UNIVERSIDAD DE TARAPACÁ
Universidad del Estado

Ingeniería
Computación e Informática

PRESENTACIÓN INFORME
“ROBOT SEPARADOR DE BLOQUES”

Integrantes:

Nicolas Olivares
Sebastián Cahuachia
Víctor Breems
Gabriel Delgado
Willy Cruz

Asignatura: Proyecto I
Profesor: Baris Nikolai Klobertanz Quiroz
Plan de Proyecto – Fase 2

DICIEMBRE 2025

Índice de CONTENIDOS

INTRODUCCIÓN

En esta presentación presentaremos el informe del proyecto fase 2 además de la corrección de errores que se presentaron en el informe. A continuación se presentará el proceso, junto con su análisis y los resultados obtenidos.

REQUERIMIENTOS FUNCIONALES

Los requerimientos funcionales describen las acciones y comportamientos que el sistema es capaz de realizar:

RF-01 – Detección de color
El sistema detecta automáticamente el color de una pieza LEGO utilizando el sensor de color del kit LEGO SPIKE PRIME.

RF-02 – Clasificación por color
El sistema clasifica las piezas detectadas en compartimentos físicos separados según su color: amarillo, azul, verde, rojo y morado.

RF-03 – Desplazamiento controlado
El robot se desplaza de forma controlada para posicionarse correctamente frente al compartimento correspondiente al color detectado.

RF-04 – Control mediante interfaz gráfica
El sistema permite iniciar, mover, pausar y reanudar el proceso de clasificación mediante una interfaz gráfica de usuario (GUI).

RF-05 – Operación autónoma
Una vez iniciado el proceso, el robot opera de forma autónoma, sin intervención del usuario durante el ciclo de clasificación.

REQUERIMIENTOS NO FUNCIONALES

Se redefinieron los RNF como atributos medibles y verificables:

RNF-01 (Disponibilidad):
El robot debe operar de forma continua durante al menos 30 minutos sin fallas.

RNF-02 (Robustez):
El sistema continúa operando ante lecturas erróneas ocasionales del sensor.

RNF-03 (Rendimiento):
Clasificación de una pieza en un tiempo máximo de 5 segundos.

RNF-04 (Usabilidad):
La interfaz permite el uso sin capacitación previa.

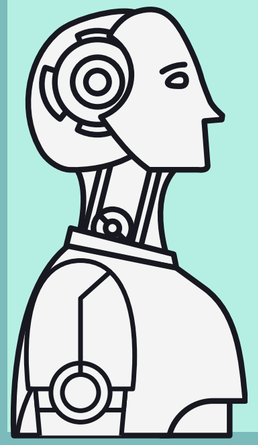
ARQUITECTURA

- Cliente
- Aplicación Python con interfaz gráfica (GUI).
- Servidor
- Hub LEGO SPIKE PRIME que ejecuta instrucciones.
- Comunicación
- Bluetooth Low Energy (BLE) mediante librería Pybricksdev.
- Generación de archivos temporales enviados al hub.

- Servidor físico: el HUB LEGO SPIKE PRIME (hardware) que ejecuta software interno (firmware/runtime) y controla motores y sensores.

  → Es “el aparato” que presta el servicio físico (mover, leer sensor).
- Servidor lógico: el software que representa/controla al hub desde el lado del PC (por ejemplo pybricksdev + pybricks como capa que construye/manda instrucciones y gestiona la comunicación).

  → Puede correr en el computador del cliente, pero cumple un rol de “servidor” porque orquesta la ejecución en el hub.



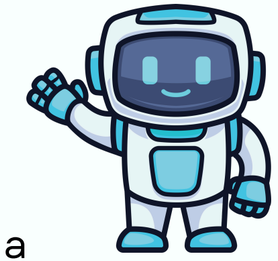
ARQUITECTURA DE SOFTWARE

En el presente proyecto se utiliza una arquitectura de tipo cliente-servidor.

- **Cliente**: Corresponde al hardware del robot. Su responsabilidad es ejecutar las instrucciones a bajo nivel a través de su firmware para activar el motor y sensor.

- **Servidor**: Está constituida con la librería de Pybricksdev que genera el archivos temporales. Su función es mandar esos archivos temporales a la Hub de LEGO SPIKE PRIME.

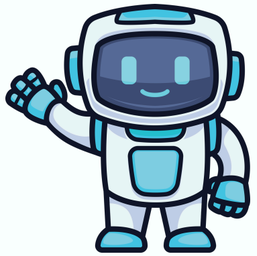
- **Comunicación**: La comunicación se realiza mediante una conexión con bluetooth con la librería de Pybricksdev que está encargado de enviar comando a la Hub de LEGO SPIKE PRIME



PROBLEMAS ENCONTRADOS Y SOLUCIONADOS

1. Problema: Dificultades para conectar la PC y el Hub de LEGO SPIKE PRIME.

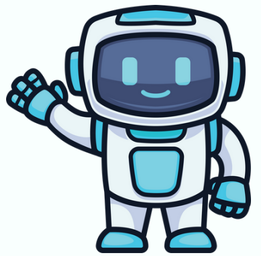
Solución: Se implementa la librería pybricksdev en Python. Esta librería optimiza la comunicación vía BLE (Bluetooth Low Energy), permitiendo la ejecución de scripts directamente en el Hub de LEGO SPIKE PRIME, eliminando el retraso que generaba el firmware original.



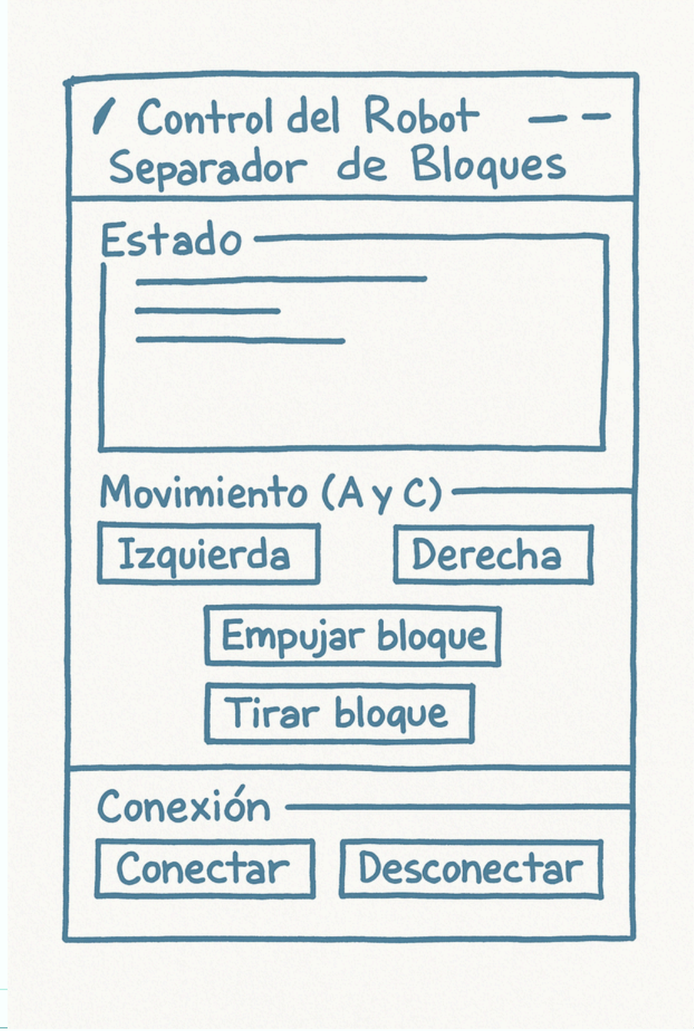
2. Problema: El sensor arrojaba lecturas erróneas debido a las condiciones de luz ambiental, confundiendo colores similares.
Solución: Se utilizó la clase ColorSensor de la librería Pybricks, la cual permite acceder a los valores crudos de reflexión y HSV (Matiz, Saturación). Esto permitió calibrar el sensor por software para filtrar el ruido lumínico, en lugar de usar la detección de color básica predeterminada.

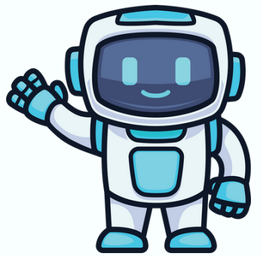
IMPLEMENTACION SERVIDOR

- El cliente genera dinámicamente programas Pybricks.
- Los archivos .py se envían temporalmente al hub.
- El hub ejecuta las instrucciones y luego se eliminan los archivos.
- Se aclaró el rol del servidor como ejecutor de comandos y controlador de motores y sensores.



Diseño inicial de la interfaz gráfica de usuario (GUI)





Fundamentos de los movimientos

Para justificar el movimiento de giro del robot, se considera un giro de 90° realizado en un tiempo aproximado de 1,5 segundos.
La velocidad angular se calcula mediante la fórmula:

$$\omega = \theta / t$$

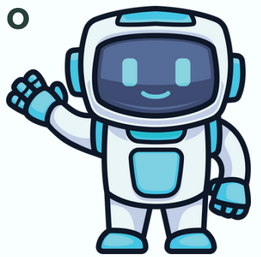
Donde:

$$\theta = 90° = \pi/2 \text{ rad}$$

$$t = 1{,}5 \text{ s}$$

$$\omega = (\pi/2) / 1{,}5 \approx 1{,}05 \text{ rad/s}$$

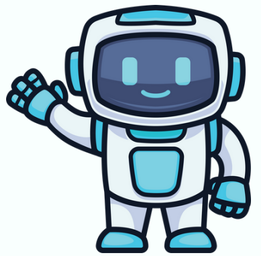
Este valor permite un giro controlado y preciso, evitando errores en la alineación del robot frente a los compartimentos.



Generación dinámica del programa que ejecuta el robot (create_program)

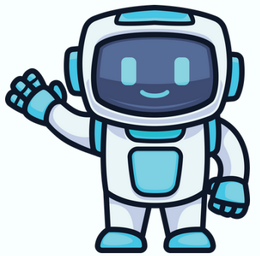
El cliente genera automáticamente un programa en Pybricks según el comando que el usuario seleccione en la interfaz

(por ejemplo: izquierda, derecha, empujar o tirar). Para ello, la función create_program(drive_cmd, claw_cmd) convierte cada comando en una instrucción de movimiento para los motores mediante run(...). Luego, el programa creado configura el hub (PrimeHub), define los motores según los puertos utilizados y ejecuta la acción solicitada. Finalmente, se incorpora un tiempo de espera breve y se detienen los motores, con el fin de mantener un movimiento controlado y evitar que el robot continúe funcionando sin supervisión.



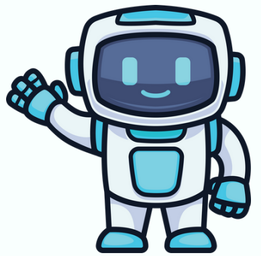
Generación dinámica del programa que ejecuta el robot (create_program)

```
def create_program(drive_cmd: str, claw_cmd: str) -> str:
    drive_commands = {
        'izquierda': "motorC.run(-300)",
        'derecha': "motorC.run(300)",
        'empujar': "motorF.run(500)",
        'tirar': "motorF.run(-500)"
    }

    drive_code = drive_commands.get(drive_cmd, "motorC.stop(); motorF.stop()")

    program = f"""
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait

hub = PrimeHub()

motorC = Motor(Port.C)
motorF = Motor(Port.F)

{drive_code}

wait(300)
motorC.stop()
motorF.stop()
"""
    return program
```



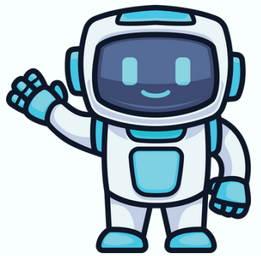
Gestión de conexión y envío de comandos por Bluetooth (BLEWorker._runner)

La conexión Bluetooth y el envío de comandos al robot se gestionan mediante un componente asíncrono que funciona como “worker”. En el método _runner() se busca el hub disponible utilizando find_device(), luego se crea la conexión con PybricksHubBLE y se establece mediante connect(). Una vez conectado, el sistema se mantiene en ejecución esperando nuevas instrucciones a través de una cola (asyncio.Queue). Cada comando recibido se procesa enviándolo al robot mediante la función execute_command(...), y el estado del proceso (conexión, ejecución o errores) se informa mediante mensajes registrados en el panel de estado.



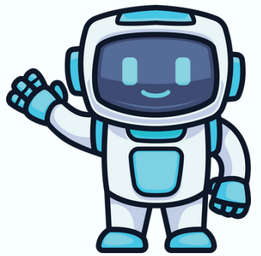
Gestión de conexión y envío de comandos por Bluetooth (BLEWorker._runner)

```python
async def _runner(self):
    try:
        self.log("Buscando hub Bluetooth...")
        device = await find_device()
        if not device:
            self.log("No se encontró hub.")
            return

        self.hub = PybricksHubBLE(device)
        await self.hub.connect()
        self.log("Conectado al hub. Listo para mover.")
        self.running.set()

        while True:
            drive_cmd = await self.queue.get()
            await execute_command(self.hub, drive_cmd, self.avanzar_activo, self.log)

    except asyncio.CancelledError:
        pass
    except Exception as e:
        self.log(f'Error en worker: {e}')
    finally:
        if self.hub:
            try:
                await self.hub.disconnect()
                self.log("Hub desconectado.")
            except Exception as e:
                self.log(f'Error al desconectar: {e}')
        self.running.clear()
```



Interfaz gráfica: controles de movimiento (botones)

```
frame_move = ttk.LabelFrame(root, text="Movimiento (A y C)")
frame_move.pack(fill="both", padx=10, pady=10)

ttk.Button(frame_move, text="⬅️ Izquierda", command=lambda: worker.send_command("izquierda")).grid(row=1, column=0, pady=5)
ttk.Button(frame_move, text="➡️ Derecha", command=lambda: worker.send_command("derecha")).grid(row=1, column=2, pady=5)
ttk.Button(frame_move, text="Empujar bloque", command=lambda: worker.send_command("empujar")).grid(row=2, column=1, pady=5)
ttk.Button(frame_move, text="Tirar bloque", command=lambda: worker.send_command("tirar")).grid(row=3, column=1, pady=5)
```

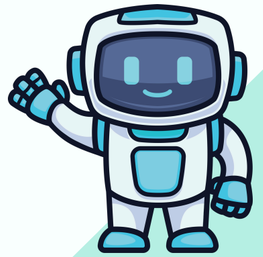


ESTADO ACTUAL DEL PROYECTO

Actualmente, el robot cumple con la función principal de identificar y clasificar bloques LEGO por color. La interfaz gráfica (GUI) permite el control manual del robot y la supervisión del estado de conexión en tiempo real, asegurando una interacción fluida entre el usuario y el sistema.

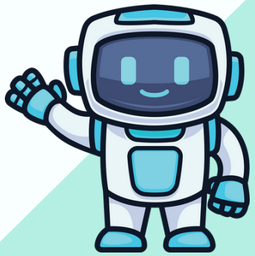
La detección de color funciona de correctamente, presentando algunas variaciones bajo diferentes condiciones de luz, por lo que sigue en proceso de calibración. Las funcionalidades autónomas están parcialmente implementadas, permitiendo ciclos básicos de clasificación automática, pero requieren optimización para aumentar la estabilidad y reducir tiempos de ejecución.

ID	Funciones	Estado	Resultado
RF-01	Posición inicial	Completo	El robot vuelve a la posición inicial al momento de tirar un bloque.
RF-02	Movimiento del robot	Completo	El robot se mueve de manera eficiente a las posición de cada cuadrado para separar los bloques de color.(amarillo, azul, verde, rojo y morado).
RF-03	Control vía Interfaz (GUI)	Completo	La conexión Bluetooth permite iniciar, pausar y mover el robot manualmente.
RF-04	Detección de color	Funcional	Aún tenemos problemas pero puede detectar algunos colores.



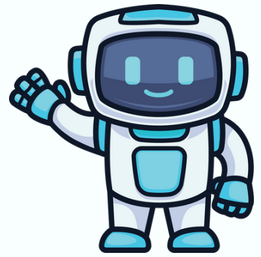
F O R M A T O

- Se corrigió la portada utilizando el formato oficial, ajustando el título y el espaciado de los nombres. Además, se optimizó la distribución del contenido, eliminando separaciones innecesarias entre títulos y texto, y reduciendo espacios excesivos en tablas y secciones para mejorar la presentación general del documento.



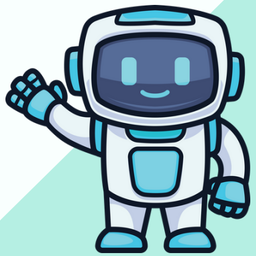
Formato, Redacción y Referencias

- Portada institucional correcta.
- Orden de títulos y tablas.
- Referencias citadas y coherentes con el contenido técnico.



C O N C L U S I Ó N

Como conclusión, aprendimos a integrar hardware y software en una arquitectura cliente-servidor, también nos dimos cuenta que durante el desarrollo del informe cometimos algunos errores que pasamos por alto, principalmente errores de espacios “exagerados”, ser un poco superficiales en algunas explicaciones, no haber distinguido bien entre servidor fisico y logico.



MUCHAS
GRACIAS