UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño de un laboratorio basado en sistemas embebidos para robótica educativa

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Adrian Jonathan Calzada Maldonado

DIRECTORA DE TESIS

Dra. Josefina Bárcenas López

Ciudad Universitaria, Cd. Mx., 2025

Agradecimientos

A la Dra. Josefina Bárcenas López, por todo su apoyo, la confianza, paciencia y oportunidad de desarrollar este proyecto bajo su supervisión.

Al M.I. Víctor Hugo García Ortega por todo su apoyo, sus enseñanzas y asesoría durante el desarrollo del proyecto.

Al Dr. Enrique Ruiz-Velasco Sánchez por permitirme incursionar en el campo de la robótica pedagógica y desarrollar mi proyecto de tesis.

Al Instituto de Ciencias Aplicadas y Tecnología (ICAT), por todo el apoyo y la oportunidad de desarrollar mi proyecto de tesis.

A la Dirección General de Asuntos del Personal Académico (DGAPA), por su apoyo en el proyecto de Laboratorio de Educatrónica PAPIIT IT400222 en el cual se desarrolló este proyecto de tesis.

A la UNAM por la oportunidad de poder formarme en esta gran institución, por todos los aprendizajes, apoyos y oportunidades que me brindaron a lo largo de mi formación académica.

Agradecimientos Personales

A mis padres Silvia Maldonado Hernández y Jesús Ismael Calzada Domínguez, por todo su apoyo, por nunca dejarme solo y motivarme día con día a salir adelante.

A mi pareja Bibiana Cambrón de la Cruz, por todo su apoyo, motivación y confianza para poder seguir adelante y motivarme cada día en seguirme superando.

A mi hermano Brayan Ismael Calzada Maldonado, por todo su apoyo y confianza.

A mis primos Miguel Ángel Mendoza Maldonado, Uriel Calzada Mendoza y Leonardo Felipe Calzada Maldonado, por todo su apoyo tanto a mi como a mi familia y nunca dejarnos en los momentos más difíciles.

A mi tío Juan Francisco Calzada Domínguez, por todo el apoyo tanto a mi como a mi familia, por sus consejos y siempre estar en los momentos más difíciles.

A mis amigos Sinuhe Mazuti Osorio Rivero, Edgar Hernández Hernández, Héctor Enrique Zamora Cerriteño, por todo el apoyo que me brindaron en los momentos más difíciles y todos los buenos momentos, “Las risas no faltaron”.

A mi familia y amigos, por todo el apoyo, por todos los ánimos y confianza para poder concluir y superar esta etapa de mi vida.

Índice general

Contenido

Índice de figuras

Índice de tablas

Introducción

El presente trabajo de tesis busca ofrecer de manera detallada y secuencial el desarrollo de un prototipo de Laboratorio para Robótica Educativa, que cuente con todos los elementos necesarios en aspectos de *hardware* y *software* para toda aquella persona interesada en aprender conocimientos básicos de programación mediante el uso de robots educativos.

En el primer capítulo se abordarán tanto la problemática que dio origen al desarrollo de este proyecto, como la propuesta y todos los elementos necesarios para validar el desarrollo de este proyecto.

El segundo capítulo partirá realizando una investigación de cada uno de los componentes necesarios para poder realizar el diseño tanto de una aplicación de escritorio, como de una aplicación para un Sistema en Chip, se presentarán algunos ejemplos de diferentes plataformas dedicadas a la robótica educativa, y de algunas aplicaciones de escritorio.

En el tercer capítulo se realizará el diseño tanto de la aplicación de escritorio como la del Sistema en *Chip,* se realizará la elección de los materiales más fáciles de conseguir y económicos para el público en general para el desarrollo del robot.

En el cuarto capítulo se realizará la implementación de ambas aplicaciones, la de escritorio y la del Sistema en Chip, se realizarán las conexiones necesarias para poder controlar el robot educativo desde la aplicación de escritorio.

Finalmente, en el capítulo cinco se pondrá a prueba el prototipo del Laboratorio para Robótica Educativo, se harán pruebas con usuarios finales para comprobar que se cumplen los objetivos del proyecto, en base a estas pruebas se presentarán las conclusiones, así como propuestas en las cuales el proyecto puede mejorar tanto en *Hardware* como *Software.*

Capítulo 1. Contextualización del Laboratorio de Robótica Educativa

1.1 Problemática

En la actualidad existen una gran cantidad de plataformas y productos que se enfocan a la robótica educativa, sin embargo, esto no significa que todas las personas puedan tener acceso a estos recursos, la mayoría de estas plataformas o productos suelen tener costos muy altos o necesitan que las personas obtengan diferentes materiales para poder hacer uso de estas lo cual muchas veces puede complicar el acceso pues estos materiales son difíciles de conseguir o son muy costosos.

Estas plataformas no solo son difíciles de acceder para muchas personas, sino que también enseñan a programar utilizando métodos de programación basados en las tecnologías que ellos utilizan, al no enseñar a programar utilizando un lenguaje natural complica que las personas puedan adaptarse a otras metodologías de programación, a diferencia de cómo lo harían al aprender a programar utilizando lenguaje natural.

1.2 Justificación

La robótica educativa se ha convertido en una herramienta de vital importancia en la formación de competencias en ciencia, tecnología, ingeniería y matemáticas. Sin embargo, a pesar del creciente desarrollo de estas plataformas y productos dedicados a este fin, la accesibilidad a estos recursos sigue siendo un desafío significativo para muchas personas. El problema dentro de estas plataformas es que la mayoría tiene costos muy elevados o requieren de materiales específicos, que son difíciles de conseguir. Debido a esto existe una gran exclusión de un amplio sector de la población en la obtención de los beneficios de la educación en robótica.

Además de los obstáculos financieros, existe una limitación en cuanto a enseñanza en las plataformas actuales, ya que estas plataformas enseñan programación mediante métodos y lenguajes basados en sus tecnologías, la falta de plataformas que utilicen un lenguaje natural para la programación limita más el acceso, principalmente para aquellas personas que no tienen conocimientos previos en programación.

La propuesta de desarrollar un prototipo de laboratorio educativo utilizando un sistema embebido para la manipulación de un robot educativo aborda estas problemáticas de manera integral. La justificación de esta propuesta se basa en los siguientes puntos clave:

- **Accesibilidad Económica y Material:** Al desarrollar un sistema embebido de bajo costo que utilice materiales accesibles, se puede reducir la dificultad de acceso que limita la participación de una gran cantidad de personas en la robótica educativa.

- **Uso de Lenguaje Natural:** La implementación de una aplicación de escritorio que permite a los usuarios interactuar con el robot educativo mediante instrucciones en lenguaje natural representa un avance importante en la pedagogía de la programación. Este método reduce la curva de aprendizaje y hace que la programación sea más intuitiva y accesible, especialmente para principiantes.

- **Innovación y Aprendizaje Práctico:** El prototipo propuesto incluye la manipulación de un elevador de 7 niveles controlado mediante un Sistema en Chip (SoC), lo cual proporciona una experiencia de aprendizaje práctica y tangible. Los estudiantes pueden ver directamente cómo sus instrucciones en lenguaje natural se traducen en acciones físicas, lo que refuerza la comprensión de los principios de programación y robótica.

1.3 Propuesta

Se propone el desarrollo de un prototipo de Laboratorio educativo con el cual se brindará la oportunidad de aprender los conocimientos de base para programar mediante lenguaje natural (lengua materna) y la manipulación de un robot educativo, dicha propuesta consiste en desarrollar los siguientes componentes.

- Desarrollo de una aplicación de escritorio con la cual el usuario podrá realizar la creación de sus programas, compilarlos, simular y ejecutarlos.

- Desarrollo de una aplicación para el Sistema en Chip *(SoC)* con la cual se puedan interpretar los comandos enviados desde la aplicación de escritorio y en base a dichos comandos realizar la manipulación del robot educativo.

- Desarrollo de un lenguaje de programación basado en lenguaje natural para realizar la codificación de los programas para manipular el robot educativo.

- Adaptación de un robot elevador de 7 niveles para realizar la manipulación de éste mediante un Sistema en Chip (*SoC*).

1.4 Objetivo general

Implementar un prototipo de laboratorio educativo, mediante el desarrollo de un sistema embebido para el aprendizaje de las estructuras básicas de programación utilizando un robot pedagógico.

1.5 Objetivos específicos

- Implementar un sistema embebido usando tecnologías de Sistemas en Chip (SoC) para la manipulación de un robot educativo.

- Implementar un lenguaje para el aprendizaje de las estructuras básicas de programación usando un conjunto de instrucciones basadas en lenguaje natural.

- Desarrollar una interfaz gráfica para la interconexión y manipulación del sistema embebido y el robot educativo utilizando lenguaje de programación Python.

1.6 Antecedentes

La robótica educativa se originó en los años 60 con científicos del *MIT(Massachusetts Institute of Technology,* por sus siglas en inglés*),* incluyendo a Wally Feurzeig, Seymour Papert, Cynthia Solomon y Daniel Bobrow. Su objetivo era enseñar a los niños a interactuar y programar robots. Más tarde, se estableció una colaboración con *LEGO* para desarrollar robots educativos.

Seymour Papert es considerado como el padre de la robótica educativa. Promovió el aprendizaje construccionista, que enfatiza el aprendizaje activo a través de la creación de modelos (prototipos tecnológicos). En 1968, publicó *"Mindstorms: Children, Computers and Powerful Ideas",* destacando la importancia de la robótica en la educación.

En 1967, Freurzeig, Papert y Solomon crearon el lenguaje de programación *LOGO*, diseñado para enseñar programación a través de una tortuga robótica.

Papert desarrolló el robot tortuga en 1969, mejorando un diseño anterior de William Gray Walter. Estos robots fueron los primeros dispositivos educativos STEM. En 1980, se integró el lenguaje LOGO en los robots tortuga, permitiendo a los estudiantes programar movimientos y dibujos.

En los años 90, la robótica educativa se expandió significativamente, con la creación de dispositivos específicos y la implementación de talleres y actividades en Europa para la educación primaria.

(Robótica, E.; 25 febrero de 2023; Historia y evolución de la robótica educativa.)

Después de eso conforme han ido avanzando los años, se han desarrollado gran cantidad de plataformas y trabajos enfocados a la robótica educativa, algunos de esos trabajos se mencionan a continuación.

- *LEGO Mindstorms Fix the factory.* Es un juego para dispositivos móviles que forma parte del mundo de *LEGO Mindstorms*, es una aplicación diseñada para enseñar conceptos básicos de programación y lógica a través de la resolución de desafiantes rompecabezas y puzzles.

  La aplicación dispone de una interfaz gráfica de programación en la que se pueden arrastrar y soltar bloques de código los cuales representan acciones y comandos que se utilizan para guiar a los robots y resolver los desafíos en cada nivel. (*Mindstorms fix the factory;* (2016, 8 junio))

- *mBot*: Es un *kit* de robótica educativa diseñado para enseñar a los estudiantes los fundamentos básicos de la programación y la robótica, este *kit* incluye bloques de construcción de plástico y componentes electrónicos que los estudiantes pueden utilizar para construir un robot. Este robot se puede controlar mediante un microcontrolador programable y un módulo inalámbrico que permite a los estudiantes programar y controlar el robot a través de una computadora o un dispositivo móvil. (Dani; 31 mayo de 2021; Robótica educativa 45 entornos virtuales para programar robots emulados.)

- Mejía Perales Marisol Itzel (2018) en su tesis titulada “La robótica pedagógica en la enseñanza de las ciencias y la tecnología” se centra en el uso de la robótica como herramienta educativa para mejorar la enseñanza y el aprendizaje en las áreas de ciencias y tecnología. La investigación explora cómo la robótica pedagógica puede fomentar un aprendizaje activo y constructivista, permitiendo a los estudiantes interactuar con la tecnología de una manera práctica y motivadora.
  Esté trabajo destaca la importancia de la robótica en el desarrollo de habilidades cognitivas, pensamiento crítico y resolución de problemas, aspectos fundamentales en la educación moderna. Además, analiza como la integración de la robótica en el currículo escolar puede transformar la manera en que los estudiantes abordan las ciencias y la tecnología, promoviendo un aprendizaje más profundo y significativo a través de la experimentación y la manipulación directa de robots. (Mejía Perales Marisol Itzel; 2018; La robótica pedagógica en la enseñanza de las ciencias y la tecnología)

- Gabriela Diego Hernández (2010) en su tesis titulada “Desarrollo de escenarios virtuales para *software* de robótica educativa en 3D”, propone el diseño de un sistema para escenarios virtuales para *software* de robótica educativa en 3d. El objetivo de esta tesis es desarrollar un entorno virtual en el que se pueda comprender el comportamiento de diferentes robots educativos por medio de la simulación en 3D siendo esta de bajo costo y fácil de entender para todos los usuarios.

  El objetivo de este sistema está enfocado en que escuelas o instituciones de bajos recursos puedan interactuar con robots educativos mediante el método de simulación de diferentes robots educativos. (Gabriela Diego Hernández; febrero 2017; Desarrollo de escenarios virtuales para *software* de robótica educativa en 3D)

- Uso de la robótica educativa como herramienta en los procesos de enseñanza.

  El artículo Uso de la robótica educativa como herramienta en los procesos de enseñanza publicado por María Luisa Pinto Salamanca, Nelson Barrera Lombana y Wilson Javier Pérez Holguín (2010) propone el uso de un robot de la plataforma *Lego* para la enseñanza con niños de preescolar y primaria, el *kit* utilizado para este proyecto fue el *Lego Mindstorms NXT* en conjunto con una plataforma robótica educativa “*AMIBOT*” desarrollada para el control del robot. El objetivo de este robot fue ayudar a los niños por medio de su programación a reforzar los conocimientos sobre números, colores, e incluso sobre geometría.

  Este robot se puede programar para realizar tareas las cuales obedecen a una secuencia numérica, el robot se puede mover de diferente forma dependiendo del color de la superficie sobre la que se está desplazando y por último puede dibujar figuras geométricas de diferente forma y tamaño dependiendo del uso que se le quiera dar. (Pinto-Salamanca, M. L., Barrera-Lombana, N., & Pérez-Holguín, W. J.; 1 julio de 2010; Uso de la robótica educativa como herramienta en los procesos de enseñanza.)

- La robótica educativa, una herramienta para la enseñanza-aprendizaje de las ciencias y las tecnologías.

  Este artículo publicado por (Moreno, et al., 2012) hace mención del proyecto implementado en Panamá en el cual se busca demostrar la importancia de la robótica educativa para facilitar y motivar la enseñanza y aprendizaje de las ciencias y tecnologías. Para este proyecto se utilizaron *kits* de *lego*

Mindstorms con profesores y estudiantes de nivel secundaria con la finalidad de brindar conocimientos de electrónica y programación a los participantes.

Antes de iniciar y al finalizar el proyecto se realizaron encuestas a todos los participantes con la finalidad de demostrar que el uso de la robótica en temas de educación puede ser beneficioso, los resultados obtenidos fue un aumento de seguridad de los participantes en conocimientos lógico-matemático, inteligencia emocional, interpersonal e intrapersonal, así como en la gestión de la información. (Moreno, Et al.; 2012; *LA ROBÓTICA EDUCATIVA, UNA HERRAMIENTA PARA LA ENSEÑANZA-APRENDIZAJE DE LAS CIENCIAS y LAS TECNOLOGÍAS*.)

1.7 Alcances

El proyecto consiste en el desarrollo de un prototipo de un laboratorio para robótica educativa, el cual solo podrá realizar la manipulación de un tipo de robot educativo, cuenta con una cantidad limitada de instrucciones básicas de programación, al ser un prototipo se pueden realizar mejoras buscando tener un funcionamiento óptimo del laboratorio, entre estas mejoras se encuentran las siguientes opciones:

- ➢ Aumentar la cantidad de instrucciones que el compilador es capaz de recibir y ejecutar.

- ➢ Agregar la capacidad de manipular más tipos de robots educativos tanto en simulación como en control mediante el SoC.

- ➢ Modificar la interfaz de la aplicación de escritorio para hacerla más amigable e intuitiva para los usuarios.

Capítulo 2. Fundamentos de Programación y Sistemas Embebidos

2.1 Lenguajes de programación

2.1.1 ¿Qué es un lenguaje de programación?

Un lenguaje de programación es una herramienta la cual se conforma de una serie de símbolos y reglas de sintaxis y semántica las cuales nos permiten crear conjuntos de instrucciones a través de las cuales los humanos pueden interactuar con las computadoras.

Los lenguajes de programación son utilizados para diseñar e implementar programas para poder controlar el funcionamiento de los dispositivos físicos y lógicos de una computadora.

Los lenguajes de programación se pueden clasificar dependiendo de su finalidad, las principales clasificaciones de estos son:

- Lenguajes de programación de bajo nivel. Son lenguajes de programación que están más cerca del lenguaje máquina y del *hardware* de una computadora, están diseñados para tener un control directo sobre los recursos del sistema y ofrecer un mayor rendimiento y mejor uso de la memoria de la computadora para aplicaciones que requieren de un rendimiento máximo tales como sistemas operativos, controladores de dispositivos o *software* embebido.
- Lenguajes de programación de alto nivel. Son lenguajes de programación que están diseñados para ser mucho más comprensibles y legibles para los seres humanos, se caracterizan por utilizar un tipo de sintaxis más parecida al lenguaje natural con la finalidad de facilitar a los programadores el escribir y comprender el código, así como poder desarrollar aplicaciones de manera más eficiente.
- Lenguajes de programación orientados a objetos. Los lenguajes de programación orientada a objetos utilizan un paradigma de programación que organiza el código en torno a objetos los cuales son entidades que representan conceptos del mundo real y que tienen propiedades y comportamientos asociados. Dentro de este paradigma los objetos son la unidad fundamental de programación y se utilizan para modelar y manipular los datos.

- Lenguajes de programación funcionales. Son lenguajes de programación que utilizan un paradigma el cual se enfoca en tratar la programación como una evaluación de funciones matemáticas y en evitar el cambio de estado y mutabilidad de los datos, dentro de este tipo de lenguajes las funciones se pueden pasar como argumentos a otras funciones, devolverse como resultados y almacenarlas dentro de variables. Estos tipos de lenguaje se utilizan principalmente para el procesamiento de datos, la programación concurrente y la programación de sistemas distribuidos.
- Lenguajes de programación de *scripting*. Son lenguajes de programación que tienen como finalidad escribir *scripts* que son programas para automatizar tareas específicas o realizar diferentes acciones dentro de un entorno determinado. Los *scripts* se interpretan en tiempo de ejecución lo cual facilita su ejecución sin la necesidad de compilar, son comúnmente utilizados en los ámbitos de automatización, administración de sistemas, procesamiento de datos y desarrollo *web*, entre otros.
- Lenguajes de programación *web*. Son lenguajes de programación que son específicamente usados para el desarrollo de aplicaciones y sitios *web*, se centran en permitir la creación de contenido dinámico, la interacción con el usuario y la manipulación de datos en el entorno *web*. Dentro del desarrollo *web* es común utilizar combinación de diferentes lenguajes aprovechando sus diferentes ventajas para lograr funcionalidades más completas y eficientes dentro de las aplicaciones o páginas *web*.
- Lenguajes de programación de bases de datos. Son lenguajes de programación diseñados específicamente para trabajar con sistemas de gestión de bases de datos permitiendo a los desarrolladores y administradores de bases de datos manipular los datos almacenados mediante el uso de un conjunto de comandos para realizar operaciones de consulta, inserción, modificación y eliminación de datos.

2.1.2 Importancia de los lenguajes de programación

El continuo desarrollo de los lenguajes de programación es de vital importancia debido a que son la herramienta fundamental para escribir *software*, sin ellos no se podría disfrutar los constantes avances tecnológicos y la amplia variedad de aplicaciones y servicios informáticos que se utilizan en la vida cotidiana.

Los lenguajes de programación son fundamentales debido a su gran importancia y utilidad en diferentes áreas tales como:

- Desarrollo de *software*.
- Automatización de tareas y procesos.

- Resolución de problemas.

- Flexibilidad y personalización en el desarrollo de *software*.

- Innovación y avance tecnológico.

Aprender a programar se ha convertido en una necesidad hoy día, conocer sobre programación puede ser una oportunidad no solo para abrir puertas a nuevos empleos, sino que puede cambiar la forma de ver el mundo a nuestro alrededor y comprender mejor el funcionamiento de las cosas. El instruirse dentro de la programación puede volver a las personas más analíticas y capaces de dar solución a problemas que otras personas encuentran sumamente complejos.

2.2 Sistemas en Chip SoC

Javier Romero describe los Sistemas en Chip *(SoC)* como chips electrónicos los cuales integran una gran cantidad de funciones y componentes esenciales de un sistema en un solo chip, el tener todos estos componentes incluidos en un solo chip tiene grandes ventajas como lo son el ahorro de energía pues estos chips están diseñados específicamente para tener un bajo consumo de está, una necesidad menor de espacio donde colocarlos pues todo ya viene incluido en un solo chip y esto también permite simplificar el diseño de los circuitos con los que se desea trabajar. (Javier Romero; (05/10/2021); ¿Qué es un *SOC* y para qué sirve?)

2.2.1 Ejemplos de SoC

Algunos ejemplos de *SoC* son:

- *Procesadores Intel Atom Serie C*: Este procesador es un *SoC* que ofrece inteligencia artificial eficiente en espacios más pequeños en el perímetro de la red. Se utiliza en una gran variedad de cargas de trabajo que requieren muy poca alimentación, alta densidad e integración de E/S. ((s. f.); *Intel Atom® C Processors Series.*)

- *Procesador AMD Ryzen Embedded serie V1000:* Los *SoC AMD Ryzen Embedded v1000* ofrecen gráficos y procesamiento multimedia similares a los de una *GPU* independiente, así como un rendimiento de procesamiento de hasta 3,61 *TFLOPS* con una potencia de diseño térmico (*TDP*) mínima de 12 W y máxima de 54W a fin de equipar a los diseñadores de sistemas para que alcancen nuevos niveles de eficiencia de procesamiento y versatilidad de diseño. Asimismo, hay disponible una opción con temperatura industrial (*iTemp*) capaz de alcanzar temperaturas de operación mínimas de -40°C. ((s. f.); *AMD RyzenTM Embedded Serie V1000.*)

- *Samsung Exynos 2200*: El *Exynos* 2200 está diseñado con una potente unidad de procesamiento de gráficos (*GPU*) *Samsung Xclipse* basada en la arquitectura *AMD RDNA 2*. Con sus núcleos de *CPU* basados en *ARM.* El *Exynos* habilitará una mejor experiencia *gaming* en los teléfonos móviles, además de perfeccionar la experiencia general en las aplicaciones de redes sociales y fotografía. ((s. f.); *Samsung* presenta el revolucionario procesador *Exynos 2200* con *GPU Xclipse* e impulsado por arquitectura *AMD RDNA* 2.)

- *Raspberry Pi Pico:* Raspberry Pi Pico es una placa de microcontrolador de alto rendimiento y bajo costo con interfaces digitales flexibles. Incorpora el chip de microcontrolador RP2040 de Raspberry Pi, con un procesador Arm Cortex M0+ de doble núcleo que funciona hasta 133 MHz, 264 KB de SRAM integrados y 2 MB de memoria Flash incorporada, así como 26 pines GPIO multifunción, para el desarrollo de software, están disponibles los SDK de C/C++ de Raspberry Pi o MicroPython. (Raspberry Pi Pico - Waveshare Wiki. (s. f.))

- *ESP8266*: El ESP8266 es un Soc (System on Chip), con capacidades de 2.4 GHz Wi-Fi (802.11 b / g / n, soporte WPA / WPA2), 16 GPIO de propósito general (entrada / salida), I^2C, convertidor analógico-digital (ADC de 10 bits), SPI, I2S, UART y modulación de ancho de pulso (PWM), emplea un CPU RISC de 32 bits basado en Tensilica Xtensa LX106 funcionando a 80 MHz. Tiene una memoria ROM de inicio de 64 KB, memoria RAM de instrucciones de 64 KB y 96 KB de RAM de datos. Memoria flash externa de 4MB pero este último varía entre diferentes versiones de modulo. (Usando ESP8266 con el IDE de Arduino. (s. f.). Naylamp Mechatronics - Perú.)

2.2.2 Protocolos de comunicación

Alejandro Daniel Perez (PEREZ, A; et al;) menciona que los protocolos empleados en los sistemas embebidos son protocolos que fueron obtenidos a partir de protocolos ya existentes en las computadoras y sistemas mayores, otras veces fueron adoptados a partir de protocolos utilizados en electrónica de consumo u industrial. Con el correr de los años estos mismos protocolos fueron sufriendo adaptaciones, dejando algunos en desuso y otros en plena vigencia con características que los hacen robustos, incluso a las condiciones más adversas. (PEREZ, A; et al;)

***Bus I2C**: I2C* es el acrónimo de *Bus Inter-Integrated Circuit*. El bus *I2C* fue desarrollado a comienzos de los años ochenta, por *Philips Semiconductors.* El propósito original de este protocolo era proveer una manera sencilla de conectar la *CPU* con los chips periféricos en los televisores y demás sistemas de complejidad media-alta.

Hoy por hoy este bus es aceptado por la industria como un estándar de facto. Asimismo, fue adoptado por los fabricantes líderes como *ST Microelectronics, Atmel, Texas Instruments* entre otros. (*Embedded Systems Academy; 2015; History of the I2C Bus.*)

Can Bus: Es un protocolo serial creado a mediados de los ochenta por la compañía alemana *Bosch*. Está optimizado para enviar pequeñas cantidades de datos entre múltiples nodos. *CAN* tiene una tasa máxima de transmisión de *1MB/s,* sin embargo, su operación a bajas velocidades lo hace robusto al ruido y le permite cubrir largas distancias.

El bus *CAN* fue originalmente diseñado para su uso en automóviles, pero fue haciéndose popular en otros ámbitos como control de líneas de ensamble industriales, como protocolo de transmisión dentro de barcos, etc. Pese a este éxito en crecimiento, la especificación de *Bosch* no define un estándar respecto a los voltajes o a los conectores o cables: cada organización define múltiples estándares a nivel físico. La capa física (hardware básico que se necesita para la transferencia de bits entre los distintos nodos que componen la red) más común y utilizada es el estándar *ISO 11898-1,* pero puede ser implementado de otras maneras. (*Bosch, R; 1991;s CAN BUS Specification.*)

USB: El *Universal Serial Bus* más conocido por la sigla *USB*, es un bus estándar industrial que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre computadoras, periféricos y dispositivos electrónicos.

Su desarrollo partió de un grupo de empresas del sector que buscaban unificar la forma de conectar periféricos a sus equipos, por aquella época poco compatibles entre sí, entre las que estaban *Intel, Microsoft, IBM, Compaq, DEC, NEC y Nortel.* La primera especificación completa 1.0 se publicó en 1996, pero en 1998 con la especificación 1.1 comenzó a usarse de forma masiva. (*Knowledgebase; USB History - Premium USB;* (s. f.))

SPI: El *bus SPI* (del inglés *Serial Peripheral Interface*) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos desarrollados en un principio por Motorola. El bus de interfaz de periféricos serie o bus *SPI* es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de *bit* serie regulado por un reloj.

Es un protocolo *full-duplex* que funciona bajo un paradigma maestro-esclavo. Incluye una línea de reloj, dato entrante, dato saliente y un pin de *chip select*, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj. (Microchip Technology; 2014; Overview and Use of the PICmicro Serial Peripheral Interface.)

(Alejandro Daniel Perez; (2016); "Protocolos de comunicación entre microcontroladores. Caso de estudio: Protocolo CAN". Tesina de Licenciatura en Sistemas.)

2.3 Sistemas embebidos

2.3.1 ¿Qué es un sistema embebido?

Un sistema embebido es un sistema informático compuesto por una combinación de *hardware* y *software* diseñado para realizar tareas específicas, sus componentes están todos integrados dentro de una placa base, cuenta con microcontrolador o un microprocesador el cual se encarga de realizar todo el procesamiento central, tiene interfaces de entrada/salida e incluso una memoria de tamaño reducido dentro del mismo chip.

Son utilizados principalmente en tareas que necesiten ejecución en tiempo real, además de ser utilizados también en el diseño y desarrollo de aplicaciones y prototipos con sistemas embebidos desde entornos gráficos, se pueden programar utilizando diferentes lenguajes de programación mediante compiladores específicos. (Trbl; 2023; Sistemas embebidos y sus características | conceptos fundamentales. TRBL Services)

2.3.2 Tipos de sistemas embebidos

Existen diferentes tipos de sistemas embebidos los cuales se pueden definir mediante sus requisitos de rendimiento o sus requisitos funcionales.

- Sistemas integrados móviles.

  Este tipo de sistemas son más pequeños y fáciles de usar, cuentan con una memoria limitada, pero tienen ventaja al permitir mejor portabilidad y manejabilidad, algunos ejemplos de estos sistemas son:

  - Cámaras digitales.
  - Teléfonos móviles.
  - Reloj inteligente.
  - Rastreador de *fitness*.

- Sistemas integrados de red.

  Estos sistemas se pueden conectar a redes alámbricas o inalámbricas con la finalidad de realizar tareas específicas y dar salida a los dispositivos conectados, algunos ejemplos de estos sistemas son:
  - Cajeros automáticos.
  - Sistemas de seguridad para el hogar.
  - Máquinas expendedoras de tarjetas.

- Sistemas integrados independientes.

  Son sistemas autónomos que no tienen dependencia alguna hacia un sistema anfitrión para poder cumplir con sus tareas asignadas, algunos ejemplos de estos sistemas son:

  - Hornos de microondas.
  - Lavadoras.
  - Consolas de videojuegos.

- Sistemas integrados en tiempo real.

  Este tipo de sistemas son diseñados para realizar tareas específicas dentro de un límite de tiempo predefinido, estos sistemas pueden clasificarse en 2 tipos:

  - Sistemas embebidos en tiempo real suave: Para estos sistemas el finalizar con las tareas es de vital importancia mientras que los plazos de tiempo no son prioridad.
  - Sistemas embebidos en tiempo real duro: Este tipo de sistemas les da prioridad a los plazos de tiempo para realizar sus tareas.

  Algunos ejemplos de estos sistemas son:
  - Sistema de sonido de un ordenador. (Sistema de tiempo real suave)
  - Sistema de control de una aeronave. (Sistema duro en tiempo real) (*Luke;* 22 de diciembre de 2022; Explicación de los sistemas embebidos - *Rebound Electronics.)*

2.3.3 Lenguajes de programación para sistemas embebidos

- Lenguaje ensamblador.

  Es un lenguaje de bajo nivel utilizado para el diseño de programas que pueden ser ejecutados directamente sobre un procesador o microcontrolador, las instrucciones del lenguaje ensamblador se traducen directamente a lenguaje máquina y cada una de estas instrucciones representa una operación específica la cual el procesador es capaz de realizar. Es utilizado en sistemas embebidos gracias a que permite un control más preciso y directo sobre el *hardware* del microcontrolador. (*Marker G;* 2022; El lenguaje ensamblador. Tecnología + Informática.)

- Lenguaje C/C++.
  Estos lenguajes son muy utilizados en la programación de sistemas embebidos gracias a su eficiencia, portabilidad y su gran capacidad para acceder y manipular *hardware* a un bajo nivel, son lenguajes de nivel medio que permiten desarrollar códigos más eficientes y de alto rendimiento gracias a que permiten un enfoque tanto en programación estructurada como orientada a objetos. (*Trbl;* 2023; Sistema embebido|Descubre C/C++ para el desarrollo de *software* embebido. *TRBL Services.)*

- *Python*.
  Es un lenguaje de programación de alto nivel, interpretado, es decir un lenguaje en el que las instrucciones se ejecutan directamente sin necesidad de compilarse, el cual tiene una sintaxis clara y legible, es un lenguaje muy popular que no es muy comúnmente utilizado para el desarrollo de sistemas embebidos ya que presenta algunas limitantes como el tiempo de ejecución, recursos limitados de los sistemas embebidos, dependencia de librerías externas. Si se desea utilizar *Python* para el desarrollo de sistemas embebidos se pueden utilizar las siguientes opciones que están optimizadas para los sistemas embebidos.

    - *MicroPython*: Es una versión optimizada para sistemas embebidos y microcontroladores el cual reduce el tamaño del intérprete para tener una forma eficiente de ejecutar código.
    - *PyPy*: Es una versión que utiliza una técnica llamada *“just in time compilation”* con la cual se mejora el rendimiento en comparación con el intérprete normal de *Python*.

- Microcontroladores con soporte nativo para *Python*: Existen algunos microcontroladores y placas de desarrollo que cuentan con soporte nativo para ejecutar código *Python*. ((s. f.); *Python*_para_Microcontroladores.)

- *Java*.
  Es un lenguaje de programación de alto nivel orientado a objetos el cual es ampliamente utilizado para el desarrollo de múltiples tipos de aplicaciones, se puede utilizar en el desarrollo de sistemas embebidos, aunque esto no es muy común, existen versiones de *java* las cuales están optimizadas para sistemas embebidos lo cual permite su ejecución en dispositivos con recursos limitados. (2023; ¿Qué es java incrustado? - Definición de *techopedia* - Desarrollo 2023)

- *VHDL*.
  Es un lenguaje de descripción de *hardware* utilizado para diseñar, modelar y realizar simulaciones de sistemas digitales, este lenguaje se utiliza ampliamente en la industria para realizar el desarrollo de sistemas embebidos, circuitos integrados y otros dispositivos electrónicos, aunque se enfoca principalmente en la descripción de *hardware* y no en la programación lógica del sistema. (Sergio Noriega; 2017; Introducción al Diseño lógico con *VHDL.*)

- *RUST*.
  Es un lenguaje de programación moderno de alto nivel que ha ido teniendo mucha popularidad para el desarrollo de sistemas embebidos gracias al enfoque que tiene en cuanto a temas de seguridad, rendimiento y la concurrencia. Debido a que el lenguaje es moderno puede requerir un esfuerzo adicional en comparación con otros lenguajes, sin embargo, la seguridad y rendimiento que ofrece este lenguaje puede ser de vital importancia en proyectos que requieran alta confiabilidad y eficiencia en sistemas embebidos. (*Syntonize;* 2022; *Rust* como lenguaje de programación y sus beneficios - *Syntonize*.)

2.3.4 Plataformas para sistemas embebidos

Existen diferentes plataformas y sistemas operativos que se pueden utilizar para el desarrollo de sistemas embebidos, dichas plataformas y sistemas proporcionan un entorno para desarrollo, herramientas y bibliotecas diseñadas para las limitantes que presentan los sistemas embebidos, algunas de estas plataformas y sistemas son los siguientes.

- *Arduino*.
  Es una plataforma de *hardware* y *software* que puede ser utilizada para el desarrollo de sistemas embebidos básicos, cuenta con una placa de desarrollo con un microcontrolador, un entorno de desarrollo basado en *Wiring*, lo cual facilita la creación de proyectos electrónicos. Aunque *Arduino* es una plataforma muy útil para proyectos básicos, puede tener limitaciones en términos de potencia de procesamiento, capacidad de almacenamiento y conectividad en comparación con otras plataformas. (Fernández, Y.; 2022; Qué es arduino, cómo funciona y qué puedes hacer con uno. *Xataka*)

- *Raspberry Pi.*
  Es una plataforma de computadora con tamaño reducido la cual se utiliza para el desarrollo de sistemas embebidos de nivel superior, combina un potente procesador con una variedad de interfaces y puertos lo cual nos permite desarrollar aplicaciones más complejas, además de ser compatible con una gran variedad de sistemas operativos. Debido a que *Raspberry Pi* tiene un enfoque más general puede tener ciertas limitaciones en cuanto a ejecución en tiempo real, capacidad de respuesta en tiempo crítico y consumo de energía en comparación con otros microcontroladores más dedicados. (*Trbl;* 2023; *Arduino vs Raspberry Pi* | ¿Cómo elegir? ¿Para qué sirven? 10 tips. *TRBL Services)*

- *BeagleBone.*
  Es una plataforma de computadora de una sola placa la cual se puede utilizar como ordenador autónomo o integrado en un sistema, es una placa de bajo coste y de código abierto basada en *Linux* la cual está diseñada para el desarrollo de sistemas embebidos. (Ecarletti; (s. f.); ¿Qué es *BeagleBone Blue*? | Robots didácticos.)

- *ARM mbed.*
  Es una plataforma con un ecosistema diseñado para el desarrollo de sistemas embebidos basados en microcontroladores *ARM (*Advanced Risc Machine), incluye *hardware, software* y una gran variedad de herramientas para poder simplificar y acelerar el proceso de creación de sistemas embebidos especialmente para dispositivos enfocados en *IoT*. (Jacob Beningo; 2019; Acelere el desarrollo de productos *IoT* utilizando el ecosistema *Mbed*.)

- *FreeRTOS.*
  Es un sistema operativo de tiempo real de código abierto utilizado principalmente en el desarrollo de sistemas embebidos y aplicaciones críticas en tiempo real, se utiliza en una amplia variedad de aplicaciones como dispositivos *IoT*, sistemas de control en tiempo real, sistemas de comunicación, sistemas de automatización, se enfoca principalmente en la eficiencia y capacidad de respuesta en tiempo real. (Guillermo; (s. f.); Introducción a los *RTOS*.)

- *Linux* embebido.
  Es una adaptación del sistema operativo *Linux* para su uso en sistemas embebidos, este sistema es flexible, de bajo costo y código abierto, ha sido portado a diferentes microprocesadores de propósito específico. *Linux* permite múltiples proveedores de *software*, desarrollo y soporte, tiene un núcleo estable y facilita la capacidad de leer, modificar y redistribuir el código fuente. (*Trbl; 2023; LINUX* embebido | Qué es, cómo funciona y para qué se usa. *TRBL Services*.)

2.4 Componentes Electrónicos para el Robot Educativo

Los componentes electrónicos son dispositivos que se suelen encapsular generalmente en un material cerámico, metálico o plástico, y terminar en dos o más terminales o patillas metálicas. Las resistencias, el condensador, un micrófono, las baterías y los transformadores son solo algunos de los elementos más conocidos de este tipo, estos elementos se pueden clasificar de diferentes formas:

Al clasificarlos por su funcionamiento podemos encontrar 2 tipos de componentes:

- **Activos:** Necesitan ser alimentados por una fuente de poder para poder realizar su tarea, son aquellos capaces de generar, modificar o ampliar una señal eléctrica. Los más comunes son las baterías, transistores, diodos.

- **Pasivos:** Son los componentes que no proporcionan ganancia, pero consumen energía. También son los encargados de la conexión entre los diferentes componentes activos. Los más conocidos y utilizados son los resistores, condensadores y bobinas.

Si se clasifican por su estructura física se dividen en 2 tipos:

- **Discretos:** Son aquellas estructuras que solo cuentan con un solo componente eléctrico, el cual puede ser activo o pasivo.

- **Integrados:** Es un circuito electrónico que contiene componentes como transistores, diodos, resistencias y capacitores, que se encuentran interconectados en un material semiconductor, como por ejemplo un amplificador operacional o puerta lógica.

Se pueden clasificar por su tipo de energía en 3 tipos diferentes:

- **Electromagnéticos:** Son aquellos componentes que aprovechan las propiedades electromagnéticas de los materiales.

- **Electroacústicos:** Aquí entran los componentes eléctricos que transforman la energía acústica en eléctrica y viceversa, tales como los micrófonos, altavoces, bocinas y auriculares.

- **Optoelectrónicos:** Componentes que transforman la energía lumínica en eléctrica y viceversa.

También se pueden clasificar por su material base de fabricación:

- **Semiconductores:** Aquí entran los diodos y transistores, componentes capaces de dirigir o controlar el flujo de la corriente eléctrica.

- **No Semiconductores:** En este grupo están los componentes que cumplen una sola función ya sea como conductores o como aislantes, algunos ejemplos de estos materiales son: bulbos, resistencias, capacitores, bobinas, etc. (Crodriguez, & Crodriguez. (2023, 28 junio). ¿Qué son los componentes electrónicos y cuáles se utilizan más?)

2.4.1 Displays

Un *display* es un dispositivo específicamente diseñado para poder mostrar datos específicos en un formato legible y comprensible para el usuario, existen diferentes tipos de display que se pueden encontrar en la actualidad.

- Pantalla *LED*: Esta es una pantalla electrónica que basa su funcionamiento en diodos emisores de luz, (*leds*). Estos se agrupan en módulos que dan lugar a *píxeles,* los cuales pueden formar imágenes, textos o incluso videos.

- Pantalla *LCD*: Estas pantallas funcionan gracias al cristal líquido, detrás del cual se sitúan los diodos emisores de luz. Gracias a estos se puede crear la imagen en el cristal, la cual suele ser bastante luminosa. Es habitual encontrarla en televisores, monitores y también en *smartphones* y *tablets.*

- Pantalla *OLED:* Este tipo de pantallas digitales trabajan a partir de diodos de carbono emisores de luz. Estos llegan a funcionar como si fueran *píxeles* y generan las imágenes. Estas se caracterizan por ser mucho más brillantes y por contar con mayor nitidez. (*Led, M. T.;* 4 enero de 2018; Tipos de pantallas electrónicas más comunes - Maupe.)

2.4.2 Módulo UART

UART significa receptor/transmisor asíncrono universal y define un protocolo, o conjunto de reglas para intercambiar datos en serie entre dos dispositivos. El *UART* es muy simple y solo utiliza dos cables entre el transmisor y receptor para transmitir y recibir en ambas direcciones. Ambos terminales también tienen una conexión a tierra. La comunicación en *UART* puede ser *simplex* (los datos se envían en una sola dirección), *semidúplex* (cada lado transmite, pero solo uno a la vez), o *dúplex* completo (ambos lados pueden transmitir en simultáneo). Los datos en el *UART* se transmiten en la forma de tramas. (Rohde & Schwarz; (s. f.); Entendiendo el *UART.*)

2.4.3 Motores

El motor eléctrico es una máquina electromecánica que convierte la energía eléctrica en energía mecánica. El principio de funcionamiento del motor eléctrico depende sobre todo de la interacción entre el campo magnético y el eléctrico.

Los motores eléctricos se clasifican principalmente en dos tipos o en categorías, dependiendo del tipo de energía eléctrica aplicada: motores de corriente continua *(DC)* y motores de corriente alterna *(AC)*. (Aula; mayo 2023; Cómo funciona un motor eléctrico: tipos y partes.)

2.4.4 Sensores

Los sensores son dispositivos que detectan y miden magnitudes físicas como la temperatura, la presión, la humedad, la posición, el movimiento, la fuerza, la luz y muchos otros. La mayoría de los sensores están basados en principios físicos como la resistencia eléctrica, la capacidad, la inductancia, la luz o el magnetismo, entre otros.

Una vez que los sensores detectan una magnitud física, convierten esa información en una señal eléctrica, que puede ser procesada por un circuito electrónico. Los sensores electrónicos son un componente fundamental en la mayoría de los sistemas electrónicos modernos, ya que permiten que las máquinas y dispositivos tengan una retroalimentación del entorno que los rodea.

Existen diferentes tipos de sensores:

- **Temperatura:** Miden la temperatura de un objeto o ambiente y convierten la información en una señal eléctrica. Estos sensores se utilizan en una amplia variedad de aplicaciones, desde la industria alimentaria hasta la medicina.

- **Presión:** Miden la presión de un fluido o gas y convierten la información en una señal eléctrica. Se utilizan en la industria automotriz, aeronáutica y en la medicina.

- **Humedad:** Miden la humedad de un ambiente y convierten la información en una señal eléctrica y se utilizan en la industria agrícola, de alimentos y en la construcción.

- **Posición:** Miden la posición de un objeto en relación con un punto de referencia y convierten la información en una señal eléctrica. Estos sensores se utilizan en la industria automotriz, aeronáutica y en la robótica.

- **Movimiento:** Miden la velocidad y dirección del movimiento de un objeto y convierten la información en una señal eléctrica. Así, se desarrollan especialmente pensando en la industria de la seguridad, la robótica y en la electrónica de consumo. (Rodriguez, A; marzo 2023; Sensores electrónicos: ¿Qué son y cómo funcionan?)

2.5 Metodologías para el desarrollo de software.

Una metodología para el desarrollo de *software* es un conjunto de prácticas, técnicas y herramientas que se utilizan para poder planificar, diseñar, construir, probar y entregar software de alta calidad de manera eficiente. Estas metodologías estructuran el ciclo de vida del *software*, establecen roles y responsabilidades, y definen procesos para la gestión de proyectos, la comunicación y el seguimiento del progreso.

Existen 2 clasificaciones para las metodologías de desarrollo de software, metodologías tradicionales y metodologías ágiles.

2.5.1 Metodologías Tradicionales

Las metodologías tradicionales son rígidas y establecen los requisitos y procesos al inicio del proyecto, lo que limita la flexibilidad y los ajustes durante el desarrollo.

Las etapas del proyecto suceden de forma secuencial impidiendo avanzar sin completar la etapa anterior, entre las metodologías tradicionales podemos encontrar algunos ejemplos como:

- **Método *Waterfall* o Cascada:** Esté método organiza las actividades del proyecto en fases secuenciales como análisis de requisitos, diseño del sistema, codificación y mantenimiento, asegurando que cada etapa esté completa antes de avanzar.

- **Método Incremental:** Similar al *Waterfall,* pero agrega funcionalidad en cada etapa, permitiendo verificaciones parciales del software en desarrollo.

- **Diseño Rápido de Aplicaciones (RAD):** Desarrolla *software* rápidamente mediante la creación de prototipos para que los usuarios los prueben y definan sus necesidades prioritarias.

- **ITIL:** Es un conjunto de normas y prácticas para la gobernanza de *TI* en empresas, útil en la ejecución, implementación y mantenimiento de desarrollos de *software.*

- **Metodología en V:** Es una variación de la metodología *Waterfall*, donde las fases de desarrollo y pruebas están alineadas, formando una V, cada fase de desarrollo tiene una fase de prueba correspondiente, lo que asegura una validación continua y rigurosa.

2.5.2 Metodologías Ágiles

Las metodologías ágiles son flexibles y adaptables, permitiendo hacer ajustes rápidos y continuos a medida que surgen nuevas necesidades. Se basan en ciclos de desarrollo cortos e iterativos, con equipos que trabajan en estrecha colaboración con el cliente, dentro de este tipo de metodología podemos encontrar algunos ejemplos como:

- **DEVOPS:** Integra desarrollo, operaciones y seguridad, mejorando la agilidad y la respuesta rápida en procesos de desarrollo.

- **SCRUM:** Es un *framework* que divide el trabajo en roles bien definidos y ciclos cortos, mejorando la adaptabilidad y eficiencia.

- **Kanban:** Utiliza tableros visuales para gestionar el flujo de trabajo, limitando el trabajo en progreso y mejorando la eficiencia mediante el seguimiento y eliminación de cuellos de botella.

(Valtx; 19 julio de 2022; Metodologías para el desarrollo de software: ¿Qué son y para qué sirven?)

2.6 Robótica educativa

2.6.1 ¿Qué es la robótica educativa?

La robótica educativa o robótica pedagógica es una disciplina enfocada para que los estudiantes se inicien desde una corta edad en los temas de robótica y la programación, la robótica educativa se engloba dentro del modelo denominado Educatrónica, un modelo de enseñanza-aprendizaje orientado a la innovación educativa a través de la enseñanza de las ciencias, tecnología, matemáticas y arte con una visión desde las humanidades. (Ruiz-Velasco&Bárcenas; 2022)

2.6.2 ¿Qué es el Modelo de Educatrónica?

Se encarga de armonizar las disciplinas pedagógica y tecnológica con el objeto de facilitar la construcción de conocimiento, de habilidades y competencias de base: tecnológicas, de información y comunicación; de nociones científicas y artísticas, sustentadas en teorías pedagógicas apropiadas que despliegan situaciones didácticas diseñadas ad hoc, y que resultan afines con el uso e integración inteligente y efectiva de tecnologías accesibles, disponibles y con un alto grado de utilidad.

Se apoya en metodologías activas de aprendizaje, donde los estudiantes aprenden por su cuenta y los docentes actúan como guías para ellos, gracias a esta metodología los estudiantes tienen la oportunidad de desarrollar habilidades del pensamiento: pensamiento creativo, computacional, matemático.

2.6.3 Definición de robot

Un robot es una máquina que está diseñada para realizar tareas de forma automática o semiautomática, pueden ser programados para llevar a cabo diversas actividades y pueden estar equipados con *hardware* como sensores y actuadores para poder interactuar con el entorno.

Los robots pueden ser físicos o virtuales los cuales existen solamente en entornos de *software* siendo capaces de simular el comportamiento de un robot físico, se pueden controlar mediante programación previamente definida, mediante el uso de un control remoto o utilizando algoritmos de inteligencia artificial que les permita aprender y adaptarse a su entorno. (Innovación Digital, R., & Innovación Digital, R. (2023). Robots: qué son, funcionamiento y modelos. ¿Nos sustituirán? Innovación Digital 360)

En la actualidad existen una gran variedad de robots, los robots se pueden clasificar dependiendo de sus características y el uso que se les da, entre esta variedad de robots existen de tipo industrial, educativos, médicos quirúrgicos, de atención al cliente, etc.

- **Robot Industrial:** Este tipo de robot es el más estandarizado y conocido, utilizado principalmente en el sector industrial, para trabajos duros que requieren de una gran carga física. Los robots industriales están compuestos por brazos robóticos mecánicos. Se les exige precisión y productividad, por lo que son empleados en tareas repetitivas que, de ser realizadas por humanos, podrían resultar en errores.

- **Robot Educativo:** Este tipo de robot es una herramienta diseñada para facilitar el aprendizaje y la enseñanza de diversas disciplinas, especialmente en el ámbito de la ciencia, tecnología, ingeniería y matemáticas. Estos robots están diseñados para ser interactivos y accesibles permitiendo a los estudiantes aprender conceptos de programación, robótica, electrónica y mecánica de una forma más práctica.

- **Robot Médico Quirúrgico:** Los robots en el sector de la medicina están ganando cada vez más importancia. Destacan los robots quirúrgicos, como el robot Da Vinci, que son esenciales para realizar intervenciones con gran precisión. Además, existen robots auxiliares que asisten en las operaciones proporcionando material a los especialistas. También hay robots asistenciales que ayudan a los pacientes a moverse, exoesqueletos que colaboran en la rehabilitación y robots que eliminan virus de las habitaciones.

- **Robot de Atención al Cliente:** Este tipo de robots se está implementando cada vez más en lugares con gran afluencia de público, como aeropuertos, centros comerciales, bancos y hoteles. Los *chatbots* o *bots* se utilizan en el sector financiero y bancario para atender al público, proporcionando soporte y aliviando la carga de trabajo de los empleados al resolver dudas en informar a los usuarios. Son muy utilizados en el sector turístico y hotelero debido a su capacidad para manejar diferentes idiomas.

(*Tipos de Robots: Características y Ejemplos*, 2020)

2.6.4 ¿Qué es un robot educativo?

Un robot educativo es un dispositivo programable y diseñado para ser utilizado en el ámbito educativo, desde la educación infantil hasta niveles avanzados, con el propósito de fortalecer los aprendizajes fundamentales en áreas como matemáticas, informática y lenguaje (conocido como sistema *STEAM*). Estos robots se utilizan como herramientas pedagógicas para fomentar el aprendizaje práctico y atractivo,

ya que permiten a los estudiantes interactuar con ellos a través de la construcción, ensamblaje y programación de robots educacionales interactivos. Además, un robot educativo debe ser capaz de proporcionar conocimiento confiable y diversión a los estudiantes, a menudo sirviendo como una fuente de aprendizaje que se percibe como un juguete, lo que facilita su aceptación y uso en entornos educativos. (Euroinnova Formación; 2023; Blogs educativos primaria. Euroinnova *Business School*.)

2.6.5 Tipos de robots educativos

Existen varios tipos de robots educativos utilizados en entornos educativos para promover el aprendizaje y el desarrollo de habilidades en los estudiantes, estos pueden ser clasificados de la siguiente manera.

- Robots de codificación física: Especialmente diseñados para los más pequeños, ya que aprenden programación sin pantallas, simplemente pulsando botones al estilo de un juguete. Algunos de los más utilizados en el entorno escolar son la abeja *Bee-Bot* [Figura 1] y el cubo *Cubetto* [Figura 2]. Estas propuestas introducen a los niños a los conceptos básicos mientras se divierten.

Figura 1. Robot Bee-Bot.
Fuente: BEE-BOT: Robot abeja educativo infantil programable. (s. f.). https://www.ro-botica.com/Producto/BEE-BOT/

Figura 2. Robot Cubetto.
Fuente: Cubetto: Un robot que enseña código y programación a los niños. https://cubetto.vicensvives.com/

- Robots programables de iniciación: Orientados a los niños de los primeros cursos de primaria, conservan la dinámica del juguete (con aspecto atractivo para los menores), pero interactúan con ellos introduciendo el *software* mediante una aplicación (con interfaz de juego), con la que controlan el robot desde una tableta o un teléfono inteligente, algunos ejemplos de estos son el *robot Botley* [Figura 3] y el *robot Dash* [Figura 4].

Figura 3. Robot Botley.
Fuente: BOTLEY EL ROBOT - Rayuelainfancia. Rayuelainfancia. https://www.rayuelainfancia.com/ciencia-naturaleza/7417-botley-el-robot.html

Figura 4. Robot Dash.
Fuente: Raquel. (2022, 22 septiembre). *Robot Dash: robótica y programación en casa o el aula*. Robots Para Niños. https://www.robotsparaninos.com/dash-and-dot-mucho-mas-que-robot-educativo/

- Robots programables por ordenador: Son para alumnos de primaria y secundaria, físicamente ya no recuerdan a adorables juguetes y se programan a través del *PC* o tabletas mediante un lenguaje de programación (por ejemplo, los basados en bloques). Integran sensores de varios tipos (luz, sonido, táctil…). Algunos ejemplos de estos robots son *ELEGOO UNO R3* [Figura 5] y *LEGO Mindstorms-EV3* [Figura 6].

Figura 5. Robot ELEGOO UNO R3.
Fuente: ELEGOO Official. *Smart Robot Car Kit v4.0 (With Camera)*.
https://www.elegoo.com/products/elegoo-smart-robot-car-kit-v-4-0

Figura 6. LEGO MINDSTORMS EV3.
Fuente: The LEGO® Group. *The LEGO® Group*. https://www.lego.com/es-mx/product/lego-mindstorms-ev3-31313

- Robots mediante *kits*: Son para estudiantes de robótica educativa de secundaria, el ensamblaje se convierte en un desafío extra para integrar sensores, motores y otros componentes de *hardware.* Integran construcción con programación a través de una pantalla. Los alumnos crean su propio prototipo y programan sus propias soluciones. Algunos ejemplos de estos robots son el *robot Makeblock mBot Ranger* [Figura 7] y el *robot Clementoni Robomaker* [Figura 8]. (*Observatory, D. E.;* (s. f.); La robótica educativa revoluciona las aulas y potencia el aprendizaje)

Figura 7. Robot Makeblock mBot Ranger.
Fuente: *MBot Ranger Robot Kit (Bluetooth Version)*. CreativaShop.
https://creativashop.com/products/mbot-ranger-robot-kit-bluetooth-version

Figura 8. Clementoni Robomaker.
Fuente: *RoboMaker, set de iniciación*. Clementoni ES.
https://es.clementoni.com/products/robomaker-set-de-iniciacion

2.7 Plataformas para robótica educativa

El texto generado por ChatGPT indicó que:

“Una plataforma para robótica educativa es un conjunto de herramientas, *software* y *hardware*, diseñado para enseñar y promover el aprendizaje de la robótica entre estudiantes de diferentes edades y niveles educativos. Estas plataformas están diseñadas para ser accesibles, intuitivas y motivadoras, brindando a los estudiantes la oportunidad de aprender conceptos de ciencia, tecnología, ingeniería y matemáticas (*STEM*) de manera práctica y divertida.

Estas plataformas suelen incluir *kits* de robótica que contienen componentes electrónicos, sensores, motores y piezas mecánicas, junto con un conjunto de instrucciones y guías para construir y programar robots. Además, proporcionan un entorno de programación visual o basado en bloques que permite a los estudiantes programar y controlar el comportamiento de los robots de forma sencilla, sin necesidad de tener conocimientos avanzados de programación.

Algunas plataformas de robótica educativa también ofrecen recursos en línea, como tutoriales, proyectos, retos y comunidades en línea donde los estudiantes pueden compartir sus creaciones y colaborar con otros entusiastas de la robótica. Estas plataformas fomentan la creatividad, el pensamiento crítico, la resolución de problemas y el trabajo en equipo, al mismo tiempo introducen a los estudiantes en los fundamentos de la robótica y la programación.

En resumen, una plataforma para robótica educativa es una herramienta integral que combina *kits* de robótica, *software* de programación y recursos en línea, con el objetivo de enseñar a los estudiantes habilidades *STEM* mientras exploran el mundo de la robótica de manera práctica y divertida.” (*OpenAI;* (2023, junio 22); Definición de una plataforma para robótica educativa [Mensaje en un modelo de lenguaje]. *ChatGPT.*)

2.7.1 Ejemplos de plataformas para robótica educativa

- *Complubot – Robot Crystalino.* Este robot es una plataforma educativa diseñada por *Complubot* para iniciarse en el mundo de la robótica trabajando el prototipado electrónico, pero sin necesidad de realizar ningún tipo de soldadura. *Crystalino* es un robot modular 100% compatible con *Arduino*, basado en la controladora *Compluino* UNO y que se programa de forma sencilla gracias a una potente librería de funciones. ((2020, 8 septiembre); *Crystalino* - Robot Educativo Compatible *Arduino* |.)

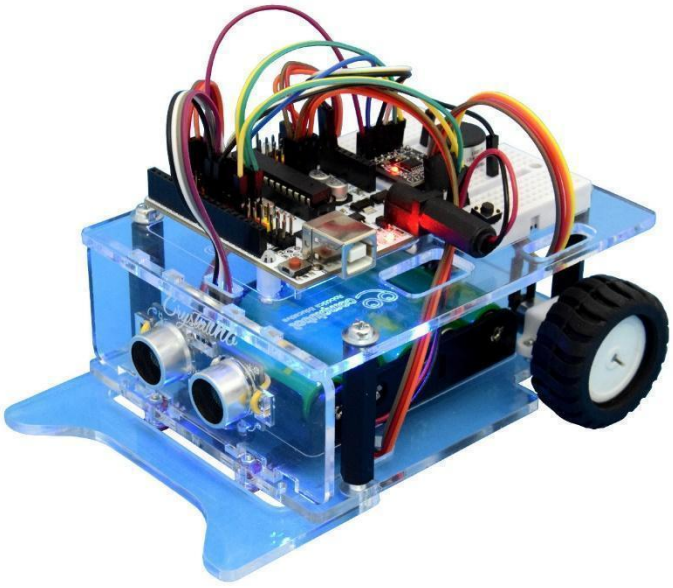

Figura 9. Robot Crystalino.
Fuente: *Crystalino - Robot educativo compatible Arduino* |. (2020, 8 septiembre). |. https://complubot.com/proyectos/crystalino/

- *Complubot* – Robótica con *Crumble*. Es una plataforma completa de soluciones en robótica educativa en la cual se permite la programación, el prototipado electrónico, el diseño mecánico y la creatividad mediante el uso de manuales y cuadernos de actividades. Utiliza como base el controlador *Crumble* el cual permite controlar la velocidad de hasta 2 motores *DC* en avance y retroceso. También dispone de 4 terminales los cuales pueden ser configurados de forma individual como entradas o salidas.

  El programa de *Crumble* es un *software* que funciona en múltiples sistemas operativos, es un entorno de programación gráfico inspirado en *Scratch*. ((2021, 2 junio); *Starting with robotics* |.)

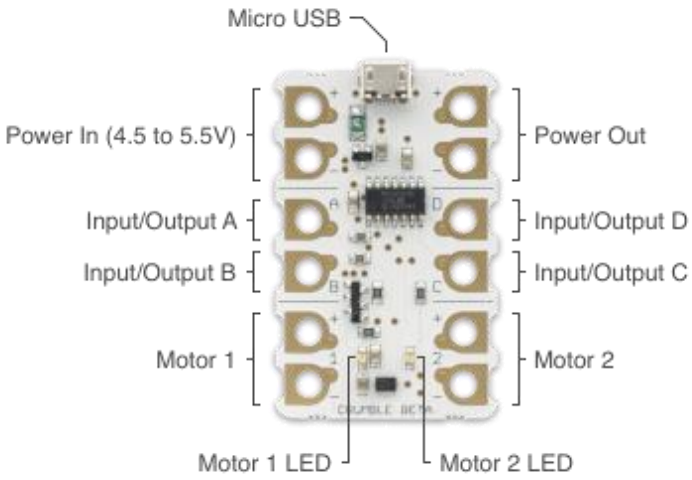

Figura 10. Controlador *Crumble.*
Fuente: Fernández, J. I. H. (2016, 22 octubre). *Crumble, un dispositivo muy interesante para iniciarse en la robótica*. Programamos. https://programamos.es/crumble-un-dispositivo-muy-interesante-para-iniciarse-en-la-robotica/

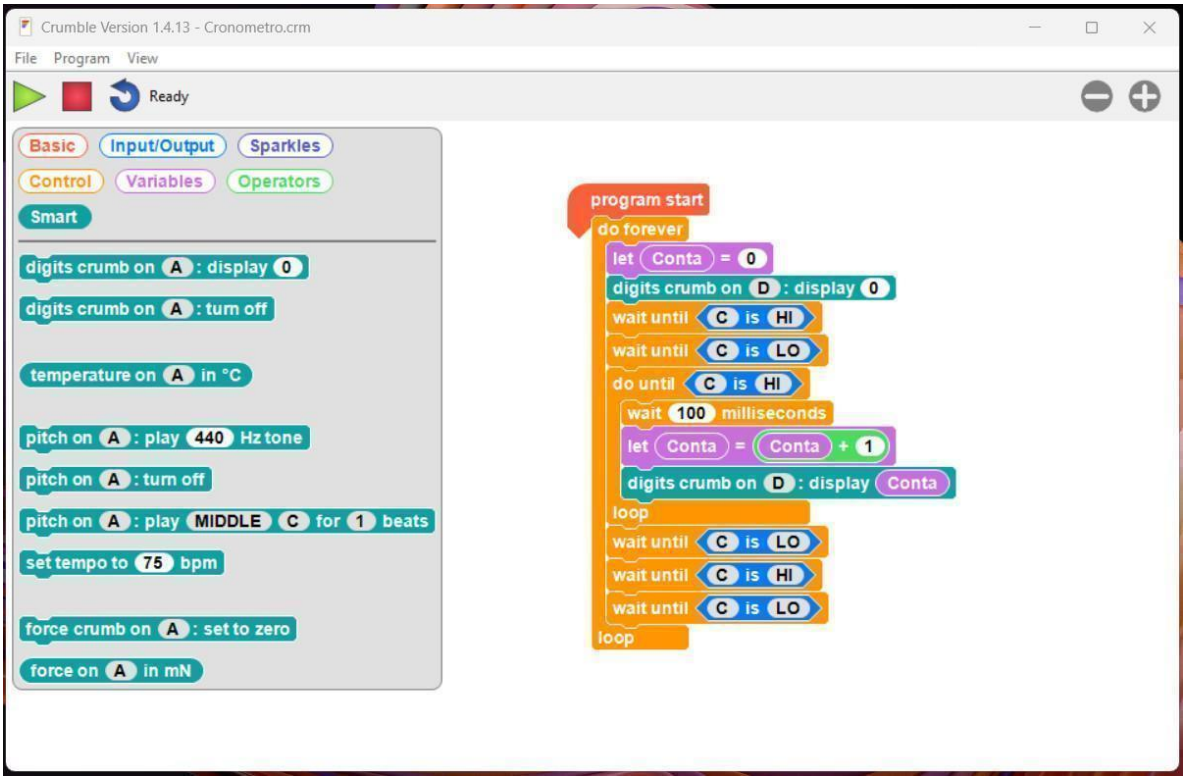

Figura 11. Software Crumble.
Fuente: Fernández, J. I. H. (2016, 22 octubre). *Crumble, un dispositivo muy interesante para iniciarse en la robótica*. Programamos. https://programamos.es/crumble-un-dispositivo-muy-interesante-para-iniciarse-en-la-robotica/

- Simulador *mBot.* Es una herramienta que permite a los usuarios interactuar con el robot educativo *mbot* sin necesidad de tener acceso al robot físico. Este *kit* está diseñado para enseñar conceptos básicos de programación, electrónica y robótica de manera divertida y accesible.
  El simulador *mBot* proporciona una interfaz virtual donde los usuarios pueden programar y controlar un robot virtual *mBot*. Permite escribir código, diseñar algoritmos y probarlos en un entorno virtual antes de implementarlos en el robot físico.
  El simulador *mBot* suele incluir una representación visual del *robot* y su entorno, lo que permite a los usuarios ver el efecto de sus programas en tiempo real. También cuenta con herramientas de depuración y simulación de sensores, como detectores de obstáculos y seguidores de líneas virtuales.

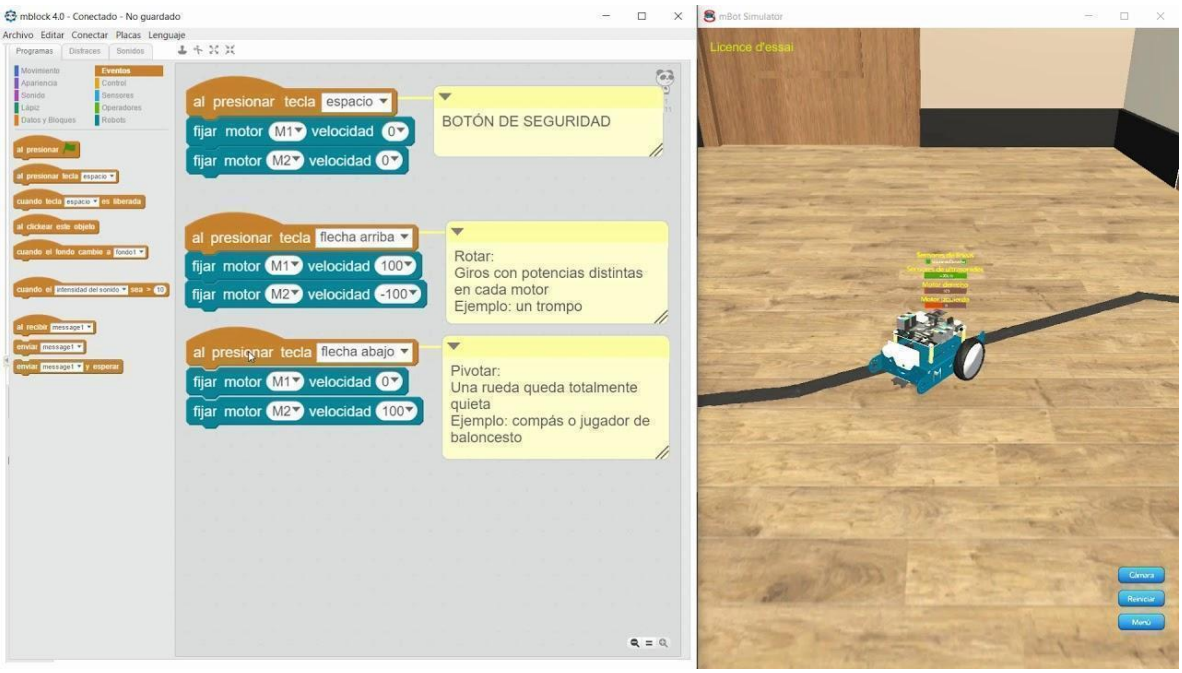

Figura 12. Simulador *mBot.*
Fuente: Dani. (2022, 31 mayo). *Robótica educativa 45 Entornos virtuales para programar robots emulados*. Juegos Robótica. https://juegosrobotica.es/podcast-045/#

- *Open Roberta Lab.* Es un entorno de programación en línea y una comunidad para aprender y enseñar robótica y programación la cual tiene como objetivo que estas mismas sean accesibles para una amplia gama de usuarios, especialmente niños y principiantes. La plataforma ofrece un lenguaje de programación que permite a los usuarios diseñar programas arrastrando y soltando bloques de código. Este enfoque simplifica el proceso de codificación y ayuda a los usuarios a centrarse en los aspectos lógicos de la programación.

Open Roberta Lab es compatible con varias plataformas de *hardware* como *LEGO Mindstorms EV3, Arduino, Calliope mini* y *micro:bit,* ofrece una amplia gama de tutoriales, programas de ejemplo y proyectos para guiar a los usuarios a través del proceso de aprendizaje. (Dani; (2022, 31 mayo); Robótica educativa 45 entornos virtuales para programar robots emulados.)

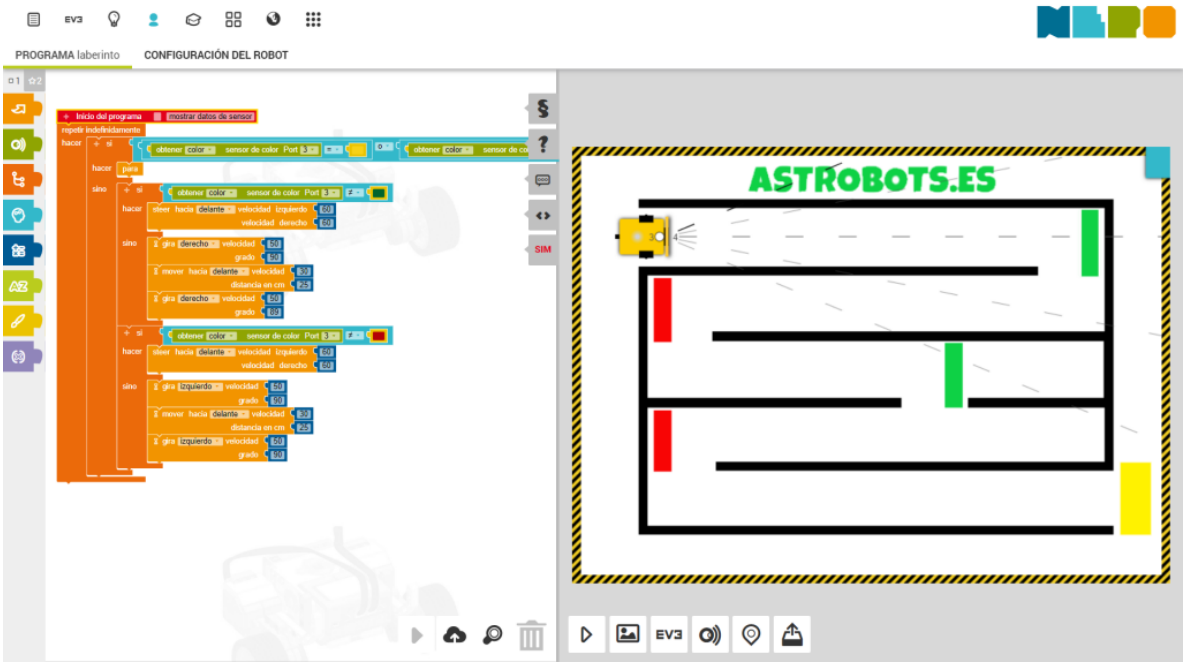

Figura 13. Open Roberta Lab.
Fuente: Dani. (2022, 31 mayo). *Robótica educativa 45 Entornos virtuales para programar robots emulados*. Juegos Robótica. https://juegosrobotica.es/podcast-045/#

- *LEGO Mindstorms Fix the factory.* Es un juego para dispositivos móviles que forma parte del mundo de *LEGO Mindstorms*, es una aplicación diseñada para enseñar conceptos básicos de programación y lógica a través de la resolución de desafiantes rompecabezas y puzzles.
  La aplicación dispone de una interfaz gráfica de programación en la que se pueden arrastrar y soltar bloques de código los cuales representan acciones y comandos que se utilizan para guiar a los robots y resolver los desafíos en cada nivel. (*Mindstorms fix the factory;* (2016, 8 junio))

Figura 14. LEGO Mindstorms fix the factory
Fuente: MOB.org. (2024, 16 mayo). *Descargar LEGO Mindstorms: Fix the factory gratis para Android | mob.org*. https://mob.org/es/android/games/g-lego_mindstorms_fix_the_factory

- *Kodable.* Es una plataforma educativa diseñada para enseñar fundamentos de programación y lógica de manera divertida e interactiva a niños de nivel preescolar y primaria.
  Utiliza una interfaz gráfica que permite a los niños arrastrar y soltar bloques de código para crear secuencias de instrucciones, con esto los niños programan personajes llamados *"Fuzzes"* para que resuelvan desafíos y puzles en un entorno de juego.
  Kodable se basa en un enfoque pedagógico de "aprender jugando", donde los niños adquieren habilidades de resolución de problemas, pensamiento lógico y secuenciación mientras se divierten con los diferentes niveles y desafíos del juego. (De Miguel, R; (2023, 20 septiembre); *Kodable*: una plataforma para programar mediante el juego.)

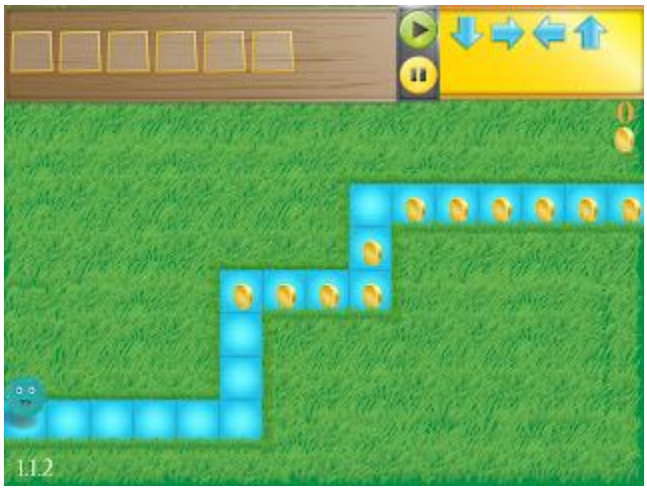

Figura 15. Aplicación *Kodable.*
Fuente: *Inicio a la programación con KODABLE*.
https://fabricandoconocimientos.blogspot.com/2013/11/inicio-la-programacion-con-kodable.html

2.8 Aplicaciones de escritorio

Una aplicación de escritorio es un programa de *software* que se instala en una computadora con un sistema operativo como *Windows* o *Mac* y se utiliza para realizar diversas tareas, como navegación por Internet, edición de documentos y visualización de videos. Estas aplicaciones se caracterizan por tener ventanas interactivas, soporte nativo para pantallas táctiles, seguridad mejorada y compatibilidad con varios dispositivos. Su propósito principal es maximizar la eficiencia, aumentar la funcionalidad, mejorar la seguridad y asegurar la compatibilidad con otros dispositivos. A diferencia de las aplicaciones móviles, las aplicaciones de escritorio suelen tener una interfaz más robusta y pueden acceder a bases de datos corporativos, lo que las hace adecuadas para empresas y tareas más complejas. Sin embargo, requieren un computador compatible con el sistema operativo y actualizaciones regulares. (Trucoteca; (2023, 11 febrero); ¿Qué son las aplicaciones en el escritorio?)

2.8.1 Ejemplos de aplicaciones de escritorio

- *Microsoft Word.* Es un procesador de texto utilizado tanto en el sector profesional como en el educativo.

Figura 16. Aplicación de escritorio *Word*.
Fuente: Equipo editorial, Etecé. (2023, 19 noviembre). *Word - Concepto, significado y reseña histórica*. Concepto. https://concepto.de/word/

- *Corel Draw.* Es un *software* de diseño gráfico profesional diseñado para la creación de gráficos y diseños de página, la edición de fotografías y el diseño de sitios *web.* ((s. f.); *CorelDRAW Graphics Suite* | *Free Trial.*)

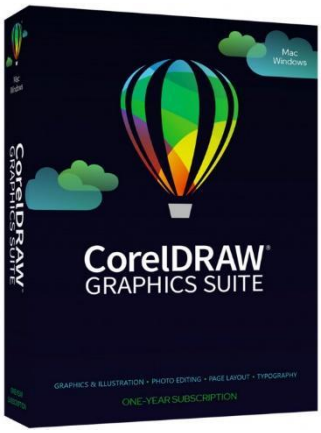

Figura 17. Aplicación de escritorio *CorelDraw.*
Fuente: *CorelDraw Graphics Suite X8, compre programas Corel na Software*. Software - o Software Que Você Precisa Está Aqui! Entre Já!
https://software.com.mx/p/coreldraw-graphics-suite

- *Adobe PDF.* Es un programa para *Windows* que permite visualizar, leer, imprimir y añadir notas en documentos *PDF* de manera gratuita. ((s. f.); Explicación de *adobe PDF.*)

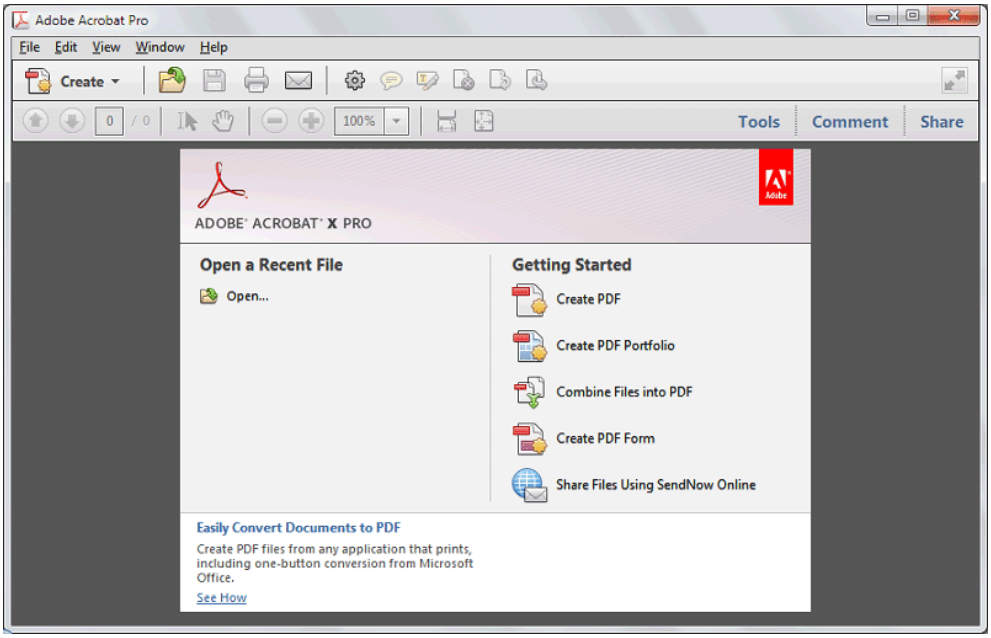

Figura 18. Aplicación de escritorio *Adobe PDF.*
Fuente: PDFelement. (2024, 17 abril). Qué es Adobe Acrobat - Cosas Que Debes Saber. *@PDFelement*. https://pdf.wondershare.es/pdf-software-comparison/what-is-adobe-acrobat.html

2.8.2 Ventajas y desventajas

Ventajas de utilizar aplicaciones de escritorio.

- Los datos están centralizados.
- Son programas mucho más estables y robustos.
- La carga de datos se produce con una mayor rapidez.
- No necesita de internet para poder funcionar.
- Suelen ser más económicas.
- La seguridad es mayor en este tipo de aplicaciones.
- Se pueden hacer copias de seguridad en todo momento.

Desventajas de utilizar aplicaciones de escritorio.

- Requiere de instalación en cada cliente a utilizarse.
- Se desarrolla para un sistema en específico por lo cual si se tiene otro sistema habrá que esperar a que esté disponible para este.
- Cada que sale una actualización se requiere realizar la actualización en cada uno de los clientes con la aplicación instalada. (Euroinnova Formación; 2023; Blogs educativos primaria. Euroinnova Business School.)

2.8.3 Lenguajes de programación para el desarrollo de aplicaciones de escritorio

- *C#:* Lenguaje de programación desarrollado por *Microsoft* y ampliamente utilizado para el desarrollo de aplicaciones de escritorio en el *framework .NET.*
- *Java:* Lenguaje de programación popular que se utiliza para el desarrollo de aplicaciones de escritorio como empresariales.
- *C++:* Lenguaje de programación de propósito general y de alto rendimiento que se utiliza ampliamente en el desarrollo de aplicaciones de escritorio.
- *Python:* Lenguaje de programación que se utiliza en una gran variedad de aplicaciones, incluyendo el desarrollo de aplicaciones de escritorio.

- *JavaScript:* Es conocido principalmente en el desarrollo de aplicaciones *web,* aunque también se puede utilizar para el desarrollo de aplicaciones de escritorio utilizando *frameworks* como *Electron.* (Euroinnova Formación; 2023; Blogs educativos primaria. Euroinnova Business School.)

2.8.4 Frameworks para el desarrollo de aplicaciones de escritorio

En el artículo "¿Qué es un *framework* en programación?" Se menciona que estas son herramientas esenciales para los programadores gracias a que proporcionan una base sólida para construir proyectos completos que pueden reutilizarse. Son entornos de trabajo preestablecidos los cuales incluyen herramientas y características para acelerar el desarrollo de proyectos y facilitar el trabajo de los programadores (Felipe Cristancho, 2022). (Cristancho, F., Cristancho, F., & Cristancho, F; (2023); ¿Qué es un *framework* en programación? Actualizado 2023. *Talently* Blog.)

En el artículo "Los mejores 5 *frameworks* para crear aplicaciones *desktops* (escritorio) con *JavaScript, HTML, y CSS*" (Colectiva, N; (2022, 23 agosto); Los 5 mejores *frameworks* para crear aplicaciones *desktops* (Escritorio) con *JavaScript, HTML y CSS.*) se mencionan los siguientes *frameworks* que podrían utilizarse para desarrollar la aplicación deseada.

- *Electrón JS:* Es un *framework* que se puede utilizar para desarrollar aplicaciones multiplataforma, para utilizar este *framework* se requiere tener conocimientos sobre *HTML, Javascript y CSS.* Es una plataforma de código abierto que facilita el desarrollo de aplicaciones de escritorio mediante el uso de tecnologías *web*.

- *Tauri:* Es un *framework* que se puede utilizar para el desarrollo de aplicaciones de escritorio para todos los principales sistemas operativos, permite crear aplicaciones más rápidas y ligeras utilizando lenguaje de programación *RUST.*

- *Neutralino JS:* Es un *framework* con el que se puede empezar en el mundo del desarrollo de aplicaciones de escritorio, viene con un *SDK* y una *API* nativa de *JavaScript* la cual te permite tener acceso a funciones del sistema operativo, para utilizar este *framework* es necesario tener conocimientos sobre *HTML, JavaScript y CSS.* (Colectiva, N; (2022, 23 agosto); Los 5 mejores *frameworks* para crear aplicaciones *desktops* (Escritorio) con *JavaScript, HTML y CSS.*)

En el artículo "*Frameworks* de *Python* para *Desktop* se mencionan los siguientes *frameworks* para desarrollar la aplicación de escritorio utilizando lenguaje *Python:*

- *Tkinter: Tkinter* es uno de los *frameworks* más populares para *Python,* es la combinación de *TK,* un *toolkit* que te permite crear diferentes *Widgets* gráficos y realizar la manipulación de estos por medio de código *Python,* este *framework* es muy útil para empezar a crear aplicaciones *desktops* con *Python* ya que permite diseñar interfaces básicas.

- *PyQT:* Es un *framework* para diseñar aplicaciones de escritorio más avanzadas gracias a que está basada en un *framework* multiplataforma, incluye un programa con el cual podrás diseñar de manera visual tus interfaces, las *API´S* que posee nos permiten utilizar protocolos de red o módulos para conectarse a bases de datos y más opciones.

- *WxPython:* Es un *framework* muy similar a *PyQT* que permite desarrollar aplicaciones de escritorio multiplataforma, la forma de programar con este *framework* es orientada a objetos además de ser *Open Source* por lo que te permitirá desarrollar lo que gustes. (*Frameworks* de *Python* para *desktop* | *FAZT Web;* (2022, 6 noviembre).)

Capítulo 3 Metodología para el Diseño de Sistemas Integrados o Embebidos

Para la implementación de este prototipo se tomó en cuenta una adaptación del modelo en V para el desarrollo de sistemas embebidos, la cual consta de 7 etapas, en las cuales se parte de un análisis y diseño, siguiendo una implementación y por último una depuración e integración final. Las etapas que tiene este modelo se muestran en la Figura 19.

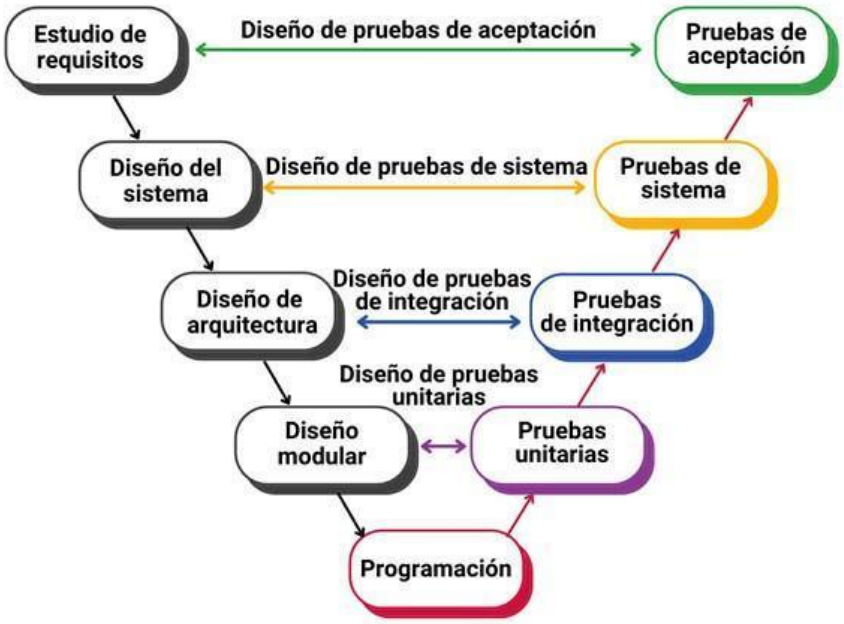

Figura 19. Modelo en V
Fuente: De Informática Profesional SA, S. (2023, 29 noviembre). *¿Qué es V-Model?* https://es.linkedin.com/pulse/qu%C3%A9-es-v-model-servicios-de-informatica-profesion-qotgf

Partiendo de la especificación de requisitos, se pretende definir y documentar los diferentes requerimientos del sistema a implementar siguiendo un diseño global el cual tiene como objetivo obtener una visión general del sistema. El diseño en detalle consiste en detallar cada bloque de la fase anterior, aquí se pretende especificar el diseño del sistema embebido, el receptor y la aplicación móvil, seguida de la implementación de cada uno de estos. El test unitario verifica cada módulo de HW y SW de manera individual, en donde se depurará cada uno de los módulos hasta obtener el resultado deseado. La fase de integración acopla los diferentes módulos

del sistema siguiendo el test operacional, en donde se realizan las últimas pruebas sobre un escenario real. (PEREZ, A; et al. “Una metodología para el desarrollo de *hardware* y *software* embebidos en sistemas críticos de seguridad”. *Systemics, Cybernetics and Informatics Journal*, vol 3, Num. 2, 2006, pp. 70-75.)

El desarrollo del sistema se realizará de la siguiente manera.

- Especificación de los diferentes requisitos de *Hardware* y *Software*: En esta sección se realizará la elección de los componentes del sistema.

- Diseño del sistema: En esta etapa se realizará el análisis funcional identificando todas las funciones que tendrá el sistema, además de realizar la especificación de cada uno de los elementos del sistema.

- Implementación: Los módulos diseñados en *Software* son integrados con el *Hardware.*

- *Test* Unitario: Se realizará la verificación de cada uno de los módulos por separado, realizando modificaciones en caso de ser necesario hasta conseguir el resultado esperado de cada uno de ellos.

- *Test* de Integración: Se realiza la verificación de todo el sistema funcionando en conjunto para comprobar que se cumplen los resultados deseados.

- *Test* Operacional: Se realizará la validación del sistema para comprobar que los resultados obtenidos cumplen con los requerimientos especificados.

3.1 Requerimientos

A continuación, se detalla cada uno de los requerimientos generales, funcionales y no funcionales que deberá de tener el sistema para asegurar el cumplimiento de cada uno de los objetivos planteados.

3.1.1 Requerimientos generales

Requerimientos generales del sistema				
ID	Nombre	Prioridad	Descripción	Origen
RG1	Aplicación de escritorio	Alta	Diseñar e implementar una aplicación de escritorio para la comunicación con el sistema en chip	Definición del problema
RG2	Lenguaje de programación	Alta	Diseñar un lenguaje de programación para el sistema a implementar	Definición del problema
RG3	Compilador	Alta	Diseñar e implementar un compilador para los programas realizados con el lenguaje de programación a diseñar	Definición del problema
RG4	Sistema en Chip	Alta	Realizar la implementación de un sistema embebido con tecnología *SoC* para permitir la comunicación con la aplicación de escritorio y realizar la manipulación del robot educativo.	Definición del problema
RG5	Robot educativo	Alta	Realizar la implementación de un robot elevador con la finalidad de manipularlo con el *SoC*	Definición del problema

Tabla 1. Requerimientos generales del sistema.

3.1.2 Requerimientos funcionales

Requerimientos funcionales				
ID	Nombre	Prioridad	Descripción	Origen
RF1	Aplicación de Escritorio	Alta	Desarrollo de la interfaz de escritorio para la interacción de los usuarios con el sistema	RG1
RF2	Mostrar piso actual	Media	Mostrar mediante el uso de un *display* el piso actual en el que se encuentra el elevador	RG5
RF3	Función para subir y bajar	Alta	El sistema deberá tener implementada una función para que el elevador pueda subir y bajar entre los diferentes pisos	RG5
RF4	Función para abrir y cerrar puertas	Alta	El sistema deberá tener implementada una función para que el elevador pueda abrir y cerrar sus puertas.	RG5
RF5	Interfaz de comunicación entre la aplicación y el *SoC*	Alta	Seleccionar la interfaz para realizar la comunicación entre la aplicación de escritorio y el *SoC*	RG1, RG4
RF6	Identificar el piso actual	Media	Instalar sensores para poder identificar el piso en el que se encuentra el elevador	RG5
RF7	Manejo de programas definidos con lenguaje natural	Alta	El sistema debe permitir el manejo de programas basados en lenguaje natural.	RG2
RF8	Funciones básicas para el manejo de los programas	Alta	El sistema deberá tener implementadas las siguientes funciones básicas para el manejo de los programas en lenguaje natural: crear, abrir, modificar, compilar, ejecutar, guardar y simular	RG1
RF9	Debe compilar los programas	Alta	El sistema deberá tener implementado un compilador para los programas creados con lenguaje natural	RG3
RF10	Mostrar el estado del editor	Media	El sistema deberá mostrar el estado del editor, si el programa pudo compilarse de manera correcta o no, si el programa se pudo guardar, etc.	RG1
RF11	Definir la trama entre las interfaces	Alta	Se debe definir la trama que tendrán tanto la aplicación de escritorio como la aplicación del *SoC* para el intercambio de datos.	RG1, RG4
RF12	Simulación	Alta	El sistema debe de permitir ver la simulación del comportamiento del elevador	RG1, RG5

Tabla 2. Requerimientos funcionales del sistema.

3.1.3 Requerimientos no funcionales

Requerimientos no funcionales del sistema				
ID	Nombre	Prioridad	Descripción	Origen
RNF1	Usabilidad	Alta	El sistema debe ser fácil de usar para los usuarios	RG1
RNF2	Portabilidad	Alta	El sistema debe de ser fácil de replicar y configurar para los usuarios	RG1, RG4, RG5
RNF3	Desempeño	Alta	La ejecución de los comandos debe realizarse antes de la llegada del siguiente comando.	RG1, RG2, RG3, RG4, RG5
RNF4	Robot elevador educativo	Alta	Selección de un robot educativo del tipo elevador para la manipulación de éste por medio del *SoC*	RG5
RFN5	Alimentación	Alta	El sistema debe funcionar con un mismo nivel de voltaje	RG4, RG5

Tabla 3. Requerimientos no funcionales del sistema.

3.2 Análisis de componentes

3.2.1 Display

Para poder mostrar el piso actual en el que se encuentra el elevador se deberá de utilizar un *display*, se tienen las siguientes opciones de *display.*

Tipo	*Display* de 7 segmentos	*LCD* alfanumérico	Matriz de *LED*
Uso	Simple y fácil de programar. Diseñado específicamente para números y letras.	Versátil y flexible. Puede mostrar texto y gráficos.	Permite diseños creativos.
Controlador	No requiere controlador	*I2C: PCF8574*	*MAX7219*
Voltaje de Operación	*7.2 a 8.5 VDC*	*3.3 a 5 VDC*	*4.7 a 5.3 VDC*
Corriente de Operación	*12 a 20 mA*	*90 a 120 mA*	*320 mA a 2 A*

Tabla 4. Comparación de *display.*

Tomando en cuenta que se necesita mostrar el piso actual además de la instrucción que se está ejecutando se tomó la decisión de utilizar un *display lcd* de 16x2 con el cual tenemos la facilidad de mostrar mensajes combinando letras, números y caracteres especiales, además de ser muy económicos y fáciles de conseguir.

Este *display* se puede controlar mediante el uso de un controlador *I2C PCF8574* por lo tanto también se utilizará dicho controlador para reducir la cantidad de pines que se necesitan por parte del sistema en chip para realizar el control de dicho *display.*

3.2.2 Módulo UART

Módulo	*FT232*	*CH9102F*	*CH340C*
Voltaje de Operación	*3.3 a 5 V*	*3.3 a 5 V*	*3.3 a 5 V*
Corriente de Operación	*10 a 25 mA*	*3 a 15 mA*	*3 a 9 mA*
Velocidad de Transmisión	*300 bps a 1 Mbps*	*4 Mbps*	*2 Mbps*
Protocolo	*USB y TTL*	*USB*	*USB*

Tabla 5. Comparación de módulos *UART*.

Considerando que los 3 tipos de módulo *UART* son muy similares se tomó la decisión de utilizar el módulo *FT232* ya que la reputación de este chip es muy conocida en comparación con los otros dos, por lo mismo me asegura una amplia disponibilidad del componente por lo cual lo hace la mejor opción para este proyecto.

3.2.3 Motor

Para el diseño del elevador se deben utilizar 2 motores los cuales van a realizar los movimientos de subir, bajar, abrir y cerrar puertas.

Características	Motor de *CA*	Motor de *CC*	Motor paso a paso
Control de velocidad	Es controlado utilizando inversores de frecuencia.	Control preciso utilizando variadores de velocidad.	Control preciso de pasos, velocidad y dirección.
Control de posición	Es menos preciso en comparación con los motores de *CC* y pasos.	Excelente control de posición y precisión.	Excelente control de posición y precisión.
Costo	Es menos costoso que los motores de *CC*.	Varía dependiendo de la calidad y la marca.	Varía según el tamaño y la marca.

Tabla 6. Comparación de motores.

Considerando que se necesita tener un control exacto sobre la posición del elevador al estar utilizando los motores se decidió utilizar motores de *CC*, utilizar este tipo de motores además de ser económicos nos permitirá tener un control más exacto sin batallar tanto como lo harías al utilizar otro tipo de motor.

A continuación, se presenta una tabla comparando algunas características de los diferentes motores de *CC.*

Característica	Motorreductor Biaxial Amarillo	Motor Tipo *Pololu*	Servomotor
Voltaje de Operación	*3 a 6 V*	*3 V*	*4.8 a 6 V*
Corriente de operación	*80 a 950 mA*	*0.045 a 0.4 mA*	*150 a 200 mA*
Torque	*0.7 kg/cm*	*0.16 kg/cm*	*1.8 kg/cm*

Tabla 7 Comparación de motores de *CC.*

Tomando en cuenta el voltaje de operación y la precisión que ofrecen los motores de tipo *Pololu* se tomó la decisión de utilizarlos ya que la precisión para el manejo del robot elevador será de vital importancia.

3.2.4 Sensores

Se utilizarán sensores para poder conocer la posición del elevador.

Característica	Sensor de Proximidad Magnético.	Sensor Ultrasónico.	Sensor Infrarrojo.	Sensor de efecto *Hall*.
Principio de detección.	Basado en campos magnéticos.	Emisión y detección de ondas ultrasónicas.	Emisión y detección de radiación infrarroja.	Basado en efecto *Hall* y campos magnéticos.
Precisión.	Buena precisión.	Buena en distancias cortas y medianas.	Adecuada para muchas aplicaciones.	Buena en campos magnéticos.
Costo.	Económico.	Varía dependiendo de la marca y alcance.	Económico.	Varía dependiendo de tipo y marca.
Facilidad de implementación.	Fácil de implementar.	Fácil de implementar.	Fácil de implementar.	Fácil de implementar.

Tabla 8. Comparación de sensores.

Tomando en cuenta que se necesitará utilizar un sensor por cada piso, se tomó la decisión de utilizar sensores de efecto *Hall* ya que en comparación con los demás sensores son más económicos además de tener más experiencia trabajando con ellos.

A continuación, se presenta una tabla comparando los diferentes sensores de efecto *Hall.*

Característica	*TLE49641MXTSA1*	*SS49E*	*Módulo KY-024*	*Módulo KY-024*
Voltaje de Operación	*3 a 32 V*	*5 a 8 V*	*3.3 a 5 V*	*3.3 a 5 V*
Salida	Digital	Analógica	Digital	Digital y Analógica
Temperatura de Operación	*-40°C a 170°C*	*-40°C a 100°C*	*-25°C a 85°C*	*-40°C a 85°C*

Tabla 9. Comparación de sensores de efecto *Hall.*

Tomando en cuenta el voltaje de operación, que se necesita trabajar con una salida digital y las dimensiones del sensor se utilizará el *TLE49641MXTSA1.*

3.2.5 Sistema en Chip (SoC)

El Sistema en Chip se utilizará para realizar la comunicación entre la aplicación de escritorio y el robot educativo.

Característica	*MSP430*	*PIC16F872*	*ESP8266*	*Raspberry Pi Pico*
Tipo de Dispositivo	Microcontrolador	Microcontrolador	*Módulo WI-FI*	Microcontrolador
Arquitectura	*RISC*	*RISC*	*Tensilica Xtensa*	*ARM Cortex-MO+*
Sistema Operativo	No	No	*Basado en Espressif SDK*	No
Lenguajes de Programación	C, ensamblador	C, ensamblador	*C, Python*	*MicroPython, C/C++*
Aplicaciones	Control de sistemas embebidos, sensores	Control de sistemas embebidos, aplicaciones industriales	*IoT*, desarrollo de aplicaciones *web*	Control de sistemas embebidos, proyectos *IoT*

Tabla 10. Comparación de *SoCs.*

Tomando en cuenta los componentes que se van a utilizar se decidió usar el sistema en chip *Raspberry Pi Pico*, además de ser económico, fácil de conseguir y de tener mayor experiencia con los lenguajes que se pueden utilizar para programarlo se tomó en cuenta la cantidad de pines de entrada y salida que se van a necesitar para realizar la manipulación de todos los componentes, en total se necesitan 23 pines de entrada y salida y este *SoC* cuenta con 26, por lo tanto es la mejor opción para trabajar en este proyecto.

3.3 Análisis de herramientas

Aplicación de escritorio:

Para el desarrollo de la aplicación de escritorio se tenían las opciones de utilizar lenguaje *Python o JavaScript*, el lenguaje que se seleccionó fue *Python* debido a que tras realizar una investigación se encontraron más cursos y tutoriales sobre desarrollo en *Python*, además de tener más experiencia trabajando con este lenguaje.

El *IDE* que se utilizará para el desarrollo de la aplicación será *Visual Studio Code,* debido a que se tiene experiencia previa trabajando con él.

El *framework* nos permitirá utilizar herramientas para diseño de componentes gráficos para la aplicación:

Framework	Ventajas	Desventajas
Tkinter	Incluido en la biblioteca estándar. Facilidad de uso. Documentación amplia. Multiplataforma.	Funcionalidad limitada. Menos *widgets.*
PyQt	Aplicaciones nativas. Gran comunidad y soporte. Multiplataforma.	Licencia. Curva de aprendizaje. Requisitos de instalación.
WxPython	Amplia selección de *Widgets.* Documentación completa. Multiplataforma.	Curva de aprendizaje. Menos popularidad. Problemas de empaquetamiento.

Tabla 11 Comparación de *frameworks* de desarrollo.

Tomando en cuenta las ventajas y desventajas de los diferentes *framework* se tomó la decisión de utilizar *Tkinter*, la aplicación a desarrollar no es muy compleja y la ventaja es que viene incluido con la instalación de *Python3* por lo tanto es la mejor opción para utilizar.

3.4 Lista de materiales

- 1 *Display LCD 16x2*
- 1 Controlador *I2C PCF8574*
- 1 Módulo *UART FT232*
- 2 Motores de *CC* tipo *Pololu*
- 1 Puente H
- 7 Sensores de Efecto *Hall TLE49641MXTSA1*
- 1 *SoC Raspberry Pi Pico*

3.5 Diseño del sistema. Arquitectura del sistema

3.5.1 Diagrama de componentes

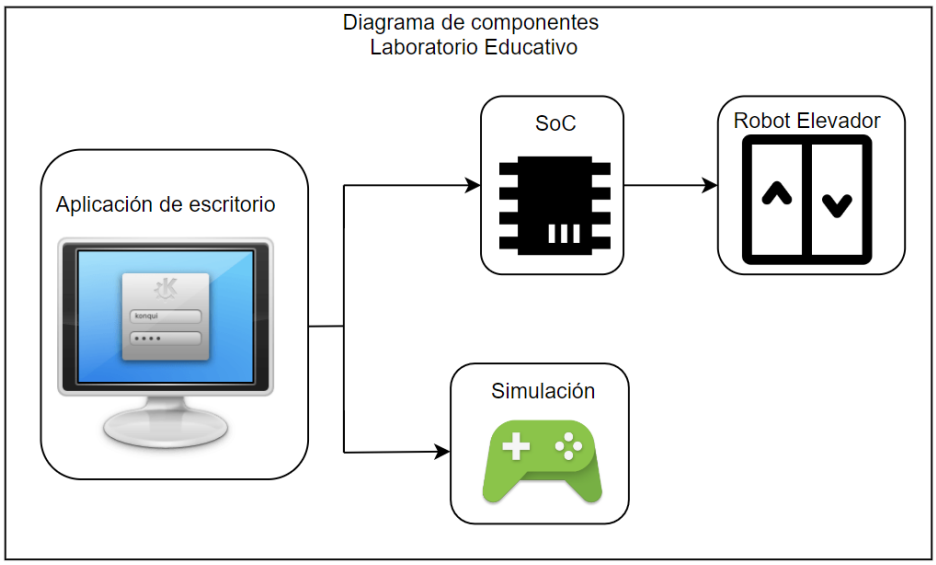

Figura 20. Diagrama general del sistema – Fuente: Elaboración propia

3.5.2 Arquitectura general

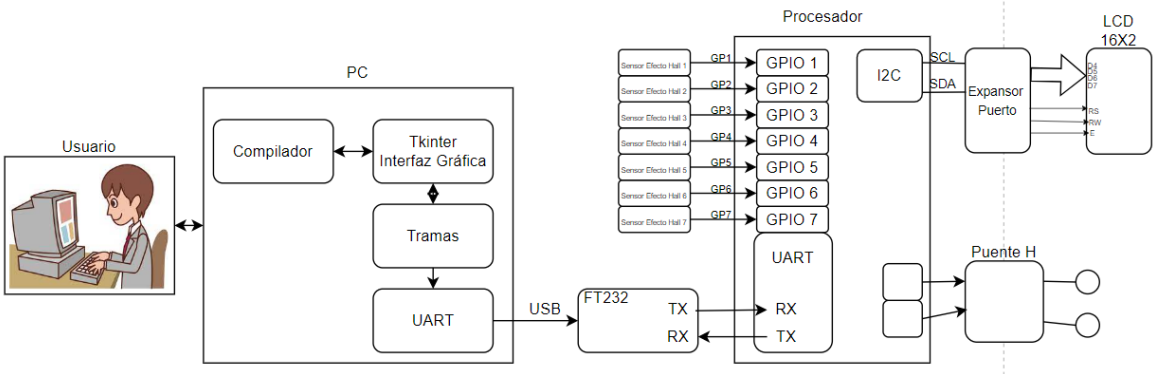

Figura 21. Diagrama de la arquitectura general del sistema. – Fuente: Elaboración propia

3.5.3 Diagrama a bloques del sistema

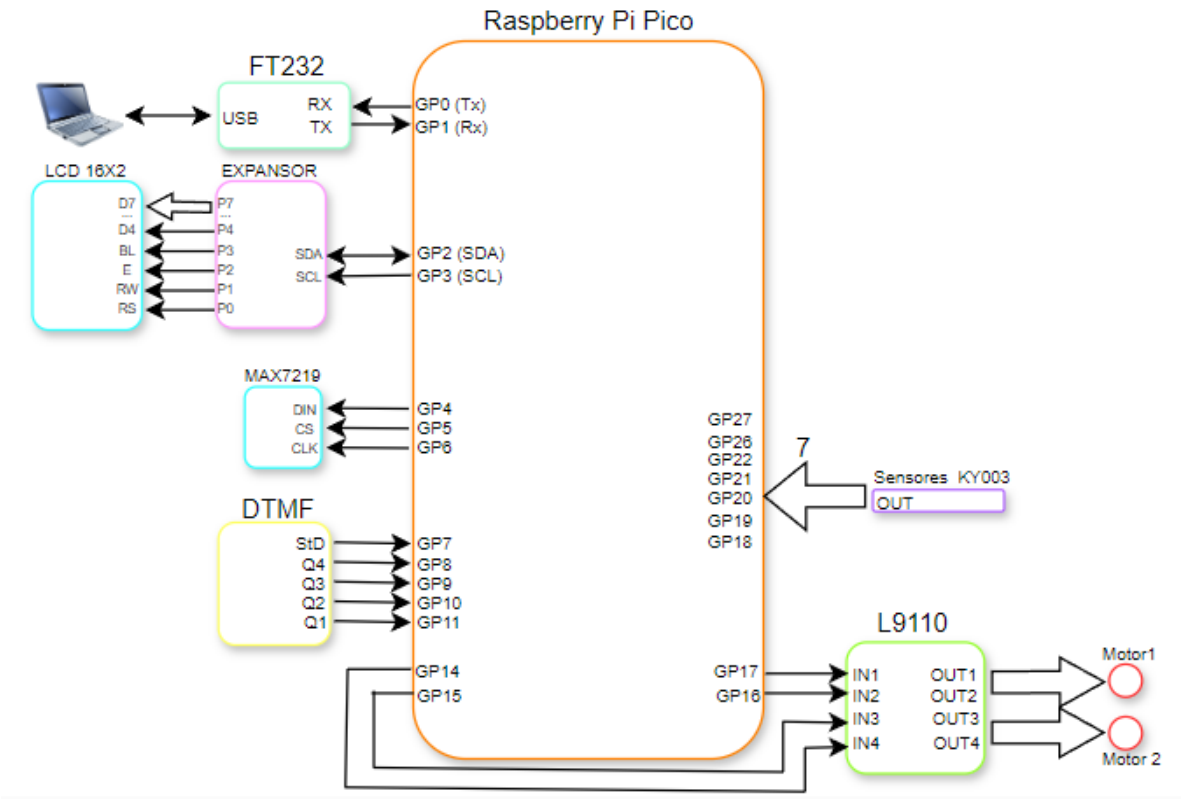

Figura 22. Diagrama de componentes del sistema físico. – Fuente: Elaboración propia

3.5.4 Diseño en detalle del sistema

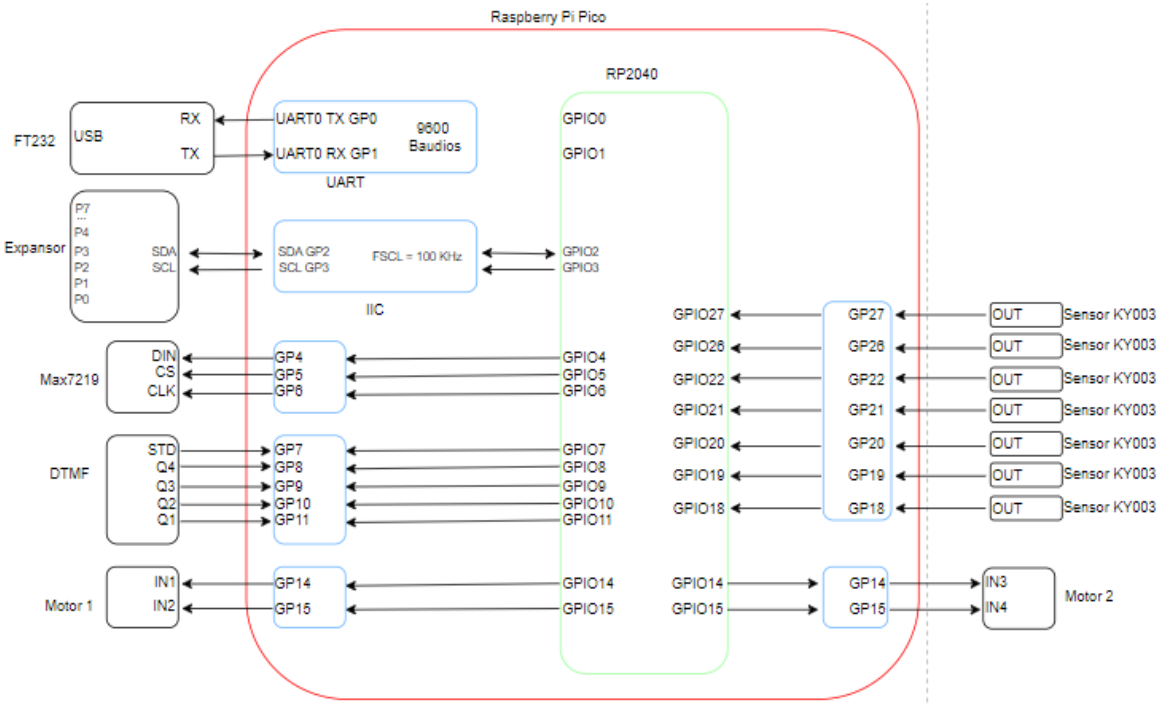

Figura 23. Diseño en detalle del sistema: Fuente: Elaboración propia

3.5.5 Diagrama de casos de uso

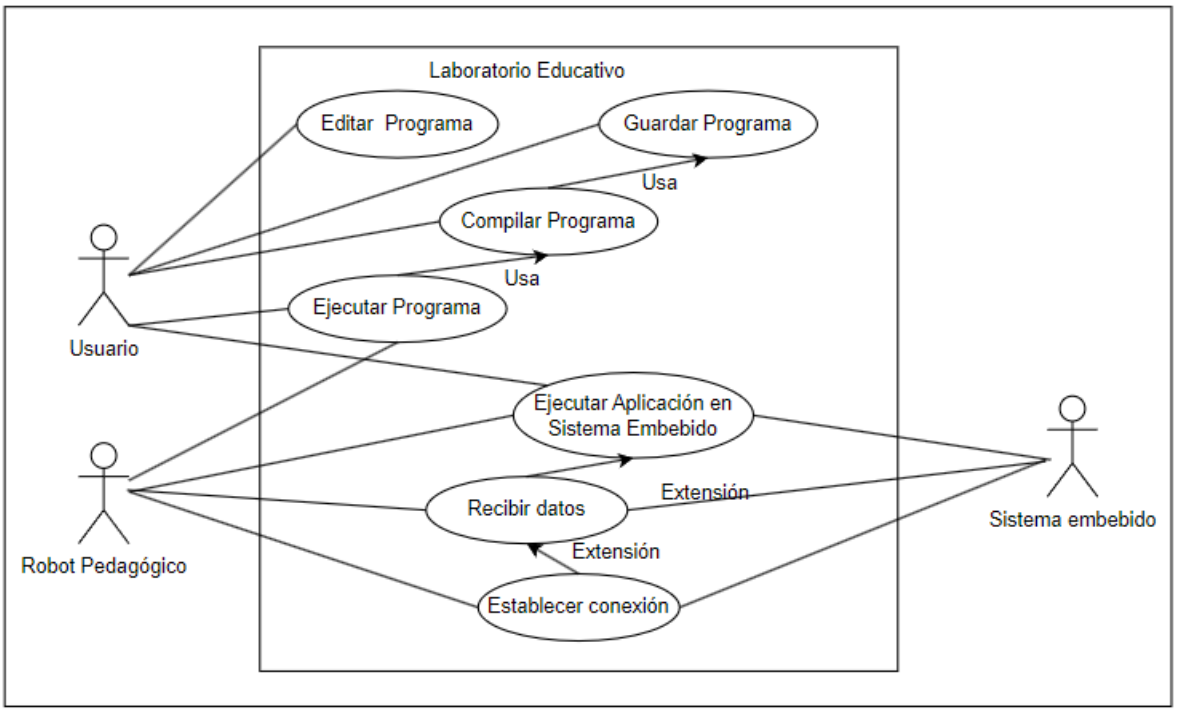

Figura 24. Diagrama de casos de uso. – Fuente: Elaboración propia

3.5.6 Diagrama Aplicación de Escritorio

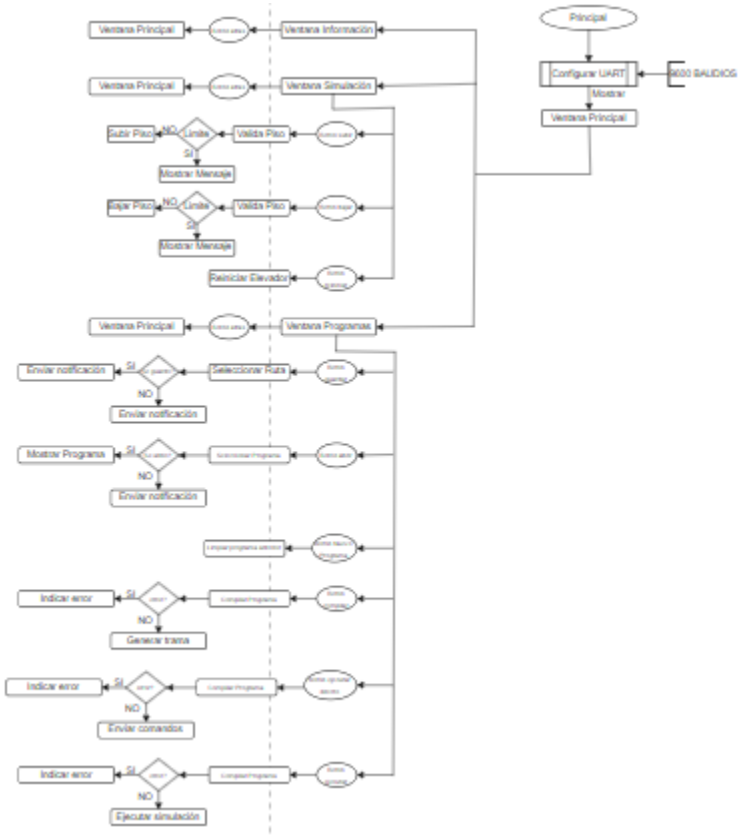

Figura 25. Diagrama de Aplicación de escritorio – Fuente: Elaboración propia

3.5.7 Diagrama de Aplicación en SoC

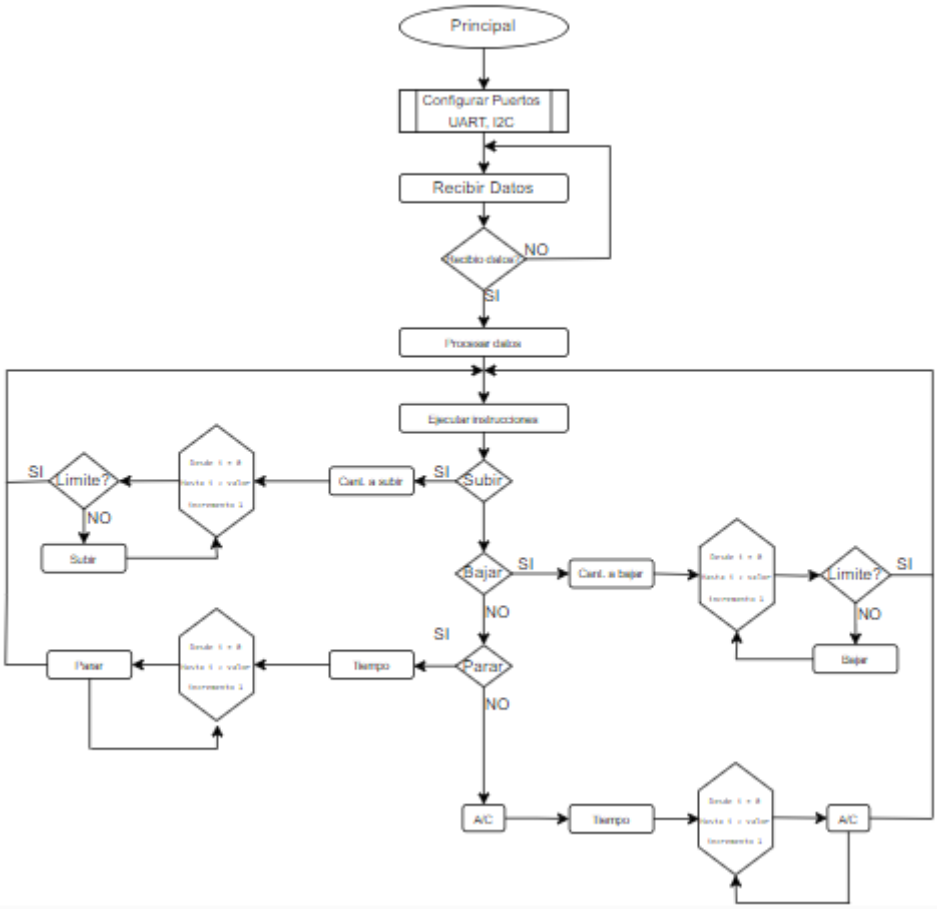

Figura 26. Diagrama de Aplicación en SoC – Fuente: Elaboración propia

3.5.8 Prototipo de la aplicación de escritorio

Para desarrollar el prototipo de la aplicación de escritorio se piensa utilizar el lenguaje Python, esté lenguaje incluye un *framework "Tkinter"* el cual permite realizar el desarrollo de aplicaciones con entorno gráfico.

Como se mencionó en el capítulo anterior el sistema se desarrollará siguiendo la metodología en V, y tendrá el siguiente flujo:

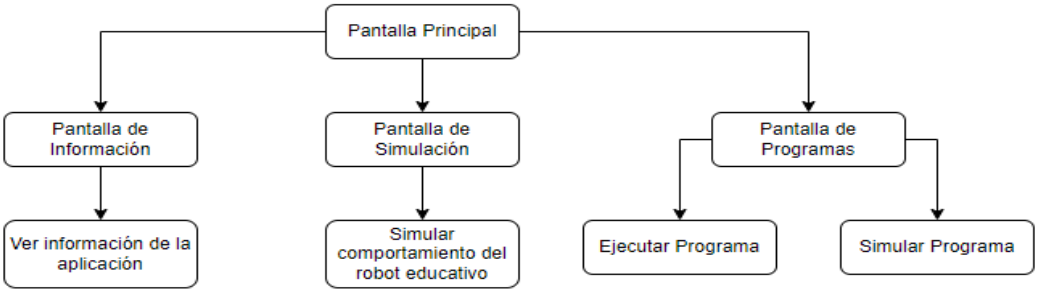

Figura 27. Diagrama de flujo de aplicación de escritorio – Fuente: Creación propia

3.6 Casos de uso

Caso de uso 1: Editar un programa.

Caso de uso	Editar un programa
Objetivo en contexto	El usuario procede a realizar la modificación de un programa existente.
Final exitoso	El programa fue modificado con éxito.
Final fallido	El programa no pudo ser modificado.
Autores principales	Usuario, Aplicación de escritorio
Autores secundarios	Compilador
Flujo	1- Ejecutar aplicación de escritorio. 2- Seleccionar opción de “Empezar”. 3- Seleccionar la opción “Abrir”. 4- Buscar y seleccionar programa a modificar. 5- Realizar modificaciones al programa abierto. 6- Seleccionar la opción “Guardar”.

Tabla 12. Caso de uso 1: Editar un programa.

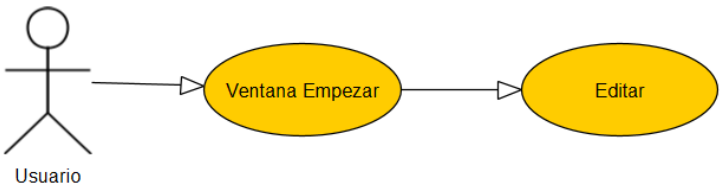

Figura 28. Caso de uso 1: Editar un programa. – Fuente: Elaboración propia

Caso de uso 2: Guardar un programa.

Caso de uso	Guardar un programa
Objetivo en contexto	El usuario procede a guardar los cambios o un programa nuevo.
Final exitoso	El programa fue guardado con éxito.
Final fallido	El programa no pudo ser guardado.
Autores principales	Usuario, Aplicación de escritorio
Autores secundarios	Compilador
Flujo	1- Ejecutar aplicación de escritorio. 2- Seleccionar opción de “Empezar”. 3- Seleccionar la opción “Abrir” o ingresar el nombre del programa nuevo. 4- En caso de seleccionar la opción “Abrir” buscar y seleccionar programa a modificar. 5- Realizar modificaciones al programa abierto, o ingresar código del programa nuevo. 6- Seleccionar la opción “Guardar”.

Tabla 13. Caso de uso 2: Guardar un programa.

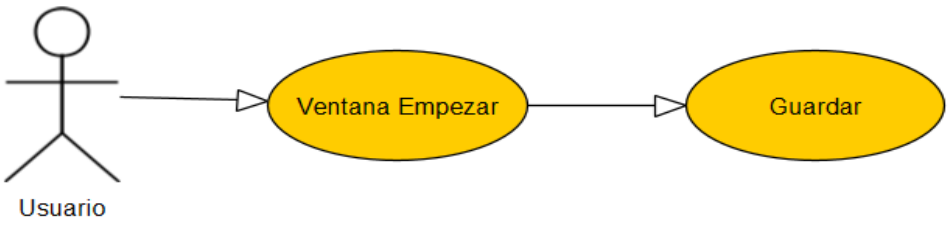

Figura 29. Caso de uso 2: Guardar un programa. – Fuente: Elaboración propia

Caso de uso 3: Compilar un programa.

Caso de uso	Compilar un programa
Objetivo en contexto	El usuario procede a realizar la compilación de un programa existente.
Final exitoso	El programa fue compilado con éxito.
Final fallido	El programa no pudo ser compilado.
Autores principales	Usuario, Aplicación de escritorio, compilador
Autores secundarios	N/A
Flujo	1- Ejecutar aplicación de escritorio. 2- Seleccionar opción de “Empezar”. 3- Seleccionar la opción “Abrir”. 4- Buscar y seleccionar programa a modificar. 5- Seleccionar la opción “Guardar”. 6- Seleccionar la opción “Compilar”.

Tabla 14. Caso de uso 3: Compilar un programa.

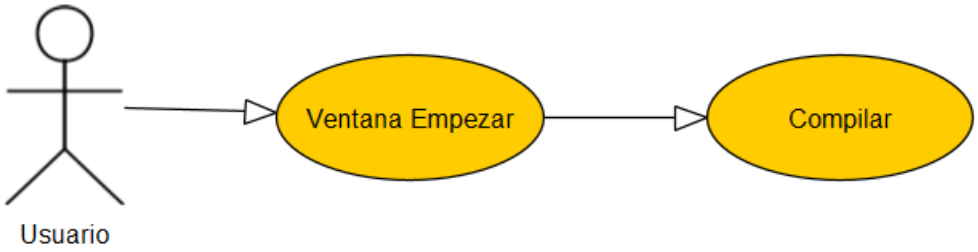

Figura 30. Caso de uso 3: Compilar un programa. – Fuente: Elaboración propia

Caso de uso 4: Ejecutar un programa.

Caso de uso	Ejecutar un programa
Objetivo en contexto	El usuario procede a realizar la ejecución de un programa existente.
Final exitoso	El programa fue ejecutado con éxito.
Final fallido	El programa no pudo ser ejecutado.
Autores principales	Usuario, Aplicación de escritorio, compilador
Autores secundarios	N/A
Flujo	1- Ejecutar aplicación de escritorio. 2- Seleccionar opción de “Empezar”. 3- Seleccionar la opción “Abrir”. 4- Buscar y seleccionar programa a modificar. 5- Seleccionar la opción “Guardar”. 6- Seleccionar la opción “Compilar”. 7- Seleccionar la opción “Ejecutar”. 8- Comunicar con aplicación del SoC. 9- Ejecución del SoC.

Tabla 15. Caso de uso 4: Ejecutar un programa.

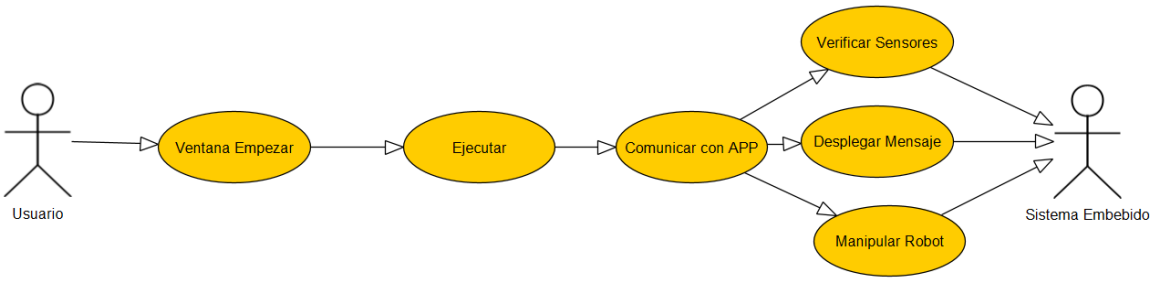

Figura 31. Caso de uso 4: Ejecutar un programa. – Fuente: Elaboración propia

Caso de uso 5: Ejecutar ventana de simulación.

Caso de uso	Ejecutar ventana de simulación
Objetivo en contexto	El usuario procede a realizar la ejecución de la ventana de simulación.
Final exitoso	Se ejecutó la simulación con éxito.
Final fallido	No se pudo realizar la simulación.
Autores principales	Usuario, Aplicación de escritorio
Autores secundarios	N/A
Flujo	1- Ejecutar aplicación de escritorio. 2- Seleccionar opción de “Simulación”. 3- Ingresar el piso inicial. 4- Proceder a realizar simulación.

Tabla 16. Caso de uso 5: Ejecutar ventana de simulación.

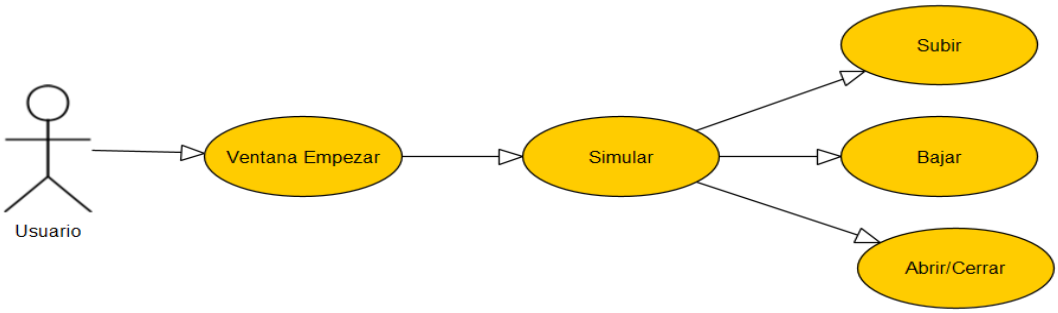

Figura 32. Caso de uso 5: Ejecutar ventana de simulación. – Fuente: Elaboración propia

Caso de uso 6: Ejecutar ventana de información.

Caso de uso	Ejecutar ventana de información
Objetivo en contexto	El usuario procede a realizar la ejecución de la ventana de información.
Final exitoso	Se visualizó la información con éxito.
Final fallido	No se pudo visualizar la información.
Autores principales	Usuario, Aplicación de escritorio
Autores secundarios	N/A
Flujo	1- Ejecutar aplicación de escritorio. 2- Seleccionar opción de “información”.

Tabla 17. Caso de uso 6: Ejecutar ventana de información.

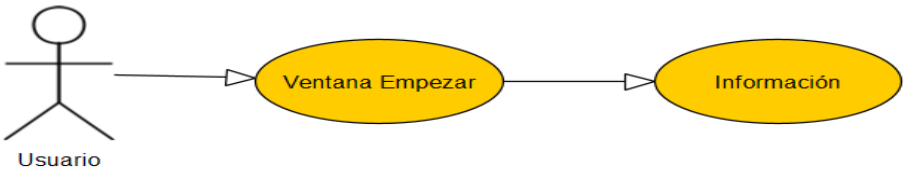

Figura 33. Caso de uso 6: Ejecutar ventana de información. – Fuente: Elaboración propia

Capítulo 4 Implementación

4.1 Implementación SoC Raspberry Pi Pico

La funcionalidad para la tarjeta *Raspberry Pi Pico* se programó en lenguaje *MicroPython* utilizando el *IDE Thonny* a continuación se muestra el código implementado:

```
#Función para recibir los datos de la aplicacion de escritorio mediante protocolo UART
def recibir_datos():
    global datos_recibidos, bandera
    del datos_recibidos[:]
    bandera = True
    print("Recibiendo datos")
    mostrar_lcd("Recibiendo datos")
    while bandera:
        if uart.any():
            data = uart.readline()
            instruccion = data.decode().strip()
            if instruccion == '32 200':
                bandera = False
            datos_recibidos.append(instruccion)
            print("instruccion: " + instruccion)
        time.sleep(0.1)
    print("tamaño: " + str(len(datos_recibidos)))
```

Figura 34. Función para recibir datos mediante *UART* – Fuente: Elaboración propia

```
#Función que obtiene las instrucciones y los valores de ejecucion para el control del robot
def obtener_instrucciones():
    global datos_recibidos, instrucciones, valores
    del instrucciones[:]
    del valores[:]
    for i in range(1, len(datos_recibidos)-1):
        partes = datos_recibidos[i].split(" ")
        instrucciones.append(partes[0])
        valores.append(partes[1])
    print(instrucciones)
    print(valores)
```

Figura 35. Función para obtener las instrucciones recibidas mediante *UART* – Fuente: Elaboración propia

```
#Función para mostrar mensajes en el Display LCD 16x2
def mostrar_lcd(cadena):
    lcd.backlight_off()
    lcd.backlight_on()
    lcd.clear()
    lcd.putstr(cadena)
```

Figura 36. Función para mostrar mensaje en el *Display LCD* 16x2 – Fuente: Elaboración propia

```
#Función para subir
def subir(valor):
    global piso_actual
    piso_destino = piso_actual + int(valor)
    print("destino: " + str(piso_destino))
    mostrar_lcd("  Instruccion        Subir")
    bandera_piso = True
    motor1.value(1)
    motor2.value(0)
    while bandera_piso:
        if piso_destino == 1:
            if piso1.value() == 0:
                piso_actual = 1
                bandera_piso = False
        elif piso_destino == 2:
            if piso2.value() == 0:
                piso_actual = 2
                bandera_piso = False
        elif piso_destino == 3:
            if piso3.value() == 0:
                piso_actual = 3
                bandera_piso = False
        elif piso_destino == 4:
            if piso4.value() == 0:
                piso_actual = 4
                bandera_piso = False
        elif piso_destino == 5:
            if piso5.value() == 0:
                piso_actual = 5
                bandera_piso = False
        elif piso_destino == 6:
            if piso6.value() == 0:
                piso_actual = 6
                bandera_piso = False
        elif piso_destino == 7:
            if piso7.value() == 0:
                piso_actual = 7
                bandera_piso = False
```

Figura 37. Función para subir el robot elevador – Fuente: Elaboración propia

```
        elif piso_destino > 7:
            if piso7.value() == 0:
                piso_actual = 7
                mostrar_lcd("No se puede subir más")
                bandera_piso = False
    motor1.value(0)
    motor2.value(0)
    time.sleep(1)
```

Figura 38. Función para subir el robot elevador – Fuente: Elaboración propia

```
#Funcion para bajar
def bajar(valor):
    global piso_actual
    bandera_piso = True
    piso_destino = piso_actual - int(valor)
    print("destino: " + str(piso_destino))
    mostrar_lcd("  Instruccion        Bajar")
    motor1.value(0)
    motor2.value(1)
    while bandera_piso:
        if piso_destino == 1:
            if piso1.value() == 0:
                piso_actual = 1
                bandera_piso = False
        elif piso_destino == 2:
            if piso2.value() == 0:
                piso_actual = 2
                bandera_piso = False
        elif piso_destino == 3:
            if piso3.value() == 0:
                piso_actual = 3
                bandera_piso = False
        elif piso_destino == 4:
            if piso4.value() == 0:
                piso_actual = 4
                bandera_piso = False
        elif piso_destino == 5:
```

Figura 39. Función para bajar el robot elevador – Fuente: Elaboración propia

```
            if piso5.value() == 0:
                piso_actual = 5
                bandera_piso = False
        elif piso_destino == 6:
            if piso6.value() == 0:
                piso_actual = 6
                bandera_piso = False
        elif piso_destino == 7:
            if piso7.value() == 0:
                piso_actual = 7
                bandera_piso = False
        elif piso_destino < 0:
            if piso1.value() == 0:
                piso_actual = 1
                mostrar_lcd("No se puede bajar más")
                bandera_piso = False
    motor1.value(0)
    motor2.value(0)
    time.sleep(1)
```

Figura 40. Función para bajar el robot elevador – Fuente: Elaboración propia

```
#Función para reiniciar la posicion del robot hacia el pi
def reiniciar():
    global piso_actual
    bandera_piso = True
    mostrar_lcd("Reiniciando...")
    motor1.value(0)
    motor2.value(1)
    while bandera_piso:
        if piso1.value() == 0:
            piso_actual = 1
            bandera_piso = False
    motor1.value(0)
    motor2.value(0)
    time.sleep(1)
```

Figura 41. Función para reiniciar el elevador al piso 1 – Fuente: Elaboración propia

```
#Función para detener el robot
def parar(valor):
    mostrar_lcd("  Instruccion        Parar")
    motor1.value(0)
    motor2.value(0)
    time.sleep(int(valor))
```

Figura 42. Función para detener el robot elevador – Fuente: Elaboración propia

```
#Función para abrir las puertas del robot
def abrir(valor):
    mostrar_lcd("  Instruccion   Abrir Puertas")
    led.value(1)
    time.sleep(int(valor))
    led.value(0)
```

Figura 43. Función para abrir puertas del robot elevador – Fuente: Elaboración propia

```
#Función para realizar el control de la instrucción a eje
def control(instruccion, valor):
    if instruccion == '0':
        parar(valor)
    elif instruccion == '1':
        subir(valor)
    elif instruccion == '2':
        bajar(valor)
    elif instruccion == '4':
        abrir(valor)
```

Figura 44. Función para controlar la instrucción a ejecutar – Fuente: Elaboración propia

```
#Función para ejecutar las instrucciones
def ejecutar():
    rango = len(instrucciones)
    for i in range(0,rango):
        print("Ejecutando: " + instrucciones[i] + " " + valores[i])
        control(instrucciones[i], valores[i])
```

Figura 45. Función para realizar la ejecución del robot elevador – Fuente: Elaboración propia

```
#Ejecución del robot
if __name__ == "__main__":
    while True:
        recibir_datos()
        reiniciar()
        obtener_instrucciones()
        obtener_piso_actual()
        ejecutar()
        mostrar_lcd("Fin de ejecucion")
```

Figura 46. Ciclo de ejecución para el control del robot elevador – Fuente: Elaboración propia

4.2 Implementación aplicación de escritorio

La aplicación de escritorio fue desarrollada utilizando lenguaje *Python* con el *IDE* de programación *Visual Studio Code*:

```
def detener_elevador():
    print("Ejecutando parar elevador")
    instruccion.config(state=tkinter.NORMAL)
    instruccion.delete(0, tkinter.END)
    instruccion.insert(0, "Deteniendo")
    instruccion.config(state=tkinter.DISABLED)
```

Figura 47. Función para detener el elevador durante la simulación – Fuente: Elaboración propia

```
def abrir_puertas():
    print("Ejecutando abrir puertas")
    instruccion.config(state=tkinter.NORMAL)
    instruccion.delete(0, tkinter.END)
    instruccion.insert(0, "Abriendo")
    instruccion.config(state=tkinter.DISABLED)
```

Figura 48. Función para abrir las puertas del elevador durante la simulación – Fuente: Elaboración propia

```
def desplazar_imagen_arriba():
    global pActual, imagen_id, pisoActual, ventanaSimulacion, instruccion
    time.sleep(1)
    print("Ejecutando desplazar arriba")
    instruccion.config(state=tkinter.NORMAL)
    instruccion.delete(0, tkinter.END)
    instruccion.insert(0,"Subiendo")
    instruccion.config(state=tkinter.DISABLED)
    nueva_x = cElevador.coords(imagen_id)[0]
    if pActual < 7:
        nueva_y = cElevador.coords(imagen_id)[1] - 60
    else:
        nueva_y = cElevador.coords(imagen_id)[1]
    pActual = pActual + 1
    pisoActual.config(state=tkinter.NORMAL)
    pisoActual.delete(0, tkinter.END)
    pisoActual.insert(0,pActual)
    pisoActual.config(state=tkinter.DISABLED)
    valida_pActual()
    cElevador.coords(imagen_id, nueva_x, nueva_y)
```

Figura 49. Función para subir el elevador durante la simulación – Fuente: Elaboración propia

```python
def desplazar_imagen_abajo():
    time.sleep(1)
    global pActual, imagen_id, pisoActual, ventanaSimulacion, instruccion
    print("Ejecutando desplazar abajo")
    instruccion.config(state=tkinter.NORMAL)
    instruccion.delete(0, tkinter.END)
    instruccion.insert(0,"Bajando")
    instruccion.config(state=tkinter.DISABLED)
    nueva_x = cElevador.coords(imagen_id)[0]
    if pActual > 0:
        nueva_y = cElevador.coords(imagen_id)[1] + 60
    else:
        nueva_y = cElevador.coords(imagen_id)[1]
    pActual = pActual - 1
    pisoActual.config(state=tkinter.NORMAL)
    pisoActual.delete(0, tkinter.END)
    pisoActual.insert(0,pActual)
    pisoActual.config(state=tkinter.DISABLED)
    valida_pActual()
    cElevador.coords(imagen_id, nueva_x, nueva_y)
```

Figura 50. Función para bajar el elevador durante la simulación – Fuente: Elaboración propia

```python
def reiniciar_imagen():
    global pActual, imagen_id, pisoActual, ventanaSimulacion
    print("Ejecutando reiniciar")
    pActual = 0
    pisoActual.config(state=tkinter.NORMAL)
    pisoActual.delete(0, tkinter.END)
    pisoActual.insert(0,pActual)
    pisoActual.config(state=tkinter.DISABLED)
    cElevador.coords(imagen_id,10,460)
```

Figura 51. Función para regresar el elevador a la planta baja durante la simulación – Fuente: Elaboración propia

```python
def valida_pActual():
    global pActual, imagen_id, pisoActual, ventanaSimulacion
    if pActual > 7:
        pActual = 7
        pisoActual.config(state=tkinter.NORMAL)
        pisoActual.delete(0, tkinter.END)
        pisoActual.insert(0,pActual)
        pisoActual.config(state=tkinter.DISABLED)
        messagebox.showinfo("Piso Actual", "Piso 7 alcanzado no se puede subir más")
    elif pActual < 0:
        pActual = 0
        pisoActual.config(state=tkinter.NORMAL)
        pisoActual.delete(0, tkinter.END)
        pisoActual.insert(0,pActual)
        pisoActual.config(state=tkinter.DISABLED)
        messagebox.showinfo("Piso Actual", "Piso 0 alcanzado no se puede bajar más")
```

Figura 52. Función para obtener el piso actual del elevador – Fuente: Elaboración propia

```
def funcGuardar():
    archivo_guardado = filedialog.asksaveasfile(
        title="Guardar Programa",
        initialdir="C:/Users/LENKABITS/Desktop/Tesis",
        filetypes=(("Programas", "*.txt"), ("Programas", "*.txt")),
        initialfile=nombrePrograma.get(),
        defaultextension= ".txt"
    )
    nomProg = archivo_guardado.name
    posNP = nomProg.rfind('/')
    nomProgF = nomProg[posNP + 1:]
    nombrePrograma.delete(0, "end")
    nombrePrograma.insert(0, nomProgF)
    contenido = textoPrograma.get("1.0","end-1c")
    if archivo_guardado:
        with open(archivo_guardado.name, 'w') as archivo:
            archivo.write(contenido)
    notificacion("Programa: " + nomProgF + " guardado correctamente")
```

Figura 53. Función para guardar el programa creado por el usuario – Fuente: Elaboración propia

```
def funcCompilar():
    global prog_comp, pCompilado
    contenido = textoPrograma.get("1.0", "end-1c")
    lineas = contenido.splitlines()
    prog_comp.clear()
    for valor in lineas:
        prog_comp.append(valor.replace(" ",""))
    valida_instrucciones()
    if pCompilado == True:
        genera_trama()
```

Figura 54. Función para compilar el programa creado por el usuario – Fuente: Elaboración propia

```
def genera_trama():
    global prog_comp, tramaProg, tramaProg2, rutaComandos
    ruta_trama = "C:/Users/LENKABITS/Desktop/Tesis/trama_" + nomProgAbiertoF
    rutaComandos = ruta_trama
    print("ruta:" + rutaComandos)
    for i in prog_comp:
        if i[0] == 'i' or i[0] == 'I':
            tramaProg.append(16)
            tramaProg2.append(100)
        if i[0] == 'f' or i[0] == 'F':
            tramaProg.append(32)
            tramaProg2.append(200)
        if i[0] == 's' or i[0] == 'S':
            tramaProg.append(1)
            tramaProg2.append(i[1])
        if i[0] == 'b' or i[0] == 'B':
            tramaProg.append(2)
            tramaProg2.append(i[1])
        if i[0] == 'p' or i[0] == 'P':
            tramaProg.append(0)
            tramaProg2.append(i[1])
        if i[0] == 'a' or i[0] == 'A':
            tramaProg.append(4)
            tramaProg2.append(i[1])
    with open(ruta_trama, 'w') as archT:
        j = 0
        for i in prog_comp:
            archT.write(str(tramaProg[j]) + " " + str(tramaProg2[j]) + "\n")
            j = j + 1
```

Figura 55. Función para generar la trama del programa compilado – Fuente: Elaboración propia

```python
def funcEjecutar():
    global ventanaSimulacion, pCompilado
    if pCompilado == True:
        print("Ejecutando")
        fVenatanaSimulacion()
        ventanaSimulacion.wait_visibility()
        print("Ejecutando instrucciones")
        hilo = threading.Thread(target=ventanaSimulacion.after(300,ejecutar_instruccion()))
        hilo.start()
        hilo2 = threading.Thread(target=ventanaSimulacion.after(300, enviar_comandos()))
        #ventanaSimulacion.after(300,ejecutar_instruccion())
        print("Fin instrucciones")
    notificacion("Se ejecutó el programa")
```

Figura 56. Función para ejecutar las instrucciones programadas por el usuario – Fuente: Elaboración propia

```python
def funcAbrirP():
    global nombrePrograma, textoPrograma, nomProgAbiertoF
    archivo_abierto = filedialog.askopenfilename(
        title="Abrir Programa",
        initialdir="C:/Users/LENKABITS/Desktop/Tesis",
        filetypes=(("Programas", "*.txt"), ("Programas", "*.txt"))
    )
    nomProgAbierto = archivo_abierto
    posNPA = nomProgAbierto.rfind('/')
    nomProgAbiertoF = nomProgAbierto[posNPA + 1:]
    nombrePrograma.delete(0,"end")
    nombrePrograma.insert(0,nomProgAbiertoF)
    if archivo_abierto:
        with open(archivo_abierto, 'r') as archivo:
            lineas = archivo.readlines()
        textoPrograma.delete('1.0',tkinter.END)
        for linea in lineas:
            textoPrograma.insert(tkinter.END, linea)
    notificacion("Programa: " + nomProgAbiertoF + " abierto correctamente")
```

Figura 57. Función para abrir los programas guardados por el usuario – Fuente: Elaboración propia

```python
def enviar_comandos():
    global rutaComandos
    print("ruta:" + rutaComandos)
    #rutaComandos = "C:/Users/adria/OneDrive/Escritorio/Tesis/trama_Prueba_PROGRAMA.txt"
    try:
        with open(rutaComandos, 'r') as comandos:
            for linea in comandos:
                comand = linea
                puerto_serie.write(comand.encode())
#           puerto_serie.close()
    except FileNotFoundError:
        print(rutaComandos)
```

Figura 58. Función para enviar comandos por *UART* – Fuente: Elaboración propia

```python
ventanaPrincipal.mainloop()
```

Figura 59. Ejecución de la aplicación de escritorio – Fuente: Elaboración propia

Capítulo 5 Pruebas y Conclusión

5.1 Pruebas unitarias

5.1.1 Sensor de efecto Hall

Para determinar el piso en el que se encuentra el elevador se utilizarán sensores de efecto *Hall*, por lo cual se tomará la distancia a la que el sensor se activa utilizando imanes de neodimio de (15x2) [MM] y (8x2) [MM] y teniendo una alimentación de 3.3 y 5 *volts* para los sensores, a continuación, se muestran 2 tablas con las mediciones de las pruebas realizadas con el sensor de efecto *Hall.*

15 [MM] x 2 [MM]

Cantidad de imanes	3.3 [V]	5 [V]
5	2.11 [CM]	2.11 [CM]
4	2.11 [CM]	2.11 [CM]
3	1.45 [CM]	1.45 [CM]
2	1.04 [CM]	1.04 [CM]
1	0.8 [CM]	0.8 [CM]

Tabla 18. Pruebas de sensor de efecto *Hall* con imanes de 15 [MM] x 2 [MM]

8 [MM] x 2 [MM]

Cantidad de imanes	3.3 [V]	5 [V]
5	1.35 [CM]	1.35 [CM]
4	1.35 [CM]	1.35 [CM]
3	0.89 [CM]	0.89 [CM]
2	0.75 [CM]	0.75 [CM]
1	0.62 [CM]	0.62 [CM]

Tabla 19. Pruebas de sensor de efecto *Hall* con imanes de 8 [MM] x 2 [MM]

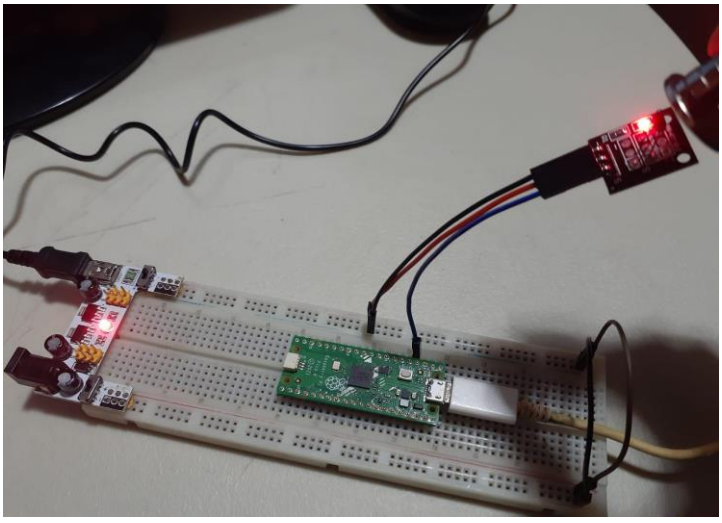

Figura 60. Prueba unitaria sensor de efecto *Hall* – Fuente: Elaboración propia

5.1.2 Display LCD 16 x 2

El *display LCD* se utilizará para mostrar mensajes sobre el funcionamiento del robot elevador, por lo tanto, se realizará la prueba del funcionamiento del *display* mostrando un mensaje de prueba.

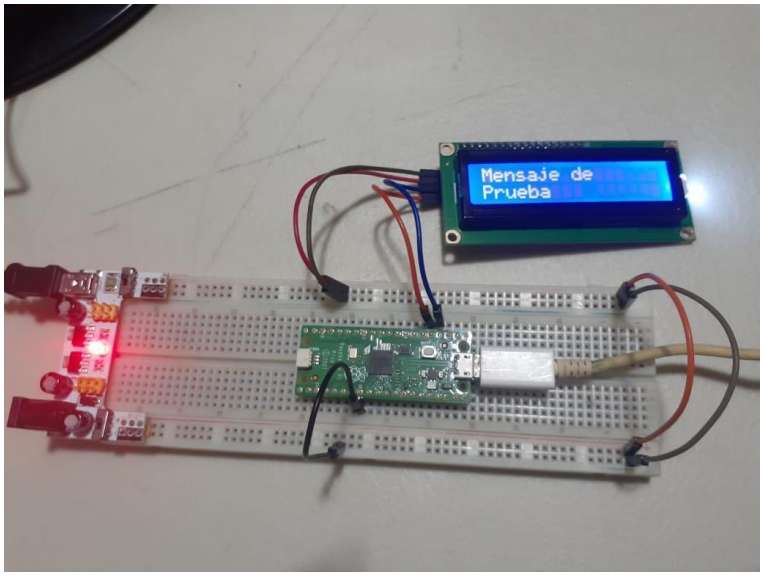

Figura 61. Prueba unitaria *display LCD* 16x2 – Fuente: Elaboración propia

5.1.3 Motor

El motor se utilizará para poder mover el robot elevador entre pisos por lo cual se realizará la prueba del funcionamiento de giro en ambos sentidos para simular la acción de subir y bajar, para esto el motor será controlado mediante un puente H el cual recibirá la instrucción de girar hacia un lado u otro.

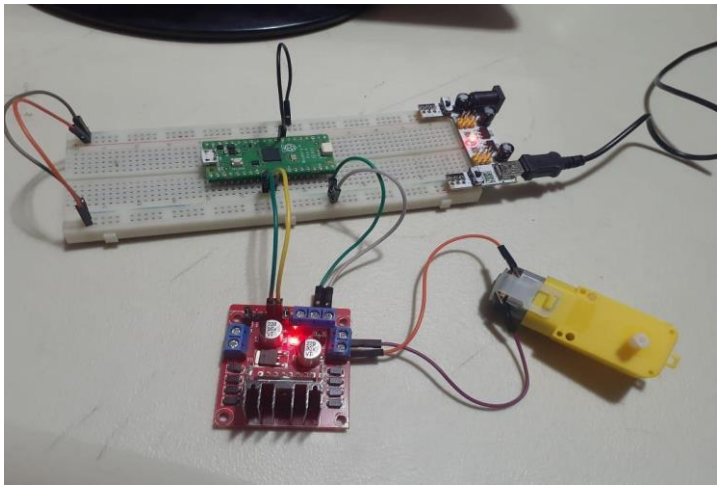

Figura 62. Prueba unitaria motor – Fuente: Elaboración propia

5.1.4 Módulo FT232

La comunicación entre la aplicación de escritorio y la tarjeta *Raspberry Pi Pico* se realizará utilizando el protocolo *UART* del módulo *FT232* por lo cual se realizará la prueba para validar la conexión entre la computadora y la tarjeta y el envío correcto de datos de la computadora hacia la tarjeta.

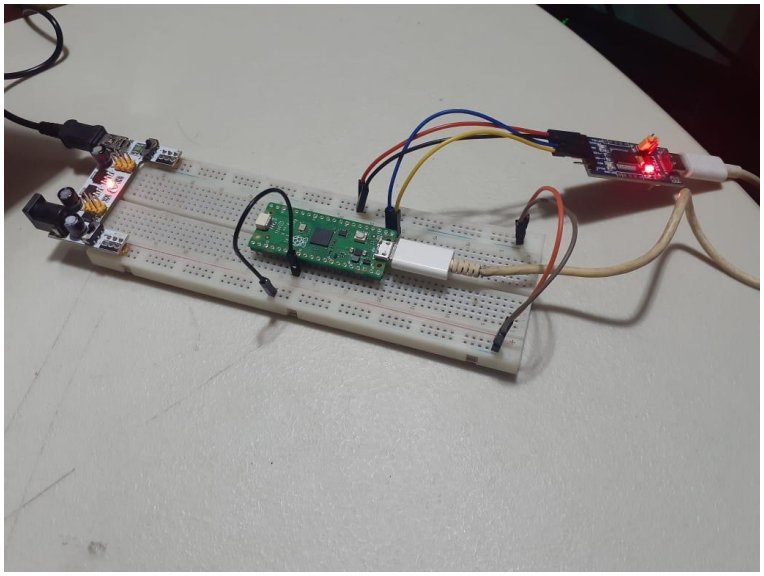

Figura 63. Prueba unitaria módulo *FT232* – Fuente: Elaboración propia

```
Consola
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Recibiendo datos
instruccion: 16 100
instruccion: 1 2
instruccion: 0 3
instruccion: 2 1
instruccion: 4 3
instruccion: 1 3
instruccion: 2 1
instruccion: 1 2
instruccion: 32 200
```

Figura 64. Prueba unitaria datos recibidos – Fuente: Elaboración propia

5.2 Pruebas de integración

5.2.1 Hardware

Antes de realizar la prueba entre el *hardware* y la aplicación de escritorio se grabará en la tarjeta las instrucciones de un programa que la aplicación debería de mandar a la tarjeta y se ejecutará de forma automática para validar el correcto funcionamiento de todos los componentes.

Programa a ejecutar

```
I
S 2
P 3
B 1
A 3
s 3
B 1
S 2
F
```

Figura 65. Programa a ejecutar – Fuente: Elaboración propia

Trama generada y recibida en la tarjeta

```
16 100
1 2
0 3
2 1
4 3
1 3
2 1
1 2
32 200
```

Figura 66. Trama del programa a ejecutar – Fuente: Elaboración propia

La trama será guardada en un arreglo antes de iniciar la ejecución con lo cual se sustituye el proceso de envío de información entre la aplicación y la tarjeta todo el demás proceso se deja tal y como esta.

```
#Variables
bandera = True
#datos_recibidos = []
datos_recibidos = ["16 100", "1 2", "0 3", "2 1", "4 3", "1 3", "2 1", "1 2", "32 200"]
instrucciones = []
valores = []
I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16
piso_actual = 1
```

Figura 67. Trama guardada dentro del código de la *Raspberry Pi Pico* – Fuente: Elaboración propia

5.3 Pruebas de usuario

Una vez terminada la aplicación tanto de escritorio como del Sistema en Chip se realizarán las pruebas de usuario, estas pruebas se llevarán a cabo el día 20 de junio de 2024 en el Laboratorio de Telemática ubicado en el Instituto de Ciencias Aplicadas y Tecnología, las pruebas las llevaran a cabo los estudiantes que se encuentran prestando sus servicios dentro de dicho laboratorio.

Las pruebas realizadas evaluarán que el sistema sea agradable y llamativo para el usuario en cuanto a diseño, que el sistema sea intuitivo y fácil de usar.

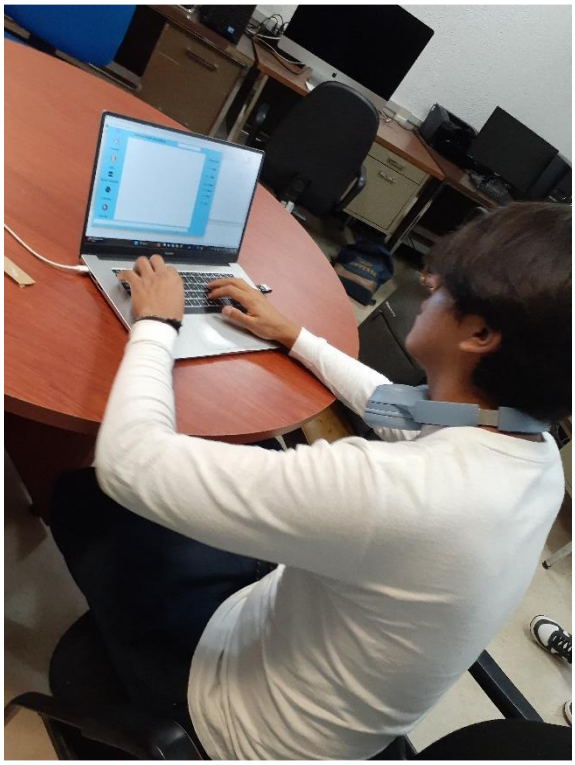

Figura 68. Prueba Usuario 1 Fuente: Elaboración propia

Figura 69. Prueba Usuario 2 Fuente: Elaboración propia

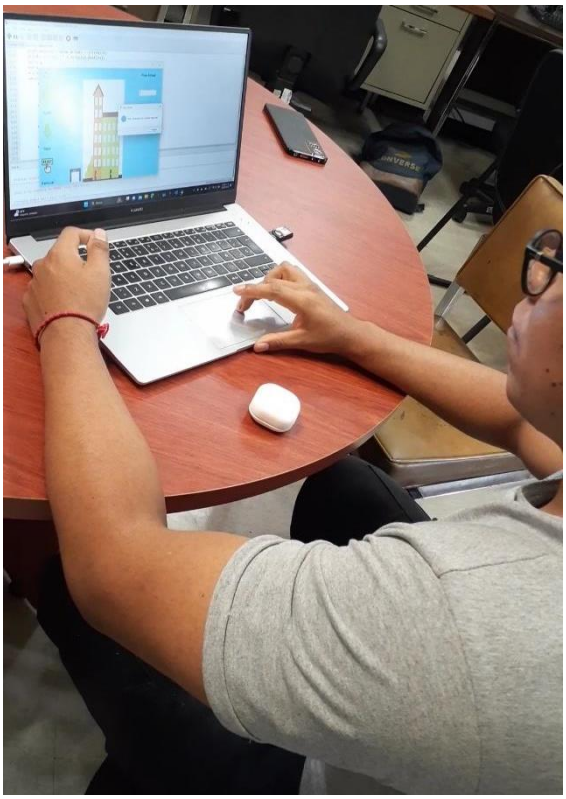

Figura 70. Prueba Usuario 3 Fuente: Elaboración propia

5.4 Resultados

Después de realizar las pruebas por parte de los usuarios del ICAT se obtuvieron los siguientes comentarios:

- **Usuario 1:** El sistema funciona bien, la vista es agradable, es intuitivo y permite usarlo sin ningún problema, aunque estaría bien que las notificaciones de error de compilación pudieran mostrarse en una ventana emergente y no solo en la barra de notificaciones, algunos podrían no darse cuenta de los mensajes en está barra.

- **Usuario 2:** El sistema está muy bien, es fácil de usar se entiende muy bien, el diseño es atractivo, solo si pudieran ser un poco más específicas las instrucciones para realizar los programas estaría mejor, usuarios sin conocimiento previo del sistema podrían tener algunas dificultades sino se especifica un poco más en las instrucciones.

- **Usuario 3:** El sistema funciona muy bien, la interfaz es muy agradable a la vista, es muy intuitivo y fácil de entender, no tuve dificultades para realizar los programas las instrucciones son claras y fáciles de entender.

5.5 Conclusión

El desarrollo de este trabajo de tesis me permitió entender la importancia de que las personas empiecen desde una corta edad a aprender fundamentos de programación, pues esto es algo que a futuro les ayudará debido al gran avance que está teniendo la tecnología día con día, me permitió conocer múltiples plataformas que brindan a las personas la posibilidad de aprender estos conocimientos de programación, aunque me hizo darme cuenta de la desigualdad en este aspecto ya que no todas las personas pueden tener el acceso a estos recursos o la forma en la que enseñan a programar está muy limitada a la tecnología que usan dichas empresas, por eso entendí la importancia de desarrollar un sistema que sea de fácil acceso y que tenga un costo muy bajo para poderlo replicar.

Tras observar las pruebas de usuario y los resultados me di cuenta de la gran importancia de pensar en los usuarios a la hora de desarrollar este tipo de sistemas, es de vital importancia pensar en que no todos los usuarios tienen los conocimientos previos para poder usar el sistema y presentar instrucciones muy claras para que cualquier usuario pueda usar los sistemas ya sea que tenga conocimientos previos o no.

Los resultados obtenidos me ayudaron a darme cuenta de las posibles mejoras que el sistema puede tener, como la inclusión de más instrucciones de programación, permitir que el sistema pueda manipular más de un tipo de robot educativo, agregar un decodificador de tonos DTMF para poder acoplarlo con otros sistemas, mejorar la interfaz de usuario, etc.

Referencias

[1] *Trbl.* (2023). Sistemas embebidos y sus características | conceptos fundamentales. *TRBL Services.* Recuperado de https://tech.tribalyte.eu/blog-sistema-embebido-caracteristicas#El_sistema_embebido_Que_es_exactamente

[2] *Luke.* (2022, 22 diciembre). Explicación de los sistemas embebidos - *Rebound Electronics.* Recuperado de https://reboundeu.com/es/insights/blog/embedded-systems-explained-15/

[3] *Marker, G.* (2022). El lenguaje ensamblador. Tecnología + Informática. Recuperado de https://www.tecnologia-informatica.com/el-lenguaje-ensamblador/

[4] *Trbl.* (2023b). Sistema embebido|Descubre C/C++ para el desarrollo de *software* embebido. *TRBL Services.* Recuperado de https://tech.tribalyte.eu/blog-que-es-lenguaje-c-donde-se-aplica

[5] *Python*_para_Microcontroladores. Recuperado de https://pythononhardware.funpython.org/

[6] ¿Qué es java incrustado? - Definición de *techopedia* - Desarrollo 2023. (2023). Recuperado de https://es.theastrologypage.com/embedded-java

[7] Sergio Noriega (2017). Introducción al Diseño lógico con *VHDL.* Recuperado de https://catedra.ing.unlp.edu.ar/electrotecnia/islyd/Tema%2012b%20Logica%20Programable%20VHDL%20%202017.pdf

[8] *Syntonize.* (2022, 11 agosto). *Rust* como lenguaje de programación y sus beneficios - *Syntonize*. Recuperado de https://www.syntonize.com/rust-como-lenguaje-de-programacion-y-sus-beneficios/#:~:text=Rust%20es%20un%20lenguaje%20multiprop%C3%B3sito,de%20aplicaciones%20con%20sistemas%20embebidos

[9] Fernández, Y. (2022). Qué es arduino, cómo funciona y qué puedes hacer con uno. *Xataka*. Recuperado de https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno

[10] *Trbl.* (2023a). *Arduino vs Raspberry Pi* | ¿Cómo elegir? ¿Para qué sirven? 10 tips. *TRBL Services.* Recuperado de https://tech.tribalyte.eu/blog-arduino-vs-raspberry-pi

[11] Ecarletti. (s. f.). ¿Qué es *BeagleBone Blue*? | Robots didácticos. Recuperado de https://robots-argentina.com.ar/didactica/que-es-beaglebone-blue/

[12] Jacob Beningo. (2019). Acelere el desarrollo de productos *IoT* utilizando el ecosistema *Mbed*. Recuperado de https://www.digikey.com.mx/es/articles/accelerate-iot-product-development-using-mbed-ecosystem

[13] Guillermo. (s. f.). Introducción a los *RTOS*. Recuperado de http://www.guillehg.com/index.php?option=com_content&view=article&id=49&Itemid=691

[14] *Trbl. (2023b). LINUX* embebido | Qué es, cómo funciona y para qué se usa. *TRBL Services*. Recuperado de https://tech.tribalyte.eu/blog-linux-embebido-que-es

[15] Ruiz-Velasco&Bárcenas; 2022

[18] Euroinnova Formación. (2023b). Blogs educativos primaria. Euroinnova *Business School*. Recuperado de https://www.euroinnova.edu.es/blog/robots-educacionales#robots-educacionalesnbsp

[19] *Observatory, D. E.* (s. f.). La robótica educativa revoluciona las aulas y potencia el aprendizaje interactivo. www.linkedin.com. Recuperado de https://www.linkedin.com/pulse/la-rob%C3%B3tica-educativa-revoluciona-las-aulas-y/?originalSubdomain=es

[20] *OpenAI.* (2023, junio 22). Definición de una plataforma para robótica educativa [Mensaje en un modelo de lenguaje]. *ChatGPT.* https://www.openai.com

[21] *Crystalino* - Robot Educativo Compatible *Arduino* |. (2020, 8 septiembre). Recuperado de https://complubot.com/proyectos/Crystalino/

[22] *Starting with robotics* |. (2021, 2 junio). Recuperado de https://complubot.com/proyectos/swr/

[23] Dani. (2022, 31 mayo). Robótica educativa 45 entornos virtuales para programar robots emulados. Recuperado de https://juegosrobotica.es/podcast-045/#

[24] *Mindstorms fix the factory.* (2016, 8 junio). Recuperado de https://www3.gobiernodecanarias.org/medusa/ecoescuela/recursosdigitales/2015/03/06/mindstorms-fix-the-factory/

[25] De Miguel, R. (2023, 20 septiembre). *Kodable*: una plataforma para programar mediante el juego. Recuperado de https://www.educaciontrespuntocero.com/tecnologia/kodable-programar/

[26] Trucoteca. (2023, 11 febrero). ¿Qué son las aplicaciones en el escritorio?. Recuperado de https://trucoteca.com/que-son-las-aplicaciones-en-el-escritorio/

[27] *CorelDRAW Graphics Suite | Free Trial.* (s. f.). Recuperado de https://www.coreldraw.com/la/

[28] Explicación de *adobe PDF.* (s. f.). Recuperado de https://helpx.adobe.com/mx/incopy/using/pdf.html

[29] Euroinnova Formación. (2023c). Blogs educativos primaria. Euroinnova Business School. Recuperado de https://www.euroinnova.mx/blog/aplicaciones-de-escritorio

[30] Cristancho, F., Cristancho, F., & Cristancho, F. (2023). ¿Qué es un *framework* en programación? Actualizado 2023. *Talently* Blog. Recuperado de https://talently.tech/blog/que-es-un-framework-en-programacion/

[31] Colectiva, N. (2022, 23 agosto). Los 5 mejores *frameworks* para crear aplicaciones *desktops* (Escritorio) con *JavaScript, HTML y CSS.* Recuperado de https://blog.nubecolectiva.com/los-5-mejores-frameworks-para-crear-aplicaciones-desktop-escritorio-con-javascript-html-y-css/

[32] *Frameworks* de *Python* para *desktop* | *FAZT Web.* (2022, 6 noviembre). Recuperado de https://faztweb.com/contenido/python-frameworks-desktop

[33] ¿Qué es un *SOC* y para qué sirve? - definición. (2021, 10 mayo). Recuperado de https://www.geeknetic.es/SoC/que-es-y-para-que-sirve

[34] *Intel Atom® C Processors Series.* (s. f.-b). Recuperado de https://www.intel.la/content/www/xl/es/products/details/processors/atom/c.html

[35] *AMD RyzenTM Embedded Serie V1000.* (s. f.). Recuperado de https://www.amd.com/es/products/embedded-ryzen-v1000-series

[36] *Samsung* presenta el revolucionario procesador *Exynos 2200* con *GPU Xclipse* e impulsado por arquitectura *AMD RDNA* 2. (s. f.). Recuperado de https://news.samsung.com/co/samsung-presenta-el-revolucionario-procesador-exynos-2200-con-gpu-xclipse-e-impulsado-por-arquitectura-amd-rdna-2

[37] Raspberry Pi Pico - Waveshare Wiki. (s. f.). https://www.waveshare.com/wiki/Raspberry_Pi_Pico

[38] *Usando ESP8266 con el IDE de Arduino. (s. f.). Naylamp Mechatronics - Perú. https://naylampmechatronics.com/blog/56_usando-esp8266-con-el-ide-de-arduino.html*

[39] PEREZ, A; et al. “Una metodología para el desarrollo de *hardware* y *software* embebidos en sistemas críticos de seguridad”. *Systemics, Cybernetics and Informatics Journal*, vol 3, Num. 2, 2006, pp. 70-75.

[40] *Knowledgebase | USB History - Premium USB.* (s. f.). Recuperado de https://www.premiumusb.com/usb-history

[41] *Bosch, R. (1991). CAN BUS Specification.* Recuperado de https://www.kvaser.com/software/7330130980914/V1/can2spec.pdf

[42] *Embedded Systems Academy. (2015). History of the I2C Bus.* Recuperado de https://www.esacademy.com/en/library/technical-articlesand-documents/miscellaneous/i2c-bus/general-introduction/history-of-thei2c-bus.html

[43] *Microchip Technology. (2014). Overview and Use of the PICmicro Serial Peripheral Interface.* https://ww1.microchip.com/downloads/en/devicedoc/spi.pdf

[44] Alejandro Daniel Perez. (2016). "Protocolos de comunicación entre microcontroladores. Caso de estudio: Protocolo *CAN*". Tesina de Licenciatura en Sistemas.

[45] Crodriguez, & Crodriguez. (2023, 28 junio). ¿Qué son los componentes electrónicos y cuáles se utilizan más? Recuperado de https://sdindustrial.com.mx/blog/componentes-electronicos-que-son/

[46] *Led, M. T.* (2018, 4 enero). Tipos de pantallas electrónicas más comunes - Maupe. Recuperado de https://www.maupe.com/Empresa/tipos-de-pantallas-electronicas-mas-comunes/

[47] Rohde & Schwarz. (s. f.). Entendiendo el *UART.* Recuperado de https://www.rohde-schwarz.com/lat/productos/prueba-y-medicion/essentials-test-equipment/digital-oscilloscopes/entendiendo-el-uart_254524.html

[48] Aula. (2023b, mayo 23). Cómo funciona un motor eléctrico: tipos y partes. Recuperado de https://www.cursosaula21.com/como-funciona-un-motor-electrico/

[49] Rodriguez, A. (2023, 8 marzo). Sensores electrónicos: ¿Qué son y cómo funcionan? diarioelectronicohoy.com. Recuperado de https://www.diarioelectronicohoy.com

[50] *Tipos de Robots: Características y ejemplos*. (2020, 28 marzo). Tipos de Robot. Recuperado de https://tiposderobot.com/

[51] Valtx. (2022, 19 julio). Metodologías para el desarrollo de software: ¿Qué son y para qué sirven? Valtx. https://www.valtx.pe/blog/metodologias-para-el-desarrollo-de-software-que-son-y-para-que-sirven

[52] Robótica, E. (2023, 25 febrero). Historia y evolución de la robótica educativa. Educación Robótica. https://educacionrobotica.com/historia-y-evolucion-robotica-educativa/

[53] Pinto-Salamanca, M. L., Barrera-Lombana, N., & Pérez-Holguín, W. J. (2010, 1 julio). Uso de la robótica educativa como herramienta en los procesos de enseñanza. https://revistas.uptc.edu.co/index.php/ingenieria_sogamoso/article/view/912

[54] Gabriela Diego Hernández; febrero 2017; Desarrollo de escenarios virtuales para *software* de robótica educativa en 3D. http://132.248.9.195/ptd2017/febrero/0755299/Index.html

[55] Moreno, I., Muñoz, L., Serracín, J. R., Quintero, J., Patiño, K. P., & Quiel, J. (2012). LA ROBÓTICA EDUCATIVA, UNA HERRAMIENTA PARA LA ENSEÑANZA-APRENDIZAJE DE LAS CIENCIAS y LAS TECNOLOGÍAS. Redalyc.org. https://www.redalyc.org/articulo.oa?id=201024390005

Referencias de figuras

Figura 1. *Robot Bee-Bot* https://s3.eu-central-1.amazonaws.com/robotica-es/uploads/items/24562_beebot1.png

Figura 2. *Robot Cubetto* https://aprenderconrobots.com/wp-content/uploads/The_Cubetto_Playset.jpg

Figura 3. *Robot Botley* https://www.rayuelainfancia.com/14534-large_default/botley-el-robot.jpg

Figura 4. *Robot Dash* https://www.robotsparaninos.com/wp-content/uploads/2016/06/Dash-and-Dot-robot-educativo-Laberinto-1024x682.jpg

Figura 5. *Robot ELEGO UNO R3* https://cdn.shopify.com/s/files/1/0296/9026/5648/products/elegoo-uno-r3-project-smart-robot-car-kit-v-30-plus-arduino-stem-kits-elegoo-shop-231477_900x.jpg?v=1622707708

Figura 6. *LEGO MINDSTORMS EV3* https://www.lego.com/cdn/cs/set/assets/blt40ee5eaab1a56b38/31313.jpg?format=webply&fit=bounds&quality=70&width=640&height=640&dpr=1.5

Figura 7. *Robot Makeblock mBot Ranger* https://www.makeblock.es/thumb/500x350/KIT_RANGER_makeblock_KIT_RANGER.jpg

Figura 8. *Clementoni Robomaker* https://cdn.shopify.com/s/files/1/0650/5666/9929/products/55331.PT01_a7a96dc4-7cba-486c-ab3d-ad9413dbc680.jpg?v=1657815265

Figura 9. *Robot Crystalino* https://complubot.com/wp-content/uploads/2018/11/Crystalino_00_1200-768x665.jpg

Figura 10. Controlador *Crumble* https://complubot.com/wp-content/uploads/2015/02/Basic-Connections-595x275.png

Figura 11. *Software Crumble* https://complubot.com/wp-content/uploads/2023/04/Sofware_Crumble_Smart_Crumbs_01-768x502.jpg

Figura 12. Simulador mBot https://i.ytimg.com/vi/Z766xEvOqy0/maxresdefault.jpg