UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
SEDE VIÑA DEL MAR – JOSÉ MIGUEL CARRERA

DESARROLLO DE MAQUETA DIDÁCTICA PARA DOCENCIA EN CONTROL AUTOMÁTICO

Trabajo de Titulación para optar al Título Profesional de Ingeniero de Ejecución en CONTROL E INSTRUMENTACIÓN INDUSTRIAL

Alumno:
Jorge Andrés Núñez Núñez

Profesor Guía:
Mag. Guelis Montenegro Zamora

2023

RESUMEN

KEYWORDS: MAQUETA, DIDÁCTICO, HARDWARE, FIRMWARE, DESARROLLO, DISEÑO, MODELO, SOFTWARE, CONTROL, PROCESOS, I+D.

Existe un problema al momento de relacionar materia teórica con la práctica en unidades de competencia asociadas a control y automatización industrial. Basados en experiencia personal y comentarios de compañeros / colegas, gran porcentaje de los alumnos se ven afectados por no entender correctamente cómo aplicar la teoría dentro de laboratorios prácticos o en el área laboral al momento de egresar de la carrera. Este trabajo tiene el objetivo de generar material didáctico que permita la integración de la práctica al momento que se enseña la teoría de una forma rápida y simple. Concretamente, intenta generar maquetas didácticas transportables que contengan un firmware de ejemplo y que pueda ser utilizado por docentes y/o alumnos.

Para cumplir el objetivo, se realizó el diseño, fabricación y firmware de dos maquetas de control de procesos llamadas “Demostración de control de un Servomotor” y “Barra y Bola”. Para la maqueta “Barra y Bola”, también se realizó el análisis e identificación del sistema para generar a modo de ejemplo, un controlador PID que sirva para el control de ésta. Además, se integra la consideración para que las maquetas cumplan requisitos de transportabilidad y economía, además, del diseño del hardware mediante placas de desarrollo y la generación de firmware de prueba para dar un punto inicial al usuario de las maquetas.

Ayudar a la educación para el control automático, permite la generación de grandes profesionales capaces de responder correctamente en cada proyecto de su vida.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

SIGLAS Y SIMBOLOGÍAS

A. SIGLAS:

AC	:	Alternate Current
AI	:	Adobe Ilustrator
ADC	:	Analog to Digital Converter
CAD	:	Computer aided Desing (Diseño asistido por computadora)
CNC	:	Control Numérico Computarizado
DXF	:	Drawing Exchange Format (Formato de archivo CAD)
DC	:	Direct Current
DAC	:	Digital to Analog Converter
GND	:	Ground (Referencia digital en electrónica)
GPIO	:	General Purpose Input/Output
IOT	:	Internet Of Things
I2C	:	Inter Integrated Circuit
IDE	:	Integrated Development Enviroment
MRUA	:	Movimiento Rectilíneo Uniformemente Acelerado
OLED	:	Organic Light Emitting Diode
PWM	:	Pulse Width Modulation
PLA	:	Poliácido Láctico
PID	:	Proporcional Integral Derivativo
PCB	:	Printed Circuit Board
PC	:	Personal Computer
RAM	:	Random Access Memory
SCL	:	Serial Clock
SDA	:	Serial Data
STL	:	STereoLithography (Formato de archivo CAD)
ToF	:	Tiempo de Vuelo (Time Of Flight)
TTL	:	Transistor-Transistor Logic
USB	:	Universal Serial Bus
USB	:	Universal Serial Bus
3D	:	Tres dimensiones

B. SIMBOLOGÍA:

A	:	Amperios
cm	:	Centímetro
KΩ	:	Kilo ohm (medida de resistencia)
KB	:	Kilobyte
mAh	:	Miliamper hora
ms	:	Milisegundos
mm	:	Milímetro
Mhz	:	Megahertz
V	:	Voltaje
VDC	:	Voltaje de corriente continua
VAC	:	Voltaje de corriente alterna
°C	:	Grados Celsius
“	:	Pulgadas

INTRODUCCIÓN

El control y la automatización industrial forman un campo interdisciplinario esencial que combina ingeniería, tecnología y sistemas para optimizar y gestionar de manera eficiente los procesos industriales. Este campo es crucial en la industria, ya que busca maximizar la productividad, la calidad y la seguridad en una variedad de industrias, al reducir la intervención humana directa y mejorar la toma de decisiones basada en datos. También, la seguridad para los trabajadores aumenta debido a que con maquinaria automatizada se pueden realizar tareas peligrosas o riesgosas, protegiendo así a los trabajadores de situaciones potencialmente dañinas.

En las casas Universitarias, para poder enseñar las técnicas de control y automatización, se introducen maquetas que normalmente consisten en equipos robustos de gran calidad, lo cual eleva el peso, el tamaño y el precio de éstas. Al momento de enseñar material teórico en las salas, cuesta poder relacionarlos posteriormente en los laboratorios prácticos, debido a los días transcurridos desde un evento a otro y que estas maquetas no pueden ser transportadas. También ocurre que, si se desea enseñar teoría en el momento de la práctica, el avance de aprendizaje práctico es casi nulo debido a que las explicaciones requieren harto tiempo.

Este trabajo parte analizando la problemática, lo efectos de ésta en los alumnos de las unidades de competencia relacionadas a control industrial y explora las posibles soluciones. Posteriormente se realiza el diseño de la solución propuesta desde cero, seleccionando componentes que tendrá cada maqueta, diseñando y fabricando piezas 3D, implementando la electrónica necesaria, la programación de firmware base y analizando una maqueta para la identificación y control del sistema.

CAPÍTULO 1: PROBLEMAS Y OBJETIVOS

1 PROBLEMAS Y OBJETIVOS

En el presente capítulo se describe la problemática detectada en el desarrollo de educación de las unidades de aprendizaje con relación a “Control automático”, el cual puede ser solucionado con una herramienta para docentes y alumnos con la finalidad de entregar o llevar a la práctica de mejor forma los conceptos teóricos.

1.1 ANTECEDENTES GENERALES

En las carreras de Control e instrumentación industrial impartidas en las distintas casas de estudio es obligatorio la enseñanza de las técnicas de **Control automático**, ya que gracias a éstas y los avances tecnológicos se puede automatizar procesos de producción, método empleado en las industrias modernas para optimizar tiempos, eliminar margen de error, evitar accidentes del personal, entre otros.

Basándose en la experiencia educativa obtenida en la Universidad Técnica Federico Santa María sede José Miguel Carrera, Viña del Mar, se detectó la dificultad que implica el completo entendimiento de esta unidad de aprendizaje y la dificultad para el docente al momento de entregar los conocimientos, ya que, la teoría consiste en cálculos matemáticos complejos, lo que resulta ser abstracto para alumnos que están en etapa de aprendizaje básica, donde la mayoría de las veces no basta con observar maquetas, virtuales y físicas, ubicadas en laboratorios específicos que requieren de horarios especiales y disponibilidad para su uso específico.

1.2 PROBLEMÁTICA

Aún con laboratorios equipados para la enseñanza práctica, estos son dispositivos costosos y de gran tamaño, por lo que necesariamente deben estar en un lugar fijo, el cual suele ser un laboratorio designado para estos. Por lo que para demostrar lo teórico hay que agendar clases de laboratorio o utilizar distintos tipos de herramientas de aprendizaje, como lo son los materiales audiovisuales y simulaciones.

1.1.1 Definición del problema

Como consecuencia de lo planteado anteriormente, se detectó, como problema principal al momento de enseñar las técnicas de control automático, la existencia de una desconexión del aprendizaje teórico con lo práctico, puesto que la teoría requiere demostración práctica constante, para relacionar y comprender completamente la materia.

Esta desconexión produce que el proceso de aprendizaje sea lento al momento de ejecutar laboratorios, concluyendo a bajas calificaciones para los alumnos y a preocupación o malos comentarios para el docente.

Las causas detectadas del problema consisten básicamente en la incapacidad de horario para el uso de laboratorios, puesto que no se puede tener un curso completo en estos y existen varios cursos o paralelos que demandan estos laboratorios. Las maquetas son grandes, pesadas y costosas, por lo que trasladar una maqueta hacia otro lugar implica harto trabajo y puede resultar peligroso.

1.1.2 Importancia de resolverlo

La importancia de resolver el problema impacta directamente a la vida profesional de los alumnos, a cómo estos se desenvuelvan en la distintas empresas y la capacidad que estos tengan al momento de resolver problemas.

1.1.3 Involucrados

Los involucrados o afectados directamente por este problema son:

- **Alumnos:** No logran el completo entendimiento de la teoría de las técnicas de control automático.
- **Educadores:** Por más que los educadores sean de calidad, no se logra entregar completamente las enseñanzas a los alumnos.
- **Centros de educación:** Afecta a la reputación de los centros educativos, debido al bajo rendimiento de los alumnos en las distintas empresas o áreas donde estos se desenvuelvan.
- **Empresas finales:** El rendimiento de los trabajadores de las empresas afecta al funcionamiento de éstas, debido a errores o fallas cometidos por la falta de conocimiento teórico.

1.3 SOLUCIÓN AL PROBLEMA

En base a la problemática mostrada, con sus causas e importancia de resolverlo, se busca realizar un proyecto o sistema que permita poder enseñar y reforzar lo teórico de las técnicas de control automático, con lo práctico, de forma simple y cómoda.

1.3.1 Requerimientos

Los requerimientos más importantes para la solución al problema consisten básicamente, en un sistema que permita aplicar la teoría de las técnicas de control automático en la práctica, que pueda ser transportable y de bajo costo.

1.3.2 Alternativas de solución

Para dar solución a la problemática se proponen cuatro alternativas, las cuales deberán ser sometidas a evaluación según criterios pertinentes para escoger cuales son factibles de aplicar, ya sea técnica como económicamente y así finalmente escoger la más óptima.

1.3.2.1 Alternativa N°1: “Hacer nada”

Una alternativa posible consiste en no desarrollar el proyecto y que la forma de educar las técnicas de control automático siga como está actualmente, asumiendo que la materia no se recibe de la mejor forma por los alumnos y que producirá bajas calificaciones. Por el lado del educador, se puede ver afectado ante las críticas del nivel de reprobados y tendrá que disminuir la exigencia del ramo, disminuyendo la calidad de los futuros profesionales.

1.3.2.2 Alternativa N°2: “Realidad Virtual”

La segunda alternativa consiste en la adquisición de tecnología para realidad virtual en donde se pueda realizar la inmersión práctica de la teoría en la industria.

Si bien, existe tecnología para esto, resulta ser una opción interesante pero costosa y también requeriría de un lugar específico y mantener especial cuidado con los equipos.

También habría que buscar o desarrollar aplicaciones para interacción virtual con la industria, considerando varios escenarios posibles y permitiendo implementar sistemas personalizados.

1.3.2.3 Alternativa N°3: “Implementación de maqueta didáctica”

La tercera alternativa consiste en desarrollar y fabricar una maqueta de control automático que sea pequeña, liviana y transportable. Además, debe contar con la capacidad de ser utilizada de forma rápida, mediante un microcontrolador y un firmware de ejemplo que permita la interacción inicial con ésta.

La meta de esta alternativa es poder lograr una maqueta económica en comparación a las maquetas industriales robustas presentes en la Universidad. Esta reducción de precio permite la fácil adquisición para docentes e incluso para varios alumnos, así poder enseñar, estudiar y practicar la teoría de manera simple.

También permite replicar la maqueta, ya que todas las partes y piezas diseñadas, firmware y hardware serán accesibles por algún medio digital, como Drive, Github, entre otros. A esto se le denomina de código o fuente abierto, ya que se pone a disposición toda la información del proyecto a la comunidad para que reutilicen o realicen los cambios que estimen convenientes.

1.3.2.4 Alternativa N°4: “Comprar plataforma grande”

Implica la compra de las maquetas actuales, induciendo o manteniendo la problemática intacta, debido a la nula transportabilidad del sistema hacia salas donde se esté impartiendo la teoría.

1.3.3 Evaluación de las alternativas de solución

Basándose en las alternativas de solución planteadas anteriormente, se ejecutará un análisis en base a criterios simples para determinar cuál de las alternativas es factible de aplicar. Como se observa en la Tabla 1-1, los criterios están en una escala del 1 al 5 donde 1 corresponde a “Muy deficiente” y 5 corresponde a la calificación “Óptima”. La suma de estos criterios resultará en un puntaje total para cada alternativa, siendo seleccionada para ejecutarse la que contenga mayor puntaje.

Tabla 1-1 Tabla de puntuación de alternativas de solución

Muy deficiente	**Deficiente**	**Aceptable**	**Bueno**	**Óptimo**
1	2	3	4	5

Fuente: Elaboración propia basada en las necesidades del proyecto.

Los criterios a utilizar serán cinco, siendo estos los siguientes: Innovación, beneficio, costo económico, factibilidad y transportabilidad. Innovación ayuda a determinar el nivel de aporte con las ideas o tecnologías empleadas que aporta. El beneficio permite determinar la ganancia que producirá en los involucrados en el problema. El costo económico es determinante debido a que se busca una solución económica. La factibilidad es la capacidad técnica que se maneja para realizar la alternativa evaluada. La transportabilidad corresponde a la opción de poder mover las maquetas hacia los lugares donde sean necesarias, para poder enseñar o poder aprender de forma amigable. La selección de la solución se muestra resaltada en azul dentro de Tabla 1-2.

Tabla 1-2 Grilla de evaluación de criterios.

Característica/Alternativa	Alternativa 1	Alternativa 2	Alternativa 3	Alternativa 4
Innovación	1	5	5	1
Beneficio	1	5	5	5
Costo económico	5	1	3	1
Factibilidad	5	1	5	3
Transportabilidad	1	3	5	1
Puntuación total	**13**	**15**	**23**	**11**

Fuente: Elaboración propia basada en comparación de alternativas.

1.3.4 Alternativa seleccionada

El resultado de la evaluación de criterios indica como mejor propuesta la **alternativa 3**, debido a su alta puntuación de **Innovación**, permitiendo maquetas educativas transportables de fácil uso. El alto **Beneficio** para todas las partes, tanto docentes, alumnos, casas Universitarias y empresas obtendrán mejores resultados al momento de enseñar y de aprender. La **Factibilidad** de realización también es alta debido a que se cuenta con el Software, maquinaria y conocimientos necesarios para diseñar y fabricar un dispositivo completamente funcional. Además de contar con los conocimientos electrónicos para la generación de circuito electrónico y de firmware que controlen las maquetas. Gracias al hecho de que las maquetas sean diseñadas de forma propia, permite que la **Transportabilidad** también obtenga un puntaje alto, ya que estas maquetas estarán diseñadas en base a que puedan ser movidas de un lugar a otro sin mayor complicación. Por último, el **Costo económico**, no obtiene el mayor puntaje debido a igual es un gasto mayor si se desea adquirir más de un equipo de maquetas portátiles.

Para medir el avance de esta solución, se ha generado la Tabla 1-3 con requerimientos específicos. La tabla, cuenta con la descripción y el criterio de aceptación para cada requerimiento.

Tabla 1-3 Requerimientos específicos de la alternativa seleccionada

N	Requerimiento	Descripción	Criterio de aceptación
1	Selección de maquetas	Seleccionar las maquetas que mejor se adapten al diseño y construcción, además que permitan explicar el funcionamiento de controlador PID.	Nombres y cantidades de maquetas a confeccionar.
2	Establecer tecnología básica	Establecer si se usará una maleta para el transporte, si usará una batería de litio, panel solar, red eléctrica, entre otros.	Obtención de la tecnología básica para el funcionamiento de las maquetas.
3	Diseño y fabricación de piezas 3D	Diseñar todas las partes y piezas para el funcionamiento de las maquetas seleccionadas.	Diseño demostrable de partes y piezas en software de CAD 3D Autodesk Inventor.
4	Confección y pruebas de las maquetas	Se debe construir y probar que las maquetas funcionen de forma correcta.	Demostrar el funcionamiento de las maquetas.
5	Programación de firmware básico	Se debe generar un firmware de prueba básico funcional para las maquetas.	Demostrar la existencia de un o más firmware para las maquetas y que éstas trabajan sin problemas con éste.
6	Obtener la función de transferencia de una de las maquetas	Se debe lograr la obtención de una función de transferencia para el control de una de las maquetas generadas.	Mostrar la función de transferencia.
7	Documentación	Generar diagrama de flujo y estudio de costos para las maquetas a generar. También contar con todo el firmware y piezas en alguna plataforma online como Google Drive o Github para la replicación del equipo.	Demostrar al menos un diagrama de flujo simple para el funcionamiento general de los firmwares y obtener valores de todos los materiales, partes y piezas, así como el valor o costo de la mano de obra. También hacer entrega de un link para la descarga de toda la información.

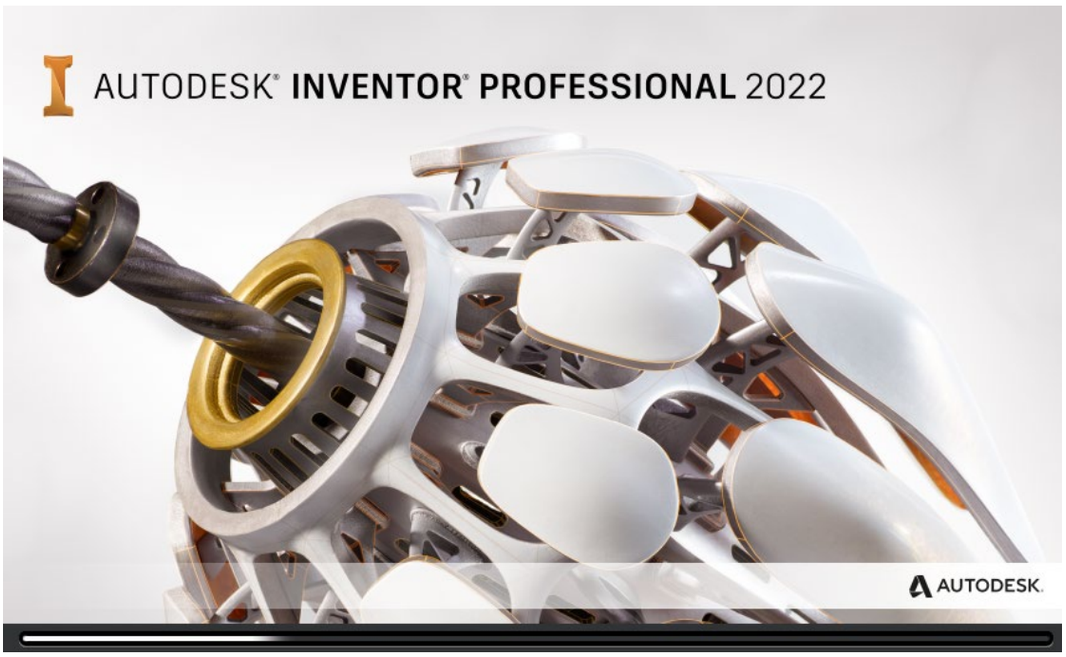
Fuente: Elaboración propia.

1.4 SOFTWARES A USAR

Se presentan los software a utilizar para el completo desarrollo de la solución seleccionada. Estos permitirán el diseño 3D, la conversión para cortes láser, impresión 3D, programación y análisis del sistema.

1.4.1 Autocad Inventor

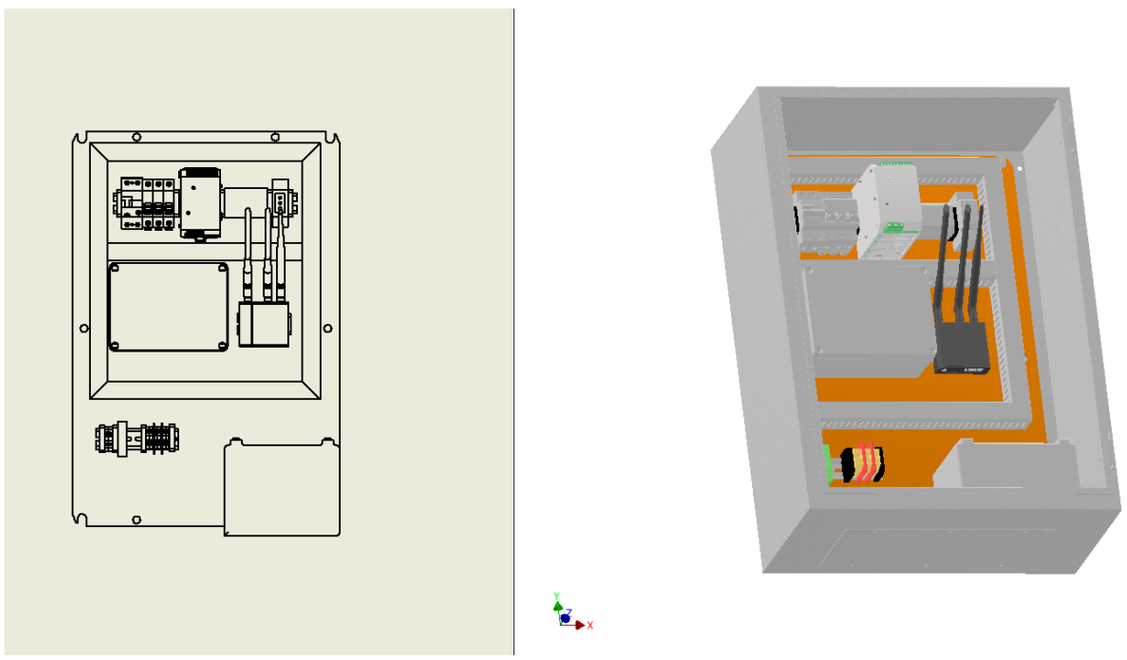
Es un software CAD 3D permite el diseño mecánico, documentar y simular productos. El logo se muestra en la Figura 1-1.



Fuente: Captura de pantalla, elaboración propia.

Figura 1-1 Pantalla de carga Autodesk Inventor.

Contiene herramientas como: plegado de metal (chapas de metal), diseño de estructuras, tubos y tuberías, cables y arneses, presentaciones, renderizar, simulaciones, diseño de mecanizado, entre otros. Estas herramientas son útiles para el Control Industrial, ya que permiten creaciones y previsualizaciones de tableros eléctricos o de control, además de soportes, carcasas y cableados, como se muestra en la Figura 1-2.



Fuente: Elaboración propia.

Figura 1-2 Ejemplo de diseño 3D con Autodesk Inventor.

1.4.2 RD Works V8

Software para el grabado o corte con máquinas laser. Soporta dibujar puntos, líneas, polilínea, elipses, entre otros. También permite la apertura de archivos vectorizados como Dxf, ai, plt, dst, dsb, entre otros. El logo del software se muestra en la Figura 1-3.

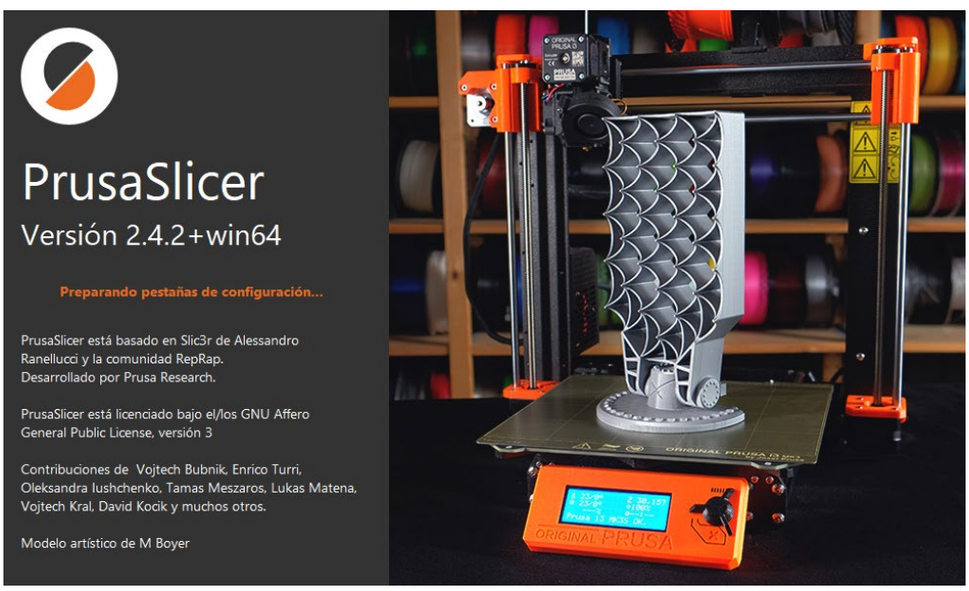
Fuente: https://www.fm-laser.ch/en/shop/product/rdworks-rdworks-v8-01-48-86?category=26#attr=

Figura 1-3 Logo de RDWorks V8.

1.4.3 Prusa Slicer

Completo software de código abierto que permite transformar documentos con extensión stl a documentos GCODE para la impresión de objetos 3D en impresoras 3D.

También agrega dentro del GCODE todas las ordenes que la impresora 3D debe ejecutar, tales como temperaturas, velocidad de movimiento, posición entre otros. La pantalla de carga de la aplicación se muestra en la Figura 1-4.



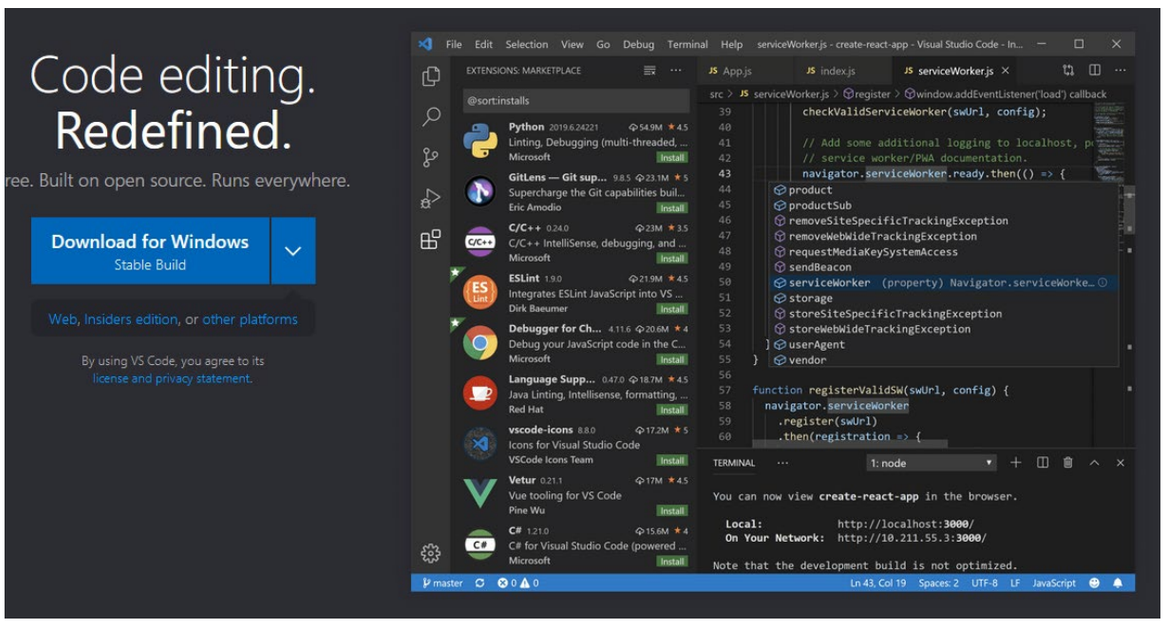
Fuente: Captura de pantalla, elaboración propia.

Figura 1-4 Pantalla de inicio Prusa Slicer

Un software Slicer para impresión 3D permite “rebanar” por capas el modelo 3D, calculando las intersecciones del objeto 3D de los planos a la altura de capa de impresión. Cada una de estas secciones permite calcular los movimientos que debe realizar el cabezal para rellenar cada área del objeto e imprimir el objeto de forma correcta.

1.4.4 Visual Code Studio

Software editor de código multiplataforma y multilenguaje. Permite control integrado de Git o repositorios, soporte para depuración, finalizado inteligente de código, fragmentos y refactorización de código. La página de inicio del software se muestra en la Figura 1-5.

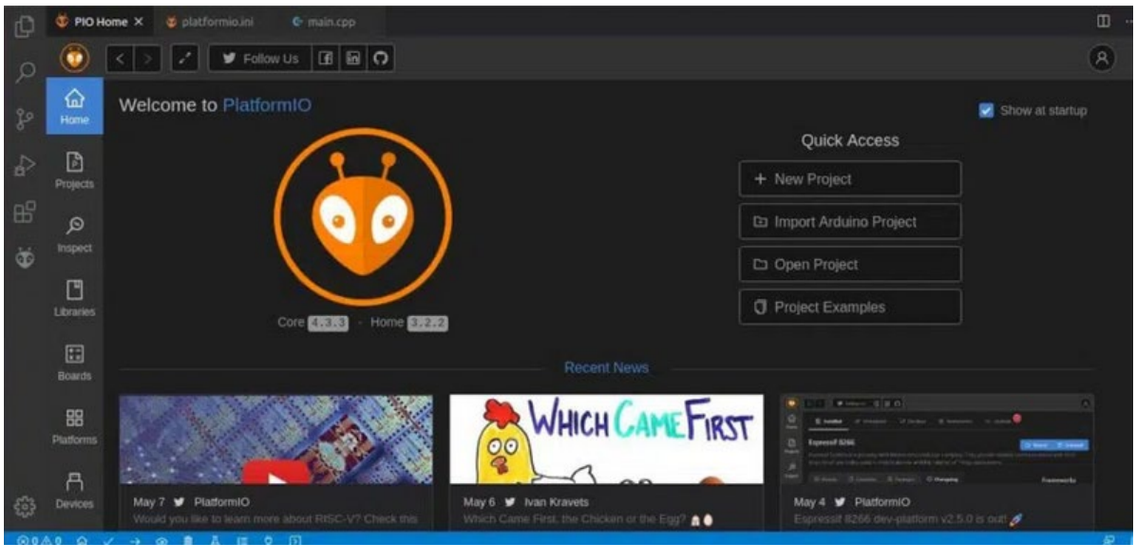


Fuente: https://code.visualstudio.com/

Figura 1-5 Inicio página web de Visual Code Studio.

1.4.5 Platformio

Es un IDE abierta para C/C++ (Integrated Development Enviroment) orientado al hardware. Este IDE es compatible con diversos editores de texto, como Visual Code Studio, Sublimetext, entre otros. En la Figura 1-6 se muestra la ventana de inicio del IDE.

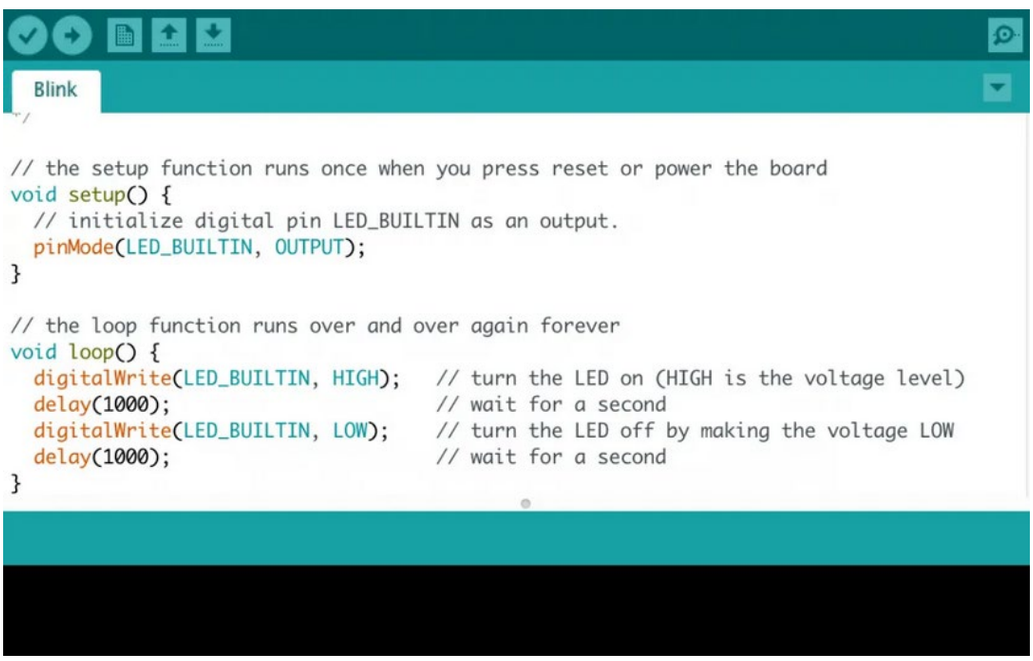


Fuente: Captura de pantalla, elaboración propia.

Figura 1-6 Pantalla de inicio de Platformio en Visual Code Studio.

1.4.6 Arduino

IDE C/C++ abierta para hardware compatible con Arduino. Se hizo popular gracias a estas placas de desarrollo. En la Figura 1-7 se muestra el entorno de programación de Arduino.

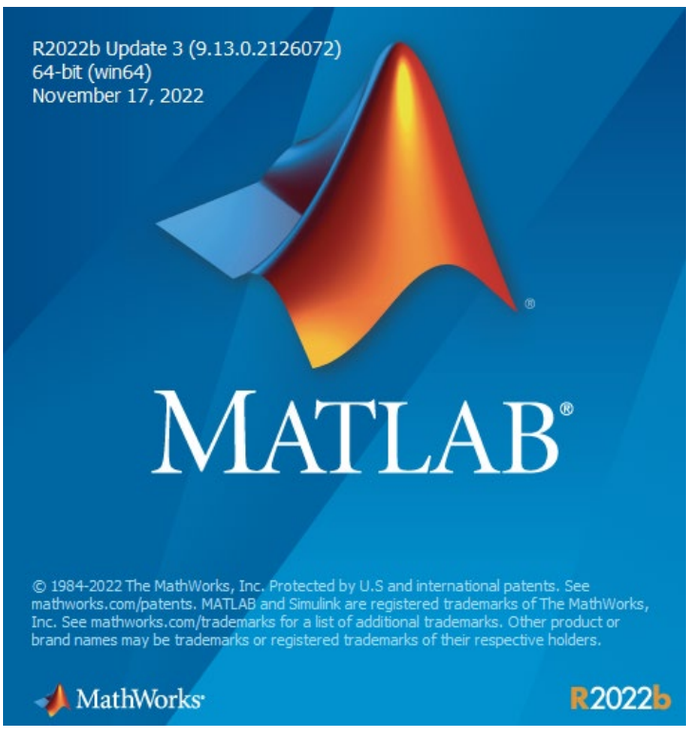


Fuente: Captura de pantalla, elaboración propia.

Figura 1-7 Muestra de código C++ en IDE Arduino.

1.4.7 Matlab

Es un software de cómputo numérico con un lenguaje de programación propio llamado lenguaje M. Permite la utilización de varias herramientas además de incrementar su potencial mediante pluggins creados por Matlab o terceros. Esta aplicación es utilizada durante la carrera para el análisis, identificación y obtención de sistemas de control. En la Figura 1-8 se muestra el logo de Matlab.



Fuente: Captura de pantalla, elaboración propia.

Figura 1-8 Pantalla de inicio de Matlab.

1.5 MAQUINARIA

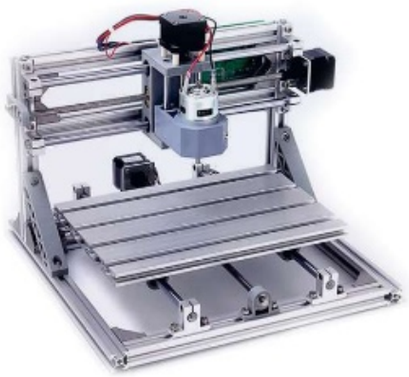
A continuación, se presentan las posibles maquinarias que se utilizarán para el proceso de fabricación de la maqueta didáctica. Estos equipos son requeridos para cortar acrílico, MDF y/o imprimir partes y piezas en 3D.

1.5.1 Fresadora CNC 3018

CNC o Control Numérico Computarizado, es una máquina automatizada para la realización de cortes en metal, madera, plásticos u otros materiales. Permite generar partes y piezas mediante el método de desbaste.

El control de estas máquinas se realiza por un lenguaje llamado GCODE, el cual le da todas las indicaciones necesarias de posiciones, velocidades, incluso hasta el cambio de brocas a realizar (en caso de ser una CNC automática).

En particular, el modelo 3018 tiene una mesa y área de trabajo de 30cm x 18cm, lo cual es útil para trabajos pequeños, como lo son las maquetas. En la Figura 1-9 se observa una CNC 3018.



Fuente: https://www.suministro.cl/cnc-de-aprendizaje-y-hobbie-3018-2500mw

Figura 1-9 CNC 3018.

1.5.2 Cortadora láser

La cortadora láser es un tipo de CNC controlada por lenguaje GCODE. Se suele utilizar para cortar materiales mediante calor focalizado en un punto. Es importante saber que existen cortadoras / grabadoras láser que son dañinas a la vista y otras que no. En la Figura 1-10 se muestra una cortadora láser similar a la que puede ser utilizada.

Fuente: https://importacionesfabian.cl/producto/cortadora-y-grabadora-cnc-laser-1000x600-100w-reci/

Figura 1-10 CNC Láser.

1.5.3 Impresora 3D – Creality CR10 Pro V2

Las impresoras 3D son otro tipo de CNC, ya que también siguen instrucciones de forma automática mediante códigos GCODE, el cual les da información de temperaturas, velocidades, posición, entre otros. Estas máquinas permiten realizar partes y piezas mediante el método de adición, ya que va agregando material capa a capa hasta completar la forma de la pieza deseada. Para piezas tipo prototipo o maquetas se suele usar un filamento llamado PLA, el cual es biodegradable y no nocivo para la salud.

La Creality CR10 Pro V2 cuenta con un área de impresión de 300 x 300 x 400 mm, especial para piezas particularmente grandes o para producción en masa. En la Figura 1-11 se muestra el modelo mencionado anteriormente.

Fuente: https://www.nod.cl/tienda/impresora-3d-cr-10s-pro-creality-300x300x400mm/

Figura 1-11 Impresora 3D CR10s PRO v2.

1.6 OBJETIVOS

En esta sección se describen los objetivos, general y específicos, que se deben alcanzar durante el desarrollo de este proyecto, para demostrar la factibilidad de aplicación de un modelo de aprendizaje automático en un sistema industrial.

1.6.1 Objetivo general

Confeccionar una maqueta didáctica portátil con fines educativos en aplicaciones de control automático.

1.6.2 Objetivos específicos

Para cumplir con el objetivo se requiere cumplir los siguientes objetivos específicos:

- Determinar el tipo de maqueta a diseñar y fabricar.
- Especificar requerimientos del dispositivo.
- Creación de firmware básico.
- Diseñar modelos en 3D que componen la maqueta.
- Seleccionar la instrumentación que forma parte del proyecto.
- Identificar el sistema de una de las maquetas.
- Desarrollar diagrama de flujo de la programación del controlador.
- Realizar un estudio de costos

CAPÍTULO 2: DISEÑO E IMPLEMENTACIÓN DE MAQUETA DIDÁCTICA

2 DISEÑO E IMPLEMENTACIÓN DE MAQUETA DIDÁCTICA

En el presente capítulo se indican los pasos realizados, en forma ordenada, para la selección e implementación de las maquetas didácticas, que satisfacen la propuesta del objetivo general y objetivos específicos.

2.1 SELECCIÓN DE CARACTERÍSTICAS BÁSICAS DEL EQUIPO

La o las maquetas deben ser transportables con opciones energéticas que no dependan de la red eléctrica 220 VAC, por lo que se presentan dos opciones para el hardware base. Estas opciones no contemplan la autonomía del equipo, solo sirven como ejercicio para enlistar la cantidad de materiales a utilizar, por lo que la corriente de estos no está indicada.

2.1.1 Opción 1

La opción 1 consiste en una maleta plástica pequeña que permite transportar las maquetas de forma cómoda y segura para el usuario. Además, contempla una batería de Litio de una celda, un módulo cargador a 5 VDC con conector micro USB y un cargador solar, para cargar la batería desde cualquier sitio donde llegue suficiente luz del sol.

En resumen, los materiales necesarios son:

- Maleta plástica pequeña.
- Batería Litio 3.7 VDC.
- Cargador batería de Litio de 3.7 VDC.
- Panel solar.
- Elevador de tensión.

Como las baterías de Litio normales son de 4.2 VDC, se requiere un módulo elevador de tensión para elevar la tensión de la batería a 5 VDC compatible con la mayoría de placas de desarrollo, como lo son Arduino con la familia de microcontroladores Atmega o Espressif con sus microcontroladores que incluyen los chips ESP32.

2.1.2 Opción 2

La opción 2 implementa el uso de un cargador de celular portátil, normalmente conocido como **Power Bank**. Estos equipos incluyen de forma interna una batería de Litio de alta capacidad, un elevador de tensión interno a 5 VDC normalmente de potencia suficiente para el uso de placas de desarrollo.

Los materiales necesarios son:

- Maleta plástica pequeña.
- Power Bank (Cargador portátil de celulares).

Esta opción no contaría con un panel solar para la carga del Power Bank, aunque existe en el mercado dispositivos que incluyen un panel en una en las dos caras planas de mayor superficie de estos, por lo que aún se mantiene la opción del uso de panel solar, quedando limitado solo por la inversión que se desee realizar por el kit completo.

2.1.3 Comparación de opciones

En la Tabla 2-1 se muestra la comparación de precio. Con la **X** se resaltan los materiales a usar entre las opciones de electrónica contempladas en cada maqueta.

Tabla 2-1 Comparación de opciones para el hardware base de las maquetas.

Material	**Costo**	**Opción 1**	**Opción 2**
Maleta	$39.990	**X**	**X**
Batería 6000mAh	$28.490	**X**	
Cargador de batería solar	$7.590	**X**	
Cargador de batería	$2.175	**X**	
Panel solar	$4.265	**X**	
Elevador de tensión	$1.890	**X**	
Power Bank	$16.490	**X**	**X**
	Total	$84.400	$56.480

Fuente: Elaboración propia.

La opción 2 es la más barata y práctica en cuanto a cantidad de partes y piezas. También se estima que el precio de la maleta y del Power Bank puede disminuir buscando modelos alternativos.

Como la **Opción 2** usa menos componentes y resulta ser más económica, se implementa el uso de una maleta y un Power Bank, en particular los mostrados en la Figura 2-1.

Fuente: https://fotosol.cl/products/cargador-portatil-audiolab-10000mah-1?variant=31583435817071 y https://www.easy.cl/caja-de-herramientas-20-mj-6060-robust-1277654/p

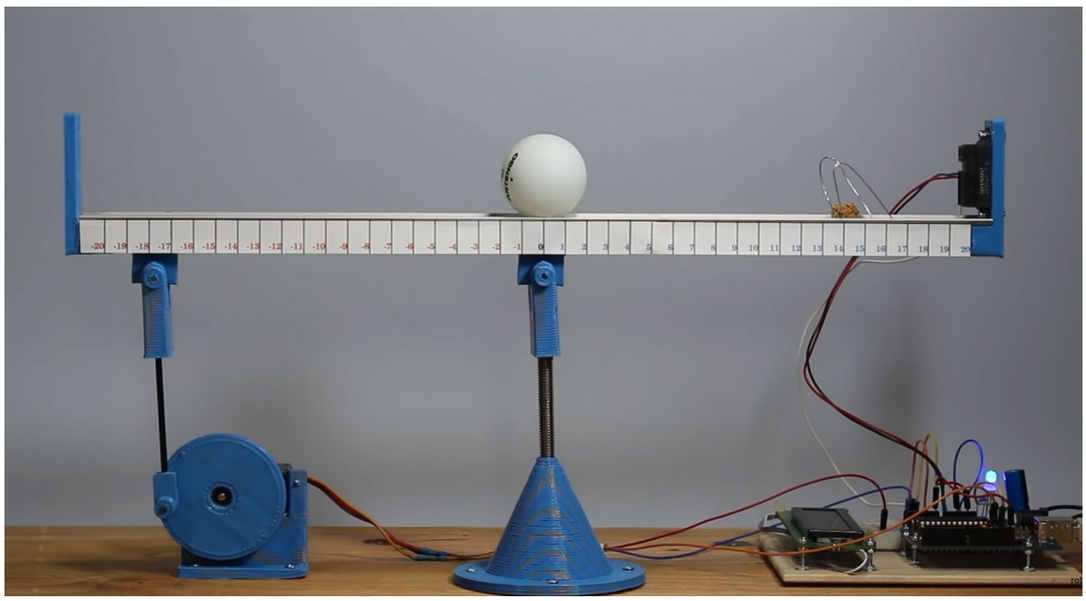
Figura 2-1 Power Bank y maleta implementada como base del sistema.

2.2 SELECCIÓN DE MAQUETAS A FABRICAR

Dentro de la gran cantidad de maquetas que existen en Internet y las distintas maquetas que se pueden diseñar, se contempla la implementación de maquetas simples de forma inicial que permitan la fácil adaptación y almacenaje dentro de la maleta, para asegurar poder transportarlas de forma cómodas, sin agregar dificultad en armado u otras cosas.

Entre las maquetas de control de nivel de líquido, robots balancín, sistemas de control de temperatura, entre otros, se han seleccionado dos maquetas de control de posición. Estas maquetas son: "*Maqueta Barra y Bola*" y "Demostración de control de un Servomotor".

La maqueta "Barra y Bola" consiste en controlar la posición de una bola puesta sobre una barra. Para mover la bola se utiliza un servomotor conectado a un extremo de la barra que permite variar la inclinación de ésta, provocando que, por la aceleración de la gravedad, la bola recorra la barra. Para cerrar el lazo de control se utiliza un sensor de distancia, normalmente con tecnología de ultrasonido o láser, como se muestra en el ejemplo de la Figura 2-2.



Fuente: https://roble.uno/control-pid-barra-y-bola-arduino/?v=028a54c4d521

Figura 2-2 Maqueta de control de posición Barra y Bola.

La maqueta de “Demostración de control de un Servomotor” consiste en controlar la posición de un rotor de motor DC, mediante la utilización de un potenciómetro conectado por medio de una banda o similar. Esto permite que, al girar el rotor el potenciómetro igual gire variando el voltaje en su punto medio y así se identifique la posición en grados en la que se encuentra el rotor. Un ejemplo armado de la maqueta se muestra en la Figura 2-3.

Fuente: https://blog.330ohms.com/2021/06/02/que-es-un-control-pid/

Figura 2-3 Maqueta control de posición: Demostración de control de un Servomotor.

2.3 SELECCIÓN ELECTRÓNICA

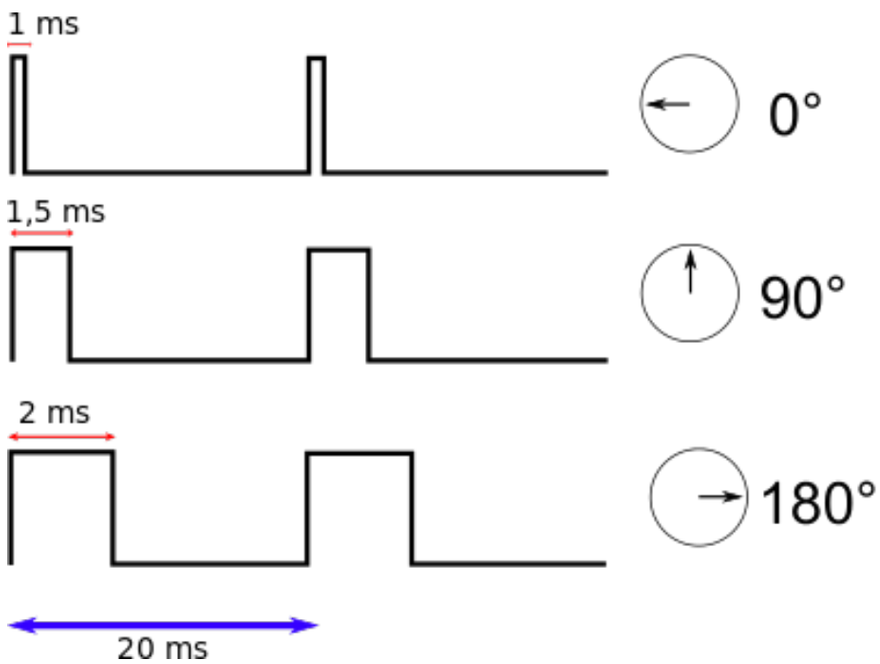
Las maquetas tienen componentes electrónicos distintos, para mostrar y explicar la electrónica usada, se analizan de forma independiente.

2.3.1 Maqueta “Barra y bola”

La electrónica de esta maqueta es reducida, solo requiere un actuador que permita ajustar la inclinación de la barra y un sensor que permita la medición de distancia o posición de la bola.

Para el actuador se puede utilizar un **Motor DC**, pero agrega complejidad el no contar con la información de la posición del rotor, por lo que hay que implementar un encoder o sistema similar con el fin de conocer los grados que ha girado y así conocer la inclinación de la barra. También se puede utilizar un **Servomotor** que implementa un motor DC y un potenciómetro en un empaquetado reducido y permite conocer el grado de rotación del rotor, el cual suele estar limitado entre de 0 a 180 grados en la mayoría de casos.

El Servomotor recibe una señal PWM a una frecuencia determinada, la cual utiliza para leer el tiempo del pulso alto y así detectar el grado de rotación en que debe estar el rotor. Normalmente el ancho del pulso es de 1 ms a 2 ms que corresponde a 0 y a 180 grados respectivamente, como se muestra en la Figura 2-4.



Fuente: https://es.wikipedia.org/wiki/Servomotor_de_modelismo

Figura 2-4 Pulso PWM y posición del rotor.

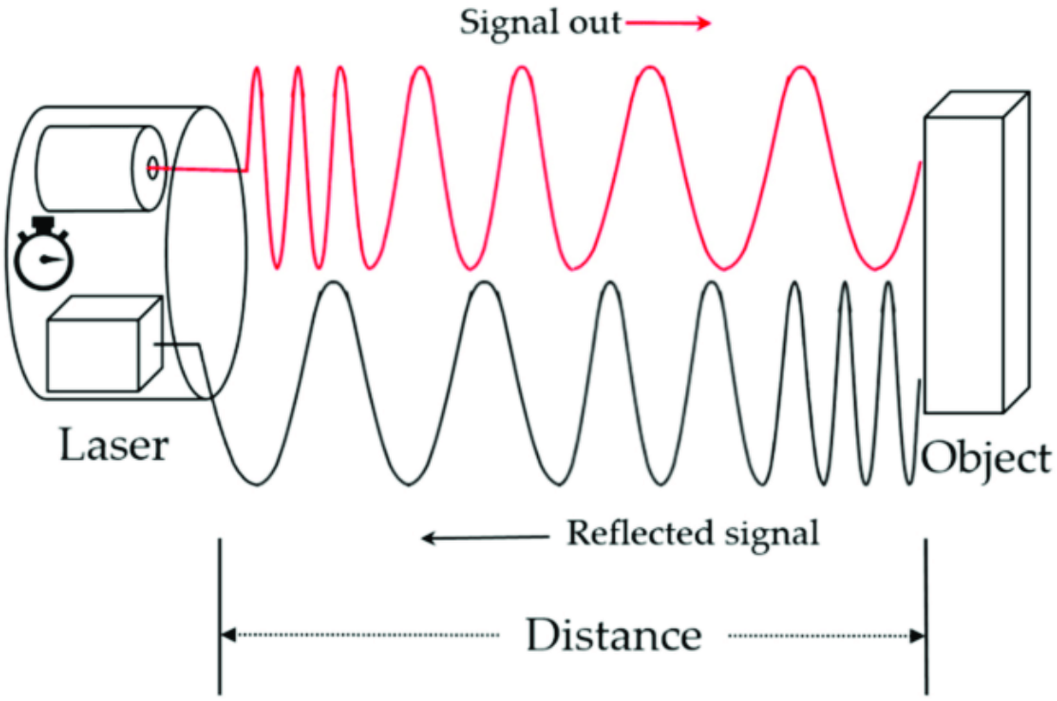
El funcionamiento del servomotor depende de tres cables, como se observa en la Figura 2-5. El cable rojo corresponde a alimentación +5 VDC, el cable café es GND y el cable amarillo es para la señal PWM. La distribución de pines, colores de los cables y tiempo de pulsos varía según el modelo y fabricante del servomotor, por lo que es importante conocer el modelo y obtener la hoja de datos correspondiente.

Fuente: https://afel.cl/producto/micro-servomotor-sg90/

Figura 2-5 Servomotor Tower Pro SG90.

Para medir la posición de la bola se puede usar un **sensor de distancia** con tecnología láser o ultrasonido, ambas basadas en la medición de tiempo de vuelo ToF de la señal emitida. En la Figura 2-6 se pueden observar dos señales (sonido o luz), una emitida y una señal que corresponde a la reflexión de la señal emitida, rebote de ésta con un objeto que se interpone en el camino. Por lo tanto, existen dos tipos de objetos actuando dentro de un sensor de distancia, un emisor y un receptor.

También existen sensores de distancia que cuentan con un solo transductor, el cual actúa de emisor y receptor. La desventaja de estos sensores es que la distancia mínima de lectura suele ser mayor a 20 cm, ya que debe cambiar de un modo a otro de forma rápida para no perder la integridad de la señal reflectada.

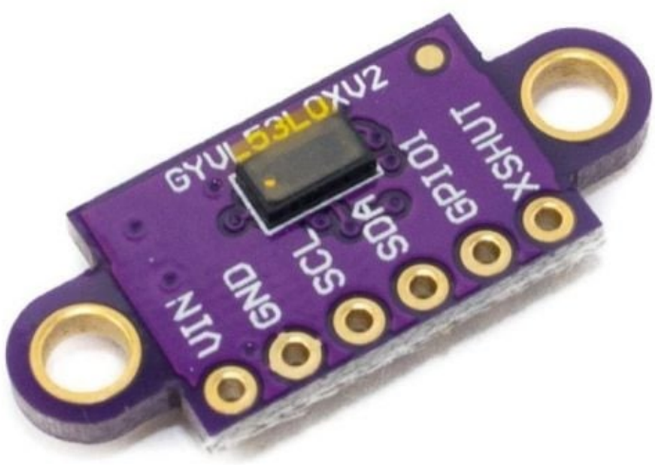


Fuente: https://www.researchgate.net/figure/The-schematic-diagram-of-a-time-of-flight-TOF-sensor_fig4_335317560

Figura 2-6 Explicación de ToF (Time of Flight).

Para la implementación de la maqueta, se utiliza el sensor láser VL53L0X mostrado en la Figura 2-7. La ventaja de este sensor en comparación a las versiones ultrasónicas más conocidas del mercado como el HC-SR04, es el tamaño reducido y también, que no presenta problemas de reflexiones falsas que tienen los sensores ultrasónicos en espacios de trabajo pequeños.

El sensor láser VL53L0X se puede alimentar con 3.3 o 5 VDC, cuenta con un rango de 50 a 1200 mm y se puede configurar y obtener el dato de distancia mediante protocolo I2C.



Fuente: https://afel.cl/producto/sensor-de-distancia-por-tof-vl53l0x/

Figura 2-7 Sensor de distancia láser VL53L0X.

El principal problema que puede presentar el uso de sensores láser será la luminosidad del lugar donde se use la maqueta, el color/material de la bola que se utilice y la distancia de lectura, mientras más lejos mayor es el error de lectura. Existe una actualización al módulo de sensor láser llamado VL5L1X, el cual es muy estable y que se ve menos afectado por esas variables, pero el precio de venta en Chile es extremadamente alto, por lo que fue descartado de esta maqueta, pero es sugerido como una futura actualización.

2.3.2 Maqueta “Demostración servo motor”

Esta maqueta presenta un grado de dificultad mayor en la electrónica debido a la cantidad de sensores y actuadores que usa. En particular, se desea lograr una maqueta con tres potenciómetros que permitan ajustar los parámetros de un controlador PID, otro potenciómetro para ajustar el valor de setpoint y un último potenciómetro para conocer la posición del rotor. Además, se quiere agregar una pantalla para poder visualizar los valores que se ajustan y la posición exacta del rotor.

Para la lectura de los potenciómetros PID y setpoint se usa un módulo ADC llamado ADS1115, el cual otorga al microcontrolador 4 ADC mediante comunicación I2C. Para el potenciómetro del rotor, se usará un ADC directo del microcontrolador, ya que la lectura será más rápida. El control del motor se realizará con un módulo puente H modelo L9110S. La pantalla a utilizar será la OLED de 1.3” y su protocolo de comunicación es por I2C.

En total, los materiales a usar son:

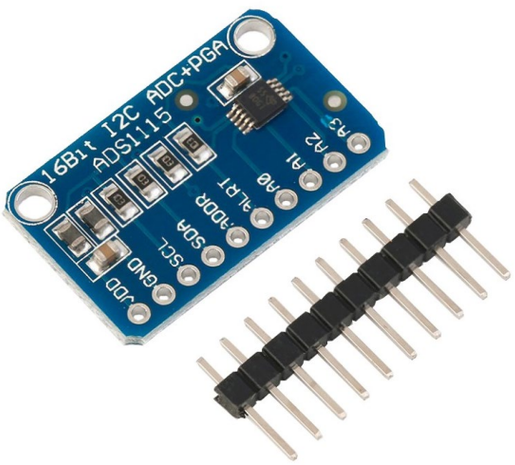
- 5 potenciómetros lineales de 100 KΩ.
- 1 Motor DC.
- 1 Pantalla I2C OLED 1.3”.
- 1 Módulo ADC ADS1115.
- 1 Módulo puente H L9110S.

Los potenciómetros se utilizan como divisor de voltaje energizando los extremos a 3.3 VDC y utilizando el punto medio para medir el voltaje de estos. El potenciómetro a utilizar es de panel, bastante típico dentro del mercado, como se muestra en la Figura 2-8.

Fuente: https://globalelectronica.cl/rotatorio-panel-y-pcb-1-vuelta/1640-pot5kb.html

Figura 2-8 Potenciómetro lineal de panel.

La lectura de voltaje de los potenciómetros será realizada por el módulo conversor análogo digital (ADC) ADS1115, el cual posee cuatro canales de conversión de 16 bits y un canal de comunicación I2C, como se observa en la Figura 2-9.



Fuente: https://maxelectronica.cl/complementos-y-accesorios/303-modulo-ads1115-conversor-adc-i2c-16-bits-4-canales.html

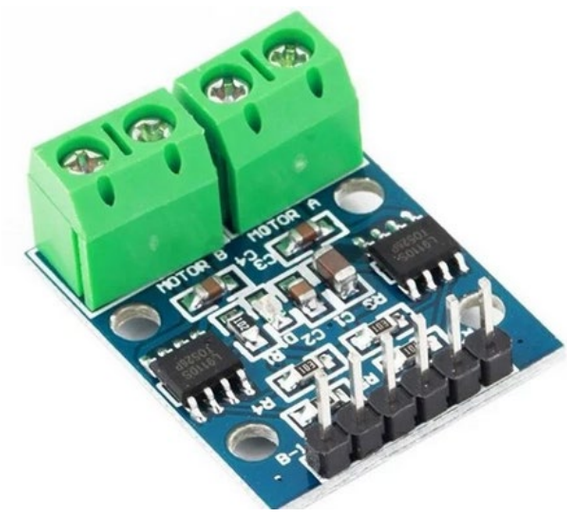
Figura 2-9 Módulo ADC ADS1115.

Para reducir el tamaño de la maqueta se utiliza un mini motor DC modelo N20, como se muestra en la Figura 2-10. Este motor resulta ser bastante pequeño y cuenta con una caja de reducción que limita las revoluciones por minuto máxima.

Fuente: https://maxelectronica.cl/moto-reductores/493-mini-motor-con-caja-reductora-modelo-n20-100rpm-6v-eje-3mm-d.html

Figura 2-10 Motor DC N20.

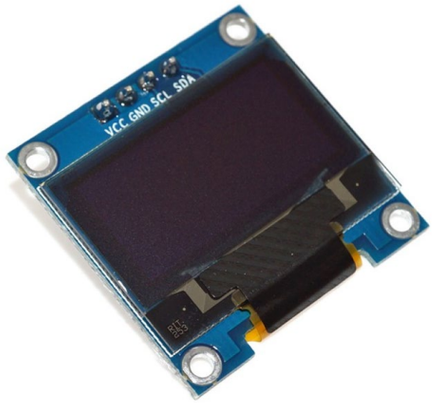
Para el control del motor se utiliza un módulo puente H pequeño modelo L9110s, el cual tiene dos componentes integrados (IC) para dos motores independientes. El módulo mostrado en la Figura 2-11, soporta voltajes de operación entre 2.5 – 12 VDC, una corriente de 800 mA por integrado e incluye diodos de protección internos. El control de velocidad se logra mediante una señal PWM y el control de sentido de giro del rotor mediante dos pines digitales.



Fuente: https://afel.cl/producto/controlador-motor-stepper-dc-puente-h-l9110s/

Figura 2-11 Módulo puente H modelo L9110s.

La pantalla usada es una de tecnología OLED de 128x64 pixeles la cual permite la implementación de texto, gráficos, animaciones, entre otras cosas. Otra ventaja es que funciona sobre el puerto I2C, por lo que ocupa solo dos cables de datos y dos de alimentación, como se observa en la Figura 2-12.



Fuente: https://maxelectronica.cl/pantallas-lcd/485-pantalla-monocromatica-096-lcd-oled-grafico-i2c-128x64-blanco.html

Figura 2-12 Pantalla I2C Oled 128x64 pixeles.

2.3.3 Placa de desarrollo

La selección de la placa de desarrollo a utilizar se basa en precios y características, comparando las placas: *Arduino Uno*, *Arduino Leonardo* y *ESP32 DEVKIT 1*.

Para una mejor visualización de los precios, en la Tabla 2-2 se muestra el valor de las placas originales y las copias denominadas Clon.

Tabla 2-2 Comparación de valores de placas de desarrollo.

Modelo	Valor Original	Valor Clon
Arduino Uno	$22.990	$12.815
Arduino Leonardo	$36.990	$12.490
ESP32 DEVKIT 1	-	$8.990

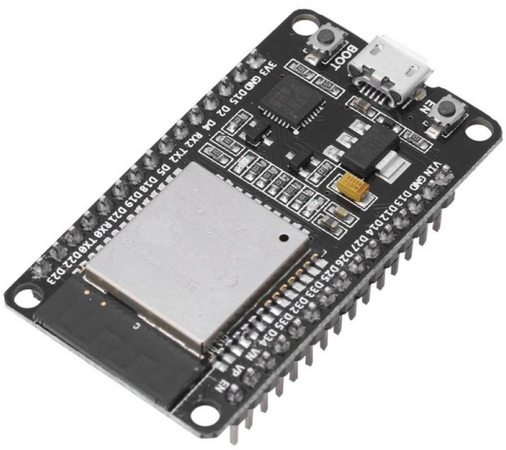
Fuente: Elaboración propia.

Las características generales de cada placa de desarrollo son las siguientes:

- **Arduino UNO:** Trabaja con el microcontrolador Atmega 328, el cual posee una velocidad de reloj de 16 Mhz y 32 KB de flash. Posee 20 pines de entrada y salida (GPIO) donde 6 de ellos tienen funcionalidad PWM y otros 6 tienen funcionalidad ADC, para realizar mediciones análogas.
- **Arduino Leonardo:** Trabaja con el microcontrolador Atmega 32u4, el cual posee interfaz USB integrada, es decir, no depende de un chip conversor USB a TTL como el Arduino UNO. El resto de las características son bastantes similares al microcontrolador Atmega 328 del Arduino UNO, ya que cuenta con 16Mhz velocidad de reloj, 32KB de flash, tiene 20 pines de entradas y salidas (GPIO) donde 7 de ellos pueden ser PWM y 12 tienen funcionalidad ADC.

- **ESP32 DEVKIT V1:** Trabaja con un encapsulado denominado ESP32 WROOM, el cuál incorpora funcionalidades WiFi, Bluetooth, un procesador de dos núcleos, arquitectura de 32bits, memoria RAM de 512KB, memoria flash de 16MB (dependiendo del modelo), frecuencia de reloj de 160Mhz, 36 pines GPIO (4 pines de entradas, 5 pines de inicio de sistema), 18 pines tienen capacidad ADC y 2 DAC.

En resumen, la placa de desarrollo **ESP32 DEVKIT V1** mostrada en la Figura 2-13, es superior a los otros dos modelos comparados, debido a las capacidades de memoria, velocidad y presencia de WiFi / Bluetooth esenciales para la extracción y visualización de datos. Además, la placa de desarrollo es de bajo costo por lo que se escoge para la medición y control de las maquetas.



Fuente: https://maxelectronica.cl/wifi-24ghz/381-tarjeta-de-desarrollo-doit-modulo-esp32-wifi-y-bluetooth-v42.html

Figura 2-13 ESP32 DEV KIT V1.

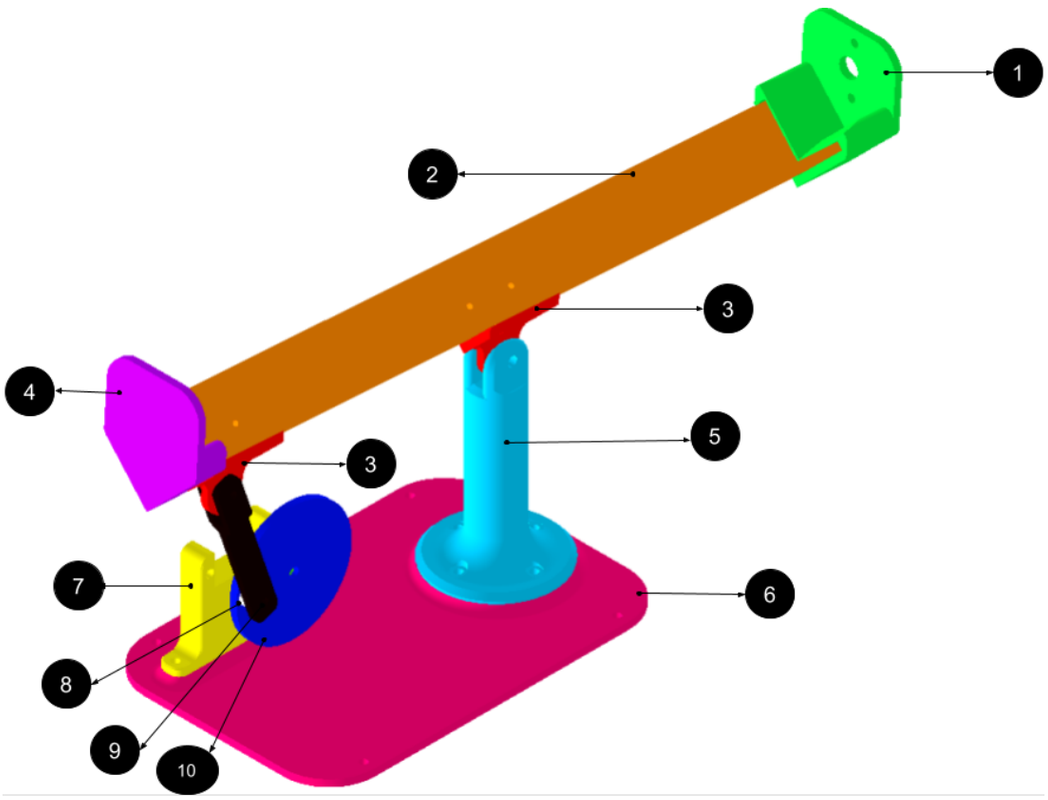
2.4 DISEÑO 3D

El diseño 3D para ambas maquetas se desarrolla de cero utilizando el software de diseño 3D llamado **Autodesk Inventor** con la licencia estudiante. El objetivo principal de los diseños consiste en generar una fácil fabricación y armado.

Es importante el diseño orientado a la impresión 3D, ya que se necesita disminuir la cantidad de soportes que se deban crear para la correcta impresión, permitiendo optimizar el consumo de material innecesario y el tiempo de impresión.

2.4.1 Partes y piezas: “Barra y bola”

Para la maqueta de “Barra y bola” se generan 10 piezas, indicadas en la Figura 2-14. El tiempo de impresión en total es de 7 horas y 38 minutos aproximadamente.



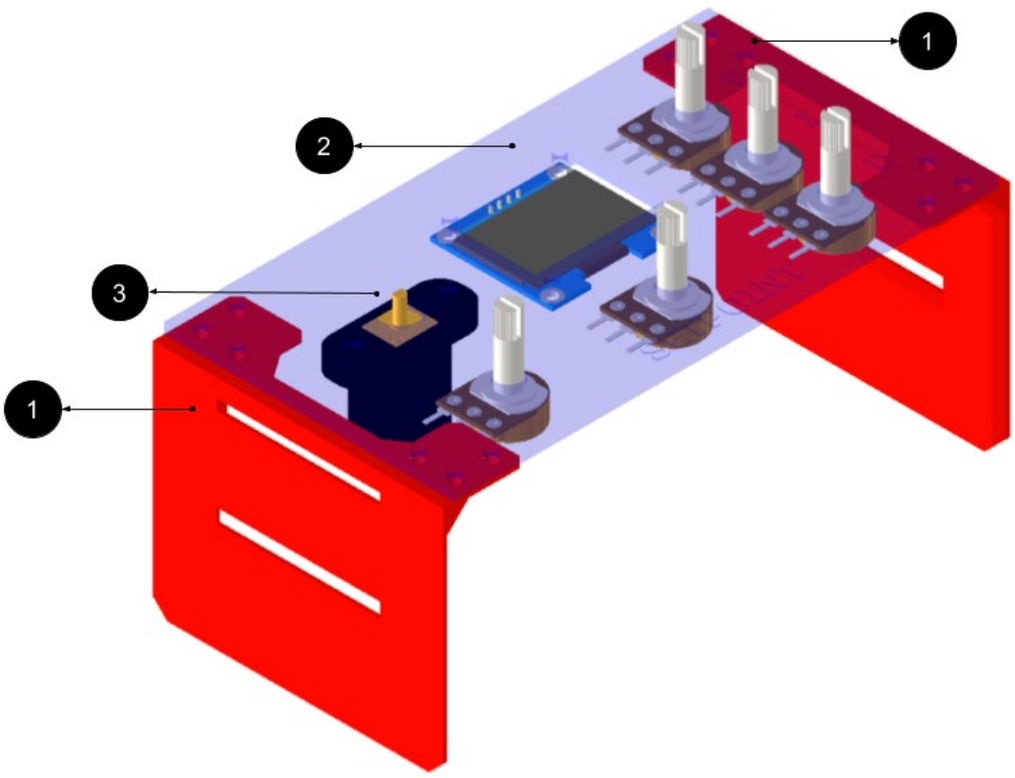
Fuente: Elaboración propia.

Figura 2-14 Partes y piezas maqueta "Barra y bola".

1. **Tapa y soporte sensor láser VL53L0X**: Esta tapa puede ser reemplazada para la utilización de otros tipos de sensores.
2. **Barra**: Guía para el movimiento de la bola.
3. **Soportes de pivote:** Dos piezas iguales que permiten la inclinación de la barra, una sujetando la parte central y la otra sujetando un extremo de ésta.
4. **Tapa**: Evita que la bola se salga de la barra.
5. **Pilar central**: Permite el punto de pivote central de la barra
6. **Base de soporte maqueta**: Permite ensamblar y dar superficie de apoyo a toda la maqueta.
7. **Soporte servomotor**: Permite posicionar de forma segura el servomotor.
8. **Separador rueda – brazo**: Evita que el brazo choque con la rueda produciendo daños.
9. **Brazo**: Permite conectar la rueda controlada por el servomotor a la barra.
10. **Rueda**: Permite transmitir de mejor forma el movimiento del servo hacia la barra.

2.4.2 Partes y piezas: “Demostración de control de un Servomotor”

Para la maqueta “Demostración de control de un Servomotor” se utiliza tres partes simples, indicadas en la Figura 2-15. El tiempo de impresión para esta maqueta es de 3 horas aproximadamente.



Fuente: Elaboración propia.

Figura 2-15 Partes y piezas maqueta "Demostración de control de un Servomotor".

1. **Soporte microcontrolador y base:** Permite adosar la tarjeta de desarrollo en uno de los costados y mantener la maqueta de pie.
2. **Panel**: Permite integrar y sujetar todos los componentes del sistema, como potenciómetros, motor, pantalla y módulos.
3. **Soporte motor DC**: Permite posicionar y asegurar el motor DC para el correcto funcionamiento del sistema.

A este sistema en particular se le debe incluir una correa y unos rodillos para poder transmitir el movimiento del eje del motor hacia el potenciómetro, así poder medir el ángulo en el que se encuentra éste, y poder hacer el control de posición de forma correcta.

2.5 FABRICACIÓN Y ARMADO DE LAS MAQUETAS

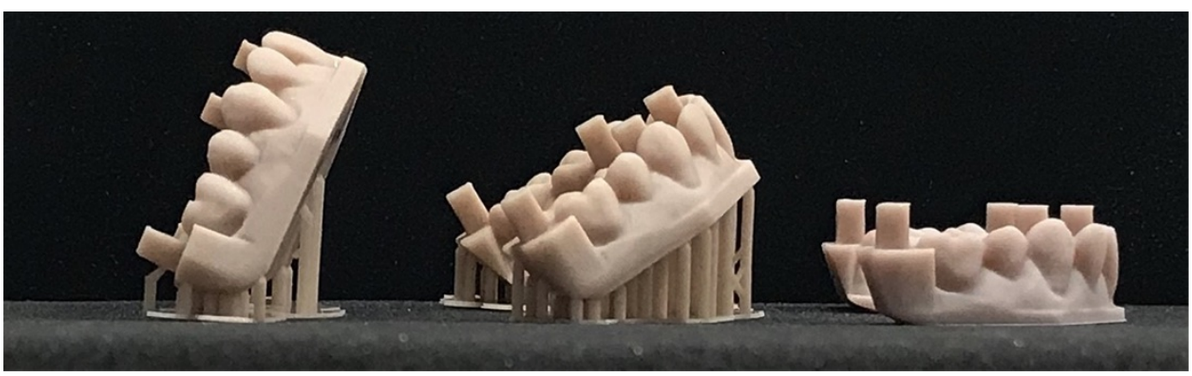
Para la fabricación de las maquetas se ha utilizado la impresora 3D Creality CR10-Pro, debido a su velocidad y gran volumen de impresión. En la Figura 2-16 se observa el funcionamiento de la impresora, imprimiendo la “Barra” de la maqueta “Barra y bola”.

Fuente: Elaboración propia.

Figura 2-16 Impresión en progreso de la barra, de la maqueta "Barra y bola".

La configuración de temperatura para el extrusor es de 230 °C y 60 °C para la cama. Estas temperaturas varían mucho dependiendo del tipo de material utilizado y por el color del filamento. Para colores claros, por lo general, se requiere mayor temperatura mientras que para colores oscuros se requiere temperaturas más bajas.

Para imprimir las piezas de forma rápida, se debe tener en cuenta el diseño y el posicionamiento que se les da a estas al momento de ingresar el archivo stl a la aplicación Slicer. Esto puede generar que el software genere soportes de forma innecesaria, aumentando el tiempo de impresión y consumiendo mayor material que finalmente se desperdicia, como se observa en la Figura 2-17.

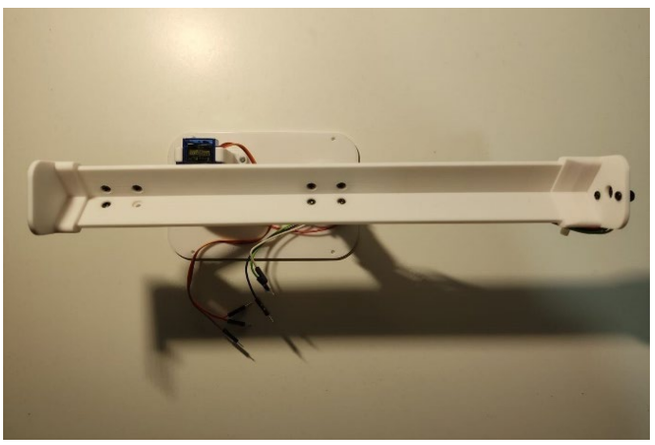


Fuente: https://www.aegisdentalnetwork.com/cced/2020/11/the-effect-of-print-angulation-on-the-surface-roughness-of-3d-printed-models.

Figura 2-17 Ejemplo de pieza 3D con soportes y sin soportes de impresión 3D

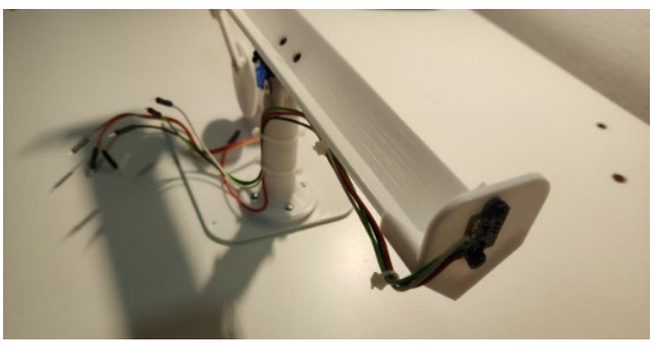
2.5.1 Armado: “Barra y bola”

El resultado después de que todas las piezas fueran impresas y ensambladas de la maqueta “**Barra y bola**” se muestra en la Figura 2-18, Figura 2-19 y Figura 2-20.



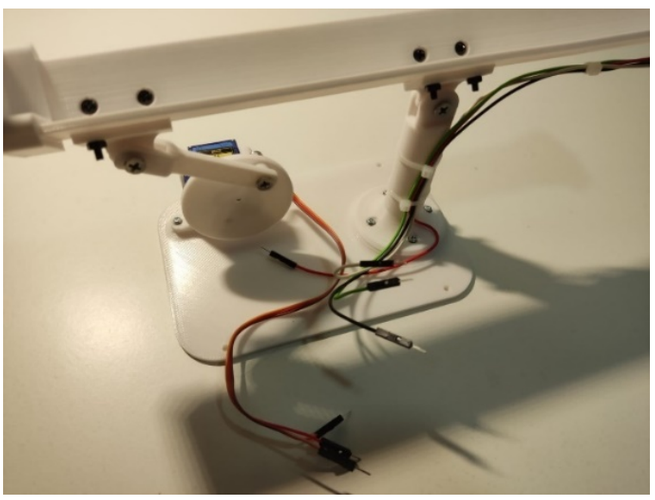
Fuente: Elaboración propia.

Figura 2-18 Vista superior maqueta "Barra y bola".



Fuente: Elaboración propia.

Figura 2-19 Vista soporte sensor láser maqueta "Barra y bola".

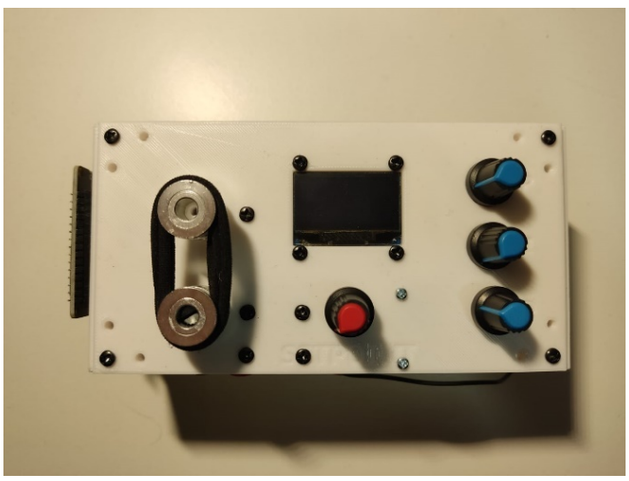


Fuente: Elaboración propia.

Figura 2-20 Vista costado maqueta "Barra y bola".

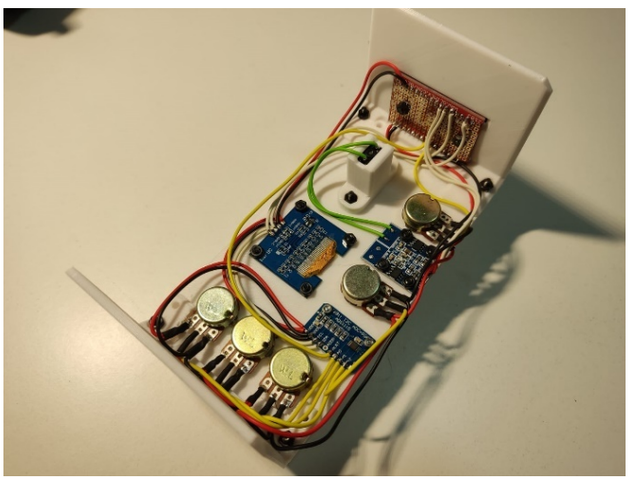
2.5.2 Armado: “Demostración de control de un Servomotor”

El resultado final para la maqueta “**Demostración de control de un Servomotor**” se muestran en la Figura 2-21 y Figura 2-22.



Fuente: Elaboración propia.

Figura 2-21 Vista frontal maqueta "Demostración de control de un Servomotor".



Fuente: Elaboración propia.

Figura 2-22 Vista electrónica maqueta "Demostración de control de un Servomotor".

En la Figura 2-21 se observa la estética del panel frontal, que interactúa con el usuario y en la Figura 2-22 se observa la parte trasera en donde están todos los módulos utilizados y las interconexiones realizadas.

2.6 IMPLEMENTACIÓN DE CIRCUITOS ELECTRÓNICOS

Para explicar de mejor forma los circuitos electrónicos implementados en las maquetas se generan esquemáticos de ambos proyectos en el software Altium Designer.

2.6.1 Circuito electrónico: “Barra y bola”

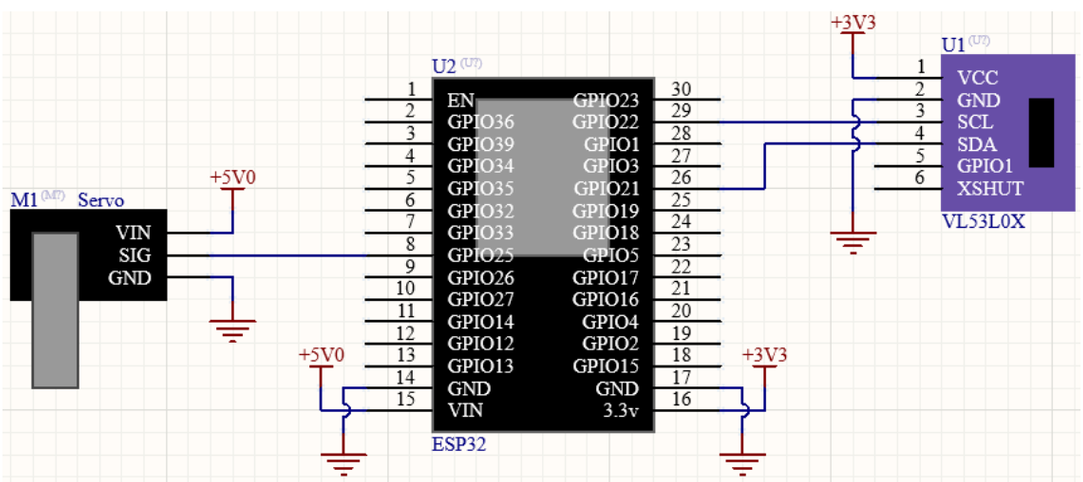
El circuito electrónico mostrado en la Figura 2-23, requiere un puerto I2C que puede ser asignado a cualquier pin del ESP32 y un puerto digital para generar PWM así controlar el servomotor.

La alimentación del sensor láser es de +3.3 VDC para la compatibilidad de pines del ESP32, ya que el Datasheet indica que no son +5 VDC compatibles. La alimentación del servomotor proviene del puerto USB +5 VDC, para el correcto funcionamiento del motor interno.

En resumen, el circuito utiliza los siguientes pines del microcontrolador:

- **GPIO22 (SCL) y GPIO21 (SDA)**: Para la comunicación I2C con el sensor de distancia láser.
- **GPIO25 (OUTPUT):** Para el control de posición del servomotor.
- **Pines para alimentación (5 VDC, 3.3 VDC y GND)**: Para la alimentación del servomotor y del sensor de distancia láser.

Como el circuito está montado en una protoboard, esta disposición de pines puede ser modificada a gusto, pero también debe ser modificado en el firmware para que funcione correctamente.



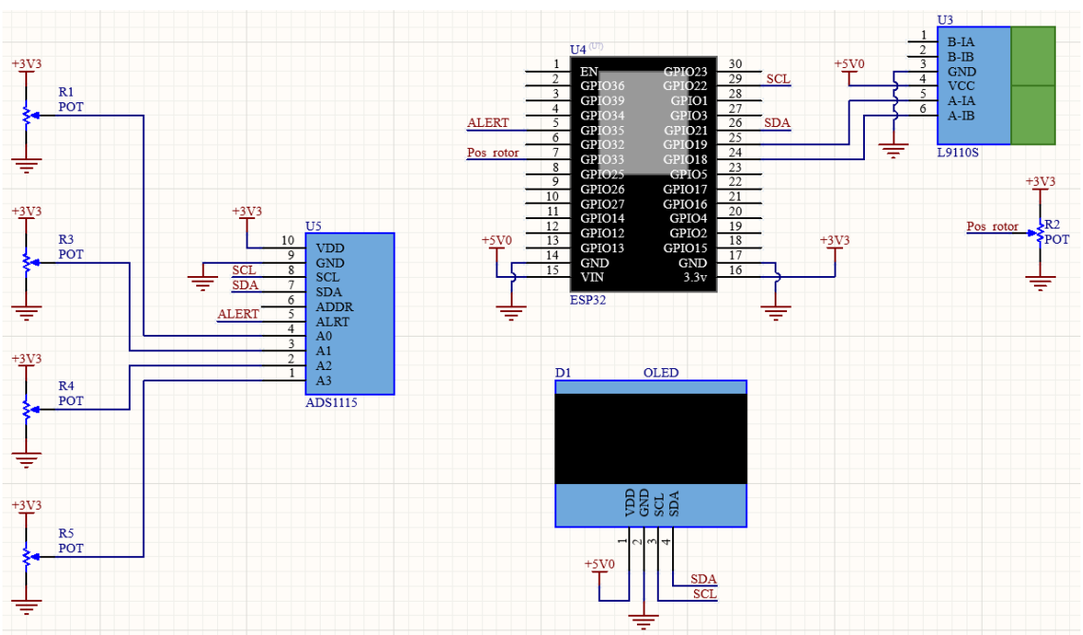
Fuente: Elaboración propia.

Figura 2-23 Diagrama electrónico maqueta "Barra y bola".

2.6.2 Circuito de maqueta “Demostración de control de un Servomotor”

Este circuito es más extenso, como se observa en la Figura 2-24. Para simplificar la explicación, el circuito se divide en los puntos: lectura análoga, comunicación I2C y control motor.

- **Lectura análoga:** compuesta por los cinco potenciómetros, alimentados con +3.3 VDC. Cuatro de ellos son leídos con el módulo ADC, que utiliza el IC ADS1115. Uno solo se conecta directo a un pin análogo del ESP32 (GPIO33).
- **Comunicación I2C:** En el puerto I2C compuesto por SCL (GPIO22) y SDA (GPIO21), se conectan el módulo ADC y la pantalla OLED, ambos alimentados a +3.3 VDC. Gracias al protocolo I2C, los módulos se pueden comunicar sin problemas con la tarjeta de desarrollo debido a las direcciones asignadas a estos. También, el módulo ADC utiliza un pin digital de la placa de desarrollo como entrada (GPIO35), para alertar cuando se ha realizado la medición análoga.
- **Control motor:** El módulo puente H contiene dos IC modelo L9110s, para el control individual de dos motores (un motor por IC). El módulo se alimenta a +5 VDC para el correcto funcionamiento del motor y utiliza dos pines digitales como salida (GPIO19 y GPIO18), con los cuales se puede configurar el giro del rotor, izquierda, derecha o detenido.

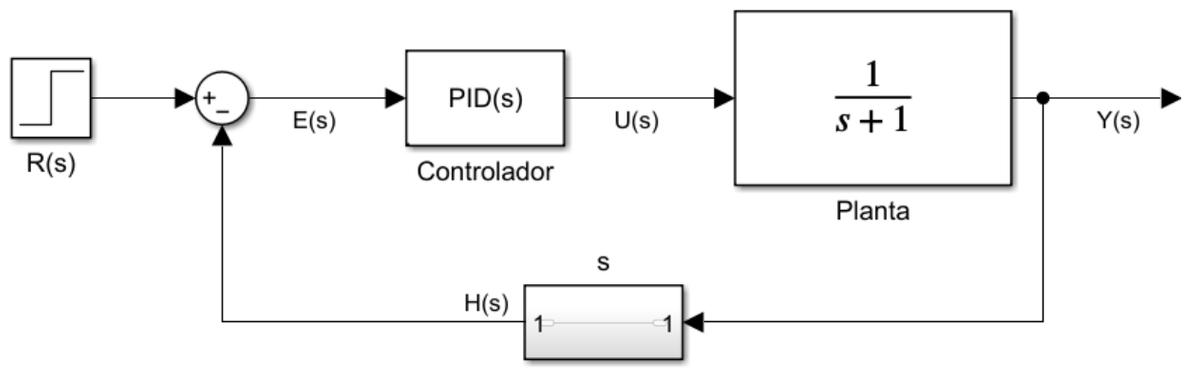


Fuente: Elaboración propia.

Figura 2-24 Diagrama electrónico maqueta "Demostración de control de un Servomotor".

2.7 CONTROLADOR PID

El controlador PID es una de las herramientas utilizadas para el control y automatización de sistemas o procesos industriales. Este controlador se basa en la medición de la salida del proceso, para calcular el error con respecto a un valor deseado y así realizar la corrección del sistema, hasta que el error sea igual a cero. A este tipo de controladores se les clasifica de lazo cerrado ya que, como se observa en la Figura 2-25, el sistema recibe una retroalimentación del estado de su salida.



Fuente: Elaboración propia, en Matlab.

Figura 2-25 Diagrama de ejemplo del funcionamiento de un controlador en lazo cerrado.

Al conjunto de bloques mostrados en la figura anterior, se denomina como un sistema, el cual consta de diferentes partes y piezas para su correcto funcionamiento.

El bloque denominado **R(s)**, corresponde al valor de referencia que se espera obtener en la salida de la planta Y(s). Este valor también puede ser llamado como: Referencia, Setpoint o consigna.

El valor **E(s)** corresponde al error presente en el sistema, este error es calculado mediante la resta del valor de referencia con la medición de la variable de salida medida de la planta H(s). Por ejemplo, si el valor de referencia es de 10 unidades y la salida de la planta es de 4 unidades, el error en E(s) será igual a 6 unidades.

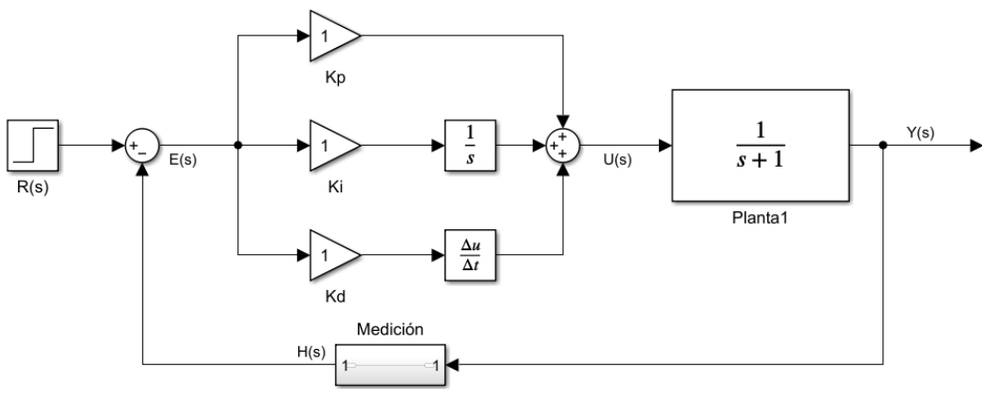
El **Controlador PID**, genera una salida denominada como **U(s)** que corresponde a la señal para el actuador, el cual realizará los cambios necesarios para afectar la planta y así poder lograr un error E(s) igual a cero.

Justo después del controlador se encuentra la **Planta**, el cual puede ser un horno, un estanque, un motor, entre otros, por lo que el valor de **Y(s)** puede adoptar cualquier unidad de medida.

En la retroalimentación está el bloque de medición que entrega la señal **H(s)** que corresponde a la señal generada por el instrumento de medición conectado a la salida Y(s).

El controlador PID se compone de tres partes fundamentales: Proporcional, integral y derivativa. La parte **Proporcional**, corresponde a la multiplicación del error por un constante denominada **Kp**. Esta parte intenta disminuir el error del sistema de una forma rápida. La parte **Derivativa**, consiste en obtener la derivada o pendiente del error, multiplicado por un factor **Kd**. Esta parte intenta anticipar el funcionamiento del sistema, aumentando la estabilidad de la planta. La parte **Integral**, se multiplica por un factor denominado **Ki**. Esta parte corresponde al error acumulado del sistema, con esto se logra eliminar el error en régimen permanente, característica que no puede realizar la parte derivativa.

Los factores Kp, Ki y Kd, permiten ajustar o sintonizar el controlador, para obtener una respuesta de acuerdo a los requerimientos de diseño. En la Figura 2-26 se muestra el sistema con controlador PID desglosado.



Fuente: Elaboración propia, en Matlab.

Figura 2-26 Diagrama de ejemplo de un sistema de lazo cerrado con controlador PID paralelo.

2.7.1 Tiempo continuo y tiempo discreto

Todos los sistemas o plantas físicas a controlar funcionan en tiempo continuo, es decir sin pausas, ya que el tiempo no se puede detener. Para poder controlar el sistema en tiempo continuo con controladores digitales, se requiere trabajar estos últimos en tiempo discreto debido a que el microcontrolador funciona con pulsos de reloj cada cierto tiempo, dependiendo de su cristal.

Es por esto, que es importante conocer el modelo matemático de la planta en tiempo continuo y contar con el controlador PID en tiempo discreto, ya que la planta funcionará en la vida real, mientras que el control lo realizará un microcontrolador.

Un método para controlar una planta de tiempo continuo, consiste en conocer o tener el modelo matemático de ésta, representada con una ecuación diferencial, ya que ésta permite describir cómo cambia o evoluciona el sistema en el tiempo. Para obtener una ecuación

diferencial, se requiere de harto conocimiento de la física para lograr entender cómo se relacionan todas las partes y piezas del sistema. Además, es importante que el modelo matemático varíe en el tiempo.

Si el modelo genera una respuesta **no lineal** se debe linealizar, de lo contrario se puede aplicar directamente la herramienta matemática llamada **Laplace**, operador **lineal** utilizado para simplificar la resolución de ecuaciones diferenciales, llevando la ecuación a otro dominio denominado **S**.

Contar con la función de transferencia en el dominio de S, permite la utilización del software **Matlab** para la correcta sintonización del controlador PID, mediante los valores para Kp, Ki y Kd que entregará con la herramienta **PID Tuner**.

Existe otro método para obtener la función de transferencia de una planta y consiste en la obtención de datos en tiempo real, para observar y analizar el comportamiento de la salida del sistema a determinado estímulo de entrada. Estos datos, pueden ser procesados por Matlab mediante la herramienta **systemIdentification** la cual, después de analizar entregará la función de transferencia que mejor se acomode a la planta.

2.7.2 Controlador PID en tiempo discreto

El controlador PID en tiempo discreto implementado, se basa en el trabajo de título "**CREACIÓN DE HARDWARE DE DESARROLLO OPEN SOURCE Y APLICACIÓN DE CONTROL PID DISCRETO**", realizado por Maximiliano Ramírez, Ingeniero de Ejecución En Control y Automatización Industrial, de la Universidad Federico Santa María sede José Miguel Carrera, Viña del Mar.

En el mencionado documento, el autor explica que la función de transferencia de un controlador PID en tiempo continuo se expresa de la siguiente forma:

$$\frac{U(s)}{E(s)} = K_p + \frac{K_i}{S} + K_d S \frac{N}{S+N}$$

Donde:

U(s): Salida del controlador PID.

E(s): Error en la planta.

Kp: Acción proporcional del controlador PID.

Ki: Acción integral del controlador PID.

Kd: Acción derivativa del controlador PID.

S: Dominio S, debido a transformada de Laplace.

N: Coeficiente de filtro de primer orden para la acción derivativa, necesario para evitar cambios bruscos en el controlador. Como la acción derivativa obtiene, en palabras simples, la pendiente del error del sistema, si el error cambia de forma rápida, la acción derivativa tenderá a infinito, por lo que es importante contar con un filtro que pueda amortiguar este cambio y que pueda ser ajustado para cada tipo de planta y velocidad de respuesta requerida.

Para poder utilizar un controlador PID en un microcontrolador, es necesario transformar la función de transferencia anterior al dominio de tiempo discreto y luego aplicar la transformada Z. Para mayor simplificación, la herramienta PID Tuner del software Maltab entrega la función del controlador en dominio discreto, como se observa en la Figura 2-27.

$$P + I \cdot \frac{T_s}{2} \frac{z+1}{z-1} + D \frac{N}{1 + N \cdot T_s \frac{1}{z-1}}$$

Fuente: Elaboración propia, captura de pantalla de PID Tuner, Matlab.

Figura 2-27 Función de transferencia controlador PID en dominio de tiempo discreto.

Donde:

P: Acción proporcional del controlador PID (Kp).

I: Acción integral del controlador PID (Ki).

D: Acción derivativa del controlador PID (Kd).

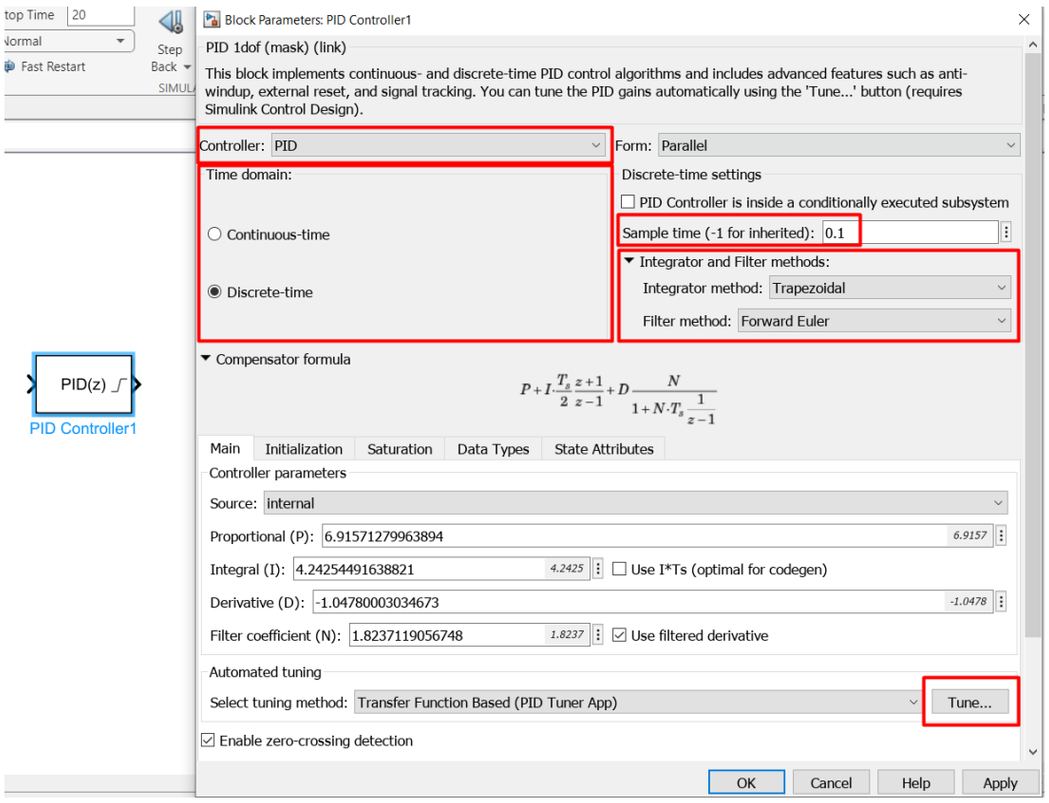
Ts: Tiempo de muestreo del sistema.

N: Coeficiente de filtro de primer orden para la acción derivativa.

Z: Dominio Z, debido a la transformada Z.

Para obtener lo mostrado en la figura anterior, se debe configurar el bloque PID de Simulink con los parámetros, tal y como se muestran en la Figura 2-28.

El coeficiente N es sintonizado por el mismo Matlab, al igual que las acciones Kp, Ki y Kd del controlador. Mientras de Ts es asignado manualmente y depende netamente del tiempo de muestreo en la que el microcontrolador o sistema en tiempo discreto adquiere los datos del sensor.

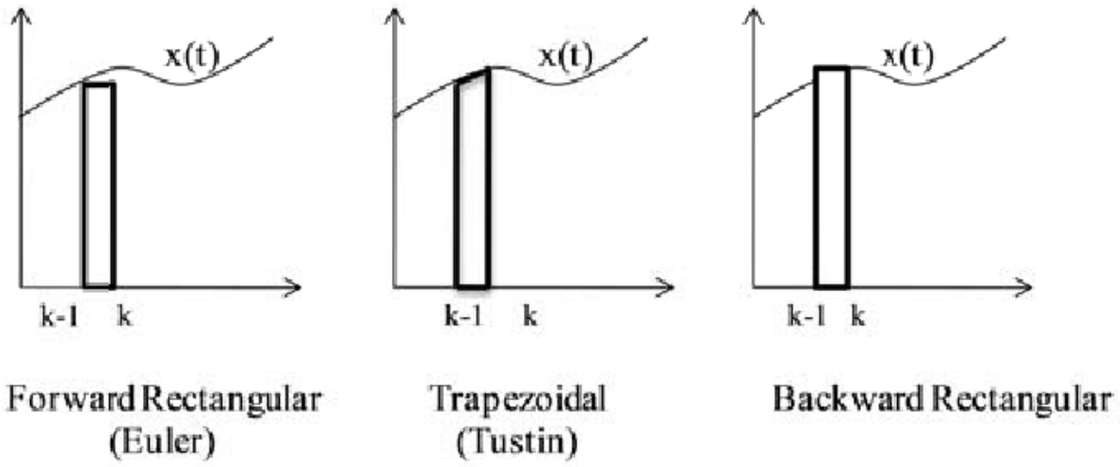


Fuente: Elaboración propia.

Figura 2-28 Configuración de bloque PID.

Los parámetros resaltados en la figura anterior son:

- **Controlador** (Controller): Permite establecer el tipo de controlador. En este caso se utiliza un controlador PID.
- **Dominio de tiempo** (Time Domain): Permite seleccionar entre el tiempo continuo y el tiempo discreto. Para el uso en microcontrolador, se utiliza el tiempo discreto.
- **Tiempo de muestreo** (Sample Time): Establece el tiempo de muestreo en 0.1 segundos. Este valor dependerá netamente del tiempo de muestreo que el microcontrolador mida los datos del sensor.
- **Método del integrador y filtro** (Integrador and Filter methods): Establece el tipo de aproximación del dominio S a la variable compleja Z. Para el integrador del controlador actual se utiliza el método **Trapezoidal** (Tustin) ya que es la mejor aproximación indicada por Matlab, porque produce la coincidencia más cercana a la señal discretizada, como se puede observar en la Figura 2-29. Por otra parte, el filtro evita que la parte Derivativa adopte valores extremadamente altos debido a la variación del error y el tiempo de muestreo. El filtro utilizado en particular implementa el método **Forward Euler,** para la simplificación de los cálculos ya que es un filtro pasa bajos de primer orden.



Fuente: https://www.semanticscholar.org/paper/Influence-of-the-discretization-method-on-the-of-Comanescu/1e444e79b36e32dbdb41b6a03bdb7d56eeb05e45.

Figura 2-29 Diferencias entre los tipos de aproximación discreta.

En el trabajo de título referenciado anteriormente, el autor desarrolla la función de transferencia de la Figura 2-27 para obtener la representación de muestras con retardos necesaria para el funcionamiento en microcontroladores o cualquier sistema discreto, obteniendo la siguiente expresión:

$$u[n] = K_p e[n] + \frac{K_i T}{2}(e[n] + e[n-1]) + i[n-1] + NK_d(e[n] - e[n-1]) + (1 - NT)d[n-1]$$

Donde:

U[n]: Salida del controlador PID.

Kp: Acción proporcional del controlador PID.

Ki: Acción integral del controlador PID.

Kd: Acción derivativa del controlador PID.

i[n-1]: Parte integral de la muestra anterior.

d[n-1]: Parte derivativa de la muestra anterior.

T: Tiempo de muestreo del sistema.

N: Coeficiente de filtro de primer orden para la acción derivativa.

e: Error en la planta.

También, la ecuación anterior puede ser divida en las tres partes que la componen: La parte proporcional, parte integral y la parte proporcional:

La **Parte proporcional** es:

$$p[n] = K_p e[n]$$

La **Parte integral** es:

$$i[n] = \frac{K_i T}{2}(e[n] + e[n-1]) + i[n-1]$$

Y la **Parte derivativa** es:

$$d[n] = NK_d(e[n] - e[n-1]) + (1 - NT)d[n-1]$$

Se debe resaltar que los valores X[n] corresponden a valores de muestras actuales y los valores X[n-1] corresponden a valores de muestras anterior. Esto en microcontroladores suele ser manejado de distintas formas, entre ellas las más simples son: Utilizar dos variables distintas o utilizar un arreglo de dos datos de tamaño.

Para el caso de las dos variables distintas, es normal utilizar la notación que se ve en la Figura 2-30, donde last_data corresponde a X[n-1] y current_data corresponde a X[n].

```
float last_data = 0.0;
float current_data = 0.0;
```

Fuente: Elaboración propia.

Figura 2-30 Inicialización de dos variables para almacenar valor anterior y actual.

Si se quisiera almacenar más datos anteriores para hacer otros cálculos, no sería optimo utilizar variables independientes. Para estos casos, se utiliza un arreglo de tamaño N según la necesidad del cálculo, como se muestra en la Figura 2-31. Esto permite almacenar la variable anterior en data[0] y la variable actual en data[1].

```
float data[2];
```

Fuente: Elaboración propia.

Figura 2-31 Declaración de un arreglo con dos espacios tipo float.

Cada quien tiene distintas formas de programar y motivos para su código, por lo que será normal ver las distintas formas para realizar o cumplir los objetivos. Puede ser que la primera opción sea más descriptiva para algunos que la segunda, debido a que se le da nombres claros y concisos a las variables, mientras que la segunda opción depende netamente del orden del programador.

2.7.3 Implementación del controlador en lenguaje C

Con la obtención de la representación de muestras con retardo del controlador PID, se implementa el algoritmo básico mostrado en la Figura 2-32, para el funcionamiento del controlador en cualquier sistema embebido.

El código de ejemplo no tiene ningún parámetro PID sintonizado por lo que, por defecto están iniciados en 1.

```c
void main() {
  // ? Parámetros del controlador
  float T = 0.1;  // Tiempo de muestreo en segundos
  float Kp = 1;   // Ganancia proporcional
  float Ki = 1;   // Ganancia integral
  float Kd = 1;   // Ganancia derivativa
  float N = 1;    // Coeficiente del filtro derivativo

  // ? Variables para almacenar muestras anteriores
  float error_anterior = 0;       // Almacena error anterior
  float integral_anterior = 0;    // Almacena parte integral anterior
  float derivativo_anterior = 0;  // Almacena parte derivativa anterior

  // Referencia de ejemplo, 50 mm.
  float referencia = 50;

  while(1) {
    // Calcula el error E(s). La función leerSensor() es un ejemplo de un sensor analógico o digital
    // que mide la señal de salida Y(s) de la planta.
    float error = referencia - leerSensor();

    // Calcula la parte o acción proporcional.
    float proporcional = Kp * error;

    // Calcula la parte o acción integral.
    float integral = Ki * 0.5 * T * (error + error_anterior) + integral_anterior;

    // Calcula la parte o acción derivativa.
    float derivativo = Kd * N * (error - error_anterior) + (1.0 - N * T) * derivativo_anterior;

    // Calcula la salida del controlador U(s).
    float output = proporcional + integral + derivativo;

    // Aplica la actuación a la planta, sea una salida análoga o digital.
    escribirSalida(output);

    // Guarda muestras anteriores para la siguiente iteración.
    integral_anterior = integral;
    derivativo_anterior = derivativo;
    error_anterior = error;

    // Aplica retraso de 100 ms, para la siguiente iteración. Este valor de retraso debe ser el
    // mismo con el que se sintoniza el controlador PID.
    delay_ms(100);
  }
}
```

Fuente: Elaboración propia, utilizando Visual Code Studio.

Figura 2-32 Implementación de controlador básico en C.

El funcionamiento del controlador puede ser mejorado para incluir la “Saturación de la salida” y una característica llamada “Anti Windup”.

La **Saturación** mostrada en la Figura 2-33, permite limitar la salida al actuador, ya que, en un sistema embebido, los microcontroladores o dispositivos en general cuentan con limitaciones físicas, por ejemplo, los DAC del ESP32 pueden ir desde los valores 0 a 3.3 VDC.

El **Anti Windup** mostrado en la Figura 2-34, limita la sobrecarga de la parte integral, ya que ésta acumula el error en cada muestreo. Si el error es persistente, provoca que la parte integral siga aumentando o disminuyendo. Si el error comienza a variar, habrá que esperar a que la parte integral se descargue para comenzar a ver un funcionamiento adecuado del controlador. Existen distintos métodos para el Anti Windup, entre los más conocidos el “Back Calculation” y “Clamping”. Debido a la facilidad de implementación y calculo para el microcontrolador, se utiliza el Anti Windup por Clamping.

Es importante incluir esta saturación y el Anti Windup para generar un correcto control del sistema, ya que puede ocurrir que el controlador intente generar control con valores que se escapan completamente de la capacidad del hardware, provocando que la planta no se comporte como fue diseñado inicialmente.

```
// Saturación
// Si el valor de salida es mayor al valor máximo, la salida es seteada al valor_máximo.
// Si el valor de salida es menor al valor mínimo, la salida es seteada al valor mínimo.
if(output > valor_maximo) output = valor_maximo;
else if(output < valor_minimo) output = valor_minimo;
```

Fuente: Elaboración propia.

Figura 2-33 Saturación de la salida.

```
// Antiwindup
// Si el valor de la parte integral es mayor al valor máximo de la salida, la parte integral
// es seteada al valor máximo de salida.
// Si el valor de la parte integral es menor al valor mínimo de la salida, la parte integral
// es seteada al valor minimo de la salida.
if(integral > valor_maximo) integral = valor_maximo;
else if(integral < valor_minimo) integral = valor_minimo;
```

Fuente: Elaboración propia.

Figura 2-34 Anti Windup por Clamping de la parte integral.

Finalmente, el algoritmo completo del controlador, queda como se muestra en la Figura 2-35.

```c
void main() {
  // ? Parámetros del controlador
  float T = 0.1;  // Tiempo de muestreo en segundos
  float Kp = 1;   // Ganancia proporcional
  float Ki = 1;   // Ganancia integral
  float Kd = 1;   // Ganancia derivativa
  float N = 1;    // Coeficiente del filtro derivativo

  // ? Variables para almacenar muestras anteriores
  float error_anterior = 0;       // Almacena error anterior
  float integral_anterior = 0;    // Almacena parte integral anterior
  float derivativo_anterior = 0;  // Almacena parte derivativa anterior

  // ? Saturación
  float valor_maximo = 100;
  float valor_minimo = 0;

  // Referencia de ejemplo, 50 mm.
  float referencia = 50;

  while(1) {
    // Calcula el error E(s). La función leerSensor() es un ejemplo de un sensor analógico o digital
    // que mide la señal de salida Y(s) de la planta.
    float error = referencia - leerSensor();

    // Calcula la parte o acción proporcional.
    float proporcional = Kp * error;

    // Calcula la parte o acción integral.
    float integral = Ki * 0.5 * T * (error + error_anterior) + integral_anterior;

    // Antiwindup
    // Si el valor de la parte integral es mayor al valor máximo de la salida, la parte integral
    // es seteada al valor máximo de salida.
    // Si el valor de la parte integral es menor al valor mínimo de la salida, la parte integral
    // es seteada al valor minimo de la salida.
    if(integral > valor_maximo) integral = valor_maximo;
    else if(integral < valor_minimo) integral = valor_minimo;

    // Calcula la parte o acción derivativa.
    float derivativo = Kd * N * (error - error_anterior) + (1.0 - N * T) * derivativo_anterior;

    // Calcula la salida del controlador U(s).
    float output = proporcional + integral + derivativo;

    // Saturación
    // Si el valor de salida es mayor al valor máximo, la salida es seteada al valor_máximo.
    // Si el valor de salida es menor al valor mínimo, la salida es seteada al valor mínimo.
    if(output > valor_maximo) output = valor_maximo;
    else if(output < valor_minimo) output = valor_minimo;

    // Aplica la actuación a la planta, sea una salida análoga o digital.
    escribirSalida(output);

    // Guarda muestras anteriores para la siguiente iteración.
    integral_anterior = integral;
    derivativo_anterior = derivativo;
    error_anterior = error;

    // Aplica retraso de 100 ms, para la siguiente iteración. Este valor de retraso debe ser el
    // mismo con el que se sintoniza el controlador PID.
    delay_ms(100);
  }
}
```

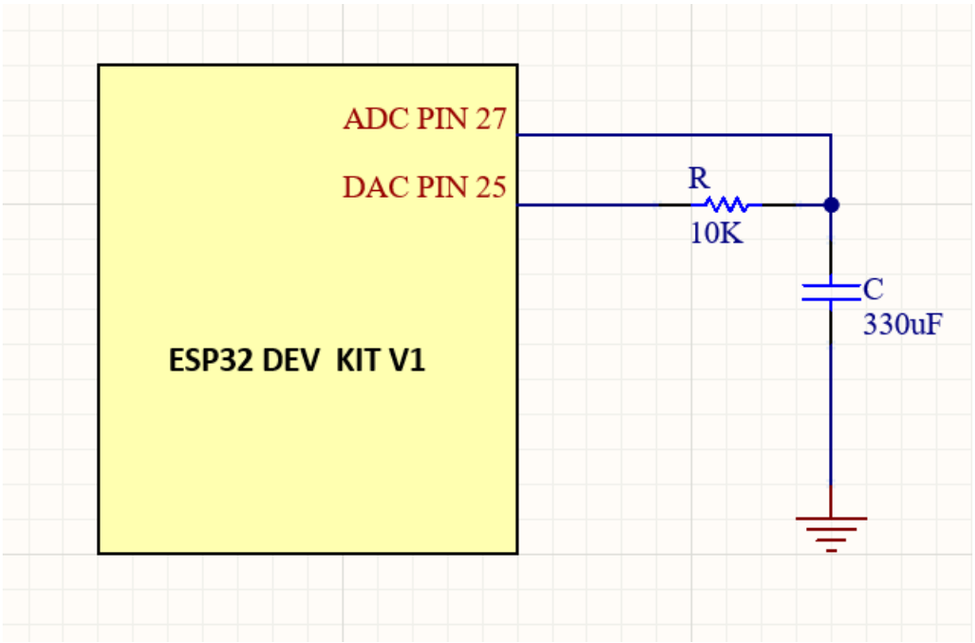
Fuente: Elaboración propia.

Figura 2-35 Implementación de controlador en C.

2.7.4 Validación del controlador en ESP32 DEV KIT V1

Para validar el controlador se utiliza un circuito RC como planta a controlar, por la rapidez de implementación en hardware y en firmware. Cabe destacar que en la presente sección no se pretende ahondar en cómo obtener la función de transferencia de un circuito RC, por lo que solo se procederá a la utilización de dicha función.

El circuito propuesto es el mostrado en la Figura 2-36Figura 2-36 Circuito RC implementado., donde se ocupará el DAC del pin 25 para generar el control de la planta y el pin 27 como ADC para cerrar el lazo y leer así el parámetro de salida de la planta.



Fuente: Elaboración propia.

Figura 2-36 Circuito RC implementado.

La función de transferencia de un circuito RC está dada por:

$$\frac{Y(S)}{U(S)} = \frac{1}{\tau S + 1}$$

Donde:

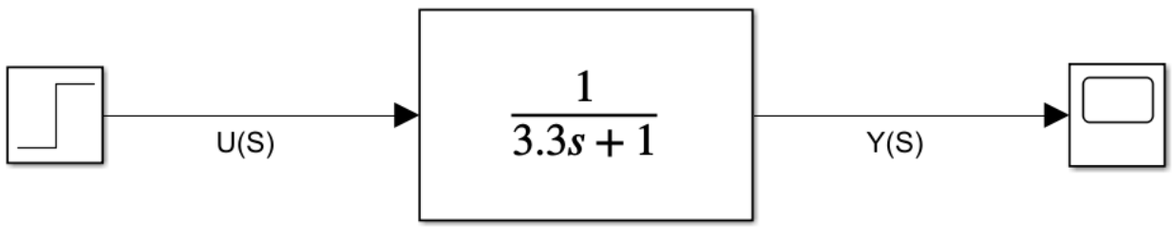
$$\tau = RC$$

$$\tau = 10K\Omega * 330uF$$

$$\tau = 3.3\ s$$

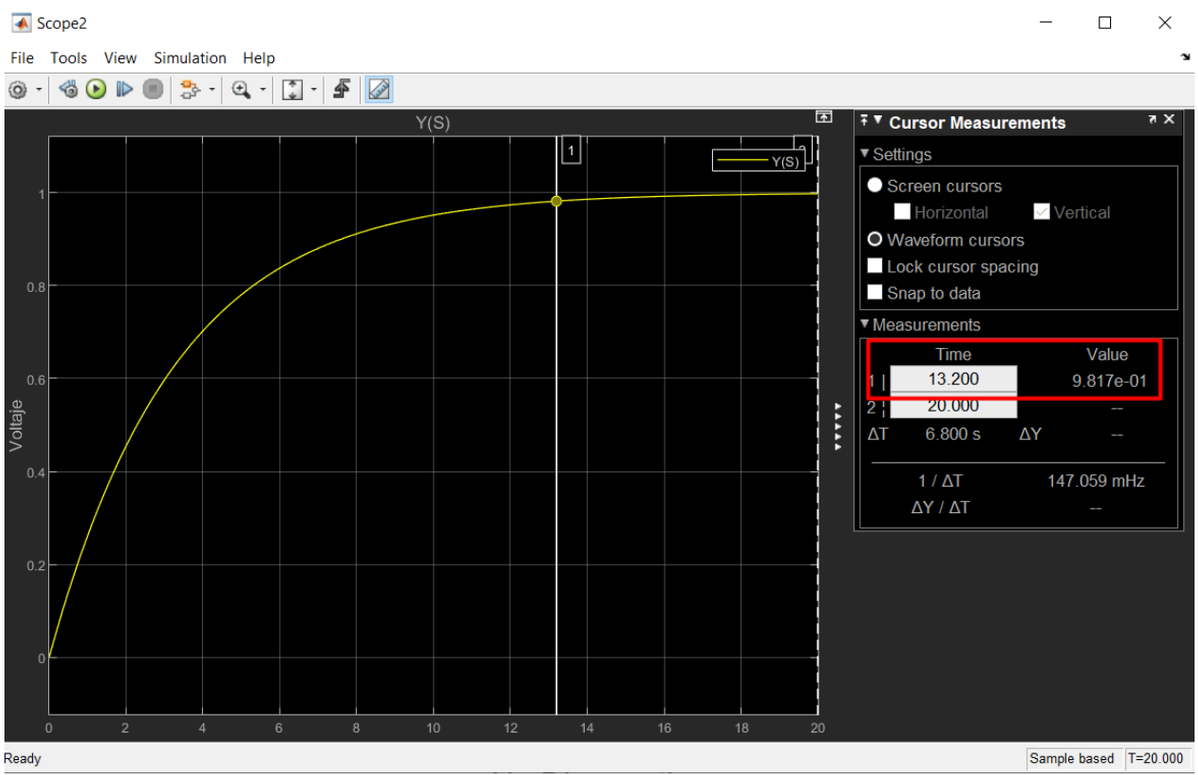
Por lo que el capacitor teóricamente deberá llegar al 98% de su carga a los 13.2 segundos (4 tau).

Si se agrega la función de transferencia de la planta RC a Simulink de Matlab y se aplica un escalón de 1 VDC, como se observa en la Figura 2-37. En la respuesta de la planta sin controlador (a lazo abierto), mostrada en la Figura 2-38, se puede observar que a los 13.2 segundos la respuesta del sistema es de 0.98 VDC.



Fuente: Elaboración propia.

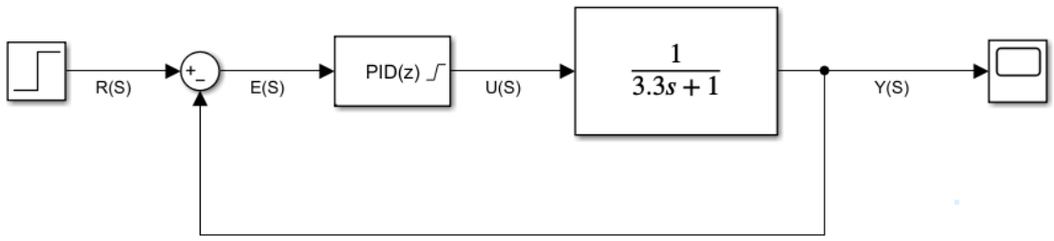
Figura 2-37 Planta de circuito RC.



Fuente: Elaboración propia.

Figura 2-38 Respuesta de la planta RC a lazo abierto.

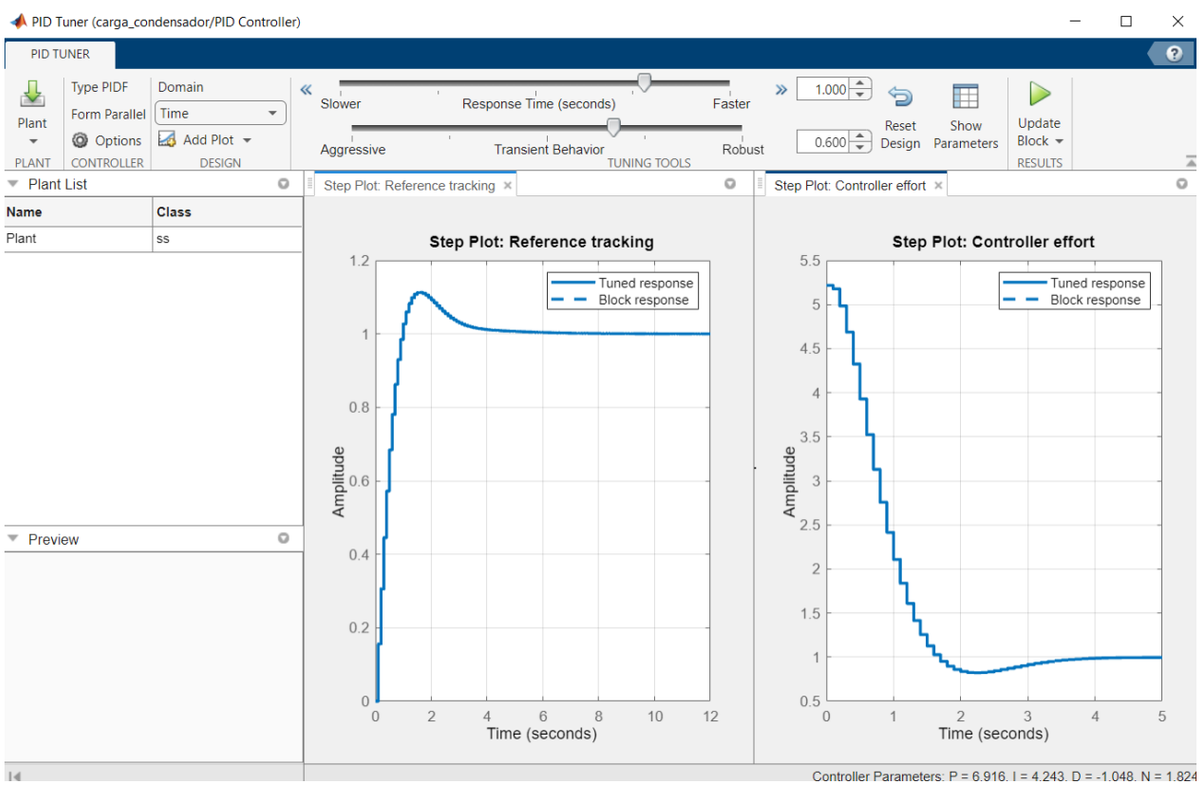
Al agregar el controlador PID y cerrar el lazo, el diagrama en Simulink queda como se muestra en la Figura 2-39.



Fuente: Elaboración propia.

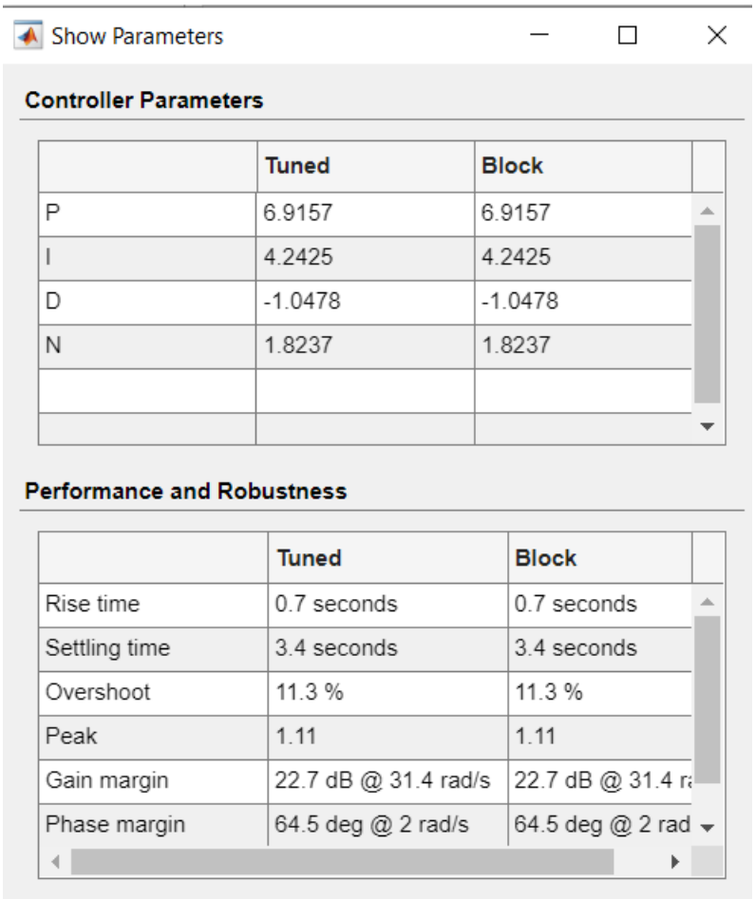
Figura 2-39 Planta RC con controlador PID y lazo cerrado.

La herramienta **PID Tuner** entrega valores recomendados por defecto, valores óptimos que pueden ser o no ocupados para el control de la planta. En este caso, se usará un tiempo de respuesta de 1 segundo, por lo que debería tardar 1 segundo en alcanzar el voltaje seteado de 1 VDC. La respuesta de la planta y el esfuerzo del controlador se observan en la Figura 2-40, mientras que los parámetros sintonizados se observan en la Figura 2-41. Estas figuras indican que la salida de la planta hay un **overshoot** del 11.3% y que el **rise time** será de 0.7 segundos.



Fuente: Elaboración propia.

Figura 2-40 Respuesta de la salida y esfuerzo del controlador a 1 segundo de respuesta.



Fuente: Elaboración propia.

Figura 2-41 Parámetros de sintonización PID.

El firmware utilizado para la validación del controlador se muestra en las Figura 2-42, Figura 2-43, Figura 2-44 y Figura 2-45. En la Figura 2-42 se crean las variables y objetos para el funcionamiento del controlador PID. En la Figura 2-43 se inicializan los valores del controlador PID dentro de la función setup() y en la función loop() se realiza la medición cada 0.1 segundos (T) si es que el la bandera booleana enable_controller está activada. En la Figura 2-44 se observa la función readConsole(), que permite escribir comandos para habilitar/deshabilitar el controlador PID, establecer el valor de referencia y establecer un valor inicial para el DAC, mediante el terminal serial. Por último, en la Figura 2-45 se puede observar el controlador implementado con la función PIDController().

```cpp
#include <Arduino.h>
#include <Wire.h>

// ? Pines del ESP32
const uint8_t PIN_DAC = 25;
const uint8_t PIN_ADC = 27;


// * Máximo y mínimo
// ? Carga condensador
const float MAX_OUT = 3.3;  // Valor máximo del DAC
const float MIN_OUT = 0.0;  // Valor mínimo del DAC

// ? Flags control
// Bandera áara activar, desactivar el controlador
bool enable_controller = false;

// ? Setpoint
float setpoint = 1;

// ? Periodo de muestreo
float T    = 0.1;

// * Variables para controlador PID
float Kp   = 6.9157;
float Ki   = 4.2425;
float Kd   = -1.0478;
float N    = 1.8237;


struct VariablesPID {
  float last_error;           // Error anterior
  float last_integral;        // Valor integral anterior
  float last_derivative;      // Valor derivativo anterior

  float error;                // Valor actual del error
  float proportional;         // Valor actual de la parte proporcional
  float integral;             // Valor actual de la parte integral
  float derivative;           // Valor actual de la parte derviativa
  float output;               // Valor actual de la salida
} pid;

uint32_t last_time = 0;

//* Prototipo de funciones
// Controlador PID
float PIDController(float in_value);
// Lectura del terminal serial para realizar cambio de setpoint,
// Habilitar, deshabilitar el controlador, entre otros.
void readConsole();
// --------------------------
```

Fuente: Elaboración propia.

Figura 2-42 Creación de variables y parámetros.

```
// -------------------------

void setup() {
  // Inicia terminal serial
  Serial.begin(115200);

  // Inicia estructura con todsos los su valores en 0
  pid = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

  // Inicia la salida del DAC en 0
  dacWrite(PIN_DAC, 0);
}

void loop() {
  readConsole();

  // ? Carga condensador
  if(enable_controller) {
    if(millis() - last_time >= 100) {
      last_time = millis();
      float adc_value = analogRead(PIN_ADC);
      adc_value = adc_value * 3.3 / 4095; // Convierte 2^12 a voltaje

      float dac_value = PIDController(adc_value);
      dac_value = dac_value * 255 / 3.3; // Convierte voltaje a 2^8

      dacWrite(PIN_DAC, dac_value);
    }
  }
}
```

Fuente: Elaboración propia.

Figura 2-43 Funciones setup() y loop().

```
void readConsole() {

  if (Serial.available()) {
    char c = Serial.read();

    switch(c) {
      case 'c': // habilita / deshabilita PID
      {
        enable_controller = !enable_controller;
        // Serial.printf("Controlador PID %s\n", enable_controller ? "habilitado" : "deshabilitado");
      } break;

      case 's': // Setea setpoint
      {
        setpoint = Serial.readStringUntil(',').toFloat();
        // Serial.printf("Setpoint puesto a %.2f\n", setpoint);
      } break;

      case 'd': // Setea valor en dac
      {
        if(enable_controller) {
          // Serial.println("Controlador habilitado! no puedes modificar esto");
          break;
        }

        float value = Serial.readStringUntil(',').toFloat();
        float dac_value = value * 255 / 3.3;

        dacWrite(PIN_DAC, (uint8_t) dac_value);
      } break;
    }
  }
}
```

Fuente: Elaboración propia.

Figura 2-44 Función readConsole().

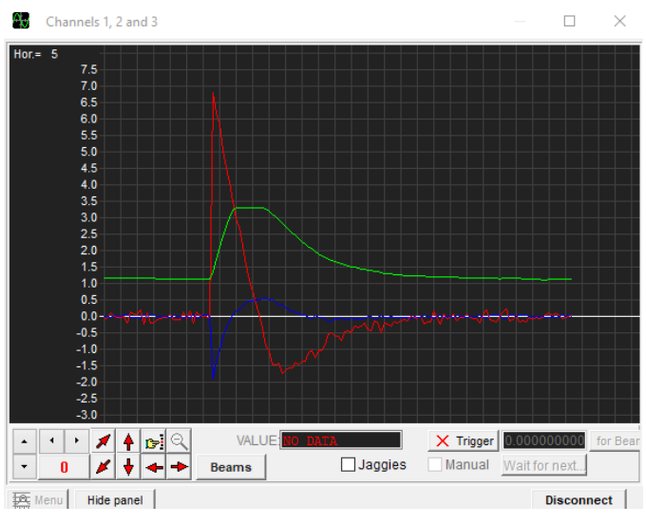
```
// PIN void PIDController()
float PIDController(float in_value) {
    pid.error = setpoint - in_value; // Calcula el error
    pid.proportional = Kp * pid.error;      // Caclula la parte proporcional
    // Calcula la parte integral si no está saturada la salida.
    pid.integral = Ki * 0.5 * T * (pid.error + pid.last_error) + pid.last_integral;
    // Antiwindup
    if(pid.integral > MAX_OUT) pid.integral = MAX_OUT;
    else if(pid.integral < MIN_OUT) pid.integral = MIN_OUT;
    // Calcula la parte derivativa del controlador
    pid.derivative = N * Kd * (pid.error - pid.last_error) + (1.0 - N * T) * pid.last_derivative;
    // Suma todas las partes para obtener la salida
    pid.output = pid.proportional + pid.integral + pid.derivative;

    if(pid.output > MAX_OUT) pid.output = MAX_OUT;
    else if(pid.output < MIN_OUT) pid.output = MIN_OUT;

    Serial.printf("%.3f,%.3f,%.3f,%.3f,%.3f,%.3f\r" , pid.proportional, pid.integral
                                                     , pid.derivative, in_value
                                                     , pid.output, pid.error);

    pid.last_integral   = pid.integral;
    pid.last_derivative = pid.derivative;
    pid.last_error      = pid.error;

    return pid.output;
}
/* ------------------------------------------------------------------------------------------ */
```

Fuente: Elaboración propia.

Figura 2-45 Función PIDController().

En la función PIDController() se imprime al terminal serial con la función printf() del objeto Serial, los valores de la parte proporcional, integral, derivativa, el valor de entrada (ADC), el valor de salida (DAC) y el error, para ser visualizados con un software open source llamado “Serial Oscilloscope”.

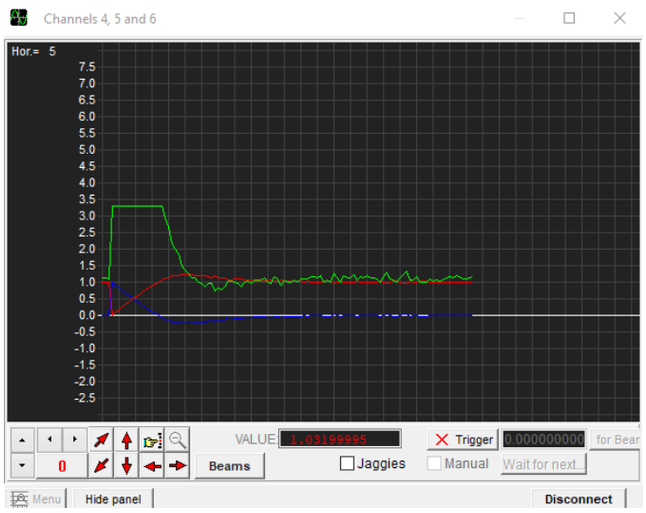
Las respuestas de las partes proporcional, integral y derivativa se observan en la Figura 2-46, donde la línea roja corresponde a la parte proporcional, la verde a la parte integral y la azul a la parte derivativa.



Fuente: Elaboración propia.

Figura 2-46 Respuesta planta RC parte PID.

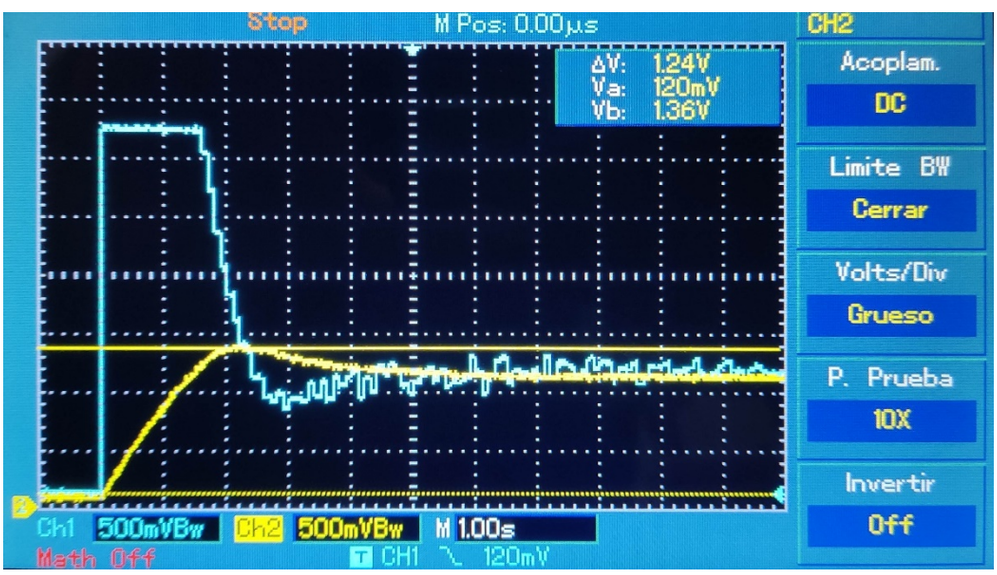
La respuesta de la planta, la salida del actuador y el error, se observan en la Figura 2-47. La línea de color rojo muestra la respuesta de la planta Y(S), mientras que en verde se observa el valor de actuación U(S) generado por el controlador PID y en azul se muestra el error E(S).



Fuente: Elaboración propia.

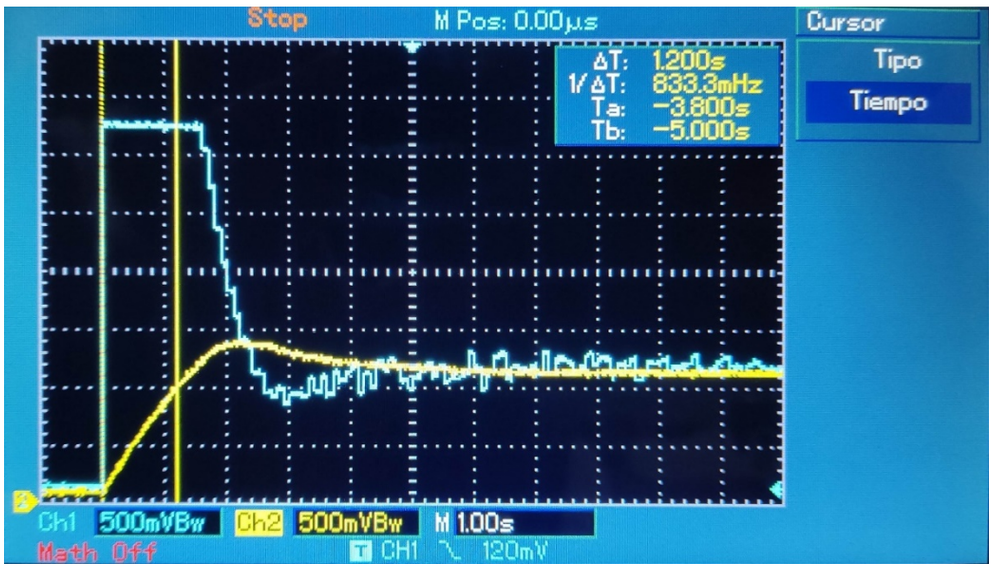
Figura 2-47 Respuesta de la planta RC, Salida de la planta, actuación y error.

El software Serial Oscilloscope se utiliza para visualizar los valores del controlador, pero no se puede realizar un mayor análisis, ya que no tiene cursores para revisar tiempos o valores, por lo que se realizan mediciones con un osciloscopio físico mostradas en la Figura 2-48, Figura 2-49 y Figura 2-50. La señal azul corresponde a la señal de salida del controlador (DAC) y la señal amarilla corresponde a la salida de la planta RC.



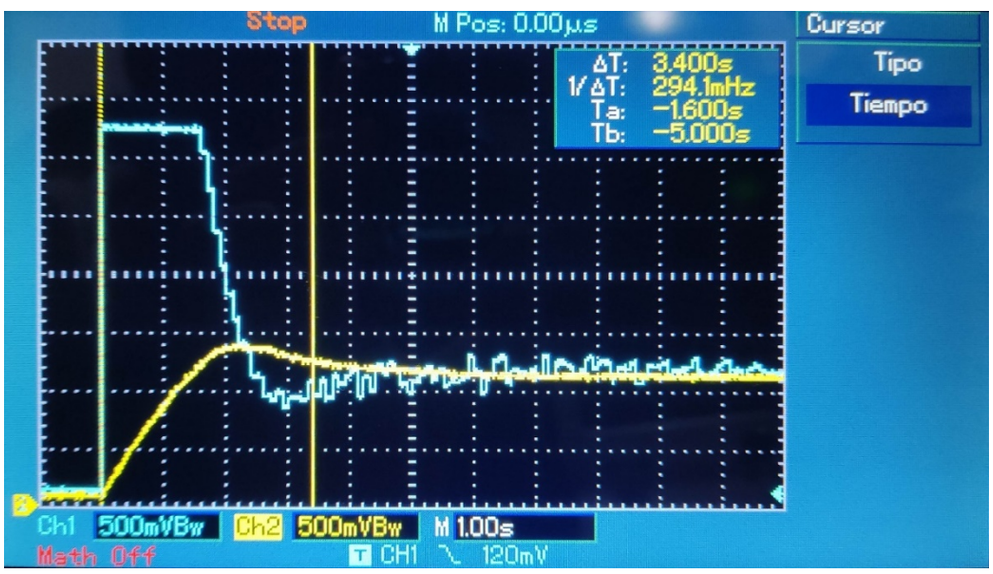
Fuente: Elaboración propia.

Figura 2-48 Respuesta de la salida en osciloscopio, cursores en voltaje.



Fuente: Elaboración propia.

Figura 2-49 Respuesta de la salida en osciloscopio, cursores en tiempo para medir tiempo de respuesta.



Fuente: Elaboración propia.

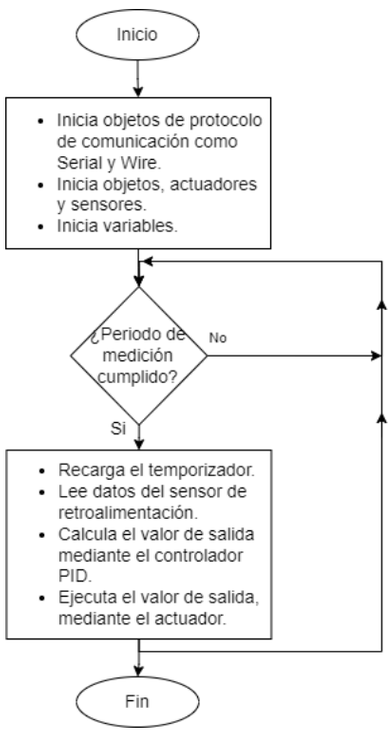
Figura 2-50 Respuesta de la salida en osciloscopio, cursores en tiempo para medir tiempo de asentamiento.

En la Figura 2-48 se puede observar que el overshoot alcanzado es aproximadamente de 24%. En la Figura 2-49 el tiempo de respuesta es de 1.2 segundos y en la Figura 2-50 el tiempo de asentamiento es de 3.4 segundos. Estos valores están cercanos a los esperados teóricamente y se concluye que son válidos, debido a que se considera que las diferencias prácticas que existen están dadas en gran medida por la no linealidad de la lectura del puerto análogo y la generación de voltaje con poca resolución (8 bits) del microcontrolador ESP32 utilizado en la

placa de desarrollo, además la tolerancia de los componentes utilizados (±20%), la saturación del sistema, entre otros.

2.8 FIRMWARE

Para cada maqueta se realizan firmwares distintos, debido a las diferencias de hardware entre ellas, como: Lecturas de puertos análogos, manipulación de pantalla y el tipo de actuador. Aun así, la base común y esencial para cada maqueta es el controlador PID, por lo que en la Figura 2-51 se muestra un diagrama general que simplifica ambos firmwares.



Fuente: Elaboración propia.

Figura 2-51 Diagrama funcionamiento básico maqueta.

El sistema inicia creando los objetos, constantes y variables, posteriormente inicializa todos los periféricos que se utilizan como el puerto Serial, la comunicación I2C denominada en la librería como Wire, entre otros.

Posteriormente se crea una variable para almacenar un temporizador interno y se pregunta en cada ciclo del bucle si el tiempo transcurrido es mayor o igual al tiempo de muestreo asignado. Si el tiempo ha transcurrido, se recarga el temporizador para permitir una nueva lectura, realiza la lectura del sensor y calcula el dato de salida mediante el controlador PID programado y sintonizado.

2.8.1 Firmware básico “Demostración de control de un Servomotor”

El firmware para la maqueta “Demostración de control de un Servomotor” consiste en poder ajustar de forma manual los parámetros PID del controlador y punto de referencia, mediante el uso de potenciómetros. Para poder visualizar los valores configurados, se implementa una pantalla I2C, la cual mostrará el valor de Kp, Ki, Kd, la referencia y posición actual. También, mediante consola serial, se puede ajustar los valores máximos para Kp, Ki y Kd, para la conversión de los potenciómetros.

El firmware implementa las librerías mostradas en la Figura 2-52.

```
//* Librerías
#include <Arduino.h>
#include <Wire.h>

#include "U8g2lib.h"
#include "ADS1X15.h"
```

Fuente: Elaboración propia.

Figura 2-52 Librerías usadas "Demostración de control de un Servomotor".

- **Arduino.h:** Corresponde a la abstracción de todos los recursos que ofrece el bootloader Arduino, por ejemplo, la utilización de funciones como pinMode(), entre otros.
- **Wire.h:** Permite la comunicación con el sensor ADS1115 y la pantalla OLED.
- **U8g2lib.h:** Permite el uso de la pantalla OLED.
- **ADS1X15.h:** Permite el uso del módulo ADC con el IC ADS1115.

Posteriormente se agrega la configuración de pines utilizada en la maqueta y las variables a leer por cada potenciómetro, como se muestra en la Figura 2-53.

```
//* Enum para identificar los pines usados en placa
enum DemoPins : uint8_t {
  I2C_SDA   = 21,
  I2C_SCL   = 22,
  ANALOG_1  = 33,
  MOTOR_A_1A  = 18,
  MOTOR_A_1B  = 19,
  ADC_ALERT   = 25
};

//* Variable / potenciómetro leído
enum Variable : uint8_t {
  PROPORTIONAL,
  INTEGRAL,
  DERIVATIVE,
  SETPOINT,
  ROTOR,
  MAX
};
```

Fuente: Elaboración propia.

Figura 2-53 Declaración de pines.

En las Figura 2-54 y Figura 2-55, se crean los objetos y variables a utilizar.

```
//* Creación de objetos
U8G2_SH1106_128X64_NONAME_F_HW_I2C display (U8G2_R0, U8X8_PIN_NONE);
ADS1115 adc(0x48);

//* Variables
struct AnalogInputs {
  uint16_t raw;
  uint16_t filtered;
  float scaled;
};

AnalogInputs analogs_inputs[MAX];
uint8_t ads_channel = 0;

volatile bool ready = false;

//* Variables del controlador
bool enable_controller = false;

float period = 0.1;

float max_kp = 10.0;
float max_ki = 10.0;
float max_kd = 10.0;

float max_out = 255.0;
float min_out = -255.0;
```

Fuente: Elaboración propia.

Figura 2-54 Objetos y variables.

```
struct VariablesPID {
  float Kp;                  // Valor de sintonización proporcional
  float Ki;                  // Valor de sintonización integral
  float Kd;                  // Valor de sintonización derivativa
  float N_coef;              // Valor coeficiente filtro derivativo

  float setpoint;            // Valor del setpoint
  float error;               // Valor actual del error

  float proportional;        // Valor actual de la parte proporcional
  float integral;            // Valor actual de la parte integral
  float derivative;          // Valor actual de la parte derviativa
  float output;              // Valor actual de la salida

  float last_error;          // Error anterior
  float last_integral;       // Valor integral anterior
  float last_derivative;     // Valor derivativo anterior
} pid;
```

Fuente: Elaboración propia.

Figura 2-55 Objetos y variables, controlador PID.

En la Figura 2-56 se muestran los prototipos de funciones, necesarios para la compilación del código en Visual Code Studio.

```
//* Prototipos de funciones
void displayView();
void readAnalogsInputs();
void readConsole();
float PIDController(float in_value);
void IRAM_ATTR adcReady();
```

Fuente: Elaboración propia.

Figura 2-56 Prototipo de funciones.

En la función **setup()** se inicializan todos los módulos y pines, como se muestran en la Figura 2-57. Primero inicializa puertos seriales como I2C y UART, después inicializa la pantalla Oled, el sensor ADS1115, posteriormente se configuran los pines de salida para el control del motor y finalmente inicia una interrupción para detectar que el módulo ADS1115 haya terminado la lectura de cada canal.

```
// PIN setup()
void setup() {
  // Inicializa puertos de comunicacion
  Wire.begin(I2C_SDA, I2C_SCL);
  Serial.begin(115200);

  // Inicia y limpia pantalla I2C
  display.begin();
  display.clear();

  // Inicia y configura el chip ADS1115
  adc.begin();
  adc.setGain(1);
  adc.setDataRate(4);
  adc.setComparatorThresholdHigh(0x8000);
  adc.setComparatorThresholdLow(0x0000);
  adc.setComparatorQueConvert(0);
  adc.setMode(1);

  // Establece y "apaga" pines de control de motor como salida
  pinMode((uint8_t) DemoPins::MOTOR_A_1A, OUTPUT);
  pinMode((uint8_t) DemoPins::MOTOR_A_1B, OUTPUT);

  digitalWrite((uint8_t) DemoPins::MOTOR_A_1A, LOW);
  digitalWrite((uint8_t) DemoPins::MOTOR_A_1B, LOW);

  // Establece el pin ADC alert como entrada, con pullup interno
  // y lo configura como interrupción.
  pinMode((uint8_t) DemoPins::ADC_ALERT, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt((uint8_t) DemoPins::ADC_ALERT), adcReady, RISING);

  // Inicializa los valores del controlador PID en 0
  pid = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
          0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
  // La constante N para el filtro derivativo se ignora por lo que se setea a 1.0
  pid.N_coef = 1.0;
}
```

Fuente: Elaboración propia.

Figura 2-57 Función setup().

La función **loop()** mostrada en la Figura 2-58, como su nombre indica es un bucle infinito, en el que se ejecutan las funciones periódicamente como la función readConsole(), displayView(), readAnalogsInputs() y se ejecuta el control sobre la planta cada 100 ms.

```
// PIN loop()
void loop() {
  static uint32_t last_time = 0;
  // Lee datos del monitor serial
  readConsole();
  // Maneja la pantalla
  displayView();
  // Lee las entradas análogas
  readAnalogsInputs();

  // Lectura y actuación cada 100 ms
  if(millis() - last_time >= 100) {
    last_time = millis();
    // Como es una maqueta de ejemplo, si Ki = 0, los valores
    // de la integral se reinician.
    if(pid.Ki == 0) {
      pid.integral = 0;
      pid.last_integral = 0;
    }
    // Lee y escala la señal del potenciómetro conectado al rotor del motor
    analogs_inputs[ROTOR].raw = analogRead(ANALOG_1);
    analogs_inputs[ROTOR].scaled = analogs_inputs[ROTOR].raw * 180.0 / 3380.0;
    // Ingresa el valor leído al controlador PID
    float motor_speed = PIDController(analogs_inputs[ROTOR].scaled);
    // Dependiendo del signo de la velocidad del motor,
    // los pintes de control del motor se apagan o no
    if((int32_t) motor_speed >= 0) {
      analogWrite(MOTOR_A_1A, 0);
      analogWrite(MOTOR_A_1B, motor_speed);
    } else { // Si no, gira a la izquierda y hace positiva la velocidad
      motor_speed *= -1;
      analogWrite(MOTOR_A_1A, motor_speed);
      analogWrite(MOTOR_A_1B, 0);
    }
  }
}
```

Fuente: Elaboración propia.

Figura 2-58 Función loop().

La función **readConsole()** se muestra en la Figura 2-59. Solo permite ajustar 3 variables por el momento, el máximo valor para Kp, Ki y Kd.

```
// PIN readConsole()
void readConsole() {
  if(Serial.available()) {
    char c = Serial.read();

    switch(c) {
      case 'p': max_kp = Serial.readStringUntil(',').toFloat(); break;
      case 'i': max_ki = Serial.readStringUntil(',').toFloat(); break;
      case 'd': max_kd = Serial.readStringUntil(',').toFloat(); break;
    }
  }
}
```

Fuente: Elaboración propia.

Figura 2-59 Función readConsole().

La función **displayView()** está configurada para refrescar los datos cada 100 ms con el fin de no saturar el puerto I2C con escrituras innecesarias, como se muestra en la Figura 2-60. Básicamente la función utiliza métodos de su clase para acomodar el puntero en el pixel correspondiente y escribir lo que se desee.

```
// PIN displayView()
void displayView() {
  static uint32_t last_time = 0;
  char text_buffer[20];

  if(millis() - last_time >= 100) {
    last_time = millis();
    display.setFont(u8g2_font_6x12_mf);
    display.enableUTF8Print();
    display.setFontMode(0);
    display.clearBuffer();
    sprintf(text_buffer, "P: %.1f", analogs_inputs[0].scaled);
    display.drawStr(0, 12, text_buffer);
    sprintf(text_buffer, "I: %.1f", analogs_inputs[1].scaled);
    display.drawStr(0, 24, text_buffer);
    sprintf(text_buffer, "D: %.1f", analogs_inputs[2].scaled);
    display.drawStr(0, 36, text_buffer);
    display.drawStr(0, 48, "Setpoint");
    sprintf(text_buffer, "%.0f°", analogs_inputs[3].scaled);
    uint8_t x = (display.getStrWidth("Setpoint") - display.getStrWidth(text_buffer)) / 2;
    display.drawUTF8(x, 60, text_buffer);

    display.drawStr(64, 12, "Position");
    display.setFont(u8g2_font_10x20_mf);
    sprintf(text_buffer, "%.0f°", analogs_inputs[4].scaled);
    display.drawUTF8(64, 30, text_buffer);

    display.sendBuffer();
  }
}
```

Fuente: Elaboración propia.

Figura 2-60 Función displayView().

La función readADSInputs() se muestra en la Figura 2-61. Permite leer los valores análogos de los cuatro canales del circuito integrado ADS1115.

El funcionamiento del integrado es particular, ya que por protocolo I2C se solicita la lectura del canal ADC que se desee leer, luego se debe esperar que el IC avise mediante el pin de interrupción, que la lectura ha terminado y está lista para poder ser consumida. Esta acción se repite para los cuatro canales.

El valor leído se escala de 0 - 26363 a 0 – 100, siendo que el módulo ADS1115 cuenta con conversores ADC de 16 bits. Esto se debe a que, si bien la conversión es de 16 bits, teóricamente se lograrían lecturas hasta 65535, pero realmente se consideran 15 bits ya que el último bit es para el signo del valor, quedando así la lectura hasta 32768.

El valor de máximo 26363 se explica porque el conversor internamente tiene un voltaje de referencia, el cual puede ser programado a 0.256, 0.512, 1.024, 2.048, 4.096 y 6.144 VDC. Para este firmware se utiliza el valor de 2.048 VDC, ya que es el valor próximo al voltaje de alimentación de +3.3 VDC del microcontrolador que utiliza la placa de desarrollo, así se logra utilizar la escala completa del ADC.

```
// PIN readAnalogsInputs()
void readAnalogsInputs() {
  static uint32_t last_time = 0;
  static bool firts = true;

  if(millis() - last_time >= 100) {
    if(firts) {
      ads_channel = 0;
      adc.requestADC(ads_channel);
      firts = false;
    }

    if(ready) {
      int16_t temp = adc.getValue();
      if(temp < 0) temp = 0;
      analogs_inputs[ads_channel].raw = temp;

      switch(ads_channel) {
        case PROPORTIONAL:
          analogs_inputs[PROPORTIONAL].scaled = temp * max_kp / 26363.0; // P
          break;
        case INTEGRAL:
          analogs_inputs[INTEGRAL].scaled = temp * max_ki / 26363.0;    // I
          break;
        case DERIVATIVE:
          analogs_inputs[DERIVATIVE].scaled = temp * max_kd / 26363.0;  // D
          break;
        case SETPOINT:
          analogs_inputs[SETPOINT].scaled = temp * 180.0 / 26363.0; // Setpoint
          break;
      }

      ads_channel++;

      if(ads_channel > 3) {
        last_time = millis();
        ads_channel = 0;
        pid.Kp        = analogs_inputs[PROPORTIONAL].scaled;
        pid.Ki        = analogs_inputs[INTEGRAL].scaled;
        pid.Kd        = analogs_inputs[DERIVATIVE].scaled;
        pid.setpoint  = analogs_inputs[SETPOINT].scaled;
      }

      adc.requestADC(ads_channel);
      ready = false;
    }
  }
}
```

Fuente: Elaboración propia.

Figura 2-61 Función readAnalogsInputs().

La función PIDController() se explica en detalle dentro del mismo código, mediante comentarios, como se observa la Figura 2-62.

```
// PIN  PIDController()
float PIDController(float in_value) {
  pid.error = pid.setpoint - in_value; // Calcula el error
  pid.proportional = pid.Kp * pid.error;      // Caclula la parte proporcional

  // Calcula la parte integral si no está saturada la salida.
  pid.integral = pid.Ki * 0.5 * period * (pid.error + pid.last_error) + pid.last_integral;
  // Anti Windup
  if(pid.integral > max_out) pid.integral = max_out;
  else if(pid.integral < min_out) pid.integral = min_out;

  // Calcula la parte derivativa del controlador
  pid.derivative =  pid.N_coef * pid.Kd * (pid.error - pid.last_error)
                  + (1.0 - pid.N_coef * period) * pid.last_derivative;

  // Suma todas las partes para obtener la salida
  pid.output = pid.proportional + pid.integral + pid.derivative;

  if(pid.output > max_out) pid.output = max_out;
  else if(pid.output < min_out) pid.output = min_out;

  Serial.printf("s: %.2f - p: %.2f - i: %.2f - d: %.2f - o: %.2f\n" , pid.setpoint
                                                                    , pid.proportional
                                                                    , pid.integral
                                                                    , pid.derivative
                                                                    , pid.output);

  pid.last_integral   = pid.integral;
  pid.last_derivative = pid.derivative;
  pid.last_error      = pid.error;

  if(pid.error == 0) pid.output = 0;

  return pid.output;
}
```

Fuente: Elaboración propia.

Figura 2-62 Función PIDController().

2.8.2 Firmware básico “Barra y bola”

Este firmware tiene una dificultad menor al anterior debido a la cantidad de partes y piezas que colaboran dentro de la maqueta. La idea principal de esta maqueta consiste en poder posicionar la bola a la posición de referencia, utilizando como indicador de posición un sensor láser y como actuador un servomotor que permitirá ajustar el grado de inclinación de la barra, por lo que el firmware debe ser capaz de manejar estos dispositivos a la perfección y también poder ajustar los parámetros Kp, Ki, Kd, coeficiente de filtro derivativo y setpoint, sin necesidad de cargar nuevamente el firmware, por lo que se habilita la escritura de comandos por terminal serial. El firmware se puede dividir en declaración de librerías, declaración de constantes y variables, temporizador y control PID.

Las librerías utilizadas se muestran en la Figura 2-63.

```
#include <Arduino.h>
#include <Wire.h>

#include "ESP32_Servo.h"
#include "VL53L0X.h"
```

Fuente: Elaboración propia.

Figura 2-63 Librerías utilizadas.

Donde:

- **Arduino.h:** Contiene todas las herramientas y abstracciones del microcontrolador a utilizar en la tarjeta de desarrollo.
- **Wire.h:** Permite el uso del puerto serial I2C, para la lectura del sensor de distancia.
- **ESP32_Servo.h:** Permite la manipulación de servos mediante el uso de salidas PWM con pulsos de tiempos acotados según el modelo del servomotor.
- **VL53L0X.h:** Permite el uso del sensor de distancia láser VL53L0X.

Posteriormente se crean los objetos, constantes y variables del sistema, como se muestra en la Figura 2-64 y Figura 2-65. La mayoría de las variables están comentadas por lo que solo se explicarán en el código sin entrar en mayor detalle en el presente documento.

```
// * Objetos
Servo servo;    // Objeto para manejo de servomotor
VL53L0X laser;  // Objeto para manejo de sensor láser

// * Pines
const uint8_t PIN_SDA = 21;
const uint8_t PIN_SCL = 22;
const uint8_t PIN_SERVO = 27;

// * Valores del servomotor
const float MAX_OUT = 90;
const float MIN_OUT = -90;
static const uint8_t SERVO_INITIAL_ANGLE = 90;

// * Calibración de distancia medición
// (Calibración calculada por dos puntos medidos)
static const float M = 0.8772;
static const float B = 0;

// * Flags control
bool enable_controller = false;

// * Variables para controlador PID
float period = 0.1;
```

Fuente: Elaboración propia.

Figura 2-64 Declaración de objetos, constantes y variables.

```
struct VariablesPID {
  float Kp;                   // Valor de sintonización proporcional
  float Ki;                   // Valor de sintonización integral
  float Kd;                   // Valor de sintonización derivativa
  float N_coef;               // Valor coeficiente filtro derivativo

  float setpoint;             // Valor del setpoint
  float error;                // Valor actual del error

  float proportional;         // Valor actual de la parte proporcional
  float integral;             // Valor actual de la parte integral
  float derivative;           // Valor actual de la parte derviativa
  float output;               // Valor actual de la salida

  float last_error;           // Error anterior
  float last_integral;        // Valor integral anterior
  float last_derivative;      // Valor derivativo anterior
} pid;
```

Fuente: Elaboración propia.

Figura 2-65 Variables controlador PID.

Luego están los prototipos de funciones, utilizados para indicarle al compilador de Visual Code Studio que se ocuparán esas funciones y que existen dentro del firmware. Las funciones utilizadas se muestran la Figura 2-66.

```
// * Prototipo de funciones
float PIDController(float in_value);
void readConsole();
```

Fuente: Elaboración propia.

Figura 2-66 Prototipo de funciones.

La función **setup()** y **loop()** no se incluyen dentro de estos prototipos de funciones, ya que son declaradas dentro de la librería “Arduino.h”.

En la función setup() se agrega todo el inicio del sistema y todo lo que deba ejecutarse una sola vez durante el transcurso del firmware, como la configuración e inicio de puertos seriales, sensores y/o actuadores. Las inicializaciones de objetos se muestran la Figura 2-67.

```
// PIN setup()
void setup() {
  // Inicializa puertos seriales
  Serial.begin(115200);
  Wire.begin(PIN_SDA, PIN_SCL);

  // Inicializa valores PID en 0
  pid = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
         , 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

  pid.Kp = 0.84198;
  pid.Ki = 0.47872;
  pid.Kd = 0.33821;
  pid.N_coef = 7.9883;
  pid.setpoint = 150.0;

  // Inicia servomotor
  // Configura el pin del servo y los tiempos de trabajo para este
  servo.attach(PIN_SERVO, 1000, 2000);
  // Mueve el servo a la posición inicial
  servo.write(SERVO_INITIAL_ANGLE);

  // Inicia sensor de distancia
  laser.setTimeout(250);      // Setea timeout
  bool status = laser.init(); // Intenta iniciar el sensor

  if(status) {
    // Configura parámetros del sensor
    laser.setSignalRateLimit(0.1);
    laser.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 18);
    laser.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, 14);
    laser.setMeasurementTimingBudget(100e3);
  } else {
    // Si no inicia el sensor, bloquea el código
    while(1);
  }
}
```

Fuente: Elaboración propia.

Figura 2-67 Contenido de función setup() Inicio de objetos.

La función **loop()** contiene la función readConsole() y la ejecución del controlador si la bandera booleana enable_controller está habilitada, como se muestra en la Figura 2-68.

```
void loop() {
  // Lee los comandos escritos en consola
  readConsole();

  // Si el controlador está habilitado
  if(enable_controller) {
    // Lee el dato de distancia
    uint16_t distance = laser.readRangeSingleMillimeters();
    // Ajusta la distancia según la calibración
    float calibrated = ((float) (distance) * M) + B;
    // Ingresa el valor medido calibrado al controlador PID
    float servo_pos = PIDController(calibrated);
    // Mapea la salida del controlador a valores
    // aceptados por el actuador y escribe al servomotor
    servo_pos = map(servo_pos, -90, 90, 0, 180);
    servo.write(servo_pos);
  }
}
```

Fuente: Elaboración propia.

Figura 2-68 Contenido de función loop() funcionamiento del sistema.

La función **readConsole()** Permite leer comandos desde el puerto serial, para configurar la posición del servo de forma manual, para activar o desactivar el controlador PID, setear el valor de referencia, realizar una medición del sensor, para reiniciar la posición del servo al valor por defecto, configurar los valores de PID, como se muestra en la Figura 2-69.

```
void readConsole() {

  if (Serial.available()) {
    char c = Serial.read();

    switch(c) {
      case 'c': //? habilita / deshabilita PID
      {…
      } break;

      case 's': //? Setea setpoint
      {…
      } break;

      case 'P': //? Cambia la posición del servomotor
      {…
      } break;

      case 'm': //? Mide y muestra la distancia
      {…
      } break;

      case 'p': //? Cambia el factor proporcional (Kp)
      {…
      } break;

      case 'i': //? Cambia el factor integral (Ki)
      {…
      } break;

      case 'd': //? Cambia el factor derivativo (Kd)
      {…
      } break;

      case 'n': //? Cambia el coeficiente del filtro
      {…
      } break;
    }
  }
}
```

Fuente: Elaboración propia.

Figura 2-69 Contenido de la función readConsole().

Por último, está la función **PIDController()**, misma utilizada en la maqueta anterior, como se muestra en la Figura 2-70.

```
// PIN  PIDController()
float PIDController(float in_value) {
  pid.error = pid.setpoint - in_value; // Calcula el error
  pid.proportional = pid.Kp * pid.error;      // Caclula la parte proporcional

  // Calcula la parte integral si no está saturada la salida.
  pid.integral = pid.Ki * 0.5 * period * (pid.error + pid.last_error) + pid.last_integral;
  // Anti Windup
  if(pid.integral > MAX_OUT) pid.integral = MAX_OUT;
  else if(pid.integral < MIN_OUT) pid.integral = MIN_OUT;

  // Calcula la parte derivativa del controlador
  pid.derivative =  pid.N_coef * pid.Kd * (pid.error - pid.last_error)
                  + (1.0 - pid.N_coef * period) * pid.last_derivative;

  // Suma todas las partes para obtener la salida
  pid.output = pid.proportional + pid.integral + pid.derivative;

  if(pid.output > MAX_OUT) pid.output = MAX_OUT;
  else if(pid.output < MIN_OUT) pid.output = MIN_OUT;

  Serial.printf("s: %.2f - p: %.2f - i: %.2f - d: %.2f - o: %.2f\n" , pid.setpoint
                                                                    , pid.proportional
                                                                    , pid.integral
                                                                    , pid.derivative
                                                                    , pid.output);

  pid.last_integral   = pid.integral;
  pid.last_derivative = pid.derivative;
  pid.last_error      = pid.error;

  if(pid.error == 0) pid.output = 0;

  return pid.output;
}
```

Fuente: Elaboración propia.

Figura 2-70 Función PIDController().

2.9 DETECCIÓN Y CONTROL DE SISTEMA BARRA Y BOLA

Para el análisis de la maqueta de control “Barra y Bola” se considera como un sistema desconocido por lo que se procede a analizar la naturaleza de éste, recordando que dentro del control industrial existen plantas o sistemas **estables** e **inestables**.

Los sistemas estables son aquellos que ante una entrada limitada o acotada, su salida también es limitada o acotada. Contrario es el caso de sistemas inestables, cuya respuesta a una entrada limitada o acotada su respuesta es oscilatoria o tiende a infinito.

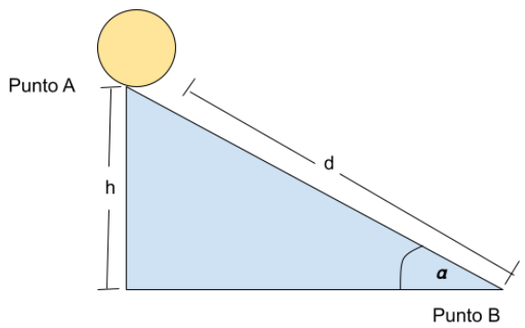
Para poder analizar y lograr el mejor controlador PID para el sistema, se realiza un cálculo teórico simple. Junto al análisis teórico se realizan pruebas prácticas para simular y controlar la planta, utilizando el software Matlab.

2.9.1 Análisis teórico

Para detectar la estabilidad del sistema “Barra y Bola”, de forma inicial se realiza un análisis mental considerando una barra de forma horizontal con inclinación de 0°. En uno de los

extremos de esta barra se ubica una bola y transcurrido un tiempo, la inclinación de la barra cambia a 20°. El cambio de inclinación hará que la bola comience a rodar hacia el final de la barra, pero si la barra es infinita, la bola continuará alejándose de su punto inicial hasta que alguna perturbación externa remueva o frene su avance.

El ejemplo anterior corresponde a un ejercicio de **MRUA** (Movimiento Rectilíneo Uniformemente Acelerado), concretamente a un ejercicio de: "*Esfera sobre un plano inclinado*", como se observa en la Figura 2-71.



Fuente: Elaboración propia.

Figura 2-71 Ejemplo de ejercicio "Esfera sobre un plano inclinado".

El cálculo matemático para obtener la posición de la bola en este ejercicio, puede ser abordado obteniendo la energía en el punto A y la energía en el punto B, así poder igualar y despejar las variables correspondientes, por lo tanto:

La esfera en el punto A, en tiempo 0 se encuentra en reposo y su energía potencial gravitatoria corresponde a:

$$E_{pa} = mgh$$

Donde:

Epa: Energía en el punto A

m: Masa de la bola (kg).

g: Gravedad ($^{m}/_{s^2}$).

h: Altura (m).

Dejando la altura en función al grado de inclinación, trigonométricamente se obtiene:

$$E_{pa} = mgdsen(\alpha)$$

La energía presente en el punto B corresponde a energía cinética de rotación y traslación:

$$E_{pb} = E_{ct} + E_{cr}$$

Donde:

Epb: Energía en el punto B.

Ect: Energía cinética de traslación.

Ecr: Energía cinética de rotación.

La energía cinética de traslación y rotación se definen de la siguiente forma:

$$E_{ct} = \frac{1}{2}mv^2$$

$$E_{cr} = \frac{1}{2}I\omega^2$$

Donde:

m: Masa de la bola (m).

v: Velocidad de desplazamiento de la bola (m/s).

I: Momento de inercia de la bola.

ω: Velocidad angular de la bola (rad/s).

El momento de inercia de la bola está dada por tablas, y varía según la geometría del objeto. Para esta maqueta se utiliza una bola hueca con una fina capa de perímetro por lo que su momento de inercia es:

$$I = \frac{2}{3}mr^2$$

Donde:

m: Masa de la bola (kilogramos).

r: Radio de la bola (m).

También, la velocidad angular puede es equivalente a la velocidad dividido entre el radio de la bola:

$$\omega = \frac{v}{r}$$

Como la conservación de la energía indica que la energía del punto A debe ser igual a la energía del punto B, es correcto afirmar que:

$$m\ g\ d\ sen(\alpha) = = \frac{1}{2}mv^2 + \frac{1}{2}\left(\frac{2}{3}mr^2\right)\frac{v^2}{r^2}$$

$$g\ d\ sen(\alpha) = \frac{1}{2}mv^2 + \frac{5}{6}mv^2$$

$$g\ d\ sen(\alpha) = mv^2\frac{4}{3}$$

Por lo tanto, para obtener la velocidad de la bola:

$$v^2 = \frac{3}{4}\frac{g\ d\ sen(\alpha)}{m}$$

$$v = \sqrt{\frac{3}{4}\frac{g\ d\ sen(\alpha)}{m}}$$

Dentro de la cinemática, se conoce que la velocidad final del objeto en plano inclinado es:

$$v^2 = 2\ a\ d$$

Donde:

a = Aceleración ($^{m}/_{s^2}$).

d = Distancia recorrida.

Por lo tanto, sustituyendo términos:

$$g\ d\ sen(\alpha) = \frac{4}{3}\ m\ 2\ a\ d$$

Despejando la aceleración se obtiene:

$$a = \frac{3}{4}\frac{g\ sen(\alpha)}{2\ m}$$

Con la aceleración obtenida y estableciendo que a es una constante, debido a que se conocen todos los valores y que el ángulo α es un valor que puede ser definido, se puede proceder a integrar para obtener la velocidad y volver a integrar para obtener la posición, resultado en lo siguiente:

$$y = \frac{1}{2} * a * t^2 + V_0 * t + y_0$$

Donde:

y: Posición de la bola.

V_0: Velocidad inicial.

y_0: Posición inicial.

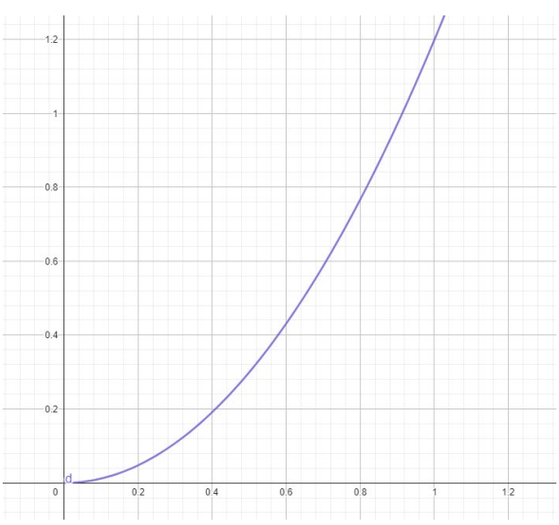
t: Tiempo transcurrido.

a: Aceleración constante.

Dando como resultado la siguiente ecuación en función al tiempo:

$$y(t) = \frac{1}{2}\frac{3}{4}\frac{g\ sen(\alpha)}{2\ m}t^2$$

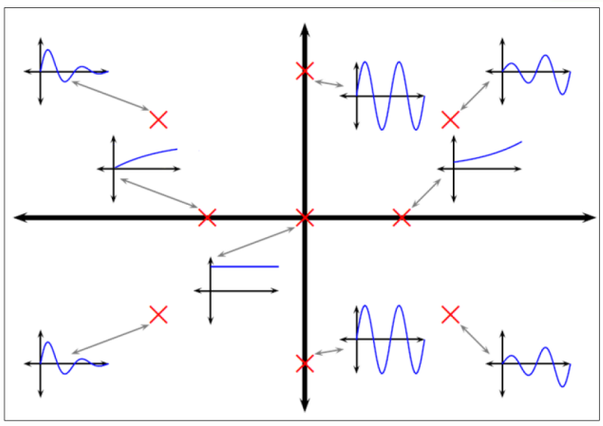
Si se sustituyen los términos con valores ficticios por ejemplo m = 0.02 y α = 45°, la respuesta en el tiempo dará como resultado una gráfica como la mostrada en la Figura 2-72.



Fuente: Elaboración propia.

Figura 2-72 Gráfica de posición de la bola con respecto al tiempo.

El comportamiento de la maqueta “Barra y Bola” es inestable, ya que los polos se ubican en el semiplano derecho del **Plano S**, como se muestra en la Figura 2-73.



Fuente: Diapositiva 5 de la presentación PowerPoint “Estabilidad de Sistemas (Routh-Hurwitz)” mostrado en la unidad de competencia “Control de procesos” de la carrera Ingeniería en Control y Automatización Industrial, de la Universidad Federico Santa María, sede Viña del Mar.

Figura 2-73 Plano S.

Como la respuesta se encuentra en el semiplano derecho y además con el eje Y igual a 0, indica que el sistema es inestable y no posee polos con parte imaginaria, por lo que es un sistema controlable.

El análisis anterior puede ser utilizado para obtener la función de transferencia del sistema barra-bola, así lo desarrollan en varias fuentes donde muestran cómo obtener el modelo matemático del comportamiento para esta maqueta. El problema en las páginas consultadas, es que no contemplan el efecto del servo motor, ya que realizan el control sobre simulaciones ideales en el mismo software Matlab, esto provoca que el sistema en la realidad no tenga una respuesta adecuada a los parámetros ajustados del controlador.

2.9.2 Análisis práctico

Para poder obtener la función de transferencia del sistema se ha utilizado el método de Adquisición de datos para el análisis con la herramienta SystemIdentification. Esta forma normalmente aplica para sistemas estables debido a que, al aplicar un escalón de entrada la salida se estabilizará en un valor fijo transcurrido el tiempo.

En el sistema barra y bola utilizar este método no es óptimo ya que, al aplicar un escalón la bola rodará hasta llegar al extremo de la barra, lo cual no es el comportamiento real del sistema.

Aun así, utilizar este método permite la obtención de una función de transferencia que cumple con la finalidad de ajustar y controlar la planta para que la bola se posicione en las distancias puestas como referencia.

Los pasos realizados para lograr este análisis son los siguientes:

- Obtener mediciones reales del sistema.
- Obtener la función de transferencia.
- Sintonizar el controlador.
- Observar los resultados.

2.9.2.1 Obtención de mediciones reales del sistema

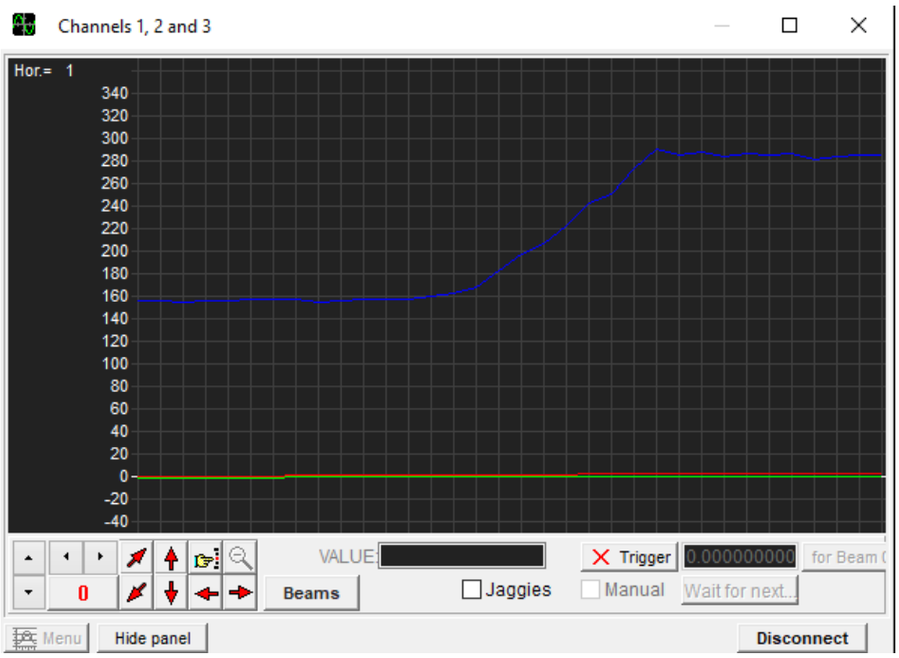
Para obtener datos reales del sistema, se implementó una función extra dentro del firmware Barra y bola, mostrada en la Figura 2-74, que permite generar una inclinación de 20 grados de la barra, transcurrido 1 segundo. En todo momento se mide y se grafica la respuesta con el software “Serial Oscilloscope”, el cual permite almacenar la información registrada en un archivo CSV.

```
void datalogger() {
  if(enable_datalogger) {
    // Lee el dato de distancia
    uint16_t distance = laser.readRangeSingleMillimeters();
    // Ajusta la distancia según la calibración
    float calibrated = ((float) (distance) * M) + B;
    // Aumenta el contador
    counter++;

    // Cuando el contador llegue a 10, significa que habrá transcurrido
    // 1 segundo, por lo que cambiará la posición del servomotor para
    // mover la bola.
    if(counter == 10) {
      angle += 20;
      servo.write(angle);
    }

    Serial.printf("%.3f,%.3f\r", angle, calibrated);
  }
}
```

Fuente: Elaboración propia.

Figura 2-74 Función datalogger() para obtención de datos del sistema.

Para la obtención de datos la bola se posiciona al medio de la barra (150 mm). Al ejecutar la función datalogger(), la gráfica generada muestra la respuesta calculada teóricamente de color azul, como se observa en la Figura 2-75.



Fuente: Elaboración propia.

Figura 2-75 Posición de la bola graficada.

Si bien, el sistema es inestable, la gráfica azul de la figura anterior muestra que se ha estacionado en un valor. Esto corresponde a la "saturación" mecánica de la maqueta, ya que cuenta con un tope al final del recorrido la cual impide que la bola siga su trayectoria.

2.9.2.2 Obtención de la función de transferencia

Los datos obtenidos en CSV pueden ser importados como un arreglo a Matlab, luego de importarlo se guarda con nombre "Datasetsmallsetp.mat", para posteriormente poder utilizarlo en el código que se muestra en la Figura 2-76.

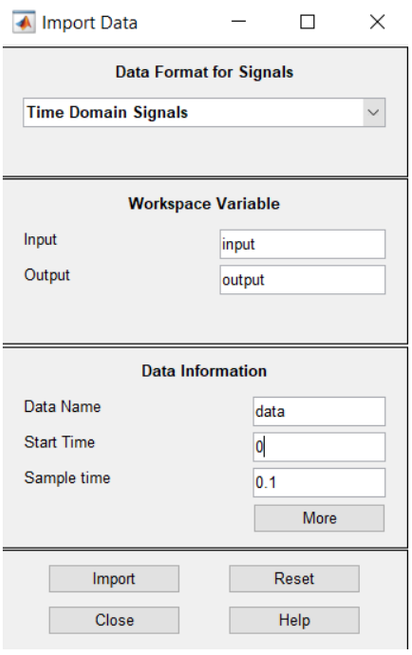
A los datos de actuación (posición del rotor) y respuesta de la planta (distancia en mm) se le restan los offsets, es decir, la posición del servomotor queda de 0 a 20 grados y el dato de posición de 0 a 150 mm, en vez de 90 a 110 grados y 150 a 300 mm, respectivamente.

```
load("Datasetsmallstep.mat");

time = Datasetsmallstep(:,1) / 1000;
input = Datasetsmallstep(:,2) - 90;
output = Datasetsmallstep(:,3) - 150;


plot(time, input, time, output);
systemIdentification();
```

Fuente: Elaboración propia.

Figura 2-76 Código implementado en Matlab para análisis de datos.

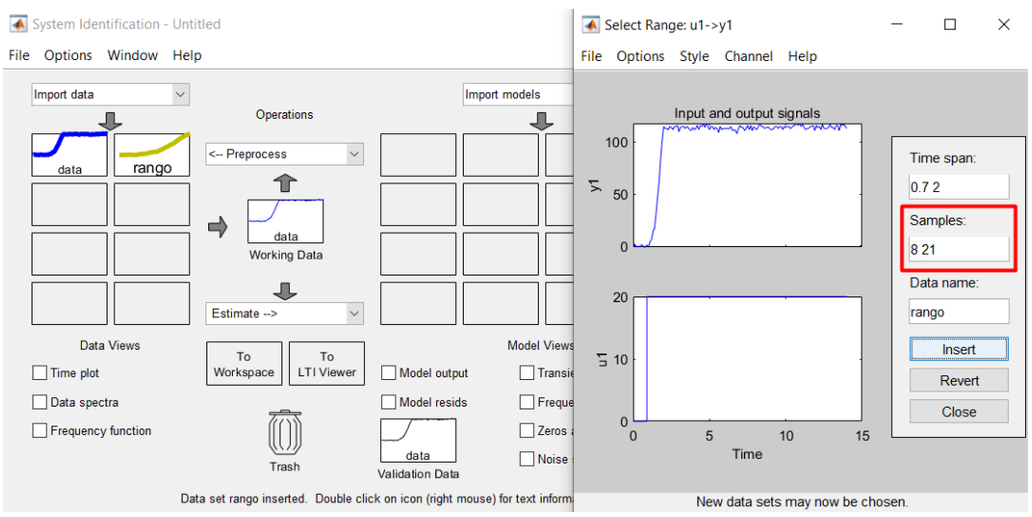
Al usar la herramienta systemIdentification, se importan los datos de entrada y salida, con tiempo de inicio en cero y tiempo de muestreo de 0.1 segundos, como se muestra en la Figura 2-77.



Fuente: Elaboración propia.

Figura 2-77 Importación de datos systemIdentification.

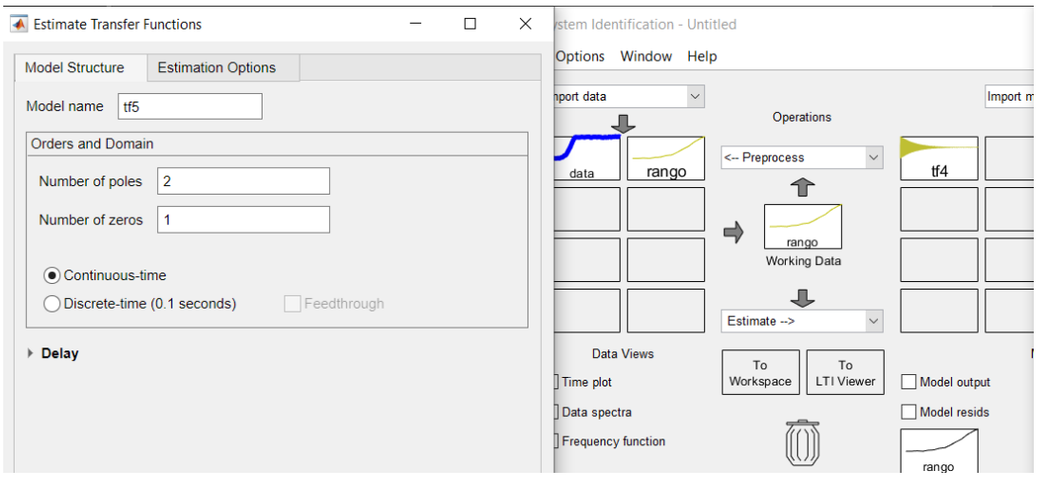
Con los datos importados, se selecciona el rango de la información a utilizar. Para lograr obtener un comportamiento similar al real, se seleccionan datos cercanos al momento de iniciar el escalón y tiempo antes de llegar al límite de la barra donde el dispositivo se satura físicamente. Con los datos obtenidos, el rango para este caso es de 8 a 21 muestras, como se observa en la Figura 2-78.



Fuente: Elaboración propia.

Figura 2-78 Selección del rango de datos.

La identificación del sistema se realiza con “Transfer function model”, contemplando 2 polos y 1 cero como se observa en la Figura 2-79.



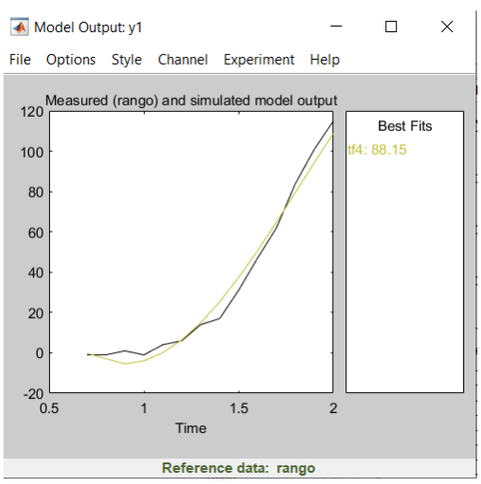
Fuente: Elaboración propia.

Figura 2-79 Identificación del sistema.

De la operación realizado por Matlab, se obtiene que la función de transferencia de la maqueta “Barra y bola” es la siguiente:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{1.433s + 11.71}{s^2 + 1.743e^{-6}s + 2.837}$$

La coincidencia de la función de transferencia con respecto a la señal original es del 88.15%, como se observa en la Figura 2-80. Este valor depende del sistema y de la precisión que se requiere. Idealmente es mejor tener una coincidencia del 100%, pero dentro de todas las pruebas e identificaciones del sistema, fue la coincidencia más alta obtenida.

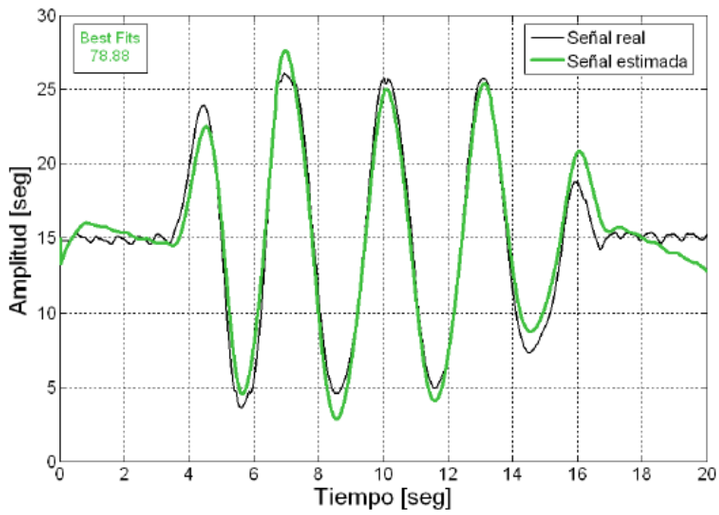


Fuente: Elaboración propia.

Figura 2-80 Coincidencia de la señal original vs la FT obtenida.

Para identificar el sistema correctamente se realizaron múltiples pruebas, con distintos rangos de medición, distintos escalones e incluso, el periodo de adquisición de datos de 50 ms en vez de 100 ms.

Para obtener una mejor respuesta o coincidencia mayor al 88% podría utilizarse otras formas de obtención de datos. Otro documento consultado, utilizó systemIdentification con datos de la bola oscilando en torno al centro de la barra, como se muestra en la Figura 2-81. Esta respuesta le permitió al autor realizar un control de la maqueta de forma correcta, por lo que se recomienda a los usuarios de la maqueta, probar distintos métodos para mejorar el control actual de ésta.

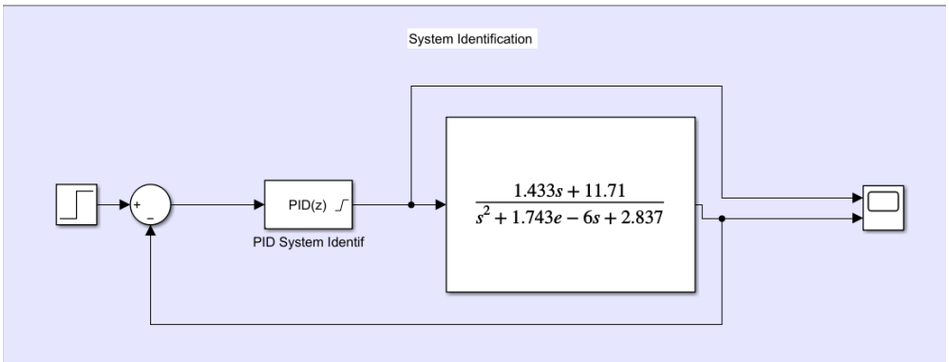


Fuente: Desarrollo de un sistema Ball and Beam, Para implementar estrategias de control mediante LabView - Capitulo 3.5.

Figura 2-81 Ejemplo de otras formas para identificar el sistema Barra y Bola.

2.9.2.3 Sintonización del controlador PID

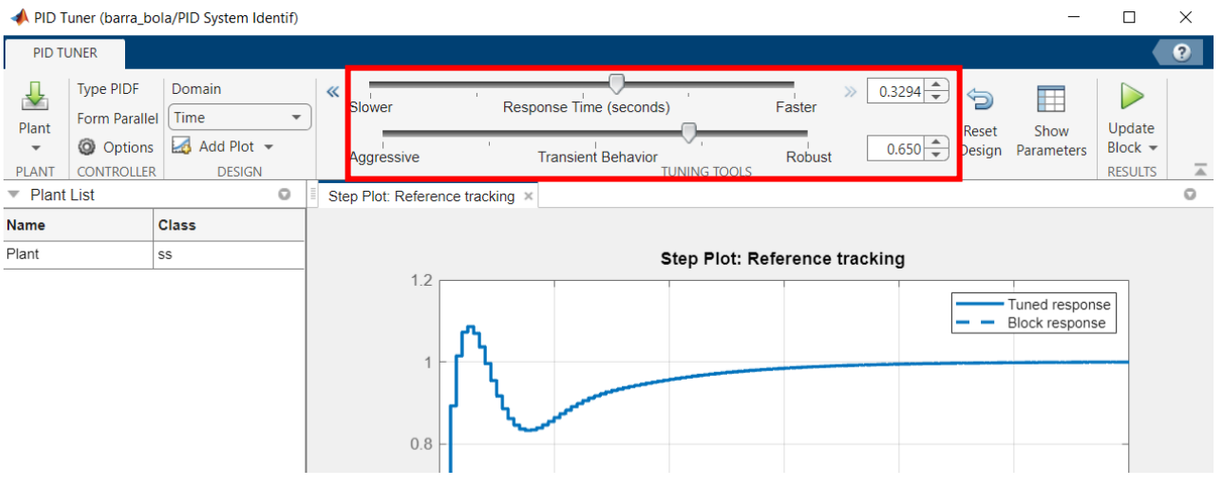
Para sintonizar el controlador PID se ha implementado una simulación con la herramienta Simulink del software Matlab, como se muestra en la Figura 2-82.



Fuente: Elaboración propia.

Figura 2-82 Sistema implementado en Simulink.

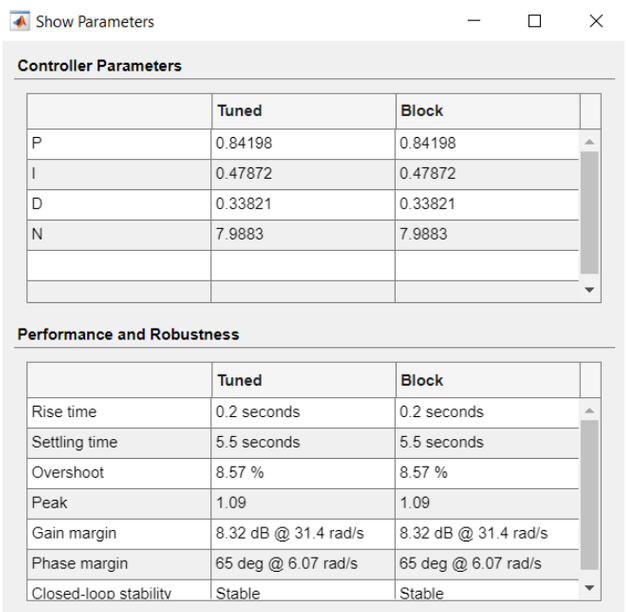
El criterio principal de sintonización para la maqueta consiste en obtener una respuesta rápida, menor a 1 segundo y un tiempo de asentamiento menor a 10 segundos. Con el bloque PID ajustado según los parámetros indicados en la Figura 2-28, por defecto se obtiene una velocidad de respuesta de 0.3294 segundos en la herramienta PID Tuner, como se muestra en la Figura 2-83.



Fuente: Elaboración propia.

Figura 2-83 Respuesta de maqueta Barra y Bola sintonizada.

En la Figura 2-84 se puede observar los valores para Kp, Ki, Kd y coeficiente N. También, se observa que el sistema tendrá un overshoot del 8.57%, un rise time de 0.2 segundos y un tiempo de asentamiento de 5.5 segundos, por lo que el controlador cumple con los requerimientos de diseño establecido inicialmente.



Fuente: Elaboración propia.

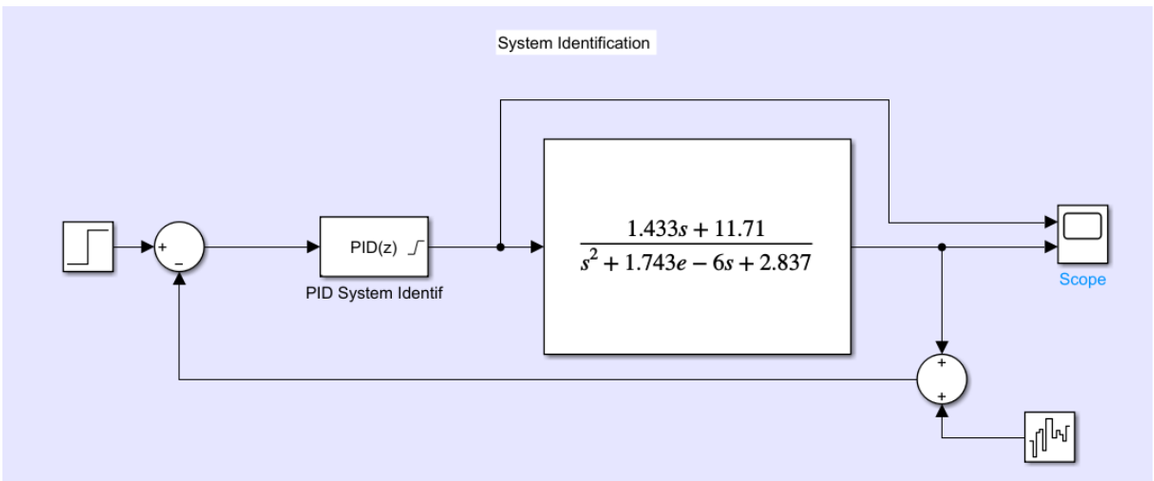
Figura 2-84 Parámetros PID configurados.

2.9.2.4 Resultados

Con el firmware y los valores sintonizados en la maqueta, se observa que el comportamiento de ésta última, cumple con el objetivo de posicionar la pelota a los distintos puntos de referencia que se le asignen.

Dentro del sistema se observa vibración en la planta, debido al error de distancia medido aproximadamente de ±3 mm del sensor láser. Esto provoca que la salida del actuador varíe constantemente. Para poder visualizar de forma simulada el efecto provocado en el actuador, por el error del sensor, se ha agregado un bloque de ruido a la simulación de Simulink, como se observa en la Figura 2-85.

En la práctica, el sensor de distancia láser presenta mayor error mientras más lejos deba medir, fenómeno normal para el sensor de distancia utilizado debido a al corto alcance de éste que es de 1200 mm. El error puede disminuir si se implementan estrategias digitales como filtros, o si se cambia el sensor actual (VL53L0X) al modelo VL53L1X que tiene un alcance de 4000 mm, el cual, al permitir rangos de lecturas largos se logra una mayor precisión a 300 mm.

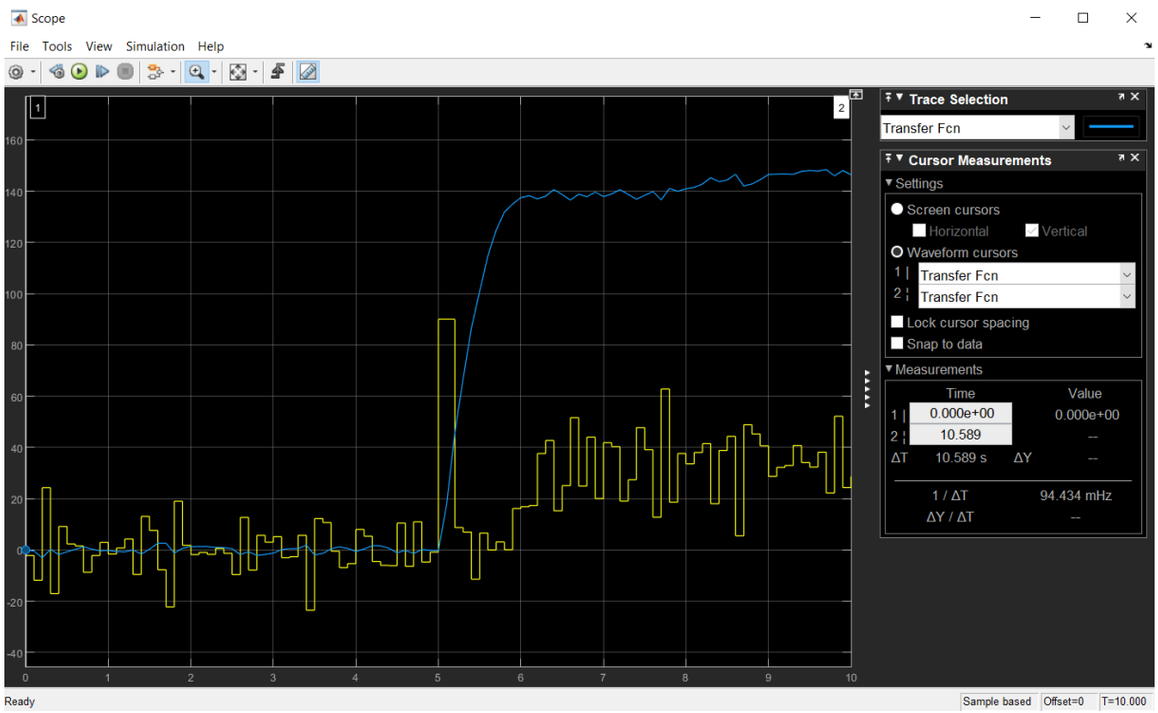


Fuente: Elaboración propia.

Figura 2-85 Planta simulada en Simulink con ruido, simulando error en la medición.

Si bien las señales ingresadas al bloque “Scope” son de distintas unidades de medida y no corresponde que se grafiquen juntas, se hace para facilitar la visualización sobreponiendo las señales en el mismo gráfico.

En la Figura 2-86 se puede observar el ruido de entrada y la respuesta que genera hacia el servomotor. La línea azul corresponde a la señal de salida medida, mientras que la línea amarilla representa la señal de actuación, el comportamiento que tendrá el servomotor para corregir el error en la salida.



Fuente: Elaboración propia.

Figura 2-86 Respuesta de la planta con error de ±3 mm.

CAPÍTULO 3: RESULTADOS, EVALUACIÓN ECONÓMICA Y MEJORAS

3 RESULTADOS, EVALUACIÓN ECONÓMICA Y MEJORAS

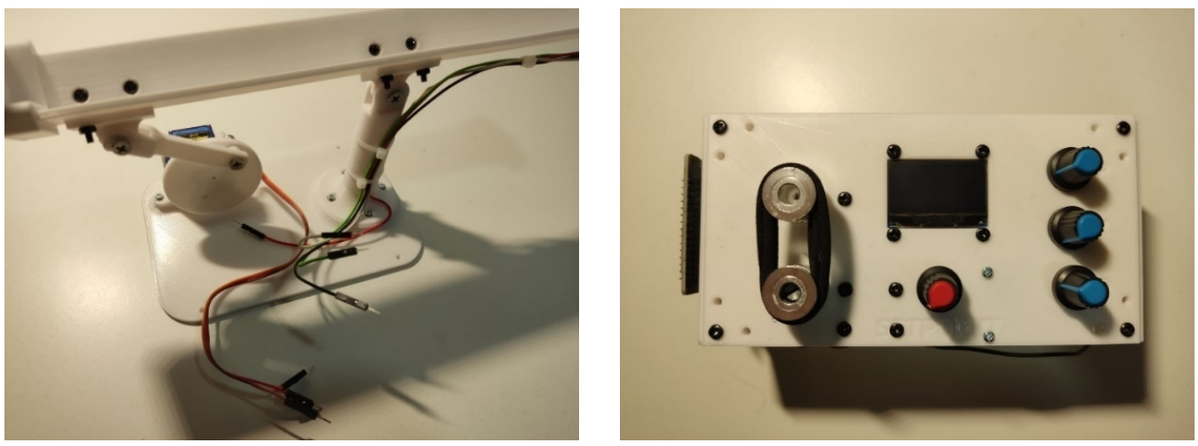
En el presente capítulo se da a conocer los resultados obtenidos en cuanto al cumplimientos de objetivos, general y específicos. También se muestra una breve evaluación económica para estimación de costos en cuanto a la replicación de las maquetas y propuestas de mejoras para funcionamiento o fabricación de estas.

3.1 RESULTADOS

Se muestran los resultados obtenidos según los objetivos planteados en el capítulo 1 y según lo logrado físicamente en hasta el término del proyecto.

3.1.1 Confeccionar una maqueta didáctica con fines educativos en aplicaciones de control automático

Se completó el objetivo ya que se diseñó y confeccionó dos maquetas de control de posición., las cuales se muestran en la Figura 3-1.



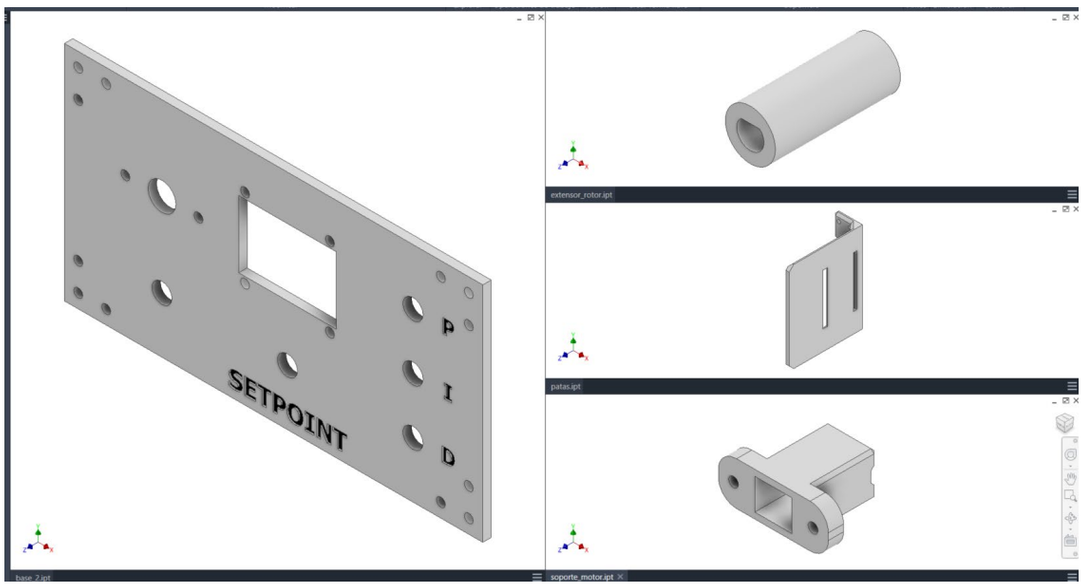
Fuente: Elaboración propia.

Figura 3-1 Maqueta "Barra y bola" a la izquierda, maqueta "Demostración de control de un Servomotor" a la derecha.

Se tienen todos los diseños 3D en formato ipt para ser visualizados con el software Autodesk Inventor, como se muestran en la Figura 3-2 y en la Figura 3-3. También se ha generado el archivo STL para la replicación con impresoras 3D.

Fuente: Elaboración propia.

Figura 3-2 Partes y piezas maqueta "Barra y bola".



Fuente: Elaboración propia.

Figura 3-3 Partes y piezas maqueta "Demostración de control de un Servomotor".

3.1.2 Determinar el tipo de maqueta a diseñar y fabricar

Se determinó que el tipo de maqueta sería de posición, “Barra y bola” y “Demostración de control de un Servomotor”, debido a la facilidad y el tiempo de fabricación de estos.

3.1.3 Especificar requerimientos del dispositivo

Se estableció como requerimientos del dispositivo la capacidad de no depender de red eléctrica 220 VAC y de ser transportable mediante la utilización de una maleta, solucionándose con una batería “Power Bank” y una maleta de seguridad, mostradas en la Figura 3-4.

Fuente: https://fotosol.cl/products/cargador-portatil-audiolab-10000mah-1?variant=31583435817071 y https://www.easy.cl/caja-de-herramientas-20-mj-6060-robust-1277654/p

Figura 3-4 Power Bank y maleta utilizadas en el proyecto.

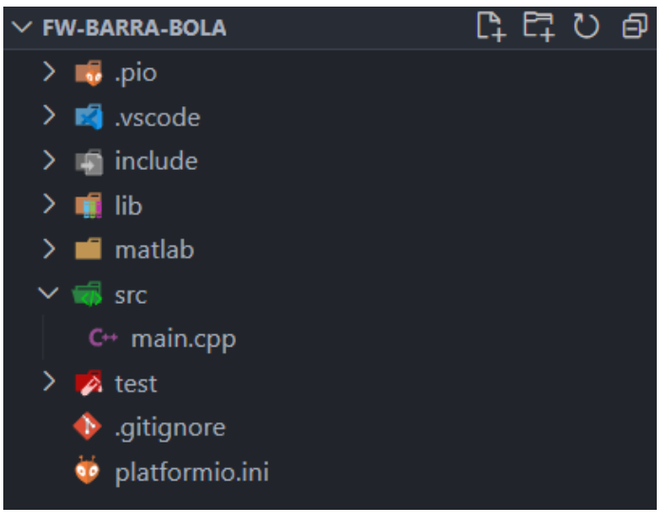
Estos dos materiales permiten el transporte seguro de las maletas y la portabilidad. La batería puede ser cargada por USB en cualquier puerto, ya sea en un PC o un adaptador (cargador de celular).

3.1.4 Creación de firmware básico

Se generó un firmware básico para testear el controlador PID con un sistema para cargar un condensador completamente en menos de 5.

Para las maquetas “Demostración de control de un Servomotor” y “Barra y bola” se realizó un firmware en donde se pueden ajustar los valores de PID e inicializar el control sobre las plantas.

El firmware se realizó en el IDE Visual Studio Code, utilizando la plataforma PlatformIO el cual permite la integración y utilización de varios modelos de placas de desarrollo con Arduino. En la Figura 3-5 se muestran los directorios con todo el código generado.

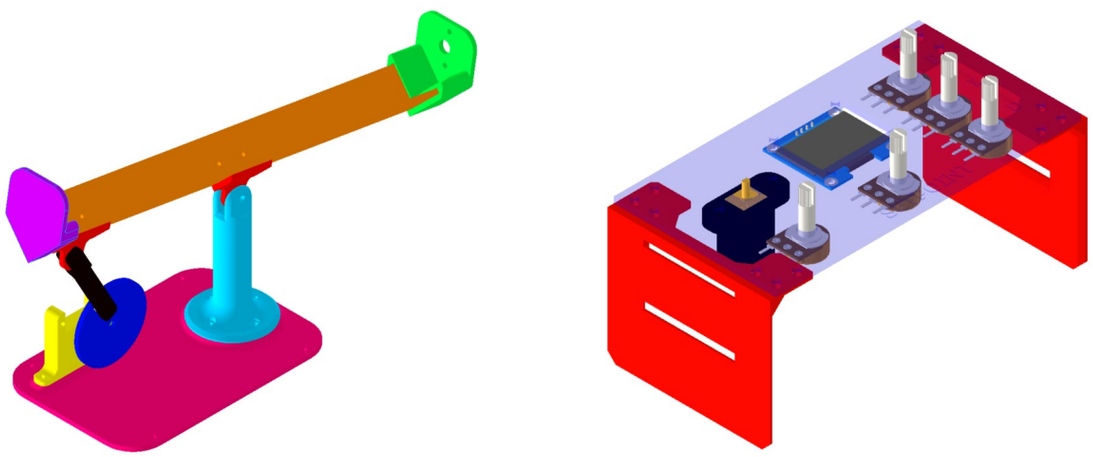


Fuente: Elaboración propia.

Figura 3-5 Directorios firmware maqueta "Barra y bola".

3.1.5 Diseñar modelos en 3D que componen la maqueta

Se diseñaron los modelos 3D para la fabricación de la maqueta, mediante la utilización del software Autodesk Inventor, como se observa en la Figura 3-6.



Fuente: Elaboración propia.

Figura 3-6 Modelos 3D de maquetas, ensambladas.

Para la maqueta de “Barra y bora” el tiempo de impresión de todas sus partes y piezas es de 7 horas y 38 minutos aproximadamente, mientras que para el “Demostración de funcionamiento de un servomotor” es de 3 horas aproximadamente.

El tiempo de impresión depende del posicionamiento de las piezas en el software Slicer que se utilice como, por ejemplo, Prusa Slicer. También depende de la configuración y capacidades de la impresora 3D utilizada.

3.1.6 Seleccionar la instrumentación que forma parte del proyecto

Se seleccionó los módulos que componen ambas maquetas y el microcontrolador según criterio precio/calidad, además de la disponibilidad dentro de Chile.

3.1.7 Identificar el sistema de una de las maquetas

Se ha realizado la identificación del sistema de la maqueta “Barra y bola” por medio del software Matlab. Concretamente se logró:

- Realizar una pequeña demostración teórica del sistema.
- Obtención de datos desde la maqueta.
- Analizar los datos.
- Diseñar un controlador PID.
- Generar el control de la maqueta.

3.1.8 Desarrollar diagrama de flujo de la programación del controlador

Se ha desarrollado un diagrama básico del funcionamiento que se repite en ambas maquetas, como se muestra en la Figura 2-51. El funcionamiento de pantalla y módulos no se ha explicado en detalle con diagramas, debido a que no es relevante para el sistema.

3.1.9 Realizar un estudio de costos

Se realizó varias listas de costos dentro del siguiente punto de este capítulo, abarcando el costo total del proyecto, costo de maquetas independientes y costo total del proyecto con componentes importados desde el extranjero.

3.1.10 Dispositivo final

El equipo logrado al final del proyecto consiste en dos maquetas de control de posición, que funcionan de forma independiente entre ellas. Ambas se almacenan en la misma maleta, pero el microcontrolador debe ser programado de forma independiente por cada maqueta.

Las maquetas generadas consisten en “Barra y bola” y una maqueta “Demostración de control de un Servomotor”. Ambas maquetas se alimentan desde un Power Bank, que permiten

la alimentación de la placa de desarrollo utilizada y desde esta misma obtener niveles de voltaje como +3.3 VDC. En la Figura 3-7 se muestra la maleta con las dos maquetas fabricadas.

Fuente: Elaboración propia.

Figura 3-7 Maquetas almacenadas en la maleta de transporte.

3.1.11 Repositorio

Se ha generado un repositorio con todos los datos útiles de este proyecto, entre ellos las distintas partes y piezas de las maquetas, firmwares y ensamblaje.

Para acceder al repositorio se puede usar el siguiente enlace https://github.com/JojeNN97/maquetas-control-automatico. Ahí se encontrará todo el material generado para las maquetas y se espera continuar agregando nuevas maquetas, si la venta de estas iniciales resulta correctamente.

3.2 EVALUACIÓN ECONÓMICA

Para conocer el valor o coste de la ejecución y posterior venta de las maquetas, se realizan tablas para conocer el material, precio en CLP y precio en UF del día jueves 8 de junio del 2023, conversión que está a $36.060,75 CLP.

Las tablas contempladas para la evaluación son:

- Valor de ambas maquetas con maleta.
- Valor solo de la maqueta “Barra y bola”.
- Valor solo de la maqueta “Demostración de control de un Servomotor”.

- Valor de las maquetas y maleta, con proveedores extranjeros.

Dentro de los valores se incluye la “Fabricación de la maqueta” que corresponde a las HH propias del trabajo de imprimir y cablear las maquetas. No se contempla el tiempo de desarrollo de las partes y firmware, puesto a que este se entrega completamente gratis en plataformas web. El concepto de HH puede ser ignorado si no se requiere el armado de las maquetas.

El tiempo considerado para la fabricación de las maquetas es de 3 horas en total para cada una, considerando las piezas previamente impresas. Calculando el valor hora a $10.000 CLP, en total serían $30.000 CLP por maqueta. Si se desean ambas maquetas se aplica un descuento del 50% para la segunda, por lo que en total se obtiene un valor de $45.000.

3.2.1 Valor de ambas maquetas con maleta

En la Tabla 3-1, se presentan los costos asociados a la confección de ambas maquetas y la maleta de transporte:

Tabla 3-1 Costo total del proyecto.

Material	**Precio CLP**	**Precio UF**
Maleta	$39.990	1,11
PLA (1 Kg)	$20.000	0,55
ESP32 DEVKIT V1	$8.990	0,25
Potenciómetro 100 KΩ (5 unidades)	$3.000	0,08
Perillas potenciómetro (5 unidades)	$950	0,03
1 motor DC N20	$6.990	0,19
OLED 1.3” 128x64	$4.990	0,14
Sensor láser VL53L0X	$3.990	0,11
Servomotor SG90	$2.990	0,08
Puente H L9110s	$2.365	0,07
ADS1115	$5.360	0,15
Perno M3 8mm (26 unidades)	$1.000	0,03
Tuerca M3 (26 unidades)	$1.000	0,03
Perno M4 12 mm	$1.000	0,03
Tuerca a presión M4	$1.000	0,03
Polea dentada (2 unidades)	$7.000	0,19
Fabricación maqueta (HH)	$45.000	1,25
Power Bank	$16.000	0,44

Fuente: Elaboración propia.

Total CLP: $171.615

Total UF: 4,76

3.2.2 Valor de la maqueta “Barra y bola”

También se estimó el costo para solo la maqueta “Barra y bola” sin maleta, como se muestra en la Tabla 3-2.

Tabla 3-2 Costo maqueta "Barra y bola".

Material	**Precio CLP**	**Precio UF**
PLA	$20.000	0,55
ESP32 DEVKIT V1	$8.990	0,25
Sensor láser VL53L0X	$3.990	0,11
Servomotor SG90	$2.990	0,08
Perno M3 8mm (26 unidades)	$1.000	0,03
Tuerca M3 (26 unidades)	$1.000	0,03
Perno M4 12 mm	$1.000	0,03
Tuerca a presión M4	$1.000	0,03
Fabricación maqueta (HH)	$30.000	0,83
Power Bank	$16.000	0,44

Fuente: Elaboración propia.

Total CLP: $85.970

Total UF: 2,38

3.2.3 Valor de la maqueta “Demostración de control de un Servomotor”

En la Tabla 3-3 se muestra el costo total de la maqueta “Demostración de control de un Servomotor” sin maleta de transporte.

Tabla 3-3 Costo total de maqueta "Demostración de control de un Servomotor".

Material	**Precio CLP**	**Precio UF**
PLA	$20.000	0,55
ESP32 DEVKIT V1	$8.990	0,25
Potenciómetros (5 unidades)	$3.000	0,08
Perillas potenciómetro (5 unidades)	$950	0,03
1 motor DC N20	$6.990	0,19
OLED 1.3” 128x64 px	$4.990	0,14
Puente H L9110s	$2.365	0,07
ADS1115	$5.360	0,15
Perno M3 8mm (26 unidades)	$1.000	0,03
Tuerca M3 (26 unidades)	$1.000	0,03
Polea dentada (2 unidades)	$7.000	0,19
Fabricación maqueta (HH)	$30.000	0,83
Power bank	$16.000	0,44

Fuente: Elaboración propia.

Total CLP: $107.645

Total UF: 2,98

3.2.4 Alternativa de compras

Los precios pueden ser reducidos si los materiales son importados desde el extranjero, concretamente desde tiendas de Aliexpress. En la Tabla 3-4 se pueden observar una comparativa con los precios nacionales versus precios encontrados en el extranjero, siendo estos últimos menores. Además, la mayoría de los valores no superan los 30 USD, por lo que no hay que preocuparse del pago de arancel aduanero (6% del valor del producto) y del IVA (%19) del valor del producto.

El tiempo de entrega de Aliexpress actualmente es de dos semanas aproximadamente, gracias a la adquisición de un avión propio de ellos que viaja a Chile constantemente, por lo que la adquisición de materiales es rápida en comparación a las cuatro a seis semanas que demoraba antiguamente.

Tabla 3-4 Alternativa de compras.

Material	**Precio Nacional**	**Precio Extranjero**
Maleta	$39.990	$25.950
PLA	$20.000	$20.000
ESP32 DEVKIT V1	$8.990	$4.000
Potenciómetros (5 unidades)	$3.000	$2.024
Perillas potenciómetro (5 unidades)	$950	$430
1 motor DC N20	$6.990	$2.792
OLED 1.3” 128x64 px	$4.990	$3.201
Sensor láser VL53L0X	$3.990	$2.435
Servomotor SG90	$2.990	$1.916
Puente H L9110s	$2.365	$1.596
ADS1115	$5.360	$2.955
Perno M3 8mm (26 unidades)	$1.000	$1.258
Tuerca M3 (26 unidades)	$1.000	$575
Perno M4 12 mm	$1.000	$2.280
Tuerca a presión M4	$1.000	$1.423
Polea dentada (2 unidades)	$7.000	$2.650
Fabricación maqueta (HH)	$45.000	$45.000
Power bank	$16.000	$5.392

Fuente: Elaboración propia.

Total nacional: $171.615

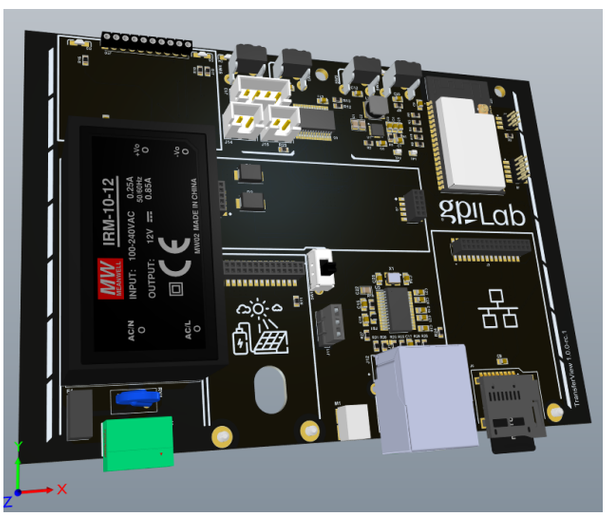
Total internacional: $125.877

Ahorro: $45.738

3.3 MEJORAS

Se enlistan las mejoras que se pueden realizar con el proyecto, dependiendo del presupuesto y la necesidad del cliente.

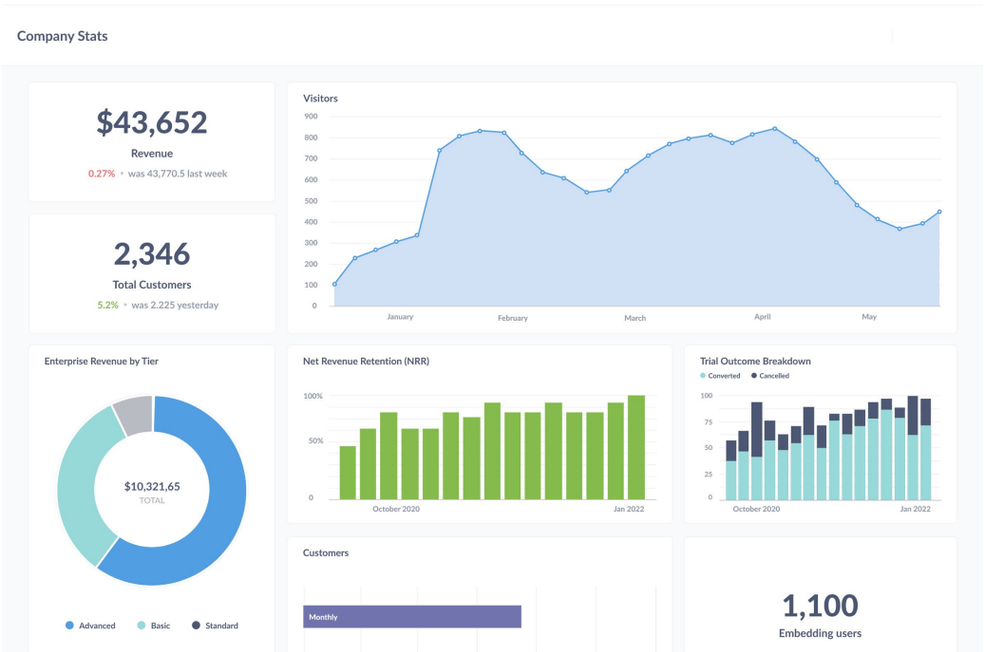
- Comprar la mayoría de componentes en el extranjero: Ayudaría a reducir costos, más aún si los componentes se traen en lotes grandes. Los precios en Aliexpress son bastante reducidos y el tiempo de envío a Chile también ha mejorado.
- Diseñar, fabricar y montar una PCB para cada maqueta, compatible con el módulo Arduino UNO, ESP32 DEVKIT V1 y salidas / entradas industriales para comunicación con PLCs, permitiría replicar las experiencias en distintos microcontroladores, logrando un mayor manejo de herramientas. Un ejemplo de PCB propia se muestra en la Figura 3-8. Estas placas electrónicas pueden ser diseñadas para cada cliente e incluir sus logos y marcas.



Fuente: Elaboración propia.

Figura 3-8 Ejemplo PCB particular.

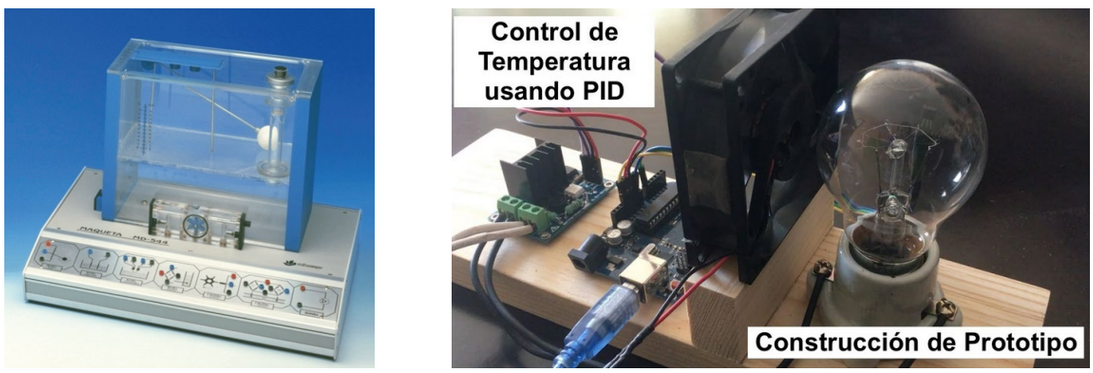
- Aumentar la robustez de los diseños 3D, necesario para demostrar o simular entorno industrial.
- Implementar un filtro pasa bajo de segundo orden y cambiar el sensor de distancia VL5X1L para la maqueta “Barra y bola”.
- Agregar un Dashboard en cada maqueta para la fácil manipulación y visualización de datos, como se muestra en la Figura 3-9.



Fuente: https://www.metabase.com/docs/latest/dashboards/start

Figura 3-9 Ejemplo de Dashboard.

- Probar otros métodos de control en las maquetas.
- Obtener los modelos matemáticos de los sistemas.
- Diseñar y fabricar otras maquetas, para control de temperatura, control de nivel, cintas transportadoras, entre otros: Ayudaría en contar con la experiencia de generar modelos de plantas y/o aplicar métodos de control distintos en cada una de las maquetas. Se muestran maquetas de ejemplo en la Figura 3-10.

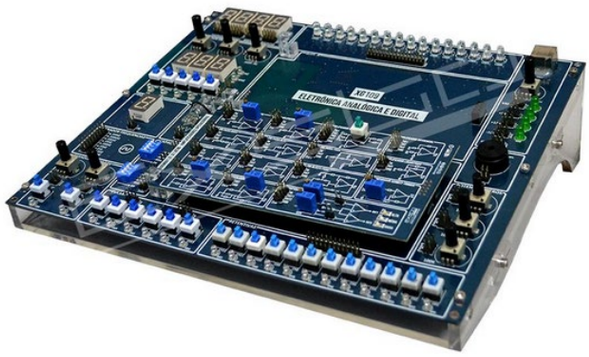


Fuente: https://www.alecop.com/equipamiento-didactico/areas/captadores-regulacion-de-procesos-y-automatas-programables/estudio-de-los-captadores-de-magnitudes-fisicas-serie-540/ y https://www.youtube.com/watch?v=1rXs4XCuros

Figura 3-10 Maquetas de ejemplo, control de nivel a la izquierda y control de temperatura a la derecha.

- Implementar dentro de las maletas, laboratorios completos de asignaturas, por ejemplo, microcontroladores, sistemas lineales, entendiéndose que se invierte tiempo

implementando la electrónica y perdiendo el foco principal de los cursos. En la Figura 3-11 se muestra un ejemplo del laboratorio.



Fuente: https://zamtsu.com/producto/kit-didactico-de-electronica-analogica-digital-con-sensores-xg204-exsto/

Figura 3-11 Ejemplo maqueta didáctica de experiencias.

CONCLUSIONES

Existen muchos tipos de maquetas para el control de procesos de fácil fabricación e implementación, que, además, utilizando softwares para modelado 3D como Autodesk Inventor con licencia estudiante se pueden lograr diseños propios y económicos.

En la actualidad existe una variedad de tutoriales en Internet para la implementación de sistemas ya listos, para fabricación con impresoras 3D entre otro, que permiten la rápida creación de material de estudio o de enseñanza.

Se entiende que uno de los bloqueos más importante para no utilizar el material existente es debido al tiempo que consume y la necesidad de lograr maquetas personalizadas. Para esto se puede externalizar el trabajo y lograr distintas maquetas para la enseñanza.

El desarrollo de maleta con las maquetas didácticas se cumplió de forma completa y las partes y piezas en formato STL se encuentran listas para ser replicadas en cualquier impresora 3D, también se demostró la transportabilidad, bajo costo y funcionalidad.

BIBLIOGRAFÍA

Ramirez, M., (2022). CREACIÓN DE HARDWARE DE DESARROLLO OPEN SOURCE Y APLICACIÓN DE CONTROL PID DISCRETO [Ingeniero de Ejecución en CONTROL E INSTRUMENTACIÓN INDUSTRIAL, Universidad Técnica Federico Santa María].

OGATA, Katsuhiko. Ingeniería de Control Moderna. 5ª ed. Prentice-Hall, 2010. 904 p. ISBN 9788483226605.

MOHAMMAD, Nabil. Digital Feedback Control Tutorial with Arduino [en línea]. <https://www.udemy.com/course/digital-feedback-control-tutorial-with-arduino>.

(S/f). Researchgate.net. Recuperado el 8 de junio de 2023, de https://www.researchgate.net/publication/353754375_SG90_Servo_Characterization

Obando, O. y Romero, H. y (2010). *DESARROLLO DE UN SISTEMA BALL AND BEAM, PARA IMPLEMENTAR ESTRATEGIAS DE CONTROL MEDIANTE LABVIEW* [Especialista en Control e Instrumentación Industrial, Universidad Pontificia Bolivariana Seccional Bucaramamga].

Jover, G., (2019). Diseño, implementación y control de un prototipo de Ball & Beam [Máster en Ingeniería Mecatrónica, Universitat Politecnica de Valencia].

4 ANEXOS

4.1 FIRMWARE CONTROLADOR PID EN C

```
void main() {
 // ? Parámetros del controlador
 float T = 0.1;  // Tiempo de muestreo en segundos
 float Kp = 1;   // Ganancia proporcional
 float Ki = 1;   // Ganancia integral
 float Kd = 1;   // Ganancia derivativa
 float N = 1;    // Coeficiente del filtro derivativo

 // ? Variables para almacenar muestras anteriores
 float error_anterior = 0;       // Almacena error anterior
 float integral_anterior = 0;    // Almacena parte integral anterior
 float derivativo_anterior = 0;  // Almacena parte derivativa anterior

 // ? Saturación
 float valor_maximo = 100;
 float valor_minimo = 0;

 // Referencia de ejemplo, 50 mm.
 float referencia = 50;

 while(1) {
  // Calcula el error E(s). La función leerSensor() es un ejemplo de un sensor analógico o digital
  // que mide la señal de salida Y(s) de la planta.
  float error = referencia - leerSensor();

  // Calcula la parte o acción proporcional.
  float proporcional = Kp * error;

  // Calcula la parte o acción integral.
  float integral = Ki * 0.5 * T * (error + error_anterior) + integral_anterior;

  // Antiwindup
  // Si el valor de la parte integral es mayor al valor máximo de la salida, la parte integral
```

```
  // es seteada al valor máximo de salida.
  // Si el valor de la parte integral es menor al valor mínimo de la salida, la parte integral
  // es seteada al valor minimo de la salida.
  if(integral > valor_maximo) integral = valor_maximo;
  else if(integral < valor_minimo) integral = valor_minimo;

  // Calcula la parte o acción derivativa.
  float derivativo = Kd * N * (error - error_anterior) + (1.0 - N * T) * derivativo_anterior;

  // Calcula la salida del controlador U(s).
  float output = proporcional + integral + derivativo;

  // Saturación
  // Si el valor de salida es mayor al valor máximo, la salida es seteada al valor_máximo.
  // Si el valor de salida es menor al valor mínimo, la salida es seteada al valor mínimo.
  if(output > valor_maximo) output = valor_maximo;
  else if(output < valor_minimo) output = valor_minimo;

  // Aplica la actuación a la planta, sea una salida análoga o digital.
  escribirSalida(output);

  // Guarda muestras anteriores para la siguiente iteración.
  integral_anterior = integral;
  derivativo_anterior = derivativo;
  error_anterior = error;

  // Aplica retraso de 100 ms, para la siguiente iteración. Este valor de retraso debe ser el
  // mismo con el que se sintoniza el controlador PID.
  delay_ms(100);
 }
}
```

4.2 FIRMWARE PLANTA RC

```
#include <Arduino.h>
#include <Wire.h>

// ? Pines del ESP32
const uint8_t PIN_DAC = 25;
const uint8_t PIN_ADC = 27;

// * Máximo y mínimo
// ? Carga condensador
const float MAX_OUT = 3.3;  // Valor máximo del DAC
const float MIN_OUT = 0.0;  // Valor mínimo del DAC

// ? Flags control
// Bandera áara activar, desactivar el controlador
bool enable_controller = false;

// ? Setpoint
float setpoint = 1;

// ? Periodo de muestreo
float T   = 0.1;

// * Variables para controlador PID
float Kp  = 6.9157;
float Ki  = 4.2425;
float Kd  = -1.0478;
float N   = 1.8237;

struct VariablesPID {
  float last_error;         // Error anterior
  float last_integral;      // Valor integral anterior
  float last_derivative;    // Valor derivativo anterior
```

```
  float error;              // Valor actual del error
  float proportional;       // Valor actual de la parte proporcional
  float integral;           // Valor actual de la parte integral
  float derivative;         // Valor actual de la parte derviativa
  float output;             // Valor actual de la salida
} pid;

uint32_t last_time = 0;

//* Prototipo de funciones
// Controlador PID
float PIDController(float in_value);
// Lectura del terminal serial para realizar cambio de setpoint,
// Habilitar, deshabilitar el controlador, entre otros.
void readConsole();
// -------------------------

void setup() {
  // Inicia terminal serial
  Serial.begin(115200);

  // Inicia estructura con todsos los su valores en 0
  pid = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

  // Inicia la salida del DAC en 0
  dacWrite(PIN_DAC, 0);
}

void loop() {
  readConsole();

  // ? Carga condensador
  if(enable_controller) {
    if(millis() - last_time >= 100) {
      last_time = millis();
```

```
    float adc_value = analogRead(PIN_ADC);
    adc_value = adc_value * 3.3 / 4095; // Convierte 2^12 a voltaje

    float dac_value = PIDController(adc_value);
    dac_value = dac_value * 255 / 3.3; // Convierte voltaje a 2^8

    dacWrite(PIN_DAC, dac_value);
   }
  }
}

void readConsole() {

  if (Serial.available()) {
   char c = Serial.read();

   switch(c) {
    case 'c': // habilita / deshabilita PID
    {
     enable_controller = !enable_controller;
     // Serial.printf("Controlador PID %s\n", enable_controller ? "habilitado" : "deshabilitado");
    } break;

    case 's': // Setea setpoint
    {
     setpoint = Serial.readStringUntil(',').toFloat();
     // Serial.printf("Setpoint puesto a %.2f\n", setpoint);
    } break;

    case 'd': // Setea valor en dac
    {
     if(enable_controller) {
      // Serial.println("Controlador habilitado! no puedes modificar esto");
       break;
     }
```

```
      float value = Serial.readStringUntil(',').toFloat();
      float dac_value = value * 255 / 3.3;

      dacWrite(PIN_DAC, (uint8_t) dac_value);
     } break;
    }
  }
}

// PIN void PIDController()
float PIDController(float in_value) {
   pid.error = setpoint - in_value; // Calcula el error
   pid.proportional = Kp * pid.error;     // Caclula la parte proporcional
   // Calcula la parte integral si no está saturada la salida.
   pid.integral = Ki * 0.5 * T * (pid.error + pid.last_error) + pid.last_integral;
   // Antiwindup
   if(pid.integral > MAX_OUT) pid.integral = MAX_OUT;
   else if(pid.integral < MIN_OUT) pid.integral = MIN_OUT;
   // Calcula la parte derivativa del controlador
   pid.derivative = N * Kd * (pid.error - pid.last_error) + (1.0 - N * T) * pid.last_derivative;
   // Suma todas las partes para obtener la salida
   pid.output = pid.proportional + pid.integral + pid.derivative;

   if(pid.output > MAX_OUT) pid.output = MAX_OUT;
   else if(pid.output < MIN_OUT) pid.output = MIN_OUT;

   Serial.printf("%.3f,%.3f,%.3f,%.3f,%.3f,%.3f\r" , pid.proportional, pid.integral
                                    , pid.derivative, in_value
                                    , pid.output, pid.error);

   pid.last_integral   = pid.integral;
   pid.last_derivative = pid.derivative;
   pid.last_error      = pid.error;
```

```
    return pid.output;
}
/* ------------------------------------------------------------------------------------------ */
```

4.3 FIRMWARE MAQUETA BARRA Y BOLA

```
#include <Arduino.h>
#include <Wire.h>

#include "ESP32_Servo.h"
#include "VL53L0X.h"

// * Objetos
Servo servo;    // Objeto para manejo de servomotor
VL53L0X laser;  // Objeto para manejo de sensor láser

// * Pines
const uint8_t PIN_SDA = 21;
const uint8_t PIN_SCL = 22;
const uint8_t PIN_SERVO = 27;

// * Valores del servomotor
const float MAX_OUT = 90;
const float MIN_OUT = -90;
static const uint8_t SERVO_INITIAL_ANGLE = 90;

// * Calibración de distancia medición
// (Calibración calculada por dos puntos medidos)
static const float M = 0.8772;
static const float B = 0;

// * Flags control
bool enable_controller = false;

// * Datalogger
bool enable_datalogger = false;
uint32_t counter = 0;
uint8_t angle = SERVO_INITIAL_ANGLE;
```

```
// * DataloggerOsc
bool enable_datalogger_osc = false;
uint32_t counter_osc = 0;
uint8_t angle_osc = SERVO_INITIAL_ANGLE;
uint8_t angle_osc_step = 0;
uint8_t factor_time = 0;

// * Variables para controlador PID
float period = 0.1;

struct VariablesPID {
  float Kp;              // Valor de sintonización proporcional
  float Ki;              // Valor de sintonización integral
  float Kd;              // Valor de sintonización derivativa
  float N_coef;          // Valor coeficiente filtro derivativo

  float setpoint;        // Valor del setpoint
  float error;           // Valor actual del error

  float proportional;    // Valor actual de la parte proporcional
  float integral;        // Valor actual de la parte integral
  float derivative;      // Valor actual de la parte derviativa
  float output;          // Valor actual de la salida

  float last_error;      // Error anterior
  float last_integral;   // Valor integral anterior
  float last_derivative; // Valor derivativo anterior
} pid;

uint32_t last_time = 0;

// * Prototipo de funciones
float PIDController(float in_value);
void readConsole();
void datalogger();
```

```
void dataloggerOsc();

// PIN setup()
void setup() {
  // Inicializa puertos seriales
  Serial.begin(115200);
  Wire.begin(PIN_SDA, PIN_SCL);

  // Inicializa valores PID en 0
  pid = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
       , 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

  pid.Kp = 0.84198;
  pid.Ki = 0.47872;
  pid.Kd = 0.33821;
  pid.N_coef = 7.9883;
  pid.setpoint = 150.0;

  // Inicia servomotor
  // Configura el pin del servo y los tiempos de trabajo para este
  servo.attach(PIN_SERVO, 1000, 2000);
  // Mueve el servo a la posición inicial
  servo.write(SERVO_INITIAL_ANGLE);

  // Inicia sensor de distancia
  laser.setTimeout(250);      // Setea timeout
  bool status = laser.init(); // Intenta iniciar el sensor

  if(status) {
    // Configura parámetros del sensor
    laser.setSignalRateLimit(0.1);
    laser.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 18);
    laser.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, 14);
    laser.setMeasurementTimingBudget(100e3);
  } else {
```

```
    // Si no inicia el sensor, bloquea el código
    while(1);
  }
}

void loop() {
  // Lee los comandos escritos en consola
  readConsole();

  // Activa datalogger
  datalogger();

  // Activa datalogger 2
  dataloggerOsc();

  // Si el controlador está habilitado
  if(enable_controller) {
    // Lee el dato de distancia
    uint16_t distance = laser.readRangeSingleMillimeters();
    // Ajusta la distancia según la calibración
    float calibrated = ((float) (distance) * M) + B;
    // Ingresa el valor medido calibrado al controlador PID
    float servo_pos = PIDController(calibrated);
    // Agrega offset para valores aceptados por el actuador
    servo_pos += 90;
    servo.write(servo_pos);
  }
}

void readConsole() {

  if (Serial.available()) {
    char c = Serial.read();

    switch(c) {
```

```
case 'c': //? habilita / deshabilita PID
{
  enable_controller = !enable_controller;
  if(!enable_controller) servo.write(SERVO_INITIAL_ANGLE);
} break;

case 's': //? Setea setpoint
{
  pid.setpoint = Serial.readStringUntil(',').toFloat();
} break;

case 'P': //? Cambia la posición del servomotor
{
  if(enable_controller) {
    break;
  }

  uint8_t position = Serial.readStringUntil(',').toInt();
  servo.write(position);
} break;

case 'm': //? Mide y muestra la distancia
{
  uint16_t distance = laser.readRangeSingleMillimeters();
  Serial.printf("Distancia: %u\n", distance);
} break;

case 'p': //? Cambia el factor proporcional (Kp)
{
  pid.Kp = Serial.readStringUntil(',').toFloat();
} break;

case 'i': //? Cambia el factor integral (Ki)
{
  pid.Ki = Serial.readStringUntil(',').toFloat();
```

```
} break;

case 'd': //? Cambia el factor derivativo (Kd)
{
  pid.Kd = Serial.readStringUntil(',').toFloat();
} break;

case 'n': //? Cambia el coeficiente del filtro
{
  pid.N_coef = Serial.readStringUntil(',').toFloat();
} break;

case 'D': //? Habilita / deshabilita datalogger
{
  enable_datalogger = !enable_datalogger;

  angle = SERVO_INITIAL_ANGLE;

  if(!enable_controller) {
    servo.write(angle);
    counter = 0;
  }
} break;

case '1':
{
  enable_datalogger_osc = !enable_datalogger_osc;

  angle_osc = SERVO_INITIAL_ANGLE;

  if(!enable_datalogger_osc) {
    servo.write(angle_osc);
    counter_osc = 0;
  }
} break;
```

```
    case '2':
    {
      factor_time = Serial.readStringUntil(',').toInt();
    } break;

    case '3':
    {
      angle_osc_step = Serial.readStringUntil(',').toInt();
    } break;
   }
  }
}

// PIN  PIDController()
float PIDController(float in_value) {
  pid.error = pid.setpoint - in_value; // Calcula el error
  pid.proportional = pid.Kp * pid.error;      // Caclula la parte proporcional

  // Calcula la parte integral si no está saturada la salida.
  pid.integral = pid.Ki * 0.5 * period * (pid.error + pid.last_error) + pid.last_integral;
  // Anti Windup
  if(pid.integral > MAX_OUT) pid.integral = MAX_OUT;
  else if(pid.integral < MIN_OUT) pid.integral = MIN_OUT;

  // Calcula la parte derivativa del controlador
  pid.derivative =  pid.N_coef * pid.Kd * (pid.error - pid.last_error)
                + (1.0 - pid.N_coef * period) * pid.last_derivative;

  // Suma todas las partes para obtener la salida
  pid.output = pid.proportional + pid.integral + pid.derivative;

  if(pid.output > MAX_OUT) pid.output = MAX_OUT;
  else if(pid.output < MIN_OUT) pid.output = MIN_OUT;
```

```
  Serial.printf("%.3f,%.3f,%.3f,%.3f,%.3f,%.3f\r" , pid.proportional
                                    , pid.integral
                                    , pid.derivative
                                    , in_value
                                    , pid.output
                                    , pid.error);

  pid.last_integral   = pid.integral;
  pid.last_derivative = pid.derivative;
  pid.last_error      = pid.error;

  if(pid.error == 0) pid.output = 0;

  return pid.output;
}

void datalogger() {
  if(enable_datalogger) {
    // Lee el dato de distancia
    uint16_t distance = laser.readRangeSingleMillimeters();
    // Ajusta la distancia según la calibración
    float calibrated = ((float) (distance) * M) + B;
    // Aumenta el contador
    counter++;

    // Cuando el contador llegue a 10, significa que habrá transcurrido
    // 1 segundo, por lo que cambiará la posición del servomotor para
    // mover la bola.
    if(counter == 5) {
      angle = SERVO_INITIAL_ANGLE + 20;
      servo.write(angle);
    }

    Serial.printf("%.1f,%u,%.3f\r", counter * 0.1, angle, calibrated);
  }
```

```
}

void dataloggerOsc() {
  if(enable_datalogger_osc) {
    // Lee el dato de distancia
    uint16_t distance = laser.readRangeSingleMillimeters();
    // Ajusta la distancia según la calibración
    float calibrated = ((float) (distance) * M) + B;
    // Aumenta el contador
    counter_osc++;

    // Cuando el contador llegue a 10, significa que habrá transcurrido
    // 1 segundo, por lo que cambiará la posición del servomotor para
    // mover la bola.
    if(counter_osc % factor_time == 0) {
      if(angle_osc >= SERVO_INITIAL_ANGLE) {
        angle_osc = SERVO_INITIAL_ANGLE - angle_osc_step;
        // Serial.printf("1: %u\n", angle_osc);
      } else {
        angle_osc = SERVO_INITIAL_ANGLE + angle_osc_step;
        // Serial.printf("2: %u\n", angle_osc);
      }

      servo.write(angle_osc);
    }

    Serial.printf("%.1f,%u,%.3f\r", counter_osc * 0.1, angle_osc, calibrated);
  }
}

/* ---------------------------------------------------------------------------------- */
```

4.4 FIRMWARE MAQUETA DEMOSTRACIÓN DE CONTROL DE UN SERVOMOTOR

```
//* Librerías
#include <Arduino.h>
#include <Wire.h>

#include "U8g2lib.h"
#include "ADS1X15.h"

//* Enum para identificar los pines usados en placa
enum DemoPins : uint8_t {
  I2C_SDA   = 21,
  I2C_SCL   = 22,
  ANALOG_1  = 33,
  MOTOR_A_1A  = 18,
  MOTOR_A_1B  = 19,
  ADC_ALERT   = 25
};

//* Variable / potenciómetro leído
enum Variable : uint8_t {
  PROPORTIONAL,
  INTEGRAL,
  DERIVATIVE,
  SETPOINT,
  ROTOR,
  MAX
};

//* Creación de objetos
U8G2_SH1106_128X64_NONAME_F_HW_I2C display (U8G2_R0, U8X8_PIN_NONE);
ADS1115 adc(0x48);

//* Variables
struct AnalogInputs {
```

```
  uint16_t raw;
  uint16_t filtered;
  float scaled;
};

AnalogInputs analogs_inputs[MAX];
uint8_t ads_channel = 0;

volatile bool ready = false;

//* Variables del controlador
bool enable_controller = false;

float period = 0.1;

float max_kp = 10.0;
float max_ki = 10.0;
float max_kd = 10.0;

float max_out = 255.0;
float min_out = -255.0;

struct VariablesPID {
  float Kp;               // Valor de sintonización proporcional
  float Ki;               // Valor de sintonización integral
  float Kd;               // Valor de sintonización derivativa
  float N_coef;           // Valor coeficiente filtro derivativo

  float setpoint;         // Valor del setpoint
  float error;            // Valor actual del error

  float proportional;     // Valor actual de la parte proporcional
  float integral;         // Valor actual de la parte integral
  float derivative;       // Valor actual de la parte derviativa
  float output;           // Valor actual de la salida
```

```
  float last_error;          // Error anterior
  float last_integral;       // Valor integral anterior
  float last_derivative;     // Valor derivativo anterior
} pid;

//* Prototipos de funciones
void displayView();
void readAnalogsInputs();
void readConsole();
float PIDController(float in_value);
void IRAM_ATTR adcReady();

// PIN setup()
void setup() {
  // Inicializa puertos de comunicacion
  Wire.begin(I2C_SDA, I2C_SCL);
  Serial.begin(115200);

  // Inicia y limpia pantalla I2C
  display.begin();
  display.clear();

  // Inicia y configura el chip ADS1115
  adc.begin();
  adc.setGain(1);
  adc.setDataRate(4);
  adc.setComparatorThresholdHigh(0x8000);
  adc.setComparatorThresholdLow(0x0000);
  adc.setComparatorQueConvert(0);
  adc.setMode(1);

  // Establece y "apaga" pines de control de motor como salida
  pinMode((uint8_t) DemoPins::MOTOR_A_1A, OUTPUT);
  pinMode((uint8_t) DemoPins::MOTOR_A_1B, OUTPUT);
```

```
  digitalWrite((uint8_t) DemoPins::MOTOR_A_1A, LOW);
  digitalWrite((uint8_t) DemoPins::MOTOR_A_1B, LOW);

  // Establece el pin ADC alert como entrada, con pullup interno
  // y lo configura como interrupción.
  pinMode((uint8_t) DemoPins::ADC_ALERT, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt((uint8_t) DemoPins::ADC_ALERT), adcReady, RISING);

  // Inicializa los valores del controlador PID en 0
  pid = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
  // La constante N para el filtro derivativo se ignora por lo que se setea a 1.0
  pid.N_coef = 1.0;
}

// PIN loop()
void loop() {
  static uint32_t last_time = 0;
  // Lee datos del monitor serial
  readConsole();
  // Maneja la pantalla
  displayView();
  // Lee las entradas análogas
  readAnalogsInputs();

  // Lectura y actuación cada 100 ms
  if(millis() - last_time >= 100) {
    last_time = millis();
    // Como es una maqueta de ejemplo, si Ki = 0, los valores
    // de la integral se reinician.
    if(pid.Ki == 0) {
      pid.integral = 0;
      pid.last_integral = 0;
    }
```

```
    // Lee y escala la señal del potenciómetro conectado al rotor del motor
    analogs_inputs[ROTOR].raw = analogRead(ANALOG_1);
    analogs_inputs[ROTOR].scaled = analogs_inputs[ROTOR].raw * 180.0 / 3380.0;
    // Ingresa el valor leído al controlador PID
    float motor_speed = PIDController(analogs_inputs[ROTOR].scaled);
    // Dependiendo del signo de la velocidad del motor,
    // los pintes de control del motor se apagan o no
    if((int32_t) motor_speed >= 0) {
      analogWrite(MOTOR_A_1A, 0);
      analogWrite(MOTOR_A_1B, motor_speed);
    } else { // Si no, gira a la izquierda y hace positiva la velocidad
      motor_speed *= -1;
      analogWrite(MOTOR_A_1A, motor_speed);
      analogWrite(MOTOR_A_1B, 0);
    }
  }
}

// PIN  PIDController()
float PIDController(float in_value) {
  pid.error = pid.setpoint - in_value; // Calcula el error
  pid.proportional = pid.Kp * pid.error;      // Caclula la parte proporcional

  // Calcula la parte integral si no está saturada la salida.
  pid.integral = pid.Ki * 0.5 * period * (pid.error + pid.last_error) + pid.last_integral;
  // Anti Windup
  if(pid.integral > max_out) pid.integral = max_out;
  else if(pid.integral < min_out) pid.integral = min_out;

  // Calcula la parte derivativa del controlador
  pid.derivative =  pid.N_coef * pid.Kd * (pid.error - pid.last_error)
                + (1.0 - pid.N_coef * period) * pid.last_derivative;

  // Suma todas las partes para obtener la salida
  pid.output = pid.proportional + pid.integral + pid.derivative;
```

```
  if(pid.output > max_out) pid.output = max_out;
  else if(pid.output < min_out) pid.output = min_out;

  Serial.printf("s: %.2f - p: %.2f - i: %.2f - d: %.2f - o: %.2f\n" , pid.setpoint
                                                , pid.proportional
                                                , pid.integral
                                                , pid.derivative
                                                , pid.output);

  pid.last_integral   = pid.integral;
  pid.last_derivative = pid.derivative;
  pid.last_error      = pid.error;

  if(pid.error == 0) pid.output = 0;

  return pid.output;
}

// PIN readConsole()
void readConsole() {
  if(Serial.available()) {
    char c = Serial.read();

    switch(c) {
      case 'p': max_kp = Serial.readStringUntil(',').toFloat(); break;
      case 'i': max_ki = Serial.readStringUntil(',').toFloat(); break;
      case 'd': max_kd = Serial.readStringUntil(',').toFloat(); break;
    }
  }
}

// PIN displayView()
void displayView() {
  static uint32_t last_time = 0;
```

```
  char text_buffer[20];

  if(millis() - last_time >= 100) {
    last_time = millis();
    display.setFont(u8g2_font_6x12_mf);
    display.enableUTF8Print();
    display.setFontMode(0);
    display.clearBuffer();
    sprintf(text_buffer, "P: %.1f", analogs_inputs[0].scaled);
    display.drawStr(0, 12, text_buffer);
    sprintf(text_buffer, "I: %.1f", analogs_inputs[1].scaled);
    display.drawStr(0, 24, text_buffer);
    sprintf(text_buffer, "D: %.1f", analogs_inputs[2].scaled);
    display.drawStr(0, 36, text_buffer);
    display.drawStr(0, 48, "Setpoint");
    sprintf(text_buffer, "%.0f°", analogs_inputs[3].scaled);
    uint8_t x = (display.getStrWidth("Setpoint") - display.getStrWidth(text_buffer)) / 2;
    display.drawUTF8(x, 60, text_buffer);

    display.drawStr(64, 12, "Position");
    display.setFont(u8g2_font_10x20_mf);
    sprintf(text_buffer, "%.0f°", analogs_inputs[4].scaled);
    display.drawUTF8(64, 30, text_buffer);

    display.sendBuffer();
  }
}

// PIN readAnalogsInputs()
void readAnalogsInputs() {
  static uint32_t last_time = 0;
  static bool firts = true;

  if(millis() - last_time >= 100) {
    if(firts) {
```

```
 ads_channel = 0;
 adc.requestADC(ads_channel);
 firts = false;
}

if(ready) {
 int16_t temp = adc.getValue();
 if(temp < 0) temp = 0;
 analogs_inputs[ads_channel].raw = temp;

 switch(ads_channel) {
  case PROPORTIONAL:
   analogs_inputs[PROPORTIONAL].scaled = temp * max_kp / 26363.0; // P
   break;
  case INTEGRAL:
   analogs_inputs[INTEGRAL].scaled = temp * max_ki / 26363.0;    // I
   break;
  case DERIVATIVE:
   analogs_inputs[DERIVATIVE].scaled = temp * max_kd / 26363.0;  // D
   break;
  case SETPOINT:
   analogs_inputs[SETPOINT].scaled = temp * 180.0 / 26363.0; // Setpoint
   break;
 }

 ads_channel++;

 if(ads_channel > 3) {
  last_time = millis();
  ads_channel = 0;
  pid.Kp       = analogs_inputs[PROPORTIONAL].scaled;
  pid.Ki       = analogs_inputs[INTEGRAL].scaled;
  pid.Kd       = analogs_inputs[DERIVATIVE].scaled;
  pid.setpoint  = analogs_inputs[SETPOINT].scaled;
 }
```

```
    adc.requestADC(ads_channel);
    ready = false;
   }
  }
}

// PIN adcReady()
void IRAM_ATTR adcReady() { ready = true; }
```