Aplicación de la Inteligencia Artificial al control de un robot Lego en configuración de péndulo invertido

Alumno: **Miguel Aguilera Zorzo**

Dirigido por: **Matilde Santos Peñas**

2018-2019

TRABAJO DE FIN DE GRADO DEL GRADO INGENIERÍA DEL SOFTWARE,
FACULTAD DE INFORMÁTICA, UNIVERSIDAD COMPLUTENSE DE MADRID

Nota del autor

Los expertos más competentes son incapaces de describir el conocimiento que usan para resolver los problemas.

Contenido

Resumen

En este trabajo se describe el diseño e implementación de la estabilización de un robot en configuración de péndulo invertido mediante la aplicación de algoritmos de control.
Para ello se ha construido un sistema que consiste en un robot LEGO. Este robot debe equilibrarse sobre sus dos ruedas mientras es capaz de moverse autónomamente esquivando obstáculos.
Se ha desarrollado un modelo matemático del sistema mediante las herramientas de simulación *Matlab*/*Simulink* sobre el que sea ha trabajado inicialmente. Se han aplicado técnicas de control LQR y PID para su estabilidad, implementado la sintonía inteligente de los parámetros del PID con un sistema experto. Se han implementado estas estrategias de control en el sistema real, utilizando los sensores y componentes hardware y software del lego EV3. El robot cuenta además con la habilidad de esquivar obstáculos gracias al sensor de ultrasonidos.
La implementación en código ha sido programada en Java, usando el paradigma de orientación a objetos.
Los resultados son satisfactorios y se ha conseguido procesar las señales de los sensores y estabilizar el robot.

Palabras clave: Control, péndulo invertido, LQR, PID, Sintonía experta, Lego, EV3

Summary

In this paper, is described the design and implementation of the stabilization of a robot in the inverted pendulum configuration through the application of control algorithms.
To achieve this purpose, a system consisting of a LEGO robot has been built. This robot must keep balance on its two wheels while being able to move autonomously dodging obstacles.
A mathematical model of the system has been developed using the Matlab / Simulink simulation tools on which it has worked. LQR and PID control techniques have been applied for stability, implementing intelligent tuning of the PID parameters with an expert system.
These control strategies have been implemented in the real system, using the lego EV3 hardware and software sensors as well as it's components. Furthermore, the robot also has the ability to dodge obstacles thanks to the ultrasonic sensor.
The code implementation has been programmed in Java, using the object orientation paradigm.
The results are satisfactory and the sensor signals have been detected and the robot has been stabilized.

Palabras clave: Control, péndulo invertido, LQR, PID, Sintonía experta, Lego, EV3

Keywords: Control, inverted pendulum, LQR, PID, expert tuning, Lego, EV3

1 Introducción

Los robots han pasado, con rapidez, a formar parte de nuestro día a día. Saber cómo modelar un sistema, que esté representado de manera fidedigna, para poder probar su comportamiento en entornos simulados, virtuales, se ha convertido en una parte muy importante en el diseño de robots eficientes. Sin este conocimiento, el diseño de estrategias de control y la sintonía de los parámetros se vuelve demasiado compleja, y las pruebas en entornos reales suponen demasiados riesgos.

Pero la implementación en un sistema real presenta una serie de retos con lo que hay que enfrentarse, y que permiten probar lo que se ha diseñado y comprobar su eficiencia.

En este trabajo se ha construir un robot en configuración de péndulo invertido con un *LEGO EV3* como sistema real. Se ha desarrollado un modelo que se ha analizado y simulador para diseñar algoritmos de control para su estabilidad (LQR, PID). Además se ha utilizado una técnica de la inteligencia artificial para sintonizar los parámetros del regulador PID.

Se ha hecho uso de las herramientas *Matlab*, *Simulink* y del sistema operativo *lejOS.*

1.1 Objetivos

Este proyecto tiene como objetivo general el control de un robot en configuración de péndulo invertido que sea capaz de mantener el equilibrio de manera autónoma. Para ello se deberán diseñar estrategias de control que permitan el equilibrio del robot. Esto servirá para hacerse una idea general de cómo funcionan las técnicas de control en un sistema real.

Como primer objetivo específico tenemos el estudio de los antecedentes, donde se tendrá que buscar información y artículos que sirvan para desarrollar el tema planteado. El siguiente objetivo será el modelado y la simulación del sistema basados en los datos recogidos en la etapa anterior. El tercero será construir el robot dónde probar el modelo obtenido, para el cual se utilizará el bloque programable *LEGO EV3.* Los últimos objetivos son la aplicación de las técnicas de control estudiadas sobre un entorno físico para comprobar su corrección y dotar al robot de la capacidad de detectar obstáculos y evitarlos.

1.2 Estructura

La memoria se ha estructurado de la siguiente manera: después de esta introducción, en el capítulo 2 se hará un estudio de los antecedentes y de los fundamentos teóricos, que servirán como base para el resto del trabajo.

En el capítulo 3, aplicaremos los fundamentos teóricos para hallar unas ecuaciones que representen el sistema real y que permitan probar estrategias para la estabilización de un péndulo invertido.

Estas se utilizarán en el capítulo 4, donde se comprobarán los resultados a través de simulaciones con Matlab/*Simulink*.

En el capítulo 5 se explican brevemente las técnicas utilizadas para conseguir el control inteligente, tanto la sintonía experta del PID, como la detección de obstáculos.

En el capítulo 6 se explicará la implementación en Java de todo lo desarrollado anteriormente, consiguiendo estabilizar el robot lego real en configuración de péndulo invertido.

Al final, se expondrán las conclusiones del desarrollo de este proyecto y se sugerirán posibles mejoras y/o extensiones.

1.3 Asignaturas relacionadas

En este proyecto se han usado los conocimientos adquiridos en diversas asignaturas cursadas a lo largo de la carrera.

Para el diseño del modelo y la construcción del robot se ha aplicado lo aprendido en Robótica e Inteligencia Artificial Aplicada al Control. Asignaturas como Técnicas de Programación y Fundamentos de la programación han proporcionado una base sólida para generar un código eficiente para el robot. Se han podido aplicar también conocimientos de Administración de Sistemas y Redes, puesto que el robot se basa en un sistema Linux.

2 Estado del arte y fundamentos teóricos

2.1 Antecedentes

El control de péndulo invertido constituye un problema clásico dentro del campo del control no lineal y este tiene aplicaciones en la vida cotidiana, tales como grúas y segways.
En los años 70 se empezaron a realizar proyectos que implicaban péndulos invertidos, y lo largo de los años, se han ido viendo diferentes variantes del problema. El pionero y referencia en este campo, Furuta K. Yamakita ha brindado enormes aportes al estudio de este problema, dándole nombre a un modelo de péndulo, el péndulo de Furuta. [7]
En 2002, el Laboratorio de Electrónica Industrial del Instituto Federal Suizo de Tecnología de Lausana, Suiza, construyó un prototipo de un vehículo de dos ruedas en configuración de péndulo invertido. El sistema de control estaba compuesto por dos controladores estatales desacoplados.[1]
En 2003 se escribe un documento que trata sobre el modelado de dos ruedas del péndulo invertido y el diseño del Control de modo de deslizamiento integral proporcional (PISMC) para el sistema. El modelo final se representa en el espacio de estados. propone un controlador robusto basado en el control de modo deslizante para realizar la estabilización robusta y el rechazo de perturbaciones del sistema.[2]
En 2004 se crea la plataforma de movilidad robótica Segway (RMP) basada en el Segway Human Transporter (SHT). El Segway RMP es más rápido, barato y ágil que las plataformas comparables existentes en ese momento, es resistente, tiene una huella pequeña, un radio de giro cero y, sin embargo, puede llevar una carga útil mayor. Esta nueva geometría de la plataforma ofreció a los investigadores la oportunidad de examinar temas novedosos, incluidas las modalidades de detección y actuación en altura.[3]
Steven Hassenplug construyó con éxito un robot de equilibrio llamado *Legway* utilizando el kit de robótica LEGO Mindstorms. Se utilizan dos sensores del detector de proximidad electro-óptico (EOPD) para proporcionar el ángulo de inclinación del robot al controlador que programó en BrickOS, un lenguaje de programación similar a C / C ++ específicamente para LEGO Mindstorms [20]
En 2007 se diseñó un sistema de control para un robot de péndulo invertido sobre dos ruedas basado en Lego Mindstorm. Se utiliza un control de H 2 y en las simulaciones se compara con el rendimiento del LQR. [6]

Se hacen experimentos parecidos en la Escuela Superior de Ingenieros de Sevilla, utilizando componentes comerciales y de bajo coste. Para el control del vehículo se han implementado dos leyes de control (lineal y no lineal), habiéndose probado con éxito en diferentes experimentos.[4]
EN 2012 se hace un trabajo para verificar y validar el buen comportamiento del control CACM-RL en este tipo de sistema. Aprender mientras se mantiene el equilibrio es una tarea compleja por lo que es fácil en plataformas estables, pero en sistemas inestables es muy difícil. [13]
En 2018 se hace una implementación de un control no lineal Fuzzy Takagi-Sugeno con funciones de pertenencia superpuestas aplicadas a un mecanismo de péndulo invertido móvil y su comparación con LQR en términos de comportamiento y robustez del espacio de estado.[14]
Más actualmente, en 2019, se siguen haciendo estudios sobre distintas técnicas de control aplicadas sobre este problema, como en *Two-wheel Balancing Robot; Review on Control Methods and Experiments* [15]. Es un tema, que como se puede ver, tiene bastante interés a día de hoy, y más con los nuevos algoritmos de inteligencia artificial, aquellos como las redes neuronales y la lógica borrosa. Con esto, se parte de una base sólida y se pretenden adquirir los conocimientos básicos del control aplicados a la Ingeniería del Software.

2.2 Control

El control se encuentra presente en todas partes, pero ¿qué significa control? El control que vamos a tratar es una disciplina de la ingeniería, en que la que se estudian las arquitecturas, los mecanismos y los algoritmos que lo hacen posible. Un punto muy importante es la automatización de procesos, que hará que un sistema no tenga que estar manejado por un operario, aunque como veremos posteriormente, los conocimientos de este serán ampliamente valorados.
El control de procesos se divide en el estudio de las plantas a controlar y en las técnicas empleadas. A su vez, su objetivo será mantener la salida de un sistema dentro un de un rango previamente fijado.
En este trabajo, la planta a controlar consistirá en un robot con configuración de péndulo invertido. El péndulo invertido es un péndulo que tiene el centro de masa encima de su punto pivotal, es inestable, y sin ayuda, se derrumba. Para estabilizar el sistema, se debe observar el ángulo del robot, y calcular la señal pertinente para recuperar el ángulo acotado en la

referencia. Es un problema clásico de la dinámica y de la teoría del control, y es ampliamente usado para testear algoritmos de control (PID, redes neuronales, etc.), por lo que servirá perfectamente para demostrar lo aprendido en Inteligencia Artificial Aplicada al Control.
Se irán aplicando diversos mecanismos de control, y se observarán los resultados para comprobar su corrección y elegir el óptimo. Para ello, se probará el control LQR, el control PID y se simulará un "sistema experto" para sintonizar de manera inteligente las ganancias del PID; todo esto, será explicado a continuación.
La realimentación es un problema crítico a la hora de diseñar un mecanismo de control eficiente. Normalmente, el objetivo del sistema es alcanzar cierta estabilidad respecto a un punto de referencia, pero sin una retroalimentación de la salida del sistema, la tarea de darle estabilidad al sistema es mucho más difícil de conseguir, y mucho menos de corregir *in-situ*.
Nuestro sistema deberá deducir los valores a partir de un determinado estado y para ello, se deberá implementar un sistema que pueda valorar la diferencia entre la situación actual y la situación de referencia deseada.
En este caso, se hace uso de una retroalimentación negativa, pues lo que se quiere es saber en cada momento cómo de alejado se está de la referencia.

2.2.1 Control Inteligente

El control convencional está típicamente basado en modelos matemáticos donde la estructura de entrada-salida está fijada. Los objetivos suelen estar especificados de manera no cualitativa, siendo una especificación pobre cuando se aplica a problemas de la vida real.
Por otro lado, el control inteligente se inicia con la intención de aplicar técnicas de inteligencia artificial al control. Igual que los seres humanos manejan información imprecisa, difusa e incompleta, el control inteligente estará enfocado en resolver problemas con este tipo de información, algo impensable para el control convencional. Por ello, el controlador inteligente es un controlador heurístico, no lineal y adaptativo.

2.2.2 Regulador LQR

El regulador cuadrático lineal (LQR) es una técnica de control óptimo, que tiene en cuenta los estados del sistema dinámico y la entrada de control para tomar las decisiones de control óptimas. Esto es simple y robusto.[6]

Un LQR minimiza la suma de los esfuerzos de control. Esto hace que estos esfuerzos se optimicen, y que las desviaciones respecto al valor deseado sean menores. Para ello, se asigna unos determinados pesos a las variables que realimentan el sistema esperando que mejoren la estabilidad. El objetivo entonces es calcular las ganancias óptimas para las variables de estado utilizadas en el proceso de retroalimentación y la función de coste cuadrático a minimizar es la siguiente:

$$J(x,u) = \int_0^\infty \left(x^T(t) \cdot Qx(t) + R \cdot u^2(t) \right) dt \quad (1)$$

Usaremos dos matrices, matriz de coste(Q) y matriz de rendimiento(R). Normalmente la matriz de coste es una matriz identidad con algunos valores sustituidos por los pesos. Se realizarán pruebas de control para determinar qué vector K proporciona el control más eficiente. La matriz de rendimiento permanecerá como matriz unitaria.

Dada la ley de control, se sabe que [10]:

$$u = -Kx \quad (2)$$

Si se utiliza la ecuación de entrada del espacio de estados y se combina con la anterior, se obtiene:

$$x' = (A - B \cdot K)\, x \quad (3)$$

K será un vector, que contendrá las ganancias correspondientes a las variables del sistema y que pueden ser calculadas del siguiente modo:

$$K = R^{-1} B^T P \quad (4)$$

Dónde P puede ser despejada a través de la siguiente fórmula algebraica de Riccati:

$$A^T P + PA - PBR^{-1} B^T P + Q = 0 \quad (5)$$

2.2.3 Control PID

Un controlador PID (Proporcional, integral, derivativo) es un mecanismo de control con retroalimentación, habitualmente usado en sistemas de control industrial. Debido a su intuitividad y su relativa simplicidad, además del rendimiento satisfactorio que puede proporcionar con una amplia gama de procesos, se ha convertido en la práctica en el controlador estándar en entornos industriales.[5]

Un PID es un mecanismo de control que corrige la diferencia entre el punto de referencia y la salida de un sistema. Su señal de control es calculada de la siguiente manera:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt} \quad (6)$$

Este calcula constantemente el error *e(t),* como la diferencia hasta el punto deseado y una medida de proceso variable, y aplica una corrección basada en términos proporcionales, integrales y derivativos.

- *Proporcional:* Consiste en el producto entre la señal de error y la constante proporcional para lograr que el error en estado estacionario se aproxime a cero. Produce una señal de control proporcional a la señal de error.
- *Integral:* Tiene como propósito disminuir y eliminar el error estacionario. El error es integrado, lo cual tiene la función de promediarlo o sumarlo por un período determinado. Proporciona una corrección para compensar las perturbaciones y mantener en el punto de consigna la variable a controlar.
- *Derivativa:* La función de la acción derivativa es mantener el error al mínimo, corrigiéndose proporcionalmente a la misma velocidad que se produce. Anticipa el efecto de la acción proporcional para estabilizar más rápidamente la variable controlada después de una perturbación.

Estas señales son sumadas y serán la entrada al sistema, como se muestra en la Figura 1.

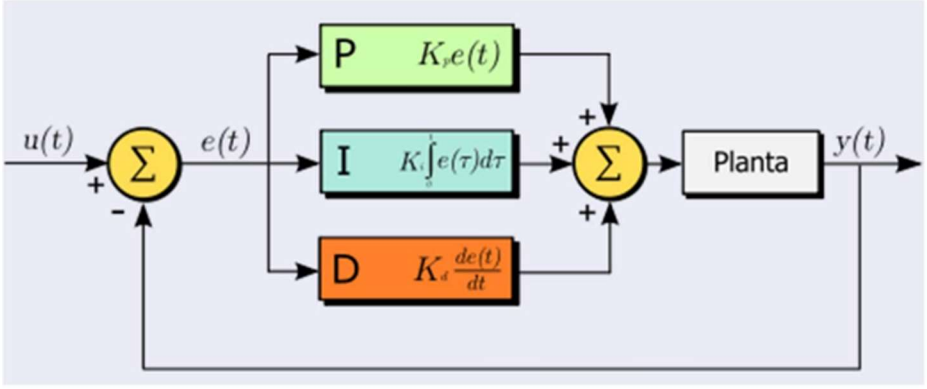


Figura 1. Modelo de ganancias PID.

El vector de parámetros de sintonía (*Kp,Ki,Kd*) representa las ganancias proporcional, integral, y derivativa. Se obtienen para calcular una señal de control adecuada. Según se indica en [17]:

- **Constante proporcional (Kp):** Constante de proporcionalidad en la acción de control proporcional. Si Kp pequeña, la acción proporcional será pequeña, y si Kp es grande, la acción proporcional también lo será.

- **Constante de tiempo integral (Ti):** El tiempo requerido para que la acción integral contribuya a la salida del controlador en igual medida que la acción proporcional. Si Ti es pequeño, la acción integral es grande, y si Ti es grande, la acción integral será pequeña.
- **Constante de tiempo derivativa (Td):** El tiempo requerido para que la acción proporcional contribuya a la salida del controlador en igual medida que la acción derivativa. Si Td es pequeño, la acción derivativa será pequeña, y si Td es grande, la acción derivativa también lo será.

Este método no garantiza la estabilidad del sistema, ya que depende del error medido, por ello, el control global está formado por varios controladores de distinto tipo trabajando simultáneamente.

2.2.4 Control Experto

En este trabajo se tratará de dar un nuevo enfoque al problema por lo que se hará uso de un sistema experto. Este será un algoritmo o programa que tendrá el conocimiento de un experto para resolver una tarea de cierto dominio, por lo que la eficiencia de este sistema será equiparable a la de un experto humano. Se da este enfoque, porque a pesar de que la generación de un sistema experto sea generalmente cara, el mantenimiento y el coste marginal de su uso repetido es relativamente barato.[18]

Sus características principales son el conocimiento, el dominio, el razonamiento y la estructura. La estructura se basa en tres partes: una base de hechos, un motor de inferencia y una base de conocimiento. La base de conocimiento es proporcionada por el experto, y representa en forma de reglas los conocimientos específicos sobre una determinada tarea. La base de hechos es proporcionada por el usuario y es información concreta del problema. El motor de inferencia será el encargado de aplicar el conocimiento generando nuevos hechos obtenidos al combinar las reglas previamente proporcionadas.

La representación del conocimiento entonces está representada en forma de reglas, que tendrán la siguiente forma:

$$si\ \left(condicion_0 \& \ldots \& condicion_i \right) \quad (7)$$

$$entonces\ (\ conclusiones) \quad (8)$$

Esto nos presenta la posibilidad para introducir cierta corrección dinámica e inteligente en el sistema. Se hará uso de ciertas reglas, a partir de conocimientos y estudios anteriores para sintonizar dinámicamente las ganancias de nuestro controlador PID.
Para programar las ganancias, se usará el ángulo y la velocidad del cuerpo como referencia, que hará que separemos nuestra zona de trabajo en dos: Estable e Inestable:

- En la zona Estable, la situación del robot no supone apenas alteraciones en su estabilidad.
- En la zona Inestable, la situación del robot es bastante comprometida, y por lo tanto necesitará de correcciones más potentes y rápidas.

La tarea más importante y compleja de los sistemas expertos es la extracción de conocimientos de los expertos y su formalización simbólica. En este caso, utilizaremos las simulaciones para generar la fuente de conocimiento, utilizando la teoría para elegir distintas ganancias que consideremos oportunas para probar. Se estructurará el conocimiento en reglas, que serán utilizadas para alcanzar la correcta sintonía de los parámetros del PID. Se trabaja en base a hipótesis, y siempre sería posible mejorar las reglas o la combinación de éstas.

3 Modelo matemático

Gran parte de la carrera de ingeniería del software consiste en hacer un modelo abstracto de la realidad para plasmarlo en código; por ello, uno de los pasos más importantes es encontrar un modelo factible del sistema que se desea controlar. La corrección de este primer modelo se irá arrastrando en el resto del proceso, y si se ha planificado torpemente tendrá consecuencias catastróficas al final del proyecto.

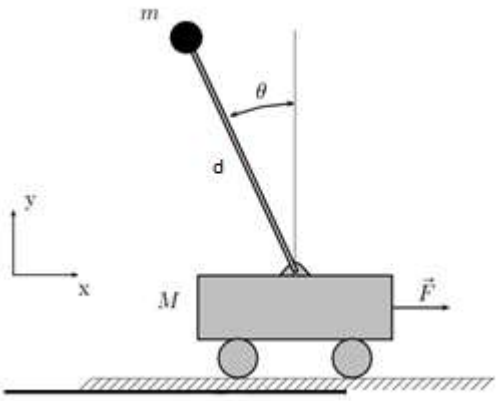
Para ello haremos uso de *Matlab* y *Simulink*. Utilizaremos *Matlab* para hacer la planta, y *Simulink* será usado para generar y simular los modelos. Lo importante del problema es analizar correctamente las salidas proporcionadas por el sistema, con determinadas entradas, para poder analizar el error y corregirlo en consecuencia.

Es importante que, aunque el modelo tenga que ser fidedigno a la realidad, hay que sopesar si la complejidad que introduce representa una mejora notable en el sistema. Esto hace que se tengan que pasar por alto variables como el nivel de batería del robot, que hace que la potencia suministrada a los motores varíe ligeramente.

El péndulo invertido es uno de los problemas más utilizados e importantes en la teoría de control, donde el sistema está compuesto por dos elementos: un carro y un péndulo de rotación libre sobre éste. El carro debe moverse por un espacio mientras compensa el desplazamiento del péndulo. Para ello, se requiere el control de un sistema **no-lineal** e **inestable** y esto es útil al permitir estudiar las diferencias entre el control de bucles abiertos y cerrados.

3.1 Obtención del modelo

Normalmente, el problema del péndulo invertido consiste en un carro móvil sobre el que reposa un sólido que debe mantenerse en equilibrio, como se observa en la Figura 2. El caso tratado es un poco distinto y se debe adaptar el problema, pensando que las ruedas son el carro y el péndulo el cuerpo, como se puede ver en la Figura 3. Con esto se podrá fijarse en las fórmulas usadas en el caso original y adaptarlas.



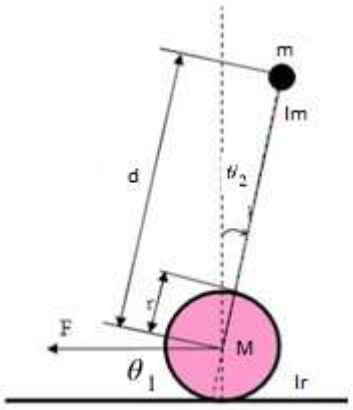


Figura 2. Configuración de péndulo invertido tradicional. *Figura 3. Configuración de péndulo invertido adaptada.*

La dificultad de considerar un problema no lineal lleva a buscar una linealización de este. Esto es posible siempre que se linealice en un rango en el cual ambos sistemas tienen respuestas muy parecidas. En este caso, el punto de consigna, es decir, el punto donde el ángulo del robot es cero con respecto a la vertical, representa este rango. Esto simplifica enormemente la resolución del problema.

Se puede realizar un modelo teórico completo del sistema de carro de péndulo utilizando la dinámica de Lagrange. Primero se eligen las coordenadas generalizadas para después derivar expresiones para las fuerzas generalizadas, las funciones de energía y el Lagrangiano. Finalmente, podemos usar la ecuación de Lagrange para derivar las ecuaciones de movimiento [11]. En base a esto, se pasa a definir los parámetros que componen el sistema.

Parámetros del sistema

Constante	Descripción	Valor
I_{motor}	Momento de inercia del motor	$0.000842\ \ kg \cdot m^2$
I_{ruedas}	Momento de inercia de las ruedas	$0.00000625\ kg \cdot m^2$
I_{cuerpo}	Momento de inercia del cuerpo	$0.09\ kg \cdot m^2$
$radio$	Radio de las ruedas	$0.027\ m$
M	Masa de la rueda	$0.029\ kg$
m	Masa del cuerpo	$0.7\ kg$

d	Distancia del eje al centro de gravedad	$0.15\ m$
b	Fricción del motor	$0.00129\ N \cdot m \cdot s$
g	Aceleración de la gravedad	$9.8\ m/s^2$

Variable	Descripción	Unidad
θ_1	Ángulo de la rueda	rad
θ_1'	Velocidad angular de la rueda	$rad\ /\ s$
θ_1''	Aceleración de la rueda	$rad\ /\ s^2$
θ_2	Ángulo del cuerpo	rad
θ_2'	Velocidad angular del cuerpo	$rad\ /\ s$
θ_2''	Aceleración del cuerpo	$rad\ /\ s^2$

3.2 Modelo matemático

El sistema a diseñar está basado en 6 variables: los ángulos de la rueda y el cuerpo, sus velocidades angulares y sus aceleraciones.

El ángulo de la rueda (θ_1) podrá ser obtenido directamente de los *encoders* de los motores, dónde derivando, se obtiene la velocidad angular (θ_1').

El giroscopio será el encargado de obtener la velocidad angular (θ_2'), que siendo integrada proporcionará el ángulo del cuerpo (θ_2).

La señal de control u, representa la potencia aplicada a los motores, que será proporcionada para llevar al robot a un estado de equilibrio.

Partiendo de las ecuaciones basadas en distintos artículos [21][8][16][19] se puede observar la relación de la velocidad (primera derivada) y de la aceleración (segunda derivada) de las ruedas respecto a los parámetros del robot:

$$X_1' = radio \cdot \theta_1' \quad (9)$$

$$m \cdot X_1'' = F_x - F_y \quad (10)$$

$$I_{ruedas} \cdot \theta_1'' = radio \cdot F_x \quad (11)$$

La primera ecuación (9) relaciona el ángulo de las ruedas con su posición a través del radio.

La segunda (10) relaciona las fuerzas en los ejes X e Y con la aceleración de las ruedas y su masa.

La tercera ecuación (11) relaciona el momento de inercia de las ruedas y la aceleración del cuerpo con el radio y su fricción en el eje de las X.

$$M \cdot X_2'' = F_y \quad (12)$$

$$M \cdot Z_2'' = F_z - M \cdot g \quad (13)$$

$$I_{cuerpo} \cdot \theta_1'' = d \cdot F_z \cdot \sin(\theta_2) - d \cdot F_y \cdot \cos(\theta_2) + u \quad (14)$$

Estas ecuaciones hacen referencia al cuerpo del robot.

En la primera ecuación (12) se relaciona la masa del cuerpo y su aceleración en el eje X con la fricción en el eje Y.

En la segunda ecuación (13) se relaciona la masa del cuerpo y su aceleración en el eje Z con la diferencia entre la fricción con el eje Z y la masa del cuerpo por la gravedad.

En la tercera ecuación (14), se observa cómo el momento de inercia del cuerpo multiplicada por su aceleración, quedan relacionadas con la distancia al punto de equilibrio, el ángulo del cuerpo, las fricciones en los respectivos ejes, y la señal de control u.

Con esto, la velocidad y las aceleraciones en los ejes x e y están relacionadas.

La relación entre las ruedas y el cuerpo es representada por estas ecuaciones (15)(16):

$$X_2 = X_1 + d \cdot \sin(\theta_2) \quad (15)$$

$$Z_2 = d \cdot \cos(\theta_2) \quad (16)$$

Se deben utilizar ciertas aproximaciones (17-19) para linealizar el sistema, suponiendo que los ángulos son muy pequeños:

$$\theta_2^2 \approx 0 \quad (17)$$

$$\cos(\theta_2) \approx 1 \quad (18)$$

$$\sin(\theta_2) \approx \theta_2 \quad (19)$$

Tomando estas simplificaciones (20-25) en base a las ecuaciones anteriores (9-19):

$$\varepsilon_1 = I_{ruedas} + I_{motor} \cdot radio + (m \cdot M) \cdot radio^2 \quad (20)$$

$$\varepsilon_2 = M \cdot d \cdot radio^2 \quad (21)$$

$$\varepsilon_3 = b \cdot radio^2 \quad (22)$$

$$\varepsilon_4 = I_{cuerpo} + M \cdot d \quad (23)$$

$$\varepsilon_5 = -M \cdot g \cdot d \quad (24)$$

$$den = \left(\varepsilon_4 \cdot \varepsilon_1 \right) - \varepsilon_2{}^2 \tag{25}$$

Se derivan las siguientes ecuaciones (26)(27):

$$\varepsilon_1 \cdot \theta_1'' + \varepsilon_2 \cdot \theta_2'' + \varepsilon_3 \cdot \theta_1' = radio \cdot u \tag{26}$$

$$\varepsilon_2 \cdot \theta_1'' + \varepsilon_4 \cdot \theta_2'' + \varepsilon_5 \cdot \theta_2' = u \tag{27}$$

Con esto se da por terminada la modelización del sistema a través de ecuaciones matemáticas.

3.3 Modelo en el Espacio de Estados

Habiendo obtenido las ecuaciones matemáticas que definen el sistema, se deben transformar al espacio de estados. Debido a la naturaleza continua y dinámica del problema del péndulo invertido, el esquema matemático utilizado para modelarlo también será dinámico, es decir, los cambios que se van a producir en el sistema no son lineales; La respuesta que haya que aplicar varía enormemente en cada caso. Podemos diferenciar entre dos estados básicos en los que se encuentra el robot: en uno se intenta mantener erguido desde un equilibrio, y en otro el robot trata de recuperar el equilibrio después de una perturbación externa (un escalón, por ejemplo). Hay que tener en cuenta, que no sólo es importante el ángulo de inclinación del robot, también se debe tener presente la velocidad anterior de las ruedas, la velocidad angular de la caída, etc. Estos estados sirven para representar las condiciones actuales del robot, y vienen dados por cuatro parámetros variables:

Ángulo de las ruedas	θ_1
Velocidad angular de las ruedas	θ_1'
Ángulo del cuerpo	θ_2
Velocidad angular cuerpo	θ_2'

Se hace imperativo formular las salidas del sistema en función del valor de las entradas y del estado (28)(29).

$$\theta'(t) = f(x(t), u(t)) \tag{28}$$

$$y(t) = h(x(t), u(t)) \tag{29}$$

El espacio de estados tiene ecuaciones matriciales y las variables son vectores. Por esto, la ecuación quedaría de la siguiente forma (30)(31):

$$x' = Ax + Bu \tag{30}$$

$$Y = Cx + Du \tag{31}$$

La primera ecuación calcula el siguiente estado a partir de la señal de control y el estado actual, mientras que la segunda ecuación representa la salida, donde se podrán escoger las variables de estado que sean oportunas.

A partir de las variables de estado $x = \left(x_1', x_2', x_3', x_4' \right) = \left(\theta_1,\ \theta_1',\ \theta_2,\ \theta_2' \right)$ podemos obtener las ecuaciones en el espacio de estados (32):

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \end{bmatrix} = \begin{bmatrix} \theta_2' \\ \theta_2'' \\ \theta_1' \\ \theta_2'' \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ a_1 \cdot x_1 + a_2 \cdot x_4 + b_1 \cdot u \\ x_4 \\ a_3 \cdot x_1 + a_4 \cdot x_4 + b_2 \cdot u \end{bmatrix} \tag{32}$$

Quedando las matrices A, B, C y D de la siguiente manera (33)(34):

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ a_1 & 0 & 0 & a_2 \\ 0 & 0 & 0 & 1 \\ a_3 & 0 & 0 & a_4 \end{bmatrix} \quad B = \begin{bmatrix} 0 & b_1 & 0 & b_2 \end{bmatrix} \tag{33}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{34}$$

dónde:

$$a_1 = \frac{\left(-\varepsilon_{1.} \varepsilon_5 \right)}{den} \tag{35}$$

$$a_2 = \frac{\left(\varepsilon_2 \cdot \varepsilon_3 \right)}{den} \tag{36}$$

$$a_3 = \frac{\left(\varepsilon_2 \cdot \varepsilon_5 \right)}{den} \tag{37}$$

$$a_4 = \frac{\left(-\varepsilon_3 \cdot \varepsilon_4 \right)}{den} \tag{38}$$

$$b_1 = \frac{\left(\varepsilon_1 - \left(\varepsilon_2 \cdot radio\right)\right)}{den} \tag{39}$$

$$b_2 = \frac{\left(-\varepsilon_2 - \left(\varepsilon_4 \cdot radio\right)\right)}{den} \tag{40}$$

Una vez hecho esto, hemos llegado a nuestro espacio de estados (41)(42), en el que las salidas son calculadas a través de una única entrada(u).

$$x' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ a_1 & 0 & 0 & a_2 \\ 0 & 0 & 0 & 1 \\ a_3 & 0 & 0 & a_4 \end{bmatrix} \cdot x + \begin{bmatrix} 0 \\ b_1 \\ 0 \\ b_2 \end{bmatrix} \cdot u \tag{41}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot x + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \cdot u \tag{42}$$

Este modelo se va a validar en el siguiente capítulo para usarlo para diseñar estrategias de control que permitan estabilizar el robot lego.

4. Simulación

En este apartado se simularán y analizarán los resultados del sistema, explicando cómo se ha desarrollado cada hito hasta llegar a un control adecuado. Para la simulación de los controladores se hará uso de la herramienta *Simulink* de *Matlab,* la cual permite generar modelos de manera sencilla a través de bloques predefinidos.

Para ello, se escribe un script "*EV3Model.m*" que contiene las constantes y las variables del modelo, así como las fórmulas y la representación en el espacio de estados. Se hace el cálculo del control LQR con las funciones predeterminadas y se especifican los parámetros de PID para que puedan ser usados por los modelos "*.slx*". Por último, se programarán las reglas de sintonía experta para comprobar su corrección.

En las simulaciones podemos observar los ángulos y de las velocidades del cuerpo del robot, y estas estarán representadas en el eje Y como radianes y radianes por segundo respectivamente.

4.1 Lazo abierto

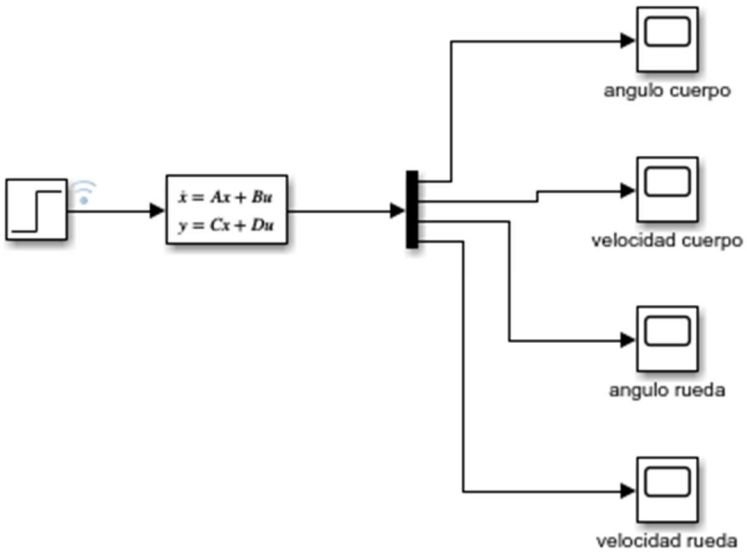


Figura 4. Modelo Simulink Sistema lazo abierto

El primer paso en el control de un sistema es comprobar cómo se comporta el sistema en lazo abierto y para ello se construye el esquema de la siguiente forma (Figura 4). Se utiliza el bloque de espacio de estados, al que le indicaremos los parámetros A, B, C y D mostrados anteriormente, y aplicando un *demultiplexor* a la salida, podremos obtener las 4 variables que componen el sistema. Al final, se aplicará una perturbación en forma de escalón como

entrada al sistema, y este, al tratarse de un sistema inestable, y no tener ningún control, hará que la salida crezca o decrezca exponencialmente, como podemos observar en la Figura 5:

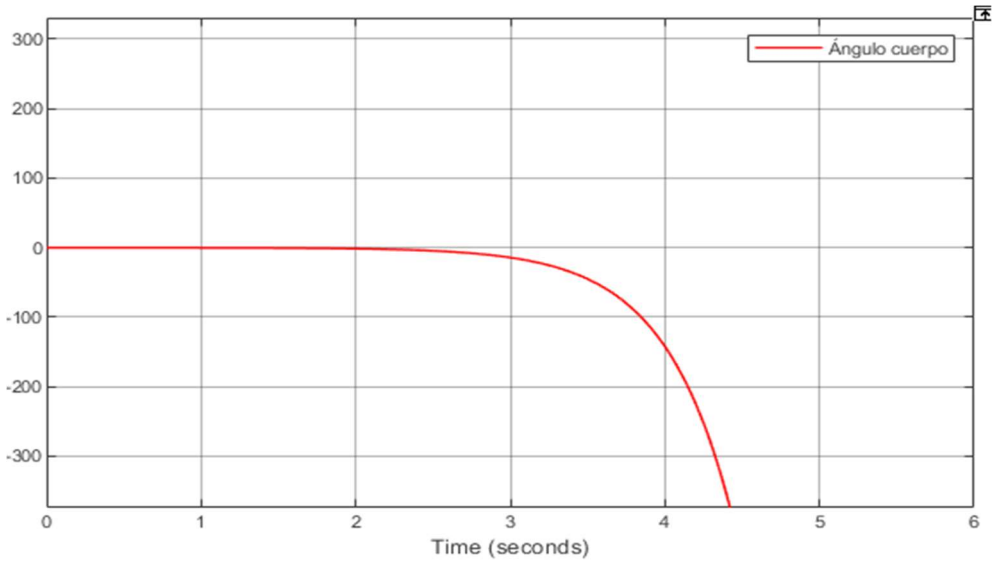


Figura 5. Ángulo del cuerpo en lazo abierto

Esto indica, que se debe aplicar un mecanismo de control para hacer que el sistema se vuelva estable, ya que el modelo actual no permite la estabilización del péndulo, causando el derrumbe del robot.

Fichero: *lazo_abierto.slx*

4.2 Lazo cerrado

El siguiente paso será comprobar cómo se comporta el sistema si se le aplica retroalimentación. Como ya se ha mencionado anteriormente, la retroalimentación es un mecanismo de control en sí mismo. Para añadirla al modelo anterior, se aplicará el escalón sólo a la salida del ángulo del cuerpo, y esta se sumará al resto de las señales. La suma total se restará a la posición de consigna (en este caso 0) para obtener la señal de control adecuada, que será aplicada para estabilizar el sistema. Con esto se da por construido el sistema en lazo cerrado como se observa en la Figura 6:

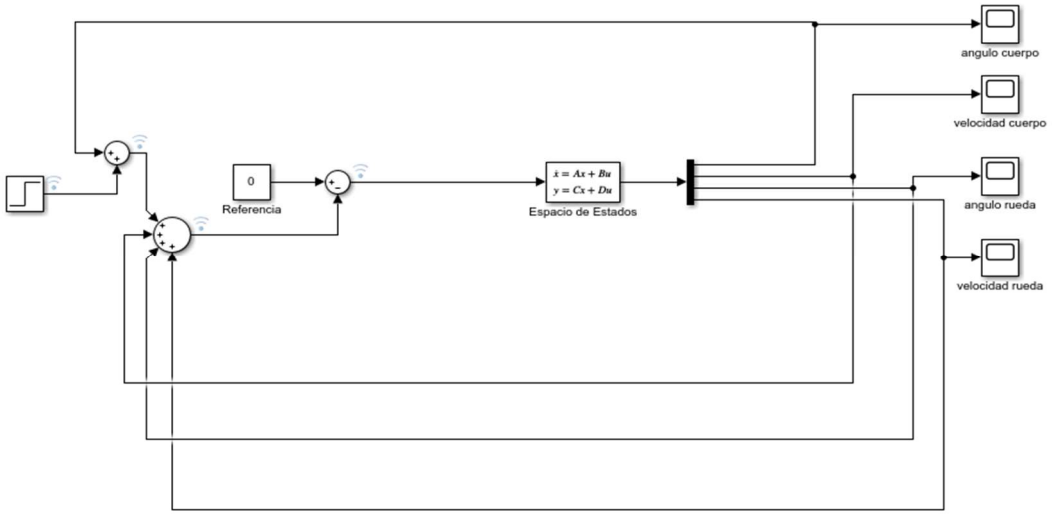


Figura 6. Modelo simulink Sistema lazo cerrado

Se aplica una perturbación (step) al ángulo del cuerpo, y se comprueba cómo responde el sistema. La respuesta del ángulo y de la velocidad se muestran respectivamente en la Figura 7 y en la Figura 8.

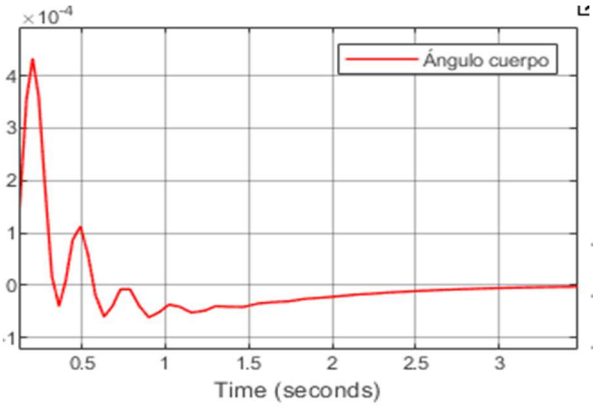


Figura 7. Velocidad cuerpo. rad/s

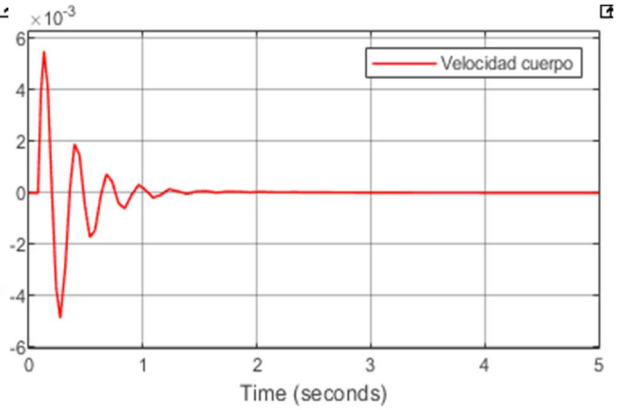


Figura 8. Ángulo Cuerpo en lazo cerrado. radianes.

Podemos observar cómo el sistema se estabiliza, pero no es lo suficientemente estable ya que se busca una respuesta más uniforme y con menos picos. Se tomará este control como ineficiente y se procede a aplicar otras técnicas. A pesar de que esta técnica resulta inválida, permite comprobar que aplicando más mecanismos de control, ya sea en paralelo o en serie, el sistema se podría estabilizar de la manera deseada.

Fichero: *lazo_cerrado.slx*

4.3 Aplicación de un control LQR

El primer método de control que se aplicará será un regulador cuadrático lineal. Se procede de la siguiente manera. Dadas las matrices de coste(Q) y matriz de rendimiento(R) (43) y utilizando la función “lqr” estándar de Matlab se obtiene el vector K (44):

$$Q = \begin{bmatrix} 300 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 75 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R = 1 \tag{43}$$

$$K = \begin{bmatrix} -0.00052046 & -2.2656 & 0.0007 & 0.0007 \end{bmatrix} \tag{44}$$

El modelo de *Simulink* resultante se muestra en la Figura 9.

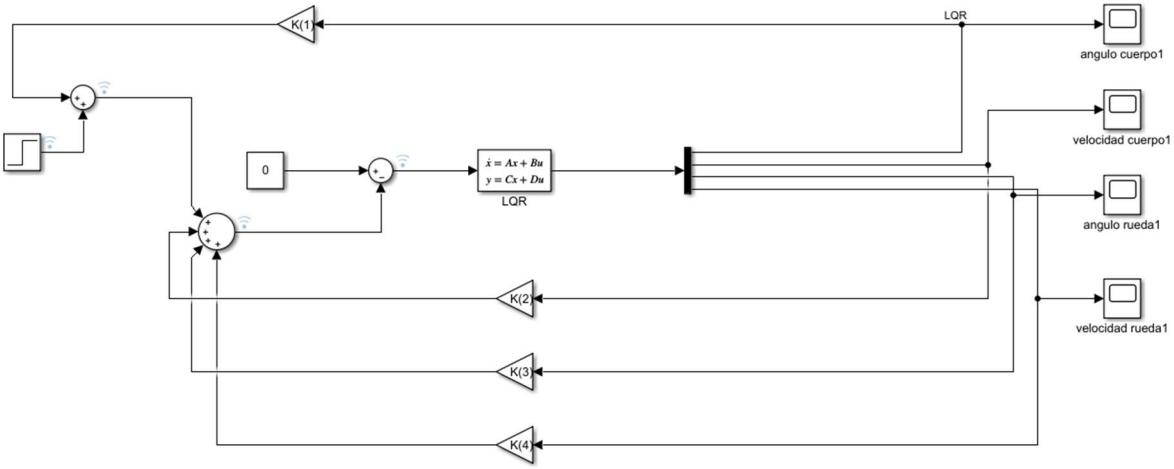


Figura 9. Modelo simulink Control LQR

Se han añadido los bloques de ganancia a las señales correspondientes, **K[1]** para el ángulo del cuerpo, **K[2]** para la velocidad angular del cuerpo, **K[3]** para el ángulo de las ruedas, **K[4]** para la velocidad angular de las ruedas. A su vez, se ha tenido que modificar el espacio de estado como se muestra en las ecuaciones (45) y (46):

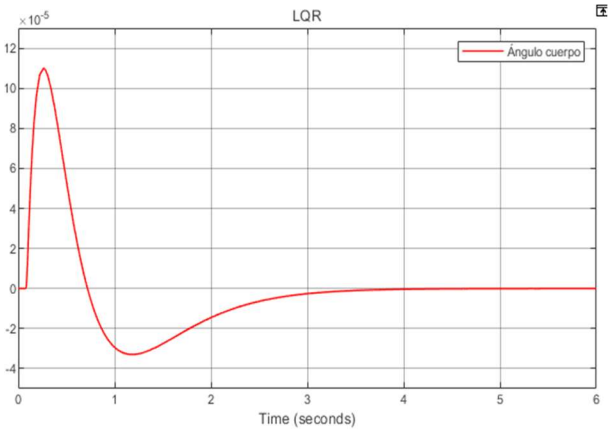
$$X = A_c x + Bu \tag{45}$$

$$Y = Cx + Du \tag{46}$$

donde Ac esta descrito a continuación (47):

$$A_c = A - B \cdot K \tag{47}$$

La gráfica resultante para el ángulo se muestra en la Figura 10 y para la velocidad en la Figura 11.



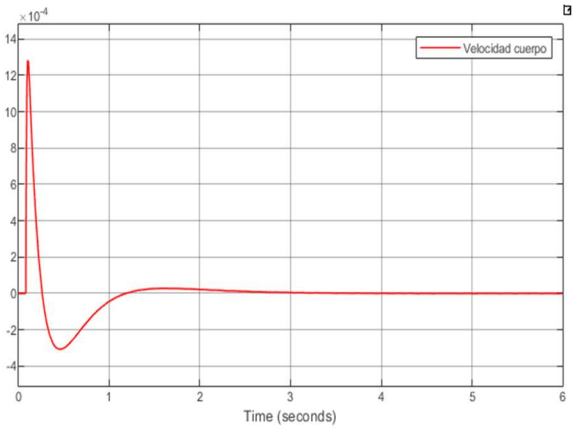


Figura 10.Velocidad cuerpo, control LQR

Figura 11. Ángulo cuerpo, control LQR.

.

Se puede ver que el sistema se estabiliza de una manera mucho más fluida, sin tantos picos consiguiendo el equilibrio total respecto al punto de consigna en 4 segundos. Este control ya permite el equilibrio del robot, pero en este trabajo se busca hacer un estudio de las distintas técnicas de control y tratar de sintonizar de manera inteligente los parámetros de un controlador PID, por lo que se procede a aplicar los siguientes pasos tomando estos resultados como referencia.

Fichero: *lqr.slx*

4.4 Aplicación del PID

En el siguiente paso se aplicará el PID junto con el control LQR, garantizando un control más fiable sobre el sistema. Para ello, se debe añadir el bloque PID al modelo de Simulink como se muestra en la Figura 12; o bien, implementarlo a partir del bloque integral y derivada como se muestra en la Figura 13:

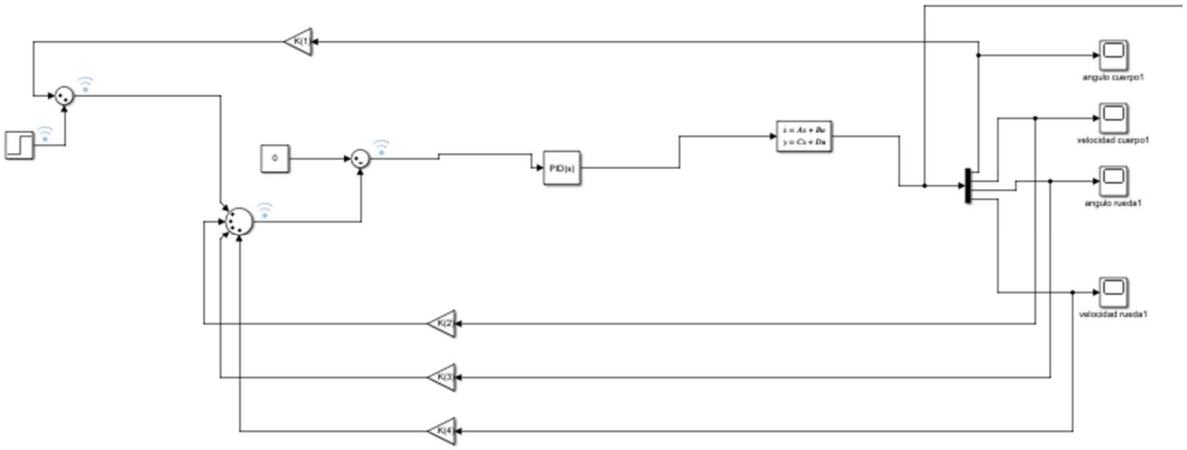


Figura 12. Modelo simulink PID. Bloque PID.

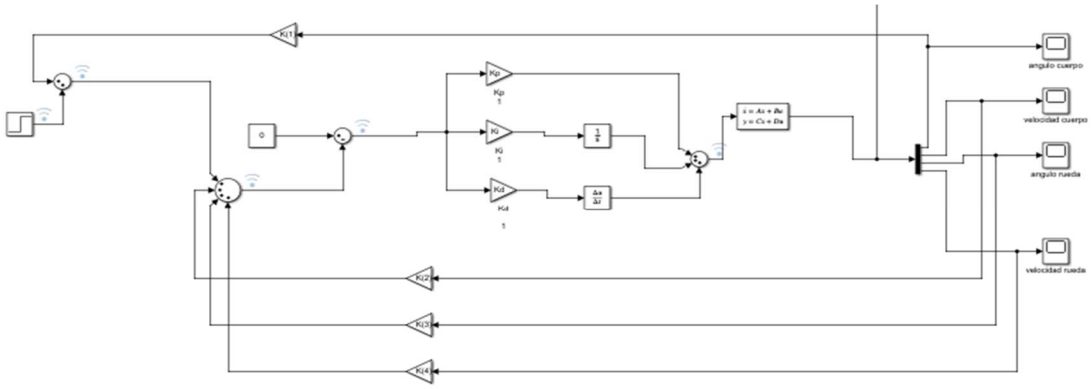


Figura 13. Modelo simulink PID. Bloques separados.

Este sistema no se puede sintonizar mediante métodos comúnmente utilizadas, como Ziegler Nichols, debido a su complejidad. Por ello usando ejemplos anteriores, trabajos parecidos, conocimiento adquirido en Inteligencia Artificial Aplicada al Control y a base de prueba e innumerables errores, se llegan a los siguientes parámetros aceptables PID:

Kp = 0.9

Ki = 0.04

Kd = 0.003

El resultado para el ángulo sería el mostrado en la Figura 14 y para la velocidad en la Figura 15.

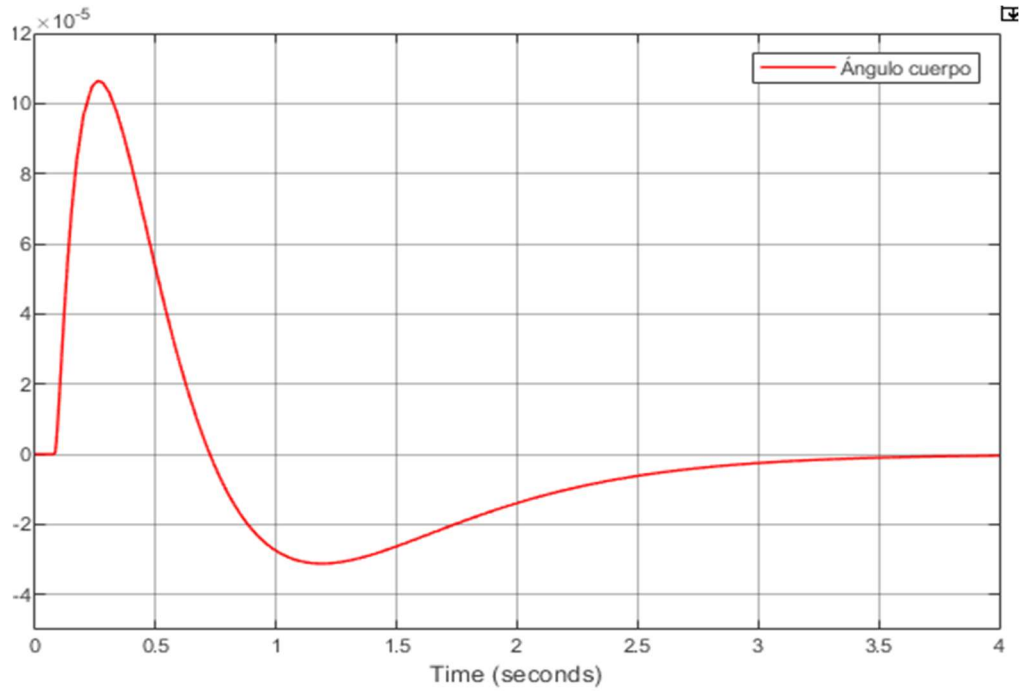


Figura 14. Angulo cuerpo control PID.

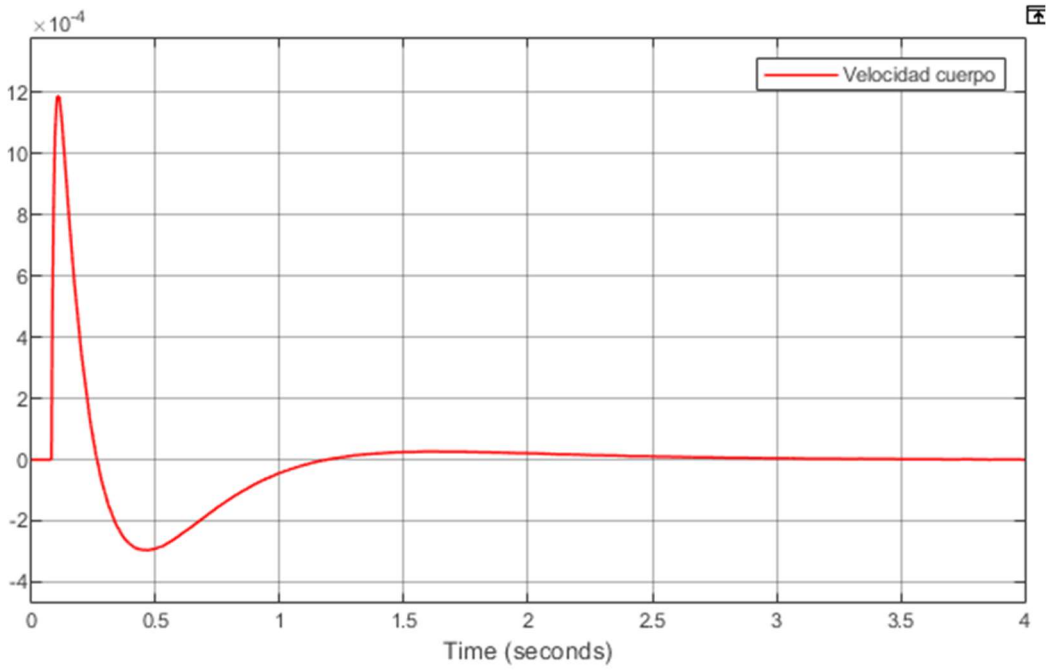


Figura 15. Velocidad cuerpo control PID.

Aparentemente, las respuestas son iguales que sólo aplicando LQR, pero superponiendo ambas gráficas se puede ver la diferencia entre el control LQR (rojo) y el LQR más el PID

(azul), aunque sea pequeña, tanto para el ángulo en la figura 16 como para la velocidad en la Figura 17:

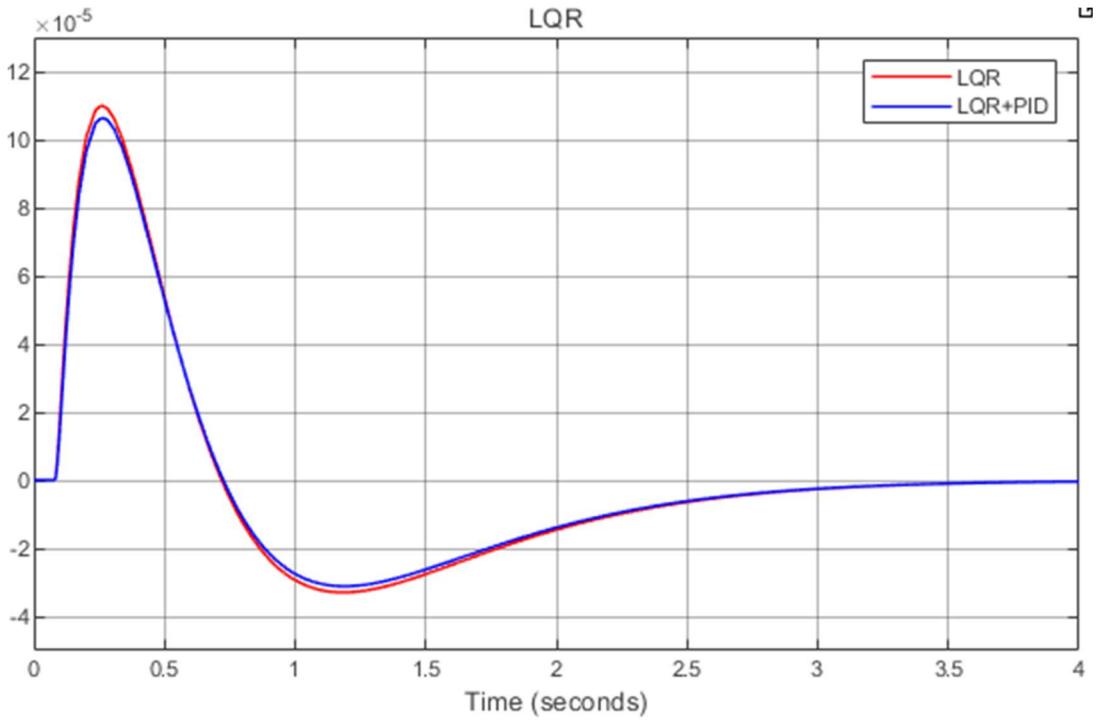


Figura 16. Comparación ángulos cuerpo con control LQR y PID.

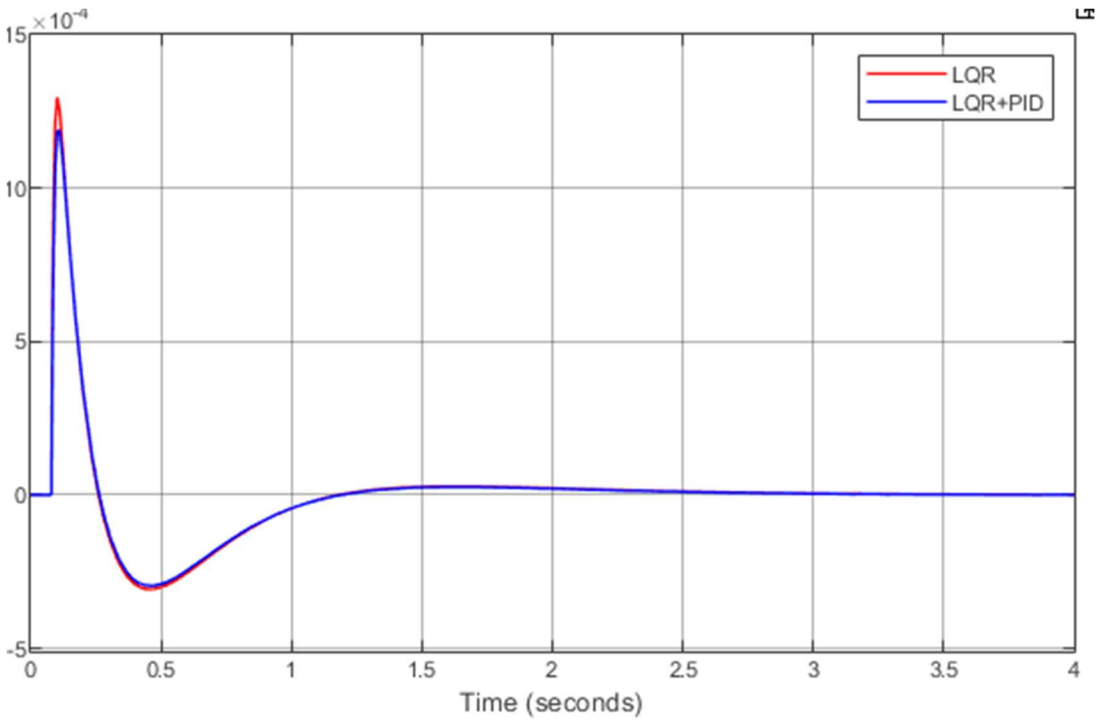


Figura 17. Comparación las velocidades del cuerpo con control LQR y PID.

Se observa una mejora tanto en el tiempo de respuesta como en la sobre elongación, siendo menor la velocidad adquirida por el cuerpo y por tanto menor esfuerzo en equilibrar el robot.

Habiendo obtenido resultados satisfactorios en las simulaciones, se daría por terminada la modelización del sistema, siendo capaz de estabilizarse el sistema en 3 segundos.
A partir de esta modelización satisfactoria, se empieza a desarrollar un sistema experto que permita la sintonía experta de los parámetros PID. Se tomarán de referencia los datos anteriores para después aplicar discriminación entre dos estados (ESTABLE e INESTABLE) que servirán de rangos para sintonizar de manera dinámica las ganancias de los parámetros PID. Esto será explicado con mayor detalle en el capítulo 5.
Fichero: *lqr_pid.slx.*

5. Aplicación de técnicas de Control Inteligente

5.1 Sintonía experta de los parámetros del controlador PID

Aplicaremos el control experto para la sintonía inteligente de los parámetros del PID. De esta forma, el sistema se comportará de una manera específica partiendo de un determinado estado, que será acotado en las pruebas. Esto permite un mayor control sobre el equilibrio del sistema, y la aplicación de técnicas inteligentes sobre un problema de control. Se pueden encontrar las relaciones del incremento o decremento de los parámetros del PID con las variables de la salida resultante, como se muestra en la siguiente tabla [9]:

Parámetro	Tiempo de subida	Sobreelongación	Tiempo de estacionamiento	Error en estado estacionario	Estabilidad
Kp	Disminuye	Aumenta	Cambio pequeño	Disminuye	Peor
Ki	Disminuye	Aumenta	Aumenta	Se elimina	Peor
Kd	Cambio pequeño	Disminuye	Disminuye	No efecto, teóricamente	Mejor si Kd pequeña

Se llevan a cabo pruebas modificando individualmente cada parámetro para observar cómo se comporta el sistema, esto lleva a generar una base de conocimiento que podrá ser aplicada.

5.1.1 Variación de variable proporcional (Kp)

Se estudian las variaciones en la salida al modificar la constante proporcional. Para ello, primero se intenta hallar un rango de valores aceptables para definir las correcciones, utilizando incrementos y decrementos sobre la variable proporcional original (0.93). Los resultados se pueden observar en la Figura 18, siendo el valor de referencia la línea roja.

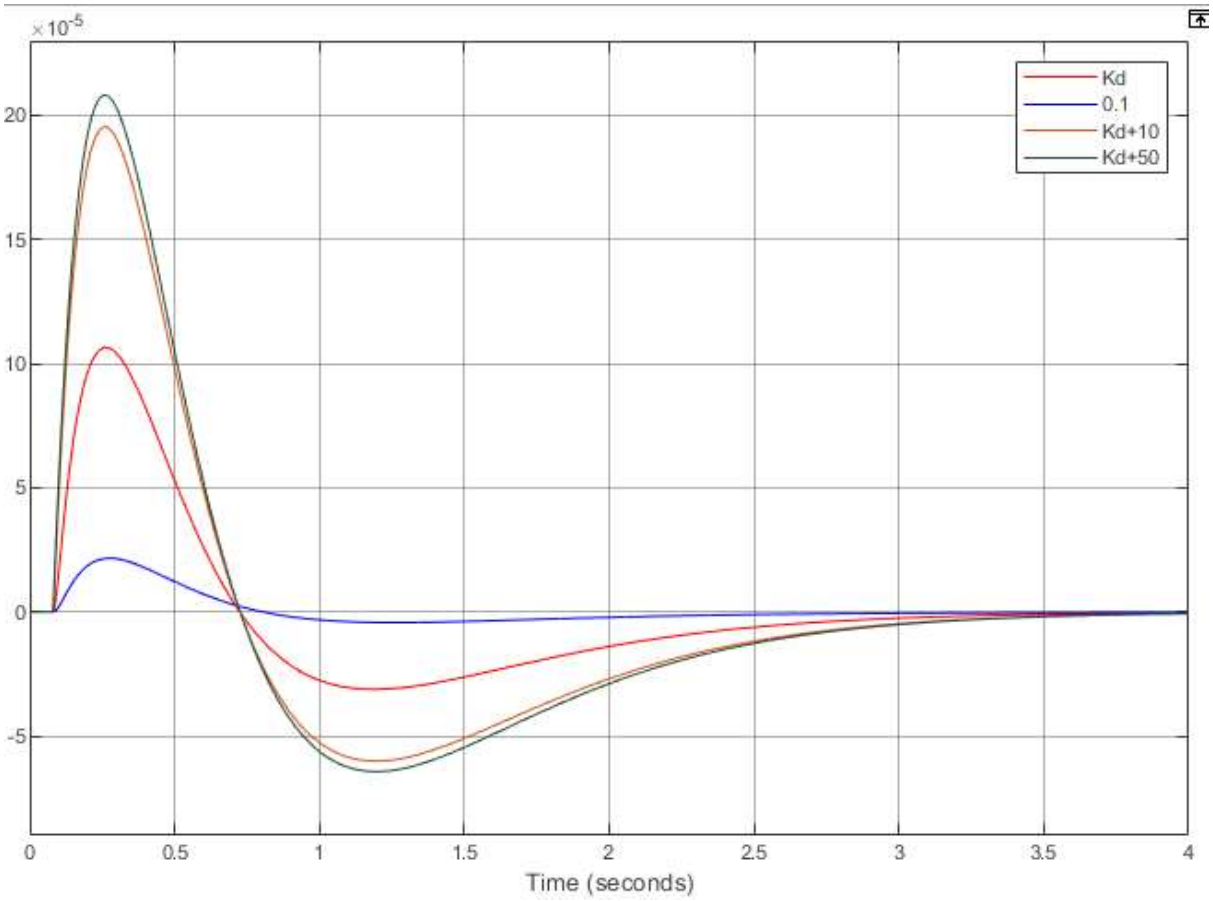


Figura 18. Sintonía Kp.

Se observa que las escalas aceptables para aumentar o disminuir valores se hallan en el orden de magnitud de 10^{-1}, por ello, en las simulaciones se muestran 4 ejemplos. El primero representará la constante proporcional previamente obtenida; los siguientes tendrán un incremento de 0.2, 0.5 y 0.7 respectivamente. Los resultados se presentan en la Figura 19 y cómo podemos observar, aumentar la constante proporcional hace que el tiempo de subida disminuya, esto a su vez produce que la sobre elongación aumente.

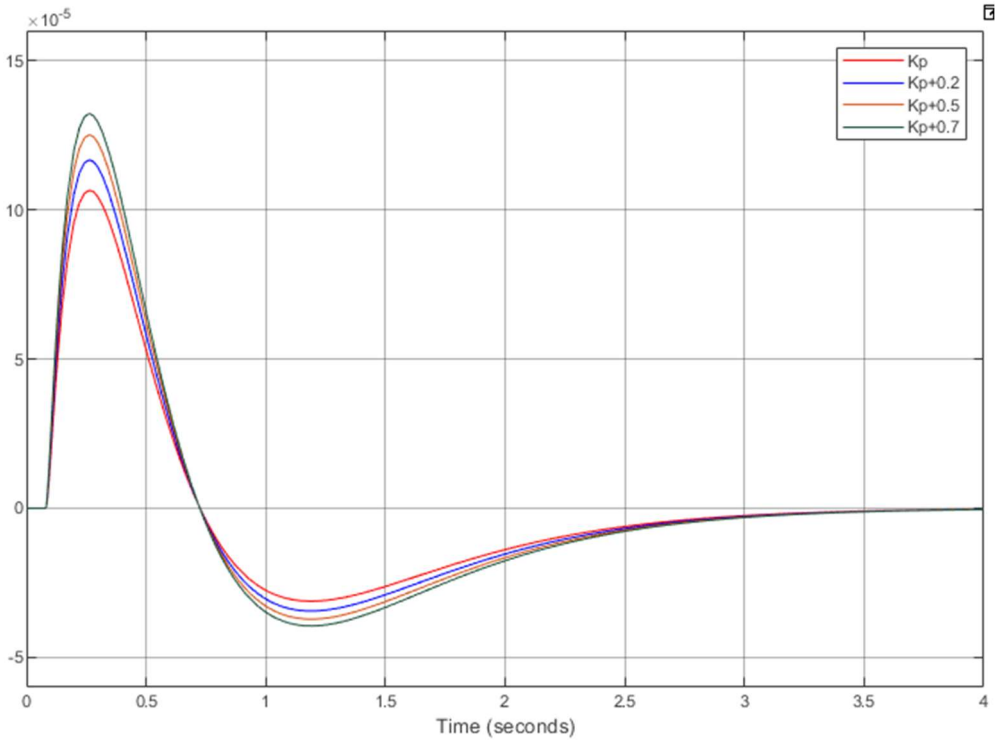


Figura 19. Comparación ángulo cuerpo.Variable Kp.

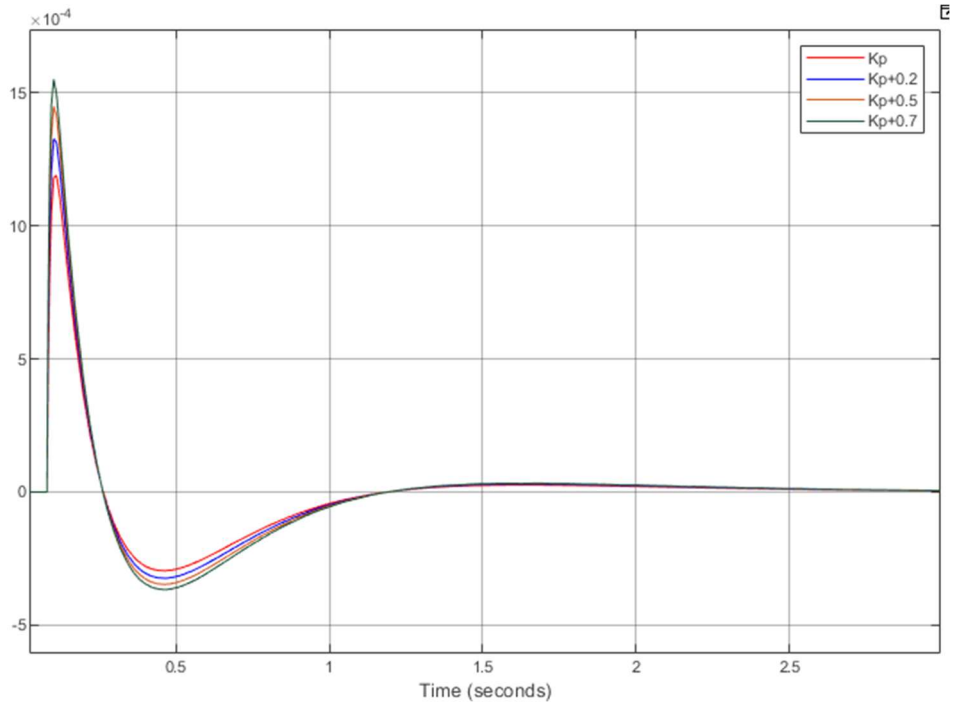


Figura 20. Comparación velocidad cuerpo. Variable Kp.

Se puede observar en la Figura 20 como la velocidad del cuerpo también se va volviendo más pronunciada, situación que podría interesar en ciertos contextos.

5.1.2 Variación de Ki

Se emplea el mismo proceso anterior para sintonizar la variable integral. En la figura 21 se pueden observar los resultados de esta simulación, y se discierne que los mayores afectados son el error estacionario y la sobreelongación. Como en el caso anterior, el orden de magnitud de los cambios es de 10^{-1}.

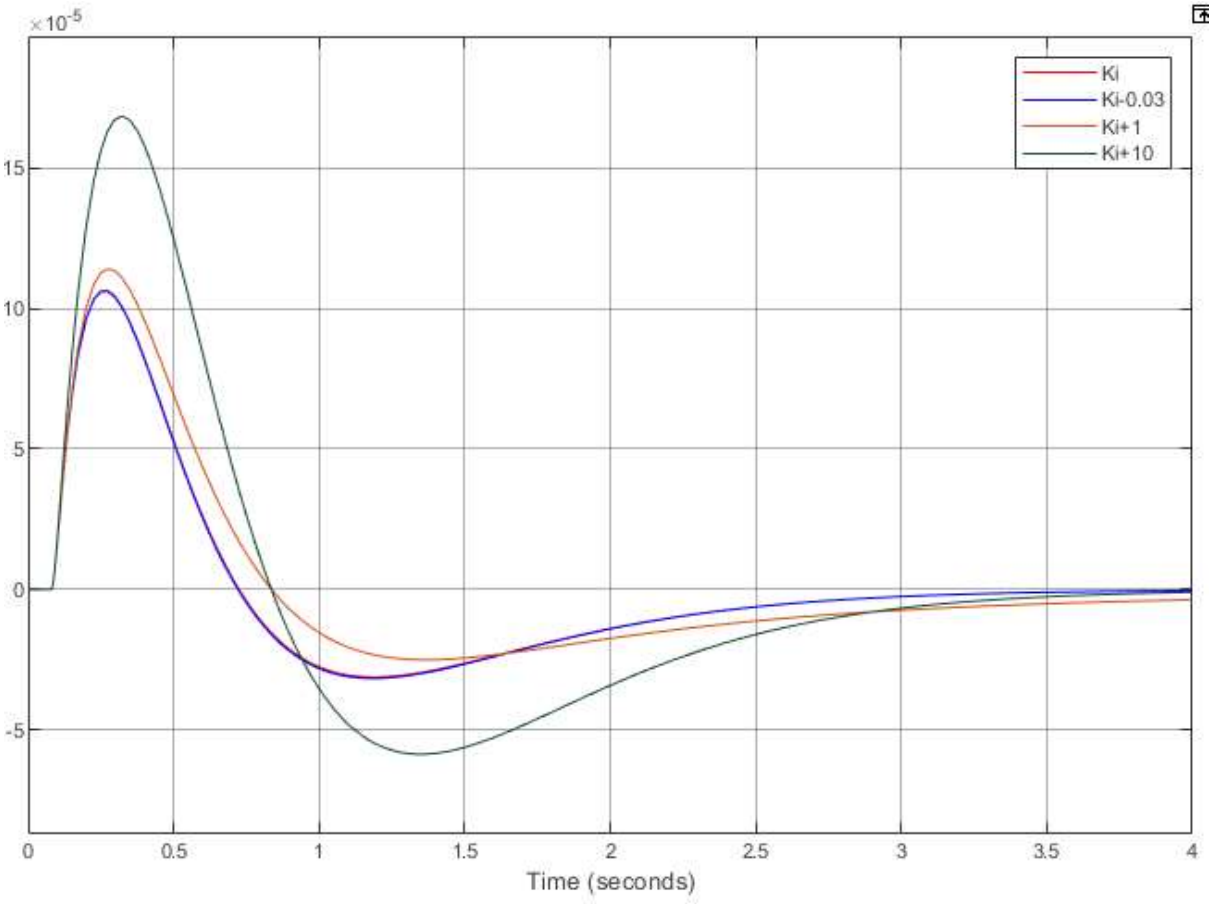


Figura 21. Sintonía Ki

En la figura 22 se muestran los resultados de las simulaciones con un rango mejor acotado de la variable, aumentando 0.2 consecutivamente, y se observar cómo se modifica mínimamente el error estacionario y como el tiempo de subida mejora ligeramente, aumentado levemente la sobre elongación.

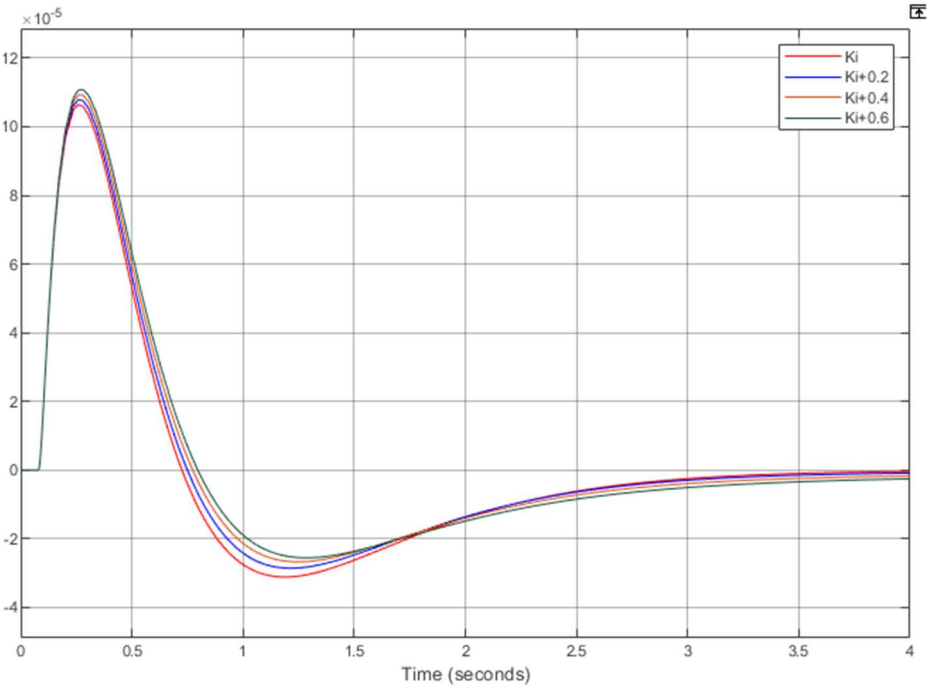


Figura 22. Comparación ángulo cuerpo. Variable Ki

La velocidad del cuerpo apenas sufre variaciones por lo que los cambios no son significativos, como se observa en la Figura 23.

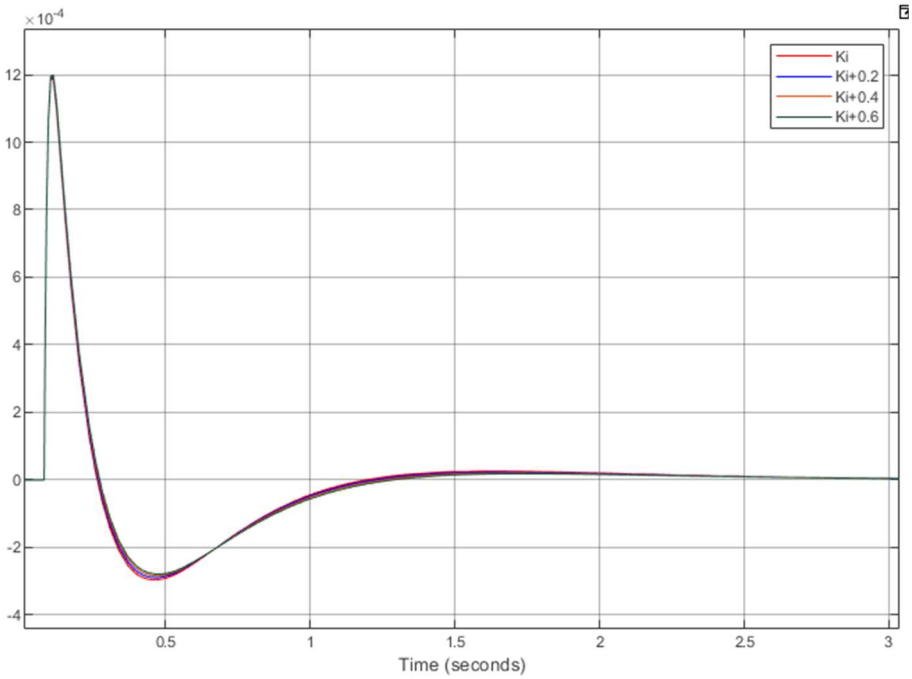


Figura 23. Comparación velocidad cuerpo. Variable Ki.

5.1.3 Variación de Kd

Como se puede observar en la Figura 24, el margen de sintonía de la variable derivativa es muy pequeño, siendo apenas imperceptibles las variaciones hasta pasado un rango en el que la señal se vuelve incontrolable. Se puede ver incluso el comportamiento extremo cuando pasamos de 0.02(naranja) a 0.05(verde).

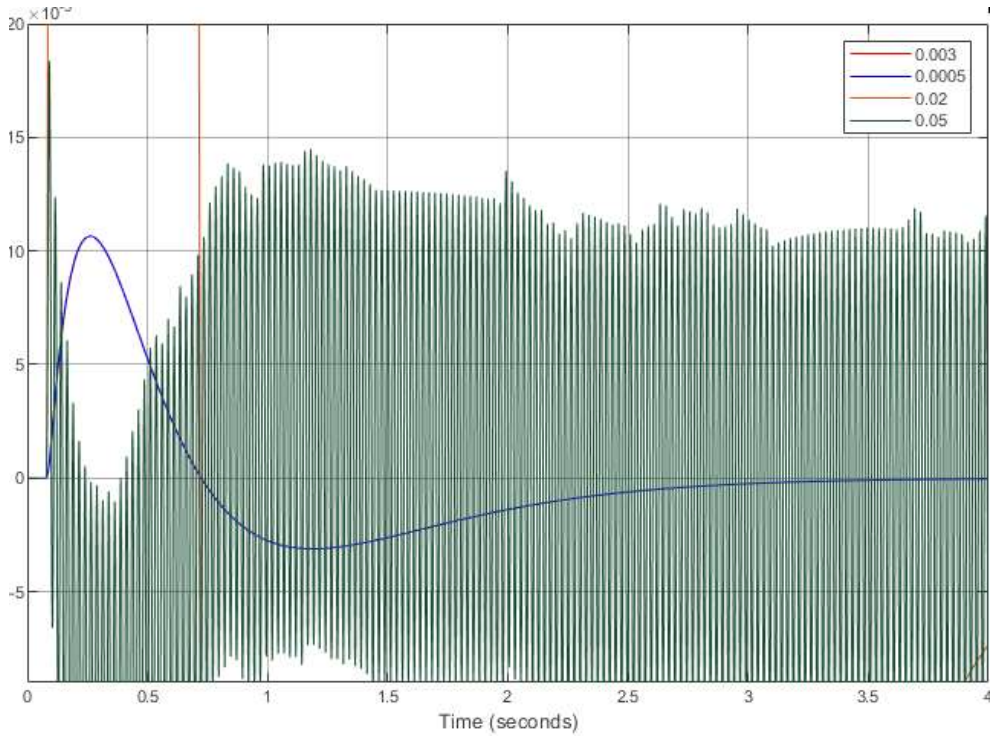


Figura 24. Sintonía Kd

Se descubre que al sumar más de 0.009 el tiempo de cálculo de la simulación se vuelve insostenible. Por lo tanto, el orden de magnitud para modificar de manera factible la constante derivativa es de 10^{-3}. Podemos ver la diferencia del ángulo con distintos parámetros en la Figura 25. Se puede observar como las gráficas se desplazan hacia la derecha, indicando que la constante derivativa reduce ligeramente el tiempo de subida. Esto se comprueba al observar la velocidad del cuerpo, que disminuye según el parámetro aumenta (Figura 26). Al ser las respuestas tan parecidas en el rango viable, se hace un aumento sobre el pico de la respuesta para que los cambios puedan ser visibles.

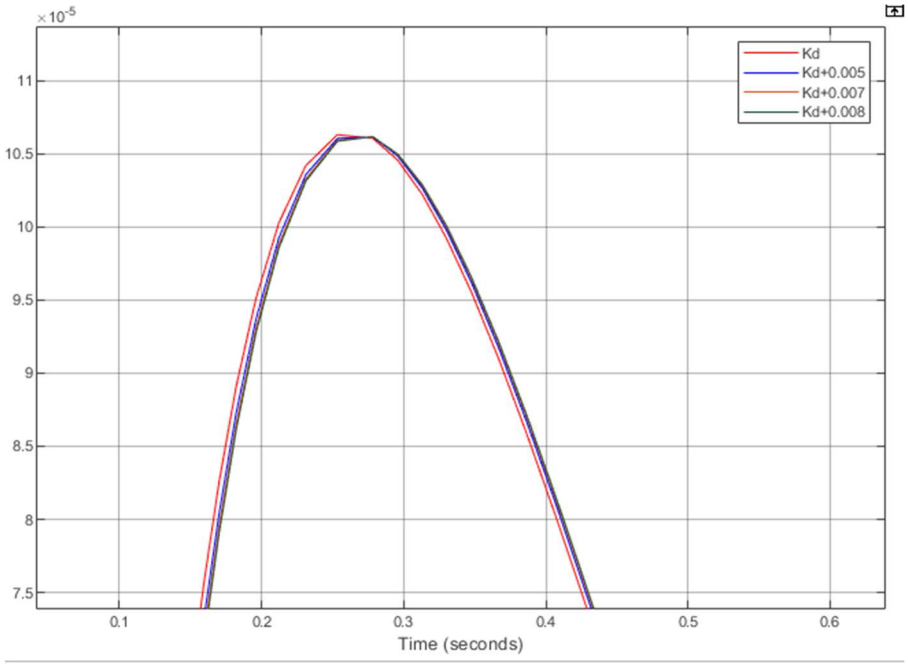


Figura 25. Comparación ángulo cuerpo. Variable Ki

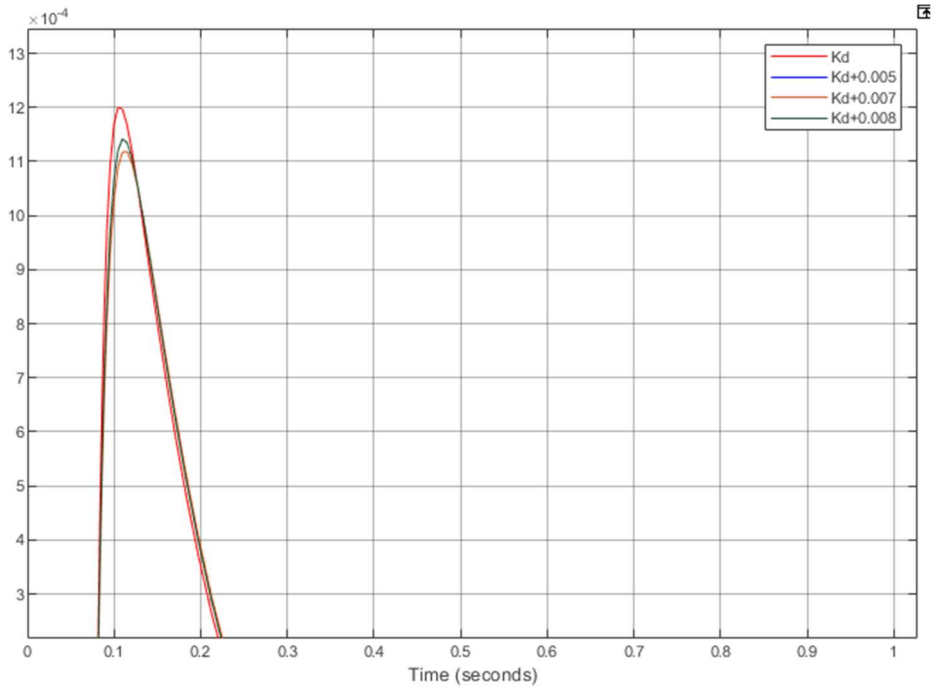


Figura 26. Comparación velocidad cuerpo. Variable Ki.

5.1.4 Discusión de los resultados

Las pruebas en el sistema han llevado a las siguientes conclusiones:

Kp alto:

Tr disminuye, *tp* diminuye, *Mp* aumenta, *ts* aumenta, *ys* aumenta

Kp bajo:

Tr aumenta, *tp* aumenta, *Mp* aumenta, *ts* aumenta, *ys* aumenta

Ki alto:
Tr disminuye, *tp* aumenta, *Mp* aumenta, *ts* disminuye, *ys* igual
Ki bajo:
Tr aumenta, *tp* disminuye, *Mp* disminuye, *ts* disminuye, *ys* aumenta
Kd alto:
Tr aumenta, *tp* aumenta, *Mp* disminuye, *ts* aumenta, *ys* igual
Kd bajo:
Tr disminuye, *tp* disminuye, *Mp* aumenta, *ts* aumenta, *ys* igual
siendo:
tr= *Tiempo de subida*, **tp**= *Tiempo de pico*, **Mp**= *Sobreelongación*, **ts**= *Tiempo de asentamiento*, **ys**= *Estado estacionario*

y a base de prueba y error, se deduce que los estados deberán de ser acotados por las siguientes condiciones:

Si ángulo alto
aumentar **kp**(+0.05) aumentar **ki**(+0.01) disminuir **kd**(-0.00005)
Si ángulo bajo
disminuir **kp**(-0.04) disminuir **ki**(-0.02) aumentar **kd**(+0.00007)
Si velocidad baja
aumentar **kp**(+0.1) aumentar **ki** (+0.1) disminuir **kd** (-0.000005)
Si velocidad alta
Disminuir **kp** (-0.2) disminuir **Ki** (-0.05) aumentar **Kd** (+0.000007)

Los sistemas expertos no son perfectos, mucho menos en la teoría de control, ya que están sujetos a las reglas que han sido descritas por los expertos. Por ello, en el programa a desarrollar se implementará de una manera sencilla, ya que la construcción de este tipo de sistemas es muy costosa. Esto lleva a integrarlo en el bucle de equilibrio. Como se ha visto anteriormente, el sistema experto deberá contar con ciertos rangos para elegir el valor óptimo, para ello, se separarán dos zonas: Estable e Inestable.
En la zona estable las correcciones a realizar serán más leves, ya que el robot se encuentra casi en equilibrio, en la zona inestable se optará por una variación mayor en los parámetros de controlador PID, para así controlar de manera agresiva el equilibrio del sistema. Se entiende que estas reglas son sencillas, pero que exponen con acierto la sintonía experta de un controlador PID.
A continuación, se muestran los rangos de los parámetros que constituirán una parte de la base de conocimiento:
Rango en zona **ESTABLE**:

Kp (0.9-0.99), Ki (0.03-0.04), Kd (0.0029-0.0031)

Rango en zona **INESTABLE**:

Kp (0.8-1.4), Ki (0.01-0.5), Kd (0.001-0.005)

En el robot este estado estará marcado por las variaciones del ángulo y la velocidad del cuerpo. La condición que separa estas zonas es: si el ángulo del cuerpo es de más de 15 grados y la velocidad es de más de 45 grados por segundo, se activa el modo inestable, si no permanecerá en modo estable.

Con esto se concluye la sintonía experta de los parámetros del controlador PID, y su implementación será expuesta en el capítulo 6.

5.2 Detección de obstáculos

En este apartado, se dotará al robot de la capacidad de detectar obstáculos y así evitar colisiones con el entorno. Para esto se hará uso del sensor de ultrasonidos, estos sensores miden la distancia a través de ondas ultrasónicas emitidas por el cabezal, que al rebotar con el objeto y retornar, posibilitan calcular la distancia al objeto, contando el tiempo entre la emisión y la recepción. La distancia se puede calcular con la siguiente fórmula (48):

$$L = ½ * T * C \qquad (48)$$

donde L es la distancia, T es el tiempo entre la emisión y la recepción, y C es la velocidad del sonido.
En el robot esto será implementado de la siguiente forma: Un hilo independiente al de balanceo, observará el valor del sensor y actuará en consecuencia en base a una serie de reglas. Para ello, se ha generado el siguiente pseudocódigo (49-51):

$$si\ (\ obstaculo\ muy\ cerca)\ \{ir\ atrás;\ giro\ cerrado\ derecha\} \quad (49)$$
$$si\ (\ obstaculo\ cerca)\ \{reducir\ velocidad;\ girar\ derecha\} \quad (50)$$
$$si\ (\ no\ obstaculo)\ \{velocidad\ rapida;\ recto\} \quad (51)$$

Esto no supone un control óptimo en la detección de obstáculos, ya que se dispone de un único sensor ultrasónico, y no se hace ningún tipo de mapeo por el espacio recorrido. Pero se demuestra que en escenarios sencillos, girar hacia la derecha hasta no detectar ningún obstáculo para avanzar, basta para evitar que el robot colisione con el entorno.
No se ha hecho demasiado hincapié en este punto, ya que el generar un buen algoritmo que recorra un espacio de manera consciente es una tarea costosa y el problema se centra en el control del péndulo invertido. Se implementa en el robot de manera anecdótica, ya que

permite ver cómo se comportan varios hilos de tareas trabajando en paralelo. Esto hace que la parte de programación tenga más interés y no se vea reducido a una sola clase ejecutando todo el código.

6 Implementación

6.2 Construcción del robot

Se ha elegido la plataforma de Lego EV3 para la construcción del robot por proporcionar gran cantidad de ventajas. Destaca lo extendido que está y lo fácil que es iniciarse en la programación de robots a partir de estas herramientas, a pesar de no tener conocimientos avanzados de robótica. Los sensores son accesibles y de fácil configuración y los bloques de LEGO permiten una estructura fácil de montar y sólida. El bloque EV3 es un bloque programable basado en un procesador ARM9 de *300 Mhz*, dónde se ejecutará el código programado. Las **características técnicas** más destacadas son [22]:

- Sistema operativo LINUX
- Procesador ARM9 de 300 Mhz
- Memoria RAM: 64 MB
- Memoria Flash: 16 MB
- Resolución de pantalla: 178x128 en blanco y negro
- Comunicación USB 2.0
- Tarjeta microSD hasta 32 GB máx.
- Puertos conectores RJ12 de los sensores y motores
- Alimentación de 6 pilas alcalinas AA o batería ionLitio recargable.

El principal sensor que se utiliza en este proyecto es el giroscopio. Con este, se detectará la velocidad angular que servirá para calcular la posición respecto al ángulo deseado del robot. Este sensor devuelve un valor con el número de grados de rotación por segundo, integrando y derivando se podrá obtener el ángulo y la aceleración, respectivamente.
Para equilibrar el robot, será necesario dotar al bloque de cierta movilidad. Esto se consigue gracias a dos servomotores de corriente continua, a estos se les indicará individualmente, la cantidad de *rpm* (revoluciones por minuto) a las que deben funcionar, siendo esta potencia nuestra señal u.
Cómo también se vio anteriormente, un sensor de ultrasonidos será el encargado de proporcionar al robot la capacidad de detectar obstáculos.
Estos tres componentes son suficientes para hacer que el robot se mantenga en equilibrio y se mueva por un espacio determinado sin chocar con obstáculos. En las figuras 27 y 28 se puede ver la primera configuración del robot.

Para la construcción el robot, el diseño se ha basado en las reglas del VIII edición del Concurso PRODEL de Control Inteligente, estando este pensado para el modelo anterior *Lego Mindstorm NTX*. Esto ha supuesto algunos problemas, ya que, por ejemplo, la dirección de los motores es la contraria a dónde mira el robot, por lo que la potencia suministrada a los servos deberá estar negada. Esto supone algunos cambios que afectan a la velocidad y la dirección que habrá que indicarle al robot para que se desplace. La estructura actual del robot es bastante inestable siendo imperativo que en futuros trabajos se consiga una estructura más estable en estado estacionario, siendo suficiente para probar el funcionamiento de los distintos algoritmos de control.

Figura 27. Robot Frontal

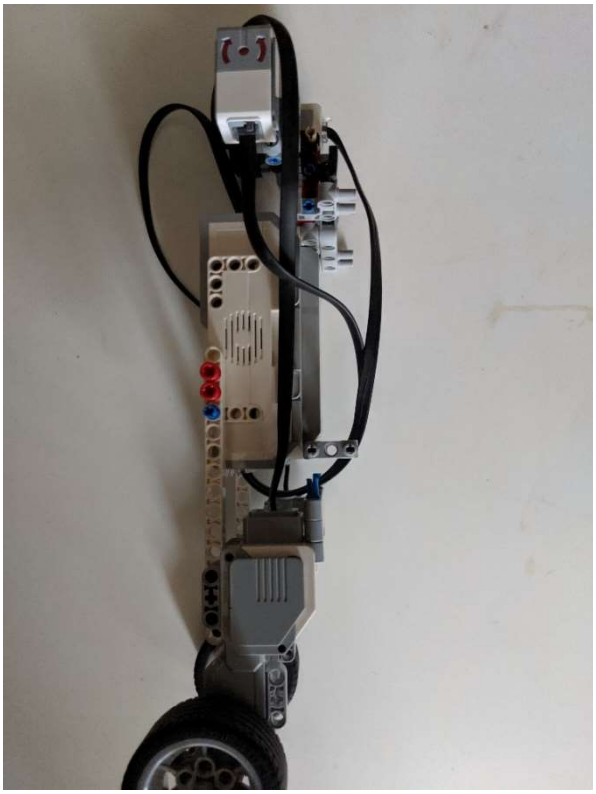


Figura 28. Robot. Lateral.

El robot en su configuración actual se muestra en las figuras 29 y 30, se ha optado por esta construcción para obtener una mayor precisión en las lecturas del giroscopio, y que el sensor de ultrasonidos pudiera detectar mejor los obstáculos.

Figura 29. Robot. Frontal

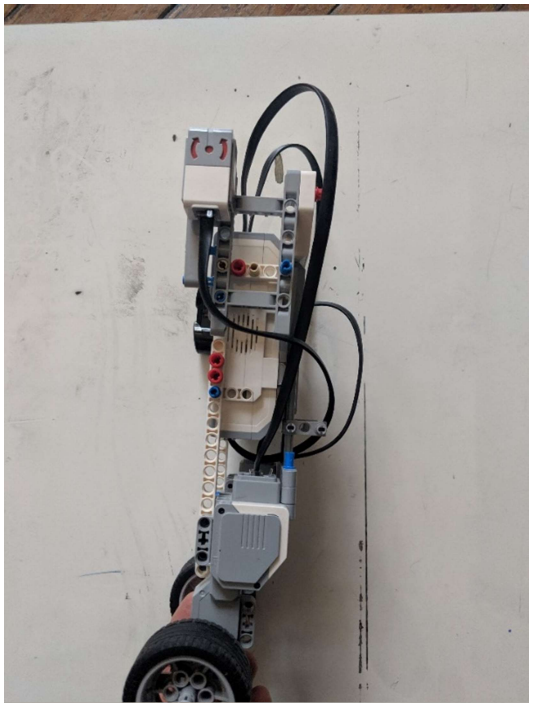


Figura 30. Robot. Lateral.

6.2 Especificaciones técnicas para el desarrollo

A la hora de empezar a implementar la solución, se ha comprobado que se podían utilizar diversos lenguajes de programación (Java, Python, C, etc.). En este caso, se opta por utilizar una de las tecnologías más conocidas: Java.

Java es un lenguaje orientado a objetos, y por ello se ha decidido tener presente los principios de programación sujetos a este paradigma. A pesar de tener una gran cantidad de documentación y de funcionalidad, se hace ver que el manejo de hilos y de la concurrencia, no es del todo óptimo en Java para operaciones que tardan tan poco tiempo. Esto nos lleva a pensar que para el futuro, será mejor optar por otros lenguajes de programación más orientados a trabajar con microcontroladores directamente (C por ejemplo).

Para poder llevar esta solución a Java y al mundo OO (Orientado a objetos) se ha hecho uso de lejOS. leJOS es una pequeña máquina virtual de Java y en 2013 se hizo un port al bloque EV3 de Lego. leJOS ofrece lo siguiente [12]:

- Programación orientada a objetos.
- Tasks.
- Arrays.
- Recursión
- Sincronización.
- Excepciones.
- La mayoría de las clases java.lang, java.io y java.util

Se intentó hacer uso de *ev3devjava*, que también ofrece el soporte Java, pero las librerías están desactualizadas y *lejOS* ofrece una mejor compatibilidad y un uso más eficiente. La instalación se realiza a través de una tarjeta SD donde va flasheado el sistema.
Al hacer esto, el robot iniciará el sistema operativo elegido en la tarjeta y se podrán ejecutar programas **Java**(*.jar*) utilizando a su vez las librerías proporcionadas para el control nativo del robot.

6.3 Implementación Java de la funcionalidad

6.3.1 Estructura

Se ha intentado estructurar el código según los paradigmas de programación orientada a objetos. Para ello, primero identificamos el que sería nuestro objeto principal: el robot.
Esta clase de robot (**EV3Block**) deberá de contar con métodos de acceso a los sensores, así como aquellos parámetros que vayan a ser compartidos por distintas "tareas" (Velocidad, aceleración, dirección). Esto permite que el programa pueda ser extendido y mantenido de manera sencilla. El programa tendrá dos "tareas" principales que serán ejecutadas en paralelo. Estas tareas son:

- **Equilibrio** - El robot hace uso de la lógica de control LQR y PID y del sensor Giroscopio para estabilizar el ángulo del cuerpo del robot.
- **Obstáculos** - El robot hace uso del algoritmo evitador y del sensor de Ultrasonidos para evitar colisiones con el entorno.

Una vez pensada la estructura, se procede a describir los parámetros de cada clase para su correcto entendimiento.

6.3.1.1 Parámetros

6.3.1.1.1 EV3Block

6.3.1.1.1.1 Constantes

Constante	**Tipo**	**Descripción**	**Valor**
GYROSCOPE	*Port*	Número de puerto del giróscopo	SensorPort.S2
ULTRASONIC	*Port*	Número de puerto del sensor de ultrasonidos	SensorPort.S3
RIGHT_MOTOR	*Port*	Número de puerto del motor derecho	MotorPort.A
LEFT_MOTOR	*Port*	Número de puerto del motor izquierdo	MotorPort.D

6.3.1.1.1.2 Variables

Variable	**Tipo**	**Descripción**
_SPEED	*double*	Velocidad del robot
_STEERING	*double*	Dirección del robot
_SHOULD_STOP	*boolean*	Variable de parada
gyroSensor	*EV3GyroSensor*	Giroscopio
gyroReader	*SampleProvider*	Lector de muestras del giroscopio
ultrasonicSensor	*EV3UltrasonicSensor*	Sensor de ultrasonidos
ultrasonicReader	*SampleProvider*	Lector de muestras del sensor ultrasónico

screen	*GraphicsLCD*	Pantalla
motorR	*EncoderMotor*	Motor derecho
motorL	*EncoderMotor*	Motor izquierdo

6.3.1.1.2 Equilibrio

6.3.1.1.2.1 Constantes

Constante	**Tipo**	**Descripción**	**Valor**
max_index	*int*	Índice máximo del array del encoder	7
gain_angle	*double*	Ganancia del ángulo del cuerpo	21.5
gain_rate	*double*	Ganancia de la velocidad angular del cuerpo	1.1520
gain_position	*double*	Ganancia de la posición de la rueda	0.1728
gain_speed	*double*	Ganancia de la velocidad de la rueda	0.1152
Kp	*double*	Constante P para el PID	0.9
Ki	*double*	Constante I para el PID	0.04
Kd	*double*	Constante D para el PID	0.005

6.3.1.1.2.2 Variables

Parámetro	**Descripción**	**Tipo**	**Variable**
θ_2	Ángulo del cuerpo	*double*	angle
θ_2'	Velocidad angular cuerpo	*double*	rate
θ_1	Ángulo de la rueda	*double*	position
θ_1'	Velocidad angular rueda	*double*	speed

Variable	**Tipo**	**Descripción**
ev3block	*EV3Block*	Objeto que representa una instancia del robot
reference_position	*double*	Posición de referencia a alcanzar
robot_speed	*double*	Velocidad global del robot
actual_tacho_count	*double*	Número de rotaciones del encoder en la iteración actual
distance_traveled	*double*	Diferencia entre el número de rotaciones de la iteración actual con la anterior.
previous_tacho_count	*double*	Número de rotaciones del encoder en la iteración anterior
encoder[]	*double*	Array que almacena las posiciones de los motores
now	*long*	Tiempo en milisegundos de la iteración actual
previous_now	*long*	Tiempo en milisegundos de la iteración anterior
dt	*double*	time step
output	*double*	Salida del sistema
input	*double*	Entrada del sistema
acc_err	*double*	Error integral
diff_err	*double*	Error derivativo
curr_err	*double*	Error proporcional
prev_err	*double*	Error anterior

NowOutOfBound	*boolean*	Booleano que controla si se está fuera de los límites en la iteración actual
OutOfBoundCount	*int*	Número de veces consecutivas que el robot no está dentro de los límites deseados
PrevOutOfBound	*boolean*	Booleano que controla si se está fuera de los límites en la iteración anterior

6.3.1.1.3 Obstáculos

6.3.1.1.3.1 Constantes

Constante	**Tipo**	**Descripción**	**Valor**
_CRITICAL_DISTANCE	*double*	Distancia en la que el robot debe dar marcha atrás	*0.2*
_SAFE_DISTANCE	*double*	Distancia considerada segura por el robot	0.4
RIGHT_SHARP	*int*	Giro cerrado a la derecha	20
RIGHT	*int*	Giro a la derecha	10
STRAIGHT	*int*	Movimiento hacia delante	0
FAST	*double*	Velocidad rápida	-0.1
FORWARD	*double*	Velocidad hacia detrás	-0.05
BACKWARDS	*double*	Velocidad hacia delante	0.05

6.3.1.1.3.2 Variables

Variable	**Tipo**	**Descripción**
distance	*double*	Distancia recogida por el sensor

6.3.1.2 Métodos

A continuación, se expondrán los diferentes métodos de las distintas clases. Posteriormente, se procederá a explicar cada apartado, proporcionando detalles y problemas propios de la implementación con el fin de que el programa quede correctamente explicado.

6.3.1.2.1 Ev3Block

Método	Tipo	Descripción
getRate	*double*	Devuelve la velocidad angular del giroscopio **returns:** *-sample[0]*
moveMotorsPwr(d)	*void*	Da potencia "d" a los motores **Parámetros:** • *final double d*: Potencia calculada por los algoritmos de control.
restart	*void*	Resetea los sensores
logToFile(text)	*void*	Genera un log **Parámetros:** • *String text*: Texto a guardar en el log, viene de las diversas tareas.
getRightTachoCount	*double*	Devuelve las vueltas del encoder derecho **returns:** *motorR.getTachoCount()*
getLeftTachoCount	*double*	Devuelve las vueltas del encoder izquierdo **returns:** *motorL.getTachoCount()*
stopMotors	*void*	Para los motores

6.3.1.2.1.1 getRate

Esta función es la encargada de leer la velocidad angular del sensor.

```
public double getRate() {
    float[] sample = { 0 };
    this.gyroReader.fetchSample(sample, 0);

    return -sample[0];
}
```

Código 1. getRate

Se coge una muestra de "gyroReader", estando este en *rateMode*. Se niega la salida porque los motores están colocados al revés.

6.3.1.2.1.2 moveMotorsPwr(d)

Esta función es la encargada de mover los motores con una potencia "d"

```
public void moveMotorsPwr(final double d) {

    this.motorR.setPower((int) (d + _STEERING));
    this.motorL.setPower((int) (d - _STEERING));

}
```

Código 2. moveMotorsPwr(d)

Al motor derecho se le suma la dirección, mientras que al izquierdo se le resta. De esta forma podremos mover las ruedas de forma que el robot gire. El rango de potencia que acepta el robot es de [-100,100].

6.3.1.2.1.3 restart

Esta función se encarga de devolver el valor original a los sensores.

```
public void restart() {
    this.motorR.resetTachoCount();
    this.motorL.resetTachoCount();
    this.gyroSensor.reset();
}
```

Código 3. restart

Se reinician los motores y el sensor de giroscopio. El sensor de ultrasonidos no necesita ser reestablecido.

6.3.1.2.1.4 logToFile(text)

Esta función guarda la cadena enviada "*text*" en el fichero de texto *log.txt*

```
public void logToFile(String text) {
    long now = System.currentTimeMillis();
    try(FileWriter fw = new FileWriter("log.txt", true);
            BufferedWriter bw = new BufferedWriter(fw);
            PrintWriter out = new PrintWriter(bw))
        {
            out.println(now+": \n");
            out.println(text);
```

Código 4. logToFile(text)

Se ha hecho esencial el uso de esta función, para el posterior estudio de los resultados de los experimentos.

6.3.1.2.1.5 getRightTachoCount

Esta función recoge el número de rotaciones del *encoder* del motor derecho. Los datos del motor izquierdo se recogen de la misma manera usando *motorL*.

```
public double getRightTachoCount() {
    return motorR.getTachoCount();
}
```

Código 5. getRightTachoCount

6.3.1.2.1.6 stopMotors

Esta función hace que que se deje de suministrar potencia a los motores, haciendo que estos se paren totalmente.

```
public void stopMotors() {
    motorR.flt();
    motorL.flt();
}
```

Código 6. stopMotors

Se hace uso de esta función cuando el robot sale del área acotada, los motores son detenidos para evitar daños al robot o a terceros.

6.3.1.2.2 Bucle de Equilibrio

Metodo	Tipo	Descripción
run	*void*	Bucle principal
initialOffset	*void*	Calcula el ángulo inicial
manageStepTime	*void*	Maneja el tiempo de step
manageAngle	*void*	Maneja el control del ángulo a través del giroscopio
managePosition	*void*	Maneja la posición a través de datos recogidos desde los encoders
LQR	*void*	Aplica el control LQR a la entrada del sistema
PID	*void*	Aplica el control PID a la entrada del sistema

checkForErrors	*void*	Controla los errores
balance	*void*	Mantiene al robot en equilibrio
logDataToFile	*void*	Guarda las variables de una iteración en un fichero de texto
control_experto	*void*	Simula la sintonía de un sistema experto

6.3.1.2.2.1 run

Este método representa el bucle principal de la tarea de Equilibrio.

```
public void run() {
    //Set priority to this thread
    Thread.currentThread().setPriority(10);
    //Initialize the block
    ev3block.restart();
    Sound.beep();
    // Get the initial angle
    initial_offset();
    // The reference position is the inital angle
    reference_position = angle;
    Sound.twoBeeps();
    previous_now = System.nanoTime();

    while (Button.ESCAPE.isUp() && !ev3block.SHOULD_STOP()) {

        manageStepTime();

        manageAngle();

        managePosition();

        LQR();

        PID(dt);

        checkForErrors();

        balance();

        Delay.msDelay(10);
    }
}
```

Código 7. Equilibrio. run

Lo primero que se hace es ponerle prioridad máxima a este hilo para asegurarse de su correcta ejecución. Se procede a la inicialización el robot, dando un pitido cuando esta tarea ha finalizado. Lo siguiente que hay que hacer, es calcular el ángulo inicial, para ello llamamos a *initial_offset*. El resultado quedará guardado en "*angle*", y esto marcará nuestra posición de referencia, es decir, el punto en el que el robot buscará el equilibrio. Se hacen sonar dos pitidos que marcarán el comienzo de la ejecución del bucle de equilibrio. Esta señal será utilizada para dejar de sujetar el robot en una posición de equilibrio. Una vez dentro del

bucle, se irán ejecutando los métodos comentados posteriormente, hasta que, o bien se pulse el botón de escape, o la condición de parada se dispare.

6.3.1.2.2.2 initialOffset

Esta función calcula el ángulo inicial del robot.

```
public void initial_offset() {
    double total_sum = 0;
    try {
        Thread.sleep(100);
        for (int i = 0; i < 30; i++) {
            total_sum += ev3block.getRate();
            Thread.sleep(5);
        }
        angle = total_sum / 30;
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Código 8. initialOffset.

Dado que el sensor del giroscopio no es muy preciso y las muestras tienen ruido, se toman 30 muestras y después se calcula la media de las muestras recogidas. Con esto, conseguimos eliminar ese ruido inicial y conseguir una posición de referencia más fidedigna a la real.

6.3.1.2.2.3 manageStepTime

Esta función maneja el tiempo de cada iteración.

```
private void manageStepTime() {
        // Get actual time
        now = System.nanoTime();
        // Calculate time step in seconds
        dt = (now - previous_now) / 1000000000.0;
        // Save value for next iteration
        previous_now = now;
}
```

Código 9. manageStepTime

Para ello, se coge el tiempo actual en milisegundos, se hace la diferencia con el tiempo anterior y se convierte a segundos. El tiempo de la iteración actual se guarda para la siguiente iteración dado que este tiempo será utilizado en futuras operaciones.

6.3.1.2.2.4 manageAngle

Esta función controla el ángulo del cuerpo del robot a través de la velocidad angular.

```
private void manageAngle() {
    // Get the rate
    rate = ev3block.getRate();
    // Calculate the current angle
    angle = angle + (rate * dt);
}
```

Código 10. manageAngle

Se llama al método del robot que coge la velocidad angular del sensor y se calcula el ángulo actual. Para ello, basta con sumar al ángulo anterior el producto de la velocidad angular con el tiempo de iteración. Con esto, obtenemos el número de grados actual.

6.3.1.2.2.5 managePosition

Esta función se encarga de controlar los parámetros relacionados con la posición del robot, es decir, la velocidad de las ruedas y su posición.

```
private void managePosition(){
    // Set the previous tachocount
     previous_tachoCount = actual_tachoCount;
    // Get the encoders tacho count
    actual_tachoCount = ev3block.getRightTachoCount() + ev3block.getLeftTachoCount();
    // Get the actual position
    distance_traveled = actual_tachoCount - previous_tachoCount;
    // Add the actual position to the previous position
    position = (position + distance_traveled);

    // Save the position in the encoder array
    encoder[index] = distance_traveled;
    // encoder[index] = actual_tachoCount;
    previous_index = index;
    index = (index + 1) % max_index;
    // Get the encoder data
    int encoder_data = 0;
    for(int i=0; i < max_index;i++) {
        encoder_data += encoder[i];
    }
    // Calculate the speed
    speed = (encoder_data)/(max_index*dt) ;

   robot_speed = ev3block.get_SPEED();
   // Update the reference position depending on the speed
   reference_position -= (robot_speed * dt);
```

Código 11. managePosition

Los valores de rotación de los *encoders* se utilizan para calcular la distancia recorrida, este valor será el que permanezca guardado en el array "*encoder*", y servirá para calcular la velocidad local del robot. La posición actual se calcula con la posición anterior y la distancia recorrida, siendo esta la diferencia entre la cuenta de la iteración actual y la cuenta de la iteración anterior.

Una vez tenemos estos datos, actualizamos la posición de referencia dependiendo del producto de la velocidad del robot con el tiempo de iteración.

El cálculo de la posición es relativo, ya que no está implementada una manera de que el robot compare su posición respecto a un espacio real. Esto lleva a ir acumulando el error indefinidamente, por lo que cuanto más se alejen los cálculos del punto inicial, mayor será la diferencia de lo medido respecto a la realidad.

6.3.1.2.2.6 LQR

Esta función aplica el control LQR a las variables recogidas en las funciones anteriores.

```
private void LQR() {
    // Apply the gains to the variables
    input = gain_angle * angle + gain_rate * rate + gain_position * (position  + reference_position)
            + gain_speed * speed;

}
```

Código 12. LQR.

Para ello, basta con multiplicar cada variable con su respectiva ganancia, anteriormente calculada, y sumarlas. Esto representará la señal de entrada al sistema en su primera fase, y esta será enviada al controlador PID.

6.3.1.2.2.7 PID

Esta función aplica el control PID a la entrada suministrada por la función LQR y devuelve la entrada suministrada a los servos del robot.

```
private void PID(double dt) {
    //Porportional error
    curr_err = input - reference_position;
    //Integral error
    acc_err = (acc_err + curr_err) * dt;
    //differential error
    diff_err = (curr_err - prev_err) / dt;
    //Previous error
    prev_err = curr_err;

    // Calculated output
    output = Kp * curr_err + Ki * acc_err + Kd * diff_err;
}
```

Código 13. PID.

Para ello se calcula el error proporcional restando la posición de referencia a la entrada. El error integral se calcula sumando el error actual con el error integral anterior y multiplicando por el tiempo de iteración. El error diferencial se calcula restando el error actual con el error anterior y dividiendo por el tiempo de iteración.
Multiplicando los errores calculados por las constantes correspondientes, obtenemos la entrada que debemos aplicar a nuestro sistema, *output*.

6.3.1.2.2.8 checkForErrors

Esta función se encarga de comprobar que el robot sigue dentro de los límites establecidos.

```
private void checkForErrors() {
    //If the angle surpasses 45
    if (Math.abs(angle) > 45) {
        //Set error condition
        NowOutOfBound = true;
    }else {
        NowOutOfBound = false;
    }
    //If error condition is met two times simultaneously
    if (NowOutOfBound && PrevOutOfBound) {
        // Add one to the count
        OutOfBoundCount++;
    } else {
        //If not, reset the count
        OutOfBoundCount = 0;
    }
    //If the error count reaches 3
    if (OutOfBoundCount > 3 ) {
        //Set the stop condition
        ev3block.set_SHOULD_STOP(true);
        //stop the motors
        ev3block.stopMotors();
        //make an error sound
        Sound.buzz();
    } else {
        //Save the state for the posterior iteration
        PrevOutOfBound = NowOutOfBound;
    }
}
```

Código 14. checkForErrors.

Si el ángulo del robot es mayor de 40 varias veces seguidas, se puede asumir que el robot no va a volver a equilibrarse, y por lo tanto debemos de parar los motores. Cuando esto pase, se avisará con un zumbido y se detendrá la ejecución del programa para evitar daños a los componentes.

6.3.1.2.2.9 balance

Esta función se encarga de suministrar la potencia a los motores del robot.

```
private void balance() {
    if (output > 100)
        output = 100;
    if (output < -100)
        output = -100;
    ev3block.moveMotorsPwr(output);
}
```

Código 15. balance

Hay que darle la potencia calculada en los pasos anteriores a los motores si queremos mantener el robot en equilibrio. El robot admite potencias entre [-100,100], por lo que se limita el valor obtenido a este rango. Después, se llama a la función del robot que se encargará de suministrar la potencia necesaria para que el robot se estabilice.

6.3.1.2.2.10 logDataToFile

Esta función envía las variables de una iteración al robot para que sean guardadas en un fichero de texto plano.

```
public void logDataToFile() {
    String stringToSave =
            " input: " + input + "\n" +
            " output: " + output + "\n" +
            " P error: " + curr_err + "\n" +
            " I error: " + acc_err + "\n" +
            " D error: " + diff_err + "\n" +
            " reference position: " + reference_position + "\n" +
            " angle: " + angle + "\n" +
            " rate: " + rate + "\n" +
            " position: " + position + "\n" +
            " speed: " + speed + "\n";

    ev3block.logToFile(stringToSave);
}
```

Código 16. logDataToFile

6.3.1.2.2.10 sistema_experto

Esta función simula el comportamiento de un sistema experto.

```
if(!stopTuning) {
if(angle > 20 || angle < -20) {
    if(i==0) {
        Kp += 0.05;
        Ki += 0.01;
        Kd -= 0.00005;
    }else {
        Kp += 0.01;
        Ki += 0.005;
        Kd -= 0.00001;
    }
}else {
    if(i==0) {
        Kp -= 0.04;
        Ki -= 0.02;
        Kd += 0.00007;
    }else {
        Kp -= 0.005;
        Ki -= 0.01;
        Kd += 0.00003;
    }
}
```

Código 18. Sistema experto, sintonía.

```
if(i==0) {
    if(Kp<0.89 || Kp>0.99) {
        stopTuning = true;
    }
    if(Ki<0.03 || Ki>0.04) {
        stopTuning = true;
    }
    if(Kd<0.0029 || Kd>0.0031) {
        stopTuning = true;
    }
}else {
    if(Kp<0.8 || Kp>1.4) {
        stopTuning = true;
    }
    if(Ki<0.01 || Ki>0.5) {
        stopTuning = true;
    }
    if(Kd<0.0001 || Kd>0.005) {
        stopTuning = true;
    }
}
```

Código 17. Sistema experto. condiciones de parada.

El estado Estable está representado como 0 y el estado Inestable como 1, esta distinción hace que las variaciones en los parámetros actúen en consecuencia. Con esto, observando el ángulo y la velocidad aplicaremos ligeros cambios en los parámetros del PID cómo se ve en código 18. Para que estos no tomen valores exagerados, se han delimitado unas condiciones de parada de la sintonía que coinciden con los rangos mencionados en el capítulo 5. Cuando

se tomen valores fuera de ese rango, la variable ***stopTuning*** tomará el valor *true* y la sintonía se detendrá. Está variable volverá a ser *false* cuando se pase de un estado a otro, permitiendo que la sintonía se pueda reiniciar. Con esto se consigue una simulación básica de la sintonía experta de los parámetros de un PID.

6.3.1.2.3 Bucle de Obstáculos

Métodos	Tipo	Descripción
run	void	Bucle principal
setSteeringAndSpeed(steering,speed)	void	Marca la dirección y la velocidad del robot Parámetros: ● int *steering*: Dirección del robot ● double *speed*: Velocidad del robot

6.3.1.2.3.1 run

Este será el método principal de la tarea de evitar obstáculos.

```
public void run() {
    while(!ev3block.SHOULD_STOP())
    {
        //Get data from the ultrasonic sensor
        double distance= ev3block.getDistance();
        if (distance< _CRITICAL_DISTANCE) {
            //If distance is critical, make two beeps
             Sound.twoBeeps();
            //Go backwards and turn sharp right
             setSteeringAndSpeed(RIGHT_SHARP,BACKWARDS);

        }else if(distance > _CRITICAL_DISTANCE && distance< _SAFE_DISTANCE) {
            //If obstacle is close, make a beep.
            Sound.beep();
            //Go forward and turn right
            setSteeringAndSpeed(RIGHT,FORWARD);

        }else if (distance> _SAFE_DISTANCE) {
            //Normal conditions go straight and forward
             setSteeringAndSpeed(STRAIGHT,FAST);
        }
```

Código 19. Obstaculos. run

Según el pseudocódigo anteriormente citado, se generará el código Java correspondiente. Para ello se hace uso de dos constantes, *CRITICAL_DISTANCE* y *SAFE_DISTANCE*, que servirán de referencia para acotar el estado dónde se encuentre el robot respecto del obstáculo. Si el robot se encuentra en una posición muy comprometida, pitará dos veces y tratará de dar una potencia negativa a los motores y de girar bruscamente a la derecha. Si el

obstáculo no está demasiado cerca, simplemente se reducirá la velocidad y se girará ligeramente a la derecha. En el caso de no haber ningún obstáculo, el robot continúa recto a la velocidad normal.

Hay que tener en cuenta, que los giros que debe dar el robot no deben de ser ni demasiado rápidos ni demasiado cerrados, ya que esto dificultará muchísimo la tarea concurrente de equilibrio.

Dado que los motores están al revés, las velocidades hacia delante serán negativas, y las velocidades hacia atrás serán positivas.

6.3.1.2.3.2 setSteeringAndSpeed

```
private void setSteeringAndSpeed(int steering, double speed) {
     ev3block.set_STEERING(steering);
     ev3block.set_SPEED(speed);
}
```

Código 20. setSteeringAndSpeed

Esta función se encarga de cambiar las variables del robot, para que la tarea de equilibrio tenga en cuenta los obstáculos. Con esto, el robot será capaz de mantener el equilibrio a la vez que esquiva ciertos obstáculos.

7. Conclusiones y trabajos futuros

Este trabajo ha servido para aplicar técnicas de control a un sistema real junto con una estrategia inteligente para la sintonía de uno de los reguladores implementados. Se ha conseguido comprender los mecanismos de control sobre un robot lego, así como la importancia del modelado para después plasmarlo en la realidad.

El problema es en sí complejo; una de las primeras dificultades encontradas fue la de obtener unas ecuaciones para representar el modelo del robot. Sería deseable estudiar en un futuro la mejora de estas ecuaciones, aunque analizar y generar el modelo de su dinámica ha permitido comprender de manera más profunda los mecanismos del control, y como la inteligencia artificial jugará un papel importante sobre estos en los próximos años.

Al aplicar el modelo sobre el sistema real, se han descubierto serias diferencias con lo planteado sobre las simulaciones. Parámetros como el nivel de la batería afectan a la potencia de los servos, pero no se toman en cuenta por aumentar demasiado la complejidad del modelo.

Al utilizar un sistema operativo que permite programar con un lenguaje de más alto nivel (Java), con una buena documentación, ha permitido un manejo nativo eficiente del robot. Se entiende que para proyectos de prototipado es factible el uso de Java, pero si se precisa para un uso industrial o que necesite mayor fiabilidad, será necesario replantear el problema en lenguajes de programación de más bajo nivel. El uso de Java y su orientación a objetos, ayuda a que el proyecto pueda ser usado didácticamente, ya que la construcción del robot supone un reto interesante, haciendo que los conocimientos puedan ser adquiridos de manera más amena.

A pesar de conseguir la estabilidad del robot, para posibles futuros trabajos, una buena modificación sería una mejor sintonía de los parámetros PID, quizás utilizando algún otro método de inteligencia artificial para ello, como redes neuronales, algoritmos genéticos… estas ideas quedan como posibles proyectos futuros.

Conclusions and future projects

This project has served to apply control techniques to a real system along with an intelligent strategy for the tuning of one of the implemented regulators. It has been possible to

understand the control mechanisms on a lego robot, as well as the importance of modeling and then translate it into reality.

The problem is complex itself; one of the first difficulties encountered was to obtain some equations to represent the robot's model. It would be desirable to study in the future the improvement of these equations, although analyzing and generating the model of its dynamics has allowed us to understand more deeply the mechanisms of control, and how artificial intelligence will play an important role in these in the coming years.

When applying the model on the real system, serious differences have been discovered with what was raised about the simulations. Parameters such as the battery's level affect the power of the servos, but are not taken into account due to the complexity of the model is greatly increased.

By using an operating system that allows programming with a higher level language (Java), with good documentation, has allowed an efficient native handling of the robot. It is understood that the use of Java is feasible for prototyping projects, but if it is required for industrial use or which requires greater reliability, it will be necessary to rethink the problem in lower-level programming languages. The use of Java and its orientation to objectives, helps the project to be used didactically, since the construction of the robot is an interesting challenge, making the knowledge to be acquired more enjoyable.

Despite achieving the stability of the robot, for possible future work, a good modification would be a better tuning of the PID parameters, perhaps using some other artificial intelligence method for it, such as neural networks, genetic algorithms ... these ideas remain as possible future projects.

Bibliografía

[1] Grasser, F., D'arrigo, A., Colombi, S., & Rufer, A. C. (2002). JOE: a mobile, inverted pendulum. *IEEE Transactions on industrial electronics*, *49*(1), 107-114.

[2] Salerno, A., & Angeles, J. (2003, September). On the nonlinear controllability of a quasiholonomic mobile robot. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)* (Vol. 3, pp. 3379-3384). IEEE.

[3] Nguyen, H. G., Morrell, J., Mullens, K. D., Burmeister, A. B., Miles, S., Farrington, N., ... & Gage, D. W. (2004, December). Segway robotic mobility platform. In *Mobile Robots XVII* (Vol. 5609, pp. 207-220). International Society for Optics and Photonics.

[4] Viguria, A., Prieto, A., Fiacchini, M., Cano, R., Rubio, F. R., Aracil, J., & Canudas-de-Wit, C. (2009). Desarrollo y experimentación de un vehículo basado en péndulo invertido (ppcar). *Revista Iberoamericana de Automática e Informática Industrial*, *3*(4), 53-62.

[5] Visioli, A. (2006). *Practical PID control*. Springer Science & Business Media.

[6] K. Ogata. Modern Control Engineering, 4th ed., New Delhi: Pearson Education (Singapore) Pvt. Ltd., 2005.

[7] Furuta, K., Yamakita, M., & Kobayashi, S. (1991, October). Swing up control of inverted pendulum. In *Proceedings IECON'91: 1991 International Conference on Industrial Electronics, Control and Instrumentation* (pp. 2193-2198). IEEE.

[8] Wiklund, M., Kristenson, A., & Aström, K. J. (1993). A new strategy for swinging up an inverted pendulum. *IFAC Proceedings Volumes*, *26*(2), 757-760.

[9] Ang, K. H., Chong, G., & Li, Y. (2005). PID control system analysis, design, and technology. *IEEE transactions on control systems technology*, *13*(4), 559-576.

[10] Xu, B. (2019). A Comparative Study of PID and LQR Control Strategies Applied to Inverted Pendulum Systems.

[11]Williams, James H. Jr., Fundamentals of Applied Dynamics, John Wiley & Sons, New York, 1996.

[12]Documentación Lejos. http://www.lejos.org/ev3/docs/

[13] Gómez, M., Arribas, T., & Sánchez, S. (2012). Optimal control based on CACM-RL in a two-wheeled inverted pendulum. *International Journal of Advanced Robotic Systems*, *9*(6), 235.

[14] Rodriguez-Garavito, C. H., Arevalo-Castiblanco, M. F., & Patiño-Forero, A. A. (2018, June). Implementation of a Non-linear Fuzzy Takagi-Sugeno Controller Applied to a Mobile Inverted Pendulum. In *The 13th International Conference on Soft Computing Models in Industrial and Environmental Applications* (pp. 344-353). Springer, Cham.

[15] International Journal of Recent Technology and Engineering (IJRTE)ISSN: 2277-3878,Volume-7, Issue-6S,March 2019

[16] Modelado y control del péndulo invertido sobre carro mediante sistemas híbridos.

Universidad de Sevilla. Escuala técnica superior de ingenieros

[17] García, F. M. (2007). El controlador PID. *Online] Disponible en: http://es. slideshare. net/MnicaMoreno/el-controlador-pid*.

[18] Castillo, E., Gutiérrez, J. M., & Hadi, A. S. (1997). Sistemas expertos y modelos de redes probabilısticas. *Academia de Ingenierıa*. (pp 7-9)

[19] García-Nieto, Sergio, Martínez, Miguel, Llosá, Ángel, & Sanchis, Javier. (2007). Estrategias de Enseñanza a Distancia sobre Control No-Lineal Aplicada al Péndulo Invertido. *Información tecnológica*, *18*(5), 85-98. https://dx.doi.org/10.4067/S0718-07642007000500011

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

[21] Stimac, A. K. (1999). *Standup and stabilization of the inverted pendulum* (Doctoral dissertation, Massachusetts Institute of Technology, Dept. of Mechanical Engineering).

[22] http://canaltic.com/rb/legoev3/15_el_bloque_ev3.html