Universidad Politécnica de Cartagena

industriales
etsii UPCT

Título del Proyecto

APLICACIÓN DE TECNICAS CLÁSICAS Y DE CONTROL MODERNO SOBRE MINIROBOT CON CONFIGURACIÓN SEGWAY.

Titulación: Ingeniero en Automática y Electrónica Industrial

Alumno: José Antonio Cerezuela Barreto

Directores: Julio José Ibarrola Lacalle
José Manuel Cano Izquierdo

Cartagena, 15 de febrero de 2013

Índice general

1. Introducción

En el Departamento de Ingeniería de Sistemas y Automática de la UPCT se han llevado a cabo en los últimos años varios proyectos basados en el set de robótica programable LEGO Mindstorms NXT.

Entre los proyectos precursores, está el proyecto fin de carrera titulado *"Estudio de las posibilidades didácticas en la ingeniería de control del LEGO Mindstorms NXT"*, año 2008, de Guillermo Nieves Molina. El objetivo principal de este proyecto era el de estudiar la viabilidad del set LEGO Mindstorms NXT (Sensores, Actuadores y Unidad de Control) como medio para implementar sobre él algoritmos de control. Tras su evaluación favorable se adoptó como herramienta para proyectos posteriores.

Siguiendo con el uso de este hardware, este proyecto trabajará sobre una construcción particular denominada NXTway-G, que emula la forma y el comportamiento del producto comercial Segway y consistirá fundamentalmente en la aplicación de técnicas clásicas y de control moderno sobre dicho minirobot.

1.1. Segway PT

Según Wikipedia, el Segway PT (Segway Personal Transporter) es un vehículo de transporte ligero giroscópico eléctrico de dos ruedas, con autobalanceo controlado por ordenador, inventado por Dean Kamen y presentado en diciembre de 2001. El ordenador y los motores situados en la base mantienen la base del Segway horizontal todo el tiempo. El usuario se debe inclinar hacia la dirección que quiera tomar (delante, atrás, derecha o izquierda). El motor es eléctrico y silencioso, alcanzando hasta los 20 km/h.

Imagen 1.1 Ejemplo del Segway PT y de la posición del persona sobre su base

Desde el punto de vista de control, se puede describir este sistema como un péndulo invertido sobre 2 ruedas accionadas por 2 servomotores DC independientes, siendo este un sistema inestable y no lineal, por lo que requiere en primer lugar la implementación de un sistema de control para mantenerlo en posición vertical (equilibrio dinámico) a pesar de perturbaciones externas.

Posteriormente, el algoritmo de control debe permitir además poder desplazarlo en cualquier dirección manteniendo la estabilidad vertical ante perturbaciones externas e internas moderadas, como son el efecto de las aceleraciones de sus motores que tienden a desequilibrarlo.

1.2. LEGO Mindstorms NXT

LEGO Mindstorms NXT es un sistema que combina la versatilidad del sistema constructivo LEGO con las nuevas tecnologías de sensores y actuadores junto a una unidad central (NXTBrick o "ladrillo") con microprocesador programable incorporado y un software de programación visual basado en LabView.
El kit de herramientas cuenta con todo lo necesario para crear múltiples configuraciones de robots sin necesidad de herramientas específicas.

Imagen 1.2 Robot con configuración Segway.

2. Objetivos del proyecto

2.1. Objetivos

- Construir mediante los elementos contenidos en el Kit LEGO Mindstorms NXT un minirobot similar al Segway (en adelante NXTway-G).

- Implementar algoritmos de control (tradicional y moderno) para mantener el robot vertical en equilibrio dinámico estable sobre sus 2 ruedas.

- El robot debe soportar perturbaciones externas moderadas y seguir manteniendo el equilibrio.

- Permitir el control remoto desde Joystick para desplazarse hacia delante / atrás así como girar sobre su eje vertical con aceleraciones moderadas).

- Poder subir un rampa, controlado remotamente mediante un Joystick y mantener el equilibrio verticalidad sobre la rampa.

2.2. Desarrollo de Objetivos

- Ver estado del arte actual en este campo de los minirobots y el caso concreto de aplicaciones similares al NXTway-G.

- Aprender el manejo de compiladores, lenguajes de programación, librerías y funcionamiento de sensores y actuadores utilizados.

- Modelar matemáticamente el sistema real del robot.

- Diseñar los sistemas de control (tradicional y moderno) requeridos.

- Simular mediante Matlab el comportamiento conjunto del sistema y los diferentes algoritmos de control propuestos.

- Implementar y probar sistema de control directamente sobre el robot mediante la programación, compilación y ejecución del código por el microprocesador existente en el ladrillo de LEGO.

2.3. Consideraciones

- El cálculo de los parámetros adecuados para estabilizar el sistema así como la simulación será realizado con el programa de cálculo Matlab 7.6.0.

- Los algoritmos de control en tiempo real serán programados mediante el lenguaje RobotC (similar a C) con capacidad de operaciones en coma flotante. Este software permite compilar y transferir el código al ladrillo tanto vía USB como Bluetooth.

- La conexión habitual entre PC/Joystick y NXTBrick del Robot será inalámbrica por Bluetooth.

- Se realizará una simulación de los 2 métodos de control utilizados con el software de Matlab/Simulink.

- Se utilizará un joystick conectado vía USB al PC para el control remoto por Bluetooth de los movimientos del robot.

3. Construcción del minirobot tipo Segway

El primer paso del proyecto consistirá en construir con los elementos y piezas disponibles el minirobot tipo Segway.

3.1. Material del que se dispone:

En la imagen 3.1 se muestran los elementos principales del Kit LEGO Mindstorm NXT. Destacar la calidad de los materiales y su sólida construcción típicos de LEGO.

1 Kit LEGO Mindstorms NXT

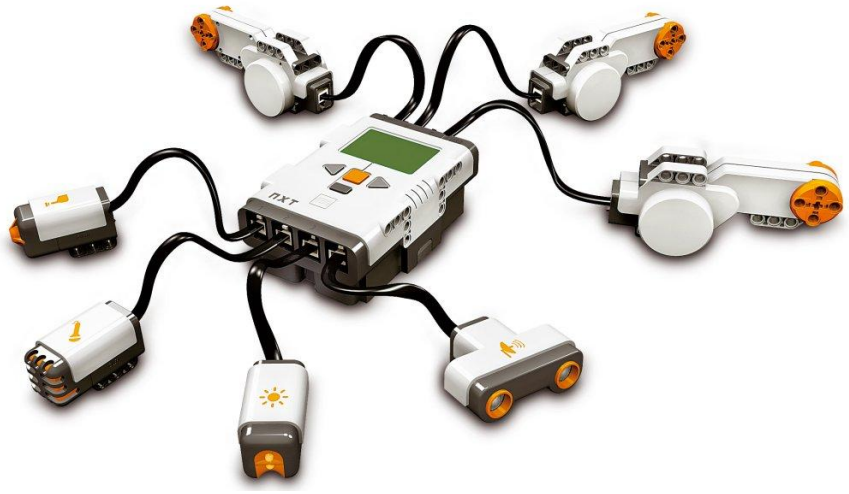

Imagen 3.1 NXTBrick con Sensores y Servomotores conectados

3.1.1. Componentes requeridos

1. **LEGO NXTBrick (CPU & I/O):**

El ladrillo permite la conexión directa de sensores/actuadores, dispone de un display LCD frontal de 100 x 64 pixel, 4 botones de goma que hacen de interface con el programa y sistema de comunicación vía USB/Bluetooth.

El módulo tiene 4 conexiones para entradas de sensores y 3 salidas hacia motores.

Imagen 3.2 NXTBrick de LEGO

Las características principales del módulo de control son las siguientes:

Procesador Principal:

Atmel 32-bit ARM processor, AT91SAM7S256
256 KB FLASH, 64 KB RAM, 48 MHz

Co-procesador:

Atmel 8-bit AVR processor, ATmega48
4 KB FLASH, 512 Byte RAM, 8 MHz

Interface I/O

- 4 Puertos entrada (Interface de 6 hilos que soporta tanto señales digitales como analógicas).
- 3 Puertos entrada (Interface de 6 hilos que soporta las señales de realimentación de los encoders en los servomotores).
- 4 botones de goma para interface con el usuario

Comunicación

- USB 2.0 (12 Mbit/s)
- Bluetooth (inalámbrica)
  CSR BlueCoreTM 4 v2.0 + EDR System

2. SENSORES

- Giróscopo electrónico de HighTechnic

- Sensor de luz

3. ACTUADORES/ OUTPUT from NXTBrick:

- Motores DC controlador por PWM con realimentación de posición por encoder ya incluido (servomotores).

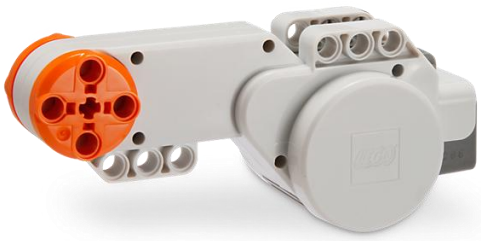

Imagen 3.4 Servomotor DC

4. PIEZAS DE CONTRUCCIÓN LEGO

Imagen 3.5 Ruedas finas de mayor diámetro y piezas LEGO Technic

5. ELEMENTOS AUXILIARES:

- Cables Conexionado de Sensores (izq.) /Actuadores, cable USB (der.)

- USB Bluetooth Dongle compatible con LEGO NXT

Imagen 2.4 Bluetooth dongle

- Set de Batería Litio recargable + Cargador

- Joystick con conexión a PC por USB

3.2. Construcción del Robot NXTway-G

El montaje del NXTway-G para este proyecto tiene una configuración "esbelta", estando el centro de masas del ladrillo a 140 mm respecto al eje de las ruedas y tiene una altura máxima con ruedas de 320 mm.

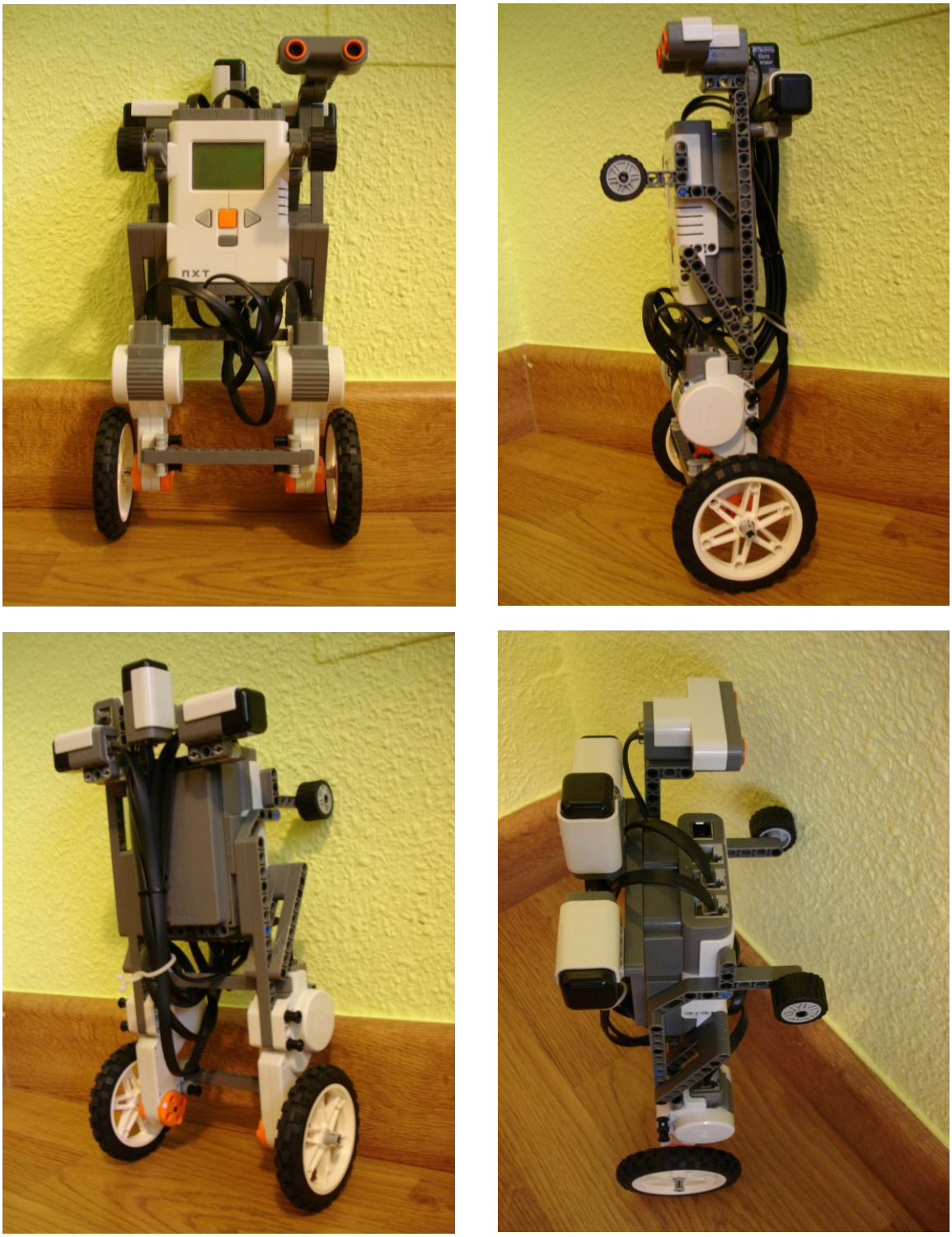

Imagen 3.10 Vistas del robot con configuración Segway construido para el proyecto.

Indicar que es importante que la rigidez del robot sea alta, y esto lo conseguimos con una configuración adecuada y la utilización de rigidizadores que limiten las holguras.

El robot con las ruedas de mayor radio ofrecen una mejor capacidad de control permitiendo mayor velocidad lineal para una misma velocidad angular máxima de los motores (limitada esta por la tensión de la batería y configuración reductora internas de estos).
Notar que a medida que la aumenta la velocidad angular del motor el par disponible va disminuyendo, y por tanto la efectividad de las señales de control.

3.3. Descripción de los Sensores

Como el objetivo de tener un primer contacto con los sensores utilizados en este proyecto, su modo de operación, errores de medida, offset, derivas en el tiempo, acceso a la información mediante código de programa en C, posibilidades de configuración,… se procede a realizar un estudio de los principios de operación, usos y configuraciones así como comprobar la operación real de estos.

Llama la atención ver que el Giróscopo, el Acelerómetro y la Brújula que el aspecto exterior de sus encapsulados de plástico son exactamente iguales (tamaño, colores,…) y solo se diferencian por un pequeño texto en la cara inferior del extremo en plástico negro. En la figura 3.11 se muestra el aspecto externo de estos sensores de HiTechnic fabricados para LEGO Mindstorms NXT.

Las imágenes 3.11 y 3.12 muestran el aspecto exterior los sensores utilizados en este proyecto (Giróscopo en Imagen 3.11 y Sensor Óptico en Imagen 3.12).

Imagen 3.11 Vista exterior encapsulados plástico con conector

Imagen 3.12 Vista exterior sensor de luz con diodo emisor y receptor

3.3.1. Sensor Óptico usado como sensor proximidad

1. INTRODUCCIÓN

El sensor óptico incluye un fototransistor para la medida de iluminación, junto a un diodo LED que puede activarse o no, dependiendo de si la medición se está efectuando en un ambiente oscuro o iluminado.

El rango de medida es más amplio que el espectro visible por el ojo humano, lo que puede inducir a confusiones con fuentes de luz que no se consideren en un principio, como pilotos infrarrojos.

El pico de sensibilidad se obtiene para las longitudes de onda de 900 nm, mientras que en el espectro de los 400 a los 700 oscila entre un 30 y un 60% de sensibilidad.

El margen de medición comprende desde los 0 hasta los 1000 lux, con una curva de respuesta bastante lineal.

2. INSTALACIÓN DEL SENSOR EN EL NXTway

Como se aprecia en la imagen 3.13, el sensor de luz se ha acoplado a la parte trasera del robot, de modo que cuando esté en equilibrio vertical los sensores estén a una distancia de la superficie lo más cercana a los 6 mm recomendados en sus especificaciones. A medida que el sensor se aleje o acerque, la señal variará dando una indicación del grado de inclinación del robot.

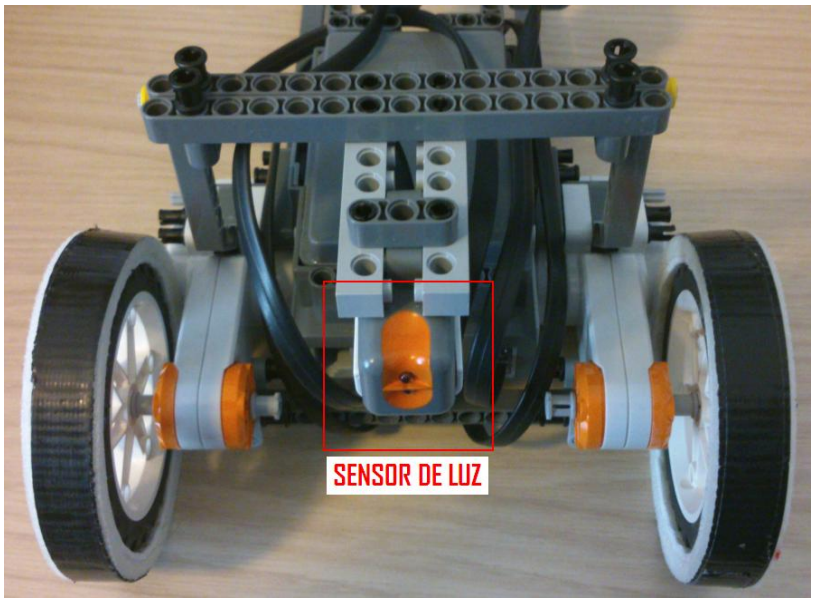

Imagen 3.13.0 Vista inferior de la posición del sensor óptico acoplado al minirobot

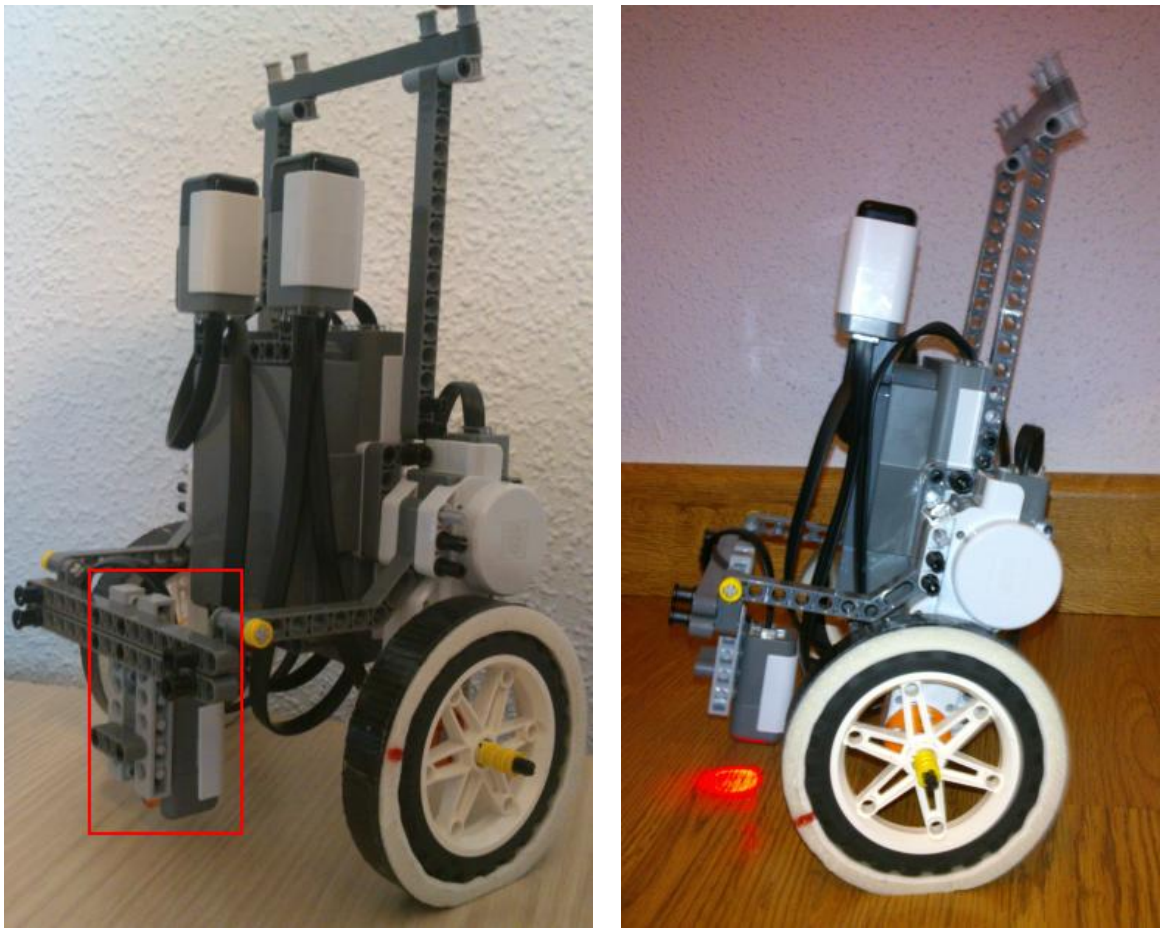

Imagen 3.13.1 Vista disposición sensor de luz en posición vertical y durante oscilación.

3.3.2. Sensores de Movimiento

1. INTRODUCCIÓN

El movimiento de cualquier cuerpo sólido puede ser descrito como la composición de una translación y una rotación.

Los acelerómetros angulares indican cómo está girando el cuerpo en el espacio. Por ejemplo, en una aeronave suelen haber 3 acelerómetros que miden el cabeceo, guiñada y alabeo (pitch, yaw, roll).

Los acelerómetros lineales indican como se está trasladando el cuerpo en el espacio e igualmente suele haber 3 de estos dispositivos que describan el movimiento de translación en los tres ejes de coordenadas (x, y, z).

El giróscopo da una medida de la velocidad angular con la que gira un cuerpo en un plano dado. Si disponemos de 3 giróscopos podemos describir la velocidad angular total del cuerpo.

Los giróscopos junto con los acelerómetros constituyen los sensores de movimiento principales de los Sistemas de Navegación Inercial (INS). Su necesidad surge cuando no es posible tener una referencia externa para determinar la posición y orientación en el espacio en tiempo real, por lo que se recurren a calcularla partiendo de la posición inicial conocida e integrando continuamente los valores dados por los giróscopos y acelerómetros. Los valores finalmente obtenidos son con respecto a un sistema de referencia inercial.

Todos los sistemas de navegación inerciales tienen una deriva o error con el tiempo consecuencia de ir integrando pequeños errores de medida y por tanto desvirtuando el valor final. Es por ello que se complementan con sistema de posicionamiento global (GPS) que dan la posición real y permiten resetear el error acumulado del sistema de navegación inercial pero cuya respuesta en tiempo es insuficiente para realizar el control de navegación automático de la nave o vehículo.

El giróscopo HiTechnic del que se dispone no incorpora integración por hardware para poder obtener el valor de ángulo directamente, proporcionando solamente la velocidad angular instantánea en grados cegesimales/seg en la que su eje de referencia varía. Se requiere por tanto integrar numéricamente la señal del giróscopo mediante la ejecución de una subrutina en el microprocesador del NXTBrick de LEGO.

Es importante señalar que cuanto mayor sea la frecuencia de muestreo al sensor así como el orden del método numérico empleado, mayor debería ser la precisión del

proceso de integración y por tanto menor la deriva o incremento del error de medida con el tiempo. Este error de deriva se puede minimizar aunque no es posible evitar.

En el caso concreto de nuestro NXTway-G, el ángulo de inclinación del cuerpo en el tiempo vendrá calculado respecto al ángulo de referencia inicial por integración. Dado que le indicamos en el programa que lo estamos dejando en una posición teórica vertical ($\psi = 0$, si cuando iniciamos el programa de control no está en equilibrio el funcionamiento será deficiente al proporcionar una información inicial incorrecta.

Aunque no se ha implementado en este proyecto, es importante recordar la capacidad que proporciona el Filtro de Kalman a la hora de integrar la información proveniente de diferentes sensores o fuentes de información con diferentes incertidumbres en cuanto a su precisión.

2. EL GIROSCOPIO

Dado que el uso de un giroscopio no es muy común, introduciremos en primer lugar los principios básicos en los que se basa, sus aplicaciones y los nuevos desarrollos electrónicos de giróscopos.

En la imagen 3.14 se puede ver un ejemplo de giróscopo clásico. Este es un dispositivo mecánico formado esencialmente por un cuerpo con simetría de rotación que gira alrededor de su eje de simetría. Cuando se somete el giroscopio a un momento de fuerza que tiende a cambiar la orientación del eje de rotación su comportamiento es aparentemente paradójico ya que el eje de rotación, en lugar de cambiar de dirección como lo haría un cuerpo que no girase, cambia de orientación en una dirección perpendicular a la dirección “intuitiva”.

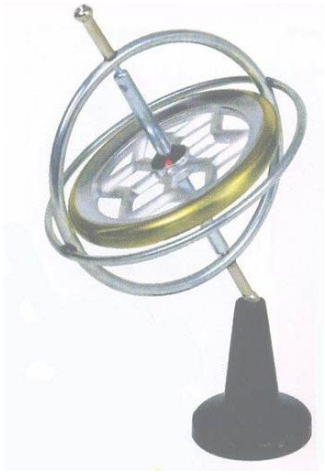

Imagen 3.14 Giróscopo mecánico clásico

El giroscopio fue inventado en 1852 por Léon Foucault montando una masa rotatoria en un soporte de Cardano para un experimento de demostración de la rotación de la tierra. Foucault también se dio cuenta de que su aparato podía servir para indicar el Norte (si se impiden ciertos movimientos del soporte del giroscopio, este se alinea con el meridiano), siendo el precursor de la invención del girocompás.

Los giroscopios se han utilizado en girocompases y giropilotos. Los giroscopios también se han utilizado para disminuir el balanceo de navíos, para estabilizar plataformas de tiro y para estabilizar plataformas inerciales sobre las cuales están fijados captadores de aceleración para la navegación inercial en aviones y misiles construidos antes de la aparición del GPS, es decir, en sistemas de control relativamente complejos.

Como curiosidad, decir que un giróscopo clásico para una aeronave, dependiendo de su calidad podía tener un precio entre los 10.000 y los 100.000 dólares.

En la actualidad se han desarrollado giróscopos electrónicos de más bajo coste, lo que ha permitido usarlos en nuevas aplicaciones en las que antes eran descartados por su elevado precio.

2.1. EL GIROSCOPIO ELECTRÓNICO

Los **giróscopos electrónicos** como el mostrado en la Imagen 3.15 son más recientes, consiguiendo unos precios muy inferiores a los mecánicos, consumos muy bajos y precisión elevada para su precio.

Imagen 3.15 Giróscopo electrónico piezoeléctrico

El giróscopo piezoeléctrico de un canal se utiliza habitualmente en aviones y helicópteros de radiocontrol para compensar de forma automática los giros bruscos.

El giróscopo es completamente electrónico, sin partes móviles, lo que hace que sea muy pequeño, tenga un bajo consumo y una respuesta muy rápida. La entrada del giróscopo se conecta en lugar del servo que se quiere compensar y el servo se conecta a su vez al giróscopo. Tan pronto como se produce un movimiento angular (por ejemplo inclinación) el giróscopo manda al servo una señal proporcional para compensar el movimiento. Por ejemplo, aplicado al rotor de cola de un helicóptero, cuando el helicóptero acelera, la cola tiende a girar lateralmente como consecuencia del par rotor. Si se utiliza el giróscopo en este caso, lo que hace es mandar una señal de control al servo que controla la cola, para aumentar el paso y compensar la desviación.

En el caso de los robots, la salida en lugar de conectarse a un servo, se conecta a un microcontrolador que puede leer la señal del giróscopo y así saber cuando se produce un giro. Los giróscopos son esenciales en los robots de tipo balancín así como en los sistemas de guiado de precisión en el que hay que medir y compensar el momento de giro.

Por tanto, para poder mantener el NXTway-G en posición vertical se necesita una referencia de lo que es "vertical o $\psi = 0$".

El giróscopo fabricado por HiTechnic nos proporciona la velocidad angular en grados/s y se han de considerar las siguientes características de operación:

<u>Offset</u>

Se puede definir el offset del sensor como el valor que proporciona el giróscopo cuando su velocidad angular es nula (0 grados/s).
Todos los giróscopos tienen un offset cuyo valor es específico de cada unidad y que puede variar con la temperatura u otros factores internos como tolerancias de fabricación.
En el giroscopio utilizado para el proyecto, se han detectado diferentes offsets que van desde 593 a 596 según diferentes momentos en los que se ha comprobado, es decir, 3 grados de diferencia absoluta.

Por tanto, antes de usar el sensor es preciso realizar una medida promedio para calcular el valor del offset en ese momento en particular. Posteriormente, para calcular el valor neto de velocidad angular (el que es realmente util), se requiere restar al valor bruto proporcionado por el sensor el offset calculado anteriormente.

Rango de Medida

El sensor detecta el giro en un solo plano de trabajo, con un rango de **± 360 grados/s,** es decir, indica también el sentido de la velocidad angular. Es por tanto clave colocar el sensor adecuadamente sobre el NXTway para que mida la velocidad angular de inclinación del cuerpo.

Configuración

Para poder acceder al sensor, es necesario definir en cuál de los 4 puertos de entrada está conectado así como cargar en el compilador las librerías particulares para los sensores de HiTechnic.

3.4. Descripción de los Servomotores

Con el Kit de LEGO Mindstorms NXT se reciben 3 servomotores, aunque solo se usarán 2 de ellos para accionar las ruedas del NXTway-G. Estos motores son mucho mayores que las versiones anteriores (RXC) y aunque son menos eficientes los valores de par de salida son significativamente mejorados.

Una gran ventaja es la de integrar internamente un **encoder incremental óptico** y de este modo poder enviar las señales de realimentación de posición por el mismo cable de potencia. La resolución del encoder es de 1 grado, es decir, 360 pulsos por revolución. El firmware ya tiene implementados unos contadores de pulsos, que permiten tanto la lectura como escritura de estos registros.

En la última versión de RobotC existe una función ya implementada que permite sincronizar el giro de los motores, de moto que uno actúa como maestro y el otro hace el seguimiento de este como esclavo. Esta función es especialmente útil cuando se requiere que el robot vaya en línea recta a pesar de diferentes perturbaciones o pares resistentes entre ambas ruedas, derivando esto en un giro indeseado aún aplicando la misma alimentación sobre ambos motores.

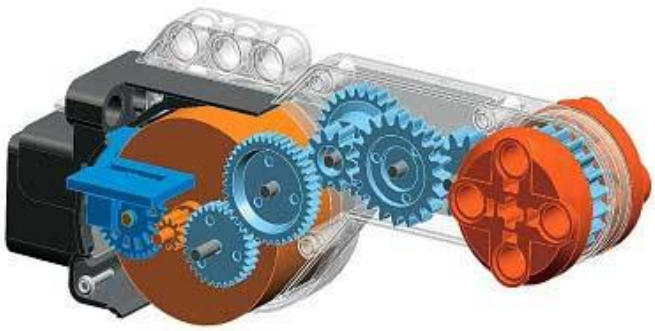

Imagen 3.16 Vista interior servomotor – Sistema de reducción por tren de engranajes.

Las imágenes 3.6, 3.7 y 3.8 muestran mediante gráficas el comportamiento de los servomotores con diferentes tensiones y cargas aplicadas.

Motor sin carga con tensión de 9V.

Motor sin carga con tensión de 7,2 V. Como el lógico la velocidad de giro es proporcionalmente inferior.

Par resistente de 11.5 N.cm y 9V sin Power Control.

En caso de no usar Power Control, se observa como el motor está bloqueado hasta que no se aplica una señal superior al 40%.

Par resistente de 11.5 N.cm y 9V con Power Control. Hasta un 70% de tensión aplicada, la velocidad es la misma que un motor sin carga. A partir de este valor el motor ya está a plena potencia y por tanto no responde aunque se incremente la entrada al motor.

Par resistente de 11.5 N.cm y 7.2V en alimentación. Hasta un 50% de señal a los motores.

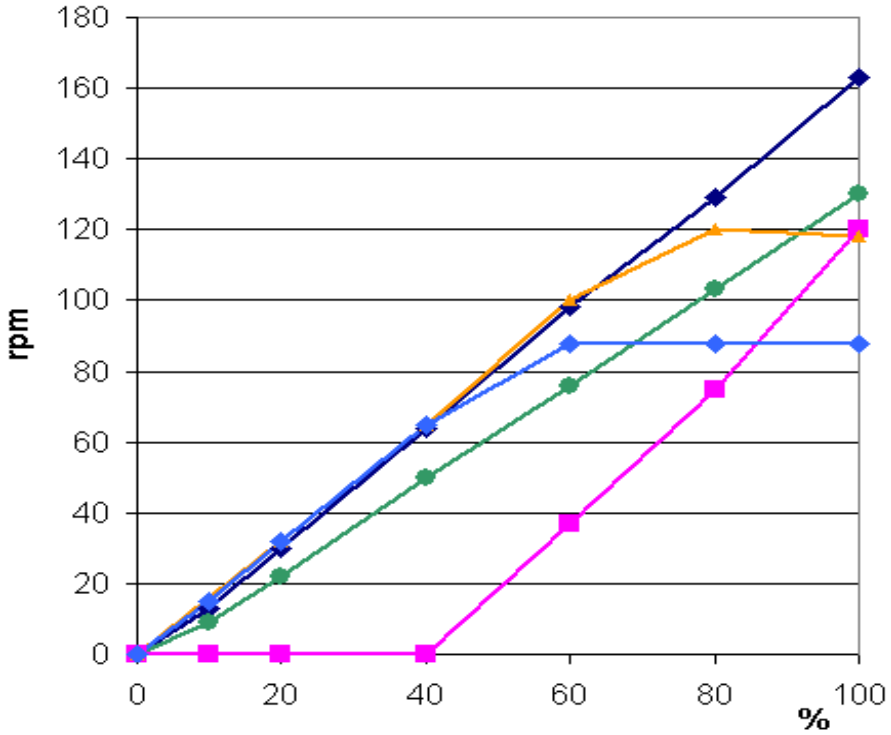

Imagen 3.17 Velocidad angular vs tensión para diferentes cargas

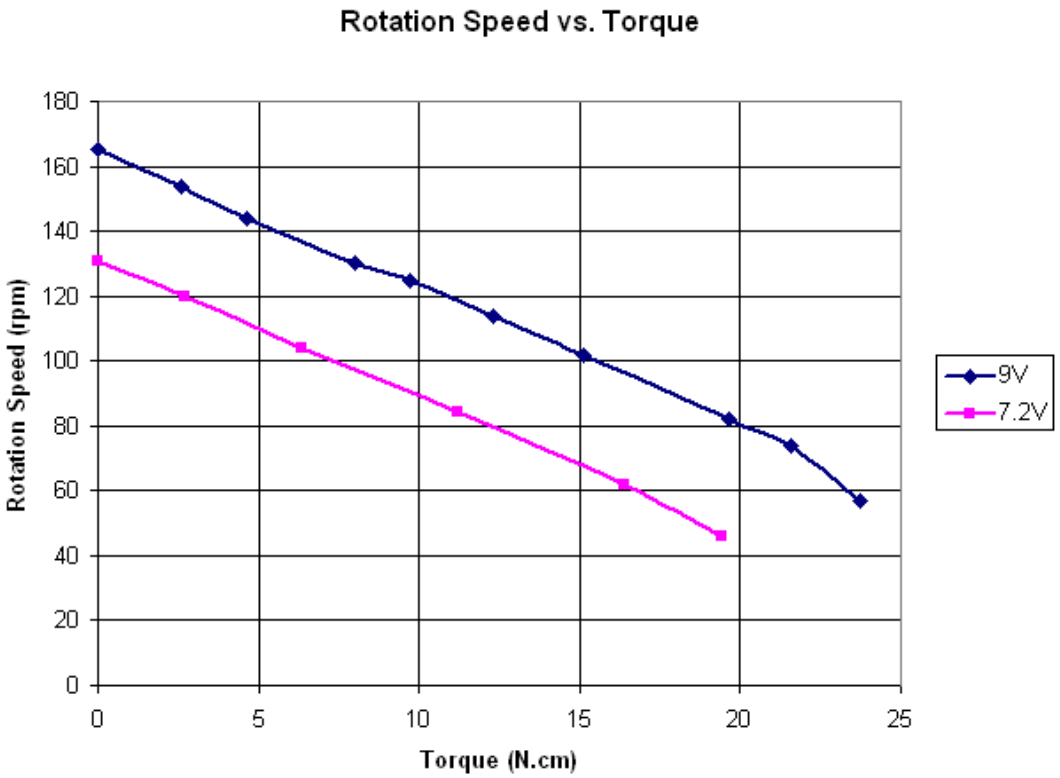

Imagen 3.18 Par disponible vs velocidad angular motor.

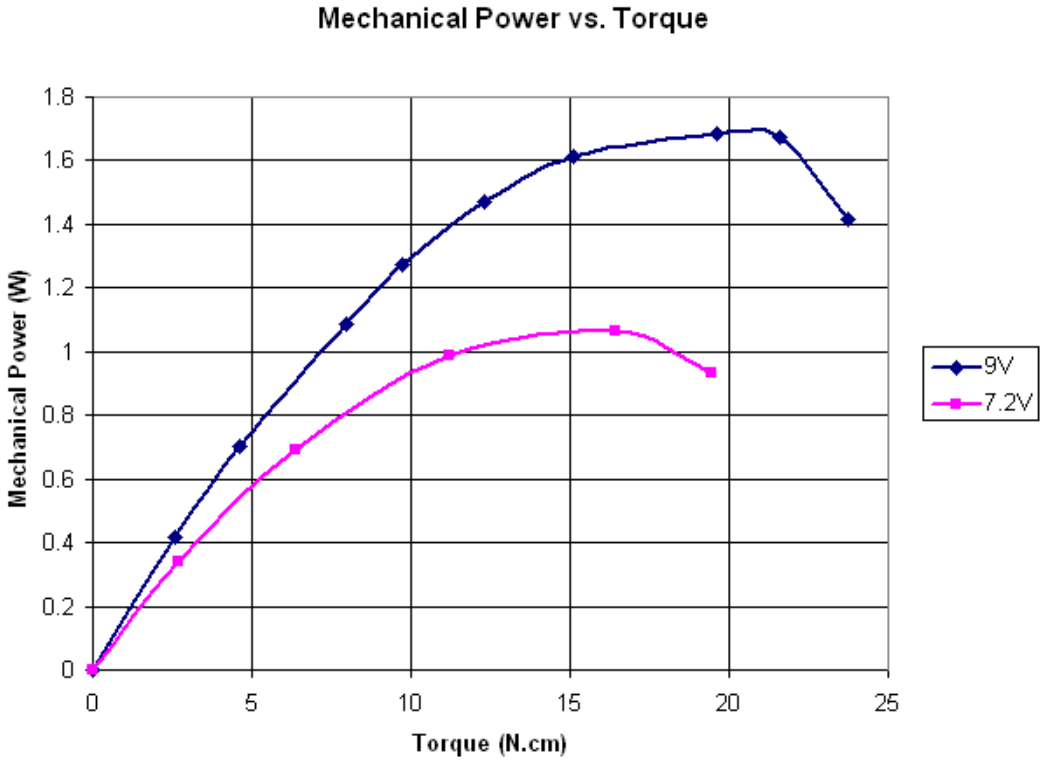

Imagen 3.19 Potencia vs Par para las 2 tensión de alimentación típicas

4. Software de Programación para NXTBrick

4.1. Introducción

Existen varias posibilidades a la hora de seleccionar el software que se va a utilizar para la programación del microprocesador del NXTBrick de LEGO. En este proyecto se ha seleccionado inicialmente 2 de los más comunes y utilizados en el mundo de la programación a bajo nivel de los minirobots.

El primero es un compilador gratuito denominado Bricx Command Center 3.3 (Brixc CC) desarrollado por John Hansen para adecuar la evolución del antiguo lenguaje NQC hacia el nuevo lenguaje NXC (Not eXactly C).

El segundo es un software más profesional y potente de bajo coste ($30/licencia) denominado RobotC for Mindstorms 3.5.4 y que ha sido desarrollado por Carnegie Mellon Robotics Academy (Universidad de Carnegie en Pittsburg, Pensilvania).
http://www.robotc.net/
http://www.education.rec.ri.cmu.edu/content/LEGO/index.htm

4.2. Comparativa RobotC vs BricxCC

Ambos programas tienen un formato IDE (Entorno de Desarrollo Integrado) y permiten la programación de el microprocesador Atmel (32-bit ARM7 @ 48MHz) existente en el NXTBrick de LEGO.

Tabla comparativa entre RobotC y BricxCC

	RobotC	BricxCC
Precio (Mellon /LEGO) por licencia	$30/$49 *	Gratuito
Lenguaje de Programación	ANSI C	NXC
Operaciones en Coma Flotante	Sí	No
Operaciones Trigonométricas	Sí	No
Debugger Disponible	Sí	No
Visualización Variables online	Sí	No
Incorporan soporte de nuevos sensores de terceras empresas	Sí	Sí
Soporte Comunicación Bluetooth	Sí	Sí
Funciones uso Joystick	Sí	No
Soporte	Sí	Foros
Información Disponible	Foros	Foros/Libros

*Se puede descargar versión de evaluación gratuita de 30 días desde su página Web.

La imagen 5.1 da una idea de la potencia de RobotC como herramienta para programación de aplicaciones más avanzadas, y en este caso en particular del robot ganador del concurso mundial de First Tech Challeng (FTC).

Imagen 5.1 Robot ganador en la World Championship FTC programado con RobotC

4.2.1. RobotC para este proyecto

La conclusión final tras el manejo de ambos programas es que para aplicaciones de cierta envergadura o complejidad, BricxCC queda descartado a favor de RobotC especialmente por disponer de operaciones en coma flotante, debugger y monitorización de variables. Indicar que si se requiere realizar operaciones (divisiones con decimales, operaciones trigonométricas,…) BricxCC no es operativo. En cuanto a las otras características hacen el desarrollo más fiable y cómodo.

4.3. Toolbox de Matlab para LEGO Mindstorms.

La orientación típica en la utilización del sistema LEGO Mindstorms ha sido la de implementar físicamente las máquinas o robots, programar en pseudoC los algoritmos de control offline y por último descargar y ejecutar en el microcontrolador embebido en el NXTbrick dicho programa.

"MATLAB® meets LEGO Mindstorms".

Con este proyecto, la universidad de Aachen en Alemania decidió dar un paso más en este sistema LEGO Mindstorms implementando un Toolbox que permiten a Matlab interactuar con el robot vía UBS y/o Bluetooth y de este modo poder utilizar los Robots de LEGO para las prácticas de laboratorio de Métodos numéricos y procesado de señales. Ahora es posible utilizar de modo más práctico librería tan

útiles como Image Processing Toolbox™, Optimization Toolbox™, and Signal Processing Toolbox™.

Inicialmente los alumnos construyen las máquinas o robots con los componentes de LEGO Mindstorms NXT education kit y posteriormente realizan los programas en Matlab que permite interactuar con el microprocesador del NXT vía Bluetooth y controlar la máquina diseñada.

TOOLBOX "**RWTHMindstormsNXT**"

En la versión 2.0 del toolbox el sistema es compatible con comunicación Bluetooth y USB y para sistemas operativos Windows o Linux.

En esencia, sus aplicaciones son programas secuenciales en las que la lógica es implementada en Matlab a partir de la información que le envía el NXT Brick de sus sensores y controla la operación de sus motores.

4.4. Software de LEGO

Junto con el robot, LEGO proporciona un CD con el software de programación gráfica basado en LabView. Este software es adecuado para niños o aplicaciones que requieran poca complejidad y cálculos. En caso contrario se vuelve poco manejable o inservible.

Al instalar este software, se instalan automáticamente los drivers para Windows denominados "Fantom". En caso de no querer instalar el software se deben descargar de la página Web de LEGO e instalarlos en el PC.

http://mindstorms.LEGO.com/en-us/support/files/Driver.aspx

La última versión instalada es Fantom Driver 1.1.3

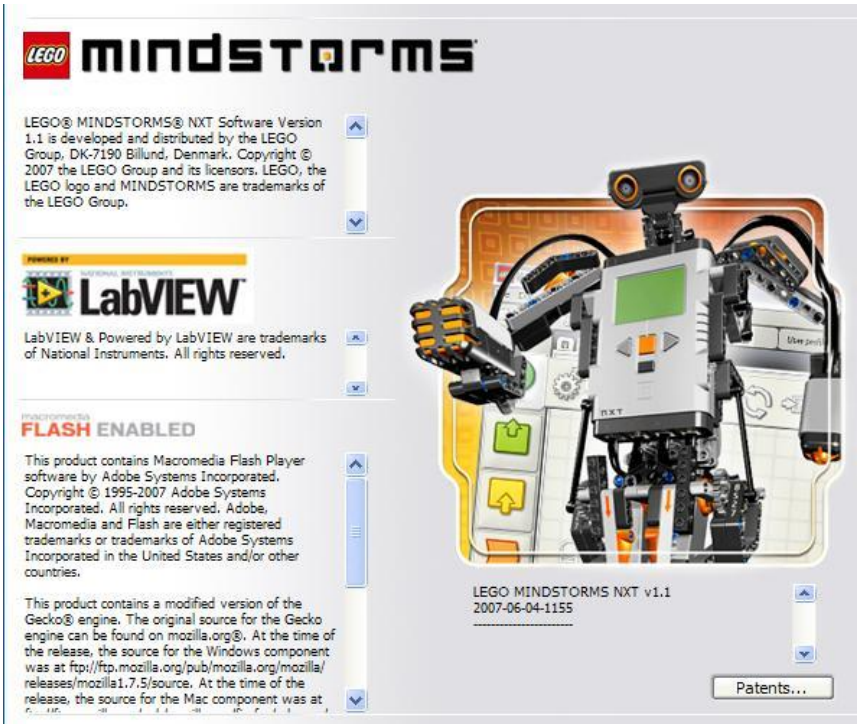

Imagen 4.2 Versión disponible LEGO Mindstorms NXT v1.1.

Ya está disponible para la venta la versión mejorada LEGO Mindstorms NXT v2.0.

4.5. Software de Programación RobotC 3.54

La última versión de RobotC (Ver. 3.54) se ha descargado desde la página web de la Universidad Carnegie Mellon y corre sobre PC con Windows XP SP3. La licencia de prueba es totalmente funcional y tiene una validez de 30 días.

http://www.education.rec.ri.cmu.edu/content/LEGO/index.htm

4.5.1. Firmware en NXTBrick

El firmware es el sistema operativo que corre en el NXTBrick. Este nos permite interaccionar con los sensores/actuadores/pantalla, proporcionar las herramientas del API que incorpora los comandos y funciones para la programación y ejecuta los programas ya compilados y residentes en la memoria del NXT.

Una ventaja importante del NXT respecto a anteriores versiones es que el firmware está almacenado en la memoria flash de 64K, y por tanto no se pierde aunque se retiren las baterías.

El NXTBrick de LEGO viene con un firmware propio que es actualizable. Para poder utilizar el software de RobotC se ha de descargar previamente el firmware propio desarrollado por RobotC y transferirlo al NXTBrick vía cable USB. Si se dispone de conexión a Internet, la actualización se puede realizar de modo automático desde RobotC accediendo al menú Robot →Download Firmware.

Imagen 4.3 Licencia inicial de RobotC comprada por la UPCT

4.5.2. Conexión por Bluetooth

Mediante Bluetooth, RobotC permite cargar/descargar los programas entre el PC←→NXTBrick, visualizar los valores de las variables del programa en ejecución así como interaccionar con los comandos del Joystick que permiten control remoto. Se utilizará igualmente para enviar comandos, valores de variables y recuperar archivos e datos almacenados de las pruebas de desempeño. Es notable la comodidad que ello proporciona al no tener que estar constantemente conectando y desconectando el cable USB en el periodo de pruebas.

Bluetooth Dongle

Para permitir la comunicación online entre PC ←→NXTBrick se requiere disponer de Dispositivo de Comunicación Bluetooth (ya sea interno de portátil o mediante un lápiz USB-Bluetooth compatible con LEGO).

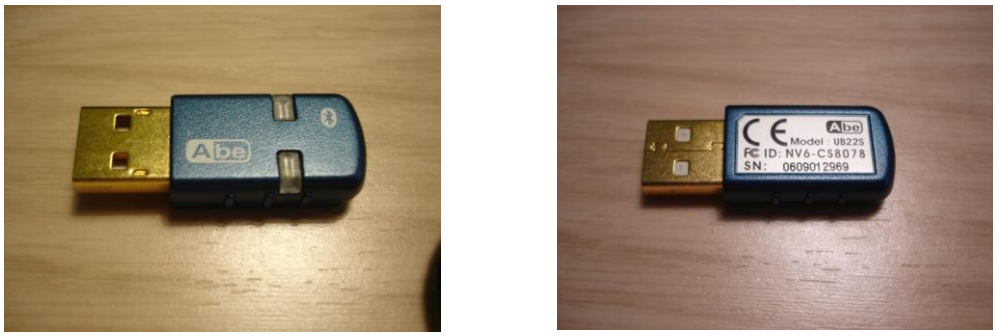

Imagen 4.4 Vista superior en inferior de Bluetooth dongle recomendado por LEGO.

A pesar de que Bluetooth 2.0 es un estándar, no todos los dispositivos Bluetooth son compatible para comunicar con el NXT, por lo que antes de comprar un adaptador Bluetooth es conveniente ver en los foros los que ya están probados y funcionan o comprar el que vende LEGO que garantiza la compatibilidad.

Un requerimiento es que han de soportar los servicios SPP (Serial Port Profile), es decir, que emulen el comportamiento de un puerto serie convencional. Se ha comprobado que hay incompatibilidades con la pila (stack) utilizada por el driver de Bluetooth para Windows.

Para este proyecto se ha usado el Adaptador USB-Bluetooth que distribuye LEGO.

Los detalles de este dispositivo son los siguientes:

Fabricante:	Abe™
Denominación:	USB Bluetooth Dongle
Modelo:	UB22S
Versión Bluetooth:	v2.0 + EDR
Alcance:	10 mts

Conexión PC <-> NXTBrick por Bluetooth

El Dongle adquirido no va provisto del CD con su driver por lo que se utiliza el que hay por defecto en Windows XP.

Al conectar este dispositivo a un puerto USB cualquiera, Windows lo reconoce automáticamente e instala los drivers apropiados.

Tras reiniciar el PC hay que añadir el NXT como nuevo dispositivo.

Para esto seguimos los siguientes pasos:

1. Encendemos el NXTBrick de NXT
2. Entramos en su Menú "Bluetooth"
    a. Activamos Bluetooth del NXT dentro del menú "ON/OFF" a ON.
    b. Activamos Visibilidad en el submenú "Visibility" activando "Visible".

Hecho esto, en el display LCD general deberá aparecer como se muestra en las imágenes inferiores.

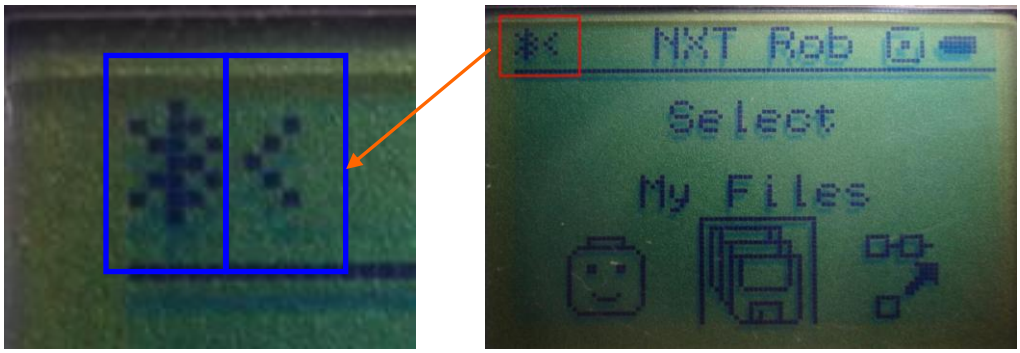

Imagen 4.5 Detalle en display LCD del icono Bluetooth y visibilidad activada

El primer símbolo en la figura de la derecha en recuadro azul (LCD display) de la imagen indica que Bluetooth está activo y el segundo que es visible para otros

dispositivos. Esto último es necesario para que podamos detectarlo desde el PC y añadirlo como nuevo dispositivo.

3. Añadimos el NXT en el Administrador de Dispositivos Bluetooth (Panel de Control – Dispositivos Bluetooth).

    a. Agregar dispositivo
    b. Seleccionar opción "Usar clave de paso que está en la documentación" e introducir la clave "1234". Ver imagen 4.6
    c. Esperar a que Windows detecte NXT
    d. Verificar que está en la lista de dispositivos

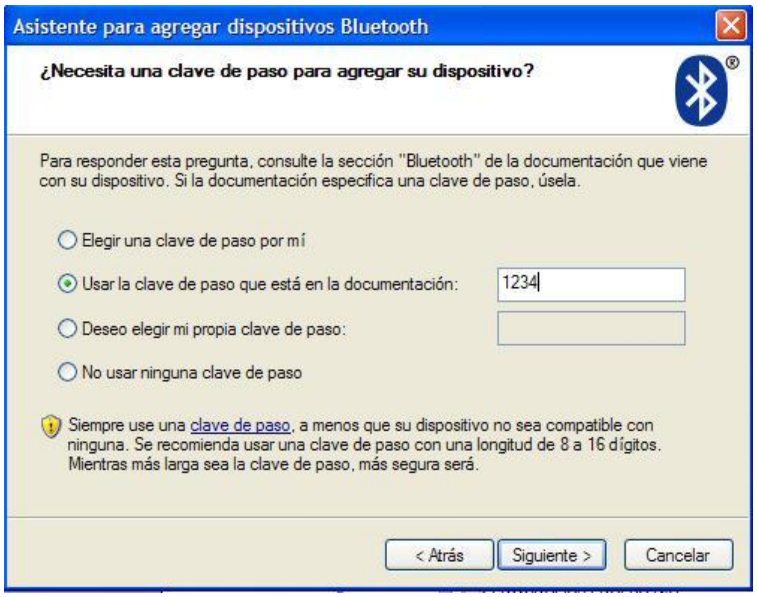

Imagen 4.6 Clave usada para conexión vía Bluetooth

4. Verificamos que el servicio SPP (imprescindible para la comunicación NXT←→PC) tiene la casilla habilitada y el número de puerto serie asignado. Esta información está disponible dentro del Administrador de Bluetooth en Propiedades→Servicios.

Indicar que este puerto serie es un puerto de comunicación virtual y puede tener valores que en principio pueden sorprendernos como por ejemplo COM 37.
Este valor es asignado automáticamente por Windows.

Cada vez que se elimina el dispositivo de la lista y se vuelve a conectar, Windows le asignará un valor mayor y no siempre consecutivo.

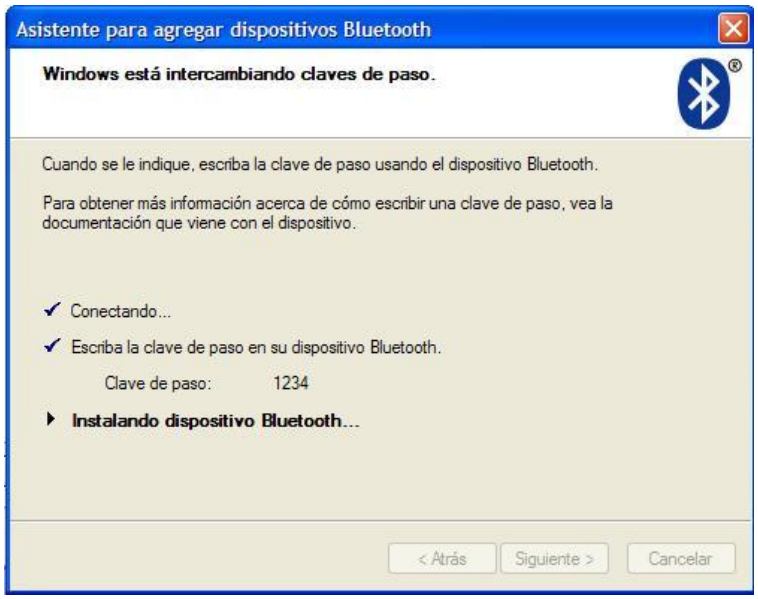

Imagen 4.7 Proceso configuración de dispositivo Bluetooth

Imagen 4.8 NXT Robot ya reconocido como dispositivo conectado

5. De los 2 puertos que aparecen en la imagen 4.9 se utilizará el puerto de salida, que es el que tiene la compatibilidad SPP.

En esta imagen vemos que Windows ya ha detectado el Robot NXT como nuevo dispositivo y que tiene la clave de paso habilitada. La clave por defecto es "1234".

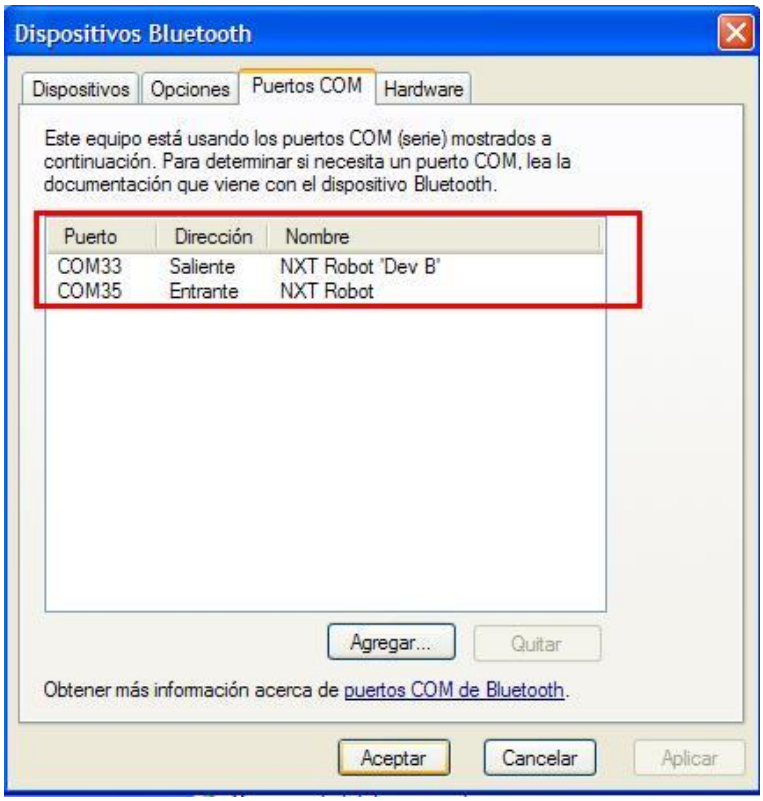

Imagen 4.9 Puertos COM asignados para conexión Bluetooth

5. Modelización y Control

5.1. Introducción

La teoría de control moderna está basada en la representación matemática de los sistemas dinámicos por medio del concepto de estado, el cual permite un mayor conocimiento del comportamiento interno de dichos sistemas.

Este conocimiento más profundo por parte del controlador del sistema permite realizar un control más potente que la teoría clásica de control, donde la utilización de la señal de salida como única fuente de realimentación empobrece la información disponible por el regulador para controlar dicho sistema.

Las principales ventajas del la teoría de control moderna frente a la clásica son que permiten los siguientes tipos de control avanzado:

- **Control Multivariable**.
  Es aplicable a sistemas multivariables en los que existen un elevado grado de interacción de los sistemas (MIMO) frente a los sistemas de control clásicos que están restringidos a una entrada una salida (SISO).

- **Control no lineal.**
  Es aplicable a sistemas con relaciones no lineales entre sus variables.

- **Control Adaptativo.**
  Es aplicable a sistemas cuyos parámetros varían en el tiempo.

- **Control Óptimo.**
  Es aplicable la optimización del comportamiento del sistema.

Como se puede observar, la teoría de control moderna ha sido un gran avance al permitir controlar sistemas que antes de su aparición estaban fuera del alcance o con desempeños peores que los conseguidos en la actualidad.

En la teoría clásica se usa la función de transferencia en el dominio de la frecuencia para describir la dinámica de un sistema lineal e invariante en el tiempo (LTI). Dicha función es única y proviene de la transformación de la ecuación diferencial que describe el sistema a una ecuación algebraica mediante la transformada de Laplace, lo que nos hace la vida más fácil a la hora de estudiar y resolver dichos sistemas.

En la teoría moderna, la representación del sistema en el espacio de estados es un formalismo para el tratamiento y resolución de sistemas dinámicos deterministas.

La representación en el espacio de estados de un mismo sistema no es única, y depende de las variables de estado elegidas. Lo que si se garantiza es que todas ellas son equivalentes y relacionadas por transformaciones lineales en el espacio de estados (cambio de las bases de los vectores de estado).

La representación de un sistema LTI (o linealizado) en el espacio de estados viene dado por:

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

Siendo:
A → Matriz del sistema (n x n)
B → Matriz de entradas (n x m)
C → Matriz de salidas (p x n)
D → Es nula en la mayoría de sistemas (p x n).

Una representación gráfica vectorial del modelo de estado se puede apreciar más abajo en la imagen 5.2:

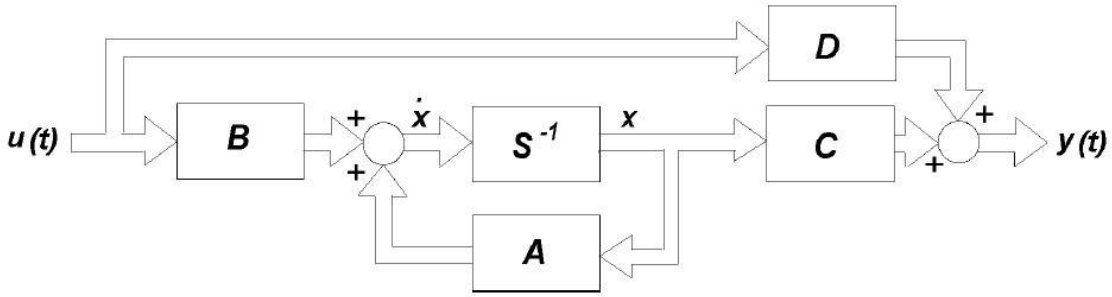

Imagen 5.2 Esquema de bloques – Representación Sistema en el Espacio de Estados

5.2. Obtención del Modelo de Estado

Una vez que se ha modelizado matemáticamente el sistema, es decir, ya disponemos de las ecuaciones diferenciales que los describen, tenemos que decidir que variables de estado elegimos de entre las infinitas opciones disponibles.

De modo global, las variables de estado pueden elegirse según estos criterios:

1) Variables de estado como magnitudes físicas del sistema
2) Variables de estado como salidas de los integradores del sistema
3) Variables de estado de fase
4) Variables de estado de Jordan

Estas variables están ordenadas en sentido decreciente con su significado físico y creciente en simplicidad de representación matemática.

La primera opción es más genérica y se puede aplicar a sistemas lineales o no lineales.
Las dos últimas pueden aplicarse a sistemas lineales.
La segunda opción permite elegir variables que puedan tener sentido físico y que a la vez se pueda sistematizar su elección para la obtención de un modelo de estado sencillo.

5.3. Modelo Simplificado del NXTway-G

Con el objetivo de simplificar el sistema para poder modelizarlo de modo más sencillo, se asumirá lo siguiente:

- El cuerpo del robot se asemeja a un poliedro de volumen W x H x D (anchura, altura y espesor respectivamente).
- El centro de masas del cuerpo se encuentra en el centro del poliedro a una distancia H/2.
- El momento de inercia de las ruedas se calcula considerando la masa concentrada en su perímetro.

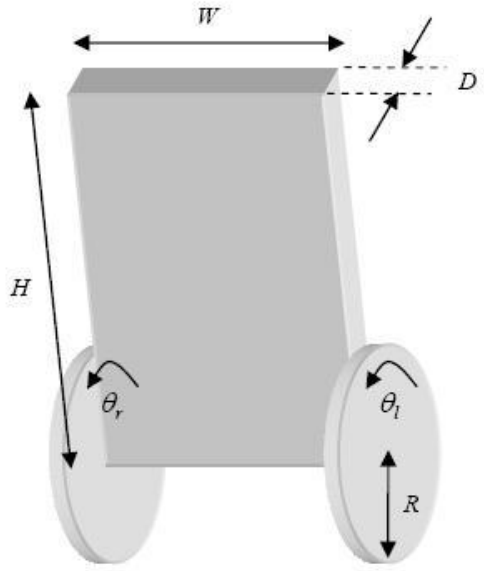

Imagen 5.3 Esquema simplificado del modelo NXTway-G

Centro de Masas del Cuerpo.

Para determinar la posición del centro de masas del cuerpo, se ha colocado el robot (sin ruedas) sobre una línea de apoyo paralela al eje de las ruedas y se ha desplazado hasta encontrar el punto de equilibrio, se ha marcado dicho punto y se ha medido la distancia al eje de las ruedas. El valor obtenido es de 140 mm.
Este método empírico es mucho más sencillo y fiable que el cálculo analítico dada da diversidad de componentes que componen el cuerpo (Motores, barras conexión, sensores,…).

5.3.1. Parámetros del Robot

Parámetros	Unidades	Descripción
g = 9.81	m/s^2	Aceleración de la gravedad
m = 0.03	Kg	Peso de cada rueda
R = 0.045	m	Radio de Rueda
M = 0.6	Kg	Masa de Péndulo
W=0.142	m	Ancho de Cuerpo
D = 0.045	m	Grosor de Cuerpo
H = 0.280	m	Altura de Cuerpo
L = 140	m	Distancia del CM al eje de ruedas
Parámetros	**Unidades**	**Descripción**
$J_{\psi} = (\frac{ML^2}{3})$	$Kg*m^2$	Momento de Inercia Inclinación Cuerpo
$J_{\Phi} = M(\frac{W^2 + D^2}{12})$	$Kg*m^2$	Momento de Inercia Cuerpo respecto Eje Vertical
$J_W = m(\frac{R^2}{2})$	$Kg*m^2$	Momento de Inercia de Rueda
$J_m = 1*10^{-5}$	$Kg*m^2$	Momento de Inercia de Motor DC
$R_m = 6.69$	Ω	Resistencia Eléctrica Motor DC
$K_b = 0.468$	(V* s/rad)	Fuerza contraelectromotriz Motor DC
$K_t = 0.317$	Nm/A	Constante de Par de Motor DC
n = 1		Relación de Transmisión
$f_m = 0.0022$		Coeficiente rozamiento Cuerpo – Motor DC
$f_w = 0$		Coeficiente rozamiento Rueda –Suelo

Tabla 5.1 Valores principales del NXTway-G para obtener modelo matemático.

1. **Momento de Inercia del Cuerpo respecto al Eje de las Ruedas.**

   Se ha simplificado suponiendo que el cuerpo es una caja homogénea de medidas 280 x 140 x 45 mm. Ver imagen 5.3.

El momento de inercia respecto a un eje de las ruedas viene dado por:

$$J_{\psi} = (\frac{ML^2}{3}) \quad \rightarrow \quad J_{\psi} = 0.6(\frac{0.14^2}{3}) = 3.92*10^{-3} \ \text{Kg*m}^2 \qquad (5.1)$$

2. Momento de Inercia del Cuerpo respecto a su Eje Central Vertical

$$J_{\Phi} = M(\frac{W^2 + D^2}{12}) \quad \rightarrow \quad J_{\Phi} = 0.6(\frac{0.14^2 + 0.045^2}{12}) = 1.08*10^{-3} \ \text{Kg*m}^2 \qquad (5.2)$$

3. Momento de Inercia de Ruedas

Para el cálculo del momento de inercia de las ruedas se han pesado estas y se ha supuesto que el momento de inercia es un anillo

$$J_W = m(\frac{R^2}{2}) \quad \rightarrow \quad J_W = 0.03(\frac{0.045^2}{2}) = 3.038*10^{-5} \ \text{Kg*m}^2 \qquad (5.3)$$

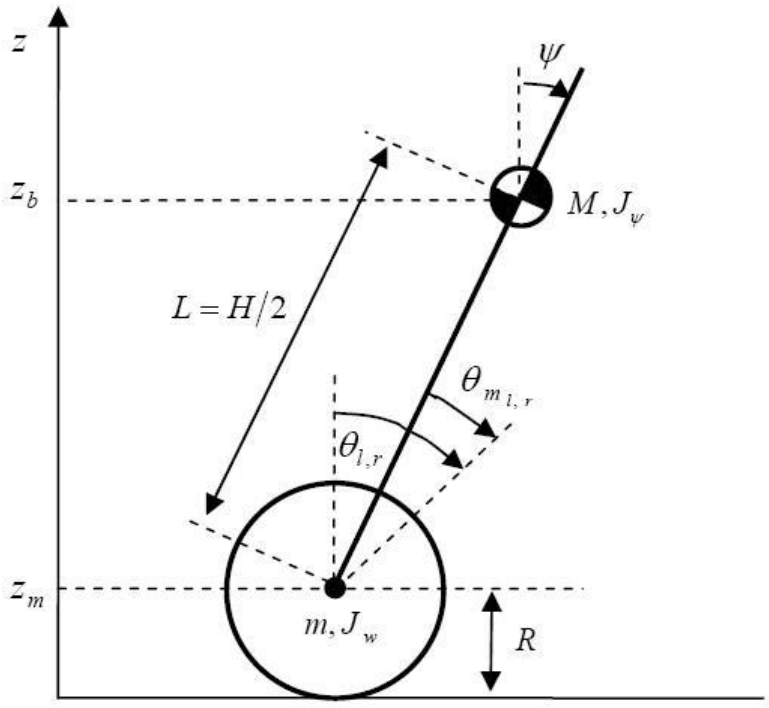

Imagen 5.4 Esquema con parámetros del modelo simplificado. Vista lateral.

ψ : Ángulo de inclinación de Cuerpo

$\theta_{l,r}$: Ángulo de giro de Ruedas (l = Izq. , r = derecha).

$\phi_{ml,\,mr}$: Ángulo de giro de Motor DC (l = Izq. , r = derecha)

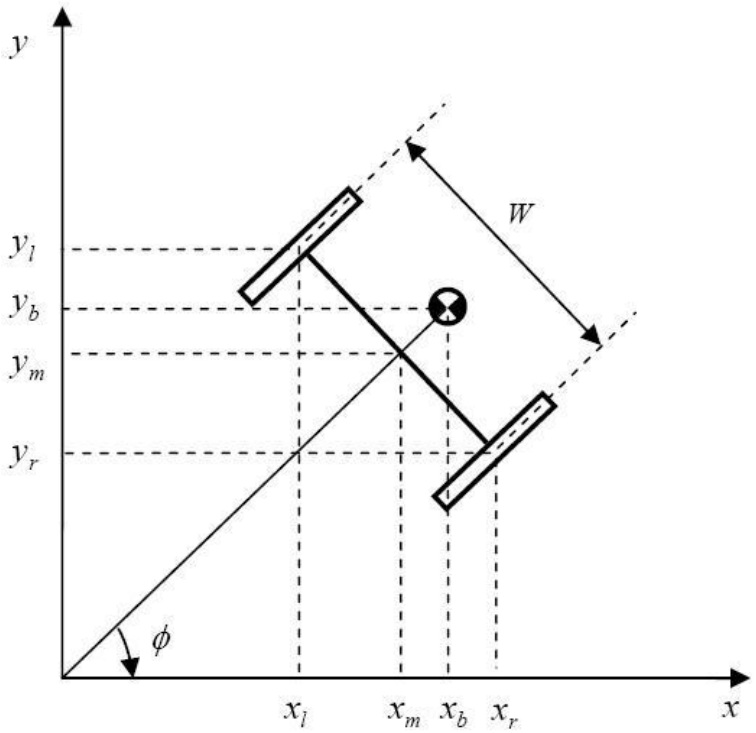

Imagen 5.5 Esquema con parámetros del modelo simplificado. Vista planta.

5.4. Ecuaciones Diferenciales del Sistema

Ecuaciones para el movimiento de las ruedas:

Obtenemos el valor de θ y ϕ para la aproximación de giros de ruedas pequeños:

$$(\theta,\phi) = (\frac{1}{2}(\theta_l + \theta_r), \frac{R}{W}(\theta_r - \theta_l)) \tag{5.4}$$

Coordenadas del CM del conjunto (rueda izq. + der.):

$$(x_m, y_m, z_m) = (R\theta\cos\phi, R\theta\sin\phi, R) \tag{5.5}$$

Coordenadas del CM de la rueda izquierda:

$$(x_l, y_l, z_l) = (x_m - \frac{W}{2}\sin\phi, y_m + \frac{W}{2}\cos\phi, z_m) \tag{5.6}$$

Coordenadas de CM de la rueda derecha:

$$(x_r, y_r, z_r) = (x_m + \frac{W}{2}\sin\phi, y_m - \frac{W}{2}\cos\phi, z_m) \tag{5.7}$$

Coordenadas de CM del cuerpo:

$$(x_b, y_b, z_b) = (x_m + L\sin\psi\cos\phi, y_m + L\sin\psi\sin\phi, z_m + L\cos\psi) \tag{5.8}$$

Obtendremos las ecuaciones dinámicas del sistema por el método de Lagrange, que es más adecuado que las ecuaciones de Newton (mecánica clásica) dado que se tiene un sistema de referencia no inercial.

La energía cinética de translación viene dada por:

$$T_1 = \frac{1}{2}m((\dot{x}_l)^2 + (\dot{y}_l)^2 + (\dot{z}_l)^2) + \frac{1}{2}m((\dot{x}_r)^2 + (\dot{y}_r)^2 + (\dot{z}_r)^2) + \frac{1}{2}M((\dot{x}_b)^2 + (\dot{y}_b)^2 + (\dot{z}_b)^2) \tag{5.9}$$

La energía cinética de rotación viene dada por:

$$T_2 = \frac{1}{2}J_w(\dot{\theta}_l)^2 + \frac{1}{2}J_w(\dot{\theta}_r)^2 + \frac{1}{2}J_\psi(\dot{\psi})^2 + \frac{1}{2}J_\phi(\dot{\phi}_l)^2 + \frac{1}{2}n^2J_m(\dot{\theta}_l - \dot{\psi})^2 + \frac{1}{2}n^2J_m(\dot{\theta}_r - \dot{\psi})^2 \tag{5.10}$$

La energía potencial viene dada por:

$$U = mgz_l + mgz_r + Mgz_b \tag{5.11}$$

El Lagrangiano viene dado por la expresión:

$$L = T_1 + T_2 - U \tag{5.12}$$

Las ecuaciones de Lagrange son:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = F_\theta \tag{5.13}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\psi}}\right) - \frac{\partial L}{\partial \psi} = F_\psi \tag{5.14}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}}\right) - \frac{\partial L}{\partial \phi} = F_\phi \tag{5.15}$$

$$\begin{aligned} F_\theta = {} & (2m+M)R^2 + 2J_w + 2n^2J_m\,\ddot{\theta} + (MLR\cos\psi - 2n^2J_m)\ddot{\psi} \\ & - MLR(\dot{\psi})^2\sin\psi - (2m+M)R^2\theta + MLR\sin\psi\,(\dot{\phi})^2 \end{aligned} \tag{5.16}$$

$$\begin{aligned} F_\psi = {} & (MLR\cos\psi - 2n^2J_m)\ddot{\theta} + (ML^2 + J_\psi + 2n^2J_m)\ddot{\psi} - MgL\sin\psi \\ & - (MLR\theta + ML^2\sin\psi)(\dot{\phi})^2\cos\psi \end{aligned} \tag{5.17}$$

$$\begin{aligned} F_\phi = {} & \left[\frac{1}{2}mW^2 + J_\phi + \frac{W^2}{2R^2}(J_w + n^2J_m) + (2m+M)R^2\theta^2 + 2MLR\theta\sin\psi\right]\ddot{\phi} \\ & + 2\left[(2m+M)R^2\,\dot{\theta}\theta + ML^2\,\dot{\psi}\sin\psi\cos\psi + MLR(\dot{\theta}\sin\psi + \theta\dot{\psi}\cos\psi\right] \end{aligned}$$

Las fuerzas que actúan son las siguientes:

$$F_l = nK_t i_l + f_m(\dot{\psi} - \dot{\theta}_l) - f_w \dot{\theta}_l \tag{5.18}$$

$$F_r = nK_t i_r + f_m(\dot{\psi} - \dot{\theta}_r) - f_w \dot{\theta}_r \tag{5.19}$$

$$F_\psi = -nK_t i_l - nK_t i_r - f_m(\dot{\psi} - \dot{\theta}_l) - f_m(\dot{\psi} - \dot{\theta}_r) \tag{5.20}$$

siendo i_r , i_l la corriente de los motores derecho e izquierdo respectivamente.

Dado que el control se realiza variando la tensión aplicada al motor de continua mediante PWM, se puede relacionar la intensidad con la tensión mediante la ecuación:

La fuerza de fricción viscosa es proporcional a la velocidad angular relativa entre la rueda y el cuerpo.

$$v_r = R_m i_r + L_m i_r + K_b(\dot{\psi} - \dot{\theta}_r) \tag{5.21}$$

$$v_l = R_m i_l + L_m i_l + K_b(\dot{\psi} - \dot{\theta}_l) \tag{5.22}$$

Si consideramos la inductancia no significativa, tenemos:

$$i_{r,l} = \frac{v_{l,r} + K_b(\dot{\psi} - \dot{\theta}_{l,r}}{R_m} \tag{5.23}$$

De este modo podemos finalmente expresar las fuerzas en función de la tensión aplicada a los motores:

$$F_\theta = \frac{\alpha}{2}(v_l + v_r) - (\beta + f_w)\dot{\theta} + \beta\dot{\psi} \tag{5.24}$$

$$F_{\psi} = -\alpha(v_l + v_r) + 2\beta\dot{\theta} - 2\beta\dot{\psi} \tag{5.25}$$

$$F_{\phi} = \frac{R}{W}\alpha(v_r - v_l) - (\beta + \frac{W}{R} f_w)\dot{\phi} \tag{5.26}$$

siendo:

$$\alpha = \frac{nK}{R_m} \tag{5.27}$$

$$\beta = \frac{nK_t K_b}{R_m} + f_m \tag{5.28}$$

5.5. Ecuación de Estado del sistema

Dado que estamos trabajando con la teoría moderna de control, representaremos el sistema es el espacio de estados. Las ecuaciones de la dinámica del sistema son no lineales por lo que el primer paso será linealizarlas en torno al punto de equilibrio, que en este caso será $\psi \cong 0$. (Cuerpo en posición vertical).

5.5.1. Linealización de ecuaciones en torno al punto de equilibrio.

Tenemos por tanto que cuánto $\psi \cong 0$ se puede aproximar:

a) $\sin\psi \cong \psi$ b) $\cos\psi \cong 1$

así como despreciar los términos de segundo orden del tipo $(\dot{\psi})^2$.

Obtenemos con esto las siguientes ecuaciones diferenciales linealizadas:

$$F_{\theta} = (2m + M)R^2 + 2J_w + 2n^2 J_m \ddot{\theta} + (MLR - 2n^2 J_m)\ddot{\psi} \tag{5.29}$$

$$F_{\psi} = (MLR - 2n^2 J_m)\ddot{\theta} + (ML^2 + J_{\psi} + 2n^2 J_m)\ddot{\psi} - MgL\psi \tag{5.30}$$

$$F_{\phi} = \left[\frac{1}{2} m W^2 + J_{\phi} + \frac{W^2}{2R^2} (J_w + n^2 J_m) \right] \ddot{\phi} \tag{5.31}$$

En los sistemas de seguimiento, es importante analizar si la planta es de tipo 0 o 1. Los de tipo 1 son aquellos que tienen un integrador en la trayectoria directa y que por tanto tenderán hacia un error de seguimiento nulo con el tiempo. En los casos que el servosistema no disponga de un integrador, se lo añadiremos artificialmente entre el comparador del error de posición y la planta.

Nota: Hay que tener en cuenta la posición de los ceros en origen de la planta, que pueden cancelar el integrador que estamos insertando.

En la ubicación de polos se ha de tener en cuenta las limitaciones prácticas que tiene el sistema como son:

- Velocidad lineal máxima (función del voltaje de la batería)
- Aceleración máxima (función de velocidad angular)

En las simulaciones con Matlab es posible obtener comportamientos suficientemente rápidos pero cuando observamos los que se le está requiriendo a los motores vemos que en la realidad no es posible obtener. Por ello la posición de los polos se han de ir situando de modo práctico hasta ver que en la simulación no se exceden las limitaciones de velocidad y par disponible exigidos a los motores.

5.6. Control en Simulación

El software Matlab es especialmente útil y potente a la hora de simular el comportamiento dinámico de un sistema ante entradas típicas como impulso, escalón, rampa así como ver la evolución del sistema ante condiciones iniciales no nulas.

5.6.1. Control PD para mantener verticalidad

En primer lugar, se ha implementado un lazo de control simple Proporcional Derivativo (en adelante PD) para intentar mantener la posición del robot en equilibrio vertical. En la imagen 5.9 puede verse la configuración de este lazo de control.

La gran ventaja del control PID tradicional es que no se requiere un modelo matemático del sistema para poder realizar un control aceptable y que por tanto no está sujeto a las imprecisiones o carencias que dicho modelo puede tener.

En nuestro caso el control ha resultado satisfactorio una vez que se han sintonizado los parámetros proporcional y derivativo (K_P y K_d respectivamente) para hacer el sistema lo suficientemente rápido para mantener el equilibrio pero con unas ganancias que no hagan vibrar el sistema de modo excesivo y lo vuelva inestable.

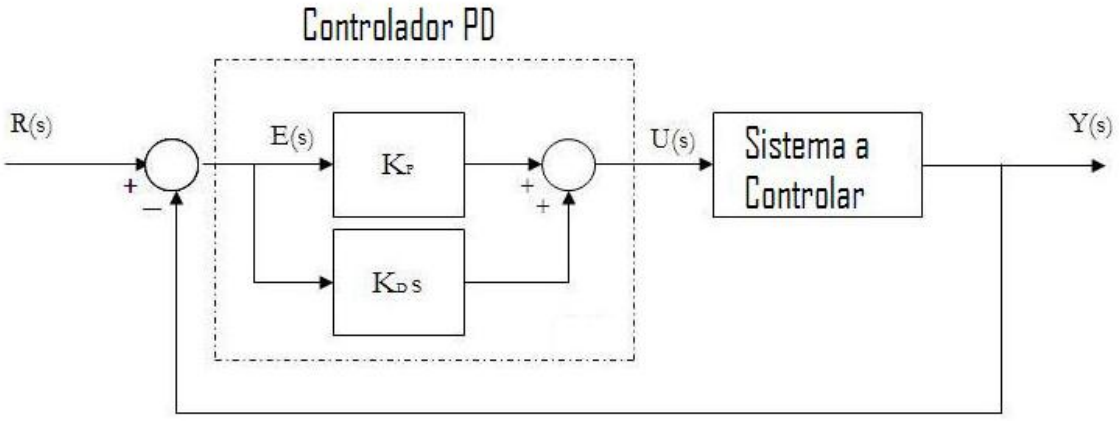

Imagen 5.9 Esquema de bloques de un controlador Proporcional Derivativo

Tras el ajuste de la ganancia proporcional y derivativa se mantiene en equilibrio con gran suavidad y prácticamente sin vibración. La derivada del error de verticalidad se obtiene directamente del valor neto calculado a partir de la señal del giroscopio electrónico (grados/s).

Se ha comprobado que el sistema es muy sensible a valores elevados de factor derivativo, oscilando bruscamente si se supera cierto valor y volviéndose muy "nervioso".

La frecuencia de envío de señales de control a los motores cada 20 ms ha dado buen resultado. Las pruebas con periodos de control de 100 ms o superiores muestras una pérdida significativa de suavidad del sistema así como robustez del sistema.

5.6.2. Control por Realimentación de Estado

El control por realimentación de estado permite cambiar la dinámica intrínseca del sistema al realimentar las variables de estado y de este modo reposicionar la posición de los polos del sistema compuesto. De este modo se puede convertir un sistema inestable en otro estable y con determinadas características como su tiempo de respuesta, grado de amortiguación, etc.

La imagen 5.10 muestra la estructura típica de señales y bloques para un sistema con realimentación de estado.

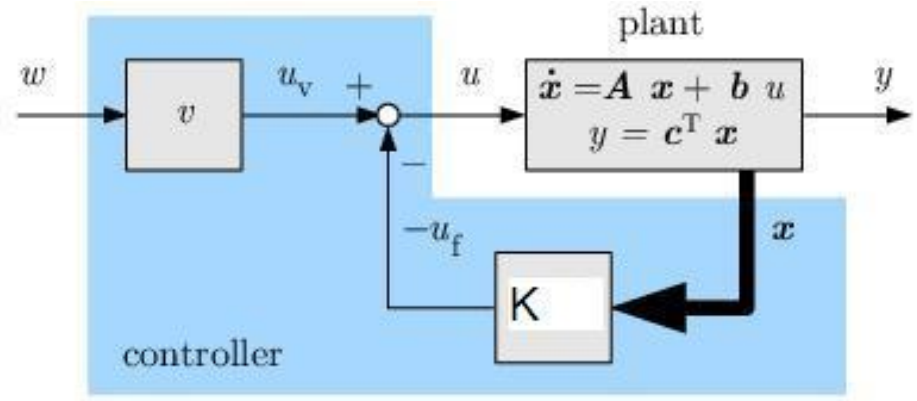

Imagen 5.10 Diagrama de bloques para sistema con realimentación de estado

Por tanto, nuestro primer objetivo será el diseño de un controlador que nos permita cambiar la dinámica del sistema compuesto.

En nuestro caso, el primer cambio fundamental es desplazar la posición de los polos desde la zona derecha del espacio de estados (sistema inestable) a la zona izquierda (sistema estable). Como condición adicional se le requerirá que el sistema sea lo suficientemente rápido para que las oscilaciones respecto al punto de equilibrio sean muy pequeñas y no se aleje de la zona en la que se ha linealizado el sistema. Un sistema linealizado muy lento en respuesta hará que no pueda mantener el equilibrio al degenerar la calidad del modelo a medida que se aleja del punto de linealización.

La matriz de realimentación de estado K con 4 valores es la que define el cambio de respuesta del sistema. Esta matriz la calcularemos a partir de la posición de los polos deseados (que especifiquemos nosotros al sistema compuesto).

Ejemplo de cálculo con Matlab de la matriz de realimentación de estado, conocidas las matrices A, B, C, D que representan la dinámica del sistema.

SIMULACIÓN CON MATLAB Nº1

Veamos la simulación del comportamiento del sistema ante una entrada de referencia de posición de 0 a 100 mm.

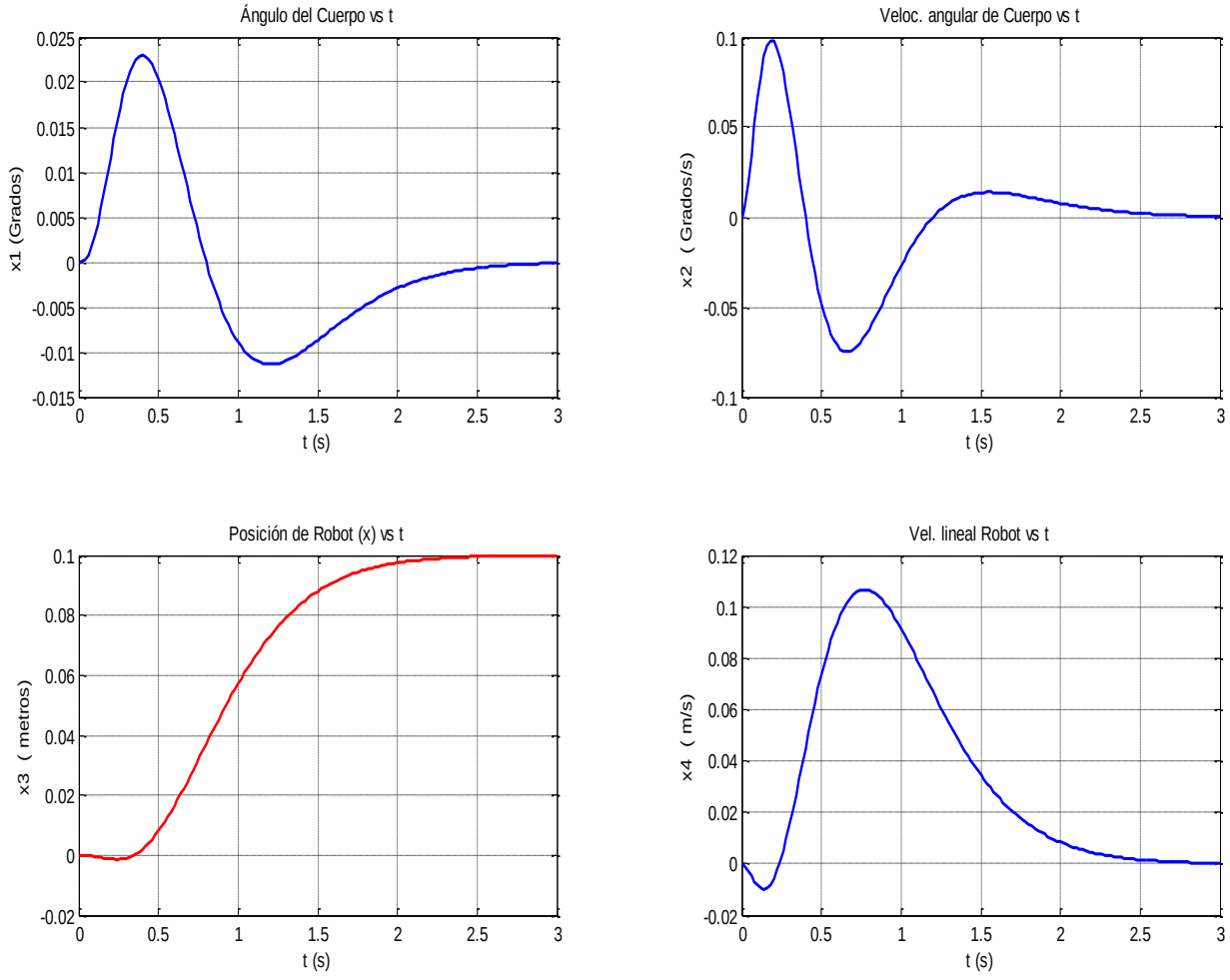

Imagen 5.11 Graficas con la respuesta a escalón del sistema realimentado (Polos -5)

En la primera simulación, los polos se han posicionado todos en el mismo punto de la parte real negativa con valor –5.

La velocidad media para este caso es de 0.03 m/s, lo cual puede ser conseguida por el robot al ser inferior a su valor máximo.

La aceleración requerida para obtener un incremento de velocidad de 0.12 m/s en un tiempo de 0.6 segundos es de 0.2 m/s^2. Para una masa total de 0.66 Kg se requiere una fuerza de 0.132 N (0.2*0.66) o un par de 0.087 N*m (0.132*0.041).

Dado que el par máximo de estos motores es de 16.7 N*cm, es decir, 0.167 N*m se comprueba que se dispone de suficiente margen para el control al requerir este la mitad del par máximo disponible.

Recordar que a medida que aumenta la velocidad de giro el par disponible va disminuyendo progresivamente por lo que perderemos capacidad de control.

Como se puede observar en las gráficas del comportamiento del robot, estando este inicialmente en posición de equilibrio, se desplaza ligeramente hacia atrás para dejar caer el péndulo hacia delante y permitir acelerar hacia la posición de referencia. Estando en movimiento y con suficiente antelación a la llegada al punto de referencia, el sistema de control inclina el péndulo hacia atrás en ángulo necesario para permitir la deceleración sin desestabilizar el sistema. El comportamiento del sistema conjunto es subamortiguado, es decir, no existe sobrepaso de la posición de referencia establecida.

Una característica que diferencia claramente el comportamiento teórico de Matlab respecto al robot real es que este último mantiene un equilibrio oscilante en torno al ángulo vertical. En la simulación en tiempo continuo y sin errores iniciales y perturbaciones se tiene un sistema realmente estable y sin oscilaciones.

Esto se debe a que matemáticamente, el sistema con realimentación de estado en modo continuo, es un sistema estable, por lo que de no existir perturbaciones el sistema se estabilizará en posición y ángulo cero.

El sistema real difiere en que siempre existe un pequeño error de ángulo entre el 0 real y el que toma el robot como 0 (al conectarlo en “nuestra” aproximación a la posición de equilibrio vertical). Este pequeño error hace que el sistema, inicialmente, esté ligeramente desestabilizado en un sentido y por tanto tienda a desplazarse en ese mismo sentido para mantener el péndulo vertical.

Por tanto, cualquier error inicial entre “vertical real” y “vertical teórica” dará un comportamiento del sistema inestable en posición de carro aunque no de equilibrio

del péndulo. El sistema tendrá que realizar una aceleración determinada para compensar este error de equilibrio, lo que mantenido en el tiempo hará que el robot empiece a ganar velocidad en un sentido hasta saturar los motores, ser incapaz de segur dando el par de aceleración para mantener el equilibrio y finalmente caerá el robot al suelo.

La clave para mantener el equilibrio a pesar de la deriva del giro es hacer que el error de posición sea proporcionalmente mucho más grande que el error provocado por el ángulo de inclinación, tratando de minimizar el error combinado que incluyen ambas señales".

SIMULACIÓN CON MATLAB Nº2

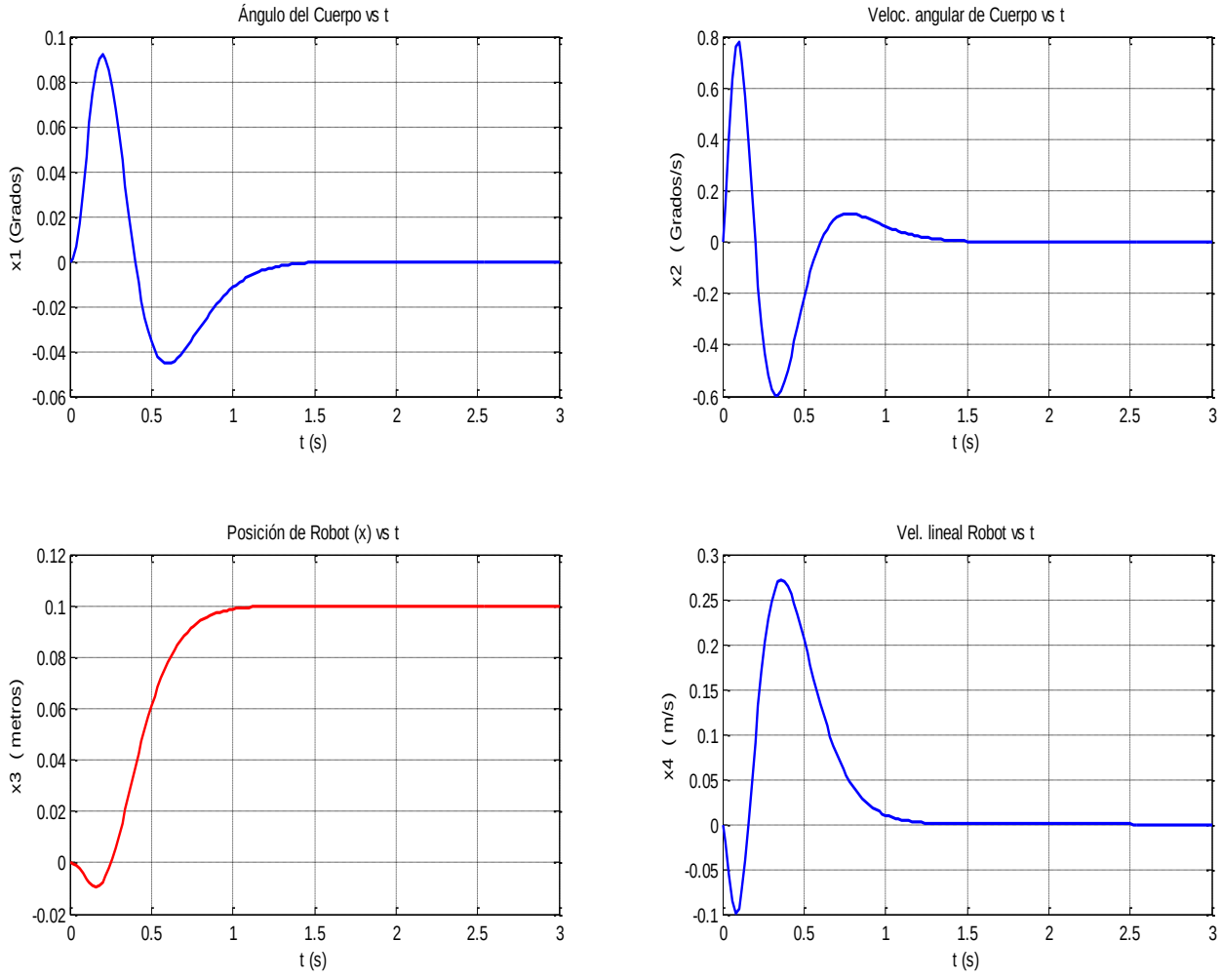

Imagen 5.12 Graficas con la respuesta a escalón del sistema realimentado (Polos -10)

Para este caso los 5 polos del sistema están en eje real negativo con valor –10.
Para este caso el sistema es mucho más rápido que el anterior, llegando sin sobrepaso al set point en 1 segundo a cambio de requerir velocidades

6. Pruebas Reales en Minirobot

6.1. Sistemas de control en tiempo real

Cuando hablamos de sistemas que requieren de un control en tiempo real, donde pequeños retardos pueden causar una operación deficiente del sistema de control, es necesario realizar una combinación adecuada de reparto de funciones entre lo que realiza Matlab (no crítico en tiempo) y lo que realiza el programa interno del NXT Brick (baja capacidad de memoria, lenguaje de bajo nivel pero alta velocidad de procesamiento).

Es importante tener en cuenta las limitaciones de comunicación que establece el canal Bluetooth, no tanto por la tasa de información que puede transferir de modo global sino por la limitación de los tiempos muertos entre envío y recepción.

La aplicación para el control de la estabilidad del sistema intrínsecamente inestable **es crítica en tiempo**, y un pequeño retardo en la transmisión del estado del sistema hará que, dado que es un sistema inestable, se deteriore la calidad de control e incluso haga inviable dicho control.

A la hora de implementar de modo práctico los algoritmos de control se debe usar la posibilidad de ejecución en paralelo de tareas críticas en el tiempo (como por ejemplo la integración de señal del Gyro) y dejar la secuencia principal (main) para añadir retardos (tipo Wait) y que no afecten a la calidad del control.

6.2. Control Proporcional Derivativo (PD)

6.2.1. Estabilidad con realimentación sensor de luz – Control PD

La prueba se ha realizado con el LED auxiliar encendido con el objetivo de minimizar el efecto de los cambios de iluminación exterior.
Una gran ventaja del sensor de luz es que no tiene deriva (siempre y cuando se mantengan la iluminación exterior constante y el color o reflectividad de la superficie sean homogéneos). Esto permite mantener un equilibrio indefinido en el tiempo sin que la señal de realimentación se degrade.

El rango de medida bruto que proporciona el sensor va desde 0 unidades (total oscuridad) hasta 100 unidades (luz muy intensa). Dependiendo de la iluminación del lugar, la reflectividad de la superficie y la distancia inicial a esto, el sensor proporcionará un valor de iluminación inicial concreto cuando el NXTway está en equilibrio vertical. Este valor es el que corresponde a "NXTway en equilibrio" para este caso concreto, y las desviaciones respecto a este valor serán interpretadas como inclinación hacia adelante o hacía atrás. En nuestro caso, dado que el sensor está en la parte trasera, cuando el NXT se inclina hacia atrás, la intensidad de la luz reflejada detectada será mayor (al estar más cerca tanto emisor como receptor), el valor absoluto incrementará, por tanto el valor neto será positivo. En este caso la señal de error y la de salida a motores tendrán signo contrario

Durante los 27 segundos de oscilación, los valores netos respecto al valor de referencia inicial (cuando el robot está en posición de equilibrio vertical) están siempre en torno al 0 original, con oscilaciones dentro de ±3 unidades.

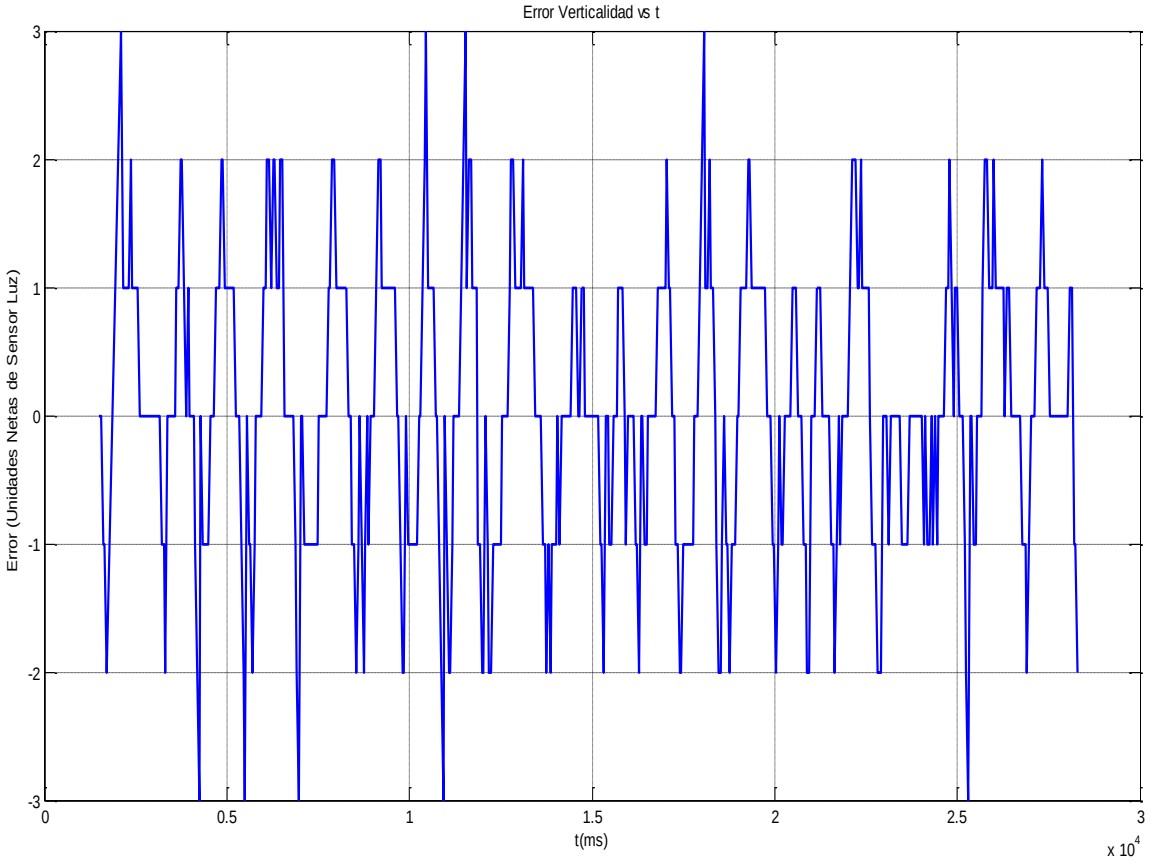

Imagen 6.1 Graficas con señal neta del sensor de luz durante la oscilación estable

Tras arrancar el programa de control, se observa como el robot oscila en torno a la posición de inicio, pero, dado que en el control solo hay realimentación de señal equivalente al ángulo de verticalidad del robot, nada impide que el con el tiempo el robot se desplace de su posición inicial sin control (al no haber control de posición implementado, solo de verticalidad).

Nota: La posición del robot se obtiene mediante el promedio del ángulo de giro (en grados) de ambas ruedas enviado por los encoder integrados en los motores. El desplazamiento lineal se obtiene pasando dicho ángulo a radianes y multiplicando por el radio de la rueda.

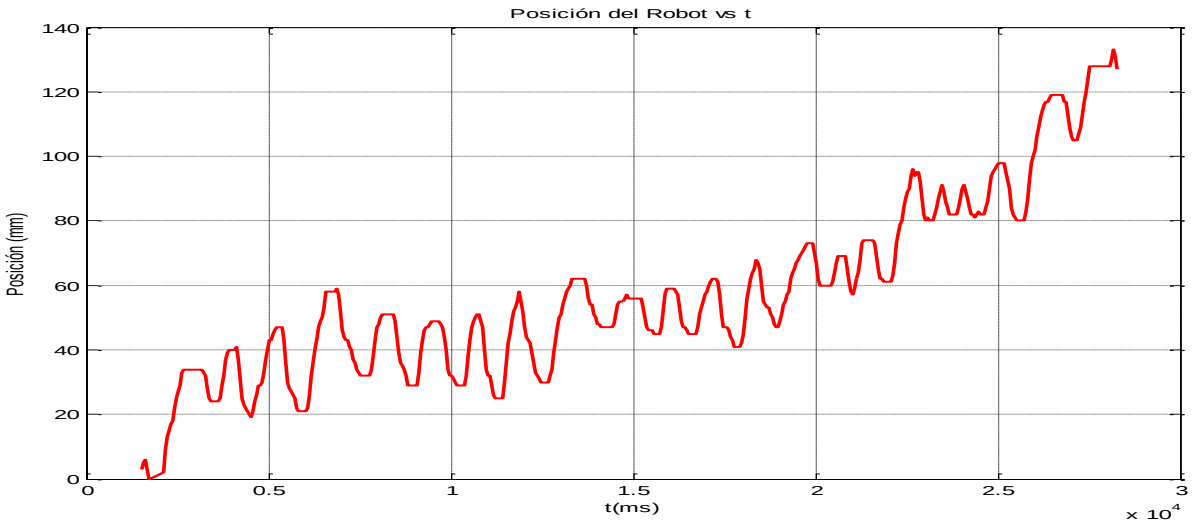

Imagen 6.2 Gráfica con posición robot con oscilación estable usando sensor de luz

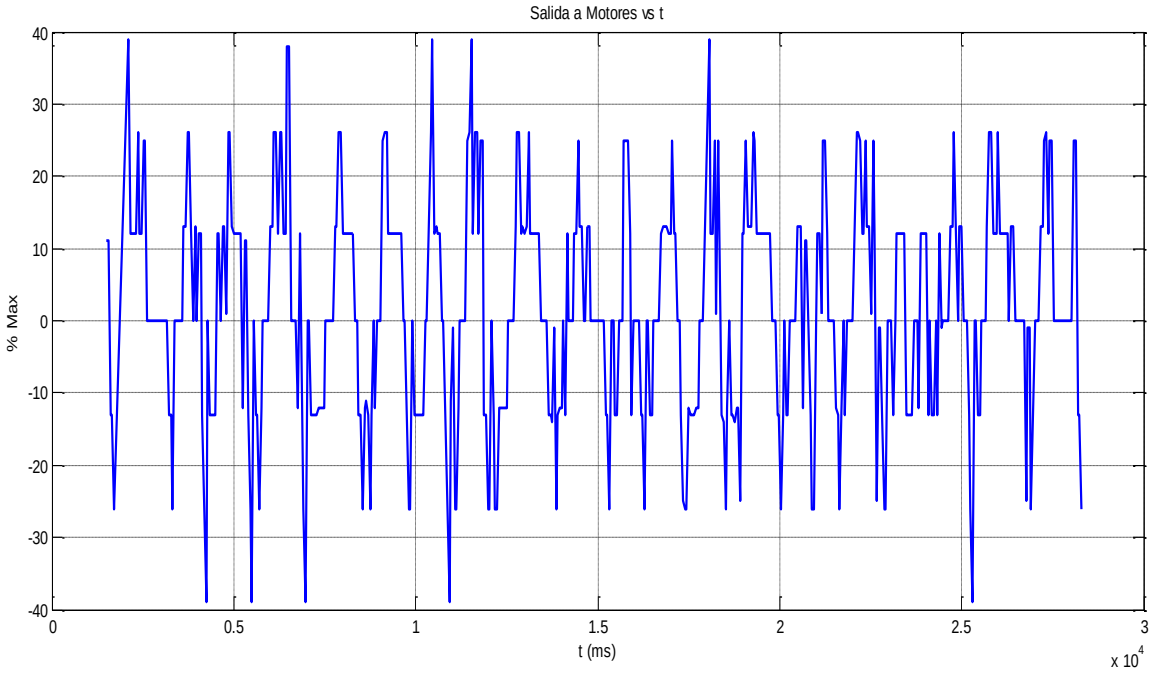

Imagen 6.3 Gráfica con tensión aplicada a motores durante equilibrio con sensor luz

El valor de control para los motores se mueve en el rango de ± 30 %, lo cual es un control relativamente suave, quedando margen en los actuadores para poder controlar la posición si se desea.

6.2.2. Estabilidad realimentación ángulo Giróscopo – Control PD

El gran problema que presenta el Giróscopo electrónico, como ya se ha comentado anteriormente, es su deriva y como consecuencia el aumento del error de medida con el tiempo.

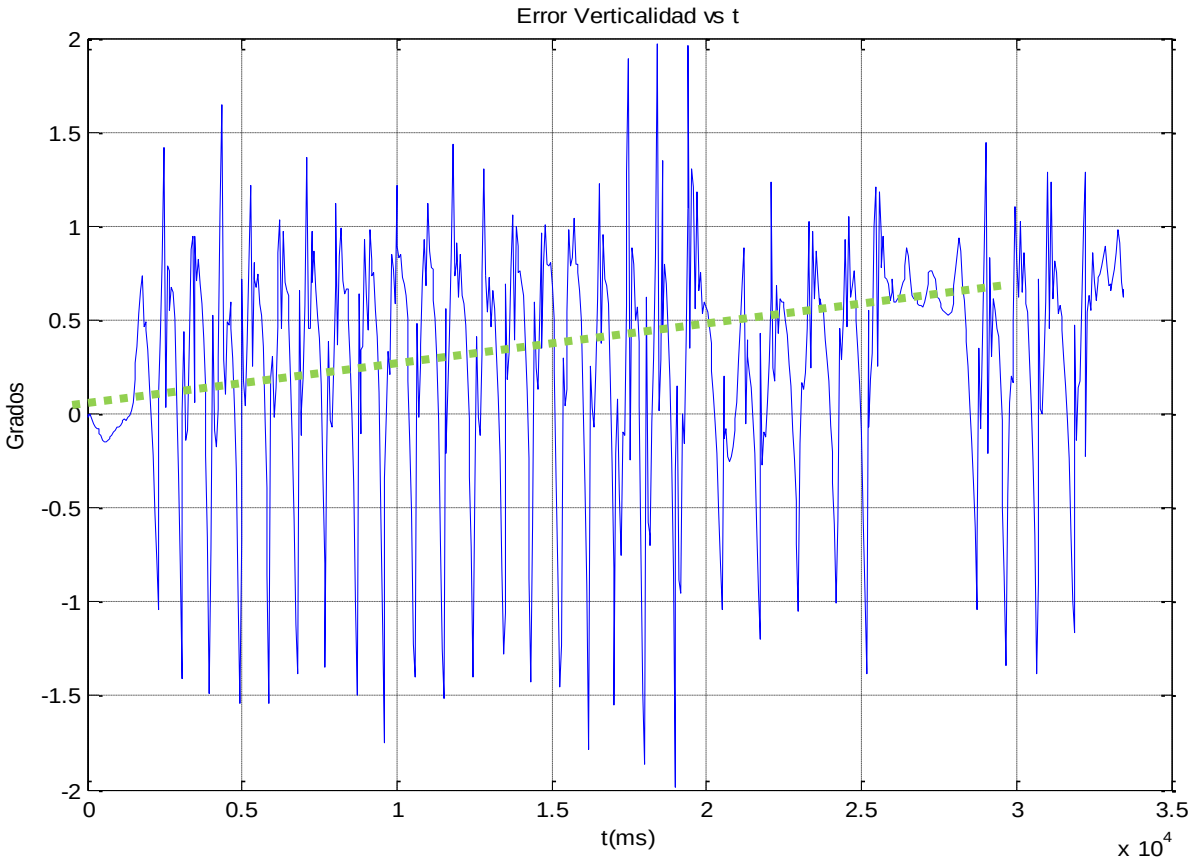

Imagen 6.4 Gráfica con ángulo de inclinación de Gyro (Compensado por software)

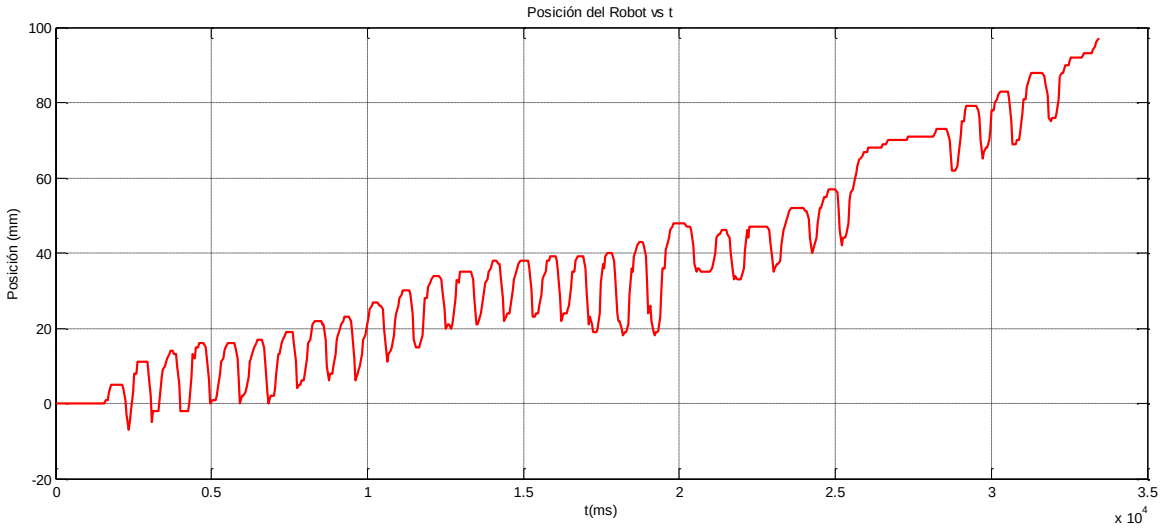

Imagen 6.5 Gráfica posición robot con oscilación dinámica estable usando Giróscopo

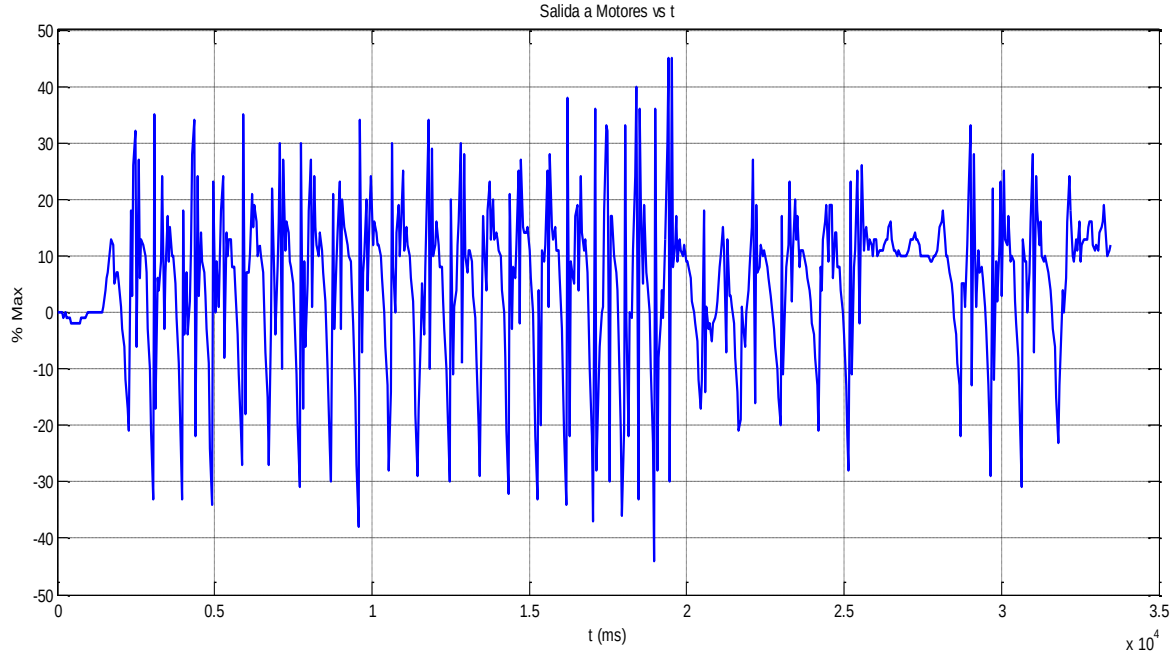

Imagen 6.6 Gráfica con voltaje aplicado a los motores durante la oscilación del robot

6.3. Control por Realimentación de Estado (RE) simple

6.3.1. Equilibrio del Sistema

Se observa que el equilibrio obtenido es muy suave, con unas oscilaciones mínimas para mantener la estabilidad dinámica. En ocasiones se desvía de su posición de referencia para poder mantener la estabilidad. En gráfica inferior se muestran 2 de estos desplazamientos en t =1 y 11 segundos.

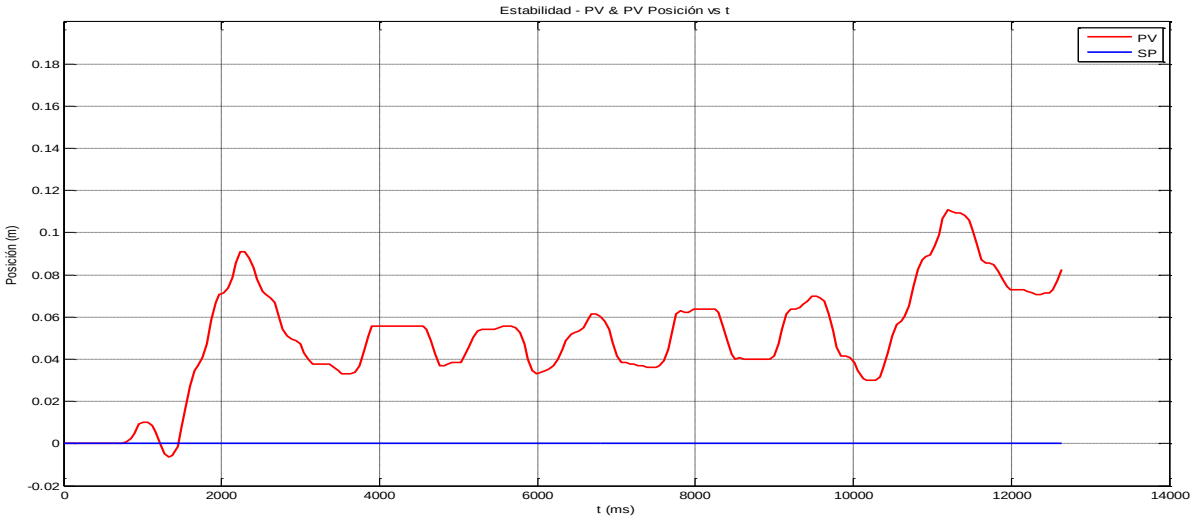

Imagen 6.7 Gráfica de estabilidad dinámica en torno a SP=0 por RE

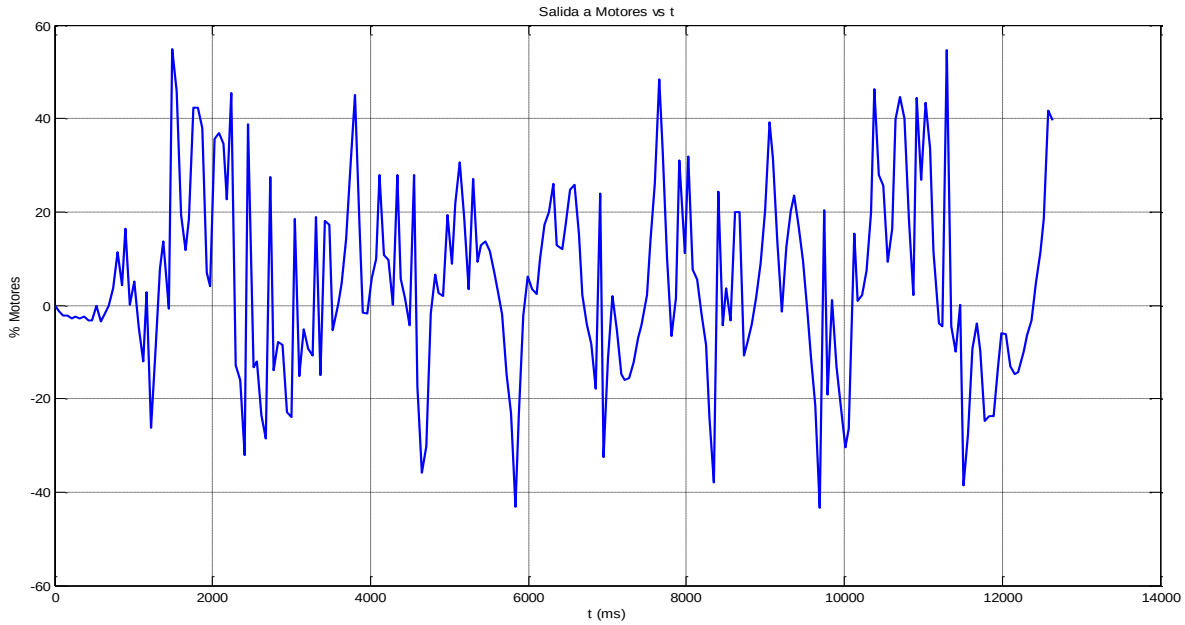

Imagen 6.8 Gráfica de señal salida a motores con control por RE

La línea verde punteada en la imagen 6.9 muestra como el valor del giro promedio va derivando, teniendo a los 10 segundos un error de -2 grados.

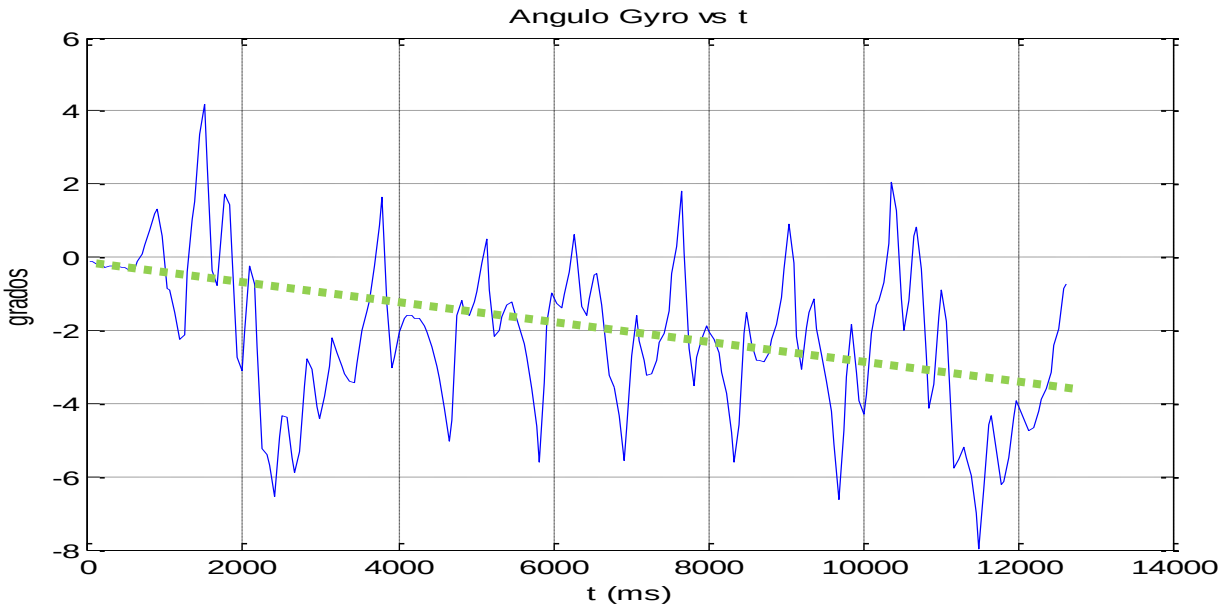

Imagen 6.9 Gráfica con valor de ángulo inclinación de cuerpo y su deriva

6.3.2. Respuesta de estabilidad ante perturbaciones externas

Durante los primeros 7 segundos oscila en torno al punto de equilibrio SP=0. En P1 se le da un impulso con la mano (-) que lo desplaza de su posición inicial.
De los 7 a los 9 segundos, ya está estabilizado y oscila durante 2 ciclos de modo estable.
Se le da un nuevo impulso en sentido contrario (+) esta vez más fuerte P2 a los 9 segundos y se desplaza casi 0.3m. El carro retorna a posición de referencia y se estabiliza en segundo17. Oscila de nuevo en posición estable 2 ciclos y se dá el último impulso positivo P3 que parece va a comportarse de modo similar al P2.

Aumentando la componente integral se corregirían estos errores respecto SP=0, pero se debe llegar a un compromiso pues el sistema empeora su respuesta dinámica y además una deriva en la señal de inclinación, con una ponderación adecuada, si mantiene la estabilidad se traduce en un error de posición.

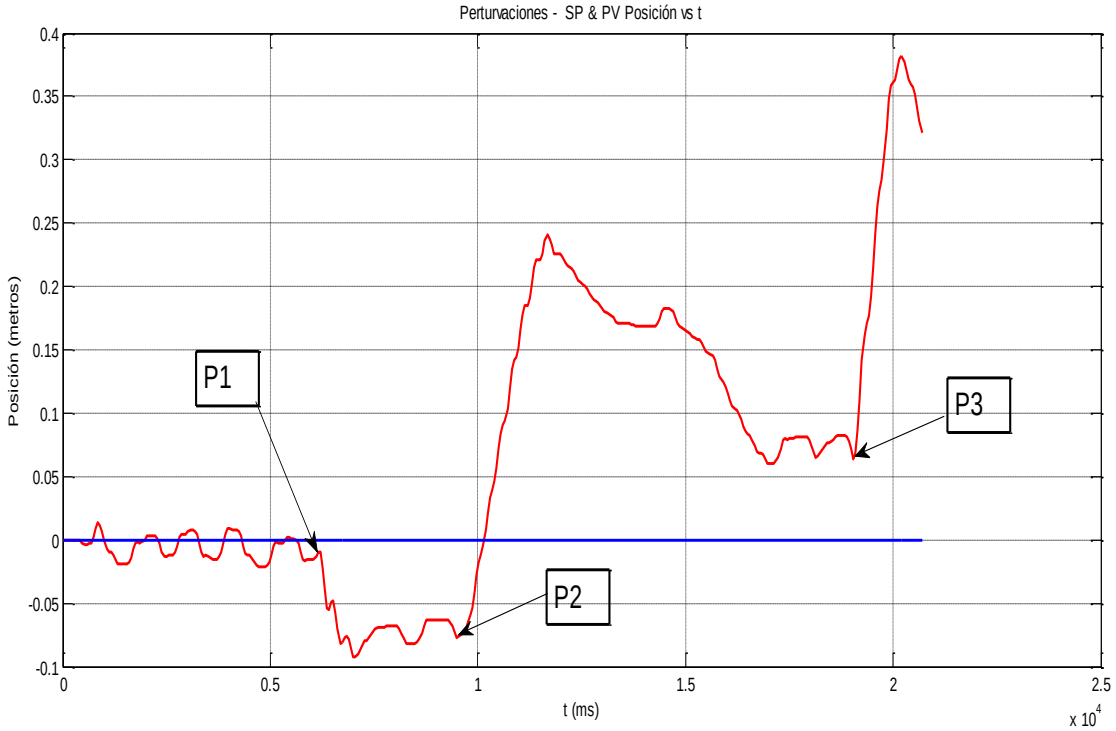

Imagen 6.10 Gráfica de comportamiento del sistema con RE ante perturbaciones

6.4. Realimentación de Estado + PID sobre error combinado

La configuración de este sistema es el que nos permite una configuración más robusta, soportando de modo excelente las perturbaciones así como mostrar un comportamiento dinámico mejorado, con una respuesta suave pero que garantizan el seguimiento de la consigna.

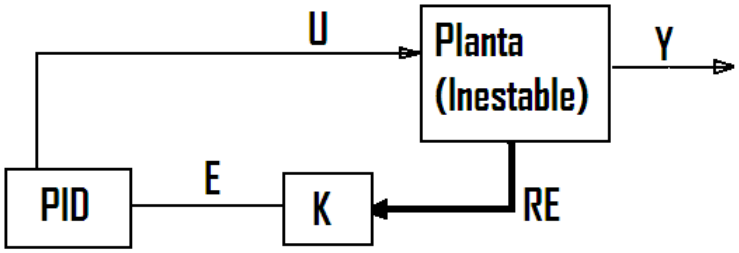

Imagen 6.11 Esquema de bloques combinando realimentación de estado y controlador PID

Una vez que tenemos un sistema estable, el proceso de sintonización es más intuitivo. El PID añade la función integral en el lazo que permite realizar el seguimiento de la consigna de un modo más efectivo.

6.4.1. Estabilidad Vertical (RE + PID)

En la imagen 6.12 se observa que se parte de una posición incial ligeramente desequilibrada. El sistema se desplaza para compensar este ligero desequilibrio hasta que encuentra su punto de estabilidad en posición 50 mm respecto a posición origen.

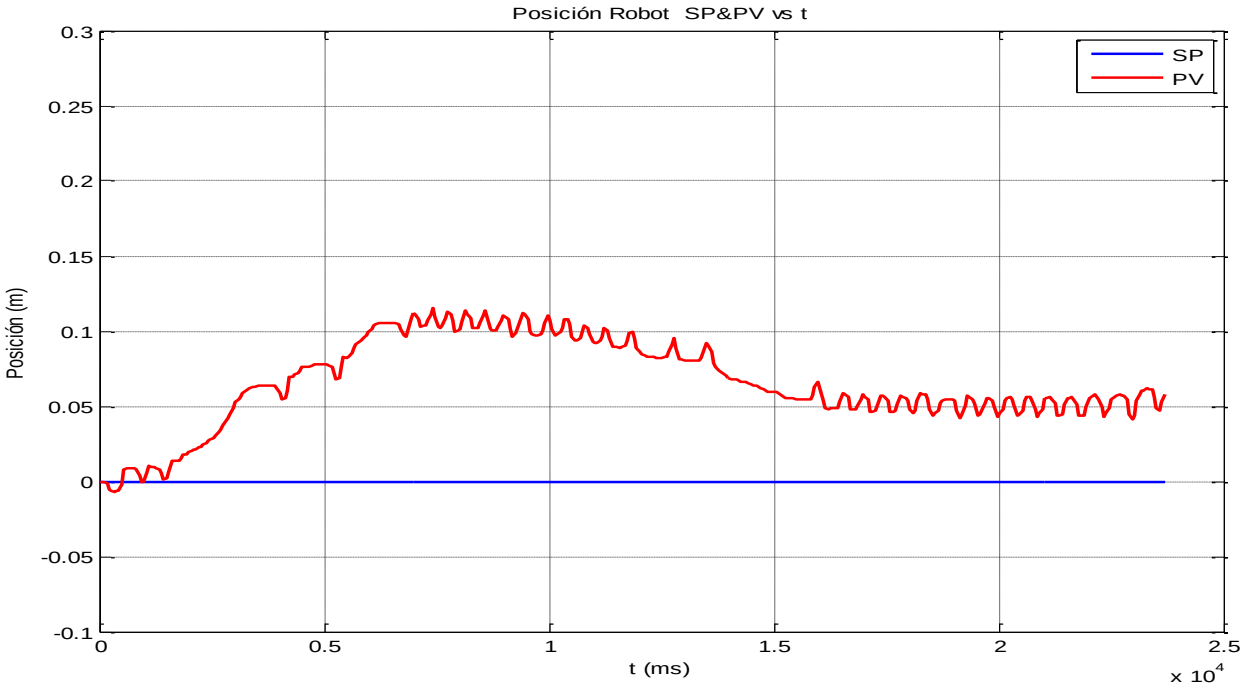

Imagen 6.12 Gráfica de posición robot con SP=0 y deriva en señal Gyro

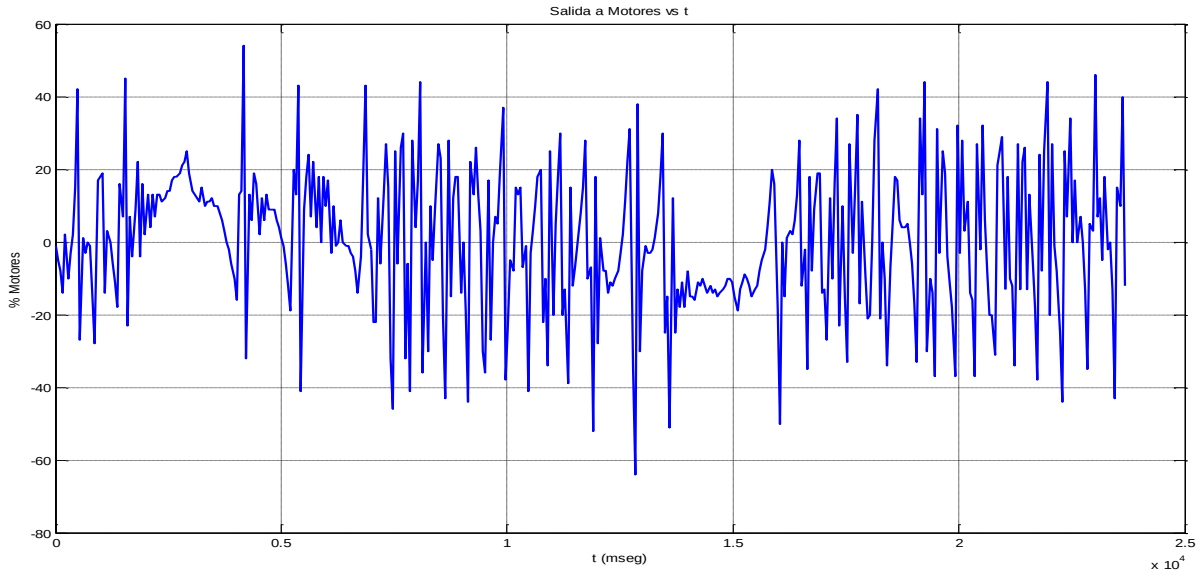

Imagen 6.13 Gráfica con señal de salida a motores para mantener estabilidad vertical

6.4.2. Estabilidad ante perturbaciones externas (RE + PID)

En esta prueba de estabilidad, se dan impulsos con la mano intentando desestabilizarlo (P1, P2, P3). Se comprueba en la gráfica como el robot mantiene el equilibrio y además retorna a su posición de origen, dado que la consigna de posición es siempre 0.
Las pequeñas oscilaciones son los periodos de estabilidad dinámica donde el robot balancea sobre su punto de equilibrio.

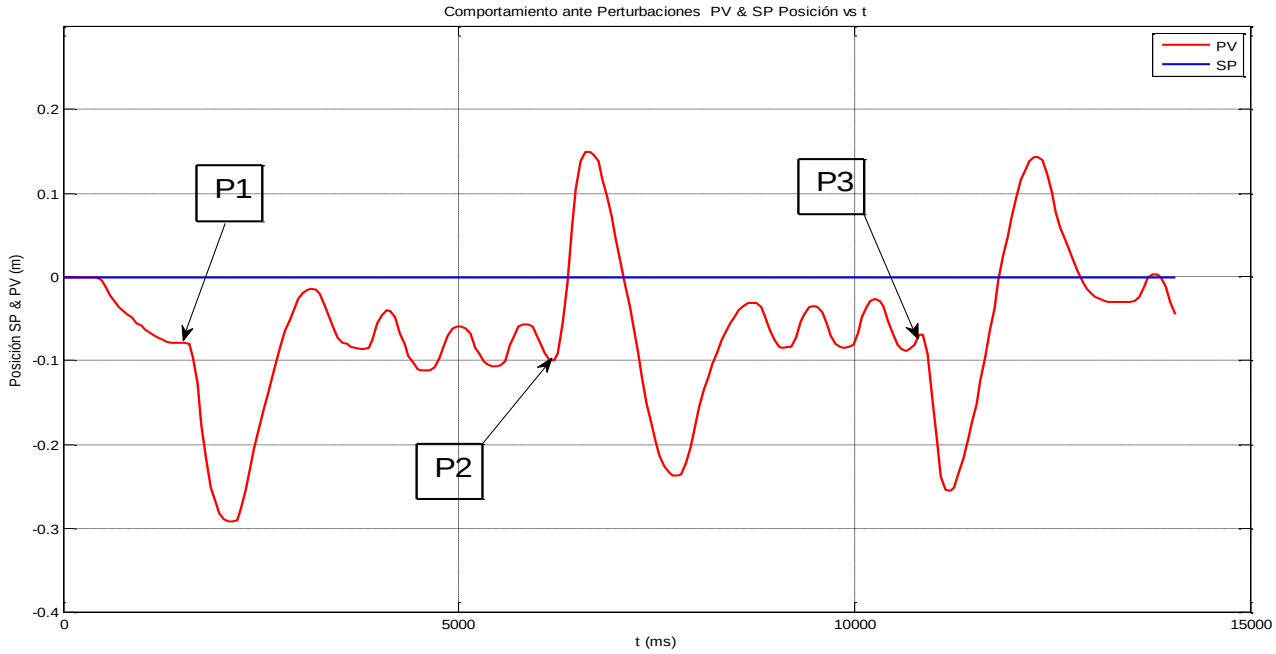

Imagen 6.14 Gráfica que presenta la respuesta del sistema ante perturbaciones externas

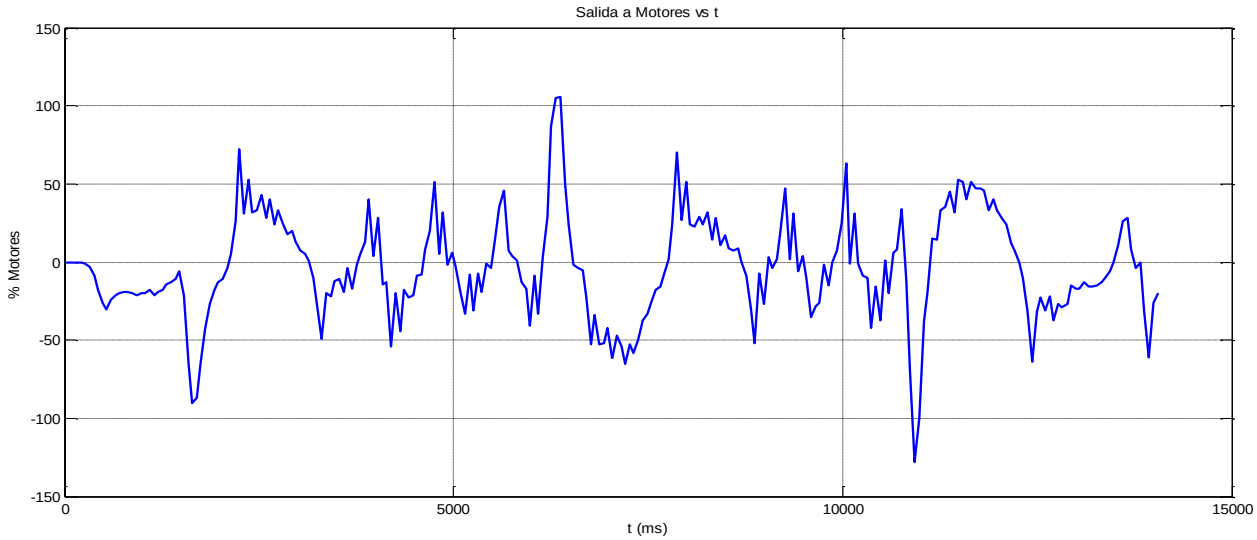

Imagen 6.15 Gráfica con señal de salida a motores ante perturbaciones externas

6.4.3. Función Seguimiento SP posición (RE + PID)

En la imagen 6.16 se muestra como el sistema hace el seguimiento del SP de posición establecido. La dinámica del sistema se observa que es especialmente buena, aunque existe un pequeño offset de posición mantenido.

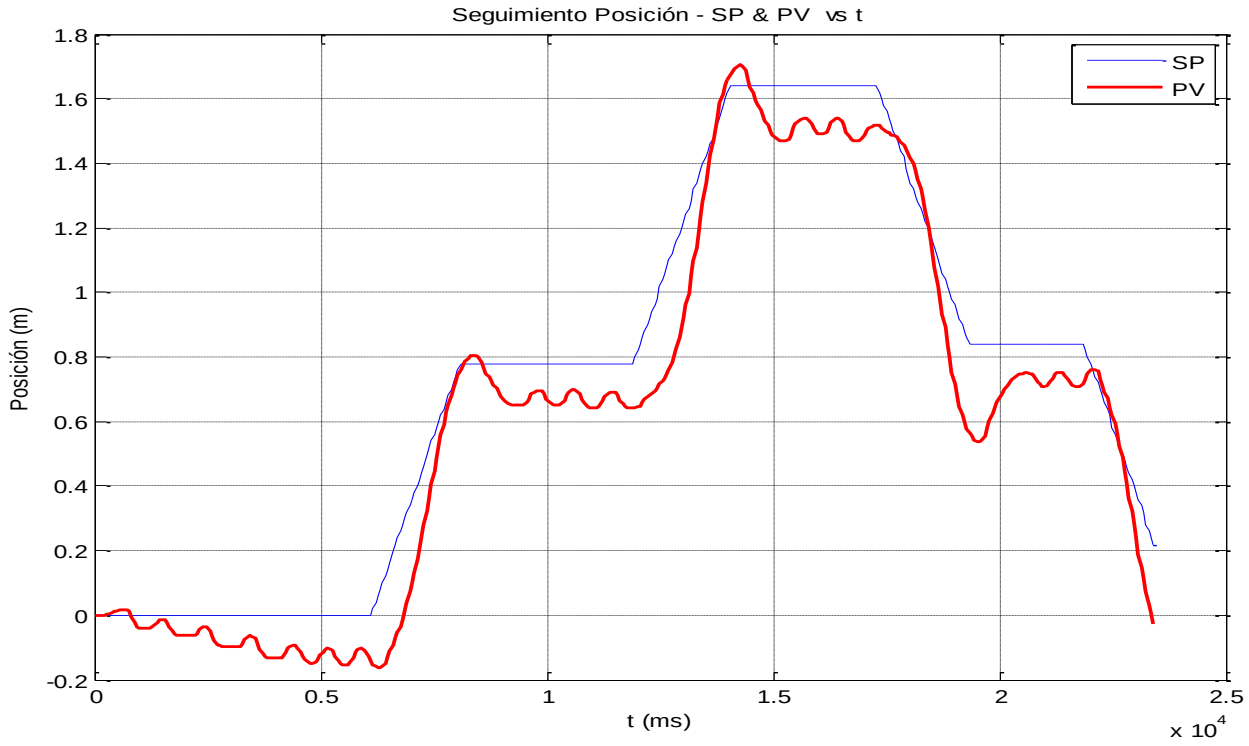

Imagen 6.16 Gráfica que muestra el seguimiento a una referencia de posición (SP)

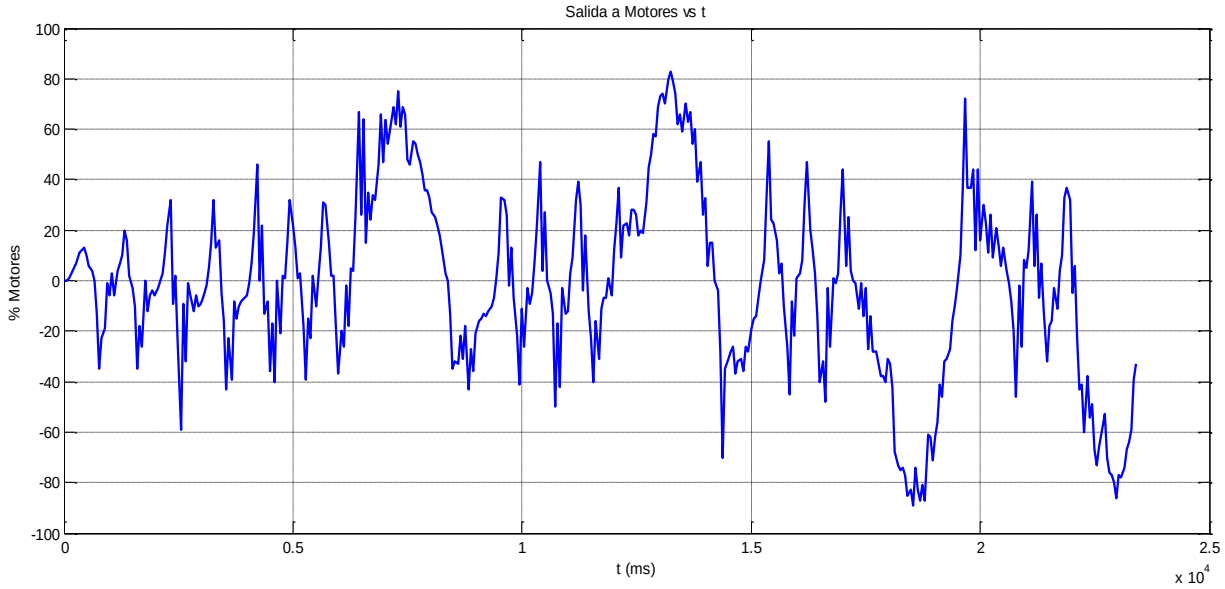

Imagen 6.17 Gráfica con señal de salida a motores al seguir SP posición

6.4.4. Equilibrio + Seguimiento SP + Subida Rampa (RE + PID)

Se ha utilizado una rampa de aproximadamente 20 grados de inclinación.

En la imagen 6.18 se puede observar lo siguiente:

- Durante los 6 primeros segundos está oscilando en posición de resposo en el suelo.
- Al avanzar, la rampa tiene un pequeño escalón que tiene que salvar, y son las oscilaciones provocadas tras "saltar" el desnivel.
- Entre los 12 y los 17 segundos el robot está **estabilizado y oscilando sobre la rampa**
- La siguiente orden de avance, fuerza el robot a seguir subiendo, pero esta vez de modo suave ya que está en medio de la rampa y no hay obstáculo alguno.
- A partir de los 22 segundos se la va dando orden de bajar en equilibrio.

En la gráfica de señal de salida a motores (Imagen 6.19) se puede observar la diferencia de la carga requerida entre la línea punteada en color naranja (0 a 6 s), donde el promedio es próximo a 0 en el suelo y la línea punteada en color rojo (12 a 17 s) donde el NXTway está **sobre la rampa en balanceo estable** y requiere por tanto una potencia promedio positiva en torno al 17% para contrarrestar el peso del robot, que de otra manera caería por la rampa.

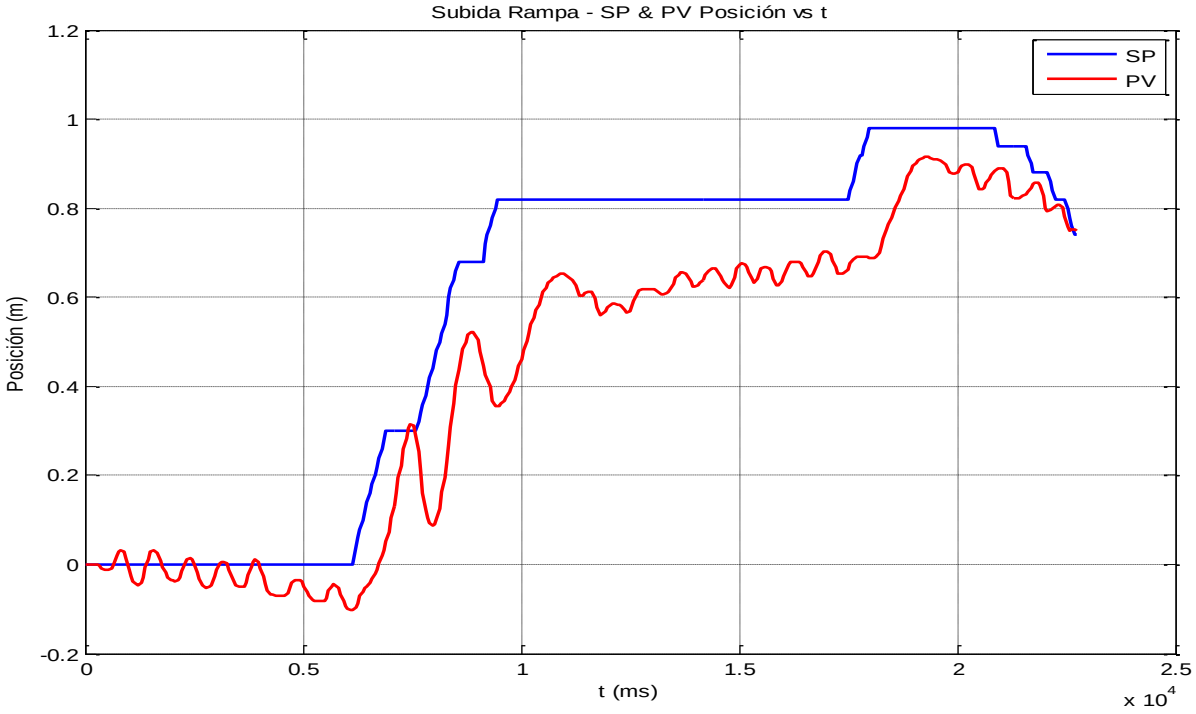

Imagen 6.18 Gráfica con el comportamiento durante la subida por una rampa

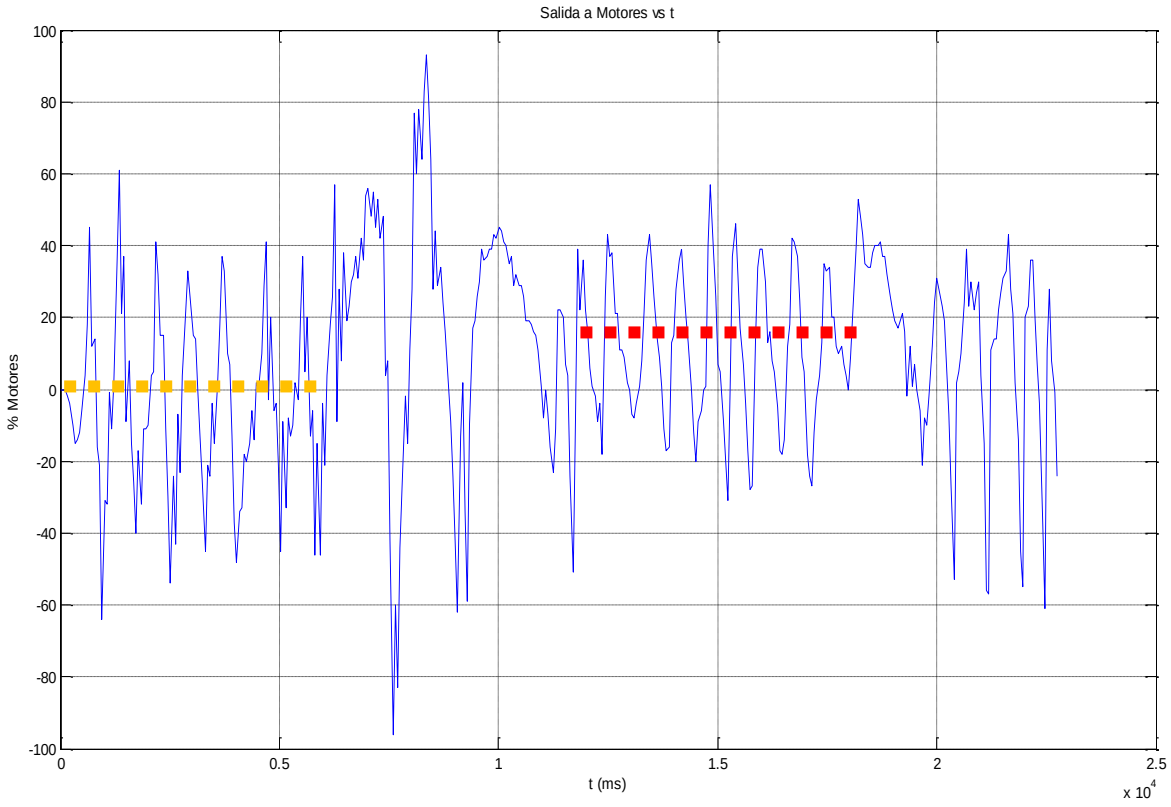

Imagen 6.19 Gráfica donde se observa la potencia media motores del robot sobre rampa

6.5. Equilibrio + Seguimiento Posición Referencia

El segundo paso en nivel de control del sistema consiste en poder controlar además de la posición vertical del cuerpo, el desplazamiento del robot siguiendo una referencia de entrada, es decir, **estamos diseñando un servosistema con el robot**.

En estos sistemas se requiere un control de la ganancia con la realimentación de estado de modo que sea independiente de la posición de los polos en lazo cerrado.

El objetivo será por tanto el **de compatibilizar la realimentación de estado** (convierte el sistema inestable en estable) **con la realimentación de salida** de modo que se consiga tener los polos en posición del espacio complejo deseado así como un error de posición nulo.

Determinamos la posición de los polos en el plano complejo. Estos polos determinan el polinomio característico de ese sistema y con ello un vector con sus coeficientes.

El objetivo es que la matriz final que refleja el sistema con realimentación de estado y salida Ar tenga los polos en la posición que se ha establecido previamente.

Identificamos coeficientes de la última fila de la matriz Ar con el correspondiente del polinomio característico obtenido. Se nos genera un sistema de ecuaciones con 5 incógnitas (k0, k1, k2, k3, k4)

6.5.1. Limitación práctica de velocidad giro motores

Velocidad de giro máxima
De los datos experimentales sobre los motores obtenemos que para una tensión de batería de 7.2V se consigue una velocidad angular máxima de 82 rpm y para una tensión de 9V se consigue una velocidad de 117 rpm.

Dado que con la batería recargable de LEGO la tensión típica tras la carga completa será aproximadamente de 8V, podemos considerar inicialmente como referencia la velocidad máxima de giro sin carga de 100 rpm.
Con las ruedas de sección delgada (radio = 0.041 mm) avanza 0.2576 m/rev.

Nota: La **velocidad lineal máxima (sin carga)** sería de 25.7 m/min o bien **de 0.43 m/s**.

6.6. Disposición General del Equipo

Con objeto de dejar claro el layout de elementos y como se conectan y comunican, en la figura 6.20 se muestra todos componentes principales.

- Portatil ejecutando el software RobotC
- NXTway-G con conexión Bluetooth habilitado y conectado con PC
- Joystick conectado por cable USB al PC
- USB-Bluetooth Dongle conectado a puerto USB PC ya configurado

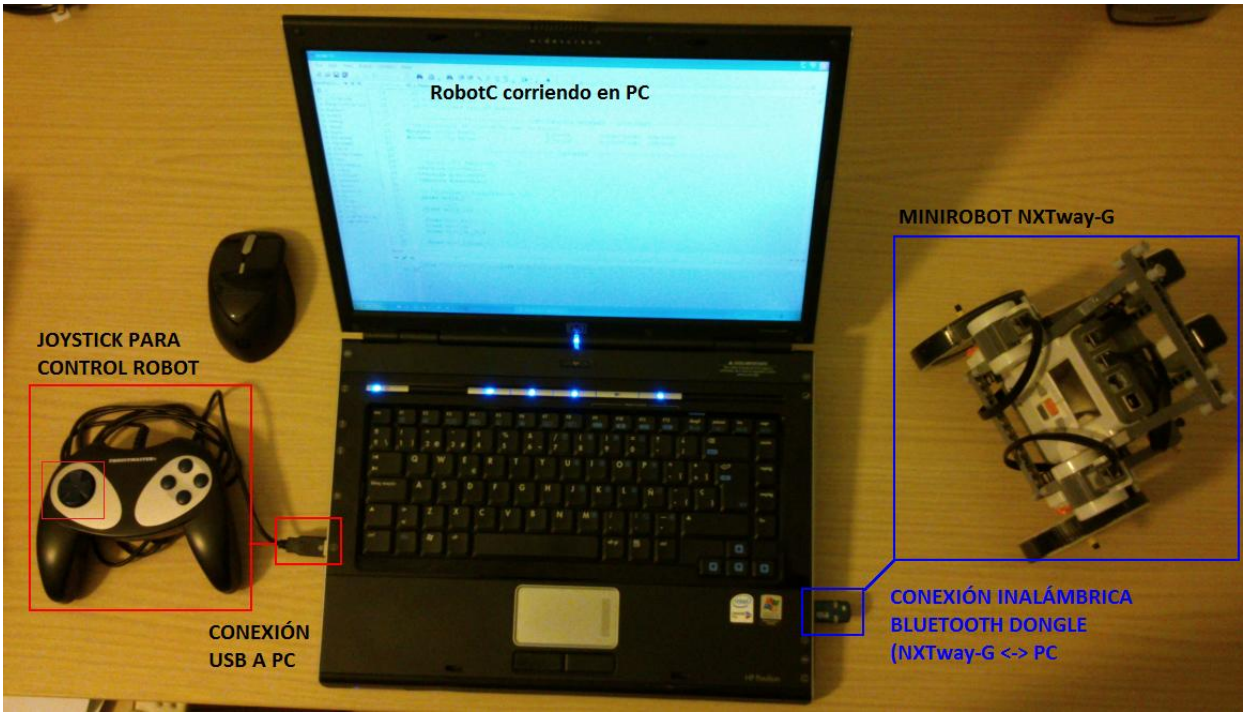

Imagen 6.20 Layout de elementos principales requeridos para las pruebas del proyecto

7. Conclusiones y futuros trabajos

El principal objetivo de este proyecto era el de diseñar e implementar algoritmos de control sobre un sistema real mediante el uso en el set de robótica programable LEGO Mindstorms NXT.

Con objeto de comprobar los límites del sistema de control dinámico se eligió un sistema inherentemente inestable, con una configuración de péndulo invertido sobre ruedas (Segway).

Se han implementado las siguientes pruebas de modo satisfactorio:

- Controlador PD usando la señal del sensor óptico como realimentación de la "verticalidad" del sistema. El sistema se mantiene estable y con oscilaciones moderadas. La ventaja de este sistema es que no tiene deriva, por lo que es posible mantener el sistema oscilando de modo indefinido.

- Controlador PD usando la señal del giróscopo electrónico e integrando esta numéricamente. Igualmente es posible mantener el equilibrio en posición vertical del NXT pero encontramos el problema de la deriva del giróscopo, que provoca que con el tiempo se desplace en la dirección

- Control por realimentación de estado (RE)
  Se comprueba que tanto la estabilidad del sistema como la resistencia a perturbaciones son aceptables. Se prueban diferentes posicionamientos de los polos para variar la dinámica del sistema. No es fácil encontrar un posicionamiento de polos que manteniendo la estabilidad y comportamiento suave (sin oscilaciones bruscas) tenga una respuesta dinámica elevada.

- Implementar control por realimentación de estado y controlador PID
  Este sistema, una vez sintonizado el PID muestra un comportamiento dinámico excelente, resistiendo las perturbaciones de modo óptimo.
  Esta robustez ha permitido controlar remotamente el desplazamiento del robot mediante Joystick así como poder subir una rampa próxima a 20°.
  La velocidad a la que puede girar el NXTway-G sobre su eje central es realmente elevada.

Como conclusión, el grado de estabilidad y robustez que se puede conseguir con el NXTway-G es muy bueno, especialmente considerando el bajo coste del conjunto.

Una vez finalizado este proyecto y tras la experiencia de controlar un sistema real inestable con diferentes métodos de control, es interesante mencionar lo siguiente:

- Las características de este **sistema** requieren de un **control próximo al tiempo real**, es decir, con tiempos de reacción entre la entrada y salida en el rango de los ms.
  La velocidad del procesador y la capacidad de ejecución en paralelo de diferentes tareas criticas en tiempo por el microcontrolador del NXTBrick es una característica clave que permite poder implementar sistemas de control lo suficientemente rápidos como para controlar sistemas con tiempos de respuesta muy bajos. Esto confirma la capacidad de LEGO Mindstorms NXT para tareas de control de mayor complejidad.

  Se ha comprobado cómo para el mismo algoritmo de control, a medida que el tiempo de ciclo se alarga, la calidad del control se va degradando hasta no ser capaz de mantener ni siquiera la estabilidad del sistema, y mucho más cuando se aplican perturbaciones externas.

- Para poder desplazar y girar el NXTway, el grado de estabilidad del sistema debe ser lo más elevado posible así como su tiempo de respuesta y estabilización el menor posible. Esto permitirá soportar las perturbaciones internas que provocan las aceleraciones de los motores que tienden a desequilibrar al sistema cuando se desplaza o gira.

- El efecto de una amortiguación externa (rodar sobre moqueta, una alfombra, etc) mejora de forma significativa la estabilidad y suavidad del comportamiento del NXTway-G. Tras observar esto, se ha incorporado sobre la superficie de ambas ruedas de una fina capa de espuma adhesiva. El efecto es equivalente y muestra una mejora general de su comportamiento dinámico.

- La deriva del giróscopo (el error de medida del ángulo va aumentando con el tiempo) es inevitable. Esto supone un problema real importante que se ha de resolver y que no se considera en los modelos matemáticos, en los cuales todo funciona de modo preciso.
  La clave para mantener la estabilidad a pesar de la deriva consiste en usar una realimentación combinada de ángulo de inclinación y posición de robot como mínimo, de modo que un avance del robot ponderado es equivalente a un error del ángulo. Esto permite mantener el robot en posición vertical pero a cambio desplaza la posición de oscilación estable del robot respecto a posición inicial.

Dado el éxito comercial del que disfruta LEGO Mindstorms NXT y la cantidad de aficionados y foros que puedes encontrar en internet, van apareciendo nuevos retos cada vez más impresionantes.

Como propuesta de nuevos proyectos, se podrían considerar los siguientes:

1. Con un grado de complejidad más elevado, construir y controlar un sistema dinámico en equilibrio oscilante sobre una bola denominado NXT Ballbot.
   https://www.youtube.com/watch?v=f8jxGsg3p0Y

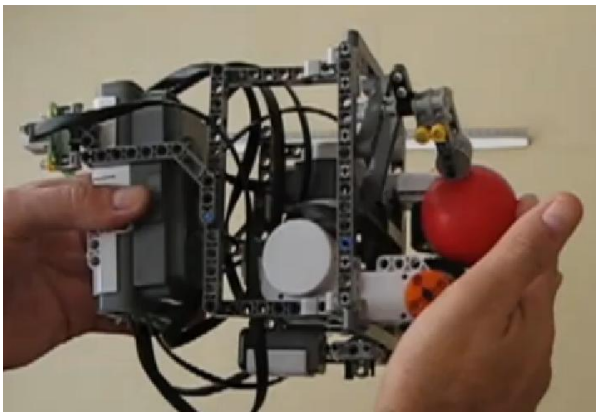

Imagen 7.1 NXT Ballbot. Sistema oscilante sobre una bola

2. Construir e implementar el control con para un minirobot con configuración monociclo (NXT balancing Unicycle). Ryno motors ya lo comercializa para transporte personal. http://rynomotors.com/

Imagen 7.2 Monociclo autoestabilizado para transporte unipersonal

Por otro lado, sería interesante aplicar las librerías desarrolladas por la universidad de Aagen para Matlab (4.3) ya que esta es una herramienta de trabajo muy utilizada en ingeniería de control y permite una capacidad de procesamiento desde el PC muy superior a lo que es posible realizar directamente en el NXTBrick

8. Bibliografía

Libros

[1.1] Deitel, H.M. y Deitel P.J. *Como programar en C/C++.* 2a. ed., Prentice Hall, 1995.

[1.2] Dorf,R.C. y Bishop, R.H. *Sistemas de Control Moderno*. 10a. ed., Prentice Hall, 2007.

[1.3] Hansen J.C. *LEGO Mindstorms NXT – Power Programming.* 1a. ed., Variant Press, 2007.

[1.4] Hebertt Sira, R. [et.al]. *Control de sistemas no lineales*. Prentice Hall, 2005

[1.5] Ogata, K. *Ingeniería de Control Moderna,* 3a. ed., Prentice Hall, 1998.

[1.6] S. Dominguez [et.al]. *Control en el Espacio de Estado*. 2a. ed., Prentice Hall, 2006

Documentos

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

Páginas Web

[3.1] http://es.wikipedia.org/wiki/Segway
Entrada sobre el Segway en Wikipedia. Consulta de 2013.

[3.2] http://mindstorms.lego.com/en-gb/Default.aspx
Página Oficial LEGO Mindstorms NXT. Consulta de 2013.

[3.3] http://www.HiTechnic.com
Web de HiTechnic, empresa fabricante de hardware para el NXT. Consulta de 2013.

[3.4] http://www.philohome.com/nxt.htm
Web de Phillipe Hurbain, coauthor del libro Extreme NXT Consulta de 2013.

[3.5] http://www.teamhassenplug.org/robots/legway

Web de Steve Hassenplug, creador del LegWay, entre otros montajes Consulta de 2013.

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

[3.7] http://www.mathworks.com/matlabcentral/fileexchange/19147-nxtway-gs-self-balancing-two-wheeled-robot-controller-design
Web de Matlab donde simulan comportamiento de NXTWay-G con Simulink Consulta 2009

[3.8] http://www.lfb.rwth-aachen.de/en/education/ws07/mindstorms.html
Web del proyecto Matlab meets Mindstorms Consulta de 2013.

[3.9] http://www.robotc.net/purchase/nxt/
Página oficial del Software de Programación RobotC. Consulta 2013

9. Anexos

9.1. Programas en Matlab

9.1.1. Modelo Matemático Complejo

```
%Este programa calcula la matriz de realimentación de estado K y la
%ganancia Ko así como las matrices Ar y Br resultantes en lazo cerrado.

%Para girar simplemente se introduce un offset en u_r y u_l.

clc;
clear;
%**************** ELECCIÓN DE VARIABLES DE ESTADO*******************
%*****************************************************************

% x1 = ángulo rueda (Encoder)
% x2 = ángulo cuerpo (Integración Gyro)
% x3 = velocidad angular rueda (Derivación Encoder)
% x4 = velocidad angular cuerpo (Gyro)

%********************** DATOS SISTEMA ****************************
%*****************************************************************
%Declaración de variables y asignación de valores
g=9.81;
m=0.03;
R=0.041;
Jw=(m*R^2)/2;
M=0.6;
W=0.142;
D=0.045;
H=0.280;
L=H/2;
Jpsi=(M*L^2)/3;
Jfi=M*(W^2+D^2)/12;
Jm=1e-5;
Rm=6.69;
Kb=0.468;
Kt=0.317;
n=1;
fm=0.0022;
fw=0;

%Cálculo de matriz de estado y entrada (A,B)
alfa=(n*Kt)/Rm;
alfa

beta=((n*Kt*Kb)/Rm)+ fm;
beta
```

```
E(1,1)=(2*m+M)*R^2+2*Jw+2*n^2*Jm;
E(1,2)=M*L*R-2*n^2*Jm;
E(2,1)=M*L*R-2*n^2*Jm;
E(2,2)=M*L^2+Jpsi+2*n^2*Jm;
E

F=[(beta+fw) -beta;-2*beta 2*beta];
F

G=[0 0;0 -M*g*L];
G

H=[alfa/2 alfa/2;-alfa -alfa];
H

I=beta+(W/R)*fw;
I

J=(R/W)*alfa;
J

K=(0.5*m*W^2+Jpsi+(W^2/(2*R^2))*(Jw+n^2*Jm));
K

%Determinación de los elementos de la ecuación de estado

det_E=det(E); %Calculamos el determinante como variable para operación más eficaz

%****************** MATRICES DE ESTADO A,B,C,D **********************
%**********************************************************************
A=zeros(4,4);
A(1,:)=[0 0 1 0];
A(2,:)=[0 0 0 1];
A(3,2)=-M*g*L*E(1,2)/det_E;
A(4,2)=M*g*L*E(1,1)/det_E;
A(3,3)=-[(beta+fw)*E(2,2)+2*beta*E(1,2)]/det_E;
A(4,3)=[(beta+fw)*E(1,2)+2*beta*E(1,1)]/det_E;
A(3,4)=beta*E(2,2)+2*E(1,2)/det_E;
A(4,4)=-beta*(E(1,2)+2*E(1,1))/det_E;
A

B=zeros(4,1);
B(3,:)=alfa*(E(2,2)/2+E(1,2))/det_E;
B(4,:)=-alfa*(E(1,2)/2+E(1,1))/det_E;
B

C=[1 0 0 0];

D=[0];
```

```
%********************** POSICIÓN DE LAS RAICES **********************
%*********************************************************************
%Ahora al añadir un integrador (1/s) al sistema este pasa
%de orden 4 a 5. Se añade por tanto una raiz nº5.

raiz1 = -1;
raiz2 = -1;
raiz3 = -1;
raiz4 = -1;
raiz5 = -3;        %Raiz adicional
raices = [raiz1, raiz2, raiz3, raiz4 raiz5];


%********************** FUNCION DE TRANSFERENCIA ********************
%*********************************************************************
%Podemos obtener la/s funcion/es de transferencia mediante la función ss2tf
% x/u  y  fi/u
[num,den]=ss2tf(A,B,C,D)


%Para obtenerlo mediante Matlab el polinomia a partir de los polos directamente usamos
%función poly() poniendo las raices reales y/o complejas en un vector fila
%Pd es el polinomio característico deseado


%********************** Q, Q_inv, Tc_inv, Tc ************************
%*********************************************************************

%Obtengamos la matriz de transformación a la forma canónica controlable Tc y Tc_inv (variable de
fase):
%El primer paso es obtener la matriz de controlabilidad
dim_A = size(A)
Q = zeros(dim_A(1,1),dim_A(1,2));
for i=0:dim_A(1,1)-1
   Q(:,i+1) = A^i*B;
end
Q
rango_Q= rank(Q)
%Muestra matriz inversa de Q (existe al tener det(Q)!= 0)
Q_inv = inv(Q)

%Toma la última fila (lr = last row) de Q_inv, requerida para generar la matriz Tc_inv
Q_inv_lr = Q_inv(dim_A(1,1),:)

Tc_inv =zeros(dim_A(1,1),dim_A(1,2));
for i=0:dim_A(1,1)-1
   Tc_inv(i+1,:)=Q_inv_lr*A^i;
end

%Muestra matriz Tc_inv
Tc_inv
```

```
%Muestra matriz Tc_inv
Tc = inv(Tc_inv)

%************* POLINOMIO CARACTERÍSTICO DESEADO ************************
%*************************************************************************************************
Pd = poly(raices)

%Dado que ahora tenemos un sistema de tipo 0 ( sin integrador), aumentamos
%el tipo añadiendo un integrador y por tanto tenemos una variable de estado
%adicional x0. Obtenemos las matrices equivalentes de orden superior
%(5)Ai,Bi,Ci


%******** MATRICES EQUIVALENTES CON REALIMENTACIÓN DE ESTAD0 Y SALIDA ******
%*************************************************************************************************************************

%Se requiere ahora trabajar con notacion simbolica en Matlab
syms s K0   kc1 kc2 kc3 kc4   k1 k2 k3 k4   x0 x1 x2 x3 x4   a0 a1 a2 a3   b0 b1 b2 b3
K=[k1 k2 k3 k4]
Kc=[kc1 kc2 kc3 kc4]
X=[x0 x1 x2 x3 x4]


%************ MATRIZ Arc CALCULADA SIN CAMBIO DE BASE**********************
%*************************************************************************************************

Prueba_c1 = [0;B*K0];
Prueba_c2 = [-C;A+B*K];
Prueba=[Prueba_c1 Prueba_c2]

I=eye(5);
matriz_det = s*I-Prueba

poly_real_calc = det(s*I-Prueba);

poly_real_calc = collect(poly_real_calc)
pretty(poly_real_calc)

roots(poly_real_calc)

%Si igualamos coeficientes nos queda el sistema lineal F*K=G. Despejar K

K = [-0.1500    1.2000   -1.1500   -2.3000]

F=[0 -2 0 2;-2 0 2 0;0 20 0 0;20 0 0 0]
G=[7 38 21.7 13]
K=linsolve(F,G')
```

9.1.2. Programa con Modelo Simplificado

```
% Autor: Jose A. Cerezuela
%  Proyecto fin de carrera: Aplicacion de tecnicas clasicas y de
control
% sobre minirobot con configuracion Segway.  Año 2013
%
%SERVOCONTROL - MODELO SIMPLE - PENDULO CON CARRO. OGATA
%Este programa calcula la matriz de realimentación de estado K y la
%ganancia Ko así como las matrices totales con realim estado y salida.

%Para girar simplemente se introduce un offset en u_r y u_l.

clc;
clear;
%**************************                DATOS                SISTEMA
*****************************
%*************************************************************************
****
%Gravedad
g = 9.81;
%Masa del Cuerpo (Kg)
m = 0.6;
%Masa del Carro (Kg) (Peso Ruedas)
M = 0.06;
%Distancia CM cuerpo a Eje Giro (metros)
l = 0.140;


%**************************      POSICIÓN      DE      LAS      RAICES
***********************
%*************************************************************************
*****
%Ahora al añadir un integrador (1/s) al sistema este pasa
%de orden 4 a 5. Se añade por tanto una raiz nº5.

raiz1 = -10;
raiz2 = -10;
raiz3 = -10;
raiz4 = -10;
raiz5 = -10;          %Raiz adicional
raices = [raiz1, raiz2, raiz3, raiz4 raiz5];

%******************        ELECCIÓN        DE        VARIABLES        DE
ESTADO*********************
%*************************************************************************
****

% x1 = tita (angulo cuerpo)
% x2 = d_tita
% x3 = posición robot
% x4 = velocidad lineal robot

%*******************       MATRICES       DE       ESTADO       A,B,C,D
************************
%*************************************************************************
****
```

```
%Obtención de Matriz de Estado (4x4)
A=zeros(4);
A(1,2)=1;
A(2,1)=(M+m)*g/(l*M);
A(3,4)=1;
A(4,1)=-(m*g)/M ;
%Muestra matriz A
A
%Obtención de Matriz de Entrada (4x1)
B=zeros(4,1);
B(2,1)= -1/(M*l);                                                    %CORREGIR
2
B(4,1)= 1/M ;                                              %CORREGIR 2
%Muestra matriz B
B
%Obtención de Matriz de Salida (1x4)
C=[0 0 1 0];

%Muestra matriz C
C
%Muestra matriz D
D=zeros(1,1)

%Matrices de estado y salida del sistema compuesto (real. estado y
salida
%con integrador y constante integral).
A1 = [A zeros(4,1);-C 0];
B1 = [B;0];

%************************       Q,       Q_inv,       Tc_inv,       Tc
************************
%*********************************************************************
****

%Obtengamos la matriz de transformación a la forma canónica
controlable Tc y Tc_inv (variable de fase):
%El primer paso es obtener la matriz de controlabilidad

dim_A1 = size(A1)

Q = zeros(dim_A1(1,1),dim_A1(1,2));
for i=0:dim_A1(1,1)-1
    Q(:,i+1) = A1^i*B1;
end
Q
rango_Q= rank(Q)
%Muestra matriz inversa de Q (existe al tener det(Q)!= 0)
Q_inv = inv(Q)

%Toma la última fila (lr = last row) de Q_inv, requerida para generar
la matriz Tc_inv
Q_inv_lr = Q_inv(dim_A1(1,1),:)

Tc_inv =zeros(dim_A1(1,1),dim_A1(1,2));
for i=0:dim_A1(1,1)-1
    Tc_inv(i+1,:)=Q_inv_lr*A1^i;
end
```

```
%Muestra matriz Tc_inv
Tc_inv

%Muestra matriz Tc_inv
Tc = inv(Tc_inv)

%**************         POLINOMIO         CARACTERÍSTICO         DESEADO
************************
%*************************************************************************
****
Pd = poly(raices)

%Dado que ahora tenemos un sistema de tipo 0 ( sin integrador),
aumentamos
%el tipo añadiendo un integrador y por tanto tenemos una variable de
estado
%adicional x0. Obtenemos la matrices equivalentes de orden superior
%(5)Ai,Bi,Ci


Phi = polyvalm(Pd,A1);

%La matriz de realimentación de estado viene dada por:
KK = [0 0 0 0 1]*Q_inv *Phi
%Desglosamos sus componentes
k1 = KK(1);
k2 = KK(2);
k3 = KK(3);
k4 = KK(4);
KI = -KK(5);     % ¡Ojo!. Hay que poner negativo (* -1) el valor que
devuelve KI

K=[k1 k2 k3 k4];

%% Respuesta a impulso en entrada (referencia)
%Prueba a condiciones iniciales no nulas
%AA,BB,CC,DD son las matrices de representación de estado del sistema
%compuesto (Realim. estado y salida con integrador).

AA = [A-B*K B*KI; -C 0]
BB = [0 0 0 0 1]'
CC = [C 0]
DD = [0]

%Simulamos que damos un Set Point de 100 mm para que lo siga el robot.
t=[0:0.02:3];                       %Simulación de 3 segundos
U=ones(size(t))*0.1;

sys=ss(AA,BB,CC,DD);
[Y,t,X]=lsim(sys,U,t);              %Simulación de entrada U en t

%Gráfica con todas las señales juntas (vectores fila)
x1 = X(:,1)';
x2 = X(:,2)';
x3 = X(:,3)';
x4 = X(:,4)';

% x1 = tita (angulo cuerpo)
```

```
% x2 = d_tita
% x3 = posición robot
% x4 = velocidad lineal robot

subplot(2,2,1);
plot(t,x1);grid
title('Ángulo del Cuerpo vs t')
xlabel('t (s)')
ylabel('x1 (Grados)')


subplot(2,2,2);
plot(t,x2);grid
title('Veloc. angular de Cuerpo vs t')
xlabel('t (s)')
ylabel('x2  ( Grados/s)')

subplot(2,2,3);
plot(t,x3);grid
title('Posición de Robot (x) vs t')
xlabel('t (s)')
ylabel('x3  ( m)')

subplot(2,2,4);
plot(t,x4);grid
title('Vel. lineal Robot vs t')
xlabel('t (s)')
ylabel('x4  ( m/s)')
```

```
%SERVOCONTROL - MODELO AVANZADO - PENDULO CON CARRO. OGATA
%Este programa calcula la matriz de realimentación de estado K y la
%ganancia Ko así como las matrices totales con realim estado y salida.

%****************** ELECCIÓN DE VARIABLES DE ESTADO*******************
%*********************************************************************************************

% x1 = ángulo rueda (Encoder)
% x2 = ángulo cuerpo (Integración Gyro)
% x3 = velocidad angular rueda (Derivación Encoder)
% x4 = velocidad angular cuerpo (Gyro)

clc;
clear;
%************************* DATOS SISTEMA ******************************
%**********************************************************************************

%Declaración de variables y asignación de valores
g=9.81;
m=0.03;
R=0.041;
Jw=(m*R^2)/2;
M=0.6;
W=0.142;
D=0.045;
H=0.280;
L=H/2;
Jpsi=(M*L^2)/3;
Jfi=M*(W^2+D^2)/12;
Jm=1e-5;
Rm=6.69;
Kb=0.468;
Kt=0.317;
n=1;
fm=0.0022;
fw=0;

%Cálculo de matriz de estado y entrada (A,B)
alfa=(n*Kt)/Rm;
alfa

beta=((n*Kt*Kb)/Rm)+ fm;
beta

E(1,1)=(2*m+M)*R^2+2*Jw+2*n^2*Jm;
E(1,2)=M*L*R-2*n^2*Jm;
E(2,1)=M*L*R-2*n^2*Jm;
E(2,2)=M*L^2+Jpsi+2*n^2*Jm;
E

F=[(beta+fw) -beta;-2*beta 2*beta];
```

```
F

G=[0 0;0 -M*g*L];
G

H=[alfa/2 alfa/2;-alfa -alfa];
H

I=beta+(W/R)*fw;
I

J=(R/W)*alfa;
J

K=(0.5*m*W^2+Jpsi+(W^2/(2*R^2))*(Jw+n^2*Jm));
K

%Determinación de los elementos de la ecuación de estado

det_E=det(E); %Calculamos el determinante como variable para operación más eficaz

%****************** MATRICES DE ESTADO A,B,C,D ****************************
%**************************************************************************************************
A=zeros(4,4);
A(1,:)=[0 0 1 0];
A(2,:)=[0 0 0 1];
A(3,2)=-M*g*L*E(1,2)/det_E;
A(4,2)=M*g*L*E(1,1)/det_E;
A(3,3)=-[(beta+fw)*E(2,2)+2*beta*E(1,2)]/det_E;
A(4,3)=[(beta+fw)*E(1,2)+2*beta*E(1,1)]/det_E;
A(3,4)=beta*E(2,2)+2*E(1,2)/det_E;
A(4,4)=-beta*(E(1,2)+2*E(1,1))/det_E;
A

B=zeros(4,1);
B(3,:)=alfa*(E(2,2)/2+E(1,2))/det_E;
B(4,:)=-alfa*(E(1,2)/2+E(1,1))/det_E;
B

C=[1 0 0 0];              %Realim. Ángulo de Rueda (x1)

D=[0];


%************************ POSICIÓN DE LAS RAICES **********************
%**************************************************************************************
%Ahora al añadir un integrador (1/s) al sistema este pasa
%de orden 4 a 5. Se añade por tanto una raiz nº5.

raiz1 = -1+3*i;
raiz2 = -1-3*i;
raiz3 = -2;
```

```
raiz4 = -2;
raiz5 = -2;        %Raiz adicional
raices = [raiz1, raiz2, raiz3, raiz4 raiz5];

%Matrices de estado y salida del sistema compuesto (real. estado y salida
%con integrador y constante integral).
A1 = [A zeros(4,1);-C 0];
B1 = [B;0];

%************************ Q, Q_inv, Tc_inv, Tc ***************************
%*************************************************************************

%Obtengamos la matriz de transformación a la forma canónica controlable Tc y Tc_inv (variable de
fase):
%El primer paso es obtener la matriz de controlabilidad
dim_A1 = size(A1)
Q = zeros(dim_A1(1,1),dim_A1(1,2));
for i=0:dim_A1(1,1)-1
   Q(:,i+1) = A1^i*B1;
end
Q
rango_Q= rank(Q)
%Muestra matriz inversa de Q (existe al tener det(Q)!= 0)
Q_inv = inv(Q)

%Toma la última fila (lr = last row) de Q_inv, requerida para generar la matriz Tc_inv
Q_inv_lr = Q_inv(dim_A1(1,1),:)

Tc_inv =zeros(dim_A1(1,1),dim_A1(1,2));
for i=0:dim_A1(1,1)-1
   Tc_inv(i+1,:)=Q_inv_lr*A1^i;
end

%Muestra matriz Tc_inv
Tc_inv

%Muestra matriz Tc_inv
Tc = inv(Tc_inv)

%************* POLINOMIO CARACTERÍSTICO DESEADO ***********************
%**********************************************************************************************
Pd = poly(raices)

%Dado que ahora tenemos un sistema de tipo 0 ( sin integrador), aumentamos
%el tipo añadiendo un integrador y por tanto tenemos una variable de estado
%adicional x0. Obtenemos la matrices equivalentes de orden superior
%(5)Ai,Bi,Ci


Phi = polyvalm(Pd,A1);

%La matriz de realimentación de estado viene dada por:
```

```
KK = [0 0 0 0 1]*Q_inv *Phi
%Desglosamos sus componentes
k1 = KK(1);
k2 = KK(2);
k3 = KK(3);
k4 = KK(4);
KI = -KK(5);    % ¡Ojo!. Hay que poner negativo (* -1) el valor que devuelve KI

K=[k1 k2 k3 k4];

%% Respuesta a impulso en entrada (referencia)
%Prueba a condiciones iniciales no nulas
%AA,BB,CC,DD son las matrices de representación de estado del sistema
%compuesto (Realim. estado y salida con integrador).

AA = [A-B*K B*KI; -C 0]
BB = [0 0 0 0 1]'
CC = [C 0]
DD = [0]

%Simulamos que damos un Set Point de 90 grados (1/4 vuelta de rueda) para que lo siga el robot.
t=[0:0.02:5];                   %Simulación de 5 segundos
U=ones(size(t))*90;

sys=ss(AA,BB,CC,DD);
[Y,t,X]=lsim(sys,U,t);              %Simulación de entrada U en t

%Gráfica con todas las señales juntas
x1 = X(:,1)';
x2 = X(:,2)';
x3 = X(:,3)';
x4 = X(:,4)';

% x1 = ángulo rueda (Encoder)
% x2 = ángulo cuerpo (Integración Gyro)
% x3 = velocidad angular rueda (Derivación Encoder)
% x4 = velocidad angular cuerpo (Gyro)


subplot(2,2,1);
plot(t,x1);grid
title('Ángulo de Rueda vs t')
xlabel('t (seg)')
ylabel('x1 ( º )')

subplot(2,2,2);
plot(t,x2);grid
title('Ángulo de Cuerpo vs t')
xlabel('t (seg)')
ylabel('x2  ( º )')

subplot(2,2,3);
```

```
plot(t,x3);grid
title('Veloc. Angular Rueda vs t')
xlabel('t (seg)')
ylabel('x3  ( º/seg )')

subplot(2,2,4);
plot(t,x4);grid
title('Vel. Angular Cuerpo(d_x) vs t')
xlabel('t (seg)')
ylabel('x4  ( º/seg )')
```

9.2. Programas en RobotC

9.2.1. Control PD- Estabilidad NXTway con Sensor de Luz

```
/*
* Autor: Jose A. Cerezuela
* Proyecto fin de carrera: Aplicacion de tecnicas clasicas y de control
* sobre minirobot con configuracion Segway.  Año 2013
*
// Este programa mantiene en equilibrio vertical (oscilante estable) el minirobot NXTway
// mediante un control PD simple.
// Se utiliza el SENSOR DE LUZ como realimentacion de verticalidad (lazo cerrado PD)
// Sin realimentacion/control de posicion del robot (lazo abierto)
// Incluye tarea toma datos en fichero TomaDatos1.txt
*/

//------------------------------- INCLUDE FUNCIONES JOYSTICK --------------------------------
#include "JoystickDriver.c"

//------------------------------- CONFIGURACION SENSORES / ACTUADORES ----------------------
---------
//Configuracion de Motores en Menu de RobotC
#pragma config(Motor,  motorA,          R_Motor,       tmotorNormal, openLoop, )
#pragma config(Motor,  motorB,          L_Motor,       tmotorNormal, openLoop, )
#pragma config(Sensor, S3,          lightSensor,        sensorLightActive)


//---------------------------------------- VARIABLES -------------------------------------

//Variables - Sensores
tSensors GyroSensor = S1;
tSensors AccelSensor = S2;
tSensors LightSensor = S3;
tSensors SonarSensor = S4;

  // Variables - Sensor Luz
  int Light_Raw = 0;
  float Light_Offset = 0;
  float Light_Net = 0;
  float old_Light_Net = 0;
  float der_Light_Net =0;
  float delta_Light = 0;
  float delta_t = 20;

  // Variables para controlador PD
  float error = 0;
  float d_error_pos_filtrado = 0;
  float d_error_neg_filtrado = 0;
  float kp = 13;
  float kd = 5;
```

```
  int u_PD = 0;
  float PID_Prop = 0;
  float PID_Deriv = 0;

  // Variables Giro Ruedas;
  float giro_ruedas = 0;
  int pos_real = 0;
          int radio_rueda = 45; // medida en mm

  //*********************************************************************
  //***************************** SUBRUTINAS ****************************
  //*********************************************************************

//*****************  TAREA TOMA DATOS A ARCHIVO TomaDatos1.txt ***************
task toma_datos()
{
// TOMA DE DATOS EN FICHERO INTERNO  TomaDatos1.txt
  TFileHandle etiqueta;     //Etiqueta del archivo
  TFileIOResult resultado; //Flag resultado de cualquier operacion con un archivo

//Definicion del tamano del archivo
int longitud = 15000;    //Longitud del archivo en bytes

//Para guardar a fichero
string datosalida;

// Borra y abre un nuevo archivo para almacenar datos comportamiento robot
Delete("TomaDatos1.txt",resultado);
OpenWrite(etiqueta,resultado,"TomaDatos1.txt",longitud);

//Inicializo el timer
ClearTimer(T1);
time1[T1]=0;

wait10Msec(100); // Espera  para tomar datos

int j;
for(j= 0; j<= 1500; j++)  // Toma 1500 filas de datos en archivo TomaDatos1.txt
{

// Primera columna el Tiempo en ms
sprintf(datosalida, "%d", time1[T1]);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);

// Segunda columna el Error (angulo inclinacion robot)
sprintf(datosalida, "%f", error);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);

// Tercera columna Salida a Motores
sprintf(datosalida, "%d", u_PD);
```

```
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);

// Cuarta columna Posicion Real robot (mm)
sprintf(datosalida, "%d",pos_real );
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);

// Nueva Fila de datos
datosalida="\r\n";
WriteText(etiqueta,resultado,datosalida);

// Retardo entre toma de datos
wait10Msec(5);                          // 20 datos por segundo

        }
CloseAllHandles(resultado);
}

//************************** CALCULAR OFFSET LUZ ******************************
//Subrutina usada para el Offset SENSOR LUZ
 void Calc_Luz_Offset()
 {
        Light_Offset = SensorValue(LightSensor);
        nxtDisplayString(1,"LOffset %f",Light_Offset);
 }

 //************************** DERIVA VALOR LUZ *******************************
 //Deriva el valor de senal de luz
        task derivada_luz()
        {
        while(true)
                {
                wait1Msec(delta_t);
                delta_Light = Light_Net - old_Light_Net;

                if(delta_Light != 0)  der_Light_Net = delta_Light/ delta_t;
                else der_Light_Net = 0;

                old_Light_Net = Light_Net;
                }
        }

//*****************************************************************************
//************************** FUNCION MAIN *************************************
//*****************************************************************************

  task main ()
{
//Configuracion de diversos sensores
SensorType[GyroSensor] = sensorI2CHiTechnicGyro;
SensorType[AccelSensor] = sensorI2CCustomFast;
```

```
SensorType[LightSensor] = sensorLightActive;
SensorType[SonarSensor] = sensorSONAR;


  //Resetea valor de encoder ambas ruedas
  nMotorEncoder[motorA]=0;
  nMotorEncoder[motorB]=0;

 //Llamada Funcion para almacenar Offset Luz en equilibrio inicial ----------
Calc_Luz_Offset();

//********************** Inicio de TAREAS ************************
 //-------------------------------------------------------------------------

    // Llamada de Tarea toma de Datos Robot en archivo .txt en Ladrillo
StartTask(toma_datos);

// Llamada de Tarea calculo derivada luz
StartTask(derivada_luz);

// Salida controlador PD (Prop.Deriv) discreto para mantener estabilidad
while(true)
            {

//********************** Calculo valor LUZ NETO ************************
 //-------------------------------------------------------------------------
Light_Raw = SensorValue(LightSensor);
Light_Net = Light_Raw - Light_Offset;


//Calcula el desplazamiento lineal robot (resolucion de 1 grado) 360 = 1 vuelta
   giro_ruedas = (nMotorEncoder[motorA] + nMotorEncoder[motorB])/2;
pos_real = degreesToRadians(giro_ruedas)*radio_rueda; //Posicion en mm


   //********************** Calculo Senal Control Motores ********************
 //-------------------------------------------------------------------------
   //Calculo de Salida Control PD
   error = -Light_Net; // + giro_ruedas/100;

  //Calculo Componente Proporcional de PD
   PID_Prop = (kp*error);

   //Calculo Componente Derivativa de PD
  if(der_Light_Net > 0)
  {
             d_error_pos_filtrado = der_Light_Net;
            PID_Deriv = -(kd*d_error_pos_filtrado);
 }
           if(der_Light_Net < 0)
            {
            d_error_neg_filtrado = der_Light_Net;
```

```
            PID_Deriv = -(kd*d_error_neg_filtrado);
            }

//Senal salida a motores (P+D)
u_PD = PID_Prop + PID_Deriv;

//Limita valores maximos a motores (+100, -100)
if (u_PD > 100) u_PD = 100;
if (u_PD < -100) u_PD = -100;

 //Activacion de Motores con Salida de Controlador
 //nSyncedMotors = synchAC;
  motor[motorA] = u_PD;
  motor[motorB] = u_PD;

          }
 }
```

9.2.1. Control PD - Estabilidad NXTway-G con Sensor Gyro

```
/*
* Autor: Jose A. Cerezuela
* Proyecto fin de carrera: Aplicacion de tecnicas clasicas y de control
* sobre minirobot con configuracion Segway.  Año 2013
*
// Este programa mantiene en equilibrio vertical (oscilante estable) el minirobot NXTway
// mediante un control PD simple.
// Se utiliza el SENSOR GIROSCOPIO ELECTRONICO como realimentacion de verticalidad
// Sin realimentacion/control de posicion del robot (lazo abierto)
// Incluye tarea toma datos en fichero TomaDatos1.txt
*/
// Se utiliza subprograma inicial para determinar el offset promedio del Gyro.

//------------------------------- INCLUDE FUNCIONES JOYSTICK -------------------------------
#include "JoystickDriver.c"

//------------------------------- CONFIGURACION SENSORES / ACTUADORES ----------------------
//Configuracion de Motores en Menu de RobotC
#pragma config(Motor,  motorA,          R_Motor,       tmotorNXT, openLoop, encoder )
#pragma config(Motor,  motorB,          L_Motor,       tmotorNXT, openLoop, encoder )
#pragma config(Sensor, S3,     lightSensor,         sensorLightActive)
//-------------------------------------------

//---------------------------------------- VARIABLES -----------------------------------------

//Variables - Sensores
tSensors GyroSensor = S1;
tSensors AccelSensor = S2;
tSensors LightSensor = S3;
tSensors SonarSensor = S4;

 // Variables - Integrador de Gyro
float delta_t = 20; //Sample Rate en ms para integracion
int dt = 0.010; // Tiempo de calculo basico general 10ms

float angle_net = 0;
float Gyro_Raw = 0;
float Gyro_Net = 0;
float Gyro_Net_Old = 0;
float Gyro_Offset_Acum = 0;

float Gyro_Offset = 0;

float PID_Prop = 0;
float PID_Deriv = 0;

// Variables para controlador PD
 float error = 0;
```

```
float d_error_pos_filtrado = 0;
float d_error_neg_filtrado = 0;
float kp = 18;
float kd = 0.5;
int u_PD = 0;

// Variables Giro Ruedas;
float giro_ruedas = 0;
float pos_real = 0;
int radio_rueda = 45; // medida en mm

//Variable Joystick
int move_fw = 0;
int move_bw = 0;
int move_right = 0;
int move_left =0;

//*********************************************************************************
//***************************** SUBRUTINAS ********************************
//*********************************************************************************

  ////**************  TAREA TOMA DATOS A ARCHIVO .txt ***********
task toma_datos()
{
// TOMA DE DATOS EN FICHERO INTERNO  TomaDatos1.txt
  TFileHandle etiqueta;     //Etiqueta del archivo
  TFileIOResult resultado; //Flagindica el resultado de cualquier operacion con un archivo

//Definir tamano de archivo almacenamiento de datos
int longitud = 15000;    //Longitud del archivo en bytes
//Para guardar a fichero
string datosalida;

// Borra y abre un nuevo archivo para almacenar datos comportamiento robot
Delete("TomaDatos1.txt",resultado);
OpenWrite(etiqueta,resultado,"TomaDatos1.txt",longitud);

//Inicializo el timer
ClearTimer(T1);
time1[T1]=0;

int j;
for(j= 0; j<= 2500; j++)  // Toma 2500 filas de datos en archivo .txt
{

// Primera columna el Tiempo en ms
sprintf(datosalida, "%d", time1[T1]);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);
nxtDisplayString(1,datosalida);

                    // Segunda columna el Error
```

```
sprintf(datosalida, "%f", error);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);
nxtDisplayString(2,datosalida);

// Tercera columna Salida a Motores (%)
sprintf(datosalida, "%d", u_PD);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);
nxtDisplayString(3,datosalida);

// Tercera columna giro medio ruedas (grados)
sprintf(datosalida, "%d",giro_ruedas );
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);
nxtDisplayString(4,datosalida);

// Nueva Fila de datos
datosalida="\r\n";
WriteText(etiqueta,resultado,datosalida);

// Retardo entre toma de datos
wait10Msec(5);

        }
        CloseAllHandles(resultado);
}
 //************************** CALCULAR OFFSET GYRO ********************************

//Subrutina usada para el Offset promedio del Sensor Gyro
 void Calc_Gyro_Offset()
 {
 int i=0;
 int veces = 50;

 while(i<veces)
  {
        Gyro_Raw = SensorValue[GyroSensor];
        wait1Msec(2);
        Gyro_Raw = (Gyro_Raw + SensorValue[GyroSensor])/2;

         Gyro_Offset_Acum = Gyro_Offset_Acum + Gyro_Raw;
        wait1Msec(50);
         i++;
  }

 Gyro_Offset = (Gyro_Offset_Acum/veces);
nxtDisplayString(3,"Offset %d",Gyro_Offset);
PlayTone(2000,20);

}
```

```
 //************************** INTEGRAR ****************************************
 //Subrutina usada para integrar la velocidad angular dada por el Gyro
 task Integrar()
 {
          while(true)
          {
          //Metodo antiguo
          Gyro_Raw = SensorValue[GyroSensor];
          wait1Msec(2);
            Gyro_Raw = ((Gyro_Raw + SensorValue[GyroSensor])/2)-Gyro_Offset;
          wait1Msec(delta_t);
          Gyro_Raw = SensorValue(GyroSensor);
          Gyro_Net = Gyro_Raw - Gyro_Offset;
          angle_net = angle_net + ((Gyro_Net + Gyro_Net_Old)/2) * (delta_t/1000);
            Gyro_Net_Old = Gyro_Net;
          angle_net = angle_net - 0.01; // Correcion aprox. de deriva giro
          }
 }


//*****************************************************************************
//************************** FUNCION MAIN ************************************
//*****************************************************************************
  task main ()
{
 //---------------------------------------------------------------------------
//Configuracion de Sonar (LO CONFIGURO AQUI PUES CON EL AUTOCONFIG DE ROBOTC EN
SONAR NO FUNCIONA
SensorType[GyroSensor] = sensorI2CHiTechnicGyro;
SensorType[AccelSensor] = sensorI2CCustomFast;
SensorType[LightSensor] = sensorLightActive;
SensorType[SonarSensor] = sensorSONAR;

// Llamada Subrutina para calculo offset Gyro
Calc_Gyro_Offset();

// Llamada de Tarea toma de Datos Robot en archivo .txt en Ladrillo
StartTask(Integrar);

// Llamada de Tarea toma de Datos Robot en archivo .txt en Ladrillo
StartTask(toma_datos);

 //Resetea valor de encoder ambas ruedas
  nMotorEncoder[motorA]=0;
  nMotorEncoder[motorB]=0;

  //---------------------------------------------------------------------------
  // Salida controlador PD (Prop.Deriv) discreto para mantener estabilidad
  while(true)
 {
//Calcula el desplazamiento lineal robot (resolucion de 1 grado) 360 = 1 vuelta
 giro_ruedas = (nMotorEncoder[motorA] + nMotorEncoder[motorB])/2;
```

```
pos_real = degreesToRadians(giro_ruedas)*radio_rueda; //Posicion en mm
 nxtDisplayString(1,"%d",giro_ruedas);

  //Compensación por deriva
  error = angle_net ;//+ (giro_ruedas/100);

  //Calculo PID_Prog
  PID_Prop = (kp*error);

  //Calculo PID_Deriv
  if(Gyro_Net > 0)
  {
  d_error_pos_filtrado = Gyro_Net;
  PID_Deriv = (kd*d_error_pos_filtrado);
  }
  if(Gyro_Net < 0)
  {
  d_error_neg_filtrado = Gyro_Net;
  PID_Deriv = (kd*d_error_neg_filtrado);
  }

 //Entrada Joystick
  getJoystickSettings(joystick);
  move_fw = (joystick.joy1_y1)/3;
  move_bw = (joystick.joy1_y2)/3;
  move_right = (joystick.joy1_x1)/3;
  move_left = (joystick.joy1_x2)/3;

  u_PD = PID_Prop + PID_Deriv;

  if (u_PD > 100) u_PD = 100;
  if (u_PD < -100) u_PD = -100;

  //-------------------------------------------------------------------------
  //Activacion de Motores con Salida de Controlador

 //nSyncedMotors = synchAC;
  motor[motorA] = u_PD;// + move_fw + move_bw + move_right - move_left;
  motor[motorB] = u_PD;// + move_fw + move_bw - move_right + move_left;

   //wait1Msec(5);

  }

}
```

9.2.1. Control Realimentación Estado NXTway-G (Sensor Gyro)

```
/*
* Autor: Jose A. Cerezuela
* Proyecto fin de carrera: Aplicacion de tecnicas clasicas y de control
* sobre minirobot con configuracion Segway.  Año 2013
*
// EJECUCION DE TAREAS EN PARALELO - MODELO BASICO CARRO - REAL ESTADO SOLO
// Este programa pretende mantener en equilibrio vertical del Robot
// mediante realimentacion de estado simple. La matriz K de realim. de estado
// en funcion de la posicion de los polos se ha obtenido mediante programa Matlab.

// SELECCION DE VARIABLES DE ESTADO
// x1 = posicion x
// x2 = velocidad lineal
// x3 = angulo cuerpo
// x4 = velocidad angular cuerpo
*/
///------------------------------- CONFIGURACION SENSORES / ACTUADORES ----------------------
----------
//Configuracion de Motores en Menu de RobotC
#pragma config(Motor,  motorA,          R_Motor,       tmotorNormal, openLoop, )
#pragma config(Motor,  motorB,          L_Motor,       tmotorNormal, openLoop, )

//---------------------------------------- VARIABLES -----------------------------------------

  //Variables - Sensores
  tSensors GyroSensor = S1;
           tSensors AccelSensor = S2;
  tSensors SonarSensor = S4;

  // Variables - Integrador de Gyro
  float delta_t = 20;

  float angle_net = 0;

  float Gyro_Raw = 0;
  float Gyro_Net = 0;
  float Gyro_Net_Old = 0;

  float Gyro_Offset = 0;


  // Variables para calculo velocidad robot
  long  Wheel_angle_l = 0;
  long  Wheel_angle_r = 0;
  float Wheel_mean_pos = 0;
  float Wheel_mean_angle = 0;

  float Wheel_mean_angle_old = 0;
  float Wheel_mean_angle_new = 0;
```

```
  float  Wheel_angle_incr = 0;
  float  radio_rueda = 0.045;

  float Calt_time = 10;                          // en ms

  float Wheel_ang_speed = 0;
  float Wheel_lin_speed = 0;

  //Control de posicion de robot
          float pos_real =0;                                          //   Posicin Rotot (desplaz. sobre
suelo)
  float pos = 0;                                                      // Posicion incluyendo en sp_pos
( sumado) para control error
  float    sp_pos = 0;                                        // Set Point deseado en posicion

  //Variables de espacio de estado
  float x1 = 0;
  float x2 = 0;
  float x3 = 0;
  float x4 = 0;

  // Matriz de realimentacion de estado
          // x1 = posicion x
          // x2 = velocidad lineal
          // x3 = angulo cuerpo
          // x4 = velocidad angular cuerpo
  //                    k1       k2       k3       k4
  // 4 Raices en -30 -->   347.1429   46.2857   53.4480    3.7440
  // 4 Raices en -25 -->   167.4107   26.7857   33.9368    2.2950
  // 4 Raices en -20 -->   68.5714   13.7143   21.3480    1.2960
  // 4 Raices en -15 -->   21.6964    5.7857   13.6568    0.6570
  // 4 Raices en -10 -->   4.2857    1.7143    9.2880    0.2880

  float k1 = 150;  //15;
  float k2 = 16;    //6;
  float k3 = 13;    //16;
  float k4 = 0.1;   //1;

  // Senal de salida a motores con la realimentacion de estado
  float u_motor_ss = 0;
  float u_motor_out = 0;

  // Correccion de deriva de gyro cada 100 ms
  float incr_ang_gyro = -0.1;


  //*************************************************************************************
  //****************************** SUBRUTINAS ***********************************
  //*************************************************************************************

  //********************  TAREA TOMA DATOS A ARCHIVO .txt *********************
          task toma_datos()
```

```
        {
          // TOMA DE DATOS EN FICHERO INTERNO  TomaDatos1.txt
 TFileHandle etiqueta;     //Etiqueta del archivo
 TFileIOResult resultado; //Flagindica el resultado de cualquier operacion con un archivo

        //Definir tamano de archivo almacenamiento de datos
        int longitud = 25000;    //Longitud del archivo en bytes
        //Para guardar a fichero
        string datosalida;

        // Borra y abre un nuevo archivo para almacenar datos comportamiento robot
        Delete("TomaDatos1.txt",resultado);
        OpenWrite(etiqueta,resultado,"TomaDatos1.txt",longitud);

        //Inicializo el timer
        ClearTimer(T1);
        time1[T1]=0;

        int j;
        for(j= 0; j<= 2500; j++)  // Toma 2500 filas de datos en archivo .txt
        {

        // Primera columna el Tiempo en ms
        sprintf(datosalida, "%d", time1[T1]);
        strcat(datosalida,"\t");
        WriteText(etiqueta,resultado,datosalida);
        nxtDisplayString(1,datosalida);

                // Segunda columna el angulo Gyro
        sprintf(datosalida, "%f", angle_net);
        strcat(datosalida,"\t");
        WriteText(etiqueta,resultado,datosalida);

        // Tercera columna Salida a Motores (%)
        sprintf(datosalida, "%f",u_motor_ss );
        strcat(datosalida,"\t");
        WriteText(etiqueta,resultado,datosalida);

        // Cuarta columna giro medio ruedas (grados)
        sprintf(datosalida, "%f",Wheel_mean_pos );
        strcat(datosalida,"\t");
        WriteText(etiqueta,resultado,datosalida);

        // Quinta columna Set Point Posicion
        sprintf(datosalida, "%f", sp_pos);
        strcat(datosalida,"\t");
        WriteText(etiqueta,resultado,datosalida);

        // Sexta columna Posicion Robot Suelo (mm)
        sprintf(datosalida, "%f", pos_real);
        strcat(datosalida,"\t");
        WriteText(etiqueta,resultado,datosalida);
```

```
        //Septima columna Velocidad angular(rad/sg)
        sprintf(datosalida, "%f",Wheel_ang_speed);
        strcat(datosalida,"\t");
        WriteText(etiqueta,resultado,datosalida);

        // Nueva Fila de datos
        datosalida="\r\n";
        WriteText(etiqueta,resultado,datosalida);

        // Retardo entre toma de datos
        wait10Msec(5);

        }

        CloseAllHandles(resultado);
}

 //************************** CALCULO DE OFFSET INICIAL GYRO ***************************************
 void Calc_Gyro_Offset()
 {
        int Gyro_Offset_Acum = 0;

        int i=0;
 int veces = 20;
        while(i<veces)
  {
  Gyro_Offset_Acum = Gyro_Offset_Acum + SensorValue(GyroSensor);
  wait1Msec(50);
  i++;
  }
  Gyro_Offset = (Gyro_Offset_Acum/veces);
  PlayTone(2000,20);

  }

 //************************** CALCULO DE ENTRADA A MOTORES ***************************************
 //Subrutina usada para calcular la potencia de entrada a motores u .(En este caso se considera iguales
 //para der. e izq.)

 void Motor_input_ss()
 {
 // SELECCION DE VARIABLES DE ESTADO
        // x1 = posicion x
        // x2 = velocidad lineal
        // x3 = angulo cuerpo
        // x4 = velocidad angular cuerpo

        x1 = Wheel_mean_pos;
        x2 = Wheel_lin_speed;
        x3 = angle_net;
        x4 = Gyro_Net;
```

```
        u_motor_ss = (x1*k1 + x2*k2 + x3*k3 + x3*k4);
        u_motor_out = u_motor_ss;

        if (u_motor_ss > 100) u_motor_out = 100;
  if (u_motor_ss < -100) u_motor_out = -100;

}

 //*********************************************************************
//***************************** TAREAS ********************************
//*********************************************************************

 //************************** INTEGRAR *****************************************
//Tarea usada para integrar la velocidad angular dada por el Gyro
task Integrar()
{
          while(true)
          {
            wait1Msec(delta_t);

                    Gyro_Raw = SensorValue(GyroSensor);
                    Gyro_Net = Gyro_Raw - Gyro_Offset;

                              if(abs(Gyro_Net)>1)
                              {
                                angle_net = angle_net + ((Gyro_Net + Gyro_Net_Old)/2) * (delta_t/1000);

                              }

                    Gyro_Net_Old = Gyro_Net;
                  }

}

 //************************** CALCULO DE VELOCIDAD LINEAL ROBOT ******************************
//Subrutina usada para calcular la posicion y velocidad lineal del robot

task Robot_Speed()
{
        nMotorEncoder[motorA]=0;        //Reseteo contador encoder motorA
        nMotorEncoder[motorB]=0;        //Reseteo contador encoder motorB

        while(true)
         {
                  Wheel_angle_r = nMotorEncoder[motorA];
                   Wheel_angle_l = nMotorEncoder[motorB];
                   Wheel_mean_angle = (Wheel_angle_l + Wheel_angle_r)/2;

                   //Angulo medio ruedas(grados) en t-1
                   Wheel_mean_angle_old = Wheel_mean_angle;
```

```
                wait1Msec(Calt_time);

                Wheel_angle_r = nMotorEncoder[motorA];   //Grados
                Wheel_angle_l = nMotorEncoder[motorB];   //Grados
                Wheel_mean_angle = (Wheel_angle_l + Wheel_angle_r)/2;

                //Angulo medio ruedas(grados) en t-1
                Wheel_mean_angle_new = Wheel_mean_angle;
                        pos_real  =  degreesToRadians(Wheel_mean_angle)*radio_rueda;  //Posicion
robot en metros

                //Calculo velocidad angular (radianes/segundo)
                Wheel_angle_incr = (Wheel_mean_angle_new - Wheel_mean_angle_old);
                Wheel_ang_speed   =   (Wheel_angle_incr   /   (Calt_time/1000))*(PI/180);          //
rad/Segundo

                 // Calculo velocidad lineal ruedas (metros/segundo)
                Wheel_lin_speed = Wheel_ang_speed * radio_rueda;

                //Posicion ruedas en metros
                Wheel_mean_pos = (Wheel_mean_angle_new *(PI/180)* radio_rueda) ;
          }

 }


 //************************** TAREA COMPENSACION DERIVA GYRO ********************
 //Subrutina para compensar la deriva del Gyro
 task Drift_comp()
 {
   while(true)
          {
           angle_net = angle_net + incr_ang_gyro;
            wait1Msec(100);
          }
 }


//*********************************************************************************
//************************** FUNCION MAIN ***********************************
//*********************************************************************************

task main()
{

 //Configuracion de Sonar
 SensorType[GyroSensor] = sensorI2CHiTechnicGyro;
 SensorType[AccelSensor] = sensorI2CCustomFast;
 SensorType[SonarSensor] = sensorSONAR;

 //Inicio de Subrutinas
 Calc_Gyro_Offset();
```

```
        nVolume = 4;

  // Boost priority so that all tasks start immediately
  nSchedulePriority = kHighPriority;

  StartTask(Integrar);                    // Integra senal de Gyro
  StartTask(Robot_Speed);                 // Calcula velocidad lineal robot
  StartTask(Drift_comp);          // Compensacion de la deriva del Gyro aprox.
        StartTask(toma_datos);            //  Tarea toma de Datos

  // Restore to normal priority
  nSchedulePriority = kDefaultTaskPriority;

  while (true)
  {

    Motor_input_ss();

  //Activacion de Motores con Salida de Controlador
    motor[motorA] = u_motor_ss;
    motor[motorB] = u_motor_ss;
    nxtDisplayString(1, "Angulo Neto%d", angle_net);

  }
  return;

}
```

9.2.2. Control Combinado RE + PID NXTway-G (Sensor Gyro)

```
/*
* Autor: Jose A. Cerezuela
* Proyecto fin de carrera: Aplicacion de tecnicas clasicas y de control
* sobre minirobot con configuracion Segway.  Año 2013
*
// Este programa mantiene en equilibrio vertical (oscilante estable) el minirobot NXTway
// mediante un control combinado de realimentacion de estado y PID simple.
// Se utiliza el SENSOR GYRO como realimentacion de verticalidad asi como los ENCODERS
// para determinar el desplazamiento del robot.
// Incluye tarea toma datos en fichero TomaDatos1.txt
*/

//------------------------------- INCLUDE FUNCIONES JOYSTICK -------------------------------
#include "JoystickDriver.c"

//------------------------------- CONFIGURACION SENSORES / ACTUADORES ------------------
//Configuracion de Motores en Menu de RobotC
#pragma config(Motor,  motorA,          R_Motor,       tmotorNXT, openLoop, encoder )
#pragma config(Motor,  motorB,          L_Motor,       tmotorNXT, openLoop, encoder )
#pragma config(Sensor, S3,     lightSensor,         sensorLightActive)

//Variables - Sensores
  tSensors GyroSensor  = S1;
tSensors AccelSensor = S2;
tSensors LightSensor = S3;
 tSensors SonarSensor = S4;

const float diametro_rueda = 90;                        //diámetro en mm
const float radio_rueda = diametro_rueda/(2000;    // radio en metros

  //Variables Globales
 float Gyro_Raw = 0;                        // Valor bruto de gyro (grados/seg)
 float angle_net = 0;                       // Angulo de inclinacion del robot (grados)
 float Gyro_Net = 0;                        // Velocidad angular ruedas (grados/seg)
float Gyro_Offset = 0;                      // Offset inicial del Gyro (grados/seg)
float Gyro_Offset_Acum = 0;                 // Acumulador para calculo promedio Gyro Offset

float error = 0;//Error                     // Error desviacion de angle_net, Gyro_Net, pos
float d_error = 0;                          // Derivada Error
float i_error = 0;                          // Integral Error
float prev_error = 0;                       // Previous Error/ Error found in previous loop cycle

  // Variables para calculo velocidad robot
  long  Wheel_angle_l = 0;
  long  Wheel_angle_r = 0;
  float Wheel_mean_pos = 0;
  float Wheel_mean_angle = 0;

  float Wheel_mean_angle_old = 0;
  float Wheel_mean_angle_new = 0;
```

```
 float  Wheel_angle_incr = 0;
 float  Wheel_radius = 0.046;
 float Calt_time = 10;              // en ms
 float Wheel_ang_speed = 0;         //Velocidad angular en grados
 float Wheel_lin_speed = 0;         //Velocidad lineal en m/s


 float u_pid = 0;                   // Valor de salida controlador PID
 float pos_real =0;                 //  Posición Rotot (desplaz. sobre suelo)
 float pos = 0;                     // Posición incluyendo en sp_pos ( sumado) para control error
 float    sp_pos = 0;               // Set Point deseado en posicion

 int motorpower = 0;                // Potencia aplicada a los motores
 int pot_giro = 0;                  // Potencia para generar giro robor sobre su eje vertical

 int encoder = 0;

float gain_ang_vel;
float gain_ang;
float gain_posic;
float kp;
float ki;
float kd;
float dt;

 //Variable Joystick
 int move_Y_axis = 0;  // FW = 128  ; BW = -127
int rotate_X_axis = 0;  //Left = -128  ; Right = 127


 //************************** CALCULAR OFFSET GYRO *********************************
//Subrutina usada para el Offset promedio del Sensor Gyro
 void Calc_Gyro_Offset()
 {
 int i=0;
 int veces = 50;

 while(i<veces)
 {
           Gyro_Raw = SensorRaw[GyroSensor];
          wait1Msec(2);
          Gyro_Raw = (Gyro_Raw + SensorRaw[GyroSensor])/2;

           Gyro_Offset_Acum = Gyro_Offset_Acum + Gyro_Raw;
          wait1Msec(20);
           i++;
  }
 Gyro_Offset = (Gyro_Offset_Acum/veces);
nxtDisplayString(3,"Offset %d",Gyro_Offset);
   PlayTone(2000,20);

 }
```

```
 //************************* CALCULO DE VELOCIDAD LINEAL ROBOT ******************************
 //Subrutina usada para calcular la posicion y velocidad lineal del robot

 task Robot_Speed()
 {
 nMotorEncoder[motorA]=0;       //Reset contador encoder motorA
 nMotorEncoder[motorB]=0;       //Reset contador encoder motorB

 while(true)
           {

            Wheel_angle_l = nMotorEncoder[motorA];
           Wheel_angle_r = nMotorEncoder[motorB];
           Wheel_mean_angle = (Wheel_angle_l + Wheel_angle_r)/2;

           //Angulo medio ruedas(grados) en t-1
           Wheel_mean_angle_old = Wheel_mean_angle;

            wait1Msec(Calt_time);

           Wheel_angle_l = nMotorEncoder[motorA];   //Grados
           Wheel_angle_r = nMotorEncoder[motorB];   //Grados
           Wheel_mean_angle = (Wheel_angle_l + Wheel_angle_r)/2;
            //Angulo medio ruedas(grados) en t-1
            Wheel_mean_angle_new = Wheel_mean_angle;

           //Calculo velocidad angular (radianes/segundo)
            Wheel_angle_incr = (Wheel_mean_angle_new - Wheel_mean_angle_old);
           Wheel_ang_speed = (Wheel_angle_incr / (Calt_time/1000))*(PI/180);   // rad/Segundo

           // Calculo velocidad lineal ruedas (metros/segundo)
           Wheel_lin_speed = Wheel_ang_speed * Wheel_radius;

           //Posicion ruedas en metros
           Wheel_mean_pos = (Wheel_mean_angle_new *(PI/180)* Wheel_radius) ;
           }
  }

//*********************  TAREA TOMA DATOS A ARCHIVO .txt **********************
task toma_datos()
{
// TOMA DE DATOS EN FICHERO INTERNO  TomaDatos1.txt
  TFileHandle etiqueta;     //Etiqueta del archivo
  TFileIOResult resultado; //Flag que nos indica el resultado de cualquier operacion con un archivo

//Definicion del tamano del archivo
int longitud = 25000;       //Longitud del archivo en bytes 25k

//Para guardar a fichero
string datosalida;
```

```
// Borra y abre un nuevo archivo para almacenar datos comportamiento robot
Delete("TomaDatos1.txt",resultado);
OpenWrite(etiqueta,resultado,"TomaDatos1.txt",longitud);

//Inicializo el timer
ClearTimer(T2);
time1[T2]=0;

int j;
for(j= 0; j<= 2500; j++)  // Toma 1000 filas de datos en archivo .txt
{

// Primera columna el Tiempo en ms
sprintf(datosalida, "%d", time1[T2]);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);

// Segunda columna el Inclinacion Gyro
sprintf(datosalida, "%f", Gyro_Net);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);
nxtDisplayString(1,datosalida);


// Tercera columna el Error
sprintf(datosalida, "%f", error);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);
nxtDisplayString(2,datosalida);

// Cuarta columna Salida a Motores de PID
sprintf(datosalida, "%d", u_pid);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);
nxtDisplayString(3,datosalida);

// Quinta columna Set Point Posicion
sprintf(datosalida, "%f", -sp_pos);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);
nxtDisplayString(4,datosalida);

// Sexta columna Posicion Robot Suelo (mm)
sprintf(datosalida, "%f", pos_real);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);
nxtDisplayString(5,datosalida);

//Septima columna Velocidad angular(rad/s)
sprintf(datosalida, "%f",Wheel_ang_speed);
strcat(datosalida,"\t");
WriteText(etiqueta,resultado,datosalida);
```

```
nxtDisplayString(6,datosalida);

// Nueva Fila de datos
datosalida="\r\n";
WriteText(etiqueta,resultado,datosalida);

// Retardo entre toma de datos
wait10Msec(5);                          // Toma 20 datos por segundo

        }
        CloseAllHandles(resultado);
}

// ********************* Tarea de balanceo de robot en equilibrio *********************
//*********************************************************************************************
task balanceo()
{
  dt = 0.010;                   // Valor de delta_t referencia calculos

  // Parametros de control
  gain_ang = 20;
  gain_ang_vel = 0.5;
  gain_posic = 800;

  kp = 0.025;
  ki = 0.35;
  kd = 0.0002;

  //Inicializacion encoder motores
  nMotorEncoder[motorA] = 0;
  nMotorEncoder[motorB] = 0;

   ClearTimer(T1);              // This timer is used in the driver. Do not use it for other purposes!

  while(true)
  {
  //Lectura Valor del sensor Gyro
 Gyro_Raw = SensorValue[GyroSensor];
wait1Msec(2);
 Gyro_Raw = Gyro_Raw+SensorValue[GyroSensor];

//Calculo de velocidad angular neta e Integracion de angulo
 Gyro_Net = Gyro_Raw/2 - Gyro_Offset;
 Gyro_Offset = Gyro_Offset*0.997 + (0.003*(Gyro_Net+Gyro_Offset));
 angle_net = angle_net + Gyro_Net*dt;

//Calculo de posicion de robot
encoder = (nMotorEncoder[motorA] + nMotorEncoder[motorB])/2;
pos_real = encoder*(PI/180) * radio_rueda;
pos = pos_real + sp_pos;

//Calcula valor combinado del error (ponderado)
```

```
  error = (gain_ang * angle_net) + (gain_ang_vel * Gyro_Net)+ (gain_posic * pos);
d_error = (error - prev_error)/dt;
i_error = i_error + error*dt;
prev_error = error;

//Valor de salida del controlador PID
u_pid = (kp*error + ki*i_error + kd*d_error)/radio_rueda;


//Entrada giro de Joystick
pot_giro = rotate_X_axis;

//Control de Motores  Potenccia y  Giro Robot
  motorpower =   u_pid;

  motor[motorA] = motorpower - pot_giro  ;
  motor[motorB] = motorpower + pot_giro ;

  while(time1[T1] < dt*1000)
          {
            wait1Msec(1);
          }
          ClearTimer(T1);
    }
}


//********************* FUNCION MAIN ******************************
//*****************************************************************************

task main()
{
  //Inicia funcion de balanceo
  Calc_Gyro_Offset();
StartTask(balanceo);
StartTask(Robot_Speed);
StartTask(toma_datos);

  while(true)
  {
//Entrada Joystick
 getJoystickSettings(joystick);
 move_Y_axis = (joystick.joy1_y1)/10;
 rotate_X_axis = (joystick.joy1_x1)/1.7;

 /*Sincroniza motores si no pulsamos
 if (abs(rotate_X_axis)<=1)
{
 nSyncedMotors = synchAB;
nSyncedTurnRatio = 100;}
*/
 nxtDisplayString(1,"FW-BW: %d",move_Y_axis);
```

```
nxtDisplayString(3,"ROT: %d",rotate_X_axis);

// Control de Avance o Retroceso con Joystick
if(move_Y_axis >1)  // Mover hacia delante
{
sp_pos = sp_pos - 0.02;
 wait1Msec(100);
//PlayTone(500,20);
 }

if(move_Y_axis <-1)  // Mover hacia atras
{
sp_pos = sp_pos + 0.02;
wait1Msec(100);
 //PlayTone(2000,20);
  }

  }
 }
```