



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.....	4
1.1. Introducción.....	4
1.2. Objetivos.....	4
1.2.1. Objetivo General.....	4
1.2.2. Objetivo Especifico.....	5
1.3. Restricciones.....	5
1.4. Entregables.....	5
2. Organización del Personal.....	6
2.1. Descripción de los Roles.....	6
2.2. Personal que cumplirá los Roles.....	6
2.3. Mecanismos de Comunicación.....	7
3. Planificación del Proyecto.....	7
3.1. Actividades.....	7
3.2. Asignación de Tiempo.....	10
3.3. Gestión de Riesgos.....	11
4. Planificación de los Recursos.....	13
4.1. Hardware.....	13
4.2. Software.....	14
4.3. Costo de Empleados.....	14
4.4. Estimación de Costos.....	15
5. Análisis y Diseño.....	15
5.1. Especificación de Requerimiento.....	15
5.2. Arquitectura.....	16
5.3. Interfaz.....	17
6. Implementación.....	18
6.1. Fundamentos de Projectiles.....	18
6.2. Descripción de los Programas.....	20
6.3. Diagramas.....	26
7. Resultados.....	27
7.1. Estado actual del Proyecto.....	27
7.2. Problemas Encontrados y Solución Propuesta.....	27
8. Conclusión.....	28
9. Referencias.....	28

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. Personal que cumplirá los Roles

Rol	Persona asignada
Jefe de grupo	Diego López
Programador	Diego López y Sebastián Becerra
Ensamblador	Gustavo Morales
Diseñador	Diego López
Documentador	Sergio Huanca, Sebastian Becerra y Bryan Vega

2.3. 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

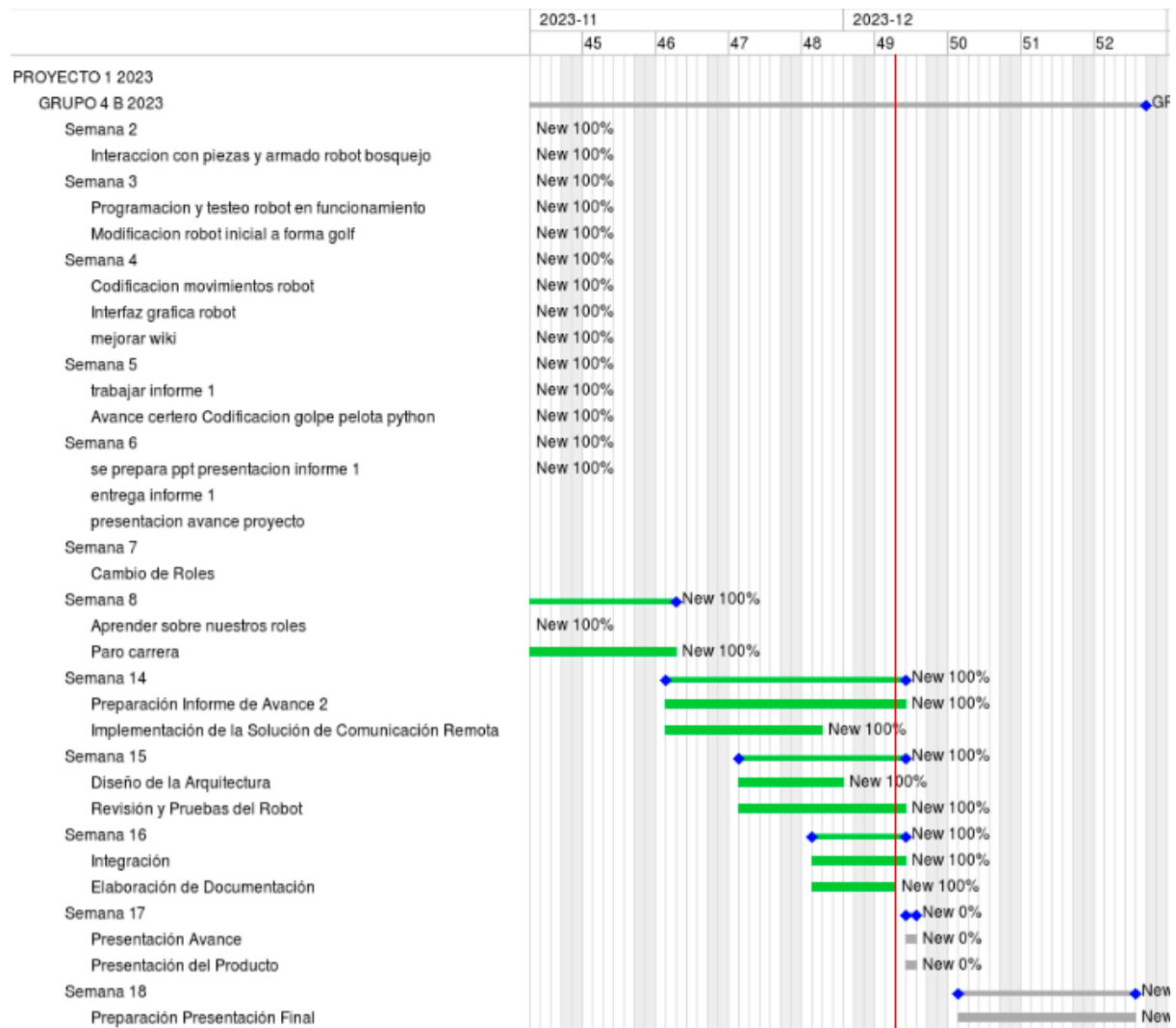
Administrar Redmine	Conocer la forma en la que se usa, sus funcionalidades y su aplicación	Sebastián Becerra
Informe avance	Añadir nuevos puntos del informe avance	Sebastián Becerra y Sergio Huanca
Estimación de costos	Calcular el presupuesto del proyecto	Sergio Huanca

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 conurrencia	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 (72 Horas Trabajadas)	\$288,000
Internet fibra óptica (Cantidad x mes trabajado)	\$24,000	4 c/u (4 Meses Trabajados)	\$96,000
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)			\$1,538,656

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 (16 semanas)			\$0

4.3. Costo de Empleados

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	72	\$1,152,000
Sebastián Becerra	\$14,500	72	\$1,044,000
Gustavo Morales	\$13,000	72	\$936,150
Bryan Vega	\$11,000	72	\$792,000
Sergio Huanca	\$10,000	72	\$720,000
Total (16 semanas)			\$4,644,150

4.4. Estimación de Costos

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.

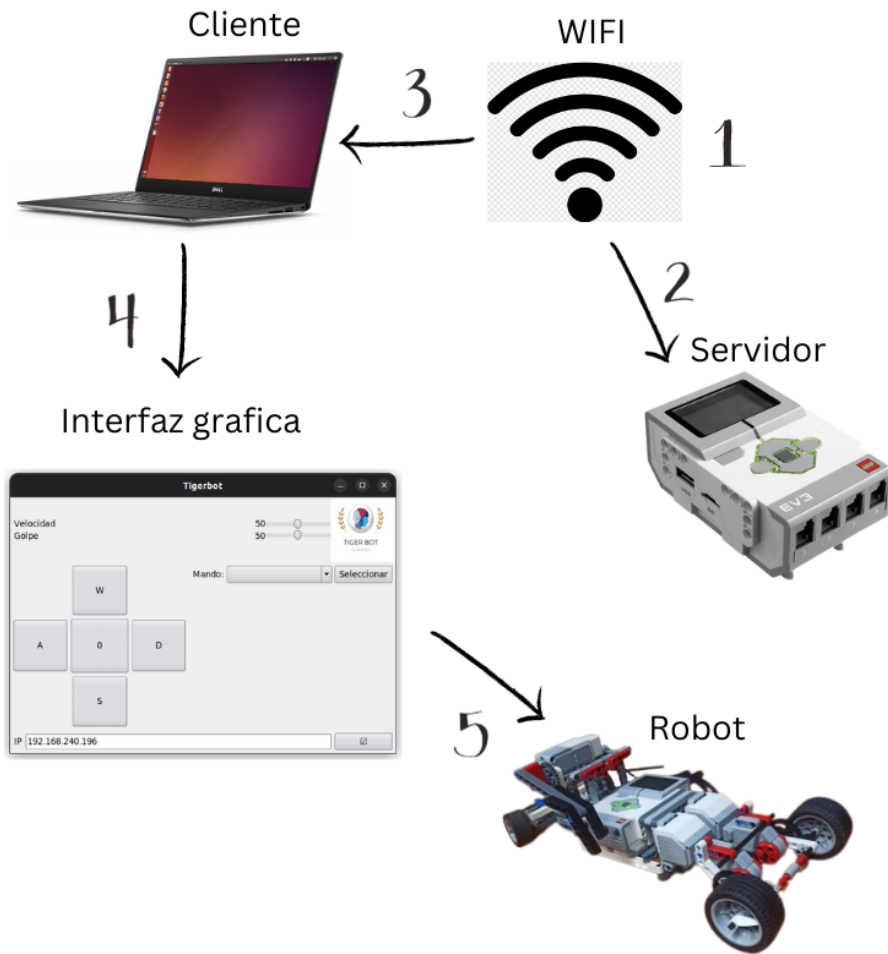
Total	Costo Total
Total Software	\$0
Total Hardware	\$1,538,656
Total Empleados	\$4,644,150
Total	\$6,182,806

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

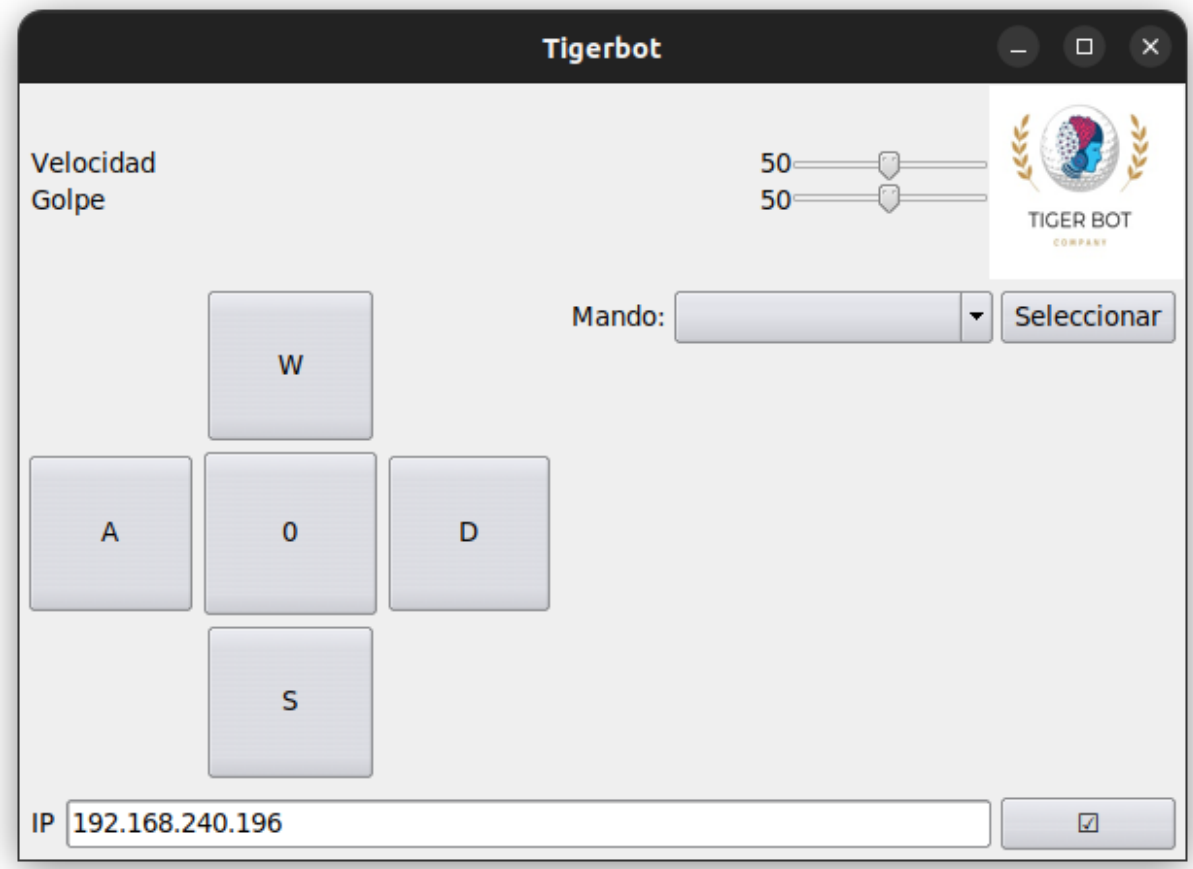
5.2. Arquitectura



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



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, virar a la izquierda y a la derecha. También contará con la opción de ajustar manualmente su velocidad.

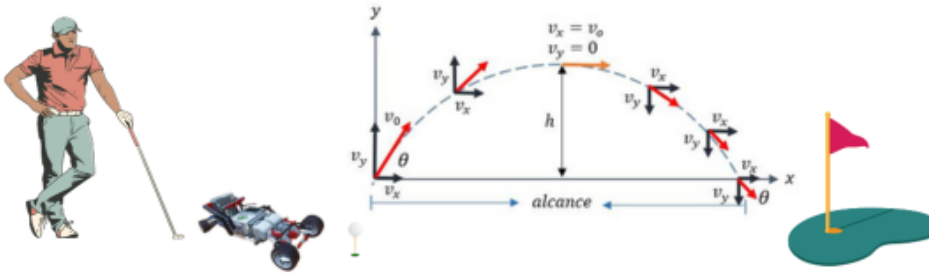
2-Conexión mando: Contará con conexión a mando para su movimiento

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 Projectiles



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:

$$g=9.8 \text{ m/s}^2, \theta= 45^\circ, y_0=0.02 \text{ m}, y_f= 0\text{m}, x_f=1.64 \text{ m}, t_v= 1.13\text{s}$$

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

M.R.U (Eje horizontal)

$$x(t)=x_0+v_0x*t$$

$$1.64=0+v_0*\cos(45)*1.13$$

$$v_0=1.64/(\cos(45)*1.13)$$

$$v_0=2.7627 \text{ m/s}$$

Valor v_0 para los tiros aprox \approx

M.R.U.A (Eje vertical)

$$y(t) = y_0 + v_0 t + \frac{1}{2} g t^2$$

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

6.2. Descripción de los Programas

Cliente socket

```
1 #!/usr/bin/env python3
2 import threading
3 import tkinter
4 from tkinter import *
5 from ttkthemes import ThemedTk
6 from PIL import Image, ImageTk
7 from Gamepad_Process_Pipe import Gamepad, get_gamepads
8 from Robot_Client import Robot
9 from functools import wraps
10
11 def hilo(func):
12     @wraps(func)
13     def run(*args, **kwargs):
14         t = threading.Thread(target=func, args=args, kwargs=kwargs, daemon=True)
15         t.start()
16         return t
17     return run
18
19 def dict_velocidad(parent: tk.Widget) -> (dict[str, dict[str, ttk.Widget]], tk.IntVar, tk.IntVar):
20     mov_sli_val = tk.DoubleVar(value=50)
21     gol_sli_val = tk.DoubleVar(value=50)
22     mov_lab_val = tk.IntVar(value=int(mov_sli_val.get()))
23     gol_lab_val = tk.IntVar(value=int(gol_sli_val.get()))
24
25     """Cambia el valor del label (entero) dependiendo del valor del slider (double)"""
26     mov_sli_val.trace_add("write", lambda x, y, z: mov_lab_val.set(int(mov_sli_val.get())))
27     gol_sli_val.trace_add("write", lambda x, y, z: gol_lab_val.set(int(gol_sli_val.get())))
28
29     return {
30         "movimiento": {
31             "label": tk.Label(parent, text="Velocidad"),
32             "label_value": tk.Label(parent, textvariable=mov_lab_val),
33             "slider": tk.Scale(parent, from_=0, to=100, variable=mov_sli_val, orient=tk.HORIZONTAL),
34             #, 'value': mov_lab_val
35         },
36         "golpe": {
37             "label": tk.Label(parent, text="Golpe"),
38             "label_value": tk.Label(parent, textvariable=gol_lab_val),
39             "slider": tk.Scale(parent, from_=0, to=100, variable=gol_sli_val, orient=tk.HORIZONTAL),
40             #, 'value': gol_lab_val
41         }
42     }, mov_lab_val, gol_lab_val
43     # Si quisiera modificar el valor de otras formas, tendría que reemplazar el x_lab_val con x_sli_val
44 def dict_control(parent: tk.Widget) -> dict[str, ttk.Button]:
45     return {
46         "up": tk.Button(parent, text="W"),
47         "left": tk.Button(parent, text="A"),
48         "down": tk.Button(parent, text="S"),
49         "right": tk.Button(parent, text="D"),
50         "center": tk.Button(parent, text="0")
51     }
52
53 def dict_mando(parent: tk.Widget) -> dict[str, ttk.Widget]:
54     return {
55         "label": tk.Label(parent, text="Mando:"),
56         "combo": tk.Combobox(parent, values=None, state="readonly"),
57         "button": tk.Button(parent, text="Seleccionar")
58     }
59 def dict_mecánica(parent: tk.Widget) -> dict[str, ttk.Widget]:
60     return {
61         "label": tk.Label(parent, text="Metros:"),
62         "entry": tk.Entry(parent),
63         "button": tk.Button(parent, text="e")
64     }
65 def dict_ip(parent: tk.Widget) -> dict[str, ttk.Widget]:
66     return {
67         "label": tk.Label(parent, text="IP:"),
68         "entry": tk.Entry(parent),
69         "button": tk.Button(parent, text="e")
70     }
71
72 class Interfaz():
73     def __init__(self, ventana_principal: tk.Tk):
74         #Propiedades de la Ventana
75         ventana_principal.geometry("600x400")
76         ventana_principal.title("Tigerbot")
77
78         #Variables del objeto
79         self.ventana: ThemedTk = ventana_principal
80         self.logo: tk.PhotoImage = ImageTk.PhotoImage(Image.open("tigerbot.png").resize((100, 100)))
81
82         #Escena
83         self.root: ttk.Frame = None
84
85         #Control Robot
86         self.robot: Robot = None
87
88         #Diccionarios
89         self.velocidad: dict = None
90         self.control: dict = None
91         self.mando: dict = None
92         self.ip: dict = None
93
94         #Mando
95         self.manilla_actual: Gamepad = None
96         self.manilla_actual_keys: dict = None
97
98         # #Styles
99         # ttk.Style().configure("Y.TFrame", background = "yellow")
100        # ttk.Style().configure("P.TFrame", background = "pink")
101        # ttk.Style().configure("C.TFrame", background = "cyan")
102        # ttk.Style().configure("G.TFrame", background = "green")
```

```

183     self.crear_gui()
184     self.asignar_eventos()
185     self.entrada_pipe()
186
187 def crear_marco(self):
188     """Crea o recrea un marco que ocupa toda la ventana"""
189     if self.root:
190         self.root.destroy()
191     self.root = ttk.Frame(self.ventana)
192     self.root.pack(fill=tk.BOTH, expand=True)
193
194 def crear_gui(self):
195     def crear_top(parent: ttk.Frame) -> ttk.Frame:
196         top_frame = ttk.Frame(parent)
197         top_frame.rowconfigure(0, weight=1)
198
199         top_frame.columnconfigure(0, weight=1)
200         velocidad_frame = ttk.Frame(top_frame)
201         velocidad_frame.grid(row=0, column=0, sticky=tk.NSEW)
202
203         velocidad, vel_mov, vel_gol = dict_velocidad(velocidad_frame)
204
205         #Posiciona y les da espacio a cada widget en el diccionario
206         for row, vel_dict in enumerate(velocidad.values()):
207             velocidad_frame.rowconfigure(row, weight=1) #Comenta para remover el reparto equitativo de espacio entre las filas
208             for column, widget in enumerate(vel_dict.values()):
209                 widget = ttk.Widget
210                 if row == 0:
211                     if column == 0:
212                         velocidad_frame.columnconfigure(column, weight=1)
213                         widget.grid(row=row, column=column, sticky=tk.SW, padx=(5, 0))
214                     else:
215                         widget.grid(row=row, column=column, sticky=tk.SE)
216                 else:
217                     if column == 0:
218                         velocidad_frame.columnconfigure(column, weight=1)
219                         widget.grid(row=row, column=column, sticky=tk.NW, padx=(5, 0))
220                     else:
221                         widget.grid(row=row, column=column, sticky=tk.NE)
222
223         #Agrega las claves después de poner los widgets
224         velocidad['movimiento'].update({'value': vel_mov})
225         velocidad['golpe'].update({'value': vel_gol})
226
227         ttk.Label(top_frame, image=self.logo).grid(row=0, column=1, sticky=tk.NE)
228
229         self.velocidad = velocidad
230
231     return top_frame

```

```

232
233 def crear_middle(parent):
234     middle_frame = ttk.Frame(parent)
235     middle_frame.rowconfigure(0, weight=1)
236
237     middle_frame.columnconfigure((0, 1), weight=1)
238     left_frame = ttk.Frame(middle_frame)
239     left_frame.grid(row=0, column=0, sticky=tk.NSEW)
240     right_frame = ttk.Frame(middle_frame)
241     right_frame.grid(row=0, column=1, sticky=tk.NSEW)
242
243     #Middle-Left
244     control = dict_control(left_frame)
245     left_frame.rowconfigure((0, 1, 2), weight=1)
246     left_frame.columnconfigure((0, 1, 2), weight=1)
247
248     control['up'].grid(row=0, column=1, sticky=tk.NSEW, padx=5, pady=(5, 2.5))
249     control['left'].grid(row=1, column=0, sticky=tk.NSEW, padx=(5, 2.5), pady=5)
250     control['down'].grid(row=2, column=1, sticky=tk.NSEW, padx=5, pady=(2.5, 5))
251     control['right'].grid(row=1, column=2, sticky=tk.NSEW, padx=(2.5, 5), pady=5)
252     control['center'].grid(row=1, column=1, sticky=tk.NSEW, padx=2.5, pady=2.5, ipadx=2.5, ipady=2.5)
253
254     #Middle-Right
255     right_frame.columnconfigure(1, weight=1)
256     right_frame.config(padding=(5,5,5,5))
257     mando = dict_mando(right_frame)
258     mando['label'].grid(row=0, column=0, sticky=tk.NSEW, padx=(0,2))
259     mando['combo'].grid(row=0, column=1, sticky=tk.NSEW, padx=(0,2))
260     mando['button'].grid(row=0, column=2, sticky=tk.NSEW, padx=(2,0))
261
262     #Columna intermedia de tamaño 5
263     right_frame.rowconfigure(1, minsize=5)
264     #Columna que ocupa todo el espacio libre
265     right_frame.rowconfigure(3, weight=1)
266     mecánica = dict_mecánica(right_frame)
267     mecánica['label'].grid(row=2, column=0, sticky=tk.NSEW, padx=(0,2))
268     mecánica['entry'].grid(row=2, column=1, sticky=tk.NSEW, padx=(0,2))
269     mecánica['button'].grid(row=2, column=2, sticky=tk.NSEW, padx=(2,0))
270
271     self.control = control
272     self.mando = mando
273     return middle_frame
274
275 def crear_bottom(parent):
276     bottom_frame = ttk.Frame(parent)
277     bottom_frame.rowconfigure(0, weight=1)
278
279     ip = dict_ip(bottom_frame)
280     for column, widget in enumerate(ip.values()):
281         if column == 0:
282             widget.grid(row=0, column=column, sticky=tk.W, padx=5, pady=5)

```

```

283         if column == 1:
284             bottom_frame.columnconfigure(column, weight=1)
285             widget.grid(row=0, column=column, sticky=tk.EW, pady=5)
286         else:
287             widget.grid(row=0, column=column, sticky=tk.E, padx=5, pady=5)
288
289     self.ip = ip
290     return bottom_frame
291
292     self.crear_marco()
293
294     cuerpo = {
295         'top': crear_top,
296         'middle': crear_middle,
297         'bottom': crear_bottom
298     }
299     #self.root.configure(padding=(5, 5, 5, 5))
300     self.root.columnconfigure(0, weight=1)
301     for index, frame_func in enumerate(cuerpo.values()):
302         #self.root.rowconfigure(index, weight=1)
303         if index == 1:
304             self.root.rowconfigure(index, weight=1)
305             frame_func(self.root).grid(row=index, column=0, sticky=tk.NSEW)
306
307     def asignar_eventos(self):
308         def click():
309             for dev in (dev for dev in get_gamepads() if self.mando['combo'].get() == dev.name):
310                 #TODO: Permitir usar manillas distintas con el mismo nombre.
311                 if self.manilla_actual is not None and self.manilla_actual.gamepad != dev:
312                     self.manilla_actual.stop()
313                     self.manilla_actual = None
314
315                 if self.manilla_actual is None:
316                     self.manilla_actual = (Gamepad(dev))
317                     self.manilla_actual.start()
318                     self.manilla_actual_keys = tuple(name for name, value in vars(self.manilla_actual).items() if isinstance(value, property))
319                     print(dev.name)
320                 else:
321                     print("Selected the same gamepad")
322
323     @hilo
324     def crear_robot():
325         try:
326             self.robot = Robot(self.ip['entry'].get())
327             print("Conexión exitosa")
328         except:
329             self.ip['entry'].delete(0, tk.END)
330             self.ip['entry'].insert(0, "Error de Conexión")
331             print("Error de Conexión")

```

```

352 #Mando
353 combo_refresh = lambda : self.mando['combo'].config(values=tuple(device.name for device in get_gamepads()))
354 self.mando['combo'].bind("<Button-1>", combo_refresh)
355 self.mando['combo'].bind("<Down>", combo_refresh)
356 self.mando['button'].config(command=click)
357
358 #Botones
359 def button_command(widget:ttk.Widget, press_command, release_command):
360     #Click
361     widget.bind("<ButtonPress>", press_command)
362     widget.bind("<ButtonRelease>", release_command)
363     #Enter
364     widget.bind("<Return>", press_command)
365     widget.bind("<KeyRelease-Return>", release_command)
366     #Barra Espaciadora
367     widget.bind("<space>", press_command)
368     widget.bind("<KeyRelease-space>", release_command)
369
370 for name, widget in self.control.items():
371     widget:ttk.Button
372     if name == "up":
373         button_command(widget,
374             lambda : self.robot.avanzar(self.velocidad['movimiento']['value'].get()),
375             lambda : self.robot.avanzar(0))
376     elif name == "left":
377         button_command(widget,
378             lambda : self.robot.izquierda(int(self.velocidad['movimiento']['value'].get()/3)),
379             lambda : self.robot.izquierda(0))
380     elif name == "down":
381         button_command(widget,
382             lambda : self.robot.retroceder(self.velocidad['movimiento']['value'].get()),
383             lambda : self.robot.retroceder(0))
384     elif name == "right":
385         button_command(widget,
386             lambda : self.robot.derecha(int(self.velocidad['movimiento']['value'].get()/3)),
387             lambda : self.robot.derecha(0))
388     elif name == "center":
389         button_command(widget,
390             lambda : self.robot.atacar(self.velocidad['golpe']['value'].get()),
391             lambda : self.robot.atacar(0))
392
393 #IP
394 # #192.168.249.196
395 self.ip['entry'].insert(0, "192.168.249.196")
396 self.ip['entry'].bind("<Return>", lambda : crear_robot())
397 self.ip['button'].config(command=crear_robot)
398 self.ip['button'].bind("<Return>", lambda : crear_robot())
399

```

```

381 @hilo
382 def entrada_pipe(self):
383     velocidad_mov_var = self.velocidad['movimiento']['value']
384     velocidad_gol_var = self.velocidad['golpe']['value']
385     #robo: Servicio que compruebe si el robot se ha desconectado
386     while True:
387         try:
388             if self.manilla_actual is not None:
389                 self.manilla_actual.exception()
390             if self.manilla_actual.output.poll():
391                 key, v = self.manilla_actual.output.recv()
392                 vel_mov = int(velocidad_mov_var.get())
393                 vel_gol = int(velocidad_gol_var.get())
394                 if self.robot is not None:
395                     match key:
396                         case "cruceta H":
397                             self.robot.izquierda(-v*int(vel_mov/3))
398                         case "cruceta V":
399                             self.robot.retroceder(v*vel_mov)
400                         case "botón A":
401                             self.robot.retroceder(v*vel_mov)
402                         case "botón B":
403                             self.robot.derecha(v*int(vel_mov/3))
404                         case "botón X":
405                             self.robot.izquierda(v*int(vel_mov/3))
406                         case "botón Y":
407                             self.robot.avanzar(v*vel_mov)
408                         case "LB":
409                             self.robot.atacar(-v*vel_gol)
410                         case "RB":
411                             self.robot.atacar(v*vel_gol)
412         except OSError as e:
413             if str(e) == '[Errno 19] No such device':
414                 print("Manilla desconectada")
415                 self.mando['combo'].set("")
416                 self.manilla_actual = None
417             elif str(e) == '[Errno 32] Broken pipe':
418                 print("Robot desconectado")
419                 self.robot = None
420         except Exception as e:
421             print(f"caught Exception (e)")
422
423
424 if __name__ == "__main__":
425     ventana = ThemedTk(theme="plastik")
426     Tigerbot = Interfaz(ventana)
427     ventana.mainloop()

```

Funciones y conexión

```
233 #IP
234 # #192.168.240.196
235 self.ip['entry'].insert(0, "192.168.240.196")
236 self.ip['entry'].bind("<Return>", lambda : crear_robot())
237 self.ip['button'].config(command=crear_robot)
238 self.ip['button'].bind("<Return>", lambda : crear_robot())
```

```
242 @hello
243 def crear_robot():
244     try:
245         self.robot = Robot(self.ip['entry'].get())
246         print("Conexión exitosa")
247     except:
248         self.ip['entry'].delete(0, tk.END)
249         self.ip['entry'].insert(0, "Error de Conexión")
250         print("Error de Conexión")
```

```
1 import socket
2
3 PORT = 2334
4 SIZE = 32
5
6 class Robot:
7
8     def __init__(self, HOST):
9         try:
10             self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11             self.s.connect((HOST, 2334))
12         except Exception as e:
13             if self.s is not None:
14                 self.s.close()
15             raise e
16
17     def avanzar(self, velocidad):
18         self.s.send(bytes(f"avanzar {velocidad}:{SIZE}", "utf-8"))
19
20     def retroceder(self, velocidad):
21         self.s.send(bytes(f"retroceder {velocidad}:{SIZE}", "utf-8"))
22
23     def virar_derecha(self):
24         self.s.send(bytes(f"virar_derecha:{SIZE}", "utf-8"))
25
26     def virar_izquierda(self):
27         self.s.send(bytes(f"virar_izquierda:{SIZE}", "utf-8"))
28
29     def izquierda(self, velocidad=10):
30         self.s.send(bytes(f"izquierda {velocidad}:{SIZE}", "utf-8"))
31
32     def derecha(self, velocidad=10):
33         self.s.send(bytes(f"derecha {velocidad}:{SIZE}", "utf-8"))
34
35     def atacar(self, velocidad):
36         self.s.send(bytes(f"atacar {velocidad}:{SIZE}", "utf-8"))
37
38     def perdonar(self):
39         self.s.send(bytes(f"perdonar:{SIZE}", "utf-8"))
40
41     def apagar(self):
42         self.s.send(bytes(f"apagar:{SIZE}", "utf-8"))
43
44     def detener(self):
45         self.s.send(bytes(f"detener:{SIZE}", "utf-8"))
```


Server

```
50 if __name__ == "__main__":
51     print("Inicializando servidor...")
52     robot = Robot()
53     exit = False
54     while True:
55         with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
56             s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
57             s.bind((HOST, PORT))
58             s.listen(5)
59             while True:
60                 try:
61                     print("Esperando Cliente...")
62                     clientsocket, address = s.accept()
63                     with clientsocket:
64                         print("Conectado a {}".format(address))
65                         data = None
66                         key = None
67                         while True:
68                             data = clientsocket.recv(SIZE).decode("utf-8").split(" ")
69                             key = data[0]
70                             if data[1] != "":
71                                 value = int(data[1])
72
73                                 if key == "apagar":
74                                     robot.apagar()
75                                 elif key == "atacar":
76                                     robot.atacar(value)
77                                 elif key == "avanzar":
78                                     robot.avanzar(value)
79                                 elif key == "derecha":
80                                     robot.derecha(value)
81                                 elif key == "detener":
82                                     robot.detener()
83                                 elif key == "izquierda":
84                                     robot.izquierda(value)
85                                 elif key == "perdonar":
86                                     robot.perdonar()
87                                 elif key == "retroceder":
88                                     robot.retroceder(value)
89                                 else:
90                                     print("Clave desconocida: {}".format(data[0]))
91                                     print("Desconectando...")
92                                     break; #Aceptar Nuevo Cliente
93                             except socket.error:
94                                 print(str(e))
95                                 print("Cliente desconectado")
96                             except KeyboardInterrupt:
97                                 exit = True
98                                 break;
99
100     except Exception as e:
101         print("Error")
102         print(str(e))
103         exit = False
104         break; #Reiniciar
105     robot.apagar()
106     if exit:
107         print("¡Saliedo...")
108         break; #Salir
109     else:
110         print("Reiniciando...")
```

Funciones server

```
1 #!/usr/bin/env python3
2 from ev3dev2.motor import MoveTank, LargeMotor, MediumMotor, OUTPUT_A, OUTPUT_B, OUTPUT_C, OUTPUT_D
3 import socket
4 import threading
5
6 HOST = ''
7 PORT = 2334
8 SIZE = 32
9
10 class Robot:
11
12     def __init__(self):
13         self.ruedas = MoveTank(OUTPUT_C, OUTPUT_A)
14         self.viraje = MediumMotor(OUTPUT_D)
15         self.brazo = LargeMotor(OUTPUT_B)
16
17         self.service = threading.Thread(target=self.daemon, args=(), daemon=True).start()
18
19     def avanzar(self, velocidad=10):
20         self.ruedas.on(left_speed=velocidad, right_speed=velocidad)
21
22     def retroceder(self, velocidad=10):
23         self.ruedas.on(left_speed=-velocidad, right_speed=-velocidad)
24
25     def izquierda(self, velocidad=10):
26         self.viraje.on(speed=velocidad)
27
28     def derecha(self, velocidad=10):
29         self.viraje.on(speed=-velocidad)
30
31     def atacar(self, velocidad=10):
32         self.brazo.on(speed=velocidad)
33
34     def perdonar(self):
35         self.brazo.off()
36
37     def apagar(self):
38         self.ruedas.off()
39         self.brazo.off()
40         self.viraje.off()
41
42     def detener(self):
43         self.ruedas.off()
44
45     def daemon(self):
46         while (True):
47             if(self.viraje.is_stalled): #Destraba las ruedas
48                 self.viraje.reset()
49
```

6.3. Diagramas

OUTPUT	INPUT
A) Rueda derecha (Large Motor)	N/A
B) Brazo (Large Motor)	N/A
C) Rueda izquierda (Large Motor)	N/A
D) Viraje (Medium Motor)	N/A

Robot_Server NF
+ruedas: MoveTank +viraje: MediumMotor +brazo: LargeMotor
avanzar (velocidad=10) retroceder(velocidad=10) izquierda(velocidad=10) derecha(velocidad=10) atacar(velocidad=10) perdonar() apagar() detener()

Cliente

GUI2_ttk_socket.py

Robot_Client.py

Interfaz	Robot
GUI	Habla

Robot_Server_NF.py <------> Robot

Escucha e Interpreta

Ejecuta

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. Conclusión

En el transcurso de este proyecto, el equipo Tigerbot ha experimentado un significativo crecimiento, aprendiendo valiosas lecciones sobre trabajo en equipo, gestión del tiempo y resiliencia frente a desafíos. La adquisición de conocimientos teóricos y prácticos en programación, física y gestión personal ha sido fundamental para superar las fechas límite y alcanzar logros destacados en el desarrollo del robot. Agradecemos los progresos individuales y colectivos, y los invitamos a seguir nuestro viaje en lo que resta del semestre, esperando presentarles con entusiasmo la versión final de Tigerbot. ¡Gracias por acompañarnos!

9. Referencias

Sadurní, J. M. (2023, 11 marzo). Ole Kirk Christiansen, la vida del carpintero que creó El Lego, uno de los juegos más famosos del mundo. *historia.nationalgeographic.com.es*.

https://historia.nationalgeographic.com.es/a/ole-kirk-christiansen-el-creador-de-lego-el-juego-de-construccion-mas-famoso-del-mundo_17770

Negocios, E., & Negocios, E. (2022). La historia de LEGO: el ladrillo de plástico que se convirtió en el «juguete del siglo». *Ekos Negocios*.

<https://ekosnegocios.com/articulo/la-historia-de-lego-el-ladrillo-de-plastico-que-se-convirtio-en-el-juguete-del-siglo#:~:text=El%20origen%20hist%C3%B3rico%20de%20la,marcas%20%C3%A1s%20valiosas%20del%20mundo.>

