Modelado y simulación de un robot Autobalanceado mediante Coppelia Sim

Belkis Marisol Damián García
Autónoma de GuerreroFacultad de ingeniería Chilpancingo, Gro., México
TEL: 7471183681
C.P. 39087
merry_pumas@hotmail.com

Eric Rodríguez Peralta
Autónoma de Guerrero
Facultad de ingeniería
Chilpancingo, Gro., México
TEL: 7471173636
C.P 39087
09063@uagro.mx

Wilfrido Campos Francisco
Autónoma de Guerrero
Facultad de ingeniería
Chilpancingo, Gro., México
Tel. 7471333310
C.P 39087
11849@uagro.mx

León Julio Cortéz Organista
Autónoma de Guerrero
Facultad de ingeniería
Chilpancingo, Gro., México
Tel. 7471102752
C.P 39087
11991@uagro.mx

RESUMEN

Un robot autobalanceado es un dispositivo que, aún teniendo su centro de masas por encima del eje de giro, consigue mantener el equilibrio. En este documento se presenta el diseño estructural y programación de un robot autobalanceado basado en el simulador CoppeliaSim, en una arquitectura cliente servidor, usando Python como lenguaje de programación. El proyecto se centra principalmente en el control de estabilidad del robot por lo que, se describe el procedimiento de diseño del controlador que busca obtener los valores de las constantes óptimas de los parámetros proporcional, integral y derivativo que permitan mantener el robot en el punto de equilibrio no estable. Se utilizó la sintonía manual para estimar los valores de los parámetros.

Palabras Clave: Coppelia, Robot autobalanceado, Simulación

INTRODUCCIÓN

En la actualidad la robotica esta asumiendo un papel importante en nuestro entorno social, especialmente en diferentes áreas de la industria. A medida que crece la complejidad de las aplicaciones en la robótica, resulta de vital importancia realizar una fase de simulación antes de la construcción y puesta en marcha del robot físico en el mundo real, de esta manera se pueden simular pruebas ahorrando costos y tiempos. Los nuevos avances en el campo de la robotica permiten el diseño y construcción de prototipos de robots virtuales que se ha convertido en un verdadero reto, pues se requiere un riguroso criterio para su simulación, de tal manera que ésta cubra todos los aspectos de adquisición de señales y movimientos en diferentes escenarios, que permita llevar a cabo un analisis cuantitativo con los resultados medibles del desempeño del robot [6].

En este trabajo de investigación, se pretende mostrar el modelado y simulación de un robot autobalanceado utilizando un controlador Proporcional Integral Derivativo (PID), programando los sensores y actuadores que componen al robot, de acuerdo a las especificaciones reales del robot que se pretende construir de manera física para esto, se hará uso de las herramientas que ofrece el simulador Coppeliasim y de la interfaz de programación de aplicaciones (API) basada en lenguaje de programación Python. El alcance de este trabajo consiste en modelar el robot en 3D e implementar un algoritmo de control que sea capaz de mantenerlo en equilibrio.

En el mercado, existen diferentes aplicaciones de simulación de robótica móvil basadas en el comportamiento de los robots, éstos permiten a los usuarios crear mundos virtuales simples de objetos rígidos y fuentes de luz que permiten crear un ambiente de trabajo para que los robots interactúen con ellos además, que este tipo de simuladores permiten "aprender" al robot de los errores, demostrando así un comportamiento muy cercano a la realidad ya que algunos simuladores utilizan un motor de física para la generación de movimiento más realista del robot. Las acciones del robot basadas en sensores son mucho más difíciles de simular ya que el movimiento del robot depende de las lecturas instantáneas del sensor en el mundo real.

De acuerdo con [7], existen diferentes entornos de simulación de robots y éstos proporcionan las siguientes características:

- Prototipado rápido del robot utilizando el propio simulador como herramienta de creación.
- Motores de física para movimientos realistas. La mayoría de ellos utiliza ODE[1]
- Representación realista en 3D. Se pueden utilizar herramientas de modelado 3D estándar o herramientas de terceros para construir los entornos.

De acuerdo con [7], existen diferentes entornos de simulación que pueden utilizarse para el modelado y simulación del robot autobalanceado que se quiere crear como: Gazebo, Coppeliasim, Webots, ARGoS y USARSim, consideradas como plataformas autónomas e independientes. De entre éstas, las posicionadas como mejores simuladores de robótica son Webots, Gazebo y Coppeliasim [3]. La tabla 1, muestra algunas de las características de estos simuladores: De entre estos simuladores, se eligió Coppeliasim de la empresa Coppelia Robotics, ya que, si bien es un software propietario, cuenta con una versión de educación completamente libre que permite diferentes modos de simulación. El entorno gráfico, permite diseñar un robot propio con los elementos que el mismo software proporciona o bien importar tu propio diseño desde cualquier entorno gráfico en un formato estándar, por lo que ofrece libertad de diseño.[2].

[1] Open Dynamics Engine

Tabla 1: Tabla comparativa de los simuladores de robótica móvil más utilizadas en el mercado.

Software	Empresa	Licencia	Lemguaje	Plataforma
Webots	Cyberbotics	Propietaria	C, C++, Python, Java, Matlab, ROS	Windows, Linux y MacOS
Copeliasim	Coppelia	Propietaria/GNU	C, C++, Python, Java, LUA, Matlab, Octave	Windows, Linux y MacOs
Gazebo	OSRF	Libre	C, C++, Python, Java, Matlab	Linux y MacOS

1. ENTORNO DE SIMULACIÓN

La simulacion de robots nos ayuda a la planificación de la predicción con el cálculo de tiempos de ciclo, con ellos se pueden anticipar problemas de diseño, plantear mejoras en la morfología del robot, mejoras en el algoritmo lo que permite evitar el desembolso para adquirir el robot real o en su caso la construcción de uno en particular [5].

La plataforma utilizada para la realización de este trabajo ha sido el simulador Coppeliasim que es una plataforma de experimentación virtual de robótica en su versión de licencia de estudiante por loque es de libre uso.

Esta plataforma se basa en una arquitectura de control distribuido, esto significa que cada objeto o modelo construido, puede controlarse individualmente mediante un script integrado, un plugin o bien haciendo uso de un cliente API[2] remoto. Esto hace que CoppeliaSim sea muy versátil e ideal para aplicaciones de múltiples robots. Otra de las grandes ventajas que ofrece esta plataforma, es la diversidad de lenguajes de programación que soportan por lo que los controladores del robot se pueden escribir en: C/C++, Python,Java, Lua, Matlab u Octave [4] [5]. Además, el simulador es multiplataforma y portable para diferentes sistemas operativos como Windows, Linux y MAC OS. Para fines prácticos de este trabajo de investigación, se trabajó en plataforma Windows.

Otras características importantes de la plataforma, es que se pueden elegir entre tres motores físicos diferentes para mayor velocidad en los cálculos dinámicos, permite simulaciones físicas e interacciones con objetos dentro de su ambiente de trabajo, por lo que permite la detección de colisiones, el cálculo de distanciasmínimas, planeación de trayectorias, mecanísmos importantes para la simulación como la cinemática directa e inversa. Es posibletambién, la creación y edición de modelos y objetos por lo que resulta sencillo diseñar y construir un robot personalizado que con sensores de proximidad y de visión además, se puede personalizar la interfaz de usuario.

Un elemento de la interfaz muy destacable es el buscador de modelos que vienen por defecto en Coppeliasim. Éste se encuentra en una ventana en la parte izquierda y están ordenados por carpetas muchos de los modelos de robots móviles, manipuladores, de los principales fabricantes de robots a nivel internacional, además de otros modelos como componentes, equipamiento, decoración, etc. El punto (4) de la Figura 1, muestra la ubicación de los diferentestipos de modelos que se pueden utilizar en el simulador. Estos modelos se pueden también modificarse o bien es posible diseñar yconstruir un modelo propio.

Para conocer con detalle cada elemento de la interfaz de usuario, Coppeliasim dispone de un manual de usuario, que también

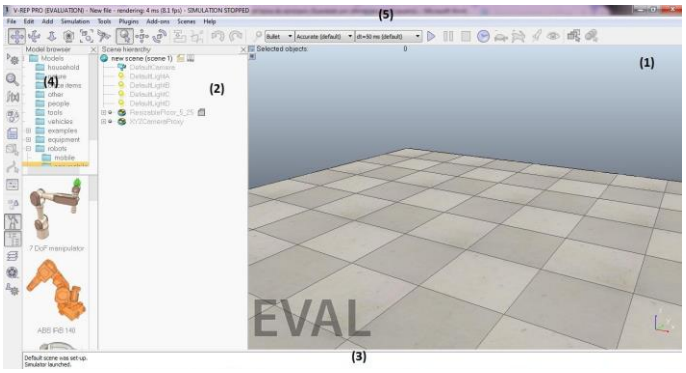

Figura 1: Interfaz de Coppeliasim.

contiene las formas de programar, el cálculo de módulos, el funcionamiento de la simulación, tutoriales, etc. [5]. Los principales elementos que se usan en Coppeliasim para crear una escena de simulación son los objetos, éstos son visibles en la jerarquía de escenas mediante un símbolo que distingue cada tipo; y en la escena de la vista principal se representan en tres dimensiones Ver Figura 1 puntos 1 y 2. Tres de los elementos centrales del simulador son:

- Objetos de escena, cálculo
- de modelos y control de
- mecanismos

1.1. Objetos de escena

Existen diferentes tipos de objetos y cada uno tiene sus opciones de configuración como el de poder asignarles propiedades especiales que les permiten interactuar con otros objetos mediante el cálculo de módulos (detección de colisiones, distancia mínima entre dosobjetos, cinemática inversa, planeación de trayectorias, entre otras). Estas propiedades se pueden combinar entre sí y formar sistemas complejos junto con otros módulos de cálculo y mecanismos de control [5]. La Figura 2, muestra los diferentes tipos de objetos de escena que pueden utilizarse.

La siguiente lista ofrece una breve descripción funcional de cadatipo de objeto en el simulador:

Formas (Shapes) : una forma es una malla rígida que se compone de caras triangulares.

Articulaciones (Joint) : un objeto de articulaciónes una articulación o actuador. Se soportan cuatro tipos: juntas angulares, juntas prismáticas, tornillos y juntas esféricas.

Sensores de proximidad (Proximity sensor) : un sensor de proximidad detecta objetos de forma geométricamente exacta dentro de su volumen de detección. Coppeliasim admite

[2] Interfaz de programación de aplicaciones (Application Programming Interface)

sensores de proximidad de tipo pirámide, cilindro, disco, cono y rayo.

Gráficos (Graph) : se utiliza un gráfico para registrar y visualizar datos de simulación.

Maniquíes (Dummy) : un maniquí es un punto con orientación. Los maniquíes son objetos polivalentes que pueden tener muchas aplicaciones diferentes.

Sensores de fuerza (Force sensors) : un sensor de fuerza es un objeto capaz de medir las fuerzas y pares que se le aplican. También tiene la capacidad de romperse si se sobrepasa un umbral determinado.

Cámaras (Camera) : una cámara es un objeto que permite ver la escena de simulación desde varios puntos de vista.

Luces (Ligth) : una luz es un objeto que permite iluminar la escena de simulación.

Caminos (Paths) : un camino es un objeto que define un camino o trayectoria en el espacio. Se puede utilizar para diversos fines, también como junta o actuador personalizado.

Árboles OC (OC trees) : un árbol OC es una estructura de datos de partición espacial formada por vóxeles.

Nubes de puntos(Point Cloud) : una nube de puntos es una estructura de árbol OC que contiene puntos.

1.2. Calculo de Modelos

Algunos de los objetos anteriores pueden tener propiedades especiales que permitan que otros objetos o módulos de cálculo interactúen con ellos. De acuerdo con [1] los objetos pueden ser:

Colisionable : los objetos colisionables se pueden probar para detectar colisiones contra otros objetos colisionables.

Medible : los objetos medibles pueden tener la distancia mínima entre ellos y otros objetos medibles calculada.

Detectable : los objetos detectables pueden ser detectados por sensores de proximidad.

Renderable : los objetos renderizables pueden ser vistos o detectados por sensores de visión.

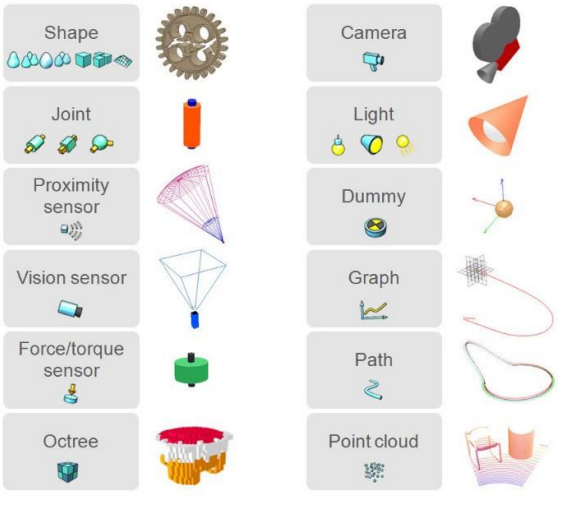

Figura 2: Tipos de objetos y su representación 3D.

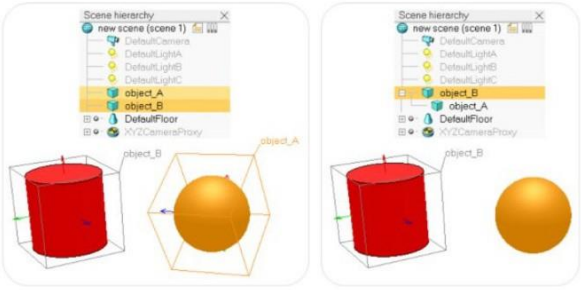

Figura 3: Interacción de modelos individual y jerárquica.

Visible : los objetos visibles se pueden ver a través, o su contenido de imagen se puede visualizar en vistas.

Cada objeto tiene una posición y orientación dentro de la escena de simulación. La configuración del objeto se refiere a su posición y orientación es decir, los objetos pueden adjuntarse a otros objetos o construir uno encima de otro por ejemplo, si un objeto A se construye sobre un objeto B, entonces el objeto B es el padre y el objeto A es el hijo a esto se le conoce como interacción jerárquica. La Figura 3, muestra ejemplo de la configuración entre dos modelos que pueden interactuar entre sí, ya sea de manera idividual o jerárquica.

1.3. Control de mecanismos

CoppeliaSim es un software que lleva un entorno de desarrollo integrado, basado en una arquitectura de control distribuida. Esto presenta una gran ventaja ya que cada objeto o modelo se puede controlar individualmente de seis maneras de programación distintas: scripts, embebidos, plugins, API remota de cliente, complementos (add-ons), nodo ROS y cliente/servidor personalizado. Además cualquiera de estas formas que se elijan, es compatible con los lenguajes de programación más extendidos actualmente, como por ejemplo C/C++, Python, Java, Matlab, etc, ver figura 4.

1.4. Interfaz de CoppeliaSim

La figura 1 muestra la interfaz gráfica del entorno de simulación de CoppeliaSim, en ésta se señala, con una numeración, algunas de las herramientas con las que cuenta y resulta importante mencionar su descripción, ya que a lo largo del artículo se hace uso de esta terminología:

1 Escena : En esta zona es donde se va a realizar el modelado del robot a simular y donde se va a producir el transcurso de la simulación.

2 Jerarquía de la escena : Este lugar muestra el contenido de una escena, es decir, los objetos que la componen. Los objetos pueden disponerse en forma de árbol jerárquico para construir un robot, permitiendo que los objetos que lo componen no se suelten o simplemente para agrupar objetos.

3 Barra de estado : Muestra información de operaciones realizadas, comandos y mensajes de error del intérprete del lenguaje Lua. También puede mostrar mensajes procedentes

de scripts desarrollados mediante la función de la API nativa de Copeliasim.

4 **Explorador de modelos** : Aquí es donde se encuentran modelos ya creados que ofrece Copeliasim para su uso, ya sean robots, sensores o simples modelos (sillas, mesas, edificios, etc.). Muestra en la parte superior una estructura de carpetas y en su parte inferior las miniaturas de los modelos que se encuentran en la carpeta seleccionada; mediante una operación de arrastrar y soltar ("drag and drop") pueden irse cargando los modelos en la escena.

5 **Barras de herramientas** : En este lugar se encuentran las funcionalidades que normalmente se usan en el simulador (deshacer, rehacer, iniciar una simulación, manipulación de objetos, etc..

2. MODELADO DEL ROBOT

Para empezar a construir el entorno de simulación, se debe diseñar el modelo del robot móvil, para fines prácticos de este trabajo de investigación, se diseñará un robot autobalanceado de un GDL (Grado de libertad), que cuenta con dos ruedas alineadas a un mismo eje que sostiene una estructura (cuerpo del robot) y que puede mantenerse equilibrado sin ningún tipo de fuerza externa, ver figura 5. Un robot con estas características es inherentemente inestable y debe ser monitoreado constantemente para permanecer en una posición de estabilidad, se basa en el principio del péndulo invertido por lo que resulta importante revisar su modelo matemático.

2.1. Péndulo invertido

El péndulo invertido es uno de los problemas clásicos en la dinámica y se hacen numerosas referencias a él, en libros que tratan sobre teoría de control dando numerosas estrategias para abordarlo.El problema físico del equilibrio de un péndulo invertido, consiste en un péndulo cuyo centro de masas se encuentra situado por encima del eje de balanceo, ver figura 6 esto crea al sistema una inestabilidad estática que puede ser compensada si se aplica un momento en el eje de giro o bien mediante un movimiento en el plano horizontal [8].

Para conseguir equilibrarlo, se estimará el ángulo formado por la vertical y se aplicará una fuerza a los motores, se utilizará además un controlador PID para que el ángulo formado por la vertical se mantenga a 0°. Si se considera un punto de balanceo fijo y un sistema de movimiento en 2 dimensiones, se obtendrá una función del comportamiento del sistema, una función en la que la aceleración ($\ddot{\Theta}$) es dependiente de la gravedad (g), de la longitud del péndulo (l) y del seno del ángulo formado (θ). La ecuación 1 muestra la dinámica del péndulo invertido.

$$\ddot{\theta} = \frac{g}{l}\sin\theta \qquad (1)$$

2.2. Diseño del robot en CoppeliaSim

Para la construcción virtual del robot en el simulador, se deben crear los objetos en la escena de simulación, para esto se utilizaron

Figura 5: Robot autobalanceado de 1 GDL.

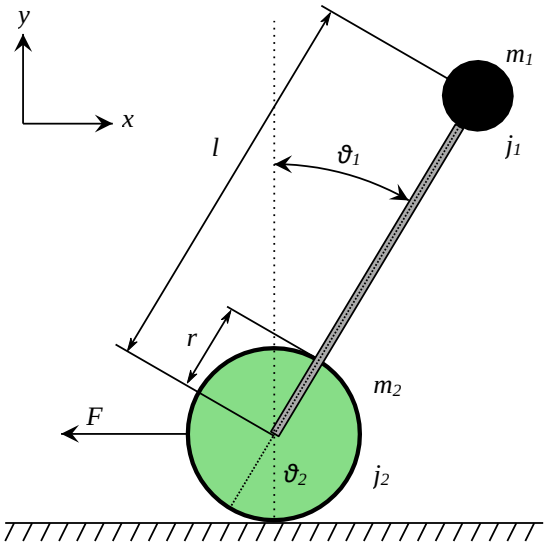

Figura 6: Ecuación dinámica del péndulo invertido.

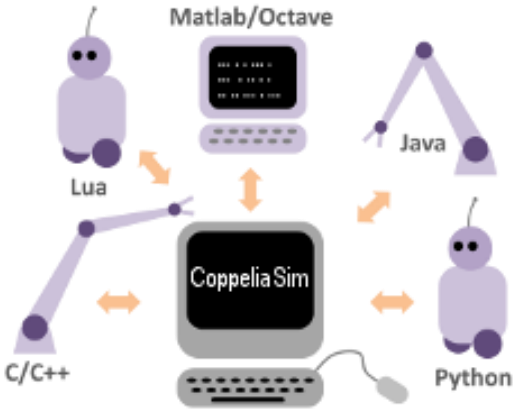

Figura 4: Lenguajes soportados.

las formas primitivas que ofrece el software conocidas como *Primitive shape*, concretamente se utilizaron dos cilíndros para simular las ruedas del robot, un cuboide para el cuerpo del robot así como dos articulaciones que van unidas a las ruedas y que realizan la función de motores para el control del equilibrio.

2.2.1. Diseño de las ruedas del robot. Para colocar objetos en la escena de trabajo, Coppeliasim ofrece múltiples posibilidades, incluso se pueden diseñar o importar, en este caso se diseñaron las ruedas del robot haciendo uso de las formas primitivas para lograrlo, simplemente se da click con el botón derecho en cualquier parte de la escena y se elige del menú desplegable la forma primitiva a utilizar (*Add- PrimitiveShape-Cylinder*). La tabla 2, muestra los valores de la geometría, posición y orientación de las ruedas del robot.

La figura 7, muestra la ventana con las propiedades dinámicas del robot, en este caso, las ruedas se han configurado como objetos que producirá una respuesta a una colisión con otros objetos en la escena que también están configurados de la misma manera, también es posible modificar la forma dinámica en que las ruedas responderán a las colisiones. Además, las ruedas se han configurado como un cuerpo dinámico por lo que su forma, posición y orientación son modificables. Debido a que se trata de un cuerpo dinámico, es posible calcular automáticamente las propiedades de masa e inercia del objeto, basándose en una densidad uniforme del material, para el caso de la masa, la forma primitiva seleccionada puede aumentar o disminuir fácilmente su masas en un factor 2 con los botones M = M*2 y M = $\frac{M}{25}$, esto es conveniente para encontrar rápidamente parámetros de simulación estables mediante prueba y error. De igual manera ocurre con los momentos inerciales sin la masa del objeto diseñado. La posición y orientación del COM (Centro de masa) de las ruedas, se determina de manera automática sin embargo, es posible modificar estos valores mediante una matríz simétrica relativos a la posición del centro de masa, en este caso el centro de masa se localiza en el centro de las ruedas. La figura 8, muestra la geometría y la posición y orientación iniciales de las ruedas del robot.

2.2.2. Diseño del cuerpo del robot. Con la idea de dar simplicidad al robot, se utilizó un cuboide como forma primitiva para el cuerpo del robot, la tabla 3, muestra la forma, posición y orientación de éste. Las condiciones de propiedades dinámicas son similares a las ruedas con la diferencia de que el centro de masa se encuentra en la base del cuerpo con la finalidad de simular el modelo del péndulo invertido y sea afectado por la gravedad. La figura 9, muestra la geometría, posición y orientación del cuerpo del robot.

2.2.3. Motores del robot. Para simular los motores que moverán las ruedas del robot se utilizan articulaciones conocidas como Joints. Una articulación es un objeto que tiene al menos un grado intrínseco de libertad (DoF). Las articulaciones se utilizan para construir mecanismos y mover objetos como es el caso de las ruedas. La posición, orientación y propiedades visuales de éstos se muestran en la tabla 4. Dado que los motores deben estar unidos a las ruedas y todo esto unido al cuerpo del robot, se debe establecer una jerarquía entre ellos, la figura 10 muestra la relación jerarquica y el diseño final del robot que se utilizará para las pruebas de equilibrio.

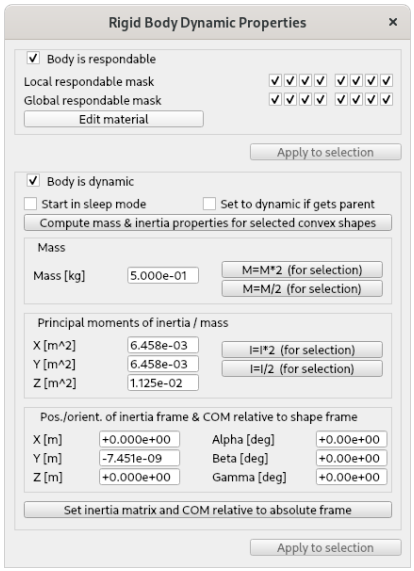

Figura 7: Propiedades dinámicas de las ruedas.

2.3. Programación del robot

El primer paso para poder manipular el robot consiste en configurar todos los elementos estructurales, este paso se realizó en el punto anterior y consiste en conocer detalladamente la configuración del robot es decir, analizar como están distribuidos los elementos que lo componen, para este caso: ruedas, cuerpo y mecanismos de movimiento (motores). En relación a las ruedas, la configuración que tiene el robot es diferencial.

La configuración diferencial es la más sencilla de todas las utilizadas típicamente en la robótica móvil y consta de dos ruedas diametralmente opuestas en un eje que es perpendicular a la dirección del robot. Cada rueda es controlada por un motor de forma tal que los giros se realizarán asignando diferentes velocidades a cada una de ellas. Con dos ruedas es imposible mantener al robot en equilibrio debido al mismo peso del cuerpo del robot además de que, se producen cabeceos al moverse el robot.

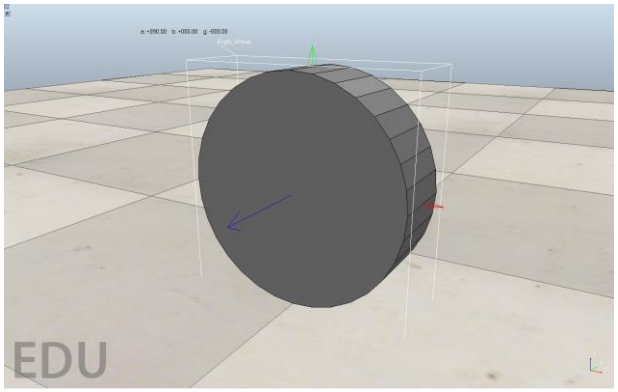

Figura 8: Geometría, forma y posición de las ruedas del robot.

Tabla 2: Propiedades de las ruedas del robot.

Geometría	Posición	Orientación	Peso
X-size[m]: +3.0000e − 01	X-coord[m]: +0.0000e + 00	Alpha[deg]: +9.0000e + 01	Mass[Kg]: +5.0000e-01
Y-size[m]: +3.0000e − 01	Y-coord[m]: −1.8000e − 01	Beta[deg]: +0.0000e + 00	
Z-size[m]: +1.0000e − 01	Z-coord[m]: +1.5000e − 01	Gamma[deg]: +0.0000e + 00	

Tabla 3: Propiedades del cuerpo del robot.

Geometría	Posición	Orientación	Peso
X-size[m]: +1.0000e − 01	X-coord[m]: +0.0000e + 00	Alpha[deg]: +0.0000e + 00	Mass[Kg]: +1.0000e+01
Y-size[m]: +2.0000e − 01	Y-coord[m]: +0.0000e + 00	Beta[deg]: +4.0930e − 12	
Z-size[m]: +5.0000e − 01	Z-coord[m]: +4.0000e − 01	Gamma[deg]: +0.0000e + 00	

Tabla 4: Posición y orientación y propiedades visuales de las articulaciones.

Posición	Orientación	Propiedades visuales
X-coord[m]: +0.0000e + 00	Alpha[deg]: +9.0000e + 01	Length[m]: +1.5000e - 01
Y-coord[m]: −1.8000e - 01	Beta[deg]: +0.0000e + 00	Diameter[m]: +2.0000e - 02
Z-coord[m]: +1.5000e - 01	Gamma[deg]: +0.0000e + 00	Max Step size [deg]: +1.0000e+01

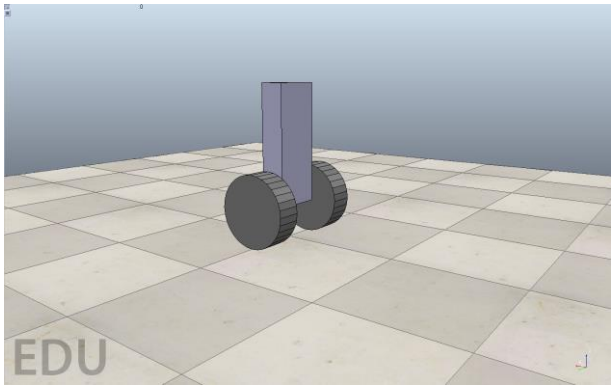

Figura 9: Geometría, forma y posición del cuerpo del robot.

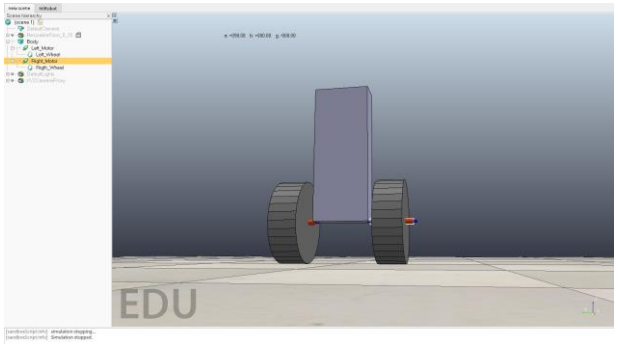

Figura 10: Relación jerárquica entre los objetos que formanel robot.

Mantener en equilibrio al robot es precisamente el reto de este trabajo de investigación. El robot diseñado consta de dos articulaciones, una por cada rueda del robot. Si cada articulación provee al robot de un grado de libertad, se puede confirmar entonces que este robot cuenta con dos grados de libertad.

De las seis diferentes maneras en que se puede programar el código de control del robot, se ha elegido la opción de la API remota debido a su sencillez y flexibilidad. Este tipo de control, permite la comunicación entre CoppeliaSim y una aplicación externa es decir, una aplicación que se ejecuta en un proceso diferente o en una computadora diferente. Este tipo de control consta de dos partes:

1. Cliente: Es el lado del cliente de la aplicación que contine aproximadamente cien funciones específicas de control y una general que pueden ser accesadas por diferentes lenguajes de programación, en este caso se decidió por utilizar Python.
2. Servidor: Es implementado por Copeliasim y un plugin que permite la comunicación con el cliente

La figura 11, muestra el diagrama de comunicación entre el simulador y el programa de control del robot. La función del programa de control consiste en comunicarse con el robot, obtener las posiciones articulares del robot (motores) y mandar las velocidades de cada una de las ruedas de tal forma que lo mantenga en equilibrio.

2.3.1. Configuración del server. Para iniciar el servidor, es necesario abrir la escena donde se construyó el robot, esto generará un script de forma automática que deberá deshabilitarse y generar uno nuevo seleccionando cualquiera de los objetos que componen la estructura del robot, este nuevo script contiene un método al que se debe pasar como parámetro el puerto con el que se conectará el cliente.

2.3.2. Configuración del cliente. Desde un script de python, se importa la librería del simulador para poder acceder a todas las

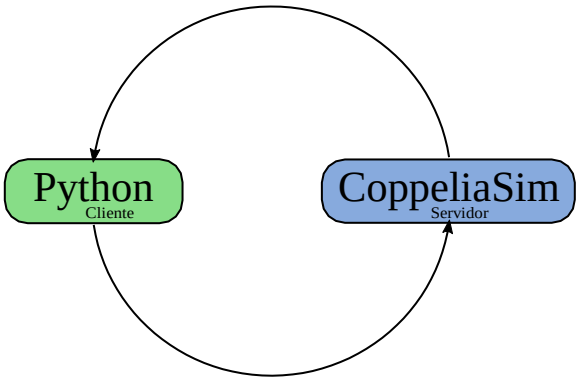

Figura 11: Diagrama de comunicación usando Remote API y Pyrhon.

funciones de control, comunicación y obtención de datos. El algoritmo 1, muestra ejemplo de como establecer la comunicación con el servidor desde el cliente.

Algoritmo 1 Comunicación cliente servidor

```
1: Procedimiento main( )
2:     clienteID = servidor(ip, puerto)
3:     si clienteID == 0 entonces      ◁ Si fue posible la conexión
4:         iniciaControlRobot()
5:     si no
6:         terminar()
7:     fin si
8: fin Procedimiento
```

2.4. Sistema de control del robot

Un sistema de control cuenta con un elemento de regulación para sí mismo o para otros sistemas. Pueden ser de ciclo abierto si no se toma en cuenta la salida del sistema o de ciclo cerrado cuando si se toma en cuenta. Debido a la inestabilidad del sistema (robot) a controlar, fue necesario hacer uso de un sistema de control de ciclo cerrado esto significa que, se tomará en cuenta la señal de salida mediante una retroalimentación, esta característica proporciona mayor estabilidad. Los sistemas de control de ciclo cerrado actúan en función de una entrada de referencia o estado deseado. Para esto, al estado deseado se le resta la señal de salida del sistema obteniendo el error. En función de esa señal de error, se realiza una acción, que se envía a los actuadores del sistema, y minimice el error con el fin de lograr la respuesta deseada.

Para fines prácticos de este trabajo, se debía conseguir que el robot (sistema) se mantuviera en equilibrio (lograr un ángulo de 0° respecto a la vertical). Para esto, se implementó un controlador PID[3].

Se trata de un mecanismo de control con una retroalimentación de ciclo cerrado, ver figura 12, en ésta es posible observar que al sistema recibe como entrada un error (e(t)) que se estima con la

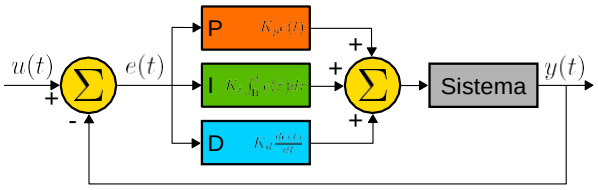

Figura 12: Diagrama de un controlador PID.

diferencia de la salida deseada (u(t)) y la salida obtenida (y(t)), el objetivo consiste en minimizar este error ajustando la entrada del sistema con base a tres parámetros (acciones): el proporcional, el integral y el derivativo, el ajuste de estos parámetros influye sobre alguna característica de la señal de salida obteniendo una respuesta mejor o peor. La ecuación 2, muestra la forma en que se obtiene la señal de salida.

$$y(t) = k_p e(t) + k_i \int_0^t e(\tau) d(\tau) + k_d \frac{de}{dt} \qquad (2)$$

La función que realiza cada uno de los parámetros es:

- La constante proporcional k_p contribuye a la estabilidad del sistema, un valor grande provocará oscilaciones al sistema, un valor muy pequeño no provocará oscilaciones pero ocasionará un valor estacionario, en ningún caso se llega al valor deseado y se provoca un offset.
- La constante integral k_i se utiliza para corregir el "offset"que se provoca por la constante proporcional por lo que se puede lograr la estabilidad sin embargo, un valor muy grande puede provocar oscilaciones y una pequeño elimina el error estacionario.
- La constante derivativa k_d sirve para minimizar las oscilaciones y las pendientes provocadas al aplicar las dos constantes anteriores, proporciona una respuesta rápida a los cambios de valores de entrada pero es muy sensible al ruido por lo que un valor muy grande puede producir un desastre en el sistema.

3. RESULTADOS

Para conseguir que el robot se mantuviera en equilibrio, se realizaron diversos ajustes al controlador PID que consistieron en proponer diversos valores para poder sintonizarlos bajo el esquema *Try And Error* que consiste en hacer que los parámetros lleguen a oscilar respecto al tiempo pero, se debe hacer de manera progresiva para esto, k_i y k_d deben valer 0:

1. Asignar un valor pequeño a la variable k_p y se va incrementando hasta que la variable comience a oscilar alrededor del valor deseado, una vez hecho esto, sedivide entre 2:

$$k_{p=}\frac{k_p}{2}$$

2. Toda vez que se obtienen el valor de la constante proporcional, se asigna un valor pequeño a k_i y se va incrementando hasta provocar una oscilación contínua, una vez hecho esto,

[3]Controlador Proporcional- Integral-Derivativo.

se divide entre 3: $$k_i = \frac{k_i}{3}$$

3. Asignar un valor pequeño a K_d y se va incrementando hasta que la variable oscile de manera permanente. El valor final de K_d debe ser la tercera parte del valor que tenía cuando el sistema comenzó a oscilar. $$k_d = \frac{k_d}{3}$$

Este método no garantiza que los valores sean precisos o los mejores por lo que se debe seguir probando. Finalmente se consiguió que el robot mantuviera el equilibrio de una manera aceptable. La prueba de equilibrio consistió en que el robot se mantuviera estable por sus propios medios.

Kp	Ki	Kd	Tiempo (ms)
13.7	0.199	1285	50
56.25	2.736	1389	10

CONCLUSIONES

Se logró diseñar e implementar de manera virtual un robot auto-balanceado en el simulador CoppeliaSim utilizando un controlador PID. Una vez diseñado y creado el robot, se desarrolló el algoritmo de control necesario para que actúe según lo esperado. El valor de los parámetros k_p, k_i y k_d que mantuvieron al robot en equilibrio de manera aceptable, se consiguieron sintonizándolos de manera empírica. La implementación de este controlador resultó ser un reto importante ya que de manera teórica, se logran resultados rápidos y posibles de acuerdo con las gráficas realizadas sin embargo, a la hora de ponerlo en práctica, los valores de los parámetros que mantuvieran al robot en equilibrio no fueron fáciles de conseguir. Dado que se pretende llevar a cabo este robot de manera física, es importante considerar algunas mejoras como:

- Diseñar el cuerpo del robot con medidas reales y con la forma real de manera que se acerque los más posible al robot físico, algo que requiere de un tiempo considerable.
- Si se utiliza el controlador PID, se puede buscar sintonizar los parámetros mediante la sintonía de oscilación (método de Ziegler-Nichols) o la sintonía basada en la curva de respuesta (Ziegler-Nichols y Cohen-Coon).
- Buscar si se logran mejores resultados con otros algoritmos de control como el control de estados o control de lógica difusa.
- Realizar otro tipo de prueba como prueba de equilibrio frente a empujes y equilibrio con movimientos y giros.

RECONOCIMIENTOS

Mis reconocimientos a la institución la Unidad Académica de Ingeniería, la cual me vio crecer como alumno para finalmente realizarme profesionalmente. A mis profesores por sus enseñanzas para adentrarme al mundo del saber.

REFERENCIAS

[1] Coppelia. [n.d.]. CoppeliaSim User Manual. https://www.coppeliarobotics.com/helpFiles/en/welcome.htm. Fecha de consulta, Marzo del 2021.

[2] Jesús de Miguel Fernández. 2017. Exploración del software de simulación V-REP. Master's thesis. Dep. de Ingeniería de Sistemas y Automática, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

[3] T. Arredondo y P. Castillo M. Torres. 2014. Survey and Comparative Study of Free Simulation Softwarefor Mobile Robots. Robótica (2014).

[4] Alberto Martín. 2016. Modelado y Simulación de un Robot Lego Mindstorm EV3 mediante V-REP y Matlab. Master's thesis. Departamento de Ingeniería de Sistemas y Automática, Universidad de Málaga.

[5] Carlos Martínez. 2015. Simulación en entornos con robots manipuladores móviles. Master's thesis. Escuela Técnica de Ingeniería Informática, Universidad Politécnica de Valencia.

[6] Pablo Parodi. 2017. Análisis de VREP como herramienta de simulación para Aerostack. Master's thesis. Escuela Técnica Superior de Ingenieros Informáticos.

[7] Javier Velasco. 2019. *Análisis y comparación de las principales plataformas de simulación robótica y su integración con ROS*. Master's thesis. Escuela Técnica Superior de Ingenieros Industriales, Universidad Politécnica de Madrid.

[8] Wikipedia. 2010. Inverted pendulum. https://en.wikipedia.org/wiki/Inverted_pendulum. Fecha de consulta: Febrero 25, 2019.