UNIVERSIDAD DE TARAPACÁ
Universidad del Estado

DEPARTAMENTO DE INGENIERÍA CIVIL EN COMPUTACIÓN E INFORMÁTICA

INFORME AVANCE
“TIGER BOT”

Integrantes:	**Diego Lopez** **Sebastian Becerra** **Gustavo Morales** **Bryan Vega** **Sergio Huanca**
Asignatura:	**Proyecto I**
Profesor:	**Humberto Urrutia**

DICIEMBRE - 2023

Historial cambios

Fecha	Versión	Descripción	Autor
12/10/23	1.0	Inicio informe avance	Sergio Huanca, Sebastian Becerra
05/12/23	2.0	Final informe avance	Sergio Huanca, Sebastian Becerra, Bryan Vega

Tabla de Contenidos

1. Panorama General

1.1. Introducción

Inicios de Ole Kirk y Lego

La historia de lego comienza en 1932 en Dinamarca cuando Ole Kirk (su fundador) abre un taller de carpintería junto a un pequeño grupo de aprendices los cuales construyó juguetes de madera de abedul. Cabe recalcar que eran tiempos de crisis por lo que tuvieron que hacer bajos sus costes de producción. La fábrica fue creciendo llegando a 40 empleados hasta que en el 47 (1947) Ole Kirk conoce a Hilary Page diseñador de la compañía Kiddicraft que había patentado bloques rectangulares de madera con conectores en su base superior. Al principio fue duro hacer el cambio de madera a plástico ya que los juguetes de plástico no eran bien vistos pero con el tiempo lograron darle un buen enfoque y hoy en día lego es visto como algo más que bloques de plástico, es visto como sinónimos de creatividad, diversión y aprendizaje. Actualmente Lego es dirigido por Kjeld Kirk Kristiansen nieto de Ole Kirk quien le ha dado a la compañía un enfoque más amplio centrándose tanto en juegos de mesa, videojuegos hasta parques temáticos.

LEGO además lleva desarrollando plataformas para que los niños aprendan robótica desde el siglo pasado. No se trata de simples cubos con los que podemos hacer obras de arte, sino que con sus productos podemos también aprender a crear robots, o al menos en su versión básica. La historia se remonta hasta 1986, que fue cuando la compañía lanzó un producto de LEGO controlado por un ordenador.

El nombre de LEGO Mindstorm RCX sirve para nombrar a dos cosas distintas: al LEGO Mindstorms original y al bloque lógico de cualquier Mindstorms. En este caso hablamos de la primera versión, la original, el primer producto educativo del fabricante.

Nuestro proyecto “Tiger Bot” ha sido desarrollado con el kit lego Mindstorm ev3 con conocimientos aprendidos tanto durante el rodaje del proyecto como con conocimientos de asignaturas pasadas.

1.2. Objetivos

1.2.1. Objetivo General

Diseñar y armar un robot con el kit Lego Mindstorms EV3 controlado a través de una interfaz gráfica remota usando Python.

1.2.2. Objetivo Específico

- Armar un prototipo funcional del robot que golpee una pelota y llegue a un objetivo aproximado.
- Estudiar módulo tkinter para escribir interfaz gráfica.
- Estudiar módulo socket para establecer comunicación remota.
- Aplicar conocimientos en mecánica clásica para predecir el tiro de la pelota.
- Gestionar tiempos de trabajo para maximizar tiempo.
- Concretar el informe avance.
- Efectuar presentación del informe avance.

1.3. Restricciones

Hay un conjunto de restricciones que deben considerarse para asegurar el éxito en el desarrollo del robot, las cuales se detallan en la siguiente tabla:

Restricción	**Descripción**
Tiempo	Fecha estipulada para entregar proyecto
Materiales	Kit de piezas lego ev3
Número Integrantes	Máximo 5 personas por grupo
Temática proyecto	Debe tener un palo de golf que golpee pelota y llegue a un objetivo aproximado
Control robot	Robot debe ser controlado a distancia
Plataforma documentación	Todos los archivos redactados, serán subidos a Redmine.

1.4. Entregables

Informe avance y presentación	Corrección de informe v1	Bitácoras semanales
Avance del proyecto	Avance Wiki	Producto final

2. Organización del Personal

Cada miembro del grupo recibe una tarea específica para desarrollar semanalmente, indicada en la bitácora. Aunque cada uno tiene un rol general dentro del equipo, se asignan roles más específicos a medida que progresa el proyecto y las tareas se desarrollan. El trabajo fuera del horario de clase se basa principalmente en el autoaprendizaje de cada integrante, con comunicación a través de WhatsApp para informar sobre el progreso alcanzado.

2.1. Descripción de los Roles

Rol	**Descripción**
Jefe de grupo	Encargado de representar al equipo de trabajo, de la organización y de la toma de decisiones.
Programador	Encargado de desarrollar e implementar el código en Python logrando así que el robot pueda ejecutar las acciones solicitadas.
Ensamblador	Encargado de diseñar y armar el robot de tal manera que pueda moverse en todas las direcciones y lance un proyectil.
Diseñador	Encargado de la estética de la interfaz gráfica.
Documentador	Encargado de realizar los informes, presentaciones, bitácoras, video, manual de usuario y wiki del proyecto.

2.2. Mecanismos de Comunicación

Previo a cualquier necesidad, coordinamos a través de un grupo de WhatsApp para establecer acuerdos sobre tareas a completar, plazos, horarios de reunión y trabajo futuro. Además, utilizamos este medio para compartir imágenes del progreso y llegar a consensos en diversos aspectos. La comunicación con el profesor se lleva a cabo directamente a través de la intranet y en encuentros presenciales.

3. Planificación del Proyecto

3.1. Actividades

Actividad	Descripción	Responsable
Redacción de bitácoras	Registro de todas las actividades que se desarrollan semanalmente	Sebastián Becerra y Sergio Huanca
Contabilizar piezas	Encargado de ver que estén las piezas y contabilizarlas	Gustavo Morales
Wiki	Se capturan y comparten ideas e información del proyecto	Sergio Huanca
Vídeos y fotos	Registro audiovisual avances proyecto	Sergio Huanca
Organización	Designación de la actividad que estará encargado cada integrante	Diego López
Redacción carta Gantt	Planificar actividades a lo largo del proyecto	Sebastián Becerra

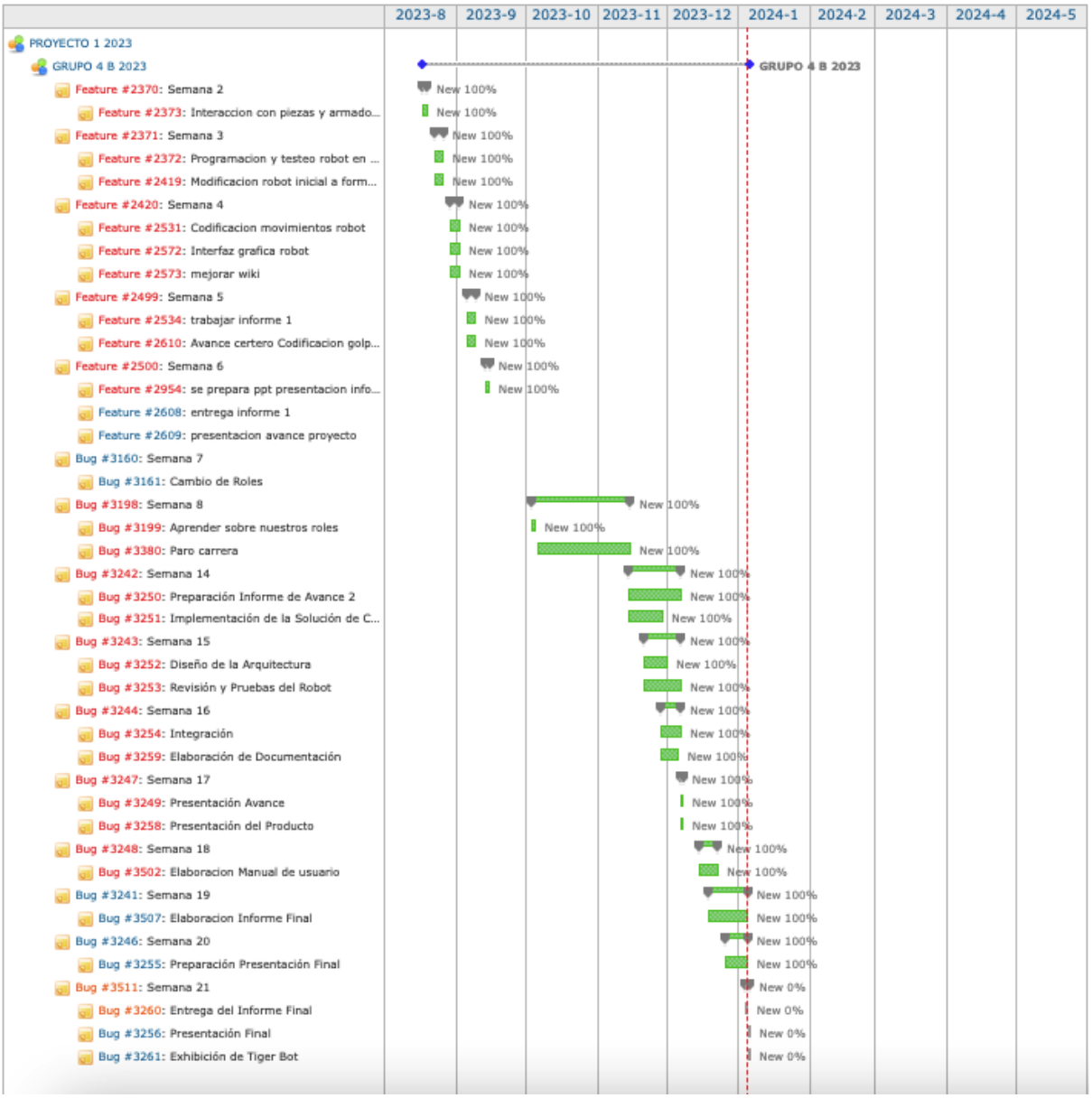
Búsqueda de ideas	Buscar nuevas ideas y formas de hacer el proyecto	Gustavo Morales-Bryan Vega
Construcción Robot	Armado y construcción robot	Diego López - Gustavo Morales - Bryan Vega

Administrar Redmine	Conocer la forma en la que se usa, sus funcionalidades y su aplicación	Sebastián Becerra
Informe avance	Añadir nuevos puntos del informe avance	Sebastián Becerra y Sergio Huanca
Estimación de costos	Calcular el presupuesto del proyecto	Sergio Huanca
Presentación avance del proyecto	Elaborar presentación del avance del proyecto y luego exponerlo	Todo el grupo
Estudiar y realizar pruebas en Python	Estudio del lenguaje de programación Python y de la librería ev3dev	Diego López y Gustavo Morales
Análisis del esqueleto del robot	Analizar y revisar la estructura del robot para ver si cabe la posibilidad de alguna modificación	Bryan Vega

Programación de los movimientos	Aplicar los conocimientos estudiados anteriormente en Python	Diego López y Gustavo Morales
Familiarizarse con LEGO EV3 GUN	Estudio de la librería ev3dev	Diego López y Gustavo Morales
Depuración Código	Proceso en el cual se identificarán y corregirán errores en el algoritmo	Diego López
Pruebas de funcionamiento	Chequear que el robot esté cumpliendo su propósito	Gustavo Morales, Bryan Vega
Documentación del código	Añadir información para explicar lo que hace cada parte del código	Diego López
Informe final	Redacción del informe final	Sebastián Becerra y Sergio Huanca
Presentación final	Creación del material para la presentación final	Sebastián Becerra y Sergio Huanca

3.2. Asignación de Tiempo

La planificación de cada tarea y actividad la llevamos a cabo utilizando la carta Gantt, lo que nos permitió mantener un enfoque preciso para establecer un orden adecuado en las tareas desarrolladas a lo largo del proyecto.



3.3. Gestión de Riesgos

Niveles de peligro:

1- Muy alto 2- Alto 3- Medio 4- Bajo

Problema	Probabilidad concurrencia	Nivel riesgo	Resolución

Programador se enferma	20%	3	Programador desde casa prueba códigos y el equipo ejecuta códigos presencial en universidad
Se echa a perder computador personal	10%	2	Trabajar archivos en la nube y pedir prestado computador carrera
Desarme robot ante caída	30%	2	Ver fotos/estructura base última sesión y repararlo
Pérdida código movimiento robot	40%	3	Enviarlo constantemente por WhatsApp
Incumplimiento tareas	20%	2	Reorganizar tiempos de trabajo además de funciones dependiendo disponibilidad integrantes grupo
Enfermedad general miembros grupo	30%	2	Integrante trabaja desde su casa en lo posible y se designa integrante presencial que reemplaza momentáneamente su puesto
Rearme completo del proyecto por incompatibilidad con lo pedido	30%	2	Se reconstruye y reprograma proyecto basado en lo que ya se tenía, reciclando códigos y funcionalidad
Descarga batería robot	20%	2	Estar constantemente cargando la batería para evitar que eso suceda sea el momento que sea

Escasez piezas	10%	2	Ir a buscar piezas requeridas donde el ayudante

4. Planificación de los Recursos

4.1. Hardware

El término 'hardware' hace referencia a todos los componentes físicos y tangibles presentes en un sistema informático o dispositivo electrónico. Entre estos componentes se encuentran la CPU, la memoria RAM, el disco duro y otros elementos fundamentales para el funcionamiento adecuado. En síntesis, el hardware representa la parte física y palpable de cualquier equipo tecnológico, posibilitando la ejecución de programas y el procesamiento de información.

Producto	**Costo**	**Cantidad**	**Total**
EV3 LEGO MiNDSTORMS	$1,147,176	1 c/u	$1,147,176
Notebook Lenovo (Arriendo x Horas trabajadas)	$800	5 c/u (91:30 Horas Trabajadas)	$366,000
Internet fibra óptica (Cantidad x mes trabajado)	$24,000	5 c/u (5 Meses Trabajados)	$120,000
Tarjeta SD 64GB	$4,490	1 c/u	$4,490
Set 6 pelotas de Ping Pong	$2,990	1 c/u	$2,990
Total (16 semanas)			**$1,640,656**

4.2. Software

El término 'software' hace referencia a los programas, aplicaciones y conjuntos de instrucciones que posibilitan a una computadora o dispositivo electrónico ejecutar tareas específicas. Es fundamental para el funcionamiento de cualquier dispositivo tecnológico, ya que sin él, el hardware carecería de las instrucciones necesarias para llevar a cabo cualquier tarea o función.

Producto	**Costo**	**Cantidad**	**Total**
S.O Linux	Gratis	1	$0
Google Docs	Gratis	1	$0
Canva	Gratis	1	$0
Librería EV3	Gratis	1	$0
Visual Studio Code	Gratis	1	$0
Total (21 semanas)			**$0**

4.3. Estimación de costos

La suma de estos componentes refleja el costo total por empleado. Es esencial gestionar eficazmente estos costos para el manejo financiero óptimo de la empresa.

Integrante	**Valor hora**	**Horas trabajadas**	**Total**
Diego López	$16,000	91:30	$1,464,000
Sebastián Becerra	$14,500	91:30	$1,326,750
Gustavo Morales	$13,000	91:30	$1,189,500
Bryan Vega	$11,000	91:30	$1,006,500
Sergio Huanca	$10,000	91:30	$915,000
Total (21 semanas)			**$5,901,750**

Este constituye el componente primordial dentro del ámbito de gestión de costos en un proyecto, un campo de conocimiento que abarca la planificación, monitoreo y control de los costos monetarios asociados a un proyecto.

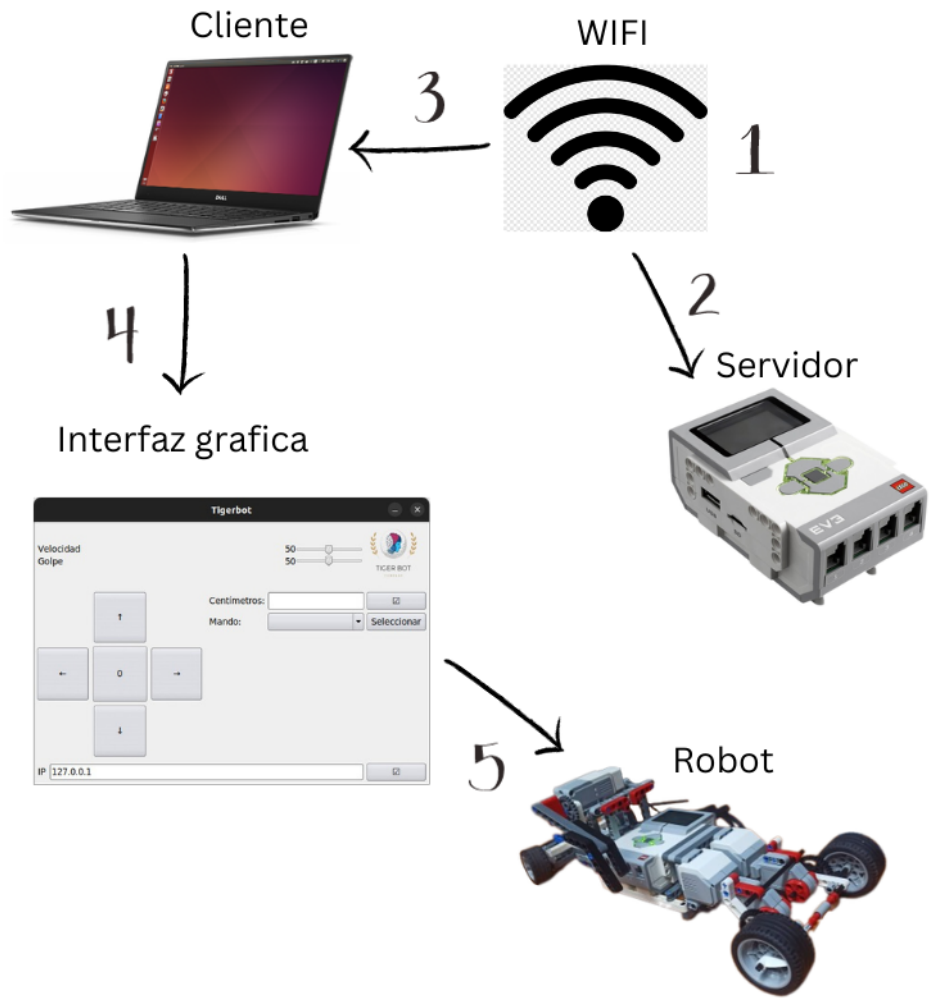
Total	**Costo Total**
Total Software	$0
Total Hardware	$1,640,656
Total Empleados	$5,901,750
Total	**$7,542,406**

5. Análisis y Diseño

5.1. Especificación de Requerimiento

Funcionales	**No funcionales**
El robot debe poder moverse.	La interfaz gráfica debe contener los movimientos del robot (avanzar, retroceder, girar izquierda, girar derecha) además del tiro de golf.
El robot debe poder golpear una pelota hacia un punto definido.	La programación debe ser en Python.
El robot debe ser controlado por una interfaz gráfica.	La estructura del robot debe ser estable al igual que sus movimientos (construido con piezas del kit Lego Mindstorm EV3 Education).
El golpe del robot debe generar movimiento parabólico en la pelota.	El robot debe poder ser controlado desde un computador con Linux.
	La interfaz gráfica debe ser de fácil entendimiento para el usuario

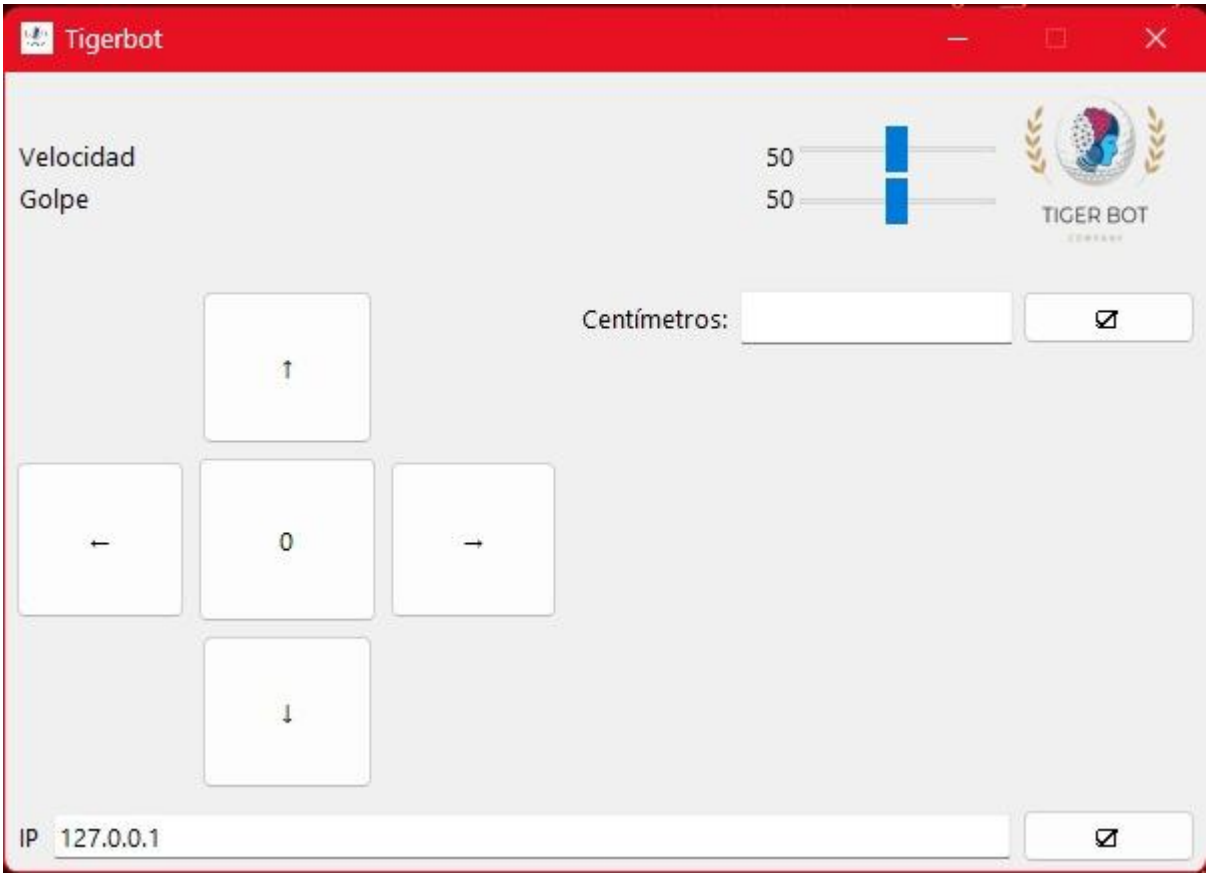
5.2. Arquitectura



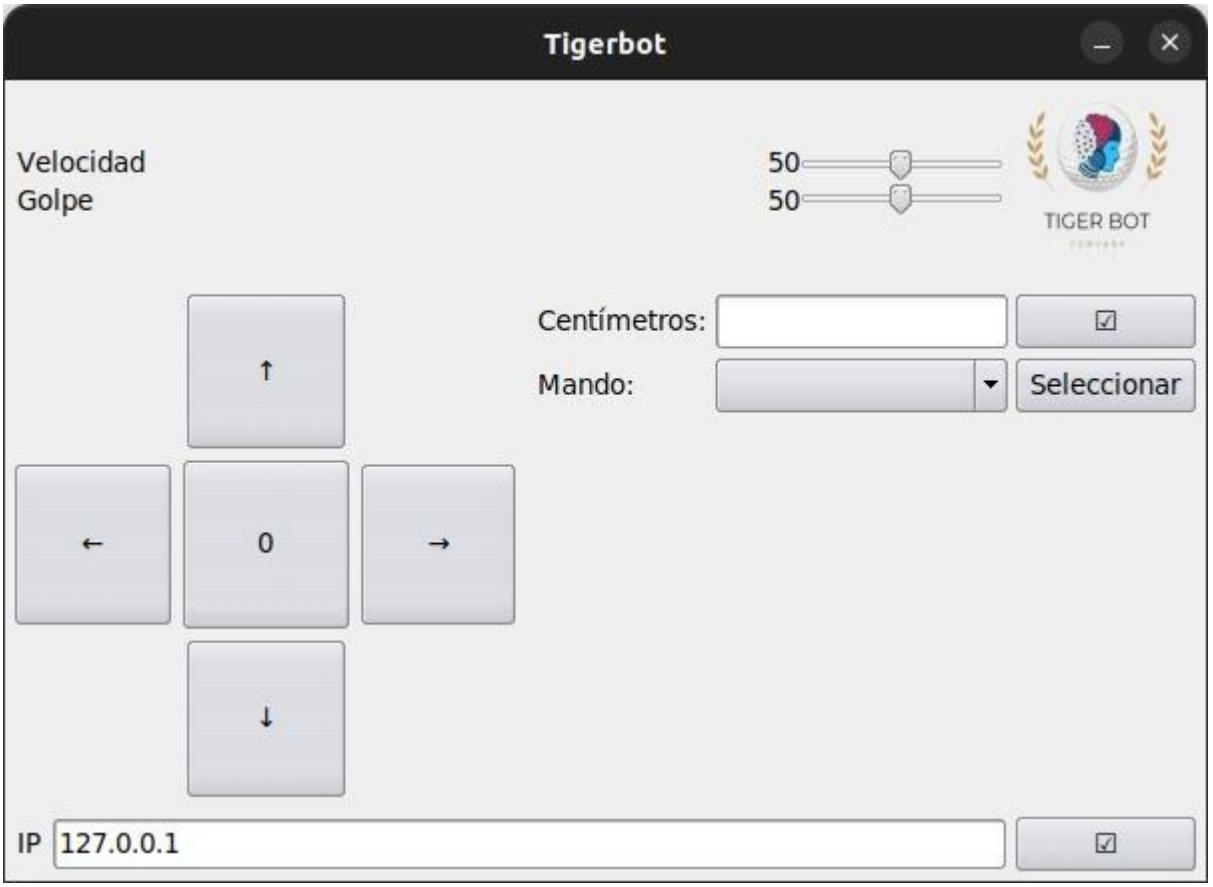
Explicación arquitectura Tigerbot

1. Se conecta el robot y un computador a la misma red wifi.
2. Se inicia el servidor para conectar la interfaz.
3. Se abre la interfaz para controlar al robot.
4. La interfaz se conecta al servidor y el usuario puede controlar el robot.
5. El robot realiza los movimientos solicitados por el usuario gracias a la conexión entre el servidor y el computador.

5.3. Interfaz



Interfaz gráfica en Windows



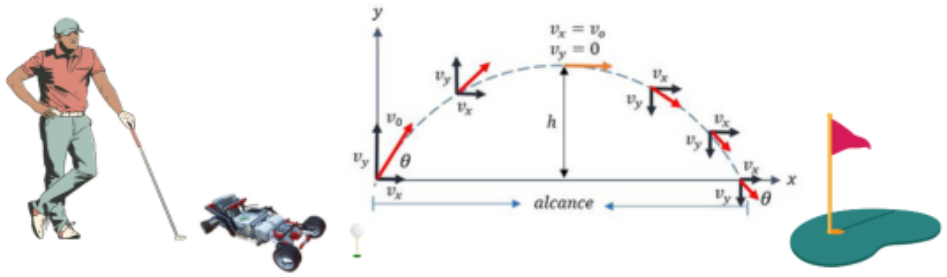
Interfaz gráfica en Linux

Se ha desarrollado una interfaz en tkinter con el propósito de mejorar la maniobrabilidad del robot y simplificar su utilización. Esta interfaz contendrá las siguientes opciones:

1-Movimiento del robot: Contará con 4 botones para el desplazamiento del robot, los cuales permitirán avanzar, retroceder, girar a la izquierda y a la derecha. También contará con la opción de ajustar manualmente su velocidad.
2-Conexión mando: Contará con conexión a mando para su movimiento
3-Acción del robot: Contará con un botón de golpear que sería representado con un “0”.
4-Servidor: Se podrá conectar a través de la dirección ip para su uso con el robot.

6. Implementación

6.1. Fundamentos de Proyectiles



Necesitamos tener 3 variables en este caso el ángulo del golpe, el tiempo de vuelo y la distancia a la que queremos enviar la pelota. Con estas 3 variables y la siguiente fórmula calculamos la velocidad inicial que necesita el brazo del robot para ejecutar el golpe y alcanzar la distancia solicitada. Cabe recalcar que definimos el ángulo por defecto

Los datos obtenidos son:

g=9.8 m/s^2, θ= 45°, y0=0.02 m, yf= 0m, xf=1.64 m, tv= 1.13s

$$v_0 = \sqrt{\frac{g\Delta x}{\sin(2\theta)}}$$

M.R.U (Eje horizontal)

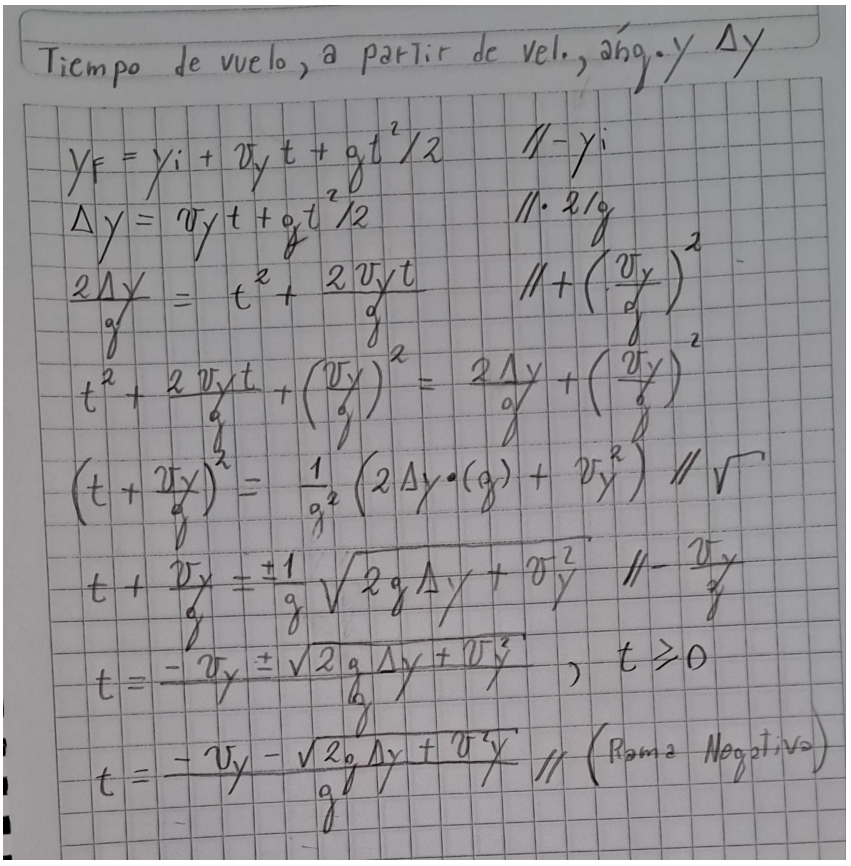
x(t)=x0+v0x∗t
1.64=0+v0*cos(45)*1.13
v0=1.64/(cos(45)*1.13)
v0=2.7627 m/s

Valor v0 para los tiros aprox ≈

M.R.U.A (Eje vertical)

$y(t)=y0+v0y*t+1/2*g*t^2$

Podemos calcular la distancia final con un ángulo dado y viendo el tiempo de vuelo.



Calculo Velocidad Inicial

$$d = \frac{-v_x\left(v_y + \sqrt{2g\Delta y + v_y^2}\right)}{g} \quad // \cdot g$$

$$dg = -v_x v_y - v_x\sqrt{2g\Delta y + v_y^2} \quad // + v_x v_y$$

$$dg + v_x v_y = -v_x\sqrt{2g\Delta y + v_y^2} \quad // ()^2$$

$$(dg)^2 + 2dg\,v_x v_y + (v_x v_y)^2 = v_x^2(2g\Delta y + v_y^2)$$

$$(dg)^2 + 2dg\,v_x v_y + (v_x v_y)^2 = v_x^2 v_y^2 + v_x^2 \cdot 2g\Delta y$$

$$// - v_x^2 v_y^2$$

$$(dg)^2 + 2dg\,v_i\cos(\theta)\,v_i\sin\theta = v_i^2\cos^2\theta \cdot 2g\Delta y$$

$$// - v_i^2\cos^2\theta \cdot 2g\Delta y - (dg)^2$$

$$2dg\,v_i^2\cos\theta\sin\theta - 2g\Delta y\,v_i^2\cos^2\theta = -(dg)^2$$

$$2v_i^2 g\cos\theta\left(d\sin\theta - \Delta y\cos\theta\right) = -(dg)^2$$

$$v_i^2 = \frac{-d^2g^2}{2g\cos\theta(d\sin\theta - \Delta y\cos\theta)} \quad // \sqrt{\cdot}$$

$$v_i = \sqrt{\frac{-\Delta x^2 g}{2\cos\theta(\Delta x\sin\theta - \Delta y\cos\theta)}}$$

Distancia Recorrida

$$x_f = x_i + v_x \cdot t \quad // - x_i$$

$$\Delta x = v_x \cdot t = d \quad // \text{Reemplazando}$$

$$d = \frac{v_x\left(-v_y - \sqrt{2g\Delta y + v_y^2}\right)}{g} \quad //$$

Ejemplo:

$\theta = 45°$, $y_f = 0$, $y_i = 0{,}05\,m$, $g = -9{,}8\,m/s^2$

$v_i = 20\,m/s$

$$d[m] = \frac{-20\cos(45°)\left[20\sin(45°) + \sqrt{-9{,}8\cdot 2\cdot(-0{,}05) + (20\sin 45°)^2}\right]}{-9{,}8}$$

$$d = 40{,}86\,m$$

6.2. Descripción de los Programas

Cliente socket

```python
#!/usr/bin/env python3
import threading
import tkinter              as     tk
from   tkinter              import ttk
from   ttkthemes            import ThemedTk
from   PIL                  import Image, ImageTk
from   Gamepad_Process_Pipe import Gamepad, get_gamepads
from   Robot_Client         import Robot
from   functools            import wraps

def hilo(func):
    @wraps(func)
    def run(*args, **kwargs):
        t = threading.Thread(target=func, args=args, kwargs=kwargs, daemon=True)
        t.start()
        return t
    return run

def dict_velocidad(parent: tk.Widget) -> (dict[str, dict[str, ttk.Widget]], tk.IntVar, tk.IntVar):
    mov_sli_val = tk.DoubleVar(value=50)
    gol_sli_val = tk.DoubleVar(value=50)
    mov_lab_val = tk.IntVar(value=int(mov_sli_val.get()))
    gol_lab_val = tk.IntVar(value=int(gol_sli_val.get()))

    """Cambia el valor del label (entero) dependiendo del valor del slider (double)"""
    mov_sli_val.trace_add("write", lambda x,y,z:mov_lab_val.set(int(mov_sli_val.get())))
    gol_sli_val.trace_add("write", lambda x,y,z:gol_lab_val.set(int(gol_sli_val.get())))

    return {
        'movimiento': {
            'label'      : ttk.Label(parent, text="Velocidad"),
            'label_value' : ttk.Label(parent, textvariable=mov_lab_val),
            'slider'     : ttk.Scale(parent, from_=0, to=100, variable=mov_sli_val, orient=tk.HORIZONTAL)
            #,'value'       : mov_lab_val
            },
        'golpe'    : {
            'label'      : ttk.Label(parent, text="Golpe"),
            'label_value' : ttk.Label(parent, textvariable=gol_lab_val),
            'slider'     : ttk.Scale(parent, from_=0, to=100, variable=gol_sli_val, orient=tk.HORIZONTAL)
            #,'value'       : gol_lab_val
            }
        }, mov_lab_val, gol_lab_val
    # Si quisiera modificar el valor de otras formas, tendría que reemplazar el x_lab_val con x_sli_val
def dict_control(parent: tk.Widget) -> dict[str, ttk.Button]:
    return {
            'up'    : ttk.Button(parent, text="W"),
            'left'  : ttk.Button(parent, text="A"),
            'down'  : ttk.Button(parent, text="S"),
            'right' : ttk.Button(parent, text="D"),
            'center': ttk.Button(parent, text="0")
        }
```

```python
def dict_mando(parent: tk.Widget) -> dict[str, ttk.Widget]:
    return {
            'label' : ttk.Label(parent, text="Mando:"),
            'combo' : ttk.Combobox(parent, values=None, state="readonly"),
            'button': ttk.Button(parent, text="Seleccionar")
    }
def dict_mecánica(parent: tk.Widget) -> dict[str, ttk.Widget]:
    return {
            'label' : ttk.Label(parent, text="Metros:"),
            'entry' : ttk.Entry(parent),
            'button': ttk.Button(parent, text="⎚")
    }
def dict_ip(parent: tk.Widget) -> dict[str, ttk.Widget]:
    return {
            'label' : ttk.Label(parent, text="IP"),
            'entry' : ttk.Entry(parent),
            'button': ttk.Button(parent, text="⎚")
    }

class Interfaz():
    def __init__(self, ventana_principal: tk.Tk):
        #Propiedades de la Ventana
        ventana_principal.geometry("600x400")
        ventana_principal.title("Tigerbot")

        #Variables del objeto
        self.ventana:   ThemedTk      = ventana_principal
        self.logo:      tk.PhotoImage = ImageTk.PhotoImage(Image.open("tigerbot.png").resize((100, 100)))

        #Escena
        self.root:      ttk.Frame = None

        #Control Robot
        self.robot:     Robot     = None

        #Diccionarios
        self.velocidad: dict = None
        self.control:   dict = None
        self.mando:     dict = None
        self.ip:        dict = None

        #Mando
        self.manilla_actual:      Gamepad = None
        self.manilla_actual_keys: dict    = None

        # #Styles
        # ttk.Style().configure("Y.TFrame", background = "yellow")
        # ttk.Style().configure("P.TFrame", background = "pink")
        # ttk.Style().configure("C.TFrame", background = "cyan")
        # ttk.Style().configure("G.TFrame", background = "green")
```

```
        self.crear_gui()
        self.asignar_eventos()
        self.entrada_pipe()

    def crear_marco(self):
        """Crea o recrea un marco que ocupa toda la ventana"""
        if self.root:
            self.root.destroy()
        self.root = ttk.Frame(self.ventana)
        self.root.pack(fill=tk.BOTH, expand=True)

    def crear_gui(self):
        def crear_top(parent: ttk.Frame) -> ttk.Frame:
            top_frame = ttk.Frame(parent)
            top_frame.rowconfigure(0, weight=1)

            top_frame.columnconfigure(0, weight=1)
            velocidad_frame = ttk.Frame(top_frame)
            velocidad_frame.grid(row=0, column=0, sticky=tk.NSEW)

            velocidad, vel_mov, vel_gol = dict_velocidad(velocidad_frame)

            #Posiciona y les da espacio a cada widget en el diccionario
            for row, vel_dict in enumerate(velocidad.values()):
                velocidad_frame.rowconfigure(row, weight=1) #Comenta para remover el reparto equitativo de espacio entre las filas
                for column, widget in enumerate(vel_dict.values()):
                    widget: ttk.Widget
                    if row == 0:
                        if column == 0:
                            velocidad_frame.columnconfigure(column, weight=1)
                            widget.grid(row=row, column=column, sticky=tk.SW, padx=(5, 0))
                        else:
                            widget.grid(row=row, column=column, sticky=tk.SE)
                    else:
                        if column == 0:
                            velocidad_frame.columnconfigure(column, weight=1)
                            widget.grid(row=row, column=column, sticky=tk.NW, padx=(5,0))
                        else:
                            widget.grid(row=row, column=column, sticky=tk.NE)


            #Agrega las claves después de poner los widgets
            velocidad['movimiento'].update({'value': vel_mov})
            velocidad['golpe'].update({'value': vel_gol})

            ttk.Label(top_frame, image=self.logo).grid(row=0, column=1, sticky=tk.NE)

            self.velocidad = velocidad

            return top_frame
```

```
        def crear_middle(parent):
            middle_frame = ttk.Frame(parent)
            middle_frame.rowconfigure(0, weight=1)

            middle_frame.columnconfigure((0, 1), weight=1)
            left_frame = ttk.Frame(middle_frame)
            left_frame.grid(row=0, column=0, sticky=tk.NSEW)
            right_frame = ttk.Frame(middle_frame)
            right_frame.grid(row=0, column=1, sticky=tk.NSEW)

            #Middle-Left
            control = dict_control(left_frame)
            left_frame.rowconfigure((0, 1, 2), weight=1)
            left_frame.columnconfigure((0, 1, 2), weight=1)

            control['up'].grid(row=0, column=1, sticky=tk.NSEW, padx=5, pady=(5, 2.5))
            control['left'].grid(row=1, column=0, sticky=tk.NSEW, padx=(5, 2.5), pady=5)
            control['down'].grid(row=2, column=1, sticky=tk.NSEW, padx=5, pady=(2.5, 5))
            control['right'].grid(row=1, column=2, sticky=tk.NSEW, padx=(2.5, 5), pady=5)
            control['center'].grid(row=1, column=1, sticky=tk.NSEW, padx=2.5, pady=2.5, ipadx=2.5, ipady=2.5)

            #Middle-Right
            right_frame.columnconfigure(1, weight=1)
            right_frame.config(padding=(5,5,5,5))
            mando = dict_mando(right_frame)
            mando['label'].grid(row=0, column=0, sticky=tk.NSEW, padx=(0,2))
            mando['combo'].grid(row=0, column=1, sticky=tk.NSEW, padx=(2,2))
            mando['button'].grid(row=0, column=2, sticky=tk.NSEW, padx=(2,0))

            #Columna intermedia de tamaño 5
            right_frame.rowconfigure(1, minsize=5)
            #Columna que ocupa todo el espacio libre
            right_frame.rowconfigure(3, weight=1)
            mecánica = dict_mecánica(right_frame)
            mecánica['label'].grid(row=2, column=0, sticky=tk.NSEW, padx=(0,2))
            mecánica['entry'].grid(row=2, column=1, sticky=tk.NSEW, padx=(2,2))
            mecánica['button'].grid(row=2, column=2, sticky=tk.NSEW, padx=(2,0))

            self.control = control
            self.mando   = mando
            return middle_frame
        def crear_bottom(parent):
            bottom_frame = ttk.Frame(parent)
            bottom_frame.rowconfigure(0, weight=1)

            ip = dict_ip(bottom_frame)
            for column, widget in enumerate(ip.values()):
                if column == 0:
                    widget.grid(row=0, column=column, sticky=tk.W, padx=5, pady=5)
```

```
                if column == 1:
                    bottom_frame.columnconfigure(column, weight=1)
                    widget.grid(row=0, column=column, sticky=tk.EW, pady=5)
                else:
                    widget.grid(row=0, column=column, sticky=tk.E, padx=5, pady=5)

            self.ip = ip
            return bottom_frame

        self.crear_marco()

        cuerpo = {
            'top'   : crear_top,
            'middle': crear_middle,
            'bottom': crear_bottom
        }
        #self.root.configure(padding=(5,5,5,5))
        self.root.columnconfigure(0, weight=1)
        for index, frame_func in enumerate(cuerpo.values()):
            #self.root.rowconfigure(index, weight=1)
            if index == 1:
                self.root.rowconfigure(index, weight=1)
            frame_func(self.root).grid(row=index, column=0, sticky=tk.NSEW)

    def asignar_eventos(self):
        def click():
            for dev in (dev for dev in get_gamepads() if self.mando['combo'].get() == dev.name):
                #TODO: Permitir usar manillas distintas con el mismo nombre.
                if self.manilla_actual is not None and self.manilla_actual.gamepad != dev:
                    self.manilla_actual.stop()
                    self.manilla_actual = None

                if self.manilla_actual is None:
                    self.manilla_actual = (Gamepad(dev))
                    self.manilla_actual.start()
                    self.manilla_actual_keys = tuple(name for name, value in vars(self.manilla_actual).items() if isinstance(value, property))
                    print(dev.name)
                else:
                    print("Selected the same gamepad")
        @hilo
        def crear_robot():
            try:
                self.robot = Robot(self.ip['entry'].get())
                print("Conexión exitosa")
            except:
                self.ip['entry'].delete(0, tk.END)
                self.ip['entry'].insert(0, "Error de Conexión")
                print("Error de Conexión")
```

```python
        #Mando
        combo_refresh = lambda _ : self.mando['combo'].config(values=tuple(device.name for device in get_gamepads()))
        self.mando['combo'].bind("<Button-1>", combo_refresh)
        self.mando['combo'].bind("<Down>", combo_refresh)
        self.mando['button'].config(command=click)

        #Botones
        def button_command(widget:ttk.Widget, press_command, release_command):
            #Click
            widget.bind('<ButtonPress>' , press_command)
            widget.bind('<ButtonRelease>'    , release_command)
            #Enter
            widget.bind('<Return>'      , press_command)
            widget.bind('<KeyRelease-Return>', release_command)
            #Barra Espaciadora
            widget.bind('<space>'       , press_command)
            widget.bind('<KeyRelease-space>' , release_command)

        for name, widget in self.control.items():
            widget:ttk.Button
            if name == "up":
                button_command(widget,
                lambda _: self.robot.avanzar(self.velocidad['movimiento']['value'].get()),
                lambda _: self.robot.avanzar(0))
            elif name == "left":
                button_command(widget,
                lambda _: self.robot.izquierda(int(self.velocidad['movimiento']['value'].get()/3)),
                lambda _: self.robot.izquierda(0))
            elif name == "down":
                button_command(widget,
                lambda _: self.robot.retroceder(self.velocidad['movimiento']['value'].get()),
                lambda _: self.robot.retroceder(0))
            elif name == "right":
                button_command(widget,
                lambda _: self.robot.derecha(int(self.velocidad['movimiento']['value'].get()/3)),
                lambda _: self.robot.derecha(0))
            elif name == "center":
                button_command(widget,
                lambda _: self.robot.atacar(self.velocidad['golpe']['value'].get()),
                lambda _: self.robot.atacar(0))

        #IP
        # #192.168.240.196
        self.ip['entry'].insert(0, "192.168.240.196")
        self.ip['entry'].bind('<Return>', lambda  : crear_robot())
        self.ip['button'].config(command=crear_robot)
        self.ip['button'].bind('<Return>', lambda _: crear_robot())

```

```python
    @hilo
    def entrada_pipe(self):
        velocidad_mov_var = self.velocidad['movimiento']['value']
        velocidad_gol_var = self.velocidad['golpe']['value']
        #TODO: Servicio que compruebe si el robot se ha desconectado
        while True:
            try:
                if self.manilla_actual is not None:
                    self.manilla_actual.exception()
                    if self.manilla_actual._output.poll():
                        key, v = self.manilla_actual._output.recv()
                        vel_mov: int = velocidad_mov_var.get()
                        vel_gol: int = velocidad_gol_var.get()
                        if self.robot is not None:
                            match key:
                                case "cruceta_H":
                                    self.robot.izquierda(-v*int(vel_mov/3))
                                case "cruceta_V":
                                    self.robot.retroceder(v*vel_mov)
                                case "botón_A":
                                    self.robot.retroceder(v*vel_mov)
                                case "botón_B":
                                    self.robot.derecha(v*int(vel_mov/3))
                                case "botón_X":
                                    self.robot.izquierda(v*int(vel_mov/3))
                                case "botón_Y":
                                    self.robot.avanzar(v*vel_mov)
                                case "LB":
                                    self.robot.atacar(-v*vel_gol)
                                case "RB":
                                    self.robot.atacar(v*vel_gol)
            except OSError as e:
                if str(e) == '[Errno 19] No such device':
                    print("Manilla Desconectada")
                    self.mando['combo'].set("")
                    self.manilla_actual = None
                elif str(e) == '[Errno 32] Broken pipe':
                    print("Robot desconectado")
                    self.robot = None
            except Exception as e:
                print(f"Caught Exception {e}")


if __name__ == "__main__":
    ventana = ThemedTk(theme="plastik")
    tigerbot = Interfaz(ventana)
    ventana.mainloop()
```

Funciones y conexión

```
        #IP
        # #192.168.240.196
        self.ip['entry'].insert(0, "192.168.240.196")
        self.ip['entry'].bind('<Return>', lambda _: crear_robot())
        self.ip['button'].config(command=crear_robot)
        self.ip['button'].bind('<Return>', lambda _: crear_robot())
```

```
        @hilo
        def crear_robot():
            try:
                self.robot = Robot(self.ip['entry'].get())
                print("Conexión exitosa")
            except:
                self.ip['entry'].delete(0, tk.END)
                self.ip['entry'].insert(0, "Error de Conexión")
                print("Error de Conexión")
```

```
import socket

PORT    = 2334
SIZE    = 32

class Robot:

    def __init__(self, HOST):
        try:
            self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.s.connect((HOST, 2334))
        except Exception as e:
            if self.s is not None:
                self.s.close()
            raise e

    def avanzar(self, velocidad):
        self.s.send(bytes(f'{f"avanzar {velocidad}":{SIZE}}', "utf-8"))

    def retroceder(self, velocidad):
        self.s.send(bytes(f'{f"retroceder {velocidad}":{SIZE}}', "utf-8"))

    def virar_derecha(self):
        self.s.send(bytes(f'{"virar_derecha":{SIZE}}', "utf-8"))

    def virar_izquierda(self):
        self.s.send(bytes(f'{"virar_izquierda":{SIZE}}', "utf-8"))

    def izquierda(self, velocidad=10):
        self.s.send(bytes(f'{f"izquierda {velocidad}":{SIZE}}', "utf-8"))

    def derecha(self, velocidad=10):
        self.s.send(bytes(f'{f"derecha {velocidad}":{SIZE}}', "utf-8"))

    def atacar(self, velocidad):
        self.s.send(bytes(f'{f"atacar {velocidad}":{SIZE}}', "utf-8"))

    def perdonar(self):
        self.s.send(bytes(f'{"perdonar":{SIZE}}', "utf-8"))

    def apagar(self):
        self.s.send(bytes(f'{"apagar":{SIZE}}', "utf-8"))

    def detener(self):
        self.s.send(bytes(f'{"detener":{SIZE}}', "utf-8"))
```

Server

```python
if __name__ == "__main__":
    print("Inicializando servidor...")
    robot = Robot()
    exit  = False
    while True:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            s.bind((HOST, PORT))
            s.listen(5)
            while True:
                try:
                    print("Esperando Cliente...")
                    clientsocket, address = s.accept()
                    with clientsocket:
                        print("Conectado a {}".format(address))
                        data = None
                        key  = None
                        while True:
                            data = clientsocket.recv(SIZE).decode("utf-8").split(" ")
                            key = data[0]
                            if data[1] != "":
                                value = int(data[1])

                            if key == "apagar":
                                robot.apagar()
                            elif key == "atacar":
                                robot.atacar(value)
                            elif key == "avanzar":
                                robot.avanzar(value)
                            elif key == "derecha":
                                robot.derecha(value)
                            elif key == "detener":
                                robot.detener()
                            elif key == "izquierda":
                                robot.izquierda(value)
                            elif key == "perdornar":
                                robot.perdonar()
                            elif key == "retroceder":
                                robot.retroceder(value)
                            else:
                                print("Clave desconocida: {}".format(data[0]))
                                print("Desconectando...")
                                break; #Aceptar Nuevo Cliente
                except socket.error:
                    print(str(e))
                    print("Cliente desconectado")
                except KeyboardInterrupt:
                    exit = True
                    break;
```

```python
                except Exception as e:
                    print("Error")
                    print(str(e))
                    exit = False
                    break; #Reiniciar
                robot.apagar()
        robot.apagar()
        if exit:
            print("\nSaliendo...")
            break; #Salir
        else:
            print("Reiniciando...")
```

Funciones server

```python
#!/usr/bin/env python3
from ev3dev2.motor      import MoveTank, LargeMotor, MediumMotor, OUTPUT_A, OUTPUT_B, OUTPUT_C, OUTPUT_D
import socket
import threading

HOST    = ''
PORT    = 2334
SIZE    = 32

class Robot:

    def __init__(self):
        self.ruedas          = MoveTank(OUTPUT_C, OUTPUT_A)
        self.viraje          = MediumMotor(OUTPUT_D)
        self.brazo           = LargeMotor(OUTPUT_B)

        self.service         = threading.Thread(target=self.daemon, args=(), daemon=True).start()

    def avanzar(self, velocidad=10):
        self.ruedas.on(left_speed=-velocidad, right_speed=-velocidad)

    def retroceder(self, velocidad=10):
        self.ruedas.on(left_speed=velocidad, right_speed=velocidad)

    def izquierda(self, velocidad=10):
        self.viraje.on(speed=velocidad)

    def derecha(self, velocidad=10):
        self.viraje.on(speed=-velocidad)

    def atacar(self, velocidad=10):
        self.brazo.on(speed=velocidad)

    def perdonar(self):
        self.brazo.off()

    def apagar(self):
        self.ruedas.off()
        self.brazo.off()
        self.viraje.off()

    def detener(self):
        self.ruedas.off()

    def daemon(self):
        while (True):
            if(self.viraje.is_stalled): #Destraba las ruedas
                self.viraje.reset()
```

7. Resultados

7.1. Estado actual del Proyecto

Actualmente nuestro proyecto cuenta con:

- ➔ La wiki del proyecto
- ➔ Informe avance formulado
- ➔ La conexión vía remota robot
- ➔ Carta Gantt actualizada
- ➔ Bitácoras subidas (durante paro trabajamos en robot y seguimos subiendo bitácoras)
- ➔ La interfaz gráfica de movimientos y control del robot la cual fue programada con la librería “tkinter”
- ➔ Servidor de conexión del robot con interfaz gráfica implementada con librería de python “socket”
- ➔ Funciones de movimiento y desplazamiento del robot además del golpe de la pelota de golf con el brazo del robot
- ➔ Estructura base apoyo pelota golpe de golf
- ➔ Control movimientos robot a través de mando tanto xbox como playstation

7.2. Problemas Encontrados y Solución Propuesta

Problemas	**Soluciones**
Tkinter entrega incorrectamente eventos de presión y soltado de tecla.	No usar el teclado para el control del robot.
Informe mal formulado	Ver la corrección del informe y arreglar errores
Dificultad para calcular movimiento parabólico	Repasar cursos pasados de mecánica clásica
Dificultad llevar movimiento parabólico a la práctica	Hicimos una base de lego para pelota golpe golf, además de utilizar una pelota de ping pong
Calcular correctamente la estimación de costos planificación de recursos	Hacer una separación entre costo de empleados y costo de software
Reorganizar la carta gantt después del paro	Añadir paro a la carta gantt y omitir semanas paradas, además de actualizar tareas y nuevas fechas

8. Pruebas

8.1 Descripción de las pruebas realizadas

1. En un primer momento realizamos el golpe de golf con la pelota “base” la cual era bastante pesada
2. El segundo intento fue ya con la plataforma hecha, la cual nos permite el movimiento parabólico pero de una manera bastante mínima en la cual con suerte se elevaba la pelota.
3. Al cambiar la pelota por una de ping pong y utilizar nuestra plataforma de lego, el movimiento parabólico fue completamente exitoso en la práctica

8.2 Resultados de las pruebas

1. Al ser la pelota “base” bastante pesada, nos dimos cuenta de que el movimiento parabólico no se llevaba a cabo en la práctica por lo que necesitábamos una plataforma para la pelota.
2. Al elevarse poco la pelota, nos dimos cuenta que necesitábamos una pelota más ligera que la pelota “base”, por lo que acudimos a una pelota más ligera en este caso una pelota de ping pong.
3. Ajustando las distintas potencias del brazo logramos hacer el tiro parabólico de golf solicitado, esta vez ya mucho más perfeccionado y controlado que en los intentos anteriores.

9. Conclusión

A lo largo del proyecto aprendimos muchas cosas, tanto en el trabajo en equipo como en la organización personal, como en la gestión de los tiempos, además de la programación del robot Lego ev3. Ha sido todo un proceso desde el día que empezamos hasta este día, nuestra entrega del proyecto final, la versión final de Tiger Bot en la cual hemos crecido tanto en nuestro aprendizaje académico como en nuestro crecimiento como personas. A lo largo del semestre como grupo hemos tenido una serie de altos y bajos los cuales hemos tenido que ir superando a medida que estos han ido ocurriendo, a pesar de planificar y gestionar y proveer las cosas siempre hay uno que otro imprevisto el cual nos ha forzado a tomar decisiones, impulsarnos en áreas que no eran de nuestro rol específico y tener flexibilidad ante cada situación que ha ocurrido. Agradecemos la oportunidad de tener un ramo como es Proyecto I en el que nos han forzado a trabajar en equipo, exponer y ponernos incómodos. Agradecemos por todos los aprendizajes llevados a cabos durante el semestre y por todos los aprendizajes que vendrán a lo largo de la carrera.