UNIVERSIDAD DE TARAPACÁ

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA CIVIL EN COMPUTACIÓN E INFORMÁTICA

Ingenierí@
Computación e Informática

Plan de Proyecto "Maquinaria de clasificación de materiales"

Alumnos:	**Bastian Hernandez**
	Saoud Ahmed
	Alex Campillay
	Marcos Caldas
	Enzo Llancabure
Asignatura:	**Proyecto I**
Profesor:	**Baris Klobertanz Quiroz**

30 de Diciembre – 2025

Historial De Cambios

Tabla N°1 “Historial de Cambios”.

Fecha	Versión	Descripción	Autor(es)
22/09	1.0	Reconocimiento del Problema y Formalización del Proyecto	Todos
26/09	1.1	Finalización de la introducción	Marcos Caldas Enzo Llancabure Bastian Hernandez
08/10	1.2	Finalización de la organización de personal	Marcos Caldas Bastian Hernandez Enzo Llancabure
13/10	1.3	Finalización de la planificación del proyecto	Enzo Llancabure Marcos Caldas Bastian Hernandez
17/10	1.4	Finalización de la planificación de recursos	Enzo Llancabure Marcos

			Caldas
17/10	1.5	Finalización de la conclusión informe I	Enzo LLancabure
17/10	1.6	Finalización de las referencias informe I	Bastian Hernandez
28/11	1.7	Corrección del informe fase I	Todos
12/12	1.8	Finalización de análisis y diseño	Bastian Hernandez, Alex Campillay
12/12	1.9	Finalización de Implementación	Saoud Ahmed Marcos Caldas
13/12	2.0	Finalización de Resultados	Marcos Caldas, Enzo Llancabure
13/12	2.1	Finalización de conclusión fase II	Enzo Llancabure
26/12	2.2	Corrección del informe fase II	Todos
28/12	2.3	Finalización de prueba de funcionamiento del sistema	Bastian Hernandez, Marcos Caldas
28/12	2.4	Finalización Conclusión Fase III	Enzo Llancabure

30/12	2.5	Finalización Y entrega de informe Final	Todos

➢ Índice

➢ Índice de tablas

➢ Indice de imagenes

1. Panel General

1.1. Introducción

En la industria minera se evidencia múltiples veces que la clasificación de materiales es una parte importante de la operación; sin embargo, la ejecución ineficiente de esto puede llevar a cuellos de botella operativos. Aún más grave estos procesos pueden llevar a la exposición de los trabajadores a entornos peligrosos si no están debidamente automatizados, evidenciando el problema claramente: La necesidad de implementar un sistema que garantice eficiencia sin comprometer la integralidad de los trabajadores.

Bajo esta premisa, en este informe se demostrará el trabajo colaborativo realizado por el equipo para cumplir el objetivo de mejorar la productividad, optimizar los procesos y elevar los estándares de seguridad de un equipo de minería. Para lograrlo se aplicaron conocimientos de ingeniería mediante una simulación de un clasificador de materiales fabricado con el set de Lego Spike Prime.

A partir de lo anterior, la estrategia seguida para la organización de actividades y asignación de responsabilidades se detalla en los siguientes apartados.

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar y programar un robot con el kit LEGO Spike Prime que sea capaz de clasificar bloques en una casilla designada según su color, simulando un proceso de clasificación de materiales de una industria minera.

1.2.2. Objetivos Específicos

- Experimentar para ser capaz de usar correctamente el set Lego Spike Prime hasta el punto de poder aplicar correctamente todas las funciones necesarias para crear el robot clasificador en un tiempo estimado de 2 semanas.

- Asignar roles a cada integrante del grupo para mantener un responsable en cada área de trabajo y poder realizar el proyecto de forma eficiente, en un plazo de 1 semana.

- Planificar y ensamblar un prototipo inicial con el set LEGO Spike Prime que pueda detectar con el sensor los colores de los bloques y mover los motores respectivamente del color, en un plazo de 3 semanas.

- Programar con el lenguaje de programación Python en la aplicacion LEGO Spike un codigo capaz de conectar el sensor de color junto con los motores al mismo tiempo para buscar la rapidez del ordenamiento de bloques, en el tiempo que el ensamblador lo tenga que necesitar.

- Construir el robot que cumpla con una buena coordinación y velocidad con el set Lego Spike Prime, librería Pybricks y Visual Studio Code, además que sea capaz de clasificar correctamente cada bloque en una casilla, antes de la presentación final.

- Documentar y subir los archivos en formato PDF a la plataforma de Redmine para registrar cada avance que se va a realizar hasta la fecha final del

proyecto.

- Implementar una interfaz para el robot clasificador que sea capaz de realizar movimientos de este, dependiendo de los botones de la interfaz, utilizando Tkinter y Visual Studio Code, una vez terminado el robot, en un plazo de 3 semanas.

- Definir un repositorio en GitHub para tener un historial de cambios, además de un sitio donde cada integrante será capaz de ver el código, dentro de la web de GitHub, hasta que se finalice el trabajo de codificación.

- Crear un manual de usuario con las instrucciones de cómo usar el robot, en Microsoft Word, al finalizar la construcción y codificación del robot, en un plazo de 1 semana.

- Realizar una presentación que sea capaz de demostrar que el robot clasificador puede realizar una buena clasificación de bloques, utilizando la web Canva, en un tiempo estimado de 1 semana.

1.3. Restricciones

Las restricciones son requerimientos mínimos que deben ser obligatoriamente cumplidos.
Las restricciones que se tiene en este proyecto son:

- Si es necesario, utilizar la extensión de Lego Spike Prime.
- Solo se debe utilizar la plataforma Redmine para los documentos y avance del proyecto.
- Se debe utilizar el Set de Lego Spike Prime.
- Tiempo limitado para la finalización del proyecto, debido al término del semestre.
- Cantidad de integrantes limitada a 5.
- La disponibilidad del robot para su uso sea codificación y construcción, está limitada al horario del departamento de ICCI (Ingenieria civil en computacion e Informatica).
- Robot debe ser capaz de reconocer el bloque y poder clasificarlo dependiendo de su color.
- La conexión entre el pc y el robot debe ser inalámbrica.

1.4. Entregables

-Bitácoras: Son informes semanales que describen el avance del equipo en el proyecto, abarcando actividades realizadas, dificultades encontradas, recomendaciones para mejorar y acciones tomadas. Preparadas por los integrantes del grupo, ofrecen un panorama exhaustivo para apoyar decisiones estratégicas, asignan responsabilidades y resaltan asuntos a tratar en grupo.

-Carta Gantt: Representación visual de la programación del proyecto, mostrando en una línea de tiempo las tareas, su duración y secuencia, facilitando la gestión del tiempo y los recursos al visualizar la evolución de las actividades a lo largo del proyecto.

-*I*nformes de avance: Este documento detalla la organización y estrategia para alcanzar los objetivos de la asignatura. Se abordará la asignación de roles, las metas del equipo y las medidas que implementarán para lograr el propósito académico. Además, se comparten las primeras impresiones durante el proceso de desarrollo y se presenta la documentación relevante recopilada a lo largo del semestre.

-Presentaciones: Se harán presentaciones con el formato de PDF, para presentar la información recopilada en los informes entregados, cada presentación tratará los temas de cada informe ya sea avance o posteriores a él.

-Manual de Usuario: Se desarrolla un manual de usuario para el funcionamiento correcto del robot enfocado en el aprendizaje para el usuario.

-Código base del cliente y servidor, publicados en un repositorio en Github.

2. Organización del Personal

La organización en un grupo es importante para lograr la finalización de este proyecto, como también un buen ambiente en el grupo.

2.1. Descripción de los Roles

Jefe de proyecto: Representante del equipo, supervisa y organiza el progreso del proyecto.

Ensamblador: Encargado del montaje y el armado de las piezas, monitorea el cumplimiento de las funcionalidades del robot, en conjunto con el programador.

Programador: Encargado del área de la codificación y la lógica de funcionamiento del robot, trabajando en colaboración con el ensamblador para integrar el software con el hardware.

Documentador: Encargado de registrar el avance del proyecto y la redacción de los informes. Para cumplir su función, debe mantener una comunicación constante con todos los miembros del equipo, con el fin de recopilar y unir la información de cada área.

2.2. Personal que Cumplirá los Roles

Tabla N°2 “Roles”.

Rol	**Responsable**
Jefe de proyecto	Bastian Hernandez
Ensamblador	Alex Campillay
Programador	Saoud Ahmed
Documentador	Enzo Llancabure Marcos Caldas Bastian Hernandez

2.3. Métodos de Comunicación

Los principales medios de comunicación que usarán a lo largo del desarrollo del proyecto serán los siguientes:

-WhatsApp: Se utilizará para mensajería haciendo uso de los grupos que ofrece la Aplicación.

-Github: Como repositorio de codificación con avance e historial de versiones de este mismo, es una comunicación enfocada a tareas técnicas.

-Discord: Reuniones a distancia aprovechando los canales de texto y voz que ofrece esta Aplicación.

-Horario del Taller de Clase: Reuniones presenciales y avance de forma presencial para comunicar el avance del proyecto mismo.

3. Planificación del Proyecto

3.1. Actividades

Tabla N°3 “Actividades”

O.E	Nombre	Encargado
O.E.1	Dominar el LEGO Spike Prime para una óptima creación de Robot.	Todos los integrantes.
O.E.2	Asignación de los Roles para cada integrante del grupo.	Todos los integrantes.
O.E.3	Diseño y Ensamblaje del primer prototipo con el set LEGO Spike Prime	Alex Campillay, Bastian Hernandez.
O.E.4	Programación del Robot Lego encargado de clasificar.	Saoud Ahmed Marcos Caldas
O.E.4	Codificación en Python para el uso del robot.	Saoud Ahmed Marcos Caldas
O.E.4	Adopción de Pybricks con Visual Studio Code.	Saoud Ahmed Marcos Caldas
O.E.5	Creación del 2do Prototipo Lego.	Alex Campillay, Bastian Hernandez, Enzo Llancabure.
O.E.5	Optimización del Prototipo Lego.	Alex Campillay, Bastian Hernandez, Enzo Llancabure.
O.E.5	Construcción final del robot LEGO clasificador.	Alex Campillay, Bastian Hernandez, Enzo Llancabure.

O.E	Nombre	Encargado
O.E.5	Pruebas Finales del Robot.	Todos los integrantes.
O.E.6	Documentación y Registro en la plataforma Redmine.	Saoud Ahmed, Bastian Hernandez, Marcos Caldas.
O.E.6	Entrega de los informes de avance.	Todos los integrantes.
O.E.6	Entrega de las presentaciones de avance.	Todos los integrantes.
O.E.7	Implementación de la Interfaz de control para el Robot LEGO clasificador.	Marcos Caldas, Saoud Ahmed, Alex Campillay.
O.E.8	Gestión del Repositorio Github con el código de control del Robot LEGO clasificador.	Todos los integrantes.
O.E.9	Elaboración del Manual de Usuario para tener las instrucciones de uso del Robot LEGO clasificador.	Todos los integrantes.
O.E.10	Presentación del Robot Clasificador para una demostración sólida del Robot LEGO clasificador.	Todos los integrantes.

3.2. Carta Gantt

Esta herramienta permite visualizar la secuencia de las actividades, organizadas en fases como la codificación, el ensamblaje y la documentación. A través de este diagrama, se muestra el porcentaje de avance de cada tarea, ayudando a la organización del grupo a la hora de completar las tareas.

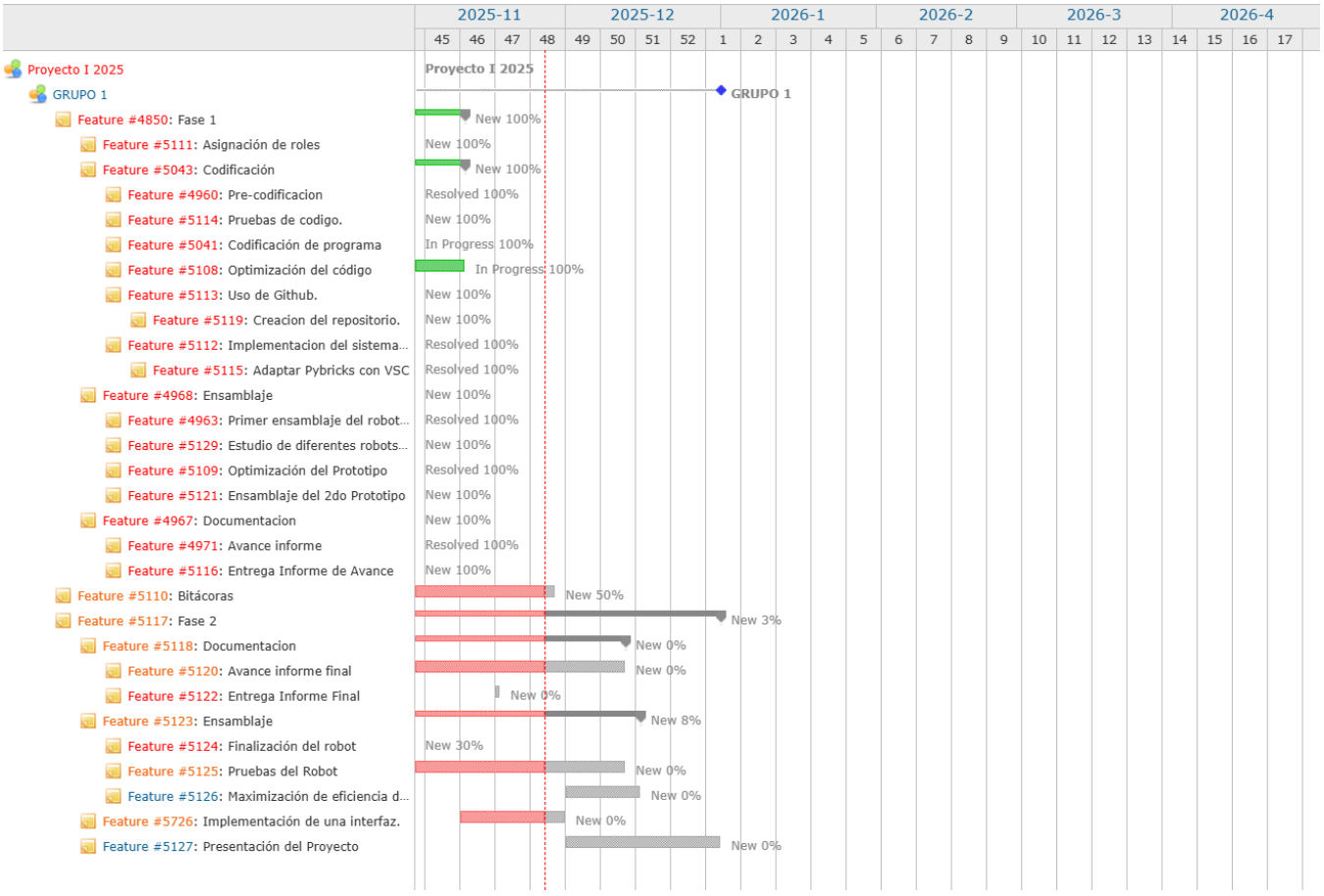

Figura 1: *Carta Gantt.*

3.3. Gestión de Riesgos

Para garantizar el cumplimiento de los objetivos del proyecto en los plazos establecidos, se ha elaborado una tabla de gestión de riesgos. Esta herramienta permite identificar, analizar y clasificar los posibles contratiempos que podrían afectar el desarrollo del prototipo robótico y la planificación general.

A continuación, se definen los niveles de severidad establecidos para clasificar cada riesgo según su impacto en el cronograma y en la operatividad del equipo:

1. **Impacto Crítico:** Problema al cual se le debe otorgar la máxima prioridad de resolución por parte del equipo, del caso contrario puede impactar en la entrega puntual del proyecto por retrasos o incluso un reinicio total de este.
2. **Impacto Alto:** Evento de máxima gravedad que compromete la viabilidad del proyecto. Requiere medidas inmediatas, pudiendo implicar el reinicio de etapas completas o la redefinición del alcance.
3. **Impacto Medio:** Riesgo que genera retrasos significativos en una o varias etapas clave. Exige una respuesta prioritaria para evitar que el desfase afecte la fecha de entrega final.
4. **Impacto Bajo:** Riesgo menor o imprevisto cotidiano que no altera la ruta del proyecto y puede ser resuelto con acciones simples sin afectar los entregables principales.

Tabla N°4 “Gestión de riesgos”

Riesgo	Nivel de Impacto	Acción Remedial
Daño grave de hardware	1	Verificar conexión del puerto, en caso de daños irreparables conseguir repuestos inmediatamente.
Horario insuficiente para el cumplimiento de tareas en conjunto	2	Coordinar los horarios disponibles del grupo.
Desempeño del robot poco eficiente	2	Ensamblar un robot más adecuado siguiendo guías en línea o un nuevo diseño adaptándolo a lo requerido.
Problema a la hora de experimentar con el robot.	2	Buscar la forma de modificar la posición de las piezas para que el robot cumpla su función.
Falla en el registro de redmine	2	Comunicar al profesor para buscar una solución.
Ausencia de piezas	2	Solicitar la extensión de Lego Spike o pedir piezas al ayudante.
Error en la codificación	2	Corregir errores sintácticos y lógicos en lo posible, de no serlo investigar una solución o explorar otro tipo de solución.
Falta de internet en la sala	3	Compartir internet del celular o usar cables ethernet.

Atraso en el cumplimiento de tareas	3	Hablar con el grupo para avanzar el proyecto fuera de clases.
Integrante falta a una clase	4	El Integrante tiene la responsabilidad de ponerse al día con lo que se avanzó en esa clase.

4. Planificación de los Recursos

Este punto muestra los recursos requeridos para la ejecución del proyecto, el cual se divide en herramientas físicas y digitales. Para la tabla de costo trabajador, se le asigna un sueldo basado en un tarifa por hora para cada rol, se toma en cuenta la complejidad de cada rol para la asignación de tarifa. El cálculo final es la suma de horas totales del equipo.

4.1. Hardware

- Set Lego Spike Prime.
- Set Lego Spike Prime Extension.
- Computador con el sistema operativo necesario para poder programar las instrucciones para el robot.
- Tablets para poder hacer la documentación necesaria.

4.2. Software

- Sistema operativo Windows para programar las funciones del robot.
- Redmine, página para la organización del proyecto.
- Canva.
- Plataforma Lego Education Spike (Code).
- Flutter para creación de aplicaciones.
- Pybricks en conjunto con VS code para la codificación pero más centralizada para los robots LEGO.

4.3. Estimación de Costos

Costo de Hardware:

Tabla N°5 "Costo de Hardware"

Producto	Cantidad	Precio (CLP)
Set Lego Spike	1	$ 622.879 [1]
Extension Lego Spike	1	$ 167.202
Notebook LOQ Gen 9	1	$ 769.993
Samsung Galaxy Tab S8 Ultra	1	$ 1.150.000
Lenovo IdeaPad 3 15	1	$ 699.990
Notebook Aspire G A515-58GM-56ZZ-1	1	$ 1.049.990
HP Pavilion Laptop 14-dv2xxx	1	$ 1.500.000
Total:	7	$ 6.005.054

Costo de Trabajador:

Tabla N°6 "Costo Trabajador"

Rol	Horas / Mes	Horas Extra	Precio / Hora (CLP)
Jefe de proyecto	18 horas	5 horas	$ 10.555
Programador	18 horas	5 horas	$ 7.500
Ensamblador	18 horas	5 horas	$ 7.000
Documentador	18 horas	5 horas	$ 3.500
Total :	-	-	$ 2.736.000

Costo de Software:

Tabla N°7 "Costo Software"

Software	Precio (CLP)
Microsoft Office	$ 8.990
Visual Studio Code	$0
Redmine	$0
Pybricks	$0
Flutter	$0
Total	$8.990

Destacado:

- *La contabilización de las horas trabajadas comienza a partir de la formación del grupo de trabajo, la fecha de inicio fue el día 22 de septiembre del 2025.*
- *Para la contabilización de las horas de trabajo, se tuvo en cuenta el tiempo de trabajo en clases.*
- *Para la contabilización de las horas extras, se tuvo en cuenta el tiempo en las que se trabajó fuera del horario de clase.*
- *El cálculo por hora se basa en el sueldo mensual promedio de cada uno de los cargos del proyecto dividido en una jornada laboral de 180 horas mensuales.*
- *Para el pago de horas extras se aplicó un pago doble sobre el valor de la hora normal.*
- *Las horas totales generales sin contar horas extras en este proyecto fueron:*
    - *42 Clases desde la fecha de inicio * 1,5 horas cada clase = 63 horas totales*
    - *Horas extras = definidas como 5 horas por mes, asumiendo que son 3 meses de clases desde septiembre 22 hasta 31 de diciembre*

      *5 * 3 = 15 + 2,5 (horas de septiembre extra) = 17,5 horas extras totales*

Con estos datos se estimó el costo total del trabajador.

Total de Costo:

Tabla N°8 “Costo Total”.

Costo Hardware	$ 6.005.054
Costo Empleados	$ 2.736.000
Costo Software	$ 8.990
Total :	$ 8.750.044

5. Análisis y Diseño

5.1 Especificación de requerimientos

Antes de especificar los requerimientos funcionales y no funcionales, se debe establecer que el cliente es una empresa minera, representada generalmente por el área de ingeniería y automatización de maquinaria, los cuales financian la implementación del sistema robótico encargado para clasificar los materiales, siendo su rol principal asegurar que la solución esperada por la empresa cumpla con los objetivos de mejorar la seguridad de los trabajadores, aumentar la eficiencia y optimizar la clasificación de materiales.
Y tomando en cuenta que el usuario no es necesariamente la misma empresa y que la probabilidad de que interactúe de forma directa con el robot diariamente es escasa, se puede representar al usuario como un operador de maquinaria dentro de la empresa. De esta manera teniendo en cuenta esos dos puntos se pueden definir los requerimientos funcionales y no funcionales.

5.1.1 Requerimientos funcionales

Algunos de los requerimientos funcionales básicos que debe cumplir el robot a la hora de ser implementado y accionado son:
RF1 : El robot debe ser capaz de identificar materiales distintos representados por cuatro colores de bloques de lego (Rojo, Amarillo, Azul, Verde).
RF2 : El robot debe ser capaz de encolar varios bloques de lego para luego atenderlos, una estructura idéntica al funcionamiento de una cola FIFO (First In, First Out) donde sale el primero que entra.
RF3 : El robot debe ser capaz de depositar el bloque de lego identificado por color en su compartimento correspondiente (los 4 compartimentos cada uno asignado a uno de los 4 colores de bloques de lego mencionados en el **RF1**)
RF4 : El robot debe ser capaz de realizar de manera automática los movimientos mencionados en los requerimientos funcionales anteriores (**RF1 , RF2, RF3**).
Además de poder realizarlos de manera manual por el usuario mediante los botones de la interfaz.
RF5-Software : El sistema debe procesar la lectura del sensor de color y enviar la

orden correcta a los motores según el color detectado.
RF6-Software : El software debe ser capaz de permitir el uso manual del robot con la interfaz implementada (Tkinter).

5.1.2 Requerimientos no funcionales

Por otro lado algunos de los atributos de calidad del software que a los stakeholders les podrían interesar para cualificar el proyecto serían:

- Disponibilidad : El robot deberá mantener una disponibilidad mínima del 98% durante la jornada operativa de la empresa minera. Ante una falla, el sistema deberá restablecer su operaciones en un tiempo máximo de 10 minutos, garantizando la eficacia del proceso.
  Métrica de Evaluación : Porcentaje de tiempo operativo

- Robustez : El sistema debe ser capaz de manejar errores de sensores, entradas incorrectas o interrupciones momentáneas de comunicación sin detenerse completamente.
  Métrica de Evaluación : Número de fallos no controlados.

- Rendimiento : El robot automatizado debe responder a las órdenes del operador o del sistema de control en un tiempo máximo de 1 segundo, asegurando el flujo de trabajo eficiente, además deberá mantener su desempeño sin interrupciones durante al menos 8 horas de operación continua.
  Métrica de Evaluación : Tiempo promedio de respuesta a comandos y estabilidad del rendimiento durante turno de trabajo.

- Usabilidad : La interfaz del control del sistema debe ser intuitiva, clara y autoexplicativa, permitiendo que un operador capacitado pueda utilizar el robot de forma autónoma tras una breve capacitación donde se le enseñaria:
  - El funcionamiento de los botones de la interfaz.
  - El movimiento del robot asociado a cada botón de la interfaz.

- Uso del manual de usuario para cualquier problema que pueda ocurrir en la operación del robot.

Métrica de Evaluación : Tiempo de capacitación requerido y cantidad de errores de operación cometidos.

5.2 Arquitectura de software

La arquitectura de software describe los componentes principales del sistema, la forma en que se organizan y cómo se comunican entre sí para cumplir con los objetivos del proyecto.

En este proyecto se utiliza una arquitectura cliente-servidor, donde la aplicación de escritorio (Tkinter) actúa como cliente y el hub del robot LEGO Spike Prime actúa como servidor, el cliente captura la interacción del usuario, genera comandos y los envía por Bluetooth. El servidor recibe, interpreta y ejecuta las instrucciones sobre los motores y sensores mediante Pybricks.

- Modelo de arquitectura cliente-servidor
  en la arquitectura cliente-servidor, el sistema se divide en dos partes principales:

  Cliente(PC-interfaz Tkinter): componente del sistema encargado de la interacción directa con el usuario. Su función principal es capturar las acciones (botones y teclas), interpretarlas y generar solicitudes o comandos que son enviados al servidor a través del medio de comunicación definido. El cliente no ejecuta acciones físicas, sino que actúa como intermediario entre el usuario y el sistema de control.

  servidor(Hub SPIKE Prime-Pybricks): componente del sistema responsable de recibir, procesar e interpretar las solicitudes enviadas por el cliente. a partir de dichas solicitudes, el servidor ejecuta la lógica necesaria para controlar los recursos físicos del sistema, como motores y sensores, realizando las acciones correspondiente en este proyecto, el servidor traduce los comandos recibidos en instrucciones comprensibles en este proyecto el servidor traduce

los comandos recibidos en instrucciones comprensibles para el microcontrolador del robot y gestiona su ejecución.

Este modelo permite separar la lógica de control del robot de la interfaz gráfica, facilitando el mantenimiento, la depuración y la escalabilidad del sistema.

- componentes del sistema
  el sistema está compuesto por los siguientes elementos:
  Cliente lógico:
  Desarrollado en Python + Tkinter.
  Proporciona la interfaz para el usuario.
  Permite al usuario controlar el robot sorter.
  Envía comandos de texto al robot utilizando Bluetooth Low Energy(BLE) mediante pybircksdev.
  Solo se encarga del envío de instrucciones por lo que no controla directamente el hardware.

  Interfaz gráfica de usuario (GUI-Tkinter):
  Forma parte del cliente lógico.
  Permite la interacción directa del usuario.
  Botones para conectar, modo automático, stop y control por colores.

  Medio de comunicación (Bluetooth Low Energy):
  Es la línea de comunicación entre el cliente y el servidor.
  Usa búsqueda y conexión BLE(find_device, PybricksHubBLE).
  Permite el envío de comandos en tiempo real.
  No procesa información, solo se encarga de transmitir los datos generados por el cliente.

  Puente de comunicación y concurrencia (Worker BLE):
  Se implementa con un hilo (thread) mas un event loop de asyncio.
  Mantiene una cola de comandos para no bloquear la GUI.
  Conecta con el hub una sola vez y ejecuta acciones según comandos en cola.

Servidor lógico:

Código desarrollado en Python, creado y gestionado en el entorno de desarrollo Visual Studio Code(VS Code), y ejecutado en el hub del robot LEGO SPIKE Prime mediante el entorno Pybricks;

Recibir comandos enviados a través de BLE..

Interpretar los mensajes recibidos.

Traducir los comandos de alto nivel a instrucciones comprensibles por el microcontrolador.

Controla motores y sensores del robot según la

Instrucción recibida.

Puede devolver información de estado si el sistema lo requiere.

- Flujo de comunicación del sistema

El flujo de trabajo comienza con la interacción del usuario a través de la interfaz gráfica de la aplicación de escritorio, desarrollada en Tkinter. La interfaz interpreta las acciones realizadas por el usuario, tales como presionar botones o utilizar el teclado, y genera un comando de texto correspondiente a cada acción.
Dicho comando es enviado mediante Bluetooth Low Energy (BLE) al hub del robot LEGO SPIKE Prime. El servidor lógico, implementado mediante Pybricks y ejecutado en el hub, recibe el comando, lo interpreta y lo traduce a instrucciones internas comprensibles para el microcontrolador del dispositivo.

Una vez interpretadas, estas instrucciones son ejecutadas sobre los motores y sensores del robot, permitiendo el control del sistema tanto en modo manual como en modo automático. El estado de ejecución o eventos relevantes, como el color detectado, pueden ser enviados de vuelta al cliente para su visualización en la interfaz gráfica.

el control del robot se realiza de la siguiente manera:
El posicionamiento del robot se controla mediante comandos generados desde el teclado o con botones de la interfaz.

El motor de empuje se acciona mediante comandos específicos asignados a controles independientes

El motor automático permite que el robot actúe de forma autónoma utilizando la información proporcionada.

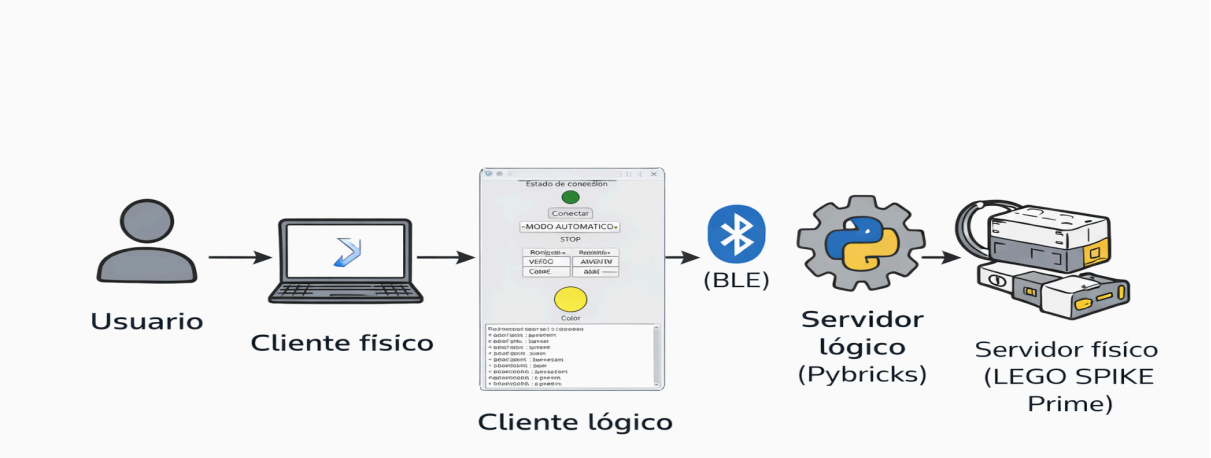

Figura 2: *Flujo de comunicación del sistema.*

- Ventajas de la arquitectura utilizada

  la arquitectura cliente-Servidor utilizada ofrece múltiples ventajas:

  Separación clara entre interfaz gráfica y control.

  Mayor facilidad para depuración y pruebas.

  Escalabilidad para agregar nuevas funciones.

  Reutilización del código del robot con distintas interfaces de control.

  Esta arquitectura permite que la aplicación de escritorio se enfoque exclusivamente en la experiencia del usuario y visualización de estado, mientras que el robot se encarga del control del hardware, mejorando la organización general del sistema y facilitando futuras modificaciones.

5.3 Diseño inicial de la interfaz gráfica de usuario (GUI)

En esta sección se presenta el diseño inicial de la interfaz gráfica de usuario (GUI) del sistema de control del robot sorter. Este diseño corresponde a un wireframe de baja fidelidad, cuyo objetivo es definir la estructura general de la aplicación, la disposición de sus elementos y la interacción básica del usuario, sin considerar detalles visuales, tipografías o estilos gráficos finales.

El wireframe permite planificar la interfaz antes de su implementación, facilitando la comprensión del funcionamiento del sistema y sirviendo como base para el desarrollo posterior en Tkinter.

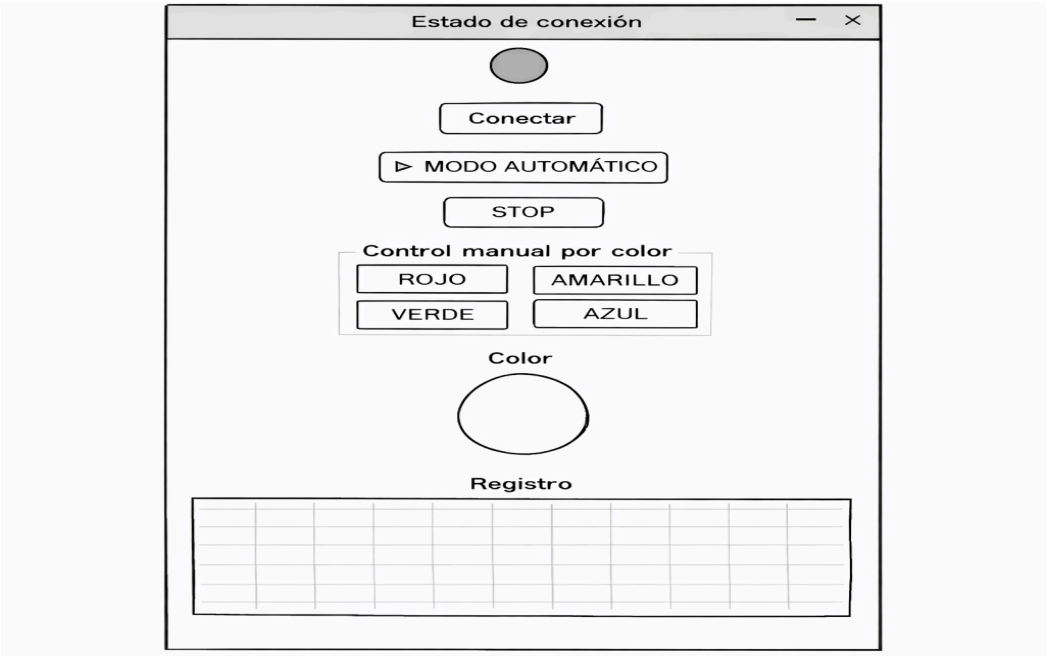

Figura 3: ***Wireframe GUI.***

Este wireframe se utilizó como referencia directa para la implementación de la interfaz gráfica de la aplicación de escritorio. desarrollada en Tkinter. Cada elemento del boceto fue traducido a componentes básicos de la biblioteca Tkinter, tales como Frame, Button, Label, Canvas y controles de entrada por teclado.

El diseño inicial permitió definir de forma clara la distribución de la ventana, la jerarquía de los elementos visuales y la lógica de interacción del usuario, antes de aplicar ajustes visuales y funcionales definitivos. De esta manera, el uso del wireframe facilitó una implementación más estructurada, reduciendo errores en etapas tempranas y asegurando coherencia entre el diseño conceptual y la interfaz

final

6. **Implementación**

En esta sección del informe se presentan los resultados obtenidos hasta el momento en el desarrollo del proyecto. Se incluye la justificación de la configuración del robot a partir de principios físicos de movimiento básico estudiados en la asignatura FI 035–Introducción a la Física, una descripción de los componentes más relevantes del sistema implementado y una captura de la interfaz gráfica de usuario ya desarrollada, junto con la explicación de sus elementos y funciones.

6.1 Fundamentos de los movimientos

Se presentan los distintos tipos de movimientos utilizados para el correcto funcionamiento del robot desde la primera aplicación física-matemática hasta la del último uso del robot. Se divide en tres secciones fundamentales que justifican a detalle el funcionamiento matemático del robot, las cuales son:

- Caída del Bloque de Lego:

Se toma en consideración un diagrama de cuerpo libre para utilizar la segunda ley de newton para determinar la aceleración ideal requerida para el correcto funcionamiento; tomando en cuenta diferentes datos, tales como: Peso del Bloque, Ángulo de inclinación de la caída y El roce efectuado del mismo plástico (DINÁMICA).

Datos:

Masa (m): 0.00232 kg.

Coeficiente de roce cinético (uk): 0,654 (estimado para plástico ABS).

Gravedad (g): 9.8 m/s^2.

Ángulo de inclinación(theta) : 60°.

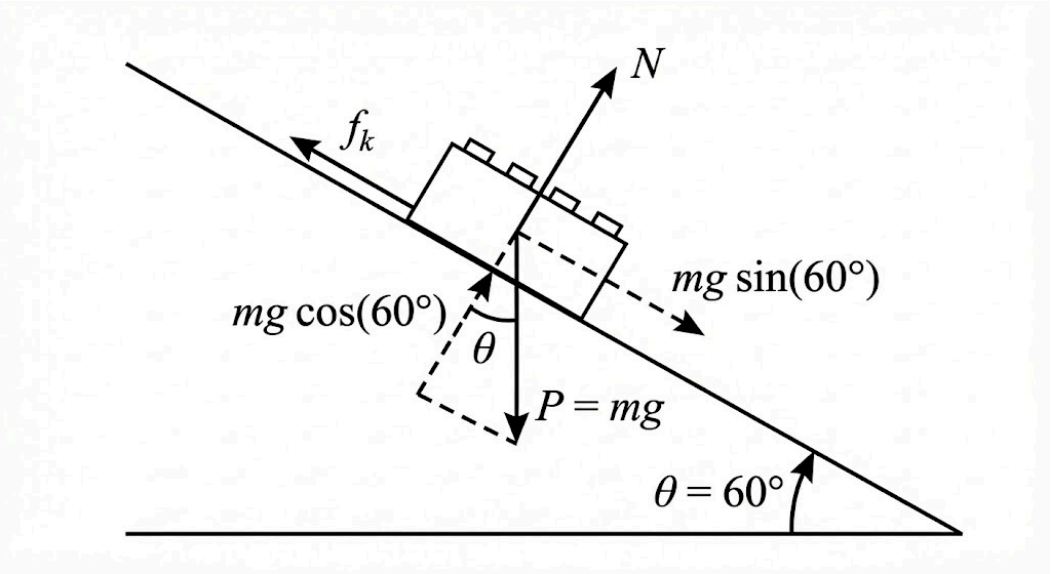

Figura 4: *"DCL"*

Desarrollo: Analizar las fuerzas al ubicar el bloque sobre la superficie inclinada, que actúan dos fuerzas opuestas en el eje del movimiento: componente horizontal Px que favorece la caída y la fuerza de roce fk que se opone a ella.

Fuerza Normal (N): Es la fuerza de reacción perpendicular a la superficie.
N = m * g * cos(60) N = 0.00232 * 9.8 * 0.5 **N = 0.01137 Newtons.**

Fuerza de Roce Cinético (fk): Es la resistencia opuesta al movimiento, calculada con el coeficiente estimado. fk = uk * N fk = 0.654 * 0.01137 fk = 0.00744 Newtons

Componente del Peso en el eje X (Px): Es la fuerza generada por la gravedad que empuja el bloque hacia abajo por la rampa. Px = m * g * sen(60) Px = 0.00232 * 9.8 * 0.866
Px = 0.01969 Newtons

Teniendo los datos ya resueltos se aplica la segunda ley de Newton (Suma de fuerzas = m * a) la cual dirá la aceleración resultante:

a = (Px - fk) / m a = (0.01969 - 0.00744) / 0.00232 a = 0.01225 / 0.00232 **a = 5.28 m/s**2**

El dato de la aceleración de 5,28 m/s**2 es la ideal, menor a ella obtendrán

resultados poco eficientes en relación al tiempo de ejecución y mayor a ella se obtendrán múltiples errores desde caída con choques y el bloque expulsado hasta el no reconocimiento del sensor de color.

- Cálculo del Tiempo Real de posicionamiento

El giro de clasificación opera en un ángulo simétrico de -35° a +35° donde el ángulo 0° representa el reposo del clasificador. A diferencia del motor de empuje, que realiza un movimiento fijo (“golpe”), el motor de posicionamiento tiene trayectorias que cambian debido al sensor de color para el correcto ordenamiento del bloque.
Determinación de la velocidad real (w) para calcular la eficiencia del giro se calcula el recorrido total del robot de un extremo a otro, de -35° a +35°.
Distancia Máxima = 35 + 35 = 70°
Fórmula = t(max) = Diferencia Angular / velocidad real
La velocidad real está limitada por el mismo Hub del lego la cual es 300°/s .

t(max)= 70°/300°/s = 0.23 s

el tiempo de recorrido es 0,23 segundos, junto con la velocidad real que esta limitada por el Hub del lego spike prime es lo mas eficiente debido a que el recorrido por deteccion de color es minimo para girar y volver del reposo, los datos y resultados son a base del peor caso posible, ir de izquierda a derecha, de extremo a extremo.

- Eficiencia Motor de empuje (radio de empuje = 4 cm)

el motor de empuje se encarga de colocar los bloques en la sección indicada de su color
se selecciona un recorrido de 180° para empujar el bloque, es eficiente ya que al seleccionar no debe dar un giro completo, como si fuera 360°tardaría más en ejecutarse, implementando así efectividad en el ángulo de empuje ya que da un ida y retorno simple y rápido, junto con el tope máximo de velocidad admitida por el Hub (300°/s) para validar que esta configuración ofrece una fuerza óptima en un tiempo mínimo, se aplican las siguientes fórmulas:

Datos:

1. **Velocidad Angular Máxima (omega):** 300°/s (Límite operativo establecido en el Hub).
2. **Radio de Palanca (r):** 4 cm (Longitud del brazo de empuje).
3. **Desplazamiento (theta):** 180°.

-Cálculo del tiempo de ciclo: determina cuanto tiempo tarda el brazo en completar el empuje al color indicado:
t = desplazamiento/velocidad angular = 180°/(300°/s) = 0.6 segundos

-Cálculo de la Fuerza de empuje (Newtons): determina la fuerza con la que se empuja el bloque (utilizamos el torque del motor = 18 N*CM).

FUERZA = TORQUE/RADIO = 18 N*CM/4CM = 4.5 NEWTONS

Los siguientes datos extraídos son necesarios ya que se necesita calcular la fuerza de empuje para asegurar que el motor tiene potencia suficiente para mover el bloque y añadimos velocidad para generar impacto teniendo en cuenta que buscamos la eficiencia respecto al tiempo.

6.2 Descripción del sistema

El sistema utilizado para el control y manejo del robot se basa en Pybricks, una plataforma que permite programar y controlar dispositivos robóticos de manera eficiente. A través de Pybricks, es posible implementar distintas formas de control del robot, lo que otorga flexibilidad al sistema y permite adaptarlo a diferentes métodos de interacción y operación.

Forma 1: Control_Automatico_Archivo_Base.py

Este corresponde al código inicial del sistema y opera de manera automática, es decir, no requiere la intervención directa de una persona para controlar el robot. Además, este código sirve como base de referencia para el desarrollo de los demás programas utilizados en el proyecto, ya que establece la lógica principal de funcionamiento del sistema.

Se puede ver el código en el repositorio github de este proyecto: "LegoSpikePrimeProyecto1Sorting".

Version1.0.0.py > ...

```python
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor, ColorSensor
from pybricks.parameters import Port, Color
from pybricks.tools import wait


hub = PrimeHub()

sensor_color = ColorSensor(Port.B)
motor_posicion = Motor(Port.A)
motor_empuje = Motor(Port.D)

while True:
    color_detectado = sensor_color.color()

    if color_detectado == Color.RED or color_detectado == Color.YELLOW:


        motor_posicion.run_target(1000, -35)
        if color_detectado == Color.RED:

            motor_empuje.run_angle(1000, -180)

        elif color_detectado == Color.YELLOW:
            motor_empuje.run_angle(1000, 180)


    elif color_detectado == Color.GREEN or color_detectado == Color.BLUE:

        motor_posicion.run_target(1000, 35)
        if color_detectado == Color.GREEN:
            motor_empuje.run_angle(1000, 180)

        elif color_detectado == Color.BLUE:
            motor_empuje.run_angle(1000, -180)

    wait(10)
```

Figura 5: *"Código Control_Automatico_Archivo_Base.py".*

Explicación del código del servidor lógico

Línea 1–4:

Se importan las librerías necesarias de Pybricks. Estas permiten utilizar el hub, los motores, el sensor de color, los puertos físicos del robot y funciones auxiliares como pausas de tiempo.

Línea 6:

Se crea una instancia del PrimeHub, que corresponde al hub físico del robot y actúa como el núcleo de control del sistema.

Línea 9:

El sensor implementado en el robot se guarda como referencia en la variable sensor_color, utilizando el puerto B. Este sensor se encarga de detectar el color del

bloque que ingresa al sistema.

Línea 10:

Se inicializa el motor_posicion en el puerto A, el cual se utiliza para posicionar el mecanismo en la casilla correspondiente.

Línea 11:

Se inicializa el motor_empuje en el puerto D, encargado de empujar o disparar el bloque hacia el lado correspondiente según el color detectado.

Línea 13:

Se implementa un bucle while True, el cual permite que el robot funcione de manera continua. El programa se mantiene en ejecución hasta que se detiene manualmente.

Línea 14:

El color detectado por el sensor se almacena en la variable color_detectado, lo que permite utilizar este valor en la lógica de decisión del sistema.

Líneas 16–36:

En esta sección se encuentra el algoritmo principal de clasificación, el cual permite distribuir los bloques en cuatro casillas distintas. Para ello, se utilizan estructuras condicionales if y elif que evalúan el color detectado:

Si el color es rojo o amarillo, el motor de posición mueve el mecanismo hacia una casilla específica.

Si el color es rojo, el motor de empuje dispara el bloque hacia un lado mediante un giro de −180°.

Si el color es amarillo, el motor de empuje lo hace hacia el lado opuesto con un giro de 180°.

Si el color es verde o azul, el motor de posición se desplaza hacia otra casilla.

Para el color verde, el motor de empuje gira 180°.

Para el color azul, el motor de empuje gira −180°.

De esta forma, el sistema clasifica automáticamente los bloques según su color y los distribuye en la casilla correspondiente.

Línea 38:

Se utiliza la función wait(10) para introducir una pequeña pausa, lo que evita lecturas excesivamente rápidas del sensor y mejora la estabilidad del sistema.

Forma 2: archivo Archivo_Control_Teclado.py

En esta versión se ocupa de base el código anterior pero se implementa una forma de manejar el robot, en este caso sería mediante teclas.

Version1.0.1.py > ...

```python
# este codigo lee la entrada del teclado para controlar el robot
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor, ColorSensor
from pybricks.parameters import Port, Color
from pybricks.tools import wait
from usys import stdin
from uselect import poll


hub = PrimeHub()

sensor_color = ColorSensor(Port.B)
motor_posicion = Motor(Port.A)
motor_empuje = Motor(Port.D)

teclado = poll()
teclado.register(stdin)

while True:
    if teclado.poll(0):
        key = stdin.read(1)
        if key=='w':
            motor_posicion.run_target(1000, -35)
            motor_empuje.run_angle(1000, -180)

        elif key == 'a':
            motor_posicion.run_target(1000, -35)
            motor_empuje.run_angle(1000, 180)
        elif key=='s':
            motor_posicion.run_target(1000, 35)
            motor_empuje.run_angle(1000, 180)

        elif key=='d':
            motor_posicion.run_target(1000, 35)
            motor_empuje.run_angle(1000, -180)

    wait(10)

```

Figura 6: *"Codigo Archivo_Control_Teclado.py".*

Puntos claves de este código:

Línea 6–7:

Se importan los módulos stdin y poll, los cuales permiten leer la entrada del teclado en tiempo real y detectar cuándo una tecla es presionada por el usuario.

Línea 15:

Se crea la variable teclado utilizando poll(), la cual se encarga de monitorear

continuamente las entradas del teclado.
Línea 16:
Se registra la entrada estándar (stdin) en la variable teclado, habilitando la lectura de las teclas presionadas.
Línea 19:
Dentro del bucle principal, se utiliza teclado.poll(0) para verificar si existe alguna entrada del teclado disponible sin detener la ejecución del programa.
Línea 20:
La tecla presionada por el usuario se lee y se almacena en la variable key.
Líneas 21–36:
Se implementa una estructura condicional que asocia distintas teclas del teclado con movimientos específicos del robot que son respectivamente con las casillas que se tienen, además de los colores que son 4.

Forma 3: Archivo_Implementado_Tkinter.py

Este archivo corresponde a la versión final del sistema, la cual permite el control del robot mediante teclas y una interfaz gráfica desarrollada con Tkinter. En este archivo se integran y unifican las funcionalidades de los dos códigos desarrollados anteriormente.

La principal novedad de este código es la implementación de la conexión mediante Bluetooth, realizada a través de la librería pybricksdev.connections.pybricks, desde la cual se importa el método PybricksHubBLE. Esta implementación permite utilizar el Bluetooth como medio de comunicación entre el cliente y el hub. Además, posibilita el envío de scripts al hub, los cuales, en este caso, corresponden a los archivos desarrollados previamente, permitiendo su ejecución directa en el dispositivo.

```
106  class BLEWorker:
107      def __init__(self, log_queue, status_cb):
108          self.loop = asyncio.new_event_loop()
109          self.queue = asyncio.Queue()
110          self.log_queue = log_queue
111          self.status_cb = status_cb
112          self.thread = threading.Thread(target=self._run, daemon=True)
113          self.hub = None
114
115      def log(self, msg):
116          self.log_queue.put(msg)
117
118      def start(self):
119          if not self.thread.is_alive():
120              self.thread.start()
121
122      def send(self, cmd):
123          self.loop.call_soon_threadsafe(self.queue.put_nowait, cmd)
124
125      def _run(self):
126          asyncio.set_event_loop(self.loop)
127          self.loop.run_until_complete(self.main())
129      async def main(self):
130          self.log("Buscando Hub Bluetooth...")
131          device = await find_device("SP----1")
132
133          if not device:
134              self.log("No se encontró el Hub")
135              self.status_cb(False)
136              return
137
138          self.hub = PybricksHubBLE(device)
139          await self.hub.connect()
140          self.log("Conectado al Hub")
141          self.status_cb(True)
142
143          while True:
144              cmd = await self.queue.get()
145              await execute(self.hub, cmd, self.log)
146
```

Figura 7: *"Código Archivo_Implementado_Tkinter.py".*

Línea 106:

Se define la clase BLEWorker, la cual se encarga de gestionar la comunicación Bluetooth entre el cliente y el hub utilizando Pybricks.

Líneas 107–114:

En el método __init__, se inicializan los componentes principales de la clase. Se crea un nuevo bucle de eventos de asyncio, una cola asíncrona para el envío de comandos, una cola de logs para registrar mensajes del sistema y un callback (status_cb) que informa el estado de la conexión. Además, se crea un hilo de ejecución independiente para manejar la comunicación Bluetooth sin bloquear la interfaz gráfica. La variable hub se inicializa como None.

Líneas 115–116:

Se define el método log, el cual envía mensajes a la cola de logs para su posterior visualización en la interfaz gráfica.

Líneas 118–120:

El método start verifica si el hilo de comunicación ya está activo. En caso de no estarlo, inicia el hilo encargado de ejecutar la comunicación Bluetooth.

Líneas 122–123:

El método send permite enviar comandos al hub de forma segura desde otros hilos, agregando dichos comandos a la cola asíncrona mediante el bucle de eventos.

Líneas 125–127:

El método privado _run configura el bucle de eventos de asyncio y ejecuta de forma continua la función principal de comunicación.

Líneas 129–130:

En el método asíncrono main, se inicia el proceso de búsqueda del hub Bluetooth. Se registra un mensaje indicando que el sistema está buscando el hub.

Líneas 131–136:

Se utiliza la función find_device para buscar el hub identificado como "SP----1". Si el dispositivo no es encontrado, se registra el error y se notifica a la interfaz gráfica que la conexión ha fallado.

Líneas 138–141:

Si el hub es encontrado, se crea una instancia de PybricksHubBLE, se establece la conexión Bluetooth y se registra un mensaje indicando que la conexión fue exitosa. Además, se actualiza el estado de conexión en la interfaz gráfica.

Líneas 142–146:

Se implementa un bucle infinito que espera comandos desde la cola. Cada comando recibido es enviado al hub mediante la función execute, permitiendo la ejecución remota de instrucciones en el robot.

6.2.1 Cliente

En el sistema desarrollado se distinguen dos tipos de cliente: el cliente físico y el cliente lógico.

El cliente físico corresponde al dispositivo desde el cual el usuario interactúa directamente con el sistema. Considerando únicamente el archivo final del proyecto, el cliente físico es el computador, ya que la interfaz desarrollada con Tkinter se visualiza en el monitor y permite la interacción directa del usuario a través del teclado y la pantalla.

Por otro lado, el cliente lógico se refiere al software encargado de gestionar la interacción con el usuario, específicamente la interfaz gráfica. En este sistema, dicho rol lo cumple Tkinter, el cual se encarga de mostrar un botón para cada color, facilitando el control manual del robot. Además, las teclas de dirección cumplen funciones específicas: la flecha derecha representa la dirección 1, mientras que la flecha izquierda corresponde a la dirección 2. Asimismo, la flecha hacia arriba ejecuta un giro de 180° del motor de empuje hacia la derecha, y la flecha hacia abajo realiza un giro de 180° del motor de empuje hacia la izquierda.

```
152  class App:
153      def __init__(self, root):
154          self.root = root
155          root.title("Control LEGO - Proyecto Final")
156          root.geometry("550x600")
157
158          self.logs = Queue()
159          self.worker = BLEWorker(self.logs, self.update_status)
160
161          # -------- Estado conexión --------
162          ttk.Label(root, text="Estado de conexión").pack()
163          self.status_panel = tk.Canvas(root, width=40, height=40)
164          self.status_panel.pack()
165          self.status_circle = self.status_panel.create_oval(5, 5, 35, 35, fill="red")
166
167          ttk.Button(root, text="Conectar", command=self.worker.start).pack(pady=5)
168
169          # -------- Automático --------
170          ttk.Button(root, text=" MODO AUTOMÁTICO",
171                     command=lambda: self.worker.send("auto")).pack(pady=8)
172
173          ttk.Button(root, text=" STOP",
174                     command=lambda: self.worker.send("stop")).pack(pady=5)
175
```

```
        # -------- Colores --------
        frame = ttk.LabelFrame(root, text="Control manual por color")
        frame.pack(pady=10)

        ttk.Button(frame, text=" ROJO", command=lambda: self.send_color("red")).grid(row=0, column=0, padx=5)
        ttk.Button(frame, text=" AMARILLO", command=lambda: self.send_color("yellow")).grid(row=0, column=1, padx=5)
        ttk.Button(frame, text=" VERDE", command=lambda: self.send_color("green")).grid(row=1, column=0, padx=5)
        ttk.Button(frame, text=" AZUL", command=lambda: self.send_color("blue")).grid(row=1, column=1, padx=5)

        # -------- Indicador color --------
        ttk.Label(root, text="Color").pack()
        self.color_canvas = tk.Canvas(root, width=120, height=120)
        self.color_canvas.pack()
        self.color_circle = self.color_canvas.create_oval(20, 20, 100, 100, fill="gray")

        # -------- Logs --------
        ttk.Label(root, text="Registro").pack()
        self.text = tk.Text(root, height=8)
        self.text.pack(fill="both", expand=True, padx=10)

        # -------- Teclado --------
        root.bind("<Up>", lambda e: self.worker.send("emp_up"))
        root.bind("<Down>", lambda e: self.worker.send("emp_down"))
        root.bind("<Right>", lambda e: self.worker.send("pos_right"))
        root.bind("<Left>", lambda e: self.worker.send("pos_left"))

        self.update_logs()

    def update_status(self, connected):
        self.status_panel.itemconfig(
            self.status_circle,
            fill="green" if connected else "red"
        )

    def send_color(self, color):
        self.worker.send(color)
        self.update_color(color.upper())

    def update_color(self, color):
        colors = {
            "RED": "red",
            "YELLOW": "yellow",
            "GREEN": "green",
            "BLUE": "blue"
        }
        self.color_canvas.itemconfig(self.color_circle, fill=colors.get(color, "gray"))

    def update_logs(self):
        try:
            while True:
                msg = self.logs.get_nowait()
                self.text.insert("end", msg + "\n")
                self.text.see("end")

                if "COLOR:" in msg:
                    self.update_color(msg.split(":")[1])

        except Empty:
            pass

        self.root.after(200, self.update_logs)
```

Figura 8: *Código parte GUI Archivo_Implementado_Tkinter.py.*

Líneas 152–156:

Se define el método __init__, el cual inicializa la ventana principal de la aplicación. En esta sección se configura el título y el tamaño de la ventana, estableciendo los parámetros visuales iniciales de la interfaz gráfica.

Líneas 158–159:

Se crea una cola de logs y una instancia de la clase BLEWorker, encargada de gestionar la comunicación Bluetooth con el hub. Además, se establece un callback para actualizar el estado de conexión en la interfaz.

Líneas 162–165:

Se implementa el indicador de estado de conexión, compuesto por una etiqueta descriptiva y un círculo que cambia de color. Este indicador muestra visualmente si el sistema se encuentra conectado o no al hub.

Línea 167:

Se crea un botón “Conectar”, el cual inicia el proceso de conexión Bluetooth al ejecutar el método start del BLEWorker.

Líneas 170–175:

Se agregan los botones correspondientes al modo automático y a la opción STOP, los cuales envían los comandos "auto" y "stop" al servidor para controlar el funcionamiento del robot.

Líneas 177–183:

Se crea un panel destinado al control manual por color, donde se incluyen botones para los colores rojo, amarillo, verde y azul. Cada botón envía el color correspondiente al servidor cuando es presionado.

Líneas 186–189:

Se implementa un indicador visual de color, representado por un círculo que cambia dinámicamente según el color seleccionado o recibido, proporcionando retroalimentación visual al usuario.

Líneas 192–194:

Se define el área de registro de eventos, donde se muestran los mensajes del sistema, tales como estados de conexión, comandos enviados y colores detectados.

Líneas 197–200:

Se habilita el control del robot mediante el teclado, asociando las teclas de dirección a comandos específicos que son enviados al servidor para controlar la posición y el motor de empuje.

Líneas 204–208:

Se define el método update_status, el cual actualiza el color del indicador de conexión según el estado recibido desde el BLEWorker.

Líneas 210–212:

El método send_color envía el color seleccionado al servidor y actualiza el indicador visual correspondiente en la interfaz.

Líneas 214–221:

El método update_color se encarga de cambiar el color del indicador gráfico según el valor recibido, utilizando un diccionario que asocia los nombres de los colores con su representación visual.

Líneas 223–236:

El método update_logs revisa periódicamente la cola de logs, muestra los mensajes en el área de registro y actualiza el indicador de color cuando se detectan mensajes relacionados con el color. Este método se ejecuta de forma cíclica mediante el temporizador de la interfaz.

Respecto a cómo se aprecia el cliente en este código, esto se realiza mediante el uso del comando lambda, el cual ejecuta la función send("") (Esta función se encuentra en la línea 122) con un String dentro de las comillas. Dicho string es posteriormente enviado al servidor lógico.

El comando se envía cuando el usuario presiona el botón correspondiente a la función que desea ejecutar. Además, el sistema permite el uso de las teclas arriba, abajo, izquierda y derecha, las cuales también envían comandos al servidor para controlar el robot.

```
196         # -------- Teclado --------
197         root.bind("<Up>", lambda e: self.worker.send("emp_up"))
198         root.bind("<Down>", lambda e: self.worker.send("emp_down"))
199         root.bind("<Right>", lambda e: self.worker.send("pos_right"))
200         root.bind("<Left>", lambda e: self.worker.send("pos_left"))
201
202         self.update_logs()
```

Figura 9: *"Código parte Teclado Archivo_Implementado_Tkinter.py".*

En el ejemplo se puede apreciar lo explicado anteriormente acerca del teclado, Up, Down, Right y Left corresponden respectivamente a arriba, abajo, derecha e izquierda. Además se envía un String que va ser la clave de un diccionario y de esa

forma el valor seria el script que se manda al hub, lo cual pertenece a la parte del servidor lógico.

6.2.2 Servidor

En el sistema desarrollado se distinguen dos tipos de servidores: el servidor físico y el servidor lógico.

El servidor físico corresponde al dispositivo encargado de recibir los mensajes enviados por el cliente. En este proyecto, dicho servidor es el LEGO Technic Large Hub, el cual recibe el código y se comunica con sus distintos puertos para ejecutar las acciones correspondientes. Estas acciones incluyen el control de los motores, así como el uso de sensores y otras funcionalidades integradas en el hub.

Por otro lado, el servidor lógico se refiere al software responsable de gestionar la comunicación con el cliente, en este caso viene a ser con el método PybricksHubBLE de la librería pybricksdev.connections.pybricks, el cual se explicó en la Sección 6.2 y también los scripts que se envían con este método.

A continuación, se muestra los scripts que se crean en el cliente y luego se envian al hub el cual el mismo servidor se encargará de interpretar y ejecutar en el cliente físico:

```
60
61      commands = {
62          "red": "print('COLOR:RED'); motor_pos.run_target(300,-35); motor_emp.run_angle(1000,-180)",
63          "yellow": "print('COLOR:YELLOW'); motor_pos.run_target(300,-35); motor_emp.run_angle(1000,180)",
64          "green": "print('COLOR:GREEN'); motor_pos.run_target(300,35); motor_emp.run_angle(1000,180)",
65          "blue": "print('COLOR:BLUE'); motor_pos.run_target(300,35); motor_emp.run_angle(1000,-180)",
66          "emp_up": "motor_emp.run_angle(1000,180)",
67          "emp_down": "motor_emp.run_angle(1000,-180)",
68          "pos_right": "motor_pos.run_target(300,35)",
69          "pos_left": "motor_pos.run_target(300,-35)",
70      }
71
72      if cmd in commands:
73          return base + commands[cmd] + "\nwait(100)"
74
75      if cmd == "stop":
76          return """
77  from pybricks.pupdevices import Motor
78  from pybricks.parameters import Port
79  Motor(Port.D).stop()
80  Motor(Port.A).stop()
81  """
```

Figura 10: *Código comandos Archivo_Implementado_Tkinter.py.*

Las instrucciones que se envían al hub corresponden principalmente a métodos de la librería Pybricks, específicamente run_target y run_angle.
El método run_target se utiliza cuando se requiere mover un motor hacia una posición específica. Este método es fundamental para posicionar el motor en las casillas designadas dentro del sistema, asegurando un desplazamiento preciso y controlado.

Por otro lado, el método run_angle se emplea para mover el motor en función de un ángulo determinado. En este proyecto, este método se utiliza para accionar el mecanismo que dispara el bloque hacia el lado izquierdo o derecho. Por esta razón, los ángulos se representan como 180° y −180°, indicando la dirección del movimiento.

Por último, los comandos se estructuran como un diccionario clave–valor, donde la clave es proporcionada por los botones de la interfaz gráfica. Estas claves se envían mediante PybricksHubBLE, permitiendo transmitir la instrucción correspondiente.

El valor asociado a cada clave corresponde al script que debe ejecutarse. Dicho script es recibido por el servidor físico (LEGO Technic Large Hub), el cual se encarga

de interpretar la instrucción y ejecutar la acción correspondiente en el robot.

6.2.3 Interfaz gráfica de usuario (GUI)

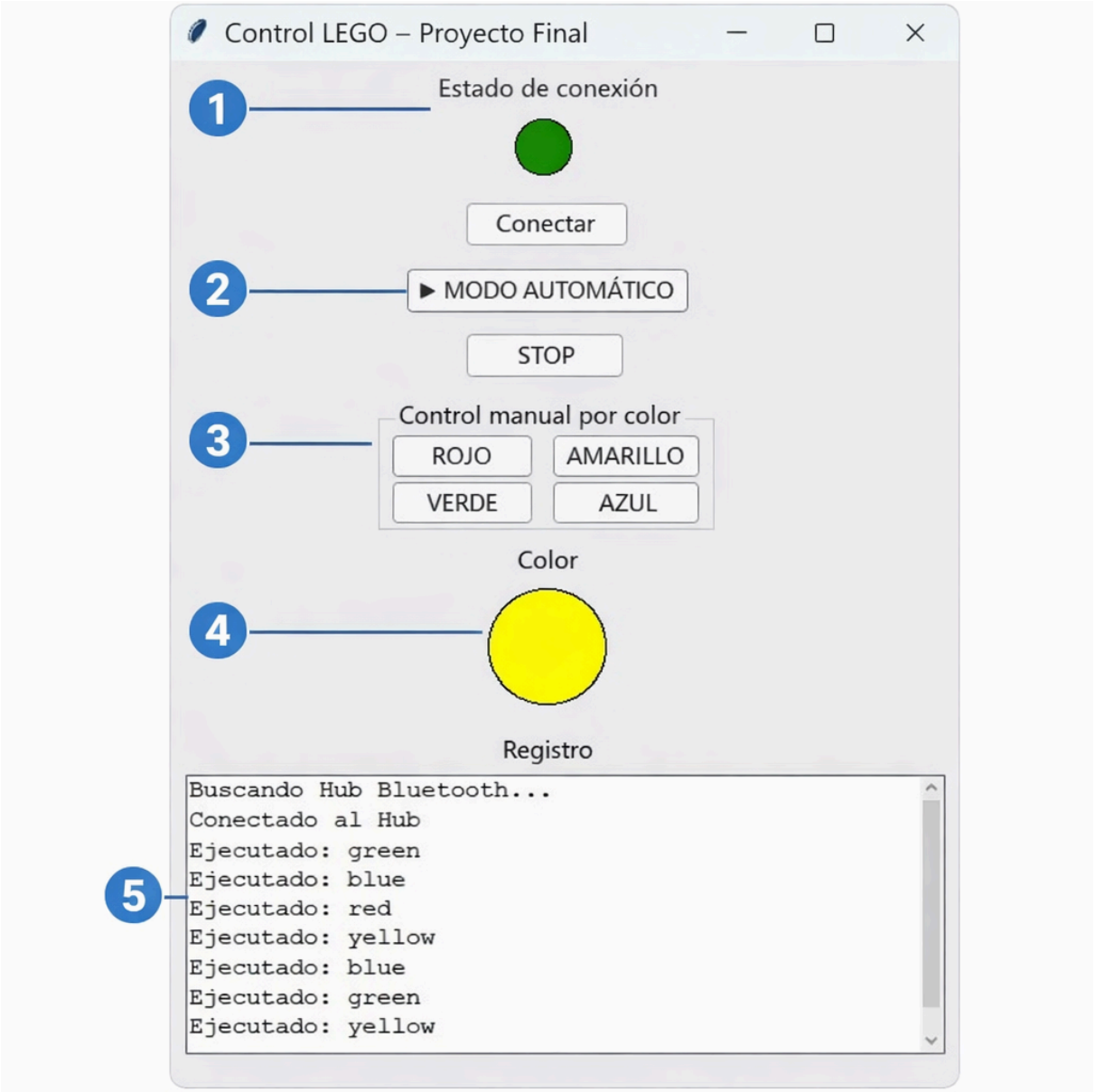

Figura 11: *Interfaz Gráfica de Usuario.*

1. Indicador de estado de conexión

   Representado mediante un indicador visual que refleja el estado actual de la conexión con el robot. Inicialmente se encuentra en estado desconectado. Al presionar el botón Conectar, el sistema inicia una búsqueda de dispositivos Bluetooth, cambiando su estado a buscando. En caso de detectar correctamente la señal específica del robot sorter,

el indicador cambia a estado conectado, proporcionando retroalimentación inmediata al usuario.

2. controles de operación (Modo Automático y Stop)

   Este sector corresponde a los controles principales de ejecución del sistema. El botón Modo automático envía una instrucción el hub para iniciar el control autonomo del robot basado en la lectura del sensor de color. El botón STOP permite detener de forma inmediata cualquier acción en ejecución, garantizando un control seguro del sistema.

3. Control manual por color

   Representado por un conjunto de cuatro botones identificados por color (rojo, amarillo, verde y azul). Cada botón envía un comando específico al hub del robot sorter, indicando la acción que debe ejecutar según el color seleccionado. Este control permite al usuario intervenir manualmente en el proceso de clasificación cuando sea necesario.

4. Indicador de color detectado

   Este elemento visual muestra el color actualmente detectado por el sensor del robot, Cuando no existe conexión o no se detecta ningún color, el indicador permanece en estado neutro. Al recibir información desde el servidor lógico, el indicador cambia al color correspondiente, facilitando el ordenamiento y otorgando claridad visual al usuario.

5. Panel de registro (logs)

   Corresponde al panel informativo donde se muestran los mensajes generados por el sistema, tales como el estado de conexión, comandos enviados y eventos relevantes (por ejemplo, ejecución de instrucciones o deteccion de color) Este panel permite supervisar el funcionamiento del sistema y facilita la depuración durante las pruebas.

7. Resultados

7.1 Estado actual del proyecto

El estado actual de la solución planteada (Maquinaria de clasificación de materiales) se encuentra en una fase de funcionalidad completa respecto a su mecánica y

cumpliento de los requerimientos funcionales definidos.

El robot ha logrado exitosamente la automatización total de clasificar bloques respecto a su color tanto de manera manual guiada por el usuario a través de la interfaz gráfica implementada con tkinter. La solución es capaz de operar de manera automática y manual replicando la tarea industrial sin exponer a funcionarios de empresas mineras.

El robot ha logrado alcanzar los siguientes requerimientos funcionales:

-Identificacion de Materiales (RF1): Mediante el sensor de color el robot es capaz de reconocer el color del bloque y separarlo en base a ello.

-Modo de operación Híbrida (RF4): El robot ha sido capaz de operar de manera automática, o sea, un bucle de ordenamiento ejecutando la clasificación de estos mismo bloques hasta no detectar ninguno. Como a su vez un modo manual que implementa una serie de botones para que el usuario pueda clasificar los bloques el mismo, todo esto gracias a la interfaz gráfica implementada.

-Flujo Continuo (RF2): El robot es capaz de reconocer y ordenar varios bloques de LEGO sucesivamente , manteniendo un constante control de la clasificación.

-Clasificación de Celdas por su color (RF3): El robot es capaz de moverse y empujar el bloque respecto al destino de su color, capaz de sincronizar el movimiento de posición con el empuje del bloque para que el bloque sea colocado en base a su color, enfatizando en su ordenamiento.

RF5-Software**:** El sistema cumple con el criterio de RF5 el cual es capaz de procesar la lectura del sensor de color y enviar la orden a los motores, obteniendo un flujo correcto de instrucciones.

RF6-Software: El software es capaz de permitir el uso manual del robot con la interfaz implementada (Tkinter).

Respecto a los requerimientos no funcionales del robot el reporte actual son los siguientes:

-Disponibilidad: La disponibilidad del robot depende netamente de su conexión con el programa, ante una pérdida de conexión con el hub y Tkinter el sistema se detiene por completo. Ante estos casos el robot debe ser reiniciado o desconectado con el hub y reintegrar la misma conexión, siendo su restauración casi de manera inmediata.

-Robustez: El robot no es tan robusto respecto al manejo mecánico debido a que si llega a presentar obstrucción por bloques, necesita intervención inmediata para el correcto flujo, carece de un mecanismo de auto barrido ante estancamiento.

-Rendimiento: El robot en condiciones de flujo normales tanto automáticas como manual cumple con los tiempos de respuesta esperados.

-Usabilidad: Para mejorar la usabilidad y comodidad se implementó un esquema en base a unos botones modelados simples, con frases explicativas de la funcionalidad a lograr, primero se conecta a través de un botón simple llamado “Conectar” el cual busca el hub y logra la conexión, una vez conectado el usuario puede seleccionar si quiere un modo automático de ordenamiento el cual se ejecuta en bucle hasta no detectar un color o también seleccionar control manual mediante la selección del usuario a través de un botón en base al color previsto y se ejecuta de manera inmediata el ordenamiento. La interfaz al ser clara y explicativa logra concisamente un tiempo de capacitación corto.

7.2 Problemas encontrados y solucionados

Durante el desarrollo del proyecto, se encontraron diferentes problemas asociadas al uso del framework Flutter para el desarrollo de la interfaz gráfica. Estos problemas se relacionan con las limitaciones del framework como al contexto por el que se quiere usar la interfaz gráfica. La aparición constante de estos problemas generó un riesgo de impacto medio-alto según los niveles de riesgo, debido a que

puede afectar la continuidad del desarrollo. Frente a esto, y con el objetivo de solucionar el problema, el equipo decidió dejar de usar el framework Flutter y optar por una alternativa más acorde a las necesidades del proyecto, se implementó la biblioteca Tkinter, la cual permitió un mayor control sobre el desarrollo de la interfaz como también eliminar los problemas de fases anteriores.

8. Prueba de funcionamiento del sistema

8.1. Descripción de prueba de funcionamiento.

La prueba que se establece para probar el funcionamiento correcto del robot se basa en el correcto posicionamiento del bloque de Lego en la rampa de encolado del robot donde se debe deslizar por el riel hacia la zona donde se ubica el sensor de color, el cual debera identificar el color del bloque dando paso al correcto almacenamiento de este en su respectivo compartimiento dando fin al proceso correspondiente al robot.

8.2. Resultados observados para la prueba de funcionamiento.

Durante la prueba de funcionamiento, el bloque de LEGO fue correctamente posicionado en la rampa de encolado, deslizándose de forma continua hasta la zona del sensor de color. El sensor identificó correctamente el color del bloque y el sistema ejecutó la secuencia programada, moviendo los motores y depositando el bloque en su compartimento correspondiente sin interrupciones ni errores. Con esto se valida el correcto funcionamiento del proceso completo de clasificación.

La interfaz se utiliza mediante los botones mostrados anteriormente, el usuario solo debe elegir el botón a seleccionar si será de manera manual o automática, todo a su vez lleva al mismo escenario del resultado observado de la prueba de funcionamiento.

Las acciones desde la interfaz de usuario independiente del botón seleccionado mandan señal vía bluetooth al Hub para la correcta orden de ejecución, a continuación adjunto un video e imagen del botón seleccionado para mostrar como

se envio la acción desde la interfaz de usuario y se representa visualmente la orden de prueba que llega al robot, con ello validando como se completa la prueba definitiva.

Demo Visual:

WhatsApp Video 2025-12-29 at 11.29.38 AM.mp4

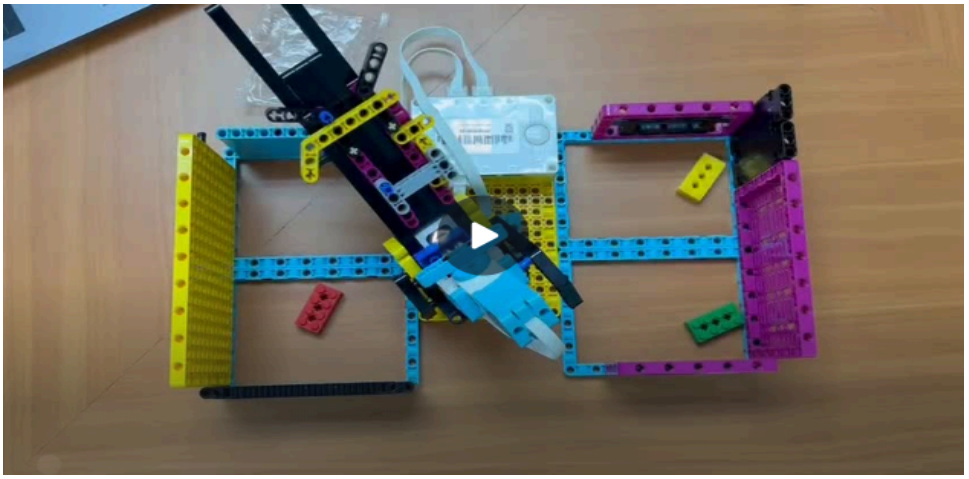

Figura 12: *DEMO*.

A continuación se adjunta los botones seleccionados acorde al video “Demo”, demostrando cómo el robot ejecuta las órdenes desde la interfaz y cómo lleva a cabo su prueba de funcionamiento completa.

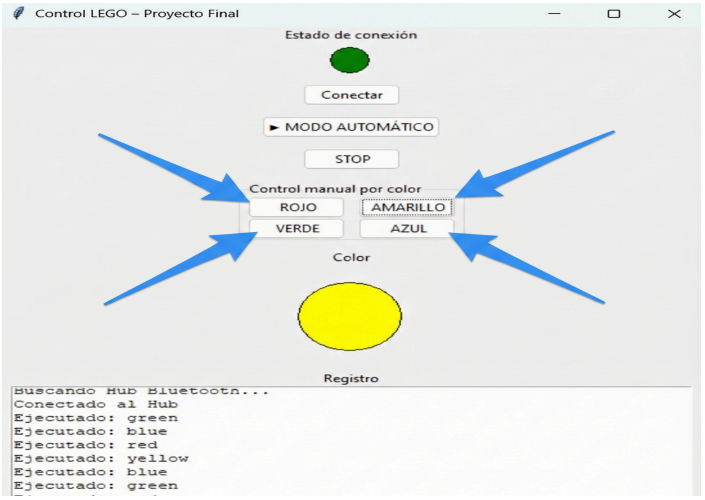

Figura 13: *Botones Seleccionados.*

9. Conclusión

Este proyecto nos permitió desarrollar una solución robótica basada en LEGO, cumpliendo el objetivo de simular un proceso de clasificación de materiales propio de la industria minera. A lo largo del desarrollo se lograron avances significativos, como la construcción de un robot capaz de clasificar bloques por color, la implementación de modos de operación automático o manual, y el diseño de una arquitectura cliente-servidor. Durante el proyecto se presentaron problemas asociados a decisiones tomadas en fases iniciales, particularmente en relación con el uso del framework Flutter para la interfaz grafica como tambien malas decisiones para escoger un modelo de robot provocando que tuvieramos 3 prototipos, para solucionarlo se tuvo que migrar a Tkinter y quedarnos con un modelo de robot en el cual nos sintamos cómodos a la hora de optimizarlo.

6. Referencias

Saoud Ahmed Lanchipa. (2025). LegoSpikePrimeProyecto1Sorting [Repositorio]. GitHub. https://github.com/saoudahmedlanchipa-code/LegoSpikePrimeProyecto1Sorting

Ubuy. (s. f.). *LEGO Education SPIKE Prime Set (45678)*. Recuperado el 15 de octubre de 2025, de https://www.ubuy.cl/sp/product/GMEH1T4-lego-education-spike-prime-set.

Tradeinn. (s. f.). *LEGO Education SPIKE Prime Expansion Set (45681)*. Recuperado el 15 de octubre de 2025, de https://www.tradeinn.com/kidinn/es/lego-juego-de-construccion-education-spike-prime-expansion-set-45681/141562226/p.

Lenovo. (s. f.). *Lenovo LOQ 15IAx9*. Recuperado el 15 de octubre de 2025, de https://www.lenovo.com/cl/es/p/notebooks/loq-laptops/lenovo-loq-15iax9/len101q0006.

RIPLEY. (s. f.). *TABLET SAMSUNG GALAXY TAB S8 PLUS + KEYBOARD COVER AMD RYZEN 9 8 GB RAM 12.4".* Recuperado el 15 de octubre de 2025, de TABLET SAMSUNG GALAXY TAB S8 PLUS + KEYBOARD COVER AMD RYZEN 9 8 GB RAM 12.4"

Paris.cl. (s. f.). *Notebook IdeaPad Slim 3 AMD Ryzen7 5825U 16GB 512GB SSD Windows 11 Home 15.6" FHD Azul Abyss*. Recuperado el 15 de octubre de 2025, de https://www.paris.cl/notebook-ideapad-slim-3-amd-ryzen7-5825u-16gb-512gb-ssd-windows-11-home-156-fhd-azul-abyss-106450999.html.

Paris.cl. (s. f.). *Notebook Gamer Acer Aspire G A515-58GM-56XX-1: Intel Core i5, NVIDIA RTX 2050, 16GB RAM, 512GB SSD, 15.6" FHD*. Recuperado el 15 de octubre de 2025, de https://www.paris.cl/notebook-gamer-aspire-g-a515-58gm-56xx-1-intel-core-i5-8-nucleos-nvidia-rtx-2050-16gb-ram-512gb-ssd-156-700915999.html.

HP. (2025). *Notebook HP Pavilion Plus 14-ew1002la*. Paris.cl. Recuperado el 27 de noviembre de 2025, de Notebook HP Pavilion Plus 14-EW1002LA Intel Ultra 7 32GB RAM 1TB SSD 14" 3K OLED Windows 11 Home HP | Paris.cl.

Glassdoor. (2025). *Sueldos para Jefe de Proyecto en Chile (Basado en datos de Claro Chile).* Recuperado el 26 de noviembre de 2025, de KEY_NOT_FOUND: ei-salaries.seo-metadata.el-details.title.singular | Glassdoor.

Glassdoor. (2025). *Sueldos para el puesto de Programador en Chile.* Recuperado el 26 de noviembre de 2025, de Sueldo: Programador (Noviembre, 2025) | Glassdoor.

Glassdoor. (2025). *Salario Mensual para Computer Hardware and Software en Latam (Chile).* Recuperado el 26 de noviembre de 2025, de KEY_NOT_FOUND: ei-salaries.seo-metadata.el-details.title.singular | Glassdoor.

Glassdoor. (2025). *Salario mensual para Analista y Documentador Técnico en TrackTec (Chile).* Recuperado el 26 de noviembre de 2025, de KEY_NOT_FOUND: ei-salaries.seo-metadata.el-details.title.singular | Glassdoor.

Bodega Digital. (2025). *Licencia Microsoft Office 2024 Professional Plus (Código Digital)*. Recuperado el 27 de noviembre de 2025, de Office 2024 Professional Plus • Bodega Digital.

BrickLink. (s.f.). *Part 3001: Brick 2 x 4*. Recuperado el 14 de diciembre de 2025, de https://www.bricklink.com/v2/catalog/catalogitem.page?P=3001

Wikipedia. (2024). *Acrilonitrilo butadieno estireno*. En Wikipedia, la enciclopedia libre. Recuperado el 15 de diciembre de 2025, de https://es.wikipedia.org/wiki/Acrilonitrilo_butadieno_estireno.

Hurbain, P. (2024). *LEGO Motor Characteristics: Comparative Test Data*. PhiloHome. Recuperado de https://www.philohome.com/motors/motorcomp.htm

Anexos

9.Anexo 1

Se adjunta fotografía de los precios reales de los productos (Hardware) mencionados en la tabla de la sección de costo de hardware.

1. SET LEGO SPIKE PRIME.

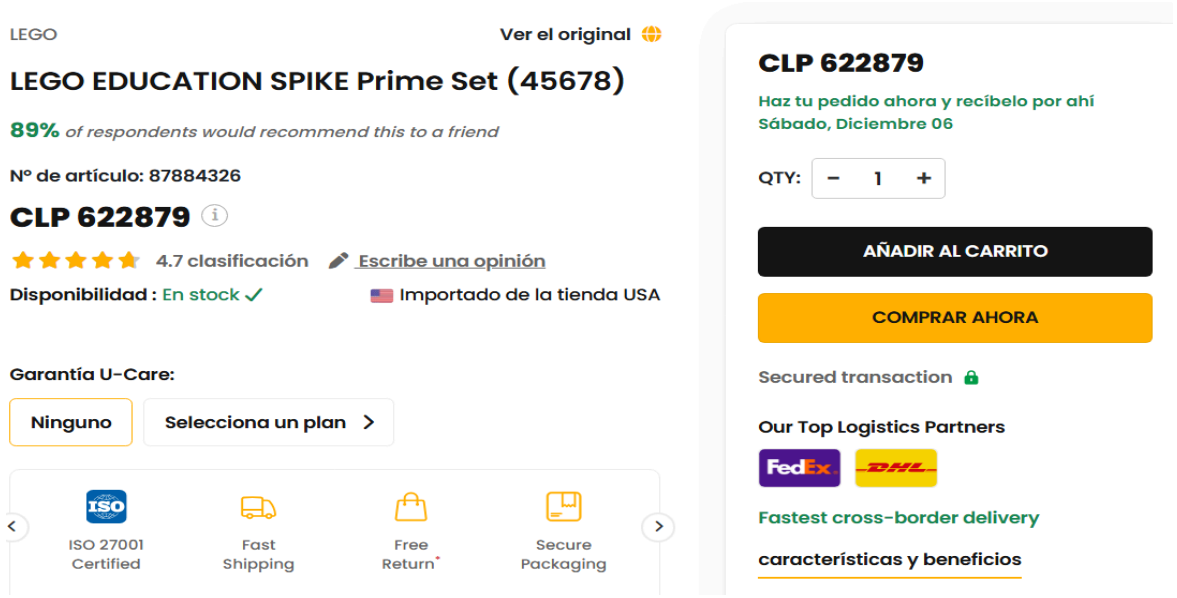

Figura 14: *LEGO SPIKE PRIME.*

2. EXTENSION LEGO SPIKE PRIME.

Figura 15: *EXTENSION LEGO SPIKE PRIME.*

3. NOTEBOOK LOQ GEN 9.

Figura 16: *Notebook loq gen 9.*

4. SAMSUNG GALAXY TAB S8 ULTRA.

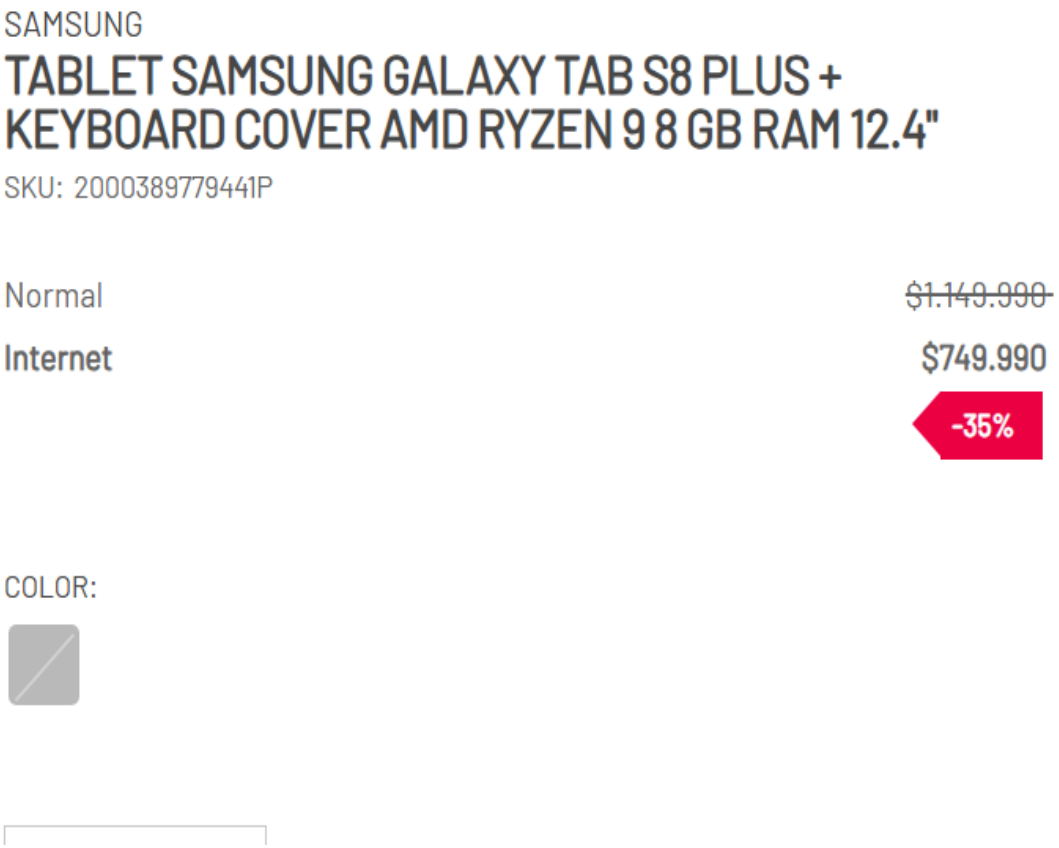

Figura 17: *GALAXY TAB S8 PLUS.*

5. LENOVO IDEAPAD 3 15.

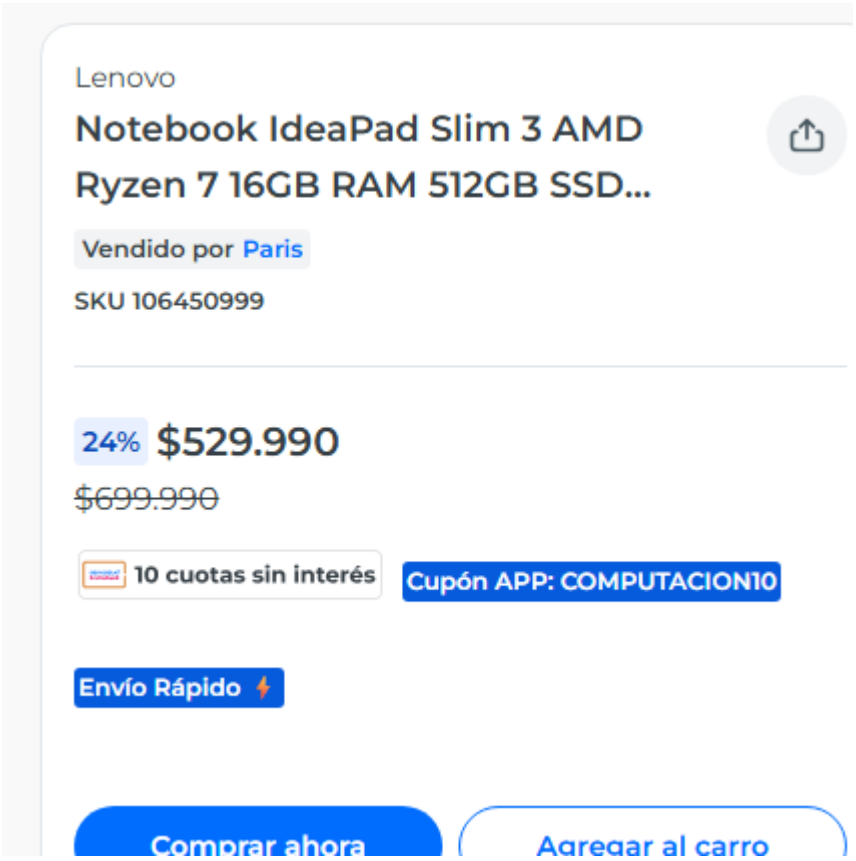

Figura 18: *IDEAPAD 3 15.*

6. NOTEBOOK ASPIRE G.

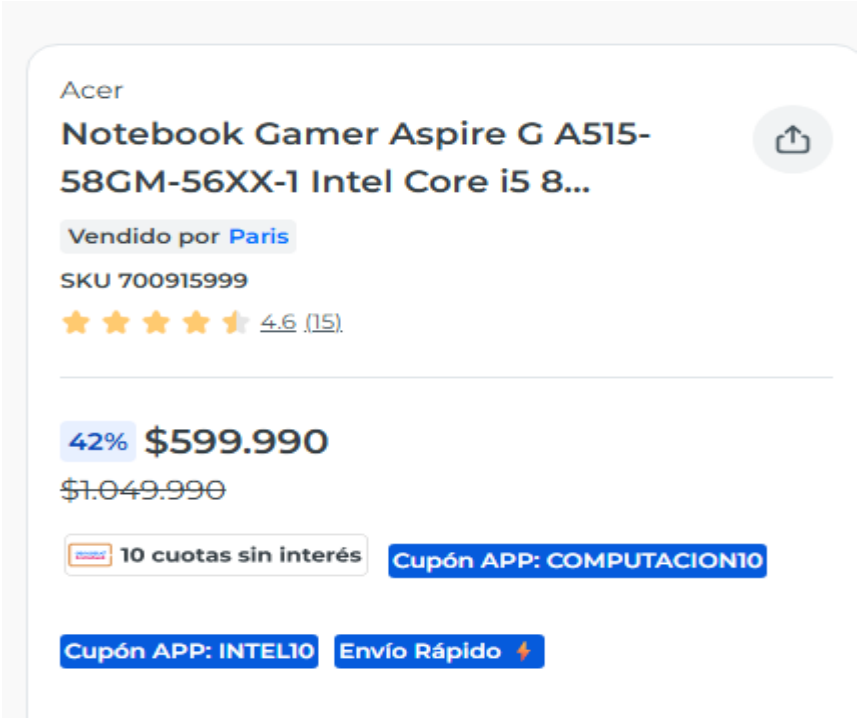

Figura 19: *GAMER ASPIRE 5.*

7. HP PAVILION LAPTOP.

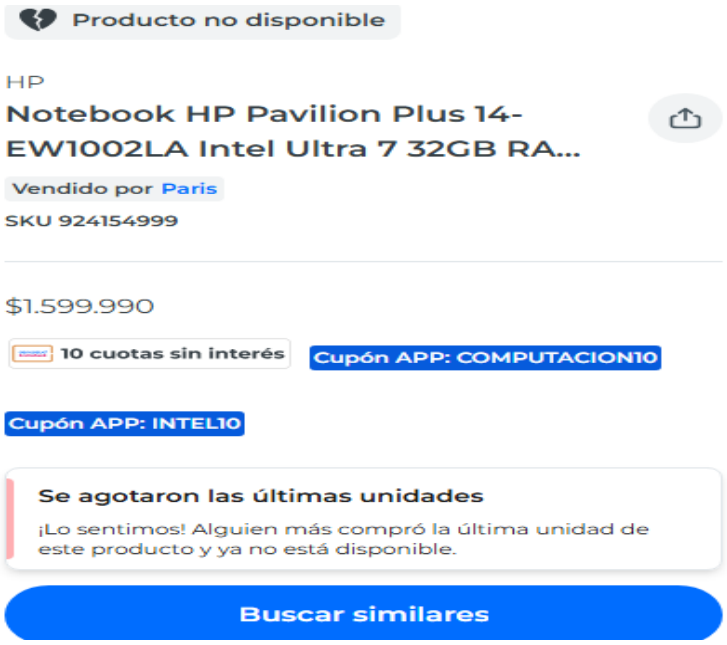

Figura 20: *HP PAVILION.*

8. TAMAÑO DEL LEGO REAL PARA LA OBTENCIÓN DE DATOS APLICANDO LA FÍSICA

Figura 21: *Bloques usados.*