UNIVERSIDAD DE TARAPACÁ
Universidad del Estado

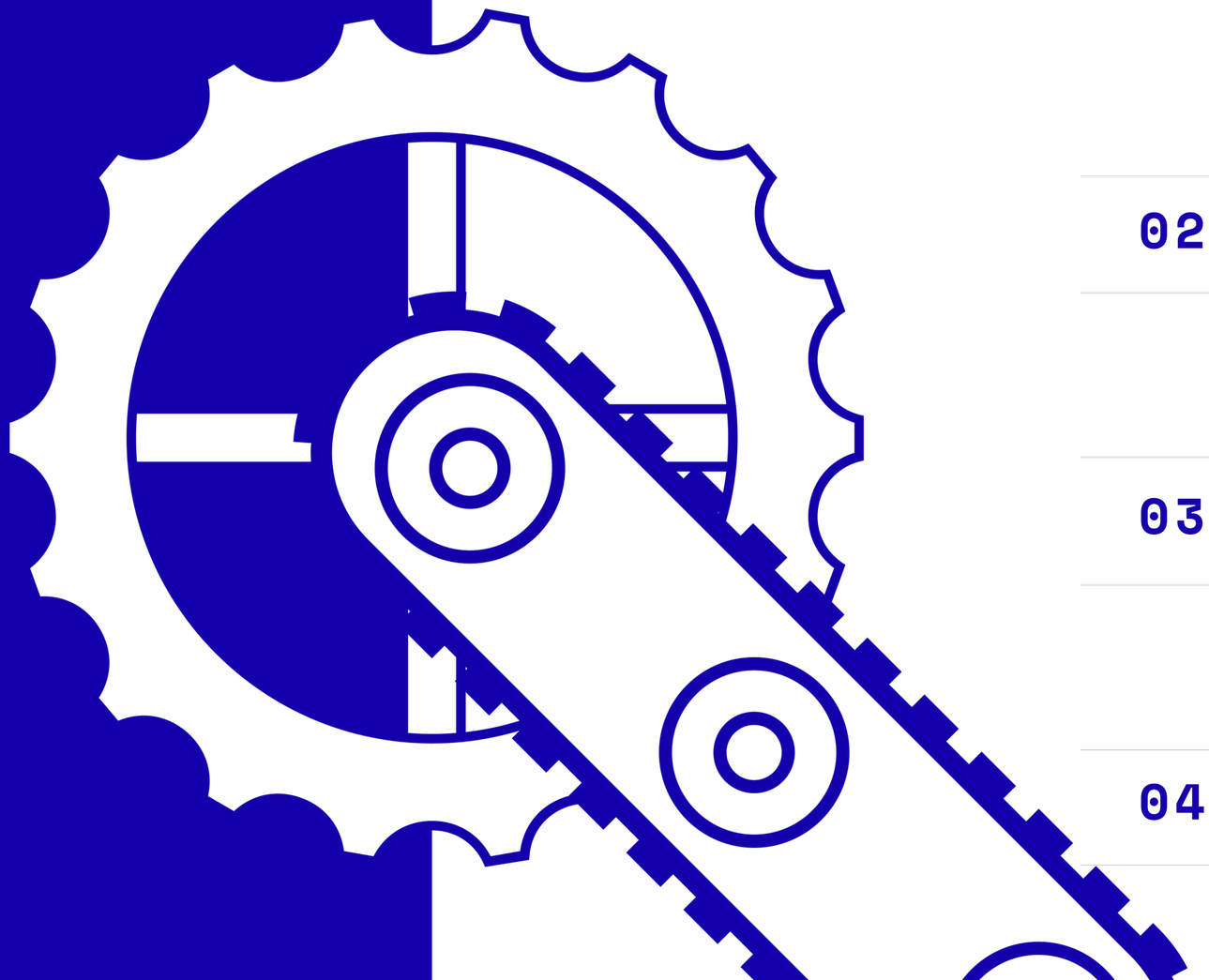
Ingeniería
Computación e Informática

PROTOTIPO DE UN VEHÍCULO ROBOT MINERO PARA EL TRASLADO DE MINERALES A GRAN ESCALA: FASE 2

Integrantes: Ayleen Humire, Brandon Quipe, German Castro, Claudio Pinazo y Daniela Poma

Profesor: Baris Klobertanz

Asignatura: Proyecto I



Contenidos

Introducción

La minería subterránea es una de las industrias con mayores riesgos críticos para los trabajadores, especialmente en las zonas de carga y transporte. Este proyecto nace de la necesidad de aplicar los conceptos de la Minería 4.0, utilizando un prototipo basado en LEGO Spike Prime y programación en Python para crear un sistema de transporte que pueda ser operado a distancia, eliminando así la exposición humana en entornos peligrosos.

ANÁLISIS Y DISEÑO

Especificación de requerimientos

REQUERIMIENTOS FUNCIONALES

Movilidad: El robot debe poder moverse a las 4 direcciones: avanzar, retroceder, girar a la izquierda y a la derecha, además de tener un sistema de velocidad que determinará la fuerza con la que se moverá.

Control: El manejo del robot debe ser mediante una interfaz gráfica accesible para el usuario mediante un archivo.exe.

Envío de información: El robot debe recibir órdenes enviadas a través de la interfaz gráfica con una respuesta rápida y sin demora.

Seguridad: Ante cualquier fallo del robot, por seguridad, debe presentar un botón que permita el apagado y absoluta detención del robot.

Especificación de requerimientos

REQUERIMIENTOS NO FUNCIONALES

Disponibilidad: Se debe garantizar un funcionamiento correcto y continuo durante las demostraciones del robot, comprobando que se puede mantener en constante uso durante horas sin problemas de batería o interrupciones para el usuario.

Robustez: Se debe garantizar que el sistema pueda manejar correctamente posibles fallos o acciones no válidas, tanto en software como en hardware, sin afectar considerablemente el rendimiento.

Rendimiento: Se debe garantizar que los comandos entregados por el usuario sean recibidos en menos de 1 segundo, logrando una interacción fluida y rápida.

Usabilidad: Se debe garantizar que la interfaz gráfica sea intuitiva y fácil de usar, permitiendo que usuarios con poca o nula experiencia técnica puedan controlar el robot sin complicaciones.

Compatibilidad: Se debe garantizar que el sistema sea ampliable y compatible para integrar otros robots o módulos sin afectar el rendimiento general.

Arquitectura de software Cliente-Servidor

Bluetooth/USB

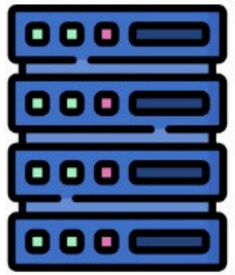
Socket stream sobre Bluetooth o USB.

Cliente

Captura las instrucciones del usuario y las envía al servidor

Servidor

Recibe, interpreta y ejecuta las instrucciones de movimiento.



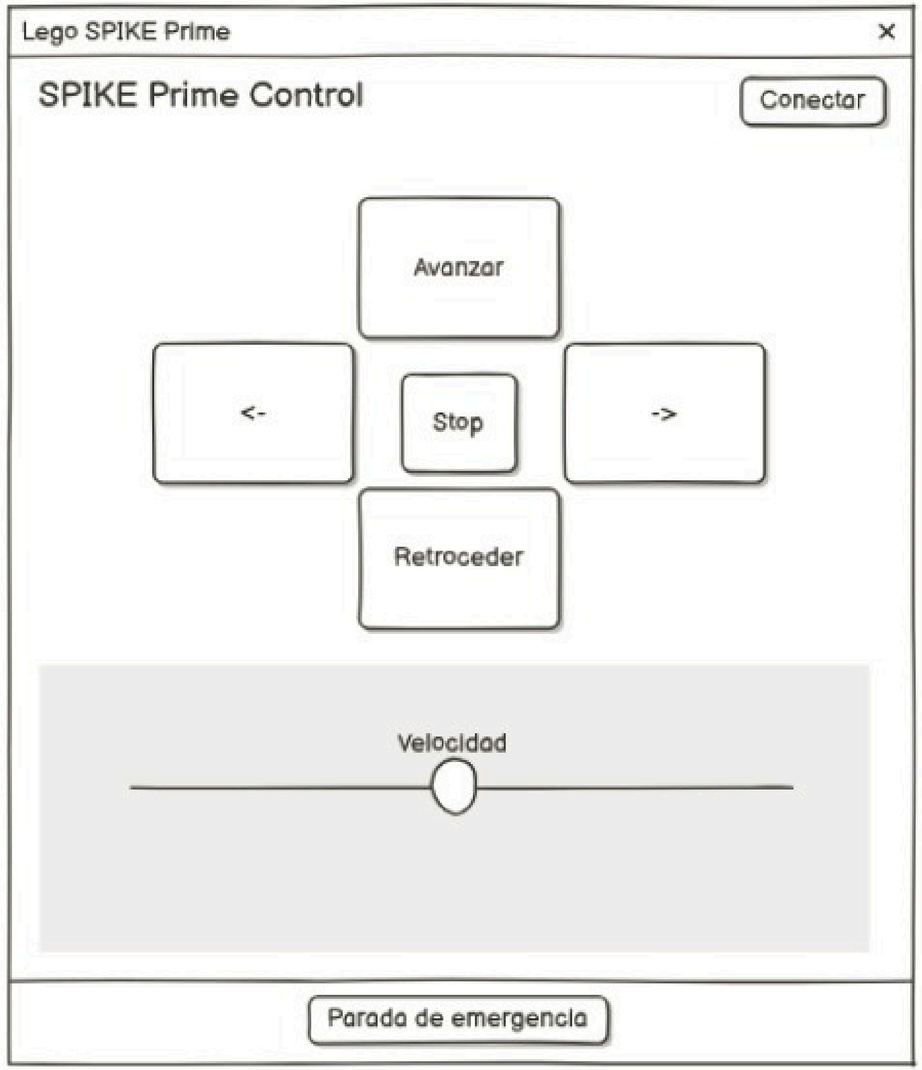
Interfaz gráfica

Interfaz sencilla donde el usuario puede indicar las instrucciones al robot.

Robot (SP-7)

El HUB recibe las instrucciones del servidor y sigue las instrucciones para los motores

Diseño inicial interfaz gráfica



- Controles direccionales.
- Control de potencia Indicadores visuales.
- Diseño inicial realizado en Balsamiq.

IMPLEMENTACIÓN

Fundamentos de los movimientos

MODELO DE MOVIMIENTO

- Movimiento rectilíneo uniformemente acelerado (MRUA)

CONSIDERACIONES FÍSICAS

- Distancia determinada: 10 metros
- Aceleración generada por dos motores traseros
- Factores considerados:
  - Masa del sistema
  - Fuerzas de roce
  - Distribución del centro de masa

RESULTADO DEL ANÁLISIS

- Configuración coherente con principios de cinemática y dinámica.
- Aceleración óptima para cumplir la distancia requerida.
- Estabilidad y eficiencia mantenidas

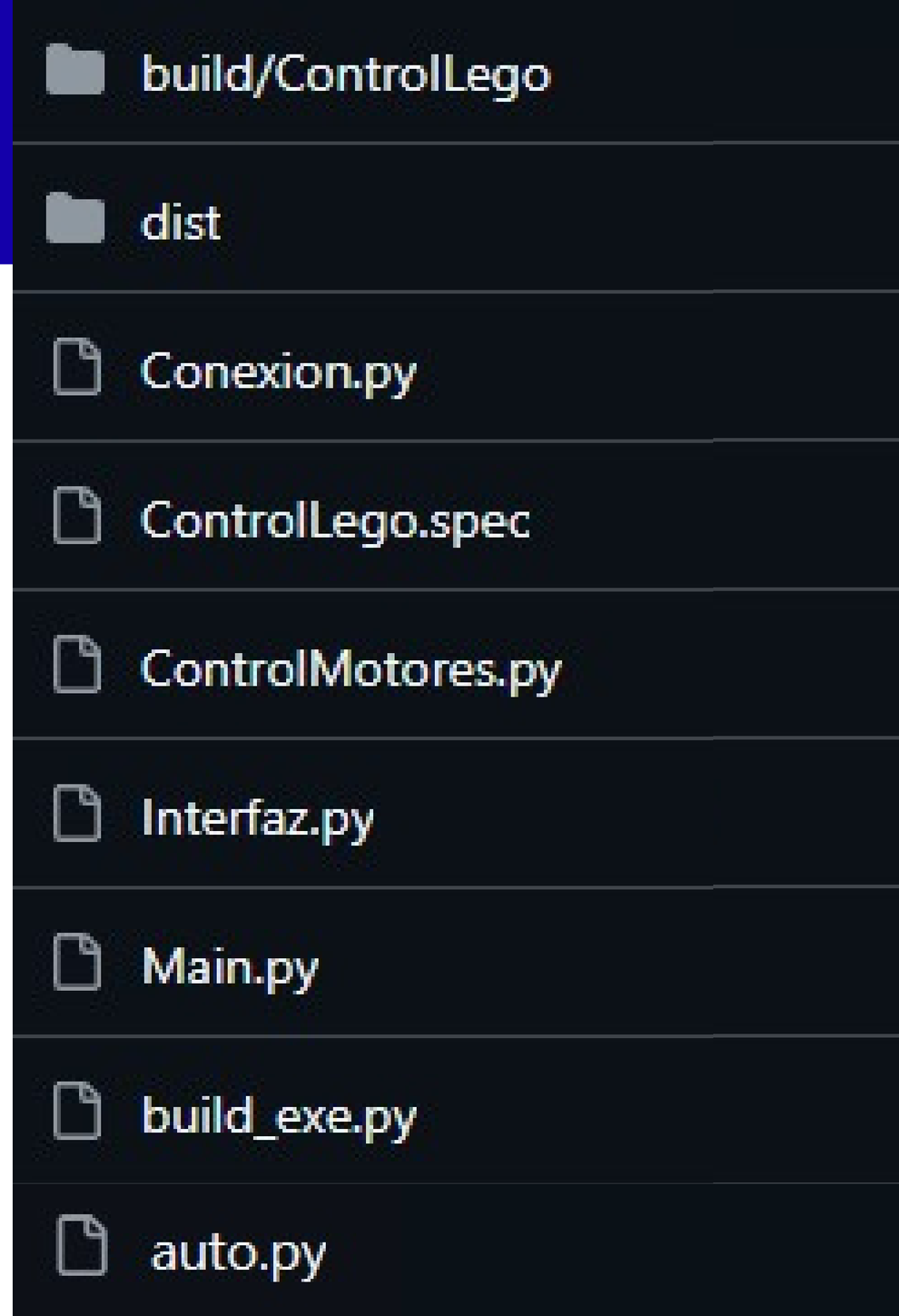
Descripción del Sistema

HARDWARE

- **LEGO Spike Prime HUB:** Actúa como el cerebro del robot, encargándose de ejecutar el programa desarrollado en MicroPython y de coordinar el funcionamiento de motores y sensores.
- **Motores:** Se utilizan motores para la tracción del vehículo, los cuales permiten controlar la velocidad y el sentido de giro de las ruedas.
- **Estructura mecánica:** Se utilizaron piezas de LEGO del propio set de Spike Prime, simulando un vehículo transportador minero.

SOFTWARE

Se expondrá el código principal del vehículo, el cual se encuentra almacenado en un repositorio de GitHub. Este código es el responsable del control de los motores y de la comunicación de forma inalámbrica al HUB, mandando las instrucciones del usuario utilizando una interfaz gráfica.



Interfaz

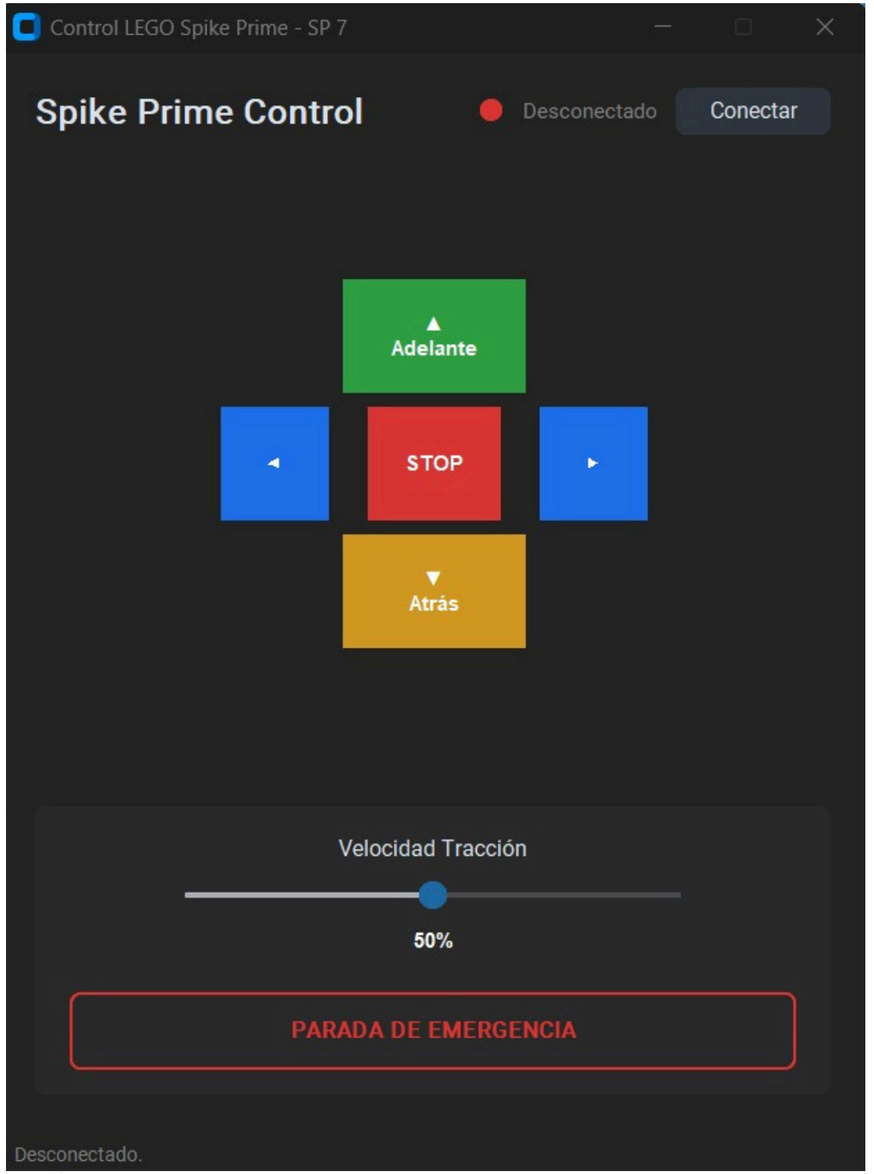
Interfaz.py

```python
import tkinter as tk
import customtkinter as ctk
from queue import Queue, Empty
from Conexion import BLEWorker

You, 53 minutes ago | 1 author (You)
class LegoGUI(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("Control LEGO Spike Prime - SP 7")
        self.geometry("500x650")
        self.resizable(False, False)

        self.log_queue = Queue()
        self.worker = BLEWorker(self.log_queue)
        self.emergency = False

        # --- RASTREADOR DE ESTADO PARA EVITAR DELAY ---
        # Esto guarda si una tecla ya está siendo presionada
        self.pressed_keys = {"w": False, "a": False, "s": False, "d": False}

        # Colores
        self.color_green = "#2EA043"
        self.color_red = "#DA3633"
        self.color_blue = "#1F6FEB"
        self.color_gray = "#30363D"
        self.color_yellow = "#D29922"

        self._build_ui()

        # Habilitar control por teclado (W A S D)
        self.bind_all("<KeyPress>", self._on_key_press)
        self.bind_all("<KeyRelease>", self._on_key_release)
```

```python
def _build_ui(self):
    # ADELANTE
    self.btn_up = tk.Button(dpad_container, text="▲\nAdelante", bg=self.color_green,
                            activebackground="#268c3b", width=14, height=4, **btn_style)
    self.btn_up.grid(row=0, column=1, pady=10)
    self.btn_up.bind("<ButtonPress-1>", lambda e: self.cmd_move("F"))
    self.btn_up.bind("<ButtonRelease-1>", lambda e: self.cmd_stop_traction())

    # IZQUIERDA
    self.btn_left = tk.Button(dpad_container, text="◀", bg=self.color_blue,
                              activebackground="#1a5cbf", width=8, height=4, **btn_style)
    self.btn_left.grid(row=1, column=0, padx=10)
    self.btn_left.bind("<ButtonPress-1>", lambda e: self.cmd_steer("L"))
    self.btn_left.bind("<ButtonRelease-1>", lambda e: self.cmd_steer("Z"))

    # STOP
    self.btn_stop = tk.Button(dpad_container, text="STOP", bg=self.color_red,
                              activebackground="#b02a28", width=10, height=4, **btn_style)
    self.btn_stop.grid(row=1, column=1, padx=5)
    self.btn_stop.config(command=self.cmd_emergency_stop)

    # DERECHA
    self.btn_right = tk.Button(dpad_container, text="▶", bg=self.color_blue,
                               activebackground="#1a5cbf", width=8, height=4, **btn_style)
    self.btn_right.grid(row=1, column=2, padx=10)
    self.btn_right.bind("<ButtonPress-1>", lambda e: self.cmd_steer("R"))
    self.btn_right.bind("<ButtonRelease-1>", lambda e: self.cmd_steer("Z"))

    # ATRAS
    self.btn_down = tk.Button(dpad_container, text="▼\nAtrás", bg=self.color_yellow,
                              activebackground="#71751f", width=14, height=4, **btn_style)
    self.btn_down.grid(row=2, column=1, pady=10)
    self.btn_down.bind("<ButtonPress-1>", lambda e: self.cmd_move("B"))
    self.btn_down.bind("<ButtonRelease-1>", lambda e: self.cmd_stop_traction())
```

Interfaz

Interfaz.py



Conexión interfaz-Spike

Conexion.py

```python
class BLEWorker:
    def __init__(self, log_queue: Queue):
        self.loop = asyncio.new_event_loop()
        self.thread = threading.Thread(target=self._thread_main, daemon=True)
        self.queue = asyncio.Queue()
        self.hub = None
        self.running = threading.Event()
        self.log_queue = log_queue

    def log(self, msg: str):
        if self.log_queue:
            self.log_queue.put(msg)

    def _thread_main(self):
        asyncio.set_event_loop(self.loop)
        self.loop.create_task(self._runner())
        self.loop.run_forever()

    async def _runner(self):
        temp_path = None
        try:
            self.log("Buscando hub 'SP-7'...")
            device = await find_device("SP-7")
            if not device:
                self.log("No se encontró hub.")
                return

            self.hub = PybricksHubBLE(device)
            await self.hub.connect()
            self.log("Conectado. Cargando script...")

            with tempfile.NamedTemporaryFile(mode='w', suffix='.py', delete=False, encoding='utf-8') as tf:
```

```python
async def _runner(self):
        with tempfile.NamedTemporaryFile(mode='w', suffix='.py', delete=False, encoding='utf-8') as tf:
            tf.write(LISTENER_SCRIPT)
            tf.flush()
            temp_path = tf.name

        # Log the temp script path and check existence to help diagnose WinError 2
        self.log(f"Temp script path: {temp_path} (exists: {os.path.exists(temp_path)})")

        await self.hub.run(temp_path, wait=False, print_output=True)
        await asyncio.sleep(2)

        self.running.set()
        self.log("¡Listo para conducir!")

        while True:
            cmd_raw = await self.queue.get()

            if self.hub and self.running.is_set():
                packet = f"{cmd_raw};"
                payload = packet.encode('utf-8')

                try:
                    await self.hub.write(payload)
                except Exception as e:
                    self.log(f"Error TX: {e}")

    except asyncio.CancelledError:
        pass
    except Exception as e:
        # Log full traceback to help identify where the error originated (WinError 2: file not found)
        tb = traceback.format_exc()
        self.log(f"Error fatal: {e} ({type(e).__name__})\n{tb}")
    finally:
        if temp_path:
            try: os.unlink(temp_path)
```

Control Motores

ControlMotores.py

```
LISTENER_SCRIPT = """
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor
from pybricks.parameters import Port, Color
from pybricks.tools import wait
import usys as sys
import uselect

# -- CONFIGURACIÓN --
hub = PrimeHub()
hub.light.on(Color.ORANGE)
      You, last week • creacion del .exe
motorA = None
motor_izq = None
motor_dir = None # Motor de dirección (Puerto C)

# Inicialización de motores de tracción
try:
    motorA = Motor(Port.A)  # Rueda derecha
except Exception: pass

try:
    motor_izq = Motor(Port.E)  # Rueda izquierda
except Exception: pass

# Inicialización del motor de dirección
try:
    motor_dir = Motor(Port.C)
    motor_dir.reset_angle(0)
except Exception: pass

# -- LOOP TIPO SOCKET --
hub.light.on(Color.GREEN) # Verde = LISTO
```

```
if len(cmd) > 0:
    action = cmd[0] # F, B, S, L, R, Z

    # --- LÓGICA DE TRACCIÓN (F=Forward, B=Back, S=Stop) ---
    if action == 'S':
        if motorA: motorA.stop()
        if motor_izq: motor_izq.stop()
        hub.light.on(Color.GREEN)

    elif action == 'F' or action == 'B':
        try:
            val_part = cmd[1:]
            if val_part == '': val_part = '0'
            speed = int(val_part)

            if action == 'B':
                speed = -speed

            if motorA: motorA.run(speed)
            if motor_izq: motor_izq.run(-speed)
            hub.light.on(Color.BLUE)
        except ValueError:
            pass

    # --- LÓGICA DE DIRECCIÓN (Puerto C) ---
    elif action in ['L', 'R', 'Z']:
        if motor_dir:
            try:
                if action == 'L':
                    motor_dir.run_target(800, -30, wait=False)
                elif action == 'R':
                    motor_dir.run_target(800, 30, wait=False)
                elif action == 'Z':
                    motor_dir.run_target(800, 0, wait=False)
            except Exception:
                pass
```

Código Spike

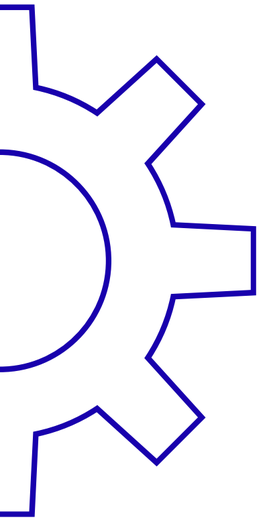
auto.py

```python
from pybricks.hubs import PrimeHub
from pybricks.pupdevices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait
import sys

# ----------------------------
# CONFIGURACIÓN DE MOTORES
# ----------------------------
hub = PrimeHub()

motorA = Motor(Port.A)   # Rueda derecha
motor_izq = Motor(Port.E)   # Rueda izquierda
motor_dir = Motor(Port.C)   # Dirección

# Variables iniciales
velocidad = 0
giro = 0

def aplicar_giro():
    """ Mueve el motor de dirección según el slider (-100 a 100). """
    motor_dir.run_target(300, giro)

def mover_adelante():
    motorA.dc(velocidad)
    motor_izq.dc(velocidad)

def mover_atras():
    motorA.dc(-velocidad)
    motor_izq.dc(-velocidad)

def detener():
    motorA.stop()
    motor_izq.stop()      You, last month • progreso real del codigo para poder
```

```python
# ENVÍA "rdy" PARA INDICAR QUE ESTÁ LISTO
# ----------------------------------------
def enviar_ready():
    # Añade un salto de línea para que el receptor basado en GATT pueda
    # recibir la notificación por líneas.
    sys.stdout.write("\x01rdy\n")
    sys.stdout.flush()

enviar_ready()

# ----------------------------------------
# BUCLE PRINCIPAL DE COMANDOS
# ----------------------------------------
while True:

    # Leer por línea en lugar de `read()` que bloquea hasta EOF.
    # Esto permite recibir comandos que el cliente BLE envía terminados
    # con un '\n'.
    cmd = sys.stdin.readline()

    if not cmd:
        wait(10)
        continue
    cmd = cmd.strip()

    # Confirma recepción al PC
    enviar_ready()
```

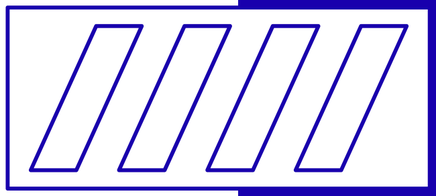
RESULTADOS

Estado actual del proyecto



Socket stream demostró ser altamente efectivo en las pruebas de conectividad. Se logró establecer un enlace estable y continuo entre la interfaz de usuario y el Hub del LEGO Spike Prime. El sistema gestiona correctamente la transmisión de múltiples instrucciones secuenciales y simultáneas, permitiendo un control fluido del vehículo.

Problemas encontrados y solucionados



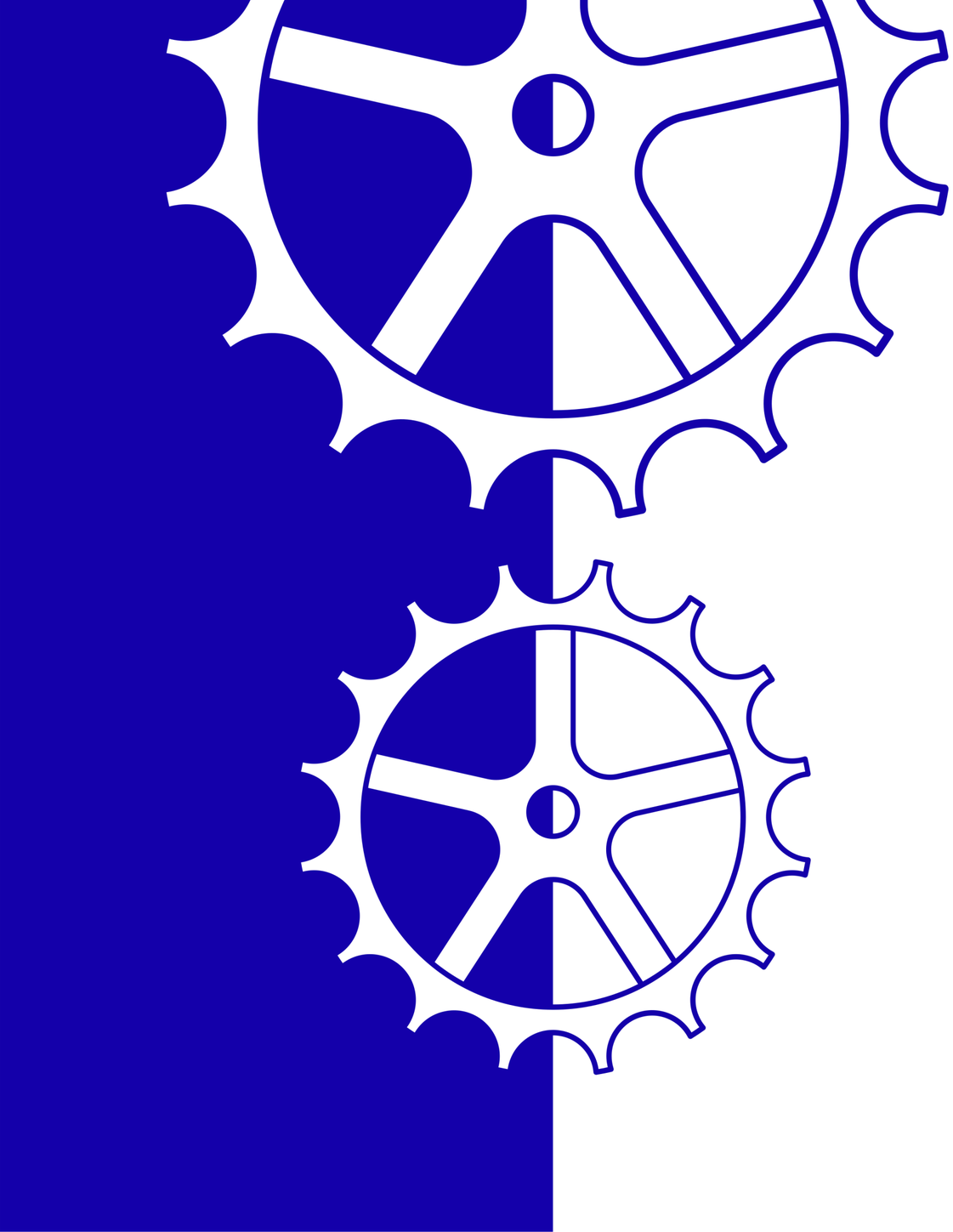
Incompatibilidad inicial entre los sensores/motores y la programación en Micro Python

Horario insuficiente para el trabajo colaborativo del equipo

Dificultad para codificar funciones avanzadas del robot

CONCLUSIONES

Si bien aún queda trabajo por realizar, este proceso ha representado una oportunidad para aplicar tanto habilidades blandas —como el trabajo en equipo, la definición de roles, la comunicación efectiva y la organización interna— como habilidades duras, entre ellas la programación en Python orientada a dispositivos físicos, el uso de herramientas como Git y GitHub, y la integración de librerías y funciones avanzadas en Visual Studio Code.



GRACIAS