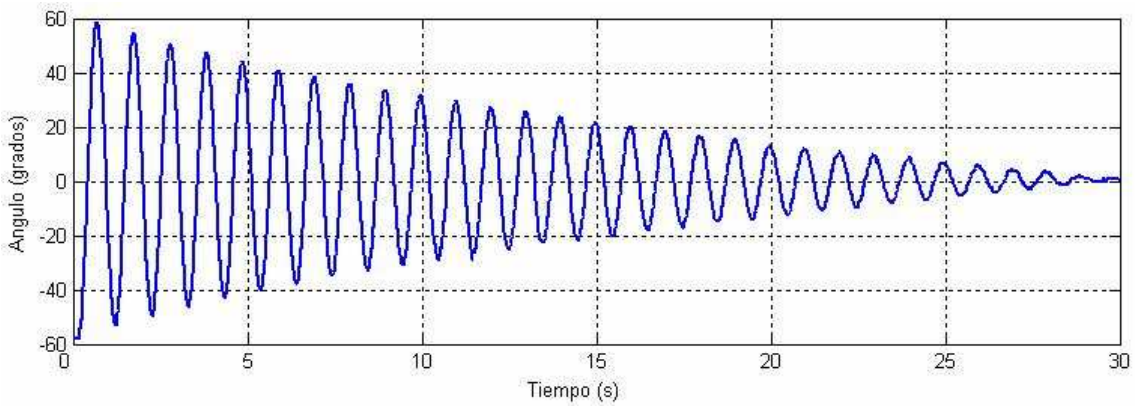


Figura 7.11. Gráfica de la señal de control y la respuesta del sistema para un impulso inesperado al péndulo.

En la Figura 7.8. se representa un impulso al péndulo desde parado. La primera oscilación es de algo más de 60 grados, por lo que el tiempo de establecimiento también se alarga hasta unos 2 segundos. El funcionamiento es correcto, aunque de nuevo tenemos unos picos que corresponden al incremento transitorio de la salida del término derivativo en los puntos de máxima inclinación de la curva. Nótese que la señal de control cambia de signo antes de que lo haga el error, produciéndose el comentado efecto de "adelanto". Los picos, que se traducen en impulsos puntuales a los motores, no afectan prácticamente al movimiento del vehículo (pues estos no tienen físicamente casi tiempo para reaccionar), por lo que podemos dar la señal de control por buena.

Finalmente comparemos la oscilación del péndulo sin y con control. El control se lleva a cabo con el PD comentado. Se comprueba que se cumplen perfectamente los objetivos buscados. Nótese que el controlador comienza a actuar a los 5 segundos.

Sin control:



Con control (activado a los 5 segundos):

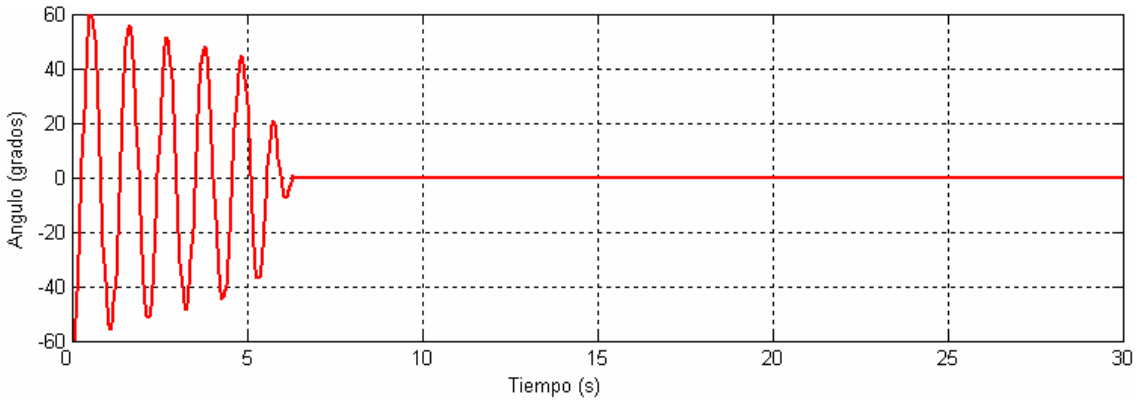


Figura 7.12. Comparativa de la oscilación del péndulo controlada y no controlada.

7.3. Control simultáneo de la posición del carro y el balanceo del péndulo

Este último experimento que se llevará a cabo incluirá el control simultáneo de las dos variables anteriores: la posición del vehículo y la oscilación del péndulo.

7.3.1. Descripción del sistema

Dada una consigna de posición, y partiendo del péndulo inmóvil, el vehículo tratará de alcanzar esa posición sin producir la oscilación en el péndulo, intentando frenarlo una vez se dirige a la posición deseada.

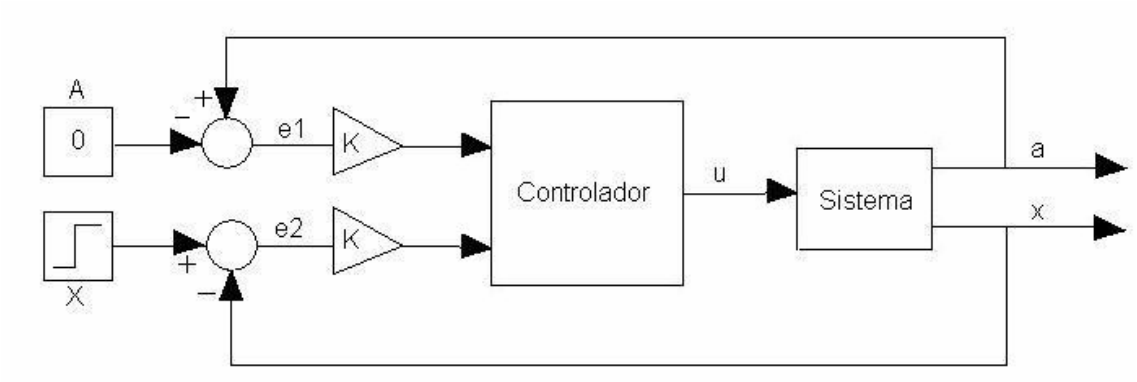
Habrá por lo tanto dos entradas al sistema:

- El ángulo de equilibrio del péndulo (A), que será una constante e igual al ángulo inicial.

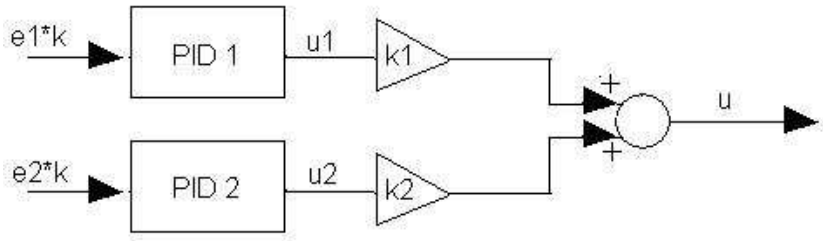
- La posición deseada del coche (X), que es especificada por el usuario y que se puede considerar una entrada escalón. Se podrá especificar tanto en posiciones positivas (mover hacia delante), como negativas (mover hacia atrás).

La idea fue entonces considerar dos subsistemas dentro del sistema completo, que son el encargado del control de posición y el encargado del control del péndulo, por lo que también existirán dos salidas distintas: posición actual (x) y ángulo actual (a). Cada uno de estos subsistemas contará con su regulador PID, y proporcionaran una salida de control conjunta que se convertirá en el movimiento final del vehículo.

La forma más sencilla de ver todo esto es representar el diagrama de bloques del sistema completo. Teniendo como base la estructura de los diagramas de bloques para los experimentos anteriores, se diseñó el siguiente esquema:



(a) Diagrama de bloques



(b) Controlador

Figura 7.13. Diagrama de bloques del controlador de posición y de ángulo (a), y ampliación del bloque Controlador (b).

En este caso hay dos señales de error (e1 y e2), cada una correspondiente a una variable a controlar. Estas señales, al igual que en casos anteriores, se multiplican por una ganancia establecida en K=0.05, para pasarlas a una magnitud adecuada al controlador como se explicó anteriormente.

El controlador estará formado a su vez por dos reguladores PID (PID1 y PID2) que actuarán en paralelo, dedicados al control del péndulo y al control de la posición respectivamente. Cada uno recibirá su señal de error correspondiente, y también cada uno generará una señal de control (u1 y u2) según sus parámetros. Finalmente, la suma de ambas salidas de control será la señal de control final (u) que irá a parar a los motores (sistema).

Como se puede apreciar en el diagrama, también se han añadido dos ganancias adicionales (k1 y k2), que multiplican a las señales de control que proporcionan los PID. Su finalidad es dar más peso a una u otra parte del controlador, es decir, dar preferencia al control de posición o al control del péndulo. Ambas ganancias tendrán valor unitario si queremos que ambos controles tengan la misma importancia. Si en lugar de eso queremos, por ejemplo, dar más prioridad al control del péndulo (que será lo habitual), incrementaremos k1 y disminuiremos k2 la misma cantidad. En principio, k1 = 1.5 y k2 = 0.5 parecen valores aceptables, aunque podrían ajustarse mejor experimentalmente. Entonces la señal de control final se puede expresar en función de las salidas de los dos controladores:

$$u = k1 \cdot u1 + k2 \cdot u2$$

Estamos ante un sistema de control que engloba a los dos anteriores, y que por tanto mantiene todas las características comentadas para cada uno de ellos. No se ha realizado por tanto una tarea de simulación concreta para este experimento, ya que se considera una unión de las realizadas en apartados anteriores.

7.3.2. Sintonía del PID experimentalmente

Al englobar este experimento a los dos ya realizados, no fue problema la elección de los controladores: en principio simplemente serían los mismos que en los casos anteriores, pudiéndose hacer más tarde algún ajuste en el caso de que fuese necesario. Sólo había que pensar ahora en implementar correctamente el diagrama de bloques y la elección de las ganancias k1 y k2. El código utilizado para este experimento corresponde al archivo “ControlPenPos.c” (ver Capítulo 8).

En principio el regulador de posición consistiría en una ganancia proporcional de valor 150, y el regulador de la oscilación del péndulo sería un PD con P = 100 y D = 130.

Una pequeña precaución se debe tener para el caso en que se decida incluir término integral en el controlador de posición: si se mantiene el péndulo inclinado hacia un lado manualmente, el vehículo intenta corregir esa desviación moviéndose indefinidamente hacia ese lado. Sin embargo, llega un momento en que la suma de errores de posición es tan grande, que la señal para controlar esta posición es mayor y el sistema “olvida” el control del péndulo. Esto hace que el vehículo cambie de sentido y se centre en corregir la posición. Esto ocurrirá aunque le hayamos dado más prioridad al control del péndulo mediante la ganancia correspondiente. Para solucionar este problema, se debe disminuir o anular el término integral en este controlador.

El problema fundamental encontrado en este experimento es el **derrape** de las ruedas. La medida de distancia se basa en el valor devuelto por el encoder. Este valor es alterado por el derrape sin conseguir un desplazamiento real del carro. Aunque este problema no era demasiado importante en los anteriores experimentos, se acentúa más aquí si el péndulo está oscilando previamente. Esto se debe a que el carro deberá moverse adelante y atrás para controlar el péndulo, derrapando mucho y alterando la medida real de distancia para el control de posición.

Por esta razón la posición de consigna no será alcanzada adecuadamente cuando se controle el péndulo previamente. Sin embargo, se conseguirá bien cuando se comience el experimento con el péndulo quieto, ya que no habrán cambios de dirección bruscos previos. Precisamente en esto último consistirá el experimento. El péndulo deberá partir desde su posición de equilibrio, y el carro se desplazará evitando la oscilación del péndulo.

Para mejorar un poco este comportamiento, se ha reducido la velocidad máxima del vehículo al 70% de potencia. Aunque esto puede alargar los tiempos de establecimiento, mejorará un poco el valor real de posición alcanzado, debido a que la rueda derrapará menos.

Se eligió un valor de consigna para los experimentos de 40 centímetros, y un periodo de muestreo de 50 milisegundos. El mejor resultado se ha obtenido con los siguientes parámetros:

Controlador de posición: P = 150
Controlador del péndulo: P = 100 D= 70
Ganancia de prioridad de la posición (k2): 0.5
Ganancia de prioridad del péndulo (k1): 1.5

En la Figura 7.14 se representan la posición, el ángulo y la señal de control para el citado experimento. Se recuerda que la máxima tensión se ha fijado en un 70%.

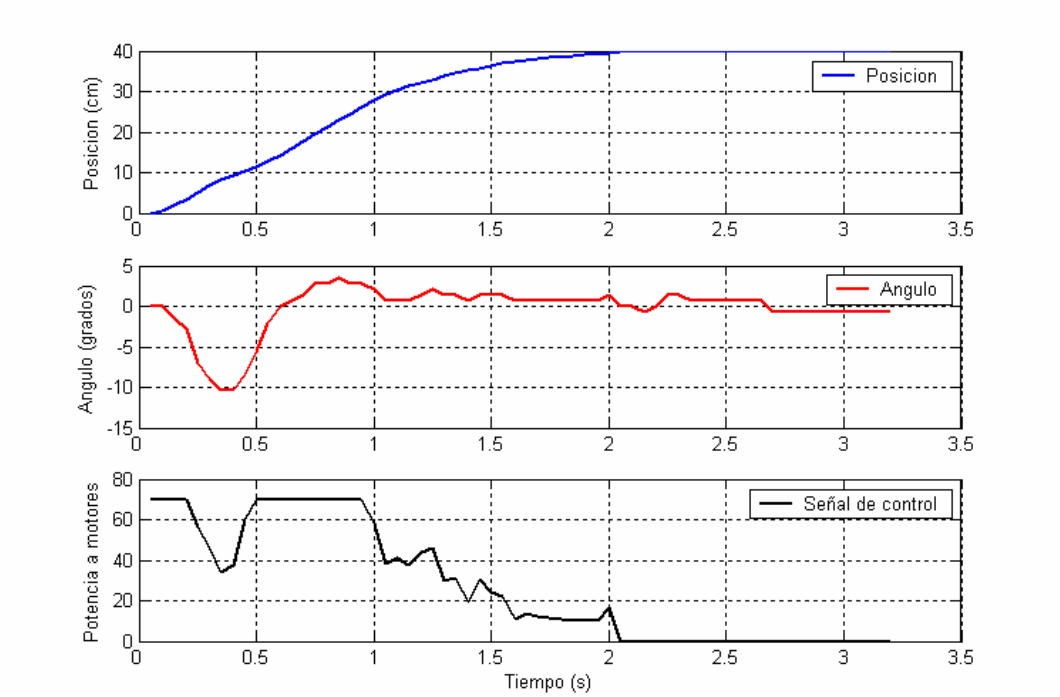
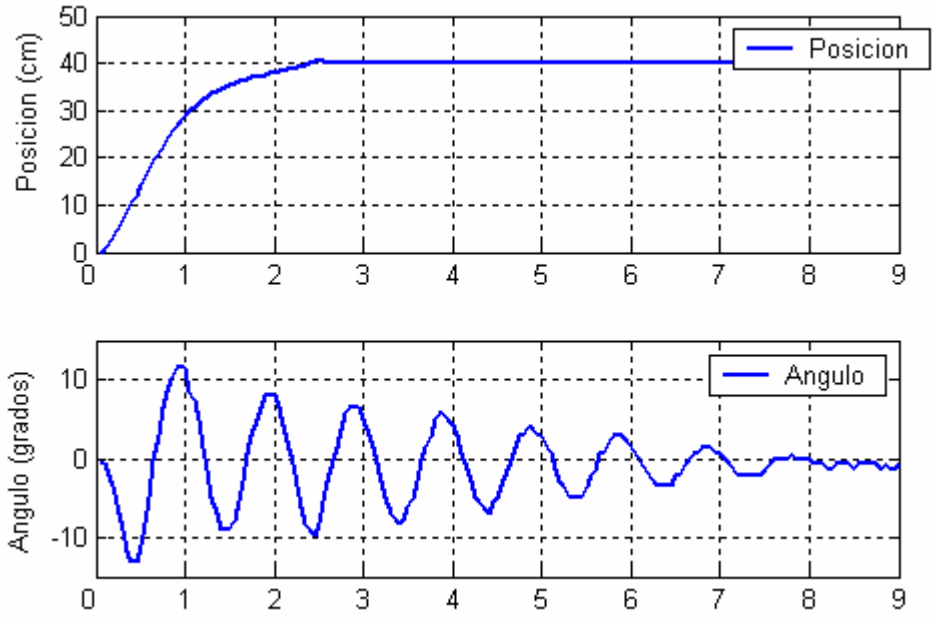


Figura 7.14. Resultado del experimento de control de la posición del carro y la oscilación del péndulo.

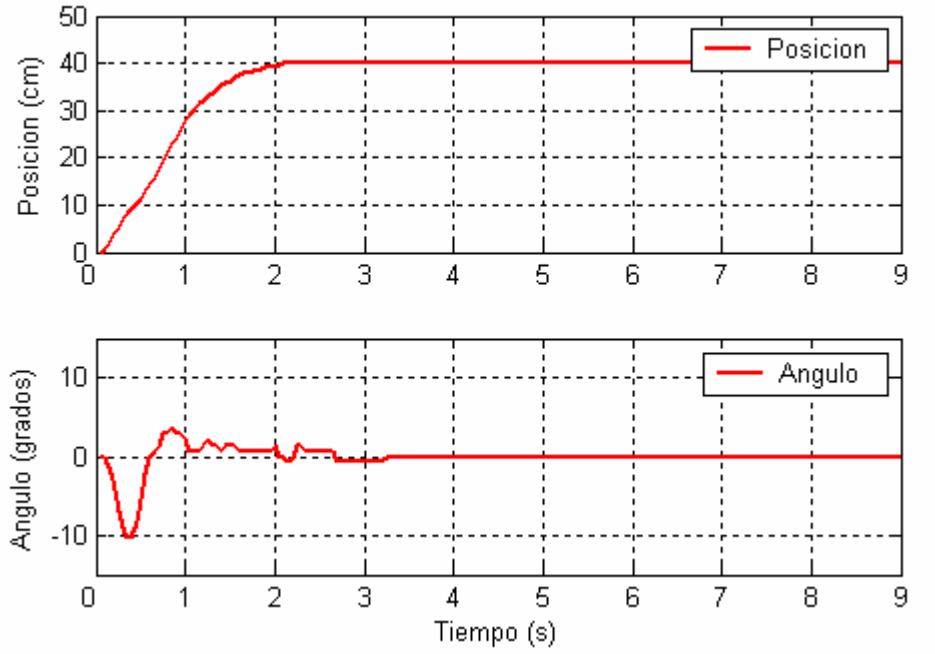
Son unos buenos resultados. Como se ve, se han obtenido unos valores para los controladores muy similares a los obtenidos en los anteriores experimentos. La única diferencia es que el término derivativo del control del péndulo se ha disminuido a un valor de 70. La razón es que el movimiento del carro produce ruido en el péndulo. En ocasiones este ruido era amplificado por la acción derivativa, lo que producía una señal de control algo brusca.

De la Figura 7.14 obtenemos alguna información. El péndulo sólo oscila al comienzo del balanceo. A partir de ahí el carro controla bien esta oscilación. El tiempo de llegada a la posición de consigna es de unos 2 segundos, mientras que en el experimento de posición (Apartado 7.1) se obtuvo un tiempo de 1.25 segundos. Este resultado es comprensible ya que hay que controlar simultáneamente la oscilación. Debido al derrape la distancia real recorrida esta en torno a unos 35 centímetros.

Para finalizar, una buena forma de comprobar el funcionamiento de este experimento es compararlo con otro en el que no se ha controlado la oscilación del péndulo (Ganancia k1 = 0). La comparación se puede observar en la Figura 7.15. Se puede apreciar que el experimento es correcto.



(a) Resultados del experimento sin controlar el péndulo



(b) Resultados del experimento controlando el péndulo

Figura 7.15. Gráficas de resultados del experimento de posición. Comparación del péndulo controlado y el péndulo sin controlar.

CAPÍTULO 8

PROGRAMACIÓN E IMPLEMENTACIÓN DEL PID

8.1. Implementación de un PID en RobotC

La expresión general de la señal de control que nos proporciona un PID en tiempo continuo se puede poner como:

$$u(t) = k_p e(t) + k_i \int_0^t e(t) + k_d \frac{de(t)}{dt}$$

donde kp, ki y kd (que para abreviar llamaremos P, I y D) corresponden a los parámetros del controlador, y *e(t)* es la señal de error. La salida *u(t)* del controlador será la suma de las aportaciones individuales de cada término.

Para trabajar en un entorno discreto como es RobotC, habrá que aproximar esta ecuación para cada instante de tiempo, que llamaremos k.

- Para la parte proporcional, la salida en el instante *k* será el error en ese instante, multiplicado por una constante (P).

$$u_p(k) = P \cdot e(k)$$

- Para la parte integral, podemos aproximar la integral por una suma, por lo que la salida de éste término en el instante de tiempo *k* será el sumatorio de todos los errores en instantes anteriores, multiplicado por la constante I.

$$u_I(k) = I \cdot \sum e(k)$$

- Para la parte derivativa, la derivada del error en un instante de tiempo *k* se puede aproximar por la diferencia entre el error en ese instante y el error en el instante de tiempo anterior. La salida de éste término del controlador en ese instante será esa diferencia multiplicada por la constante D.

$$u_d(k) = D \cdot (e(k) - e(k-1))$$

La salida final del PID en un instante será $\boldsymbol{U(k)=U_p+U_i+U_d}$. Por tanto habrá que calcular esta salida de control de la forma mencionada para cada instante de tiempo, definido por el periodo de muestreo que estemos utilizando. Para todos los programas de control de este proyecto, el periodo de muestreo se ha seleccionado de 50 milisegundos.

Esto se traduce en RobotC a unas simples líneas de código, como se muestra a continuación.

Para un instante de tiempo cualquiera:

```
salidaP=error*P;
//Añadimos el error actual a la suma de errores, para calcular la salida integral
errorsuma+=error;
salidaI=errorsuma*I;
//El error previo debe haber sido almacenado antes
salidaD=(error-errorprevio)*D;
salidacontrolador=(salidaP+salidaI+salidaD);
```

Por supuesto todas las variables deben estar declaradas antes. Aunque este segmento de código es la base del controlador, habrá que añadirle algunas operaciones más, como incluir una ganancia (para disminuir la salida del controlador que normalmente será muy grande), o acotar la salida final de entrada a los motores entre unos valores determinados. Habrá que tener en cuenta el experimento que estemos diseñando. Por otro lado, la velocidad de procesamiento del NXT es mucho mayor que el periodo de muestreo, por lo que el programa no tiene ningún problema para realizar todos estos cálculos.

8.2. Comentarios a los programas de control

Aunque todos los programas incluidos en el Anexo A incluyen sus comentarios, merece la pena explicar más detenidamente algunos aspectos de los tres programas fundamentales: los de control. Estos también pueden encontrarse en la carpeta “Códigos desarrollados en el proyecto”, dentro del CD.

8.2.1. Control de posición del carro

Se trata del código *“ControlPosicion.c”*. La programación de este experimento se presentó como la más sencilla. El funcionamiento es simple: tras pulsar el botón central del NXT para comenzar, se calcula el error en centímetros respecto de la posición deseada seleccionable por el usuario. Para ello se toma una muestra del encoder y se convierte a centímetros mediante la ecuación correspondiente a la rueda que estamos usando. El valor se compara con el de consigna. A partir de este error se calcula la señal de control correspondiente mediante el PID, y se modifica con la ya mencionada ganancia k=0.05.

Esta señal podrá ser mayor de 100, tanto de signo positivo como negativo. Sin embargo, los motores no aceptan valores mayores de esa magnitud. Para realizar el ajuste, se incluye un segmento que acota la señal de control entre 100 (valor máximo) y 10 (mínimo valor que produce movimiento en el vehículo). Finalmente, si el error supera la tolerancia que hayamos seleccionado, el vehículo se moverá adelante o atrás dependiendo del signo de la señal de control calculada, y tras esto se deja un tiempo de espera y se repite el bucle indefinidamente.

Pequeños defectos como el derrape de las ruedas sobre distintas superficies pueden hacer que en ocasiones la distancia recorrida por el vehículo no sea exactamente la buscada, aunque generalmente este error será despreciable.

```
/*****************************************************************************
Autor: Ismael Sanchez Mendoza
Proyecto Fin de Carrera: Diseño y control de un prototipo carro-péndulo basado en LEGO Mindstorms NXT.
Universidad Politecnica de Cartagena, 2009.

    Codigo para el control de la posicion del vehiculo, sin tener en
    consideracion el balanceo del pendulo. Es ajustable la posicion de
    consigna deseada del coche. El sistema se basa en un regulador PID
    de parametros ajustables.
```

```
**********************************************************************/

task main()
{
//Posicion a la que debe dirigirse el vehiculo, en centimetros.
//Valores positivos y negativos indican hacia adelante o hacia
//atras respectivamente.
float posiconsigna = 30;

//Parametros del regulador
int P = 150;
int I = 0;
int D = 0;

//Ganancia del sistema
float K = 0.05;
//Introducimos una tolerancia para los encoders en centimetros
float tolerancia = 0.5;
//Tiempo de espera entre bucles, en milisegundos
int tiempoespera = 50;

//Variable para almacenar la posicion actual del coche en cm
float posicion = 0;
//Variabla para almacenar la desviacion respecto a la posicion buscada
float error = 0;
//Variable para almacenar el error en un instante de tiempo anterior
float errorprevio = 0;
//Variable para almacenar la suma de todos los errores anteriores
float errorsuma = 0;

//Salida que nos proporcionara cada uno de los terminos del controlador
float salidaP = 0;
float salidaI = 0;
float salidaD = 0;
//Suma de las 3 salidas anteriores y teniendo en cuenta la ganancia
float salidatotal = 0;

//Variable para almacenar el valor final que entrara a los motores, una vez
//hecha la correcion.
int final = 0;

//Esperamos a pulsar el boton central para comenzar
nxtDisplayTextLine(2,"Pulsa boton");
nxtDisplayTextLine(3,"central para");
nxtDisplayTextLine(4,"iniciar");
until(nNxtButtonPressed==3);
        //Borramos el display
        eraseDisplay();
        //Reproducimos cuenta atras con sonidos
        PlayTone(400, 20);
```

```
	wait1Msec(1000);
	PlayTone(400, 20);
	wait1Msec(1000);
	PlayTone(600, 40);
	wait1Msec(700);

//Reseteamos el encoder
nMotorEncoder[motorB] = 0;
//Sincronizamos los motores, el derecho(C) esclavo del izquierdo(B)
nSyncedMotors = synchBC;


//Codigo principal para el seguimiento de consigna corrigiendo la posicion del vehiculo
while(true)
	{
		//Calculamos la posicion actual en centimetros a partir del valor del encoder.
		posición = (nMotorEncoder[motorB]*56*PI)/3600;

		//Hallamos el error correspondiente
		 errorprevio = error;
		 error = posiconsigna - posicion;

		 //Salidas que nos proporciona cada termino del controlador
		salidaP = error*P;
		 errorsuma += error;
		salidaI = errorsuma*I;
		salidaD = (error-errorprevio)*D;
		//Salida total, al multiplicar por la ganancia
		salidatotal = K*(salidaP+salidaI+salidaD);


//Segmento para corregir el valor de entrada a los motores. No se permiten valores mayores
//de 100, ni menores de 10, ya que este es el minimo valor capaz de mover el coche
		 if (abs(salidatotal) > 100)
			final = 100;
		else if(abs(salidatotal) < 10)
			final = 10;
		else
			final = abs(salidatotal);

//Segmento para mover el coche. Si el error supera la tolerancia, se movera adelante o atras
//dependiendo del signo de la senal de control. La variable "final" de entrada a los motores,
//estara comprendida entre 10% y 100%, pudiendo ser de signo + o -.
		 if (abs(error) > tolerancia)
		{
		if (salidatotal > 0)
			motor[motorB] = final;
		else
			motor[motorB] = (-1)*final;
		}
```

```
                else
                        //Reseteamos la suma de errores
                        errorsuma=0;

                  //Damos tiempo a que los motores se muevan
                 wait1Msec(tiempoespera);
                //Paramos los motores
                motor[motorB]=0;
        }
}
```

8.2.2. Control de oscilación del péndulo

Incluido en el archivo *"ControlPendulo.c"*. El funcionamiento de este experimento es casi igual al anterior, pero con algunas diferencias a tener en cuenta.

Ahora el valor del ángulo de reposo del péndulo, que se convertirá en la consigna, se consigue con una primera pulsación del botón central del NXT, por lo que se precisa que el péndulo esté quieto al pulsar este botón. Esta forma de establecer el valor de consigna se debe a que pequeñas imperfecciones o desajustes en la estructura o ejes pueden causar que el ángulo de consigna varíe de una ejecución a otra, por lo que conviene recoger este valor cada vez que se vaya a iniciar el programa.

El valor Raw obtenido en las mediciones se transforma en ángulo mediante la ecuación correspondiente al potenciómetro utilizado. A partir de ahí, todo es prácticamente igual. Se compara con la consigna (en ángulo), se obtiene el error, se obtiene la señal de control con el PID, y se mueve el coche en consecuencia. Tanto la ganancia K, como los segmentos de código para acotar la señal de control y para el movimiento del coche, son idénticos a los del experimento anterior.

```
/*********************************************************************
Autor: Ismael Sanchez Mendoza
Proyecto Fin de Carrera: Diseño y control de un prototipo carro-péndulo basado en LEGO
Mindstorms NXT.
Universidad Politecnica de Cartagena, 2009.

    Codigo para el control de la oscilacion del pendulo. El sistema cuenta
    con una ganancia K y un regulador PID de parametros ajustables.
    Se intentara mantener el valor de consigna inicial que corresponde
    al pendulo inmovil.
*********************************************************************/
```

```
task main()
{
//Terminos del regulador
int P = 100;
int I = 0;
int D = 130;
//Ganancia para mejorar el comportamiento del sistema
float K = 0.05;
//Tolerancia para el sensor de angulo, en grados
int tolerancia = 2;

//Tiempo de espera entre bucles, en milisegundos
int tiempoespera = 50;
//Variable que almacena el valor raw devuelto por el potenciómetro
float raw = 0;
//Variable para almacenar el valor de equilibrio inicial
float anguloinicial = 0;
//Variable para almacenar el angulo girado por el pendulo
float angulo = 0;
//Variable para almacenar el error de angulo
float error = 0;
//Variable para almacenar el error en un instante de tiempo anterior
float errorprevio = 0;
//Variable para almacenar la suma de todos los errores anteriores
float errorsuma = 0;

//Salida que nos proporcionara cada uno de los terminos del controlador
float salidaP = 0;
float salidaI = 0;
float salidaD = 0;
//Suma de las 3 salidas anteriores
float salidatotal = 0;
//Variable para almacenar el valor final que entrara a los motores, una vez hecha la correccion.
int final = 0;


//Segmento para inicializar el angulo de reposo que sera la consigna
nxtDisplayTextLine(2,"Pulsa boton");
nxtDisplayTextLine(3,"central para");
nxtDisplayTextLine(4,"guardar consigna");
//Esperamos a que se pulse el boton central(3)
until(nNxtButtonPressed==3);
        //Borramos el display
        eraseDisplay();
        //Reproducimos un sonido de aviso
         PlayTone(400, 20);
        //Leemos la entrada
         raw = SensorRaw[S1];
        //Inicializamos el valor del angulo mediante la ecuación correspondiente al
        //potenciometro utilizado
```

```
		 anguloinicial = (34500/((109461/raw)-107));
		//Damos tiempo a soltar el boton
		wait1Msec(600);
//Esperamos a pulsar de nuevo el boton central para empezar
nxtDisplayTextLine(2,"Pulsa boton");
nxtDisplayTextLine(3,"central para");
nxtDisplayTextLine(4,"iniciar");
until(nNxtButtonPressed==3);
		eraseDisplay();
		//Reproducimos una cuenta atras con sonidos
		PlayTone(400, 20);
		wait1Msec(1000);
		 PlayTone(400, 20);
		wait1Msec(1000);
		PlayTone(600, 40);
		wait1Msec(700);

//Sincronizamos los motores, el derecho(C) esclavo del izquierdo(B)
nSyncedMotors = synchBC;

//Codigo principal para el seguimiento de la consigna corrigiendo la posicion del vehiculo
while(true)
		{
			//Leemos la entrada
			 raw = SensorRaw[S1];

			//Obtenemos el angulo correspondiente
			angulo = (34500/((109461/raw)-107));

			 //Hallamos la desviacion respecto a la posicion inicial
			errorprevio = error;
			error = angulo - anguloinicial;

			 //Salidas que nos proporciona cada parte del controlador
			 salidaP=error*P;
			errorsuma+=error;
			 salidaI=errorsuma*I;
			 salidaD=(error-errorprevio)*D;
			//Salida completa teniendo en cuenta la ganancia K
			salidatotal=K*(salidaP+salidaI+salidaD);

//Segmento de codigo para corregir la entrada a los motores. No se permiten valores mayores de
//100, ni menores de 10, ya que este es el minimo valor capaz de mover el coche
//apreciablemente
			 if (abs(salidatotal) > 100)
				final = 100;
			else if(abs(salidatotal) < 10)
				final = 10;
			else
				final = abs(salidatotal);
```

```
//Segmento para mover el coche. Si el error supera la tolerancia, se movera adelante o atras
//dependiendo del signo de la senal de control. La variable "final" de entrada a los motores,
//estara comprendida entre 10% y 100%, con signo - o +.
                if (abs(error)>tolerancia)
                        {
                        if (salidatotal>0)
                                motor[motorB] = final;
                        else
                                motor[motorB] = (-1)*final;
                        }
                 else
                        //Reseteamos el error suma
                        errorsuma=0;

                //Damos tiempo a que los motores se muevan
                 wait1Msec(tiempoespera);
                //Paramos los motores
                 motor[motorB]=0;
        }
}
```

8.2.3. Control simultáneo de posición del carro y oscilación del péndulo

Correspondiente al código "*ControlPosPen.c*", este programa es muy parecido a los dos anteriores. Básicamente es la unión de ambos en uno solo, por lo que se utilizan dos controladores PID. Los segmentos de recogida de datos y de cálculo de las señales de control para cada controlador, tanto el de posición como el del péndulo, son idénticos a los anteriores. La única diferencia es que ahora cada señal de control está modificada por una ganancia particular (Kpos y Kpen), que servirá para dar prioridad a un controlador u otro. Habitualmente el control del péndulo deberá ser prioritario (Kpen > Kpos).

La señal de control final será la suma de ambas salidas ya modificadas. En este experimento el vehículo se moverá si alguno de los dos errores (posición o ángulo) supera su tolerancia. El tiempo de bucle, el acotamiento de la señal de control y el segmento para el movimiento del coche son iguales que en casos anteriores.

```
/*****************************************************************************
Autor: Ismael Sanchez Mendoza
Proyecto Fin de Carrera: Diseño y control de un prototipo carro-péndulo basado en LEGO
Mindstorms NXT.
```

Universidad Politecnica de Cartagena, 2009.

Codigo para controlar simultaneamente tanto la posicion del vehiculo como la oscilacion del pendulo. El coche tratara de alcanzar la posición indicada intentando que el pendulo oscile lo menos posible. El sistema cuenta con dos reguladores PID de parametros ajustables, uno para cada una de las tareas de control. Ambas senales de control quedan modificadas por un multiplicador (Kpen y Kpos), que servira para asignar mayor o menor importancia al seguimiento de la posicion o al control del pendulo.

**********/

```
task main()
{
//PARAMETROS AJUSTABLES
//Posicion a la que debe dirigirse el vehiculo, en centimetros. Valores positivos y negativos
//indican hacia adelante o hacia atras respectivamente.
float posiconsigna = 40;

//Terminos del regulador de la posicion
int Ppos = 150;
int Ipos = 0;
int Dpos = 0;

//Terminos del regulador del pendulo
int Ppen = 100;
int Ipen = 0;
int Dpen = 70;

//Ganancia para ambos reguladores
float K = 0.05;

//Ganancias adicionales para dar prioridad al seguimiento de posicion o al control del angulo.
//Normalmente debe tener mas peso el control del pendulo
float Kpos = 0.5;
float Kpen = 1.5;

//Tolerancias para el encoder y para el pendulo respectivamente.
float tolpos = 0.5;
float tolpen = 2;

//Tiempo de bucle, en milisegundos
int tiempoespera = 50;

//VARIABLES PARA RECOGIDA DE DATOS EXTERNOS
//Variable para almacenar la posicion actual del coche, en cm
float posicion = 0;
//Variable que almacena el valor raw devuelto por el potenciómetro
float lectura = 0;
//Variable para almacenar el angulo actual del pendulo en grados
float angulo = 0;
```

```
//Variabla para almacenar el valor de reposo inicial
float anguloinicial = 0;

//VARIABLES DE ERROR
//Error en la posicion
float Epos = 0;
//Error de posicion un instante anterior
float Epreviopos = 0;
//Suma de todos los errores de posicion anteriores
float Esumapos = 0;

//Error en el angulo del pendulo
float Epen = 0;
//Error angular un instante de tiempo anterior
float Epreviopen = 0;
//Suma de todos los errores angulares anteriores
float Esumapen = 0;

//SALIDAS DE LOS CONTROLADORES
//Salida que nos proporcionara cada uno de los terminos del controlador POS
float UPpos = 0;
float UIpos = 0;
float UDpos = 0;
//Salida total del controlador POS
float Utotalpos = 0;

//Salida que nos proporcionara cada uno de los terminos del controlador PEN
float UPpen = 0;
float UIpen = 0;
float UDpen = 0;
//Salida total del controlador PEN
float Utotalpen = 0;

//Suma de las salidas de los dos controladores, POS Y PEN, previamente modificadas por sus
//multiplicadores
float Usuma = 0;
//Valor final de potencia que se suministrara a los motores
float Ufinal = 0;

//Segmento de codigo para inicializar el angulo de reposo que sera la consigna
nxtDisplayTextLine(2,"Pulsa boton");
nxtDisplayTextLine(3,"central para");
nxtDisplayTextLine(4,"guardar consigna");
//Esperamos a que se pulse el boton central(3)
until(nNxtButtonPressed==3);
        //Borramos el display
        eraseDisplay();
        //Reproducimos un sonido de aviso
         PlayTone(400, 20);
        //Leemos la entrada
```

```
		lectura = SensorRaw[S1];
		//Inicializamos el valor del angulo mediante la ecuación correspondiente al
		//potenciometro utilizado
		anguloinicial = (34500/((109461/lectura)-107));
		//Tambien reseteamos el encoder
		nMotorEncoder[motorB] = 0;
		//Damos tiempo a soltar el boton
		wait1Msec(600);

//Esperamos a pulsar de nuevo el boton central para comenzar
nxtDisplayTextLine(2,"Pulsa boton");
nxtDisplayTextLine(3,"central para");
nxtDisplayTextLine(4,"iniciar");
until(nNxtButtonPressed==3);
		eraseDisplay();
		//Reproducimos una cuenta atras de 3 segundos con sonidos
		PlayTone(400, 20);
		wait1Msec(1000);
		PlayTone(400, 20);
		wait1Msec(1000);
		PlayTone(600,40);
		wait1Msec(600);

//Sincronizamos los motores, el derecho(C) esclavo del izquierdo(B)
nSyncedMotors = synchBC;

//Bucle infinito para la lectura de valores y la correspondiente correccion de la posicion del
//vehiculo, controlando a su vez la oscilacion del pendulo.
while(true)
		{
//LECTURA DE POSICION Y CALCULO DE SU SALIDA DE CONTROL
			//Calculamos la posicion actual en centimetros a partir del valor del encoder.
			posición = (nMotorEncoder[motorB]*56*PI)/3600;

			//Hallamos el error correspondiente
			Epreviopos = Epos;
			Epos = posiconsigna-posicion;

			//Calculo de salidas de cada termino del controlador
			UPpos=Epos*Ppos;
			Esumapos+=Epos;
			UIpos=Esumapos*Ipos;
			UDpos=(Epos-Epreviopos)*Dpos;
			//Salida completa teniendo en cuenta la ganancia
			Utotalpos=K*(UPpos+UIpos+UDpos);

//LECTURA DE OSCILACION DEL PENDULO Y CALCULO DE SU SALIDA DE
//CONTROL
			//Leemos la entrada
			lectura = SensorRaw[S1];
```

```
		//Obtenemos el angulo correspondiente
		angulo=(34500/((109461/lectura)-107));

		//Hallamos el error respecto a la posicion inicial
		Epreviopen=Epen;
		Epen=angulo-anguloinicial;

		//Calculo de salidas de cada termino del controlador
		UPpen=Epen*Ppen;
		Esumapen+=Epen;
		UIpen=Esumapen*Ipen;
		UDpen=(Epen-Epreviopen)*Dpen;
		//Salida completa teniendo en cuenta la ganancia
		Utotalpen=K*(UPpen+UIpen+UDpen);
//CALCULO DE LA SALIDA FINAL APLICANDO A CADA SALIDA ANTERIOR EL
//MULTIPLICADOR CORRESPONDIENTE
		Usuma=(Kpen*Utotalpen)+(Kpos*Utotalpos);

//AJUSTE FINAL DE LA ENTRADA A MOTORES Y EL CONSECUENTE MOVIMIENTO
//DEL COCHE
//Segmento de codigo para acotar la entrada a los motores entre 100 y 10, ya que este es el
minimo valor capaz de mover el coche apreciablemente
		if (abs(Usuma) > 100)
				Ufinal = 100;
			else if(abs(Usuma) < 10)
				Ufinal = 10;
			else
				Ufinal = abs(Usuma);

//Segmento para mover el coche. Si algun error supera la tolerancia, se movera adelante o atras
//dependiendo del signo de la senal de control “Usuma". La variable Ufinal de potencia a los
//motores, estara comprendida entre 10% y 100%
		if ((abs(Epos) > tolpos) || (abs(Epen) > tolpen))
			{
				if (Usuma>0)
					motor[motorB] = Ufinal;
				else
					motor[motorB] = (-1)*Ufinal;
			 }
		  else  //Reseteamos los errores suma
			{
				Esumapos=0;
				Esumapen=0;
			}

		//Damos tiempo a que los motores se muevan
		wait1Msec(tiempoespera);
		//Paramos los motores
		motor[motorB]=0;
	}
```

```
}
```

CAPÍTULO 9

CONCLUSIONES Y TRABAJOS FUTUROS

9.1. Conclusiones

A lo largo de este proyecto se han llevado a cabo una serie de tareas que son enumeradas a continuación:

- Se ha analizado de cerca la posibilidad de trabajar con el NXT en tareas de control, se han examinado trabajos previos, se ha comprobado que es posible usar el NXT en estas tareas, y se ha decidido diseñar el sistema de control concreto que se ha visto.

- Se ha llevado a cabo la construcción de una maqueta completa consistente en una estructura móvil con péndulo, y basada únicamente en elementos y piezas de LEGO. Se ha creado una guía de montaje para esta maqueta usando el software Lego Digital Designer.

- Se ha elegido el entorno RobotC como el idóneo para los programas del proyecto. Se han estudiado las posibilidades y las características de su lenguaje, con el que se ha trabajado gran parte del tiempo.

- Se ha realizado una tarea de sensorización para obtener los datos de entrada al NXT. Tras el estudio de otras alternativas como usar sensores de LEGO, se ha comprobado que es posible usar un potenciómetro con el NXT, y más en general, cualquier tipo de sensor resistivo. Se ha analizado la forma de conexión y de obtención de datos de estos sensores en el ladrillo.

Además, se ha estudiado el uso de dos potenciómetros en el NXT, sus características y sus curvas, y se ha elegido el más idóneo. También se ha realizado la tarea de montaje y conexión del potenciómetro a la maqueta.

- Se ha hecho una identificación completa del sistema. Mediante la realización de distintos experimentos se han obtenido las relaciones entre las distintas variables, y se han construido las funciones de transferencia y los diagramas de bloques correspondientes.

- Se han diseñado tres experimentos de control basados en reguladores PID, y se ha realizado una tarea de sintonización, tanto en simulación como experimentalmente. Se han hallado los valores adecuados para los controladores de los tres experimentos.

- Se han implementado en RobotC los reguladores PID, y en torno a ellos todos los sistemas de control. Se han introducido estos programas en RobotC y tras proceder a las pruebas necesarias, se han dado por buenos los tres experimentos, confirmando que cumplen los objetivos de control propuestos.

9.2. Trabajos futuros

Una vez concluidos los objetivos que se habían marcado para este proyecto, puede ser interesante continuar desarrollando algunas aplicaciones con el NXT. Algunos de estos posibles trabajos son:

- Estudiar la comunicación Bluetooth entre varios NXT o entre el NXT y un PC. Esto abriría las puertas al control remoto del NXT y a la realización de tareas que requieran varios NXT trabajando simultáneamente. Se pueden plantear gran cantidad de aplicaciones en este campo.

- Implementar un prototipo de plataforma inercial. Este sistema consistiría en un vehículo o plataforma móvil, sobre la que estaría colocada una antena o cañón. El objetivo sería controlar la antena, manteniendola apuntando a un punto del espacio mientras el vehículo se desplaza. Sería interesante usar varios NXT comunicados por Bluetooth, para el movimiento del vehículo y el movimiento de la antena.

- Estudiar otras formas de realimentación de la posición. La medida de posición con los tacómetros tiende a no ser exacta, por el citado derrape. Sería interesante obtener la medida de posición de forma externa, para reducir el error en la medida en otros trabajos. Alguna forma de hacerlo podría ser utilizando algún otro sensor como por ejemplo el de ultrasonidos.

BIBLIOGRAFÍA

1. *Estudio de las posibilidades didácticas en ingeniería de control del LEGO Mindstorms NXT.* PFC. Guillermo Nieves Molina. Universidad Politécnica de Cartagena. 2008.

2. *Extreme NXT.* Michael Gasperi; Philippe Hurbain. Apress. 2007.

3. http://www.mech.gla.ac.uk/~peterg/Lego/, web sobre los experimentos de control basados en LEGO, por Peter Gawthrop y Euan McGookin. Consultada en 2009.

4. *A lego based control experiment.* Peter Gawthrop; Euan McGookin. 2004. Publicado en [3].

5. http://web.mac.com/ryo_watanabe/iWeb/Ryo%27s%20Holiday/NXTway-G.html, web del creador del NXTWay-G, Ryo Watanabe. Consultada en 2009.

6. http://www.lego.com/eng/education/mindstorms/home.asp?pagename=input, web de LEGO Education con información sobre sensores y actuadores del RCX. Consultada en 2009.

7. http://www.legoeducation.com/store/, tienda online con diferentes productos de LEGO. Consulta en 2009.

8. http://www.robotc.net/, web del lenguaje de programación RobotC, con descarga de software, tutoriales, y foro. Consultada en 2009.

9. *Hardware Developer Kit*, Disponible en [10].

10. http://mindstorms.lego.com/Overview/NXTreme.aspx, sección de la web de LEGO, con especificaciones del software, el hardware, y documentos descargables para aplicaciones más avanzadas con el NXT. Consultada en 2009.

11. http://www.hitechnic.com/, web del fabricante de hardware para LEGO HiTechnic. Consultada en 2009.

12. http://ldd.lego.com/, web de descarga del software *Lego Digital Designer*. Consultada en 2009 .

13. http://www.laurentkneip.de/angle_sensor.html, sección de la web de Laurent Kneip referente al uso de potenciómetros como sensores de ángulo en el NXT. Consultada en 2009

14. http://www.extremenxt.com/pot.htm, sección de la web de Michael Gasperi (co-autor de *ExtremeNXT*), donde se explica el uso de un potenciómetro como sensor de rotación.

15. *Lego Technic Tora no Maki.* Isogawa Yoshihito. Documento gratuito con gran cantidad de ejemplos de montajes básicos con piezas de LEGO Technic, como diferenciales, trenes de engranajes y estructuras. Descargable en http://www.isogawastudio.co.jp/legostudio/toranomaki/en/download.html

ANEXO A

Otros códigos desarrollados en el proyecto

Todos los códigos aquí descritos, junto con los tres códigos para las tareas de control (Capítulo 8), se pueden encontrar en la carpeta “Códigos desarrollados en el proyecto”, incluida en el CD.

A.1. Lectura de valores de entrada del potenciómetro

Se trata del código correspondiente al archivo “GuardaRawPulsando.c”.

```
/*************************************************************************
Autor: Ismael Sanchez Mendoza
Proyecto Fin de Carrera: Diseño y control de un prototipo carro-péndulo basado en LEGO Mindstorms NXT.
Universidad Politecnica de Cartagena, 2009.

        Codigo para almacenar el valor Raw devuelto por el potenciometro
        mediante pulsaciones del boton central del NXT.
        Los valores se guardaran en un archivo de texto llamado "Raw pot.txt".
*************************************************************************/

//Variables globales para trabajar con archivos
TFileIOResult resultado; //Flag que nos indica el resultado de la operacion con el archivo
TFileHandle etiqueta;    //Etiqueta del archivo.
int longitud = 5000;     //Longitud del archivo en bytes

task main()
{
    //Tiempo de espera para soltar el botón, en milisegundos
    int tiemposoltar = 500;
    //Variable que almacena el valor devuelto por el potenciómetro
    int raw = 0;
    //Variable para almacenar la lectura una vez ajustada entre 0 y 999
    int rawfinal = 0;
    //String que se escribira en el archivo
```

```
    string dato;

//Borramos cualquier archivo existente con el nombre que queremos
Delete("Raw pot.txt", resultado);
//Creamos el archivo para guardar las lecturas
OpenWrite(etiqueta, resultado,"Raw pot.txt", longitud);
//Escribimos la primera linea de texto en el archivo
WriteText(etiqueta, resultado,"Valores Raw: \r\n");


//Comienza el bucle principal
while(true)
        {
        //Esperamos hasta que se pulse el boton central(3)
        until(nNxtButtonPressed == 3);

        //Reproducimos un sonido de aviso
        PlayTone(400, 20);

        //Leemos la entrada
        raw = SensorRaw[S1];

        //Acotamos las lecturas entre 0 y 999.
           if (raw > 999)
             rawfinal = 999;
              else rawfinal = raw;

        //Convertimos la lectura final en un string y lo escribimos en el archivo
        dato = rawfinal+"\r\n";
         WriteText(etiqueta, resultado, dato);

        //También lo mostramos por pantalla en decimal
        nxtDisplayTextLine(1,"%03d", rawfinal);

        //Espera para dar tiempo a soltar el boton
        wait1Msec(tiemposoltar);
        }
//Cerramos manualmente todos los archivos
CloseAllHandles(resultado);

}
```

A.2. Experimento de oscilación libre del péndulo

Se trata del código correspondiente al archivo “PruebaOscilacion.c”.

```
/*************************************************************************
Autor: Ismael Sanchez Mendoza
Proyecto Fin de Carrera: Diseño y control de un prototipo carro-péndulo basado en LEGO Mindstorms NXT.
Universidad Politecnica de Cartagena, 2009.

        Codigo para registrar la oscilacion libre del pendulo.
        Los datos se guardan en tantos archivos de texto como sea necesario.
        Si un archivo se llena se abre otro y se continua escribiendo.

        Es ajustable:
                        - Numero de lecturas
                        - Tiempo entre cada lectura

  Para un archivo de 15kb se podran almacenar aproximadamente 1400 lecturas,
  antes de pasar al archivo siguiente. Es recomendable tener en cuenta la
  cantidad de memoria flash disponible en el NXT antes de comenzar el programa.
*************************************************************************/

//Variables globales para trabajar con archivos
TFileIOResult resultado;  //Flag que nos indica el resultado de la operacion con un archivo
TFileHandle etiqueta;     //Etiqueta del archivo
int longitud = 15000;     //Longitud de cada archivo en bytes
string nombrearchivo;     //String que sera el nombre de cada archivo
int contador = 1;         //Contador para cambiar el nombre al siguiente archivo

task main()
{
//Tiempo de espera entre medidas, en milisegundos
int tiempoespera = 10;

//Numero de lecturas que se haran
int numerolecturas = 5000;

//Variable que almacena el valor raw devuelto por el potenciómetro
float raw = 0;

//Variable para almacenar el valor de equilibrio inicial
float anguloinicial = 0;

//Variable para almacenar el angulo en cada instante
float angulo = 0;

//Variable para almacenar el error respecto al angulo inicial
float error = 0;
```

```
//String con el valor del error final, que se escribira en el archivo
string dato;


//Cerramos todos los archivos abiertos
CloseAllHandles(resultado);
//Borramos cualquier archivo existente previamente
for (int a=1;a<=10;a++)
	{
	nombrearchivo="Oscilacion "+a+".txt"; //Pasamos el entero "a" a tipo string
	Delete(nombrearchivo,resultado);
	}

//Segmento de codigo para inicializar el angulo de reposo
nxtDisplayTextLine(2, "Pulsa boton");
nxtDisplayTextLine(3, "central para");
nxtDisplayTextLine(4, "guardar inicial");
until(nNxtButtonPressed==3); //Esperamos hasta que se pulse el boton central(3)
	//Borramos el display
	eraseDisplay();
	//Reproducimos un sonido de aviso
	 PlayTone(400, 20);
	//Leemos la entrada
	 raw = SensorRaw[S1];
	 //Inicializamos el valor del angulo mediante la ecuacion correspondiente
	anguloinicial = (34500/((109461/raw)-107));
	//Damos tiempo a soltar el boton
	wait1Msec(600);


//Esperamos a pulsar de nuevo el boton central para comenzar la rutina
nxtDisplayTextLine(2, "Pulsa boton");
nxtDisplayTextLine(3, "central para");
nxtDisplayTextLine(4, "iniciar");
until(nNxtButtonPressed==3);
	//Borramos display
	eraseDisplay();
	 //Reproducimos una cuenta atras con sonidos
	PlayTone(400, 20);
	wait1Msec(1000);
	PlayTone(400, 20);
	wait1Msec(1000);
	PlayTone(600, 40);
	 wait1Msec(600);

//Pasamos el entero "contador" al string "NombreArchivo"
nombrearchivo="Oscilacion "+contador+".txt";
contador++;
//Creamos el primer archivo
OpenWrite(etiqueta,resultado,nombrearchivo,longitud);
```

```
//Escribimos la primera linea de texto en el archivo
WriteText(etiqueta,resultado,"Salida en grados:\r\n");


//Iniciamos la rutina de lecturas del angulo del pendulo
for (int i = 1; i <= numerolecturas; i++)
	{
	if (resultado==0) //Si no se ha producido un error (el archivo aun no esta lleno)
		{
		//Esperamos el tiempo establecido
		wait1Msec(tiempoespera);

		//Leemos la entrada
		raw = SensorRaw[S1];

		//Obtenemos el angulo correspondiente
		angulo=(34500/((109461/raw)-107));

		//Hallamos el error respecto a la posicion inicial
		error=angulo-anguloinicial;

		//Convertimos el error en un string y lo escribimos en el archivo
		dato=error+"\r\n";
		WriteText(etiqueta,resultado,dato);
		}

	else	//Si el resultado es distinto de 0 significa que ha habido error al escribir en
		//archivo (se ha llenado) y hay que seguir escribiendo en otro
		{
		//Creamos un nuevo archivo
		nombrearchivo="Oscilacion "+contador+".txt";
		OpenWrite(etiqueta,resultado,nombrearchivo,longitud);
		contador++;
		}
	}
//Cerramos manualmente todos los archivos
CloseAllHandles(resultado);

}
```

A.3. Experimento de velocidad del vehículo

Se trata del código correspondiente al archivo “PruebaVelocidad.c”.

```
/*****************************************************************************
Autor: Ismael Sanchez Mendoza
Proyecto Fin de Carrera: Diseño y control de un prototipo carro-péndulo basado en LEGO
Mindstorms NXT.
Universidad Politecnica de Cartagena, 2009.
```

```
        Codigo para registrar la respuesta de velocidad de los motores ante
        una entrada escalon. Los datos se guardan en un archivo de texto
        llamado "Velocidad.txt". Para hallar la velocidad, se toma el valor del
        encoder(grados) en un momento determinado, y se compara con el valor un instante
        despues. El tiempo de muestreo esta fijado en 30 ms, por lo que los valores
        de velocidad estaran dados en numero de grados cada 30ms.(grados/30ms).
        Habra que tenerlo en cuenta a la hora de pasar a otras unidades.

        Es ajustable:
                        - Escalon de tension dado a los motores
*********************************************************************************/

//Variables globales para trabajar con archivos
TFileIOResult resultado;  //Flag que nos indica el resultado de la operacion con el archivo
TFileHandle etiqueta;      //Etiqueta del archivo
int longitud = 15000;      //Longitud de cada archivo en bytes


task main()
{
//Velocidad de consigna (debera estar comprendida entre -100 y 100)
int v = 100;

//Variable para almacenar el valor del encoder
int encoder = 0;

//Variable para almacenar el valor del encoder un instante anterior
int encoderprevio = 0;

//Variable para almacenar la velocidad
float velocidad = 0;

//Tiempo de espera entre medidas, en milisegundos
int tiempoespera = 30;

//String con el valor de velocidad, que se escribira en el archivo
string dato;

//Creamos el archivo de texto
Delete("Velocidad.txt",resultado);
OpenWrite(etiqueta,resultado,"Velocidad.txt",longitud);

//Esperamos a pulsar de el boton central para comenzar la rutina
nxtDisplayTextLine(2, "Pulsa boton");
nxtDisplayTextLine(3, "central para");
nxtDisplayTextLine(4, "iniciar");
until(nNxtButtonPressed==3);
        //Borramos display
        eraseDisplay();
```

```
        //Reproducimos una cuenta atras con sonidos
        PlayTone(400, 20);
        wait1Msec(1000);
        PlayTone(400, 20);
         wait1Msec(1000);
          PlayTone(600, 40);
         wait1Msec(600);

//Reseteamos el encoder
nMotorEncoder[motorB] = 0;
//Sincronizamos los motores, el derecho(C) esclavo del izquierdo(B)
nSyncedMotors = synchBC;
//Activamos el motor con la potencia elegida
motor[motorB] = v;

//Reseteamos el timer
ClearTimer(T1);
//El bucle se ejecutara durante un segundo
while(time1<1000)
        {
        encoderprevio = encoder;
        wait1Msec(tiempoespera);
        //Leemos el encoder
        encoder = nMotorEncoder[motorB];
        //Calculamos la velocidad
        velocidad = encoder-encoderprevio;

        //Pasamos la velocidad al string y lo escribimos en el archivo
        dato = velocidad+"\r\n";
        WriteText(etiqueta,resultado,dato);
        }
//Paramos los motores y cerramos todos los archivos
motor[motorB]=0;
CloseAllHandles(resultado);

}
```

A.4. Experimento de oscilación del péndulo ante entradas de tensión al carro

Se trata del código correspondiente al archivo “PruebaImpulso.c”

```
/*****************************************************************************
Autor: Ismael Sanchez Mendoza
Proyecto Fin de Carrera: Diseño y control de un prototipo carro-péndulo basado en LEGO
Mindstorms NXT.
Universidad Politecnica de Cartagena, 2009.
```

```
        Codigo para registrar la oscilacion del pendulo al aplicar una tension
        a los motores. Tanto los datos de entrada (tension) como los de salida
        (angulo) se almacenan en un mismo archivo de texto llamado "DatosImpulso.txt".
        La primera columna correspondera a la entrada y la segunda a la salida.
        Se usaran tantos archivos de texto como sea necesario. Si un archivo se
        llena se abre otro y se continua escribiendo.

        Es ajustable:
                        - Distancia que recorre el coche.
                        - Velocidad del coche
                        - Tiempo entre cada lectura.

        Para un archivo de 15kb se podran almacenar aproximadamente 1400 lecturas,
        antes de pasar al archivo siguiente. Es recomendable tener en cuenta la cantidad
        de memoria flash disponible en el NXT antes de comenzar el programa.
**************************************************************************/

//Variables globales para trabajar con archivos
TFileIOResult resultado; //Flag que nos indica el resultado de la operacion con un archivo
TFileHandle etiqueta;    //Etiqueta del archivo
int longitud = 15000;    //Longitud de cada archivo en bytes
string nombrearchivo;    //String que sera el nombre de cada archivo
int contador=1;          //Contador para cambiar el nombre al siguiente archivo

task main()
{
//Tiempo de espera entre medidas, en milisegundos
int tiempoespera = 10;

//Distancia que queremos que recorra el vehiculo hasta pararse, en cm.
float distancia = 30;

//Velocidad que queremos a los motores (-100 a 100)
int velocidad = 100;

//Valor de giro a los encoders para mover la distancia seleccionada
float giro = 0;

//Variable que almacena el valor raw devuelto por el potenciómetro
float raw = 0;

//Variable para almacenar el valor de reposo inicial
float anguloinicial = 0;

//Variable para almacenar el angulo girado por el pendulo
float angulo = 0;

//Variable para almacenar la diferencia respecto al angulo inicial
float error = 0;
```

```
//String con el valor de la entrada
string datoentrada = 0;

//String con el valor de la salida
string datosalida = 0;


//Borramos cualquier archivo previo
for (int a = 1; a <=10; a++)
	{
	//Pasamos el entero "a" a tipo string
	nombrearchivo = "DatosImpulso "+a+".txt”;
	Delete(nombrearchivo,resultado);
	}

//Segmento para inicializar el angulo de reposo
nxtDisplayTextLine(2,"Pulsa boton");
nxtDisplayTextLine(3,"central para");
nxtDisplayTextLine(4,"guardar inicial");
//Esperamos hasta que se pulse el boton central(3)
until(nNxtButtonPressed==3);
	//Borramos el display
	eraseDisplay();
	//Reproducimos un sonido de aviso
	PlayTone(400, 20);
	//Leemos la entrada
	raw = SensorRaw[S1];
	//Inicializamos el valor del angulo mediante la ecuacion correspondiente
	anguloinicial = (34500/((109461/raw)-107));
	//Damos tiempo a soltar el boton
	wait1Msec(600);

//Esperamos a pulsar de nuevo el boton central para comenzar
nxtDisplayTextLine(2,"Pulsa boton");
nxtDisplayTextLine(3,"central para");
nxtDisplayTextLine(4,"iniciar");
until(nNxtButtonPressed==3);
	//Borramos display
	eraseDisplay();
	//Reproducimos cuenta atras con sonidos
	PlayTone(400, 20);
	wait1Msec(1000);
	PlayTone(400, 20);
	wait1Msec(1000);
	PlayTone(600, 40);
	wait1Msec(700);

//Sincronizamos los motores, el derecho(C) esclavo del izquierdo(B)
nSyncedMotors = synchBC;
//Calculamos el giro necesario de los motores
```

```
giro  = ((distancia*360/(PI*5.6)));
//Establecemos ese valor como el que debe alcanzar el encoder
nMotorEncoderTarget[motorB] = giro;

//Pasamos el entero "contador" al string "nombrearchivo"
nombrearchivo = "DatosImpulso "+contador+".txt";
contador++;
//Creamos el primer archivo
OpenWrite(etiqueta,resultado,nombrearchivo,longitud);

//Guardamos en el archivo el primer valor de potencia a los motores
datoentrada = motor[motorB]+"\t";
WriteText(etiqueta,resultado,datoentrada);

//Iniciamos los motores al maximo de potencia
motor[motorB] = velocidad;

//Borramos el timer
ClearTimer(T1);
//Ejecutaremos el bucle durante un tiempo maximo de 40 segundos
while(time1<40000)
	{
	if (resultado==0) //Si el archivo de texto aun no esta lleno
		{
		//Esperamos el tiempo de bucle
		wait1Msec(tiempoespera);

		//Leemos la entrada
		raw = SensorRaw[S1];

		//Obtenemos el angulo correspondiente
		angulo = (34500/((109461/raw)-107));

		//Hallamos el error
		error = angulo-anguloinicial;

		//Convertimos el error en un string y lo escribimos en el archivo
		datosalida = error+"\r\n";
		WriteText(etiqueta,resultado,datosalida);

		//Si los motores aun no han alcanzado la posicion deseada
		if (nMotorRunState[motorB] != runStateIdle)
			{
			datoentrada = motor[motorB]+"\t";
			//Guardamos la tension en el archivo
			WriteText(etiqueta,resultado,datoentrada);
			}
		//En caso contrario
		 else if (nMotorRunState[motorB]==runStateIdle)
```

```
                    {
                    //La tension a motores sera cero
                    datoentrada="0"+"\t";
                    WriteText(etiqueta,resultado,datoentrada);
                    }
            }

        else if (resultado!=0) //Si resultado es distinto de 0 significa que ha habido error al
                                //escribir en archivo (se ha llenado) y hay que crear otro
            {
            //Creamos un nuevo archivo
            nombrearchivo = "DatosImpulso "+contador+".txt";
            OpenWrite(etiqueta,resultado,nombrearchivo,longitud);
            contador++;
            }
        }
//Cerramos manualmente todos los archivos
CloseAllHandles(resultado);

}
```

A.5. Recogida de datos en los programas de control

Para almacenar en un archivo las entradas y salidas de los experimentos de control, es necesario realizar una pequeña modificación a estos códigos. Tan sólo habrá que añadir unas líneas de código para trabajar con archivos, de forma similar a la que se puede ver en los programas anteriores. Los tres programas de control modificados para almacenar los datos en un archivo son: "*ControlPosDatos.c*", "*ControlPenDatos.c*" y "*ControlPenPosDatos.c*". Se pueden encontrar en la carpeta "*Programas desarrollados en el proyecto*", dentro del CD.

Las líneas a añadir son:

- Declaración de las variables para trabajar con archivos, al comienzo del código. Por ejemplo:

  ```
  TFileIOResult resultado; //Flag que nos indica el resultado de la operacion con un
                           archivo
  TFileHandle etiqueta;    //Etiqueta del archivo
  int longitud = 15000;    //Longitud de cada archivo en bytes
  string nombrearchivo;    //String que sera el nombre del archivo
  ```

  El nombre del archivo deberá contener como máximo 15 caracteres y una extensión. En este caso será ".txt".

- Antes de comenzar el bucle de control, borramos cualquier archivo con el mismo nombre y creamos el nuevo archivo de texto listo para escribir en él.

En caso de que no se borre el archivo anterior, habrá un error al intentar crearlo.

```
Delete(nombrearchivo,resultado);
OpenWrite(etiqueta,resultado,nombrearchivo,longitud);
```

- En el momento que nos convenga, convertimos el dato a guardar en un string, y seguidamente lo almacenamos en el archivo. La conversión a string la hace RobotC directamente. Sólo hay que asignar cualquier valor a una variable declarada como tipo string. Esta variable ("dato" en el ejemplo), deberá estar declarada como string al comienzo del programa.

```
dato = error+"\t";  // Convertimos a string el error (por ejemplo)
WriteText(etiqueta,resultado,dato); // Escribimos en el archivo
```

Los caracteres "\r\n" y "\t" sirven para indicar a RobotC que efectúe en el archivo de texto un salto de línea o una tabulación respectivamente. Esto es útil para organizar los datos en un mismo archivo al almacenar varias variables.

- Finalmente, es conveniente al finalizar el programa cerrar manualmente todos los archivos abiertos.

```
CloseAllHandles(resultado);
```

ANEXO B

Guía de montaje de la maqueta usando LDD

En el CD del proyecto se encuentra la documentación necesaria para visualizar la maqueta construida en LDD y generar la guía de montaje.

Para ello es necesario instalar previamente el software LEGO Digital Designer. Es importante tener en cuenta que el software solo es operativo en Windows y en Mac OSX, y el programa ocupa unos 100 MB en el disco duro. El ejecutable para la instalación se encuentra siguiendo la siguiente ruta:

CD:\Guía de montaje\Instalador de Lego Digital Designer\ SetupLDD-PC-2_3_19.exe

Se trata de la versión 2.3.19 para PC. En cualquier caso, este instalador también se puede descargar gratuitamente de su web [12].

Una vez instalado el programa, hay que abrir el archivo que contiene el modelo construido y que se puede encontrar en:

CD:\Guía de montaje\Maqueta carro-péndulo.lxf

Al abrir el archivo aparecerá un aviso de que no se podrá comprobar el precio total del conjunto debido a algunas de las piezas usadas. Esto no es importante.

En la ventana principal aparecerá la imagen del modelo construido, con las salvedades que se indican en el Capítulo 3 de la memoria. Para generar la guía de montaje, hay que hacer clic en uno de los tres botones superiores (*Building Guide Mode).* Una vez hecho esto, aparecerá una pequeña ventana donde se puede seleccionar el numero de piezas por paso (3 o 4 es un buen numero) y algunas opciones más.

Para generar la guía, en esa misma ventana hay que hacer clic en *Generate Building Guide*. Una vez generada, se nos presenta una animación del montaje que podemos reproducir paso a paso a través de la ventana mencionada (ver Figura B.1).

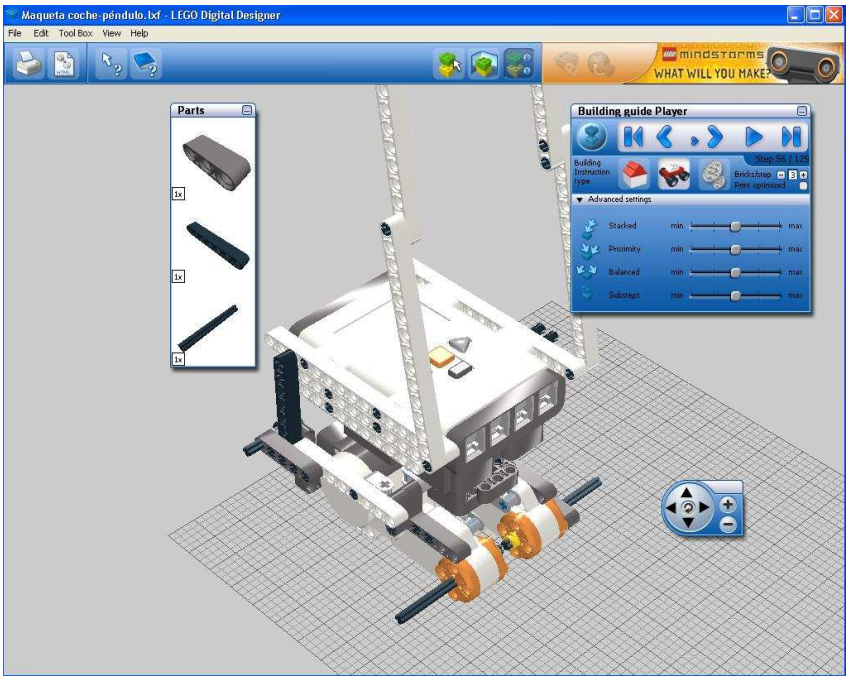


Figura B.1. Captura de pantalla de la animación del montaje en LDD

Por último, para llevar a cabo el montaje del péndulo y del potenciómetro que no se encuentran representados en LDD, se adjuntan en el CD unas fotos aclaratorias, que pueden encontrarse en los directorios:

CD:\Guía de montaje\Montaje del péndulo

CD:\Guía de montaje\Montaje del potenciómetro