Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Herramienta de diseño para sistemas de control de las plantas del Laboratorio de Control Automático del Instituto Tecnológico de Costa Rica

Informe de Trabajo Final de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

André Martínez Arana

II Semestre, 2018

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADUACIÓN

ACTA DE APROBACIÓN

Defensa de Proyecto de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del proyecto de graduación denominado Herramienta de diseño para sistemas de control de las plantas del Laboratorio de Control Automático del Instituto Tecnológico de Costa Rica, realizado el señor André Martínez Arana y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

TEC | Tecnológico de Costa Rica
Ingeniería Electrónica

Dr. Ing. Carlos Meza Benavides — Profesor lector

Ing. Sergio Morales Hernández — Profesor lector

Ing. Carlos Mauricio Segura Quirós — Profesor asesor

Cartago, 20 de Noviembre, 2018

Declaro que el presente documento de tesis ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente documento.

André Martínez Arana

Céd: 1-1566-0905

Cartago, Noviembre 2018

Resumen

La Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica tiene como proyecto la realización de una unidad de prototipado rápido para sistemas de control en distintas plantas con las que se encuentra equipado el Laboratorio de Control Automático. Dicho proyecto busca disminuir los tiempos de implementación de sistemas de control que los estudiantes demoran en aplicar a las plantas del laboratorio.

La unidad de prototipado rápido se dividió en dos partes: la unidad de hardware, que interactúa con la planta, y la herramienta de diseño que interactúa con el usuario.

El presente trabajo corresponde a la herramienta de diseño, en donde se desarrolló una herramienta con interfaz gráfica de usuario, con capacidad de simulación y con un protocolo de comunicación hacia la unidad de hardware. Dicha herramienta provee al usuario con los instrumentos necesarios para el diseño de distintos sistemas de control, disminuye posibles errores cometidos por el usuario, y traduce a lenguaje de máquina los diseños generados para la comunicación y transmisión de datos hacia la unidad de hardware.

Palabras clave: prototipado rápido, sistemas de control, control automático, interfaz gráfica de usuario, simulación, herramienta de diseño.

Abstract

The School of Electronic Engineering of Instituto Tecnologico de Costa Rica, has as its project the realization of a rapid prototyping unit for control systems in different plants with which the Automatic Control Laboratory is equipped. The project seeks to reduce the implementation times that students take in applying control systems to the laboratory plants.

The rapid prototyping unit was divided into two parts: the hardware unit, which interacts with the plant, and the design tool that interacts with the user.

The present work corresponds to the design tool, where a tool with graphical user interface was developed, with simulation capability and with a communication protocol to the hardware unit. This tool provides the user with the necessary tools for the design of different control systems, decreases possible errors committed by the user, and translates into machine language the designs generated for the communication and transmission of data to the hardware unit.

Keywords: rapid prototyping, control systems, automatic control, graphical user interface, simulation, design tool.

A mis padres, a quienes les debo todo lo que soy.

Índice general

Índice de figuras VIII

Índice de tablas XI

Índice de código XII

Capítulo 1: Introducción 1

1.1 Problemática actual con la Unidad Controladora de Procesos y el Entorno de Desarrollo Integrado 3

1.2 Herramienta de diseño para sistemas de control y la unidad de hardware 3

Capítulo 2: Meta y objetivos 5

2.1 Meta 5

2.2 Objetivo general 5

2.3 Objetivos específicos 5

Capítulo 3: Marco teórico 6

3.1 Entorno de desarrollo integrado para una unidad controladora de procesos 6

3.1.1 TeleLAB 6

3.1.2 Unidad Controladora de Procesos 6

3.1.3 Entorno de Desarrollo Integrado 6

3.2 Estructura de un computador 7

3.2.1 Unidad central de procesamiento 8

3.2.2 Memoria principal 8

3.2.3 Periféricos 8

3.3 Lenguaje de programación 8

3.3.1 Sintaxis y semántica 9

3.3.2 Lenguaje de alto nivel 9

3.3.3 Traductor 10

3.3.4 Compilador 10

3.3.5 Intérprete 10

3.4 Interacción persona-computadora 11

3.4.1 Interfaz gráfica de usuario 11
Capítulo 4: Herramienta de diseño para sistemas de control 13
4.1 Interfaz de usuario 13
4.1.1 Requerimientos de la interfaz gráfica 14
4.1.2 Tkinter 15
4.1.3 Estructura/Diseño de la interfaz gráfica 15
4.1.4 Módulos y sus parámetros 16
4.1.5 Estructura básica de los bloques 35
4.1.6 Principales funciones de los objetos 37
4.1.7 Listas enlazadas 39
4.1.8 Netlist 42
4.1.9 Archivo XML 43
4.2 Simulación del sistema de control 45
4.2.1 Ball & Beam 47
4.2.1 Recorrido de árboles 48
4.2.2 Simulación de los módulos 51
4.3 Protocolo de comunicación 63
Capítulo 5: Análisis y resultados de la herramienta de diseño 66
5.1 Interfaz gráfica de usuario de la herramienta de diseño 66
5.2 Resultados de simulación 67
5.3 Protocolo de comunicación y el archivo interpretable 72
Capítulo 6: Conclusiones 76
Bibliografía 77

Índice de figuras

Figura 1.1 Metodología en el Laboratorio de Control Automático. 1

Figura 1.2. Diagrama general de la solución. 4

Figura 1.3 Solución de la herramienta de diseño. 4

Figura 3.2.1. Estructura básica de un computador. 7

Figura 3.3.1. Traducción por compilador 10

Figura 3.3.2. Traducción por intérprete. 10

Figura 4.1.1. Ventana principal de la herramienta de diseño. 13

Figura 4.1.2 Estructura de la herramienta de diseño. 15

Figura 4.1.3 Interfaz Gráfica de Usuario 16

Figura 4.1.4 Módulos de la herramienta de diseño. 17

Figura 4.1.5. Icono ADC. 17

Figura 4.1.6. Ventana ADC. 18

Figura 4.1.7. Icono PID. 19

Figura 4.1.8. Estructura interna del bloque PID. 19

Figura 4.1.9. Ventana PID. 20

Figura 4.1.10. Icono CTE. 21

Figura 4.1.11. Icono DER. 22

Figura 4.1.12. Icono SAT. 22

Figura 4.1.13. Ventana SAT. 23

Figura 4.1.14. Icono SUM. 23

Figura 4.1.15. Icono RST. 24

Figura 4.1.16. Icono RE. 24

Figura 4.1.17. Distintas configuraciones del bloque RE. 25

Figura 4.1.18. Estructura interna del bloque RE. 25

Figura 4.1.19. Estructura interna del bloque REI. 26

Figura 4.1.20. Ventana RE. 26

Figura 4.1.21. Icono FT. 27

Figura 4.1.22. Ventana FT. 28
Figura 4.1.23. Icono QDEC. 29
Figura 4.1.24. Ventana QDEC. 29
Figura 4.1.25. Icono PWM. 29
Figura 4.1.26. Ventana PWM. 30
Figura 4.1.27. Icono DAC. 30
Figura 4.1.28. Icono FLT. 31
Figura 4.1.29. Ventana FLT. 31
Figura 4.1.30. Icono SCOPE. 32
Figura 4.1.31. Icono STEP. 32
Figura 4.1.32. Icono TRP. 33
Figura 4.1.33. Propiedades de los tipos de objetos. 36
Figura 4.1.34. Posición de las terminales de conexión del bloque ADC. 38
Figura 4.1.35. Creación de un nodo en una lista enlazada. 40
Figura 4.1.36. Asignación del nuevo nodo cabeza. 40
Figura 4.1.37. Remover elemento de lista enlazada. 41
Figura 4.1.38. Estructura del NETLIST. 42
Figura 4.1.39. Ejemplo de sistema de control. 42
Figura 4.2.1. Diagrama Ball & Beam. 47
Figura 4.2.2. Árbol binario. 49
Figura 4.2.3. Sistema de control con 6 bloques. 49
Figura 4.2.4. Árbol del sistema de control de la figura 4.2.3. 50
Figura 4.2.5. Función escalón. 52
Figura 4.2.6. Señal trapecio. 53
Figura 4.2.7. Control PID, I_PD y PI_D. 54
Figura 4.2.8. Diagrama interno de los bloques RE y REI. 56
Figura 4.2.9. Sistema con bloque saturador. 60
Figura 4.2.10. Función de los bloques sumador y restador. 60
Figura 4.2.11. Bloques SALP (izquierda) y ENTP (derecha). 61

Figura 4.3.1. Ejemplo del archivo interpretable. 64
Figura 4.3.2. Estructura de la trama de datos de la sección CON 64
Figura 4.3.3. Protocolo de comunicación. 65
Figura 5.1.1. Sistema de control diseñado con la herramienta. 67
Figura 5.2.1. Ejemplo de sistema de control con bloque de control PID. 67
Figura 5.2.2. Gráfico de respuesta del bloque PID ante una entrada constante y trapecio. 69
Figura 5.2.3. Sistema de control con bloque FT. 69
Figura 5.2.4. Gráfico de respuesta del bloque FT ante una entrada constante y trapecio. 70
Figura 5.2.5. Sistema de control con bloque RE. 70
Figura 5.2.6. Sistema de control con bloque REI. 71
Figura 5.2.7. Respuesta de los bloques RE y REI ante una entrada escalón. 71
Figura 5.3.1. Sistema de control implementado con un bloque FT. 72
Figura 5.3.2. Implementación de la herramienta de diseño de sistemas de control. 75

Índice de tablas

Tabla 4.1.1 Requerimientos de la interfaz gráfica de usuario. 14
Tabla 4.1.2 Código hexadecimal para los parámetros del bloque ADC. 18
Tabla 4.1.3 Código hexadecimal para los parámetros del bloque PID. 21
Tabla 4.1.4 Código hexadecimal para los parámetros del bloque SAT. 23
Tabla 4.1.5 Código hexadecimal para los parámetros del bloque RE. 27
Tabla 4.1.6 Código hexadecimal para los parámetros del bloque FT. 28
Tabla 4.1.7 Código hexadecimal para los parámetros del bloque DAC 30
Tabla 4.1.8 Código hexadecimal para los parámetros del bloque FLT. 32
Tabla 4.1.9 Código hexadecimal para los parámetros de los módulos. 33

Índice de código

Código 4.1. Clase NodoADC 35

Código 4.2. Asignación de acciones de usuario al objeto ADC 36

Código 4.3. clickIzquierdoADC 37

Código 4.4. enMovimientoADC 38

Código 4.5. Función para remover un elemento de la lista 41

Código 4.6. Estructura básica para el recorrido de un árbol binario 51

Código 4.7. Función Simulación Bloque STEP 51

Código 4.8. Función Simulación Bloque TRAPECIO 52

Código 4.9. Función Simulación Bloque PID 55

Código 4.10. Función Simulación Bloque RE de 2 estados 57

Código 4.11. Función Simulación Bloque FT 58

Código 4.12. Función Simulación Bloque SAT 59

Código 4.13. Función Simulación Bloque SAT 61

Código 4.14. Función actualizarSalidaPlanta 62

Capítulo 1: Introducción

La Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica cuenta con un laboratorio de Control Automático en donde los estudiantes diseñan e implementan sistemas de control a distintas plantas con las que se encuentra equipado el laboratorio. Dichas plantas simulan problemas y retos reales en la ingeniería para que el estudiante logre aplicar lo estudiado en el curso de Control Automático, fomentando así el aprendizaje de los temas abarcados en dicho curso.

Como parte de la metodología de aprendizaje del curso, cada grupo de estudiantes (de 3 a 4 integrantes) selecciona una planta con la que deben trabajar a lo largo del curso lectivo, es decir, alrededor de 15 semanas. Durante este periodo, los grupos de trabajo deben pasar por cuatro etapas en la realización del sistemas de control de la planta: Modelo, Diseño, Simulación e Implementación.

La etapa de Modelo consiste en obtener el modelo analítico y empírico de la planta seleccionada. Durante el Diseño, los estudiantes proceden a diseñar el regulador correspondiente ante posibles perturbaciones. En la tercera etapa se verifica el correcto diseño y funcionamiento del regulador a implementar por medio de una Simulación, y por último, en la etapa de Implementación, los grupos de trabajo implementan su diseño del sistema de control en hardware.

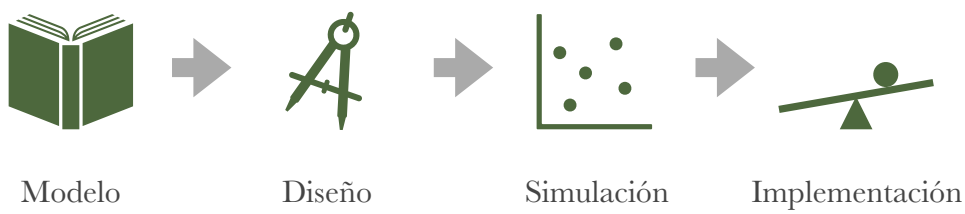

Figura 1.1 Metodología en el Laboratorio de Control Automático.

Si bien existe una distribución clara de las tareas para realizar el sistema de control en la planta, los estudiantes presentan otros tipos de problemas para lograr completar cada etapa. El principal problema al que se enfrentan los grupos de trabajo, es el tiempo que se requiere para familiarizarse con los programas y herramientas a utilizar. El tiempo que se toma en aprender a utilizar estas herramientas de diseño e implementación no está contemplado en los objetivos del curso y no forma parte del contenido de los mismos, siendo el objetivo principal el diseño de sistemas de control.

Esta extensa duración en el tiempo de familiarización de las herramientas conlleva a un menor tiempo para el aprendizaje de los estudiantes sobre los distintos tipos de sistemas de control, por lo que es común que los estudiantes, por cuestiones de tiempo, experimenten únicamente con un solo tipo de sistema de control.

La necesidad de disminuir el tiempo en la implementación de los sistemas de control diseñados por los estudiantes fue motivo de un proyecto de graduación en el año 2008, titulado "Unidad Controladora de Procesos para el diseño, análisis, simulación e implementación de sistemas de control automático a través de redes TCP/IP" [2]. El proyecto consistió en crear una herramienta para la adquisición de datos y empleo de señales de control en las plantas de forma remota.

Otro proyecto en el año 2010, titulado "Diseño de un Entorno de Desarrollo Integrado para una Unidad Controladora de Procesos" [1], presentaba la misma intención de solventar el problema del tiempo y lograr una mejor interacción con el usuario y la Unidad Controladora de Procesos del proyecto realizado en el año 2008, sin embargo, existió una nueva problemática conforme pasaron los años.

1.1 Problemática actual con la Unidad Controladora de Procesos y el Entorno de Desarrollo Integrado

Bajo el proyecto de la Unidad Controladora de Procesos, el usuario debía ingresar las instrucciones por medio de lenguaje de máquina, siguiendo un protocolo de comunicación que involucraba la traducción a un sistema hexadecimal para su debido funcionamiento. Esta brecha entre el usuario y el lenguaje de máquina resultó ser una de las razones para la elaboración del proyecto del Entorno de Desarrollo Integrado (IDE) [1], el cual buscaba disminuir el tiempo de desarrollo y posibles errores en la programación de la unidad, con el uso de aplicaciones gráficas.

Los dos proyectos fueron utilizados en el Laboratorio de Control Automático por varios años, sin embargo, la tecnología en ambos proyectos se volvió obsoleta y su funcionalidad se vio afectada, por lo que surge de nuevo la necesidad de una nueva herramienta para lograr un prototipado rápido de sistemas de control en las plantas del laboratorio.

1.2 Herramienta de diseño para sistemas de control y la unidad de hardware

El presente proyecto realiza una herramienta de diseño, fácil de utilizar, amigable con el usuario, con interfaz gráfica y capacidad de simulación, logrando una comunicación rápida y eficiente con la unidad de hardware para la debida implementación del sistema de control en las plantas del Laboratorio de Control Automático, permitiendo así un desarrollo rápido de los sistemas de control diseñados por parte de los estudiantes.

Figura 1.2. Diagrama general de la solución.

Con una interfaz gráfica y con tecnologías mas modernas, la herramienta de diseño permite a los estudiantes disminuir el tiempo actual que se demora en la implementación de sus diseños de sistemas de control, logrando un mejor aprendizaje al aportarles un mayor tiempo para la experimentación con otros sistemas de control. La interacción con la unidad de hardware la realiza el usuario por medio de la interfaz de la herramienta, por lo que la herramienta debe eliminar cualquier posible error ingresado por el usuario hacia la unidad.

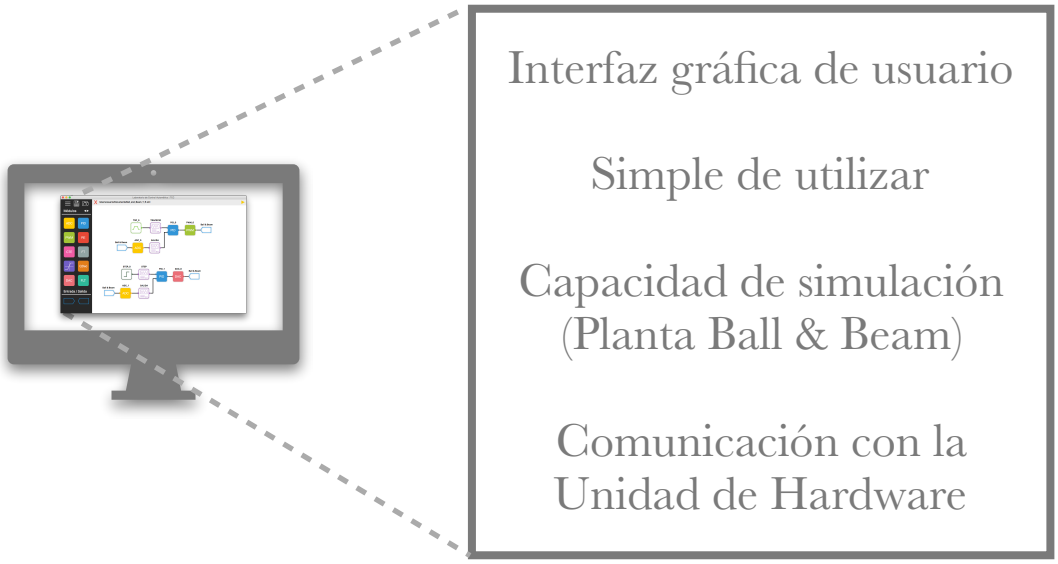

Figura 1.3 Solución de la herramienta de diseño.

Capítulo 2: Meta y objetivos

2.1 Meta

• Agilizar el proceso de diseño e implementación de sistemas de control para el aumento del tiempo de aprendizaje de los estudiantes de Laboratorio de Control Automático.

2.2 Objetivo general

• Desarrollar una herramienta de diseño, con capacidad de simulación, que permita a los estudiantes el prototipado de control automático de las plantas del Laboratorio de Control Automático del Instituto Tecnológico de Costa Rica.

2.3 Objetivos específicos

• Diseñar una interfaz gráfica, simple y amigable con el usuario, que permita el prototipado rápido de las plantas de control del Laboratorio de Control Automático del Instituto Tecnológico de Costa Rica.

• Generar la simulación del sistema de control de la planta “Ball & Beam” dentro de la interfaz gráfica de la herramienta de control.

• Diseñar el protocolo de comunicación entre la herramienta de diseño y la unidad de prototipado.

Capítulo 3: Marco teórico

3.1 Entorno de desarrollo integrado para una unidad controladora de procesos

3.1.1 TeleLAB

El proyecto TeleLAB de la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica consistió en la creación de un laboratorio de control automático para ser accesado de manera remota por los estudiantes por medio de una conexión a internet. Con TeleLAB, los estudiantes diseñaban y experimentaban con distintos sistemas de control a diferentes plantas del laboratorio de forma remota y sin necesidad de estar de forma presencial en el laboratorio. [1], [2]

3.1.2 Unidad Controladora de Procesos

La unidad controlador de procesos fue desarrollada como parte del proyecto TeleLAB. Su función consistió en permitir el diseño, análisis, simulación e implementación de sistemas de control a distintas plantas del laboratorio de control automático. Su método de programación consistía en lenguaje de máquina siguiendo un sistema hexadecimal. [2]

3.1.3 Entorno de Desarrollo Integrado

El Entorno de Desarrollo Integrado surgió como solución para mejorar la interacción con el usuario en el proyecto TeleLAB y consistió en la elaboración de una herramienta para realizar los experimentos de sistemas de control automático por medio

de aplicaciones gráficas. El entorno de desarrollo integrado facilitó la interacción del usuario con la Unidad Controladora de Procesos y su lenguaje de máquina. [1]

3.2 Estructura de un computador

La estructura de un computador puede variar según su función, sin embargo, la representación más básica de un computador moderno viene dado por la figura 3.2.1. En su funcionamiento más simple, un computador es capaz del procesamiento, almacenamiento y movimiento de datos. [13]

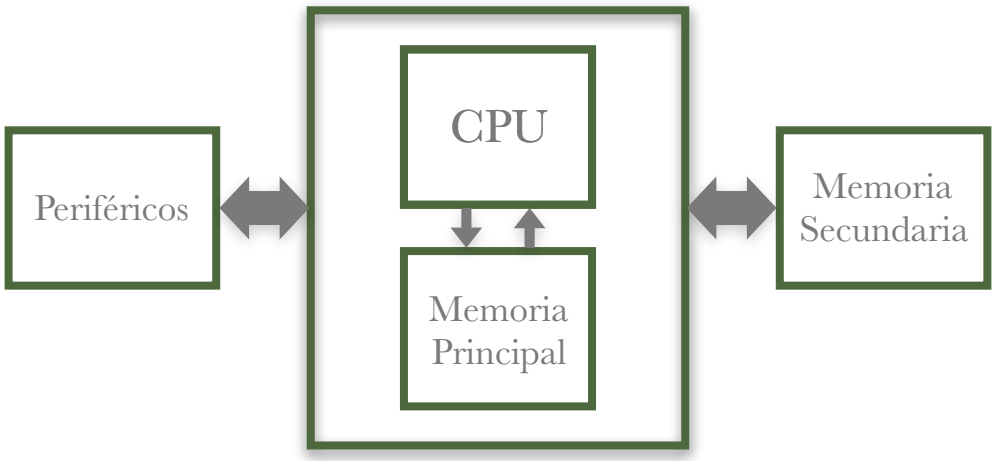

Figura 3.2.1. Estructura básica de un computador.

Al estar en constante procesamiento de datos, es decir, en su función de procesamiento, un computador debe de almacenar los datos procesados de manera temporal, por lo que requiere de al menos una función de almacenamiento de datos para el posterior uso, lectura y recuperación de la información procesada.

La función de movimiento de datos es la encargada del envío de datos hacia y desde los periféricos. Por último, debe existir un control entre las funciones para indicar el tiempo en que debe ejecutarse cada función. El control es ordenado por instrucciones

ingresadas por el usuario, y la unidad de control actúa acorde a estas operaciones y los recursos disponibles. [11]

La estructura básica del computador consiste entonces de una unidad de procesamiento, memoria principal y periféricos.

3.2.1 Unidad central de procesamiento

La unidad central de procesamiento, CPU por sus siglas en inglés, es donde se realizan todas las operaciones básicas del computador así como las operaciones aritméticas. La unidad controla la función de procesamiento de datos. [11]

3.2.2 Memoria principal

La función de almacenamiento es realizada por la memoria principal (memoria RAM, Memoria de acceso aleatorio). Las instrucciones y datos de los programas utilizados por la unidad central de procesamiento son almacenados en la memoria principal. [11]

3.2.3 Periféricos

La interacción entre el computador y el usuario se realiza mediante los periféricos. Ejemplos de periféricos son el ratón (*mouse*), teclado, monitor, y su información es procesada por la unidad central de procesamiento del computador. [11], [13]

3.3 Lenguaje de programación

Un programa es un conjunto de instrucciones que indican al computador las funciones que debe seguir. Para la generación de estas instrucciones es necesario la

implementación de un lenguaje entendible por el ser humano y traducible a un sistema computacional. Este lenguaje artificial es llamado lenguaje de programación y es utilizado para evitar ambigüedades e imprecisiones en la creación de instrucciones, y además crear notaciones para la expresión de computaciones de manera inequívoca. [13]

Los lenguajes de programación presentan una estructura, o forma, denominada sintaxis, y un significado llamado semántica.

3.3.1 Sintaxis y semántica

La sintaxis se refiere a las distintas formas en la que los símbolos pueden ser combinados para lograr la creación de instrucciones o programas en el lenguaje de programación. Esta creación de instrucciones es llevada a cabo bajo una serie de normas que determinan la correcta escritura de las ordenes. Por otro lado, la semántica se refiere al significado de una cadena de símbolos válido dentro del lenguaje, logrando así una interpretación de la misma. [10]

3.3.2 Lenguaje de alto nivel

Python, C++, Java y BASIC son ejemplos de lenguajes de programación de alto nivel, en donde todos comparten las características de poseer una sintaxis y semántica inequívoca. Los lenguajes de alto nivel están diseñados para ser entendibles por las personas. La adición de dos número a y b puede representarse de la forma $c = a + b$, sin embargo, los computadores entienden únicamente lenguaje de bajo nivel, es decir, instrucciones representadas por unos ‘1’ y ceros ‘0’. Este lenguaje es conocido como lenguaje de máquina. [13]

Para la traducción de lenguaje de alto nivel a lenguaje de máquina es necesario de un compilador o intérprete para la obtención de la notación binaria de la instrucción o programa deseado.

3.3.3 Traductor

El proceso de traducir lenguaje de alto nivel a lenguaje de máquina para que el computador logre ejecutar un programa o instrucción es llamado traducción. La traducción puede llevarse a cabo de dos formas: por compilador o por intérprete.

3.3.4 Compilador

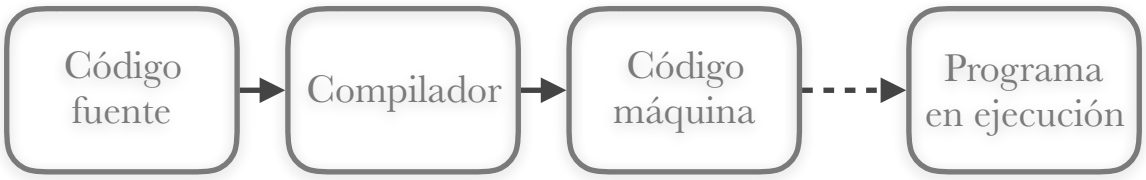

Figura 3.3.1. Traducción por compilador

Un compilador es un programa que toma otro programa escrito en lenguaje de alto nivel y realiza la traducción a un programa equivalente en código máquina, es decir, lenguaje de máquina [9]. La traducción por compilador se realiza a todo el código fuente por completo y es realizada una sola vez.

C es un lenguaje de programación que utiliza la traducción con compilador.

3.3.5 Intérprete

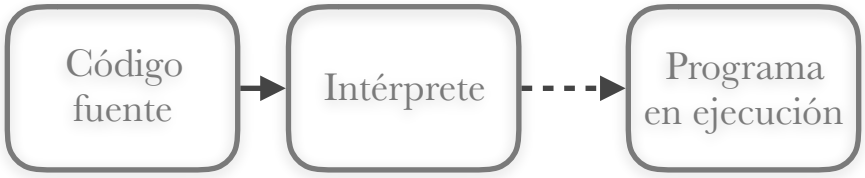

Figura 3.3.2. Traducción por intérprete.

La traducción con intérprete es un programa que simula un computador capaz de entender lenguaje de alto nivel, y tanto la traducción como ejecución se realiza instrucción por instrucción [9]. No crea un programa equivalente en código máquina.

El lenguaje de programación Python utiliza la traducción con intérprete.

3.4 Interacción persona-computadora

La interacción persona-computadora es el estudio, planeamiento y diseño de la forma en que las personas y computadoras trabajan de forma conjunta para satisfacer las necesidades del usuario. Para su implementación, es necesario tomar en cuenta las necesidades y expectativas del usuario, sus habilidades y limitaciones físicas, y las limitaciones del hardware y software del computador. [5]

Se entiende como interfaz de usuario a la parte de la computadora y su software con la que el usuario se encuentra en contacto, es decir, el usuario observa, escucha y toca [5]. El usuario interactúa con el computador mediante dispositivos de entrada y salida. Como ejemplos de dispositivos de entrada comunes está el teclado y el puntero, y el principal ejemplo de dispositivo de salida es el monitor.

3.4.1 Interfaz gráfica de usuario

La interfaz gráfica de usuario se refiere a la interacción por parte de un usuario con una colección de elementos y objetos que pueden ser visibles, escuchados y manipulados [5]. La interacción del usuario con un objeto se denomina acción, y esta puede referirse a la selección, señalamiento y modificación de un objeto.

Dentro de la interfaz gráfica de usuario, la información puede ser representada en forma de texto o mediante imágenes gráficas llamados iconos, en donde cada ícono representa un objeto o acción.

Los sistemas gráficos presentan una serie de ventajas [5]:

- Los símbolos son reconocidos más fácilmente que el texto
- Aprendizaje más rápido
- Uso y resolución de problemas de manera más rápida
- Más fáciles de recordar
- Disminuyen los errores

Sin embargo, entre las desventajas de los sistemas gráficos se encuentran:

- Mayor complejidad de diseño
- Aprendizaje de la herramienta es aun necesario
- Limitaciones del hardware

Capítulo 4: Herramienta de diseño para sistemas de control

4.1 Interfaz de usuario

El diseño de la interfaz gráfica de usuario se diseñó de manera que fuera amigable con el usuario y fácil de utilizar, evitando así las demoras en la familiarización con nuevas herramientas.

La interfaz presenta elementos comunes en aplicaciones de software como lo son iconos, colores y alertas, haciendo de la aplicación un programa intuitivo y sin la necesidad de un extenso aprendizaje previo a su uso.

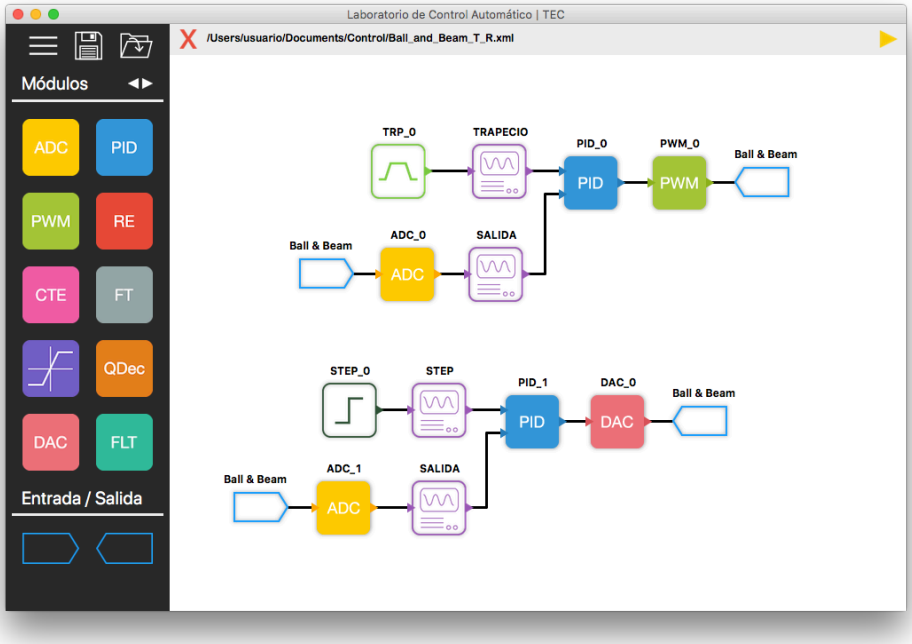

Figura 4.1.1. Ventana principal de la herramienta de diseño.

Para el diseño de la interfaz gráfica de usuario, se partió de la necesidad de los distintos módulos necesarios para la implementación de los sistemas de control establecidos: PID, I_PD, PI_D, RE, REI y FT. Además de contar con los módulos y bloques necesarios, la herramienta presenta la capacidad de crear conexiones entre los bloques, por lo que se requería de entradas y salidas en cada uno de los bloques, los cuales simulan, a nivel gráfico, las conexiones reales de los módulos en la unidad de hardware.

Cada módulo, según su funcionalidad, presenta opciones configurables al usuario para la debida implementación de su diseño del sistema de control, por lo que la forma del ícono dentro de la interfaz puede cambiar una vez que se encuentra dentro del workspace (cambio en el número de entradas o salidas de un bloque.)

4.1.1 Requerimientos de la interfaz gráfica

Para lograr una herramienta de diseño intuitiva, se requería que la interfaz cumpliera con los siguientes requerimientos:

Tabla 4.1.1 Requerimientos de la interfaz gráfica de usuario.

Mantener la interfaz simple	Evitar el uso innecesario de elementos.
Uso de elementos comunes de interfaz de usuario	Uso de elementos comunes de interfaz de usuario como iconos de abrir y guardar, mensajes de alerta, botones, menús, etc.
Diseño de página	Determinar la ubicación de los elementos y ventanas de manera eficiente.
Uso adecuado de colores y texturas	Uso de colores para una mejor atención del usuario.
Uso de tipografía para claridad	El uso adecuado del tamaño y tipo de letra logra una mejor atención y legibilidad.
Comunicación por parte de la interfaz hacia el usuario	Uso de mensajes y colores para indicar alguna alerta o error, entre otros.

4.1.2 Tkinter

Para la programación de la interfaz gráfica de usuario se utilizó la herramienta Tkinter, biblioteca de interfaz gráfica de usuario integrada en todas las distribuciones de Python. Tkinter es la interfaz de Python a Tk, lenguaje de herramientas de comando Tcl/Tk. Además de ser software de código abierto, Tkinter es portable a través de todos distintos sistemas operativos (Windows, Linux y MacOS), es de fácil acceso dado que viene instalado en todas las distribuciones de Python y es relativamente simple de utilizar.

4.1.3 Estructura/Diseño de la interfaz gráfica

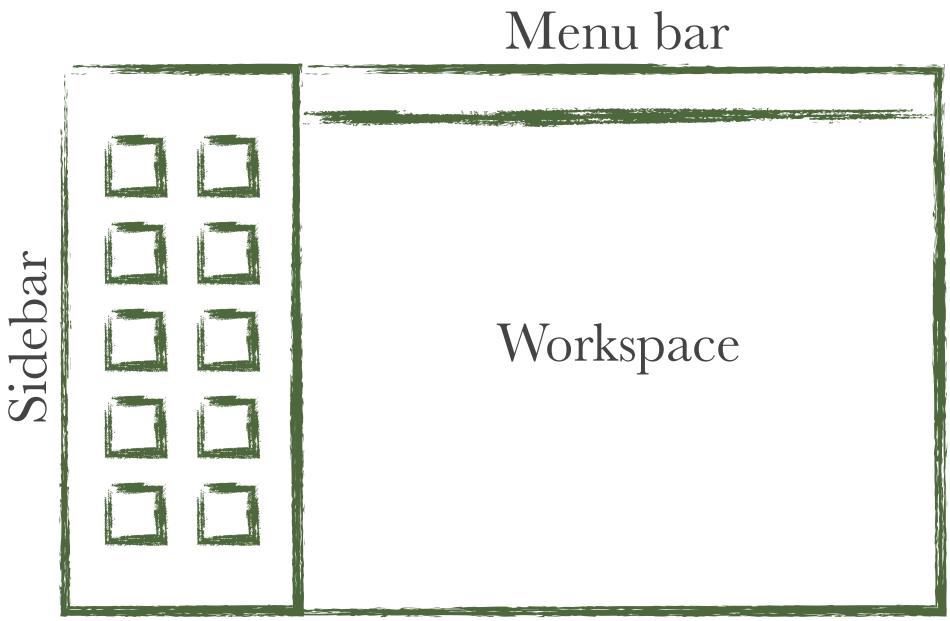

Figura 4.1.2 Estructura de la herramienta de diseño.

La estructura de la herramienta de diseño se dividió en tres secciones: Sidebar, menu bar y workspace. La sección del sidebar se diseñó para contener todos los bloques/módulos disponibles para el usuario, en donde mediante con el puntero del mouse, el usuario puede arrastrar y soltar un bloque al área del workspace para su debida configuración. Además de contener los bloques utilizados para el diseño de sistemas de

control, el sidebar contiene también los iconos de abrir y guardar archivos, además de un icono de configuración.

La sección menu bar, contiene el iconos de eliminar bloques y conexiones, además del icono de simulación y generación del archivo interpretable para la comunicación con la unidad de hardware. La función de la sección menu bar es también indicar al usuario cuando la opción del icono eliminar está seleccionada.

Por último, el workspace es la sección de la herramienta en donde el usuario logra la conexión de los distintos módulos que añadió del sidebar. Desde la sección del workspace, el usuario configura y manipula libremente los módulos añadidos y elimina o crea conexiones entre los módulos del workspace.

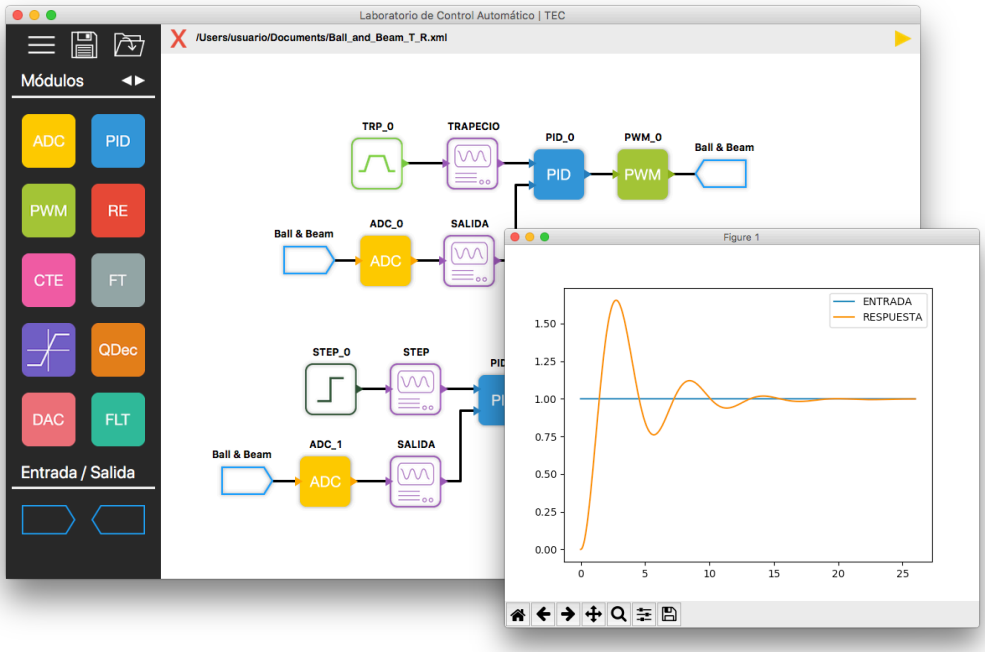

Figura 4.1.3 Interfaz Gráfica de Usuario

4.1.4 Módulos y sus parámetros

Figura 4.1.4 Módulos de la herramienta de diseño.

La herramienta cuenta con 16 distintos módulos, 1 bloque de entrada y 1 bloque de salida. Los módulos son los siguientes: ADC, PID, CTE, DER, SAT, SUM, RST, RE, FT, QDEC, PWM, DAC, FLT, SCOPE, STEP, TRP.

Bloque ADC

Figura 4.1.5. Icono ADC.

El bloque ADC cuenta con 1 entrada y 1 salida. Es la representación de un módulo Convertidor Analógico Digital, en donde la entrada es una señal analógica, y la salida es la señal digital equivalente a la entrada. Dentro de los parámetros configurables por el usuario, se encuentra el rango de operación, el cual puede ser bipolar o unipolar, además

del tiempo de muestreo en milisegundos. La figura 4.1.6. muestra la ventana presente al usuario para la configuración de estos parámetros.

Figura 4.1.6. Ventana ADC.

Para la comunicación con la unidad de hardware se utilizó un código que representa a cada parámetro. Este código hexadecimal viene dado por la tabla 4.1.2.

Tabla 4.1.2 Código hexadecimal para los parámetros del bloque ADC.

Parámetro Rango	Código Hexadecimal
0-10 V	0x30
0-5 V	0x31
0-2 V	0x32
0-1 V	0x33
±1 V	0x34
±2 V	0x35
±5 V	0x36
±10 V	0x37

Bloque PID

Figura 4.1.7. Icono PID.

El bloque PID cuenta con 2 entradas y 1 salida. El bloque PID representa el sistema de control PID, I_PD y PI_D, en donde su configuración se realiza desde la ventana de configuración del bloque.

Dado que el sistema de control PID siempre presenta retroalimentación negativa, la entrada 1 representa la señal de referencia y la entrada 2 es la señal de retroalimentación. Esto se realizó para disminuir el número de bloques necesarios de la herramienta de diseño y permitir una simplificación en la elaboración de los sistemas de control. La señal de salida del control PID viene dada en la salida 1 del bloque.

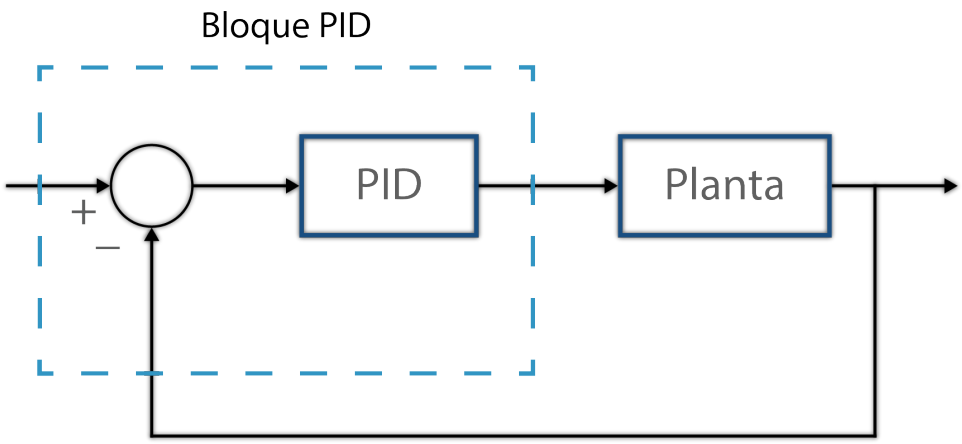

Figura 4.1.8. Estructura interna del bloque PID.

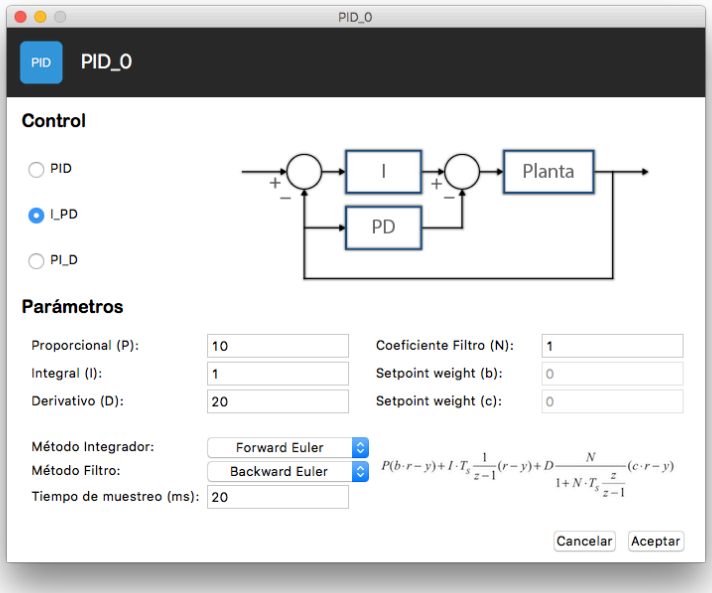

Figura 4.1.9. Ventana PID.

Dentro de los parámetros configurables del PID se encuentra el tipo (PID, I_PD y PI_D), la constante proporcional (P), integral (I) y derivativo (D), además del coeficiente del filtro N. El tipo también modifica los parámetros (b) y (c), según la configuración seleccionada.

Para el método del integrador se implementaron los métodos Forward Euler, Backward Euler y Trapezoidal. Estos mismos métodos de aproximación son utilizados para las opciones del método de filtro. Por último, se le solicita al usuario el tiempo de muestreo en milisegundos.

El código en hexadecimal para la representación de cada uno de estos parámetros viene dado por la tabla 4.1.3.

Tabla 4.1.3 Código hexadecimal para los parámetros del bloque PID.

Parámetro	Código Hexadecimal
Tipo: PID, I_PD, PI_D	0x3030
P	0x3031
I	0x3032
D	0x3033
C	0x3034
B	0x3035
N	0x3036
Método Integrador: FE, BE, TR	0x3037
Método Filtro: FE, BE, TR	0x3038
Tiempo de muestreo	0x3039

Bloque CTE (CONSTANTE)

Figura 4.1.10. Icono CTE.

El bloque CTE cuenta con 1 salida. La salida del bloque consiste en un escalar configurado por el usuario.

El valor de la constante puede ser positivo o negativo y puede ser un número flotante o entero. El código hexadecimal para representar este parámetro dentro del bloque es el 0x3030.

Bloque DER (Derivativo)

Figura 4.1.11. Icono DER.

El bloque DER cuenta con 2 entradas y 1 salida. La salida 1 del bloque derivativo es la misma señal que la ingresada por la entrada 1, mientras que la salida 2 del bloque es la derivada de la señal en la entrada. El bloque derivativo no presenta opciones de configuración.

Bloque SAT (SATURADOR)

Figura 4.1.12. Icono SAT.

El bloque SAT cuenta con 1 entrada y 1 salida. La función del bloque es saturar la señal de entrada entre los límites inferior y superior. Si la señal de entrada es mayor al límite superior, este será saturado, mismo caso en que la señal de entrada sea menor al límite inferior.

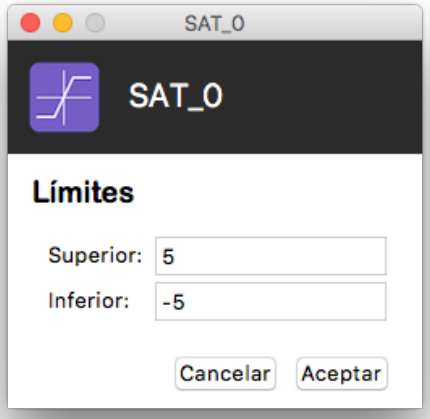

Figura 4.1.13. Ventana SAT.

Las opciones configurables al usuario son tanto el límite superior como el inferior. Ambos parámetros pueden ser reales positivos o negativos, y sus códigos en hexadecimal vienen dados por la tabla 4.1.4.

Tabla 4.1.4 Código hexadecimal para los parámetros del bloque SAT.

Parámetro	Código Hexadecimal
Límite Inferior	0x3030
Límite Superior	0x3031

Bloque SUM (SUMADOR)

Figura 4.1.14. Icono SUM.

El bloque SUM cuenta con 2 entradas y 1 salida, en donde la salida es la suma de ambas señales de entrada. El bloque no presenta opciones configurables.

Bloque RST (RESTADOR)

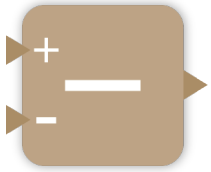

Figura 4.1.15. Icono RST.

El bloque RST cuenta con 2 entradas y 1 salida. La salida del bloque viene dado por la substracción de la señal de la entrada 2 a la entrada 1. Al ser un restador, el bloque no presenta opciones configurables y está limitado a 2 señales de entrada.

Bloque RE (RETROALIMENTACIÓN DE ESTADOS)

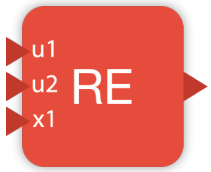

Figura 4.1.16. Icono RE.

El bloque RE puede contar con 3-6 entradas y 1-2 salidas. El número de entradas del bloque dependerá de la configuración del número de estados ingresados por el usuario. En cuanto al número de salidas, el usuario ingresa la cantidad de salidas requeridas. Dado que el número de entradas y salidas del bloque son configurables por el usuario, el icono del bloque RE puede variar a cualquiera de las configuraciones de la figura 4.1.17., sin incluir la configuración de retroalimentación de estado integral REI.

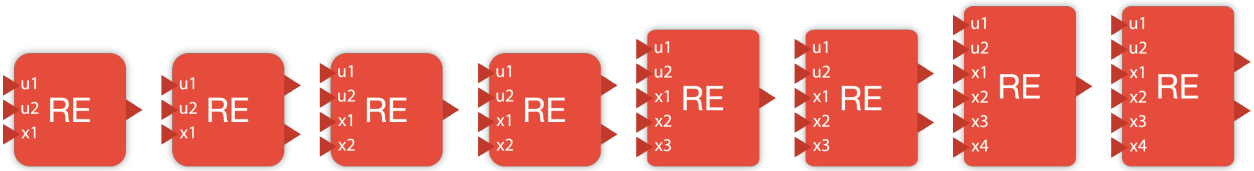

Figura 4.1.17. Distintas configuraciones del bloque RE.

El bloque RE tiene como parámetro configurable el tipo (Realimentación de estados RE o Realimentación de Estado Integral). Los parámetros configurables varían según el tipo seleccionado.

La estructura del bloque RE tiene como parámetros la ganancia K_0 (ganancia calculada para lograr un error de estado estacionario igual a cero) y la matriz de retroalimentación de estados K.

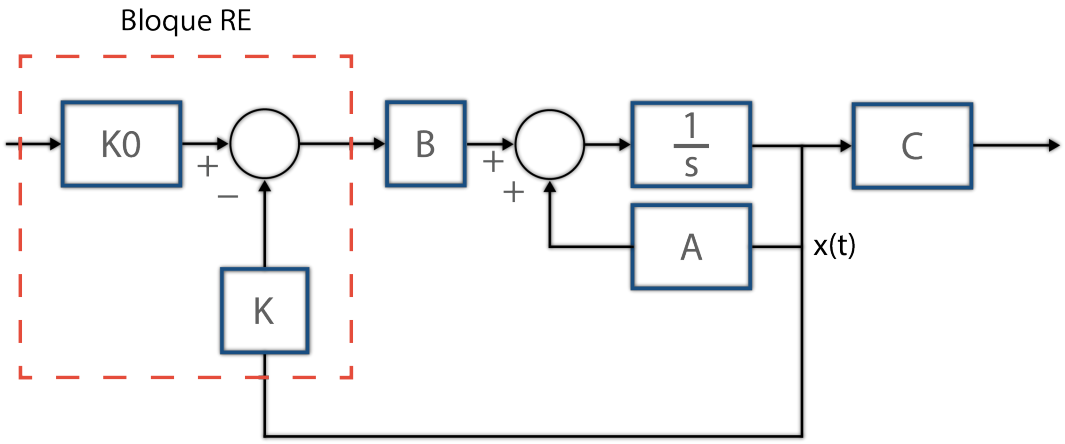

Figura 4.1.18. Estructura interna del bloque RE.

La estructura del bloque REI tiene como parámetros la matriz K y la matriz K_I. La retroalimentación por la salida de la planta viene dada con la selección de los estados correspondientes a cada salida.

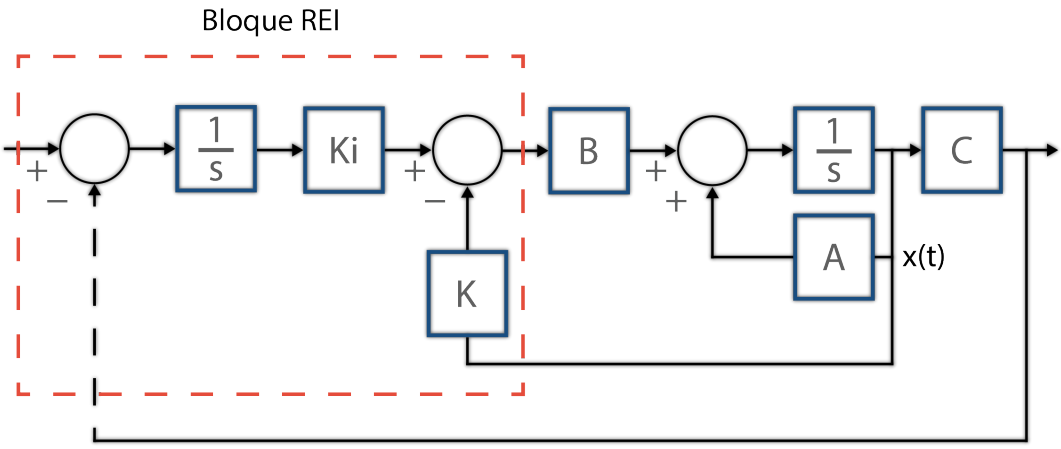

Figura 4.1.19. Estructura interna del bloque REI.

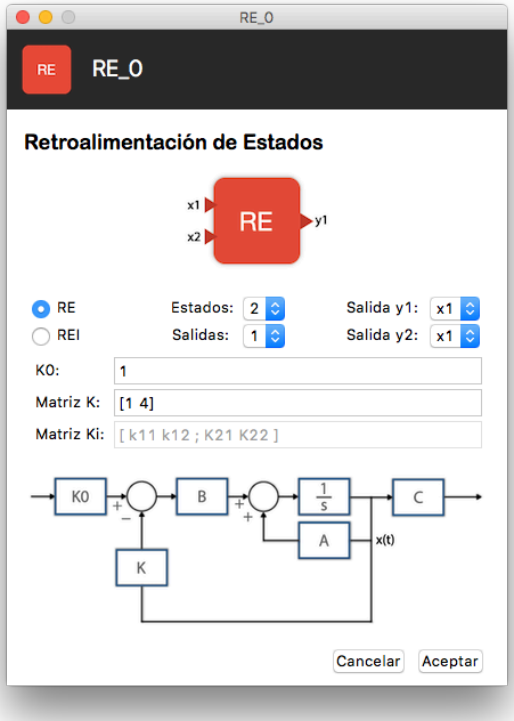

Figura 4.1.20. Ventana RE.

El código hexadecimal de los parámetros del bloque RE viene dado por la tabla 4.1.5.

Tabla 4.1.5 Código hexadecimal para los parámetros del bloque RE.

Parámetro	Código Hexadecimal
Tipo: RE, REI	0x3030
N (número de estados)	0x3031
M (número de salidas)	0x3032
Matriz K	0x3033
Matriz Ki	0x3034
Estado Salida y1	0x3035
Estado Salida y2	0x3036
K0	0x3037

Bloque FT (FUNCIÓN DE TRANSFERENCIA)

Figura 4.1.21. Icono FT.

El bloque FT puede contar con 1-2 entradas y 1 salida. El orden de la función de transferencia es configurable por el usuario y puede variar entre orden 1 y orden 2. Como parámetros, además del orden, se solicita el numerador y el denominador de la función de transferencia en forma de arreglo. La retroalimentación es opcional, y, al estar seleccionada, la segunda entrada del icono actúa como la retroalimentación del bloque. La señal de salida es la respuesta de la función de transferencia ante la señal de entrada del bloque.

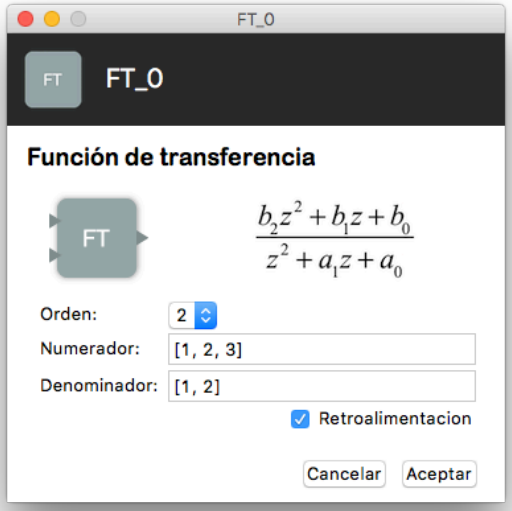

Figura 4.1.22. Ventana FT.

Los parámetros configurables del bloque de función de transferencia son interpretados por el código hexadecimal, mostrados en la tabla 4.1.6.

Tabla 4.1.6 Código hexadecimal para los parámetros del bloque FT.

Parámetro	Código Hexadecimal
Tipo FT: (con o sin retroalimentación)	0x3030
Orden	0x3031
Numerador	0x3032
Denominador	0x3033

Bloque QDEC (CODIFICADOR DE CUADRATURA)

Figura 4.1.23. Icono QDEC.

El bloque QDEC cuenta con 1 entrada y 1 salida. El bloque no presenta opciones configurables por el usuario y modela a un codificador de cuadratura de la unidad de hardware.

Figura 4.1.24. Ventana QDEC.

Bloque PWM

Figura 4.1.25. Icono PWM.

El bloque PWM cuenta con 1 entrada y 1 salida. El bloque de modulación por ancho de pulsos modela al bloque PWM de la unidad de hardware. El bloque PWM tiene como parámetro el periodo en microsegundos con código hexadecimal 0x3031.

Figura 4.1.26. Ventana PWM.

Bloque DAC

Figura 4.1.27. Icono DAC.

El bloque DAC cuenta con 1 entrada y 1 salida. Modela al bloque Convertidor Digital Analógico de la unidad de hardware y la señal de salida es la señal analógica correspondiente a la señal digital en la entrada. Presenta como parámetros configurables al rango de tensión y al tiempo de muestreo en milisegundos. Los códigos hexadecimales de cada parámetro configurable se presentan en la tabla 4.1.7.

Tabla 4.1.7 Código hexadecimal para los parámetros del bloque DAC

Parámetro Rango	Código Hexadecimal
0-10 V	0x30
0-5 V	0x31
0-2 V	0x32
0-1 V	0x33

Parámetro Rango	Código Hexadecimal
±1 V	0x34
±2 V	0x35
±5 V	0x36
±10 V	0x37

Bloque FLT (FILTRO)

Figura 4.1.28. Icono FLT.

El bloque ADC cuenta con 1 entrada y 1 salida. Su función es realizar una ecuación de diferencias y sus parámetros son los coeficientes de un filtro, por lo que su salida es la señal filtrada de la señal de entrada.

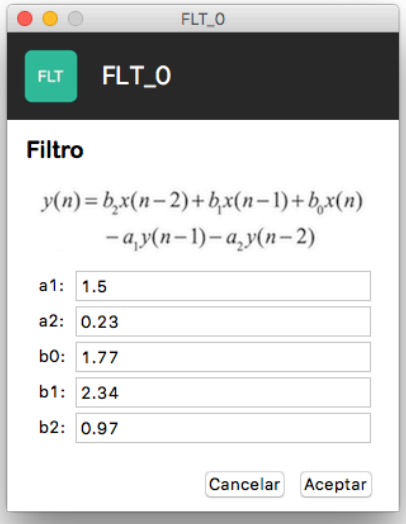

Figura 4.1.29. Ventana FLT.

Tabla 4.1.8 Código hexadecimal para los parámetros del bloque FLT.

Parámetro	Código Hexadecimal
Coeficiente a1	0x3030
Coeficiente a2	0x3031
Coeficiente b0	0x3032
Coeficiente b1	0x3033
Coeficiente b2	0x3034

Bloque SCOPE (OSCILOSCOPIO)

Figura 4.1.30. Icono SCOPE.

El bloque SCOPE cuenta con 1 entrada y 1 salida. Su propósito de funcionamiento es la captura de datos, en donde la señal de entrada es capturada y luego es transmitida sin modificación por la salida del bloque. El bloque SCOPE guarda la señal de entrada en un vector de datos para luego ser graficados en una gráfica contra tiempo. Su única función es por motivos de simulación y su funcionamiento no es transmitido a la unidad de hardware.

Bloque STEP (ESCALÓN)

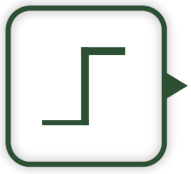

Figura 4.1.31. Icono STEP.

El bloque STEP cuenta con 1 salida. La salida corresponde a la señal escalón generada según las opciones ingresadas por el usuario. Estos valores incluyen el tiempo inicial del escalón, el valor inicial y el valor final que tomará la señal escalón.

Bloque TRP (TRAPECIO)

Figura 4.1.32. Icono TRP.

El bloque TRP cuenta con 1 salida. Su salida corresponde a una señal trapecio que es generada según los parámetros ingresados por el usuario: valor final, tiempo inicial, tiempo de rampa y tiempo en alto.

Tabla 4.1.9 Código hexadecimal para los parámetros de los módulos.

MÓDULO	PARÁMETROS	CÓDIGO HEXADECIMAL	VALOR
ADC	Rango	0x3030	Opciones de 0-7
	Tiempo de muestreo	0x3031	Entero
PID	Tipo	0x3030	Opciones de 0-2
	P	0x3031	Entero/Float
	I	0x3032	Entero/Float
	D	0x3033	Entero/Float
	c	0x3034	Entero
	b	0x3035	Entero
	N	0x3036	Entero/Float
	Método integrador	0x3037	Opciones de 1-3
	Método Filtro	0x3038	Opciones de 1-3
	Tiempo de muestreo	0x3039	Entero

MÓDULO	PARÁMETROS	CÓDIGO HEXADECIMAL	VALOR
PWM	Periodo	0x3031	Entero
RE	Tipo	0x3030	Opciones de 0-1
	Número de estados	0x3031	Entero
	Número de salidas	0x3030	Entero
	Matriz K	0x3031	Matriz
	Matriz Ki	0x3032	Matriz
	Estado y1	0x3033	Opciones de 1-4
	Estado y2	0x3034	Opciones de 1-4
	K0	0x3035	Entero/Float
CTE	Valor constante	0x3030	Entero/Float
FT	Tipo: Retroalimentación	0x3030	Opciones de 0-1
	Orden	0x3031	Opciones de 1-2
	Numerador	0x3032	Vector
	Denominador	0x3033	Vector
SAT	Límite inferior	0x3030	Entero/Float
	Límite superior	0x3031	Entero/Float
QDEC	Tiempo de muestreo	0x3030	Entero
DAC	Rango	0x3030	Opciones de 0-7
	Tiempo de muestreo	0x3031	Entero
FLT	Variable a1	0x3030	Entero/Float
	Variable a2	0x3031	Entero/Float
	Variable b0	0x3032	Entero/Float
	Variable b1	0x3033	Entero/Float
	Variable b2	0x3034	Entero/Float
DER	-	-	-
SUM	-	-	-
RST	-	-	-

4.1.5 Estructura básica de los bloques

Cada objeto cuenta con ciertas características que son comunes para todos los tipos de objetos dentro del workspace. El código 4.1 muestra la clase NodoADC (objeto ADC) y sus propiedades. Al inicializarse, cada objeto presenta un identificador. Este identificador es un número entero y empieza desde el cero. El identificador es guardado en memoria bajo la variable dato. El número de identificación permite rastrear un bloque en caso de una futura modificación.

El tipo de bloque es guardado como la variable tipo, y es la variable encargada de distinguir entre los distintos tipos de bloques en el workspace. Ejemplo de tipos son ADC, PID, SAT, etc. Para el ejemplo del código 4.1, el tipo es un "ADC". Además del identificador y del tipo de bloque, se asigna a cada objeto un nombre, el cual está compuesto por su tipo y el número de identificación.

El objeto ingresado por el usuario debe contener una posición 'x' y una posición 'y' dentro del workspace, el cual permitirá conocer la ubicación del objeto en caso de que el usuario arrastre el bloque a una nueva ubicación.

Código 4.1. Clase NodoADC

```
class NodoADC:
    def __init__(self, id_ADC, tupla_listas, px, py):

        self.dato = id_ADC
        self.tipo = "ADC"
        self.nombre = "ADC_"+str(id_ADC)
        self.posicionx = px
        self.posiciony = py
        self.tipoNetlist = "A"
        ...
        # Entrada y Salida
        self.entrada = None
        self.salida = None

        self.siguiente = None
```

Según el tipo de bloque, el número de entradas y salidas puede variar. En el caso del bloque ADC, el módulo cuenta con 1 entrada y 1 salida, por lo que ambas terminales se inicializan en None para indicar que aún no hay conexión en ambas terminales.

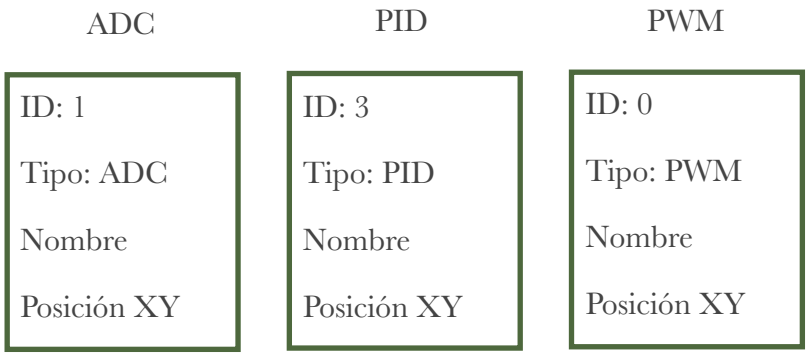

Figura 4.1.33. Propiedades de los tipos de objetos.

Código 4.2. Asignación de acciones de usuario al objeto ADC

```
temp.window.tag_bind(temp.nombre, "<DoubleButton-1>", temp.dobleClickADC)
temp.window.tag_bind(temp.nombre, "<Button-1>", temp.clickIzquierdoADC)
temp.window.tag_bind(temp.nombre, "<B1-Motion>", temp.enMovimientoADC)
temp.window.tag_bind(temp.nombre, "<Button-2>", temp.clickDerechoADC)
```

Luego de la inicialización del objeto, es necesario la creación del *bind*, o unión, a funciones de acciones ingresadas por el usuario. Estas acciones pueden ser 'Doble click', 'Liberación del puntero', 'Un solo click', 'Click y arrastre', 'Click derecho', etc. Las funciones son luego unidas a cada comando para lograr la acción deseada por el usuario.

4.1.6 Principales funciones de los objetos

dobleclickADC

La función dobleclickADC es utilizada cuando el usuario ingresa la acción de 'doble click' sobre el icono del objeto. Su función es abrir la ventana de opciones configurables al usuario y mostrar las opciones ingresadas por el mismo. La ventana de opciones de configuración cuenta con botones comunes como 'aceptar' y 'cancelar', 'cerrar ventana', botones de selección del rango y el ingreso de valores por el usuario para el tiempo de muestreo. Si bien estas configuraciones son para el bloque ADC, todos los distintos módulos presentan opciones similares con mismos elementos en común.

clickIzquierdoADC

Su función es determinar la posición del puntero dentro del ícono para reconocer si el usuario está creando una nueva conexión en la entrada o en la salida del bloque. Una vez que el usuario realiza un 'click' sobre el ícono, este informa a los demás bloques que se está creando una conexión.

Código 4.3. clickIzquierdoADC

```
if 38 > self.dx > 28 and 6 > abs(self.dy):  # Salida ADC

    if self.creandoLinea is not None and self.salida is None and not self.tipoCreandoLinea:
        tupla = (self.posicionx + 32, self.posiciony - 1)
        self.salida = self.creandoLinea
        self.tupla_listas[2].ultimoNodo(self.creandoLinea, self.nombre, tupla)

    if self.creandoLinea is None and self.salida is None:
        coord = [self.posicionx + 32, self.posiciony-1]
        self.tupla_listas[2].agregar(self.tupla_listas, self.nombre, coord, True)
        self.salida = self.creandoLinea

if -38 < self.dx < -28 and 6 > abs(self.dy):  # Entrada ADC

    if self.creandoLinea is not None and self.entrada is None and self.tipoCreandoLinea:
        tupla = (self.posicionx-32, self.posiciony-1)
        self.entrada = self.creandoLinea
        self.tupla_listas[2].ultimoNodo(self.creandoLinea, self.nombre, tupla)
```

```
if self.creandoLinea is None and self.entrada is None:
    coord = [self.posicionx - 32, self.posiciony-1]
    self.tupla_listas[2].agregar(self.tupla_listas, self.nombre, coord, False)
    self.entrada = self.creandoLinea
```

La función del código 4.3. es determinar si el 'click' del usuario ha sido en la salida o entrada del bloque. self.dx es la localidad del puntero (en píxeles) sobre el ícono.

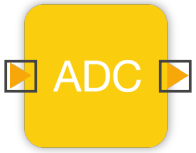

Figura 4.1.34. Posición de las terminales de conexión del bloque ADC.

Una vez determinada la ubicación del 'click' sobre el ícono, se procede a determinar si existe o no una conexión en la entrada o salida del bloque. De no existir una conexión en la terminal correspondiente, se procede a crear una nueva conexión. Si el usuario ya estaba realizando una conexión y realizó un 'click' sobre la terminal, se procede a finalizar la conexión entre los dos bloques llamando a la función ultimoNodo.

enMovimientoADC

Función encargada en caso de arrastrar el icono del bloque dentro del worksapce. Al arrastrar el bloque, es necesario el cálculo de la nueva posición del objeto, así como de mover la ubicación de la imagen del ícono y la etiqueta con su nombre a la nueva posición deseada. En caso de existir conexiones atadas al bloque, es necesario el nuevo cálculo de la ubicación de las líneas que conforman la conexión en las terminales del objeto. Esta última función se aprecia en el código 4.4. bajo el nombre moverLinea.

Código 4.4. enMovimientoADC

```
self.window.move(self.nombre,
                 event.x-self.posicionx-self.dx,
                 event.y-self.posiciony-self.dy)
```

```
self.window.move(self.labelNombre,
                 event.x-self.posicionx-self.dx,
                 event.y-self.posiciony-self.dy)

# Movimiento de la Conexión
# ----------------------------------
if self.entrada is not None and self.entrada != self.salida:
    self.tupla_listas[2].moverLinea(self.entrada, self.nombre,
                                    event.x-self.posicionx-self.dx,
                                    event.y-self.posiciony-self.dy)

if self.salida is not None:
    self.tupla_listas[2].moverLinea(self.salida, self.nombre,
                                    event.x-self.posicionx-self.dx,
                                    event.y-self.posiciony-self.dy)
# ----------------------------------
self.posicionx = event.x - self.dx
self.posiciony = event.y - self.dy
```

4.1.7 Listas enlazadas

Para la creación de los objetos en memoria se utilizó listas enlazadas. Las listas enlazadas permiten la implementación de estructuras de datos, y la constante modificación de los objetos en el workspace hace más eficiente el recorrido en memoria de una lista que el de un arreglo o vector.

Para la implementación de listas enlazadas se parte de la creación de un nodo cabeza, el cual apunta a None. Para la creación de una lista, es necesario cambiar el puntero de None a otro nodo del mismo tipo para generar así la lista enlazada. La ilustración de este proceso se observa en la figura 4.1.35.

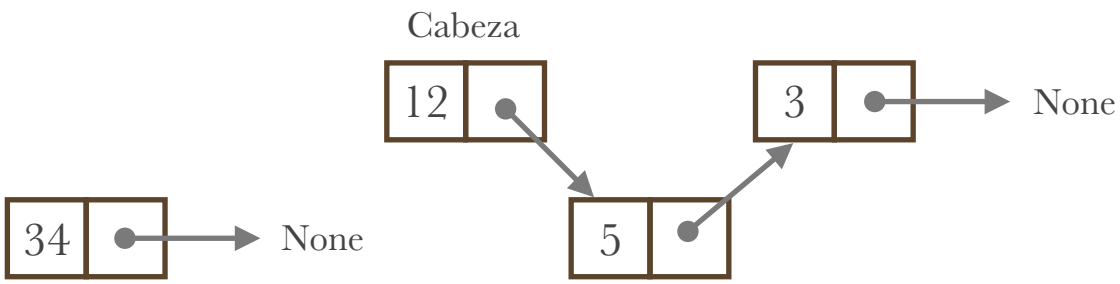

Figura 4.1.35. Creación de un nodo en una lista enlazada.

La creación del nodo 34 apunta a None y la lista enlazada conformada por 12, 5 y 3 no tiene relación con el nuevo nodo, por lo que es necesario cambiar el puntero del nodo 34 hacia la actual cabeza de la lista: nodo 12.

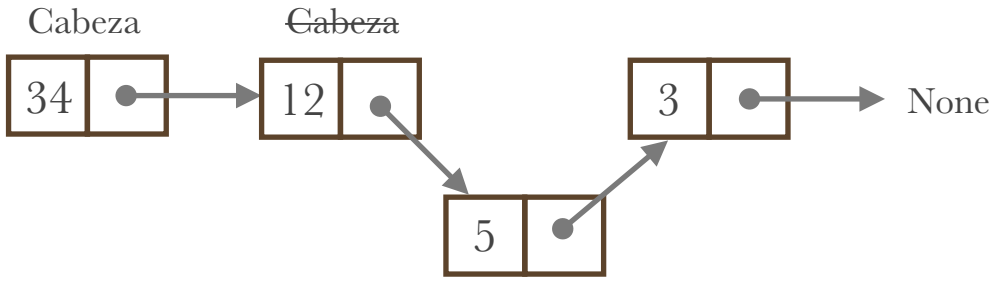

Figura 4.1.36. Asignación del nuevo nodo cabeza.

Una vez que el nuevo nodo apunta a la cabeza de la lista enlazada es necesario cambiar el nuevo nodo a nodo cabeza para lograr mantener la referencia correcta a la lista.

La generación de una lista enlazada para cada tipo de bloque logra un mejor ordenamiento de los bloques dentro del workspace, por lo que si es necesario acceder a un nodo de la lista, su búsqueda resulta ser muy fácil dentro de la lista enlazada correspondiente.

Código 4.5. Función para remover un elemento de la lista

```
def remover(self, nombre):
    actual = self.cabeza
    previo = None
    encontrado = False
    while not encontrado:
        if actual.nombre == nombre:
            encontrado = True
        else:
            previo = actual
            actual = actual.siguiente
    if previo == None:
        self.cabeza = actual.siguiente
    else:
        previo.siguiente = actual.siguiente
```

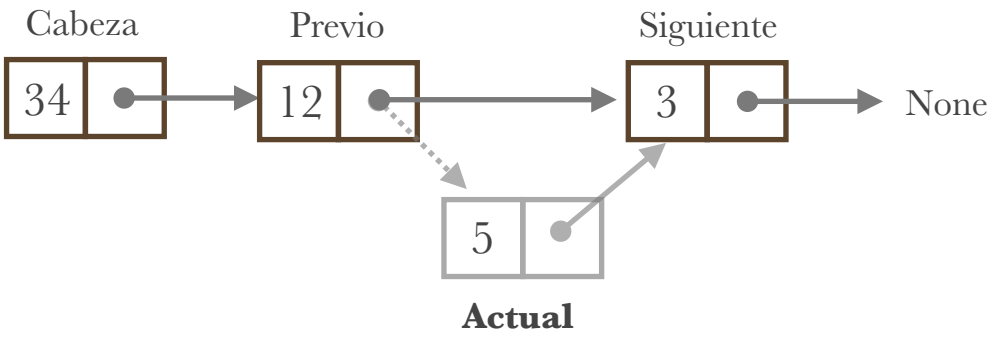

Figura 4.1.37. Remover elemento de lista enlazada.

La acción de remover un elemento de una lista enlazada se visualiza con la figura 4.1.37., y con el código 4.5. Una vez encontrado el objeto a eliminar (actual), se asigna al elemento anterior (previo) el puntero hacia el siguiente elemento que apunta actual, por lo que se remueve cualquier referencia hacia el nodo actual y así borrándolo de la lista enlazada.

4.1.8 Netlist

Para el debido registro de las comunicaciones de los bloques y módulos en el workspace es necesario la creación del netlist. La estructura del netlist se presenta en la figura 4.1.38.

Tipo	ID	Salida	Tipo	ID	Entrada
ADC	0	1	PID	1	1
Origen			**Destino**		

Figura 4.1.38. Estructura del NETLIST.

Cada trama del netlist está conformada por el bloque de origen y el bloque de destino de la conexión. En la figura 4.1.38., la estructura es la siguiente: la primer sección indica el tipo de bloque (ADC), seguido por el identificador de ese bloque (0) y el numero de salida (1) por el que está conectado al bloque de destino (PID). Por último, la sección de ID indica el identificador del bloque (ID = 1), seguido del número de entrada al que está conectado el bloque de destino.

Un ejemplo de NETLIST se obtiene con el siguiente sistema de control:

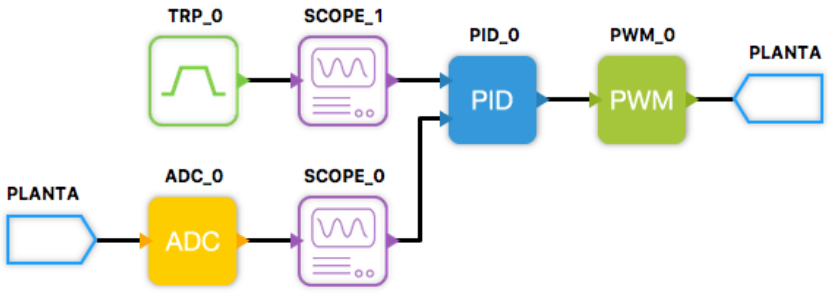

Figura 4.1.39. Ejemplo de sistema de control.

```
NETLIST
PWM,0,1,ENTP,0,1          # ENTP: Bloque Entrada a la planta
PID,0,1,PWM,0,1
SCOPE,1,1,PID,0,1
SCOPE,0,1,PID,0,2
ADC,0,1,SCOPE,0,1
SALP,0,1,ADC,0,1          # SALP: Bloque Salida de la planta
TRAPECIO,0,1,SCOPE,1,1
```

Al analizar el ejemplo anterior línea por línea del netlist generado por el sistema de control, se obtiene lo siguiente:

El PWM_0 está conectado por la salida 1 a ENTP_0, por la entrada 1 de ENTP. El PID_0 esta conectado por la salida 1 a PWM_0 en su entrada 1. El bloque SCOPE_1 está conectado por la salida 1 al bloque PID_0 por medio de la entrada 1. El bloque SCOPE_0 está conectado por la salida 1 al bloque PID_0 por medio de la entrada 2. ADC_0 está conectado por la entrada 1 con el bloque SCOPE_0 por medio de la entrada 1 de ADC_0. SALP_0 se conecta en su salida 1 con la entrada 1 del ADC_0 y, por último, el bloque TRAPECIO_0 se conecta en su salida 1 al bloque SCOPE_1 por la entrada 1.

Para el registro del Netlist, la información de cada conexión se guarda en una lista enlazada. Cada conexión guarda los tipos y nombres de los bloques involucrados en la conexión, generando así el Netlist de la configuración.

4.1.9 Archivo XML

Para guardar y abrir archivos se utilizó el lenguaje “Extensible Markup Language”, en donde cada bloque y conexión presenta su apartado de parámetros así como información básica del objeto dentro del Workspace. Para la generación del archivo, es necesario que el usuario presione el ícono de guardar y seleccione el directorio en donde

desea guardar el archivo XML. Una vez obtenido el directorio, la herramienta genera el archivo XML obteniendo los parámetros de los bloques e información de cada lista enlazada. Cada parámetro se encuentra dentro de dos indicadores para indicar el inicio y fin del parámetro dentro del documento:

<Nombre_Parámetro> Parámetro </Nombre_Parámetro>.

Una vez guardado el archivo, es posible recuperar la información al presionar el ícono de abrir y luego seleccionar el archivo deseado. Una vez obtenido el directorio del archivo, la herramienta lee línea por línea el archivo XML, y crea de nuevo los objetos correspondientes, asignando los parámetros guardados en el archivo.

Ejemplo de archivo XML generado por la herramienta de diseño:

```
<?xml version='1.0' encoding='utf-8'?>
<Laboratorio_Control_Automatico>
  <Configuracion>
        <Dispositivo>/dev/cu.usbmodemFA131</Dispositivo>
        <Tiempo_Muestreo>20</Tiempo_Muestreo>
        <Numero_Muestras>1000</Numero_Muestras>
  </Configuracion>
  <Linea>
        <Nombre>Linea_6</Nombre>
        <Bloque_Inicio>TRAPECIO_0</Bloque_Inicio>
        <Bloque_Fin>SCOPE_1</Bloque_Fin>
        <Nodos>[302, 164, 319, 164, 319, 164, 340, 164]</Nodos>
  </Linea>
  <ADC>
        <Nombre>ADC_0</Nombre>
        <Posicion_X>269</Posicion_X>
        <Posicion_Y>274</Posicion_Y>
        <Entrada_1>Linea_0</Entrada_1>
        <Salida_1>Linea_1</Salida_1>
        <Variable_Rango>7</Variable_Rango>
```

```
        <Variable_Tiempo_Muestreo>20</Variable_Tiempo_Muestreo>
    </ADC>
    <PID>
        <Nombre>PID_0</Nombre>
        <Posicion_X>474</Posicion_X>
        <Posicion_Y>178</Posicion_Y>
        <Entrada_1>Linea_3</Entrada_1>
        <Entrada_2>Linea_2</Entrada_2>
        <Salida_1>Linea_4</Salida_1>
        <Variable_Tipo_Pid>1</Variable_Tipo_Pid>
        <Variable_Parametro_P>10</Variable_Parametro_P>
        <Variable_Parametro_I>0</Variable_Parametro_I>
        <Variable_Parametro_D>20</Variable_Parametro_D>
        <Variable_Parametro_C>1</Variable_Parametro_C>
        <Variable_Parametro_B>1</Variable_Parametro_B>
        <Variable_Parametro_N>1</Variable_Parametro_N>
        <Variable_Metodo_Integrador>Backward Euler
        </Variable_Metodo_Integrador>
        <Variable_Metodo_Filtro>Forward Euler</Variable_Metodo_Filtro>
        <Variable_Tiempo_Muestreo>20</Variable_Tiempo_Muestreo>
    </PID>
    <PWM>
        <Nombre>PWM_0</Nombre>
        <Posicion_X>576</Posicion_X>
        <Posicion_Y>178</Posicion_Y>
        <Entrada_1>Linea_4</Entrada_1>
        <Salida_1>Linea_5</Salida_1>
        <Variable_Resolucion>2</Variable_Resolucion>
        <Variable_Periodo>500</Variable_Periodo>
    </PWM>
</Laboratorio_Control_Automatico>
```

4.2 Simulación del sistema de control

La herramienta de diseño cuenta también con capacidad de simulación. La herramienta está diseñada para lograr la simulación de distintos sistemas de control, entre ellos: PID, I_PD, PI_D, realimentación de estados, realimentación de estado integral y

función de transferencia. La planta con capacidad de simulación es la planta "Ball & Beam". Para lograr la simulación de la planta, es necesario colocar los bloques de control y hacer click en el botón de simulación.

Para la opción de simulación, se añadieron dos bloques que simulan el comportamiento de señales de entrada o referencia. Estos bloques son el STEP (escalón) y TRAPECIO. Ambos bloques son utilizados como señal de entrada al sistema de control para lograr observar la respuesta del sistema correspondiente.

Además de los bloques STEP y TRAPECIO, se añadió un bloque SCOPE (bloque osciloscopio), el cual es utilizado en modo simulación para la captura de datos y la generación de un gráfico con los datos capturados. Los bloques SCOPE pueden insertarse en cualquier conexión como cualquier otro bloque, y no influyen en el comportamiento del sistema diseñado. Para lograr observar una señal en el tiempo, es necesario la colocación de al menos un bloque SCOPE.

Las señales capturadas por los bloques SCOPES fueron graficadas utilizando las bibliotecas matplotlib y NumPy.

4.2.1 Ball & Beam

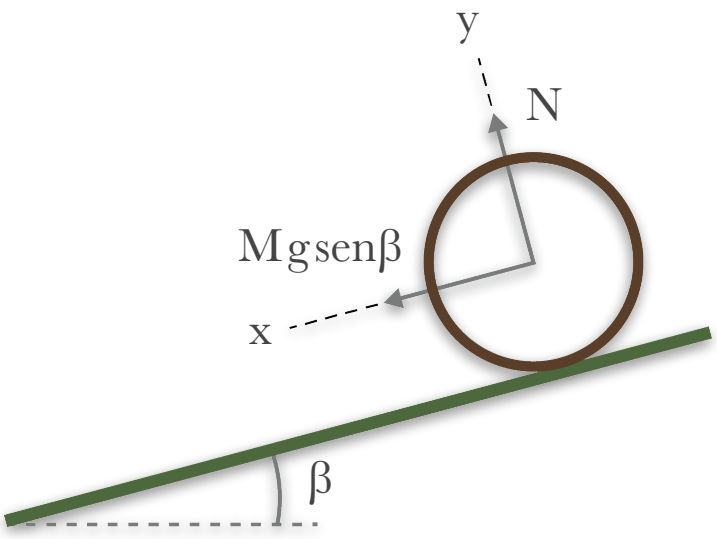

Figura 4.2.1. Diagrama Ball & Beam.

Para la obtención del modelo de la planta Ball & Beam se parte de que la posición de la bola en la barra dependerá del ángulo de inclinación β.

Se tienen las siguientes ecuaciones de movimiento,

$$\sum F = Ma_{cm} \tag{4.1}$$

$$\sum \tau = I_{cm}\alpha \tag{4.2}$$

en donde la sumatoria de fuerzas y torque viene dado por

$$\sum F = Mg\,\mathrm{sen}\beta - f = Ma_{cm} \tag{4.3}$$

$$\sum \tau = fR = I_{cm}\alpha \tag{4.4}$$

Partiendo de la relación entre la velocidad del centro de masa v_{cm} y su rapidez angular ω

$$v_{cm} = R\omega \tag{4.5}$$

se obtiene

$$a_{cm-x} = R\alpha_z \tag{4.6}$$

Dado que el momento de inercia de una esfera es $I_{cm} = \frac{2}{5}MR^2$, y despejando α de la ecuación (4.6), entonces

$$fR = \frac{2}{5}MRa_{cm-x} \tag{4.7}$$

Despejando f de (4.3) y (4.7) e igualando ambas ecuaciones, se obtiene

$$\frac{2}{5}Ma_{cm-x} = Mg\,\text{sen}\beta - Ma_{cm-x} \tag{4.8}$$

despejando a_{cm-x} y aproximando $\text{sen}\beta \approx \beta$ para valores pequeños de β

$$a_{cm-x} = \frac{5}{7}g\beta \tag{4.9}$$

Reemplazando $g = 9.8$ y haciendo $a_{cm-x} = \frac{d^2}{dt^2}x(t)$ y $\beta(t)$

$$\frac{d^2}{dt^2}x(t) = 7\beta(t) \tag{4.10}$$

Por último, aplicando la transformada de Laplace a ambos lados de la ecuación

$$s^2X(s) = 7B(s) \tag{4.11}$$

se obtiene entonces

$$\frac{X(s)}{B(s)} = \frac{7}{s^2} \tag{4.12}$$

4.2.1 Recorrido de árboles

Para la simulación de los sistemas de control dentro del workspace se utilizó el recorrido de árboles binarios. Sin importar el número de entradas y salidas, es necesario que cada entrada de cada bloque esté lista antes de poder realizar la función necesaria para generar la señal de salida correspondiente. El recorrido de árbol binario utilizado es el recorrido postorden, el cual se basa en recorrer en primer lugar, el sub-árbol izquierdo de cada nodo. En segundo lugar se recorre el sub-árbol derecho y por último se recorre el nodo raíz que posee dichos sub-árboles izquierdo y derecho.

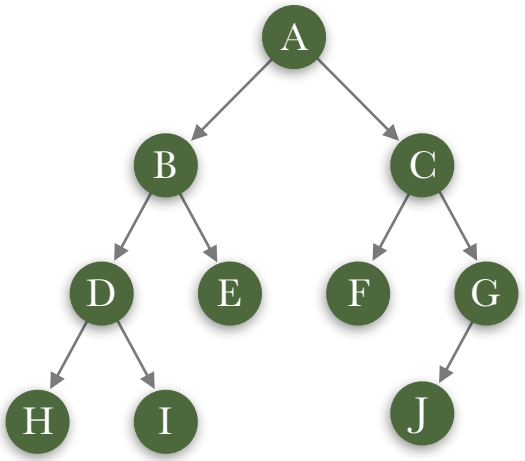

Figura 4.2.2. Árbol binario.

Para la figura del árbol binario de la figura 4.2.2., y en recorrido postorden, el recorrido es el siguiente: H, I, D, E, B, F, J, G, C, A. El principio de este recorrido es recorrer primeramente cada sub-árbol antes de realizar la función del nodo raíz. Dado que en un sistema de control el bloque raíz dependerá de las señales generadas por los bloques en sus entradas, es necesario recorrer en primer lugar esos bloques antes de generar la señal de salida.

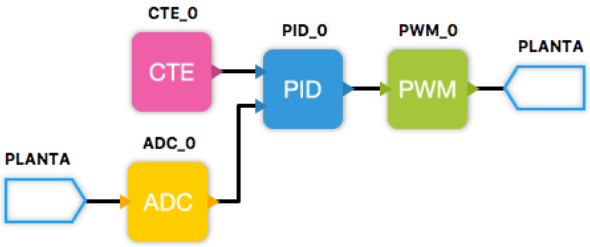

Figura 4.2.3. Sistema de control con 6 bloques.

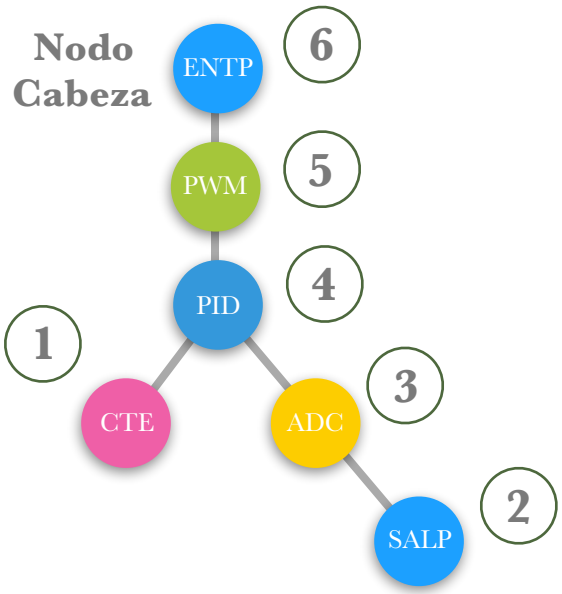

Figura 4.2.4. Árbol del sistema de control de la figura 4.2.3.

En el caso del sistema de la figura 4.2.4, el recorrido necesario para generar la simulación es el siguiente: CTE, SALP, ADC, PID, PWM, ENTP. (SALP: Salida de la Planta, ENTP: Entrada a la Planta). En este caso, el bloque PID debe esperar al valor retornado por CTE y por el valor retornado por ADC, sin embargo, ADC debe esperar al valor retornado por SALP para poder retornar su salida hacia el bloque PID.

Para encontrar el nodo raíz o cabeza superior de un árbol, se procede según los siguientes puntos:

- El bloque es de tipo ENTP y presenta un bloque en su entrada.
- El bloque presenta al menos una conexión en alguna de sus entradas, y además no presenta bloques en su salida.
- Los bloques sin entradas nunca serán nodos cabeza de un árbol.

Es importante resaltar que puede existir más de un árbol (sistema de control) dentro del workspace, y que un árbol debe estar compuesto de al menos dos bloques.

Código 4.6. Estructura básica para el recorrido de un árbol binario

```
def senal_simulacion(self, n, ts, conexion):

   if self.entrada1 is not None:
        sim_entrada1 = a.obtenerBloqueConexion(self.entrada1, self.nombre)
        ent1 = sim_entrada1.senal_simulacion(n, ts, self.entrada1)
    else:
        ent1 = 0.0

    if self.entrada2 is not None:
        sim_entrada2 = a.obtenerBloqueConexion(self.entrada2, self.nombre)
        ent2 = sim_entrada2.senal_simulacion(n, ts, self.entrada2)
    else:
        ent2 = 0.0
```

Mediante un método recursivo es posible el recorrido del árbol binario. En caso de existir una conexión en una entrada del bloque, se solicita a la entrada retornar el valor correspondiente, por otro lado, si no hay conexión en la entrada, el valor de dicha entrada es 0.

4.2.2 Simulación de los módulos

Los módulos dentro de la herramienta de diseño tienen capacidad de simulación. Cada bloque presenta una función específica, el cual generará una señal de salida según cambie la señal de entrada del bloque. Los bloques sin terminales de entrada generan señales de salida constantes (bloque CTE <constante>) o cambiantes en el tiempo (bloques STEP <escalón>, y TRP <trapecio>).

Bloque STEP

Código 4.7. Función Simulación Bloque STEP

```
def senal_simulacion(self, n, ts, conexion):

    if self.variable_escalon > ts:
        return self.variable_valor_inicial
    else:
```

```
        return self.variable_valor_final
```

La estructura básica de la función simulación se presenta en el código 4.7. La función recibe como parámetros al número de muestra “n”, el tiempo “ts” y la conexión con el siguiente bloque “conexión” a quien debe retornar el valor del escalón. Al ser una señal escalón, el bloque puede retornar dos posibles valores: variable_valor_inicial y variable_valor_final. Ambos parámetros son los configurados e ingresados por el usuario desde la interfaz gráfica. Si el tiempo ts es mayor al tiempo de inicio del escalón, la función retorna el escalar variable_valor_inicial, caso contrario, retorna variable_valor_final.

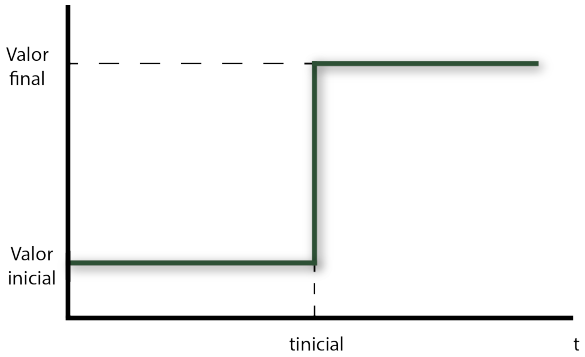

Figura 4.2.5. Función escalón.

Bloque TRAPECIO

Código 4.8. Función Simulación Bloque TRAPECIO

```
    def senal_simulacion(self, n, ts, conexion):

        if n == 0: # Inicialización de las variables
            self.t1 = self.variable_tiempo_inicio
            self.t2 = self.t1 + self.variable_tiempo_rampa
            self.t3 = self.t2 + self.variable_tiempo_alto
            self.t4 = self.t3 + self.variable_tiempo_rampa
            self.m = 0
            self.n = 0
            return 0
```

```
if n == 1:
    self.t_muestreo = ts
    if self.variable_tiempo_rampa != 0:
        self.m = self.variable_amplitud / ((self.t2 - self.t1) / self.t_muestreo)
    else:
        self.m = self.variable_amplitud
    self.n = 0

if ts < self.t1:
    return 0


if self.t1 <= ts < self.t2:
    b = self.m * self.n
    self.n = self.n + 1
    return b

if self.t2 <= ts < self.t3:
    return self.variable_amplitud

if self.t3 <= ts < self.t4:
    b = self.m * self.n
    self.n = self.n - 1
    return b

if self.t4 <= ts:
    return 0
```

El código 4.8. muestra la generación de la señal trapecio para el bloque TRP. Al igual que el bloque STEP, la función "senal_simulacion" recibe los parámetros numero de muestra "n", tiempo "ts" y el siguiente bloque "conexión".

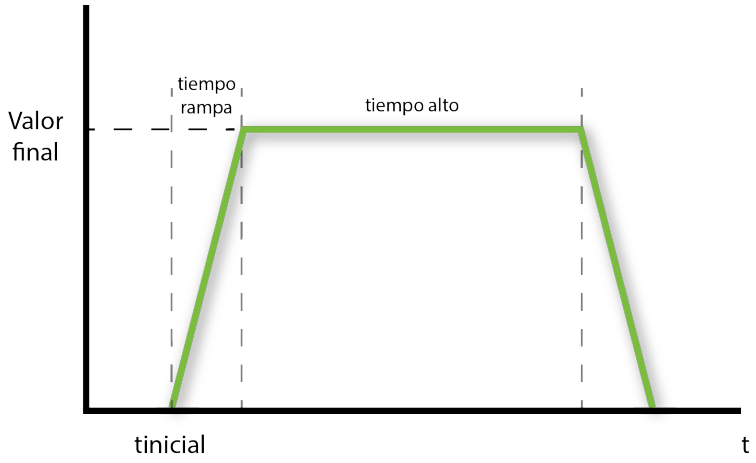

Figura 4.2.6. Señal trapecio.

El código señal_simulación genera la señal trapecio mostrada en la figura 4.2.6., en donde existe un tiempo inicial, tiempo de rampa y tiempo en alto. Según el parámetro ts recibido por la función simulación, se procede a calcular el valor de salida respectivo a la señal trapecio.

Bloque PID

El bloque cuenta con capacidad de simulación del controlador PID y sus variaciones I_PD y PI_D. Para la simulación del bloque PID, tanto el método integrador como método filtro pueden seleccionarse en "Backward Euler", "Forward Euler" y "Trapezoidal".

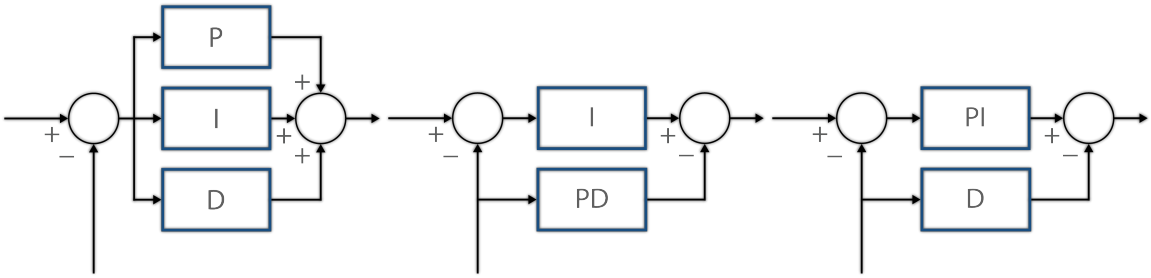

Figura 4.2.7. Control PID, I_PD y PI_D.

Para la selección del tipo de control, el usuario elige entre las distintas configuraciones del PID, en donde según la opción elegida, se implementa la siguiente ecuación con los distintos valores de las constantes b y c.

$$PID(z) = P(z)[bR(z) - Y(z)] + I(z)[R(z) - Y(z)] + D(z)[cR(z) - Y(z)] \tag{4.13}$$

El tipo PID presenta ambos valores de b y c iguales a 1. La variación I_PD tiene como valores b y c ambos iguales a 0 y, por último, la variación PI_D presenta valores b y c iguales a 1 y 0 respectivamente.

Código 4.9. Función Simulación Bloque PID

```
def senal_simulacion(self, n, ts, conexion):


   ...
    if self.entrada1 is not None:
        sim_entrada1 = a.obtenerBloqueConexion(self.entrada1, self.nombre)
        ent1 = sim_entrada1.senal_simulacion(n, ts, self.entrada1)
    else:
        ent1 = 0.0

    if self.entrada2 is not None:
        sim_entrada2 = a.obtenerBloqueConexion(self.entrada2, self.nombre)
        ent2 = sim_entrada2.senal_simulacion(n, ts, self.entrada2)
    else:
        ent2 = 0.0
    ...
```

La función simulación es la encargada de encontrar bloques en las entradas del módulo PID. En caso de encontrar un bloque en la conexión de la entrada '1', el bloque llama a la función simulación de ese bloque para que retorne de forma recursiva un valor que será la entrada al bloque PID. En caso de no existir una conexión en alguna de las dos entradas, entonces el valor será de cero para dicha entrada.

Bloque RE

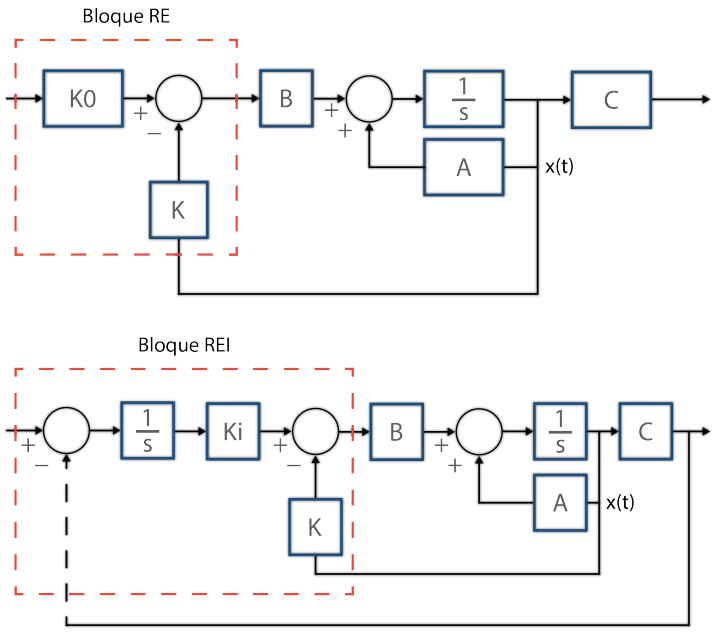

Figura 4.2.8. Diagrama interno de los bloques RE y REI.

La simulación del sistema de control por realimentación de estados y realimentación de estado integral se logra mediante el bloque RE. Las entradas 1 y 2 del bloque corresponden a las entradas del sistema de control. Las entradas 3, 4, 5 y 6 corresponden a los estados actuales de la planta.

Para la simulación del bloque RE, la señal de referencia es multiplicada por un escalar K0. Luego, la señal **Kx**, donde **x** es el vector de estados, es retroalimentada y restada a la señal de referencia, obteniendo entonces la siguiente señal de control:

$$\mathbf{u} = K_0\mathbf{r} - \mathbf{Kx} \tag{4.14}$$

En el caso de un bloque REI, existe un integrador para la señal $\mathbf{r} - \mathbf{Cx}$, esta señal es luego restada por $-\mathbf{Kx}$, obteniendo la señal de control

$$\mathbf{u} = \mathbf{Ki}\xi - \mathbf{Kx} \tag{4.15}$$

En donde ξ es la señal de salida del integrador.

La simulación del bloque RE y REI viene dado entonces por el código 4.10.

Código 4.10. Función Simulación Bloque RE de 2 estados

```
...
if self.variable_tipo == 1: # Tipo RE

    matriz_x = np.array([[ent3], [ent4]])
    matriz_k = np.matrix(self.variable_matriz_k)

    u1 = ent1 * self.variable_k0
    u2 = matriz_k.dot(matriz_x)
    u2 = u2.item(0)

    return u1-u2


if self.variable_tipo == 2:  # Tipo REI

    self.x_n = ent1 - self.y_n  # self.y_n -> Salida de la Planta

    matriz_ki = np.matrix(self.variable_matriz_ki)
    ki = matriz_ki.item(0)
    ts = 0.02

    self.u1_n = self.u1_n_1 + ki * ts * self.x_n_1

    matriz_x = np.array([[ent3], [ent4]])
    matriz_k = np.matrix(self.variable_matriz_k)

    self.u2_n = matriz_k.dot(matriz_x)
    self.u2_n = self.u2_n.item(0)

    u = self.u1_n - self.u2_n
    self.x_n_1 = self.x_n
    self.u1_n_1 = self.u1_n

    return u
```

Para el bloque RE de dos estados, la matriz de estados x viene dada por las entradas 3 y cuatro, es decir, estado 1 y estado 2. Una vez obtenida la matriz x, se multiplica con la

matriz k ingresada por el usuario. El bloque entonces retorna $K_0\mathbf{r} - \mathbf{Kx}$, siendo r la señal obtenida en la entrada 1 del bloque.

Para el caso del tipo REI, se implementa un integrador a la señal **r – Cx**. La señal **Cx** es conocida por parámetros ingresados por el usuario.

Bloque FT

El bloque FT simula la función de transferencia de grado 2

$$\frac{Y(z)}{X(z)} = \frac{b_2 z^2 + b_1 z + b_0}{z^2 + a_1 z + a_0} \tag{4.16}$$

con ecuación de diferencias

$$y(n) = -a_0 y(n-1) + b_1 x(n) + b_0 x(n-1) \tag{4.17}$$

y la función de transferencia de grado 1

$$\frac{Y(z)}{X(z)} = \frac{b_1 z + b_0}{z + a_0} \tag{4.18}$$

con ecuación de diferencias

$$y(n) = -a_1 y(n-1) - a_0 y(n-2) + b_2 x(n) + b_1 x(n-1) + b_0 x(n-2) \tag{4.19}$$

La entrada 1 del bloque corresponde a la señal $x(n)$ de la ecuación de diferencias, mientras que la señal $y(n)$ representa la salida del bloque. Su implementación está dada por el código 4.11.

Código 4.11. Función Simulación Bloque FT

```
if self.variable_orden == 1:
# Grado 1
# y(n) = -a0 * y(n - 1) + b1 * x(n) + b0 * x(n - 1)
    self.y_n = -float(den[0])*self.y_n_1+float(num[0])*self.x_n+float(num[1])*self.x_n_1
    self.y_n_1 = self.y_n
    self.x_n_1 = self.x_n
else:
```

```
        # Grado 2
        # y(n) = -a1 * y(n - 1) - a0 * y(n - 2) + b2 * x(n) + b1 * x(n - 1) + b0 * x(n - 2)
        self.y_n = -float(den[0]) * self.y_n_1 - float(den[1]) * self.y_n_2 + float(num[0]) *
                   self.x_n + float(num[1]) * self.x_n_1 + float(num[2]) * self.x_n_2
        ...
        self.y_n_2 = self.y_n_1
        self.y_n_1 = self.y_n
        self.x_n_2 = self.x_n_1
        self.x_n_1 = self.x_n

    return float(self.y_n)
```

La variable **den** corresponde al denominador que contiene los parámetros $[a_1, a_0]$, mientras que la variable **num** contiene los parámetros $[b_2, b_1, b_0]$.

Bloque FLT

El bloque Filtro simula la ecuación de diferencias:

$$y(n) = b_2 x(n-2) + b_1 x(n-1) + b_0 x(n) - a_1 y(n-1) - a_2 y(n-2) \tag{4.20}$$

Los parámetros a1, a2, b0, b1 y b2 son configurables por el usuario y son ingresados desde la interfaz gráfica. La entrada 1 del bloque corresponde a la señal $x(n)$, mientras que la salida es la señal $y(n)$.

Bloque SAT

El bloque SAT, simula el comportamiento de un saturador, en donde la salida del sistema se satura en caso de superar los límites superior o inferior establecidos por el usuario.

Código 4.12. Función Simulación Bloque SAT

```
def senal_simulacion(self, n, ts, conexion):
    ...
    if self.entrada is not None:
        sim_entrada1 = a.obtenerBloqueConexion(self.entrada, self.nombre)
```

```
        ent1 = sim_entrada1.senal_simulacion3(n, ts, self.entrada)

        if ent1 > self.variable_limite_superior:
            return self.variable_limite_superior

        if ent1 < self.variable_limite_inferior:
            return self.variable_limite_inferior

        return ent1
    else:
        return 0
```

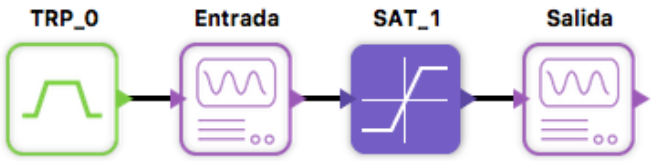

Figura 4.2.9. Sistema con bloque saturador.

El sistema de la figura 4.2.9. corresponde a una entrada de forma trapecio a un bloque saturador. La amplitud de la señal trapecio es 1, y el bloque saturador tiene como límite superior una magnitud de 0.8. La señal de entrada y salida del bloque son capturadas por 2 bloques SCOPE.

Bloques SUM (SUMADOR) y RST (RESTADOR)

La simulación del bloque SUM consiste en retornar la suma de las dos señales obtenidas en las entradas del bloque, es decir, la entrada 1 y la entrada 2. El resultado es retornado por la terminal de salida hacia la conexión con el bloque siguiente. Por otro lado, el bloque RST resta ambas señales de entrada y la señal de salida es el resultado de esta operación.

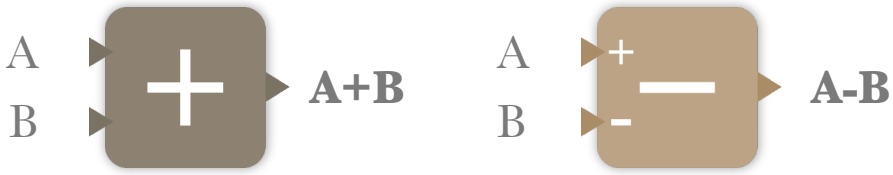

Figura 4.2.10. Función de los bloques sumador y restador.

Bloques ENTP (Entrada a la PLANTA) y SALP (Salida de la PLANTA)

Figura 4.2.11. Bloques SALP (izquierda) y ENTP (derecha).

Los bloques ENTP y SALP son los encargados de generar la simulación de la planta Ball & Beam. Cada bloque posee una sola terminal: 1 salida para SALP y 1 entrada en el caso de ENTP. La señal generada por el bloque ENTP es transmitida al bloque SALP en el próximo recorrido del árbol. Además de generar la señal de salida de la planta, el bloque también calcula los estados siguientes de la planta Ball & Beam.

$$\mathbf{x}(k+1) = \mathbf{A_d}\mathbf{x}(k) + \mathbf{B_d}u(k) \tag{4.21}$$

$$y(k) = \mathbf{C_d}\mathbf{x}(k) + \mathbf{D_d}u(k) \tag{4.22}$$

Código 4.13. Función Simulación Bloque SAT

```
def senal_simulacion(self, n, ts, conexion):

    if n == 0:
        self.y_n = 0
        self.y_n_1 = 0
        self.y_n_2 = 0
        self.x_n = 0
        self.x_n_1 = 0
        self.x_n_2 = 0
        self.ex1_n = 0
        self.ex1_n_1 = 0
        self.ex2_n = 0
        self.ex2_n_1 = 0

        self.matriz_x = 0
    ...
    if self.entrada is not None:
        sim_entrada1 = a.obtenerBloqueConexion(self.entrada, self.nombre)
        ent1 = sim_entrada1.senal_simulacion3(n, ts, self.entrada)
```

```
        else:
            ent1 = 0

        self.x_n = ent1

        # y(n) = 2*y(n-1) - y*(n-2) + 0.00000525*x(n) + 0.0000105*x(n-1) + 0.00000525*x(n-2)
        self.y_n = 2 * self.y_n_1 - self.y_n_2 + 0.00000525 * self.x_n + 0.0000105 * self.x_n_1
                    + 0.00000525 * self.x_n_2
        ...
        self.x_n_2 = self.x_n_1
        self.x_n_1 = self.x_n
        self.y_n_2 = self.y_n_1
        self.y_n_1 = self.y_n

        # ------------------------------------------------------
        self.matriz_x_n = np.matrix([[self.ex1_n], [self.ex2_n]])

        ad = np.matrix([[1, 0.02], [0, 1]])
        bd = np.matrix([[4.2e-05 * ent1], [0.0042 * ent1]])
        cd = np.matrix([[1, 0]])

        self.matriz_x_n_1 = ad.dot(self.matriz_x_n) + bd

        self.ex1_n = self.matriz_x_n_1.item(0)
        self.ex2_n = self.matriz_x_n_1.item(1)
        self.actualizarSalidaPlanta()
```

La función del bloque ENTP es generar la señal de salida de la planta Ball & Beam ante una señal de entrada. La señal de salida es calculada con la ecuación de diferencias de la planta, y sus estados son calculados con la ecuación 4.21. Una vez determinados estos valores, se procede a llamar a la función actualizarSalidaPlanta.

Código 4.14. Función actualizarSalidaPlanta

```
    def actualizarSalidaPlanta(self):

        actual = self.tupla_listas[12].cabeza
        while actual is not None:

            if actual.variable_nombre_bloque == "." + self.variable_nombre_bloque:
                actual.y_n = self.x_n
                actual = actual.siguiente

            elif actual.variable_nombre_bloque == self.variable_nombre_bloque + "_X1":
```

```
            actual.y_n = self.ex1_n
            actual = actual.siguiente

        elif actual.variable_nombre_bloque == self.variable_nombre_bloque + "_X2":
            actual.y_n = self.ex2_n
            actual = actual.siguiente

        else:
            actual.y_n = self.y_n
            actual = actual.siguiente
...
```

La función del código 4.14 es modificar el parámetro de salida del bloque SALP. Este último bloque al no poseer terminales de entrada, retornará un valor que es modificado en cada recorrido por el bloque ENTP.

El bloque ENTP también tiene la capacidad de transmitir el valor de cada estado de la planta Ball & Beam. Para transmitir el valor de un estado, es necesario que el bloque SALP tenga el mismo nombre que el bloque ENTP seguido de "_X1" o "_X2", en donde este último parámetro indica si se desea recibir el estado 1 o el estado 2 respectivamente en el bloque SALP. En caso de que el nombre presente un punto ".", antes del nombre en SALP, ENTP transmitirá únicamente la señal de entrada del bloque sin modificar.

4.3 Protocolo de comunicación

Con el protocolo de comunicación, la herramienta de diseño es capaz de lograr la comunicación con la unidad de hardware. El objetivo de la comunicación es enviar a la unidad de hardware el archivo interpretable que contiene el netlist y la lista de bloques y parámetros indicados por el usuario desde la interfaz gráfica de la herramienta de diseño.

Ejemplo de un archivo interpretable a transmitir mediante el protocolo de comunicación:

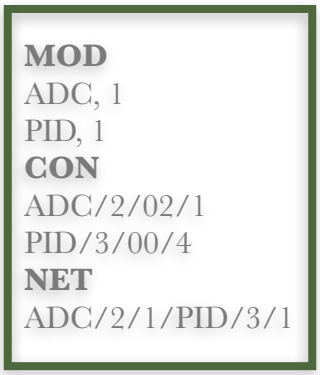

Figura 4.3.1. Ejemplo del archivo interpretable.

MOD	CON	NET
ADC,1	ADC/0/00/7/	SALP,0,1,ADC,0,1
PID,1	ADC/0/01/20/	ADC,0,1,PID,0,2
DAC,2	ADC/0/02/1/	DAC,0,1,ENTP,0,1
...	PID/0/00/0/	PID,0,1,DAC,0,1
	...	CTE,0,1,PID,0,1

El archivo interpretable consta de tres secciones: Módulos, Configuración y Netlist. La sección de Módulos, **MOD**, indica el número de módulos por tipo para la debida inicialización en la unidad de hardware. La sección de configuración **CON**, se encarga de indicar los parámetros correspondientes de cada uno de los módulos y, por último, la sección de Netlist **NET**, contiene las conexiones correspondientes entre cada uno de los bloques.

Figura 4.3.2. Estructura de la trama de datos de la sección CON

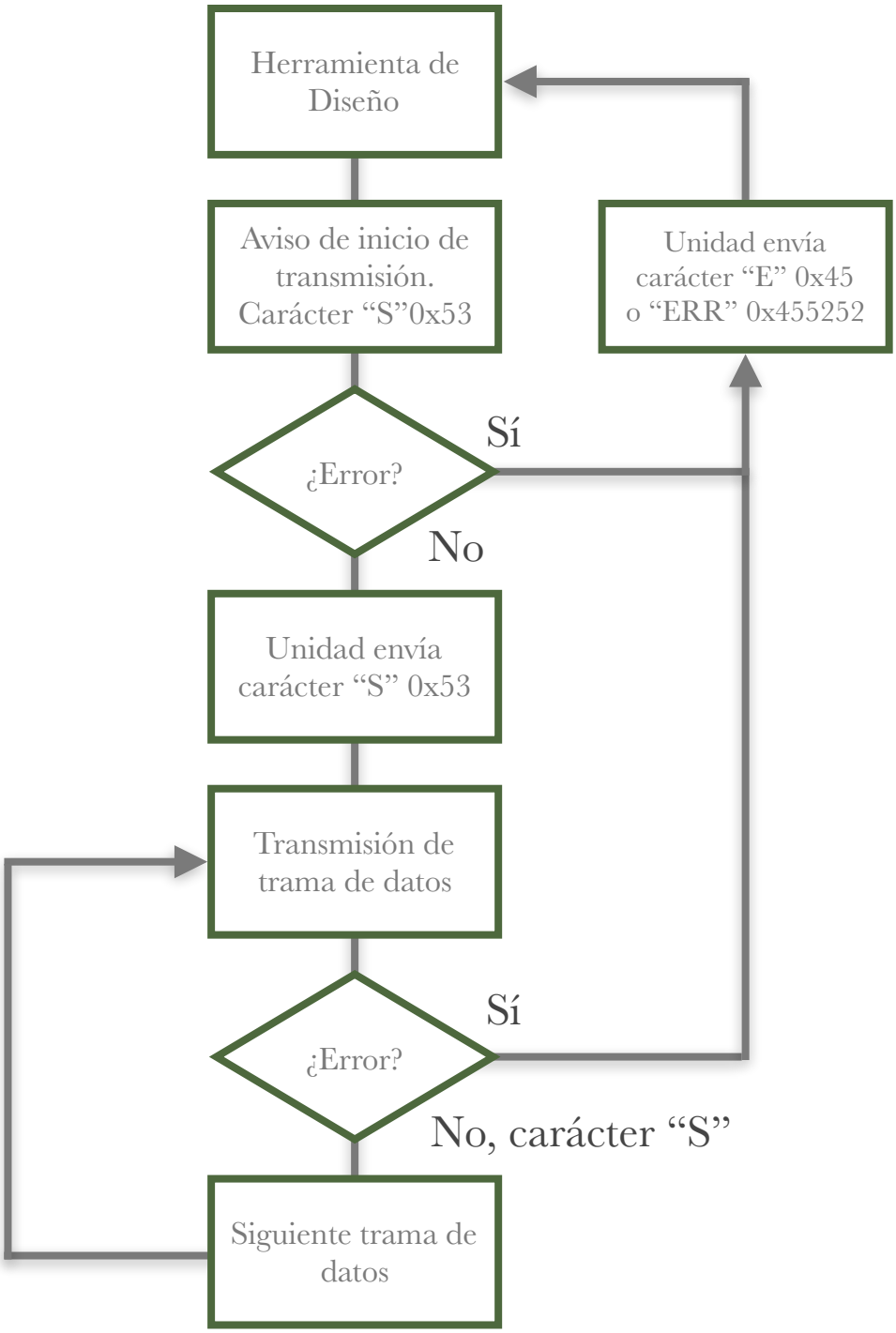

Figura 4.3.3. Protocolo de comunicación.

Capítulo 5: Análisis y resultados de la herramienta de diseño

La herramienta de diseñó se dividió en tres secciones: diseño de interfaz gráfica de usuario, capacidad de simulación y el protocolo de comunicación con la unidad de hardware.

5.1 Interfaz gráfica de usuario de la herramienta de diseño

La interfaz con el usuario incorpora los elementos y bloques necesarios para la elaboración de sistemas de control de distintas plantas del curso de Laboratorio de Control Automático del Instituto Tecnológico de Costa Rica.

Los bloques que incorpora la herramienta de diseño son ADC (Convertidor Analógico-Digital), PID, PWM, RE (Realimentación de Estados), CTE (Constante), FT (Función de Transferencia), SAT (Saturador), QDEC (Codificador de Cuadratura), DAC (Convertidor Digital-Analógico), FLT (Filtro), SCOPE (Osciloscopio), STEP (Escalón), TRP (Trapecio), DER (Derivativo), SUM (Sumador) y RST (Restador).

Los bloques disponibles se encuentran en la sección del sidebar de la interfaz gráfica y son colocados por el usuario dentro del workspace mediante la acción de “arrastrar y soltar”. La simplicidad en el uso de la herramienta de diseño permite la creación de sistemas de control en poco tiempo, disminuyendo así el tiempo de aprendizaje de la herramienta.

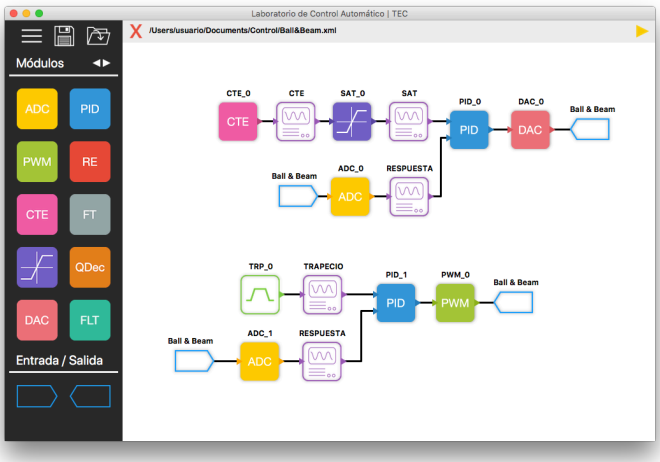

Figura 5.1.1. Sistema de control diseñado con la herramienta.

Además de incluir los módulos necesarios para la creación de sistemas de control, la herramienta incorpora iconos y botones comunes en la creación de interfaz de usuario. Entre estos botones e iconos se incluye los iconos de abrir y guardar, botones de "aceptar" y "cancelar", al igual que mensajes de error o de avisos al usuario. La herramienta también es aprueba de errores y en caso de ingresar parámetros inválidos se notifica al usuario sobre su error mediante mensajes de error.

5.2 Resultados de simulación

La herramienta de diseño logra la simulación de distintos sistemas de control para la planta Ball & Beam: PID, I_PD, PI_D, FT, RE y REI.

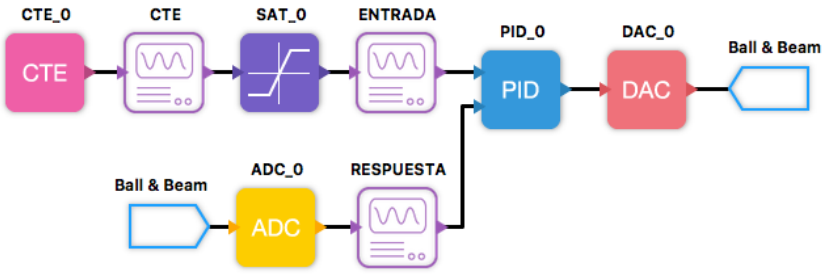

Figura 5.2.1. Ejemplo de sistema de control con bloque de control PID.

Para la configuración del control PID, se construye el sistema mostrado en la figura 5.2.1. Los bloques SCOPE (con nombres ENTRADA, RESPUESTA y CTE) se ubican en las conexiones en donde se desea la captura de datos. Para la configuración de este sistema, la constante de entrada se estableció en 0.30, mientras que los parámetros del bloque PID y SAT son los siguientes:

Bloque PID

- Tipo: PID
- Parámetro P: 10
- Parámetro I: 5
- Parámetro D: 10
- Tiempo de muestreo (ms): 10

Bloque SAT

- Límite superior: 0.25
- Límite inferior: - 0.25

Una vez establecidos los parámetros del control PID, la herramienta de diseño, junto con la ecuación del control PID, genera la señal de control hacia la planta y muestra de forma gráfica la respuesta ante la entrada.

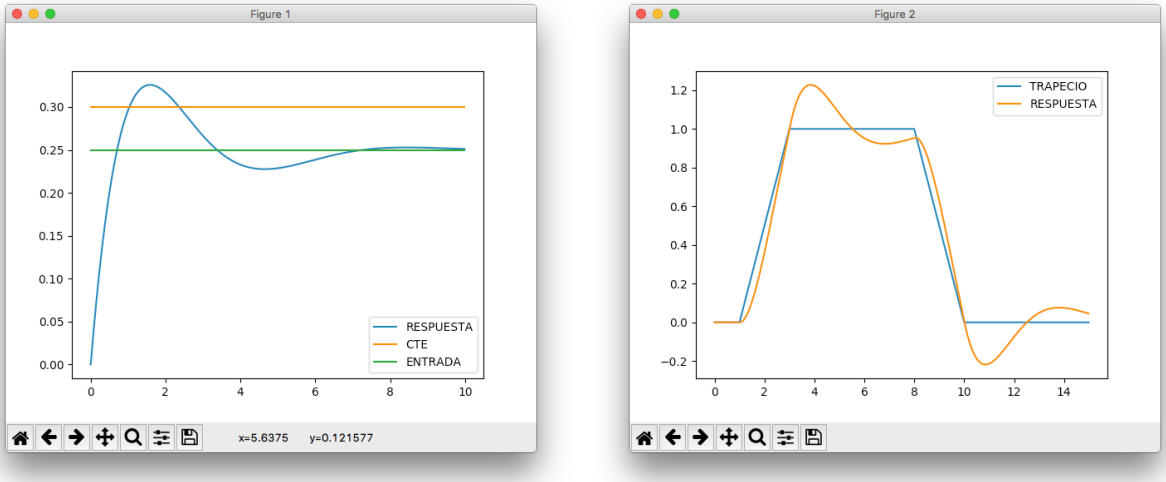

Figura 5.2.2. Gráfico de respuesta del bloque PID ante una entrada constante y trapecio.

La simulación del bloque FT viene dado por el siguiente sistema:

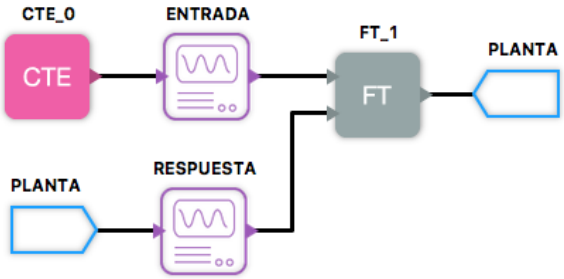

Figura 5.2.3. Sistema de control con bloque FT.

Los parámetros ingresados en el bloque FT son los siguientes:

Bloque FT

- Orden: 1
- Numerador: [165.6, -162.9]
- Denominador: [-0.9148]
- Con retroalimentación

La simulación del sistema de control con bloque de función de transferencia ante una entrada constante de valor 1 y ante una entrada trapecio de valor final 1 y tiempo de rampa igual a 2 se muestra en la figura 5.2.4., en donde el diseño implementado requería de un tiempo de estabilización menor a los 2 segundos.

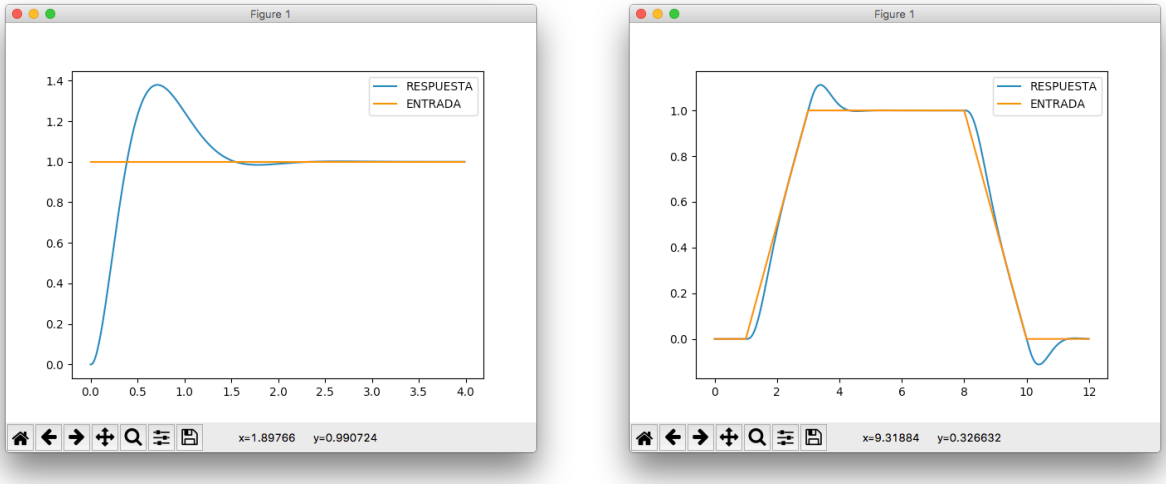

Figura 5.2.4. Gráfico de respuesta del bloque FT ante una entrada constante y trapecio.

La simulación del bloque RE se realizó con una entrada escalón con tiempo inicial igual a 3 segundos y valor final igual a 1. La constante K0 se estableció en 1 y la matriz K en [1 2]. Para el bloque REI, los parámetros configurables se establecieron en matriz K igual a [1 2], y matriz Ki en [1].

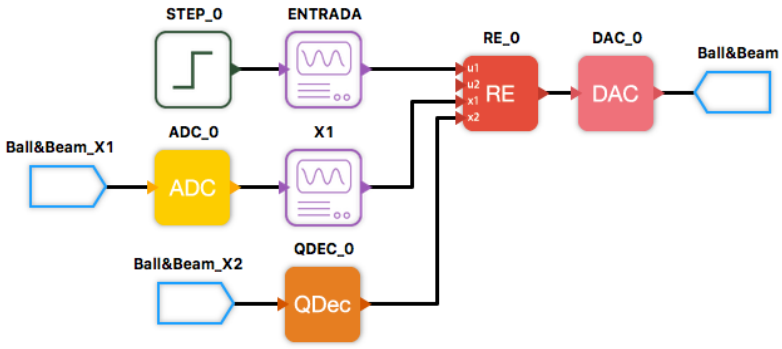

Figura 5.2.5. Sistema de control con bloque RE.

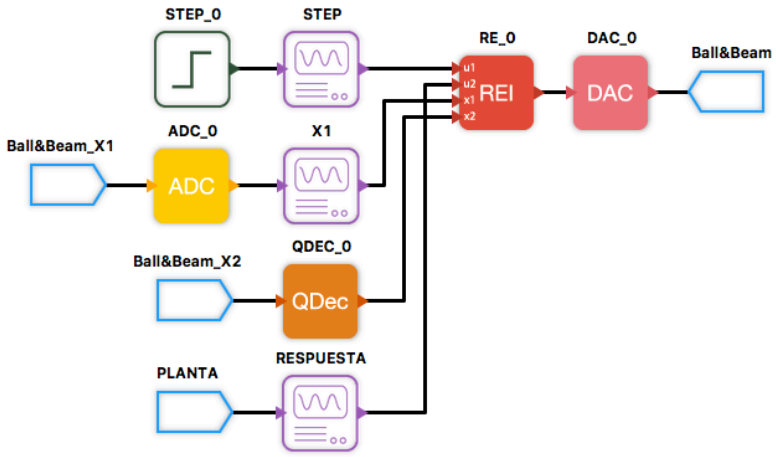

Figura 5.2.6. Sistema de control con bloque REI.

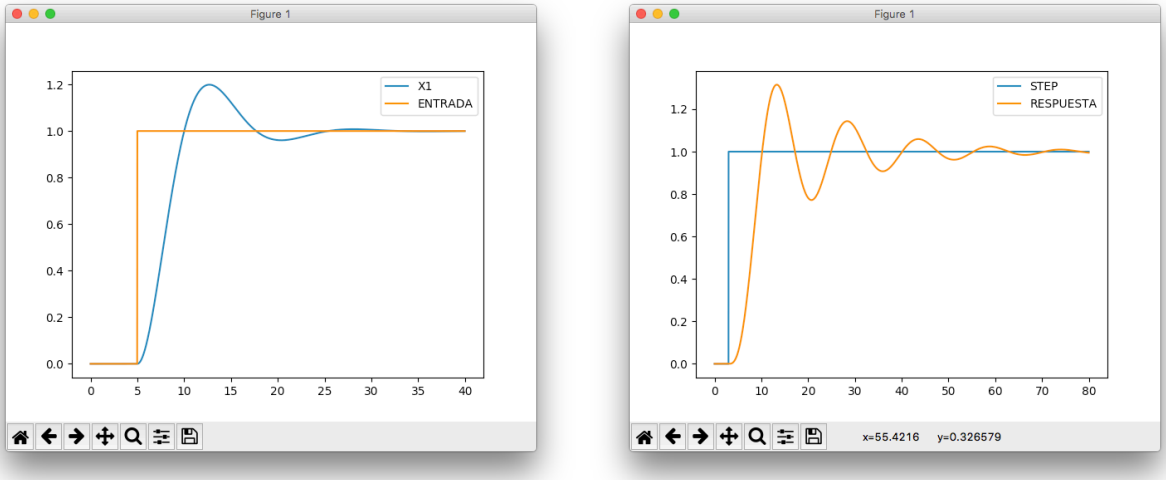

Figura 5.2.7. Respuesta de los bloques RE y REI ante una entrada escalón.

Al presionar el botón de simulación, la herramienta de diseño genera los gráficos de entrada/respuesta contra tiempo como lo muestra la figura 5.2.7. Las señales de cada bloque SCOPE se muestran con colores distintos, en donde el eje vertical indica la amplitud de la señal, mientras que el eje horizontal muestra el tiempo en segundos.

5.3 Protocolo de comunicación y el archivo interpretable

Se procedió a realizar el siguiente sistema de control dentro de la herramienta de diseño.

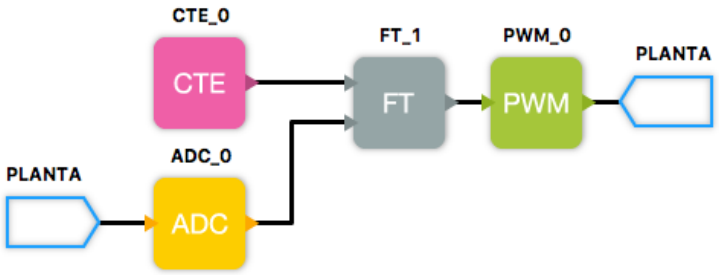

Figura 5.3.1. Sistema de control implementado con un bloque FT.

En donde los parámetros configurables de cada bloque están dados por:

Bloque CTE_0

- Valor: 1

Bloque ADC_0

- Rango: ±10V
- Tiempo de muestreo (ms): 10

Bloque FT_1

- Orden: 1
- Numerador: [165.587, -162.874]
- Denominador: [-0.914]
- Con retroalimentación

Bloque PWM_0

- Periodo (us): 500

Al generar el archivo interpretable, el protocolo de comunicación implementado es visualizado mediante el archivo de texto llamado netlist.txt, el cual contiene las tres secciones MOD, CON y NET. Cada línea del archivo representa la trama de datos a enviar por la comunicación serial. Para el caso del ejemplo de la figura 5.3.1, el archivo interpretable corresponde al siguiente:

```
MOD
ADC,1
PWM,1
CTE,1
FT,1
ENTP,1
SALP,1

CON
ADC/0/00/7/
ADC/0/01/10/
ADC/0/02/1/
PWM/0/00/1/
PWM/0/01/500/
PWM/0/02/1/
CTE/0/00/1/
FT/1/00/1/
FT/1/01/1/
FT/1/02/[165.587,-162.874]/
FT/1/03/[-0.914]/

NET
ADC,0,1,FT,1,2
CTE,0,1,FT,1,1
SALP,0,1,ADC,0,1
FT,1,1,PWM,0,1
PWM,0,1,ENTP,0,1
```

La herramienta logra la generación del archivo y sus tres secciones. La sección MOD indica la existencia dentro del workspace de 1 bloque ADC, 1 bloque PWM, 1 bloque CTE, 1 bloque FT, 1 bloque ENTP y 1 bloque SALP, los cuales corresponden al sistema de la figura 5.3.1. Seguido de la sección MOD, CON muestra las tramas de los parámetros de los bloques a enviar por la comunicación serial. Siguiendo la tabla con los

códigos hexadecimal descritos en la tabla 4.1.9, la sección CON contiene de manera esperada los parámetros ingresados mencionados anteriormente.

```
CON
ADC/0/00/7/                     ADC_0, parámetro 00, rango = 7
ADC/0/01/10/                    ADC_0, parámetro 01, tiempo muestreo = 10
ADC/0/02/1/                     ADC_0, parámetro 02, ganancia = 1
PWM/0/00/1/                     PWM_0, parámetro 00, tipo = 1
PWM/0/01/500/                   PWM_0, parámetro 01, periodo = 500
PWM/0/02/1/                     PWM_0, parámetro 02, amplitud = 1
CTE/0/00/1/                     CTE_0, parámetro 00, valor = 1
FT/1/00/1/                      FT_1,  parámetro 00, tipo/retroalimentación = 1
FT/1/01/1/                      FT_1,  parámetro 01, orden = 1
FT/1/02/[165.587,-162.874]/     FT_1,  parámetro 02, numerador = [165.587,-162.874]
FT/1/03/[-0.914]/               FT_1,  parámetro 03, denominador = [-0.914]
```

Por último, la sección NET del archivo interpretable contiene el netlist de las conexiones de los módulos dentro del workspace. Cada línea dentro de la sección NET corresponde a cada trama de datos a enviar por la comunicación serial.

```
NET
ADC,0,1,FT,1,2          ADC_0 por su salida 1 se conecta a FT_1 por su entrada 2
CTE,0,1,FT,1,1          CTE_0 por su salida 1 se conecta a FT_1 por su entrada 1
SALP,0,1,ADC,0,1        SALP_0 por su salida 1 se conecta a ADC_0 por su entrada 1
FT,1,1,PWM,0,1          FT_1 por su salida 1 se conecta a PWM_0 por su entrada 1
PWM,0,1,ENTP,0,1        PWM_0 por su salida 1 se conecta a ENTP_0 por su entrada 1
```

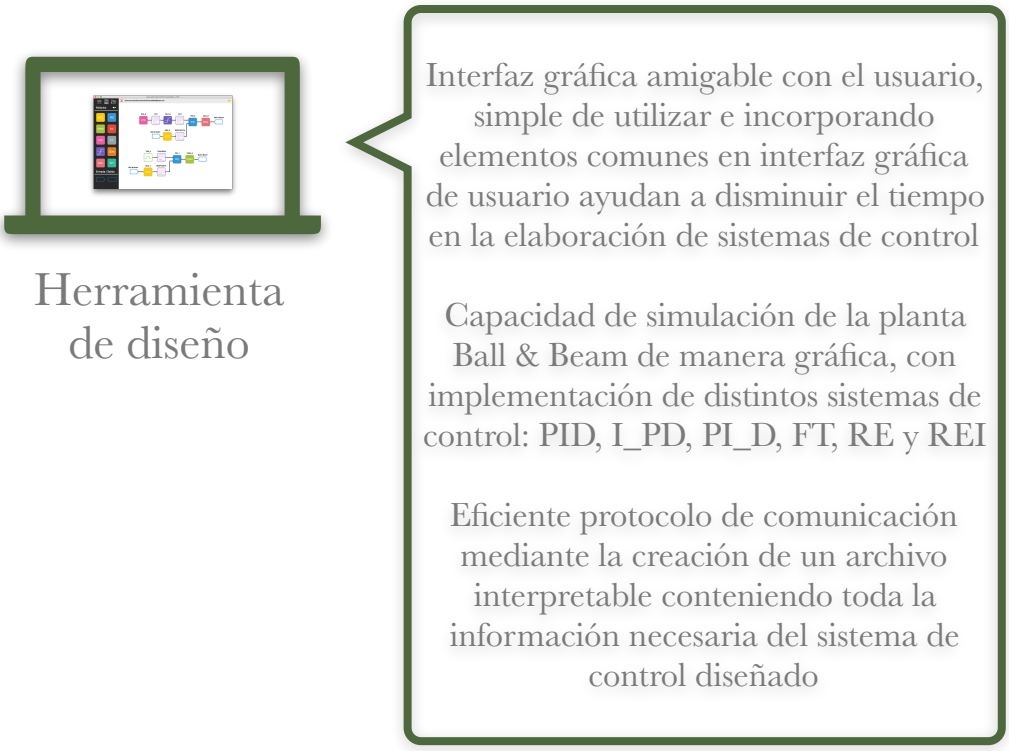

Figura 5.3.2. Implementación de la herramienta de diseño de sistemas de control.

Capítulo 6: Conclusiones

• La incorporación de una interfaz gráfica en la herramienta de diseño elimina la necesidad de implementar sistemas de control en lenguaje de máquina por parte del usuario.

• El uso de una biblioteca de interfaz gráfica como Tkinter simplificó el desarrollo de la herramienta de diseño.

• La implementación del lenguaje de marcado XML en las opciones de guardar y abrir archivos permite la visualización y entendimiento por parte del usuario de las configuraciones y parámetros de los módulos utilizados.

• La generación de un archivo interpretable crea una independencia entre la herramienta de diseño y la unidad de hardware en caso de un futuro cambio o modificación de la unidad.

• El uso de las bibliotecas de funciones matemáticas de alto nivel, NumPy y matplotlib, facilitó la implementación de la capacidad de simulación de la herramienta de diseño.

Bibliografía

[1] Caravaca, Oscar. Diseño de un Entorno de Desarrollo Integrado para una Unidad Controladora de Procesos. 2010.

[2] Castro, Daniel. Unidad Controladora de Procesos para el diseño, análisis, simulación e implementación de sistemas de control automático a través de redes TCP/IP. 2008.

[3] Chaudhary, Bhaskar. Tkinter GUI Application Development Blueprints. Packt Publishing, 2015.

[4] Control Tutorials for MATLAB & SIMULINK. University of Michigan, Carnegie Mellon University, University of Detroit Mercy. [Octubre 2018]
http://ctms.engin.umich.edu/CTMS/index.php?aux=Home

[5] Galitz, Wilbert O. The Essential Guide to User Interface Design. Segunda edición. John Wiley & Sons, Inc. 2002.

[6] Halterman, Richard L. Learning to program with Python. 2011.

[7] MathWorks. The MathWorks, Inc. Documentation. [Octubre 2018]
https://www.mathworks.com/help/

[8] Ogata, Katsuhiko. Modern Control Engineering. Quinta edición. Prentice Hall. 2010.

[9] Rodriguez, Jesús. Introducción a la programación. Teoría y Práctica. Editorial Club Universitario. [Noviembre 2018] https://www.editorial-club-universitario.es/pdf/405.pdf

[10] Slonneger, Kenneth. Kurtz, Barry L. Formal Syntax and Semantics of Programming Languages. Addison-Wesley Publishing Company. 1995

[11] Stallings, William. Computer Organization and Architecture. Designing for Performance. Novena Edición. Prentice Hall. 2013.

[12] Young, Hugh D. Freedman, Roger A. University Physics With Modern Physics. 13a Edición. Pearson Education, Inc. 2012

[13] Zelle, John M. Python Programming: An Introduction to Computer Science. Versión 1.0rc2. 2002.