Modelo escala Lego
Vehículo Minero

Integrantes:

Cristofer Lazaro

Francisca Albornoz

Brayan Cahuachia

Ruth Huanca

Abraham Canaviri

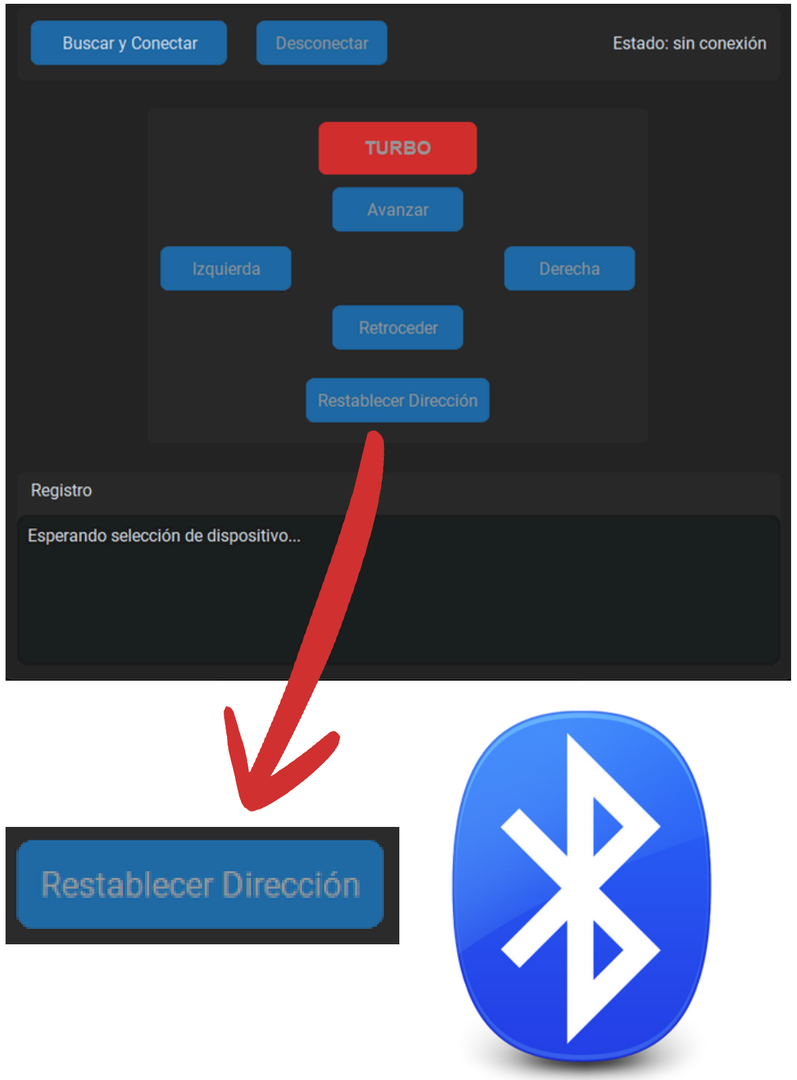
Asignatura: Proyecto I

Profesor: Baris Klobertanz

ÍNDICE

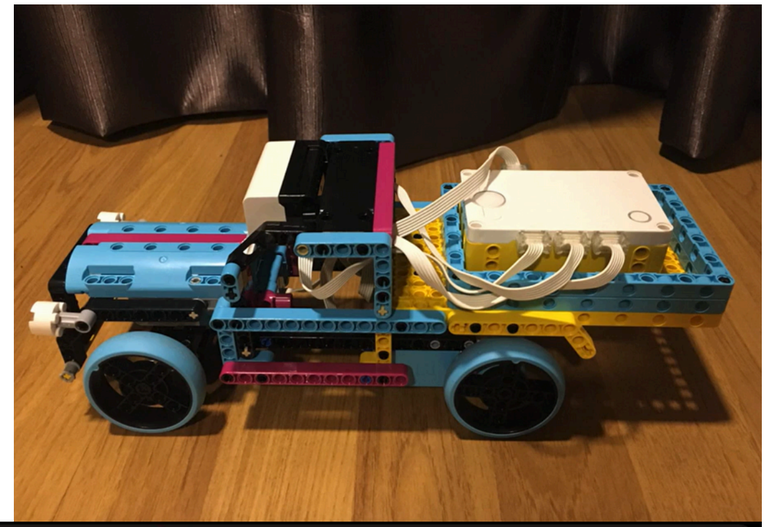
REQUERIMIENTOS FUNCIONALES

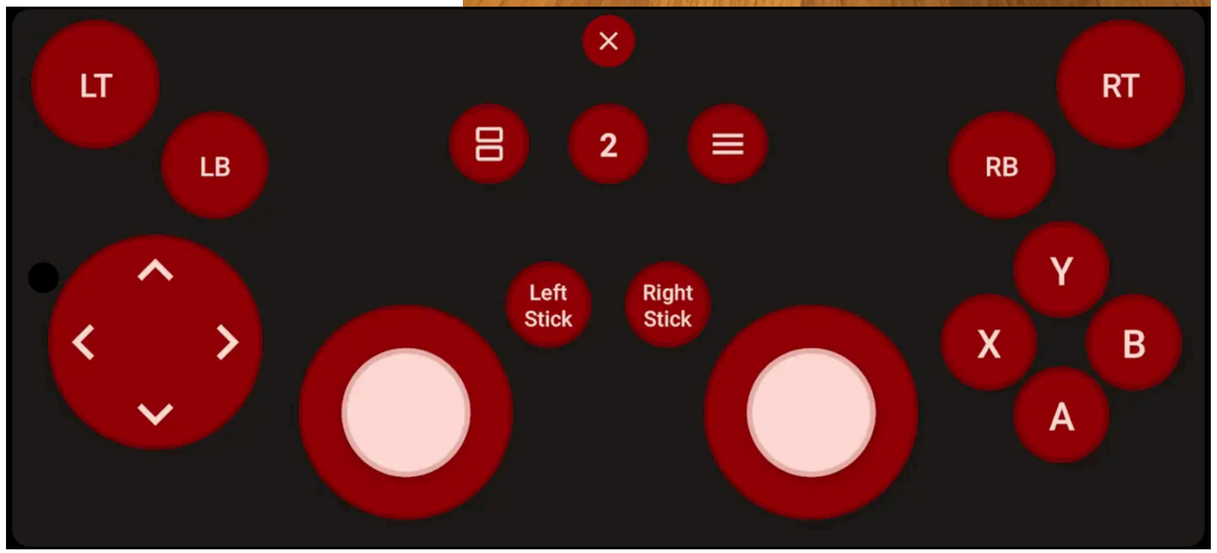
- Control
- Gestión de conexión inalámbrica
- Interfaz de usuario (GUI)
- Restablecer trayectoria



REQUERIMIENTOS NO FUNCIONALES

- Diseño Intuitivo
- Tiempo
- Rendimiento

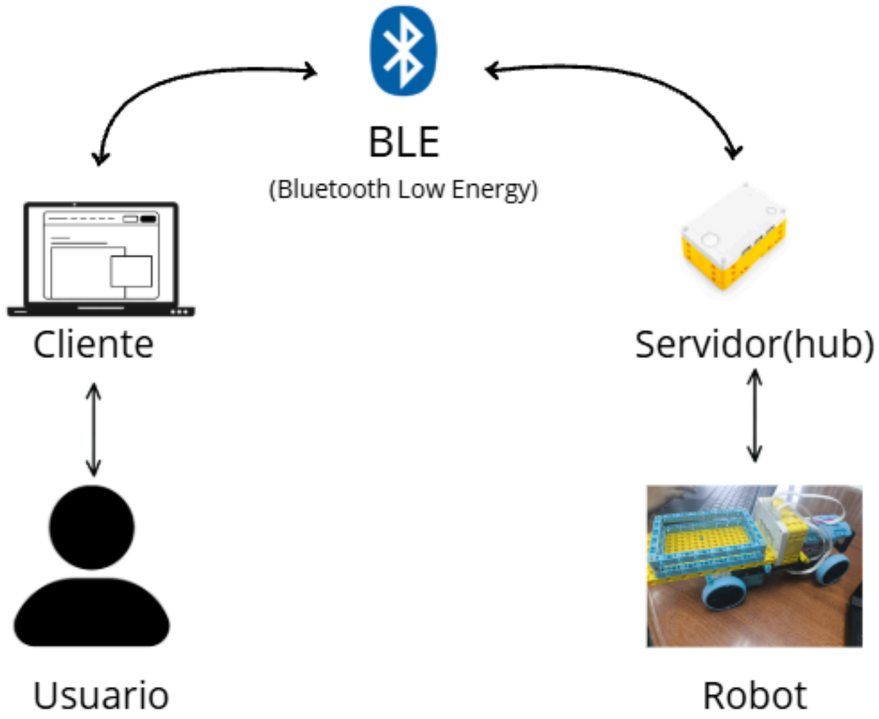




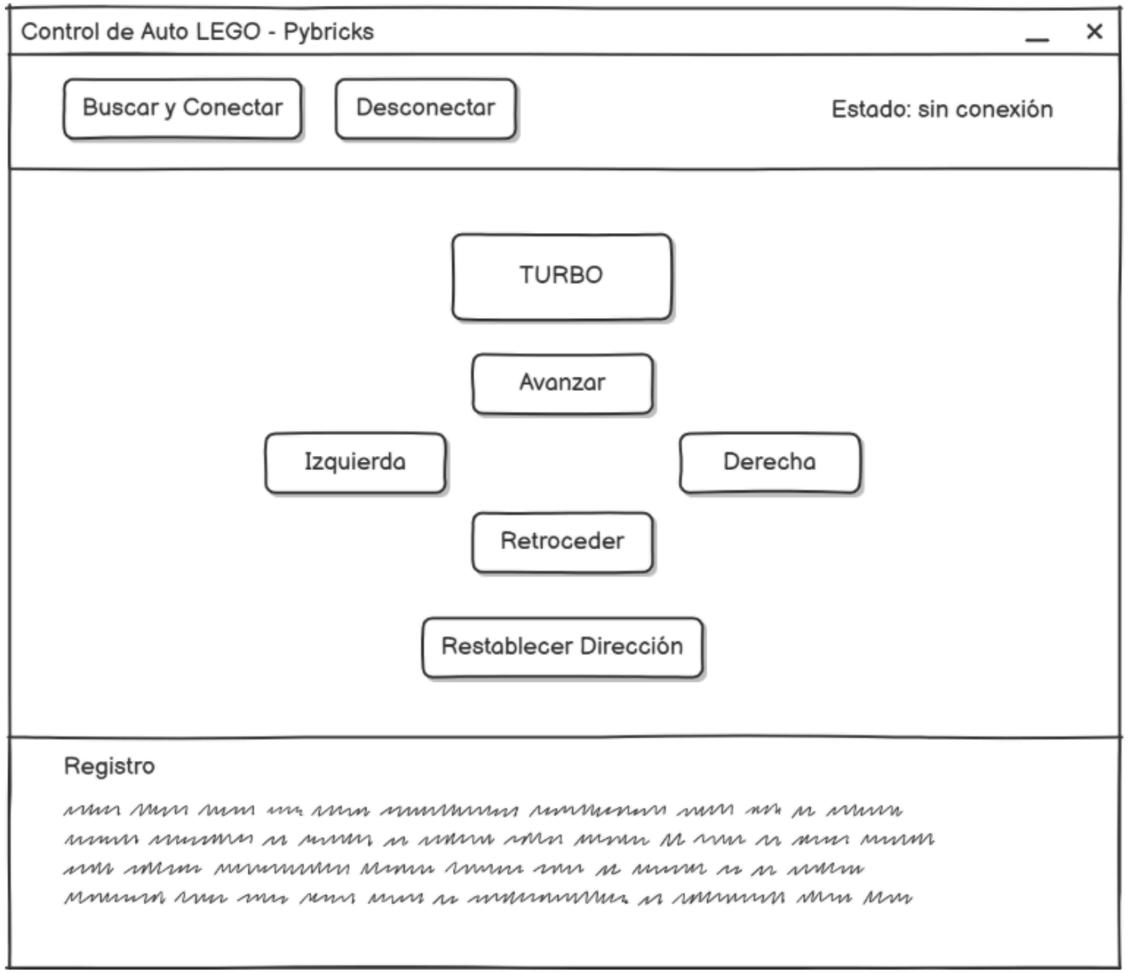
ARQUITECTURA DE SOFTWARE

Se ha diseñado una arquitectura cliente-servidor.

- Cliente: Desarrollado en Python, se encarga de capturar los eventos que el usuario quiera hacer.
- Comunicación (Bluetooth): Es el medio de comunicación entre el cliente y el servidor
- Servidor (Hub): Se mantiene a la espera de los scripts enviados por el por el usuario a través del cliente.



WIREFRAME GUI



IMPLEMENTACIÓN

Fundamentos de movimientos

Cliente-Servidor

Interfaz gráfica de usuario

FUNDAMENTOS DE MOVIMIENTO

FI 035

"Determinar la aceleración media necesaria para que el vehículo recorra una distancia de 1 metro en el menor tiempo posible."

Meta: 2.0 s
Modelo ideal

Distancia a recorrer (d): 1 m.
Tiempo objetivo (t): 2.0 s.
Velocidad inicial (v0): 0 m/s (el robot parte del reposo).
Incógnita: Aceleración media (a).

$$d = v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2$$

$$d = \frac{1}{2} \cdot a \cdot t^2$$

$$a = \frac{2d}{t^2}$$

$$a = \frac{2 \cdot 1m}{(2s)^2} = 0.5\ m/s^2$$

IMPLEMENTACIÓN CLIENTE

Control del robot

- clase _update_list

```
def _update_list(self, devices):
    self.label.configure(text="Selecciona tu Hub:")
    self.btn_refresh.configure(state="normal")

    found_any = False
    for dev in devices:
        name = dev.name if dev.name else "Desconocido"
        address = dev.address

        if name != "Desconocido":
            found_any = True
            btn = ctk.CTkButton(
                self.scroll_frame,
                text=f"{name}\n({address})",
                command=lambda d=dev: self._on_device_click(d),
                height=50,
                anchor="w"
            )
            btn.pack(pady=5, padx=5, fill="x")

    if not found_any:
        lbl = ctk.CTkLabel(self.scroll_frame, text="No se encontraron dispositivos con nombre.")
        lbl.pack(pady=20)

def _on_device_click(self, device):
    self.on_select_callback(device)
    self.destroy()
```

- Funcionamiento de _runner

```
async def _runner(self):
    try:
        while True:
            self.log("Esperando selección de dispositivo...")
            await self._connect_request.wait()

            device = self._target_device
            if not device:
                self._connect_request.clear()
                continue

            try:
                self.log(f"Conectando a {device.name}...")
                self.hub = PybricksHubBLE(device)
                await self.hub.connect()
                self.log("Conectado. Listo para conducir.")
                self.running.set()

                while self.running.is_set():
                    drive_cmd = await self.queue.get()
                    await execute_command(self.hub, drive_cmd, self.log)

            except Exception as e:
                self.log(f"Error de conexión/ejecución: {e}")
            finally:
                if self.hub:
                    try:
                        await self.hub.disconnect()
                        self.log("Hub desconectado.")
                    except:
                        pass
                self.hub = None
                self.running.clear()
                self._connect_request.clear()
                self._target_device = None

    except asyncio.CancelledError:
```

IMPLEMENTACIÓN SERVIDOR

- Perfiles de movimiento

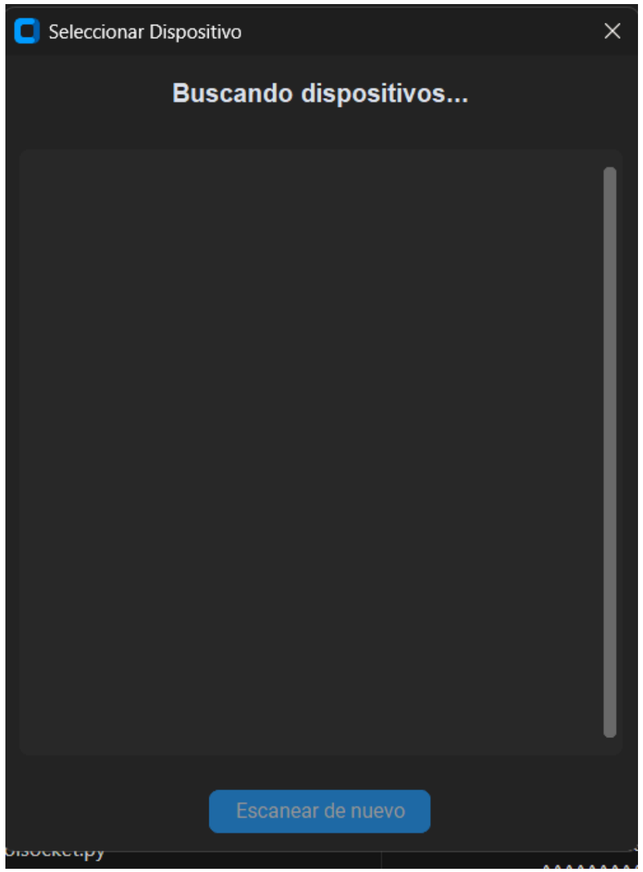
```python
def create_program(drive_cmd: str) -> str:
    drive_commands = {
        'run_forward': "motorB.run(-800)\nmotorF.run(800)",
        'run_turbo':   "motorB.run(-1100)\nmotorF.run(1100)",
        'run_backward': "motorB.run(500)\nmotorF.run(-500)",
        'stop': "motorB.stop()\nmotorF.stop()\nmotorD.stop()",
        'izquierda': "motorD.run_target(300, -35)\nmotorD.stop()",
        'derecha': "motorD.run_target(300, 35)\nmotorD.stop()",
        'centro': "motorD.run_target(300, 0)\nmotorD.stop()",
    }
```

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

hub = PrimeHub()
motorB = Motor(Port.B)
motorF = Motor(Port.F)
motorD = Motor(Port.D)

{drive_code}
{wait_code}
motorB.stop()
motorF.stop()
"""
    return program
```

INTERFAZ GRÁFICA DE USUARIO



ESTADO ACTUAL

Asociados a Requerimientos Funcionales.

- Diseño del modelo
- Control
- Gestión de la conexión
- Interfaz gráfica
- Restablecimiento de dirección

PROBLEMAS ENCONTRADOS Y SOLUCIONADOS

- Envío de comandos en vacío
- Conexión Automática "Ciega"

CONCLUSIÓN

Puntos corregidos:

- Requerimientos funcionales y no funcionales
- Implementación servidor
- Estado actual
- Problemas y soluciones

CONCLUSIÓN

Avance en el desarrollo del control, la conexión inalámbrica, la GUI y el restablecimiento de dirección.
Influencia de la fecha límite en las decisiones.
Se realizan pruebas de respuesta del robot a través del GUI.

GRACIAS