

UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA:

INGENIERÍA ELECTRÓNICA

Trabajo de titulación previo a la obtención de:

INGENIERA ELECTRÓNICA

TEMA:

**DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR ÓPTIMO LQG,
PARA UN SISTEMA DE PÉNDULO INVERTIDO APLICADO EN UN
EQUIPO LEGO MINDSTORMS.**

AUTORA:

DIANA KARINA SOLÓRZANO PEÑAFIEL

TUTOR:

CARLOS GERMÁN PILLAJO ANGOS

Quito, julio 2018

CESIÓN DE DERECHOS DE AUTOR

Yo, Diana Karina Solórzano Peñafiel, con documento de identificación N° 0604165753, manifiesto mi voluntad y cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autora del trabajo de titulación intitulado: “DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR ÓPTIMO LQG, PARA UN SISTEMA DE PÉNDULO INVERTIDO APLICADO EN UN EQUIPO LEGO MINDSTORMS”, mismo que ha sido desarrollado para optar por el título de Ingeniera Electrónica, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en mi condición de autora me reservo los derechos morales de la obra antes citada. En concordancia, suscribo este documento en el momento que hago la entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Politécnica Salesiana.



Diana Karina Solórzano Peñafiel

0604165753

Quito, julio 2018

DECLARATORIA DE COAUTORÍA DEL DOCENTE TUTOR

Yo declaro que bajo mi dirección y asesoría fue desarrollado el trabajo de titulación, “DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR ÓPTIMO LQG, PARA UN SISTEMA DE PÉNDULO INVERTIDO APLICADO EN UN EQUIPO LEGO MINDSTORMS” realizado por Diana Karina Solórzano Peñafiel, obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana, para ser considerados como trabajo final de titulación.

Quito, julio 2018

A handwritten signature in blue ink, consisting of several overlapping loops and lines, positioned over a horizontal line.

Carlos Germán Pillajo Angos

1709255119

DEDICATORIA

Dedico este trabajo a mi abuelita Angelina porque sé que todos los días guía mis pasos desde el cielo, dedico también a mi madre Nieves Peñafiel por su paciencia y el cariño brindado durante mi vida universitaria, a mi padre Alberto Solórzano por darme su ejemplo de superación y motivarme a ser mejor persona.

Dedico también a mis hermanos que han estado en los momentos difíciles, a mis sobrinas Jholy, Mela y Aitana porque siempre han sido mi felicidad y motivación más grande, a Ricardo y su familia por todo el ánimo y fuerzas que me han brindado para la culminación de este proyecto y de mi carrera.

AGRADECIMIENTO

Agradezco a Dios por haberme permitido culminar con esta etapa más de mi vida, a mis padres por todo el apoyo brindado durante este tiempo. Agradezco a la Universidad Politécnica Salesiana, a todos los docentes por sus enseñanzas y su calidad moral.

Agradezco a mi tutor MSc Carlos Pillajo por su ayuda para la culminación de este proyecto, siempre guiándome de la mejor manera con sus conocimientos y ánimos para salir adelante.

Agradezco a mis amigas y amigos que conocí durante toda mi carrera y que han estado conmigo dándome su apoyo, alentándome para lograr con mí objetivo, muchos éxitos y les deseo lo mejor del mundo. Agradezco a Ricardo por ser parte importante de mi superación personal, gracias por tu cariño, consejos y ánimos para ser mejor persona y salir adelante con mis sueños.

CONTENIDO

CONTENIDO	v
ÍNDICE DE TABLAS	vii
ÍNDICE DE FIGURAS	viii
ABSTRACT	x
INTRODUCCIÓN	xi
CAPÍTULO 1	1
ANTECEDENTES.....	1
1.1 Planteamiento del problema	1
1.2 Justificación del proyecto	1
1.3 Objetivos	2
1.3.1 Objetivo General	2
1.3.2 Objetivos específicos	2
CAPÍTULO 2	3
MARCO TEÓRICO.....	3
2.1 Péndulo Invertido	3
2.2 Lego Mindstorms NXT	4
2.2.1 Sensor Giroscopio Hitechnic.....	5
2.2.2 Servo Motores y Encoders	5
2.3 Sistema MIMO (Multi-Input Multi-Output)	6
2.4 Representación en espacio de estados	7
2.5 Control Proporcional-Integral-Derivativo (PID).....	8
2.7 Control Lineal Cuadrático Gaussiano - LQG.....	9
2.7.1 Ventajas y desventajas del control LQG.....	11
2.8 Regulador Cuadrático Lineal - LQR	11
2.9 Filtro de Kalman	12
2.10 Índices de rendimiento IAE e ISE.....	14

CAPÍTULO 3	15
DISEÑO E IMPLEMENTACIÓN DE LOS CONTROLADORES	15
3.1 Modelo de la dinámica del péndulo invertido sobre dos ruedas	15
3.2 Variables en espacio de estados del modelo	22
3.3 Diseño del controlador PID	24
3.4 Diseño del controlador LQG	33
CAPÍTULO 4	40
PRUEBAS Y RESULTADOS	40
4.1 Prueba 1: Control de posición mediante los controladores PID y LQG en simulación.	40
4.2 Prueba 2: Control de ángulo mediante los controladores PID y LQG en simulación	42
4.3 Prueba 3: Control de ángulo mediante los controladores PID y LQG en el equipo real	43
4.4 Prueba 4: Control de posición mediante los controladores PID y LQG en el equipo real	44
4.5 Prueba 5: Gráficas de índices de rendimiento IAE para el control PID y función de costo para el control LQG.	45
CONCLUSIONES	48
RECOMENDACIONES	49
CITAS BIBLIOGRÁFICAS	50
ANEXO I	53
ANEXO II	54
ANEXO III	55
ANEXO IV	61

ÍNDICE DE TABLAS

Tabla 2.1: Constantes del controlador PID	8
Tabla 3.1: Descripción de parámetros físicos del equipo Lego Mindstorms	17
Tabla 3.2 Valores de las constantes PID	27
Tabla 3.3: Valores de constantes PID mediante algoritmos genéticos	28
Tabla 3.4: Valores de implementación del PID	31
Tabla 4.1: Valores obtenidos del sistema PID y LQG posición en simulación	41
Tabla 4.2: Índices de rendimiento de los controladores.....	41
Tabla 4.3: Valores del sistema PID y LQG ángulo en simulación	42
Tabla 4.4: Índices de rendimiento del control de ángulo en simulación.....	42
Tabla 4.5: Valores de los controladores PID y LQG de ángulo en el sistema real. ...	43
Tabla 4.6: Índices de rendimiento PID y LQG de ángulo en el sistema real.....	44
Tabla 4.7: Valores del Sistema PID y LQG de posición en el equipo real	45
Tabla 4.8: Índices de rendimiento de la salidas del sistema	45

ÍNDICE DE FIGURAS

Figura 2.1: Péndulo Invertido	3
Figura 2.2: Equipo Lego Mindstorms	4
Figura 2.3: Sensor Giroscopio NXT Hitechnic.....	5
Figura 2.4: Servo Motor y Encoder interno del robot Lego NXT	6
Figura 2.5: Sistema MIMO	6
Figura 2.6: Diagrama de bloques en variables de estado.....	8
Figura 2.7: Diagrama de bloques Proporcional Integral Derivativo.....	9
Figura 2.8: Diagrama de bloques LQG.....	10
Figura 2.9: Diagrama de bloques control LQR.....	12
Figura 2.10: Representación de un Filtro de Kalman	13
Figura 3.1: Movimientos generados del equipo Lego Mindstorms	15
Figura 3.2: Vista frontal del robot Lego Mindstoms	16
Figura 3.3: Diagrama de cuerpo libre del robot Lego Mindstorms.....	16
Figura 3.4: Modelo de un motor DC.....	20
Figura 3.5. Representación del sistema no lineal.....	25
Figura 3.6: Representación de entradas y salidas para la identificación.....	26
Figura 3.7: IDEN de Matlab para la función de transferencia del ángulo	26
Figura 3.8: IDEN de Matlab para la función de transferencia del posición.....	27
Figura 3.9: Simulación del controlador PID	28
Figura 3.10: Diagrama de flujo de la programación en Matlab.	29
Figura 3.11: Diagrama de flujo para la implementación del control PID.....	31
Figura 3.12: Diagrama de flujo para encontrar las matrices Q y R.....	34
Figura 3.13: Simulación del control LQG.....	36
Figura 3.14: Diagrama de flujo para la simulación del control LQG	37
Figura 3.15: Diagrama de flujo para la implementación del control LQG	38
Figura 4.1: Respuesta del sistema PID y LQG posición en simulación.....	41
Figura 4.2: Respuesta del sistema PID y LQG ángulo en simulación	42
Figura 4.3: Respuesta del sistema PID y LQG ángulo en el equipo real	43
Figura 4.4: Respuesta del sistema PID y LQG posición en el equipo real	44
Figura 4.5: Índice de rendimiento IAE para el ángulo.....	46
Figura 4.6: Índice de rendimiento IAE para la posición.	46
Figura 4.7: Respuesta de la función de costo vs Tiempo.....	47

RESUMEN

En el presente trabajo se realizan los controles LQG (Lineal Cuadrático Gaussiano) y PID aplicado al equipo Lego Mindstorms NXT en configuración de péndulo invertido sobre dos ruedas, como primer punto se obtiene el modelo matemático del sistema en ecuaciones de estado y las funciones de transferencia para el control del ángulo y control de posición. Con las ecuaciones de estados se procede a diseñar los controladores en el software Matlab y Simulink donde se busca la constante K del controlador LQR, y la constante L del Filtro de Kalman como observador, los cuales juntos forman el control LQG; para la simulación del control PID se obtienen los valores de los compensadores K_p , K_i , K_d para cada de una de sus salidas, también se realiza la aproximación de las constantes mediante algoritmos heurísticos basados en disminuir el error en estado estacionario.

La implementación de los controladores PID y LQG se realiza en el software RobotC, donde se adquiere los datos del sensor de giroscopio para el control de ángulo y la señal de los Encoders de las ruedas para el control de posición. Una vez obtenidas las señales de salida de los dos controladores en simulación y en el equipo real, se toman los valores de los controladores PID y LQG para contrastar el rendimiento de cada uno de ellos, mediante estos valores se obtienen el máximo sobreimpulso, tiempo de asentamiento y los índices de rendimiento IAE (Integral del valor absoluto del error) e ISE (Integral del cuadrado del error).

ABSTRACT

In this work LQG (Linear–Quadratic–Gaussian) and PID (Proportional–Integral–Derivative) controllers are performed, applying to Lego Mindstorms NXT set in two-wheeled inverted pendulum configuration, as a first point the mathematical model of the system is obtained from equations of the state and transfer functions for angle and position control. Controllers are designed with equations of state by MATLAB and Simulink software where LQR (Linear–Quadratic Regulator) variable K should be found, and Kalman Filter variable L as an observer, which both together deploy the LQG control; to simulate the PID control, K_p , K_i and K_d compensators values are got to each one of the outputs, variables approach are also performed by heuristic algorithms based on decreasing the error in steady state.

PID and LQG controllers implementation are performed by RobotC software, where gyroscope sensor data is got to control the angle and the signal of the encoders, and for the control of the wheels position. Once the dual controller simulation output signals are got in the real device, PID and LQG controllers values are taken to match the performance of each one of them, maximum overshoot is obtained by these values, settlement time and performance index. IAE (Integral Absolute Error) and ISE (Integrated Squared Error)

INTRODUCCIÓN

En este trabajo surge a partir de la idea de implementar un control óptimo a un equipo real debido que existen muchos proyectos que han sido limitados a la simulación ya que no se considera el aspecto físico y mecánico del sistema, a partir de esto se realiza un controlador LQG (Lineal Cuadrático Gaussiano) a un equipo Lego Mindstorms NXT en forma de un péndulo invertido sobre dos ruedas, el péndulo es uno de los equipos más utilizados dentro de los sistemas de control debido a su inestabilidad y falta de robustez, por lo cual la importancia de ser aplicada a un sistema físico, por lo que se necesita obtener un modelamiento matemático en donde se involucre la parte mecánica y eléctrica del sistema en ecuaciones de estado. Se realiza el control PID para un sistema MIMO en donde se necesita obtener dos funciones de transferencia para el ángulo y posición respectivamente, con ello obtener los compensadores K_p , K_i , K_d para realizar el control de cada una de las variables y obtener la respuesta del sistema. Una vez realizado el diseño de los controladores PID y LQG es necesario contrastar los resultados de ambos los controladores mediante los índices de rendimiento IAE para el control PID y la función de costo del control LQG; también es necesaria la implementación de los controladores en el sistema físico para mostrar el comportamiento del sistema en la planta real y a partir de esto saber cuál es el mejor controlador.

En el capítulo 1 se muestran los objetivos, planteamiento del problema y la justificación del trabajo.

En el capítulo 2 se considera la parte teórica donde se detallan las partes importantes de equipo Lego Mindstorms y los aspectos físicos que serán de importancia para la realización del trabajo.

En el capítulo 3 se realiza el modelamiento matemático para encontrar las ecuaciones de estados que describen el sistema, y proceder a la simular el control LQR, obteniendo las matrices Q y R basadas en las entradas y salidas, que se busca reducir la función de costo, se realiza el diseño del filtro de Kalman obteniendo la matriz L del observador que busca predecir el comportamiento de la planta.

En el capítulo 4 se presenta el análisis y resultados de la simulación e implementación de los controladores, se toman los valores de máximo sobreimpulso, tiempo de asentamiento e índices de rendimiento IAE e ISE.

CAPÍTULO 1

ANTECEDENTES

En el capítulo 1 se presenta el planteamiento del problema a resolver, así como la justificación del problema y objetivos generales y específicos.

1.1 Planteamiento del problema

Se necesita implementar un control óptimo LQG (Linear Quadratic Gaussian) en un sistema físico, en donde se tomen en cuenta aspectos que no son considerados en una simulación como ruido en la lectura de sensores, perturbaciones, y otros efectos de un ambiente real. Se encuentran varios proyectos de investigación en donde se aplican algunos tipos de control, pero muy pocos implementados en un sistema físico.

Controladores básicos como el PID son muy empleados para simulación y sistemas físicos, pero este controlador es empleado para sistemas de una entrada y una salida (SISO), también se conoce que este controlador es limitado ante perturbaciones y ruidos existentes propios de un sistema físico, el control PID no utiliza una función de costo para minimizar el error entre la referencia y el punto de equilibrio, y también no considera el esfuerzo de control.

La aplicación de un controlador diferente al PID en un sistema físico de péndulo invertido en dos ruedas mejorará el problema de robustez ante perturbaciones y la estabilidad del sistema, tomando en cuenta el ruido y las perturbaciones propias que existen un sistema real.

1.2 Justificación del proyecto

Se realizará el estudio de la aplicación de un control LQG en un robot móvil en configuración de péndulo invertido sobre dos ruedas, para el estudio y análisis de las perturbaciones y problemas que intervienen en la estabilidad del equipo cuando se implementa el controlador en un sistema real. Existen varios temas de investigación orientados a diferentes tipos de control, pero pocos implementados en un sistema real, por el que se plantea aplicar un control LQG óptimo que toma en cuenta las perturbaciones externas, y mejora la estabilidad mediante la aplicación del control.

1.3 Objetivos

1.3.1 Objetivo General

Diseñar e implementar un controlador óptimo LQG, para estabilizar en un punto de equilibrio, un equipo Lego Mindstorms en configuración péndulo invertido sobre dos ruedas.

1.3.2 Objetivos específicos

- Investigar y extraer información sobre el diseño del control LQG para el posterior desarrollo del modelamiento matemático del sistema y diseño del algoritmo.
- Analizar e implementar el equipo Lego Mindstorms en forma de un péndulo invertido sobre dos ruedas, para obtener el modelo matemático, su función de transferencia y realizar la simulación del sistema.
- Diseñar e implementar algoritmos de control PID, así como el LQG, para verificar el rendimiento y contrastar el resultado de cada uno de ellos.
- Realizar pruebas de funcionamiento de los controladores para validar la respuesta del sistema, comparando los resultados obtenidos de los controladores implementados.

Los beneficiarios para este trabajo son los investigadores y las personas interesados en la teoría de control, ya que el desarrollo de un controlador óptimo es un aporte a nuevos conocimientos que aún se pueden ir mejorando.

CAPÍTULO 2

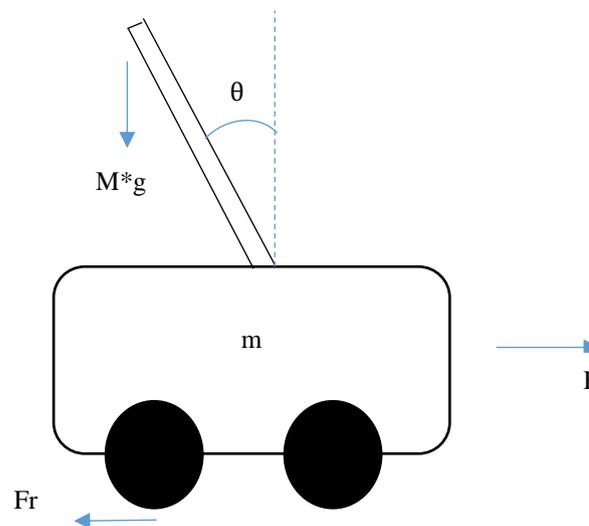
MARCO TEÓRICO

En el Capítulo 2 se describe el equipo Lego Mindstorms con sus sensores y actuadores, utilizado para la construcción del péndulo invertido sobre dos ruedas, posteriormente se describe el diseño de los controladores PID y LQG.

2.1 Péndulo Invertido

El péndulo invertido es uno de los prototipos más comunes, debido a que posee similitud con el cuerpo humano al activar las partes de su cuerpo para obtener estabilidad. Debido a su inestabilidad y falta de robustez en el sistema ha sido objeto de estudio en el campo de la investigación o de laboratorio. Existen algunos prototipos como son: péndulo invertido sobre un carro en donde gira libremente sobre un riel, o prototipos de robots móviles sobre dos ruedas acoplado una tarjeta de adquisición, también sobre un carro que sostiene el péndulo como se muestra en la Figura 2.1. (Loya, Arroyo, Rodriguez , & Jaramillo, 2015)

Figura 2.1: Péndulo Invertido



Prototipo de un péndulo invertido, Fuente: (Messner & Tilbury, 2011)

Donde:

m = Masa del carro.

M = Masa de la barra.

g = Gravedad.

θ = Ángulo de inclinación.

F = Fuerza

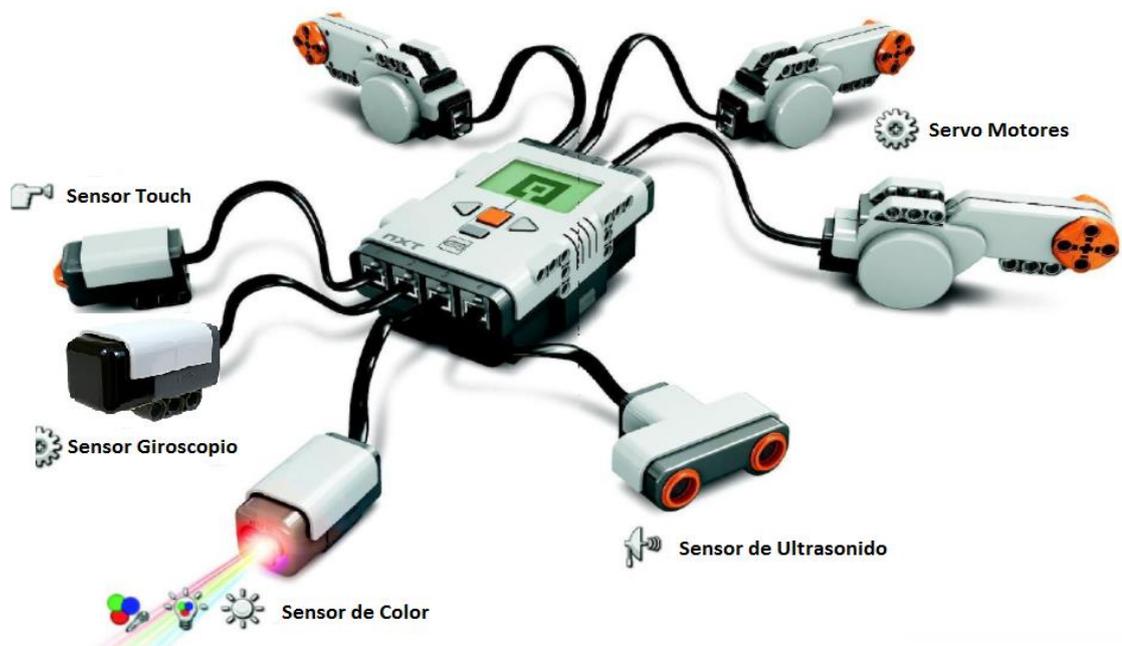
Fr = Fuerza de fricción.

2.2 Lego Mindstorms NXT

El equipo Lego Mindstorms es un set de construcción que ha sido utilizado principalmente para robots móviles, aunque en la actualidad tiene muchas aplicaciones en la ingeniería debido a la versatilidad del equipo por su parte mecánica. El equipo Lego Mindstorms NXT posee un cerebro llamado Brick que está formado por un microcontrolador de 32 bits ARM 7, posee una pantalla LCD, 4 botones de navegación, 3 puertos de entrada para conectar sensores y 4 puertos de salidas para conectar actuadores mediante cables RJ12 directamente al Brick del robot. (Zaldívar Navarro, Cuevas Jiménez, & Pérez Cisneros , 2015)

El equipo Lego Mindstorms posee una memoria Flash de 4 KB, 512 Byte de memoria RAM, y un oscilador de 48MHz, soporta una frecuencia de muestreo de 16.2 KHz; el equipo Lego dispone de comunicación inalámbrica Bluetooth y un puerto de comunicación USB de alta velocidad (12Mbit/s), en la Figura 2.2 se presenta el equipo Lego Mindstorms con sus principales sensores y actuadores. (Lego Mindstorms, 2017)

Figura 2.2: Equipo Lego Mindstorms



Aspecto físico del Kit Lego Mindstorms con sensores y actuadores, Fuente:

(Mina, 2015)

El equipo Lego Mindstorms es compatible con varios lenguajes de programación como son: NXC, LEJOS NXJ, ECROBOT, Toolbox MATLAB RWTH-Mindstorms, algunos de software libre y otros que se necesita adquirir su licencia, el equipo Lego posee su plataforma gráfica llamada LEGO MINDSTORMS NXT, existe el programa RobotC que está basado en una plataforma de lenguaje C, el software RobotC tiene varias ventajas y desventajas como: (Zaldívar Navarro, Cuevas Jiménez, & Pérez Cisneros , 2015)

Ventajas

- Amplio número de instrucciones.
- Tamaño moderado de código en aplicaciones complejas.
- Posee una herramienta de programación en lenguaje C.

Desventajas

- Requiere del aprendizaje de funciones de programación.
- Software con pago.

2.2.1 Sensor Giroscopio Hitechnic

El sensor giroscopio Hitechnic que se presenta en la Figura 2.3, mide la rotación y devuelve el valor en grados por segundo, el sensor se utiliza para determinar el ángulo de inclinación respecto a un eje. Tiene un rango de medición de $\pm 360^\circ$ por segundo, y la velocidad de rotación es de 300 muestras por segundo. (HiTechnic, 2012)

Figura 2.3: Sensor Giroscopio NXT Hitechnic



Aspecto físico del sensor giroscopio, Fuente: (HiTechnic, 2012)

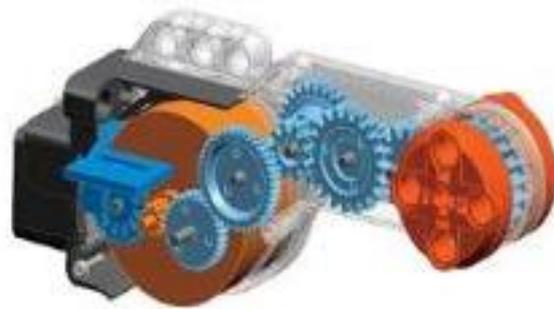
2.2.2 Servo Motores y Encoders

El Kit Lego Mindstorms NXT viene equipado de 3 servomotores de corriente continua DC que son polarizados con 9 V y conectan directamente a las baterías del equipo

Lego Mindstorms, el servo motor posee internamente engranajes con una relación de 1:48, un torque de 11.5 N.cm, y una velocidad de 170 rpm sin carga y 160 rpm con carga.

Dentro del servo motor se encuentra acoplado una rueda que posee 12 ranuras y está conectado a un sensor óptico o tacómetro que permite determinar la posición de la rueda y medir la revoluciones por minuto (rpm). El tacómetro mide las rotaciones del motor en grados o rotaciones completas abarcando los 360° , por lo que posee una precisión de ± 1 grado aproximadamente debido a su retroalimentación rotacional, el encoder mide la velocidad angular en rad/s y dicha información es enviada al cerebro del equipo (Brick) para procesar su información. La velocidad de los motores se puede modificar entre (+100,-100) mediante software, de acuerdo con la aplicación orientada del equipo, el servo motor se puede observar en la Figura 2.4. (Edubrick, 2010)

Figura 2.4: Servo Motor y Encoder interno del robot Lego NXT

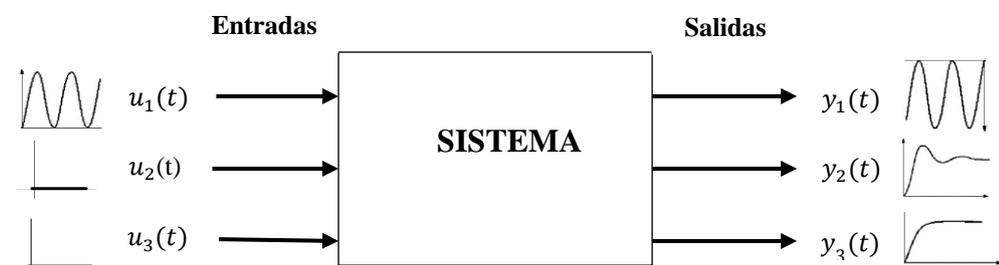


Vista interna del Servo Motor, Fuente:(Santos, 2010)

2.3 Sistema MIMO (Multi-Input Multi-Output)

Se conoce como sistema MIMO al sistema que tiene múltiples entradas y múltiples salidas, en un sistema MIMO si se modifica una de las entradas también cambiara su salida y las otras salidas, como se presenta en la Figura 2.5. (Ogata, 2003)

Figura 2.5: Sistema MIMO



Representación de un sistema MIMO, Elaborado por: Diana Solórzano

2.4 Representación en espacio de estados

En la teoría de control clásico, el modelo del sistema es representado por una función de transferencia y depende solamente de la información de las entradas y salidas del sistema, sin conocer la estructura interior de la planta. El control moderno resuelve muchas limitaciones ya que utiliza una información más completa de la dinámica de la planta. Por lo cual la representación en espacio de estados describe la parte eléctrica y mecánica de la planta, los cuales muestra el comportamiento del sistema. Se conoce como estado a la mínima cantidad de información obtenida del sistema, que describe el comportamiento dinámico de la planta, en un instante de tiempo, conociendo su entrada en un instante $t = t_0$, se determinar cualquier variable en un estado posterior $t = t + 1$. (Dominguez, Campoy, Sebastian , & Jiménez, 2006).

El modelo de sistema completo para un sistema lineal invariante en el tiempo consiste en un conjunto de n estados o ecuaciones, definidas en términos de las matrices A y B , y un conjunto de ecuaciones de salida que relacionan cualquier variable de salida de interés con las variables de estado y las entradas, y se expresan en términos de las matrices C y D . Las ecuaciones en las variables de estados, para la entrada se muestran en la ecuación 2.1 y para la salida del sistema se encuentra en la ecuación 2.2.

$$\dot{x}(t) = Ax(t) + Bu(t) \quad \text{Ec. (2.1)}$$

$$y(t) = Cx(t) + Du(t) \quad \text{Ec. (2.2)}$$

Donde:

$\dot{x}(t)$ = Señal de control

$y(t)$ = Salida

$A(t)$ = Matriz $n \times n$ en donde n representa el número de estados del sistema.

$B(t)$ = Matriz $n \times m$ en donde m es el número de entradas del sistema.

$C(t)$ = Matriz $r \times n$ en donde r es el número de salidas a controlarse.

$D(t)$ = Matriz $r \times m$ del sistema.

Las matrices A y B son propiedades del sistema y están determinadas por el sistema estructura y elementos. Las matrices de salida de la ecuación C y D son determinadas

por elección particular de variables de salida. La representación en diagrama de bloques de las variables de estado, se muestran en la Figura 2.6.

Figura 2.6: Diagrama de bloques en variables de estado

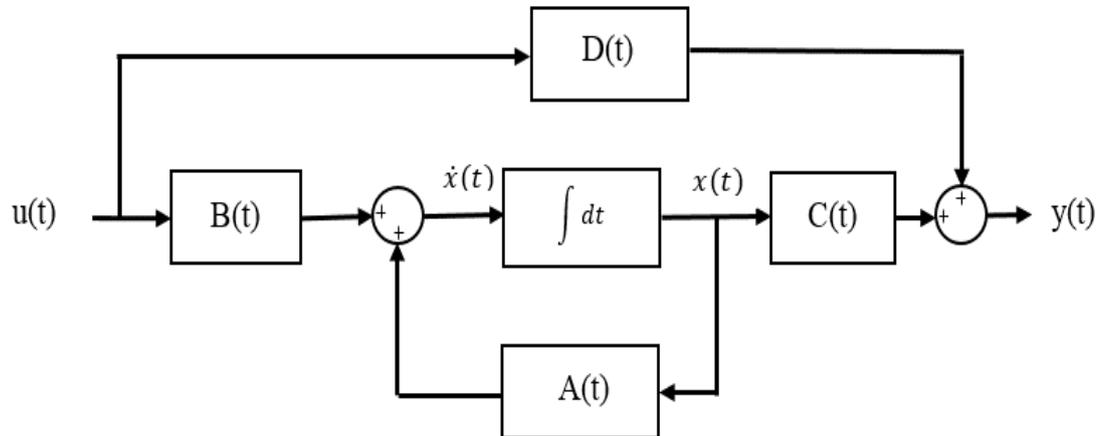


Diagrama de bloques para un sistema lineal en espacio-estados, Fuente: (Rowell, 2002)

2.5 Control Proporcional-Integral-Derivativo (PID)

El control PID es uno de los controles más utilizados en el campo industrial, debido a la simplicidad y fácil aplicación. Su implementación trata de encontrar valores de cada una de sus ganancias con el objetivo de minimizar el error, entre la entrada o setpoint y la salida del sistema. (Rairán Antolines, 2007)

El control PID recibe su nombre por las tres acciones básicas del control, P para su acción proporcional, I para su acción integral y D para la acción derivativa. El control PID es un elemento en el sistema de lazo cerrado donde la entrada es una señal de error y su salida que se convierte en el elemento correctivo. (Ramirez, 2015).

En la Tabla 2.1 se muestran las constantes K_p , K_i , K_d y las características de cada una cuando son modificadas en el sistema.

Tabla 2.1: Constantes del controlador PID

Constante	Tiempo de levantamiento	Sobreimpulso	Tiempo de asentamiento	Error en estado estable
K_p	Decrece	Aumenta	Pequeño cambio	Decrece
K_i	Decrece	Aumenta	Aumenta	Elimina
K_d	Pequeño cambio	Decrece	Decrece	Pequeño cambio

Características de las constantes del control PID, Fuente: (DISEÑO DE UN CONTROLADOR PID PARA EL SISTEMA PITCH CONTROLLER , 2012)

La suma de los controles P, I, D representado en términos de Laplace se muestra en la ecuación 2.3.

$$G_c(s) = K_p + K_d s + \frac{K_i}{s} \quad \text{Ec. (2.3)}$$

Donde:

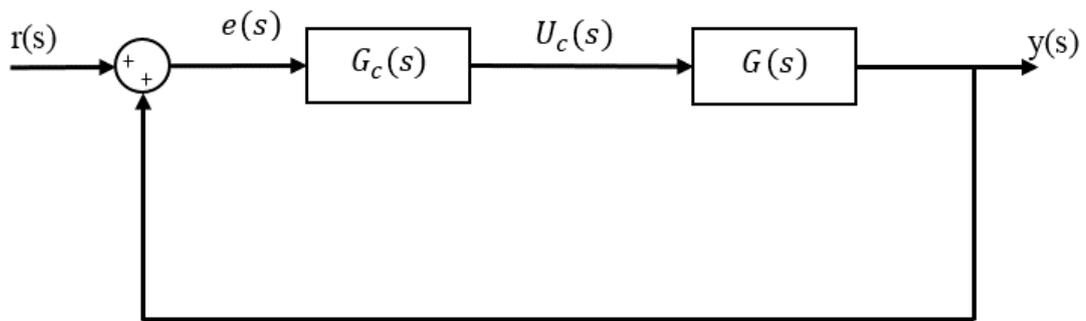
K_p = Constante de proporcionalidad.

K_i = Tiempo requerido para que la acción integral.

K_d = Tiempo requerido para que la acción derivativa. (Morillo, 2007)

Se representa el control PID en diagrama de bloques, se muestra en la Figura 2.7.

Figura 2.7: Diagrama de bloques Proporcional Integral Derivativo



Representación del control PID en diagrama de bloques, Fuente: (Morillo, 2007)

Donde:

$r(t)$ = Referencia o setpoint.

$U_c(s)$ = Señal de control.

$e(s)$ = Error.

2.7 Control Lineal Cuadrático Gaussiano - LQG

El controlador Lineal Cuadrático Gaussiano (Linear Quadratic Gaussian) con sus siglas en inglés LQG conocido así, debido a que se introduce al sistema perturbaciones, conocidas como ruido blanco o gaussiano, ha sido aplicado en muchos sistemas dinámicos debido a los parámetros desconocidos que esto conlleva, el original problema del controlador se dividió en el óptimo problema de filtrado sobre

observaciones lineales. Este es uno de los controles más fundamentales dentro de los controladores óptimos. El controlador LQG es la combinación de un controlador LQR y un Estimador de Kalman es decir un estimador lineal cuadrático (LQE), los cuales pueden ser diseñados de forma independiente, o uno con relación a otro. El modelo del proceso se muestra en la ecuación 2.4 y 2.5, donde se representa las ecuaciones de entrada y salida con perturbación.

$$\dot{x} = A x(t) + B u(t) + W_k \quad \text{Ec. (2.4)}$$

$$y = C x(t) + D u(t) + V_k \quad \text{Ec. (2.5)}$$

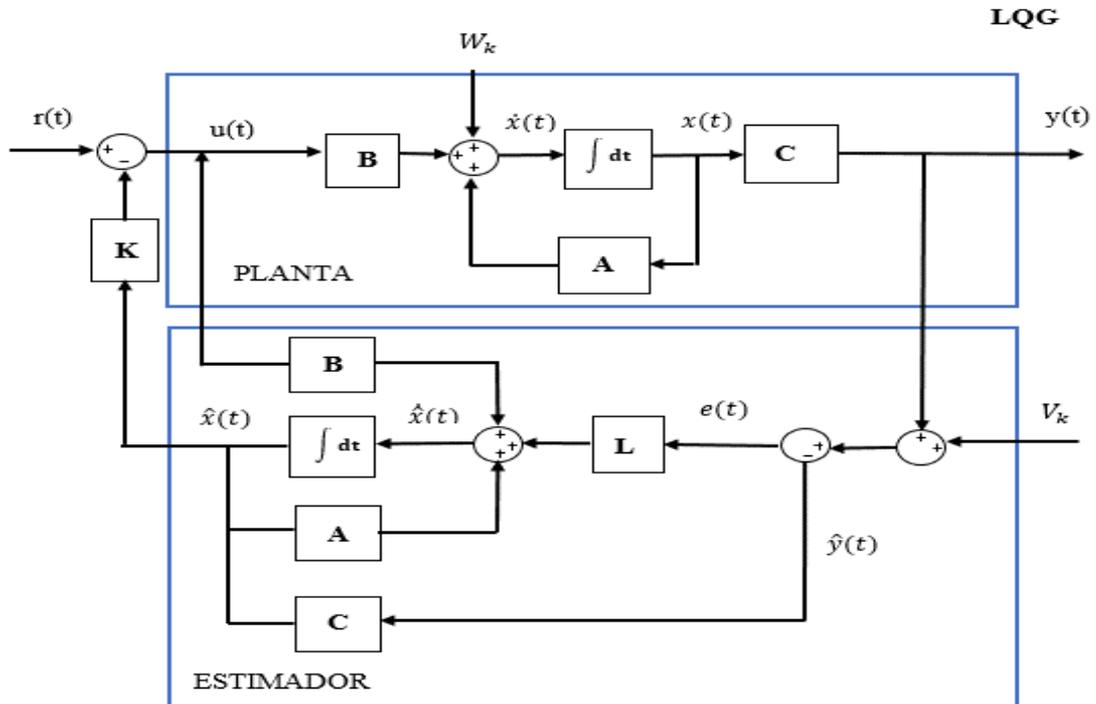
Donde:

W_k = Ruido del Sistema

V_k = Ruido del estado

En la Figura 2.8 se muestra la representación en diagrama de bloques del controlador Lineal Cuadrático Gaussiano LQG.

Figura 2.8: Diagrama de bloques LQG



Representación del control LQG en diagrama de bloques, Elaborado: Diana Solórzano

Donde:

L = constante del filtro de Kalman.

K = constante del regulador LQR.

2.7.1 Ventajas y desventajas del control LQG

Las ventajas tienen el control LQG son:

- Se aplican en sistemas robustos debido a que es capaz de rechazar perturbaciones y seguir la referencia, al que se le conoce como principio de modelamiento interno.
- Minimiza la varianza del error de estimación a partir de la caracterización de los ruidos.

Las desventajas tienen el control LQG son:

- El control LQG tiene limitaciones de estabilización, cuando se trabaja con estructuras mecánica móviles, para solucionar este problema se realiza un filtro observador y el diseño de una matriz de ganancia de realimentación.
- Se indica que, si existe un desajuste debido a la inexactitud del modelamiento, en los parámetros de la curva lineal o no linealidades (es decir, el sistema real no es realmente lineal), entonces el controlador resultante se degradará y el sistema puede incluso volverse inestable.

2.8 Regulador Cuadrático Lineal - LQR

El control LQR por sus palabras en inglés (Linear Quadratic Regulator) conocido como regulador cuadrático lineal, es una estrategia de control óptimo; que, mediante la optimización de la función de coste, calcula la ley de control para optimizar mediante su rendimiento.

El control LQR contempla el problema de regulación, que resuelve a partir de la optimización de un índice cuadrático, con una solución lineal. El control LQR minimiza una función cuadrática o índice de coste que se muestra en la ecuación 2.6.

$$J = \int_{t_0}^{t_f} (x^T Q x + u^T R u) dt \quad \text{Ec. (2.6)}$$

Donde x representa los estados del sistema; las matrices de pesos Q , R son matrices simétricas positivas.

Para la elección de las matrices Q y R , no existen reglas que se puedan emplear de una manera general, en donde consiste escoger los valores Q y R diagonales. Asignando valores grandes aquellas variables que se desean minimizar, estos valores deben ser positivos o cero. (Universitat Politècnica de Catalunya)

Figura 2.9: Diagrama de bloques control LQR

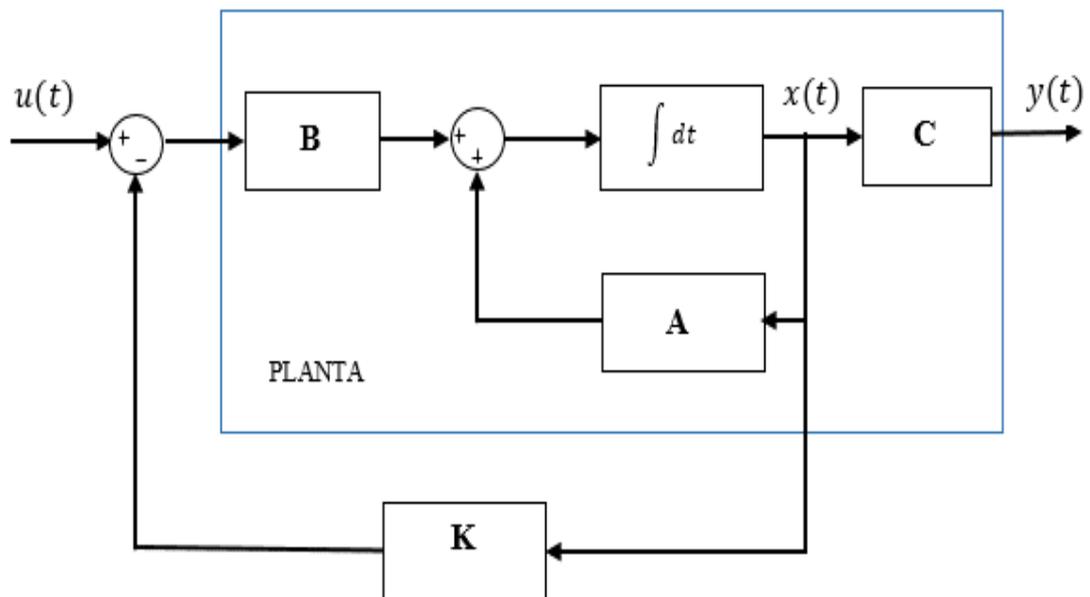


Diagrama de bloques del control LQR, Fuente: (Villacres & Viscaino, 2016)

De la Figura 2.9 se observa que la señal de control $u(t)$ está dada por la ecuación 2.7:

$$u(t) = -Kx(t) \quad \text{Ec.(2.7)}$$

Donde

K = Matriz de ganancias realimentadas.

2.9 Filtro de Kalman

Es un estimador de estados que puede comportarse como filtro o predictor, en el diseño de este filtro se toma en cuenta la presencia de ruidos estocásticos o gaussianos dentro del proceso. Por lo cual se conoce como un estimador óptimo, debido a la aproximación obtenida en la cual se busca tener el menor error posible, al considerar el ruido del sistema y del proceso no significa que se obtenga una aproximación perfecta.

Suponiendo la planta dinámica se prescribe como se muestran en las ecuaciones 2.8 y 2.9:

$$x_{k+1} = A x_k + B u_k + W_k \quad \text{Ec.(2.8)}$$

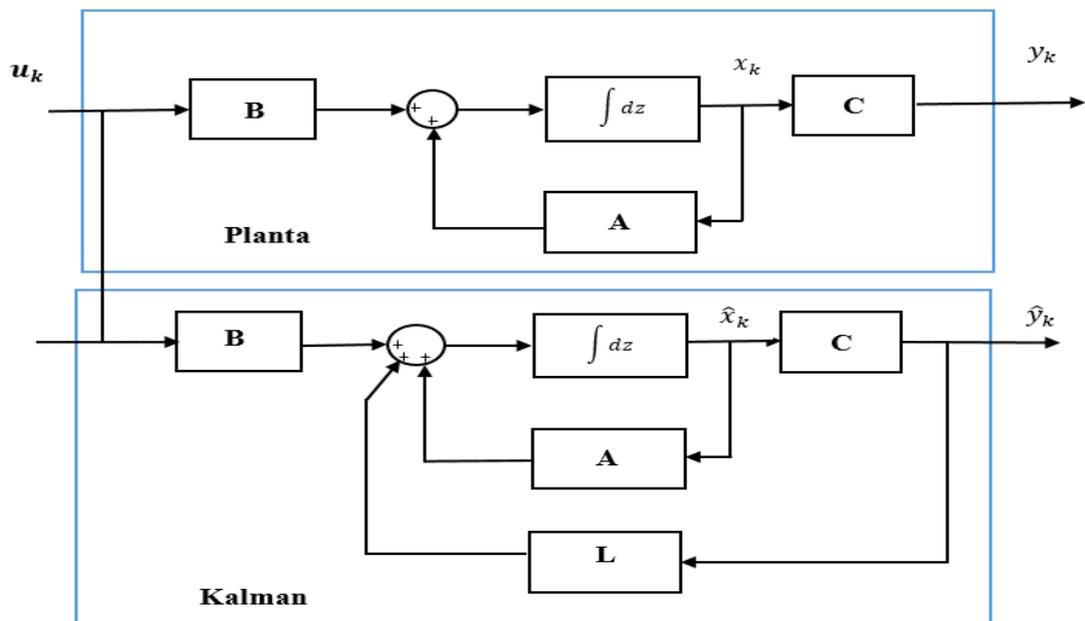
$$y_k = C x_k + D u_k + V_k \quad \text{Ec.(2.9)}$$

Donde el estado $x_k \in R^n$, la entrada de control $u_k \in R^m$, la salida y_k , y las matrices A, B, C son matrices constantes conocidas de dimensiones apropiadas. Todas las variables son deterministas, de modo que si el estado inicial x_0 es conocido por la ecuación 2.8 se puede resolver exactamente para x_k, y_k para $k \geq 0$. Se diseña un estimador cuya salida y_k converge con k con estado real x_k de la ecuación 2.9 cuando el estado inicial x_0 es desconocido, pero u_k y y_k se dan exactamente. Un estimador u observador tiene la forma:

$$\hat{x}_{k+1} = A \hat{x}_k + L(y_k - C \hat{x}_k) + B u_k \quad \text{Ec.(2.10)}$$

Donde las matrices A, B, C son las mismas de la ecuación 2.8, en donde es necesario encontrar solamente la matriz del observador L . Se debe tomar en cuenta que el observar consiste en dos partes, la réplica de A, B, C de la dinámica de la planta cuyo estado de ser estimado y la corrección del error es descrita por $L(y_k - \hat{y}_k) = L \tilde{y}_k$, donde la salida estimada es $\hat{y}_k = C \hat{x}_k$ y $\tilde{y}_k = y_k - \hat{y}_k$ es el residuo. (Lewis, Xie, & Popa , 2008)

Figura 2.10: Representación de un Filtro de Kalman



Filtro de Kalman como observador, Fuente: (Lewis, Xie, & Popa , 2008)

2.10 Índices de rendimiento IAE e ISE

Se definen como índices de rendimiento a los valores que son utilizados como parámetros para evaluar la calidad de la respuesta del sistema ante una entrada. Por lo tanto, la optimización de los controladores dependerá del mejor índice de rendimiento.

El índice del valor absoluto del error (IAE), es más sensible a los errores pequeños y menos insensibles a los errores grandes. La fórmula se muestra en la ecuación 2.11.

$$IAE = \int_0^{\infty} e(t)dt \quad \text{Ec.(2.11)}$$

El índice de la integral del cuadrado del error (ISE), es insensible a pequeños errores, pero a grandes errores contribuyen grandemente al valor de la integral. El valor de ISE tendrá una respuesta con pequeños sobrepasamientos, pero con largos tiempos de estabilización debido a que los errores pequeños errores a lo largo del tiempo no contribuyen a la integral del error. La fórmula se muestra en la ecuación 2.12. (Acedo Sánchez, 2006)

$$ISE = \int_0^{\infty} [e(t)]^2 dt \quad \text{Ec.(2.12)}$$

CAPÍTULO 3

DISEÑO E IMPLEMENTACIÓN DE LOS CONTROLADORES

En el capítulo 3 se realiza el modelamiento matemático del péndulo invertido sobre dos ruedas, las variables de estado del sistema y el diseño de los controladores PID y LQG.

3.1 Modelo de la dinámica del péndulo invertido sobre dos ruedas

El equipo Lego Mindstorms tiene 2 movimientos en el plano debido a su inclinación y rotación, cuando el robot se inclina realiza el movimiento en Pitch o Cabeceo y en él se genera el ángulo psi (ψ), cuando el robot gira realiza el movimiento en Yaw y forma el ángulo fi (ϕ), el equipo Lego no se desplaza en el eje z por lo cual no genera el movimiento en Roll, los movimientos se muestran en la Figura 3.1.

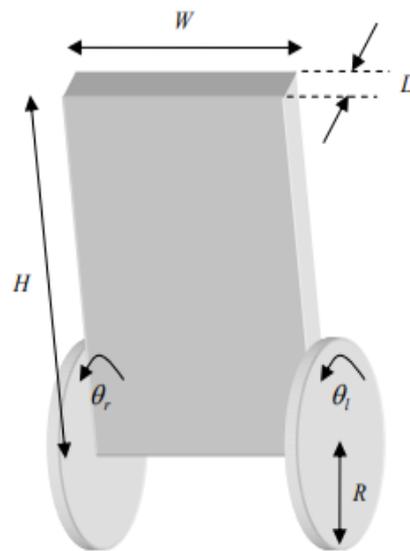
Figura 3.1: Movimientos generados del equipo Lego Mindstorms



Movimientos Pitch, Yaw, Roll del equipo Lego y los ángulos formados por los movimientos, Elaborado por: Diana Solórzano.

El robot Lego NXT se puede considerar como un modelo de péndulo invertido de dos ruedas como se muestra en la Figura 3.2.

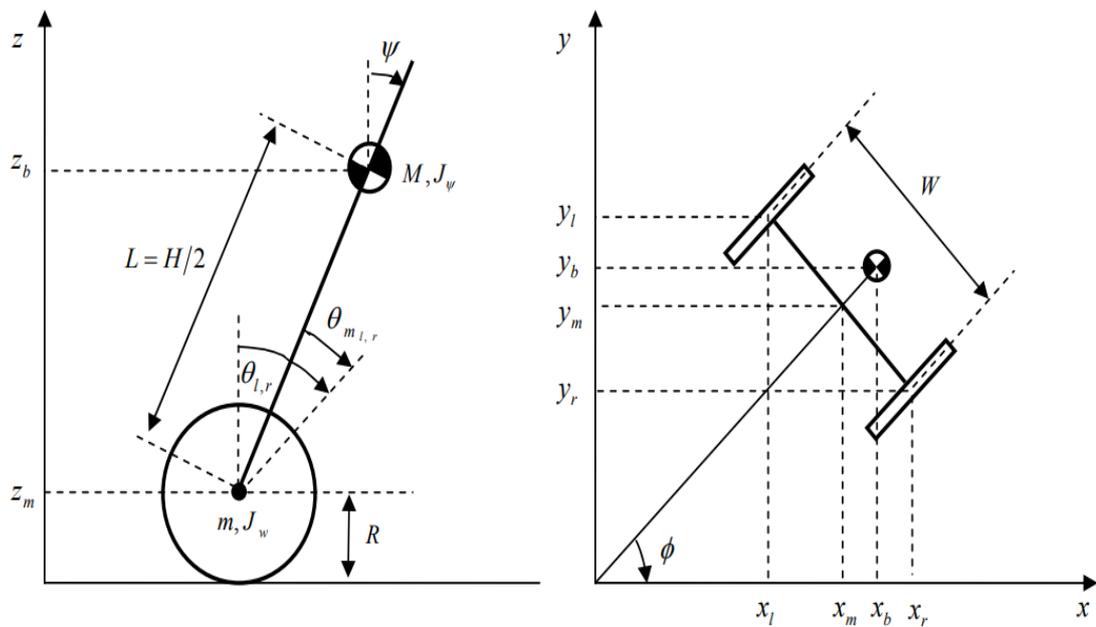
Figura 3.2: Vista frontal del robot Lego Mindstorms



Parámetros del robot Lego vista desde el frente, Fuente: (Yamamoto, 2009)

En la Figura 3.3 se muestra el diagrama de cuerpo libre del equipo Lego Mindstorms y los parámetros del sistema.

Figura 3.3: Diagrama de cuerpo libre del robot Lego Mindstorms



Parámetros del robot Lego, Fuente: (Yamamoto, 2009)

Donde:

ψ = Ángulo psi formado del movimiento en Pitch.

θ = Ángulo theta generado por la posición angular del promedio de la rueda derecha y la rueda izquierda.

Las entradas del sistema son:

v_{izq} = Voltaje para la rueda del motor izquierdo.

v_{der} = Voltaje para la rueda del motor derecho.

Las salidas del sistema son:

ψ = Ángulo del robot con respecto a la vertical.

θ = Posición de las ruedas.

En la Tabla 3.1 se muestran los parámetros que se utilizan para el modelamiento matemático y los respectivos valores correspondientes al Robot Lego Mindstorms NXT.

Tabla 3.1: Descripción de parámetros físicos del equipo Lego Mindstorms

Parámetros	Descripción
$g = 9.8 [m/sec^2]$	Gravedad
$m = 0.03 [kg]$	Peso de la rueda
$R = 0.028 [m]$	Radio de la rueda
$J_w = mR^2/2 [kgm^2]$	Momento de inercia de la rueda
$M = 0.6 [kg]$	Peso del cuerpo
$W = 0.09 [m]$	Ancho del cuerpo
$D = 0.05 [m]$	Profundidad del cuerpo
$H = 0.26 [m]$	Altura del cuerpo
$L = H/2 [m]$	Distancia del centro de masa del eje de la rueda
$J_m = 1 \times 10^{-5} [kgm^2]$	Momento de inercia del motor DC
$J_\phi = M(W^2 + D^2)/12 [kgm^2]$	Momento de inercia (yaw)
$J_\psi = ML^2/3 [kgm^2]$	Momento de inercia (pitch)
$R_m = 6.69 [\Omega]$	Resistencia del motor DC
$K_b = 0.468 [V \text{ sec}/rad]$	Constante de velocidad del motor DC
$K_t = 0.317 [Nm/A]$	Constante de torque del motor DC
$n = 1$	Relación de transmisión
$f_m = 0.0022$	Coefficiente de fricción entre el cuerpo y el motor
$f_w = 0$	Coefficiente de fricción entre la rueda y el suelo

Tabla de parámetros internos del Equipo LEGO, Fuente: (Yamamoto, 2009)

Las ecuaciones que describen el movimiento del sistema se obtienen mediante el método de Euler - Lagrange, sobre el sistema de coordenadas de la Figura 3.3 (Yamamoto, 2009)

$$(\theta) = \left(\frac{1}{2}(\theta_{izq} + \theta_{der}) \right) \quad \text{Ec. (3.1)}$$

$$(x_{md}, y_{md}, z_{md}) = \left(\int \dot{x}_{md} dt, \int \dot{y}_{md} dt, R \right), (\dot{x}_{md}, \dot{y}_{md}) \quad \text{Ec. (3.2)}$$

$$= (R \dot{\theta} \cos \phi) \quad \text{Ec. (3.3)}$$

$$(x_{izq}, y_{izq}, z_{izq}) = \left(x_{med} - \frac{W}{2}, y_{med} + \frac{W}{2}, z_{med} \right)$$

$$(x_{der}, y_{der}, z_{der}) = \left(x_{med} + \frac{W}{2}, y_{med} - \frac{W}{2}, z_{med} \right) \quad \text{Ec. (3.4)}$$

$$(x_c, y_c, z_c) = (x_{med} + L \sin \psi \cos \phi, y_{med} + L \sin \psi, z_{med} + L \cos \psi) \quad \text{Ec. (3.5)}$$

Donde:

$(x_{med}, y_{med}, z_{med})$ = Coordenadas del centro de masa del equipo Lego.

$(x_{izq}, y_{izq}, z_{izq})$ = Coordenadas de la rueda izquierda en el centro de masa.

$(x_{der}, y_{der}, z_{der})$ = Coordenadas de la rueda derecha en el centro de masa.

(x_c, y_c, z_c) = Coordenadas del péndulo invertido en el centro del cuerpo.

θ_{izq} = Ángulo de desplazamiento de la rueda izquierda.

θ_{der} = Ángulo de desplazamiento de la rueda derecha.

Utilizando las ecuaciones 3.1, 3.2, 3.3, 3.4 y 3.5. De las coordenadas de la ecuación 3.5 se establece la energía cinética traslacional T_1 , la energía cinética rotacional T_2 y la energía potencial U que está dada por las ecuaciones 3.6, 3.7, 3.8. (Yamamoto, 2009)

$$T_1 = \frac{1}{2}m(\dot{x}_{izq}^2 + \dot{y}_{izq}^2 + \dot{z}_{izq}^2) + \frac{1}{2}m(\dot{x}_{der}^2 + \dot{y}_{der}^2 + \dot{z}_{der}^2) + \frac{1}{2}M(\dot{x}_c^2 + \dot{y}_c^2 + \dot{z}_c^2) \quad \text{Ec. (3.6)}$$

$$T_2 = \frac{1}{2}J_w \dot{\theta}_{izq}^2 + \frac{1}{2}J_w \dot{\theta}_{der}^2 + \frac{1}{2}J_\psi \dot{\psi}^2 + \frac{1}{2}n^2 J_{med} (\dot{\theta}_{izq} - \dot{\psi})^2 + \frac{1}{2}n^2 J_{med} (\dot{\theta}_{der} - \dot{\psi})^2 \quad \text{Ec. (3.7)}$$

$$U = mgz_{izq} + mgz_{der} + Mgz_c \quad \text{Ec. (3.8)}$$

z_{izq} = coordenda en z de la rueda izquierda.

z_{der} = coordenada en z de la rueda derecha.

z_c = coordenada en z del centro de masa.

Los últimos términos de la ecuación 3.8 corresponden a la energía cinética generada en la armadura de los dos motores DC del equipo Lego Mindstorms. (Yamamoto, 2009)

Las expresiones de Lagrange se muestran en las ecuaciones 3.10 y 3.11, está dada por la ecuación 3.9.

$$L = T_1 + T_2 - U \quad \text{Ec. (3.9)}$$

Las ecuaciones de Lagrange son:

$$F_\theta = \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\theta}} \right) - \frac{\partial L}{\partial \theta} \quad \text{Ec. (3.10)}$$

$$F_\psi = \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\psi}} \right) - \frac{\partial L}{\partial \psi} \quad \text{Ec. (3.11)}$$

Donde:

F_θ = Fuerza generalizada del ángulo de la posición angular promedio de las ruedas derecha e izquierda. (Herrera Garzón, 2014)

F_ψ = Fuerza generalizada del ángulo generado por el movimiento en Pitch.

Resolviendo las ecuaciones 3.10 y 3.11 se obtienen las ecuaciones 3.12 y 3.13. (Yamamoto, 2009)

$$F_\theta = [(2m + M)R^2 + 2J_w + 2n^2J_m]\ddot{\theta} + (MLR \cos \psi - 2n^2J_m)\ddot{\psi} - MLR\dot{\psi}^2 \sin \psi \quad \text{Ec. (3.12)}$$

$$F_\psi = [MLR \cos \psi - 2n^2J_m]\ddot{\theta} + (ML^2 + J_w + 2n^2J_m)\ddot{\psi} - MgL \sin \psi \quad \text{Ec. (3.13)}$$

Considerando el torque del motor DC y la fricción viscosa, la fuerza generalizada se presentan las ecuaciones 3.14, 3.15, 3.16 y 3.17. (Yamamoto, 2009)

$$(F_\theta, F_\psi) = (F_{izq} + F_{der}, F_\psi) \quad \text{Ec. (3.14)}$$

$$F_{izq} = nK_t i_{izq} + f_m(\dot{\psi} - \dot{\theta}_{izq}) - f_w \dot{\theta}_{izq} \quad \text{Ec. (3.15)}$$

$$F_{der} = nK_t i_{der} + f_m(\dot{\psi} - \dot{\theta}_{der}) - f_w \dot{\theta}_{der} \quad \text{Ec. (3.16)}$$

$$F_\psi = -nK_t i_{izq} - nK_t i_{der} - f_m(\dot{\psi} - \dot{\theta}_{izq}) - f_m(\dot{\psi} - \dot{\theta}_{der}) \quad \text{Ec. (3.17)}$$

Donde:

i_{der} = Corriente del motor DC derecho.

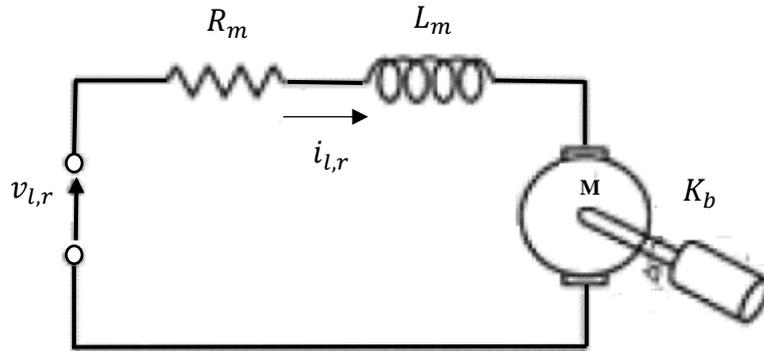
i_{izq} = Corriente del motor DC izquierdo.

f_m = Coeficiente de fricción entre el cuerpo del equipo Lego y los motores.

f_w = Coeficiente de fricción entre las ruedas y la superficie en la que se desliza.

No se puede utilizar directamente la corriente del motor DC, porque el sistema está basado en el control PWM. Por lo tanto, se evalúa la relación entre corriente y voltaje usando la ecuación del motor DC. La ecuación general del motor se muestra en la ecuación 3.18. (Herrera Garzón, 2014)

Figura 3.4: Modelo de un motor DC



Modelo de la estructura interna de un motor, Fuente: (Castillo, 2008)

Se obtienen las ecuaciones 3.18 y 3.19 mediante la Ley de Kirchhoff, basado en la Figura 3.4.

$$v_{izq,der} + K_b = L_m \dot{i}_{izq,der} + R_m i_{izq,der} \quad \text{Ec. (3.18)}$$

$$L_m \dot{i}_{izq,der} = v_{izq,der} + K_b - R_m i_{izq,der} \quad \text{Ec. (3.19)}$$

Se desprecia la fricción interna del robot con la superficie en la que se desliza. (Villacres & Viscaino, 2016)

Donde:

L_m = Inductancia interna del motor DC

R_m = Resistencia interna del motor DC

K_b = Constante de velocidad del motor DC.

La inductancia interna del motor DC es despreciable, a partir de la ecuación 3.19 se obtiene la ecuación 3.20.

$$i_{izq,der} = \frac{v_{izq,der} + K_b}{R_m} \quad \text{Ec. (3.20)}$$

De la ecuación 3.20 la representación generalizada de las fuerzas se puede expresar utilizando el voltaje del motor, como se muestra en la ecuación 3.21 y 3.22, con las ecuaciones 3.23 y 3.24 se realiza el reemplazo de los valores. (Yamamoto, 2009)

$$F_\theta = \alpha(v_{izq} + v_{der}) - 2(\beta + f_w)\dot{\theta} + 2\beta\dot{\psi} \quad \text{Ec. (3.21)}$$

$$F_\psi = -\alpha(v_{izq} + v_{der}) + 2\beta\dot{\theta} - 2\beta\dot{\psi} \quad \text{Ec. (3.22)}$$

Donde:

$$\alpha = \frac{nK_t}{R_m} \quad \text{Ec. (3.23)}$$

$$\beta = \frac{nK_tK_b}{R_m} + f_m \quad \text{Ec. (3.24)}$$

Igualando las ecuaciones 3.10, 3.11 y las ecuaciones 3.12, 3.13 se obtiene las ecuaciones 3.25, 3.26 que muestran el comportamiento del sistema no lineal. (Villacres & Viscaino, 2016)

$$[(2m + M)R^2 + 2J_w + 2n^2J_m]\ddot{\theta} + (MLR \cos \psi - 2n^2J_m)\ddot{\psi} - MLR\dot{\psi}^2 \sin \psi = \alpha(v_l + v_r) - 2(\beta + f_w)\dot{\theta} + 2\beta\dot{\psi} \quad \text{Ec. (3.25)}$$

$$[MLR \cos \psi - 2n^2J_m]\ddot{\theta} + (ML^2 + J_w + 2n^2J_m)\ddot{\psi} - MgL \sin \psi - ML^2\dot{\phi}^2 \sin \psi \cos \psi = -\alpha(v_l + v_r) + 2\beta\dot{\theta} - 2\beta\dot{\psi} \quad \text{Ec. (3.26)}$$

Para hallar $\ddot{\theta}$ en la ecuación 3.26 reemplazando en ecuación 3.25 se obtiene la ecuación 3.27 y 3.28. (Herrera, Camacho, Chamorro, & Gómez, 2015)

$$\Delta = (MLR \cos \psi - 2n^2 J_m)^2 - [(2m + M)R^2 + 2J_w + 2J_m](ML^2 + J_\psi + 2n^2 J_m) \quad \text{Ec. (3.27)}$$

$$\ddot{\psi} = \left\{ [\alpha(v_l + v_r) - 2(\beta + f_w)\dot{\theta} + 2\beta\ddot{\psi}\sin\psi](MLR\cos\psi - 2n^2 J_m) - [(2m + M)^2 + 2J_w + 2n^2 J_m][MgL \sin\psi + ML^2 \dot{\phi}^2 \sin\psi \cos\psi - \alpha(v_l + v_r) + 2\beta\dot{\theta} - 2\beta\dot{\psi}] \right\} \left(\frac{1}{\Delta} \right) \quad \text{Ec. (3.28)}$$

Para encontrar $\ddot{\psi}$ en la ecuación 3.25 reemplazando en 3.26 se obtiene la ecuación 3.29.

$$\ddot{\theta} = \left\{ [MgL \sin\psi + ML^2 \dot{\phi}^2 \sin\psi \cos\psi - \alpha(v_l + v_r) + 2\beta\dot{\theta} - 2\beta\dot{\psi}](MLR\cos\psi - 2n^2 J_m) - [ML^2 + J_\psi + 2n^2 J_m][MLR\dot{\psi}^2 \sin\psi + \alpha(v_l + v_r) - 2(\beta + f_m)\dot{\theta} + 2\beta\dot{\psi}] \right\} \left(\frac{1}{\Delta} \right) \quad \text{Ec. (3.29)}$$

3.2 Variables en espacio de estados del modelo

A partir del modelamiento matemático se obtiene la matriz de estados que se muestra en la ecuación 3.30.

$$x = [\psi, \dot{\psi}, \theta, \dot{\theta}]^T = [x_1, x_2, x_3, x_4]^T \quad \text{Ec. (3.30)}$$

El vector de entrada se muestra en la ecuación 3.31:

$$U = [v_{izq} \ v_{der}]^T = [u_1 \ u_2]^T \quad \text{Ec. (3.31)}$$

Donde U representa las señales de control de voltaje, u_2 para la rueda derecha y u_1 para la rueda izquierda. La ecuación de estados está en función de los estados y la entrada, y la función de entradas y salida como se muestra en la ecuación 3.32.

$$\dot{X}(t) = f(t, X(t), U(t)) \quad \text{Ec. (3.32)}$$

La ecuación de salida considera todos los estados pueden ser medidos es decir a través de los sensores para tener un sistema retroalimentado, el modelo en espacio de estados está dado por la ecuación 3.33, 3.34, 3.35, 3.36, 3.37.

$$\Delta_e = (MLR \cos x_1 - 2n^2 J_m)^2 - [(2m + M)R^2 + J_w + 2n^2 J_m](ML^2 + J_\psi + 2n^2 J_m) \quad \text{Ec. (3.33)}$$

$$\dot{x}_1 = x_2 \quad \text{Ec. (3.34)}$$

$$\begin{aligned} \dot{x}_2 = \{ & [\alpha(u_2 + u_1) - 2(\beta + f_w)x_4 + 2\beta x_2 \\ & + MLRx_2^2 \sin x_1](MLR \cos x_1 - 2n^2 J_m) \\ & - [(2m + M)R + 2J_w + 2n^2 J_m][MgL \sin x_1 \\ & + ML^2 x_6^2 \sin x_1 \cos x_1 - \alpha(u_2 + u_1) + 2\beta x_4 \\ & - 2\beta x_2] \left(\frac{1}{\Delta_e} \right) \end{aligned} \quad \text{Ec. (3.35)}$$

$$\dot{x}_3 = x_4 \quad \text{Ec. (3.36)}$$

$$\begin{aligned} \dot{x}_4 = \{ & [MgL \sin x_1 \\ & + ML^2 x_6^2 \sin x_1 \cos x_1 - \alpha(u_2 + u_1) + 2\beta x_4 \\ & - 2\beta x_2](MLR \cos x_1 - 2) \end{aligned} \quad \text{Ec. (3.37)}$$

Para encontrar las matrices A, B, C es necesario linealizar el sistema para el cual se utiliza el método de mínimos cuadrados en el cual se trabaja en los puntos de equilibrio del sistema. A partir de las ecuaciones 3.33, 3.34, 3.35, 3.36 y 3.37 se obtiene las matrices de estado.

Se obtiene las ecuaciones de movimiento en su punto de equilibrio, significa que se considera que $\psi \rightarrow 0$, debido a esto se puede asumir que $(\sin \psi \rightarrow \psi, \cos \psi \rightarrow 1)$, y reemplazando en las ecuaciones 3.28, 3.29, se obtienen las matrices mostradas en las ecuaciones 3.38, 3.39, 3.40, y mediante el modelo matemático se obtienen las ecuaciones 3.41, 3.42, 3.43, 3.44, 3.45, 3.46, 3.47 y 3.48. (Villacres & Viscaino, 2016)

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & A32 & A33 & A34 \\ 0 & A42 & A44 & A44 \end{bmatrix} \quad \text{Ec. (3.38)}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ B3 & B3 \\ B4 & B4 \end{bmatrix} \quad \text{Ec. (3.39)}$$

Donde:

$$H1 = \begin{bmatrix} (2m + M)R^2 + 2J_w + 2n^2J_m & MLR - 2n^2J_m \\ MLR - 2n^2J_m & ML^2 + J_\psi + 2n^2J_m \end{bmatrix} \quad \text{Ec. (3.40)}$$

$$A32 = -L * M * g * H1(1,2)/\det (H1) \quad \text{Ec. (3.41)}$$

$$A42 = L * M * g * H1(1,1)/\det (H1) \quad \text{Ec. (3.42)}$$

$$A33 = -[(f_w + \beta) * H1(2,2) + 2 * \beta * H1(1,2)]/\det (H1) \quad \text{Ec. (3.43)}$$

$$A43 = [(f_w + \beta) * H1(1,2) + 2 * \beta * H1(1,1)]/\det (H1) \quad \text{Ec. (3.44)}$$

$$A34 = \beta * (H1(2,2) + 2 * H1(1,2))/\det (H1) \quad \text{Ec. (3.45)}$$

$$A44 = -\beta * (H1(1,2) + 2 * H1(1,1))/\det (H1) \quad \text{Ec. (3.46)}$$

$$B3 = \alpha * (H1(2,2)/2 + H1(1,2))/\det (H1) \quad \text{Ec. (3.47)}$$

$$B4 = -\alpha * (H1(1,2)/2 + H1(1,1))/\det (H1) \quad \text{Ec. (3.48)}$$

Remplazando los valores de la Tabla 3.1 en las ecuaciones 3.38, 3.39, 3.40 se obtendrá las matrices, A, B que se muestran en las ecuaciones 3.49, 3.50.

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & -399 & -34457 & 34457 \\ 0 & 238 & 14548 & -14548 \end{bmatrix} \quad \text{Ec. (3.49)}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 23863 & 23863 \\ -2968 & -2968 \end{bmatrix} \quad \text{Ec. (3.50)}$$

Basándose en los estados del sistema x , se controlan ψ (inclinación) y θ (posición) se obtiene la matriz C, colocando el valor de 1 al estado que se controla, la respuesta se indica en las ecuaciones 3.51, 3.52.

$$x = [\psi, \dot{\psi}, \theta, \dot{\theta}] \quad \text{Ec. (3.51)}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \text{Ec. (3.52)}$$

3.3 Diseño del controlador PID

A partir de las variables de estado se realiza el diseño del controlador PID, calculando la función de transferencia del sistema continuo, se tiene la ecuación 3.53.

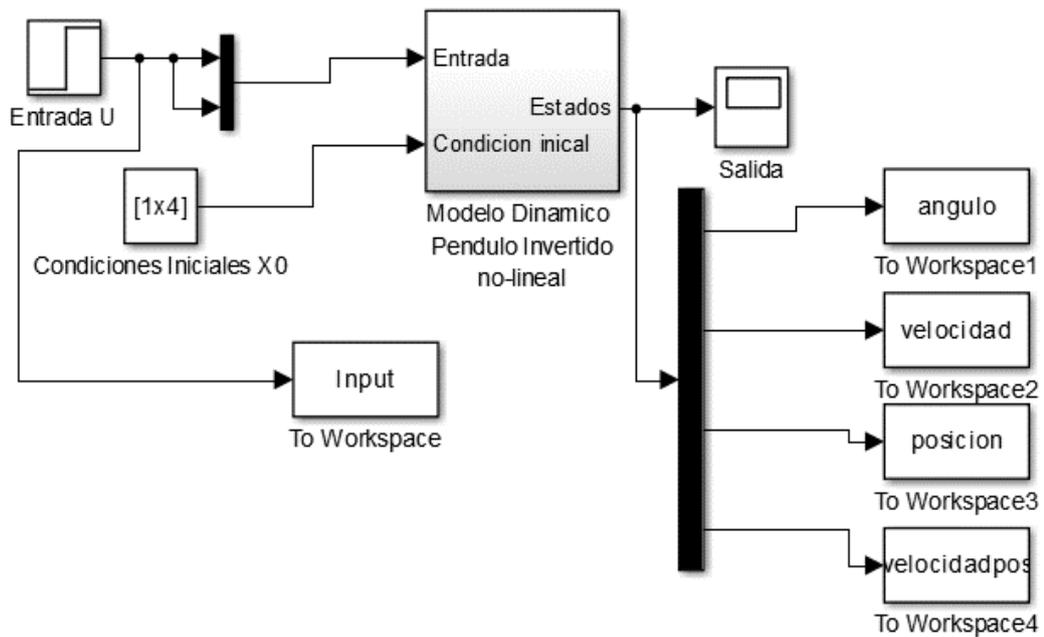
$$G_c(s) = C(sI - A)^{-1} B \quad \text{Ec.(3.53)}$$

La función de transferencia general del control PID se presenta en la ecuación 3.54.

$$G_c(s) = K_p + \frac{K_i}{s} + K_d s \quad \text{Ec.(3.54)}$$

Se representa el sistema no lineal en la Figura 3.5, su entrada y su condición inicial. La entrada es un escalón con valores de U, su estado inicial es de [1,0,1,0] es decir se coloca 1 a las variables a controlarse que son ángulo y posición, la estructura interna del bloque “Modelo Dinámico Péndulo Invertido no-lineal” y programación se muestra en el ANEXO I.

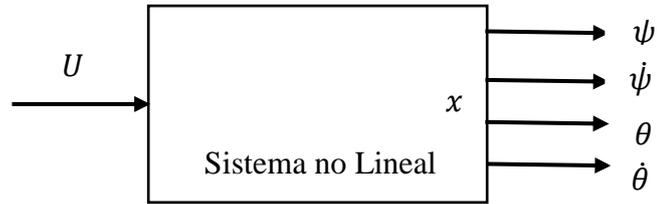
Figura 3.5. Representación del sistema no lineal



Representación del sistema no lineal, Elaborado por: Diana Solórzano

Con los valores de $\psi, \dot{\psi}, \theta, \dot{\theta}$, y mediante la entrada U del sistema en el que se toman datos randomicos en intervalos de ángulo y posición para cada uno de ellos, se tiene la función de transferencia para el ángulo y la variación de posición como se muestra en la Figura 3.6.

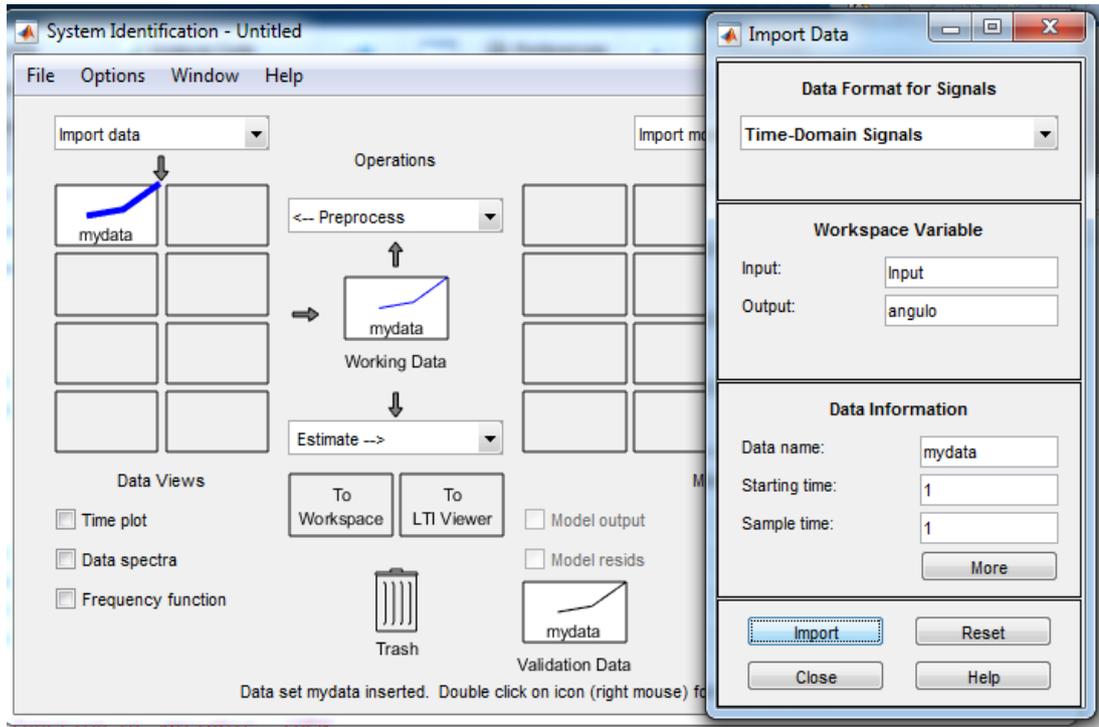
Figura 3.6: Representación de entradas y salidas para la identificación



Representación de entradas y salidas, Elaborador por: Diana Solórzano

En la Figura 3.7 se muestra la obtención de la función de transferencia para el ángulo, en donde se trabaja en el dominio del tiempo.

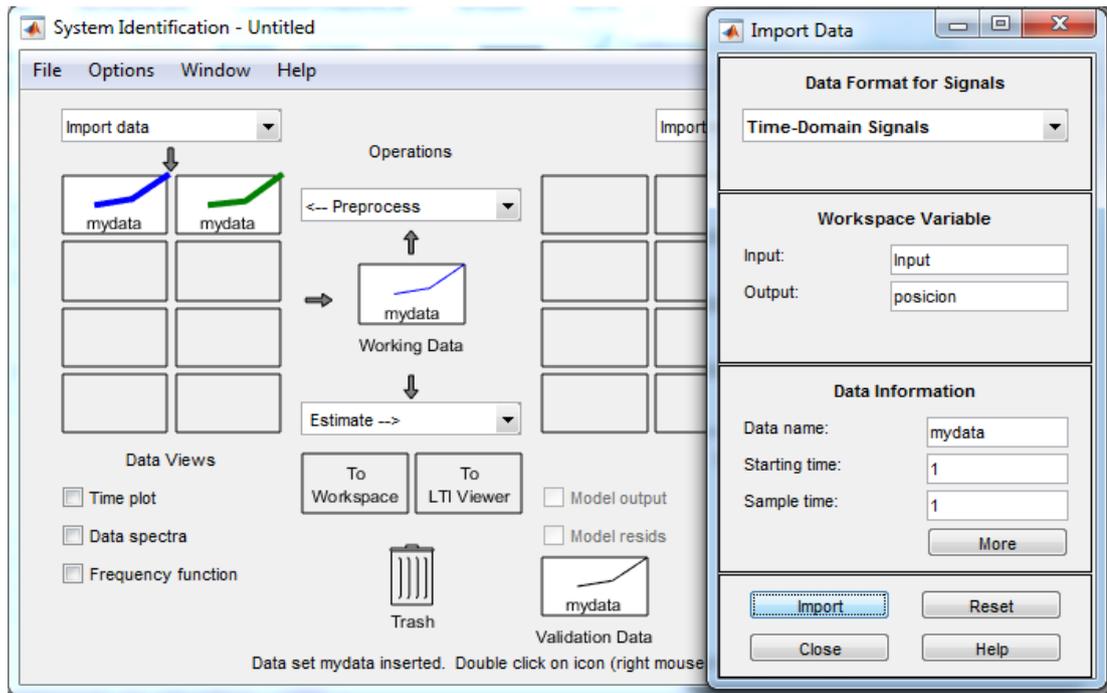
Figura 3.7: IDEN de Matlab para la función de transferencia del ángulo



Función de transferencia para el control de ángulo, Elaborado por: Diana Solórzano.

En la Figura 3.8 se obtiene los datos del modelo, para ello mediante la herramienta IDEN de Matlab se obtiene la función de transferencia para el control de posición mediante los datos obtenidos de la Figura 3.5.

Figura 3.8: IDEN de Matlab para la función de transferencia de la posición



Obtención de la función de transferencia para el control de posición, Elaborado por: Diana Solórzano.

Obteniendo la función de transferencia para el ángulo con la mejor aproximación de 76.69%, con 3 polos y 1 cero; mostrado en la ecuación 3.55.

$$\frac{\psi}{U} = \frac{1.08e04s + 1.203e04}{s^3 + 9.839s^2 + 42.69s + 36.12} \quad \text{Ec.(3.55)}$$

Se tiene la función de transferencia para la posición con la mejor aproximación 77.98% con 2 polos, como se muestra en la ecuación 3.56.

$$\frac{\theta}{U} = \frac{-1456}{s^2 + 3.52s + 0.0001725} \quad \text{Ec.(3.56)}$$

En la Tabla 3.2 se encuentran los valores de los compensadores PID mediante el comando TUNE, se encuentren valores para cada función de transferencia, el PID1 representa los valores para el control de ángulo, y el PID 2 representa los valores para el control de posición.

Tabla 3.2 Valores de las constantes PID

	Kp	Ki	Kd
PID 1	0.6	12.2	0.005
PID 2	1.2	13.8	0.003

Valores de los compensadores, Elaborado por: Diana Solórzano

Se realiza la sintonización del PID que busca reducir la integral del error absoluto (IAE), los algoritmos genéticos realizan una aproximación heurística, que busca optimizar basado en un criterio específico se detalla en el ANEXO II, con los cual se buscan los mejores valores de los compensadores K_p , K_i , K_d , que se muestran en la Tabla 3.3. Dichos valores son utilizados para la simulación de sistema. (Pillajo, Bonilla, & Hincapié, 2016).

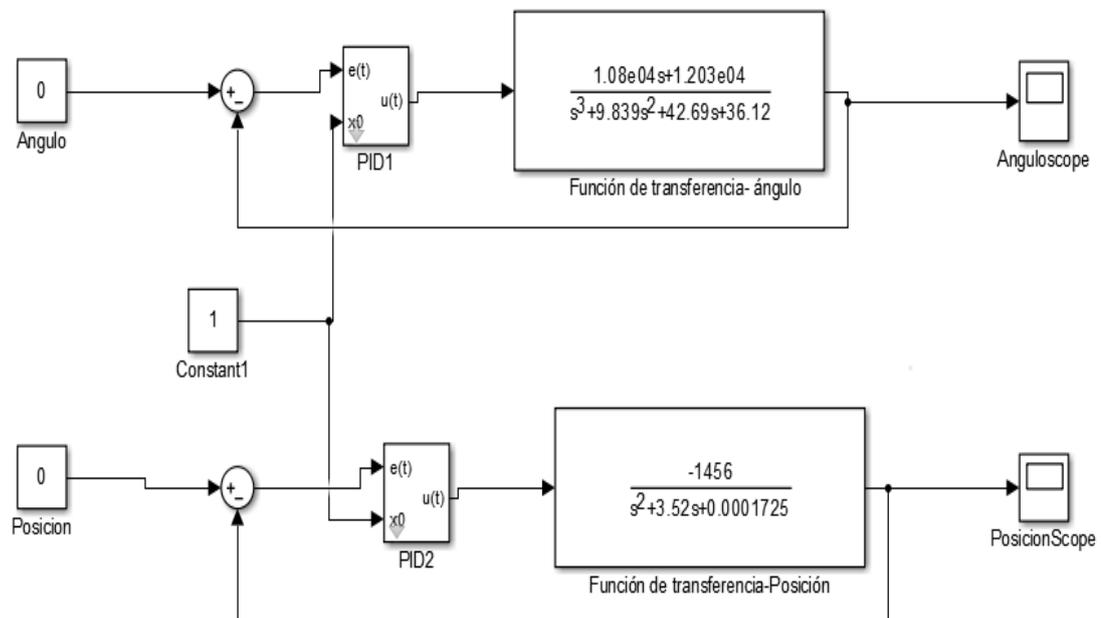
Tabla 3.3: Valores de constantes PID mediante algoritmos genéticos

	K_p	K_i	K_d
PID 1	0.7473	10.37	0.009981
PID 2	1.296	13.612	0.00621

Valores de los compensadores, Elaborado por: Diana Solórzano

En la Figura 3.9 se muestra la simulación del control PID en Simulink con las dos funciones de transferencia para ángulo y posición, y los compensadores PID para cada uno de ellos.

Figura 3.9: Simulación del controlador PID



Simulación del control PID en Simulink, Elaborado por: Diana Solórzano

La programación que se realiza en el software Matlab para obtener las señales de ángulo y posición se muestra en la Figura 3.10, el diagrama de flujo que describe cada uno de los pasos que se realizan para la simulación de los controladores, y la sintonización de los compensadores del control PID.

Figura 3.10: Diagrama de flujo de la programación en Matlab.

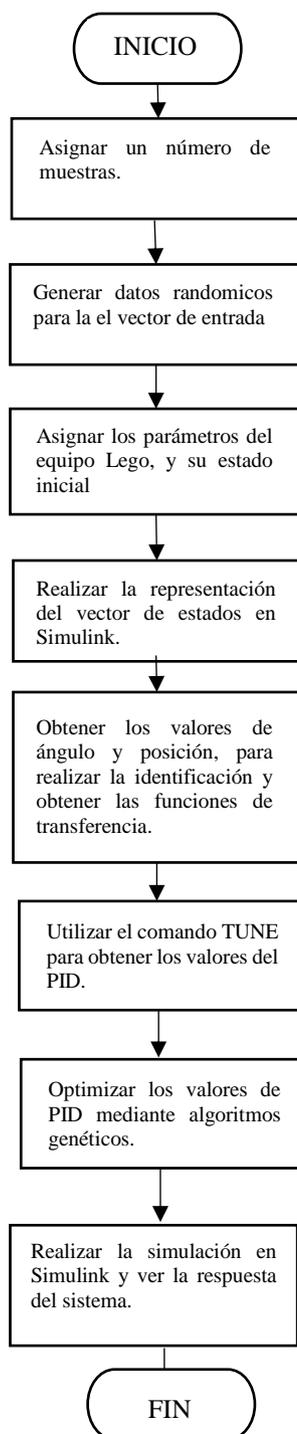


Diagrama para la simulación del control PID en Matlab, Elaborado por: Diana Solórzano

Para la implementación del control PID en el sistema real, se realiza la programación en el software RobotC, para ello se detalla los pasos a seguir mediante el diagrama de flujo que se muestra en la Figura 3.11.

Para la implementación del equipo en la planta real, es necesario realizar el acondicionamiento del sensor giroscopio para ello se necesita el valor de la

compensación offset. El sensor giroscopio mide el valor del ángulo de inclinación ψ y la velocidad del ángulo $\dot{\psi}$. Se realiza la obtención de la velocidad del ángulo de inclinación mediante el sensor Hitechnic NXT cuando el sensor está inactivo retorna un valor de 600, y a este valor se le conoce como offset en cambio cuando está en funcionamiento si se retorna un valor mayor significa que está rotando en sentido horario, en caso contrario si el valor es menor gira en sentido antihorario.

El valor de offset puede variar por diferentes factores externos, como el ruido que se introduce al sistema por lo que se necesita una medida confiable de offset, y se mantiene un valor de 300 muestras, con el sistema sin rotación y a partir de esto determinar un promedio y con este procedimiento se tiene un valor de offset de 606. (Herrera Garzón, 2014).

Se determina el valor del compensador, pero en la realidad es distinto debido a que es necesario ir ajustando el valor del offset, ya que se produce un desvío (drift) con el tiempo, para mejorar el valor se toma el promedio por lo que se realiza mediante la con media móvil exponencial, que se obtiene mediante la ecuación 3.57.

$$\dot{\psi}_{Off}[k] = \dot{\psi}_{Off} + \beta(\dot{\psi}_{Rw} - \dot{\psi}_{Off}[k - 1]) \quad \text{Ec.(3.57)}$$

Donde:

$\dot{\psi}_{Off}[k]$ = Valor actual offset

$\dot{\psi}_{Off}[k - 1]$ = Valor anterior del offset.

$\dot{\psi}_{Rw}$ = Valor actual.

β = factor de ponderación.

El factor β será un valor pequeño que garantiza que el pendulo invertido cuando este en equilibrio no tenga cambios inmediatos del offset, por un tiempo significativo prolongado el valor será elevado, a la medida móvil exponencial se compara como un filtro pasabajo.

Una vez obtenido el valor offset con la compensación y el filtro pasa bajo, el valor de la velocidad del ángulo ψ se muestra en la ecuación 3.58. (Herrera Garzón, 2014).

$$\dot{\psi} = \dot{\psi}_{Raw} - \dot{\psi}_{Offset} \quad \text{Ec.(3.58)}$$

Donde:

$\dot{\psi}_{Offset}$ = Valor offset compensado y filtrado.

Luego de realiza la implementación del control PID y se realiza la programación en el software RobotC presentado en el ANEXO III. Se procede a descargar al equipo Lego Mindstoms. Los valores que se utilizan en la implementación, del PID se muestran en la Tabla 3.4, el valor de PID 1 se utilizan para el control de ángulo y el PID 2 se utiliza para el control de posición.

Tabla 3.4: Valores de implementación del PID

	Kp	Ki	Kd
PID 1	0.40	12.2	0.005
PID 2	0.20	14.2	0.005

Valores de los compensadores en el sistema real, Elaborado por: Diana Solórzano

En la Figura 3.11 se muestra en diagrama de flujo para la programación del control PID en el software RobotC, donde se muestran los partes principales utilizados en la programación en lenguaje C.

Figura 3.11: Diagrama de flujo para la implementación del control PID

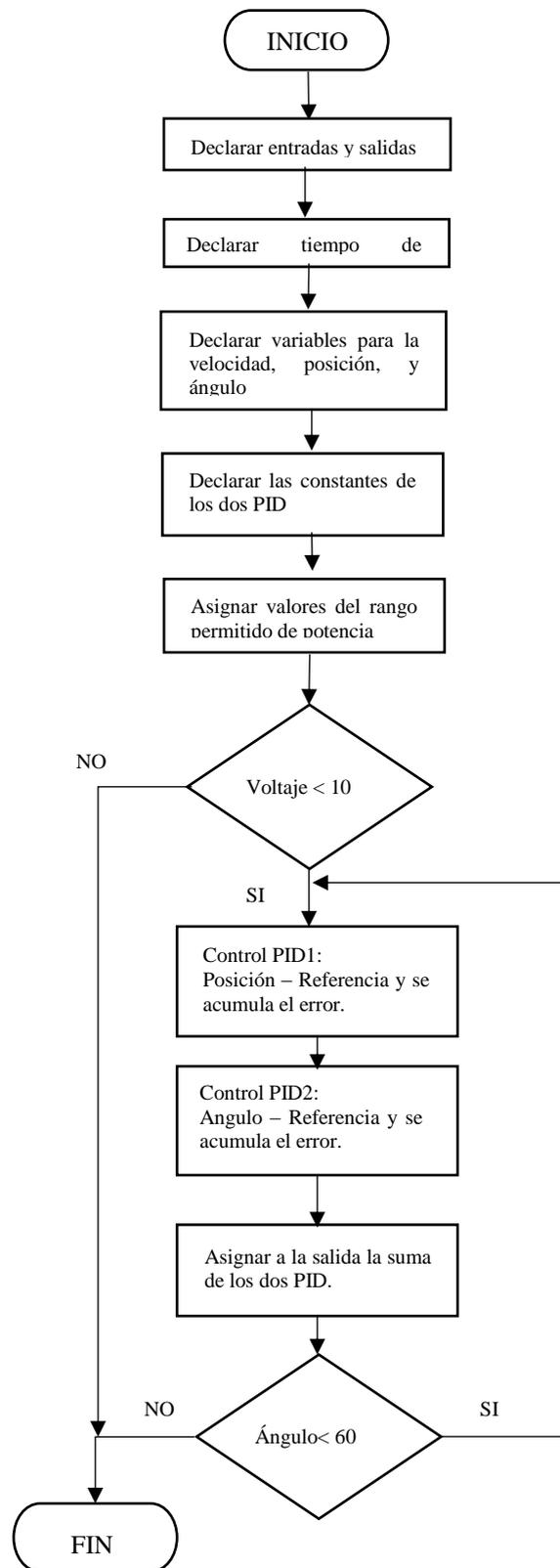


Diagrama de flujo de la programación en RobotC, Elaborado por: Diana Solórzano

3.4 Diseño del controlador LQG

El diseño del controlador se lo realiza mediante varios comandos de Matlab, el que se utiliza el comando $lqr()$, en donde se calcula la constante K del controlador LQR, para ello se necesitan las matrices de pesos Q y R. Se representa como se muestra en la ecuación 3.59.

$$K = lqr(A, B, Q, R) \quad \text{Ec.(3.59)}$$

Las matrices Q representa el error del estado, y la matriz R representa el esfuerzo de control. La matriz de ganancia P se determina por la ecuación de Riccati que se muestra en la ecuación 3.60.

$$A^T P + P A - P B Q_u^{-1} B^T P + Q_x = 0 \quad \text{Ec.(3.60)}$$

Donde:

P= Matriz definida positiva.

Se tiene que elegir las matrices Q y R. Para la estimación de la matriz Q se utilizan valores escalares en su diagonal y cero a los demás valores, luego de ello se busca minimizar el error y aumentar su función de coste. Para representar los valores que deben ser modificados en la matriz Q se muestra en la ecuación 3.61 y la matriz R depende de las entradas del sistema como se muestra en la ecuación 3.62.

$$Q = \begin{pmatrix} \psi & 0 & 0 & 0 \\ 0 & \dot{\psi} & 0 & 0 \\ 0 & 0 & \theta & 0 \\ 0 & 0 & 0 & \dot{\theta} \end{pmatrix} \quad \text{Ec.(3.61)}$$

$$R = \begin{pmatrix} u_l & 0 \\ 0 & u_r \end{pmatrix} \quad \text{Ec.(3.62)}$$

Aunque no se tiene una regla específica para la ponderación de los valores, es necesario realizar la simulación del sistema y modificar sus valores buscando la mejor aproximación de la salida deseada. Se debe tomar en cuenta que se debe cumplir las condiciones que se muestran en las ecuaciones 3.63 y 3.64.

$$Q = Q^T \geq 0 \quad \text{Ec.(3.63)}$$

$$R = R^T > 0 \quad \text{Ec.(3.64)}$$

Se obtiene la función de costo J del control LQG para ello se busca optimizar los valores de Q y R para lo cual se realiza el diagrama de flujo de la programación realizada en Matlab para obtener los valores, se muestra en la Figura 3.12.

Figura 3.12: Diagrama de flujo para encontrar las matrices Q y R.

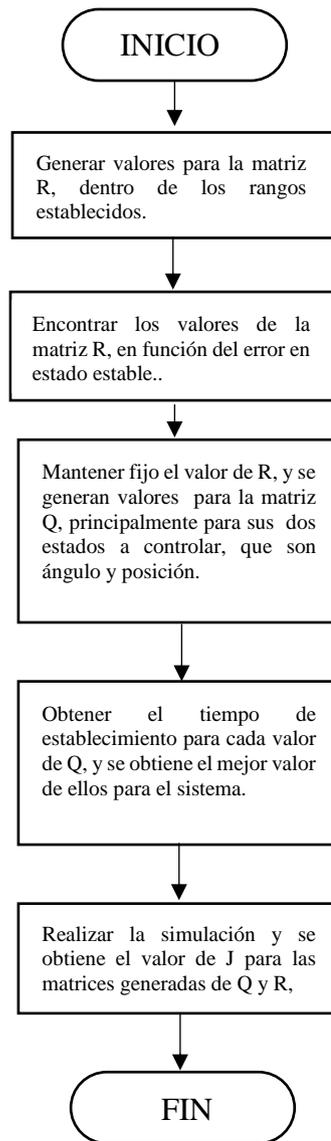


Diagrama de flujo de la obtención de la matriz Q y R. Elaborado por: Diana Solórzano.

Se se observa la mejor aproximación de la matrices Q y R ,que se muestra en a con un valor de $J = 2.11$. A partir de valores que se obtuvieron de la aproximación se muestran las matrices en las ecuaciones 3.65 y 3.66.

$$Q = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}$$

Ec.(3.65)

$$R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{Ec.(3.66)}$$

Obteniendo así una matriz K, que se tiene en la ecuación 3.67.

$$K = \begin{bmatrix} 7.0710 & -14.1534 & -0.0158 & -0.0422 \\ 7.0710 & -14.1534 & 0.0158 & 0.0422 \end{bmatrix} \quad \text{Ec.(3.67)}$$

Para obtener la constante de Kalman L , se aplica la siguiente fórmula mostrada en la ecuación 3.68 y 3.69.

$$R_{pn} = (A * Q_n * A') + Q_n \quad \text{Ec.(3.68)}$$

$$L = R_{pn} * C' * \text{inv}(C * R_{pn} * C' * R_n) \quad \text{Ec.(3.69)}$$

Donde:

$Q_n = V_n^2$ Matriz de covarianza del error.

$R_n = W_n^2$ Matriz de covarianza del error por parte del sensor giroscopio.

$W_n = 0.04$ Desviación del observador.

$V_n = 0.08$ Desviación del estado.

$X_{an} = [0; 0; 0; 0]$

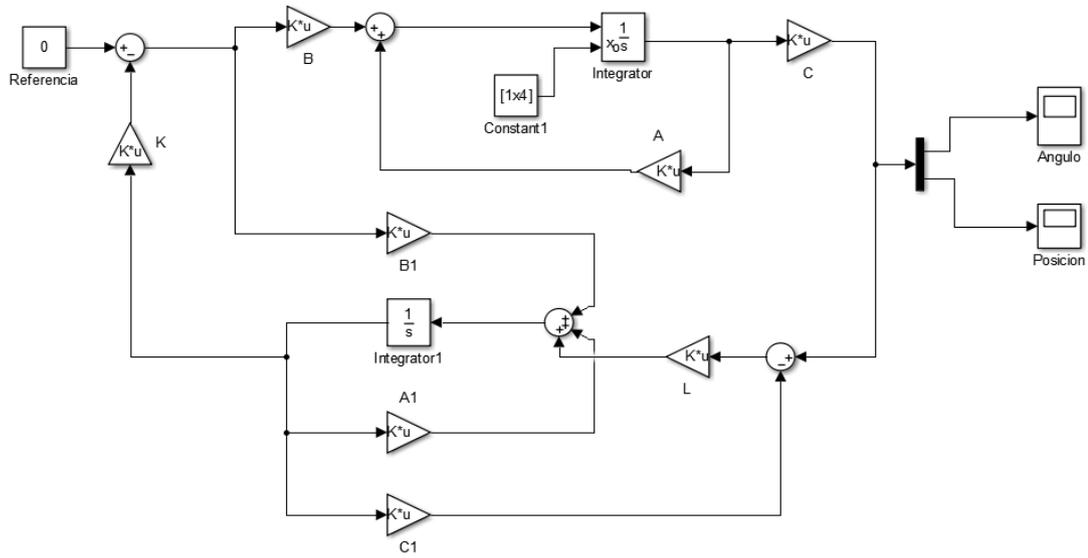
$X_p = [0; 0; 0; 0]$

Aplicando las ecuaciones 3.68 y 3.69 se obtiene la matriz que se muestra en la ecuación 3.70.

$$L = \begin{bmatrix} 1250 & 1250 & -1861.2 & 1861.2 \\ -0.1 & 0.1 & 4041.8 & -1541.38 \end{bmatrix} \quad \text{Ec.(3.70)}$$

En la Figura 3.13 se presenta la simulación del control LQG en Simulink de Matlab, en representación de sus matrices de estado y el observador como estimador de Kalman; mediante las matrices calculadas del modelamiento matemático.

Figura 3.13: Simulación del control LQG



Simulación en Simulink del control LQG y sus salidas, Elaborado por: Diana Solórzano.

Donde:

$$A = A1 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & -399 & -34457 & 34457 \\ 0 & 238 & 14548 & -14548 \end{bmatrix}$$

$$B = B1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 23863 & 23863 \\ -2968 & -2968 \end{bmatrix}$$

$$C = C1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Constant1 = Valor inicial [1,0,1,0]

El filtro de Kalman se aplica la ecuación 3.71, 3.72, 3.73, 3.74, y 3.75, para la estimación en la implementación, en donde se utilizan las matrices de estado

$$x_e = Ax_e + B u_e \quad \text{Ec.(3.71)}$$

$$u_e = u_{LQR} + u_{Kalman} \quad \text{Ec.(3.72)}$$

$$u_{LQR} = -K \cdot x \quad \text{Ec.(3.73)}$$

$$u_{Kalman} = -L \cdot x_e \quad \text{Ec.(3.74)}$$

$$U = u_e \quad \text{Ec.(3.75)}$$

En la Figura 3.14 se muestran los pasos para realizar la simulación del controlador LQG, en representación de diagrama de flujo, donde se muestran las partes principales de la programación, el código C implementado en RobotC se muestra en el ANEXO IV.

Figura 3.14: Diagrama de flujo para la simulación del control LQG

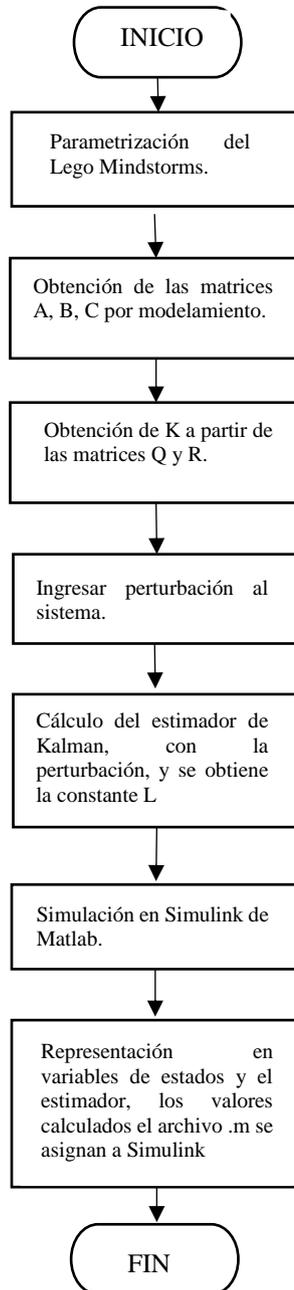


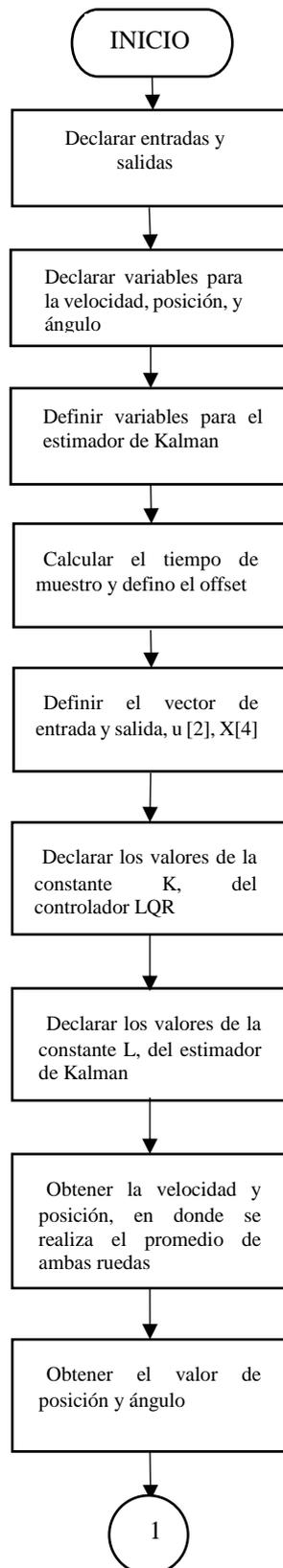
Diagrama de flujo de simulación del control LQG en Matlab y Simulink,

Elaborado por: Diana Solórzano

Para la implementación del control LQG, en el software RobotC se toma en cuenta la ley de control $u = -Kx$, donde se declara el vector de entrada y el vector K obtenido

de Matlab, luego de ello se realiza el filtro de Kalman que. De la misma manera se coloca el vector calculado y se actualiza el valor deseado. En la figura 3.15 se muestra el diagrama de flujo de la implementación en el sistema real.

Figura 3.15: Diagrama de flujo para la implementación del control LQG



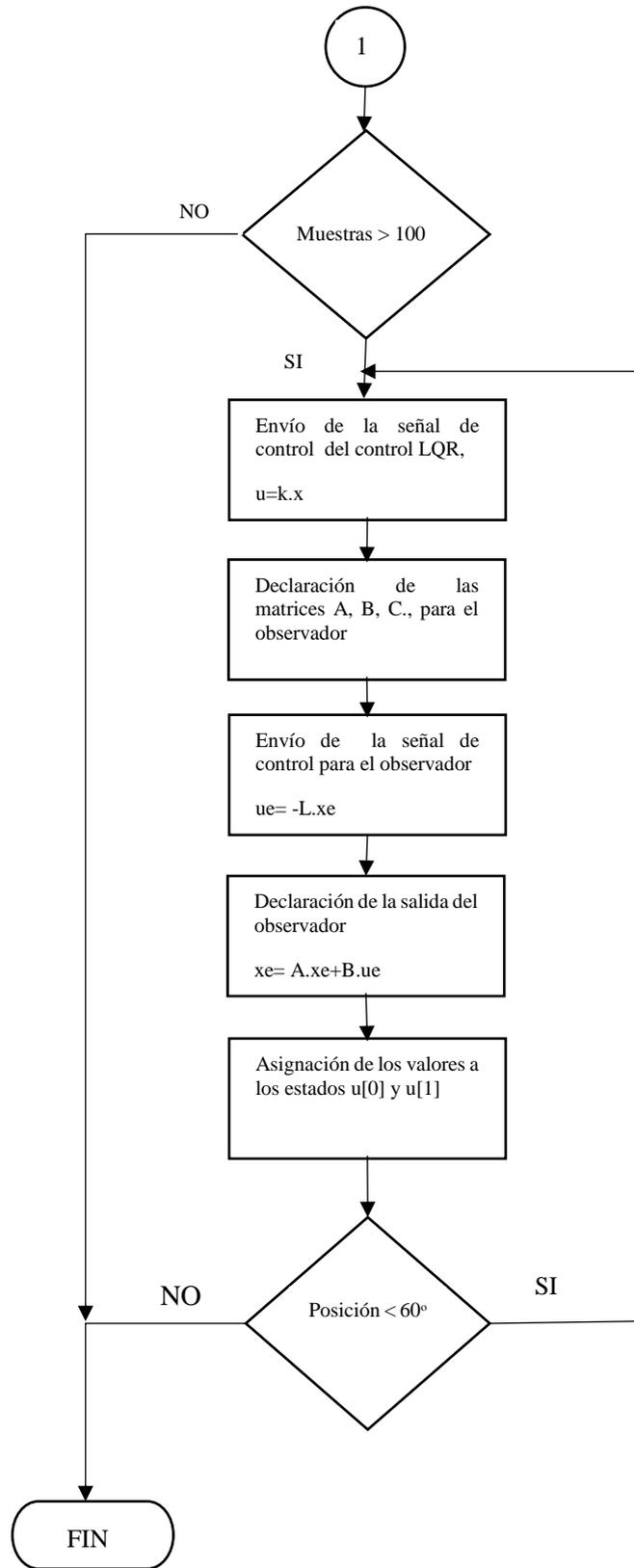


Diagrama de flujo de la programación en RobotC, Elaborado por: Diana Solórzano.

CAPÍTULO 4

PRUEBAS Y RESULTADOS

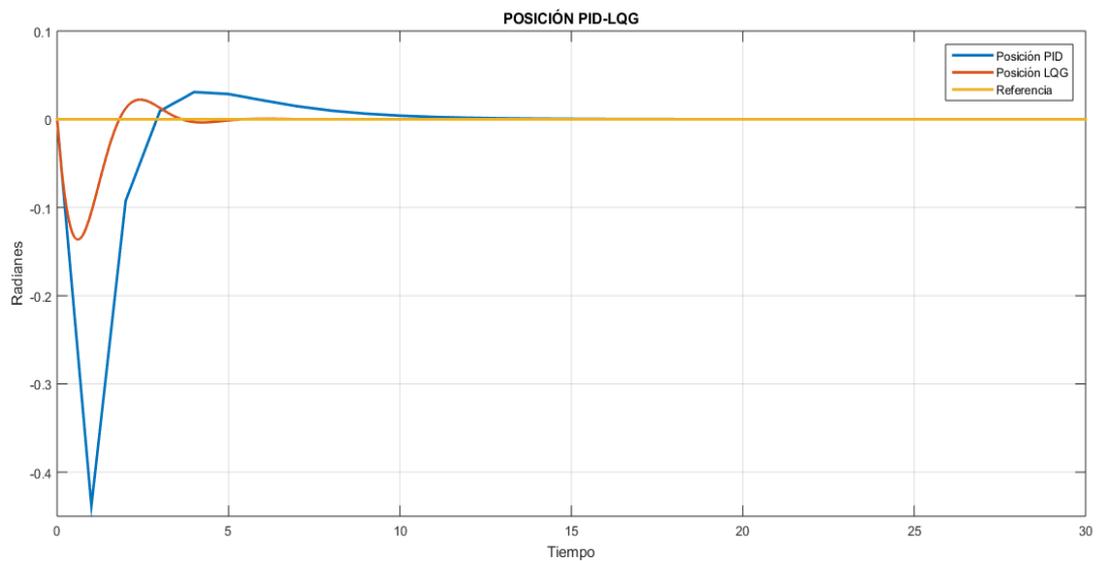
En este capítulo se realizarán pruebas de la implementación de los controladores PID y LQG en simulación y sobre el equipo real, diseñados en el capítulo 3. Para la obtención de las respuestas del sistema en simulación y en el equipo real se plantea la realización de 5 pruebas, donde se analizará los resultados de cada una de ellas.

- **Prueba 1:** En esta prueba se realiza la aplicación de los controladores PID y LQG para el control de posición en simulación, aplicando los controladores a la función de transferencia y al sistema en espacio de estados respectivamente, de lo cual se obtendrá valores de máximo sobreimpulso, tiempo de asentamiento e índices de rendimiento.
- **Prueba 2:** En esta prueba se realiza la aplicación de los controladores PID y LQG para el control de ángulo en simulación, aplicando los controladores a la función de transferencia y al sistema en espacio de estados respectivamente.
- **Prueba 3:** En esta prueba se realiza la aplicación de los controladores PID y LQG para el control de posición en el equipo Lego Mindstorms, de lo cual se obtendrá valores de máximo sobreimpulso, tiempo de asentamiento e índices de rendimiento.
- **Prueba 4:** En esta prueba se realiza la aplicación de los controladores PID y LQG de ángulo en el equipo real Lego Mindstorms, de lo cual se obtendrá valores de, máximo sobreimpulso, tiempo de asentamiento e índices de rendimiento.
- **Prueba 5:** En esta prueba se obtienen las gráficas del índice de rendimiento IAE para el control PID, así como la gráfica de la función de coste para el controlador LQG.

4.1 Prueba 1: Control de posición mediante los controladores PID y LQG en simulación.

Para la realización de la prueba 1 se emplea un tiempo de simulación de 30s con un valor de referencia constante de 0 radianes. En la Figura 4.1 se muestra las respuestas del sistema al aplicar el control PID y LQG respectivamente.

Figura 4.1: Respuesta del sistema PID y LQG posición en simulación



Respuesta del control LQG y PID de posición, Elaborado por: Diana Solórzano

En la Tabla 4.1 se muestran los valores de máximo sobreimpulso (**Mp**) y tiempo de asentamiento (**Ts**) obtenidos de la aplicación del control de posición.

Tabla 4.1: Valores obtenidos del sistema PID y LQG posición en simulación

Control	Mp	Ts
PID	-0.43 %	9s
LQG	-0.13 %	5s

Valores de la salida del control LQG y PID

En la Tabla 4.2 se presentan los índices de rendimiento IAE e ISE obtenidos de la realización de la prueba 1.

Tabla 4.2: Índices de rendimiento de los controladores

Control	IAE	ISE
PID	1.01	0.91
LQG	0.17	0.01562

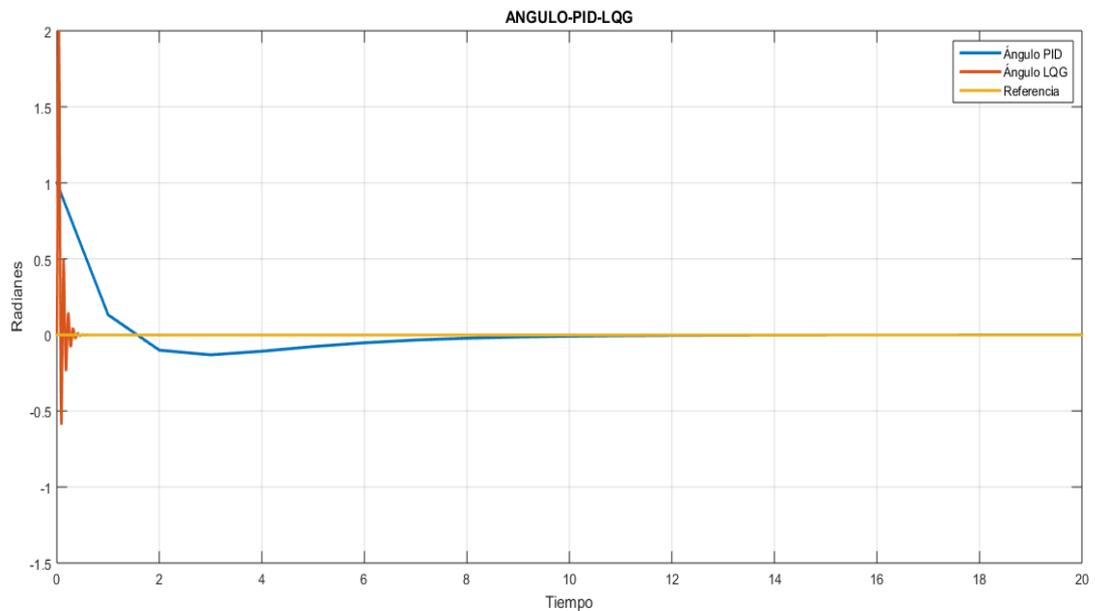
Índices de rendimiento de los controladores PID y LQG, Elaborado por: Diana Solórzano

Con los datos de la Tabla 4.1 y la Figura 4.1 se comprueba que el control LQG tiene un menor sobreimpulso, y se estabiliza 4s menos que el control PID, con los valores obtenidos en la Tabla 4.2 de índices de rendimiento se comprueba que el control LQG tiene mejor respuesta que el controlador PID.

4.2 Prueba 2: Control de ángulo mediante los controladores PID y LQG en simulación

Para la realización de la prueba 2 se emplea un tiempo de simulación de 20s con un valor de referencia constante de 0 radianes. En la Figura 4.2 se muestra las respuestas del sistema al aplicar el control PID y LQG respectivamente, para el control de ángulo.

Figura 4.2: Respuesta del sistema PID y LQG ángulo en simulación



Respuesta del control PID y LQG para el ángulo en simulación, Elaborado por: Diana Solórzano

En la Tabla 4.3 se muestran el máximo sobreimpulso y el tiempo de asentamiento para contrastar los valores obtenidos de los dos controladores.

Tabla 4.3: Valores del sistema PID y LQG ángulo en simulación

Control	Mp	Ts
PID	1%	8s
LQG	2 %	1s

Valores de la salida del control LQG y PID, Elaborado por: Diana Solórzano

En la Tabla 4.4 se muestran los índices de rendimiento del control PID y LQG.

Tabla 4.4: Índices de rendimiento del control de ángulo en simulación.

Control	IAE	ISE
PID	1,13	0.34
LQG	1.03	0.15

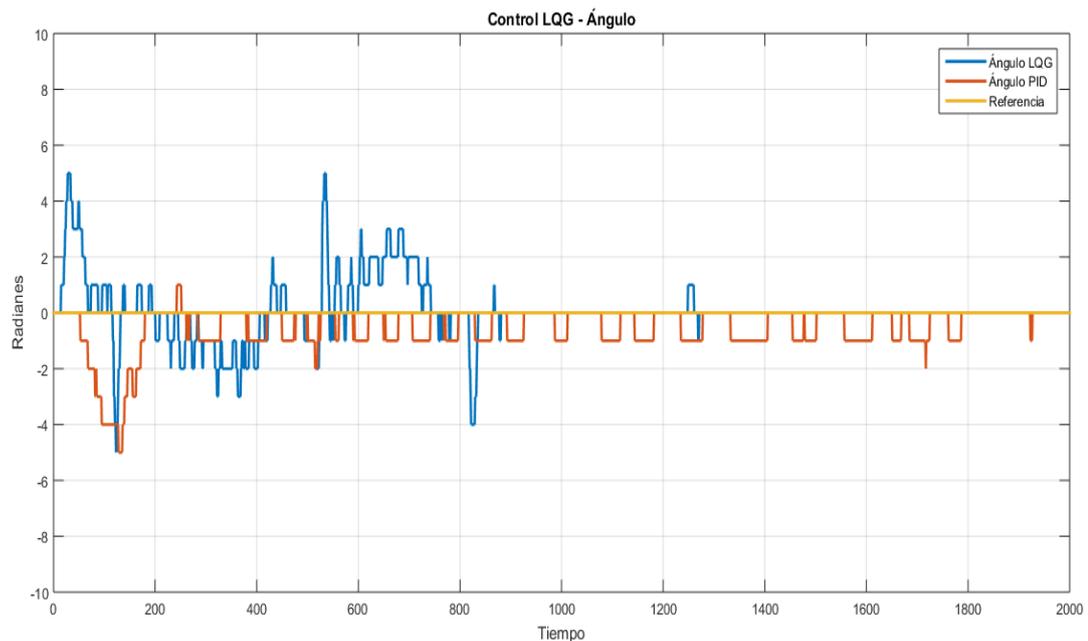
Índices de rendimiento de los controladores PID, LQG, Elaborado por: Diana Solórzano.

Con los datos de la Tabla 4.3 se demuestra que el control PID tiene un menor sobreimpulso, pero se demora en estabilizarse con respecto al control, con los valores obtenidos en la Tabla 4.4 de índices de rendimiento se comprueba que el control LQG tiene mejor rendimiento con respecto al controlador PID.

4.3 Prueba 3: Control de ángulo mediante los controladores PID y LQG en el equipo real

En la prueba 3 se realiza la implementación de los controladores PID y LQG en el sistema real, para obtener los datos de la ejecución del algoritmo, para esta prueba se emplea una referencia de 0 radianes, con un tiempo de simulación de 200 s como se muestra en la Figura 4.3.

Figura 4.3: Respuesta del sistema PID y LQG ángulo en el equipo real



Implementación del control LQG y PID en ángulo, Elaborado por: Diana Solórzano

En la Tabla 4.5 se muestran los valores de máximo sobreimpulso, y el tiempo de establecimiento, de la implementación de los controladores PID y LQG sobre el equipo real para el control del ángulo.

Tabla 4.5: Valores de los controladores PID y LQG de ángulo en el sistema real.

Control	Mp	Ts
PID	5 %	3s
LQG	-5 %	8s

Valores de la salida del control LQG y PID, Elaborado por: Diana Solórzano.

En la Tabla 4.6 se presentan los índices de rendimiento IAE e ISE, del control de ángulo, obtenidos en la prueba 3.

Tabla 4.6: Índices de rendimiento PID y LQG de ángulo en el sistema real

Control	IAE	ISE
PID	1.09	2.93
LQG	1.06	2.31

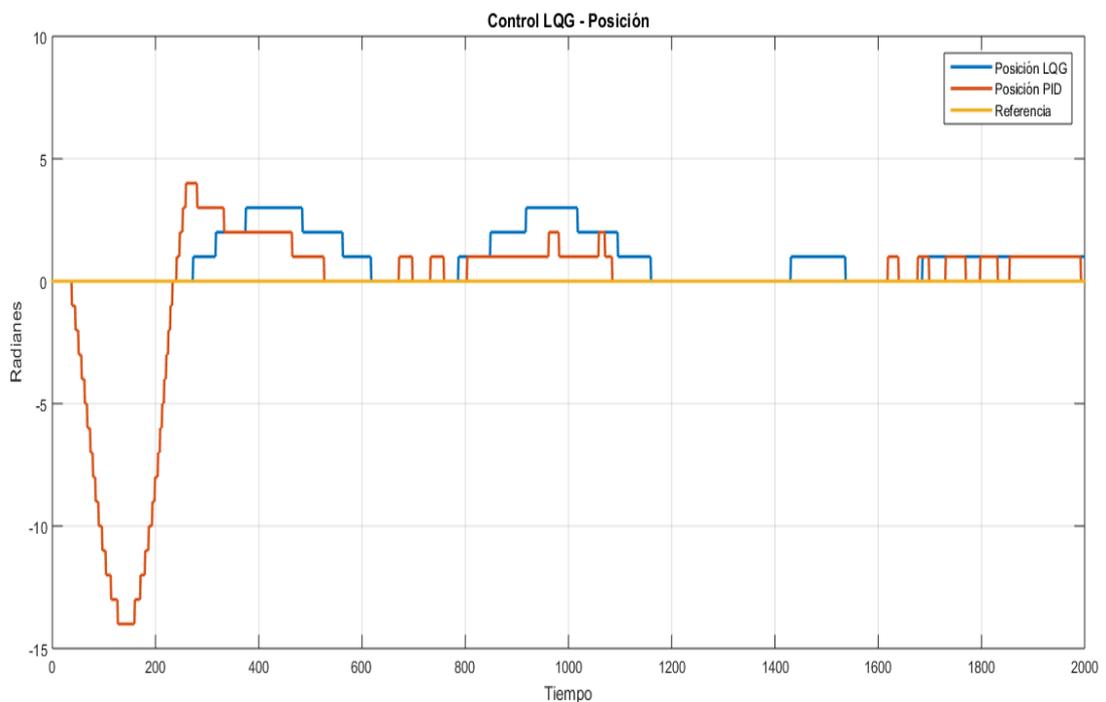
Índices de la salida del control LQG y PID, Elaborado por: Diana Solórzano.

Con los datos de la Tabla 4.5 se muestra que ambos controladores tienen el mismo máximo sobreimpulso, pero el control LQG se demora 5s más en estabilizarse, pero no genera oscilaciones como el control PID como se muestra en la Figura 4.3. Mediante los valores de la Tabla 4.6 el controlador LQG tiene mejor rendimiento.

4.4 Prueba 4: Control de posición mediante los controladores PID y LQG en el equipo real

En la prueba 4 se realiza la implementación de los controladores en el equipo Lego Mindstorms, se toma las gráficas de posición para los dos controladores LQG y PID respectivamente, y la respuesta se muestra en la Figura 4.4.

Figura 4.4: Respuesta del sistema PID y LQG posición en el equipo real



Implementación del control LQG y PID en posición en el equipo real, Elaborado por: Diana Solórzano.

En la Tabla 4.7 se muestran los valores de máximo sobreimpulso, y tiempo de establecimiento de la implementación en el equipo real, de los controladores PID y LQG para el control de posición.

Tabla 4.7: Valores del Sistema PID y LQG de posición en el equipo real

Control	Mp	Ts
PID	-14 %	11s
LQG	2.5 %	11.5 s

Valores de la salida del control LQG y PID, Elaborado por: Diana Solórzano

En la Tabla 4.8 se muestran los índices de rendimiento de las salidas de posición de los controladores.

Tabla 4.8: Índices de rendimiento de la salidas del sistema

Control	IAE	ISE
PID	3.56	2.21
LQG	2.61	4.44

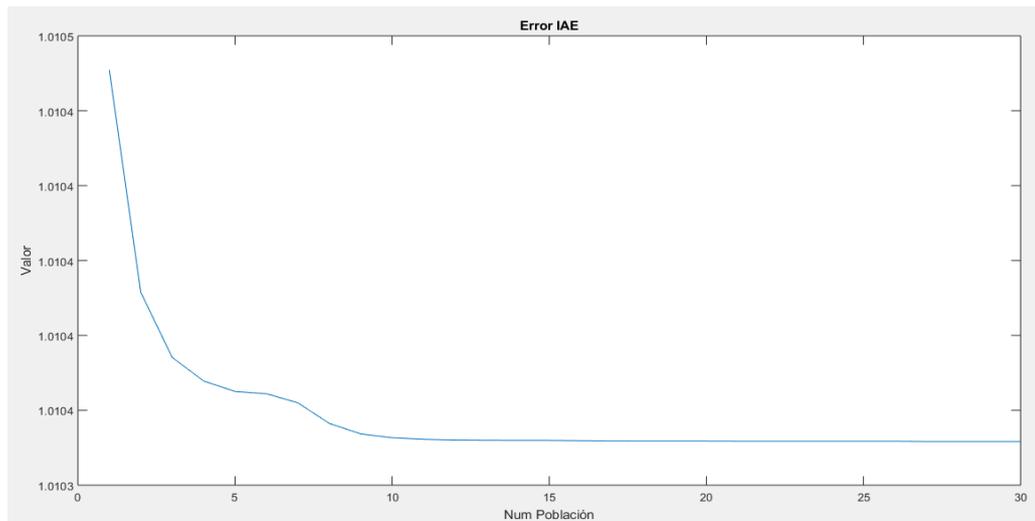
Tabla de comparación de los índices de rendimiento, Elaborado por: Diana Solórzano

En la Tabla 4.7 se tiene que el control PID tiene mayor sobreimpulso debido al desplazamiento que realiza el equipo antes de estabilizarse, el tiempo de asentamiento es similar para ambos controladores. En la Tabla 4.8 muestra que el control PID es igual de eficiente que el control LQG.

4.5 Prueba 5: Gráficas de índices de rendimiento IAE para el control PID y función de costo para el control LQG.

Para comprobar la respuesta de los controladores es necesario realizar la gráfica correspondiente al índice de rendimiento IAE que es la integral del error absoluto, con ello se verifica el error que existe entre la referencia y la salida del sistema. En la Figura 4.5 se muestra el índice de error IAE para el control de ángulo, mediante su función de transferencia.

Figura 4.5: Índice de rendimiento IAE para el ángulo.



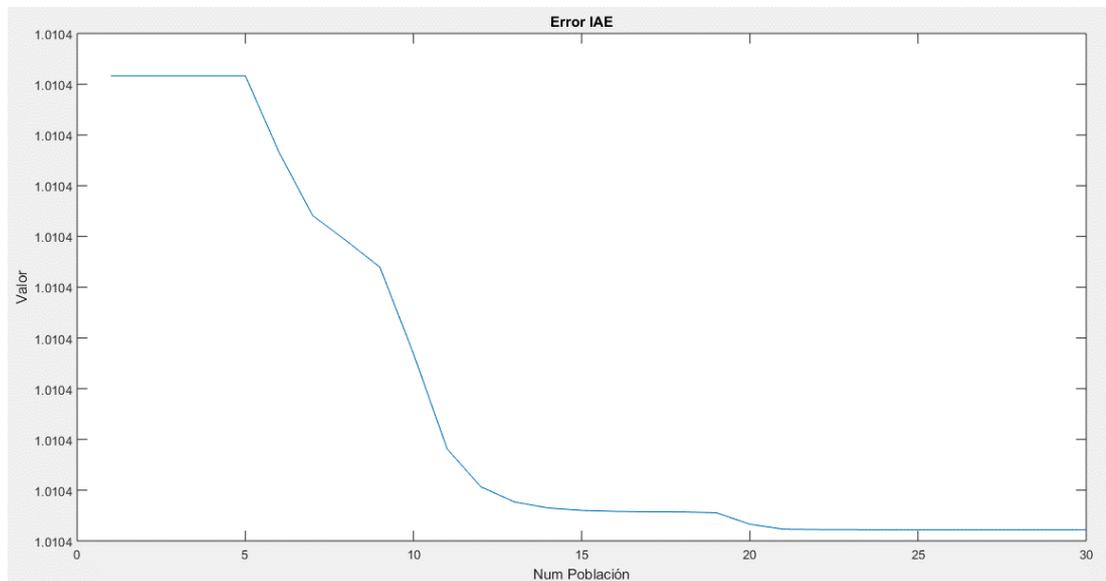
Gráfica del índice de rendimiento IAE, para el control de ángulo del PID

Elaborado por: Diana Solórzano.

Se tiene el valor de 1,01 del índice IAE, el cual se puede decir que el valor que se tiene en la salida es la óptima debido a la aproximación encontrada.

Se tiene también el índice de rendimiento para el control de posición, encontrada mediante la función de transferencia que se aplicó para el controlador PID, el resultado del IAE que se muestra en la Figura 4.6.

Figura 4.6: Índice de rendimiento IAE para la posición.



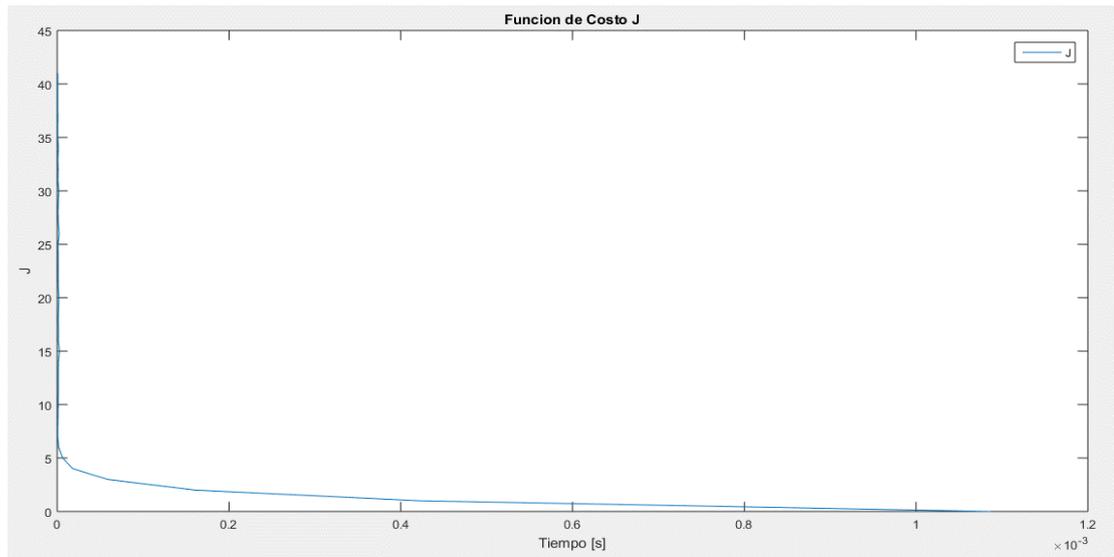
Gráfica del índice de rendimiento IAE, para el control de posición para el control PID

Elaborado por: Diana Solórzano.

Se tiene el valor de 1,01 del índice IAE, el cual se puede decir que el valor que se tiene en la salida es la óptima debido a la aproximación encontrada.

Los valores encontrados están basados en el error en estado estable para encontrar el valor de R se toma el valor más bajo de error, para la matriz Q se basa en el tiempo de establecimiento, de la misma manera se toma el mínimo valor; obtenidos dichos valores se encuentra la función de costo que se muestra en la Figura 4.7.

Figura 4.7: Respuesta de la función de costo vs Tiempo.



Función de costo de los valores óptimos Q y R. Elaborado por: Diana Solórzano.

CONCLUSIONES

Al investigar y recopilar información de las características y parámetros físicos, que intervienen en el equipo Lego Mindstorms en configuración de péndulo invertido, se pudo obtener una aproximación del modelo matemático del sistema, para el desarrollo de los controladores LQG y PID.

A partir del modelamiento matemático en variables de estado se desarrolla una función en Matlab que describe el modelo dinámico del sistema para obtener las funciones de transferencia de las variables a controlar, las cuales se utilizaron para simular el controlador PID y de esta manera obtener las constantes de compensación implementadas en el algoritmo del controlador PID aplicadas al equipo Lego Mindstorms.

Mediante la simulación basada en el modelo matemático para el control LQG y basado en función de transferencia para el control PID, se realiza su implementación en el equipo obteniendo la estabilidad del sistema para cada uno de los controladores desarrollados.

Al realizar las pruebas de funcionamiento de los controladores se pudo observar que el control LQG cumple de mejor manera con índices de rendimiento de robustez, mientras que el control PID tiene mejores índices en los tiempos de estabilización, como se puede corroborar en los cuadros presentados en el capítulo 4 de las pruebas y resultados.

RECOMENDACIONES

Se recomienda la implementación de un sensor acelerómetro en el equipo Lego Mindstorms que permita realizar el control del ángulo Yaw de rotación, para implementar un grado de libertad al sistema, que permita mejorar la estabilidad.

Se recomienda probar la estabilidad del sistema mediante el diseño y aplicación de otros controladores tales como: lineales, no lineales, avanzados y robustos.

Se recomienda aplicar los controladores a versiones recientes del equipo Lego Mindstorms, para mejorar los tiempos de adquisición de datos y tiempo de procesamiento que permitan mejorar el desempeño de los controladores.

Como trabajo futuro se podría recomendar la implementación de este sistema para que sea controlado desde la nube, es decir desde cualquier otra parte de tal manera que sería una aplicación WNCS (Wireless Network Control System)

Se recomienda que se obtenga el modelamiento matemático más exacto posible, debido a que se necesita tener una idea generalizada del sistema para la simulación y para la implementación en el equipo real.

CITAS BIBLIOGRÁFICAS

- Acedo Sánchez, J. (2006). *INSTRUMENTACIÓN Y CONTROL AVANZADO DE PROCESOS*. España: Ediciones Díaz de Santos.
- Castillo, E. d. (2008). *CONTROL DE PROCESOS*. Cataluña: urv.
- DISEÑO DE UN CONTROLADOR PID PARA EL SISTEMA PITCH CONTROLLER . (2012). En I. D. AUTOMÁTICA, *AUTÓMATAS Y SISTEMAS DE CONTROL* (pág. 3). Universidad Miguel Hernandez.
- Dominguez, S., Campoy, P., Sebastian , J., & Jiménez, A. (2006). Modelo de estado. En *Control en el espacio de estado* (pág. 4). Madrid : PEARSON.
- Edubrick. (2010). *Guía de Robótica LEGO MINDSTORMS NXT*.
- Herrera Garzón, M. A. (2014). *Modelado Discreto y Control Óptimo de Sistemas No Lineales Multivariables y su Aplicación a un Péndulo Invertido utilizando Lego Mindstorms*. Madrid: Universidad Politécnica De Madrid.
- Herrera, Camacho, Chamorro, & Gómez. (2015). *Two-Wheeled Inverted Pendulum Robot NXT Lego Mindstorms*. Revista Politécnica.
- HiTechnic. (2012). *Hitechnic HT*. Obtenido de <http://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NGY1044>
- Lego Mindstorms*. (2017). Obtenido de <https://www.lego.com/es-es/mindstorms/support>
- Lewis, F., Xie, L., & Popa , D. (2008). *Optimal and Robust Estimation*. Boca Raton: Taylor & Francis Group.
- Loya, H., Arroyo, S., Rodriguez , R., & Jaramillo, D. (Septiembre de 2015). *Estabilización de un Péndulo Invertido aplicando MPC y LQR*. Quito. Obtenido de ResearchGate.
- Messner, B., & Tilbury, D. (2011). *CONTROL TUTORIALS*. Obtenido de <http://ctms.engin.umich.edu>
- Mina, S. C. (13 de Abril de 2015). *Lego Mindstorms Education*. Obtenido de SlideShare: <https://es.slideshare.net/santimina21/trabajo-1-46960416>

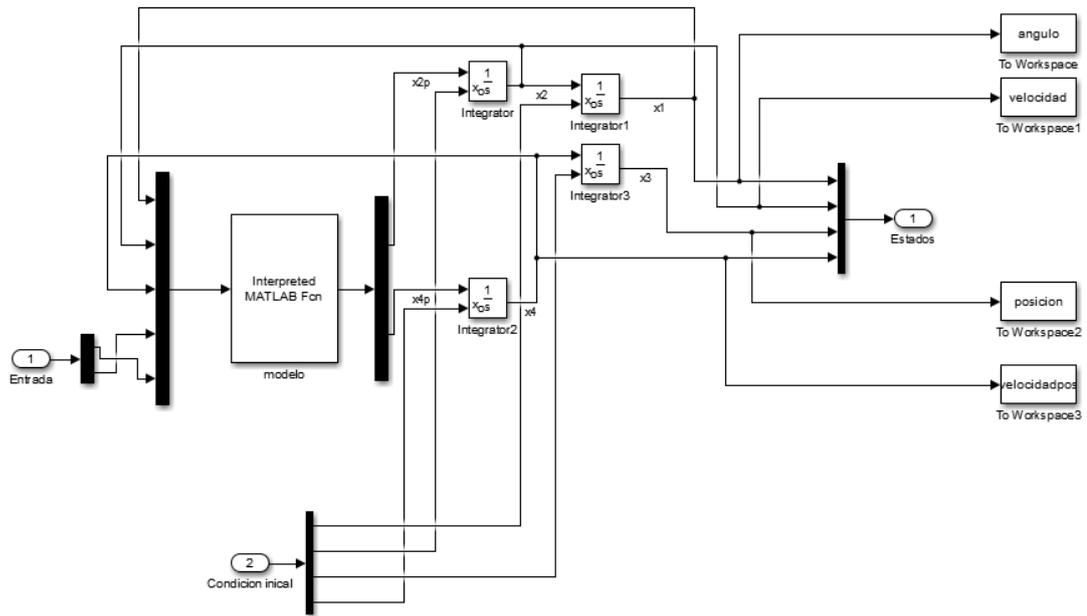
- Morillo, F. (11 de enero de 2007). Obtenido de <http://www.dia.uned.es/~fmorilla/MaterialDidactico/El%20controlador%20PID.pdf>
- Ogata, K. (2003). *Ingeniería de Control Moderna*. Madrid: PEARSON EDUCACIÓN.
- Pillajo, C., Bonilla, P., & Hincapié, R. (28 de Septiembre de 2016). *Algoritmo genético para sintonización de pid basado en la integral del error absoluto*. Obtenido de <http://dSPACE.ucuenca.edu.ec/jspui/bitstream/123456789/29773/1/5.%201615-4908-1-PB.pdf>
- Rairán Antolines, J. D. (2007). *Análisis de sistemas dinámicos y control PID*. Caldas: Fondo de publicaciones Universidad Distrital José de Caldas.
- Ramirez, A. P. (31 de enero de 2015). *INGENIERÍA DE CONTROL II*. Callao.
- Rowell, D. (Octubre de 2002). *MIT*. Obtenido de <http://web.mit.edu/2.14/www/Handouts/StateSpace.pdf>
- Santos, S. C. (2010). *ebah*. Obtenido de <http://www.ebah.com.br/content/ABAAABm8cAB/robotica?part=2#>
- Universitat Politècnica de Catalunya*. (s.f.). Obtenido de BarcelonaTech: <https://upcommons.upc.edu/bitstream/handle/2117/93642/06Sam06de15.pdf>
- Villacres, J., & Viscaino, M. (2016). *DISEÑO E IMPLEMENTACION DE TRES ESQUEMAS DE CONTROL: PID, LQR Y MODOS DESLIZANTES PARA LA ESTABILIZACIÓN DE UN PENDULO INVERTIDO SOBRE DOS RUEDAS DE UN EQUIPO LEGO MINDSTORMS CON LA APLICACIÓN DE UN PLANIFICADOR DE RUTAS MEDIANTE UN ALGORITMO RRT*. Quito: Escuela Politécnica Nacional.
- Yamamoto, Y. (2009). *NXTway-GS Model-Based Design-Control of self-balancing two-wheeled robot built with LEGO Mindstorms NXT*. Obtenido de Cybernet System Co.: http://www.pages.drexel.edu/~dml46/Tutorials/BalancingBot/files/NXTway-GS%20Model-Based_Design.pdf

Zaldívar Navarro, D., Cuevas Jiménez, E., & Pérez Cisneros , M. (2015). *Proyectos con Robots LEGO*. Bogotá: Ra-ma.

ANEXO I

Simulación del sistema no-lineal

Modelo Dinámico Péndulo Invertido no-lineal



```

for k1=1:nmuestras;
    Xin(k1,1)=(rand*pi/40)-(pi/80);    % Datos del angulo de inclinación(ψ)
    Xin(k1,2)=rand*2-1;                % Datos de la velocidad del cambio inclinación
    Xin(k1,3)=(rand*pi/40)-(pi/80);    % Datos del angulo de desplazamiento(φ)
    Xin(k1,4)=rand*2-1;                % Datos de la velocidad de desplazamiento
    Uin(k1,1)=(rand*20)-10;            % Datos Voltaje rueda derecha -9 a 9 voltios
    Uin(k1,2)=(rand*20)-10;            % Datos Voltaje rueda izquierda -9 a 9 voltios
end
    
```

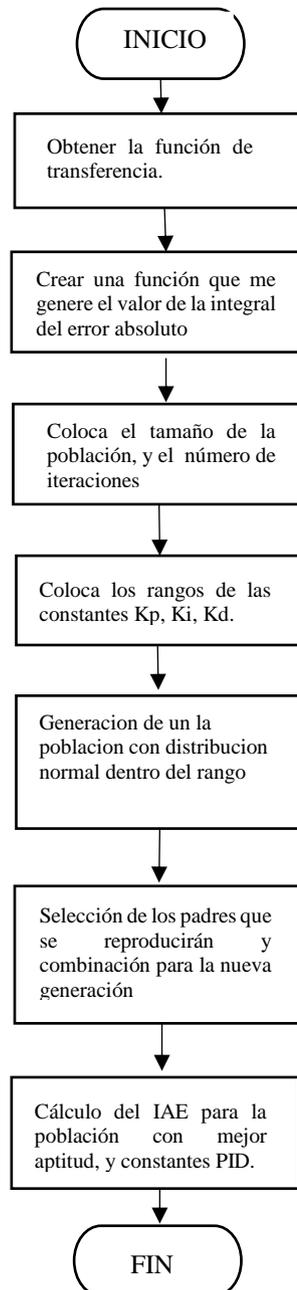
```

%% Carga Datos
load Xin.mat;           %Estados iniciales aleatorios del sistema
load Uin.mat;           %Entradas aleatorias del sistema
nmuestras=length(Xin); %Número de datos
XT=zeros(nmuestras,4);
x1=[pi/180 0 0 0]; % estado inicial del robot Lego Mindstorms

%% simulación del modelo de la planta
for k=1:nmuestras
    U=Uin(k,:);         % Entradas
    x0=Xin(k,:);        % Valores iniciales de los estado
    sim('Modelo.mdl')
    XT(k,:)=Xk1(length(Xk1),2:5); % Xk1 matriz generada por simulink
end
    
```

ANEXO II

Descripción del Algoritmo Genético



ANEXO III

Programación del control PID en RobotC

```
/****** Declaro entradas y salidas******/
#pragma config(Sensor, S1, gyroSensor, sensorI2CHiTechnicGyro)
#pragma config(Motor, motorB, rightMotor, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, leftMotor, tmotorNXT, PIDControl, encoder)

/******Defino variables para las graficas******/

#define DATALOG_SERIES_0 0 // Define para la graficas
#define DATALOG_SERIES_1 1 // Define para las graficas
#define DATALOG_SERIES_2 2 //Define para las graficas
#define DATALOG_SERIES_3 3 //Define para las graficas
#define MAX_ENCODER_VALUES ( 7 ) // Maximo numero de entradas a la matriz de los encoders, se usa para la velocidad
#define POWER_LIMIT ( 100 )//limita la potencia para los motores (<100)
#define SAMPLE_TIME_DIFFERENCE (1.5)

int sampleTime = 10; // Tiempo de muestreo en ms
// Tiempo de muestreo, convertido a segundos y corregido para retrasos en el ciclo
float dt = ( 1.0 * sampleTime + SAMPLE_TIME_DIFFERENCE ) / 1000.0;
// Diametro de las llantas en mm
short wheelDiameterInMm = 56.0;
// Convertir el diametro de las ruedas a radio en metros
float wheelRadius = wheelDiameterInMm / 2000.0;
int encoderValueIndex = 0; // Valor de indexacion del encoder para obtener la velocidad del motor
// Running encoder value for the last MAX_ENCODER_VALUES epochs
// Used for computing the average speed over MAX_ENCODER_VALUES epochs
long encoderValues[ MAX_ENCODER_VALUES ];
float robotAngle = 0.0; // Angulo (integrado de la velocidad angular) degrees
// Current of gyro rate bias in deg/s
float gyroRateBias; //
// Currently measured gyro rate in deg/s
float gyroRate;

// Constantes PID
// Posicion
float kp = 0.35;
float ki = 12;
float kd = 0.008;
// Angulo
float kpa = 0.20;
float kia = 14.2;
float kda = 0.008;

//Ganancia de la velocidad angular,
float gainAngularVelocity = 1.3; //1.3
// Ganancia para el angulo, para cambiar a otro dt
float gainAngle = 18.7; //18.7
// Ganancia para la velocidad del robot, para cambiar a otro dt
float gainRobotSpeed = 86.3; //86.3
// Gain para la posicion del robot
float gainRobotPosition = 980.0; //880

//
// Velocidad, Posicion y potencia
//
// Velocidad del robot en m/s
float robotSpeed = 0.0;
// Posicion del robot en m
float robotPosition = 0.0;

// Potencia a los motores
int rightPower, leftPower;
```

```

// Error
// Variables y contadores fuera de limite
bool NowOutOfBound = false;
bool PrevOutOfBound = false;
int OutOfBoundCount = 0;

//Obtener la tasa de cambio del giroscopio
float getGyroRate()
{
    return SensorValue(gyroSensor);
}

// Calibracion del giroscopio

float calibrate()
{
    int numberOfReadings = 300;

    playSound( soundBeepBeep ); // Sonido para obtener las muestras para la calibracion
    writeDebugStreamLine( "Resetear Giroscopio" );
    sleep( 4000 );
    playSound( soundBeepBeep );
    sleep( 3000 );
    gyroRateBias = 0.0;

    // Realiza las lecturas para actualizar las lecturas del sensor para la calibracion
    for( int i = 0; i < numberOfReadings; i++ ) {
        gyroRateBias += getGyroRate();
    }
    gyroRateBias /= numberOfReadings;
    writeDebugStreamLine("Initial gyroRateBias (not used): %f", gyroRateBias );
    sleep( 500 );
    gyroRateBias = 0.0;
    sleep(100);
    for( int i = 0; i < numberOfReadings; i++ ) {
        gyroRateBias += getGyroRate();
    }
    gyroRateBias /= numberOfReadings;
    writeDebugStreamLine("Second gyroRateBias (not used): %f", gyroRateBias );
    sleep(500);
    gyroRateBias = 0.0;
    // Realiza la calibracion actual
    for( int i = 0; i < numberOfReadings; i++ ) {
        gyroRateBias += getGyroRate();
    }
    gyroRateBias /= numberOfReadings;
    writeDebugStreamLine("Third gyroRateBias (used): %f", gyroRateBias );
    playSound( soundDownwardTones );
    sleep( 100 );
    return gyroRateBias;
}

* Inicializacion
*/
void initialize()
{
    for ( int i = 0; i < MAX_ENCODER_VALUES; i++ ) encoderValues[ i ] = 0;
    nMotorEncoder[rightMotor]=0;
    nMotorEncoder[leftMotor]=0;
    sleep( 100 );
    gyroRateBias = calibrate();
}

//Posicion de la referencia segun la velocidad actualizada
// Devuelve la posicion de la referencia actualizada
float position( float lastReferencePosition, float requestedSpeed )
{
    // retorna lastReferencePosition + requestedSpeed * dt
    return lastReferencePosition + requestedSpeed * dt *0.007;
}

```

```

//Posicion de la referencia segun la velocidad actualizada
// Devuelve la posicion de la referencia actualizada
float position( float lastReferencePosition, float requestedSpeed )
{
    // retorna lastReferencePosition + requestedSpeed * dt
    return lastReferencePosition + requestedSpeed * dt *0.007;
}

/**
 * Retorna la velocidad del motor en deg/s
 * basado en MAX_ENCODER_VALUES y los encoders de derecha e izquierdda.
 */
float myGetMotorSpeed()
{
    encoderValueIndex++;
    if ( encoderValueIndex == MAX_ENCODER_VALUES ) encoderValueIndex = 0;
    int compare_index = encoderValueIndex + 1;
    if ( compare_index == MAX_ENCODER_VALUES ) compare_index = 0;
    long rightEncoderValue = nMotorEncoder[rightMotor] ;
    long leftEncoderValue = nMotorEncoder[leftMotor] ;
    long avgEncoderValue = ( rightEncoderValue + leftEncoderValue ) / 2;
    encoderValues[ encoderValueIndex ] = avgEncoderValue;
    float tmp = ( encoderValues[ encoderValueIndex ] - encoderValues[ compare_index ] ) / ( dt * MAX_ENCODER_VALUES );
    return tmp;
}

/**
 * Leer encoders y determinar la velocidad del robot y la posicion del robot en m/s y m.
 */
void readEncoders()
{
    robotSpeed = wheelRadius * myGetMotorSpeed() / 57.3; // 57.3 = 180/pi, necesita para convertir en deg->rad
    // Se lee nuevamente los encoders realizando un promedio
    long rightEncoderValue = nMotorEncoder[rightMotor] ;
    long leftEncoderValue = nMotorEncoder[leftMotor] ;
    long avgEncoderValue = ( rightEncoderValue + leftEncoderValue ) / 2;
    robotPosition = wheelRadius * avgEncoderValue / 57.3;
}

* Leer giroscopio y la estimacion del angulo y la velocidad angular
*/
void readGyro()
{
    static float robotAngleBias = 0.0;

    float gyroRateBiasUpdateRatio = 0.2;

    float robotAngleBiasUpdateRatio = 0.0;
    float currentGyroRateMeasurement = getGyroRate();

    gyroRateBias = gyroRateBias * ( 1 - dt * gyroRateBiasUpdateRatio ) +
    currentGyroRateMeasurement * dt * gyroRateBiasUpdateRatio;

    gyroRate = currentGyroRateMeasurement - gyroRateBias;

    robotAngleBias = robotAngleBias * ( 1 - dt * robotAngleBiasUpdateRatio ) -
    ( robotPosition * gainRobotPosition / gainAngle ) * dt * robotAngleBiasUpdateRatio;

    robotAngle += gyroRate * dt - robotAngleBias;
}

```

```

/**
Combina los valores del sensor y la ganancia del sensor; para tener un unico valor
*/
float yposicion(float robotSpeed, float robotPosition){

return
robotPosition* gainRobotPosition +
robotSpeed * gainRobotSpeed;
}

float yangulo(float angularVelocity, float robotAngle){

return
robotAngle* gainAngle +
angularVelocity * gainAngularVelocity;
}

```

```

/**
* Control PID
*/
float PIDpos( float input, float reference )
{
// Parte de integración
static float accumulatedError = 0.0;
// Parte diferencial
static float diffError = 0.0;
static float previousError = 0.0;
float currentError = input - reference;
accumulatedError += currentError * dt;
diffError = ( currentError - previousError ) / dt;
previousError = currentError;
return kp * currentError + accumulatedError * ki + diffError * kd;
}

float PIDang( float inputang, float reference )
{
// Parte de integración
static float accumulatedError = 0.0;
// Differentiation part
static float diffError = 0.0;
static float previousError = 0.0;
float currentError = inputang - reference;
accumulatedError += currentError * dt;
diffError = ( currentError - previousError ) / dt;
previousError = currentError;
return kpa * currentError + accumulatedError * kia + diffError * kda;
}

```

```

/**
 * Detecta errores a la salida
 */
void errors( float pidOutput )
{
    if ( abs( pidOutput ) > 100 ) NowOutOfBound = true;
    if ( NowOutOfBound && PrevOutOfBound ) {
        OutOfBoundCount++;
    } else {
        OutOfBoundCount = 0;
    }
    if ( OutOfBoundCount > 20 ) {
        setMultipleMotors( 0, leftMotor, rightMotor );
        eraseDisplay();
        displayCenteredBigTextLine( 2, "ERROR" );
        sleep( 100 );
    }
}

float getSteer()
{
    return 0.0;
}

// Establece la potencia del motor y corrige la direccion de los motores
// que no estan sincronizados

void setMotorPower( float requestedSteering, float averagePower )
{
    float extra_pwr = 0.0;
    static float old_steering = 0.0;
    static float pwr_c = 0.0;
    static float pwr_b = 0.0;

    extra_pwr = 0.0;
    pwr_c = averagePower - round (extra_pwr);
    pwr_b = averagePower - round (extra_pwr);

    old_steering = requestedSteering;
    rightPower = pwr_b;
    leftPower = pwr_c;
    if ( rightPower > POWER_LIMIT ) rightPower = POWER_LIMIT;
    if ( rightPower < -POWER_LIMIT ) rightPower = -POWER_LIMIT;
    if ( leftPower > POWER_LIMIT ) leftPower = POWER_LIMIT;
    if ( leftPower < -POWER_LIMIT ) leftPower = -POWER_LIMIT;
    motor[motorC]=leftPower;
    motor[motorB]=rightPower;
}

```

```

task main()
{
    int counter = 0; // cuenta las entradas del bucle principal
    // Referencia de la posición en metros
    float referencePosition = 0.0;
    // Velocidad requerida en m/s
    float requestedSpeed = 0.00;
    // Valor del sensor combinado, calculado en función del ángulo, la velocidad angular, la velocidad y la posición
    float salidaPos; // salida de la posición
    float salidaAng; // salida del ángulo
    // referencia del PID y potencia (-100 to 100)
    float pidReference = 0.0;

    // salida de los dos PID
    float pidOutput;
    float pidOutput1;
    float pidOutput2;

    // Cantidad de veces que el ciclo de control no cumplió con la fecha límite.
    int controlLoopTimerOverflow = 0;

    initialize();

    // Lazo de balance
    eraseDisplay();
    displayCenteredTextLine( 2, "Balancing" );
    writeDebugStreamLine("*** Starting main loop");
    resetTimer( T1 );
    resetTimer( T2 );

    // Si el robot tiene un ángulo mayor a 60 las ruedas se detienen

    #ifndef ENABLE_HISTORY
    repeatUntil( ( getTimer( T1, seconds ) > 20 ) || ( abs( robotAngle ) > 60 ) || ( OutOfBoundCount > 20 )
    || ( counter >= HISTORY_SIZE ) ) {
    #else
    repeatUntil( ( abs( robotAngle ) > 60 ) || ( OutOfBoundCount > 20 ) ) {
        setMultipleMotors( 0, leftMotor, rightMotor );
        motor[motorC]=0.0;
        motor[motorB]=0.0;
    #endif

    // Obtener la posición esperada basada en la posición actual y la velocidad solicitada
    referencePosition = position( referencePosition, requestedSpeed );

    // Leer los encoders y calcular la posición y velocidad del robot al mismo tiempo
    readEncoders();

    // Leer el giroscopio y actualizar gyroRateBias y el ángulo del robot
    readGyro();

    // Sensor "fusion"

    salidaPos=yposicion(robotSpeed, robotPosition);
    salidaAng=yangulo(gyroRate, robotAngle);

    // PID control

    pidOutput1 = PIDpos( salidaPos, pidReference );
    pidOutput2 = PIDang( salidaAng, pidReference );
    pidOutput=pidOutput1+pidOutput2;

```

ANEXO IV

Programación del control LQG en RobotC

```

/*****
Define entradas y salidas
*****/

#pragma config(Sensor, S1, GyroSensor, sensorI2CHiTechnicGyro)
#pragma config(Motor, motorB, Right_Motor, tmotorNXT, PIDControl, encoder)
#pragma config(Motor, motorC, Left_Motor, tmotorNXT, PIDControl, encoder)

/*****
Define las graficas
*****/

#define DATALOG_SERIES_0 0
#define DATALOG_SERIES_1 1 // Define para las graficas

// ----- Para Motores -----

#define Left_Motor motorC // se define el motor C
#define Right_Motor motorB // se define el motor B
#define deg2rad PI/180 // transformacion grados a radianes
float motorPos = 0; // Posicion inicial del angulo theta
float motorSpeed=0; // Variable velocidad
float gyroAngle=0;
float valDeltaP3 = 0; // Declaracion de variables
float valDeltaP2 = 0; // Declaracion de variables
float valDeltaP1 = 0; // Declaracion de variables
float valSum = 0;
float valSumPrev=0; // Declaracion de variables
float valDelta; // Declaracion de variables
float gyroRaw;
..

//----- Define las variables para filtro de Kalman-----
float An=1;
float Cn=1;
float wn=0.8;
float vn=0.6;
float Qn=0;
float Rn=0;
float pan=0;
float xan=0;
float uan=0;
float xpn=0;
float ppn=0;
float kn=0;
float xn=0;
float yn=0;
float pn=0;
float yfn=0;

// ----- Para giroscopio -----
#define EMAOFFSET 0.0005 // Offset del sensor calculado prueba y error
float gAngleGlobal = 0; // Declaracion de variables
const tSensors GyroSensor=(tSensors) S1; // Asigna al giroscopio al puerto 1
float gOffset; // Defino variable
float gyroSpeed
float gyroAngle1; // Defino variable
..

```

```

// ----- Para Tiempo de muestreo -----
long   cLoop=0; // Define variable
long   tCalcStart; // Define variable
float  tInterval; // Tiempo inicial de muestreo

#define TIME_FALL_LIMIT 100 // Tiempo limite
#define WAIT_TIME 1 // temporizador
long   tMotorPosOK; // Define variable

// ----- Para el Controlador -----

/*****
Defino Variables Globales
*****/

float u[2]; // Vector de entrada
float X[4]; //Vector de salida
float xRef[]={0,0,0,0}; // Referencia

/////////////////////////////////Valores LQR obtenido de Matlab/////////////////////////////////

float k1[]={7.0710,-14.1534,-0.0158,-0.0422};
float k2[]={7.0710,-14.1534,0.0158,0.0422};

/////////////////////////////////Valores Estimador obtenido de Matlab(Kalman)/////////////////////////////////

float k11[]={1250,1250,-1861.2,1861.2};
float k12[]={-0.1,0.1,4041.8,-1541.38};

/*****
Subrutina obtención Posición y velocidad de los motores
*****/

void GetMotorData(float &motorSpeed, float &motorPos)
{
    long valLeft, valRight;

    // ----- Posición -----
    valLeft = nMotorEncoder[Left_Motor]; // valor del encoder
    valRight = nMotorEncoder[Right_Motor]; // valor del encoder
    motorPos=(deg2rad*(valLeft+valRight)/2)-0.1; // se obtiene la posición obteniendo el promedio de los 2 motores

    // ----- Promedio de la velocidad -----
    valSumPrev = valSum;
    valSum = motorPos;
    valDelta = valSum - valSumPrev;
    motorSpeed = (valDelta + valDeltaP1 + valDeltaP2+valDeltaP3)/(4*tInterval);
    valDeltaP3 = valDeltaP2;
    valDeltaP2 = valDeltaP1;
    valDeltaP1 = valDelta;
}

```

```

/*****
Subrutina Adquisición Giroscopio con el Filtro de Kalman
*****/

void GetGyroData(float &gyroSpeed, float &gyroAngle)
{
    Qn=vn*vn;
    Rn=wn*wn;
    pan=Qn;

    xpn=An*xan;
    ppn=An*pan*An+Qn;
    kn=ppn*Cn*(1/(Cn*ppn*Cn+Rn));
    yn=SensorValue(GyroSensor);
    xn=xpn+kn*(yn-Cn*xpn);
    pn=(1-kn*Cn);
    uan=SensorValue(GyroSensor);
    xan=xn;
    yfn=Cn*xn;

    gyroRaw = yfn;
    gOffset = EMAOFFSET * gyroRaw + (1-EMAOFFSET) * gOffset;
    gyroSpeed = deg2rad*(gyroRaw - gOffset);
    gAngleGlobal += gyroSpeed*tInterval;
    gyroAngle = gAngleGlobal;
    gyroAngle1=gyroAngle;
}

/*****
Calculo del Tiempo de muestreo
*****/
void CalcInterval(long cLoop)
{
    if (cLoop == 0)
    {
        tInterval =0.01;
        tCalcStart = time1[T1];
    }
    else {
        tInterval = (time1[T1] - tCalcStart)/(cLoop*1000.0);
    }
}

/*****
Determinacion Offset Giroscopio
*****/

#define OFFSET_SAMPLES 100 // Numero de tomas
void GetGyroOffset()
{
    float gSum;
    int i, gMin, gMax, g;
    eraseDisplay();
    nxtDisplayStringAt(0,32,"Colocar robot");
    nxtDisplayStringAt(0,24,"en posicion plana");
    nxtDisplayStringAt(0,16,"obtener offset");
    do {
        gSum = 0.0;
        gMin = 1000;
        gMax = -1000;
        for (i=0; i<OFFSET_SAMPLES; i++)
        {
            g = SensorValue(GyroSensor);
            if (g > gMax)
                gMax = g;
            if (g < gMin)
                gMin = g;
            gSum += g;
            wait1Msec(5);
        }
    } while ((gMax - gMin) > 1);
}

```

```

// ----- Promedio de las muestras -----

gOffset = gSum / OFFSET_SAMPLES+ 1.0;
}

/*****
Accion de control Saturada
*****/
void PowerControl()
{
    float powerRight=u[1]*10;
    float powerLeft=u[0]*10;

    if (abs(powerRight)<100)tMotorPosOK=timel[T1];

    if (powerLeft > 100) powerLeft = 100;
    if (powerLeft < -100) powerLeft = -100;
    if (powerRight > 100) powerRight = 100;
    if (powerRight < -100) powerRight = -100;

    motor[motorC]=powerLeft;
    motor[motorB]=powerRight;
}

/*****
Controlador LQR
*****/
float LQR()
{
    float ur=0;
    float ul=0;
    float uer=0;
    float uel=0;
    float err[]={0,0,0,0};
    float erre[]={0,0,0,0};
    float xe[]={0,0,0,0};
    float ul = 0;
    float ult = 0;
    float uel = 0;
    float uelt = 0;
    /*****
    ESTIMADOR LQG
    *****/
    float ue[2];
    float A [4]= {
        {0,0,1,1},
        {0,0,0,1},
        {0,-399,-34457,34457},
        {0,238,14548,-14548},};
    float B[4] = {
        {0,0},
        {0,0},
        {23863, -2968},
        {-2968, -0.2968},};
    float C [6] ={
        {1,0,0,0},
        {0,0,1,0}
    }
}

```

```

/*****
ACTUALIZACION DEL ESTIMADOR
*****/

xe= A*xe+B*ue;

err[0]=xRef[0]-X[0];
err[1]=xRef[1]-X[1];
err[2]=xRef[2]-X[2];
err[3]=xRef[3]-X[3];

/*****
ERROR DEL ESTIMADOR
*****/

erre[0]=xRef[0]-xe[0];
erre[1]=xRef[1]-xe[1];
erre[2]=xRef[2]-xe[2];
erre[3]=xRef[3]-xe[3];

/*****
Controlador LQR
*****/

ul=-k1[0]*err[0]-k1[1]*err[1]-k1[2]*err[2]-k1[3]*err[3]; // Aplicacion del control LQR
ur=-k2[0]*err[0]-k2[1]*err[1]-k2[2]*err[2]-k2[3]*err[3]; // Aplicacion del control

/****GRAFICAR****/
datalogDataGroupStart();
datalogAddValue(DATALOG_SERIES_0, ((SensorValue(GyroSensor)-gOffset))*0.1);
datalogAddValue(DATALOG_SERIES_1,robotPosition);
datalogDataGroupEnd();

/*****
SALIDA DE KALMAN
*****/

uel=-kl1[0]*erre[0]-kl1[1]*erre[1]-kl1[2]*erre[2]-kl1[3]*erre[3]; // Kalman
uer=-kl2[0]*erre[0]-kl2[1]*erre[1]-kl2[2]*erre[2]-kl2[3]*erre[3]; //

/*****
SEÑAL DE CONTROL
*****/

u[0]=ul+uel;
u[1]=ur+uer;

/*****
ACTUALIZACION DE LA SEÑAL DE CONTROL
*****/

ue=u;

return u;

}

```

```

/*****
Programa Principal
*****/

task main()
{
    float gyroAngle;
    // ----- Offset Gyroscopio -----

    GetGyroOffset();
    ClearSounds();
    nVolume = 3
    for (int n=0; n<2; n++)
    {
        ClearSounds();
        wait10Msec(20);
        PlayTone(784, 15);
        wait10Msec(20);
    }
    eraseDisplay();
    nxtDisplayStringAt(3,60,"Balanceando");
    wait10Msec(300);
    ClearTimer(T1); // Para inicializar el tiempo

    // ----- Algoritmo Principal -----

    nMotorEncoder[Left_Motor] = 0;           // Inicializo Posicion Angular de Motores
    nMotorEncoder[Right_Motor] = 0;        // Inicializo Posicion Angular de Motores

    while(true)
    {

        CalcInterval(cLoop++);                // Funcion determina tiempo de muestreo
        GetMotorData(motorSpeed,motorPos);    // Funcion angulo Theta y DotTheta
        GetGyroData(gyroSpeed, gyroAngle);   // Funcion angulo Psi y DotPsi
        // ----- Vector de estados -----
        X[0]=gyroAngle;
        X[1]=gyroSpeed;
        X[2]=motorPos;
        X[3]=motorSpeed;
        // ----- CONTROLADOR LQR LINEAL -----

        LQR();

        // ----- CONTROLADOR LQR -----
        PowerControl();

        // ----- Para detener al Pendulo -----
        if((time1[T1]-tMotorPosOK)>TIME_FALL_LIMIT)
        {
            break;
        }

        wait1Msec(WAIT_TIME);
    }
}

```