# UNIVERSIDAD DE TARAPACÁ

## FACULTAD DE INGENIERÍA

## DEPARTAMENTO DE INGENIERÍA CIVIL EN COMPUTACIÓN E INFORMÁTICA

# Plan de Proyecto Fase 3
# “Prototipo de un vehículo robot minero para el traslado de minerales a gran escala”


**Alumnos:** Ayleen Humire
Brandon Quispe
German Castro
Claudio Pinazo
Daniela Poma


**Asignatura:** Proyecto I

**Profesor:** Baris Klobertanz

**Diciembre – 2025**
**ARICA - CHILE**

## Historial de Cambios

| Fecha | Versión | Descripción | Autores |
|---|---|---|---|
| 09/10/2025 | 1.0 | Concepción del documento. | **Ayleen Humire, Daniela Poma** |
| 10/10/2025 | 1.1 | Recopilación de información . | **Ayleen Humire, Daniela Poma, Claudio Pinazo** |
| 16/10/2025 | 1.2 | Diseño principal del documento hecho. | **Ayleen Humire, Daniela Poma** |
| 17/10/2025 | 2.0 | Revisión y finalización del primer informe. | **Todo el grupo** |
| 28/11/2025 | 2.1 | Corrección de informe. | **Todo el grupo** |
| 08/12/2025 | 2.2 | Inicio de informe 2. | **Todo el grupo** |
| 15/12/2025 | 3.0 | Ajustes y revisión informe 2. | **Todo el grupo** |
| 24/12/2025 | 3.1 | Corrección informe 2. | **Todo el grupo** |
| 29/12/2025 | 3.2 | Elaboración y ajuste de informe 3. | **Ayleen Humire, German Castro, Brandon Quispe** |

# Índice de Contenidos

# Índice de Tablas

# Índice de Figuras

# 1. Panorama General

## 1.1 Introducción

La minería 4.0 es la transición de la minería tradicional hacia la industria 4.0, un modelo caracterizado por la digitalización y la automatización de los procesos productivos. Con base en tecnologías como el internet de las Cosas (IoT), la inteligencia artificial (IA), el big data, la robótica y la realidad aumentada, las cuales mejoran la seguridad y eficiencia de las operaciones mineras. Aún así, se muestran altos niveles de riesgo para los trabajadores en la minería subterránea, principalmente en la etapa de carga, transporte y trabajo del material. Debido a la presencia de importantes accidentes ocurridos en la mina Grasberg (Indonesia) y en la mina El Teniente (Chile), se necesita reducir la exposición humana a zonas de gran peligro mediante soluciones tecnológicas.

Teniendo en cuenta el contexto anteriormente explicado, el presente proyecto propone el desarrollo de una maqueta robótica que pueda simular un proceso mineral realista, en este caso la maqueta robótica será el de un vehículo transportador, con el fin de explorar su uso como alternativa segura para realizar diversas tareas. Para lograrlo, se utilizará el kit lego Spike Prime, para desarrollar un robot que podrá ser movilizado y controlado a través de una interfaz gráfica de usuario (GUI) implementada en el lenguaje de programación Micro Python.

En este informe se presenta la formulación inicial del proyecto, detallando la justificación del problema, la definición de objetivos, la asignación de responsabilidades de cada miembro, la estrategia empleada y las acciones que se tomaron para lograr los objetivos del proyecto. También se registrará la investigación pertinente que se llevará a cabo a lo largo del semestre.

## 1.2 Objetivos

### 1.2.1 Objetivo general

Diseñar y construir un robot utilizando el kit Lego Spike Prime que sea capaz de desplazarse, integrando principios de mecánica y programación, con el propósito de desarrollar competencias técnicas en robótica educativa y fomentar el trabajo colaborativo en equipo, además de en forma simulada poder ayudar en reducir los riesgos en las zonas de minería; beneficiando a los trabajadores.

### 1.2.2 Objetivos específicos

## Diseño del robot:

Diseñar y planificar la estructura del robot usando las piezas del kit Lego Spike Prime, definiendo el tipo de chasis, el sistema de movimiento con ruedas y la ubicación de cada componente, para establecer una base sólida que permita el desplazamiento autónomo del robot.

- Actividades:
    - Reunión inicial y definición del proyecto.
    - Investigación del kit Lego Spike Prime.
    - Asignación de roles del equipo.
    - Diseño conceptual del robot.

## Armado físico:

Ensamblar todas las partes del robot siguiendo el diseño establecido, integrando el chasis, las ruedas, el HUB, y los motores hasta obtener el robot completo y funcional.

- Actividades:
    - Construcción de la base del robot.
    - Instalación del chasis y ruedas.
    - Instalación del HUB, y motores.
    - Ensamblaje final.

## Programación:

Desarrollar el código de control del robot escribiendo las instrucciones para el movimiento y los giros, logrando que el robot se desplace.

- Actividades:
    - Configuración del HUB.
    - Programación de movimientos básicos y giros.
    - Desarrollo de navegación autónoma.
    - Creación de repositorio de código.
    - Optimización y limpieza del código.

### Pruebas y ajustes:

Evaluar el funcionamiento del robot en diferentes escenarios, identificando problemas tanto en el ensamblaje como en la programación y aplicando las correcciones necesarias, mientras se documenta el proceso completo para la presentación final del proyecto.

- Actividades:
    - Pruebas de movimiento y giros.
    - Identificación y corrección de errores.
    - Pruebas finales en pista de obstáculos.
    - Desarrollo de bitácoras semanales.
    - Elaboración y corrección de informes.
    - Preparación de presentación final.
    - Entrega de entregables.

## 1.3 Restricciones

- Se utilizará el Kit Lego Spike Prime para la creación del robot.
- Se cuenta con un tiempo límite de ejecución.
- Para conectarse con el hub del robot tenemos dos opciones: usar cable USB o Bluetooth. No se permiten otros métodos de conexión.
- Se debe de usar Redmine de forma obligatoria todas las semanas para gestionar el proyecto.
- Se debe crear la interfaz gráfica usando librerías que implementan un GUIs.
- Se debe implementar únicamente el lenguaje de programación Python.

## 1.4 Entregables

- Carta Gantt: Herramienta visual esencial para la administración de proyectos. Su función principal es planificar y mostrar el calendario de todas las actividades del proyecto durante un período específico.

- Bitácoras: Registro continuo donde anotamos todo lo que hacemos: qué investigamos (sitios web y referencias técnicas) y qué tareas realizaremos. Lo importante es incluir tanto lo que funcionó como lo que no.

- Robot: La versión y modelo final del robot hecho con piezas de LEGO Spike Prime.

- Informe: Documento donde el proyecto está especificado y justificado en base al Caso de estudio respectivamente estudiado. En él se adjuntará el desarrollo de los demás entregables (Carta Gantt, Bitácoras, y el Robot).

- Manual de Usuario: Documento técnico completo que guía al operador en la instalación, configuración, operación y mantenimiento del sistema de control del robot. Incluye procedimientos paso a paso, solución de problemas, advertencias de seguridad y especificaciones técnicas. Disponible como archivo independiente en el repositorio GitHub del proyecto.

(https://github.com/ChermanWest/Proyecto-1/blob/main/docs/MANUAL_DE_USUARIO.md).

## 2. Organización del Personal

El proyecto tiene diferentes áreas de trabajo (armado, programación, diseño y documentación), por lo que fue necesario dividir las tareas entre los integrantes del equipo de forma balanceada. Se decidió trabajar con roles específicos porque el proyecto necesita atención en varios frentes al mismo tiempo: la parte física del robot, el código, el aspecto visual y todo el papeleo administrativo. Si cada persona se enfoca en una cosa, el trabajo sale mejor y más rápido. También hace más fácil organizarse y revisar qué tanto se ha avanzado cada semana. Los roles del equipo son: Jefe de Proyecto, Ensamblador, Programador, Diseñador y Documentador.

### 2.1. Descripción de los roles

**Jefe de proyecto:** Representante del equipo, supervisa y organiza el progreso del proyecto, verifica que todo sea entregado en el tiempo acordado e interactúa con cada integrante del equipo para dar sugerencias o brindar soluciones para evitar retrasos.

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

**Programador:** Encargado del área de la codificación para los motores del robot, buscando la movilidad del robot mediante una interfaz funcional, colabora con el ensamblador.

**Documentador:** Encargado de registrar el avance del proyecto, el cual, busque dar a conocer cuáles fueron los avances y retrasos del proyecto durante las semanas, además de la redacción del informe a presentar.

**Diseñador:** Encargado de la creación del logotipo y diseñador de pruebas de campo que busquen poner a prueba las funcionalidades del vehículo.

## 2.2 Personal que cumplirá los roles

A lo largo del proyecto ha existido una rotación de roles habiendo un cambio de estos a partir de la semana de trabajo N°7.

**Tabla 1:** *Asignación de Roles del Proyecto.*

| Rol | Responsable |
|---|---|
| Jefe de proyecto | German Castro |
| Ensamblador | Claudio Pinazo |
| Programador | Brandon Quispe |
| Documentador | Ayleen Humire |
| Diseñador | Daniela Poma |

**Tabla 2:** *Asignación post cambio de roles.*

| Rol | Responsable |
|---|---|
| Jefe de proyecto | Ayleen Humire |
| Ensamblador | Daniela Poma |
| Programador | Claudio Pinazo |

| | |
|---|---|
| Documentador | German Castro |
| Diseñador | Brandon Quispe |

## 2.3. Métodos de Comunicación

Los principales medios de comunicación que se usarán son los siguientes: WhatsApp, que se utilizará para enviar mensajes, haciendo uso de los grupos de chat; Discord, que será usado para reuniones de llamada o videollamada (canales de texto y voz), y para la documentación de words se están usando los servicios de google para poder progresar en los documentos mientras todos tienen acceso a esto para poder editarlos.

# 3. Planificación del proyecto

## 3.1. Actividades

**Tabla 3:** *Actividades del proyecto y sus criterios de término.*

| Nombre | Descripción | Responsable | Producto |
|---|---|---|---|
| Reunión inicial y definición del proyecto. | Primera junta del equipo donde se conversa sobre qué se quiere lograr y cómo organizarse. | Todo el grupo. | Acta de reunión con objetivos definidos. |
| Investigación del kit Lego Spike Prime. | Revisar qué trae el kit, cómo funciona y qué se puede hacer con él. | Todo el grupo. | Conocimiento sobre la información técnica acerca del kit. |
| Asignación de roles del equipo. | Repartir las tareas según lo que cada uno sabe hacer mejor. | Todo el grupo. | Tabla con los roles de cada integrante. |
| Diseño conceptual del robot. | Hacer los primeros dibujos de cómo va a verse y funcionar el robot. | Todo el grupo. | Bocetos y esquemas del diseño. |
| Construcción de la base del robot. | Armar la parte de abajo que va a aguantar todo lo demás. | German Castro, Claudio Pinazo, Brandon Quispe. | Estructura base armada. |
| Instalación del chasis y ruedas. | Poner las ruedas y lo que permite que el robot se mueva. | Ayleen Humire, Claudio Pinazo. | Robot con ruedas funcionando. |

| | | | |
|---|---|---|---|
| Instalación del HUB, y motores. | Integración de los componentes electrónicos principales y los motores que lo mueven. | German Castro, Brandon Quispe. | Robot con componentes electrónicos y motores instalados. |
| Ensamblaje final. | Juntar todas las piezas y hacer los últimos ajustes. | Brandon Quispe, German Castro, | Robot completamente armado. |
| Configuración del HUB. | Preparar el HUB para que esté listo para recibir el código. | German Castro. | HUB configurado. |
| Programación de movimientos básicos y giros. | Escribir el código para que el robot avance y gire. | German Castro, Brandon Quispe. | Código con funciones básicas. |
| Optimización y limpieza del código. | Revisar el código para hacerlo más eficiente. | Brandon Quispe. | Código optimizado final. |
| Creación de repositorio de código. | Hacer un lugar donde guardar todas las versiones del código. | German Castro. | Repositorio en GitHub o similar. |
| Pruebas de movimiento y giros. | Probar que el robot se mueva y gire como debe. | Todo el grupo. | Registro de las pruebas hechas. |
| Identificación y corrección de errores. | Encontrar lo que no funciona bien y arreglarlo. | German Castro, Brandon Quispe. | Robot con sus problemas solucionados. |
| Pruebas finales en pista de obstáculos. | Probar el robot en la pista con obstáculos. | German Castro Brandon Quispe | Resultados de las pruebas en pista. |
| Desarrollo de bitácoras semanales. | Ir anotando cada semana qué se hizo y cómo fue el avance en el transcurso del tiempo. | Ayleen Humire, Daniela Poma, Claudio Pinazo | Bitácoras de cada semana. |
| Elaboración y corrección de informes. | Escribir los informes del proyecto y revisarlos | Todo el grupo. | Informes terminados |
| Preparación de presentación. | Armar y coordinar la presentación para mostrar todo el trabajo. | Todo el grupo. | Presentación en PowerPoint o PDF |
| Entrega de entregables | Juntar todo lo que hay que entregar y mandarlo | Todo el grupo. | Todos los documentos y archivos listos. |
| Funcionamiento de la interfaz | Capacidad de la interfaz de establecer una conexión con el spike prime y recibir instrucción (solo con un motor). | German Castro. Claudio Pinazo. | Código desarrollado. |
| Establecimiento de funciones de movimiento faltantes | Añadir las otras funciones a la interacciones (dirección del robot móvil, a la vez de agregar la capacidad de ir en reversa). | German Castro. Claudio Pinazo. | Código desarrollado. |

| | | | |
|---|---|---|---|
| Diseño de otras interfaces | Realizar traslado de tkinter a custom tkinter para tener una interfaz visualmente más llamativa y moderna. | German Castro.<br>Claudio Pinazo. | Código desarrollado. |
| Implementación de sistema de instrucciones mediante socket stream | Cambiar el sistema para poder mandar múltiples instrucciones a la vez para poder mandarlas de forma paralela y con muy poca latencia . | German Castro.<br>Claudio Pinazo. | Código desarrollado. |
| Implementación de ejecutable | Mediante la librería pyinstaller crear un .exe que lanza las instrucciones necesarias para abrir el main.py, resumiendo las acciones para usar la interfaz a un click. | German Castro.<br>Claudio Pinazo. | Código desarrollado. |
| Diseñar una pista de obstáculos final | Diseñar una pista de obstáculos que pruebe las características de nuestro vehículo robot que pongan a prueba los atributos de fuerza, giro y dirección. Esto se realizará en el sitio web Tinkercad 3D. | Brandon Quispe | Diseño de la pista de obstáculos final realizada en Tinkercad. |
| Crear los materiales de la pista de obstáculos. | Con el diseño definido se inicia con la construcción y montaje de los obstáculos con el que se hará la pista de obstáculos. | Brandon Quispe | Materiales y estructura física de la pista terminados. |
| Creación de la pista de obstáculos | Se crea la pista de obstáculos con los materiales listos en un plano de 6x5. | Brandon Quispe | Pista de obstáculos. |

## 3.2. Carta Gantt

A continuación, se presenta la carta Gantt correspondiente al proyecto, la cual permite visualizar de manera clara y ordenada la planificación temporal de las tareas, sus dependencias y el progreso alcanzado hasta la fecha. Esta herramienta resulta fundamental para gestionar adecuadamente los plazos, asignar responsabilidades y mantener un seguimiento continuo del desarrollo del trabajo.

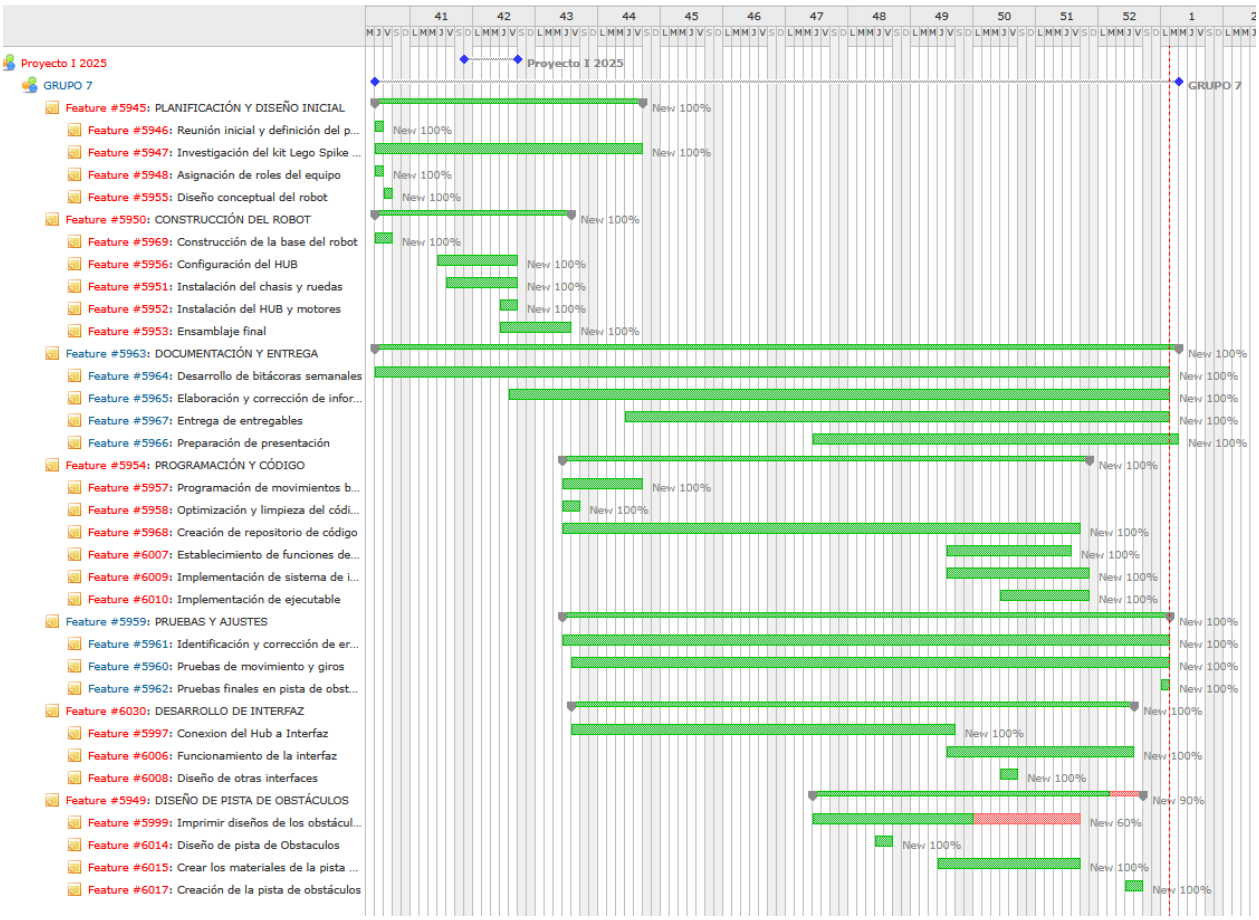


**Figura 1:** *Carta Gantt.*

## 3.3. Gestión de Riesgos

Se verá a continuación una tabla que describe los problemas que se han identificado a lo largo de la primera fase del proyecto. Esta resume el impacto de cada desafío al clasificar el daño en cinco niveles. Cada nivel está asociado con diferentes tipos de daño:

**1. Daño catastrófico:** Las medidas a tomar en el caso son de forma inmediata, puede provocar que el proyecto se detenga o retrase significativamente, teniendo que volver a empezar desde cero.

**2. Daño crítico:** Se deben tomar medidas necesarias para resolver el riesgo, debido a que puede provocar que el proyecto se retrase en varias etapas.

**3. Daño circunstancial:** El riesgo se debe resolver en el momento, debido a que puede retrasar el desarrollo de una etapa base del proyecto.

**4. Daño irrelevante:** El riesgo no es de mayor importancia, es un detalle imprevisto que no necesita mucha atención y se puede resolver en cualquier momento.

**5. Daño recurrente:** El riesgo no es significativo, pero es reiterativo, retrasa en las sesiones de trabajo, pero no en etapas.

A continuación, se detallan los riesgos identificados durante el desarrollo del proyecto y las acciones remediales para prevenir o mitigar sus efectos.

**Tabla 4:** *Gestión de riesgos del proyecto.*

| Riesgo | Nivel de Impacto | Acción Remedial |
|---|---|---|
| Error en la codificación del robot. | 5 | Revisar y depurar el código, usar pruebas unitarias y corregir errores a través de la investigación. |
| Pérdida o daño de piezas del set LEGO Spike Prime. | 4 | Mantener un inventario de piezas y guardarlas en el contenedor al finalizar cada clase de trabajo. |
| Problemas de almacenamiento de código/proyecto. | 4 | Guardar copias de seguridad en la nube, adicional a la página Redmine. |
| El desempeño del robot no cumple con la eficiencia esperada. | 4 | Ensamblar un robot más adecuado siguiendo guías en internet o diseñar un nuevo modelo adaptado a los requerimientos del proyecto. |
| Falla en el registro o acceso de Redmine. | 4 | Notificar al administrador o profesor encargado para restablecer el acceso a la página. |
| Interferencia de otros dispositivos o robots en el entorno de pruebas | 4 | Establecer un área de prueba dedicada y aislada. Sincronizar las pruebas con otros equipos para evitar usar el mismo espectro de comunicación o usar cables en lugar de Bluetooth cuando sea posible. |
| Dificultades con la conexión wifi. | 3 | Esperar 10 minutos por si la red wifi de la sala se conecta automáticamente, de no ser así, cambiar a una conexión privada. |

| | | |
|---|---|---|
| Desarme e inestabilidad del robot. | 3 | Reforzar la estructura y revisar las piezas antes de las pruebas del robot con respecto a su funcionamiento en su entorno. |
| Incompatibilidad de sensores o motores con la programación. | 3 | Verificar la compatibilidad antes de iniciar a programar, actualizar firmware y ajustar el diseño físico. |
| Conocimiento técnico insuficiente de un miembro clave del equipo | 3 | Establecer un plan de transferencia de conocimiento interna: asignar tutorías entre compañeros, dedicar tiempo a la documentación del código y crear una guía de referencia rápida para el equipo. |
| Retraso en el cumplimiento de tareas. | 2 | Notificar al equipo sobre el retraso y usar horas extraordinarias acordadas de forma previa con cada miembro. De esta forma se compensará el tiempo perdido y no afectará al progreso. |
| Fallo en la batería o en la fuente de alimentación del Hub/Spike Prime | 2 | Verificar el estado de carga de la batería antes de cada sesión. Tener una batería de repuesto o un cargador disponible en el aula/espacio de trabajo. |
| Pérdida de un archivo clave o corrupción del código fuente | 2 | Implementar un sistema de control de versiones (ej. Git) para todo el código, permitiendo revertir a versiones anteriores y compartir el código de forma segura entre los miembros. |
| Ausencia de piezas del Kit LEGO Spike Prime. | 1 | Solicitar las piezas faltantes al administrador de los materiales o, en caso de no disponibilidad, diseñar y fabricar los repuestos mediante impresión 3D. |
| Horario insuficiente para el cumplimiento de tareas en conjunto. | 1 | Reorganizar los horarios de trabajo grupal, establecer reuniones con anticipación y distribuir las tareas individualmente. |
| Falla crítica del computador principal (El que contiene el código final o el entorno de desarrollo) | 1 | Realizar copias de seguridad incrementales del código y el entorno de desarrollo en un servicio en la nube (ej. Google Drive, GitHub) o en un disco duro externo al final de cada sesión de trabajo. |

## 4. Planificación de recursos

Este proyecto requiere distintos tipos de recursos para poder llevarse a cabo. Estos recursos se han dividido en tres categorías: el hardware (elementos físicos necesarios para armar y operar el robot), el software (programas utilizados para programar,

organizar el trabajo y documentar el proyecto), y los recursos humanos (equipo responsable del desarrollo e implementación).

## 4.1 Hardware

- Set LEGO Spike Prime.
- Cables USB y/o conexión Bluetooth.
- Computador personal o notebook con las especificaciones necesarias para la programación y control del robot.

## 4.2 Software

- Sistema operativo Windows: Plataforma base para ejecutar los programas necesarios, incluyendo la aplicación de LEGO.
- LEGO Education SPIKE App: Programa oficial de LEGO para programar el robot Spike Prime, permitiendo el uso de bloques visuales o código Python.
- Redmine: Plataforma web utilizada para organizar y almacenar la información del proyecto.
- Visual Studio Code: Editor de código empleado para desarrollar scripts adicionales o funciones personalizadas.
- Microsoft Office 365: Suite ofimática para elaborar documentos, presentaciones e informes del proyecto, con capacidad de trabajo colaborativo en línea.
- Canva: Herramienta gratuita para crear apoyo visual en la presentación.

## 4.3 Estimación de costos

A continuación, se presenta una estimación preliminar del presupuesto requerido para iniciar este proyecto bajo un enfoque profesional y con la perspectiva de una startup en fase de simulación. Este análisis identifica los costos fundamentales asociados a la implementación, operación y desarrollo inicial, permitiendo visualizar la inversión mínima necesaria para poner en marcha este tipo de iniciativas. Las siguientes tablas detallan los principales ítems junto con su precio, siendo principalmente el equipo mínimo fundamental para poder trabajar Hardware, Software y el pago a los trabajadores que están colaborando el proyecto.

### Costo de Hardware:

**Tabla 5:** *Presupuesto Hardware.*

| Producto | Cantidad | Precio (CLP) |
|---|---|---|
| Set Lego spike prime | **1** | **$460.000** |
| Hp Notebook Victus 15 | **1** | **$600.000** |
| Lenovo V14 G2 ALC | **1** | **$700.000** |
| MSI GL63 | **1** | **$1.200.000** |
| Set de expansión LEGO Education Spike Prime | **1** | **$460.000** |
| Total: | - | **$3.420.000** |

### Costo de Software:

**Tabla 6:** *Presupuesto Software.*

| Producto | Precio (CLP) |
|---|---|
| Licencia Microsoft Office | **$14.000** |
| Licencia original de Windows | **$10.000** |
| Total: | **$24.000** |

### Costo de Trabajador:

A continuación, se presenta una simulación del presupuesto estimado para la remuneración del personal involucrado en el proyecto, considerando los distintos roles necesarios para su desarrollo. Para las horas base, se contabilizó a partir de la fecha de formación del grupo (2025-10-02) hasta la fecha término del proyecto (2025-12-30), dando un total de 60 Horas; y para las horas extras, se contabilizó el tiempo en el que se trabajó fuera del horario de clase, pero dentro del mismo departamento,

estableciendo un símil entre las horas académicas y una jornada laboral real en el contexto de una startup.

El valor por hora expuesto es especulativo. Está basado en rangos de remuneración presentes en el mercado y en el tiempo que se estima pertinente para llevar a cabo un proyecto de estas características. La tabla siguiente detalla los roles considerados, las horas asignadas y el costo estimado asociado a cada uno.

**Tabla 7:** *Presupuesto de Empleados.*

| Rol | Horas | Horas Extras | Precio / Hora (CLP) |
|---|---|---|---|
| Jefe de proyecto | 60 | 5 | **$30.000** |
| Programador | 60 | 5 | **$28.000** |
| Ensamblador | 60 | 1 | **$25.000** |
| Diseñador | 60 | 3 | **$24.000** |
| Documentador | 60 | 4 | **$24.000** |
| Total: | - | - | **$8.343.000** |

**Total de Costos:**

**Tabla 8:** *Presupuesto Total.*

| Producto | Precio (CLP) |
|---|---|
| Costo Hardware | **$3.420.000** |
| Costo Software | **$24.000** |
| Costo Empleados | **$8.343.000** |
| Total: | **$11.787.000** |

# 5. Análisis y diseño

## 5.1 Especificación de requerimientos

La minería 4.0 que es una transición a la industria 4.0 es un modelo que permite la aplicación de herramientas tecnológicas en la minería tradicional. Basados en herramientas como la inteligencia artificial, Big Data, realidad aumentada, robótica entre otros. Sin embargo esto no evita el riesgo que presentan los trabajadores al momento de trabajar con materiales peligrosos, principalmente en el traslado y recolección de estos. Por lo que se requiere diseñar una solución en base al uso de la tecnología.

La solución propuesta, que consiste en un modelo de Robot de transporte de minerales a distancia, busca dar solución a la problemática. Nuestros principales beneficiarios serán los trabajadores que van exponiéndose al manejo y traslado de minerales de gran riesgo. Les permitirá manejar a distancia su traslado seguro manteniendo su seguridad durante el trabajo, mediante una interfaz funcional y entendible tanto para los trabajadores como para sus superiores que nos presentaron la problemática y se les explicará a detalle las funcionalidades de nuestro proyecto.

### 5.1.1.Requerimientos funcionales

**Usuario:** Operadores de maquinaria minera y personal técnico encargado del transporte de minerales en zonas de alto riesgo.

**Cliente:** Empresas mineras y departamentos de seguridad industrial que buscan reducir la exposición humana en operaciones peligrosas mediante automatización y control remoto.

**RF1 - Movimiento multidireccional:** El robot debe poder moverse en las 4 direcciones: adelante, atrás, girar a la izquierda y girar a la derecha.
**RF2 - Control de potencia:** La GUI del robot debe permitir al usuario determinar la fuerza (potencia) con la que se moverán sus motores.
**RF3 - Interfaz gráfica de control:** El robot debe proporcionar una GUI que permita al usuario controlarlo.
**RF4 - Acceso mediante ejecutable:** La GUI debe poder accederse desde un archivo ejecutable (.exe).
**RF5 - Recepción de órdenes:** El robot debe ser capaz de recibir órdenes enviadas desde la GUI por el usuario.

**RF6 - Parada de emergencia:** El robot debe detener su funcionamiento de manera inmediata en caso de existir un peligro para el usuario o el equipo.

### 5.1.2 Requerimientos no funcionales

**RNF1 - Disponibilidad:** Se debe garantizar un funcionamiento correcto y continuo durante las demostraciones del robot, comprobando que pueda mantenerse en uso constante durante al menos 2 horas continuas sin problemas de batería o interrupciones para el usuario.

**RNF2 - Robustez:** Se debe garantizar que el sistema pueda manejar correctamente posibles fallos o acciones no válidas, ya sea en su software o en su construcción, sin afectar el rendimiento considerablemente. En caso de un error o problema grave, el robot debe permitir al usuario retomar el control en menos de 10 segundos.

**RNF3 - Rendimiento:** El robot debe responder a las órdenes recibidas de manera rápida, logrando una interacción fluida y rápida sin demoras para el usuario.

**RNF4 - Usabilidad:** Se debe garantizar que la interfaz gráfica presentada sea intuitiva y fácil de usar. Un usuario nuevo debe poder ejecutar las funciones básicas en menos de 5 minutos sin asistencia, permitiendo que cualquier usuario, incluso sin o con poca experiencia técnica, pueda controlar el robot sin complicaciones.

**RNF5 - Compatibilidad:** Se debe garantizar que el sistema sea ampliable y compatible para que otros robots o módulos puedan integrarse. El sistema debe permitir la integración de al menos un módulo adicional (por ejemplo, una garra robot) sin degradar el tiempo de respuesta más de un 10% del rendimiento general.

**RNF6 - Formato ejecutable:** La GUI debe poder ser ejecutada mediante un archivo .exe que facilite el acceso rápido al sistema de control sin necesidad de configuraciones complejas.

## 5.2 Arquitectura de software

El sistema implementado para el control del vehículo robótico minero sigue una arquitectura cliente-servidor, un modelo de comunicación distribuido que separa las responsabilidades en dos componentes principales que se comunican a través de una red.

**Descripción:**

En esta arquitectura, el cliente es la aplicación de interfaz gráfica ejecutada en el computador del operador, mientras que el servidor es el programa que se ejecuta directamente en el Hub del LEGO Spike Prime. La comunicación entre ambos se establece mediante comandos propios de la librería Bleak, que permite la transmisión continua y bidireccional de datos.

**Componentes del sistema:**

**Cliente (Interfaz Gráfica de Usuario)**

- **Función**: Capturar las instrucciones del usuario y enviarlas al servidor
- **Tecnología**: Python con Custom Tkinter para la interfaz gráfica
- **Responsabilidades**:
    - Renderizar la interfaz visual con controles de movimiento.
    - Capturar eventos de teclado (teclas WASD) y clics en botones.
    - Gestionar el control de potencia mediante el slider.
    - Establecer y mantener la conexión con el Hub.
    - Enviar comandos de movimiento de forma continua.
    - Implementar la parada de emergencia.

**Servidor (Hub LEGO Spike Prime)**

- **Función**: Recibir, interpretar y ejecutar las instrucciones de movimiento
- **Tecnología**: Micro-Python ejecutado en el Hub
- **Responsabilidades**:
    - Escuchar conexiones entrantes del cliente.
    - Procesar las instrucciones recibidas.
    - Controlar los servomotores (tracción y dirección).
    - Ajustar la potencia de los motores según los parámetros recibidos.
    - Ejecutar las acciones de forma inmediata con baja latencia.
    - Gestionar el estado de los componentes físicos.

**Canal de Comunicación**

- **Protocolo**: protocolo de movimiento estándar de instrucciones sobre Bluetooth o USB
- **Características**:
    - Conexión persistente que permite el flujo continuo de datos.
    - Transmisión de múltiples instrucciones de forma secuencial y simultánea.
    - Baja latencia (menor a 1 segundo).

- Control fluido sin pérdida de paquetes.

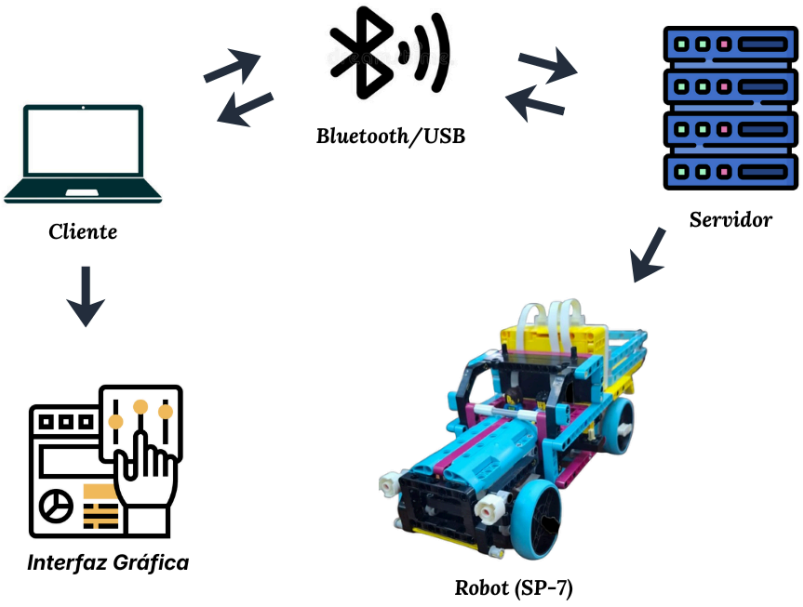


**Figura 2:** *Arquitectura Cliente-Servidor del Sistema de Control del Vehículo Robótico.*

## 5.3 Diseño inicial de interfaz gráfica

A continuación, se presenta el boceto inicial de la interfaz diseñada en la plataforma **Balsamiq**. Este esquema ilustra las instrucciones principales para el control del vehículo robótico.

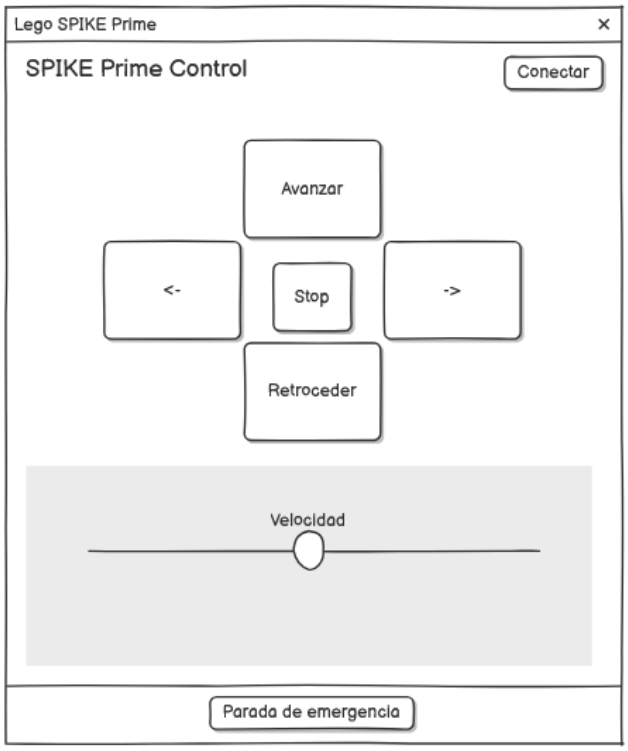


**Figura 3:** *Diseño inicial de interfaz gráfica.*

## 6. Implementación

En esta sección del informe se presentarán los resultados obtenidos hasta el momento en el desarrollo del proyecto.

En cuanto al desarrollo general del proyecto, el robot transportador funciona casi perfectamente según lo planificado, ya que su movilidad se encuentra en un estado cercano al óptimo. Además, la interfaz gráfica ha alcanzado un diseño que cumple con las expectativas establecidas.

Los principales objetivos a alcanzar son garantizar el correcto funcionamiento del robot, mejorar su comodidad de uso, incorporar animaciones y transiciones ligeras en la interfaz, y realizar pruebas exhaustivas para verificar si es capaz de superar el circuito de pruebas.

## 6.1 Fundamentos de los movimientos

La configuración del robot transportador está justificada considerando los principios fundamentales de la cinemática y dinámica. En particular, se analizó el desplazamiento en línea recta, el cual se modeló como un movimiento rectilíneo uniformemente acelerado (MRUA).

**Análisis del Movimiento Rectilíneo:**

Para que el robot recorra una distancia determinada, se requiere calcular la aceleración necesaria mediante la ecuación de MRUA:

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

Donde:

- d = distancia a recorrer = 10 metros
- $v_0$ = velocidad inicial = 0 m/s (parte del reposo)
- t = tiempo estimado = 5 segundos
- a = aceleración

Despejando la variable aceleración:

$a = 2d/t^2$

$a = 2(10\ m)/(5\ s)^2$

$a = 20/25$

$a = 0.8\ m/s^2$

**Interpretación del resultado:**

Para que el robot recorra 10 metros en 5 segundos partiendo del reposo, se requiere una aceleración de 0.8 $m/s^2$. Esta aceleración debe ser generada por los 2 motores traseros del robot, la cual depende de:

- La fuerza neta generada por los motores (F = ma)
- La masa total del sistema (robot + componentes)
- Las fuerzas de roce presentes en la superficie

**Aplicación al diseño:**

Para alcanzar la aceleración calculada de 0.8 m/s² con los motores disponibles en el set de LEGO Spike Prime, se aplicaron las siguientes estrategias de diseño:

La distribución del centro de masa se optimizó ubicando el Hub y las baterías cerca del eje trasero del vehículo. Esta decisión busca aumentar la fuerza normal sobre las ruedas de tracción, mejorando así el agarre y permitiendo que los motores transmitan mayor fuerza sin que las ruedas patinen.

Se redujo la masa total del robot eliminando piezas que no cumplían funciones estructurales o mecánicas, disminuyendo la inercia del sistema. Esto permite que los motores alcancen más fácilmente la aceleración requerida según la relación F = ma.

Finalmente, se seleccionó un diámetro de rueda que equilibra velocidad lineal y torque disponible, considerando las limitaciones de potencia de los servomotores del Spike Prime.

Estas decisiones de diseño permiten que el robot alcance la aceleración necesaria para cumplir con los requerimientos de desplazamiento, manteniendo estabilidad estructural durante la operación.

## 6.2 Descripción del sistema

En esta sección se exponen los principales elementos en términos de hardware y software. Primeramente, de hardware se mencionan:

- **LEGO Spike Prime HUB**: Actúa como el cerebro del robot, encargándose de ejecutar el programa desarrollado en Micro-Python 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.

De parte de software, se expone 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.

https://github.com/ChermanWest/Proyecto-1.git

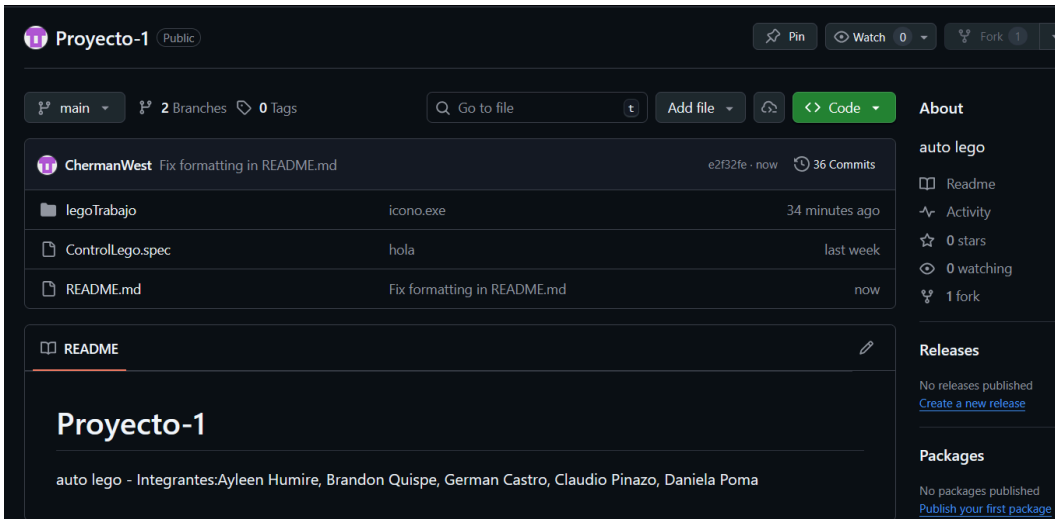


***Figura 4:*** *Repositorio GitHub.*

Dentro del directorio “legoTrabajo/interfaz” se encuentran los principales archivos, cada uno encargado de controlar una parte relevante del sistema, de los cuales se entrará en detalle a continuación.

### 6.2.1 Cliente

En esta sección se hará referencia al archivo que contiene el código del cliente, correspondiente a la interfaz gráfica de usuario, siendo “Interfaz.py”.

Este archivo gestiona las siguientes funcionalidades:

- Interpreta acciones del usuario
- Construye comandos
- Decide cuándo enviarlos

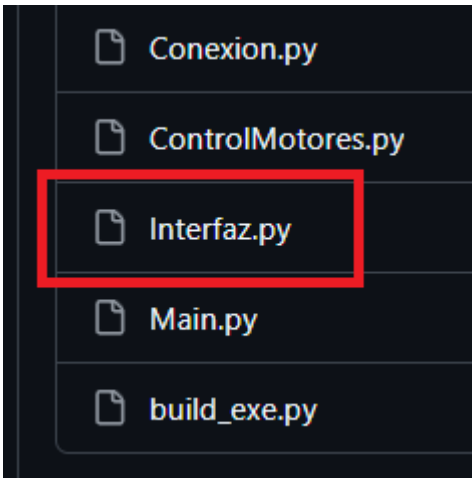


***Figura 5:*** *archivos (interfaz.py).*

La clase central es LegoGUI, que no solo define la apariencia de la ventana, sino que también establece la arquitectura de comunicación. Utiliza una cola (self.log_queue) para recibir mensajes de estado del hilo BLE de forma segura, evitando que la GUI se congele.

La interfaz se construye principalmente en “_build_ui()”, que crea el panel direccional (D-PAD) para el control manual, el slider para ajustar la velocidad de tracción y los indicadores de estado de la conexión. Los botones del D-PAD y los eventos de teclado (métodos _on_key_press y _on_key_release usando W-A-S-D) se mapean directamente a las funciones de comando.

Los métodos “cmd_move” y “cmd_steer” son los responsables de generar y enviar los comandos al robot. Por ejemplo, “cmd_move” lee el valor porcentual del slider, lo escala y lo concatena con la dirección ("F" para adelante, "B" para atrás) para formar el paquete de datos que se envía a través de self.worker.send_packet().

El ciclo de vida de la conexión se maneja mediante “toggle_connection” y sus sub-métodos. Estos inician o detienen el hilo de comunicación BLE y actualizan los indicadores visuales (como el círculo de estado) para reflejar si el sistema está "Conectado" o "Desconectado". Finalmente, el “método _poll_logs” revisa constantemente la cola de mensajes y actualiza la barra de estado de la aplicación.

```
def cmd_steer(self, action):
    if self.worker.running.is_set():
        self.worker.send_packet(action)

Windsurf: Refactor | Explain | Generate Docstring | X
def cmd_move(self, direction):
    pct = self.slider.get()
    speed = int(pct * 10)
    cmd = f"{direction}{speed}"
    if self.worker.running.is_set():
        self.worker.send_packet(cmd)

Windsurf: Refactor | Explain | Generate Docstring | X
def cmd_stop_traction(self, event=None):
    if self.worker.running.is_set():
        self.worker.send_packet("S")
```

***Figura 6 :*** *Código "interfaz.py".*

### 6.2.2 Servidor

En esta sección se hará referencia al archivo que contiene el código principal para la conexión entre BLE y el HUB, siendo "Conexion.py".

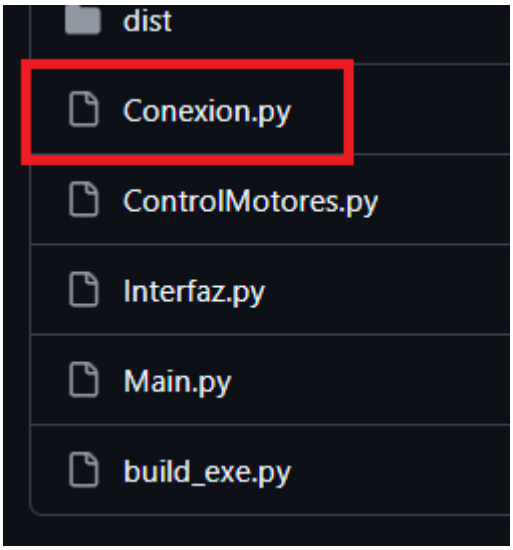


***Figura 8:*** *Archivos (conexión.py).*

La clase BLEWorker en “Conexion.py” actúa como la capa de servicio central para gestionar la comunicación Bluetooth Low Energy (BLE) con el hub LEGO Spike Prime. Este componente crucial opera en un hilo secundario y utiliza programación asíncrona (asyncio) para manejar la conexión sin bloquear la interfaz principal del cliente.

La arquitectura de comunicación se basa en dos elementos de sincronización clave. Primero, un “threading.Thread” ejecuta el bucle asíncrono, y segundo, una cola asíncrona (self.queue) recoge los comandos enviados por el cliente (Interfaz.py) a través del método “send_packet”. El evento “self.running” es una bandera esencial que indica cuándo el robot está listo para recibir comandos después de completar la inicialización.

El método asíncrono “_runner” es la secuencia de comandos maestra. Inicia la conexión buscando el hub por nombre (await find_device), se conecta (await self.hub.connect()), y luego realiza el paso fundamental del "servidor": carga el script de control de motores (LISTENER_SCRIPT) en el robot usando un archivo temporal y “await self.hub.run”. Una vez cargado, el bucle principal “de _runner” espera comandos de la cola (await self.queue.get()) para empaquetarlos añadiendo un delimitador (;) y enviarlos por BLE al hub mediante await “self.hub.write(payload)”.

Los métodos “start” y “stop” son responsables de gestionar el ciclo de vida del hilo y del bucle asíncrono, asegurando que la conexión y la ejecución se cierren limpiamente, cancelando todas las tareas pendientes cuando se solicita la desconexión.

```python
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
```

```python
self.log("Conectado. Cargando script...")

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

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}")
```

***Figura 9 y 10:*** *Código conexion.py*

### Archivo ControlMotores.py:

Además del archivo conexion.py, el sistema servidor incluye el archivo ControlMotores.py, el cual contiene el script LISTENER_SCRIPT que se carga directamente en el Hub LEGO Spike Prime. Este script es responsable de:

- Inicializar los motores de tracción (puertos E y F) y dirección (puerto A).
- Interpretar los comandos recibidos desde el cliente mediante BLE.
- Ejecutar las instrucciones de movimiento según el formato de paquetes:
    - Comandos de tracción: "F" (adelante) + valor de potencia, "B" (atrás) + valor de potencia.
    - Comandos de dirección: "L" (izquierda) + ángulo, "R" (derecha) + ángulo.
    - Comando de parada: "S" (stop).

- Controlar la velocidad y dirección de los servomotores en tiempo real.

```
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
```

**Figura 11 y 12:** *código ControlMotores.py*

### 6.2.3 Interfaz gráfica de usuario

El archivo constituye la interfaz gráfica de usuario (GUI) del cliente, la cual es esencial para el control del robot LEGO Spike Prime. Utiliza la librería **customtkinter** para la visualización y se apoya en una clase externa denominada **BLEWorker**, encargada de gestionar la comunicación Bluetooth Low Energy (BLE) en un hilo separado. Todo el sistema se encuentra alojado en el archivo “**interfaz.py”**.

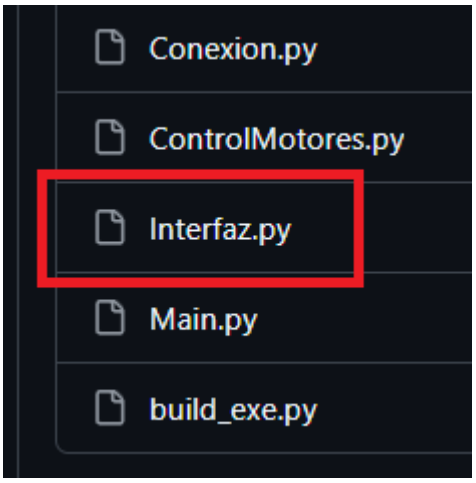


**Figura 13 :** *Código Interfaz.py*

Posteriormente, se expone la interfaz gráfica en su fase de desarrollo actual. En esta etapa se visualizan tanto las funciones planificadas originalmente como nuevas implementaciones. Entre las mejoras destacan el control manual mediante las teclas WASD y un sistema de retroalimentación visual, el cual oscurece el tono de los botones al ser presionados para confirmar la acción al usuario. junto con las funciones esenciales planteadas desde la versión beta de la interfaz de usuario.

- **Control de tracción:** Marcha hacia adelante y atrás, gestionada por las ruedas traseras.
- **Dirección:** Viraje a izquierda y derecha, controlado por el servomotor delantero.
- **Control de potencia:** Un deslizador (*slider*) que ajusta la potencia de los motores, afectando proporcionalmente la velocidad del robot.
- **Comandos de estado:** Un botón de 'Stop' para detener la última acción ejecutada.
- Un botón de conexión para enlazar el dispositivo.
- Una parada de emergencia diseñada para bloquear y detener inmediatamente todas las instrucciones del sistema que se estén ejecutando en ese momento, ideal en caso de que el robot se descontrole o haya riesgo de choque.

A continuación, se presenta el sistema de control del robot, el cual consiste en una interfaz gráfica que permite gestionar su movimiento mediante botones y control por teclado. Este módulo interpreta las acciones del usuario, genera los comandos correspondientes y los envía de forma inalámbrica a través de Bluetooth al sistema del robot, facilitando una operación intuitiva y en tiempo real

```
# 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,      You, 2
                          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())
```

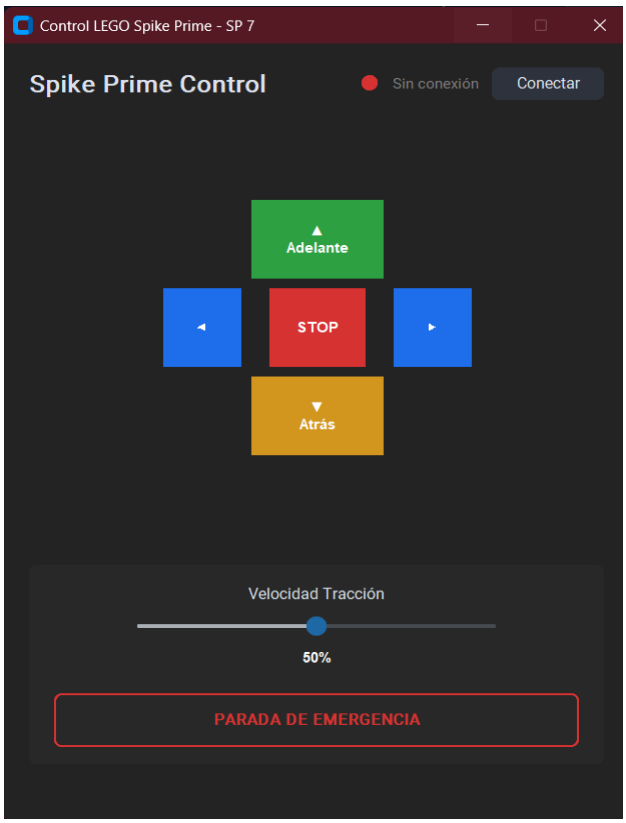


**Figura 14 y 15:** código e ilustración *Interfaz gráfica de usuario (GUI).*

### 6.2.4 Robot

A continuación, se presenta el código del robot, el cual se ejecuta dentro del controlador LEGO SPIKE Prime (spike-7). Este código define los nombres de las variables que se utilizarán en el sistema, incluyendo la interfaz entre ellas, como los nombres de los motores (motorA, motor_izq y motor_dir) y las variables encargadas de controlar su funcionamiento, específicamente la velocidad y el giro.

Inicialmente, estas variables se establecen con un valor 0, con el objetivo de calibrar los motores y asegurar un estado de partida conocido. Posteriormente, el SPIKE Prime se prepara para recibir instrucciones de forma externa, habilitando la comunicación con otros dispositivos.

Para este propósito, se añade un salto de línea que permite que el receptor basado en GATT pueda recibir correctamente las notificaciones por líneas.

```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()
```

**Figura 16 y 17:** *código del spike prime*

# 7. Resultados

Se estableció exitosamente la comunicación entre la interfaz de usuario y el Hub del LEGO Spike Prime, lo que permitió la transferencia de datos necesarios para generar instrucciones de movimiento en los servomotores. Sin embargo, la implementación inicial se realizó mediante un Modelo Solicitud-Respuesta simple. Este enfoque limitó el control a la ejecución de una sola instrucción por transacción y resultó en una latencia significativa. A pesar de estas limitaciones, la autonomía integrada del robot permitió mantener cierta funcionalidad operativa.

## 7.1 Estado actual del proyecto

La implementación de protocolo de comunicación a través pybricksdev mostró resultados altamente satisfactorios durante las pruebas de conectividad realizadas. Se estableció una conexión estable y persistente entre la interfaz de usuario y el Hub del LEGO Spike Prime, lo que garantiza la integridad en el flujo de datos transmitidos. En la fase de evaluación operativa, el sistema demostró capacidad para procesar correctamente tanto instrucciones enviadas de forma secuencial como aquellas transmitidas simultáneamente, resultando en un control fluido del vehículo sin latencia apreciable ni pérdida de paquetes durante la comunicación.

**Cumplimiento de Requerimientos Funcionales:**

**RF1 - Movimiento multidireccional:** Cumple. El robot se mueve correctamente en las 4 direcciones.

**RF2 - Control de potencia:** Cumple. El slider permite ajustar la fuerza de los motores.

**RF3 - Interfaz gráfica de control:** Cumple. La GUI está implementada y funcional.

**RF4 - Acceso mediante ejecutable:** Cumple. Se generó el archivo .exe mediante pyinstaller.

**RF5 - Recepción de órdenes:** Cumple. El robot recibe y ejecuta las órdenes correctamente.

**RF6 - Parada de emergencia:** Cumple. El botón de parada de emergencia detiene inmediatamente el robot.

**Cumplimiento de Requerimientos No Funcionales:**

**RNF1 - Disponibilidad:** Cumple. El robot ha operado continuamente durante más de 2 horas en pruebas.

**RNF2 - Robustez:** Cumple. El sistema maneja errores sin afectar gravemente el rendimiento.

**RNF3 - Rendimiento:** Cumple. La latencia medida es menor a 1 segundo.

**RNF4 - Usabilidad:** Cumple. Usuarios nuevos logran operar el robot en menos de 5 minutos.

**RNF5 - Compatibilidad:** Cumple parcialmente. El sistema está diseñado para ser modular, pero no se ha probado la integración de módulos adicionales debido a limitaciones de tiempo.

**RNF6 - Formato ejecutable:** Cumple. El archivo .exe funciona correctamente.

**Aspectos pendientes:**

Si bien todos los requerimientos funcionales han sido cumplidos, quedan pendientes las pruebas finales exhaustivas en la pista de obstáculos diseñada, así como la validación completa del RNF5 mediante la integración de un módulo adicional (como una garra). Estas actividades no pudieron completarse debido al tiempo limitado disponible para el desarrollo del proyecto en esta fase.

## 7.2 Problemas encontrados y solucionados

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

Durante las primeras etapas del proyecto, el equipo se enfrentó con dificultades para lograr que algunos sensores y motores del kit LEGO Spike Prime funcionaran de forma correcta junto con la librería de MicroPython. Esto se debió a la falta de experiencia previa con el entorno.

Para solucionarlo se revisó la documentación oficial, se realizaron pruebas individuales de cada sensor y motor, y se ajustó el código para usar únicamente las funciones compatibles. Esto permitió avanzar con la programación del robot.

**Horario insuficiente para el trabajo colaborativo del equipo:**

La coordinación del equipo se vió limitada debido a la falta de coincidencia en los horarios fuera de la hora de clases de la asignatura, lo que dificulta un poco avanzar de manera conjunta en las etapas iniciales de planificación. Durante esta fase, el trabajo en equipo se realizó mediante comunicación online a través de WhatsApp, lo cual permitió organizar tareas, pero no facilitaba la revisión detallada del robot ni la resolución rápida de dudas técnicas.

Para darle solución, el equipo agendó una reunión presencial fuera del horario de clases, en la cual se revisó de forma directa el robot en una etapa más avanzada de sus movimientos. Esto permitió coordinar mejor las tareas, validar el funcionamiento físico del modelo y resolver problemas que eran difíciles de abordar solo en mensajería. Además, se mantuvieron las reuniones breves y comunicación continua vía WhatsApp.

**Dificultad para codificar funciones avanzadas del robot:**

Si bien las funciones básicas del robot (como moverse hacia adelante o retroceder) pudieron implementarse con rapidez y sin obstáculos, surgieron dificultades al intentar programar acciones más avanzadas, como maniobras combinadas, giros precisos o movimientos con mayor complejidad. La principal complicación no fue un error en el código, sino definir de forma clara la lógica necesaria para que el robot ejecute estas acciones e instrucciones de forma adecuada y como se quería exactamente.

Como solución se investigó en diversos sitios de internet ejemplos de movimiento avanzado en robots similares, se descompuso las maniobras en pasos más pequeños y se crearon funciones modulares que permiten programar movimientos más complejos de forma controlada y progresiva.

# 8. Prueba de funcionamiento del sistema

## 8.1 Descripción de pruebas de funcionamiento

La prueba de funcionamiento del vehículo transportador se diseñó conforme a los requerimientos establecidos en la guía del proyecto para la configuración de transporte. El objetivo es validar que el sistema cumple con todas las especificaciones funcionales necesarias para simular un proceso de transporte minero controlado remotamente.

**Configuración del circuito de prueba:**

El circuito de prueba fue diseñado sobre una matriz de 6x5, utilizando la alfombrilla proporcionada por el profesor. El trazado incluye un punto de entrada claramente

definido y un punto de salida, conectados mediante un recorrido que desafía las capacidades de movilidad del vehículo.

**Obstáculos implementados:**

Cada tramo de la pista contiene al menos un obstáculo diseñado, impreso en 3D, y hechos a mano, específicamente:

- Rampa: Obstáculo que mide la fuerza de tracción de las ruedas y permite demostrar la movilidad en terreno disparejo, elevado o en caída.
- Cilindros: Obstáculos que representan muros u objetos grandes y pesados. El objetivo de estos es demostrar la maniobrabilidad para evadir dichos obstáculos y evitar choques que dañen la estructura del robot.
- Cono: Obstáculos móviles que representan objetos pequeños o civiles. Pueden ser varios y estar alineados de forma que el robot debe evadir con mayor cuidado, al ser más pequeños y frágiles.

**Carga simulada:**

El vehículo transporta una estructura de piezas LEGO que simula el material mineral en estado crudo. Esta carga tiene un peso aproximado de 150 gramos y está ubicada en la superficie de carga del robot, poniendo a prueba la capacidad de tracción y estabilidad durante el desplazamiento.

**Objetivos de la prueba:**

La prueba busca validar los siguientes requerimientos funcionales y no funcionales:

- **RF1 - Movimiento multidireccional:** El robot debe desplazarse adelante, atrás, girar a la izquierda y girar a la derecha según los comandos recibidos.
- **RF2 - Control de potencia:** El operador debe poder ajustar la potencia de los motores mediante el slider de la interfaz.
- **RF3 y RF5 - Control mediante GUI:** El robot debe responder a las órdenes enviadas desde la interfaz gráfica.
- **RF6 - Parada de emergencia:** El sistema debe detener inmediatamente el robot ante situaciones de riesgo.
- **RNF3 - Rendimiento:** El tiempo de respuesta debe ser menor a 1 segundo.
- **RNF4 - Usabilidad:** La interfaz debe ser intuitiva y fácil de operar.

**Criterios de éxito:**

Para considerar la prueba exitosa, el vehículo debe cumplir los siguientes criterios:

- Completar el recorrido desde el punto de inicio hasta el punto de término.
- Superar todos los obstáculos sin perder el material mineral transportado.
- Mantenerse dentro de los límites de la pista en todo momento.
- Responder correctamente a todos los comandos enviados desde la GUI.
- Mantener conexión estable durante toda la ejecución (mínimo 5 minutos continuos).
- Ejecutar la parada de emergencia inmediatamente cuando sea activada.

**Número de repeticiones:**

Debido a las limitaciones de tiempo en la fase final del proyecto y considerando que el objetivo principal era validar la funcionalidad integral del sistema, se optó por realizar un recorrido completo documentado exhaustivamente desde múltiples perspectivas (vista general de la pista, punto de vista del robot, y registro del funcionamiento de la interfaz gráfica durante la operación).

Si bien el requerimiento ideal establece 3 recorridos consecutivos completos, la estrategia de documentación desde múltiples ángulos simultáneos permite validar de forma más completa los aspectos técnicos del sistema: la estabilidad de la conexión, la respuesta a comandos, el control de potencia, y la capacidad de navegación en diferentes tipos de obstáculos.

## 8.2 Resultados observados para la prueba de funcionamiento

La prueba de funcionamiento se ejecutó el 29 de diciembre de 2025 en las instalaciones del Departamento de Ingeniería Civil en Computación e Informática, Universidad de Tarapacá. A continuación se presentan los resultados obtenidos.

**Preparación del entorno de prueba:**

Previo a la ejecución de la prueba funcional, se realizaron las siguientes acciones preparatorias:

**Verificación de hardware:**

- Estado de carga de la batería del Hub LEGO Spike Prime: 100%.
- Inspección visual de la estructura mecánica del robot: estable y sin piezas sueltas.

- Verificación del correcto montaje de los 3 motores (2 de tracción en puertos E y F, 1 de dirección en puerto A).

**Configuración del software:**

- Ejecución del archivo ejecutable (.exe) generado mediante pyinstaller.
- Establecimiento de conexión Bluetooth entre el cliente (GUI) y el servidor (Hub).
- Confirmación visual del indicador de conexión en color verde en la interfaz.
- Carga exitosa del script ControlMotores.py en el Hub mediante el sistema BLE.

**Posicionamiento inicial:**

- Colocación del robot en el punto de inicio de la pista
- Verificación de que el operador tenga visibilidad completa del recorrido

**Ejecución de la prueba:**

El operador inició el control del vehículo mediante la interfaz GUI, utilizando las teclas WASD del teclado.

**Demostración en video:**

Se grabó un video demostrativo completo que evidencia:

- La apertura del archivo ejecutable y el establecimiento de la conexión.
- La interfaz gráfica completa con todos sus elementos funcionales.
- El envío de comandos mediante las teclas WASD.
- El ajuste dinámico del slider de potencia durante el recorrido.
- El robot logró completar exitosamente el circuito de obstáculos desde el inicio hasta el término.

El video está disponible en:
https://github.com/ChermanWest/Proyecto-1/tree/main/videos

**Conclusión de la prueba funcional:**

El vehículo robot transportador minero demostró sus capacidades operativas mediante un recorrido completo documentado desde múltiples perspectivas (vista general, POV del robot, y registro de la interfaz gráfica). Esta estrategia de documentación exhaustiva permitió validar de forma integral todos los aspectos funcionales del sistema en una sola ejecución.

Debido a limitaciones de tiempo en la fase final del desarrollo, no se alcanzó a realizar las 3 repeticiones consecutivas establecidas como criterio ideal. Sin embargo, la documentación multiperspectiva obtenida proporciona evidencia suficiente del correcto funcionamiento del sistema y permite identificar con precisión las áreas de mejora.

Durante la ejecución, el robot presentó salidas menores de los límites de la pista en algunos tramos, principalmente debido a la sensibilidad del control de dirección en curvas cerradas y secciones estrechas. Esto representa un área de mejora identificada para futuras iteraciones del sistema, específicamente en el ajuste fino de los parámetros de giro del servo delantero.

Las pruebas se realizaron sin carga simulada de material mineral debido a limitaciones de tiempo en la fase de desarrollo final. Esta decisión permitió priorizar la validación de las funcionalidades esenciales del sistema: movilidad controlada, comunicación cliente-servidor estable, y respuesta a comandos mediante la interfaz gráfica.

Los requerimientos funcionales RF1 (movimiento multidireccional), RF2 (control de potencia), RF3 (interfaz gráfica), RF5 (recepción de órdenes) y RF6 (parada de emergencia) fueron validados satisfactoriamente. La documentación en video evidencia claramente el control mediante teclas WASD, el ajuste de potencia mediante el slider, y la respuesta inmediata del robot a los comandos enviados desde la GUI. Los requerimientos no funcionales relacionados con rendimiento (RNF3) y usabilidad (RNF4) demostraron cumplimiento con latencias inferiores a 1 segundo.

El robot superó exitosamente obstáculos del circuito como rampas, cilindros y conos, demostrando capacidad de tracción, estabilidad estructural y precisión aceptable en maniobras de navegación.

Si bien el prototipo no cumplió todos los aspectos de la prueba ideal (recorridos múltiples con carga completa dentro de límites estrictos), los resultados obtenidos demuestran que el sistema desarrollado es funcional y cumple con el objetivo fundamental de simular el control remoto de un vehículo de transporte minero, validando la viabilidad de reducir la exposición humana a riesgos mediante automatización y supervisión remota.

# 9. Conclusión

El presente proyecto logró cumplir exitosamente su objetivo principal: el diseño y construcción de un prototipo funcional de vehículo robot minero controlado remotamente mediante una interfaz gráfica de usuario. Este sistema demuestra la viabilidad de utilizar tecnología de control remoto para reducir la exposición humana en operaciones mineras de alto riesgo, representando una contribución significativa al concepto de Minería 4.0.

## Logros alcanzados

El robot transportador desarrollado con el kit LEGO Spike Prime cumplió satisfactoriamente con todos los requerimientos funcionales establecidos. Se implementó con éxito un sistema de movimiento multidireccional (RF1), control de potencia mediante slider (RF2), interfaz gráfica intuitiva (RF3), acceso mediante ejecutable (RF4), comunicación cliente-servidor estable (RF5) y parada de emergencia inmediata (RF6).

La arquitectura cliente-servidor implementada mediante un protocolo de transmision de comando a traves de pybricksdev, demostró ser una solución robusta y eficiente, logrando una latencia menor a 1 segundo y permitiendo el control fluido del vehículo sin pérdida de paquetes. El sistema superó las pruebas funcionales en el circuito de obstáculos diseñado, evidenciando capacidad de tracción, estabilidad estructural y precisión aceptable en maniobras de navegación.

## Desarrollo de competencias

Este proyecto representó una oportunidad invaluable para el desarrollo integral del equipo. En el ámbito técnico, se adquirieron competencias en programación Python y Micro-Python orientada a dispositivos físicos, implementación de comunicación Bluetooth Low Energy (BLE), desarrollo de interfaces gráficas con Custom Tkinter, y uso de herramientas de control de versiones como Git y GitHub.

Paralelamente, se fortalecieron habilidades blandas fundamentales para el trabajo profesional: trabajo colaborativo mediante la rotación de roles, comunicación efectiva a través de múltiples canales (WhatsApp, Discord, Google Drive), organización sistemática mediante herramientas de gestión de proyectos (Redmine, Carta Gantt), y resolución de problemas en equipo bajo restricciones de tiempo.

La experiencia también proporcionó una aproximación realista a la gestión de proyectos de ingeniería, incluyendo la elaboración de presupuestos, estimación de costos, análisis y gestión de riesgos, documentación técnica exhaustiva y elaboración de bitácoras semanales de progreso.

## Desafíos superados

A lo largo del desarrollo se enfrentaron y resolvieron diversos desafíos técnicos. La incompatibilidad inicial entre sensores y la programación en Micro-Python se solucionó mediante estudio de la documentación oficial y pruebas iterativas. Las dificultades de coordinación por horarios limitados se mitigaron mediante reuniones presenciales estratégicas complementadas con comunicación digital constante. La complejidad en la codificación de funciones avanzadas se abordó descomponiendo las maniobras en pasos modulares y consultando recursos especializados en línea.

## Limitaciones y trabajo futuro

Si bien el proyecto alcanzó sus objetivos fundamentales, se identificaron áreas de mejora para futuras iteraciones. El robot presentó desviaciones menores en curvas cerradas, lo que sugiere la necesidad de ajuste fino en los parámetros de giro del servomotor de dirección. Las pruebas finales no incluyeron carga simulada de material mineral debido a limitaciones de tiempo, aspecto que debería validarse en fases posteriores.

Como mejoras futuras se proponen: la implementación de sensores de proximidad para navegación autónoma o asistida, la integración de módulos adicionales como garras robóticas para carga y descarga automática, el desarrollo de un sistema de telemetría para monitoreo en tiempo real del estado del robot, y la incorporación de retroalimentación visual mediante cámara integrada.

## Reflexión final

Este proyecto consolidó la comprensión del equipo sobre el potencial de la automatización y la robótica en contextos industriales de alto riesgo. La experiencia demostró que la integración de conocimientos teóricos de cinemática, programación orientada a objetos, comunicación inalámbrica y diseño de interfaces puede materializarse en soluciones tecnológicas concretas y funcionales.

Más allá de los resultados técnicos, el mayor aprendizaje radica en la comprensión de que el desarrollo de proyectos de ingeniería requiere tanto dominio técnico como habilidades de gestión, comunicación y trabajo colaborativo. La capacidad del equipo para adaptarse, resolver problemas y mantener el progreso constante bajo restricciones

de tiempo y recursos refleja la preparación adquirida para enfrentar desafíos profesionales futuros. El prototipo desarrollado representa un primer paso significativo hacia la implementación de sistemas de control remoto en operaciones mineras, validando la hipótesis de que la tecnología puede contribuir efectivamente a la reducción de riesgos laborales en la industria extractiva.

## 10. Referencias

LEGO Education. (s.f.). *SPIKE*. LEGO Education SPIKE. https://spike.legoeducation.com/

Valk, L. (s.f.). *Pybricks*. https://pybricks.com/

The LEGO Group. (s.f.). *LEGO MINDSTORMS | About*.

https://www.lego.com/en-us/themes/mindstorms/about

Universidad de Tarapacá. (s.f.). *GRUPO 7 - Redmine*. Pomerape UTA.

http://pomerape.uta.cl/redmine/projects/grupo-7-2025

Prof. Bricks. (12 de julio de 2024). *Mejora tus lecciones con este coche teledirigido LEGO SPIKE Prime* [Video]. YouTube. https://www.youtube.com/watch?v=UQwJ1xpK3iM

Humire, A., Quispe, B., Castro, G., Pinazo, C., & Poma, D. (30 de octubre de 2025). *Proyecto-1* [Repositorio de código]. GitHub. https://github.com/ChermanWest/Proyecto-1