APLICACIÓN DE LA INTELIGENCIA ARTIFICIAL AL CONTROL DE UN ROBOT LEGO EN CONFIGURACIÓN DE PÉNDULO INVERTIDO

Por

Eva Gala de Pablo

UNIVERSIDAD COMPLUTENSE

MADRID

Doble Grado en Matemáticas e Ingeniería Informática

Facultad de Informática

Dirigido por:

Matilde Santos Peñas

Madrid, 2018-2019

Resumen

Este estudio presentó el diseño y la implementación de un robot basado en Lego Mindstorm EV3 imitación de un péndulo vertical invertido que es capaz de mantenerse en equilibrio, avanzar y detectar formas y colores. Así mismo, se confeccionó un modelo matemático análogo al prototipo mediante un espacio de estados y se revisó y simuló el sistema con un regulador lineal cuadrático (LQR) y un controlador proporcional, integral y derivativo (PID). Se implementó el código funcional en Java leJOS. Finalmente el robot fue capaz de mantenerse en equilibrio, evitar obstáculos y reconocer colores a lo largo de su recorrido.

Abstract

This research presents the design and implementation of a robot based on Lego Mindstorm EV3. The robot is a reproduction of an inverted vertical pendulum that is stabilized, moves forward and notices shapes and colours. To this end, a mathematical model analogous to the prototype was stablished through a state-space representation. Additionally, the system was reviewed and simulated with a linear–quadratic regulator and a proportional–integral–derivative controller. The code was implemented in Java leJOS. The resulting controller allowed the EV3 robot to stay steady, avoid obstacles and detect colours along its path.

Palabras clave

Péndulo invertido, Espacio de estados, robot balancín, Lego Mindstorm EV3, Regulador LQR, Controlador PID.

Keywords

Inverted pendulum, State-space representation, balancing robot, Lego Mindstorm EV3, LQR, PID Controler.

Índice general

Dedicatoria

Mamá, aunque empezaste este proyecto conmigo y viste a Gsus caminar, no pudiste verle correr...

Capítulo 1

Introducción

La materia en la que se centra este proyecto es la construcción y el control de un sistema de péndulo invertido. Esta cuestión física es conocida por ser uno de los problemas más representativos y habituales en la teoría de control. El proyecto engloba la construcción, el modelado y la programación de un robot Lego Mindstorms EV3® (EV3 o robot, de ahora en adelante).

1.1. Propósito

La elección de este proyecto como trabajo de fin de grado viene motivado por la posibilidad de profundizar en la teoría de control; un campo multidisciplinario y transversal en el que intervienen tanto las ciencias de la computación como las matemáticas. Por ejemplo, en el estudio de sistemas dinámicos se requiere la aplicación de ecuaciones diferenciales y conocimientos de álgebra avanzado. Por otro lado, el desarrollo de este proyecto requiere de una programación sólida y en varios lenguajes; y la propia construcción del robot requiere unos conocimientos básicos en robótica.

1.2. Planteamiento

El planteamiento inicial que se hizo del proyecto fue, por parte de la directora como el siguiente:

Este trabajo responde a la propuesta del concurso de control inteligente, organizada por el grupo temático de Control Inteligente del Comité Español de Automática.
El Control Inteligente nace con la intención de aplicar las técnicas de Inteligencia Artificial a los problemas de control. La Inteligencia Artificial en sí es un campo amplio que abarca lógica, optimización, probabilidad, percepción, razonamiento, toma de decisiones, aprendizaje, etc. El proyecto consiste en aplicar paradigmas de control inteligente a una planta real. El sistema a controlar es un Lego en configuración de péndulo invertido. El prototipo está disponible en la Facultad de Informática. Una vez realizado el montaje de la planta, se trata de diseñar e implementar en el sistema un controlador basado en alguna de las técnicas de control inteligente (lógica fuzzy, redes neuronales, etc.) o una combinación de varias de ellas. Para validar el control, el sistema debe realizar un recorrido a lo largo de un circuito controlando en todo momento el ángulo del péndulo

invertido, que va situado sobre un carrito con ruedas. El Anyway deberá desplazarse dentro de la pista detectando y salvando los obstáculos colocados sin salirse del perímetro de la pista. Durante el desarrollo de la prueba el robot irá buscando y detectando topos de colores que llevan asociada una determinada acción.

Este objetivo preliminar del proyecto tiene como motivación la Convocatoria del IX Concurso PRODEL de Control Inteligente [1] y el péndulo vertical como pieza fundamental. El comportamiento del robot una vez fue capaz de mantenerse erguido estuvo más abierto a las posibilidades que se quisieron explorar sujeto a la planificación y el tiempo.

En resumen, se fijó como objetivo principal del proyecto la exploración de los métodos de control para la obtención de un robot erguido. Se decidió no priorizar el comportamiento posterior del EV3, ya que puede ser un proyecto entretenido para su mejora, una vez concluido el estudio actual. Se puede encontrar una lista detallada de objetivos en la sección 1.5.

1.3. Descripción del problema

Como ya se aventuraba antes, el péndulo invertido es uno de los problemas más habituales en la teoría de control. El problema en cuestión está ilustrado en el esquema de la Figura 1.1.

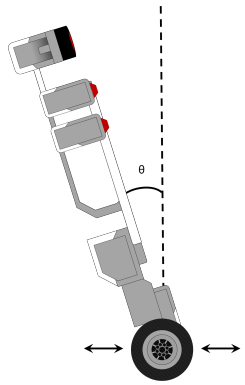


Figura 1.1: Esquema del péndulo invertido en el robot EV3.

El sistema se basa en una estructura vertical capaz realizar un movimiento unidimensional y compuesto por un péndulo y carro. Por tanto, el objetivo es que el carro se mueva para compensar el desplazamiento del péndulo ante la gravedad y la sobrecompensación[1]. La meta final es aplicar cierta fuerza para conseguir que $\theta \approx 0$.

Este problema tiene muchos ejemplos y usos prácticos, pero quizás el más importante de todos sea el propio cuerpo humano. Una persona de pie actúa como un péndulo invertido cuya estabilidad se alcanza con el movimiento de los pies. El cuerpo se equilibra con un sistema realimentado que usa las percepciones (visión, balance, etc.) para ajustar los movimientos de las piernas y así poder mantener el equilibrio. Aunque este modelo quizás sea poco visual por su alta complejidad, vale la pena ser conscientes de la importancia del estudio incluso, como tanteo a la estabilidad de robots del tipo humanoide y otras imitaciones del cuerpo humano.

[1] consideramos el fenómeno en el que, al intentar evitar la caída hacia un lado se ejerce demasiada fuerza y se provoca la caída hacia el otro

Un ejemplo más simple es su aplicación en transporte como el Segway Proffesional de Segway HT®; un vehículo de transporte ligero con autobalanceo (ver Figura 1.2).

Figura 1.2: Ejemplo aplicación péndulo invertido: Segway. Licencia Pixabay.

En este documento se hablará sobre el modelo matemático de este problema, las estrategias para el diseño de un controlador y, sobre todo, en la aplicación de todos estos conocimientos y métodos de trabajo a la ardua tarea de conseguir que EV3 se mantenga en equilibrio.

1.4. Antecedentes

Se cuenta con una gran cantidad de antecedentes; tanto teóricos, que tienen más recorrido y son más genéricos y aplicables, como con el propio robot. Aunque estos últimos son más escasos al encontrarse solo con tres modelos distintos (RCX, NXT y EV3).

1.4.1. Antecedentes teóricos

El problema del péndulo invertido es un problema recurrente como ejemplo educativo en física, dinámica y teoría del control. Es una representación muy clara y simple de un sistema mecánico inestable, y por ello cuenta con unos antecedentes muy extensos. Una de las primeras referencias al problema es la demostración de la solución de Roberge en su trabajo de fin de grado *The Mechanical Seal* en 1960 [2] intentando mantener una escoba en posición vertical.

Un extenso análisis de la historia del péndulo invertido se puede encontrar en *History of Inverted-Pendulum Systems* de Lundberg y Barton (2003) [3]. Allí se menciona a Higdon and Cannon (1963) como los primeros en publicar un estudio de sistemas con múltiples péndulos invertidos independientes haciendo referencia al trabajo de Roberge [3]. También se nombra a Truxal y las sus notas docentes (1965) como la primera referencia al modelo descrito con un espacio de estados [3]. Se concluye que, para finales de la década de los 60, múltiples libros fueron publicados incluyendo discusiones sobre este tópico y distintas soluciones [3]. Entre ellos, el propio *Modern Control Engineering* de Ogata (1970) [18] a cuya última edición se hacen múltiples referencias en este estudio.

El análisis del estudio actual se ha basado en *Non-Linear Swing-Up and Stabilizing Control of an Inverted Pendulum System* de M. Bugeja (2003) [4] y el libro *Linear optimal control Systems* de Kwakernaak y Sivan (1972) [5].

1.4.2. Antecedentes aplicados

El repertorio de proyectos de Lego Mindstorm es bastante extenso. A día de hoy, la búsqueda de "Mindstorm" en GitHub revela más de 1.500 repositorios en más de 10 lenguajes de programación diferentes y en los tres distintos modelos lanzado por Lego: el RCX, el NXT y EV3.

Bloque RCX

El primer bloque de Mindstorms, el RCX, fue lanzado en 1998 y se vendieron en total un millón de unidades. Tuvo tres versiones distintas: 1.0, 1.5 y 2.0.

En 2002 Steve Hassenplug [6] construyó el primer robot basado en un péndulo invertido con Lego Mindstorms bloque RCX, el *Legway* (ver Figura 1.3). El lenguaje de programación fue BrickOS (LegOS), un sistema operativo libre alternativo para programar en este módulo.

El método de equilibrio se basaba en el uso del sensor electro-óptico detector de proximidad para detectar la distancia al suelo y compensarla cuando se acercaba demasiado.

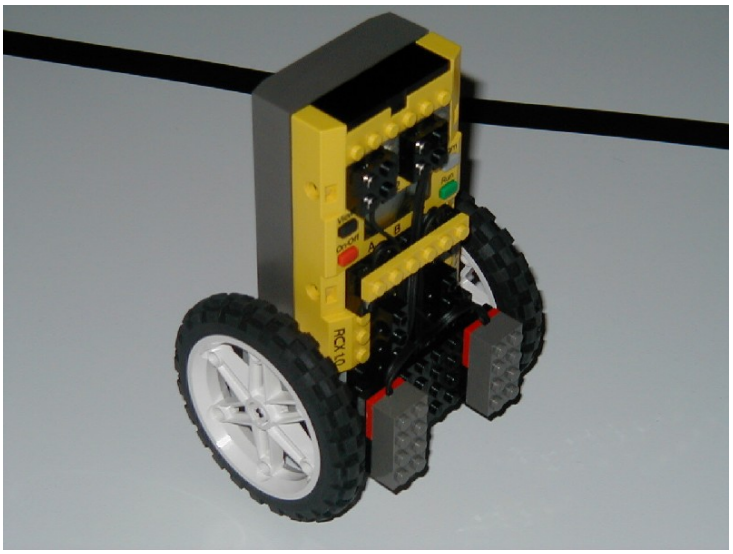


Figura 1.3: Modelo *Legway*. Imagen de la Página Oficial.

Bloque NXT

En julio de 2006 se lazó la siguiente generación: el Lego Mindstorms NXT [7] (ver Figura 1.4). Como novedad contaba con *bluetooth*, sensor ultrasónico, sensor de color, etc. Este modelo es mucho más parecido al actual, y tiene mucho más trabajo documentado.

En 2007 Philippe "Philo" Hurbain publicó su robot *NXTway* [8] inspirado en el *Legway* de Hassenplug. Su sistema de control se basaba en el detector de luz del modelo aunque "el control

Figura 1.4: Bloque NXT. CC BY-SA 3.0 de Hugo12 (Wikipedia)

es mucho más rudimentario que el (robot) de Steve, es suficiente para que el NXTway se mantenga balanceado un rato..."(cita traducida). Se programó en NBC[2] usando BricxCC[3].

Ese mismo año, Ryo Watanabe publicaba el primer robot de Lego que se mantenía en equilibrio con el sensor del giroscopio programado en RobotC [9]. Y, en el año 2008, Yorihisa Yamamoto [10] publicó el primer robot que, no solo se sostenía, si no que además se controlaba por *bluetooth*. Se programó en leJOS C.

También cabe destacar el trabajo de fin de grado de la Universidad Complutense de Madrid "Control Inteligente de péndulo invertido" de 2011 por A. Hernández Largacha y *et al.* [11]. Éste fue programado en RobotC.

EV3

En 2013 se lanzó la tercera y más moderna generación de robots de Lego Mindstorm: el EV3. Las mayores diferencias entre estos modelos se encuentran en el propio bloque. [12]

- EV3 tiene un mayor procesador y es más rápido, aunque en contra partida es más lento a la hora de inicializarse.
- EV3 puede ser controlador por tanto iOS y Android cuando NXT solo podía por el segundo.
- EV3 tiene como novedad la ranura para la tarjeta SD para cargar programas.
- El *software* de EV3 es más avanzado y se pueden escribir pequeños programas simples en el propio robot.

Dentro de los antecedentes del modelo EV3 caben mencionar los que más se han consultado en el estudio. El *Self-Balancing EV3 Robot* de L. Valk (2014) [13] programado en EV3-G. El modelo de 2015 de Lora Thola (Universitat Politènica de Catalunya) [14] programado en BricxCC. Y, por último, el *GyroBoy* por C. Bjørn Klint (2017) [15] en Java leJOS; muy similar al que se va a construir en este estudio, pero con distinta constitución y con controlador LQR.

[2] Next Byte Codes (NBC) es un lenguaje simple de programación en Lego NXT de tipo ensamblador.
[3] Bricx Command Center (BricxCC) es un entorno de desarrollo integrado (IDE) de *Windows* para programar Lego Mindstorm.

Este modelo tiene menos antecedentes, probablemente porque solo lleva seis años en el mercado y no hay suficientes diferencias con NXT como para que haya un gran cambio en el paradigma.

1.5. Objetivos

Como ya se ha dejado establecido, el objetivo principal del proyecto fue la obtención de un robot EV3 que fuera capaz de equilibrarse, avanzar e interactuar con el medio que le rodea. Para ello, se plantearon unos hitos incrementales que pretendieron culminar con la obtención de dicho robot.

I. Comprensión de los conocimientos básicos en teoría de control y los controladores LQR y PID.
II. Comprensión del modelado matemático de un péndulo invertido.
III. Obtención de un modelo relativamente fiable en MATLAB y Simulink de EV3.
IV. Construcción de EV3.
V. Comprensión de un método de programación válido para EV3.
VI. Obtención de un controlador LQR que sostenga a EV3.
VII. Obtención de un controlador PID que sostenga a EV3 mejor.
VIII. Otras funcionalidades.

1.6. Plan de trabajo

Se puede hacer una idea del plan de trabajo en el diagrama 1.5. Para que la planificación fuera acorde con los objetivos, los hitos de la planificación se han numerado según los objetivos anteriores. La escala de colores de los meses corresponde a la disponibilidad que se consideraba que se iba a tener para trabajar en el proyecto. El plan de trabajo inicial se basaba en la entrega del proyecto en la convocatoria de junio. Se tuvo que actualizar el plan de trabajo para amoldarla a la nueva disponibilidad y a la entrega del proyecto en septiembre.

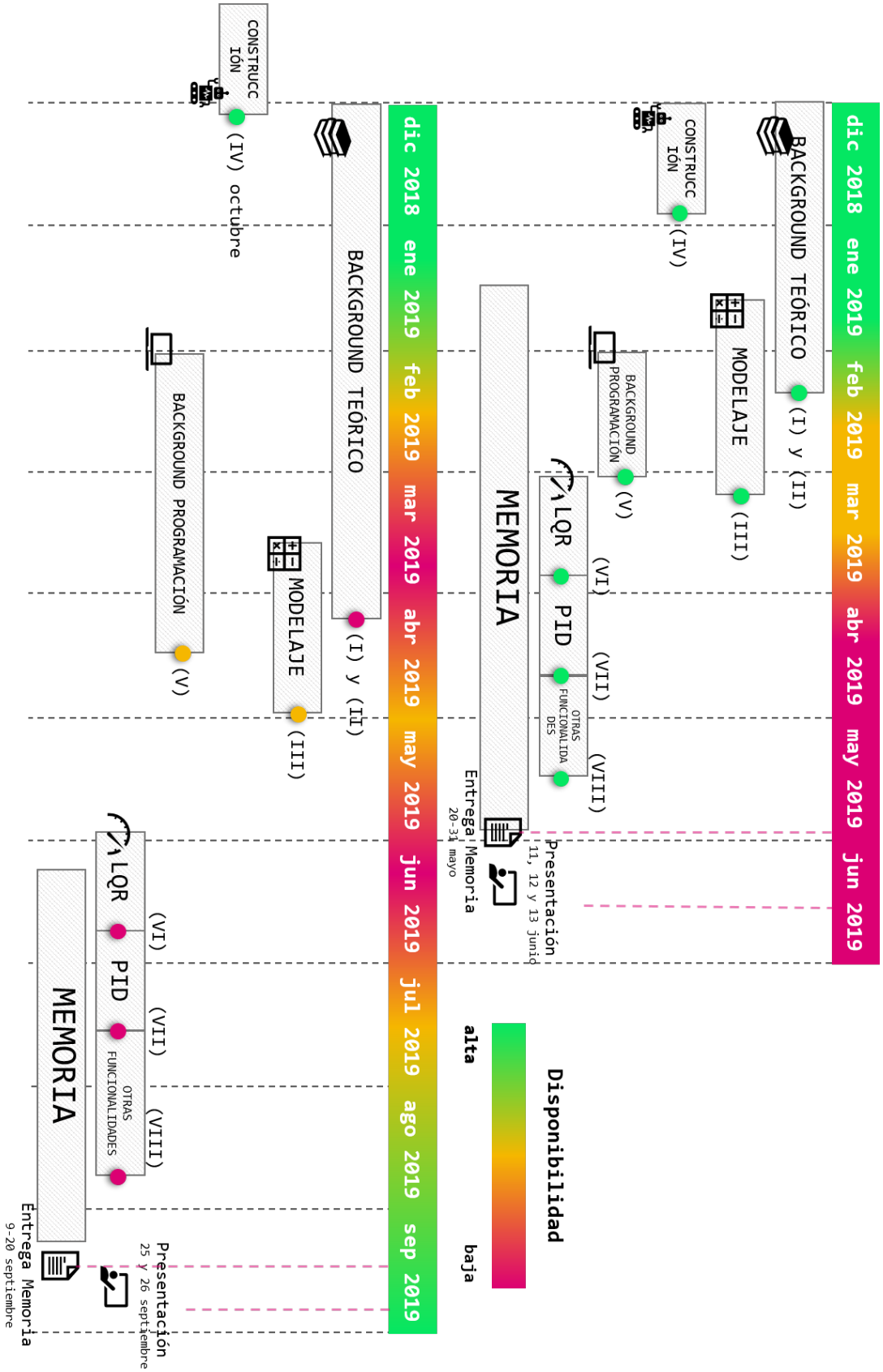


Figura 1.5: Diagrama explicativo de la planificación del proyecto.

Capítulo 2

Modelo matemático

En esta fase del proyecto se procuró encontrar una expresión matemática que representase el comportamiento aproximado del sistema. Es necesario encontrar un equilibrio entre un modelo fiable y una complejidad aceptable. Para ello, se comenzó con un estudio simple de las fuerzas existentes en un péndulo invertido estándar.

2.1. Análisis de las fuerzas y obtención de ecuaciones

Para poder obtener un buen modelo del sistema, se debe analizar por separado cada uno de los cuerpos del modelo que se han explicado en la introducción de la memoria: péndulo y carro. Así bien, cuando este sistema se lleva a la práctica con el EV3, sucede que estas partes no están separadas físicamente.

Símbolo	Significado	Unidades
m	Masa del péndulo	kg
M	Masa del carro	kg
k	Constante de fricción del carro con el suelo	$m \cdot kg/s$
c	Constante de fricción en la unión del péndulo y carro	$m \cdot kg/s$
L	Longitud del péndulo al centro de gravedad	m
I	Inercia del péndulo	$kg \cdot m^2$
θ	Ángulo que forma el péndulo con la vertical	rad
x	Desplazamiento del carro en la horizontal	m
F	Fuerza aplicada al carro	N

Cuadro 2.1: Tabla con las variables que se usarán en el modelo

El péndulo invertido se puede definir y modelar como un cuerpo rígido presente solo en dos dimensiones. La tabla 2.1 muestra las variables que se utilizaron y lo que representa cada una y la Figura 2.1 representa el diagrama de fuerzas que se ejercen sobre el péndulo invertido. Se obtienen las componentes de las fuerzas en estas dos dimensiones con las ecuaciones (2.1) y (2.2).

$$\sum F_i = ma_i \tag{2.1}$$

$$\sum F_j = ma_j \tag{2.2}$$

Estas ecuaciones representan las fuerzas de un movimiento plano en un cuerpo, de acuerdo con la

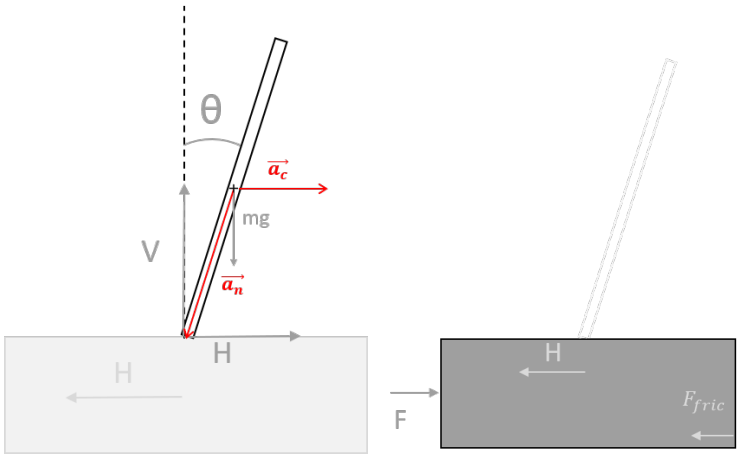


Figura 2.1: Diagramas de cuerpo del sistema

Segunda Ley de Newton. También se ha de considerar la Versión Rotacional de la Segunda Ley de Newton, la ecuación (2.3). Siendo F_i y F_j la suma de fuerzas en dirección vertical y horizontal y τ el torque, o la fuerza ejercida sobre el eje de giro. [4]

$$\tau = I\alpha \tag{2.3}$$

Empezando por el péndulo, se pueden encontrar dos aceleraciones representadas como a_c y a_n en la figura 2.1. La aceleración asociada al movimiento lineal del carro (a_c) es horizontal y la aceleración asociada al movimiento rotatorio (a_n) sigue la dirección del ángulo de giro θ.

Tomando las fuerzas y aceleraciones horizontales en el péndulo y aplicando las ecuaciones (2.1) y (2.2) se obtienen las ecuaciones (2.4) y (2.5), respectivamente. Aquí se considera H y V como la suma de fuerzas horizontales y las verticales que se ejercen sobre el péndulo respectivamente. [4]

$$H = m \cdot \left(\frac{d^2x}{dt^2} + \frac{d^2L \cdot \sin(\theta)}{dt^2} \right) \tag{2.4}$$

$$V - mg = m \cdot \left(\frac{d^2L \cdot \cos(\theta)}{dt^2} \right) \tag{2.5}$$

Si se estudian las fuerzas asociadas al movimiento rotatorio en el péndulo, con la ecuación (2.3), se obtiene la ecuación (2.6). En este caso se representa $\frac{d\theta}{dt}$ y $\frac{d^2\theta}{dt^2}$ como $\dot{\theta}$ y $\ddot{\theta}$ para simplificar la notación. [4]

$$VL\sin(\theta) - HL\cos(\theta) - c\dot{\theta} = I \cdot \ddot{\theta} \tag{2.6}$$

Por otro lado, cabe destacar que en la ecuación (2.6), el término $c\dot{\theta}$ es la fuerza asociada a la fricción de la unión del péndulo con el carro. Como en el EV3 no existe tal unión, se considera que no se debe de tener en cuenta y se fija una aproximación $c \approx 0$.

Y, por último, se calculan las fuerzas horizontales de (2.1) que se ejercen sobre el carro en la ecuación (2.7). [4]

$$F - H = M\ddot{x} + k\dot{x} \tag{2.7}$$

El termino $k\dot{x}$ hace referencia a la fricción del carro contra el suelo. Siguiendo la línea de muchas fuentes consultadas, se usa la ecuación de la fricción como si fuera fricción viscosa. Este enfoque se debe a que la fricción de la rueda con el suelo es prácticamente inexistente y la única fricción que se tiene que considerar es la fricción propia del motor que, al ser lubricada, se corresponde con una fricción viscosa. Por ello se establece que la fricción es proporcional a la velocidad del cuerpo [4].

En el artículo [16] se puede ver un estudio de los diferentes tipos de fricción que se pueden considerar y, con ello, los resultados obtenidos. En el caso de sólo considerar una fricción viscosa en la modelización, se deduce que los controladores tienden a oscilar ligeramente. Sin embargo, en las conclusiones del artículo se admite que añadir el rozamiento con el suelo, aunque mejora ligeramente los resultados, complica el modelo y sigue presentando ruido, probablemente debido a los sensores.

Ahora, para aclarar el origen de las ecuaciones principales de las que se obtendrá el modelo, se procede a calcular las derivadas apropiadas y simplificar la notación. De las ecuaciones (2.4) y (2.5) se obtienen las ecuaciones (2.8) y (2.9). [5]

$$H = m\ddot{x} + mL\left(\ddot{\theta}\cos(\theta) - \dot{\theta}^2\sin(\theta)\right) \tag{2.8}$$

$$V - mg = -mL\left(\ddot{\theta}\sin(\theta) + \dot{\theta}^2\cos(\theta)\right) \tag{2.9}$$

La expresión de $\ddot{x}$ se obtiene de la suma de (2.8) + (2.7). Para obtener $\ddot{\theta}$, (2.6) $- L\cos(\theta)$(2.8) + $L\sin(\theta)$(2.9).

$$\ddot{x} = \frac{1}{m+M}\cdot\left[F - k\dot{x} - mL\left(\ddot{\theta}\cos(\theta) + \dot{\theta}^2\sin(\theta)\right)\right] \tag{2.10}$$

$$\ddot{\theta} = \frac{1}{I + L^2 m}\cdot\left[mL\left(g\sin(\theta) - \ddot{x}\cos(\theta)\right)\right] \tag{2.11}$$

2.1.1. Linealización del modelo

Para el diseño de un controlador se va a usar una versión lineal de las ecuaciones que se han obtenido anteriormente.

Como el péndulo se encuentra estable solo en la vertical, se puede considerar que, idealmente, siempre estará en un estado cercano. Por ello, se aproxima $\theta \approx 0$ y con ello se aplican las aproximaciones de ángulos pequeños tales como $\sin(\theta) \approx \theta$, $\cos(\theta) \approx 1$ y $\dot{\theta}^2\theta \approx 0$ [4]. Se obtienen, así, las dos ecuaciones anteriores ((2.10) y (2.11)) linealizadas:

$$\ddot{x} = \frac{1}{m+M}\cdot\left[F - k\dot{x} - mL\ddot{\theta}\right] \tag{2.12}$$

$$\ddot{\theta} = \frac{1}{I + L^2 m} \cdot [mL\,(g\theta - \ddot{x})] \tag{2.13}$$

Es importante tener en cuenta que, al usar estas aproximaciones de ángulos pequeños, se está aproximando las series de Taylor del $\cos(\theta)$ y $\sin(\theta)$ al primer término. Pero, para que la expansión no contenga términos en potencia de $\pi/180$, en θ se ha definido los ángulos en radianes. Por ello, se fija las unidades del modelo a radianes.[17]

2.1.2. Diagrama de estados

Un diagrama de estados es una representación de un modelo matemático de un sistema físico mediante un conjunto de entradas, salidas y variables de estado. Estas entradas y salidas están relacionadas mediante ecuaciones diferenciales que se combinan en una ecuación diferencial matricial de primer orden. La representación de un modelo de esta forma proporciona un modo compacto y conveniente de modelar un sistema con varias entradas y salidas.[18]

En nuestro caso se tienen 4 entradas y 4 salidas, que forman un sistema de 4 dimensiones.

Para obtener un espacio de estados lineal válido a través de las ecuaciones obtenidas anteriormente es necesario sustituir $\ddot{\theta}$ en la ecuación (2.12) usando la ecuación (2.13) y, de la misma forma, sustituir $\ddot{x}$ en la ecuación (2.13) usando la ecuación(2.12). [4, 18]

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ Lmgp_1 & 0 & 0 & \frac{Lmkp_1}{M+m} \\ 0 & 0 & 0 & 1 \\ \frac{-(Lm)^2 gp_2}{I+L^2 m} & 0 & 0 & -kp_2 \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{-Lmp_1}{M+m} \\ 0 \\ p_2 \end{bmatrix} \cdot F \tag{2.14}$$

Con $p_1 = \frac{M+m}{I(M+m)+L^2 mM}$ y $p_2 = \frac{I+L^2 m}{I(M+m+L^2 mM)}$.

Para simplificar la notación, se renombran las matrices de la ecuación (2.14) como $\mathbf{A}$ y $\mathbf{B}$, el vector $[\theta \quad \dot{\theta} \quad x \quad \dot{x}]^T$ como $u(t)$ y se incluye una ecuación que represente la salida del sistema $v(t)$:

$$\dot{u}(t) = \mathbf{A}u(t) + \mathbf{B}F(t) \tag{2.15}$$

$$v(t) = \mathbf{C}u(t) + \mathbf{D}F(t) \tag{2.16}$$

Donde $\mathbf{C} \in \mathbb{R}^4 \times \mathbb{R}^4$ es la matriz identidad y $\mathbf{D} \in \mathbb{R}^4$ es un vector de ceros.

2.1.3. Función de Transferencia

A partir del sistema de estados de la sección anterior se obtuvo la función de transferencia.

Primero, de la ecuación (2.15) se obtiene la transformada de Laplace ($\int_0^{\text{inf}} f(t)e^{-st}dt$) con condiciones iniciales cero. [18]

$$tU(t) = \mathbf{A}U(t) + \mathbf{B} * V(t) \tag{2.17}$$

$$V(t) = \mathbf{C}U(t) \tag{2.18}$$

Y, de la ecuación (2.17), reordenando y sustituyendo en (2.18) se obtiene

$$U(t) = \mathbf{C}\left(t\mathbf{I} - \mathbf{A}\right)^{-1}\mathbf{B}V(t) \tag{2.19}$$

Con esto, se logra la función de transferencia. [18]

$$H(t) = \frac{U(t)}{V(t)} = \mathbf{C}\left(t\mathbf{I} - \mathbf{A}\right)^{-1}\mathbf{B} \tag{2.20}$$

El denominador de la función de transferencia se hace cero cuando $(t\mathbf{I} - \mathbf{A})$ es cero. Lo que es equivalente a que los polos de la función de transferencia son los autovalores de la matriz $\mathbf{A}$. [19, 18, 5]

Esta afirmación es especialmente útil a la hora de comprobar la inestabilidad del sistema.

Capítulo 3

Controladores

Para asegurar un correcto funcionamiento del dispositivo y un final funcional, se aplicó un desarrollo incremental del controlador. Es decir, primero se intentó obtener un controlador LQR funcional y después mejorar éste con un controlador PID.

En este capítulo, se expondrá la base teórica que se requiere para entender los controladores que después se han implementado. Pero, para poder entender esta base teórica se tendrán que definir dos conceptos fundamentales:

Sistema de lazo abierto Un sistema en el que la salida del sistema no tiene ningún efecto sobre la acción del sistema. [18]

Sistema de lazo cerrado o realimentado Un sistema que mantiene una relación determinada entre la salida y la entrada de referencia, comparándolas y usando la diferencia como medio de control. [18]

3.1. Controlador lineal cuadrático

La técnica de control LQR se usó como base para ver la veracidad del modelo matemático y hacer un primer controlador válido. La Figura 3.1 muestra un diagrama del controlador.

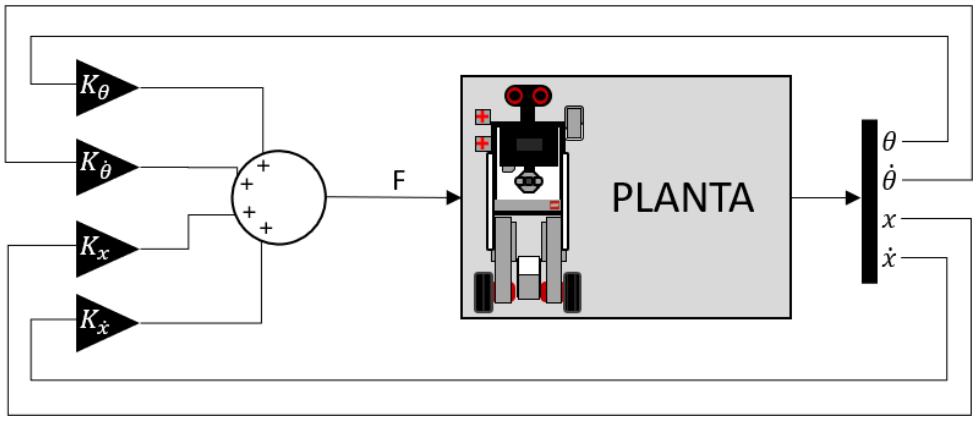


Figura 3.1: Diagrama del controlador LQR.

El objetivo de esta técnica es obtener las constantes K_θ, $K_{\dot\theta}$, K_x y $K_{\dot x}$ con los que se calculó una

suma ponderada que minimizó el error del sistema.

Se ha concluido que nuestro sistema se comporta como se rige en la ecuación (2.15). Teniendo en cuenta esto, se define la función de coste $J(t)$. [5]

$$J(t) = u^T(t)\mathbf{S}u(t) + \int_0^t \left(u^T(e)\mathbf{Q}u(e) + F(e)^T\mathbf{R}F(e)\right) de \tag{3.1}$$

Aunque este método acarrea una reducción del cálculo del factor humano, se requiere introducir al algoritmo ciertos parámetros que condicionan la prioridad de minimización. Las matrices de pesos $\mathbf{S}$, $\mathbf{Q}$ y $\mathbf{R}$ son las que parametrizan estos criterios. Las matrices $\mathbf{S}$ y $\mathbf{Q}$ son simétricas y no definidas negativas, mientras $\mathbf{R}$ es simétrica y definida positiva. La matriz $\mathbf{Q}$ es la matriz de peso para los estados intermedios, la matriz $\mathbf{R}$ es la matriz de peso para la acción de control del sistema y la matriz $\mathbf{S}$ representa el peso del estado final.

Para ver un ejemplo más ilustrativo, en el primer caso de la ecuación (3.2) el objetivo es minimizar el error total obtenido en las cuatro variables de salida del sistema en el tiempo t. Esta configuración sería práctica en un sistema que no vaya a ser perturbado y vaya a funcionar durante un determinado periodo de tiempo t.

En el segundo caso, se centra más en minimizar el gasto total de energía. Podría ser útil para obtener un proyecto más ecológico que equilibrase al robot usando la mínima cantidad de energía posible.

$$\begin{cases} \mathbf{S} & = 1 \\ \mathbf{Q} & = 0 \\ \mathbf{R} & = 0 \end{cases} \quad \Longrightarrow \quad J = ||u(t_1)||^2$$

$$\begin{cases} \mathbf{S} & = 0 \\ \mathbf{Q} & = 0 \\ \mathbf{R} & = I \end{cases} \quad \Longrightarrow \quad J = \int_0^{t_1} ||F(t)||^2 dt \tag{3.2}$$

En el caso del sistema que se diseñó, $\mathbf{Q} \in \mathcal{M}_{4\times4}(\mathbb{R})$, $\mathbf{R} \in \mathbb{R}$ y $\mathbf{S} = 0$, ya que se necesita una estabilidad hasta $t = \infty$. Por ello, se considera nuestro problema un sistema con solución en tiempo infinito cuyo coste viene definido por:

$$J_\infty = \int_0^\infty \left[u^T(t)\mathbf{Q}u(t) + F(t)^T\mathbf{R}F(t)\right] dt \tag{3.3}$$

Y, para solucionar este problema, se estima que la solución de minimización viene dada por $F(t) = -\mathbf{K}u(t)$ con $\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}$. Y esta $\mathbf{P}$ se obtiene mediante la ecuación matricial de Riccati asociada [18, 20].

$$\mathbf{PA} + \mathbf{A}^*\mathbf{P} - \mathbf{PBR}^{-1}\mathbf{B}^*\mathbf{P} + \mathbf{Q} = 0 \tag{3.4}$$

Para el cálculo de este resultado se puede utilizar el código de MATLAB

```
lqr(A,B,Q,R)
```

3.2. Controlador Proporcional, Integral y Derivativo

El algoritmo PID se basa, como su propio nombre indica, en acciones de control proporcional, integral y derivativa. Su ganancia proporcional (K_p) se calcula pensando en compensar el error actual del sistema; la integral (K_i), el cúmulo de errores pasados; y la derivativa (K_d), una predicción de los errores futuros. Así, la suma ponderada de estos errores pretende evitar una sobrecompensación o una acción de corrección basada en un pico de los datos.

Históricamente este algoritmo es uno de los más usados en control. Es importante tener en cuenta que su uso no garantiza un control óptimo o, ni siquiera, la estabilidad del mismo.

A veces, algunas aplicaciones se basan solo en alguna parte de este sistema, dejando de lado alguno de los parámetros. Por ejemplo, hay controladores proporcionales - integrales, o proporcionales - derivativos.

En general, este método se puede combinar en parte con un controlador LQR, como se ve en la figura 3.2. El controlador LQR obtiene una acción de control mediante la suma ponderada

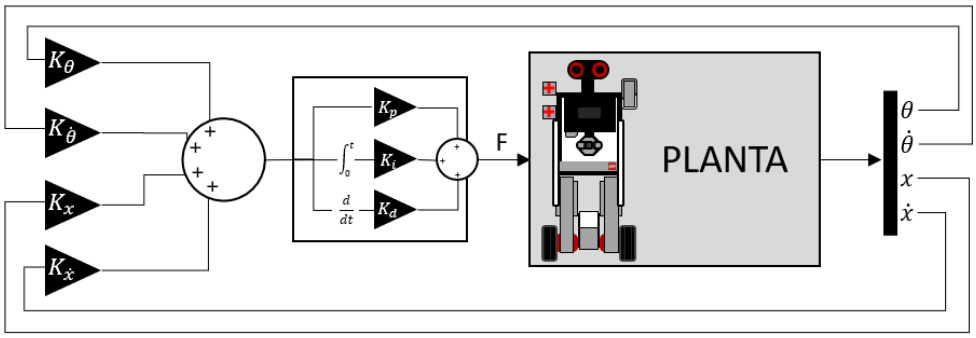


Figura 3.2: Diagrama del controlador LQR.

de las salidas del sistema. A la salida obtenida se le aplica un controlador PID para regular su compensación adecuadamente.

La forma matemática de expresar la salida que se obtiene de un controlador PID esta representada en la ecuación (3.5) siendo $e(t)$ el error obtenido en el tiempo t. [18]

$$F(t) = K_p e(t) + K_i \int_0^t e(\tilde{t})d\tilde{t} + K_d \frac{de(t)}{dt} \tag{3.5}$$

Este controlador y el determinado en el punto anterior LQR serán los que se simulen en el capítulo **??** y se apliquen a la construcción del robot EV3 en el capítulo **??**.

Capítulo 4

Simulación

En este capítulo, con el modelo matemático calculado en el capítulo 2, y antes de empezar con el regulador, se comprobó que el sistema podía, efectivamente, ser estable y tener solución.

Primero se definieron las constantes que se aplicaron a las ecuaciones obtenidas, lo cual se puede ver en la tabla 4.1.

Símbolo	Valor	
m	0,7 kg	[1]
M	0,027 kg	[1]
k	0,1 $m \cdot kg/s$	[2]
L	0,145 m	[1]
I	0,006 $kg \cdot m^2$	[2]

[1] Valores medidos empíricamente.
[2] Valores obtenidos de [19]

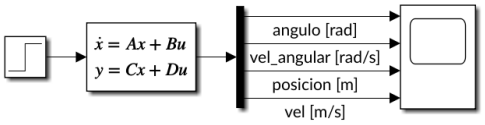
Cuadro 4.1: Valores a introducir en el modelo

Las simulaciones se hicieron en MATLAB para definir el sistema y Simulink para simular ese sistema. Para simplificar la lectura, el código que se requiere para la comprensión de esta memoria, se puede encontrar en el Anexo A.1.

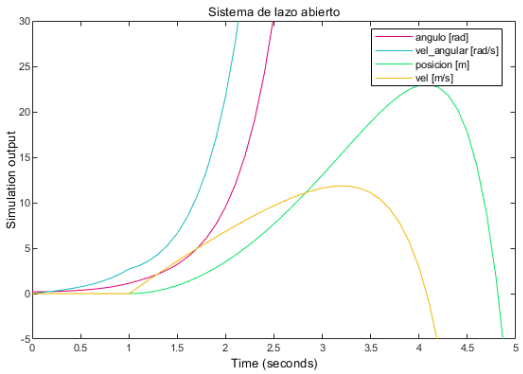
4.1. Sistema de lazo abierto

Primero se simuló el sistema en lazo abierto para ver que, siguiendo lo que la intuición dictaba, era inestable. Para ello se ejecutó el código A.1 con el sistema en la Figura 4.1a y se obtienen los resultados que se pueden ver en Figura 4.1b.

Tuvieron sentido estos resultados: si el robot se libera con una inclinación inicial distinta de cero, la gravedad le hará caer. El movimiento de desplazamiento se debe a la señal en forma de escalón que se introduce como F del sistema.



(a) Sistema simulado en lazo abierto



(b) Respuesta del sistema en simulación

Figura 4.1: Simulación del sistema en lazo abierto

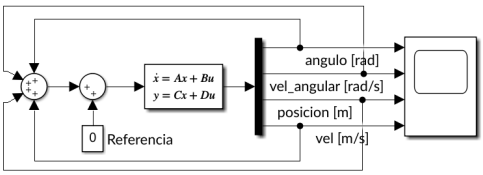
Para confirmar que es un sistema inestable, calculamos sus polos:

$$polos = \begin{bmatrix} 0 \\ 2{,}4219 \\ -2{,}4224 \\ -0{,}1500 \end{bmatrix} \tag{4.1}$$

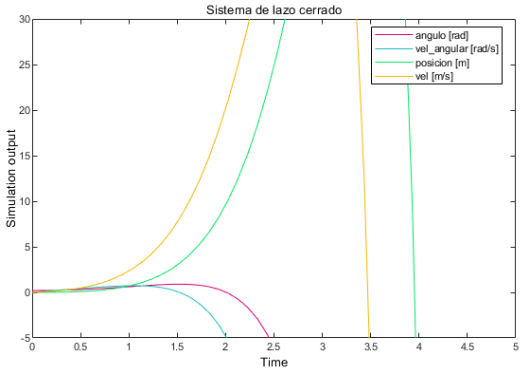
El segundo polo del sistema se encuentra en el plano positivo y confirma que el sistema es inestable.

4.1.1. Sistema de lazo cerrado

La realimentación en sí puede ser, en algunos casos, un método de control. Por tanto, se simuló el sistema introduciendo el propio cálculo de error como la suma de las variables. Se consideró este valor la fuerza a aplicar.



(a) Sistema simulado en lazo cerrado



(b) Resultado de la simulación

Figura 4.2: Respuesta del sistema en simulación

Los resultados en la Figura 4.2b concuerdan: estando el robot inicialmente con un ángulo positivo ($\theta_0 = 0{,}2 \approx 11^o$), tendería a caer y, por tanto, continuarían creciendo θ y $\dot{\theta}$. La realimentación con valores positivos impulsaría al robot a moverse hacia delante y entonces vemos el crecimiento de x y $\dot{x}$. Se genera una sobrecompensación y el robot cae, hacia atrás ahora. Podemos asumir pues, que se suceden oscilaciones de valores muy altos en el ángulo. En la realidad no se llegaría

nunca a este extremo, ya que al crecer $\theta > \pi$, el robot estaría en el suelo, sin posibilidad de reparar ese ángulo independientemente de la fuerza que se aplique o el desplazamiento que se produzca. Además no se puede considerar el comportamiento de la simulación fiable en estos casos, ya que nuestro modelo se basa en que $\theta \approx 0$ en la aproximación lineal que se ha aplicado.

4.1.2. Sistema con control LQR

Para intentar obtener un sistema estable, se añadieron las constantes K_θ, $K_{\dot{\theta}}$, K_x y $K_{\dot{x}}$ para ponderar la suma de las salidas del sistema. Así se pretendía evitar la sobrecompensación que se produjo en el sistema realimentado.

Como se explicó en el punto 3.1, se tiene que encontrar primero las matrices $\mathbf{Q}$ y $\mathbf{R}$ con los valores adecuados.
Primero, para obtener el valor adecuado de $\mathbf{Q}$ hay que tener en cuenta que se debe minimizar, ante todo, el ángulo θ y la posición x. Por ello, inicialmente se prueba con $\mathbf{Q} = \mathbf{I}$, matriz identidad.

Se modificó la matriz $\mathbf{Q}$ y se compararon los resultados del controlador obteniendo los mejores resultados con el valor de la ecuación (4.2).

$$\mathbf{Q} = \begin{bmatrix} 50 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

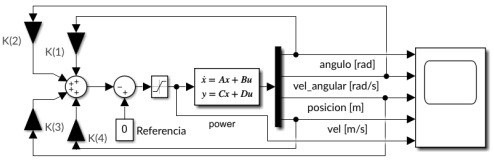
Por otro lado, para evitar picos muy altos de potencia, se tomaron dos medidas. Primero, se añadió un módulo de saturación que limitó $F(t) \in [-5, 5]\,(N)$. Con esto se pretende solucionar un problema del modelo teórico. Con el modelo teórico, si el robot comienza prácticamente tumbado, se le puede aplicar una fuerza infinita y conseguir con ello levantarlo. Esto, en la práctica, es inviable, ya que el motor del robot tiene sus límites. Por ello, para simular estos límites, se le añade un módulo de saturación.

La otra medida, es poner la $\mathbf{R} = 5$. Con eso se le da mayor valor a la fuerza en la función a minimizar, para así ahorrar gasto energético y evitar correcciones muy bruscas.

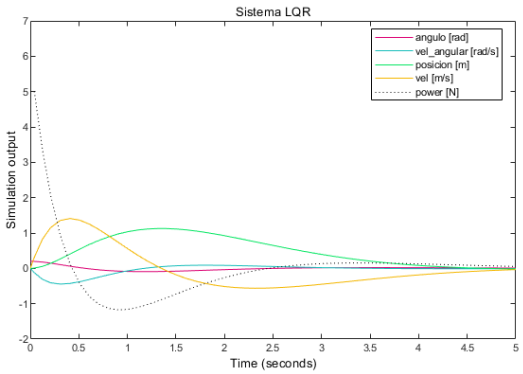
Con estos valores de $\mathbf{Q}$ y $\mathbf{R}$, se ejecutó el comando de MATLAB que se comentaba en el punto 3.1 y se obtuvo un vector $\mathbf{K}$.

$$\mathbf{K} = \begin{bmatrix} K_\theta \\ K_{\dot{\theta}} \\ K_x \\ K_{\dot{x}} \end{bmatrix} = \begin{bmatrix} -29{,}3035 \\ -12{,}0458 \\ -1{,}0000 \\ -2{,}1869 \end{bmatrix} \tag{4.3}$$

Con estos valores se obtiene un resultado que se puede observar en 4.3b. El ángulo comienza con una inclinación de 0,2 $rad \approx 11^o$ y la potencia se activa para compensar esa inclinación. Se genera un poco de sobrecompensación provocando una inclinación en la dirección contraria ($\theta < 0$), pero se acaba estabilizando en 0.



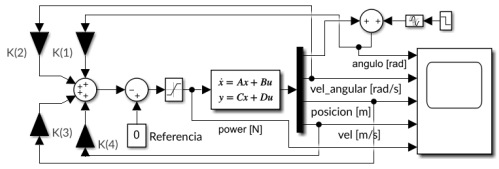
(a) Sistema simulado con controlador LQR



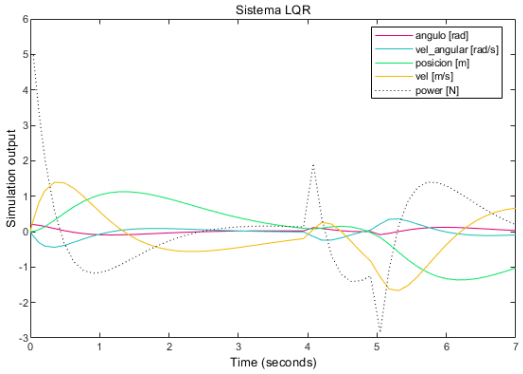
(b) Respuesta del sistema en simulación

Figura 4.3: Simulación del sistema con controlador LQR

Por otro lado, también se quiso simular una perturbación como si se empujara al robot una vez alcanzado el punto de equilibrio de la inclinación inicial. Para ello, en la figura 4.4a, se ha añadido a la señal de ángulo la suma de un escalón. Este escalón se ha retrasado 4 segundos, ya que se quería ver el resultado una vez se hubiese equilibrado, y no durante el equilibrado inicial. El empujón que se simula tiene como consecuencia que el robot se incline 0,1 $rad \approx 6^{o}$. En la simulación en 4.4b se observa que, entorno al segundo 4, el ángulo varía y, con ello, la potencia se dispara para compensarlo. Alrededor del segundo 5, el escalón vuelve a 0 y la potencia también ha de cambiar para corregir esa fuerza compensatoria. El valor que más tarda en corregirse es la posición del robot. Cabe entender que, frente a un empujón, se desplace considerablemente y tarde en volver a su posición inicial.



(a) Sistema simulado con controlador LQR y perturbación



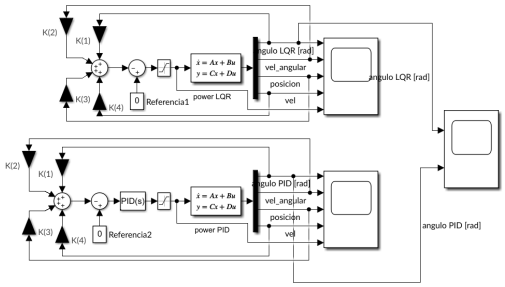
(b) Respuesta del sistema en simulación

Figura 4.4: Simulación del un sistema con controlador LQR con perturbación

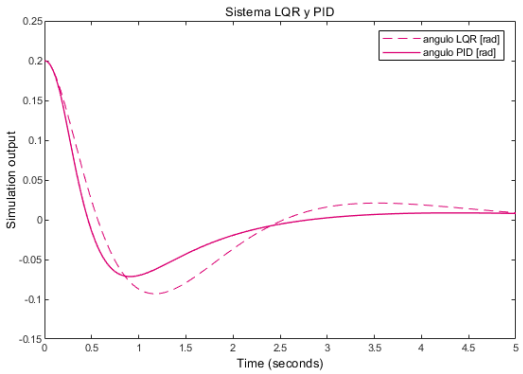
4.1.3. Simulación con control LQR y PID

Para terminar las simulaciones, se añadió un módulo PID que ponderase la señal "power" obtenida del controlador LQR. Como se explicó en 3.2, este módulo opera con el valor de la señal, su integral y su derivada para obtener un control mejorado.

En 4.5a se puede observar que se simulan ambos controles al mismo tiempo y que se comparan las salidas "ángulo" para ver qué mejoría produce este controlador. Los mejores resultados, que



(a) Sistema simulado



(b) Resultado de la simulación

Figura 4.5: Comparativa de las simulaciones de un sistema con control LQR y PID en Simulink

se pueden observar en 4.5b se obtuvieron con los siguientes valores:

$$\begin{bmatrix} K_p \\ K_i \\ K_d \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 0{,}01 \end{bmatrix} \tag{4.4}$$

Con esto se demuestra que, como se había anticipado, el problema tiene solución, aunque es inestable inicialmente. Y que el controlador PID es mejor que el controlador solo LQR.

Capítulo 5

Lenguajes de programación

Para el comienzo de la realización más práctica del proyecto se tuvo que elegir un lenguaje de programación. Se va a explicar brevemente la experiencia que se tuvo con cada lenguaje y las opciones que se valoraron para el proyecto.

5.1. EV3-G

Primero se probó el propio software de codificación diseñado por la empresa creadora del robot, el EV3-G. Para la codificación y el control del EV3, se puede descargar gratuitamente en la Página Oficial de Lego el programa Lego Mindstorm EV3. Para un aprendizaje inicial y valorar esta alternativa, se consultó Guía de uso de Lego Company de 2018 [21].

Este software está pensado para la programación en EV3 por los creadores y, por tanto, tiene bastantes funcionalidades con las que no cuentan otros. Por ejemplo, una amplia base de datos de gráficos y sonidos disponibles. También es gratuito y permite conectar fácilmente el robot al ordenador por USB o *bluetooth*.

Todas estas razones permitieron que se valorara como software a usar, pero se desechó la idea ya que está diseñado con una intención educativa y para introducir a los entornos de programación. Es muy intuitivo para un usuario principiante, pero tremendamente poco práctico para un programador avanzado. La programación se lleva a cabo a través de bloques de diferentes tipos (esperar, redondear, bucle, lectura de sensor, etc), estilo lenguaje Scratch.

Este paradigma de programación es práctico para las funciones exclusivas del robot (iniciar motores, reproducir sonidos, ect.), pero muy intrincado para funciones más generales (bucles, switchs, condicionales, etc.).

Se puede ver una muestra de este entorno de programación en la figura 5.1, con un ejemplo de un programa que escribiría “Hello world” en la pantalla del robot y reproduciría un sonido en un bucle infinito.

Figura 5.1: Código "Hello world" en Mindstorm EV3.

5.2. RobotC

RobotC es un lenguaje de programación basado en C que está específicamente diseñado para la programación en entornos como Lego Mindstorms. Es un lenguaje que se ejecuta con un *firmware* muy eficiente que permite una rapidez notable.

Se puede descargar en la Página Oficial, pero es una versión de prueba de 10 días. Para usarlo durante más tiempo se requiere obtener una licencia de $49 al año.

Esta es la razón por la que se desechó este lenguaje para el proyecto. Aunque, para un proyecto con visión comercial y a más largo recorrido, sí sería una opción a considerar.

No se llegó a profundizar más en si se requería otro sistema operativo o si se tenía que hacer algún cambio en el software del robot, aunque sería un punto a tener en cuenta.

5.3. EV3dev *Python*

EV3dev es un proyecto de código abierto que permite al usuario usar un sistema operativo basado en Debian inicializado desde una tarjeta SD. El código de este proyecto se puede encontrar en el repositorio GitHub y en la Página Oficial del proyecto se explica su uso y los lenguajes que soporta.

Que se instale el SO en una tarjeta SD es especialmente útil, ya que permite que se mantenga el propio robot intacto.

En este sistema operativo se pueden utilizar múltiples lenguajes como *Python*, Java, Go, C++, C, Prolog, JavaScript, Ruby, etc.

Se eligió *Python* al ser un lenguaje muy útil actualmente en la vida laboral, fácil de usar y al tener ciertas nociones básicas. Para la programación se utilizó el editor de código Microsoft Visual Studio Code y la extensión de EV3, que permite conectar y volcar el código en el robot.

Para la programación de *Python* enfocada al uso del robot se consultaron múltiples fuentes

Figura 5.2: Logo EV3dev obtenido de su página oficial.

[22, 23]. Se desarrolló un primer intento de código de equilibrado que se puede ver en el anexo A.2.

El principal problema que se tuvo para el uso de este lenguaje fue la lentitud. Aunque *Python* puede ser muy rápido con las librerías adecuadas, el lenguaje en sí no lo es tanto. Por ello, el tiempo de recorrido del bucle que se consiguió con la máxima optimización (sin usar librería externas) fue de 0,04 sec. Un problema, ya que el bucle para un adecuado funcionamiento no puede ser mayor que 0,01 sec. Por esta, y más razones, se considero que, aunque se había comenzado programando en este lenguaje, se tantearían otras opciones antes.

5.4. LeJOS Java

LeJOS es un *firmware* para el EV3 que incluye una máquina virtual de Java y, con ello, la posibilidad de programar en este lenguaje. También incluye un librería que permite comunicarse por *bluetooth* con el *firmware* original del EV3.

LeJOS ofrece características como: lenguaje orientado a objetos, threads, arrays, recursión, sincronización, excepciones, etc. Además, se cuenta con una documentación bastante extensa con la que comenzar a codificar [24].

Figura 5.3: EV3 iniciándose con leJOS.

Para su instalación, se procedió de forma similar a EV3dev, con una SD.

Éste fue el lenguaje en el que se realizó el proyecto. Las razones son que:

- El paradigma del lenguaje orientado a objetos sería muy práctico para una codificación más intuitiva y limpia.
- Es el lenguaje que más se ha utilizado en los estudios, y por tanto con el que se está más familiarizado, acelerando el aprendizaje del uso de EV3 y la programación en sí.
- Todas las fuentes consultadas tienden a usar este lenguaje (si no EV3-G o RobotC), lo que hace probable que sea el que más documentado esté y el más funcional.

Capítulo 6

Codificación

Una vez se decidió el algoritmo, el método, el programa y el lenguaje de programación, se procedió a iniciar el proceso de programar. Para simplificar la lectura, se va a separar este capítulo en las diferentes clases implementadas explicando, sin entrar en mucho detalle, las funcionalidades de cada una . El código completo del proyecto se puede encontrar en el repositorio GitHub[1]. También se pueden encontrar en ese mismo repositorio múltiples vídeos que muestran los resultados obtenidos.

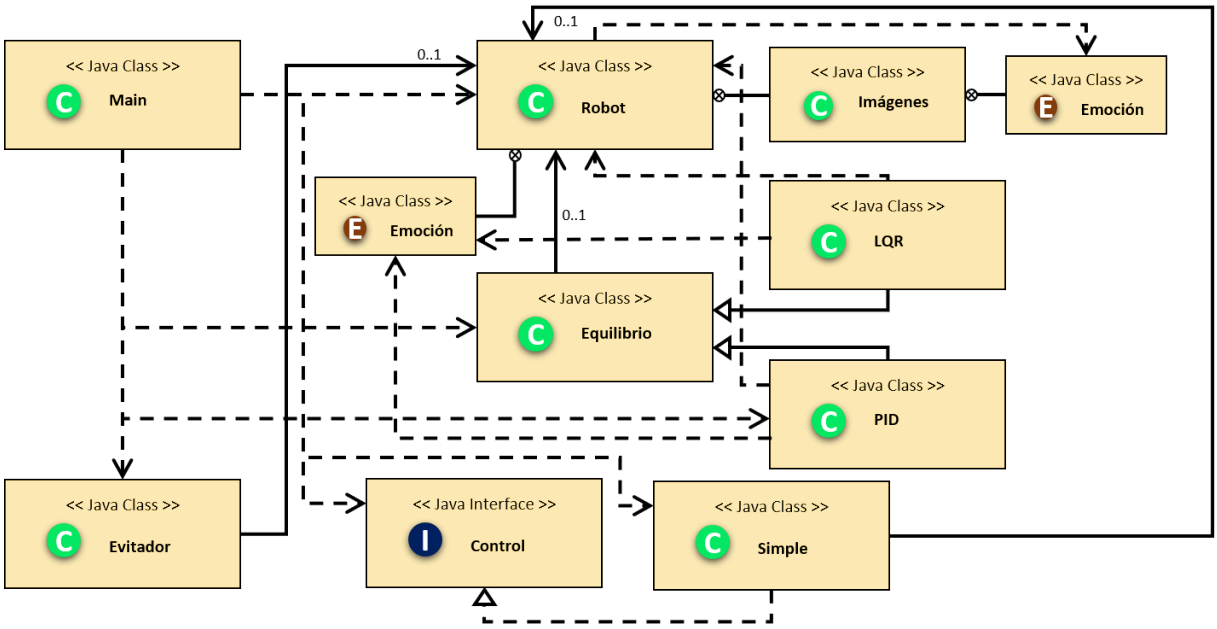


Figura 6.1: Diagrama *UML* del las clases del proyecto.

En la figura 6.1 se ve representadas las clases y las relaciones entre ellas siguiendo el estándar del Lenguaje Unificado de Modelado.

6.1. Main

El código comienza aquí y llamará a cada uno de los hilos o clases a ser ejecutadas. Es una clase bastante simple que no hace más que llamar a otras clases y comenzar otros hilos. Comienza

[1] https://github.com/vuvuxka/EV3_EGala

por llamar a la clase **Robot** (sección 6.2) para inicializar al EV3 y guardarla como atributo a proporcionar a la mayoría de las demás clases. Desde casi todas las demás clases se llamará a **Robot** para lecturas y escrituras.

A continuación, crea una instancia de una subclase de **Equilibrio** (sección 6.3), que se ejecutará en un hilo independiente (hereda de *Runnable*) y mantendrá al robot de pie.

Por otro lado, también se creará una instancia ejecutable que se dedicará a evitar posibles obstáculos y que se ejecutará en otro hilo distinto. Esta clase se llama **Evitador** (sección 6.4).

Por último, se creará una última clase que se ocupará de proporcionarle al robot instrucciones que hacer. Será una interfaz **Control** (sección 6.5) que sera implementado por distintas clases.

6.2. Robot

La clase **Robot** principalmente se ocupa de todo lo que relaciona al resto de las clases con el robot en sí. De esta forma, si se requiriese una reestructuración de los métodos asociados al robot (por una nueva versión o una librería mejorada), solo habría que hacer los cambios en esta clase.

Para cada uno de los sensores se tiene un atributo que será el encargado de leer ese sensor. Aparte, tenemos los siguientes atributos que caracterizarán el estado del robot:

Velocidad. Al cambiar este atributo, el robot incrementará su velocidad. Los valores adecuados son -50-50, con signo negativo para el retroceso.

Dirección. Al cambiar este atributo, se cambiará la dirección. Se define el valor como la intensidad con la que se efectúa el giro.

Stop. Indica si el robot debería de pararse. Es un método para sincronizar las paradas de todos los bucles.

Evitando. Indica si la clase **Evitador** está cambiando la dirección para evitar un obstáculo. Es la forma que tiene **Control** de no interferir.

6.2.1. Motores

int encoder(Motor m) devuelve a la llamada los *tacho counts* ($\approx rad$) del motor **m**.

void stop(Motor m) detiene el motor **m**.

void avance(Motor m, int vel) pone al motor **m** a la velocidad **vel**.

6.2.2. Giroscopio

double rate(int n) devuelve el valor del giroscopio en modo "velocidad angular" haciendo la media de **n** vueltas con 2 milisegundos de separación.

double angle() devuelve el valor del giroscopio en modo "ángulo".

6.2.3. Gráficos

Una de las posibilidades que tenía el EV3-G con las que no contaba leJOS era la biblioteca gráfica para reproducir en la pantalla. Con leJOS se pueden poner texto, y gráficos creados a partir de comandos muy simples (rectas, formas, etc.). Como se quería explorar la posibilidad de ponerle ojos expresivos al robot, se investigó una manera de paliar esta carencia.

Primero se requería imágenes libres de derechos para poder utilizar de forma legal, ya que las imágenes de EV3-G están protegidas por *copyright* de LEGO. Se encontró una base de datos de imágenes que imitaban a éstas pero creadas por usuarios y libres de *copyright*[2].

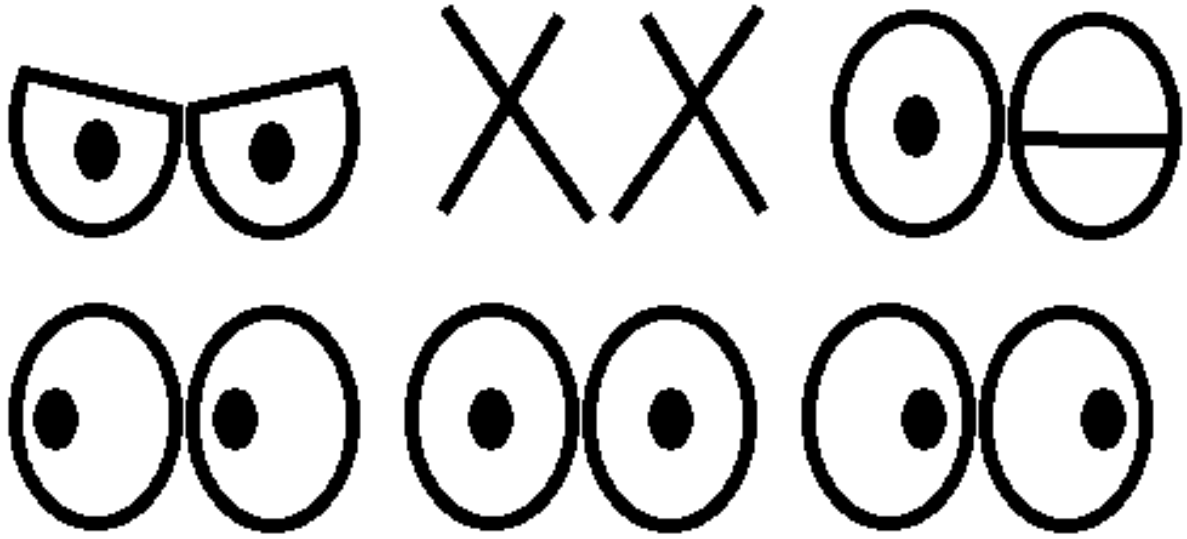


Figura 6.2: Imágenes obtenidas para los gráficos del robot

El problema es que, para poder trazar una imagen en la pantalla LCD del robot en leJOS, se debe utilizar la clase **Imagen**, que se construye con tres parámetros de entrada: *width*, *height* y *data*, un array de bytes que representa la figura. Por ello, se investigó y se encontró un *script*[3] de Python que transforma imágenes a arrays de bytes en Java. Así, se transformaron las imágenes de la Figura 6.2 y se introdujeron en la clase **Robot**.

El robot comienza el programa con la cara "neutra", cuando se produce un error de desequilibrio (el robot se cae o se coge) se cambia a "error". Si el robot se encuentra con un obstáculo que la clase **Evitador** tiene que esquivar, durante el proceso se pondrá la cara "enfadado". La implementación del interfaz **Control** también puede llamar a cambiar la cara dependiendo de las instrucciones.

6.2.4. Detector de Color

Uno de los sensores con los que cuenta el EV3 es un detector de color que se ha incorporado a la parte inferior del diseño que se ha construido. Este sensor tiene cuatro modos distintos que se especifican en la tabla 6.1.

Para la detección de color se comenzó con el modo *ColorID* que devuelve el color en forma de

[2] Repositorio GitHub: https://github.com/ev3dev/ev3dev/releases
[3] *Script* (https://www.pobot.org/Outil-de-generation-d-images-pour.html) creado por Eric P.

Color ID	Measures the color ID of a surface	Color ID	getColorIDMode()
Red	Measures the intensity of a reflected red light	N/A, Normalized to (0-1)	getRedMode()
RGB	Measures the RGB color of a surface	N/A, Normalized to (0-1)	getRGBMode()
Ambient	Measures the ambient light level	N/A, Normalized to (0-1)	getAmbientMode()

Cuadro 6.1: Modos del sensor de color [24].

valor de un enumerado con quince colores distintos. Se experimentó que el detector era bastante poco certero y confundía los colores en muchos casos. Consideraba la mayoría de los colores iguales al rojo o al negro, y había colores del enumerado que nunca detectaba.

Por ello se planteó la posibilidad de utilizar el modo *RGB* y crear un método en la clase **Robot** que descifrara los colores básicos necesarios con el código RGB obtenido por el método.

Detector RGB

RGB, de las siglas en inglés de rojo, verde y azul, es un modelo de color aditivo basado en los colores primarios de la luz. La forma estándar de representación es como un vector c con tres componentes enteras pertenecientes $\{0, \ldots, 255\}$. Cada uno de los componentes de ese vector representa el valor de ese color, y juntando los tres se obtiene un color específico. El 0 representa la ausencia de luz, y el valor máximo, 255, que el componente de ese color primario es máximo. [25]

En el caso del sensor *RGB* se han normalizado estos valores y se obtiene $\hat{c} \in \mathbb{S}_\infty$. Con $\mathbb{S}_\infty$ la parte positiva de la esfera de radio unidad en norma $\|\cdot\|_\infty$. Se representa como un cubo en $\mathbb{R}^3$ con centro $o = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$.

Se ha decidido centrarse en clasificar solo los colores representativos de la base ortogonal $\mathcal{B} = \{e_1, e_2, e_3\}$ y las sumas simples de éstos. Es decir, se han clasificado los vectores según cercanía a los vectores del conjunto $\mathcal{C} = \{\vec{0}, e_1, e_2, e_1 + e_2, e_3, e_1 + e_3, e_2 + e_3, e_1 + e_2 + e_3\} := \{\hat{e}_1, \ldots, \hat{e}_8\}$. También se puede entender como los valores binarios del 0 al 7, pero como se van a tratar como vectores, se prefiere conservar la notación vectorial.

En la tabla 6.2 se pueden ver los vectores representantes y el color asignado, y en la figura 6.3 la representación gráfica de $\mathbb{C}$.

(0,0,0)	(0,0,0)	Negro
(0,0,255)	(0,0,1)	Azul
(0,255,0)	(0,1,0)	Verde
(0,255,255)	(0,1,1)	Azul
(255,0,0)	(1,0,0)	Rojo
(255,0,255)	(1,0,1)	Rosa
(255,255,0)	(1,1,0)	Amarillo
(255,255,255)	(1,1,1)	Blanco

Cuadro 6.2: Separación de colores según $\mathcal{C}$.

Para la selección del color asignado a un vector c se utiliza la función representado en la ecuación (6.1). Se toma el representante que minimiza el cuadrado de la norma de la diferencia entre este

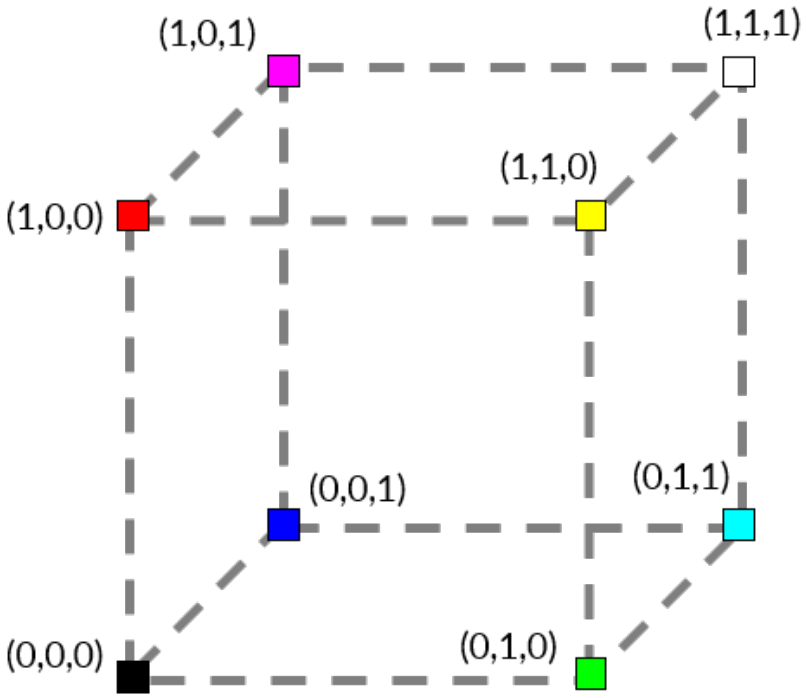


Figura 6.3: Representación gráfica de $\mathbb{S}_\infty$ con los colores.

y el vector c. Es decir, el algoritmo de k-vecinos con $k = 1$. [25]

$$\begin{array}{rcccl} \mathbb{S}_\infty & \longrightarrow & \mathcal{C} & \longrightarrow_{\text{tabla 6.2}} & \text{Colores} \\ c & \longrightarrow & \{\hat{e}_i : min\{\|c - \hat{e}_i\|_2^2 : \hat{e}_i \in \mathcal{C}\}\} & \longrightarrow_{\text{tabla 6.2}} & \text{color} \end{array} \tag{6.1}$$

Resultados

Para probar la validez del detector implementado se construyó una rejilla con distintos colores y se compró los resultados en dos casos: un entorno totalmente iluminado y un entorno a oscuras. La rejilla se puede ver en la imagen 6.4

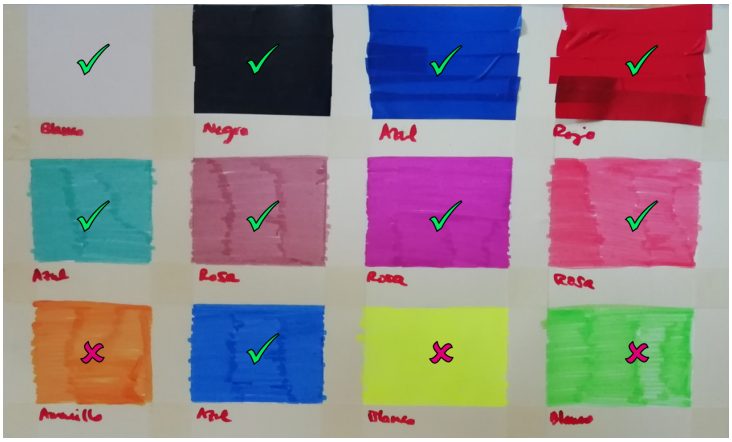


Figura 6.4: Prueba de la validez del detector de color en entorno iluminado y sin iluminar.

Se constató que los resultados para distinguir los colores negro, rojo, blanco, azul y rosa era bastante prometedores. En el entorno oscuro los resultados fueron los mismos, pudiendo considerar que la valoración no es relativa a la iluminación del entorno.

6.3. Equilibrio

Para poder cambiar fácilmente entre versiones con distintos controladores, se creó una clase abstracta llamada **Equilibrio** de la que heredan las clases que ejecuten los distintos controladores. Es una clase del tipo *Runnable* para que se ejecute en un hilo aparte que sólo gestionará el equilibrio.

Se obtuvieron dos controladores viables, como era la intención inicial.

6.3.1. Aplicación del modelo matemático

Aunque el curso lógico del proyecto es aplicar el modelo matemático que se ha calculado inicialmente, no ha sido viable. Se debe a que las unidades que utiliza el robot para la obtención de datos son muy distintas a las que se usaron inicialmente en el modelo. Podemos ver una comparativa entre ambas en la tabla 6.3. Las unidades de ángulo, la velocidad angular, la posición y la velocidad no suponen un problema para la aplicación del modelo: las conversiones son bastante simples e incluso en el sistema se usa la conversión radianes - metros. El problema se encuentra cuando al transformar la fuerza. Esta conversión es crítica de cara a la validez del modelo y requiere conocimientos de parámetros internos del motor. Además de esto, el porcentaje de vatios depende de la batería con la que cuenta el robot.

	Unidades del modelo	Unidades del sistema
Ángulo (θ)	radianes	grados
Velocidad angular ($\dot{\theta}$)	radianes/s	grados/s
Posición (x)	metros	*tacho counts*[1]
Velocidad ($\dot{x}$)	metros/s	*tacho counts*/s
Fuerza (F)	Newtons	vatios (%)

[1] *tacho counts* son equivalentes a grados hexadecimales.

Cuadro 6.3: Relación entre las unidades del modelo y del sistema

El modelo sirvió para comprender las necesidades y demostrar la existencia de una solución al problema. Para la obtención de las constantes aplicadas al sistema se recurrió a antecedentes anteriores al problema y la bibliografía encontrada. Ajustando estos valores empíricamente al sistema definido.

6.3.2. LQR

La implementación de esta clase se basó en un trabajo anterior de equilibrio del robot tipo *Gyroboy* (ver Figura 6.5)[15]. Esta consulta bibliográfica es muy útil para considerar la estructura del algoritmo, aunque el modelo *Gyroboy* es totalmente distinto del robot que se ha construido y requirió de una adaptación a la constitución del robot.

El código comienza con una inicialización del ángulo: se toma la medida del ángulo 20 veces, con separaciones de 2 milisegundos, y se considera θ_0 a la media de todas las medidas.

Una diferencia entre este controlador y el controlador PID será la forma de obtener el tiempo de bucle. En el controlador PID se fija un dt y se opta por esperar siempre a que el bucle termine en

Figura 6.5: Modelo Gyroboy [15]

ese tiempo. Se comprobó empiricamente que, incluso en las situaciones más críticas, el tiempo de ejecución del bucle no superaba o superaba por una cantidad no considerable el tiempo fijado. En cambio, en el controlador LQR utiliza como tiempo derivativo el tiempo que se empleó en el bucle anterior. Por ello, lo primero que se realiza al entrar en el bucle infinito es fijar el tiempo actual y restarlo a la medida del bucle anterior.

Después se obtiene $\dot{\theta}$ (deg) con la función de la clase **Robot** (sección 6.2.2) y se calcula θ (deg/s) con la siguiente fórmula, considerando n como el número de vuelta en el que se encuentra.

$$\theta_{n+1} = \theta_n + \dot{\theta}_{n+1} dt \tag{6.2}$$

Se obtiene el valor actual de los motores en tacho counts (m_n^d, el valor del motor derecho y m_n^i, el izquierdo) y se define s_n como la suma de los valores tomados de cada motor: $s_n = m_n^d + m_n^i$. Por otro lado, también se adopta la notación $\bigtriangledown s_n$ para definir la diferencia regresiva en s_n. Es decir, $\bigtriangledown s_n = s_n - s_{n-1}$.

$$x_{n+1} = x_n + \bigtriangledown s_n \tag{6.3}$$

$$\dot{x}_{n+1} = \frac{\sum_{i=1}^{N} (\bigtriangledown s_{n-i})}{N dt} \tag{6.4}$$

En la práctica, se ha definido $N = 4$ en la ecuación (6.4).

Se define $\bar{x}_n$ como la posición de referencia. Las clases **Evitador** 6.4 y **Control** 6.5 editan la variable velocidad del robot para dirigir el comportamiento del robot y con ella se obtiene la posición a la que se quiere mover el robot. Se calcula el error considerando la posición como $x_n + \bar{x}_n$ introduciendo un desfase de la distancia que se quiere avanzar.

Una vez calculado el error con la ecuación (6.5).

$$E_n = K_\theta \theta_n + K_{\dot{\theta}} \dot{\theta}_n + K_x (x_n + \bar{x}_n) + K_{\dot{x}} \dot{x}_n \tag{6.5}$$

Por último, se devuelve al motor el valor $E_n + d_n^d$ al motor derecho y $E_n - d_n^i$ al izquierdo. Estos valores d_n se obtienen de la clase **Robot** y representan el giro que se quiere que produzca el robot.
Las constantes del controlador LQR que se obtuvieron empiricamente son:

$$\begin{bmatrix} K_\theta \\ K_{\dot{\theta}} \\ K_x \\ K_{\dot{x}} \end{bmatrix} = \begin{bmatrix} 21{,}6 \\ 1{,}1520 \\ 0{,}1728 \\ 0{,}1152 \end{bmatrix} \tag{6.6}$$

Resultados

El controlador obtenido presenta ciertas fallos:

- Es muy sensible a una inicialización correcta. Si no se sujeta bien el robot vertical en el inicio puede tener un balanceo marcado y acabar por caer.
- La posición del robot oscila ligeramente.
- El terreno no liso supone un reto adicional para el robot. El robot es capaz de avanzar entre baldosas con uniones de 0.5 cm de longitud y 3 mm de profundidad, pero la presencia de estos escalones provoca oscilaciones de mayor intensidad. Escalones de mayor tamaño pueden provocar oscilaciones bruscas e incluso caídas.
- No acepta muy bien velocidades y direcciones. El control de dirección es muy rudimentario y tiende a caerse cuando se intenta girar mucho.
- No es muy receptivo a perturbaciones externas como empujones.

6.3.3. PID

Para la realización de esta clase, se tuvo en cuenta el código parecido en EV3-G para el modelo BALANC3R [13].

La inicialización del ángulo y la obtención de la velocidad angular se realiza por el mismo método que en la clase **LQR**. La única diferencia es que se toma la velocidad angular con cinco muestras en vez de solo una para evitar que picos afecten a la medida.

La obtención de la velocidad también es similar, aunque en este caso definimos $\bar{s}_n = \frac{m_n^d + m_n^i}{2}$, un valor medio entre el avance que ha tenido cada motor. Con los giros que se puedan producir, esta opción parece más consistente.

Además, también se decidió actualizar el método de obtención de la velocidad. En esta clase, en vez de trabajar con la diferencia regresiva ($\bigtriangledown s_n$), se trabajó con los valores absolutos de $\bar{s}_n$. De esta forma se puede llevar un historial más largo de los valores sin que suponga un gran coste en tiempo el hallar la media. Y, para el calculo de la velocidad, se estima con la diferencia en más bucles de distancia y obteniendo así un valor más seguro y estable.

$$\dot{x} = (\frac{s_n - s_{n-M})}{Ndt} \tag{6.7}$$

Los valores de x y $\dot{x}$ se transforman a metros, ya que se pensó que sería más práctico en un uso futuro del robot con más funcionalidades.

La obtención de E_n se mantiene con el uso de la ecuación (6.5), pero con distintas constantes adaptadas a metros y al controlador PID:

$$\begin{bmatrix} K_\theta \\ K_{\dot{\theta}} \\ K_x \\ K_{\dot{x}} \end{bmatrix} = \begin{bmatrix} 25 \\ 1{,}3 \\ 350 \\ 75 \end{bmatrix} \tag{6.8}$$

Para la obtención del valor derivativo e integral del error se aplican las siguientes fórmulas, siendo E_n^p el error proporcional, E_n^i el error integral y E_n^d el error derivativo.

$$\begin{aligned} E_n^p &= E_n \\ E_n^i &= E_{n-1}^i + E_n dt \\ E_n^d &= \tfrac{E_n - E_{n-1}}{dt} \end{aligned} \tag{6.9}$$

Se calcula el error a aplicar al motor como $\hat{E_n} = E_n^p K_p + E_n^i K_i + E_n^d K_d$ con las constantes de (6.10).

$$\begin{bmatrix} K_p \\ K_i \\ K_d \end{bmatrix} = \begin{bmatrix} 0{,}4 \\ 14 \\ 0{,}005 \end{bmatrix} \tag{6.10}$$

Para mejorar la aplicación de la dirección, se considero el parámetro "dirección" (d_n de ahora en adelante) como la brusquedad del giro, no la amplitud. Es decir, se imita el uso de un volante de coche. Considerar la amplitud deseada sería otro enfoque interesante, pero se decidió optar por éste al plantearse un futuro control con *joystick* o volante.

Este valor, d_n, será el que clases como **Evitador** o **Controlador** escriban para conseguir que EV3 gire. Con este valor y la ecuación (6.11) se obtendrá $\hat{d}_n$ como el valor a aplicar a los motores. Se define $c_n = m_n^i - m_n^d$ como el giro actual que tienen las ruedas.

$$\hat{d}_{n+1} = \begin{cases} c_{n+1} - c_n, & \text{si } d_n = d_{n-1} = 0 \\ 0, & \text{si } d_n = 0 \neq d_{n-1} \\ -0{,}5 d_n, & \text{si no} \end{cases} \tag{6.11}$$

La primera parte de la ecuación (6.11) pretende corregir errores que se puedan producir en la dirección cuando el robot va recto. Por ejemplo, por irregularidades en el suelo o en los motores.

Una vez obtenido este valor, se aplica a los motores añadiendo $\hat{d}_n$ sumando en el caso del motor derecho y restando en el del izquierdo, al igual que en la clase **LQR**. De esta forma, con valores positivos el robot gira en la dirección de las agujas del reloj.

6.3.4. Resultados

El controlador es notablemente mejor que el obtenido en **LQR**.

- Cuando se le pidió a un usuario no entrenado en el modelo iniciarlo dadas las siguientes instrucciones: *Encender, sostener vertical, soltar después de los 3 pitidos*, el robot inició correctamente 3 de 3 veces.
- La posición es bastante estable, aunque no perfecta. No oscila en equilibrio no perturbado, pero ante una perturbación recula y no vuelve exactamente a su posición inicial.
- El robot no da sacudidas muy pronunciadas.
- Acepta mejor giros.
- Es más receptivo a perturbaciones externas como empujones.

Aunque, por otro lado, a mejorar de cara al futuro convendría un direccionamiento absoluto en grados.

6.4. Evitador

El objetivo de la clase **Evitador** es detectar los obstáculos por medio del sensor de ultrasonidos y corregir la dirección temporalmente para evitarlo. Para ello, se recibe una señal de la clase **Robot** que representa la distancia en metros del obstáculo más cercano (b_m, siendo m el número de vuelta del bucle infinito de la clase **Evitador**) que observa el robot. En caso de no haber ningún obstáculo (o estar tapando el sensor), el resultado el ∞.

El método para corregir es bastante simple. Se mide la distancia b_m y si se se obtiene ∞, no se responde de ninguna manera. Si, en cambio, $B_{\text{crítica}} < b_m < B_{\text{ok}}$, se evita el obstáculo generando un giro no muy pronunciado a la derecha (y haciendo saber al controlador que en ese momento se está evitando un obstáculo, que no se actualice el valor de d_n). Por último, si $b_m < B_{\text{crítica}}$, se evita el obstáculo echando marcha atrás.

6.5. Control

El interfaz **Control** es el ejecutador de lo que hará el EV3 mientras no esté evitando obstáculos. Es decir, si se quisiera que avanzase en una única dirección, o que siguiese una línea, o que girase, o incluso en un futuro que se conectara a una señal *bluetooth* y fuese manejado a distancia.

A diferencia de **Evitador** y **Equilibrio**, no es una clase herencia de *Runnable*, ya que continúa con el hilo del **Main** para su ejecución.

Simple La clase **Simple** implementa el interfaz **Control** con las instrucciones de ir recto (siempre que no se encuentre un obstáculo) y con velocidad constante.

Recorrido La clase **Recorrido** implementa el interfaz **Control** con instrucciones para seguir un recorrido. El robot tiene que empezar recto con una velocidad constante. Cuando detecte el color blanco deberá de hacer un giro cambiando de dirección para evitarlo. Al encontrar el color rojo, realizar una vuelta completa. Y, por último, al encontrar el color verde, reproducir un sonido.

Capítulo 7

Conclusiones y futuros proyectos

La investigación está comprendida en un estudio matemático de las fuerzas, la construcción de un sistema que simule el robot, la simulación de este sistema para comprender su estabilidad y la aplicación de lo aprendido a un robot EV3. En este capítulo se estudian los métodos y las conclusiones que derivan del trabajo realizado. Y, posibles trabajos que realizar tomando como punto de partida el final de éste.

7.1. Conclusiones

El estudio del modelo matemático fue extenso y completo. Se cumplieron las expectativas iniciales de esta parte del trabajo e incluso se exploraron tópicos que nos se consideraron inicialmente (como el tipo de fricción que se aplica a un móvil a ruedas con motor, o el paso de un sistema de estados a una función de transferencia).

El modelo del péndulo vertical es una cuestión con una bibliografía extensa y que se ha estudia previamente muy en profundidad. Como posibilidad de ampliación se plantearía un repaso bibliográfico a un modelo no lineal y las diferencias de exactitud entre el modelo lineal y el no lineal.

Con respecto a la simulación del sistema, se consideró que fue fundamental para la comprensión del proyecto. Se ha advertido que esta fase fue la más significativa para fijar los conceptos y para comprender la aplicación que se hizo de ellos. Por ello, quizás, como sugerencia de mejora, en el futuro se extendería el tiempo dedicado a esta etapa en la planificación.

Uno de los impedimentos más grande a los que se enfrentó el proyecto es la conversión del modelo matemático al físico. El problema es el método de trato de los motores y sensores por parte de EV3 en sí. Se puede sacar en conclusión que aplicar un modelo matemático a un *hardware* específico requiere un estudio mucho mayor del propio robot, unos conocimientos físicos avanzados y más tiempo para realizar el proyecto. Una de las conclusiones más significativas del proyecto es la importancia del lenguaje y el entorno para la obtención de distintos resultados. Que se comenzara codificando en *python* no se considera un desacierto, ya he condujo a un estudio de los tiempos de ejecución y de la optimización de código en este lenguaje.

Una de las conclusiones principales es que el estudio del lenguaje que se va a usar debería de ser

una fase a incluir en todos los proyectos, junto con una comparativa y unas sólidas razones para la elección.

El código resultante del proyecto fue un código bien estructurado y que a la altura de las consideraciones iniciales. Se estima un acierto la construcción por módulos e interfaces, ya que ha facilitado mucho la depuración y la construcción. El proyecto está preparado para un crecimiento futuro e incremental con nuevas funcionalidades.

7.2. Futuros proyectos

- Estudio avanzado de la aplicación de un modelo físico a un robot EV3. Es un tema que no está presente en la bibliografía consultada y que se considera que tiene gran importancia.
- Estudio de los tiempos de ejecución de *python* y de la viabilidad de usar librería externas en la programación en EV3 que aclaren si es viable el proyecto.
- Ampliación de las funciones del robot para combinar el equilibrio con funciones más avanzadas (como una exploración basada en inteligencia artificial más compleja).

Bibliografía

[1] Comité Español de Automática (CEA), "IX edición del Concurso PRODEL de Control Inteligente," 2019.

[2] J. K. Roberge, *The mechanical seal.* Trabajo de fin de grado, Massachusetts Institute of Technology, 1960.

[3] K. H. Lundberg and T. W. Barton, *History of Inverted-Pendulum Systems*, vol. 42. IFAC, 2010.

[4] M. Bugeja, "Non-linear swing-up and stabilizing control of an inverted pendulum system," *IEEE Region 8 EUROCON 2003: Computer as a Tool - Proceedings*, vol. B, pp. 437–441, 2003.

[5] H. Kwakernaak and R. Sivan, "Linear Optimal Control Systems," *IEEE Transactions on Automatic Control*, vol. 19, pp. 631–632, oct 1974.

[6] S. Hassenplug, "Steve's LegWay," 2002.

[7] Michael McNally (Lego Americas), "What's NXT? LEGO Group Unveils LEGO(R) MINDSTORMS(TM) NXT Robotics Toolset at Consumer Electronics Show," 2006.

[8] P. P. Hurbain, "Get up, NXTway!," 2007.

[9] R. Watanabe, *Motion Control of NXTway(LEGO Segway) Control Experiments with LEGO Mindstorms NXT.* PhD thesis, Waseda University, 2007.

[10] Y. Yamamoto, "NXTway-GS Model-Based Design," 2008.

[11] A. Hernández Largacha, M. Martínez Legaspi, and J. Martín Peláez, *Control inteligente de péndulo invertido.* PhD thesis, Universidad Complutense de Madrid, 2013.

[12] A. U. o. M. Sherrard and A. U. o. M. Rhodes, "Comparison of the LEGO Mindstorms NXT and EV3 Robotics Education Platforms," *Extension Journal*, vol. 52, no. #5TOT9, 2014.

[13] L. Valk, "Tutorial: Self-Balancing EV3 Robot," 2014.

[14] P. A. Lora Thola, "Estudio e implementación de un controlador para un robot tipo Segway y de los algoritmos que lo capacitan para el seguimiento de trayectorias desconocidas," *Máster Universitario de Ingeniería de Sistemas Automáticos y Electrónica Industrial*, 2015.

[15] C. Bjørn Klint, *GyroBoy – a self-balancing robot programmed in JAVA with leJOS EV3.* PhD thesis, University of Denmark (DTU), 2017.

[16] S. A. Campbell, S. Crawford, and K. Morris, "Friction and the Inverted Pendulum Stabilization Problem," *Journal of Dynamic Systems, Measurement, and Control*, vol. 130, no. 5, p. 054502, 2008.

[17] C. H. Holbrow, J. N. Lloyd, J. C. Amato, E. Galvez, and M. E. Parks, *Modern Introductory Physics*. New York, NY: Springer New York, 2010.

[18] K. Ogata, *Ingeniería de control moderna 4ED*. Pearson, 2003.

[19] Control Tutorials for MATLAB & Simulink, "Inverted pendulum: State-space methods for controller design," 2012.

[20] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal control*. New Jersey (USA): John Wiley & Sons, Inc, 2012.

[21] Lego Mindstorm, "Guía de uso," tech. rep., 2018.

[22] N. Ward, "EV3 python," 2018.

[23] N. Ward, "EV3 Basic."

[24] EV3 LeJOS Developers Team, "Documentación LeJOS," 2013.

[25] Yongmei Cai and LinLin Zhang, "Average color vector algorithm in color recognition based on a RGB space," in *2012 IEEE 14th International Conference on Communication Technology*, pp. 1043–1047, IEEE, nov 2012.

Apéndice A

Código

A.1. Código del sistema (MATLAB)

```
%%
clc
clear
close all

%% Parametros
M = 0.7; % masa del carro
m = 0.027; % masa del pendulo
g = 9.81; % gravedad
L = 0.145; % longitud del pendulo
k = 0.1; % friccion del carro
I = 0.006; % friccion en la union

%% Sistema
% todas las unidades de salida estan en radianes
p1 =     (M + m)/(I*(M + m) + L^2*m*M);
p2 = (I + L^2*m)/(I*(M + m + L^2*m*M));
A = [                            0, 1, 0,                    0;
                       L*m*g*p1, 0, 0, (L*m*k*p1)/(M + m);
                            0, 0, 0,                    1;
     (-(L*m)^2*g*p2)/(I + L^2*m), 0, 0,               -k*p2];
B = [0; (-L*m*p1)/(M + m); 0; p2];
C = eye(4);
D = zeros(4,1);
x0 = [0.2; 0; 0; 0]; % empieza quieto y girado 11 grados

polos = eig(A);
```

A.2. Código EV3dev en Python

```
def equilibrar():
    ini.motorRight.reset()
    g = ini.Gyro.rate # Velocidad Angular del Giroscopio (grados/s)
    time.sleep(0.002)
    g = g + ini.Gyro.rate

    ini.dtheta = g/2.0 - ini.offset_gyro # Velocidad Angular (grados/s)
    ini.offset_gyro = ini.offset_gyro*0.999 + (0.001*(ini.dtheta + ini.
       offset_gyro)) # Actualizamos el offset
    ini.theta = ini.theta + ini.dtheta*ini.dt # Angulo (grados)
    ini.theta = ini.theta*0.999 - ini.theta*0.001

    ini.n = ini.n + 1
    if ini.n == ini.n_max:
        ini.n = 0
    ini.xdes = 0
    ini.x = ini.motorLeft.position + ini.motorRight.position # Posicion
       (deg)
    ini.n_ant = ini.n + 1
    if ini.n_ant == ini.n_max:
        ini.n_ant = 0
    ini.encoder[ini.n] = ini.x # Posicion (rotaciones)
    average = 0
    for i in range(ini.n_max):
        average = average + ini.encoder[i]
    average = average / ini.n_max
    ini.dx = average / ini.dt #Posicion

    ini.e = (k1*ini.theta + k2*ini.dtheta + k3*ini.x + k4*ini.dx)
    ini.de_dt = (ini.e - ini.e_prev)/ini.dt
    ini.iedt = ini.iedt + ini.e*ini.dt
    ini.e_prev = ini.e
    ini.rot_prev = ini.rotacion

    if ini.e > 1050:
        ini.e = 1050
    elif ini.e < -1050:
        ini.e = -1050
    v = int(ini.e/ini.motorLeft.max_speed*100)
    ini.motorLeft.on(speed=SpeedDPS(ini.e))
    ini.motorRight.on(speed=SpeedDPS(ini.e))
```