DESARROLLO DE PLANTAS VIRTUALES PARA EL ESTUDIO DE SISTEMAS DE CONTROL USANDO V-REP Y MATLAB

LAURA MARIA RODRIGUEZ RIVERA

HAROLD FERNANDO RUIZ BRAVO

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
SAN JUAN DE PASTO
2020

DESARROLLO DE PLANTAS VIRTUALES PARA EL ESTUDIO DE SISTEMAS DE CONTROL USANDO V-REP Y MATLAB

LAURA MARIA RODRIGUEZ RIVERA

HAROLD FERNANDO RUIZ BRAVO

TRABAJO DE GRADO PARA OPTAR POR EL TÍTULO DE INGENIERO ELECTRÓNICO

ASESOR

PhD. ANDRES DARIO PANTOJA BUCHELI

INGENIERO ELECTRÓNICO

ASESOR EXTERNO

MSc. JOHN BARCO JIMENEZ

INGENIERO ELECTRÓNICO

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
SAN JUAN DE PASTO
2020

NOTA DE RESPONSABILIDAD

“La Universidad de Nariño no se hace responsable por las opiniones o resultados obtenidos en el presente trabajo y para su publicación priman las normas sobre el derecho de autor.”

Acuerdo 1. Artículo 324. Octubre 11 de 1966, emanado del honorable Consejo Directivo de la Universidad de Nariño.

NOTA DE ACEPTACIÓN:

_____
_____
_____
_____
_____
_____

_____
Firma del presidente del jurado

_____
Firma del jurado

_____
Firma del jurado

San Juan de Pasto, 28/02/20

DEDICATORIA

“A mi padre, él es la persona con más tenacidad que conozco. Cada acto de su vida está enfocado en llevar adelante a sus hijos. Tan fuerte e insistente, una gran inspiración para mi vida”.

Laura María Rodríguez Rivera.

“A mi madre, padre y hermanos, son la muestra perfecta de sabiduría y paciencia, la energía perfecta que recarga mis ánimos para alcanzar cada una de mis metas”.

Harold Fernando Ruiz Bravo.

AGRADECIMIENTOS

“Agradecemos al departamento de Electrónica de la Universidad de Nariño por apoyarnos en las diferentes etapas de este proyecto”.

Laura Rodríguez y Harold Ruiz.

“Agradecemos al Profesor Andrés Pantoja y John Barco por apoyarnos en las diferentes etapas de nuestra formación profesional y por ayudarnos a lograr nuestras metas”.

Laura Rodríguez y Harold Ruiz.

.

RESUMEN

Las plantas virtuales son entornos basados en software, donde los usuarios interactúan en una serie de componentes gráficos que representan elementos de un modelo físico. Estas herramientas son unos de los instrumentos más versátiles en el proceso enseñanza-aprendizaje de los sistemas de control, ya que suplen la carencia de costosos laboratorios o plantas reales. En este trabajo se propuso el uso del simulador en 3D V-REP® y Matlab® para la construcción, comunicación e implementación de un sistema bola-viga, un sistema bola-placa, un péndulo de Furuta y un planeamiento de trayectorias para un robot diferencial terrestre por medio de una plataforma de co-simulación. Lo anterior es posible mediante la caracterización de modelos matemáticos y la inserción de las plantas en V-REP, junto con la creación de funciones, algoritmos y bloques en Simulink que posibilitan la comunicación transparente entre ambos programas. Finalmente, los laboratorios virtuales se evaluaron con la implementación de distintos tipos de controladores que permiten ver las diferencias y posibilidades al trabajar con plataformas especializadas que incluyen fenómenos físicos no modelados. Los resultados mostraron cómo los experimentos realizados en V-REP producen señales resultado de interacciones físicas complejas y respuestas desafiantes que deben corregirse con las estrategias de control planteadas para cada planta virtual.

ABSTRACT

Virtual plants are software-based environments where users interact on a series of graphical components that represent elements of a physical model. These tools are one of the most versatile instruments in the teaching-learning process of control systems, since they make up for the lack of expensive laboratories or real plants. In this work we propose the use of the 3D simulator V-REP® and Matlab® for the construction, communication and implementation of a ball-beam system, a ball-plate system, a Furuta pendulum and a trajectory planning for a terrestrial differential robot by means of a co-simulation platform. This is possible through the characterization of mathematical models and the insertion of the plants in V-REP, together with the creation of functions, algorithms and blocks in Simulink that enable transparent communication between both programs. Finally, virtual laboratories are evaluated with the implementation of different types of controllers that allow to see the differences and possibilities when working with specialized platforms that include unmodelled physical phenomena. The results show how the experiments performed in V-REP produce signals resulting from complex physical interactions and challenging responses that must be corrected with the control strategies proposed for each virtual plant.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

GLOSARIO

Agente: entidad física o virtual, capaz de percibir su entorno, procesarlo y actuar de manera racional, buscando optimizar el resultado.

Algoritmo: Conjunto definido de reglas o procesos que llevan a la solución de un problema en un número determinado de pasos.

API (Application Programming Interface): interfaz de programación de aplicaciones.

CD: Algoritmo de descomposición por celdas, cuyo término en inglés es *Cell Decomposition*.

Cinemática de un robot: Estudia el movimiento de un robot con respecto a un sistema de referencia.

Espacio de estados: El espacio de n dimensiones cuyos ejes coordenados están formados por el eje x_1, eje x_2,... , eje x_n se conoce como espacio de estados.

Estado: En un sistema dinámico el estado es el conjunto más pequeño de variables (llamadas variables de estado) tales que el conocimiento de dichas variables en $t = t_0$, junto con el conocimiento de la entrada para $t \geq t_0$, determinan por completo el comportamiento del sistema para cualquier tiempo $t \geq t_0$.

Frecuencia de muestreo: Se refiere a la cantidad de muestras por unidad de tiempo tomadas de una señal continua para producir una señal discreta.

Gradiente: el vector gradiente indica la dirección en la cual el campo de una función varía más rápidamente y su módulo representa el ritmo de variación de la función en la dirección de dicho vector gradiente.

Grafo: dato abstracto que consiste en un conjunto de vértices y un conjunto de aristas que establecen relaciones entre los nodos.

Odometría: es un método sencillo para estimación de posición y ampliamente usado para el cálculo de posición de los robots móviles.

PF: Algoritmo de función de planificación, cuyo término en inglés es *Planning Function*.

Planta virtual: es un entorno basado en software, donde los usuarios interactúan en una serie de componentes gráficos que representan elementos de un modelo físico.

PSO: Algoritmo de optimización de enjambre de partículas, cuyo término en inglés es *Particle Swarm Optimization*.

Restricción Holonómica: aquella en las que no interviene la velocidad y ocurre cuando los grados de libertad del robot están desacoplados. Si el número de grados de libertad controlables son todos los del robot, este es un robot holónomo.

Restricción no Holonómica: aquella que depende de la velocidad y demanda una trayectoria integrable que no puede deducirse derivando una restricción holónoma. Si el número de grados de libertad controlables no son todos los del robot, este es un robot no holónomo.

Robot: Maquina controlada por ordenador y programada para manipular objetos y realizar trabajos a la vez que interactúa con el entorno.

RRT: Algoritmo de árboles aleatorios de exploración rápida, cuyo término en inglés es *Rapidly-exploring Random Trees*.

Trayectoria: Es el camino que debe seguir un robot para realizar una tarea y alcanzar una meta, en los robots autónomos dicha trayectoria es generada por sí mismo, así como las acciones de control para mantenerse y recuperar dicha trayectoria.

Trapezoide: figura geométrica de cuatro lados, de los cuales no hay ninguno paralelo a otro.

Variable de estado: En un sistema dinámico las variables estado son las que conforman el conjunto más pequeño de variables que determinan el estado del sistema dinámico. Si para describir en su totalidad el comportamiento de un sistema dinámico se requiere por lo menos n variables $x_1, x_2, \ldots, x_n$(de tal forma que una vez dada la entrada para $t \geq t_0$ y el estado inicial $t = t_0$, el estado futuro del sistema queda completamente determinado), entonces dichas n variables se consideran un conjunto de variables de estado.

Vector de estado: Si se necesitan n variables de estado para describir completamente el comportamiento de un sistema dado, entonces estas n variables de estado se pueden considerar como los n componentes de un vector x. Dicho vector se conoce como vector de estados.

INTRODUCCIÓN

Una planta virtual es un entorno basado en software, donde los usuarios interactúan en una serie de componentes gráficos que representan elementos de un modelo físico. Estos elementos básicos se modelan en una plataforma de software especializada que forma mecanismos más complejos, de tal manera que el sistema emula estrechamente el rendimiento real de la planta. Algunos ejemplos de plantas virtuales incluyen sistemas mecánicos que trabajan en una simulación y laboratorios remotos, donde las plantas son reales (y costosas) pero operadas a través de Internet por diversos usuarios[1].

Por otro lado, estos entornos han tenido un impacto en muchas áreas de conocimiento e investigación de ingeniería, aprovechando al máximo el avance de las tecnologías basadas en simulación 3D y la relevancia que han tenido en la pedagogía y las actividades lúdicas; algunos referentes tecnológicos los podemos hallar en el proyecto Weblab[2] y en los prototipos didácticos creados por la empresa Quanser[3]. Esta penetración ha facilitado la mejora de muchos entornos de simulación 3D y 2D con fines educativos, lo que permite a los usuarios consolidar conceptos y fusionar la teoría con la práctica de una manera simple y accesible.

Una de las plataformas más extendidas en este tipo de aplicaciones es V-REP, cuyas características admiten formas simples y complejas (por ejemplo, diferentes tipos de robots), así como la inclusión de modelos 3D de un software CAD externo. Además, V-REP tiene afinidad con múltiples lenguajes de programación, incluido Matlab.

En el campo del software educativo, especialmente para las áreas técnicas, este trabajo pretendió resolver la deficiencia de equipos de laboratorio costosos y prototipos didácticos especializados en la mayoría de las instituciones educativas, mediante el uso de las tecnologías emergentes mencionadas anteriormente. Una plataforma interactiva que integró un simulador de planta y un software de control facilita la unión de conocimientos teóricos y prácticos, evitando grandes inversiones en equipos y permitiendo el estudio de diferentes tipos de plantas de control.

[1] VARGAS, H; DORMIDO, D; Duro, N; DORMIDO-CANTO, D. Creación de laboratorios virtuales y remotos usando easy java simulations y LABVIEW: El sistema heatflow como un caso de estudio. En: XXVII Jornadas de Automática, 2006, p. 1182-1188.

[2] International Inclusive Science, Technology, Engineering, and Mathematics Education. Low-Cost Engineering Laboratory Project. {En linea}. {10 de julio del 2018}. Disponible en: (https://istem.engineering.osu.edu/university-experiments/welab-low-cost-engineering-laboratory-project).

[3] QUANSER, PRODUCTS & LAB SOLUTIONS. {En linea}. {15 de Julio del 2018}. Disponible en: (https://www.quanser.com/).

Varios trabajos se han centrado en la construcción de plantas virtuales. Por ejemplo, Andaluz Víctor Hugo[4] presenta un simulador de realidad virtual 3D para el desarrollo de controladores para seguir trayectorias con un brazo robótico de 6 grados de libertad (DOF). Vásquez[5], muestra la implementación de una herramienta didáctica que permite simular, observar y analizar el comportamiento de los procesos industriales, haciendo hincapié en la comunicación con un PLC real. Utilizando las características principales del entorno V-REP, la propuesta de Domínguez[6] muestra la construcción, el diseño y la simulación de un robot LEGO EV3 en el simulador 3D V-REP. Además, para un sistema específico de bola y placa, Fábregas[7] propone una aplicación conjunta con un laboratorio virtual y remoto utilizado en un programa de posgrado en ingeniería de control, la cual usa una plataforma basada en Java. Finalmente, Sergio[8] implementa un sistema de control proporcional derivativo con compensación gravitacional y un sistema de control visual IBVS (Image-Based Visual Servoing) para el guiado de distintos tipos de robots en el Simulador en 3D V-REP.

Este trabajo describió el uso de una co-simulación entre un entorno libre de plantas virtuales (V-REP) y un lenguaje común para el estudio del control (Matlab), trabajando en una plataforma comunicada por una API (interfaz de programación de aplicaciones). Además, se elaboró una plataforma que permitió enlazar el simulador V-REP con Simulink destacando que la API usada se manipuló desde el lenguaje de programación C++. Para mostrar la metodología, se presentó un sistema bola viga, un sistema bola placa, un péndulo de Furuta y un planeamiento de trayectorias para un robot terrestre. Cabe señalar que este trabajo también se enfocó en el desarrollo de controladores clásicos y avanzados para cada uno de los casos de estudio, además se ha propuesto una metodología pedagógica que aproveche las plantas virtuales. El método se pudo adaptar para la implementación de diferentes plantas que emulan un comportamiento real en contraste con las simulaciones basadas en modelos estándar.

Para analizar las diferencias entre los resultados de la simulación y los obtenidos en las plantas de ejemplo (Matlab y Simulink), se ajustaron varios controladores (como, PID, LQR y MPC, entre otros) de acuerdo a las necesidades y características de cada planta. La comparación muestra la brecha entre un controlador real y una

[4]ANDALUZ, V.H; CHICAIZA, F.A; GALLARDO, C; QUEVEDO, W.X; VARELA, J; SÁNCHEZ, J.S; ARTEAGA, W.X. Unity3DMatLab Simulator in Real Time for Robotics. En: Proceedings of the third 3rd AVR international conference on Augmented Reality, Virtual Reality and Computer Graphics. AVR, 2016.p.246-263.

[5] VÁSQUEZ, R.D; SARMIENTO, H.O; MUÑOZ, D.S. Propuesta de implementación de plantas virtuales para la enseñanza de programas de control lógico. En: ACOFI, 2016, v.11, n.22, pp. 1-13.

[6] DOMINGUEZ, A. Modelado y Simulación de un Robot LEGO Mindstorms EV3 mediante V-REP y Matlab. Trabajo de grado (Ingeniero de sistemas). Málaga, 2016. Universidad de Málaga. Departamento de Ingeniería de Sistemas y Automática.

[7] FABREGRAS, E; DORMIDO-CANTO, S; DORMIDO, S. Virtual and Remote Laboratory with the Ball and Plate System.. En: IFAC-PapersOnLine, 2017, v.50, n.1, pp. 9132-9137.

[8] SERGIO, J. Control de Posición y Visual de Sistemas de Manipulación Autónomos. Trabajo de grado (Master Universitario en Automática y Robótica). Alicante, 2017. Universidad de Alicante. Departamento de Ingeniería de Sistemas.

simulación ideal ya que la planta virtual emula propiedades físicas como fricción, desplazamientos y deslizamientos, que están cerca de la realidad. Esta condición permitió a los estudiantes enfrentar condiciones empíricas que no se obtienen en una simulación (por ejemplo, ruido, perturbaciones, límites de actuadores y barreras físicas), mejorando la capacitación teórico-práctica y la aplicación real de los conceptos de control.

El trabajo se organizó de la siguiente manera: en el capítulo 1 se presentó el esquema de co-simulación con Matlab y Simulink; en el capítulo 2 se describió el desarrollo de las plantas virtuales; en el capítulo 3 se explicó los resultados obtenidos, teniendo en cuenta las diferencias de los controladores dentro de la simulación y las plantas reales; en el capítulo 4 se mostró las conclusiones y en el capítulo 5 se expuso el trabajo futuro. En los anexos se encuentran los videos tutoriales, algunos códigos guía y los artículos presentados en el 4th IEEE Colombian Conference on Automatic Control (CACC) y el 2nd Congreso Latinoamericano de Automática y Robótica (LACAR).

Modalidad: Investigación.

Línea: Robótica y Control.

Descripción del problema

Es bien sabido que, dentro del desarrollo metodológico de la enseñanza de la ingeniería electrónica, el área de control presenta algunas dificultades en la explicación y comprensión de conceptos. Por lo anterior, varios estudios como el presentado por el profesor Kevin Passino de la Universidad Estatal de Ohio mediante el programa weLab[2], plantea la necesidad de reforzar el conocimiento teórico con prácticas de laboratorio realizadas por los estudiantes, que refuercen los conceptos teóricos adquiridos en clase.

Los laboratorios en el área de control están generalmente compuestos por diversos dispositivos, que pueden ser comercializados por muchas empresas como QUANSER[3], quienes se dedican a fabricar plantas didácticas, siendo estas indispensables para la enseñanza e incluso para la prueba y diseño de controladores. Sin embargo, muchas universidades, y en especial las de países en vía de desarrollo, no cuentan con los recursos necesarios para adquirir estos equipos, lo cual repercute en el nivel investigativo de las instituciones y sus estudiantes, así como para la creciente demanda que tienen las industrias de profesionales bien capacitados.

Es aquí donde el uso de nuevas tecnologías se hace pertinente. En los últimos años se han creado una serie de programas y simuladores de gran calidad que permitan emular toda clase de prácticas de laboratorio de una manera muy fiable y realista. La gran mayoría de ellos son diseñados específicamente para la enseñanza de materias y temáticas especificas tales como comunicaciones, potencia o control, de forma que pueda hacerse de manera fácil y sin requerir de costosos equipos que pueden no tener los centros de enseñanza.

Por lo anteriormente expuesto, se ve la necesidad que, dentro del avance del Departamento de Electrónica, se diseñen las suficientes plantas virtuales requeridas para la pedagogía de la ingeniería y el desarrollo de la investigación en el área de control, propuesta como una de las líneas temáticas del perfil profesional.

Justificación

La Universidad de Nariño, al igual que varias universidades del país, carece de los recursos y el equipo necesario para la prueba de algoritmos de control en plantas físicas. A pesar de ello, la institución y particularmente su Departamento de Electrónica, tiene la responsabilidad de preparar de la mejor manera a sus estudiantes en los conceptos teóricos y prácticos, específicamente en el área de control. Para suplir la carencia de laboratorios y evitar las posibles falencias que los alumnos pueden llegar a tener en la práctica, se plantea aprovechar los recursos disponibles actualmente para simulación de sistemas físicos en 2D y 3D.

Existen varios simuladores en 2D y 3D como, Stage,Player y Gazebo[9] que emulan robots en dos o tres dimensiones y se programa en lenguajes C y C++. También se encuentra Microsoft Robotics Developer Studios[10], que brinda escenarios de robots reales en tres dimensiones y que se programa en cualquier lenguaje adscrito a Microsoft (C#,Visual Basic). Otro es Mariluo Robotics studio[11], que simula múltiples tipos de robots en tres dimensiones y admite muchos lenguajes como Matlab y Java. Finalmente, se encuentra V-REP[12], cuyas características permiten simular distintos tipos de robots, importar modelos en 3D y tener afinidad con muchos lenguajes de programación, entre ellos Matlab.

[9]The Player Project. Player Project. {En linea}. {15 de marzo del 2018}. Disponible en: (http://playerstage.sourceforge.net/).
[10]Microsoft. Welcome to Robotics Developer Studio. {En linea}. {15 de marzo del 2018}. Disponible en: (https://msdn.microsoft.com/en-us/library/bb648760.aspx).
[11]anyKode. Marilou: the universal mechatronic software. {En linea}. {15 de marzo del 2018}. Disponible en: (http://www.anykode.com/marilou.php).
[12]Coppelia Roboticts V-REP. V-REP User Manual. {En linea}. {10 de enero del 2018}. Disponible en: (http://www.coppeliarobotics.com/helpFiles/).

Pese a lo anterior, el presente trabajo no se enfocó en la creación de nuevos softwares de simulación, sino en el desarrollo de una interfaz de programación de aplicaciones (API) para Matlab que, usando como base la ya existente, permita un manejo más intuitivo, menos complejo y con mayor funcionalidad de las plantas virtuales. Por lo anterior se optó por usar el programa V-REP, que es ampliamente usado como una herramienta para prueba y desarrollo de controladores, que se caracteriza por tener una documentación extensa, permitir una importación de modelos en 3D y disponer de una versión educacional gratuita.

En ese sentido, se podrá instruir las temáticas del área de control de una forma más sencilla y amigable que facilite la labor del docente, para así aprovechar de mejor manera el tiempo disponible sin tener que promover la curva empinada de aprendizaje de los entornos virtuales y aplicar directamente la adaptación de plantas y la enseñanza del diseño de controladores. Resulta también interesante que el desarrollo de una plataforma flexible puede aplicarse en otras instituciones para incentivar el uso de las herramientas disponibles.

Es por ello, que en este trabajo se elaboró herramientas que facilitan el uso de estos softwares, así mismo se implementaron algoritmos que sirvieron como base y apoyo para el aprendizaje de las temáticas relacionadas con control.

Objetivo General

Desarrollar una plataforma entre V-REP y SIMULINK y adecuar la API existente con Matlab, Con el fin de diseñar e implementar diferentes tipos de plantas virtuales orientadas al estudio de sistemas de control.

Objetivos específicos

- Caracterizar diferentes tipos de plantas virtuales por medio de un modelado matemático y su desarrollo mecánico en el simulador V-REP.
- Elaborar librerías que permitan el manejo de V-REP desde Simulink, y adecuar la API existente entre V-REP y Matlab mediante la creación de funciones para permitir el intercambio de datos entre las plataformas en tiempo real en un esquema de co-simulación.
- Desarrollar controladores prototipo con distintas estrategias para la evaluación de las plantas en la plataforma de acuerdo con los requerimientos de diseño.
- Proponer un esquema de guías de laboratorio para estimular el uso de la plataforma y el diseño de nuevas plantas.

Contribuciones de este trabajo

Con la construcción, comunicación e implementación de las diferentes plantas virtuales, los docentes y estudiantes cuentan con nuevas herramientas para el estudio de sistemas de control. De esta forma, se mitigo muchas deficiencias que pueden presentar los estudiantes a la hora de programar los diferentes algoritmos de control, además de proveer un instrumento útil para los docentes.

De igual forma, las plantas virtuales se usaron para reducir la brecha existente entre la simulación y el controlador real, debido a que estas se aproximan más a la realidad que una simulación. Esto permitió que estudiantes se enfrenten a exigencias reales que no se aprecian en una simulación (e.g., ruido, perturbaciones, límites de actuadores, barreras físicas), mejorando la formación teórico-práctica y la aplicación real de los conceptos de control.

Por último, un estudio adecuado de la construcción de plantas virtuales, puede motivar a la inclusión de este tipo de instrumento pedagógicos en otros ámbitos como la automatización, robótica y potencia, por lo tanto, es posible abarcar más áreas de conocimiento práctico en ingeniería.

Marco Teórico

La elaboración de una planta virtual en un entorno de simulación 3D presenta dos procesos importantes. El primero consiste en el análisis de las ventajas y limitaciones que presenta el software con respecto a los diferentes objetos, formas, sensores y actuadores que constituyen la planta. El segundo implica la comunicación entre los objetos que forman parte de la planta virtual y un programa externo, mediante una API (interfaz de programación de aplicaciones), puntualizando que dicha comunicación en el presente trabajo se realizó con el programa Matlab y Simulink.

Entornos de Simulación 3D

En el mundo de las plantas virtuales, existen varios programas y simuladores de gran calidad que permiten emular toda clase de prácticas de laboratorios de una manera muy fiable y realista. En el mercado de los simuladores se dispone de muchas opciones; algunos de ellos son:

Stage, Player y Gazebo

Son tres aplicaciones creadas por el laboratorio de robótica de investigación de la USC ("University of Southern California"). Player proporciona una interfaz de red para una variedad de hardware de robots y sensores las cuales se escriben en cualquier lenguaje de programación (principalmente en C y C++) y se ejecutan en cualquier computadora con una conexión de red al robot. Además, dispone de varios simuladores que emulan un robot en entornos de dos y tres dimensiones, como Stage (2D) y Gazebo (3D).

Estas tres aplicaciones se pueden descargar de manera gratuita en la página del proyecto[9] y permiten crear toda clase de prácticas de laboratorio en robótica facilitando la prueba y verificación de algoritmos antes de aplicarlos en un robot real. Las principales restricciones que suponen usar este entorno es saber programar C o C++ y tener conocimientos de Linux para instalar las librerías y simuladores. Asimismo, el entorno no es multiplataforma por lo que en Windows el programa no funciona. También, dichas librerías y aplicaciones no han sido actualizadas desde 2010.

Microsotf Roboticts Developer Studio

Es una plataforma integrada diseñada para la creación, programación, simulación e implementación de diferentes plataformas robóticas[10]. Este programa cuenta con una licencia gratuita con propósitos docentes (para profesores y estudiantes), donde se pueden desarrollar algoritmos para robots usando la programación grafica que proporciona, la cual sirve de ayuda para comenzar a programar robots. El inconveniente es el tiempo necesario para aprender dicho entorno, puesto que contiene gran cantidad de módulos y programas; adicionalmente al ser una herramienta de Microsoft solo se pueden programar robots con los lenguajes de programación propietarios (C#, Visual Basic, entre otros.), por lo que no es posible utilizar otros lenguajes.

Marilou Robots Studio

Es un entorno que permite la simulación de múltiples tipos de robots (Brazos manipuladores, robots móviles, humanoides, etc.) en tres dimensiones capaces de representar fenómenos del mundo real. Este simulador[11] incluye varias librerías que facilita la programación del robot modelado en gran variedad de lenguajes (C, C++, C#, etc), y también es compatible con lenguajes como Matlab o Java.
Este entorno posibilita el modelado y simulación por medio de una interfaz gráfica que, en teoría, debería ahorrar tiempo. Está disponible para varias plataformas (Windows y distribuciones Linux), sin embargo, el principal inconveniente de este

entorno es que se necesita adquirir una licencia, incluso para la versión de estudiantes.

V-REP

Es un entorno de simulación de robótica[12] que posibilita simular distintos tipos de robots en tres dimensiones. Este programa está disponible en los sistemas operativos Windows, distribuciones Linux y MacOs, y cuenta con una documentación extensa y detallada. La plataforma tiene predefinidos gran variedad de robots, actuadores y sensores, lo que favorece la construcción de plantas virtuales, además de permitir la importación de piezas procedentes de programas de diseño CAD (como Solidwords). Igualmente, la plataforma se encuentra integrada con múltiples lenguajes de programación entre ellos Matlab.

Este es el Simulador de robótica que se eligió para modelar y simular las cuatro plantas virtuales que se presentaron en este trabajo. Toda la información básica de este entorno 3D se puede encontrar en detalle en la siguiente subsección. Las principales razones para elegir el entorno, adicionales a las ya mencionadas, son la disponibilidad de una licencia totalmente gratuita y las actualizaciones del software son constantes.

Generalidades del entorno V-REP

El software cuenta con cuatro motores para cálculos cinemáticos y dinámicos avanzados que permiten la simulación de entornos y partes físicas, motores y sensores, entre muchas opciones de trabajo en su interfaz gráfica. La interacción se realiza a través de más de 400 funciones en la API nativa del entorno de simulación[13]. Es de destacar que el simulador es capaz de emular comportamientos físicos reales como los producidos por la gravedad, deslizamientos de partes por inercias, posibles perturbaciones externas, además de colisiones y fricciones entre los diferentes objetos que se encuentran en el entorno de trabajo.

Los componentes físicos en V-REP están compuestos por diferentes tipos de "formas[14]", clasificadas según su complejidad (forma aleatoria simple y compuesta, forma convexa simple y compuesta y forma pura simple y compuesta). A estas "formas" se les pueden alterar parámetros físicos como la masa y color, entre otras características, mediante la ventana de configuración mostrada en la opción *show dynamic properties dialog* y las dimensiones eligiendo *view/modify geometry.* De esa misma manera es necesario decir que dichos componentes se articulan entre

[13]Coppelia Roboticts V-REP. V-REP Regular API Function List. {En línea}. {10 de enero del 2018}. Disponible en: (http://www.coppeliarobotics.com/helpFiles/).
[14]Coppelia Roboticts V-REP. V-REP shapes. {En línea}. {30 de abril del 2018}. Disponible en: (http://www.coppeliarobotics.com/helpFiles/en/shapes.htm).

sí por junturas[15] o *joints* que son clasificadas según el tipo (rotacionales o trasnacionales) o el modo de operación (pasivo, cinemática inversa, dependiente, movimiento o torque-fuerza). Las junturas se mueven por medio de motores que representan los actuadores principales de las plantas. Por su parte, existen gran variedad de sensores[16] para completar el lazo de realimentación en los esquemas de control. También se destacan algunos objetos adicionales presentes en el trabajo como los *dummies*[16].

En cuanto al tiempo de simulación, V-REP presenta restricciones en el tiempo de muestreo mínimo que se puede lograr, resaltando que se encuentran entre 1 y 200 milisegundos. Estos periodos dependen principalmente del equipo de cómputo utilizado, la complejidad de la planta (número y tipo de formas, actuadores y sensores) y el uso de servidores externos, como la API remota, para interconexión con otros programas.

[15]Coppelia Roboticts V-REP. V-REP joint description. {En línea}. {3 de marzo del 2018}. Disponible en: (http://www.coppeliarobotics.com/helpFiles/en/jointDescription.htm).
[16]Coppelia Roboticts V-REP. V-REP object description. {En línea}. {3 de marzo del 2018}. Disponible en: (http://www.coppeliarobotics.com/helpFiles/en/objects.htm).

1. DESARROLLO DE LA INVESTIGACIÓN

En esta sección se describe el funcionamiento y adecuación de la API existente entre V-REP y Matlab que permitió el intercambio de datos entre las plataformas en tiempo real en un esquema de co-simulación. Adicionalmente se explica el desarrollo de la plataforma que posibilito la co-simulación entre los softwares V-REP y Simulink.

1.1. MODIFICACIÓN DE API REMOTA V-REP MATLAB

La interfaz de aplicaciones de programación – API[13], se utiliza para controlar una simulación (propiedades de lectura y escritura y atributos de componentes, actuadores y sensores) desde hardware o software externo. Consiste en más de cien funciones, que se pueden invocar desde una aplicación C ++, Python, Java o un script de Matlab. Las funciones interactúan con V-REP a través de Blue Zero Middleware[17] y su complemento de interfaz con V-REP[13]. La API remota permite que una o más aplicaciones externas interactúen con el simulador en modo síncrono o asíncrono.

En el caso de una co-simulación en modo síncrono, el cliente (desarrollo en Matlab) debe estar sincronizado con el progreso de la simulación en V-REP. En este modo, cada paso de la simulación es equivalente a un período de muestreo, determina un nuevo ciclo de cálculo basado en los datos enviados a los actuadores y recopila los datos proporcionados por los sensores. De esta manera, se proporciona la retroalimentación y se sincroniza ambos programas. En el modo asíncrono, el cliente no debe estar sincronizado con el progreso de la simulación en V-REP, por lo tanto, los datos enviados a los actuadores y sensores pueden ser remitidos varias veces desde el servidor y el cliente escoge el último dato que se presentó cuando acabo un ciclo, también se resalta que en este trabajo se utilizó el modo asíncrono para elaborar la co-simulacion entre ambos programas debido a que es el modo que más se acerca a la realidad.

En el desarrollo propuesto en este trabajo, se adaptó la API remota original de V-REP y Matlab[13]. Para este propósito, se ha modificado el script "remotApi", junto con las funciones de transmisión bidireccional de información. En consecuencia, los comandos utilizados en Matlab son más simples de usar y tienen solo los parámetros necesarios, facilitando la ejecución y configuración de la co-simulación.

[17]Open Source Robotics Fundation. ROS.org | Powering the world's robots. {En línea}. {30 de marzo del 2019}. Disponible en: (http://www.ros.org/).

La co-simulación se ha realizado enviando comandos desde Matlab que, a través de la API, se ejecutan en V-REP. Una gran cantidad de comandos ofrece la versatilidad que le permite a Matlab obtener todo tipo de información de los objetos en la planta, así como establecer ciertos parámetros en los actuadores (por ejemplo, velocidades o posiciones en los motores). Sin embargo, las funciones originales son bastante complejas dada la gran cantidad de parámetros que se necesitan usar. Por esta razón, se ha modificado el script "remoto", lo cual ha facilitado el enlace entre las dos plataformas.

El manejo y configuración detallados de la API remota original y modificada entre V-REP y Matlab se encuentra en el Anexo 1.

1.2. DESARROLLO DE UNA API REMOTA V-REP SIMULINK

En el ámbito de la enseñanza de sistemas de control, una de las herramientas más populares es el entorno de programación visual Simulink, que ofrece una interfaz muy intuitiva y de fácil uso. Lastimosamente Coppelia Robotics no ofrece una forma de enlazar V-REP con Simulink, por lo que en este trabajo se ha propuesto desarrollar un bloque en Simulink que sirva para este objetivo, por este motivo se usó las S-functions y la API remota nativa de V-REP con C++.

Las S-funtions (funciones del sistema) son una representación en lenguaje informático de un bloque de Simulink escrito en MATLAB, C, C++ o Fortran. Estas funciones siguen una forma general y se pueden acomodar a sistemas continuos, discretos e híbridos. Mediante el usó de un conjunto de reglas simples, se puede implementar un algoritmo en cualquiera de los lenguajes anteriormente mencionados y utilizar el bloque S-Function para añadirlo a un modelo de Simulink, cuya interfaz se puede personalizar mediante enmascaramiento.

El S-Funtion Block, como cualquier bloque de Simulink, consiste en un conjunto de entradas, estados, parámetros y salidas, donde las salidas están en función del tiempo de simulación, las entradas, los parámetros y los estados. Como primer paso, se procedió a realizar un documento en C++ que sirve como configuración inicial, conteniendo parámetros como el tiempo de muestreo, el número de entradas, el número de salidas, y demás configuraciones iniciales. Para lo anterior se usó como base las plantillas ofrecidas por Matlab[18].

Dada la complejidad del código, esta configuración se ha hecho lo más general posible, de forma que dicha configuración no deba cambiarse si se requiere modificar el bloque (realizar una nueva planta o elegir una de las plantas

[18]MatWorks. Templates for C S-functions. {En línea}. {7 de septiembre del 2019}. Disponible en: (https://www.mathworks.com/help/simulink/sfg/templates-for-c-s-functions.html).

predefinidas). Para lograr esto, se han establecido salidas, entradas y un tiempo de muestreo dinámicos que dependan de unos parámetros predefinidos usando la guía dispuesta por MathWorks[19].

Aparte de la configuración, la S-Function sigue una serie de etapas durante el proceso de simulación, como la inicialización, actualización, obtención de salidas y terminación. En cada paso de una simulación, el motor de simulación invoca un método para realizar una tarea específica. Las etapas de inicio y fin sirven para realizar actividades de inicialización y terminación requeridas por la S-funtion sólo una vez, tales como configuración de datos de usuario o la inicialización de vectores de estado en una S-funtion. En la etapa de salida, las salidas se calculan hasta que todos los puertos de salida del bloque sean válidos para el paso de tiempo actual, es decir, que todos los valores de salida estén en un cierto rango de error y en la actualización el bloque realiza actividades de un solo paso, como la actualización de estados discretos.

Estos procesos normalmente están definidos y se programan dentro del mismo archivo de configuración, sin embargo, para una mayor facilidad de personalización, se ha separado estos procesos de la configuración en un archivo conocido como *wrapper*[20].

Este nuevo archivo se comporta de forma muy similar a un archivo de programación en Arduino, compuesto por unos encabezados, un código de inicialización, un loop principal, así como dos secciones extra que son la de finalización y la de actualización de estados. Cada una de estas secciones permite incluir la librería de la API de V-REP, inicializar la conexión, actualizar estados, actualizar las entradas y finalizar la conexión. Cabe aclarar que la API está desarrollada por Coppelia Robotics, por lo tanto, la programación para este bloque es muy similar a la que se tuvó con Matlab.

Debido a que el bloque se hizo con fines pedagógicos, la programación se ha realizado de forma que tuviera pre-programados las tres primeras plantas que se muestran más adelante en este documento y se dejó el espacio para quien desee incluir plantas adicionales, lo cual se detalla más a fondo en el Anexo 1.

Al realizar pruebas del desempeño del bloque que se ha programado, se presentaron dificultades de funcionamiento debido a que, por la naturaleza de

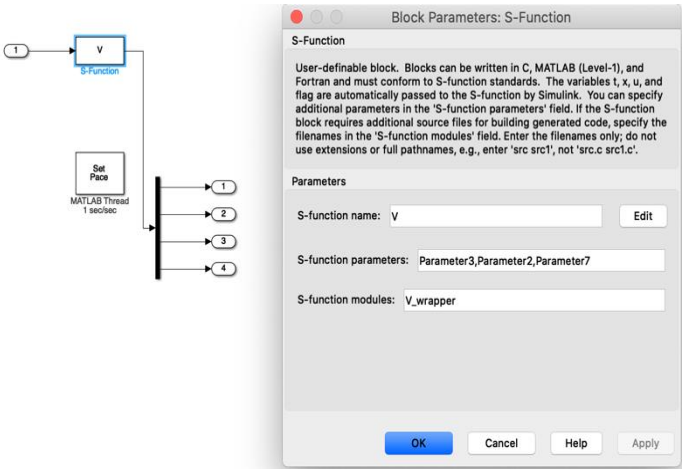
[19]MatWorks. Create a Basic C Mex S-Function. {En línea}. {12 de septiembre del 2019}. Disponible en: (MatWorks, Create a Basic C Mex S-Function, disponible en: (https://www.mathworks.com/help/simulink/sfg/example-of-a-basic-c-mex-s-function.html).

[20]San Diego State University. Mex S-Funtion Wrapper. {En línea}. {12 de septiembre del 2019}. Disponible en: (MatWorks, Create a Basic C Mex S-Function, disponible en: (https://edoras.sdsu.edu/doc/matlab/toolbox/simulink/sfg/sfcn_rtw8.html).

Simulink, la simulación no cumple exactamente el tiempo de muestreo. Por lo anterior, se agregó un bloque extra que permite sincronizar los tiempos. En este caso, se ha utilizado el bloque Simulation Pace, de la librería Aerospace Blockset. Este bloque permitió ejecutar la simulación a un ritmo específico, por lo que es posible configurar el paso a tiempo real. No obstante, esto no es una garantía, ya que, si la ejecución del código interno dura más que el tiempo de muestreo, el bloque estaría deshabilitado. En caso contrario el tiempo de simulación se detendrá hasta que la ejecución se haya cumplido, lo cual emula de cerca una simulación en tiempo real.

Ahora, puesto que el bloque está configurado para tener sólo una salida vectorial, se ha usado un demultiplexor, quedando el sistema final como se muestra en la Figura 1.

Figura 1. Sistema final

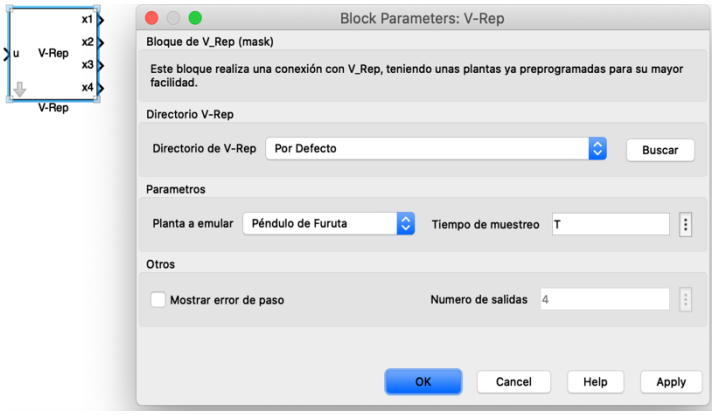


Fuente: Esta investigación, diagrama realizado en Simulink por Laura Maria Rodriguez.

Por último, el bloque original no es muy estético y es poco intuitivo, razón por la que se ha procedido a crear una máscara[21], que no es más que una interfaz personalizada para un bloque que oculta su contenido, haciéndolo parecer como un bloque promedio de Simulink, con su propio icono y cuadro de diálogo de parámetros. La máscara encapsula la lógica del bloque, proporciona acceso controlado a los parámetros del bloque y simplifica la apariencia gráfica de un modelo. Dando como resultado la interfaz mostrada en la Figura 2.

[21]MathWorks. Creating a Mask: Parameters and Dialog Pane. {En línea}. {15 de septiembre del 2019}. Disponible en: (https://www.mathworks.com/videos/creating-a-mask-parameters-and-dialog-pane-120557.html).

Figura 2. Bloque de V-REP enmascarado



Fuente: Esta investigación, diagrama realizado en Simulink por Laura María Rodríguez.

Es importante puntualizar que el funcionamiento del bloque de conexión V-REP-Simulink va estar limitado por las características de configuración de ambos programas, enfatizando que el parámetro *solve* de Simulink ha sido determinante en los resultados obtenidos de las diferentes plantas virtuales. Para los casos implementados en este documento se ha usado el *solve* predeterminado por Simulink (ode45).

2. CASOS DE ESTUDIO

Para ilustrar el proceso de implementación de plantas virtuales y su control a través de Matlab y Simulink, se ha presentado el desarrollo de un sistema Bola-Viga, un sistema Bola- Plato, un péndulo de Furuta y un planeamiento de trayectorias de un robot terrestre. El primer sistema está compuesto por un motor que gira una viga que contiene una bola. Dado que el motor está unido a una base fija, la rotación producida por el par permite que la viga ubique la bola en la posición deseada. El sistema tiene un grado de libertad y la posición de la pelota se puede medir usando un sensor de proximidad ubicado en un extremo del haz o también usando la propiedad de posición de la pelota proporcionada por la API V-REP Matlab.

La segunda planta consiste en una superficie que gira sobre una articulación central movida por dos servomotores que controlan el ángulo de rotación en las coordenadas x e y. El sistema completo tiene dos grados de libertad y la medición de la posición de la pelota en la superficie puede ser detectada por una cámara o directamente con las propiedades de la ubicación de la esfera en el espacio.

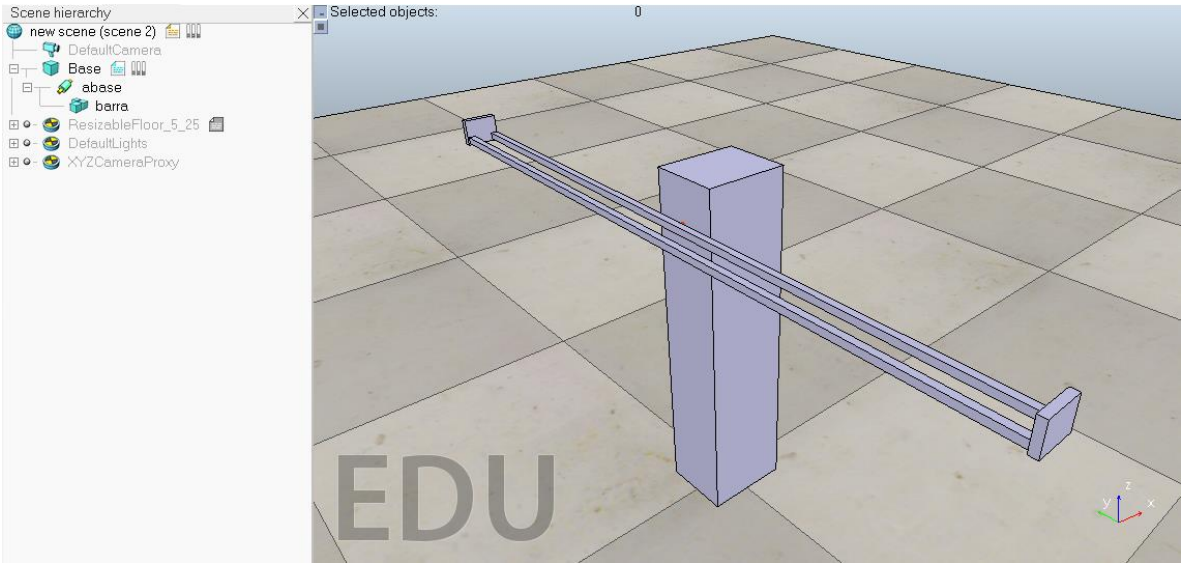
El tercer sistema consta de dos cuerpos inerciales conectados. El primer elemento es un pilar central con momento de inercia, rígidamente conectado a un brazo horizontal de longitud l_a y masa homogéneamente distribuida en línea. El segundo es un péndulo de longitud l_p y masa homogéneamente distribuida. Finalmente, la cuarta planta son varios escenarios con obstáculos (formas primitivas) en el que se encuentra un robot con configuración diferencial Pioneer_p3dx, el cual se ha guiado a través de trayectorias libres de obstáculos.

2.1. SISTEMA BOLA VIGA

2.1.1. Desarrollo mecánico

Para establecer la estructura de la planta, se ha utilizado las llamadas "formas puras" en V-REP, que son sólidos simples como barras, placas y esferas. Cada elemento se ha definido con propiedades como dimensiones, masa y posición, características que determinan la operación dentro del esquema 3D. Una vez que se han definido todos los elementos necesarios para la construcción del sistema, las diferentes piezas se han unido mediante la opción *group select shapes*, que junta las diferentes formas en un conjunto, como el que se muestra en la Figura 3. En este estado inicial, la base y el riel se han asociado mediante una juntura que permite el movimiento de la viga.

Figura 3. Modelo de formas puras sistema Bola Viga.



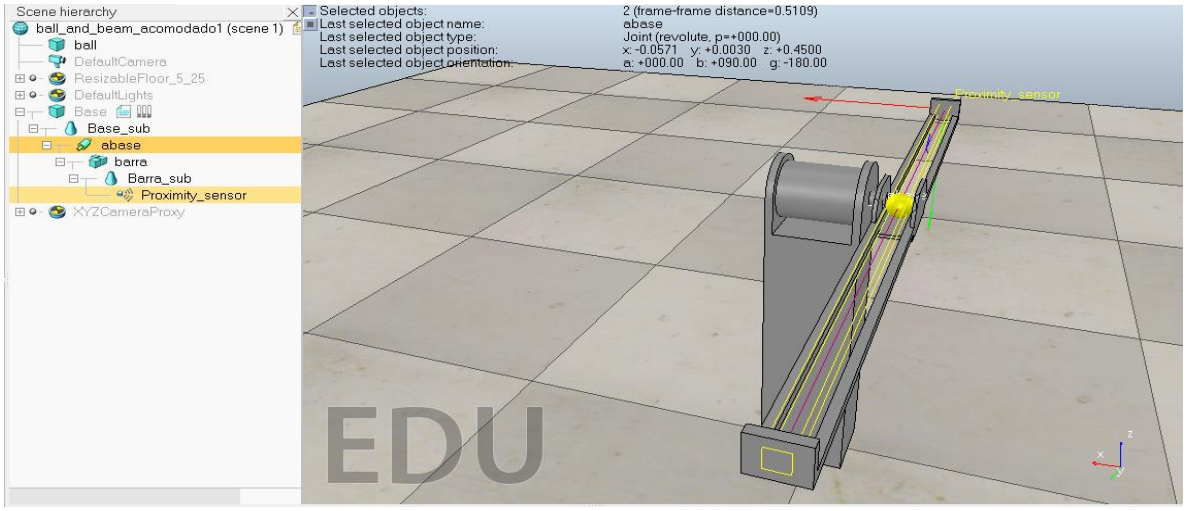
Fuente: Esta investigación, diseño realizado en V-REP por Harold Fernando Ruiz.

Como las formas originales de V-REP no son estéticas y representan formas básicas, para la emulación de un esquema real se ha necesitado importar modelos diseñados en herramientas CAD como Solidworks[22]. Para hacer esto, los archivos se importan en formatos obj, dxf o stl que contienen formas básicas pero limpias. Vale la pena señalar que, si se utilizan figuras complejas, la simulación se vuelve lenta y los tiempos de muestreo en la co-simulación aumentan considerablemente.

Las piezas que se han importado se agrupan en una estructura de árbol para que el simulador asocie cada forma básica a sus propiedades en el idioma nativo. Además, los acoplamientos entre todos los elementos son indispensables, porque si las formas no se unen correctamente, las piezas caen al suelo cuando comienza la simulación (similar a los sólidos en un entorno real). La agrupación se ha realizado jerárquicamente tomando un "padre" y asociando un "hijo" a través de un menú simple. Una vez que todas las formas puras de V-REP se han asociado entre sí con los modelos importados del software CAD, los sólidos originales de V-REP se vuelven "invisibles", produciendo un esquema pulido como el que se muestra en la Figura 4.

[22]Solidworks Corporation. Guía del estudiante para el aprendizaje del software Soidworks". {En línea}. {10 de febrero del 2018}. Disponible en: (https://www.solidworks.com/sw/docs/Student_WB_2011_ESP.pdf).

Figura 4. Sistema Bola Viga completo.



Fuente: Esta investigación, diseño realizado en V-REP por Harold Fernando ruiz.

En cuanto al actuador, la planta tiene un motor que mueve el riel. La juntura se ha configurado en el modo torque/ fuerza, mientras que el motor se ha seleccionado en el modo de velocidad angular (se selecciona la opción *motor enable*). Para capturar la posición de la pelota con un sensor real, se ha usado un sensor de proximidad, cuya distancia se exporta a Matlab para estimar la posición de la pelota. Sin embargo, si se desea facilitar la medición de la ubicación de la bola, se puede utilizar directamente las propiedades de la esfera moviéndose libremente en el riel, obtenidas a través de la API. El desarrollo mecánico del sistema y su comunicación con Matlab y Simulink se encuentran con más detalle en el Anexo 1.

2.1.2. Modelo Matemático

Según Huang[23] el espacio de estados no lineal que define el sistema es

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} x_3 \\ x_4 \\ \dfrac{(m_b\, x_1 x_4^2 - m_b g \sin x_2)}{\left(m_b + \left(\dfrac{I_b}{R^2}\right)\right)} \\ \dfrac{\left(u - (m_b g x_1 \cos x_2) - (2 m_b x_1 x_3 x_4)\right)}{\left(I_v + (m_b x_1^2)\right)} \end{bmatrix}, \quad (1)$$

[23]HUANG, J; LIN, C. Robust nonlinear control of the ball and beam system. En: Proceedings of of the American Control Conference. ACC, 1995.p.306-310.

donde x_1 y x_3 son la posición (x) y la velocidad lineal (v) de la esfera, respectivamente, mientras x_2 y x_4 son la posición (Φ) y velocidad (ω) angular del motor. La entrada u es el torque del motor, m_b y m_v son la masa de la bola y la viga, respectivamente, I_b y I_v son los momentos de inercia de la bola y la viga, L_v es la longitud de la viga, g es la gravedad y R es el radio de la esfera.

Ahora, se ha modificado el modelo de tal forma que la entrada de control sea velocidad angular (ω) puesto que a los actuadores de V-REP (articulaciones) solo se les puede asignar velocidad o posición angular. Dicho lo anterior el modelo matemático de un sistema bola viga teniendo como entrada la velocidad angular tiene el siguiente espacio de estados no lineal

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} x_3 \\ u \\ \dfrac{(m_b\, x_1 u^2 - m_b g \sin x_2)}{\left(m_b + \left(\frac{I_b}{R^2}\right)\right)} \end{bmatrix}, \tag{2}$$

donde x_1 y x_3 son la posición y la velocidad lineal de la esfera, respectivamente, mientras x_2 es la posición angular del motor y u es la velocidad del motor.

Después, se linealizó el sistema en el punto de operación (0,0,0), obteniendo el siguiente espacio de estado continuo

$$\dot{x} = \overbrace{\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ m_b u^2 & \dfrac{-g m_b}{\left(m_b + \left(\frac{I_b}{R^2}\right)\right)} & 0 \end{bmatrix}}^{A} x + \overbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}}^{B} u \tag{3}$$

Luego, se aplicó la configuración mostrada en el menú “formas” para cambiar las masas de los objetos y demás parámetros del sistema que se ilustra en la Tabla 1. Para hallar los momentos de inercia de la bola I_b y de la viga I_v se han usado las siguientes expresiones

$$I_b = \frac{2}{5}\, m_v\, R^2 \qquad\qquad I_v = \frac{1}{12}\, m_v\, {L_v}^2. \tag{4}$$

Tabla 1. Parámetros del sistema bola viga

Parámetro	Unidad de Medida	Valor
$\boldsymbol{I_b}$	$[\mathrm{K_g/m^2}]$	$4.6240*10^{-5}$
$\boldsymbol{I_v}$	$[\mathrm{K_g/m^2}]$	0.08334
$\boldsymbol{m_b}$	$[\mathrm{K_g}]$	0.1
$\boldsymbol{m_v}$	$[\mathrm{K_g}]$	1
$\boldsymbol{L_v}$	[m]	1
$\boldsymbol{R}$	[m]	0.0340
$\boldsymbol{g}$	$[\mathrm{m/s^2}]$	9.8

Fuente: Esta Investigación.

2.1.3. Controladores

Para esta planta se ha diseñado un controlador LQR discreto con integrador, un Control Predictivo basado en modelo (MPC) y un observador de estados discreto.

Discretización del sistema

El sistema bola viga en el punto de operación (0,0,0) se describe por las matrices que se mostraron en (3), las cuales al reemplazar los valores de la Tabla 1 y suponiendo que solamente se cuenta con las mediciones de la posición de la bola en el riel (x_1) y la posición angular del motor (x_2) se obtuvo el siguiente espacio de estados continuo e invariante en el tiempo:

$$\dot{x} = \overbrace{\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & -7.0071 & 0 \end{bmatrix}}^{A} x + \overbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}}^{B} u, \qquad y = \overbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}^{C} x + \overbrace{[0]}^{D} u. \tag{5}$$

Ahora, para discretizar el sistema es necesario elegir un periodo de muestreo (T) adecuado para que el sistema sea estable, por lo que se seleccionó para esta planta un T=60ms, aclarando que este tiempo puede cambiar según las condiciones de simulación que se les proporcionen a los programas. Una vez se tiene el periodo de muestreo, se discretizó el sistema. Las matrices que describen el sistema en tiempo discreto con un periodo de muestreo T y un retenedor de orden cero (ZOH) son

$$x(k+1) = \overbrace{\begin{bmatrix} 1 & -0.0126 & 0.0600 \\ 0 & 1 & 0 \\ 0 & -0.4204 & 1 \end{bmatrix}}^{G} x(k) + \overbrace{\begin{bmatrix} -0.003 \\ 0.0600 \\ -0.0126 \end{bmatrix}}^{H} u(k), \quad (6)$$

$$y(k) = \overbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}^{C} x(k) + \overbrace{[0]}^{D} u(k)$$

Controlabilidad del sistema

Antes de comenzar con el diseño del controlador, se comprueba que el sistema es controlable. Para que un sistema de orden n sea controlable es necesario que el rango de la matriz de controlabilidad sea n, donde dicha matriz está dada por

$$CO = \begin{bmatrix} H & GH & G^2H & \cdots & \boldsymbol{G^{n-1}H} \end{bmatrix}. \quad (7)$$

Para el sistema bola viga en tiempo discreto la matriz de controlabilidad tiene rango tres, por lo tanto, el sistema es controlable.

LQR Discreto

Un controlador LQR (Regulador Cuadrático Lineal) discreto con horizonte infinito se caracteriza por ser una estrategia óptima, robusta y sencilla de implementar con realimentación de estados. Su operación se basa en minimizar la función:

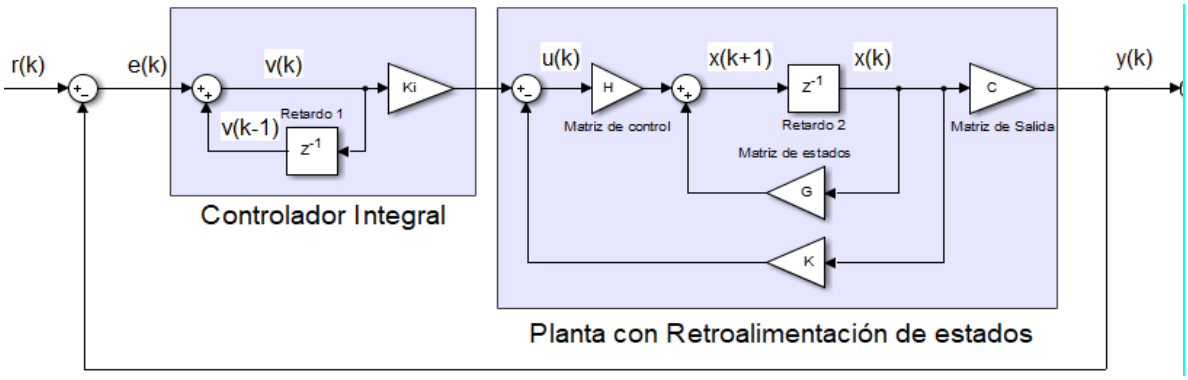
$$J = \sum_{k=0}^{\infty} x_k^T Q\, x_k + u_k^T R u_k \quad (8)$$

donde Q es una matriz positiva definida que define las prioridades en los estados, mientras que R, también positiva definida, establece la importancia de la señal de control (energía gastada) para obtener el objetivo de optimización. Para evitar errores de estado estacionario, es necesario agregar un integrador por cada una de las salidas (Figura 5). Con el fin de lograr lo anterior se ha seguido las recomendaciones expuestas en el libro Sistemas de control en tiempo discreto[24] y se ha utilizado las matrices de estados discretas ampliadas definidas como

$$GA = \begin{bmatrix} G & 0 \\ CG & 1 \end{bmatrix} \qquad HA = \begin{bmatrix} H \\ CH \end{bmatrix} \quad (9)$$

[24] OGATA.K. Sistemas de control en tiempo discreto. Segunda edición. Prentice Hall, 1995, p. 506-609.

Figura 5. Estructura de un controlador LQR discreto con integrador.



Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz.

Con base en pruebas realizadas en la simulación (usando el modelo matemático y la API de V-REP) para satisfacer un bajo tiempo de establecimiento y sobrepaso, se cambiaron varias veces las matrices de diseño (Q y R) hasta que se encontró las siguientes matrices

$$Q = \begin{bmatrix} 300 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 800 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}, \qquad R = 10. \tag{10}$$

Este diseño ha arrojado el siguiente vector de realimentación de estados, el cual se ha obtenido mediante el uso del comando “dlqr” del programa Matlab:

$$K = [-5.6072 \quad 10.0856 \quad -7.4063] \qquad K_i = -0.0698, \tag{11}$$

con los siguientes polos de lazo cerrado en el plano z

$$polos = [0.6756 + 0.2373i \quad 0.6756 - 0.2373i \quad 0.9730 \quad 0.9758]. \tag{12}$$

Control Predictivo basado en Modelo (MPC)

El control predictivo basado en modelos (MPC) es una estrategia de control que utiliza de forma explícita un modelo matemático interno del proceso a controlar (modelo de predicción). Este modelo se utiliza para predecir la evolución de las variables a controlar a lo largo de un horizonte temporal; de este modo se pueden calcular las variables manipuladas futuras para lograr que, en el horizonte de

predicción, las variables controladas converjan a los valores de referencia. Una de las propiedades más atractivas del MPC es su formulación abierta que permite la incorporación de distintos tipos de modelos de predicción, así como la consideración de restricciones sobre las señales del sistema; como se expone en el libro *Model Predictive control system design and implementation using MATLAB®*[25].

En este caso se ha formulado un MPC en variables de estado para la planta discreta bola-viga la cual es descrita por

$$\begin{aligned} x_m(k+1) &= A_m x_m(k) + B_m u(k), \\ y(k) &= C_m x_m(k), \end{aligned} \tag{13}$$

donde $u(k)$ es el vector de entrada o variable manipulada (velocidad del motor) de dimensiones $m\ x\ 1$ con $m = 1$, $y(k)$ es el vector de salida (posición de la bola) de dimensión $q\ x\ 1$ con $q = 1$, $x_m(k)$ es el vector de estados de dimensión $n_1\ x\ 1$ con $n_1 = 3$, A_m y B_m equivalen a las matrices discretas G y H encontradas anteriormente y $C_m = [1 \quad 0 \quad 0]$.

Ahora el modelo en espacio de estados se ha acondicionado para que cuente con un integrador, de esta forma el sistema no presentará error de estado estacionario, por lo que se ha seguido las recomendaciones de Wang[25] obteniendo la siguiente representación de espacio de estados

$$\begin{aligned} \overbrace{\begin{bmatrix} \Delta x_m(k+1) \\ y(k+1) \end{bmatrix}}^{x(k+1)} &= \overbrace{\begin{bmatrix} A_m & o_m^T \\ C_m A_m & I_{qxq} \end{bmatrix}}^{A} \overbrace{\begin{bmatrix} \Delta x_m(k) \\ y(k) \end{bmatrix}}^{x(k)} + \overbrace{\begin{bmatrix} B_m \\ C_m B_m \end{bmatrix}}^{B} \Delta u(k) \\ y(k) &= \overbrace{[o_m \quad I_{qxq}]}^{C} \overbrace{\begin{bmatrix} \Delta x_m(k) \\ y(k) \end{bmatrix}}^{x(k)}, \end{aligned} \tag{14}$$

donde $\Delta x_m(k)$ y $\Delta u(k)$ son variables incrementales ($\Delta x_m(k) = x_m(k) - x_m(k-1)$ y ($\Delta u(k) = u(k) - u(k-1)$) , o_m es una matriz de ceros de dimensiones $q\ x\ n_1$, I_{qxq} es la matriz identidad con dimensiones $q\ x\ q$, A_m es una matriz con dimensiones $n_1\ x\ n_1$, B_m con dimensiones $n_1\ x\ m$ y C_m con dimensiones $q\ x\ n_1$.

Una vez se ha agregado el integrador se procede a calcular la variable de control óptima para las variables predichas del sistema. En esta ventana de predicción se ha asumido que el tiempo actual es k_i y su longitud se encuentra determinada por N_p, es decir, el horizonte de predicción dado por el número de predicciones de la variable de salida. Adicionalmente, otro parámetro importante es el horizonte de

[25] WANG.L. Model predictive control system design and implementation using MATLAB®. Springer, 1995.

control (N_c) que define el número de señales de control a calcular para la trayectoria futura de control. Teniendo en cuenta todas las variables predichas se ha compactado las futuras variables de estado y de salida en la siguiente expresión

$$Y = Fx(k_i) + \Phi \Delta U, \tag{15}$$

donde

$$F = \begin{bmatrix} CA \\ CA^2 \\ CA^2 \\ \vdots \\ CA^{N_p} \end{bmatrix}, \Phi = \begin{bmatrix} CB & 0 & 0 & \dots & 0 \\ CAB & CB & 0 & \dots & 0 \\ CA^2B & CAB & CB & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ CA^{N_p-1}B & CA^{N_p-2}B & CA^{N_p-3}B & \dots & CA^{N_p-N_c}B \end{bmatrix},$$
$$Y = \begin{bmatrix} y(k_i+1|k_i) & y(k_i+2|k_i) & y(k_i+3|k_i) & \dots & y\left(k_i+N_p\middle|k_i\right) \end{bmatrix}^T, \tag{16}$$
$$\Delta U = [\Delta u(k_i) \quad \Delta u(k_i+1) \quad \Delta u(k_i+2) \quad \dots \quad \Delta u(k_i+N_c-1)]^T.$$

Teniendo en cuenta la señal de referencia $\mathrm{r}(k_i)$ y que el objetivo del horizonte de predicción es predecir la señal de salida para que el error de referencia y ΔU sea la menor posible, se tiene:

$$R_s^T = \overbrace{[1 \quad 1 \quad \cdots \quad 1]}^{N_p} r(k_i). \tag{17}$$

Ahora en el libro *Model Predictive control system design and implementation using MATLAB*®[25] se define la siguiente función de costo

$$\begin{gathered} J = (R_s - Y)^T (R_s - Y) - \Delta U^T \bar{R} \Delta U \\ J = \left(R_s - Fx(k_i)\right)^T \left(R_s - Fx(k_i)\right) - 2\Delta U^T \Phi^T \left(R_s - Fx(k_i)\right) \\ + \Delta U^T (\Phi^T \Phi + \bar{R}\,) \Delta U, \end{gathered} \tag{18}$$

donde $\bar{R}$ es un vector de dimensiones $r_w I_{N_c x N_c}$ siendo $r_w > 0$, que es usado como parámetro de ajuste en el rendimiento de lazo cerrado del sistema. Para encontrar el ΔU óptimo se minimiza la función de costo J, se obtiene la primera derivada de la función y la igualó a cero ($\frac{\partial J}{\partial \Delta U} = 0$), obteniendo

$$\Delta U = (\Phi^T \Phi + \bar{R}\,)^{-1} \Phi^T \left(R_s - Fx(k_i)\right). \tag{19}$$

Cabe destacar que la solución del problema no tiene restricciones sobre las variables de decisión. Sin embargo, en la planta virtual bola-viga se trabajó el problema de optimización con restricciones en la señal de control u_k (velocidad angular del motor), para lo cual se resolvió el problema de optimización cuadrático

$$\min_{\Delta U} \frac{1}{2}\Delta U^T \overbrace{(\Phi^T\Phi + \bar{R})}^{H}\Delta U - \Delta U^T \overbrace{\Phi^T\left(R_s - Fx(k_i)\right)}^{F_0} \; sujeto\ a\ \{a\,.\Delta U \leq b,$$
$$a = \begin{bmatrix} Triag \\ \cdots \\ -Triag \end{bmatrix}, b = \begin{bmatrix} 1u_{max} - 1u(k-1) \\ \cdots \\ 1u_{min} + 1u(k-1) \end{bmatrix}. \quad (20)$$

donde $Triag$ es una matriz triangular de dimensiones $N_c\, x\, N_c$, u_{min} y u_{max} son vectores con los valores mínimos y máximos, cuyas expresiones son:

$$u_{min} = u_{min}\overbrace{[1 \quad \cdots \quad 1]}^{N_c}, u_{max} = u_{max}\overbrace{[1 \quad \cdots \quad 1]}^{N_c}. \quad (21)$$

Finalmente, se puntualiza que los parámetros que se han usado en este controlador son: $N_c = 2$, $N_p = 40$, $\bar{R} = 1I_{N_c x N_c}$ y $[u_{min} \quad u_{max}] = [-3 \quad 3]$.

Observador discreto:

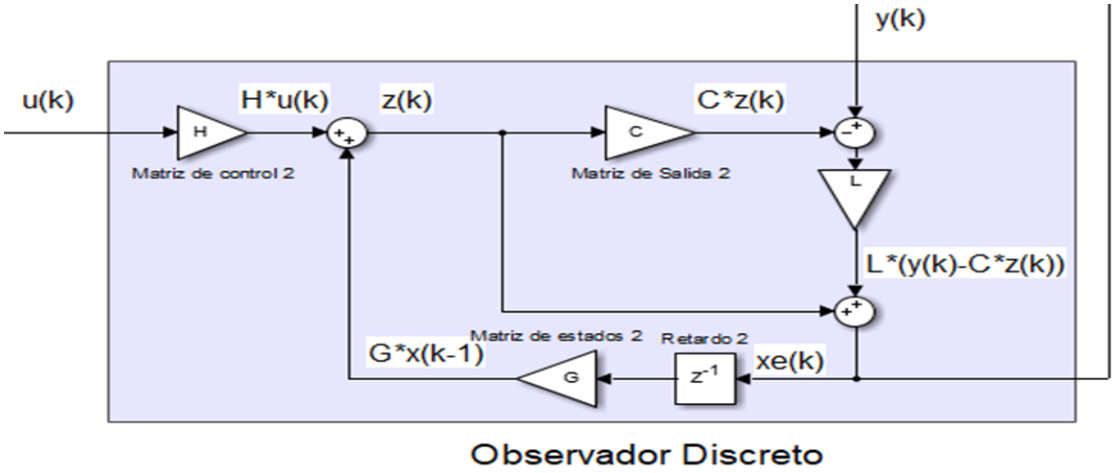
En un sistema dinámico puede darse el caso que no sea posible la medición directa de todas las variables de estado. En muchos casos prácticos solo son medibles unas cuantas variables de estado de un sistema dado. Por lo anterior es necesario estimar aquellas variables que no se pueden medir directamente, siendo pertinente el uso de observadores o estimadores de estado. En el caso específico del sistema bola-viga usualmente solo se pueden medir la posición de la bola (x_1) y el ángulo del motor (x_2), por lo que se ha necesitado la adición de un observador de estados al sistema (Figura 6). Conociendo la entrada, salida y dinámica del sistema, se ha podido estimar el valor de las variables que no es posible medir. La ecuación que describe la obtención de los estados observados es

$$z_k = G\hat{x}_k + Hu_k$$
$$\hat{x}_{k+1} = z_k + L(y_k - Cz_k), \quad (22)$$

Donde $\hat{x}_k$ son los estados observados, u_k es la señal de control (entrada del sistema) y L es la matriz de ganancias del observador.

También se enfatiza que para el sistema en cuestión se puede estimar las variables faltantes por medio de otros métodos como puede ser la aplicación de aproximaciones discretas de la derivada para hallar la velocidad lineal de la bola (x_3) y la velocidad angular del motor (x_4).

Figura 6. Diagrama general de un observador discreto lineal.



Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz.

Observabilidad del sistema lineal

Antes de continuar con el desarrollo del observador se comprueba si el sistema en cuestión es observable. Para que el sistema de orden n sea observable es necesario que el rango de la matriz de observabilidad sea igual n. Esta matriz tiene la siguiente expresión

$$CO = \begin{bmatrix} C & GC & G^2C & \cdots & \boldsymbol{G^{n-1}C} \end{bmatrix}. \qquad (23)$$

Para el sistema bola viga en tiempo discreto la matriz de observabilidad tiene un rango igual a tres por lo tanto el sistema es observable.

Matriz de ganancia del observador

La matriz de ganancias del observador L debe ser tal que el polinomio característico del sistema con observador sea igual a un polinomio deseado, que se ha determinado a partir de los polos discretos deseados en lazo cerrado para el sistema. Para que el observador tenga datos precisos lo más rápido posible, sus polos deben ser mínimo 5 veces más rápidos que los polos del controlador (LQR o MPC). Sin embargo, si se eligen polos muy rápidos las estimaciones serán muy sensibles al ruido y poco robusto a cambios de parámetros. En este caso los polos que se han usado para el LQR y MPC son

$$polosobsLQR = [\,0.7866 \quad 0.1466 \quad 0.0907],$$
$$polosobsMPC = [\,0.5488 \quad 0.0082 \quad 0.0025]. \tag{24}$$

De acuerdo con el observador elegido, el polinomio característico está dado por

$$|zI - (G - LCG)|, \tag{25}$$

asumiendo como polinomio característico deseado

$$P_{obs}(z) = (z - \mu_1)(z - \mu_2) \dots (z - \mu_n), \tag{26}$$

donde μ_i son los polos deseados en tiempo discreto y, para el caso en específico, los polos se mostraron en (24).

El cálculo de la matriz L se hizo desarrollando el polinomio característico del observador e igualando al polinomio deseado (función place de Matlab).En este caso las matrices de ganancias de los observadores para el LQR y MPC son:

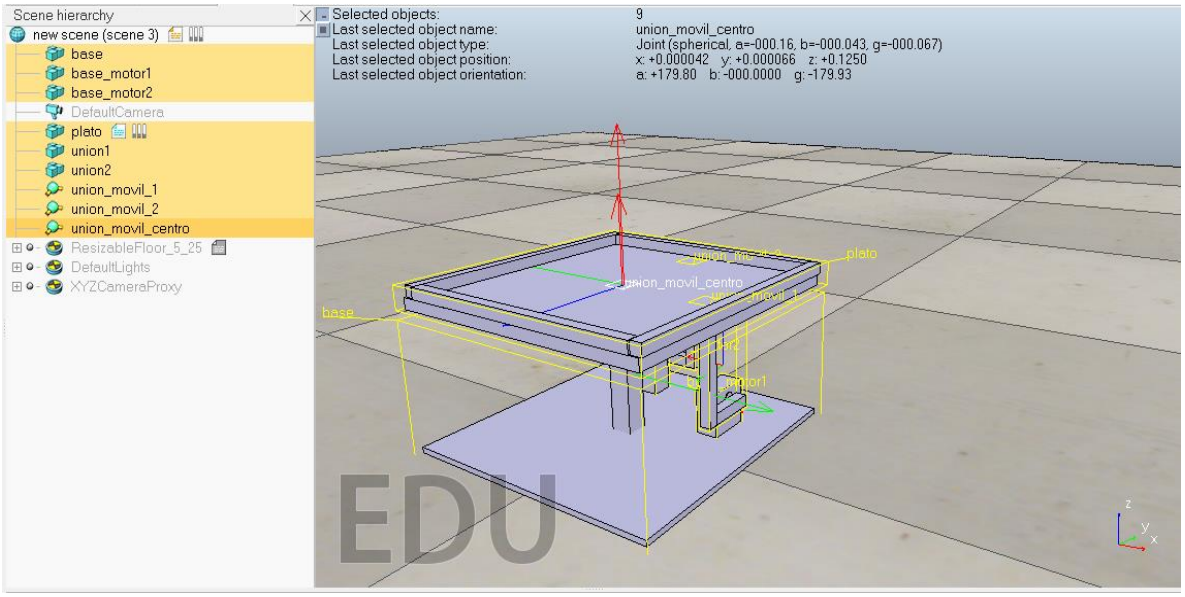
$$L = \begin{bmatrix} 0.8847 & 0 & 3.0348 \\ -0.0015 & 0.9093 & -0.3822 \end{bmatrix}^T,$$
$$LMPC = \begin{bmatrix} 0.9955 & 0 & 7.4579 \\ -0.0001 & 0.9975 & -0.3264 \end{bmatrix}^T. \tag{27}$$

2.2. SISTEMA BOLA PLATO

2.2.1. Desarrollo mecánico

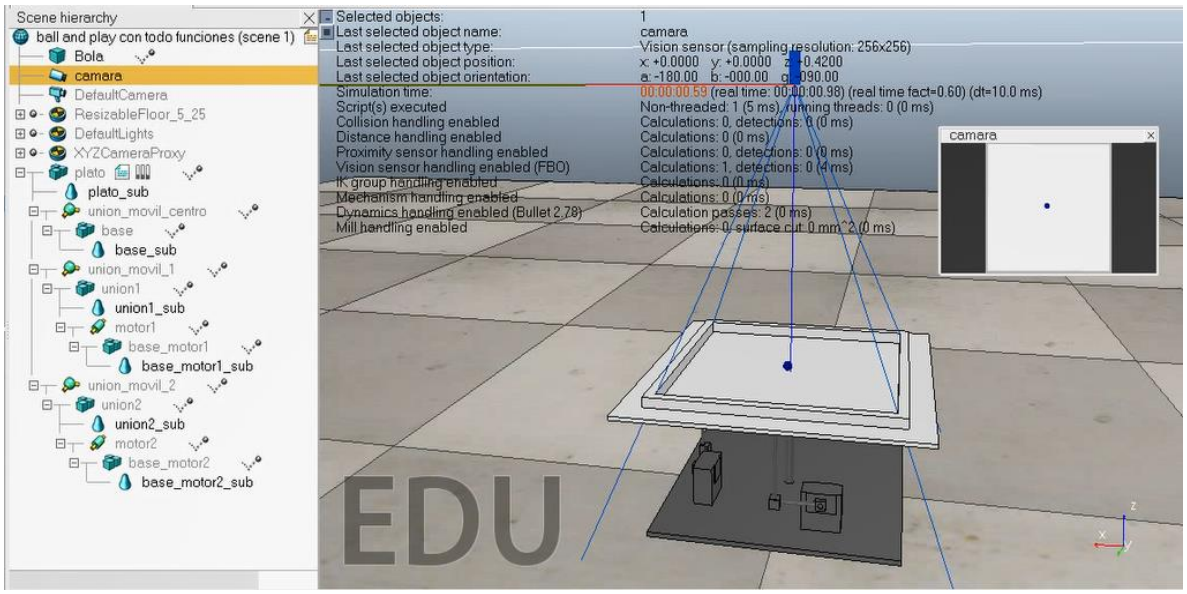
De manera similar al ejemplo anterior, para el sistema bola-plato se implementó una base de la planta con formas básicas (Figura 7) y un esquema pulido con formas importadas por CAD (Figura 8). En los modelos también se ha tenido en cuenta la agrupación de las partes importadas y la configuración de formas puras con el idioma nativo en V-REP.

Figura 7. Base estructural Sistema Bola Plato.



Fuente: Esta investigación, diseño realizado en V-REP por Harold Fernando Ruiz.

Figura 8. Modelo final del sistema Bola Plato.



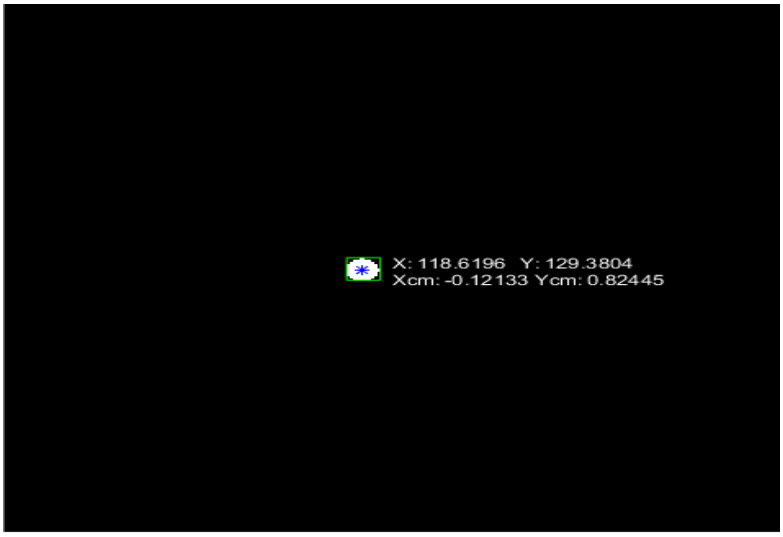
Fuente: Esta investigación, diseño realizado en V-REP por Harold Fernando Ruiz.

En cuanto a los actuadores, la planta tiene dos motores que mueven dos brazos unidos a la placa con juntas rotativas ubicadas a la misma distancia axial. Se supuso que los actuadores son simétricos para el movimiento en cada uno de los ejes, como se ilustró en la Figura 8. Las juntas se han configurado en el modo de torque/fuerza, mientras que los motores se pusieron en el modo de posición angular para habilitar la función servo (es decir, la entrada de cada motor es la posición angular). Para capturar la posición de la pelota con un sensor real, se ha usado una cámara fija, cuya imagen se exporta a Matlab como un mapa de bits que se ha procesado para estimar la posición de la pelota en la placa. Otra forma de obtener la posición de la pelota es consultando esa propiedad directamente desde la API (Anexo 1).

Al usar la cámara para obtener la posición de la esfera se realizó el procesamiento de imágenes explicado a continuación. Es de resaltar que el proceso para obtener la posición de la esfera debe implementarse de tal manera que todo el algoritmo (incluido el cálculo de control) se ejecute en un tiempo menor que el tiempo de muestreo seleccionado. Para este propósito, se ha establecido las características apropiadas de la cámara en V-REP, como la resolución, el ángulo de perspectiva y la región de interés. El color del sistema se fijó de tal manera que el contraste entre la placa (blanco) y la bola (azul oscuro) es alto, lo que facilita la implementación de algoritmos de procesamiento de imágenes.

La imagen inicial recibida en Matlab a través de la API es una imagen RGB con un tamaño de 256x256 píxeles, que corresponde al tamaño interno de la placa (22.5x22.5 cm). Para procesar la imagen se transformó a escala de grises, binariza e invierte. Según el cambio de valores en los píxeles adyacentes a la esfera, se reconoció el centroide de la misma. Además, los píxeles del centroide de la imagen (Figura 9) se cambiaron a distancias en centímetros tomando como referencia la escala utilizada por la API.

Figura 9. Imagen resultante del procesamiento de imágenes.



Fuente: Esta investigación, imagen obtenida en Matlab por Harold Fernando Ruiz.

2.2.2. Modelo Matemático

Asumiendo simetría en el sistema, además que la bola no se desliza y que está siempre en contacto con la superficie, un modelo no lineal que describe el comportamiento de la planta es sugerido por Cedeño[26], el cual es

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} x_2 \\ a\left(x_1 \dot{U_x^2} + x_3 \dot{U_x} \dot{U_y} - g\ sin(U_x)\right) \\ x_4 \\ a\left(x_3 \dot{U_y^2} + x_1 \dot{U_x} \dot{U_y} - g\ sin(U_y)\right) \end{bmatrix}, \tag{28}$$

donde x_1 y x_2 son la posición y velocidad a lo largo del eje x, respectivamente, mientras que x_3 y x_4 son las mismas variables en el eje y. Las entradas U_x y U_y son las posiciones angulares de los ejes de los servomotores, mientras que el valor de a=5/7, para este caso y tomando como referencia a Cedeño[26], reúne las características físicas del plato (dimensiones, peso, inercias), y g es la gravedad.

Para el diseño de los controladores, se linealizó el sistema alrededor del punto (0,0,0,0), resultando en un sistema definido por las matrices

[26]CEDEÑO, A; GORDÓN, M; MORALES, L. Control de posición y seguimiento de caminos en el sistema Bola-Plataforma. En: XXVII Jornadas de Automática, 2017, vol.27, no 5, p.47-53.

$$\dot{x} = \overbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}^{A} x + \overbrace{\begin{bmatrix} 0 & 0 \\ -ag & 0 \\ 0 & 0 \\ 0 & -ag \end{bmatrix}}^{B} u. \tag{29}$$

2.2.3. Controladores

Para esta planta se elaboró un controlador LQR discreto con integrador suponiendo la planta como un sistema MIMO. Adicionalmente se diseñó un PID discreto (suponiendo dos plantas SISO independientes) y un observador de estados discreto para los controladores ya mencionados.

Discretización del sistema (MIMO)

El sistema bola plato en el punto de operación (0,0,0,0) se describió en (29), que al sustituir g=9.8 m/s² y suponiendo que solamente se cuenta con las mediciones de la posición de la bola (x_1, x_3) en el plano (x,y), resultaron las siguientes matrices de estado en tiempo continuo

$$\begin{aligned} \dot{x} &= \overbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}^{A} x + \overbrace{\begin{bmatrix} 0 & 0 \\ -7.0071 & 0 \\ 0 & 0 \\ 0 & -7.0071 \end{bmatrix}}^{B} u, \\ y &= \overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}^{C} x + \overbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}}^{D} u. \end{aligned} \tag{30}$$

Ahora, al igual que en el sistema bola viga, se obtuvo las matrices en tiempo discreto resaltando que T=100 ms, resultando

$$\begin{aligned} x(k+1) &= \overbrace{\begin{bmatrix} 1 & 0.1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}}^{G} x(k) + \overbrace{\begin{bmatrix} -0.0350 & 0 \\ -0.7007 & 0 \\ 0 & -0.0350 \\ 0 & -0.7007 \end{bmatrix}}^{H} u(k), \\ y(k) &= \overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}^{C} x(k) + \overbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}}^{D} u(k). \end{aligned} \tag{31}$$

El sistema resultante es controlable y observable, después de aplicar el análisis controlabilidad y observabilidad, encontrando que el rango de dichas matrices corresponde con el número de estados (cuatro).

LQR Discreto (Sistema MIMO)

El controlador LQR se diseñó de la misma forma que en el sistema bola viga, con un esquema de control como el mostrado en la Figura 5. En este sistema, al tener dos referencias de posición (x e y), se incluyen dos integradores, aumentando el orden del sistema en dos. Las matrices ampliadas son entonces

$$GA = \begin{bmatrix} G & 0 & 0 \\ CG & 1 & 0 \\ & 0 & 1 \end{bmatrix} \qquad HA = \begin{bmatrix} H \\ CH \end{bmatrix}. \tag{32}$$

Después de hacer diversas pruebas en la simulación y en la Planta virtual se han sintonizado experimentalmente las matrices Q y R adecuadas definidas como

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 500 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 500 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix}, \qquad R = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \tag{33}$$

Este diseño arrojó las siguientes matrices de realimentación de estados

$$K = \begin{bmatrix} -0.5779 & -1.1352 & 0 & 0 \\ 0 & 0 & -0.5779 & -1.1352 \end{bmatrix}$$
$$K_i = \begin{bmatrix} -0.0150 & 0 \\ 0 & -0.0150 \end{bmatrix}, \tag{34}$$

cuyos polos de lazo cerrado son

$$[0.24 \quad 0.24 \quad 0.97 + 0.03i \quad 0.97 - 0.03i \quad 0.97 + 0.03i \quad 0.97 - 0.03i]. \tag{35}$$

PID Discreto (Sistema SISO)

Gracias a la simetría del sistema, para la planta bola-plato se pudo modelar las dinámicas con dos plantas SISO independientes. De esta manera, para el control se diseñó un controlador PID discreto mediante dos lazos independientes (uno para cada motor). La calibración se ha realizado para un solo subsistema y se replica para el segundo actuador dada la simetría asumida en la planta.

La estructura del control continuo estándar se muestra en el libro Sistemas de control en tiempo discreto[24] y se define como

$$m(t) = K\left[e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d\frac{de(t)}{dt}\right], \tag{36}$$

donde K, T_i y T_d son las constantes continuas proporcional, integral y derivativa respectivamente.

Entonces, se discretizó (36) mediante aproximaciones de la integral y derivada, y se halló la transformada Z, resultando en la siguiente función de transferencia pulso definida por

$$U(z) = \left[k_P + \frac{k_I}{1 - z^{-1}} + k_D(1 - z^{-1})\right] E(z), \tag{37}$$

donde k_P , k_I y k_D son las constantes discretas proporcional, integral y derivativa respectivamente, $U(z)$ es la señal de control y $E(z)$ el error de la salida frente a la referencia deseada.

En este caso se calibró el controlador en tiempo continuo, por lo que para su implementación es necesario hallar una relación entre las constantes continuas y discretas. Esta relación se tomó del libro Sistemas de control en tiempo discreto [25], resultando las siguientes expresiones

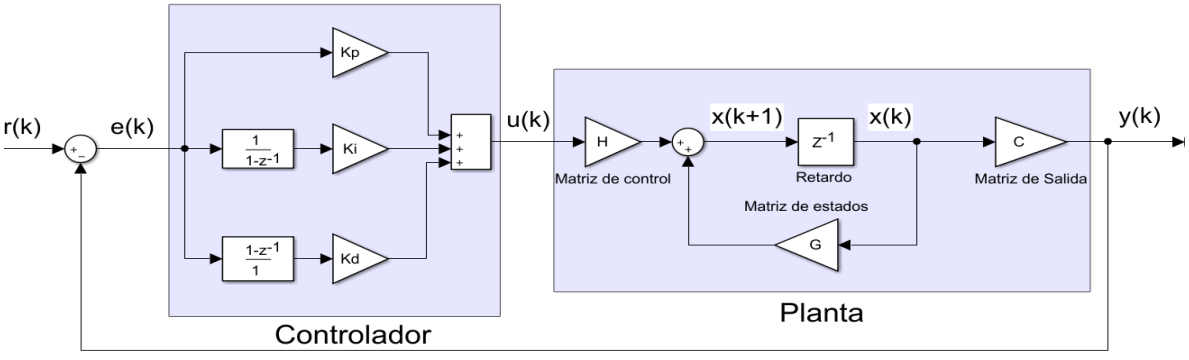
$$k_I = \frac{KT}{T_i} \qquad k_p = K - \frac{k_I}{2} \qquad k_D = \frac{K\,T_d}{T}, \tag{38}$$

cuyos resultados para este caso en particular se muestran a continuación:

$$k_I = -0.0029 \qquad k_p = -0.4626 \qquad k_D = -5.3064. \tag{39}$$

En el esquema de la Figura 10 se presenta un diagrama de la técnica de control PID.

Figura 10. Estructura de un controlador PID discreto.



Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz.

Observador discreto

En un sistema bola-plato real es sencillo medir la posición de la bola (x_1, x_3) en el plano (x,y). Debido a lo anterior se elaboró un observador de estados de orden completo (Figura 6). Para construir el observador se fijó los polos discretos y se encontró la matriz de ganancias del observador como se mostró con anterioridad en el sistema bola viga. Los polos deseados resultantes son

$$polosobs = [\,0.6703 \quad 0.0408 \quad 0.0183 \quad 0.0003\,]. \tag{40}$$

Y la matriz de ganancia del observador es:

$$L = \begin{bmatrix} 0.9999 & 8.2023 & -0.0001 & -2.1249 \\ -0.0001 & -2.2609 & 0.9999 & 7.2860 \end{bmatrix}^T. \tag{41}$$

Cabe resaltar que al igual que en el sistema bola-viga se pueden encontrar las velocidades lineales (x_3 y x_4) mediante aproximaciones discretas de la derivada usando la posición (x, y) de la pelota.

2.3. PÉNDULO DE FURUTA

2.3.1. Desarrollo mecánico

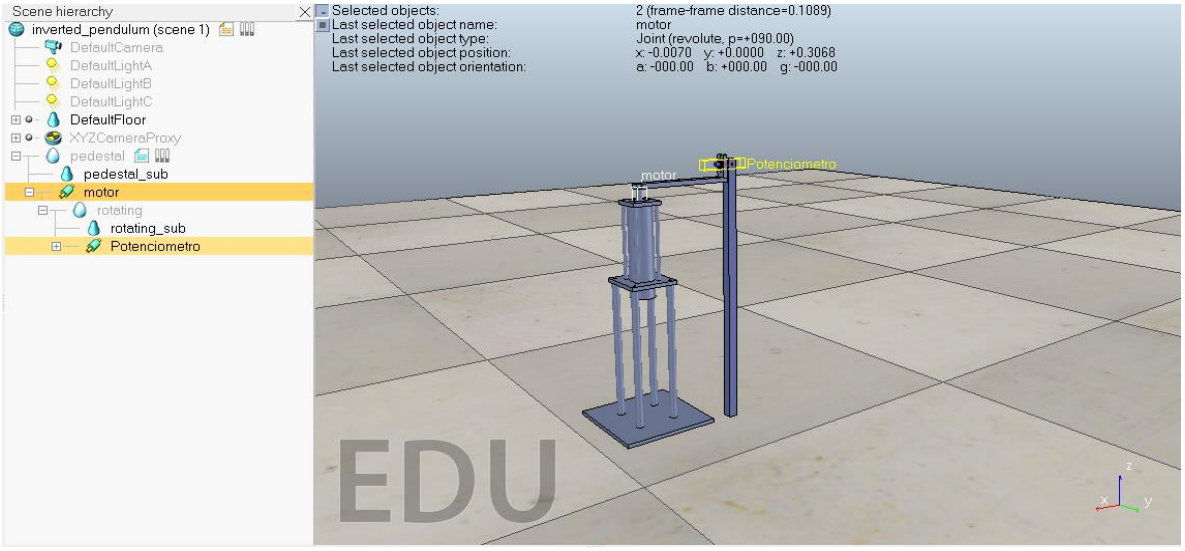
El modelo mecánico en V-REP de la planta en cuestión se encontró en el repositorio[27], por lo que en el trabajo simplemente se acopló los parámetros del modelo.

Con respecto a los sensores, la planta necesita dos articulaciones rotacionales en el modo “torque/fuerza”, las cuales permiten medir los ángulos Φ y θ desde la API (Matlab y Simulink). Una de las articulaciones se ubicó entre la base del péndulo y el brazo, y la otra en la base del brazo como se ilustra en la Figura 11.

En cuanto al actuador, la planta cuenta con un motor que se ha configurado en el modo de velocidad angular (la misma del sistema bola viga) y se ubicó en la base del brazo (Figura 11).

[27]RyogeiGoton. inverted_pendulum. {En linea}. {4 de octubre del 2018}. Disponible en: (https://github.com/RyoheiGoto/inverted_pendulum/tree/master/simulator/vrep).

Figura 11. Esquema final péndulo de Furuta.



Fuente: Diseño realizado en V-REP por RyogeiGoton.

2.3.2. Modelo Matemático

La obtención del modelo está basada en la teoría de Euler-Lagrange[28]. Primero se supuso que el momento de inercia del motor es muy bajo y se introdujo las siguientes variables de estado $x_1 = \Phi\ (\text{Posición angular del brazo}), x_2 = \dot{\Phi},\ x_3 = \theta\ (\text{Posición angular del pendulo})\ y\ x_4 = \dot{\theta}$ y τ es el torque del motor, resultando las siguientes ecuaciones

$$\alpha = \frac{1}{3}\left(m_a + m_p\right){l_a}^2 \qquad \beta = \frac{1}{3}\, m_p\, {l_p}^2$$

$$\gamma = \frac{1}{2}\, m_p\, l_a l_p \qquad \delta = \frac{1}{2}\, m_p\, g\, l_p$$

$$\dot{x_1} = x_2$$

$$\dot{x_2} = \frac{1}{(\alpha\beta - \gamma^2 + (\beta^2 + \gamma^2)\sin {x_3}^2)}[\beta\gamma\,(\sin {x_3}^2 - 1)\sin x_3 {x_2}^2 - 2\beta^2 \cos x_3 \sin x_3 x_2 x_4 + \beta\gamma \sin x_3 {x_4}^2 - \gamma\delta \cos x_3 \sin x_3 + \beta\,\tau] \tag{42}$$

$$\dot{x_3} = x_4$$

$$\dot{x_4} = \frac{1}{(\alpha\beta - \gamma^2 + (\beta^2 + \gamma^2)\sin {x_3}^2)}[\beta\gamma\,(\sin {x_3}^2 - 1)\sin x_3 {x_2}^2 - 2\beta^2 \cos x_3 \sin x_3 x_2 x_4 + \beta\gamma \sin x_3 {x_4}^2 - \gamma\delta \cos x_3 \sin x_3 + \beta\,\tau],$$

[28] OSORIO, C. Diseño, construcción y control de un péndulo invertido rotacional utilizando técnicas lineales y no lineales. Trabajo de grado (Master en automatización industrial). Bogotá, 2009. Universidad de Nacional de Colombia. Facultad de ingeniería. Departamento de Ingeniería de Sistemas.

donde g es la gravedad, m_a es la masa del brazo horizontal, l_a es la longitud del brazo horizontal, m_p es la masa del péndulo, l_p es la longitud del péndulo.

Ahora, linealizó el sistema alrededor del punto (0,0,0,0), obteniendo las siguientes matrices que describen el sistema

$$\dot{x} = \overbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-\delta\gamma}{\alpha\beta-\gamma^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{\alpha\delta}{\alpha\beta-\gamma^2} & 0 \end{bmatrix}}^{A} x + \overbrace{\begin{bmatrix} 0 \\ \frac{\beta}{\alpha\beta-\gamma^2} \\ 0 \\ \frac{-\gamma}{\alpha\beta-\gamma^2} \end{bmatrix}}^{B} u \tag{43}$$

Luego, se aplicó la configuración mostrada en “formas” (Anexo 1) para cambiar las masas de los objetos, asignando los parámetros que se muestran en la Tabla 2.

Tabla 2. Parámetros Péndulo de Furuta.

Parámetro	Unidad de Medida	Valor
$\boldsymbol{m_a}$	[K_g]	0.2
$\boldsymbol{m_p}$	[K_g]	0.6
$\boldsymbol{l_a}$	[m]	0.116
$\boldsymbol{l_p}$	[m]	0.3
$\boldsymbol{g}$	[m/s^2]	9.8

Fuente: Esta investigación.

2.3.3. Controladores

En esta planta virtual se realizó un controlador LQR y un observador de estados de orden completo, Adicionalmente se planteó un control por Gain Scheduling y una estrategia de balanceo inicial o Swing-up para posicionar el péndulo lo más cerca posible al origen desde su posición de reposo.

Puntos de equilibrio

Los puntos de equilibrio se obtienen cuando $\dot{x} \equiv 0$, es decir que la tasa de variación de x con respecto al tiempo es cero. Entonces de (42) se igualó las derivadas de las variables de estado a cero y se supuso que el péndulo trabaja en el intervalo $[-\pi, \pi]$ resultando la siguiente expresión:

$$\begin{bmatrix}0\\0\\0\\0\end{bmatrix} = \begin{bmatrix} x_2 \\ -\beta\gamma \sin x_3 - \gamma\delta \cos x_3 \sin x_3 + \beta\ \tau \\ x_4 \\ (\alpha + \beta \sin {x_3}^2) - \gamma \cos x_3\ \tau \end{bmatrix}. \tag{44}$$

Ahora se despejó τ de la fila dos de la ecuación (44), obteniendo los siguientes puntos de equilibrio

$$x_{1ss} = x_1 \quad x_{2ss} = 0 \quad x_{3ss} = x_3 \quad x_{4ss} = 0 \qquad \tau_{ss} = \frac{3 g l_a m_p \sin 2x_3}{8}. \tag{45}$$

Es clave mencionar que los puntos de equilibrio hallados anteriormente permitirán diseñar un controlador no lineal de ganancia programable (Gain Scheduling), explicado más adelante.

Discretización del sistema

Al igual que en los sistemas anteriores el primer paso para discretizar el sistema es reemplazar los parámetros de la planta que se mostraron en la Tabla 2 en (42) y (43). Adicionalmente, se supuso que solamente se cuenta con las mediciones de los estados x_1 y x_3 obteniendo las siguientes matrices de estado en tiempo continuo

$$\dot{x} = \overbrace{\begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & -108.6207 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 111.9963 & 0\end{bmatrix}}^{A} x + \overbrace{\begin{bmatrix}0\\ 636.9968\\ 0\\ -369.4458\end{bmatrix}}^{B} u, \tag{46}$$

$$y = \overbrace{\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}}^{C} x + \overbrace{[0]}^{D} u.$$

Ahora, las matrices en tiempo discreto con T=30ms, resultan en

$$x(k+1) = \overbrace{\begin{bmatrix} 1 & 0.0300 & -0.0493 & -0.0005 \\ 0 & 1 & -3.3136 & -0.0493 \\ 0 & 0 & 1.0508 & 0.0305 \\ 0 & 0 & 3.4166 & 1.0508 \end{bmatrix}}^{G} x(k) + \overbrace{\begin{bmatrix} 0.2880 \\ 19.2914 \\ -0.1677 \\ -11.2705 \end{bmatrix}}^{H} u(k), \quad (47)$$

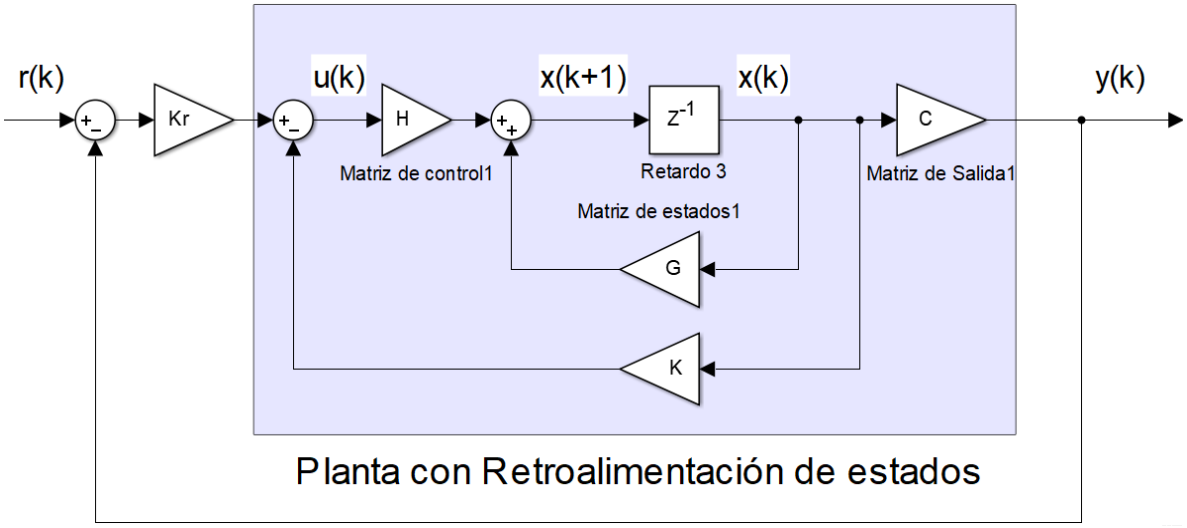
$$y(k) = \overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}^{C} x(k) + \overbrace{[0]}^{D} u(k)$$

Una vez se definió las matrices discretas, se examina la controlabilidad y observabilidad del sistema como se ha hecho con anterioridad, encontrando que este sistema es también controlable y observable.

Controlador LQR

El controlador LQR se diseñó de la misma forma que en el sistema bola viga con la diferencia que no se adicionó un integrador, dado que este introduce inestabilidad a la planta virtual y el sistema cuenta con dos integradores naturales. Por lo anterior el sistema tiene el esquema de control mostrado en la Figura 12.

Figura 12. Diagrama de bloques LQR sin integrador.



Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz.

Los parámetros Q y R se han encontrado realizando diferentes experimentos en la simulación y en la planta virtual resultando:

$$Q = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 60 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad R = 1, \tag{48}$$

con el siguiente vector de realimentación de estados

$$K_r = -0.0121 \qquad K = [-0.0457 \quad -1.4770 \quad -0.1886], \tag{49}$$

y los polos en lazo cerrado dados por

$$polos = [4.4780x10^{-5} \quad 0.9914 \quad 0.8111 \quad 0.8108]. \tag{50}$$

Gain Scheduling

La técnica de ganancia programable (Gain scheduling) es un acercamiento al control de sistemas no lineales que utiliza una familia de controladores lineales, para proporcionar un control satisfactorio en diversos puntos de operación. En este caso se parametrizó la planta mediante la variable (x_3) ("scheduling variable"), de modo que se siguió las recomendaciones dadas por el libro *Adaptive control*[29], las cuales son:

- **Determinar las variables de programación:** mediante (45) se ha identificado que para el péndulo de Furuta la variable de programación es (x_3).

- **Obtener el modelo del proceso para diferentes puntos de operación:** En el caso en específico se linealizó (42) teniendo en cuenta los puntos de operación descritos en (45). Luego se halló las ecuaciones de estado lineales en tiempo continuo, reemplazando los puntos de operación en la planta linealizada, y después se discretizó las ecuaciones lineales continuas mediante un (ZOH) y un T=30 ms. Finalmente se repitió el proceso para valores de (x_3) en un rango de [-0.4, 0.4] rad con un paso de 0.01 rad. Así se tuvieron distintos sistemas definidos en este rango.

- **Calcular los parámetros del controlador para los diferentes puntos de operación:** se calculó un LQR discreto para cada ecuación de estados lineal discreta encontrada en el punto anterior y se halló las constantes de retroalimentación teniendo en cuenta que los parámetros de entrada (Q y R) son los mostrados en (48).

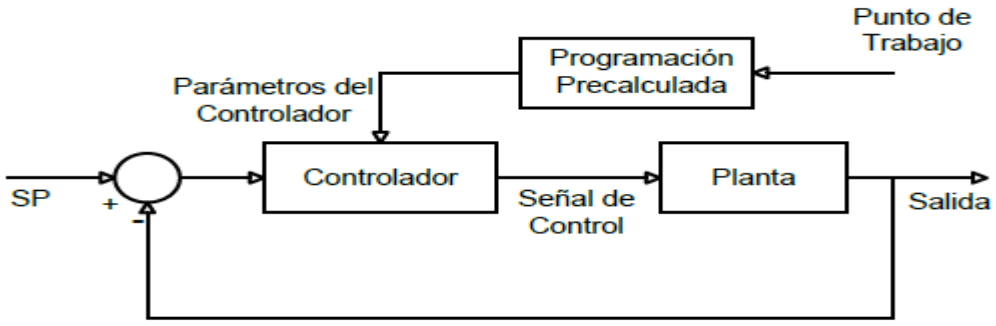
[29] ASTRÖM.K; WITTENMARK.B. Adaptive control. Second edition. Dover publications, 1995.

- **Seleccionar el controlador en función de las variables de programación:** Con las contantes de retroalimentación encontradas en cada punto de operación, se procedió a hacer una regresión que permite relacionar la variable de programación con las constantes del controlador resultando en las siguientes expresiones:

$$K_1 = -0.0577{x_3}^2 - 0.0121 \qquad K_2 = -0.2170{x_3}^2 - 0.0459$$
$$K_3 = -9.0990{x_3}^2 - 1.4690 \qquad K_4 = -1.0850{x_3}^2 - 0.1881 \tag{51}$$

Es importante resaltar que la prueba de controlabilidad del sistema se realizó en cada punto de operación para asegurar que es posible controlar el sistema. En este caso la planta en cuestión es controlable en todos los puntos de operación en el rango expuesto anteriormente. En la Figura 13, se presenta un diagrama básico de la técnica de control ganancia programable.

Figura 13. Diagrama de bloques Gain Sheduling.



Fuente: ASTRÖM.K; WITTENMARK.B. *Adaptive control*. *Second Edition*. Dover publications, 1995.

Transformación de torque a velocidad

En apartados anteriores se ha mencionado que a los actuadores de V-REP solamente se les puede asignar velocidad o posición angular. Sin embargo, los controladores que se han diseñado en esta planta tienen como entrada el torque, por lo que es necesario convertir la entrada torque en velocidad. Esta transformación se realizó por medio de una integración discreta, resaltando que en esta planta se despreció el momento de inercia generado por el péndulo. La ecuación que relaciona la velocidad y el torque es

$$w_k = \frac{3\tau\, T}{m_a\, {l_a}^2} + w_{k-1}, \tag{52}$$

donde w_k y w_{k-1} son la velocidad angular del motor en el instante de tiempo actual y anterior respectivamente, T es el tiempo de muestreo y τ es el torque del motor.

Swing up

Para este controlador se ha usado el modelo simplificado sugerido por Osorio[28]. El modelo anterior desprecia los torques de reacción ejercidos desde el péndulo hacia el brazo, y de esta forma, el control de energía del péndulo se estudia considerando la posición (x_3) y velocidad (x_4) del péndulo.

De lo anteriormente expuesto y siguiendo los pasos que ha recomendado Osorio[28] es posible definir la función de energía del péndulo dada por

$$E = \frac{1}{2}J_p x_4^2 + \frac{1}{2}m_p g l_p (cos x_3 - 1), \tag{53}$$

donde E es la energía del péndulo y J_p el momento de inercia del péndulo.

Luego, se ha escogido una función de Lyapunov de la forma

$$V = \tfrac{1}{2}(E - E_0)^2, \tag{54}$$

para hallar $\dot{V}$ se siguió las recomendaciones de Osorio[28], obteniendo la siguiente expresión

$$\dot{V} = -\tfrac{1}{2}m_p l_p u (E - E_0) x_4 cos x_3. \tag{55}$$

Ahora se escogió una función u igual a

$$u = k(E - E_0) x_4 cos x_3, \tag{56}$$

y se reemplazó (56) en (55), resultando en

$$\dot{V} = -\frac{1}{2}m_p l_p k[(E - E_0) x_4 cos x_3]^2. \tag{57}$$

Dado que es necesario que la magnitud de la energía cambie lo más rápido posible, se ha propuesto la siguiente señal de control

$$u = k(E - E_0) sgn(x_4 cos x_3), \tag{58}$$

donde u es la aceleracion angular del pendulo, k es una constante de calibración, E_0 es la energía a la que se desea que llegue el péndulo y sgn es la función signo.

Es importante resaltar que (57) sigue siendo una función semi negativa definida, lo que significa que el sistema es estable en el sentido de Lyapunov.

Una vez hecho lo anterior, se transformó la señal de control del swing up de aceleración en el punto del pivote del péndulo a velocidad angular del motor ($\dot{\Phi}$), por lo que se usó la siguiente relación:

$$\tau = \frac{\alpha}{l_a} u. \tag{59}$$

Siendo así, se expresó τ en función de la aceleración angular del motor obteniendo

$$\frac{m_a \, {l_a}^2}{3} \frac{d^2\Phi}{dt^2} = \frac{m_a \, {l_a}^2}{3 \, l_a} u. \tag{60}$$

Luego se integró la ecuación (60) para obtener $d\Phi/dt$ considerando que u no depende de Φ (modelo simplificado) resultando

$$\int \frac{d^2\Phi}{dt^2} \, d\Phi = \int \frac{1}{l_a} u \, d\Phi = \frac{1}{l_a} u\Phi + \mathrm{C} \quad (\Phi \neq 0), \tag{61}$$

Finalmente, de (61) se supuso que Φ siempre mantiene un valor constante ($\Phi = 1$), ya que Φ puede ser cualquier valor en el rango $(0\,, 2\pi)$. Adicionalmente se resalta que C es despreciable, obteniendo la ley de control para “Swing up” descrita por

$$\dot{\Phi} = \frac{2\pi}{l_a} k(E - E_0) sgn(x_4 cos x_3) \tag{62}$$

Las leyes de control que se han presentado en (58) y en (62) resultan en un swing up que hace que el péndulo pueda llegar a una posición adecuada para que cualesquiera de los controladores mostrados anteriormente puedan estabilizar el péndulo. Es de aclarar que una vez realizado el balanceo inicial, si el péndulo entra en una posición angular menor a los +- 18°, el controlador local (LQR o Ganancia programable) entra en funcionamiento. De lo contrario, sigue en activado el controlador Swing up.

Hasta ahora el control de energía planteado no presenta ningún inconveniente. Sin embargo, cuando se implementó esta técnica en el modelo de orden cuatro presentó

dificultades, principalmente debido a que los parámetros k y E_0 dependen del tiempo de muestreo y su calibración resultó bastante complicada. Teniendo en cuenta lo anterior se hizo experimentos a partir de las simulaciones y la planta virtual, donde se encontró que con los parámetros de la Tabla 2, una buena calibración para el swing up en la simulación (basado en modelo) es cuando $k = 0.010$ y $E_0 = 0.588$, mientras que para la planta virtual los parámetros más adecuados son $k = 0.005$ y $E_0 = 0.588$.

Observador de estados

En un péndulo de Furuta real es común medir solamente la posición angular del brazo (x_1) y la posición angular del péndulo (x_3). Debido a lo anterior se elaboró un observador de estados de orden completo (22), fijando polos discretos y se encontró la matriz de ganancias del observador como se mostró con anterioridad en el sistema "bola viga". Entonces, los polos deseados son

$$polosobs = [\ 0 \quad 0.0001 \quad 0.0001 \quad 0.0001\ ], \tag{63}$$

y la matriz de ganancia del observador es

$$L = \begin{bmatrix} 1 & 33.3 & 0 & 0 \\ 0 & -1.0796 & 1 & 34.4098 \end{bmatrix}^T. \tag{64}$$

En este caso también es posible usar las posiciones angulares del brazo x_1 y péndulo x_3 para hallar las velocidades angulares (x_2 y x_4) por medio de aproximaciones discretas de la derivada.

2.4. PLANEAMIENTO DE TRAYECTORIAS DE UN ROBOT TERRESTRE

2.4.1. Desarrollo mecánico

El modelo mecánico en V-REP del robot en cuestión ya se encuentra predeterminado en el simulador, por lo tanto, en este trabajo simplemente se acoplaron los parámetros del robot, presentados en la Tabla 3.

Tabla 3. Parámetros robot Pioneer_P3dx.

Parámetro	Descripción	Unidad de Medida	Valor
$\boldsymbol{M_t}$	Masa Total	$[\mathrm{K_g}]$	20.25
$\boldsymbol{M_r}$	Masa rueda derecha	$[\mathrm{K_g}]$	1.5
$\boldsymbol{M_l}$	Masa rueda izquierda	$[\mathrm{K_g}]$	1.5
$\boldsymbol{M_{rl}}$	Masa rueda loca	$[\mathrm{K_g}]$	1.25
$\boldsymbol{M_b}$	Masa de la base del robot	$[\mathrm{K_g}]$	16
$\boldsymbol{2\alpha}$	Distancia entre los centros de las ruedas	[m]	0.3
r	Radio de las ruedas	[m]	0.0975

Fuente: Esta investigación.

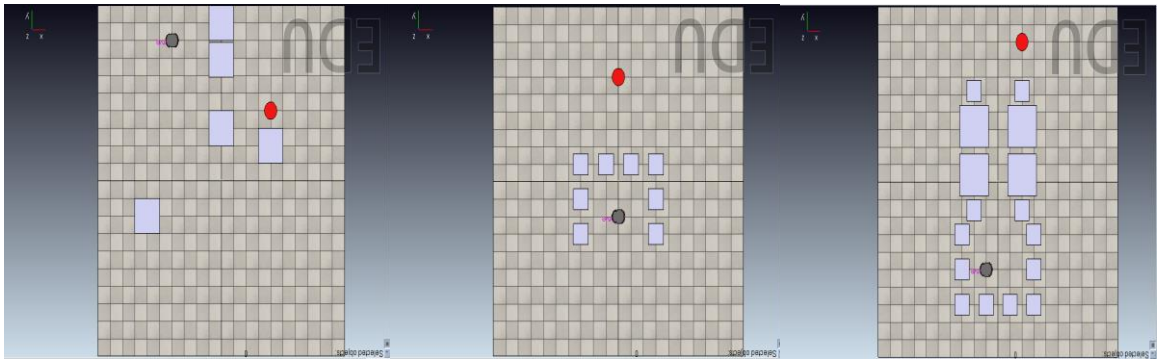
Para ilustrar el proceso de implementación de planeación de trayectorias de vehículos terrestre, se usó los escenarios expuestos por Getial[30]. Por lo tanto, se construyeron en V-REP seis escenarios, los cuales tienen como elementos más destacables el vehículo (Pioneer_p3dx), los obstáculos, el objetivo y el espacio de trabajo. En consecuencia, se divide los escenarios en dos tipos, cada uno con tres ejemplos, puesto que la construcción en el simulador y los métodos de planeamiento de trayectorias (Sección 2.4.3) son diferentes como se describe a continuación.

Escenarios tipo 1: Los obstáculos son cubos (formas primitivas), cuyo eje de referencia es el centro del cubo (configuración por defecto). El objetivo o punto deseado es un cilindro (forma primitiva) que también tiene como eje de referencia el centro del cilindro. Finalmente, el robot Pioneer incorporado en el simulador (características en Tabla 3). El espacio de trabajo tiene dimensiones de (10x10x0.04) m en coordenadas cartesianas (x, y, z).

En la Figura 14 se presenta todos los escenarios tipo uno.

[30]GETIAL, J; ERAZO, E; PANTOJA, A. Quantitative Comparison of Path Planning Methods in Mobile Robotics. En: Proceedings of the 3rd Colombian Conference on Automatic Control. CCAC, 2017.

Figura 14. Escenarios tipo uno, donde el robot se observa como el círculo gris y los puntos finales en rojo.

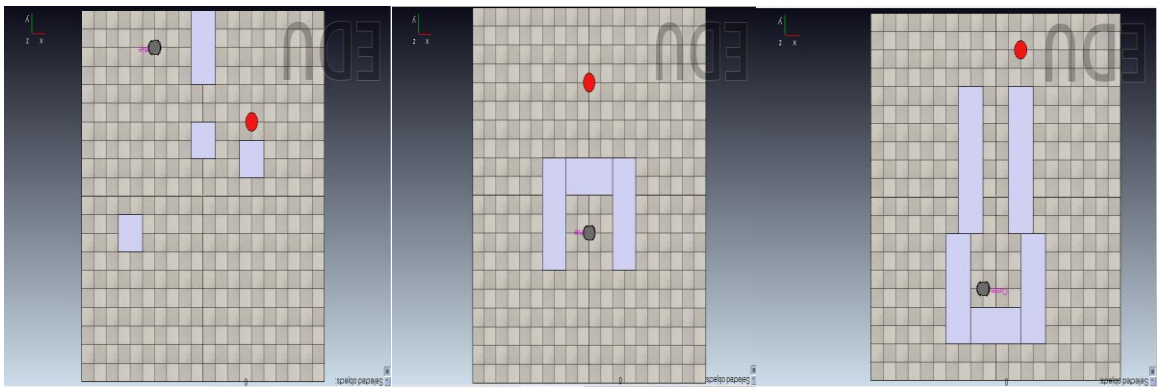


Fuente: Esta investigación, diseño realizado en V-REP por Harold Fernando Ruiz.

Escenarios tipo 2: la diferencia con los escenarios tipo uno es la cantidad de obstáculos y sus ejes de referencia. Para cambiar los ejes de referencia de los obstáculos se agregó “dummies” a la escena. Un dummy es un punto de referencia que facilitó el cambio de los ejes coordenados, para lo cual se ubicó el elemento donde se desea poner el nuevo eje coordenado del objeto y se lo posicionó en una jerarquía arriba. Entonces, se hizo que el obstáculo dependa del “dummy”. Lo anterior se realizó con el fin que a la hora de iniciar el algoritmo de planeamiento de trayectorias no sea necesario agregar las posiciones de los objetos en Matlab, sino que estas cambien con las modificaciones a la escena en V-REP.

En la Figura 15 se muestran todos los escenarios tipo dos.

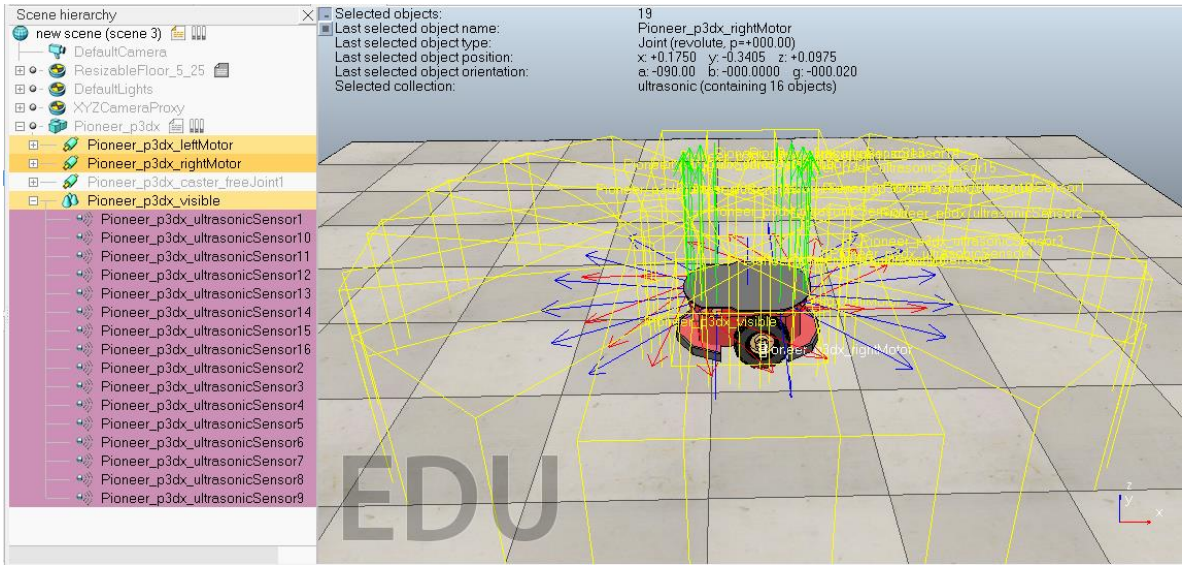
Figura 15. Escenarios tipo dos.



Fuente: Esta investigación, diseño realizado en V-REP por Harold Fernando Ruiz.

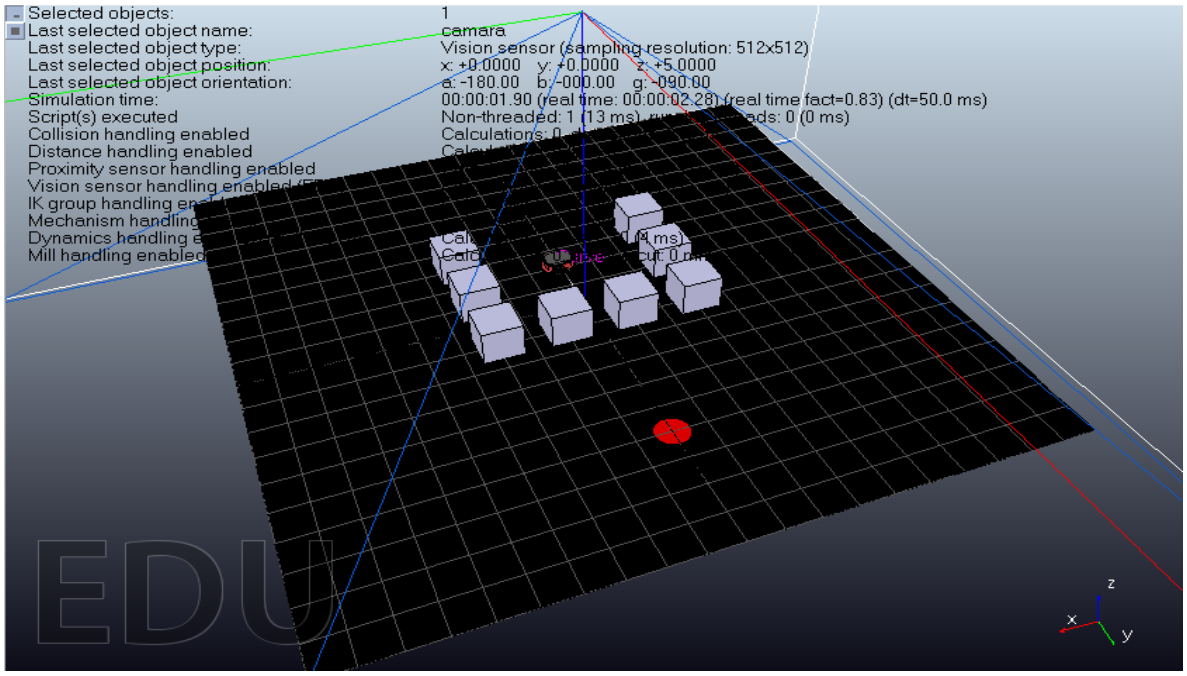
En el desarrollo del planeamiento de trayectorias se necesitaron varios sensores en el robot. Primero se midió la posición angular de la rueda izquierda θ_l y derecha θ_r mediante dos articulaciones rotacionales posicionadas en las ruedas (actuadores). La detención de obstáculos se realizó por medio de varios sensores de proximidad que ya vienen incorporados en el vehículo (Figura 16) o con un sensor de visión tipo cámara ubicado en la parte superior del escenario (Figura 17). Para el caso más simple, también se puede usar la API V-REP-Matlab, con la que también se midió la posición del vehículo en el plano (x, y) y el ángulo de giro ϕ. Los actuadores del sistema son las dos articulaciones rotacionales que se ubicaron transversalmente a cada rueda (Figura 16), que tienen la misma configuración que en el sistema bola viga.

Figura 16. Ubicación sensores de proximidad Pioneer_P3dx.



Fuente: Esta investigación, diseño realizado en V-REP por Harold Fernando Ruiz.

Figura 17. Ubicación del sensor de visión en el escenario.

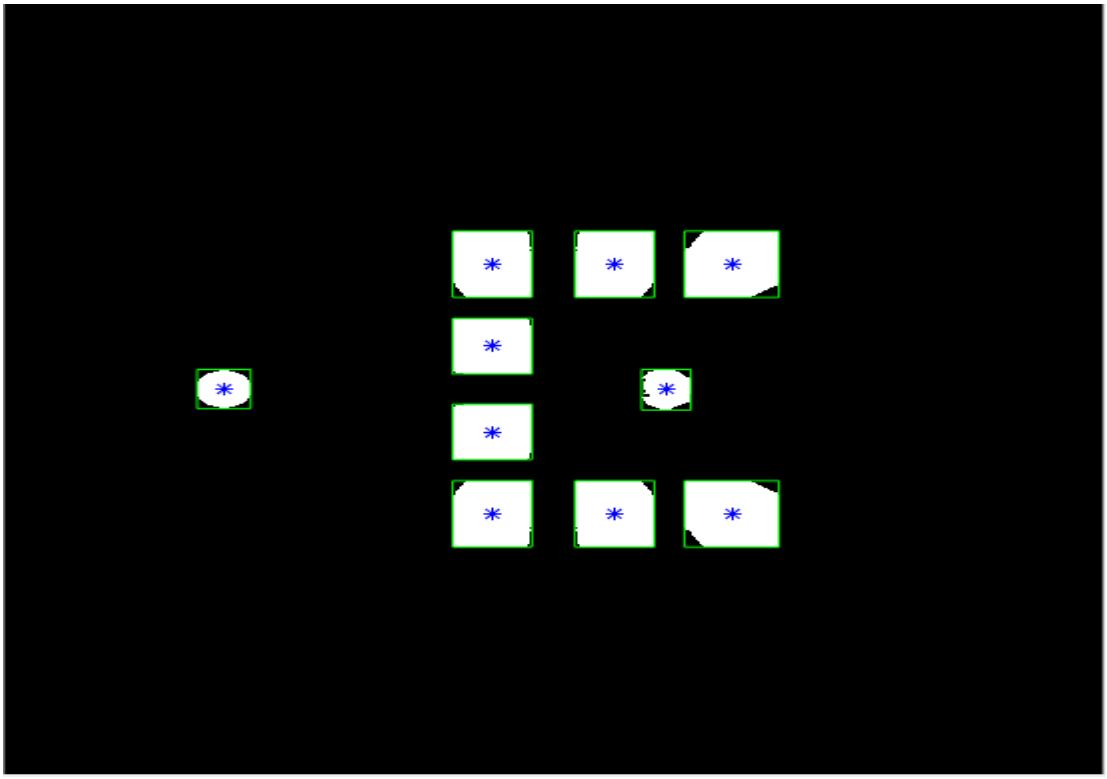


Fuente: Esta investigación, diseño realizado en V-REP por Harold Fernando Ruiz.

Si se usa la cámara se obtiene la posición de todos los objetos de la escena ejecutando el mismo procesamiento de imágenes que en el sistema Bola Plato, con la diferencia que el nuevo algoritmo debe poder diferenciar el piso (negro), los obstáculos (grises), el objetivo (rojo) y el robot (rojo con gris).

Para lograr lo anterior inicialmente se configuró la resolución de la imagen RGB a 512x512 pixeles, que corresponde al tamaño del escenario (10x10m). Adicionalmente se programó el algoritmo de procesamiento de imágenes con condiciones de área (tamaño del objeto), excentricidad (que tan circular es un objeto) y diámetro para poder diferenciar cada objeto de la escena. El objetivo final fue el calculó del centroide de cada objeto (Figura 18) y transformar esta posición a medidas en metros, tomando como referencia el espacio de trabajo de 10x10 metros.

Figura 18. Imagen resultante del procesamiento de imágenes escenario trampa.



Fuente: Esta investigación, imagen obtenida en Matlab por Harold Fernando Ruiz.

2.4.2. Modelo Matemático

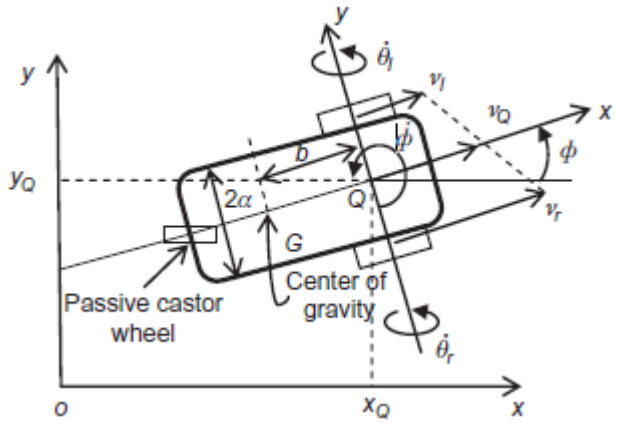
El modelo de un robot diferencial se describe en el libro *Introduce to Mobile robot control*[31], que tomando como referencia el robot de la Figura 19, se deducen las siguientes ecuaciones

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\phi}\end{bmatrix}=\frac{1}{2\alpha}\begin{bmatrix}(r/2)\cos\phi & (r/2)\cos\phi\\ (r/2)\sin\phi & (r/2)\sin\phi\\ (r/2\alpha) & -(r/2\alpha)\end{bmatrix}\begin{bmatrix}\dot{\theta}_r\\ \dot{\theta}_l\end{bmatrix},\qquad \begin{bmatrix}\dot{\theta}_r\\ \dot{\theta}_l\end{bmatrix}=\frac{1}{r}\begin{bmatrix}\cos\phi & \sin\phi & \alpha\\ \cos\phi & \sin\phi & -\alpha\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\phi}\end{bmatrix}, \tag{65}$$

donde x, y son la posición del vehículo,ϕ es el ángulo de giro del vehículo, $\dot{\theta}_l$ es la velocidad angular de la rueda izquierda, $\dot{\theta}_r$ es la velocidad angular de la rueda derecha, 2α es la distancia que hay entre los ejes de las dos ruedas y r es el radio de la rueda.

[31] TZAFESTAS.S. Introduce to Mobile robot control. Elsevier, 2013.

Figura 19. Modelo cinemático robot móvil con configuración diferencial.



Fuente: TZAFESTAS.S. *Introduce to Mobile robot control. Elsevier, 2013.*

2.4.3. Controladores

En esta planta virtual se logró que un robot móvil (Pioneer_P3dx) encuentre una ruta óptima y libre de obstáculos desde un punto inicial a un punto deseado en diferentes escenarios (Figuras 14 y 15). Para mostrar el desarrollo de estas estrategias, distintas a las de control de plantas tradicionales, se implementó cuatro algoritmos para la planeación de trayectorias descritos por Getial[30]. Adicionalmente se realizó un control de seguimiento cinemático para el robot diferencial, destacando que para el método de campos de potencial se elaboró un control proporcional adicional que permite el posicionamiento angular del robot. Además, cabe resaltar que la posición del robot se estimó mediante odometría.

Función de Navegación para campos potenciales artificiales (PF)

En esta técnica se definió el escenario como una función tridimensional $U(x)$, igual a la suma de funciones atrayentes (para el punto final) y repulsivas (para el punto inicial y los obstáculos). El robot se asumió como una partícula que encuentra una ruta libre de colisión definida como el gradiente descendente del potencial, introduciendo un tipo especial de potencial denominado función de navegación. Esta función está dada por una esfera de radio r_0, centrada en el punto cartesiano inicial x_0 que contiene n obstáculos de radio r_i ($r_{obstaculos}$) centrados en q_i, como

$$U(x) = \begin{cases} \dfrac{\|x - x_{goal}\|^2}{\|x - x_{goal}\|^{2k} + B(x)^{\frac{1}{k}}} & para\ \ B > 0 \\ 1 & para\ \ B \leq 0 \end{cases}, \qquad (66)$$

donde k es una constante de suavizado, x_{goal} la posición cartesiana del destino y B determina las posiciones de los obstáculos con la expresión

$$\begin{aligned} B(x) &= B_0(x)\prod_{i=0}^{n} B_i(x), \\ B_0(x) &= -\left\|x - x_{goal}\right\|^2 + r_0^2, \\ B_i(x) &= \left\|x - x_{goal}\right\|^2 + r_i^2. \end{aligned} \tag{67}$$

Getial[30] describe el desarrollo y aplicación de esta técnica en distintos escenarios y casos de estudio.

Optimización por enjambre de Partículas (PSO)

PSO es un algoritmo de búsqueda multi-agente inspirado en el comportamiento de enjambres de insectos, pájaros y peces, descrito por Alam[32]. En PSO para generar trayectorias se consideró a cada individuo de una población (enjambre) de tamaño n, como una partícula ubicada en el vector posición $x_i(t) \in \mathbb{R}^2$, que se mueve iterativamente de acuerdo con

$$x_i(t+1) = x_i(t) + v_i(t+1), \tag{68}$$

y una velocidad de desplazamiento v_i, cuya evolución está dada por

$$v_i(t+1) = v_i(t) + c_1\varphi_1(P_{ibest} - x_i) + c_2\varphi_2\left(P_{gbest} - x_i\right) + c_3\varphi_3(P_{nbest} - x_i), \tag{69}$$

donde φ_1, φ_2 y φ_3 son valores de una distribución aleatoria uniforme entre [0,1] (con media igual a cero y covarianza igual a uno) que representa los factores cognitivos y sociales que influyen en cada individuo, P_{ibest}, P_{gbest} y P_{nbest} son las mejores posiciones obtenidas por el individuo, del vecindario alrededor de un radio determinado (r_g) y el enjambre, respectivamente, y c_1, c_2 y c_3 son factores de ponderación.

Alam[32] presenta el cálculo de posiciones mediante una función de aptitud, con puntos máximos en las zonas ocupadas por obstáculos y mínimo global en la meta. En adición, El algoritmo se implementó con algunas recomendaciones encaminadas a la planificación de trayectorias.

[32] ALAM, M; RAFIQUE, M; KAHN, M. Mobile Robot Path Planning in Static Environments using Particle Swarm Optimization. En: International Journal of Computer Science and Electronics Engineering (IJCSEE), 2015, v.3, no.3, pp. 253-257.

Exploración rápida de árbol de azar (RRT)

RRT es un algoritmo estocástico que se basa en la construcción de un árbol de configuraciones o rutas que crece explorando a partir de una posición de partida (x_0). Mantiene la ventaja de completitud probabilística, es decir, que la probabilidad de encontrar un camino, si existe, tiende a uno exponencialmente con el número de nodos y hace que el costo de la solución encontrada converja casi seguramente al optimo; como se expone en el trabajo de Getial[30].

Descomposición de celdas (CD)

El método CD recurre a la descomposición trapezoidal del escenario, como se expone en el libro *Robot motion planning*[33], distinguiendo entre obstáculos y una posible área de recorrido. Esta técnica busca generar trayectorias en el espacio libre extendiendo rectas verticales sobre los vértices de cada objeto en el escenario. Después se elabora un mapa de ruta con los centros de los trapezoides creados y los centros de las extensiones verticales, eliminado del proceso a los trapezoides contenidos por objetos.

Una vez se tiene el mapa de ruta, se detecta la celda que contiene el punto objetivo (x_{goal}) de la ruta, tomando como partida el punto inicial (x_0). En este caso, para elegir la mejor trayectoria entre los puntos se aplica el algoritmo Dijkstra que se detalla en el libro *Robot motion planning*[33].

Parámetros de los métodos de planeamiento de trayectorias

Los parámetros que se sintonizaron mediante simulaciones y pruebas en la planta virtual se detallan en la Tabla 4, resaltando que el método CD no tiene parámetros debido a que se crea el mapa de ruta según el escenario deseado.

[33] BERG.M; CHEONG.O; OVERMARS.M. Robot motion planning Third Edition. DE: Springer, 2000, Ch. 13, p. 283-306.

Tabla 4. Parámetros usados Métodos de planeamiento de trayectorias.

PF	PSO	RRT
$\boldsymbol{k_{arreglo\ de\ obstaculos} = 4.5}$	$n = 100$	$N_{max} = 500$
$\boldsymbol{k_{trampa} = 5.1}$	$c_1 = 0.01$	$dis_p = 0.5$
$\boldsymbol{k_{pasaje\ estrecho} = 6.5}$	$c_2 = 0.025$	
$\boldsymbol{r_0 = 5}$	$c_3 = 0.5$	
	$Vel_{max} = 0.01$	
	$r_g = 0.2$	
	$nwf = 5$	

Fuente: Esta Investigación.

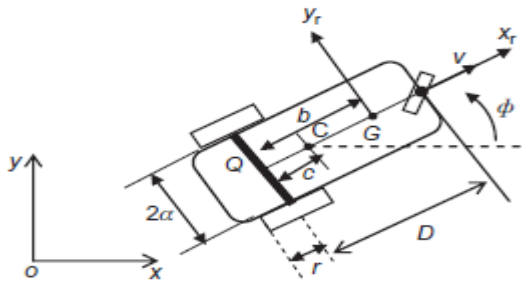
Control del Vehículo

El objetivo del control es establecer una estrategia para que el controlador siga las trayectorias planteadas por los algoritmos anteriores. Para esto se ha usado el método de seguimiento cinemático propuesto en el *libro Introduce to Mobile robot control*[31], en donde se consideró un modelo cinemático (Figura 20) sobre las coordenadas de referencia (x, y, ϕ) dado por

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}\cos\phi & -c\sin\phi\\ \sin\phi & c\cos\phi\end{bmatrix}\begin{bmatrix}v\\ w\end{bmatrix} = R(c,\phi)\begin{bmatrix}v\\ w\end{bmatrix}, \tag{70}$$

donde v, w son las velocidades lineal y angular del vehículo y c es la distancia al punto de control C (Figura 20), que para este caso en específico se fija en c=0.1 metros.

Figura 20. Cinemática de robot diferencial.



Fuente: *TZAFESTAS.S. Introduce to Mobile robot control. Elsevier, 2013.*

Ahora, mediante transformaciones matemáticas se obtiene la siguiente ley de control

$$\begin{bmatrix} v \\ w \end{bmatrix} = R^{-1}(c,\phi)\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -(1/c)\sin\phi & (1/c)\cos\phi \end{bmatrix}\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} (c \neq 0), \tag{71}$$

En ese sentido, se supuso una la ley de retroalimentación de estados dada por

$$\dot{x} = \dot{x}_d + K_x e_x, \qquad \dot{y} = \dot{y}_d + K_y e_y, \tag{72}$$

donde e_x y e_y son los errores en las coordenadas cartesianas x, y. (i.e., $e_x = x_d - x,\ e_y = y_d - y$).

Reemplazando (72) en (71) se obtiene el control cinemático no lineal

$$\begin{bmatrix} v \\ w \end{bmatrix} = R^{-1}(c,\phi)\left(\begin{bmatrix} \dot{x}_d \\ \dot{y}_d \end{bmatrix} + \begin{bmatrix} K_x & 0 \\ 0 & K_y \end{bmatrix}\begin{bmatrix} e_x \\ e_y \end{bmatrix}\right). \tag{73}$$

Adicionalmente, se supuso que las posiciones deseadas (x_d, y_d) son constantes resultando en

$$\begin{bmatrix} v \\ w \end{bmatrix} = R^{-1}(c,\phi)\left(\begin{bmatrix} K_x & 0 \\ 0 & K_y \end{bmatrix}\begin{bmatrix} e_x \\ e_y \end{bmatrix}\right), \tag{74}$$

donde K_x y K_y son constantes de calibración del controlador, que para los casos presentados en este documento (Figuras 14 y 15) se calibró en el intervalo $[0.07, 0.25]$ (dependiendo del método de planeamiento de trayectorias usado) para obtener una respuesta adecuada. Finalmente, se transformó v, w en velocidades angulares de los motores derecho e izquierdo ($\dot{\theta}_r, \dot{\theta}_l$) mediante la siguiente expresión

$$\dot{\theta}_r = \frac{v + \alpha w}{r}, \qquad \dot{\theta}_l = \frac{v - \alpha w}{r}. \tag{75}$$

Control proporcional para el método (PF)

Este controlador tiene la finalidad de posicionar el vehículo en un ángulo ϕ_d ($\phi_{deseado}$). Para este propósito se planteó un control proporcional sencillo que mueva solamente la rueda derecha (para que el robot gire sobre su propio eje), controlador necesario en el algoritmo (PF), y cuya expresión es

$$e_\phi = \phi_d - \phi, \qquad w = K_\phi e_\phi, \qquad v = 2\alpha w, \tag{76}$$

donde e_ϕ es el error de la posición angular del robot y K_ϕ es la constante proporcional del controlador.

Es clave mencionar que el valor K_ϕ se calibró por medio de la planta virtual en V-REP y se halló que un valor adecuado es $K_\phi = 2$.

Odometría de un robot con configuración diferencial

La odometría permitió estimar el desplazamiento y posición de un robot mediante las posiciones angulares de los motores izquierdo y derecho. Para realizar dicha estimación se usó las siguientes expresiones:

$$\begin{gathered} D_d = r\theta_r, \qquad D_i = r\theta_l, \qquad D_{robot} = \frac{D_d + D_i}{2}, \\ x = D_{robot}\sin\phi + x_0, \qquad y = D_{robot}\cos\phi + y_0, \end{gathered} \tag{77}$$

donde D_d y D_i son el desplazamiento de la rueda derecha e izquierda, x_0 y y_0 son las posiciones iniciales del robot y D_{robot} es el desplazamiento total del robot.

Finalmente, se destaca que en este trabajo se conoce las mediciones de la posición angular del robot ϕ y las posiciones angulares de las ruedas θ_r y θ_l, las cuales se obtuvieron mediante la API V-REP – Matlab.

3. ANALISIS Y RESULTADOS

3.1. SIMULACIONES

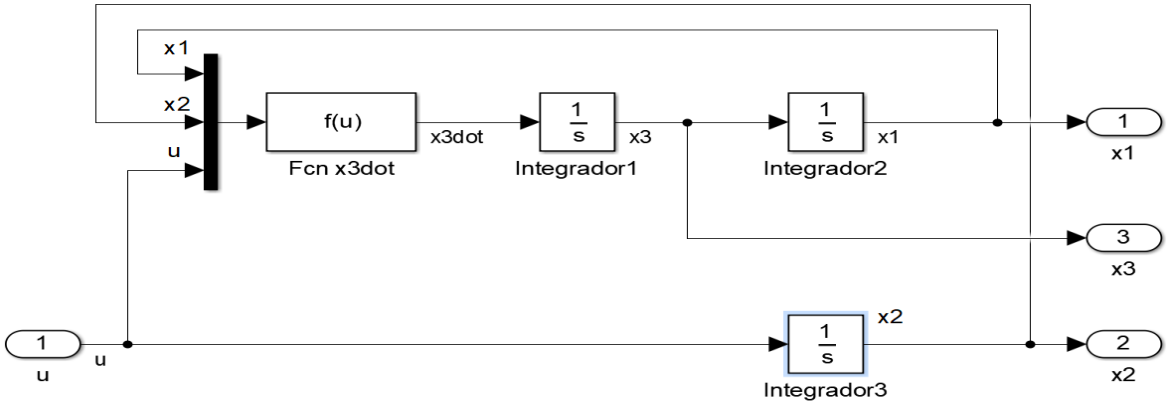
En este apartado se describió las simulaciones realizadas de los diferentes controladores que se han diseñado en la anterior sección. Las comparaciones con los datos de implementación se muestran en la sección de resultados para elaborar un mejor análisis.

3.1.1. Sistema bola viga

Sistema en lazo abierto

En esta primera simulación se ilustró la construcción del sistema no lineal en lazo abierto sin ningún tipo de control. En la Figura 21 se muestra el diagrama de bloques utilizado en Simulink.

Figura 21. Diagrama en Simulink del modelo no lineal del sistema Bola-Viga.



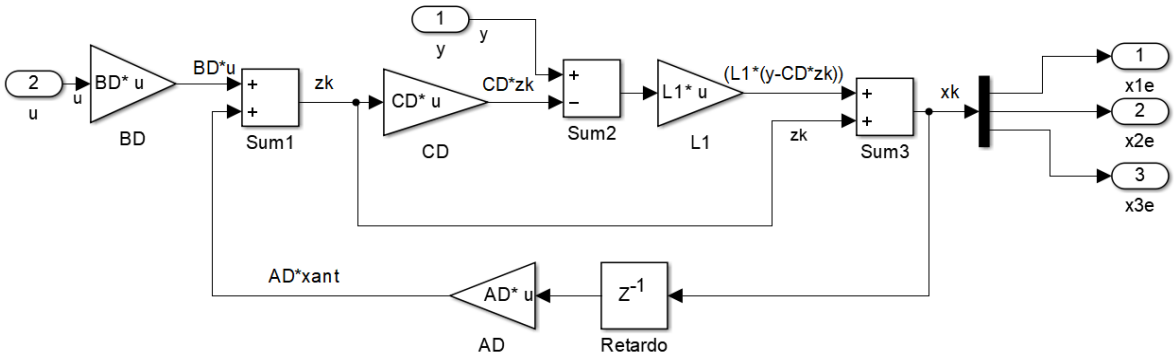
Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez.

La planta corresponde al modelo no lineal del sistema bola-viga que se describió en la Sección 2, en donde el bloque “Fcnx3dot” crea la relación que alimentan a la variable $\dot{x_3}$ (Ecuación 2). Esta es la planta que se usó en todas las simulaciones del modelo no lineal del sistema bola-viga, puntualizando que el diagrama de la Figura 21 se construyó dentro de un subsistema.

Sistema con retroalimentación de estados con integrador y un observador discreto

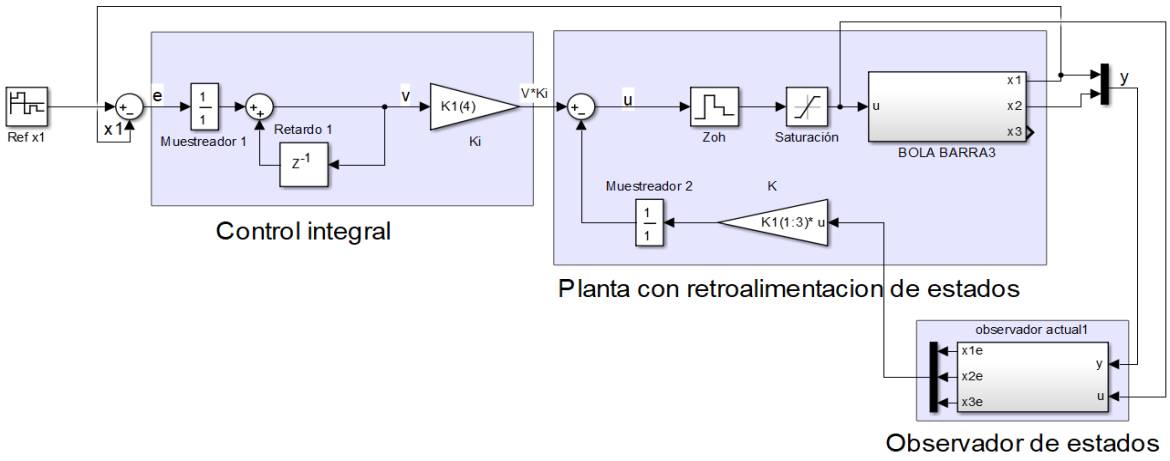
En la Figura 22 se muestra el diagrama en Simulink del observador de estados utilizado, con un vector de salida $\mathbf{y} = [x_1 \quad x_2]$ y una matriz de ganancias del observador $\boldsymbol{L1}$ encontrada en la sección (2.1.3) . Adicionalmente, en la Figura 23 se ilustra el esquema de control discreto por retroalimentación de estados con integrador y un observador discreto completo, donde $\boldsymbol{K}$ y K_i son las constantes de retroalimentación del LQR calibradas con anterioridad.

Figura 22. Diagrama en Simulink del observador discreto.



Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez

Figura 23. Diagrama en Simulink del Controlador LQR con observador de estados.



Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez.

El observador mostrado en la Figura 22 está implementado en el subsistema “observador actual 1” (Figura 23).

Simulación MPC

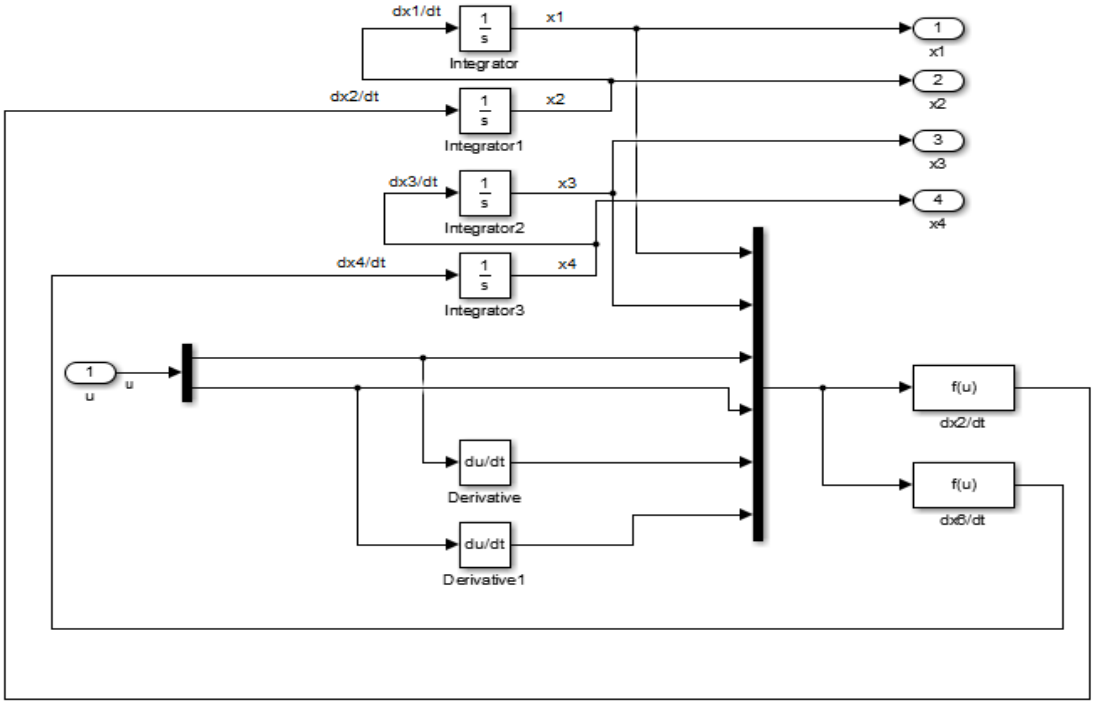
En el caso del MPC, la simulación se hizo en un script de Matlab debido a la dificultad de la implementación de restricciones y calibración de parámetros en Simulink. También se destaca que en el script se realizó el observador de estados para el controlador MPC.

3.1.2. Sistema Bola-Plato

Sistema en lazo abierto

En la Figura 24 se muestra el diagrama de bloques utilizado en Simulink para el sistema no lineal bola-plato.

Figura 24. Diagrama en Simulink del modelo no lineal de un Sistema Bola-Plato.



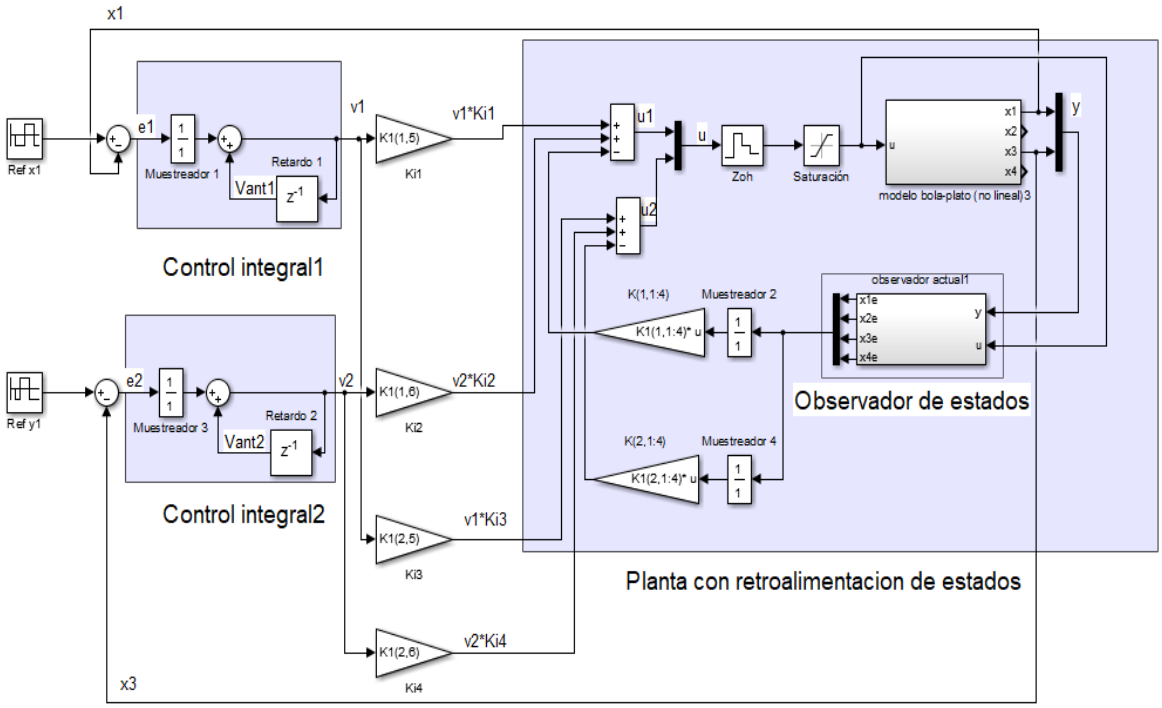
Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez.

La planta corresponde al modelo no lineal del sistema bola-plato explicado en la Sección 2. Los bloques “dx2/dt” y “dx6/dt” crean la relación que alimentan a las variables $\dot{x_2}$ y $\dot{x_4}$ (Ecuación 28).

Sistema con retroalimentación de estados con integrador (MIMO) y un observador discreto

En este caso, el diagrama del observador es el mismo que en el sistema bola-viga (Figura 22). Aquí, $\mathbf{y} = [x_1 \quad x_3]$ y $\boldsymbol{L1}$ es la matriz de ganancias $\boldsymbol{L}$ (Sección 2.2.3). Ahora, en la Figura 25 se ilustra el esquema de control discreto por retroalimentación de estados con integrador y un observador discreto completo para el sistema bola-plato (MIMO), donde la matriz $\boldsymbol{K}$ y $\boldsymbol{K_i}$ son las constantes de retroalimentación del LQR halladas en la sección anterior.

Figura 25. Control por retroalimentación de estados con integrador y observador sistema bola-plato MIMO.

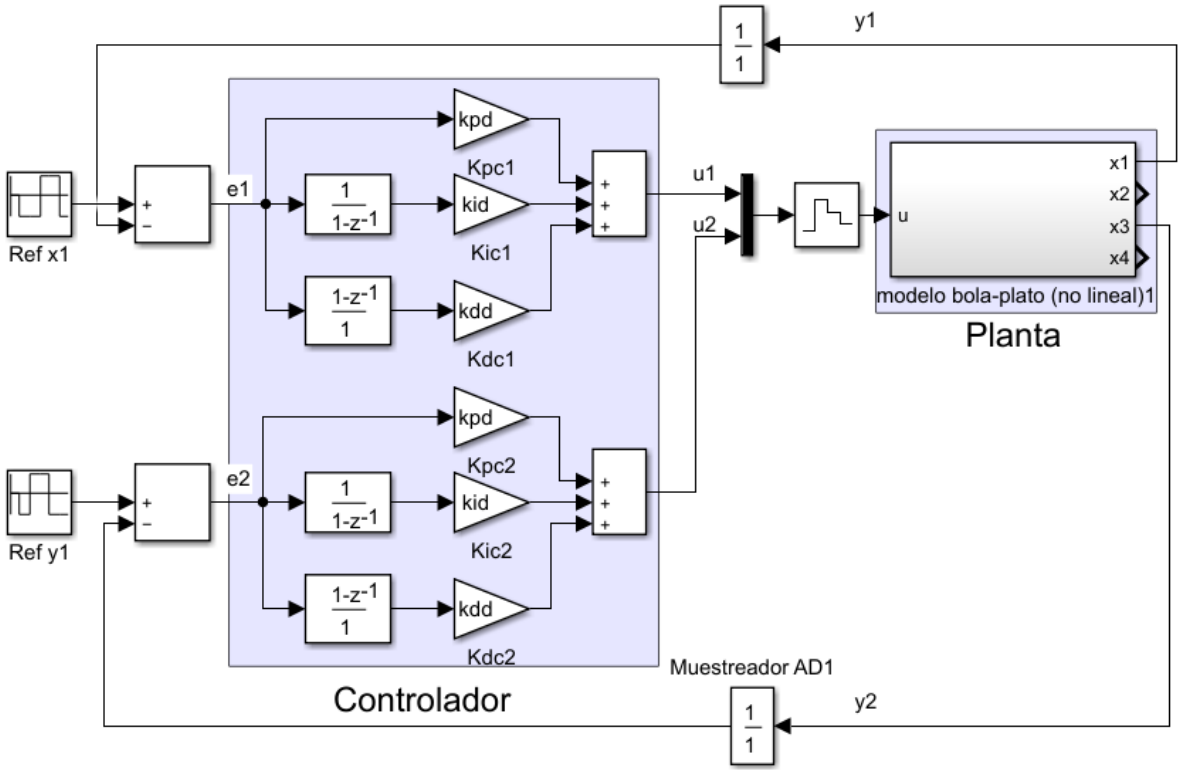


Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez.

PID discreto sistema SISO

El esquema que se realizó en Simulink para este controlador se aprecia en la Figura 26 donde kpd, kid y kdd son las constantes del controlador PID k_P , k_I y k_D (Ecuación 39).

Figura 26. Control PID sistema Bola-Plato.



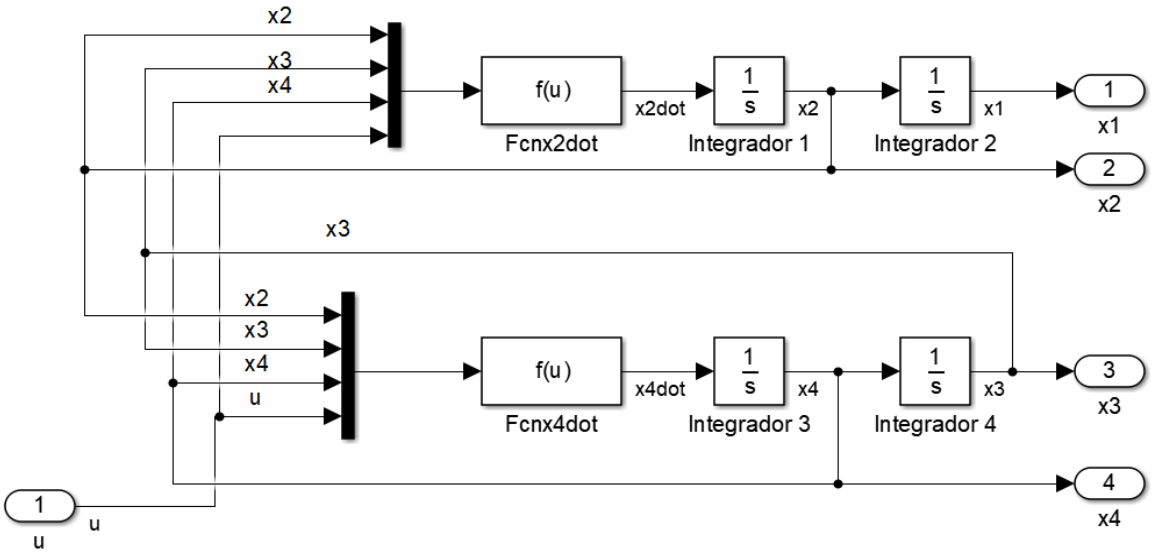
Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez.

3.1.3. Péndulo de Furuta

Sistema en lazo abierto

En la Figura 27 se muestra el diagrama de bloques utilizado en Simulink para simular el comportamiento de un Péndulo de Furuta.

Figura 27. Diagrama en Simulink del modelo no lineal del Péndulo de Furuta.



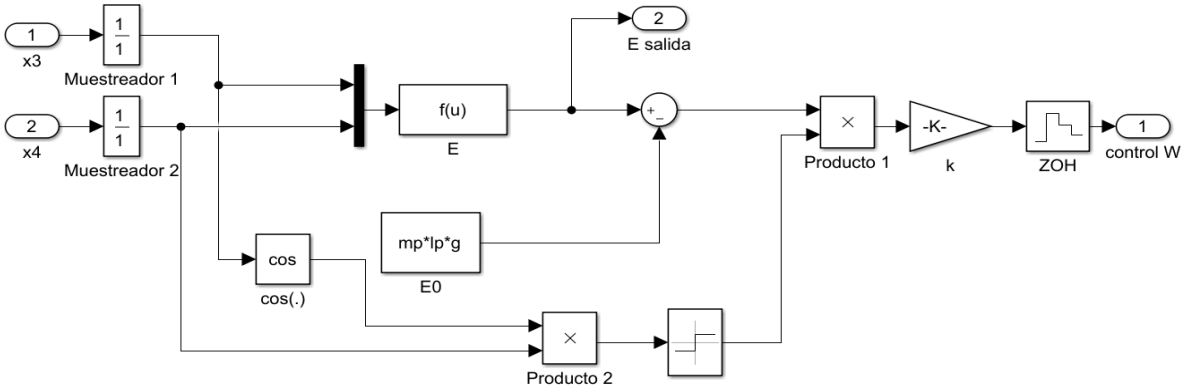
Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez.

La planta corresponde al modelo no lineal del Péndulo que se describió en la Sección 2. Los bloques “Fcnx2dot” y “Fcnx4dot” crean la relación que alimentan a las variables $\dot{x_2}$ y $\dot{x_4}$ (Ecuación 42).

Control Swing up

El diagrama que se muestra en la Figura 28 corresponde al controlador Swing up explicado anteriormente donde el bloque E crea la relación que alimenta la variable E (ecuación 53).

Figura 28. Controlador Swing up en Simulink.

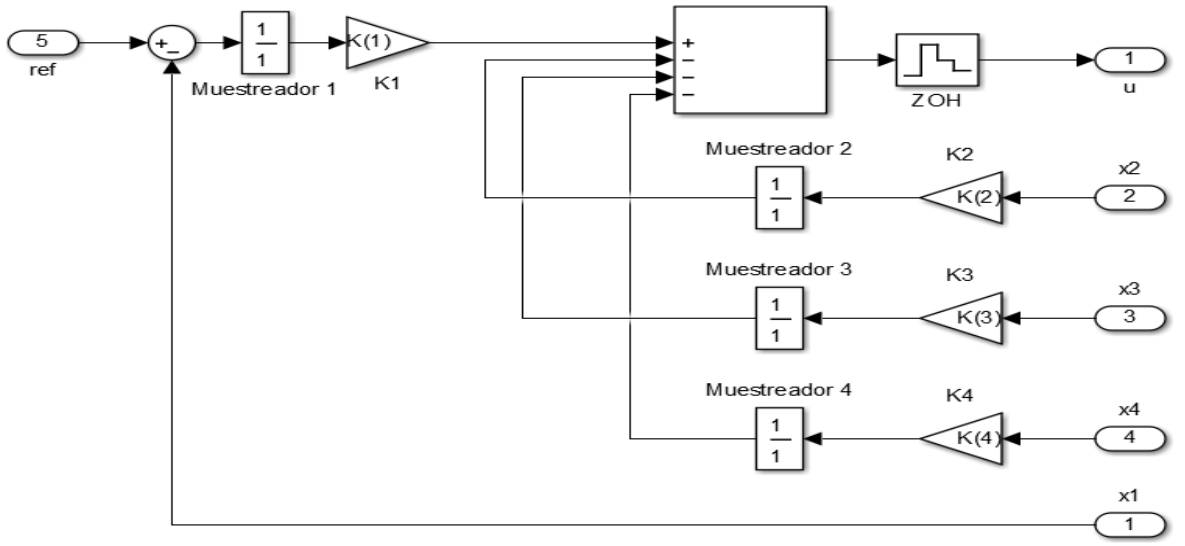


Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez.

Control LQR

En la Figura 29 se ilustra el esquema de control discreto por retroalimentación de estados para el péndulo de Furuta, donde el vector $\boldsymbol{K}$ y la constante K_r son las ganancias de retroalimentación del LQR halladas en la sección anterior.

Figura 29. Controlador LQR en Simulink.

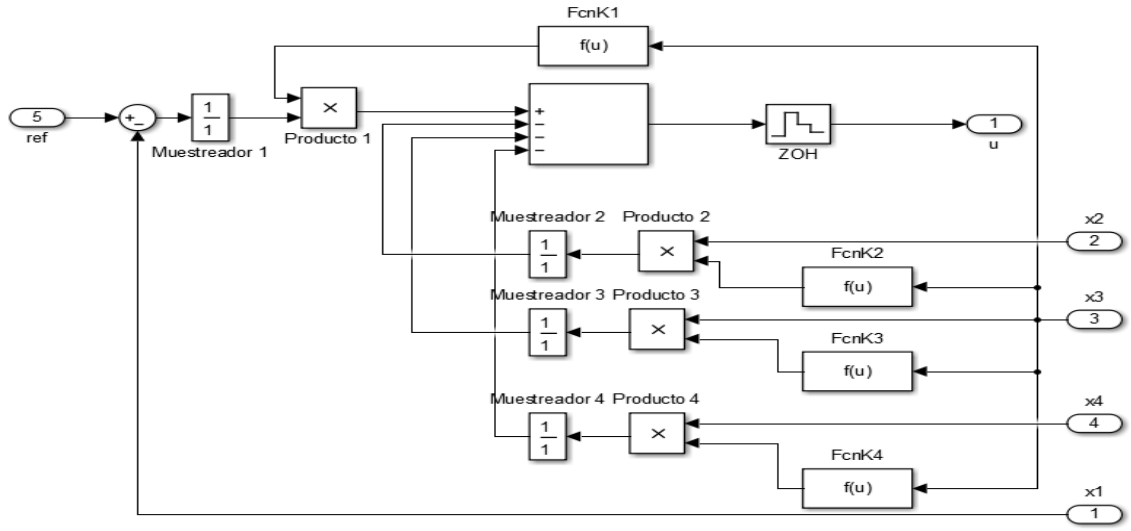


Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez.

Controlador Gain Sheduling

En la Figura 30 se aprecia el diagrama de control discreto para el gain scheduling aplicado al Péndulo de Furuta, donde las funciones FcnK1, FcnK2, FcnK3 y FcnK4 son las funciones de las variables de programación mostradas en la Ecuación 51.

Figura 30. Controlador Gain Sheduling en Simulink.



Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez.

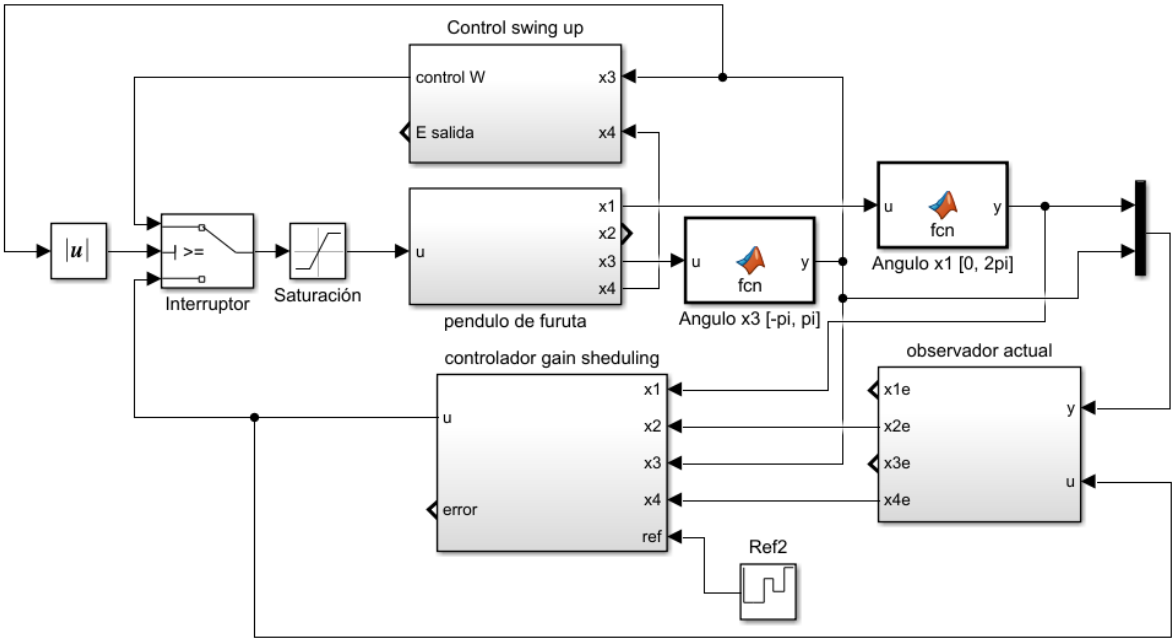
Observador de estados actual

En este caso el diagrama del observador es el mismo que en el sistema bola-plato (Figura 22), puntualizando que $\mathbf{y} = [x_1 \quad x_3]$ y $\boldsymbol{L1}$ es la matriz de ganancias $\boldsymbol{L}$ (Sección 2.3.3).

Controlador Gain Scheduling y LQR completo (con swing up y observador)

En la Figura 31 se aprecia el controlador Gain scheduling, destacando que la función "Angulo x1 y Angulo x3" sirven para que el ángulo del motor (x_1) este en el rango [0, 2π] y el ángulo del péndulo (x_3) se encuentre en el intervalo de $[-\pi, \pi]$. Adicionalmente el bloque "interruptor" cumple la función de cambiar el controlador Swing up al Gain scheduling o LQR, cambio que se hace cuando el ángulo del péndulo (x_3) se encuentra en una posición angular menor a los +- 18°. También se aclara que si se desea cambiar al controlador LQR en la simulación simplemente se cambia el subsistema "Gain scheduling" por el diagrama mostrado en la Figura 29.

Figura 31. Controlador Gain scheduling, Swing up y observador de estados para el Péndulo de Furuta.



Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez.

3.1.4. Planeamiento de trayectorias de un robot Terrestre

En este caso las simulaciones de los algoritmos de planeamiento de trayectorias no se realizaron en Simulink sino en “scripts” de Matlab, debido principalmente a la dificultad de programación en Simulink de los algoritmos detallados en la Sección 2.4.1. Adicionalmente, se resalta que en repositorio[34] se encontró las simulaciones de los planeamientos de trayectoria que se trabajan en este documento.

[34]GETIAL, J; ERAZO, E; ANDRES, P. A Quantitative Comparison of Path Planning Methods in Mobile Robotics. {En línea}. {11 de marzo del 2019}. Disponible en: (https://sites.google.com/view/pathplanningthese/simulations/unique-robot?fbclid=IwAR0cGIQz8wiUhLYtlOZYLX1I4n139ryej7nGzbK7hSwWsZk1gTMWknMtxVsI).

3.2. IMPLEMENTACIÓN

3.2.1. Implementación de los Controladores en V-REP conectado Matlab

Para poder implementar los controladores y observadores diseñados fue necesario realizar los lazos de control por medio de la API remota de V-REP. En el Anexo 1 se detalló la implementación del controlador LQR con integrador y un observador de estados discretos para el sistema Bola Viga. Este control presenta la guía para la implementación de los demás algoritmos para las plantas tradicionales. Igualmente, en el mismo Anexo se describió la elaboración de un planeamiento de trayectorias para un robot terrestre usando el algoritmo de Kevin Passino[35].

3.2.2. Implementación de los Controladores en V-REP conectado Simulink

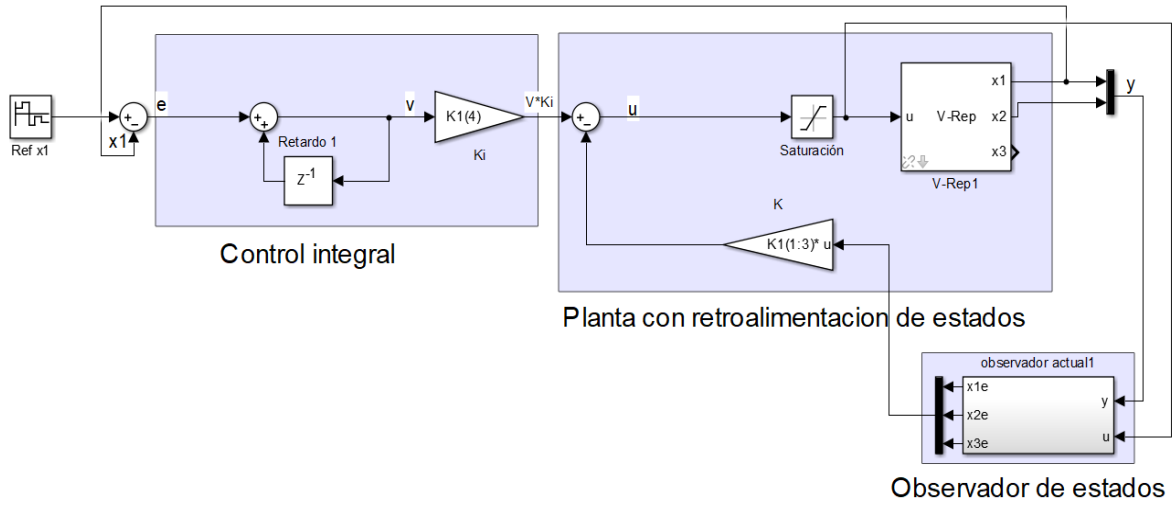
Los controladores que se realizaron por medio de la plataforma creada para Simulink se limitaron a las plantas tradicionales (excepto el control MPC y los controles del péndulo de Furuta por razones que se detallan en secciones posteriores). Dada la gran complejidad que presentó la construcción de los diferentes escenarios de planeamiento de trayectorias (sección 2.4) mediante técnicas de potencial artificial, arboles de ruta o trapezoides y a lo increíblemente tedioso que sería elaborar los controladores planteados en este documento (2,4,3) en el entorno visual Simulink, ninguno de los algoritmos de planeamiento de trayectorias se implementó en dicho programa.

Descartando los algoritmos complejos, la implementación de los controladores en Simulink resulta bastante sencillo puesto que son casi exactamente iguales a los mostrados en las simulaciones. Por lo tanto, en este documento solamente se detalló la implementación del controlador LQR con integrador para el sistema bola viga.

La implementación se realizó reemplazando el bloque del modelo matemático detallado en la Sección 3.1.1 (Figura 23) por el bloque creado que posibilita la comunicación transparente con V-REP (Anexo 1), como se aprecia en la Figura 32.

[35]PASSINO, K. Vehicule guidance for obstacle avoidance example. {En línea}. {29 de marzo del 2018}. Disponible en: (http://www2.ece.ohio-state.edu/~passino/ICbook/Code/vehic_guide.m).

Figura 32. Diagrama del esquema de control para LQR con observador de estados en la plataforma V-REP - Simulink.



Fuente: Esta investigación, diagrama realizado en Simulink por Harold Fernando Ruiz y Laura María Rodríguez.

Es clave decir que el esquema de Simulink de la Figura 32 no fue necesario agregar ni muestreadores, ni retenedores de orden cero (ZOH) ya que la simulación se realizó en tiempo discreto.

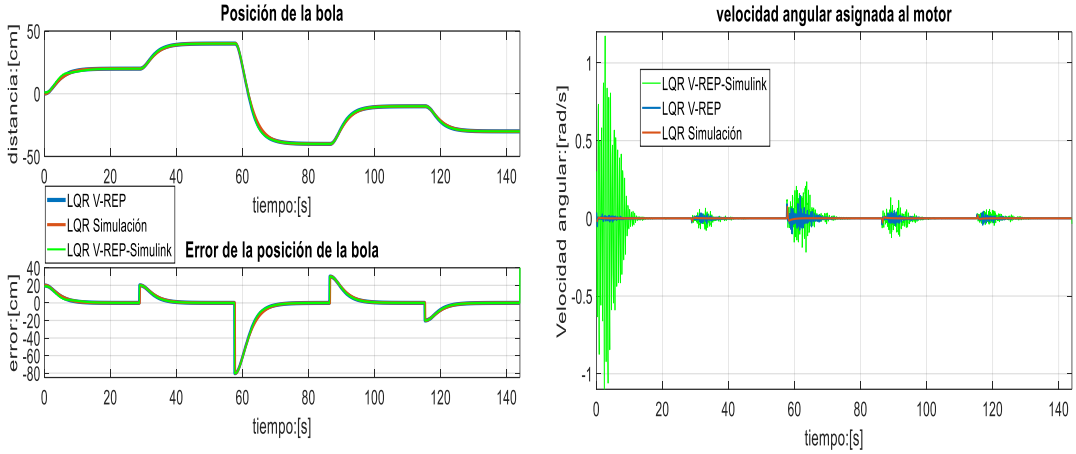
3.3. RESULTADOS

3.3.1. Sistema bola viga

Para comprobar el correcto funcionamiento de la API Remota (tanto en Simulink como en Matlab), se implementó los controladores de la Sección 3.1 y se elaboró un análisis de los resultados, puntualizando que se compara i) la simulación ideal hecha en Simulink, ii) el lazo de control realizado desde un script de Matlab conectado a V-REP y iii) el esquema en bloques en Simulink conectado a V-REP. En ese sentido tanto en la simulación como en la planta virtual en Simulink y Matlab se intentó iniciar con las mismas condiciones iniciales comenzando con la bola detenida en 0 cm, cambiando luego la posición cada 28.8 segundos a 20 cm, 40 cm, -40 cm, 10 cm y -30 cm para el caso del LQR con observador y cada 7.2 segundos para el MPC con restricciones. Los resultados se muestran en la Figura 33 para el LQR y la Figura 34 para el MPC.

Figura 33. Señales de respuesta del controlador LQR

a) b)

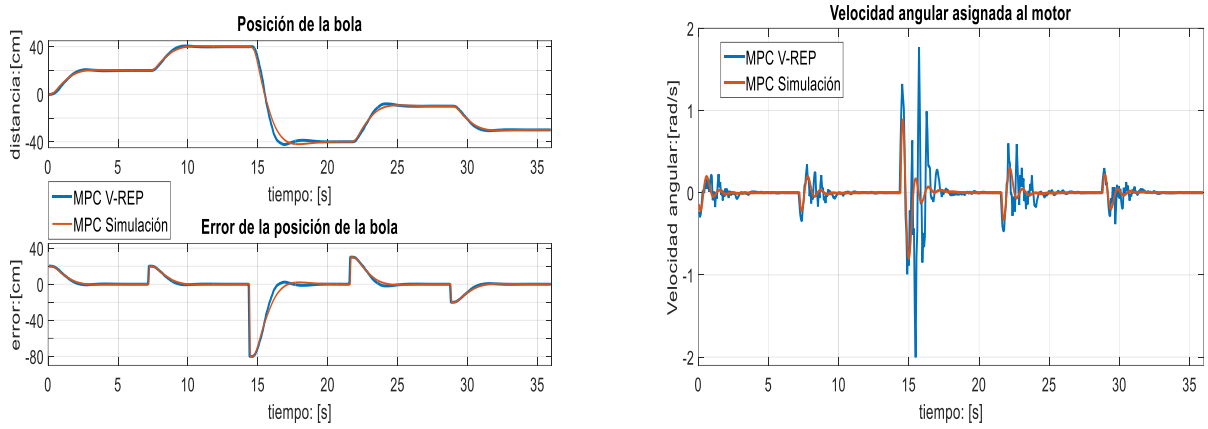


Fuente: Esta investigación, Grafica realizada en Matlab por Harold Fernando Ruiz y Laura María Rodríguez.

De la Figura 33a, la planta virtual en V-REP (Co-simulación Matlab y Simulink) tiene casi el mismo tiempo de establecimiento en comparación con el sistema simulado. Sin embargo, los factores físicos no modelados como la fricción y el contacto entre la bola y la viga, son más evidentes en las señales de control que se mostraron en la Figura 33b. En el caso de V-REP comunicado con Matlab, el motor presentó más oscilaciones en los cambios de punto de ajuste para guiar la pelota. En Simulink la señal resultó mucho más agresiva que en Matlab, debido a que las condiciones iniciales afectan en gran medida el observador de estados discreto haciendo que este sea más sensible, lo anterior se debe principalmente a los parámetros de configuración que se tenga en Simulink (principalmente el *solve*) y al desbordamiento del intervalo de muestreo (step size) generado primordialmente por una demora en él envió de datos desde la API de V-REP. Pese a todo lo anterior, las respuestas de la simulación con modelo matemático y la simulación en V-REP conectado a Simulink y Matlab son similares, mostrando la robustez del controlador LQR.

Figura 34. Señales de respuesta ante el controlador MPC

a) b)



Fuente: Esta investigación, Grafica realizada en Matlab por Harold Fernando Ruiz y Laura María Rodríguez.

Ahora, si se analiza el MPC se observó que es mucho más rápido que el LQR y aunque presenta un pequeño sobrepaso sigue bastante bien al controlado que se diseñó con el modelo matemático. Sin embargo, en la Figura 34a se mostró que la posición oscila más en la tercera referencia, debido a la acción de control (Figura 34b) mucho más agresiva en ese punto. También se enfatiza que en la acción de control (Figura 34b) se pudo observar los factores físicos no modelados como la fricción y el contacto entre la bola y la viga.

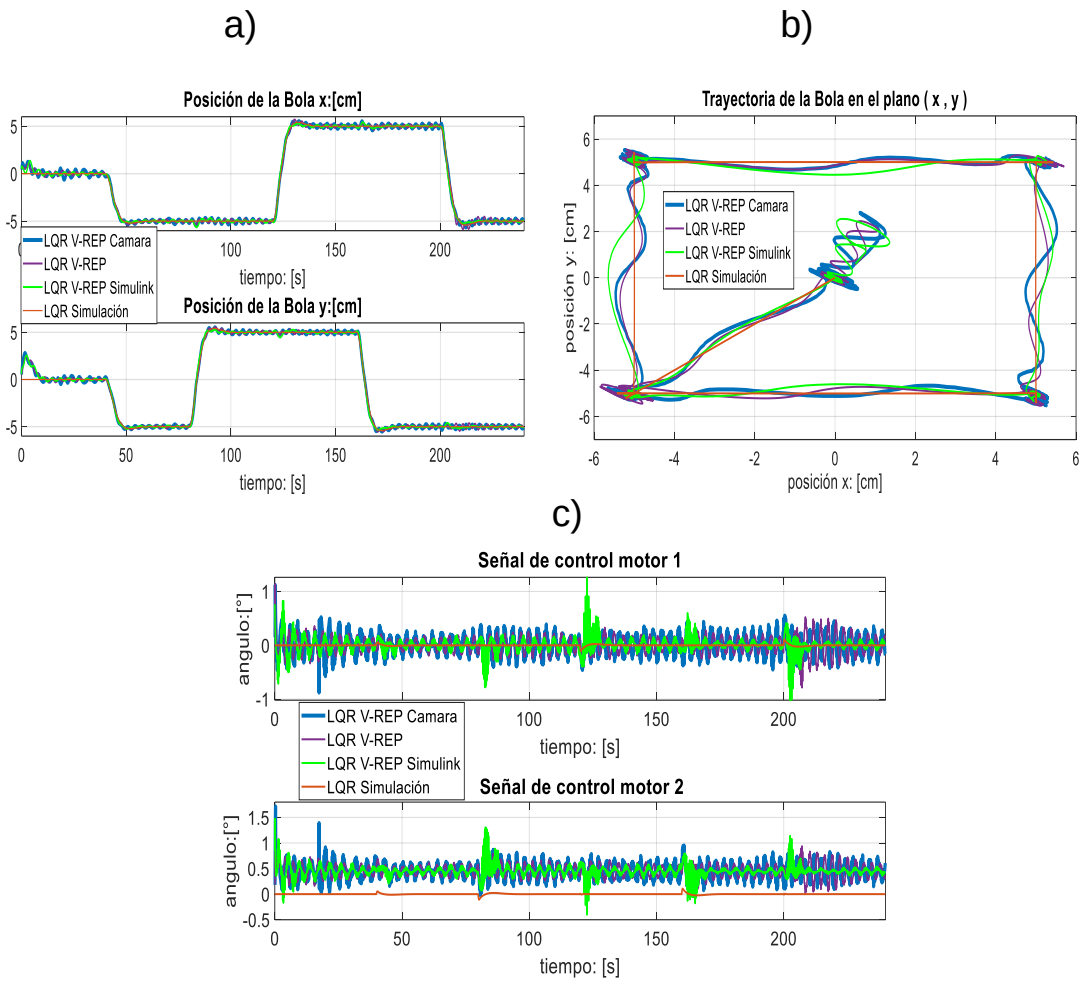
3.3.2. Sistema bola plato

Los resultados sobre la planta que se implementó son analizados comparando las respuesta de los sistemas a los controladores diseñados en cuatro escenarios: i) usando modelos ideales en una simulación, ii) con la interacción de los controladores en Matlab regulando la posición de la pelota en la planta virtual a través de la co-simulación (con la posición de la pelota obtenida directamente de la API, iii) usando la planta virtual en co-simulación pero con una cámara para obtener la posición de la pelota, y iv) usando la plataforma creada entre V-REP y Simulink. Para probar esta planta, se propuso seguir una trayectoria simple, comenzando en reposo en la posición (0,0) cm, luego cambiando el punto de ajuste cada 40 segundos a (-5, -5), (-5,5), (5,5), (5, -5) y (-5, -5) centímetros.

Las respuestas para el controlador LQR MIMO se muestra en la Figura 35, que detalla la posición de la pelota en el tiempo, la trayectoria en la placa y las señales de los actuadores, respectivamente. Se observó que la posición en cada eje se logra

sin error de estado estacionario debido al integrador adicional en el diseño. Además, las diferencias en la simulación y la planta virtual (con y sin cámara) se notaron en todos los resultados.

Figura 35. Señales de respuesta LQR trabajado como un sistema MIMO



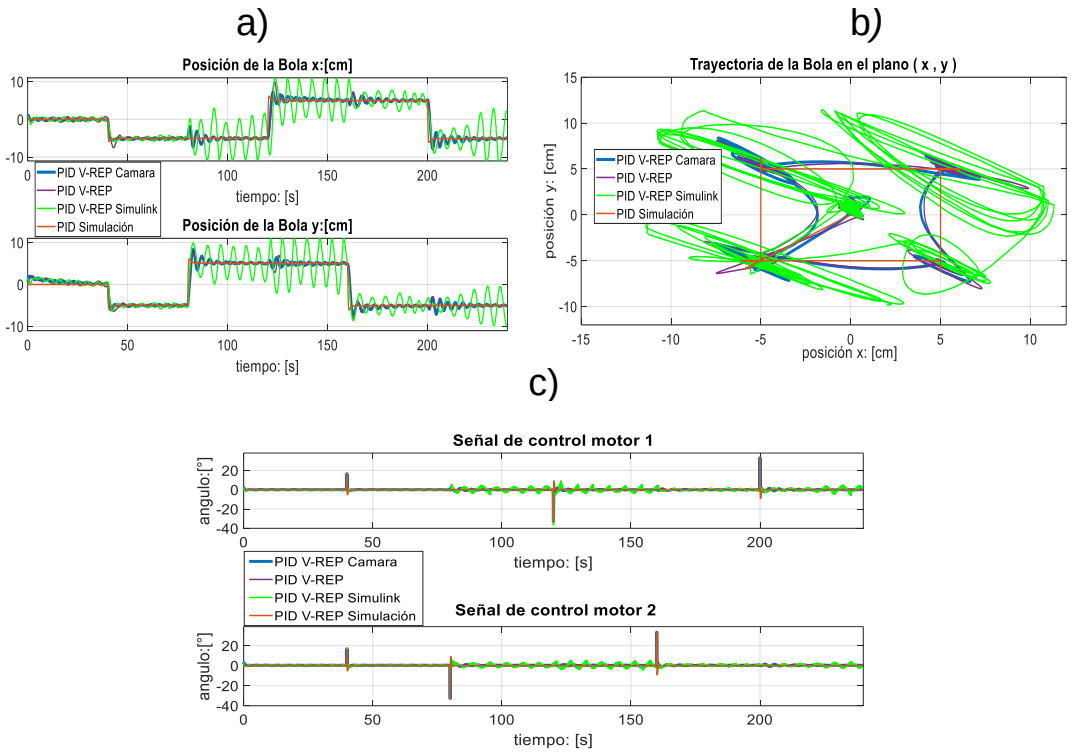
Fuente: Esta investigación, Grafica realizada en Matlab por Harold Fernando Ruiz y Laura María Rodríguez.

De la Figura 35c, se notó que la señal de control del motor 2 en la planta virtual (tanto en Matlab como en Simulink) difiere claramente de la simulación. Esta respuesta es porque la placa en la planta virtual no está perfectamente alineada con el origen, y el controlador intenta compensarlo permaneciendo en un valor constante para equilibrar la mesa en la posición horizontal. Este percance provocó que la bola no comience en (0,0) cm (Figura 35b), ya que tiende a deslizarse hacia el lado positivo del plano (x, y). Sin embargo, a pesar de los parámetros contemplados por

el entorno virtual (fricción, gravedad y contacto entre la pelota y el plato) en general el comportamiento de la planta virtual en V-REP (Script y Simulink) es similar a la simulación. Adicionalmente, se enfatiza que las señales obtenidas mediante el uso de la cámara (especialmente en las Figuras 35b y 35c) presentan más oscilaciones debido al error existente en la conversión de píxeles por la baja resolución de la imagen. También se señala que la posición inicial y las señales de control presentan un pequeño cambio, puesto que las oscilaciones aumentan dado que el controlador intenta siempre corregir el error presentado en la posición (Figura 35c).

Por otro lado, las respuestas para el controlador PID discreto son bastante diferentes, como se muestra en la Figura 36. Aunque en la Figura 36a la simulación muestra cambios suaves en los puntos de ajuste, la respuesta real en la planta virtual presenta excesos y oscilaciones. Este comportamiento es más evidente en la Figura 36b, donde la trayectoria seguida por la planta real es completamente distinta a la presentada en la simulación.

Figura 36. Señales de respuesta PID trabajado como un sistema SISO



Fuente: Esta investigación, Grafica realizada en Matlab por Harold Fernando Ruiz y Laura María Rodríguez.

Las Figuras 36a y 36b señalan que también se presentó el desplazamiento inicial de la mesa mencionado anteriormente, ya que la pelota no comienza en el origen. La compensación del controlador se evidenció en la Figura 36c, donde la señal de control para la planta virtual es más alta que la del controlador LQR (logrando picos entre 35 y -35 grados). El controlador PID, en este caso, es más agresivo que el LQR, y luego, la presencia de fricción y deslizamiento de la pelota en el entorno virtual causan grandes diferencias con la simulación. Asimismo, estas diferencias son más notables en este caso debido a la baja robustez del diseño del controlador PID. Además, el rendimiento del controlador que usa la cámara (Figura 36b) es peor debido al error de conversión mencionado anteriormente. Igualmente, en las Figuras 36a y 36b se apreció que el controlador implementado en co-simulacion con Simulink presenta el peor comportamiento de todos cuando cambia a la tercera y quinta referencia. Lo anterior se originó porque Simulink conectado a V-REP es mucho más sensible a las condiciones iniciales y a los cambios bruscos en las referencias que su contraparte en Matlab, generando mayores fluctuaciones en las señales de control (Figura 36c), lo anterior ocurre por las mismas razones mencionadas en la sección (3.3.1).

Para finalizar, se comparó los controladores, el controlador LQR presentó un mayor rendimiento en robustez, tiempo de ajuste, señal de control y error de estado estable sobre el controlador PID. Las oscilaciones en el controlador PID produjeron un tiempo de estabilización más largo, mientras que los cambios bruscos en la señal de control permitió que la bola se deslice, lo que aumenta los errores.

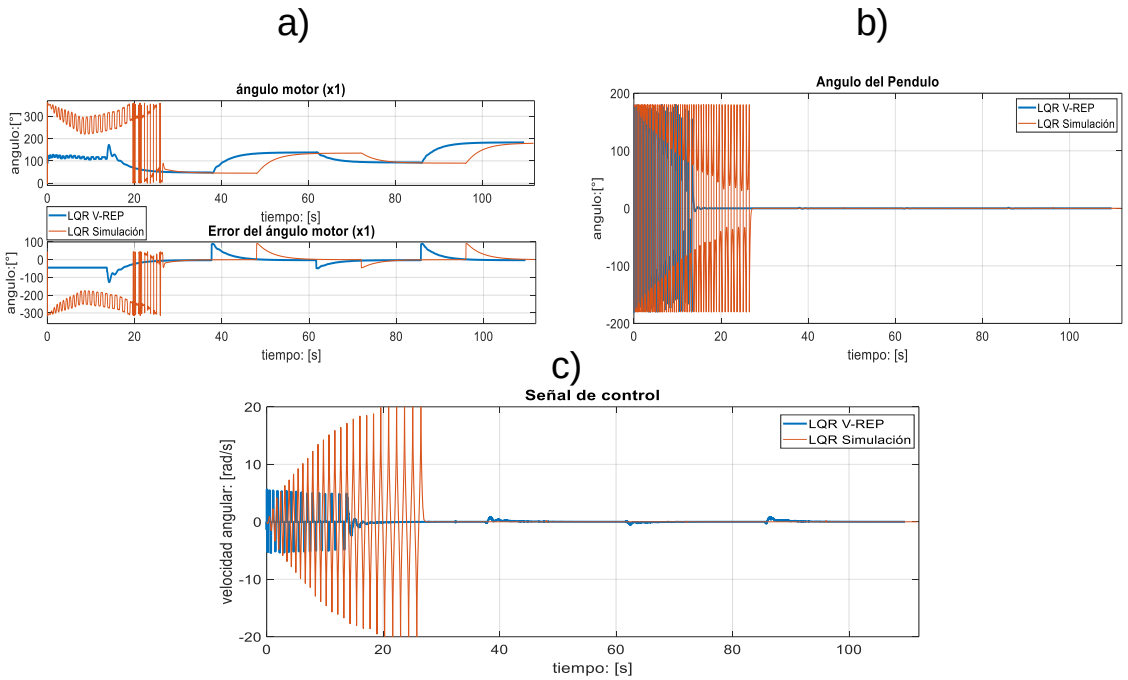
3.3.3. Péndulo de Furuta

En esta planta virtual se analizó las respuestas de los sistemas a los controladores diseñados en dos escenarios: i) usando modelos ideales en una simulación, ii) con la interacción de los controladores en Matlab regulando la posición del péndulo y brazo a través de la co-simulación.

Para probar esta planta, se propuso comenzar con el brazo del péndulo en 90° y el péndulo en -180°, luego se esperó a que los controladores posicionen el péndulo en 0° y el brazo en 45°, y después cada 24 segundos se cambió la posición del brazo a 135°, 90° y 180°.

Las respuestas para el controlador LQR se muestran en las Figura 37, detallando la posición del brazo del péndulo, la posición del péndulo y la señal de control del actuador, respectivamente. La posición angular del péndulo y el brazo se lograron casi sin error de estado estacionario puesto que al sistema en la transformación de torque a velocidad (Sección 2.3.3) se le adicionó un integrador discreto.

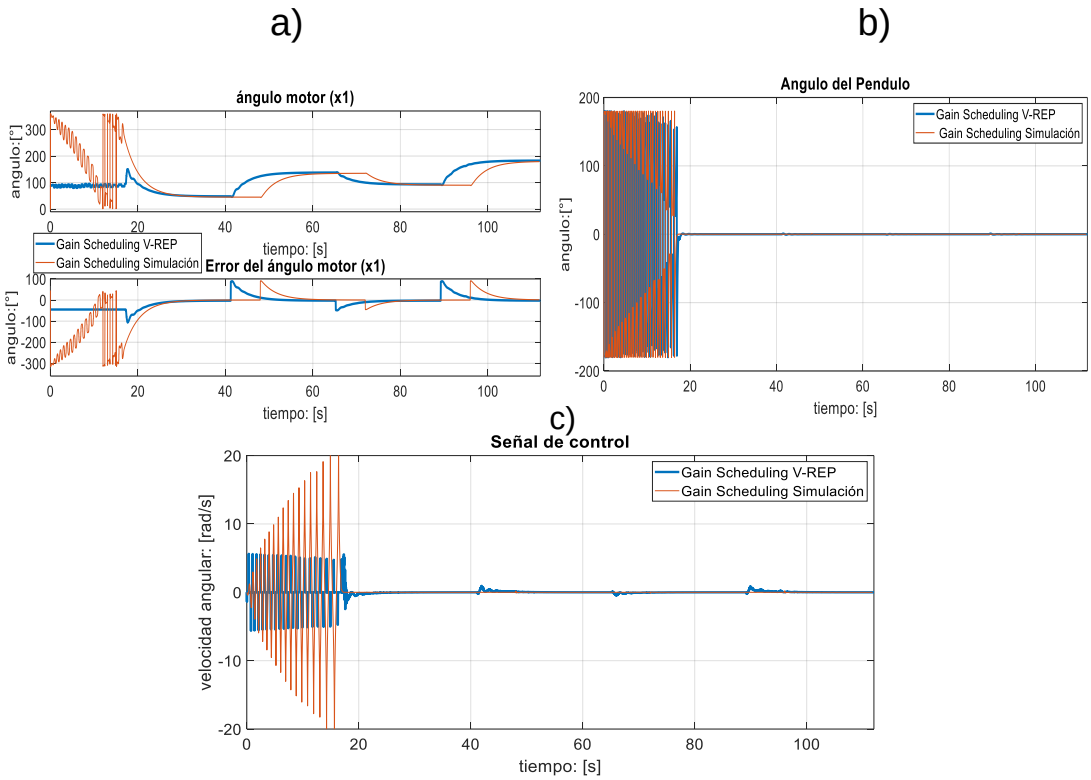
Figura 37. Señales de respuesta del Swing up y LQR



Fuente: Esta investigación, Grafica realizado en Matlab por Harold Fernando Ruiz y Laura María Rodríguez.

De la Figura 37c, se observó que la señal de control del motor en la planta virtual difiere notablemente de la simulación. Esta respuesta se debió al tiempo que estuvo activo el controlador Swing up en la planta virtual, el cual depende de las dinámicas físicas del sistema (fuerza aplicada al motor, fricciones, gravedad, etc.) y puede cambiar con cada nueva simulación. En contraste, en la simulación (modelo matemático) el tiempo de activación del swing up siempre es el mismo. Sin embargo, a pesar de las diferencias de duración del Swing up (en la simulación y en V-REP) y de los parámetros contemplados por el entorno virtual (fricción, gravedad y contacto entre el brazo y el péndulo) en general el comportamiento de la planta virtual en V-REP es similar a la simulación en Simulink (modelo matemático) una vez se estabilizó la planta en la primera referencia (Figuras 37a y 37b). Adicionalmente, se enfatiza que en la simulación (Figura 37a) el brazo en el controlador swing up debe dar varios giros para poder estabilizar el péndulo en 0°, lo cual no sucede en la planta virtual. Lo anterior se debió a la calibración de los controladores, reflejando una mejor calibración en la planta virtual. También se señala para la Figura 37c, que en la planta de V-REP se presentaron pequeñas oscilaciones cuando se cambia la posición del brazo, las cuales se generaron por las dinámicas físicas que se contemplan en el entorno virtual.

Figura 38. Señales de respuesta del Swing up y Gain Scheduling



Fuente: Esta investigación, Grafica realizado en Matlab por Harold Fernando Ruiz y Laura María Rodríguez.

Por otro lado, las respuestas para el controlador Gain Scheduling son bastante parecidas, como se mostró en la Figura 38, teniendo en cuenta que las señales en la simulación y en la planta virtual difieren por las mismas razones explicadas anteriormente.

También, si se comparan los controladores, se observó que ambos controladores tienen buenos resultados, con la señal de control dependiendo en gran medida de la inercia que lleva el péndulo cuando se hace el cambio del controlador Swing up al controlador LQR y Gain Scheduling. En general, con el controlador con Gain Scheduling fue posible estabilizar el péndulo en rangos más amplios que con el LQR.

Finalmente, se debe comentar que realizar los controladores propuestos en esta planta en la API conectada a Simulink presentan dificultades ya que es una planta con comportamiento no lineal marcado, a la cual es necesario inicializar condiciones adecuadas para el observador de estado e integrador, operación que resulta

compleja en Simulink. Entonces, a pesar de ser posible una calibración más precisa, el proceso es complejo ya que Simulink es más sensible a pequeñas variaciones en estas condiciones como ya se observó en las plantas anteriores.

3.3.4. Planeamiento de trayectorias de un robot terrestre

Los resultados en la ejecución de cada algoritmo en los escenarios propuestos (Figuras 14 y 15) se resumen en las gráficas de las trayectorias obtenidas, junto con una tabla donde se presenta el cálculo de las distancias en metros para las trayectorias originales (línea negra), odometría (línea azul) y recorrida por el robot (línea roja). Adicionalmente, se muestra los resultados obtenidos usando un sensor tipo cámara, el cual se ejecutó solamente para el escenario trampa para los algoritmos PSO y PF.

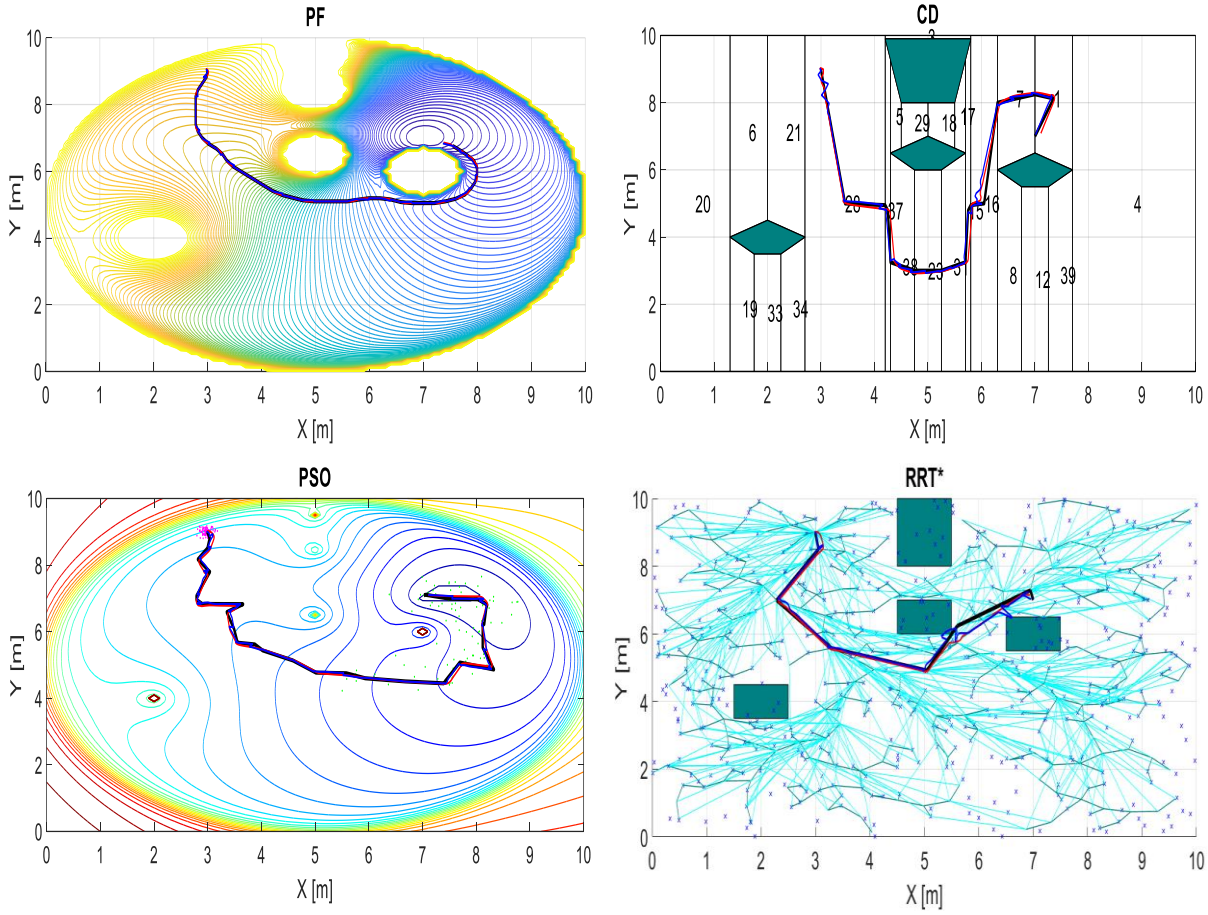
Arreglo de obstáculos

Tabla 5. Indicadores de desempeño para escenario de arreglo de obstáculos.

Método	Trayectoria original	Trayectoria odometría	Trayectoria real recorrida
PF	9.5260	10.4559	9.9492
PSO	13.2769	14.6079	13.3320
RRT	9.2366	11.5737	9.0911
CD	15.2823	16.5158	15.2898

Fuente: Esta Investigación.

Figura 39. Rutas obtenidas en escenario arreglo de obstáculos.



Fuente: Esta investigación, Grafica realizado en Matlab por Harold Fernando Ruiz y Laura María Rodríguez.

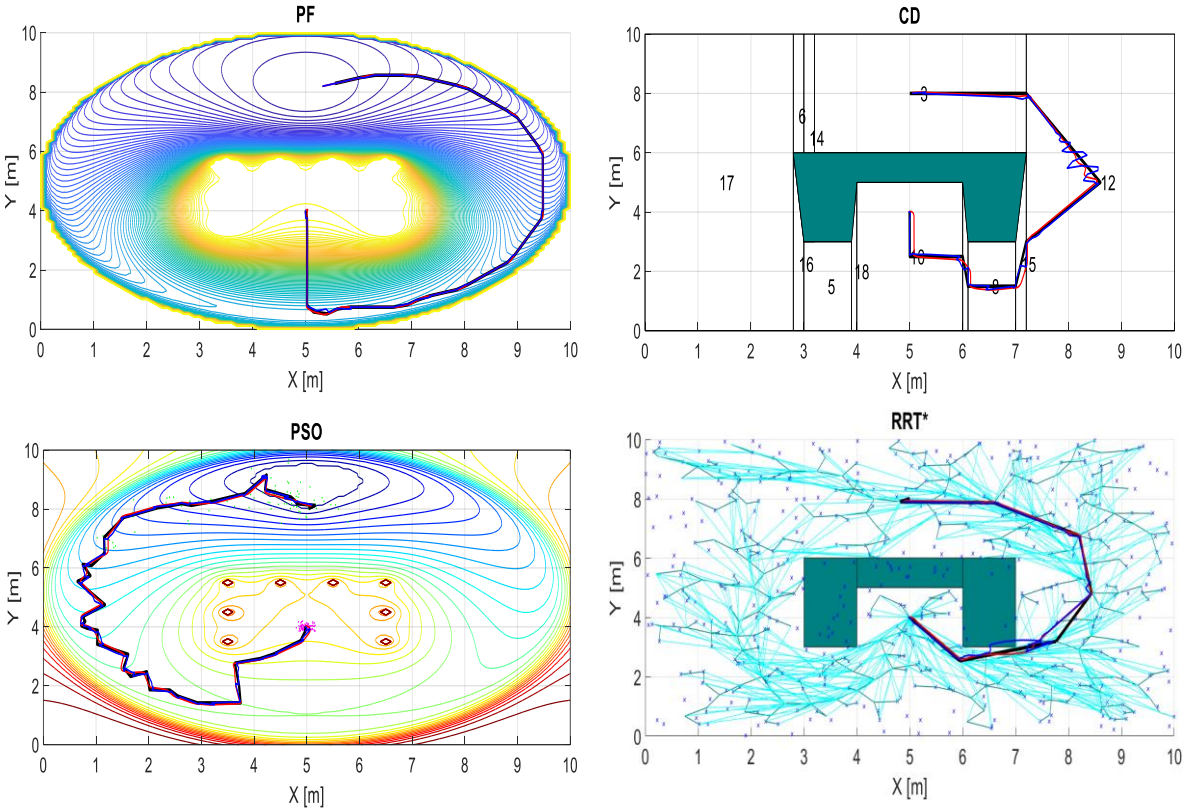
Trampa

Tabla 6. Indicadores de desempeño para escenario trampa.

Método	Trayectoria original	Trayectoria odometría	Trayectoria real recorrida
PF	*16.8144*	*17.4733*	17.2875
PSO	*18.0778*	*19.9656*	17.5971
RRT	*11.3343*	*13.0579*	11.1036
CD	*13.8700*	*16.3760*	13.8325

***Fuente*:** Esta Investigación.

Figura 40. Rutas obtenidas escenario trampa.



Fuente: Esta investigación, Grafica realizado en Matlab por Harold Fernando Ruiz y Laura María Rodríguez.

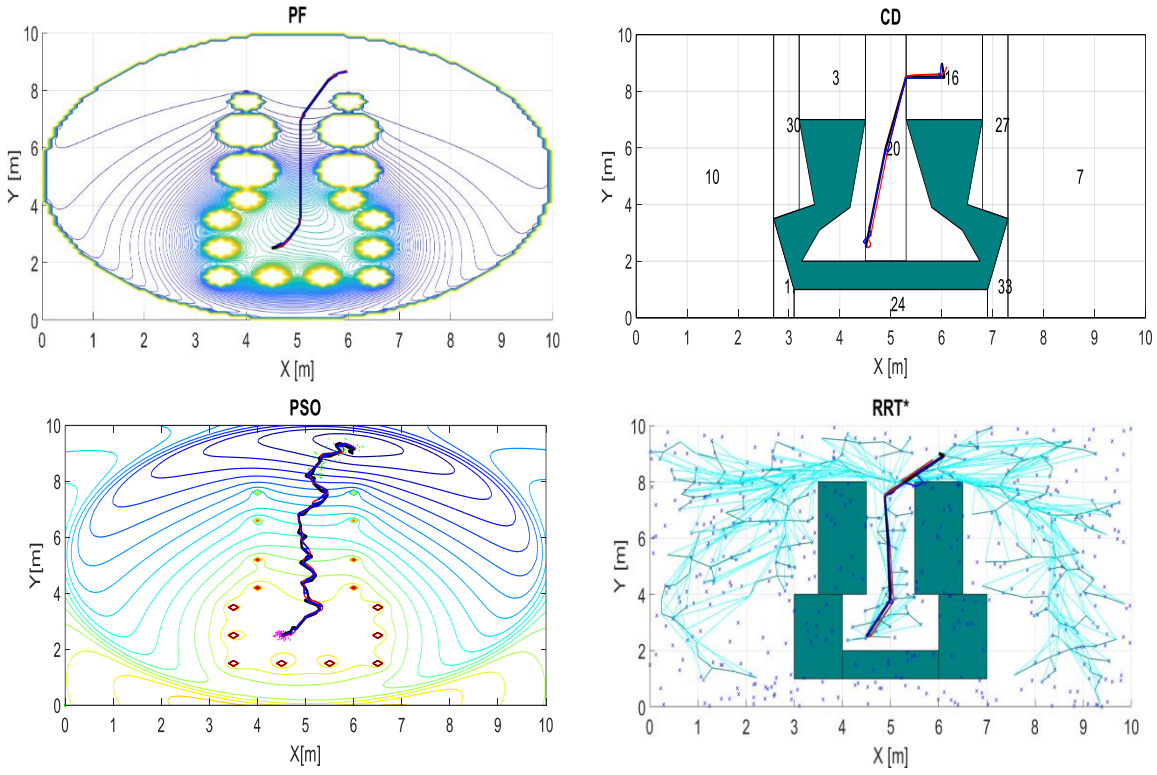
Pasaje estrecho

Tabla 7. Indicadores de desempeño para escenario pasaje estrecho.

Método	Trayectoria original	Trayectoria odometría	Trayectoria real recorrida
PF	*6.4031*	*6.6881*	6.5548
PSO	*9.7288*	*12.1630*	9.2642
RRT	*7.1198*	*7.5744*	6.8487
CD	*7.3062*	*7.4551*	7.1712

Fuente: Esta Investigación.

Figura 41. Rutas obtenidas en escenario pasaje estrecho.



Fuente: Esta investigación, Grafica realizado en Matlab por Harold Fernando Ruiz y Laura María Rodríguez.

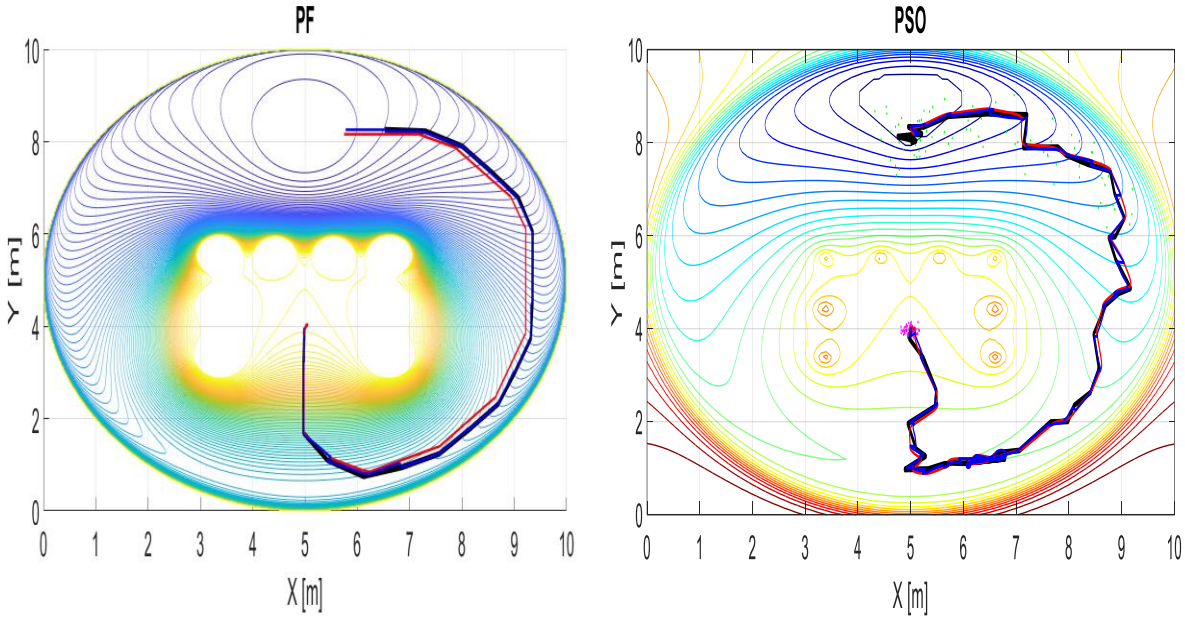
Trampa con cámara

Tabla 8. Indicadores de desempeño para escenario trampa usando la cámara.

Método	Trayectoria original	Trayectoria odometría	Trayectoria real recorrida
PF	14.6364	15.6956	15.1501
PSO	20.5396	23.4516	20.1944

Fuente: Esta Investigación.

Figura 42. Rutas obtenidas escenario trampa con cámara.



Fuente: Esta investigación, Grafica realizado en Matlab por Harold Fernando Ruiz y Laura María Rodríguez.

Análisis de resultados

Para el análisis de las trayectorias obtenidas en la sección anterior se consideran principalmente dos aspectos: i) la seguridad de la ruta (libre de colisiones) y ii) la distancia total, definida como la suma de las distancias euclidianas entre los puntos de control que definen las trayectorias (originales, odometría y real recorrida).

En ese sentido y teniendo en cuenta las Tablas 5,6 y 7 y las Figuras 39 y 40, el método RRT generó las distancias más cortas de los cuatro algoritmos probados en dos de los casos de estudio debido a la gran cantidad de ramas que se especificaron para la creación del mapa de ruta. Sin embargo, cuando es implementado en V-REP en todos los casos de estudio se presentaron colisiones debido a la cercanía de los caminos con los obstáculos (Figuras 39, 40 y 41). Estas colisiones generaron que las distancias de las trayectorias de odometría y real recorrida difieran bastante, tal como se ilustró en las Tablas (5,6 y 7). Es clave decir que a pesar de las colisiones que se presentaron, el móvil logró llegar al punto deseado, debido a que el robot al ser simulado en un entorno parecido a la realidad fue capaz de mover los obstáculos permitiendo que este pudiera llegar a la meta.

Por su parte, el método CD generó los caminos de acuerdo con el mapa trapezoidal creado (depende mucho del escenario) y en ocasiones pudo producir rutas muy cortas (Figura 41) o muy lejanas a los objetos (Figura 39 y 40). Además, si se tiene en cuenta las Figuras 39 y 40 los caminos que se produjeron pueden colisionar con los objetos como en el algoritmo RRT. Se solucionó este problema creando los trapezoides de tal forma que se consideró tanto el tamaño del obstáculo como del vehículo, lo que provocó que en ninguno de los tres casos de estudio el vehículo se estrelle con un objeto. Es importante resaltar que la solución planteada en este algoritmo se la puede aplicar al método RRT.

En el método PF se tuvieron las segundas mejores distancias recorridas en 2 de los 3 casos analizados, siendo la mejor distancia en el último caso de estudio y no presentó colisiones con los objetos. Este efecto se debió principalmente a la definición de la función de potencial, pues el camino seguido desde cualquier punto depende del descenso del gradiente. Por lo anterior una buena representación del ambiente puede suponer una ventaja frente a los algoritmos RRT y CD. Aunque en este algoritmo, si no se definen correctamente los obstáculos, se pueden llegar a presentar colisiones y afectar seriamente la seguridad de las rutas.

En el método PSO se presentaron las distancias más largas (Tablas 5,6 y 7) en 2 de los casos de estudio y en uno de los casos el robot colisionó con un objeto (Tabla 7). Este efecto se debió principalmente a la función de potencial que se usó, pues al igual que en el caso PF, el camino seguido depende del descenso del gradiente. Por lo tanto, cambiar la función de potencial a una más cercana a los

escenarios de estudio mejoraría considerablemente el camino generado por el algoritmo.

Finalmente, se observó que al usar un sensor tipo cámara (Figura 42) el escenario se transformó. Los cambios más notorios son que la posición y tamaño de algunos obstáculos cambian debido a dos factores. El primero es el error de estimación existente cuando se transformó de pixeles a metros (cambio en la posición), mientras que el segundo se generó porque la cámara toma parte de la profundidad del objeto (cambia el tamaño). Así mismo, estos errores afectaron el comportamiento de los algoritmos en los escenarios estudiados puesto que la función de potencial que se usó también cambia, lo anterior generó que la distancia total recorrida (Tabla 8) por los algoritmos PF y PSO se distancien de su homólogo de la Tabla 6. Una forma en la que se mitigó estos inconvenientes fue aumentar la resolución de la imagen (teniendo cuidado de mantener el periodo de muestreo) o directamente se cambió el ángulo de perspectiva del sensor de visión (cámara) en V-REP.

Análisis de errores generados por odometría

Teniendo en cuenta los resultados expuestos con anterioridad, se pudo destacar que la posición estimada por odometría (línea azul) y la posición real del vehículo (línea roja) presentan diferencias. Dichas discrepancias se debieron principalmente a dos factores: i) las colisiones del vehículo con algún objeto y ii) la presencia de oscilaciones en el control de movimiento del vehículo provocadas principalmente por saturación de los actuadores al moverse de un punto a otro punto muy lejano.

- **Errores de estimación debido a colisiones:** Se presentan cuando el vehículo se estrella con un objeto, por lo que las ruedas siguen en movimiento hasta poder mover el objeto con el cual se colisionó. Esto provoca que la odometría presente un error de estimación ya que según esta el vehículo se encuentra en movimiento cuando realmente está estático (por la colisión). Lo anterior se evidenció en los caminos trazados por los algoritmos RRT (Figuras 39,40 y 41), y PSO (Figura 41). Adicionalmente se destaca que dicho error provocó que las distancias recorridas y estimadas por odometría difieran bastante cómo se observó en las Tablas 5,6 y 7 para el método RRT y la Tabla 7 para PSO.

- **Errores de estimación por oscilaciones:** Se manifiestan cuando el vehículo debe moverse de un punto a otro muy lejano, generando oscilaciones dadas por la saturación de los actuadores. Lo anterior provoca que en el cálculo de la odometría se presenten oscilaciones mucho más amplias que las que realmente están ocurriendo. Dicho fenómeno se

presentó claramente en la trayectoria trazada por el algoritmo CD en la Figura 40 causando que los recorridos trazados por la odometría sean muchos más largos. Este efecto se evidenció en la Tabla 6 para el método CD, donde el vehículo no se estrella con ningún objeto, pero la distancia recorrida por el vehículo es muy diferente a la estimada por odometría. Una manera en la que se mitigó este error fue cambiando las contantes K_x y K_y por valores más pequeños, causando que el control del vehículo sea menos agresivo o en su defecto utilizar técnicas de reducción de errores sistemáticos como son *UBMmark* o *Triangular Path Test*.

4. CONCLUSIONES

Este trabajo presentó una metodología para construir plantas virtuales en V-REP y la modificación de una API remota para comunicar el entorno virtual con controladores implementados en Matlab. Adicionalmente se desarrolló una plataforma que posibilita la interacción del entorno virtual con Simulink.

Estas plataformas de co-simulacion (Matlab y Simulink) facilitaron el proceso de enseñanza-aprendizaje en las asignaturas relacionadas, permitiendo que las instituciones educativas utilicen diferentes plantas con la mayoría de las interacciones reales que generalmente no se presentan en las simulaciones con modelos matemáticos.

Para mostrar la aplicación del método, se realizó la comparación de diferentes técnicas de control (al menos dos por planta) en cuatro plantas mecánicas. Se resalta la implementación exitosa de un controlador LQR y un MPC con observador de estados para el sistema bola-viga y un controlador PID y un LQR con observador de estados en el sistema bola plato, donde se presentaron varios inconvenientes como la calibración de los parámetros de los controladores, las oscilaciones en las señales de entrada y salida de los sistemas y errores en la estimación de los estados de la planta propios de un ambiente real. Adicionalmente en estas dos plantas se presentaron algunas dificultades a la hora de elaborar los controladores en Simulink conectado a V-REP, ya que dicha plataforma fue mucho más sensible a las condiciones iniciales y a los cambios bruscos en las referencias que su contraparte en Matlab lo que generó mayores oscilaciones en las señales de control. Lo anterior fue causado principalmente por los parámetros de configuración que se tenga en Simulink (esencialmente el *solve*) y al desbordamiento del intervalo de muestreo (step size) generado primordialmente por una demora en él envió de datos desde la API de V-REP.

En el péndulo de Furuta se llevó acabo un controlador LQR, un observador de estados, un Gain schaling y un Swing up, donde se evidenciaron dificultades en la calibración de los parámetros (especialmente en el swing up), en la transformación de las señales de control (torque a velocidad) y en la implementación de los controladores en Simulink. En esta última plataforma no fue posible controlar la planta debido a la inestabilidad que generaban los observadores lineales y los integradores discretos.

Asimismo, en la última planta virtual se presentó con éxito la implementación de los cuatro métodos de planeación de trayectorias en seis escenarios construidos en V-REP, donde se presentaron imprevistos propios de plantas reales como las colisiones con los objetos, oscilaciones en las trayectorias y errores en la odometría propios de un entorno real.

Finalmente, se resalta que parte del trabajo descrito en este documento se presentó en el congreso nacional “IEEE COLOMBIAN CONFERENCE ON AUTOMATIC CONTROL” y en la conferencia internacional “LATIN AMERICAN CONGRESS ON AUTOMATION AND ROBOTICS”, cuyos documentos publicados se encuentran en los Anexos 2 y 3.

5. TRABAJO FUTURO

Teniendo en cuenta los resultados obtenidos con las plantas virtuales implementadas, donde la mayoría de señales de control son las velocidades angulares de los motores, se podría realizar una función (tanto en Matlab como en Simulink) que posibilite ingresar señales de voltajes a los actuadores de esta forma aumentar el realismo de las plantas desarrolladas en este trabajo.

En el planeamiento de trayectorias de un robot terrestre se puede explorar la implementación de otros algoritmos como los basados en control distribuido (teoría de juegos) para realizar trayectorias en un entorno con varios robots.

Finalmente, se propone realizar controladores y observadores más robustos que posibiliten el control de la planta virtual péndulo de Furuta desde Simulink. Dichos controladores deben solucionar los problemas de observabilidad e inestabilidad generados por los observadores lineales y los integradores discretos.

BIBLIOGRAFIA

VARGAS, H; DORMIDO, D; Duro, N; DORMIDO-CANTO, D. Creación de laboratorios virtuales y remotos usando easy java simulations y LABVIEW: El sistema heatflow como un caso de estudio. En: XXVII Jornadas de Automática, 2006, p. 1182-1188.

International Inclusive Science, Technology, Engineering, and Mathematics Education. Low-Cost Engineering Laboratory Project. {En linea}. {10 de julio del 2018}. Disponible en: (https://istem.engineering.osu.edu/university-experiments/welab-low-cost-engineering-laboratory-project).

QUANSER, PRODUCTS & LAB SOLUTIONS. {En linea}. {15 de Julio del 2018}. Disponible en: (https://www.quanser.com/).

ANDALUZ, V.H; CHICAIZA, F.A; GALLARDO, C; QUEVEDO, W.X; VARELA, J; SÁNCHEZ, J.S; ARTEAGA, W.X. Unity3DMatLab Simulator in Real Time for Robotics. En: Proceedings of the third 3rd AVR international conference on Augmented Reality, Virtual Reality and Computer Graphics. AVR, 2016.p.246-263.

VÁSQUEZ, R.D; SARMIENTO, H.O; MUÑOZ, D.S. Propuesta de implementación de plantas virtuales para la enseñanza de programas de control lógico. En: ACOFI, 2016, v.11, n.22, pp. 1-13.

DOMINGUEZ, A. Modelado y Simulación de un Robot LEGO Mindstorms EV3 mediante V-REP y Matlab. Trabajo de grado (Ingeniero de sistemas). Málaga, 2016. Universidad de Málaga. Departamento de Ingeniería de Sistemas y Automática.

FABREGRAS, E; DORMIDO-CANTO, S; DORMIDO, S. Virtual and Remote Laboratory with the Ball and Plate System. En: IFAC-PapersOnLine, 2017, v.50, n.1, pp. 9132-9137.

SERGIO, J. Control de Posición y Visual de Sistemas de Manipulación Autónomos. Trabajo de grado (Master Universitario en Automática y Robótica). Alicante, 2017. Universidad de Alicante. Departamento de Ingeniería de Sistemas.

The Player Project. Player Project. {En linea}. {15 de marzo del 2018}. Disponible en: (http://playerstage.sourceforge.net).

Microsoft. Welcome to Robotics Developer Studio. {En linea}. {15 de marzo del 2018}. Disponible en: (https://msdn.microsoft.com/en-us/library/bb648760.aspx).

anyKode. Marilou: the universal mechatronic software. {En linea}. {15 de marzo del 2018}. Disponible en: (http://www.anykode.com/marilou.php).

Coppelia Roboticts V-REP. V-REP User Manual. {En linea}. {10 de enero del 2018}. Disponible en: (http://www.coppeliarobotics.com/helpFiles/).

Coppelia Roboticts V-REP. V-REP Regular API Function List. {En línea}. {10 de enero del 2018}. Disponible en: (http://www.coppeliarobotics.com/helpFiles/).

Coppelia Roboticts V-REP. V-REP shapes. {En línea}. {30 de abril del 2018}. Disponible en: (http://www.coppeliarobotics.com/helpFiles/en/shapes.htm).

Coppelia Roboticts V-REP. V-REP joint description. {En línea}. {3 de marzo del 2018}. Disponible en: (http://www.coppeliarobotics.com/helpFiles/en/jointDescription.htm).

Coppelia Roboticts V-REP. V-REP object description. {En línea}. {3 de marzo del 2018}. Disponible en: (http://www.coppeliarobotics.com/helpFiles/en/objects.htm).

Open Source Robotics Fundation. ROS.org | Powering the world's robots. {En línea}. {30 de marzo del 2019}. Disponible en: (http://www.ros.org/).

MatWorks. Templates for C S-functions. {En línea}. {7 de septiembre del 2019}. Disponible en: (https://www.mathworks.com/help/simulink/sfg/templates-for-c-s-functions.html).

MatWorks. Create a Basic C Mex S-Function. {En línea}. {12 de septiembre del 2019}. Disponible en: (MatWorks, Create a Basic C Mex S-Function, disponible en: (https://www.mathworks.com/help/simulink/sfg/example-of-a-basic-c-mex-s-function.html).

San Diego State University. Mex S-Funtion Wrapper. {En línea}. {12 de septiembre del 2019}. Disponible en: (MatWorks, Create a Basic C Mex S-Function, disponible en: (https://edoras.sdsu.edu/doc/matlab/toolbox/simulink/sfg/sfcn_rtw8.html).

MathWorks. Creating a Mask: Parameters and Dialog Pane. {En línea}. {15 de septiembre del 2019}. Disponible en: (https://www.mathworks.com/videos/creating-a-mask-parameters-and-dialog-pane-120557.html).

Solidworks Corporation. Guía del estudiante para el aprendizaje del software Soidworks". {En línea}. {10 de febrero del 2018}. Disponible en: (https://www.solidworks.com/sw/docs/Student_WB_2011_ESP.pdf).

HUANG, J; LIN, C. Robust nonlinear control of the ball and beam system. En: Proceedings of of the American Control Conference. ACC, 1995.p.306-310.

OGATA.K. Sistemas de control en tiempo discreto. Segunda edición. Prentice Hall, 1995, p. 506-609.

WANG.L. Model predictive control system design and implementation using MATLAB®. Springer, 1995.

CEDEÑO, A; GORDÓN, M; MORALES, L. Control de posición y seguimiento de caminos en el sistema Bola-Plataforma. En: XXVII Jornadas de Automática, 2017, vol.27, no 5, p.47-53.

RyogeiGoton. inverted_pendulum. {En linea}. {4 de octubre del 2018}. Disponible en: (https://github.com/RyoheiGoto/inverted_pendulum/tree/master/simulator/vrep).

OSORIO, C. Diseño, construcción y control de un péndulo invertido rotacional utilizando técnicas lineales y no lineales. Trabajo de grado (Master en automatización industrial). Bogotá, 2009. Universidad de Nacional de Colombia. Facultad de ingeniería. Departamento de Ingeniería de Sistemas.

ASTRÖM.K; WITTENMARK.B. Adaptive control. Second edition. Dover publications, 1995.

GETIAL, J; ERAZO, E; PANTOJA, A. Quantitative Comparison of Path Planning Methods in Mobile Robotics. En: Proceedings of the 3rd Colombian Conference on Automatic Control. CCAC, 2017.

TZAFESTAS.S. Introduce to Mobile robot control. Elsevier, 2013.

ALAM, M; RAFIQUE, M; KAHN, M. Mobile Robot Path Planning in Static Environments using Particle Swarm Optimization. En: International Journal of Computer Science and Electronics Engineering(IJCSEE), 2015, v.3, no.3, pp. 253-257.

BERG.M; CHEONG.O; OVERMARS.M. Robot motion planning Third Edition. DE: Springer, 2000, Ch. 13, p. 283-306.

GETIAL, J; ERAZO, E; ANDRES, P. A Quantitative Comparison of Path Planning Methods in Mobile Robotics. {En línea}. {11 de marzo del 2019}. Disponible en: (https://sites.google.com/view/pathplanningthese/simulations/unique-robot?fbclid=IwAR0cGIQz8wiUhLYtlOZYLX1I4n139ryej7nGzbK7hSwWsZk1gTMWknMtxVsl).

PASSINO, K. Vehicule guidance for obstacle avoidance example. {En línea}. {29 de marzo del 2018}. Disponible en: (http://www2.ece.ohio-state.edu/~passino/ICbook/Code/vehic_guide.m).

ANEXOS

ANEXO 1. **Matlab y Simulink**

a) Sistema bola viga en V-REP

Videos donde se explica la construcción, diseño e implementación física de un sistema bola viga en el software de simulación 3D V-REP, además de detallar el proceso para realizar la comunicación V-REP-Matlab, disponible en: https://www.youtube.com/watch?v=7zkXhFoC6QA&list=PLwUJXo5BTmo6TYU_7-sGsRAFCKCiQaqei&index=1

b) Controlador LQR y observador de estados en V-REP

Videos donde se muestra como hacer un controlador LQR y un observador de estados discretos para un sistema Bola-viga el cual es implementado en V-REP, disponible en: https://www.youtube.com/watch?v=BKbIOneiM10&list=PLwUJXo5BTmo57BPEH6D15lR5uBzM3vpYR

c) Tutorial robot con tracción diferencial y planeamiento de trayectorias de un robot con configuración diferencial en V-REP

En estos videos se detalla el control de posición de un robot diferencial en el entorno de simulación V-REP, resaltando que la posición del vehículo se la obtiene mediante odometría. Adicionalmente se desarrolló un planeamiento de trayectorias para el robot Pioneer_P3dx en V-REP usando el algoritmo de Kevin Passino, destacando que la posición del vehículo se la obtiene mediante odometría, disponible en: https://www.youtube.com/watch?v=TTJT9G9dMOo&list=PLwUJXo5BTmo4xGWdcG_hGT7JK9umATWa8

d) Sistema bola-plato en V-REP (construcción en V-REP y comunicación con Matlab)

Videos explicativos en la construcción física y manejo de la API remota V-REP-Matlab en un sistema bola plato, disponible en: https://www.youtube.com/watch?v=y_tvGiKxdX8&list=PLwUJXo5BTmo75Q24r1t4jTvCSYMG_zIco

e) Transformación de las funciones de la API remota V-REP-Matlab

Video donde se explica cómo cambiar las funciones de la API remota de V-REP para que estén en español y tengan menos parámetros, disponible en: https://www.youtube.com/watch?v=rvUkWKBLTNQ&t=144s

f) Manejo e instalación de la plataforma que comunica V-REP con Simulink

Video donde se detalla el procedimiento para poder realizar la comunicación entre los programas V-REP y Simulink, destacando que se dan algunas aclaraciones generales del funcionamiento del bloque, disponible en: https://youtu.be/QrxTMsWVJTE

g) Repositorio de código

Repositorio de códigos y archivos de V-REP de importancia usados en este documente, disponible en: https://github.com/Laura-M-Rod/V-Rep-Matlab

ANEXO 2. **ARTÍCULO:** DEVELOPMENT AND CONTROL OF VIRTUAL PLANTS IN A CO-SIMULATION ENVIRONMENT

Este anexo contiene el artículo seleccionado para sustentación en la modalidad de ponencia oral y publicación en la conferencia IEEE COLOMBIAN CONFERENCE ON AUTOMATIC CONTROL celebrada en Medellín-Colombia del 15 al 18 de octubre de 2019.

2019 IEEE 4th Colombian Conference on Automatic Control (CCAC)
October 15-18, 2019. Diez Hotel, Medellin, Colombia.

Development and Control of Virtual Plants in a Co-Simulation Environment

Harold Fernando Ruiz Bravo
Departamento de Electrónica
Universidad de Nariño
Pasto,Colombia
harolfernandoruiz@hotmail.com

Laura María Rodríguez Rivera
Departamento de Electrónica
Universidad de Nariño
Pasto,Colombia
lauritamaria_rod@hotmail.com

Andrés Pantoja Buchelli
Departamento de Electrónica
Universidad de Nariño
Pasto,Colombia
ad_pantoja@udenar.edu.co

John Barco
Facultad de Ingeniería
UNICESMAG
Pasto,Colombia
jebarco@iucesmag.edu.co

***Abstract*— Virtual plants represent one of the most versatile instruments in the teaching-learning process of control courses since they may replace expensive laboratories or real plants. This work proposes a methodology to use the 3D simulator V-REP and Matlab for the construction, communication, and control of virtual plants in a co-simulation environment. The implementation of the ball-and-plate system shows the control possibilities and the differences by working with ideal-model simulations and using specialized platforms to include unmodeled physical phenomena. Using two simple controllers, the results show how the real-based experiments produce unexpected and challenging responses that should be corrected with distinct control strategies.**

Keywords*— Co-simulation, Virtual plants, V-REP, ball-and-plate.

I. INTRODUCTION

A virtual plant is a software-based environment, where users interact on a series of graphical components that represent elements of a physical model. These basic elements are modeled in a specialized software platform forming more complex mechanisms, in such a way, that the system closely emulates the actual plant performance. Some examples of virtual plants include mechanical systems working on a simulation and remote laboratories, where the plants are real (and expensive) but operated through internet by diverse users [1].

On the other hand, these environments have had an impact in many knowledge areas and engineering research, making the most of the advance of 3D simulation-based technologies and the relevance they have had in pedagogy and ludic activities [2] [3]. This penetration has facilitated the improvement of many 3D and 2D simulation environments for educational purposes, which allow users to consolidate concepts and merge theory with practice in a simple and accessible way.

One of the most widespread platforms in this type of applications is V-REP, whose features support simple and complex forms (e.g., different types of robots), as well as the inclusion of 3D models from an external CAD software. Moreover, V-REP has an affinity with multiple programming languages, including Matlab.

In the field of educational software, especially for technical areas, this work intends to solve the lack of expensive laboratory equipment and specialized didactic prototypes in most of education institutions, through using the emerging technologies above mentioned. An interactive platform integrating a plant simulator and a control software facilitates the joint of theoretical and practical knowledge, avoiding high investments in equipment and allowing the study of different kind of control plants.

Several works have focused on the construction of virtual plants. For instance, the authors in [4] present a 3D virtual reality simulator for the development of controllers to follow trajectories with a 6-DOF robotic arm. In [5], it is shown the implementation of a didactic tool that allows simulating, observing, and analyzing the behavior of industrial processes, emphasizing on the communication with a real PLC. Using the main features of the V-REP environment, the proposal in [6] shows the construction, design, and simulation of a LEGO EV3 robot in the V-REP 3D simulator. Moreover, for a specific ball-and-plate system, the authors in [10] propose a joint application with a virtual and a remote laboratory used in a graduate program in control engineering, using a Java-based platform.

This work describes the use of a co-simulation between a free environment of virtual plants (V-REP) and a common language for the study of control (Matlab), working in a platform communicated by an API (application programming interface). To show the methodology, a ball-and-plate system is presented. It should be noted that the contribution of this work does not focus on the development of advanced controllers for a particular plant but the proposal of a pedagogical methodology taking advantage of virtual plants. The method can be adapted for the implementation of different plants emulating real behavior in contrast to standard model-based simulations.

To analyze the differences between the simulation results and the ones obtained in the example plant, two controllers (i.e., PID and LQR) are tuned to track simple trajectories of the ball. The comparison shows the gap between a real controller and an ideal simulation since the virtual plant emulates physical properties as friction, displacements, and slides, which are close to reality. This condition allows students to face real conditions that are not obtained in a simulation (e.g., noise, disturbances, actuator limits, and physical barriers), improving theoretical-practical training and the actual application of control concepts.

The rest of the work is organized as follows: Section II presents the virtual environment and the co-simulation scheme, while Section III describes the development of the virtual plant. Section IV explains the results obtained, taking into account the differences of controllers within the simulation and the real plants. Finally, Section V contains the conclusions and future work.

978-1-5386-6962-4/19/$31.00 ©2019 IEEE

II. Virtual Environment and Co-Simulation Scheme

The implementation of a virtual plant in V-REP presents two important processes. The first one is the "physical" construction of the plant taking into account the properties and limitations of the emulation software considering the different objects, shapes, sensors, and actuators in the plant's mechanics. The second one involves the communication between the objects within the virtual plant and an external program by means of a remote API (Application Programming Interface).

A. V-REP Generalities

V-REP is a program with a free educational version that is used as a tool to test and develop virtual robots and mechanical systems. It is characterized by having extensive documentation, allowing importation of 3D models, and having compatibility with multiple programming languages, including Matlab.

The software has a specialized processing engine for advanced kinematic and dynamic calculations that allows the simulation of real environments and physical parts (e.g., solid pieces, motors, and sensors, just to mention a few) in its graphical interface. The interaction with other software platforms is carried out through more than 400 functions in the native API of the simulation environment, whose documentation is detailed in [6] and [7]. It is worth remarking that the simulator is capable of emulating real physical behaviors such as those produced by gravity, inertia slips of parts, possible external disturbances, as well as collisions and frictions between the different objects found in the workspace.

Physical components in V-REP are articulated with each other by joints that are classified according to the type (rotational or translational) or the operation way (passive, inverse kinematics, object dependent, motion, or torque-force). The joints are moved by motors, representing the main actuators of the plants. Furthermore, there are a high variety of sensors to complete the feedback loop in the control schemes [7].

In terms of simulation time, V-REP presents restrictions on the minimum sampling time that can be achieved (between 1 and 200 milliseconds). The sampling period depends mainly on the computer capacities, the complexity of the plant (number of simple forms, actuators, and sensors), and the use of external servers such as the remote API for interconnection with other programs.

B. V-REP Remote API

The API is used to control a simulation (read/write properties and attributes of components, actuators and sensors) from external hardware or software into V-REP. It consists of more than one hundred functions, which can be called from a C++ application, Python, Java, or a MATLAB [6], [7] script. The functions interact with V-REP through Blue Zero Middleware [8] and its interface plugin [7].

The API is provided by the V-REP developers, and allows the users to control the simulation environment of the platform using an external application, or even, a remote computer. The API offers a list of useful functions, which can be called by a routine (implemented in the aforementioned programming languages). The communication with V-REP is made through sockets, where one or more external applications interact with the simulator in synchronous or asynchronous mode, using a client-server structure. The external application is the client and the simulation environment in V-REP is the server.

The server can operate in two modes: continuous operation and temporary operation. In continuous operation, after the computer initialization, the API plug-in checks the "remoteApiConnections" file available in the V-REP root installation folder, and activates the server on the specified communication port. In this mode, the server remains available even when the simulation is not running, and the software can be controlled externally (start, stop and pause commands), using the appropriate API functions.

In the case of a co-simulation structure, the client (developed in Matlab) must be synchronized with the progress of the simulation in V-REP. In this mode, each step of the simulation (i.e., a sampling period), determines a new calculation cycle in V-REP based on the data sent to the actuators and collects the data provided by the sensors. In this way, the communication emulates the close loop between the controller and the plant by the synchronization of both programs.

Specifically, the co-simulation is performed by executing functions or commands in Matlab, that through the API, are executed in V-REP. A large number of commands provides the versatility that allows Matlab to obtain all the information from the plant objects, as well as to set certain parameters on actuators (e.g., motor's speeds or positions).

However, the original functions of the API are quite complex given the large number of parameters they need to use. For this reason, we change the remote script to provide an easy library to facilitate the link between the two platforms. To do this, the "remotApi" script is modified, as well as the functions of bidirectional transmission of information. Consequently, the commands used in Matlab are simpler to use and have only the necessary parameters, facilitating the execution and configuration of the co-simulation.

Being the script "remotApi" the intermediary between Matlab and V-REP, it is responsible for interpreting the functions executed in Matlab, and it sends the information to the simulation in V-REP. Some of the changes we propose in the main functions of the API include shortening the names of the parameters and integrating some of them as static statements within the code to avoid their repetitive use. For instance, this process is implemented for the identification of the connection port, since it is a parameter that usually is not changed along the simulation and it is used in most of the API instructions. Moreover, given the high information exchange that involves the use of these two software platforms it is not recommended to run more than one server and client at a time.

III. Case Study: Ball-and-Plate System

To illustrate the process of implementing virtual plants and their control through Matlab, we present the development of a ball-and-plate experiment. The plant consists of a surface that pivots on a central articulation moved by two servomotors that control the rotation angle on the x and y coordinates. The complete system has two degrees of freedom and the measurement of the position of the ball on the surface can be sensed by a camera or directly with the properties of the location

of the sphere in the V-REP space. The position of the ball can be measured using the position property of the ball provided by the V-REP API.

A. Ball-and-Plate Mechanical Design in V-REP

To establish the structure of the plant, we use the so-called "pure forms" in V-REP, which are simple solids such as bars, plates and spheres. Each element can be defined with properties such as dimensions, mass, and position, characteristics that determine the operation within the 3D scheme. Once all the necessary elements for the construction of the system have been defined, the different pieces are put together by means of the "group select shapes" option, which joins the different shapes in a set, as the one shown in Figure 1. In this initial state, the main pivot is embedded on a rotational articulation in the middle of the table. In the same way, the arms articulated to the servomotors are joined to control the movement of the surface.

As the original shapes of V-REP are not asthetic and represent basic shapes, for the emulation of a real scheme it is necessary to import models designed in CAD tools such as Solidworks [9], [7]. To do this, the files are imported in obj, dxf, or stl formats that contain basic, but clean shapes. It is worth noting that if complex figures are used, the simulation becomes slow and the sampling times in the co-simulation increase considerably.

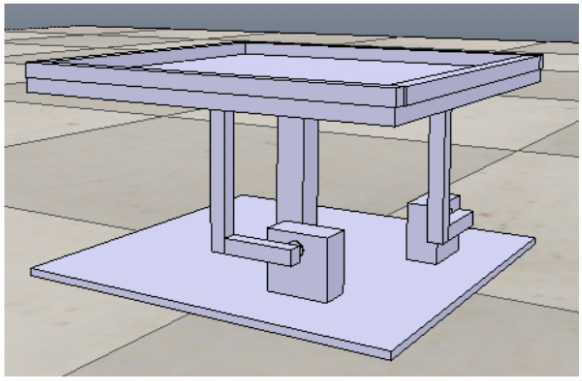


Fig. 1. Initial structural design of the ball-and-plate system in the main environment of V-REP.

The imported pieces are grouped into a tree structure so that the simulator associates each basic form to its properties in the native language. In addition, the couplings between all elements are indispensable, because if the shapes are not properly joined, the pieces fall to the floor when the simulation starts (similar to solids in a real environment). The grouping is done hierarchically by taking a "father" and associating a "son" through a simple menu. Once all the pure forms of V-REP are associated with each other and with the models imported from the CAD software, the original solids of V-REP become "invisible", producing a polished scheme as the one shown in Figure 2.

Regarding the actuators, the plant has two motors that move two arms joined to the plate with rotational joints located at the same axial distance. The actuators are assumed to be symmetrical for motion in each of the axes as illustrated in Figure 2. The joints are configured in torque-force mode, while the motors are selected in the angular-position mode to enable the servo function (i.e., the input of each motor is the angular position). To capture the position of the ball with a real sensor, V-REP can use a fixed camera, whose image is exported to Matlab as a bitmap that can be processed to estimate the position of the ball in the plate. Another way to obtain the ball's position is by consulting that property directly from the API.

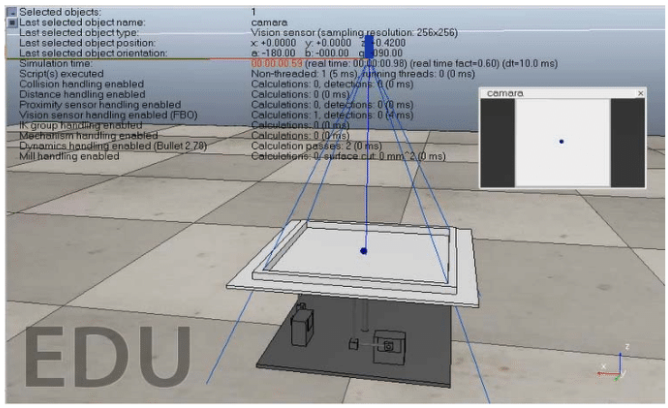


Fig. 2. Complete Ball-Plate System.

B. Estimation of ball position by image processing

The image process to obtain the position of the sphere must be implemented in such a way that all the algorithm (including the control calculation) is executed in a time less than the selected sampling time. For this purpose, appropriate characteristics of the camera in V-REP, such as resolution, perspective angle and region of interest, are set. The color of the system is fixed in such a way that the contrast between the plate (white) and the ball (blue) is high, which facilitates the implementation of image processing algorithms.

The initial image received in Matlab through the API is a RGB picture with a size of 256x256 pixels, corresponding to the internal size of the plate (22.5x22.5 cm). To process the image, the picture is transformed to grayscale, binarized and inverted. Based on the change of values in the pixels adjacent to the sphere, the centroid of the sphere is recognized. Additionally, the pixels of the centroid of the image (Figure 3) are transformed to centimeters taking as reference the scale used by the API.

Fig. 3. Resulting picture and position of the ball from the image processing routine.

C. Ball-and-plate Controllers Design

1) Plant Model

Assuming complete symmetry in the system and that the ball does not slide and is always in contact with the surface, a non-linear model describing the behavior of the plant is given by [12]

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} x_2 \\ a\left(x_1 \dot{U_x}^2 + x_3 \dot{U_x} \dot{U_y} - g\ sin(U_x)\right) \\ x_4 \\ a\left(x_3 \dot{U_y}^2 + x_1 \dot{U_x} \dot{U_y} - g\ sin(U_y)\right) \end{bmatrix}, \quad (1)$$

where x_1 and x_3 are the position and velocity along the x-axis, respectively, while x_2 and x_4 are the same variables on the y-axis. The inputs U_x and U_y are the angular positions of the servomotor axes, the value of $a = 5/7$, for this case, meets the physical characteristics of the plate (dimensions, weight, inertia), and $g = 9.8\ m/s^2$ is the gravity.

To design a lineal control, the linearization of the system at the origin point (0,0,0,0) results in

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} B = \begin{bmatrix} 0 & 0 \\ -ag & 0 \\ 0 & 0 \\ 0 & -ag \end{bmatrix}, \quad (2)$$

where the same separability characteristic of the original MIMO model is appreciated. Notice that the system can be split in two identical SISO systems of order two. The system outputs are the ball positions on the x-axis (x_1) and on the y-axis (x_3).

2) Discrete PID Controller

As an initial point of comparison, a discrete PID controller is designed using two independent loops (one for each motor). The tuning is performed in a single subsystem and replicated for the second joint given the assumed symmetry in the plant. The standard discrete control structure is defined as

$$U(z) = \left[k_P + \frac{k_I}{1 - z^{-1}} + k_D(1 - z^{-1})\right] E(z), \quad (3)$$

where k_P , k_I y k_D are the discrete proportional, integral and derivative constants, respectively, $U(z)$ is the control signal and $E(z)$ the error of the output against the desired reference. The controller parameters are obtained for low overshoot and a short set-up time, relative to the trajectory time in the simulation results. The final values are $k_P = -0.4626$, $k_I = -0.0029$, and $k_D = -5.3064$.

3) Discrete LQR Controller

A discrete lineal quadratic regulator is characterized by being an optimal, robust, and easy to implement strategy with state feedback. Its operation is based on minimizing the function

$$J = \sum_{k=0}^{\infty} (x_k^T Q\, x_k + u_k^T R u_k), \quad (4)$$

where k is the index of discrete time, Q is a symmetric definite positive matrix that establishes the priorities in the states, while R establishes the importance of the control signal (energy expended) to obtain the optimization objective. To avoid steady-state error, we add an integrator for each one of the outputs, increasing the total order of the system by two. However, for the separate plant design, each subsystem with an additional integrator is chosen and the three feedback gains are calculated (two original states plus the integral error as the new state), as it is proposed in [11].

Based on simulation calibrations, the values used for the calculation of the feedback constants are presented in Table 1. All state variables are assumed measured using the properties from the remote API.

TABLE I. LQR PARAMETERS AND CONSTANTS

LQR constant and input parameters	Value
Q	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 500 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$
R	100
K	[-0.5779 -1.1352 -0.0150]

IV. RESULTS AND DISCUSSION

The results over the implemented plant are analyzed by comparing the response of systems to the designed controllers in two scenarios: *i)* using ideal models in a Simulink simulation, *ii)* with the interaction of the controllers in Matlab regulating the ball position in the designed virtual plant through the co-simulation (with the position of the ball obtained directly from the API, and *iii)* using the virtual plant in co-simulation but using a camera to get the ball position.

A. Analysis of the ball-and-plate system.

To test this plant, we propose to follow a simple trajectory, starting in rest at position (0,0) cm, then changing setpoint every 40 seconds to (-5,-5), (-5,5), (5,5), (5,-5) and (-5,-5) centimeters.

The responses for the LQR controller are shown in Figures 4, 5 and 6, detailing the position of the ball in time, the control signals to the actuators, and the trajectory in the plate, respectively. Note that the position in every axis is achieved without steady-state error due to the additional integrator in the design. Moreover, the differences on the simulation and the virtual plant (with and without camera) are remarkable in all the results.

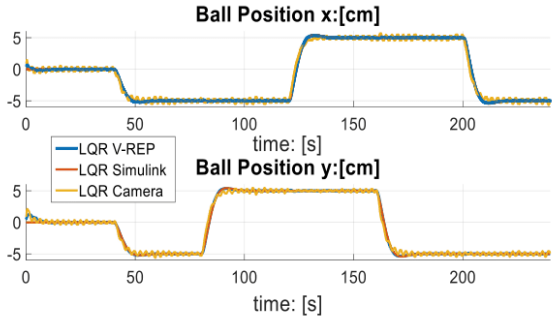


Fig. 4. Ball position response using a LQR for the three study cases.

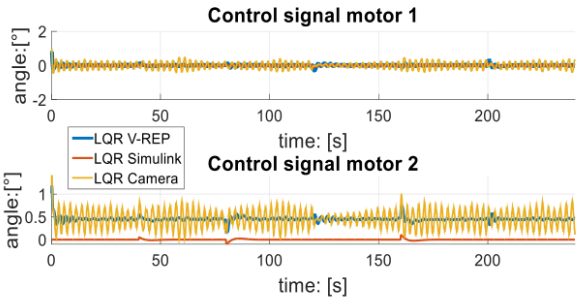


Fig. 5. Motors control signal using a LQR.

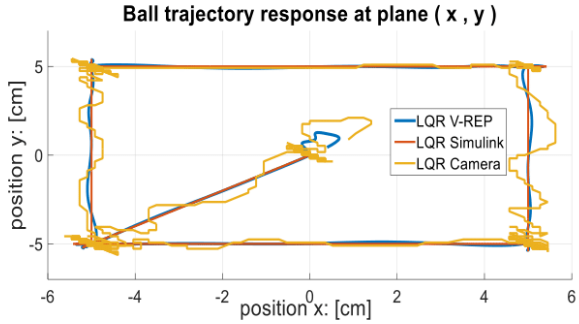


Fig. 6. Ball trajectory response using a discrete LQR.

From Figure 6, it is noticeable that the control signal of the motor 2 in the virtual plant differs clearly from the simulation. This response is because the plate in the virtual plant is not perfectly aligned to the origin, and the controller tries to compensate this lag by remaining at a constant value to equilibrate the table in a horizontal position. This lag causes the ball not to start at (0,0) cm (Figure 6), since it tends to slide to the positive side of the plane (x, y). However, despite the parameters contemplated by the virtual environment (friction, gravity, and contact between the ball and the plate) the general behavior of the virtual plant in V-REP is similar to simulation in Simulink. Additionally, it is emphasized that the signals obtained through the use of the camera (especially in Figures 5 and 6) present more oscillations due to the existing error in the conversion of pixels by the low resolution of the image. Moreover, it is also pointed out that the initial position and the control signals present also a small shift, while the oscillations increase given that the controller tries always to correct the error presented in the position (Figure 6).

On the other hand, the responses for the discrete PID controller, are quite different as it is shown in Figures 7, 8 and 9. Although in Figure 8 the simulation shows smooth changes in the setpoints, the actual response in the virtual plant presents overshoots and oscillations. This behavior is more evident in Figure 9, where the trajectory followed by the real plant is completely distinct to the one presented in simulation.

Figures 7 and 9 point out that the aforementioned initial displacement of the table is also presented, since the ball does not start at the origin. The compensation of the controller is presented in Figure 9, where the control signal for the virtual plant is higher than the one in the LQR controller (achieving peaks between 30 and -30 degrees). The PID controller, in this case, is more aggressive than the LQR, and then, the friction and slide of the ball in the virtual environment cause large differences with the simulation. Also, these differences are more remarkable in this case due to the low robustness of the PID-controller design. In addition, the performance of the controller using the camera (Figure 9) is worse due to the conversion error mentioned above.

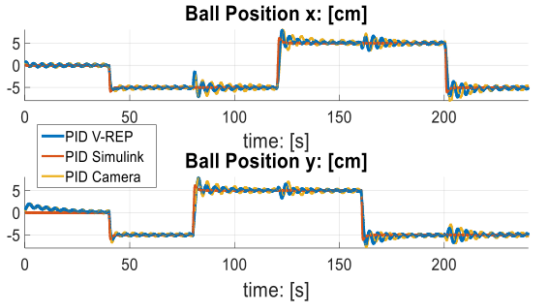


Fig. 7. Ball position response using a discrete PID.

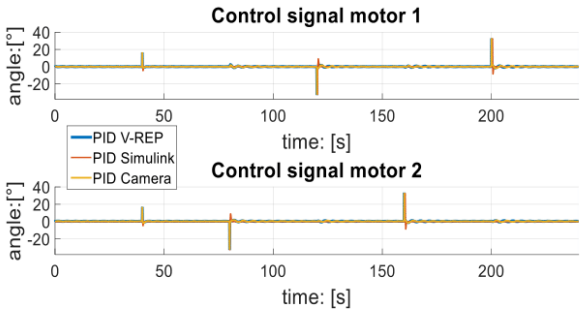


Fig. 8. Motors control signal using a PID.

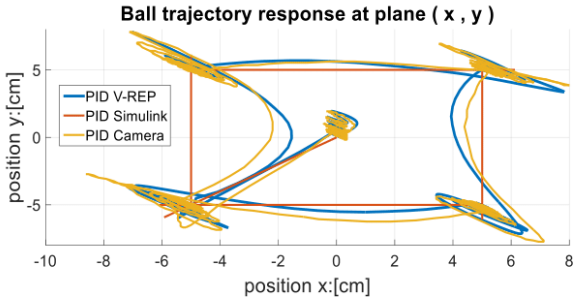


Fig. 9. Ball trajectory response using a discrete PID.

Comparing the controllers, the LQR controller presents a higher performance in robustness, set time, control signal and steady-state error over the PID controller. The oscillations in the PID controller produce a longer settling time, while the abrupt changes in the control signal allow the ball to slide, increasing the errors.

V. Conclusons and Future Work

This work presents a methodology to construct virtual plants in V-REP and the modification of a remote API to communicate the real environment with controllers implemented in Matlab. This co-simulation platform facilitates the teaching-learning

process in control, allowing the education institutions to use different plants with most of the real interactions that usually are not present in model simulations.

To show de application of the method, we have presented the design and control of a mechanical plant (ball-and-plate). The differences between simulation and the co-simulation platform are remarked using simple controllers, and the results can be used to re-tune the regulation strategies or to choose more appropriate control methods.

As future work, we are working on more complex plants such as a Furuta pendulum and adaptive environments (scenarios and obstacles) for trajectories planning with holonomic robots. Moreover, several V-REP features can be explored in order to propose more interactive and realistic plants for the application of distinct control techniques.

REFERENCES

[1] H. Vargas, D. Dormido, N. Duro, D. Dormido-Canto, "Creación de laboratorios virtuales y remotos usando easy java simulations y Labview", XXVII Jornadas de Automática, Almería, España, pp. 1182-1188, Sep, 2006.

[2] S.Dormido, "Control learning: Present and future" Annual Control Reviews, vol.28, pp.115-136, 2005.

[3] D.Guillet, A.Nguyen., Y.Rekik, "Collaborative Web-Based Experimentation inFlexible Engineering Education", IEEE Trans on Education, vol. 48, no. 4, 2005.

[4] V.H. Andaluz, F.A. Chicaiza, C. Gallardo, W.X. Quevedo, J. Varela, J.S. Sánchez, O. Arteaga, "Unity3D-MatLab Simulator in Real Time for Robotics Applications", Augmented Reality, Virtual Reality, and Computer Graphics: Third International Conf., Lecce, Italia, pp. 246-263, Jun, 2016.

[5] R.D. Vásquez, H.O. Sarmiento, D.S. Muñoz, "Propuesta de implementación de plantas virtuales para la enseñanza de programas de control lógico", *ACOFI*, vol.11, no.22, pp. 1-13, 2016.

[6] A. Domínguez, "Modelado y Simulación de un Robot LEGO Mindstorms EV3 mediante V-REP y Matlab", Proyecto final de carrera, Dep. Informática, Universidad de Málaga, 2016.

[7] Coppelia Roboticts V-REP, V-REP User Manual, available at: http://www.coppeliarobotics.com/helpFiles/, Consultated Apr 2019.

[8] Blueworkforce bluezero, V-REP shapes, available at: https://blueworkforce.github.io/bluezero/v1/, Consultated Apr 2019.

[9] Solidworks Corporation, "Guía del estudiante para el aprendizaje del software Soidworks", 2011 available at: https://www.solidworks.com/sw/docs/Student_WB_2011_ESP.pdf

[10] E Fabregas, S. Dormido-Canto, S. Dormido, "Virtual and Remote Laboratory with the Ball and Plate System", IFAC-PapersOnLine, vol. 50, no. 1, pp. 9132-9137, 2017.

[11] K. Ogata, "Discrete-Time Control Systems", 2nd edition, Ed. Prentice Hall Inc., USA, 1995, pp. 596-609.

[12] A. Cedeño, M. Gordón, L. Morales, "Control de posición y seguimiento de caminos en el sistema Bola-Plataforma", XXVII Jornadas de Automática, Quito, Ecuador, vol.27, pp. 47-53, 2017.

ANEXO 3. **ARTÍCULO:** EVALUATION OF TRAJECTORY-PLANNING TECHNIQUES FOR MOBILE ROBOTS IN V-REP

Este anexo contiene el resumen extendido seleccionado para sustentación en la modalidad de ponencia oral y publicación en la conferencia internacional LATIN AMERICAN CONGRESS ON AUTOMATION AND ROBOTICS celebrada en Cali-Colombia de octubre 30 al 1 de noviembre del 2019.

Evaluation of Trajectory-Planning Techniques for Mobile Robots in V-REP

Harold Ruiz, Laura Rodrǵuez, Andrés Pantoja, and John Barco

Abstract Path planning is a key factor in mobile robotics but testing the strategies in real environments is a difficult task. Virtual plants represent one of the most versatile instruments in the teaching-learning process of control systems since they replace the lack of expensive laboratories or real plants. In this work, we propose a co-simulation platform using the 3D simulator V-REP and Matlab for the evaluation of different trajectory-planning strategies for a differential robot in scenes with obstacles. The results are compared calculating the performance of the methods in simulation and in the virtual environment with a kinematic control for the vehicle, showing remarkable differences between ideal and real scenarios.

1 Introduction

Virtual plants are software-based experimentation environments, where users can operate different graphical components representing elements of a physical model, in such a way that the system closely emulates the actual performance of the plant [6]. One widely-used platform to implement virtual plants is V-REP [1], which has had an increasing interest in many areas of engineering simulation and research, nourished by the advancement of technologies based on 3D simulation and their relevance in pedagogy [2], [4].

Path planning is one of the more important tasks in the development of mobile robotics. However, real laboratories for testing the performance of different strategies are expensive and limited in space, configuration, and possible vehicles. In this sense, the co-simulation platform proposed in this work allows the researchers to design in

Harold Ruiz, Laura Rodríguez, and Andrés Pantoja
Departamento de Electrónica, Universidad de Nariño, Pasto, Colombia e-mail: haroldfernandoruiz@hotmail.com, lauritamaria_rod@hotmail.com, ad_pantoja@udenar.edu.co

John Barco
Facultad de Ingeniería, I.U. CESMAG, Pasto, Colombia e-mail: jebarco@iucesmag.edu.co

V-REP a wide spectrum of scenarios with configurable obstacles located in large or small areas, for the running of one or several vehicles with on-board or global sensors. Then, the planning strategies are programmed in Matlab using the information of the environment (obtained from the sensors in V-REP) and sending the control actions back to the vehicles in V-REP to execute the trajectories. Both programs synchronize the information exchange in a closed-loop configuration to emulate real scenarios where Matlab is the controller, and the plant is the vehicle in the flexible V-REP environment.

As a contribution, the paper analyzes using the co-simulation for the planning of trajectories of a terrestrial robot with four representative path-planning methods (artificial potential fields, heuristic and stochastic searches, and graph-based routing) described in [3]. The performance of each strategy is compared by the implementation of the algorithms only in simulation (ideal conditions) and in the virtual platform (real conditions), remarking the differences about the efficiency, collisions, and final distance traveled. The physical constraints of the vehicle and obstacles in the virtual plant represent a challenge to improve the real application of the strategies in mobile robotics with on-board sensors (e.g., with odometry).

2 Methodology

To illustrate the planning implementation and the robot's kinematic control in V-REP through Matlab, three scenarios with narrow passages, traps, and random allocation of obstacles are proposed to design optimal trajectories between fixed initial and final points. The scenario is composed by a differential robot (Pioneer_p3dx) and several obstacles (different sized cubes) in a plain area of 10×10 meters, with distinct levels of difficulty to achieve the plan objective.

In the simulation, many physic dynamics are neglected. For instance, the robot is assumed as a particle without friction and with ideal actuators. However, in V-REP all these conditions are taken into account for the differential robot with non-holonomic restrictions. In this case, we implement a kinematic controller described in [5], using transformations to obtain the angular velocities the wheels from the robot's angular position and the desired distance to the goal. The angular position of the wheels are measured by odometry (in V-REP), so the control is based on local measures, increasing the complexity of the general trajectory-planning strategy.

The strategies to provide the set-points to the kinematic controller are based in well-known path-planning methods such as minimization of potential fields, particle swarm optimization (PSO), rapidly-exploring random trees (RRT), and cell decomposition [3], which represent most of the general methodologies to obtain an optimal routes. Each algorithm is programmed in Matlab with the conditions in the scenario and some performance indexes such as distance, number of collisions, and achievement of the final goal are evaluated in simulation and with the virtual plant.

3 Results

The results for a particular scenario are summarized in Figure 1, where trajectories in simulation are shown in black, the ones measured by odometry are in blue, and the real paths traveled by the robot are in red. It is remarkable that the algorithms are adapted in order to deal with two main requirements *i)* the safety of the route (free of collisions) and *ii)* the minimization of the total distance, defined as the summation of the Euclidean distances between the control points in each iteration.

With the adjusted methods for the scenario, the robot is able to reach the desired position. However, according to the trajectories in Figure 1, the RRT method presents several collisions due to the closeness between set-points and obstacles to shorten the path. In addition, the CD method shows some oscillations generated by the saturation of the actuators (motors in the vehicle), while the potential field method present the more complex implementation, but the shortest final distance. A trade-off is presented for the PSO strategy, where the distance is longer, but the computational cost is reduced. Two more scenarios with narrow passages and trap arrangements of

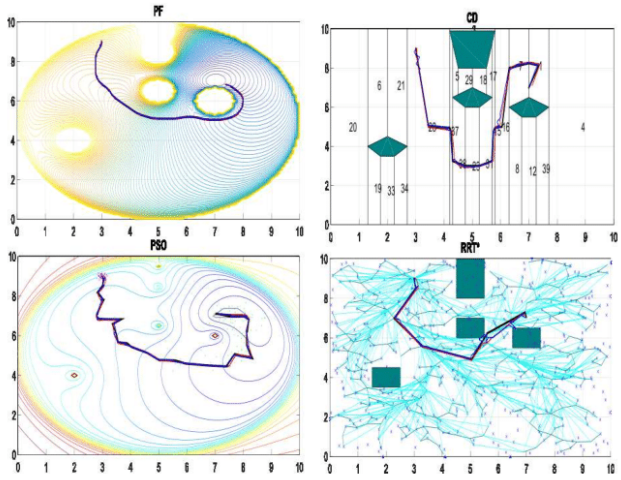


Fig. 1 Resulting trajectories for the four path-planning methods in a scenario with random obstacles.

obstacles are also analyzed to provide more tests and conclusive remarks about the performance of the strategies, as well as the differences between the simulation and the real co-simulation environment.

References

1. Coppelia Roboticts: V-rep user manual. Tech. rep. Available on: http://www.coppeliarobotics.com/helpFiles/
2. Dormido, S.: Control learning: Present and future. Annual Control Reviews **18**, 115–136 (2005)
3. Getial, J., Erazo, E., Pantoja, A.: A quantitative comparison of path planning methods in mobile robotics. In: 3rd Colombian Conference on Automatic Control (CCAC) (2017)
4. Guillet, D., Nguye, A., Rekik, Y.: Collaborative web-based experimentation inflexible engineering education. IEEE Trans on Education **48**(8) (2005)
5. Tzafestas, S.: Introduce to Mobile robot control. Ed. Elsevier (2013)
6. Vargas, H., Dormido, D., Duro, N., Dormido-Cant, D.: Creación de laboratorios virtuales y remotos usando easy java simulations y labview. In: XXVII Jornadas de Automática, pp. 1182–1188 (2006)