Diseño de un controlador PID en NXT OSEK para el autoequilibrado de un robot sobre dos ruedas.

Manuel Pedrero Luque, Raúl Pérula Martínez

Profesor tutor: José Manuel Palomares Muñoz

i62pelum@uco.es, i62pemar@uco.es

Resumen:

Este artículo expone una posible solución al problema de equilibrado de un robot sobre dos ruedas mediante el uso de un sensor de luminosidad y el diseño e implementación de un sistema de control PID (Proporcional, Integral, Derivativo) para la corrección del desequilibrio en tiempo real. Para la construcción del robot se ha utilizado un set educativo LEGO Mindstorms NXT 2.0 y para la implementación del software se ha utilizado la plataforma nxtOSEK.

Palabras Clave: Tiempo Real, Equilibrio, Lego, nxtOSEK, PID.

Abstract:

This article exposes a possible solution for balance problem in a 2-geared robot using a brightness sensor and designing and coding a PID control system (Proportional, Integral, Derivative) for correcting balance in real time. An educative set LEGO Mindstorms NXT 2.0 was used to robot construction along with nxtOSEK platform to software implementation.

Keywords: Real Time, Balance, Lego, nxtOSEK, PID.

1. Introducción

1.1. Sistemas en Tiempo Real

Se denominan Sistemas en Tiempo Real a aquellos dispositivos que deben producir unas salidas determinadas como respuestas a unas entradas *en unos límites de tiempo específicos*.

Dichos sistemas se utilizan en gran cantidad de dispositivos, desde simples alarmas hasta misiles, pasando por sistemas de monitorización, procesadores de señal o medios de transporte. La principal diferencia entre estos dispositivos y otros sistemas clásicos es la presencia de *restricciones temporales*. Una respuesta óptima que no cumpla estrictamente con el tiempo requerido para obtenerla se considera errónea, mientras que una respuesta *aceptable* dentro del límite de tiempo asignado es correcta. La importancia de obtener respuesta en el límite de tiempo es máxima, ya que las consecuencias de no obtenerla pueden ser fatales. Pensemos en un misil que no pueda salvar un obstáculo a tiempo, o el sistema activación de airbags de un automóvil.

El problema de equilibrado de un robot sobre dos ruedas que se plantea en este artículo es otro ejemplo de sistema en tiempo real. No sólo es necesario corregir la desviación que sufre el robot lo antes posible, sino que si no la detectamos la desviación y calculamos la corrección antes de un tiempo determinado, será imposible que los motores puedan corregirla, por lo que el robot caerá al suelo irremediablemente.

Este tipo de dispositivos requiere de sistemas operativos especializados y lenguajes que siguen un paradigma completamente distinto al de la programación clásica.

1.2. Lego Mindstorms NXT [1]

Lego Mindstorms NXT es una línea de productos del fabricante Lego que permite utilizar los bloques del juego de construcción para la creación de robots, y que incluye elementos propios de sistemas robóticos como sensores, actuadores y un bloque programable que hace las veces de procesador. Permite la programación de software mediante su

propio lenguaje (NXT-G) aunque a lo largo del tiempo han aparecido lenguajes alternativos e incluso sistemas operativos compatibles.

La plataforma Lego Mindstorms nació fruto de la colaboración de Lego y el MIT, y se lleva comercializando desde 1998. En la actualidad, la versión disponible es la 2.0, que utiliza el bloque de control NXT, una evolución del utilizado en versiones anteriores (RCX). Dicho bloque incluye un microcontrolador ARM7 de 32 bits, 256Kb de memoria Flash y 64Kb de RAM. Además, incluye cuatro entradas para conectar diversos sensores y tres salidas para alimentar a actuadores como luces o motores.

1.3. Plataforma nxtOSEK [2][3][4]

La plataforma nxtOSEK es un conjunto de programas de código abierto que permiten la programación en ANSI C/C++ del bloque Lego NXT y el acceso a todas sus funciones incluyendo manejo de sensores y actuadores o acceso al módulo Bluetooth. Además permite el uso de TOPPERS, una plataforma abierta que proporciona un sistema operativo en tiempo real para dispositivos embebidos y que ha demostrado su fiabilidad en sistemas industriales.

2. Antecedentes

Existen varios antecedentes a este proyecto tanto utilizando Lego Mindstorms, normalmente de ámbito educativo y/o personal, como proyectos comerciales. Se han destacado dos proyectos que guardan cierta similitud, aunque existen muchos otros que no se tratarán en este artículo.

2.1. Segway Personal Transporter [5]

El antecedente comercial más claro lo tenemos en el dispositivo Segway Personal Transporter, un conocido medio de transporte monoplaza que utiliza un motor eléctrico y se desplaza sobre dos ruedas, autoequilibrándose automáticamente por medio de gi-

roscopios que le ayudan además a mantener su estabilidad. Dicho dispositivo fue presentado en 2001 y lanzado al mercado unos meses más tarde.

2.2. Legway [6]

Legway es un proyecto no comercial realizado también con la plataforma Lego Mindstorms NXT, en este caso utilizando el lenguaje oficial NXT-G. Para mantener el equilibrio, Legway utiliza dos sensores de proximidad electro-ópticos a ambos lados del robot que le permiten detectar la distancia al suelo con gran precisión. Dichos son comercializados por una empresa externa a Lego, que proporciona además librerías para añadir compatibilidad con el lenguaje oficial.

3. Objetivos

El objetivo principal de este trabajo fue diseñar un software en tiempo real que detecte y corrija automáticamente la desviación sufrida por un robot apoyado sobre dos ruedas utilizando para ello los sensores disponibles en el laboratorio y dos servomotores para el movimiento de las ruedas.

4. Proceso de diseño

En este capítulo se describe el proceso llevado a cabo para el diseño del software de autoequilibrado del robot.

4.1. Configuración del robot

El robot se construyó siguiendo dos ideas básicas: estabilidad y disponibilidad de piezas. Para conseguir la mejor estabilidad posible, se procuró que el centro de gravedad del robot estuviese lo más bajo posible. El peso del bloque NXT es sensiblemente superior al del resto de partes, seguido del de los motores. Además, se intentó minimizar el número de piezas utilizadas, obteniéndose el diseño que se puede ver en la imagen.

***Figura 1**: Robot NXT*

Como puede observarse, el robot tiene el bloque NXT, marcado como *LEGO10* situado en la parte central, y lo más bajo posible de forma que los cables no toquen el suelo. Además podemos ver dos servomotores a derecha e izquierda que controlan las ruedas del robot, un sensor de luminosidad, marcado como *L10*, y un pulsador situado en la parte superior izquierda de la imagen y marcado como *P14.*

4.2. Primera aproximación: uso de intervalos proporcionales

Como primer enfoque, se equilibró el robot manualmente en una superficie blanca mate y se recogió el nivel de luminosidad medido por el sensor. A continuación se observó en qué medida se alteraba ese nivel al inclinar el robot a un lado y a otro, tras lo cual se estableció un límite de inclinación a partir del cual el robot intentaría corregirla poniendo sus motores en la misma dirección a la mitad de su potencia. Se estableció además un segundo nivel que haría que el robot intentase corregir la desviación con sus motores a plena potencia.

Desgraciadamente este enfoque tenía varias debilidades. La primera es que el robot no conseguía equilibrarse, ya que a pesar del trabajo de calibración con los límites de inclinación lo más que se consiguió fue que oscilase un par de veces antes de caer al suelo.

Si el límite era muy bajo, el robot tendía a sobrecorregir, por lo que oscilaba violentamente hacia el otro lado, y la propia desviación unida a la inercia de la sobrecorrección hacía imposible que se estabilizase.

Si por el contrario el límite era demasiado alto, el robot no era capaz de corregir la primera desviación, por lo que caía igualmente.

Se planteó la posibilidad de introducir más intervalos de potencia para suavizar las correcciones en la medida de lo posible pero, en lugar de eso, se intentó evolucionar a un control proporcional, como se verá a continuación.

La segunda debilidad era que al trabajar con un sensor lumínico, el valor medido dependía de la superficie de apoyo o la condición de luz, por lo que el punto de equilibrio que consideraba el sistema no se correspondía con el punto de equilibrio real, y el robot partía desde el principio con un punto objetivo erróneo.

4.3. Segunda aproximación: uso de un control proporcional y calibrado del sistema

Esta evolución de la idea anterior intenta que el robot entregue una potencia a sus motores de manera directamente proporcional a la desviación sufrida respecto de la vertical, que está relacionada también de modo proporcional con el nivel de luz medido. Además intenta evitar en la medida de lo posible que el robot se vea alterado por las distintas condiciones de luz.

El problema del establecimiento de un punto de equilibrio, se solucionó añadiendo un pulsador (etiquetado en la imagen anterior como P14) y modificando el sistema de modo que al pulsarlo se entrase en un modo de calibración. Acto seguido, se equilibraba manualmente el robot y se pulsaba de nuevo el botón. Así, el robot recogía el nivel exacto de luz de ese momento y lo usaba como base para calcular el resto de operaciones.

Para hacer el control proporcional, se estableció una inclinación límite a cada lado del robot, de modo que al llegar a ella los motores estuviesen a plena potencia. A continuación se estableció una función proporcional de modo que la potencia de los motores

disminuía gradualmente hasta pararlos completamente en el punto de equilibrio. Para calibrar la respuesta del robot, se añadió un coeficiente de calibrado proporcional K_p que multiplicaba al resultado de la función permitiendo una reacción más o menos sensible.

```
Algoritmo de control proporcional:
brillo_actual <- obtener_nivel_brillo();
error = brillo_actual - brillo_equilibrio;
mover_motores(Kp*error);
```

Como se puede observar, se intentó realizar un algoritmo lo más sencillo posible, ya que estaba pensado para ejecutarse en una tarea con un período de sólo 20 ms.

Con este enfoque y tras bastantes pruebas de calibrado, se consiguió que el robot se mantuviese oscilando tres o cuatro segundos, hasta que la oscilación crecía demasiado y el robot caía. Por otra parte, el comportamiento se mantenía en diferentes condiciones de luz gracias al sistema de calibrado.

4.4. Tercera aproximación: controlador PID [7]

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

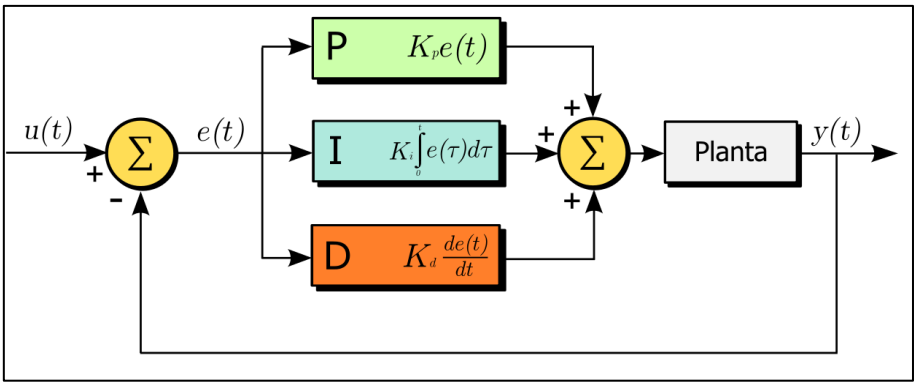

***Figura 2**: Esquena de control PID*

En dicho mecanismo se calculan tres mediciones de error y se obtiene una suma ponderada de las mismas multiplicando cada error por un determinado coeficiente, y obteniendo el resultado que hay que aplicar a los motores del robot. Cada una de estas medidas busca corregir un aspecto diferente de las desviaciones del robot.

4.4.1. Control proporcional

El control proporcional se realiza simplemente obteniendo el producto entre la señal de error que denominaremos E_p y la constante proporcional K_p de modo que produzca una salida casi nula en valores e error bajos y aumente a la vez que aumenta E_p. En la mayoría de los casos, esto sólo produce valores óptimos en un pequeño rango de la zona de control. En nuestro caso, consigue que el sistema realice una corrección mayor cuanto mayor sea la diferencia entre el valor de luminosidad actual y el valor de luminosidad en el punto de equilibrio.

$$P_{sal} = K_p \cdot e(t) \qquad (1)$$

Como se puede observar, esta función es la que ya se calculaba en el enfoque anterior, y el coeficiente que se aplicaba es equivalente al coeficiente de ponderación que aplica PID.

4.4.2. Control integral

El control integral intenta disminuir y eliminar el error en estado estacionario que provoca la función proporcional. Este control actúa cuando existe una desviación entre la variable y el punto referencia (nuestro punto de equilibrio), e integra esta desviación en el tiempo sumándola después al error proporcional.

En nuestro caso, y por simplicidad, hemos sumado acumulativamente los errores de desviación que se producían con respecto al tiempo, obteniendo un resultado equivalente y consiguiendo una mayor simplicidad. El resultado es equivalente ya que tomamos medidas cada $d(t)$, salvo que en nuestro caso nuestro diferencial es de 25 ms. Así conseguimos que si un error de desviación se mantiene con el mismo signo durante un período prolongado de tiempo, el resultado del control integral aumenta rápidamente, consiguiendo que aumente la potencia de los motores para corregirlo, en lugar de entregando una potencia fija como en el caso del control proporcional. Así:

$$I_{sal} = K_i \int_t^0 e(\tau)d\tau \cong 1/K_{ai} \cdot K_i \cdot (e(t_0) + e(t_{20}) + e(t_{40}) \ldots + e(t_n)) \qquad (2)$$

Donde K_i es un coeficiente que permite calibrar la sensibilidad de la respuesta y K_{ai} es un coeficiente de amortiguamiento que reduce el efecto de la integral en casos en los que se haya obtenido un error de desviación en un periodo relativamente largo de tiempo.

4.4.3. Control derivativo

El control derivativo se basa en el la evolución actual del error para determinar su tendencia. Es decir, predice el desvío que va a tener el robot antes de que lo tenga, lo que permite corregir los otros dos controles antes de que sea tarde. Este control resulta especialmente útil cuando el robot ha corregido un desvío hacia un lado y por inercia tiende a desviarse para el otro. Gracias a esta función, los motores reducen su potencia antes de llegar al punto de equilibrio evitando que las oscilaciones sean cada vez mayores.

La función de control derivativo está dada por:

$$D_{sal} = K_d \cdot \frac{de}{dt} \qquad (3)$$

En el caso de nuestra implementación, optamos por una solución simplificada de esta fórmula, que consistía simplemente en obtener el error en el instante t y en el instante $t-1$, es decir, 40 ms antes. Esto hace las veces de diferencial de error aproximada, que dividimos entre nuestro diferencial de tiempo que es 1. Conseguimos así reducir la complejidad del control manteniendo en la medida de lo posible su efectividad.

$$D_{sal} \cong K_d(e_{t_0} - e_{t_{-1}}) \qquad (4)$$

Como se puede observar, se añade un tercer coeficiente K_d que permitirá calibrar la respuesta a este control al igual que en los controles anteriores.

4.4.4. Control combinado PID

Una vez calculadas las respuestas de los tres controles, sólo resta combinarlas mediante una suma ponderada por los tres coeficientes que hemos mencionado:

$$PID_{sal} = \left(K_p \cdot E_p + K_i \cdot P_i \cdot K_d \cdot P_d\right) = P_{sal} + I_{sal} + D_{sal} \qquad (5)$$

Una vez implementada esta funcionalidad, se ponderó usando un procedimiento similar al descrito en [8]. Se aumentó el peso de la K_p dejando nulas K_i y K_d hasta que el robot oscilaba respecto al punto de equilibrio y a partir de ahí, se aumentó el peso de K_i y por último el de K_d hasta conseguir una respuesta adecuada. En nuestro caso, después de muchas pruebas, el comportamiento óptimo se consiguió con unos pesos de 5 para la K_p, 2 para K_i y 1 para K_d.

5. Implementación en nxtOSEK [9][10]

A la hora de implementar este procedimiento en nxtOSEK, hay que tener en cuenta que el método de programación se basa en tareas que luego son planificadas por el sistema operativo en tiempo real atendiendo a su prioridad. Para nuestra implementación se consideraron tres tareas:

- La tarea **Calibrar**: Se encarga de recoger el valor de luminosidad base que se utilizará en el resto del programa. Para ello se equilibra el robot manualmente lo mejor que se pueda y se pulsa el botón de calibración, momento en el que se recoge la medida de luminosidad y se almacena. Esta tarea sólo la activa la tarea *Equilibrar* cuando es necesario.
- La tarea **Equilibrar**, Se ejecuta de manera periódica cada 20 ms y se encarga de la implementación del esquema PID y del equilibrio del robot. Para ello recoge el valor de luminosidad de un instante, calcula las funciones Proporcional, Integral y Derivativa y obtiene la potencia que ha de suministrar a los motores. Además comprueba si se ha pulsado el botón de calibrado, lanzando la tarea Calibrar en caso afirmativo.
- La tarea **PantallaLCD**: Se ejecuta de manera periódica cada 500 ms y se encarga de mostrar por pantalla información relevante como los pesos de las funciones, los valores de las mismas en bruto y ponderados, la potencia que se le aplica a los motores, el valor de luminosidad actual y el valor de luminosidad base. Es la menos prioritaria para que no pueda interrumpir a la tarea equilibrar, que es crítica.

Como resultado de esta implementación y posterior calibrado, el robot puede mantener el equilibrio por un período prolongado de tiempo (minutos) en un ambiente de pruebas controlado, con una superficie uniforme e iluminación constante para conseguir la máxima fiabilidad posible por parte del sensor de luminosidad.

6. Conclusiones

En este artículo se han explicado las bases del control PID y se han aplicado experimentalmente a la creación de un controlador para un robot con autoequilibrado. Se ha comentado el proceso de aproximación y refinamiento empleado para mejorar el comportamiento del robot y cumplir el objetivo propuesto. Mirando en la literatura se puede comprobar como el control PID es muy usado en procesos de control y ha demostrado su eficacia en muchos otros casos.

7. Referencias

1. LEGO Mindstorms. Última visita: (20-02-2011).
   Enlace: http://mindstorms.lego.com/en-us/default.aspx

2. nxtOSEK/JSP (ANSI C/C++ with OSEK/µITRON RTOS for LEGO MIND-STORMS NXT). (Última visita: 20-02-2011).
   Enlace: http://lejos-osek.sourceforge.net/

3. OSEK Implementation Language (OIL) (Version 2.5). (Última visita: 20-02-2011).
   Enlace: http://portal.osek-vdx.org/files/pdf/specs/oil25.pdf

4. OSEK OS (Versión 2.2.3). (Última visita: 20-02-2011).
   Enlace: http://portal.osek-vdx.org/files/pdf/specs/os223.pdf

5. Segway Simply Moving. (Última visita: 20-02-2011).
   Enlace: http://www.segway.es/faq.asp

6. Steve's LegWay. (Última visita: 02-03-2011).
   Enlace: http://www.teamhassenplug.org/robots/legway/

7. PID controller for NXT Robots. (Última visita: 02-03-2011).
   Enlace: http://www.inpharmix.com/jps/
   PID_Controller_For_Lego_Mindstorms_Robots.html

8. Wikipedia: PID .Última visita: (02-03-2011).
   Enlace: http://es.wikipedia.org/wiki/Proporcional_integral_derivativo

9. Manual de Referencia OIL. UCO Moodle. (Última visita: 20-02-2011).
   Enlace: http://www3.uco.es/moodle/

10. Manual de Referencia nxtOSEK. UCO Moodle. (Última visita: 20-02-2011).
    Enlace: http://www3.uco.es/moodle/