



UNIVERSIDAD DEL BÍO-BÍO, CHILE

FACULTAD DE CIENCIAS EMPRESARIALES

Departamento de Sistemas de Información

SIMULADOR LABORATORIO CIMUBB

PROYECTO DE TÍTULO PRESENTADO POR PATRICIO ANDRÉS LABRA MEDINA
DE LA CARRERA INGENIERÍA CIVIL INFORMÁTICA
DIRIGIDA POR MÓNICA CANIUPÁN MARILEO

2024

Resumen

La tesis tiene como objetivo abordar la necesidad identificada en el Laboratorio de Sistemas Automatizados de Producción, conocido como CIMUBB, donde se imparte formación en automatización mediante robots y computadoras. La limitación de acceso al laboratorio y la dificultad para impartir clases durante situaciones excepcionales, como la pandemia, han generado la necesidad de encontrar una solución que permita brindar enseñanza sin presencia física.

La propuesta en este informe, consiste en desarrollar un simulador en Unity, un motor de videojuegos, que reproduzca la experiencia de trabajar con los brazos robóticos Scrbot utilizados en el laboratorio. Este simulador permite a los estudiantes interactuar virtualmente con los brazos robóticos, realizando ejercicios prácticos, programación y pruebas, de manera similar a como lo harían en el laboratorio real. De esta forma, se busca preservar la calidad de la enseñanza y superar las limitaciones de acceso físico al laboratorio.

El enfoque del proyecto se basa en la recreación de los brazos robóticos Scrbot, incluyendo su funcionamiento y comportamiento. Se implementan diversas funcionalidades y herramientas que faciliten la comprensión y práctica de los conceptos de automatización, permitiendo que los estudiantes realicen actividades teóricas y prácticas desde cualquier lugar.

El resultado esperado es que este simulador se convierta en una valiosa herramienta para estudiantes y profesores del laboratorio CIMUBB, mejorando la calidad de la enseñanza en sistemas automatizados de producción y brindando la posibilidad de impartir cursos de forma virtual o complementaria a las clases presenciales.

Palabras Clave — Simulador, Brazos Robóticos Scrbot, Unity, Sistemas Automatizados de Producción, Enseñanza Virtual, Automatización, Robótica.

Abstract

The thesis aims to address the identified need in the Automated Production Systems Laboratory, known as CIMUBB, where training in automation using robots and computers is provided. The limited access to the laboratory and the difficulty of conducting classes during exceptional situations, such as the pandemic, have created the need to find a solution that allows for teaching without physical presence.

The proposal in this report is to develop a simulator in Unity, a game engine, that replicates the experience of working with Scrbot robotic arms used in the laboratory. This simulator allows students to interact virtually with the robotic arms, performing practical exercises, programming, and tests, similar to how they would in the real laboratory. In this way, the aim is to preserve the quality of teaching and overcome the limitations of physical access to the laboratory.

The project's approach is based on the recreation of Scrbot robotic arms, including their functionality and behavior. Various features and tools are implemented to facilitate the understanding and practice of automation concepts, allowing students to engage in theoretical and practical activities from anywhere.

The expected outcome is for this simulator to become a valuable tool for students and professors at the CIMUBB laboratory, improving the quality of teaching in automated production systems and providing the possibility of conducting courses virtually or as a complement to in-person classes.

Keywords — Simulator, Scrbot Robotic Arms, Unity, Automated Production Systems, Virtual Teaching, Automation, Robotics.

Índice general

1. Introducción	11
2. Justificación del Proyecto	13
2.1. El Proyecto	13
2.2. Otras Soluciones	14
2.2.1. Robocell de Intelitek	14
2.2.2. Robots Personales	16
3. Definición de la institución	17
3.1. Descripción de la institución	17
3.2. Descripción del área de estudio	18
3.3. Descripción de la problemática	19
4. Definición proyecto	21
4.1. Objetivos del proyecto	21
4.1.1. Objetivo General	21
4.1.2. Objetivos Específicos	21
4.1.3. Actividades	21
4.2. Ambiente de ingeniería de software	22
4.2.1. Metodología de Desarrollo	22
4.2.2. Tecnologías	22
4.2.3. Herramientas de apoyo	23
5. Especificación de requerimientos de software	24
5.1. Alcances	24
5.2. Objetivo del software	25
5.3. Descripción global del producto	25
5.3.1. Interfaz de usuario	25
5.3.2. Interfaz de hardware	26
5.3.3. Interfaz software	26
5.4. Requerimientos específicos	26
5.4.1. Requerimientos funcionales del sistema	26

6. Factibilidad	27
6.1. Factibilidad Técnica	27
6.2. Factibilidad Operativa	29
6.3. Impacto Positivo:	29
6.4. Impacto Negativo:	30
6.5. Factibilidad Económica	30
6.6. Conclusión de la factibilidad	32
7. Análisis	33
7.1. Interacción del Usuario en la Simulación	33
7.2. Casos de uso	34
7.2.1. Actores	34
7.2.2. Diagrama de casos de uso y descripción	34
7.2.3. Especificación de los caso de uso	37
8. Diseño	47
8.1. Diseño interfaz y navegación	47
9. Código	50
9.1. Lógica Cinta Transportadora	50
9.2. Lógica Brazos Robóticos	55
9.3. Lógica tomar objeto	62
10. Pruebas	64
10.1. Elementos de prueba	64
10.2. Detalle de las pruebas solicitadas por el desarrollador	64
10.3. Respuestas	65
10.4. Conclusiones de prueba del desarrollador	68
10.5. Prueba del profesor	68
11. Plan de capacitación y entrenamiento	69
11.1. Plan de capacitación	69
12. Plan de implantación y puesta en marcha	70
12.1. Plan de implantación	70
13. Trabajo Futuro	83
13.1. Cambios que se pueden realizar	83
14. Conclusiones	85
Referencias	86
A. Código: Belt.cs	87

B. Código: MovimientoJoints IX.cs	88
C. Código: MovimientoJoints V.cs	102
D. Código: MovimientoJoints Vz.cs	115
E. Código: TomarObjeto.cs	129
F. Código: CalidadImagen.cs	131
G. Código: Camaras.cs	132
H. Código: ControlDropdown.cs	134
I. Código: MenuPausa.cs	136
J. Código: PantallaCompleta.cs	138

Índice de Figuras

2.1. Software Robocell	14
2.2. Software Scrbase	15
2.3. Software Robocell y Scrbase	15
2.4. Robot Arduino	16
3.1. Edificio CIMUBB	17
3.2. Laboratorio CIMUBB	19
3.3. SCORBOT ER V Plus	19
3.4. SCORBOT ER IX	20
7.1. BPMN de la Interacción	33
7.2. Diagrama de caso de uso: Usuario	34
7.3. Diagrama de caso de uso: Usuario	35
7.4. Diagrama de caso de uso: Usuario	36
8.1. Mockup	47
8.2. Menu Principal Simulación	48
8.3. Interfaz dentro de la Simulación	49
9.1. rigidBody Position 1	52
9.2. rigidBody Position 2	53
9.3. rigidBody MovePosition 1	53
9.4. rigidBody MovePosition 2	54
10.1. Facilidad de uso de la botonera del robot	65
10.2. Facilidad de uso del cambio de cámara	66
10.3. Facilidad de uso del cambio de brazo robótico	66
10.4. Facilidad de uso del menu	67
10.5. Facilidad de uso del menu dentro de la simulación	68
12.1. Parte 1 Tutorial	71
12.2. Parte 2 Tutorial	72
12.3. Parte 3 Tutorial	72

12.4. Parte 4 Tutorial	73
12.5. Parte 5 Tutorial	73
12.6. Página de descarga WinRAR	74
12.7. Parte 1 Tutorial Windows	74
12.8. Parte 2 Tutorial Windows	75
12.9. Parte 3 Tutorial Windows	75
12.10Parte 4 Tutorial Windows	76
12.11Parte 5 Tutorial Windows	76
12.12Parte 6 Tutorial Windows	77
12.13Parte 7 Tutorial Windows	77
12.14Parte 1 Tutorial Linux	78
12.15Parte 2.1 Tutorial Linux	78
12.16Parte 2.2.1 Tutorial Linux	79
12.17Parte 2.2.2 Tutorial Linux	79
12.18Parte 2.2.3 Tutorial Linux	80
12.19Parte 3 Tutorial Linux	80
12.20Parte 4.1.1 Tutorial Linux	81
12.21Parte 4.1.2 Tutorial Linux	81
12.22Parte 4.2 Tutorial Linux	82
13.1. Diseño interfaz: Módulo Principal	84

Índice de Tablas

4.1. Tecnologías	22
4.2. Herramientas de apoyo	23
5.1. Requerimientos funcionales	26
6.1. Requerimientos de desarrollo en computadores	27
6.2. Costo de Desarrollo del Proyecto en Unity	30
6.3. Evaluación Financiera del Proyecto	31
7.1. Especificación de casos de uso: Pulsar Botón Entrar	37
7.2. Especificación de casos de uso: Pulsar Botón Reiniciar	37
7.3. Especificación de casos de uso: Desplegar lista para cambio de resolución	38
7.4. Especificación de casos de uso: Desplegar lista para cambio de calidad imagen	38
7.5. Especificación de casos de uso: Pulsar Interruptor Pantalla Completa	38
7.6. Especificación de casos de uso: Pulsar Botón Salir	39
7.7. Especificación de casos de uso: Pulsar Botón Pausar	39
7.8. Especificación de casos de uso: Desplegar lista para cambio de cámara	39
7.9. Especificación de casos de uso: Desplegar lista para cambio de brazo robótico	40
7.10. Especificación de casos de uso: Pulsar Interruptor Mostrar Control	40
7.11. Especificación de casos de uso: Pulsar Interruptor Activar Cinta	40
7.12. Especificación de casos de uso: Pulsar Botón Rotar Base Negativa	41
7.13. Especificación de casos de uso: Pulsar Botón Rotar Base Positiva	41
7.14. Especificación de casos de uso: Pulsar Botón Rotar Shoulder Negativa	42
7.15. Especificación de casos de uso: Pulsar Botón Rotar Shoulder Positiva	42
7.16. Especificación de casos de uso: Pulsar Botón Open-Close	43
7.17. Especificación de casos de uso: Pulsar Botón Rotar Elbow Negativa	43
7.18. Especificación de casos de uso: Pulsar Botón Rotar Elbow Positiva	44
7.19. Especificación de casos de uso: Pulsar Botón Rotar Pitch Negativa	44
7.20. Especificación de casos de uso: Pulsar Botón Rotar Pitch Positiva	45
7.21. Especificación de casos de uso: Pulsar Botón Rotar Roll Negativa	45
7.22. Especificación de casos de uso: Pulsar Botón Rotar Roll Positiva	46
10.1. Facilidad de uso de la botonera del robot	65

10.2. Facilidad de uso del cambio de cámara	65
10.3. Facilidad de uso del cambio de brazo robótico	66
10.4. Facilidad de uso del menu	67
10.5. Facilidad de uso del menu dentro de la simulación	67

Capítulo 1

Introducción

La pandemia de COVID-19, que comenzó a principios de 2020, tuvo un impacto significativo en diversos sectores, y la educación no fue la excepción [1]. Con la propagación del virus, muchas instituciones educativas se vieron obligadas a cerrar temporalmente y adoptar modalidades de enseñanza en línea para garantizar la seguridad de estudiantes y personal.

En el caso específico de la Universidad del Bío-Bío, la transición a la educación en línea generó desafíos, especialmente en disciplinas prácticas que requerían acceso a laboratorios y equipamiento especializado. El laboratorio de automatización CIMUBB, que se centra en la preparación de estudiantes para las nuevas tecnologías de automatización, se vio afectado por estas limitaciones.

El uso de robots, en este caso, los brazos robóticos SCORBOT, se vio restringido durante la cuarentena, ya que los estudiantes no podían acceder físicamente al laboratorio. Esto resultó en una disminución en la experiencia práctica y en la interacción directa con los sistemas automatizados. Los desafíos adicionales surgieron al depender de videoconferencias para la enseñanza, lo que limitó la efectividad de la instrucción práctica.

La imposibilidad de acceder al laboratorio durante la pandemia llevó al profesor Luis Vera a buscar soluciones alternativas para continuar con la enseñanza práctica. Fue en este contexto que se exploraron las tecnologías de simulación, destacando Unity como una herramienta versátil que permitiría recrear virtualmente el laboratorio de automatización CIMUBB y, específicamente, dar vida a los brazos robóticos SCORBOT.

La simulación propuesta no solo aborda la limitación impuesta por la pandemia, sino que también amplía las posibilidades de aprendizaje. Al utilizar elementos de videojuegos, se ofrece a los estudiantes la oportunidad de interactuar de manera práctica con los brazos robóticos, incluso permitiéndoles realizar experimentos y desarrollar habilidades de control directo.

Esta iniciativa representa una respuesta innovadora a los desafíos educativos presentados por la pandemia, destacando cómo las tecnologías de simulación y la creatividad pueden superar barreras físicas y brindar experiencias de aprendizaje significativas, incluso en contextos adversos como los generados por el COVID-19.

En este escrito, se cuenta con catorce capítulos, los cuales están ordenados de la siguiente manera:

- Capítulo 1 Introducción: Este capítulo sirve como punto de partida para el trabajo, brindando una visión general del tema y su relevancia. Se establecen los objetivos de la investigación y se presenta la estructura del documento, preparando al lector para adentrarse en el análisis y exploración del tema en cuestión.
- Capítulo 2 Justificación: Se exponen las razones y motivos detrás de la elección de este trabajo o proyecto en particular. Se describe la importancia del tema y su relevancia en el contexto actual, destacando las problemáticas o necesidades que aborda. Además se presentan soluciones alternativas para el problema.
- Capítulo 3 Definición de la institución: Se proporciona una descripción de la institución y de la problemática a tratar.
- Capítulo 4 Definición de proyecto: En este capítulo se establecen los objetivos, actividades, tecnologías y herramientas para el proyecto.
- Capítulo 5 Especificación de requerimientos de software: Se detalla el alcance que tiene el proyecto, los requerimientos y funcionalidades que el software o proyecto debe cumplir.
- Capítulo 6 Factibilidad: Se evalúa la viabilidad del proyecto.
- Capítulo 7 Análisis: Se analiza los actores que participan en el software.
- Capítulo 8 Diseño: Se describe el diseño del software desarrollado para resolver el problema identificado.
- Capítulo 9 Código: Este capítulo se presenta y explica el código desarrollado con la finalidad de darle funcionalidad al programa desarrollado.
- Capítulo 10 Pruebas: Se presenta las pruebas realizadas del proyecto.
- Capítulo 11 Plan de capacitación y entrenamiento: Se detalla el plan para capacitar.
- Capítulo 12 Plan de implantación y puesta en marcha: Se presenta cómo se llevará a cabo la implementación y el inicio del proyecto.
- Capítulo 13 Trabajos Futuros: Se analizan ideas o propuestas para futuras mejoras o expansiones del proyecto.
- Capítulo 14 Conclusiones: Resumen de los hallazgos, conclusiones clave y posibles recomendaciones basadas en los resultados obtenidos durante el desarrollo del proyecto.

Capítulo 2

Justificación del Proyecto

2.1. El Proyecto

El proyecto se enfoca en abordar la necesidad de utilizar los brazos robóticos SCORBOT del laboratorio CIMUBB sin la posibilidad de acceder físicamente a ellos. Ante la situación desafianta presentada por la pandemia, la solución inicial fue el uso remoto de computadoras específicas en el laboratorio para permitir la operación a distancia de las máquinas disponibles. Esta solución fue efectiva para garantizar la continuidad de las actividades educativas y de investigación en un entorno virtual, pero presentó algunas limitaciones.

Una de las principales limitaciones es la dependencia del profesor que debe estar presente físicamente en el laboratorio para realizar el monitoreo y la supervisión de la operación remota de los brazos robóticos. Esto puede ser problemático si el profesor no puede asistir al laboratorio debido a restricciones de viaje, enfermedad u otros compromisos. La falta de presencia física puede aumentar el riesgo de fallas mecánicas o incidentes inesperados, lo que podría resultar en daños considerables en el edificio o los equipos.

La idea central de este proyecto es replicar el laboratorio CIMUBB y el funcionamiento de los brazos robóticos SCORBOT de manera virtual para abordar la necesidad de impartir conocimiento sin requerir la presencialidad física en el laboratorio. Esta solución se presenta como una alternativa efectiva y sostenible ante los desafíos planteados por la pandemia y las limitaciones para acceder a los equipos en persona.

Mediante la simulación, se crea un entorno virtual que intenta recrear fielmente el laboratorio y permite a los estudiantes y profesores interactuar con los brazos robóticos SCORBOT a distancia. Esta garantiza que el conocimiento y las habilidades prácticas relacionadas con el manejo de los brazos robóticos se puedan transmitir de manera efectiva, sin incurrir en costos adicionales de equipamiento o mantenimiento de los equipos reales. Los estudiantes pueden realizar prácticas, experimentos y proyectos de manera segura y eficiente desde sus computadoras, permitiéndoles adquirir experiencia valiosa en el uso de esta tecnología sin exponerse a riesgos físicos o daños en el edificio.

Además, la simulación ofrece una ventaja significativa al permitir la flexibilidad de horarios para los profesores y estudiantes, ya que no están limitados por las restricciones de disponibilidad

del laboratorio físico. Los docentes pueden impartir clases y asesorar a los estudiantes de manera remota, brindando retroalimentación y guiándolos en sus proyectos desde cualquier lugar.

2.2. Otras Soluciones

2.2.1. Robocell de Intelitek

Intelitek es una empresa dedicada a la robótica y automatización, en su pagina dicen "brindamos a las instituciones educativas entornos interactivos de aprendizaje tecnológico".

Uno de esos entornos es el software Robocell, un software que le agrega un entorno 3D al usuario, el cual podrá ver un robot con dimensiones reales.

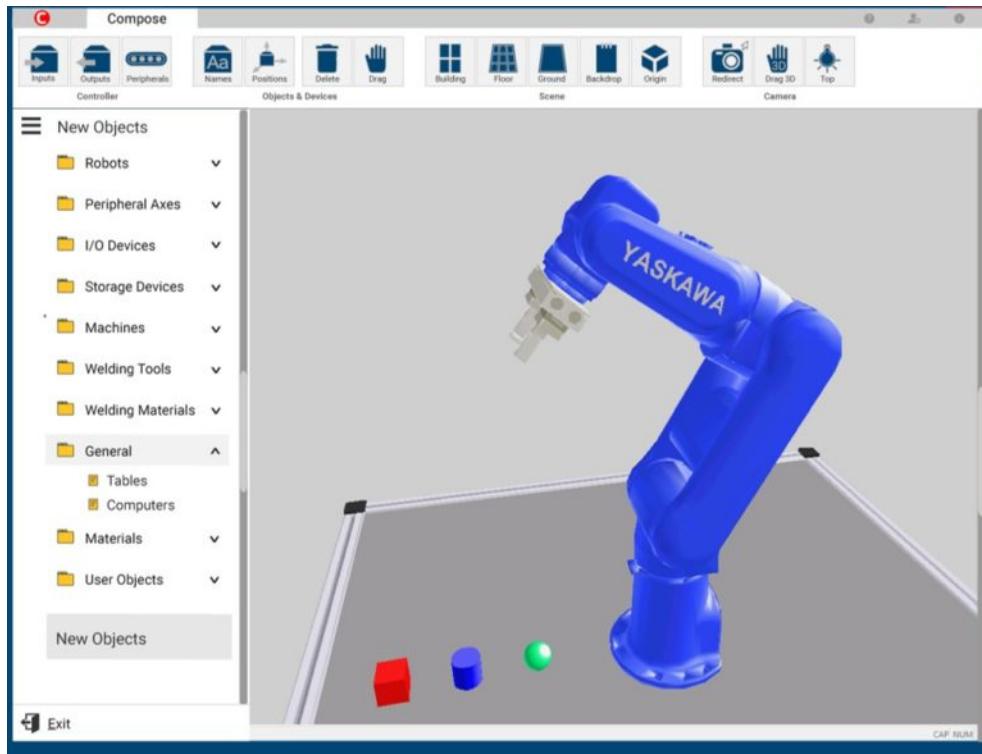


Figura 2.1: Software Robocell

En la Figura 2.1 se presenta la interfaz que ofrece la empresa Intelitek, esta interfaz como tal no es util, debido que el software fue desarrollado para tener una conexión al programa Scorbbase.

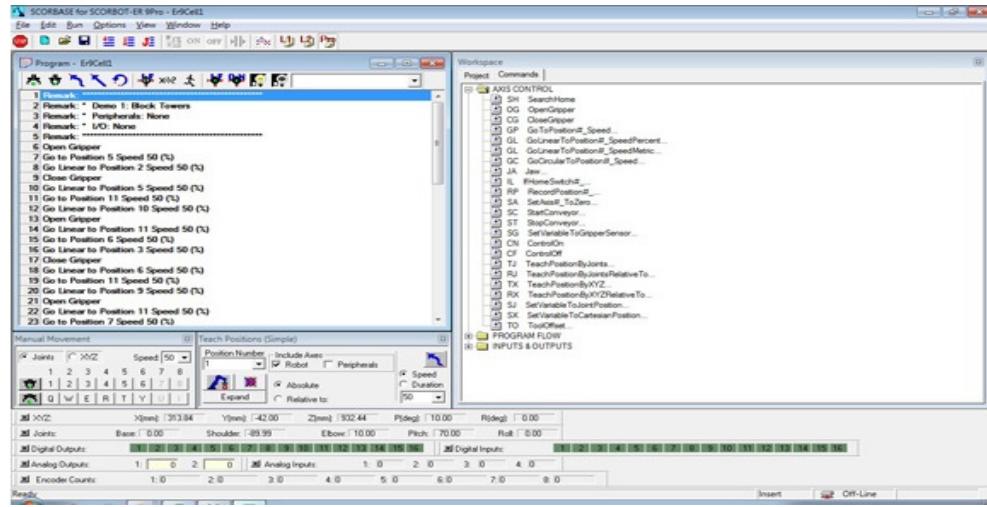


Figura 2.2: Software Scorbbase

En la Figura 2.2 se muestra el programa Scorbbase, el cual tiene como finalidad programar y operar a los brazos robóticos, además de ofrecer soporte e integración a componentes externos, por ejemplo para realizar monitoreo o cintas transportadoras.

Este programa fue creado especialmente para la operación de maquinaria física, por lo cual carece de una opción de poder visualizar el brazo robótico en uso.

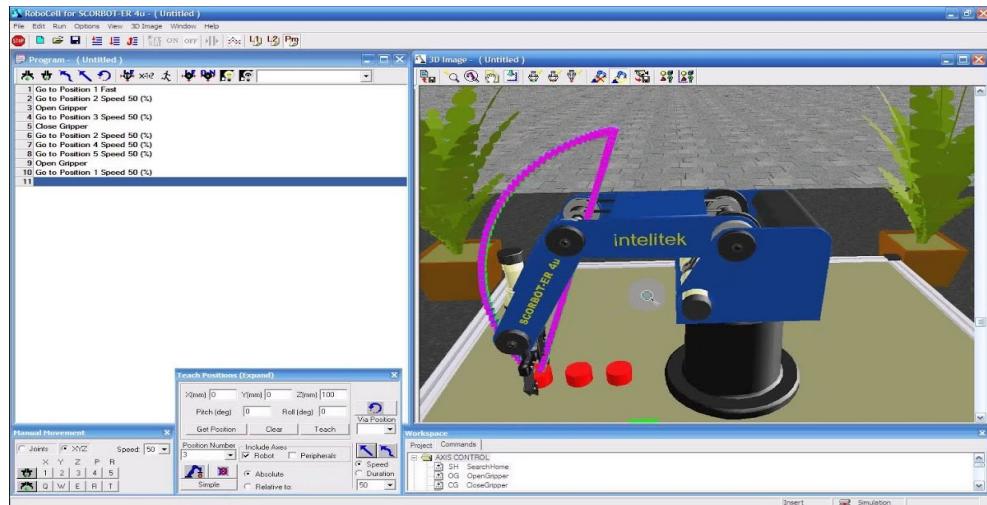


Figura 2.3: Software Robocell y Scorbbase

En la Figura 2.3 se presenta ambos programas funcionando en conjunto, estos pueden mostrar al usuario los brazos robóticos y otros componentes, además de poder programarlo mediante comandos o usar una botonera adaptada al estilo Windows.

2.2.2. Robots Personales

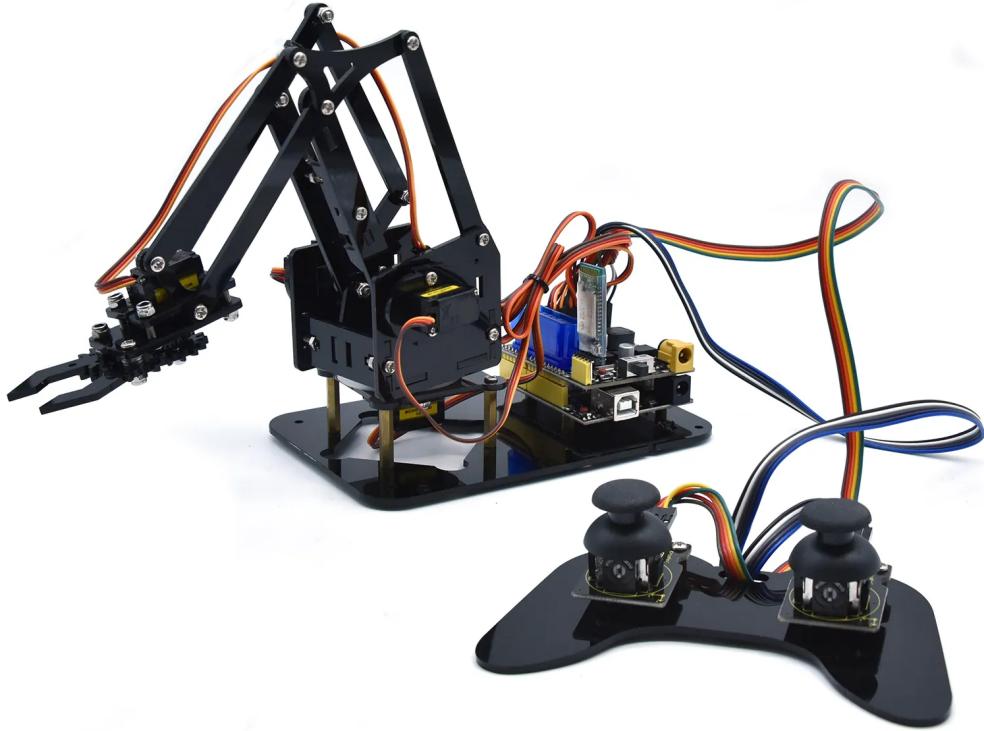


Figura 2.4: Robot Arduino

En la Figura 2.4, se presenta un kit de brazo robótico controlado por Arduino, destacando su versatilidad y accesibilidad. Este sistema permite una interacción directa a través del control físico incluido, proporcionando a los usuarios una experiencia táctil y práctica para la manipulación del brazo robótico.

Además, este kit ofrece la capacidad de programación mediante software, lo que amplía significativamente sus aplicaciones. Los usuarios pueden personalizar y automatizar movimientos del brazo robótico, lo que resulta especialmente valioso en entornos educativos y proyectos de desarrollo.

Capítulo 3

Definición de la institución

En el presente capítulo se presenta la institución y el problema que se aborda en el proyecto.

3.1. Descripción de la institución

- Nombre: Laboratorio de Sistemas Automatizados de Producción
- Dirección: Avenida Collao 1202, Concepción, Chile
- Encargado: Luis Vera
- Rubro: Automatización
- Teléfono: (41) 311 1137
- Descripción: Laboratorio de ensayos para el desarrollo de disciplinas en el ámbito de la Automatización Industrial, Tecnologías Avanzadas de Manufactura, Gestión de la Producción, Robótica, Control Numérico y Visión por Computador.



Figura 3.1: Edificio CIMUBB

3.2. Descripción del área de estudio

La presente tesis tiene como objetivo principal abordar el área de estudio relacionada al laboratorio CIMUBB, el cual, por diversas razones, se puede no contar con acceso a este para realizar demostraciones prácticas. La limitación de acceso al laboratorio físico impide brindar una enseñanza completa y práctica a los estudiantes, ya que gran parte del aprendizaje depende de la interacción directa con la maquinaria y los experimentos que se encuentran en su interior.

Con el fin de superar esta limitación, se propone desarrollar un software de simulación que permita a los estudiantes experimentar y aprender de manera virtual dentro del entorno del laboratorio. El software de simulación debe recrear fielmente los equipos, las configuraciones y las condiciones experimentales presentes en el laboratorio real, brindando una experiencia inmersiva y cercana a la realidad.

La simulación se desarrolló utilizando tecnologías de vanguardia y siguiendo los estándares y protocolos establecidos en el campo de estudio correspondiente. Se diseñó una interfaz intuitiva y amigable para facilitar la interacción de los usuarios con los diferentes elementos del laboratorio virtual, permitiéndoles realizar experimentos, ajustar parámetros, recopilar datos y observar los resultados de sus acciones.

La creación de este software de simulación tiene como objetivo principal proporcionar una alternativa efectiva y accesible para la enseñanza y el aprendizaje en ausencia del acceso físico al laboratorio. De esta manera, se busca brindar a los estudiantes una plataforma que les permita adquirir conocimientos prácticos, desarrollar habilidades experimentales y comprender los conceptos teóricos en un entorno virtual seguro y controlado.

Se espera que esta iniciativa proporcione una solución innovadora que amplíe las posibilidades de enseñanza en el área de estudio, permitiendo a los educadores complementar las clases teóricas con experiencias prácticas simuladas. Además, se busca fomentar la autonomía y la experimentación activa por parte de los estudiantes, promoviendo el descubrimiento y la resolución de problemas dentro del contexto del laboratorio virtual.

En resumen, esta investigación se enfoca en el desarrollo de un software de simulación de laboratorio para superar la limitación de acceso físico, permitiendo así una enseñanza práctica y completa en el área de estudio correspondiente. Se espera que esta solución contribuya significativamente a la formación académica y profesional de los estudiantes, brindándoles una herramienta virtual valiosa y efectiva para su aprendizaje y desarrollo.

3.3. Descripción de la problemática

A los comienzos de la pandemia, el profesor encargado del laboratorio CIMUBB, Luis Vera, comienza a notar una necesidad para sus alumnos, una necesidad que siempre estuvo presente pero, para en esa fecha, nunca había sido tan clara ni tan vital como llegó a ser.



Figura 3.2: Laboratorio CIMUBB

En la Figura 3.2 se muestra una imagen del laboratorio CIMUBB, en el cual se llevan a cabo la entrega del conocimiento sobre la automatización, para la problemática del proyecto se enfoca en los brazos robóticos SCORBOT, los cuales son tres, siendo dos modelos diferentes.



Figura 3.3: SCORBOT ER V Plus

En la Figura 3.3 se presenta el brazo robótico SCORBOT ER V Plus, es un brazo de antigua generación, teniendo cinco ejes de movimiento para el brazo robótico estático y seis ejes para el que posee una cinta de movimiento lineal.



Figura 3.4: SCORBOT ER IX

En la Figura 3.4 se presenta el brazo robótico SCORBOT ER IX, es una versión mejorada, la cual posee siete ejes de movimiento.

El laboratorio CIMUBB no cuenta con un entorno virtual, lo que dificulta significativamente el aprendizaje y la manipulación de los brazos robóticos SCORBOT cuando los estudiantes no pueden acceder físicamente al edificio. La falta de un entorno virtual restringe el acceso de los alumnos a prácticas y experimentos con los brazos robóticos, especialmente durante situaciones como cuarentenas, cortes de luz u otras fallas que requieren la presencia física de un supervisor o profesor.

La falta de acceso a un entorno virtual también limita la flexibilidad y disponibilidad para los estudiantes que deseen aprender sobre sistemas automatizados y brazos robóticos fuera de los horarios de clases regulares. Si el profesor no tiene acceso físico al laboratorio en un momento dado, los estudiantes se ven obligados a esperar hasta que el profesor pueda estar presente nuevamente en el lugar para realizar cualquier actividad relacionada con los brazos robóticos.

Capítulo 4

Definición proyecto

En este capítulo se detallan los objetivos, la metodología que se utiliza, y las tecnologías en la cual se apoya.

4.1. Objetivos del proyecto

4.1.1. Objetivo General

Desarrollar una aplicación en Unity que permita simular el laboratorio del CIMUBB, con la cual los estudiantes puedan probar el funcionamiento de los brazos robóticos SCORBOT ER V Plus y el brazo robótico SCORBOT ER IX , intentando replicar con el mayor detalle posible las estaciones de los brazos en el laboratorio.

4.1.2. Objetivos Específicos

- I.- Examinar los desafíos inherentes de la problemática que presenta el laboratorio.
- II.- Analizar tecnologías y herramientas con el fin de comprender su funcionamiento y utilizarlas de manera eficaz en la resolución del problema.
- III.- Desarrollar la aplicación.

4.1.3. Actividades

Las actividades desarrolladas en este proyecto son:

- I.- Examinar los desafíos inherentes de la problemática que presenta el laboratorio.
 - 1.- Investigación sobre los factores que originan la problemática.
- II.- Analizar tecnologías y herramientas con el fin de comprender su funcionamiento y utilizarlas de manera eficaz en la resolución del problema.
 - 1.- Exploración y análisis de diversas herramientas de programación con el propósito de comprender sus características y funcionalidades.
 - 2.- Investigación para evaluar las posibilidades que ofrecen distintas herramientas de programación y determinar cuál es la más adecuada para abordar el proyecto.

- III.- Desarrollar la aplicación
- 1.- Investigación sobre el laboratorio.
 - 2.- Diseño de entorno y equipamiento.
 - 3.- Desarrollo de código para dar funciones.
 - 4.- Diseño de interfaz gráfica.
 - 5.- Desarrollo de funcionalidad de la interfaz gráfica.

4.2. Ambiente de ingeniería de software

4.2.1. Metodología de Desarrollo

La metodología de desarrollo a utilizar a lo largo del proyecto es incremental [2], pues se debe a que facilita al desarrollo del proyecto y saber si se encuentra por un buen camino, debido a que se le presenta cada incremento al cliente y obtener una retroalimentación del mismo.

4.2.2. Tecnologías

Nombre	Logo	Descripción
C#		C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común . Su sintaxis básica deriva de C / C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

Tabla 4.1: Tecnologías

4.2.3. Herramientas de apoyo

Nombre	Logo	Descripción
Unity		Unity es un motor de desarrollo en tiempo real que te permite crear experiencias interactivas en el Editor de Unity que se utiliza para la creación de videojuegos. Estos se pueden publicar en diversas plataformas como PC, videoconsolas, móviles, etc. Gracias a su flexibilidad es una herramienta que también se usa en diferentes industrias como arquitectura, ingeniería, automotriz y de entretenimiento.
Blender		Blender es un programa informático multiplataforma, dedicado especialmente al modelado, iluminación, renderizado, la animación y creación de gráficos tridimensionales. También de composición digital utilizando la técnica procesal de nodos, edición de vídeo, escultura (incluye topología dinámica) y pintura digital.
Visual Studio Code		Es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.
GitHub		Es un servicio en la nube que ayuda a los desarrolladores a almacenar y administrar su código, al igual que llevar un registro y control de cualquier cambio sobre este código.

Tabla 4.2: Herramientas de apoyo

Capítulo 5

Especificación de requerimientos de software

En el siguiente capítulo se presentan los requerimientos de la aplicación.

5.1. Alcances

El alcance de la simulación abarca la recreación de los brazos Scrbot y otros componentes y maquinarias presentes en el laboratorio, aunque estos últimos no serán funcionales. Entre estos elementos se incluye una cinta transportadora y un objeto específico que el brazo Scrbot podrá tomar y manipular.

La simulación se centra en replicar el comportamiento y las características de los brazos Scrbot de manera precisa, garantizando un rango de movimiento y una capacidad de manipulación de objetos lo más realista posible. Además, se incluye una representación visual detallada de los brazos Scrbot y de los demás componentes y maquinarias presentes en el laboratorio.

El brazo robótico Scrbot puede realizar su movimiento en modo Joints, el cual hace girar en función de los motores. Por otra parte, el movimiento en modo XYZ no está disponible debido a fallos de diseño del modelado del robot. El brazo tampoco es capaz de realizar movimientos guardados, sin embargo en el código se está planteado la lógica de esta.

La cinta transportadora está disponible en la simulación, pero no funciona como en el entorno físico. Su presencia permite a los estudiantes experimentar con la interacción del brazo Scrbot en relación con la transferencia de objetos a lo largo de la cinta, aunque su movimiento y operación real no están habilitados del todo. Esto debido a darle más importancia al brazo como tal, sin embargo, se implementó un código rudimentario, el cual sirve pero no está pulido y tiene varias fallas.

Asimismo, se proporciona un objeto específico en la simulación que el brazo Scrbot podrá tomar y manipular. Esto permite a los usuarios practicar las habilidades de agarre y manipulación del brazo, familiarizándose con los comandos y las técnicas necesarias para realizar estas tareas.

Es importante tener en cuenta que, aunque los componentes y otras maquinarias están presentes en la simulación, su funcionalidad real no está activa.

El enfoque principal es brindar a los estudiantes una experiencia interactiva y visualmente representativa del entorno del laboratorio, centrándose en la operación y el uso del brazo Scrbot. El programa no está diseñado para personas no videntes, y además, se asume que el usuario posee los conocimientos básicos sobre computación.

5.2. Objetivo del software

El objetivo del software de simulación desarrollado es proporcionar una herramienta virtual que permita a los estudiantes experimentar, aprender y practicar el funcionamiento de los brazos Scrbot y otros componentes del laboratorio, en ausencia del acceso físico al mismo.

Los principales objetivos del software de simulación son:

- 1.- Brindar una experiencia realista: El software se diseñará con el objetivo de recrear de manera precisa y realista el comportamiento y las capacidades de los brazos Scrbot, así como otros componentes y maquinarias presentes en el laboratorio. Se buscará proporcionar una representación visual y funcionalidad detalladas, para que los estudiantes puedan interactuar con ellos de manera similar a como lo harían en el entorno físico.
- 2.- Facilitar la práctica y el aprendizaje: El software permitirá a los estudiantes practicar y adquirir habilidades en el uso y manejo de los brazos Scrbot, así como explorar la interacción con otros componentes del laboratorio. Los usuarios podrán realizar diferentes tareas y experimentos, ajustar parámetros y recopilar datos, todo dentro de un entorno virtual controlado y seguro.
- 3.- Fomentar la comprensión teórica y práctica: El software de simulación ayudará a los estudiantes a comprender los conceptos teóricos relacionados con el funcionamiento de los brazos Scrbot y su aplicación en diferentes situaciones. Podrán experimentar directamente los efectos de las acciones realizadas y observar los resultados de sus interacciones, lo que fortalecerá su comprensión de los principios subyacentes.
- 4.- Superar limitaciones de acceso: Al proporcionar una alternativa virtual al laboratorio físico, el software de simulación permitirá a los estudiantes acceder al aprendizaje práctico y experiencial en cualquier momento y lugar, sin restricciones de horarios o limitaciones de espacio físico. Esto ampliará las oportunidades de aprendizaje y brindará una opción adicional para aquellos que no tienen acceso directo al laboratorio.

5.3. Descripción global del producto

5.3.1. Interfaz de usuario

La interfaz de usuario es diseñada con un enfoque en la amabilidad y familiaridad para minimizar la resistencia al cambio y maximizar la adaptabilidad. Al crear una interfaz que los usuarios encuentren intuitiva, se facilita su aceptación y comodidad al utilizar el software. Además, la adaptabilidad es crucial para garantizar que la interfaz funcione de manera eficiente en diferentes contextos y dispositivos, permitiendo a los usuarios acceder y utilizar el programa

de manera óptima en diversas situaciones. Un diseño cuidadoso y una atención especial a la experiencia del usuario pueden fomentar una transición más fluida y exitosa hacia el software.

5.3.2. Interfaz de hardware

En el contexto de computadoras, las interfaces de hardware típicas son el teclado y el ratón para la entrada de datos, y el monitor para la salida visual. Mientras que en el caso de teléfonos y tabletas, las interfaces de hardware se basan en pantallas táctiles, que permiten a los usuarios interactuar directamente con el dispositivo mediante toques y gestos en la pantalla.

5.3.3. Interfaz software

El programa no requiere la instalación de software externo adicional, ya que funciona directamente en sistemas operativos como Windows, Linux o Android. Esto implica que los usuarios pueden ejecutar el programa sin necesidad de instalar bibliotecas o programas adicionales, lo que simplifica su implementación y uso en diferentes plataformas. La compatibilidad con estos sistemas operativos amplía la accesibilidad del programa a una variedad de dispositivos y entornos, facilitando su distribución y adopción.

5.4. Requerimientos específicos

5.4.1. Requerimientos funcionales del sistema

Se presentan los requerimientos funcionales de la aplicación, donde se destaca las características que tiene.

ID	Nombre	Descripción
RF-01.1	Mover el brazo robótico	La aplicación debe permitir al usuario controlar el movimiento del brazo robótico para posicionarlo en diferentes ubicaciones.
RF-01.2	Agarrar objeto con el brazo robótico	La aplicación debe permitir al usuario utilizar el brazo robótico para agarrar y manipular objetos según sea necesario.
RF-02	Controlar botonera	La aplicación debe permitir al usuario hacer uso de la botonera del brazo robótico.
RF-03.1	Cambiar cámara	La interfaz de la aplicación debe permitir al usuario cambiar entre diferentes cámaras para obtener diferentes perspectivas del entorno.
RF-03.2	Cambiar brazo robótico	La interfaz de la aplicación debe permitir al usuario seleccionar y cambiar entre diferentes brazos robóticos disponibles en el sistema.

Tabla 5.1: Requerimientos funcionales

Capítulo 6

Factibilidad

En el capítulo actual se mostrará la factibilidad que tiene el proyecto de ser realizado.

6.1. Factibilidad Técnica

Requerimiento para el desarrollo:

Requisitos mínimos	Windows	macOS	Linux
Versión sistema operativo	Windows 7 (Service Pack 1+), Windows 10 y Windows 11	High Sierra 10.13+	Ubuntu 20.04, Ubuntu 18.04, y CentOS 7
Procesador	Arquitectura x86 o x64 con soporte de instrucción SSE2	Arquitectura x64 con soporte de instrucción SSE2 para CPU Intel, Apple M1 o superior para CPU Apple	Arquitectura x64 con soporte de instrucción SSE2
API Gráfico	GPU compatible con DirectX versión 10, 11 o 12	GPU Intel, AMD o Apple compatible con Metal	GPU compatible con OpenGL 3.2+ o Vulkan

Tabla 6.1: Requerimientos de desarrollo en computadores

Factibilidad Técnica para Desarrollo Unity en un Notebook con Ryzen 7 5700U, 16GB RAM y Ubuntu:

1. Compatibilidad con Sistema Operativo:

- **Factible:** El sistema operativo Ubuntu 20.04 es compatible con los requisitos mínimos para el desarrollo en Unity. Además, la arquitectura x64 del procesador Ryzen 7 5700U es compatible.

2. Procesador:

- **Factible:** El Ryzen 7 5700U es un procesador x64 con soporte para instrucciones SSE2, cumpliendo así con los requisitos mínimos de Unity.

3. API Gráfico:

- **Factible:** Ubuntu 20.04 es compatible con OpenGL 3.2+, y el Ryzen 7 5700U tiene una GPU integrada que es capaz de soportar esta versión de OpenGL.

4. Memoria RAM:

- **Factible:** Con 16GB de RAM, el sistema cumple con los requisitos mínimos, lo que proporcionará suficiente memoria para el desarrollo en Unity.

5. Conexión a Internet:

- **Factible:** Dado que algunos recursos y actualizaciones de Unity pueden requerir una conexión a Internet, es recomendable tener acceso a una conexión estable.

6. Espacio en Disco:

- **Factible:** Con 400GB disponibles en un disco de 500GB, cumples con el requisito mínimo de 15GB libres que necesita Unity. Es completamente factible.

7. Compatibilidad con Controladores Gráficos:

- **Factible:** Los controladores gráficos usados en Vulkan son mesa v23, siendo así compatibles con Unity.

En resumen, después de analizar todos los aspectos técnicos, se determina que el proyecto es factible. Las especificaciones del notebook cumplen con los requisitos mínimos de Unity, y los recursos necesarios están disponibles para el desarrollo sin necesidad de hardware adicional significativo.

6.2. Factibilidad Operativa

- (a) **Recursos Humanos:** Como desarrollador del simulador con experiencia en programación y Unity, los recursos humanos necesarios ya están disponibles internamente. La capacitación para los usuarios se limita a conocimientos básicos sobre el uso de un dispositivo.
- (b) **Recursos Físicos:** Con los datos que entrega la cámara de diputados en 2022[3], se puede asumir que de los 3.89 millones de usuarios de internet fija en la zona residencial, debe tener al menos un computador, por lo cual se dice que al menos 3.89 millones de personas serían potenciales usuarios.
- (c) **Tecnología y Sistemas de Información:** La restricción de hardware para Unity (computadoras desde 2012 y celulares Android desde 2011) se considera aceptable y probablemente cubra una gran parte de los dispositivos en uso actualmente.
- (d) **Cumplimiento Normativo:** Dado que el simulador no almacena datos, no hay problemas de privacidad y seguridad que considerar.
- (e) **Experiencia del Usuario:** La interfaz del simulador se diseñó de manera simple para facilitar su uso.

La factibilidad operativa del simulador de brazos robóticos en Unity es sólida. Con recursos humanos disponibles, un amplio alcance potencial, requisitos tecnológicos razonables y la ausencia de procesos operativos complejos, el simulador parece ser una herramienta eficiente y accesible para entornos educativos.

6.3. Impacto Positivo:

- (a) **Experimentación sin riesgos:** Los entornos virtuales en Unity permiten a los investigadores y estudiantes realizar experimentos y pruebas sin riesgos ni costos asociados con el mundo real. Esto es especialmente útil en campos como la investigación médica, química o física, donde los experimentos pueden ser costosos o peligrosos.
- (b) **Acceso universal:** Los entornos virtuales pueden ser accesibles desde cualquier lugar, lo que facilita la colaboración y el acceso a recursos educativos o de investigación para personas de todo el mundo.
- (c) **Aprendizaje interactivo:** Para fines educativos, un entorno virtual en Unity puede proporcionar una experiencia de aprendizaje más interactiva y atractiva. Puede mejorar la comprensión de conceptos complejos al permitir a los estudiantes interactuar directamente con modelos y simulaciones.
- (d) **Personalización:** Los entornos virtuales pueden adaptarse para satisfacer las necesidades específicas del laboratorio o del programa educativo. Esto permite una mayor personalización y adaptabilidad a diferentes objetivos de investigación o de aprendizaje.
- (e) **Reducción de costos:** Al utilizar entornos virtuales, se pueden reducir los costos asociados con la compra de equipos especializados, materiales y mantenimiento de laboratorios físicos.

6.4. Impacto Negativo:

- (a) **Limitaciones de simulación:** Aunque los entornos virtuales son poderosos, no siempre pueden replicar completamente la complejidad del mundo real. Pueden haber limitaciones en la simulación de ciertos fenómenos o situaciones.
- (b) **Requisitos técnicos:** La implementación y el uso de entornos virtuales en Unity pueden requerir habilidades técnicas y recursos significativos. Esto podría ser un desafío para algunos laboratorios que no tienen acceso a personal técnico o hardware adecuado.
- (c) **Aislamiento:** Dependiendo de cómo se implemente, la tecnología virtual puede llevar al aislamiento del mundo real. Los estudiantes o investigadores pueden perder la experiencia táctil y la interacción directa con los objetos y fenómenos que están estudiando.

6.5. Factibilidad Económica

Costo de personal

Dado que el proyecto de tesis se desarrolla dentro de un entorno académico y no implica la contratación ni la compensación económica adicional para el personal involucrado, no se incluye un ítem de gasto en recursos humanos. La naturaleza educativa y formativa del proyecto permite que los participantes, principalmente los estudiantes responsables de la tesis, contribuyan sin generar desembolsos económicos adicionales para el proyecto. La ausencia de este gasto fortalece la viabilidad económica del proyecto en el ámbito académico.

Costo de desarrollo

Con el propósito de realizar una evaluación económica, se emplea el promedio de la tarifa por hora de un ingeniero civil informático, fijada en 6000 pesos chilenos. Esta estimación se utiliza como una medida representativa del valor del tiempo y la contribución del estudiante al proyecto, a pesar de que no se traduzca en un desembolso financiero directo en este contexto académico.

Fase	Porcentaje	Horas	Costo por Hora	Costo Total
Diseño	10 %	58.5	6000	351,000
Desarrollo	45 %	263.25	6000	1,579,500
Implementación	20 %	117	6000	702,000
Pruebas	15 %	87.75	6000	526,500
Costo Total de Desarrollo				3,159,000 CLP

Tabla 6.2: Costo de Desarrollo del Proyecto en Unity

La cuantificación no se implementa en este caso, ya que al tratarse de un proyecto de tesis, se percibe como un ahorro.

Costo de hardware La ausencia de inversión en hardware se justifica por el hecho de que la aplicación se ejecutará de manera eficiente en los computadores personales de los estudiantes, eliminando así la necesidad de adquirir equipamiento adicional. Este enfoque estratégico garantiza que los recursos tecnológicos existentes sean plenamente utilizados, minimizando cualquier requerimiento financiero relacionado con la infraestructura de hardware del proyecto.

Año	1	2	3	4	5	8
Utilidad	165,000	330,000	495,000	660,000	825,000	1,320,000
Impuesto	31,350	62,700	94,050	125,400	156,750	150,800
Utilidad después de impuesto	133,650	267,300	400,950	534,600	668,250	1,069,200
Utilidad del periodo	133,650	400,950	801,900	1,336,500	2,004,750	4,811,400

Tabla 6.3: Evaluación Financiera del Proyecto

1. **Utilidad:** Representa la ganancia generada por el proyecto en cada año después de la deducción de los costos y gastos, pero antes de impuestos. En este contexto, la "Utilidad" refleja la estimación de reducción de costos asociados con el mantenimiento del brazo robótico. Inicialmente, la reducción del costo de mantenimiento es de 165,000 CLP por mes. Gracias a la implementación del software, se estima que la eficiencia aumentará, lo que resultará en la reducción de un mes de mantenimiento por cada año hasta llegar a ahorrar un 75 % de este. Por lo tanto, la "Utilidad" refleja tanto los ahorros en costos de mantenimiento como cualquier otra ganancia generada por el proyecto, sin considerar impuestos en esta etapa del análisis.
2. **Utilidad del periodo:** Representa la ganancia neta después de impuestos y es el resultado final de restar el impuesto de la utilidad antes de impuestos. Este valor muestra la ganancia neta real que el proyecto genera después de cumplir con las obligaciones tributarias.

6.6. Conclusión de la factibilidad

En conclusión, después de analizar la factibilidad técnica, operativa y económica del proyecto de simulador de brazos robóticos en Unity, se puede afirmar que el proyecto es factible y presenta un potencial impacto positivo en diversas áreas.

Desde el punto de vista técnico, los requisitos de hardware y software necesarios para el desarrollo en Unity son cumplidos por el equipo de desarrollo propuesto, especialmente en el caso del notebook con Ryzen 7 5700U y Ubuntu. Este análisis asegura que la implementación del proyecto no requiera hardware adicional significativo.

En términos operativos, la disponibilidad de recursos humanos, físicos y tecnológicos es sólida. El proyecto cuenta con un desarrollador con experiencia en programación y Unity, y se proyecta un amplio alcance potencial gracias a la cantidad estimada de usuarios de internet fija en la zona residencial.

El impacto positivo del proyecto se refleja en la posibilidad de realizar experimentos sin riesgos, el acceso universal a la herramienta, el aprendizaje interactivo, la personalización según las necesidades y la reducción de costos asociados con laboratorios físicos.

La evaluación económica muestra que, a pesar de los costos iniciales de desarrollo, el proyecto tiene el potencial de generar utilidades a partir del segundo año. La ausencia de costos de personal y hardware adicionales contribuye a la viabilidad económica del proyecto, y el análisis financiero proyecta una utilidad neta positiva en los años subsiguientes.

En resumen, la factibilidad del proyecto se respalda sólidamente en los aspectos técnico, operativo y económico, lo que sugiere que la implementación del simulador de brazos robóticos en Unity es una propuesta viable y prometedora.

Capítulo 7

Análisis

En este capítulo que nos conlleva a analizar los actores que ejecutaran la aplicación, los flujos que esta tendrá.

7.1. Interacción del Usuario en la Simulación

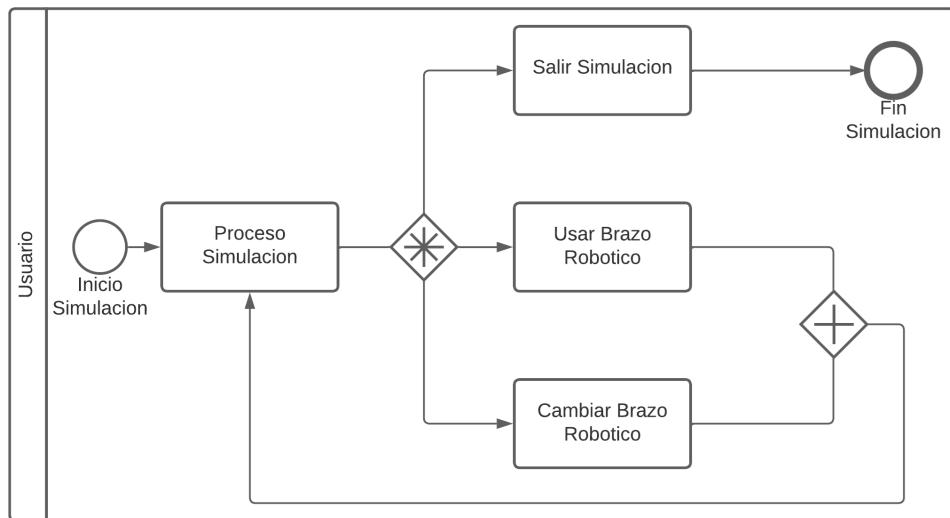


Figura 7.1: BPMN de la Interacción

En la Figura 7.1 se muestra el proceso de como el usuario interactúa con la simulación, el proceso inicia con el usuario ejecutando la simulación, una vez dentro de esta, el usuario puede elegir entre usar el brazo robótico, cambiar de brazo robótico, con las cuales lo lleva a mantenerse en el proceso de la simulación, por ultimo esta la opción de salir de la simulación que finaliza este proceso

7.2. Casos de uso

7.2.1. Actores

Usuario: Persona que utilizará la aplicación, ya sea estudiante o profesor de la Universidad del Bío-Bío, como también personas externas a esta. El usuario puede utilizar la aplicación completamente.

7.2.2. Diagrama de casos de uso y descripción

Este capítulo destaca los casos de uso del proyecto, ofreciendo una visión rápida de las interacciones que los usuarios pueden realizar.

Caso de uso: Uso de Interfaz Principal

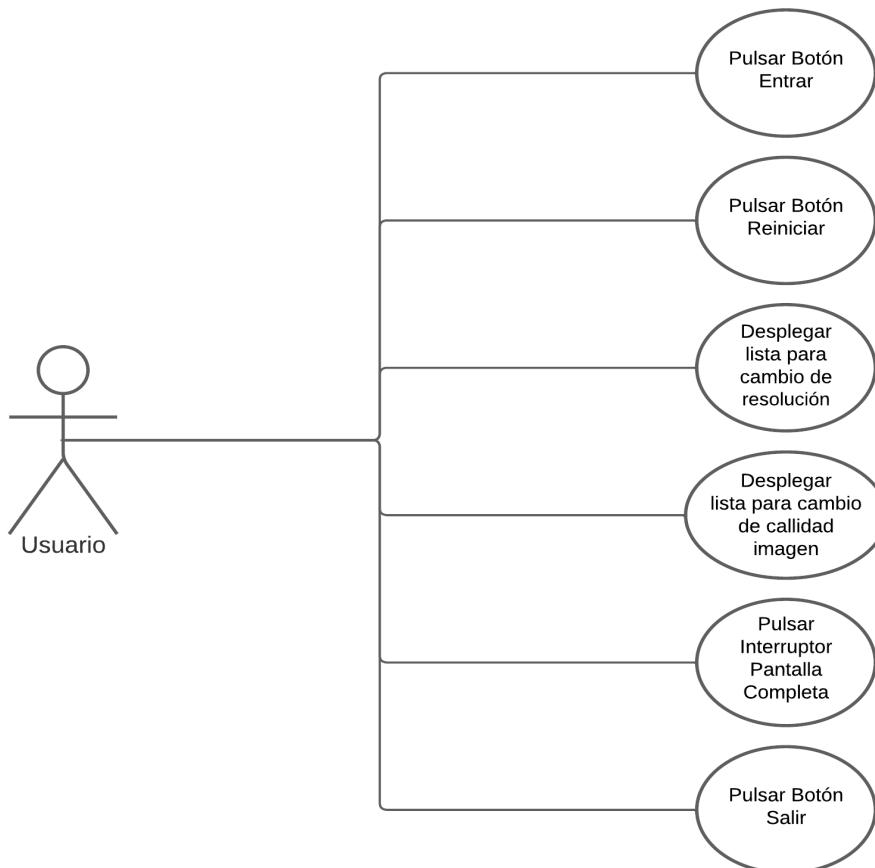


Figura 7.2: Diagrama de caso de uso: Usuario

Como se logra apreciar en la Figura 7.2, se enfoca en la configuración y ajuste de parámetros de la aplicación. Desde reiniciar la simulación hasta cambiar la resolución y calidad de la imagen, estos casos ofrecen opciones para personalizar la configuración de la aplicación según las preferencias del usuario y así poder abarcar más equipos, por ejemplo, con menos recursos.

Caso de uso: Uso de Interfaz dentro de simulación

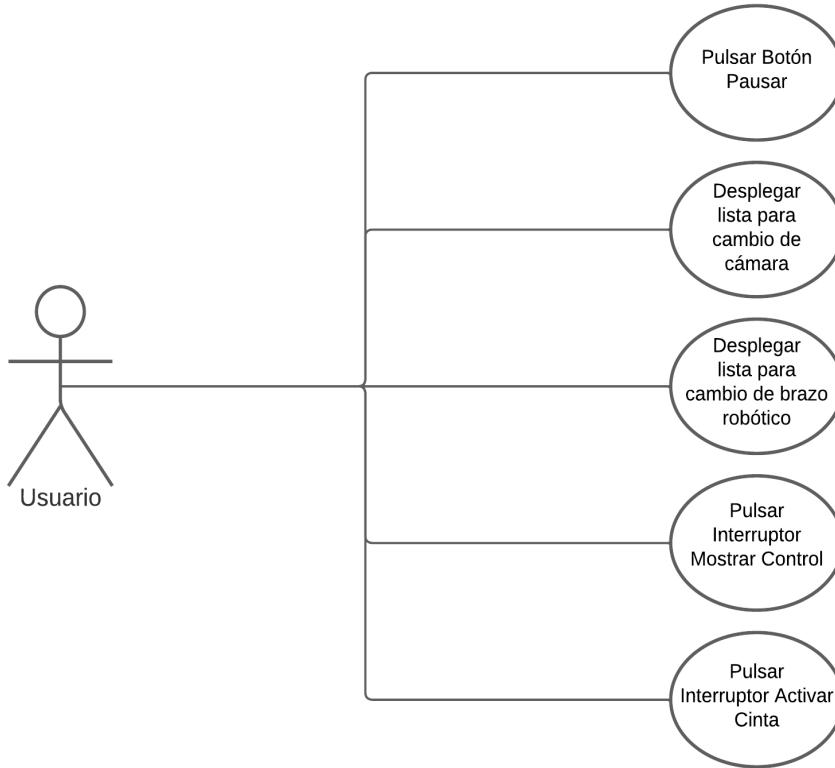


Figura 7.3: Diagrama de caso de uso: Usuario

Como se logra apreciar en la Figura 7.3, esta aborda la navegación y visualización en el sistema. Desde cambiar cámaras hasta activar o desactivar funciones, estos casos permiten a los usuarios personalizar su experiencia de visualización de manera eficiente, cambiar el dispositivo a controlar y cambiar el estado de la cinta transportadora.

Caso de uso: Uso de Botonera

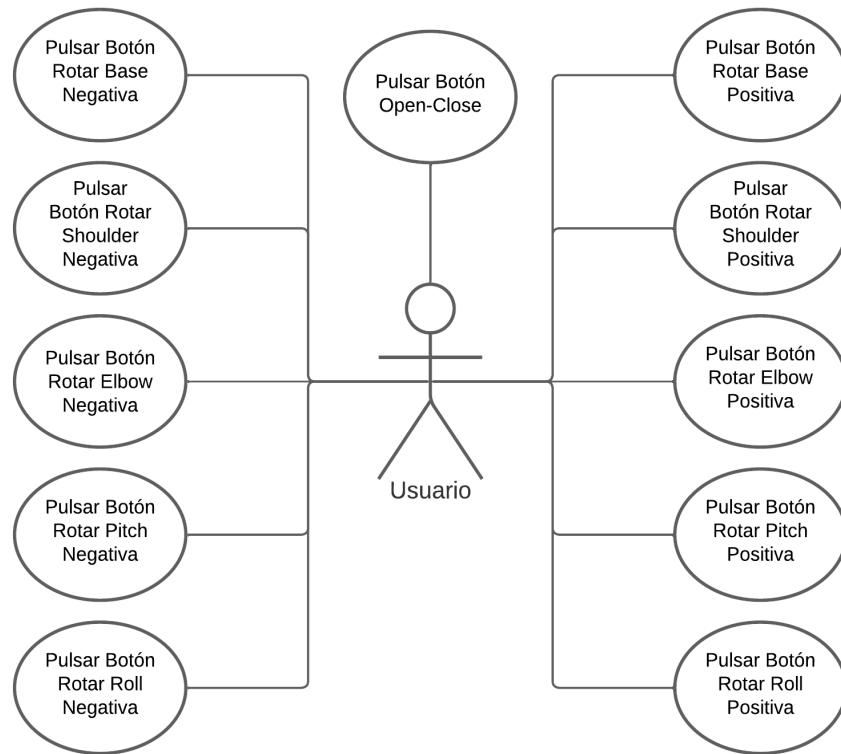


Figura 7.4: Diagrama de caso de uso: Usuario

Como se logra apreciar en la Figura 7.4, se centra en las diversas acciones de movimiento del brazo robot que se encuentran operativas en la botonera. Desde la rotación de articulaciones hasta el control de la pinza, estos casos proporcionan un panorama completo de las capacidades de manipulación del sistema.

7.2.3. Especificación de los caso de uso

Caso de Uso: Pulsar Botón Entrar	
ID	CU-01
Precondiciones	El actor ejecutó el programa.
Descripción	El actor puede ingresar a la simulación al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ul style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Entrar". 3.- El actor pulsa el botón de "Entrar".

Tabla 7.1: Especificación de casos de uso: Pulsar Botón Entrar

Caso de Uso: Pulsar Botón Reiniciar	
ID	CU-02
Precondiciones	El actor ejecutó el programa.
Descripción	El actor puede reiniciar la simulación al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ul style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Reiniciar". 3.- El actor pulsa el botón de "Reiniciar".

Tabla 7.2: Especificación de casos de uso: Pulsar Botón Reiniciar

Caso de Uso: Desplegar lista para cambio de resolución	
ID	CU-03
Precondiciones	El actor ejecutó el programa.
Descripción	El actor puede acceder a una lista para cambiar la resolución del sistema al seleccionar la opción correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza la opción para "Cambio de Resolución". 3.- El actor despliega la lista para cambio de resolución. 4.- El actor cambia la resolución.

Tabla 7.3: Especificación de casos de uso: Desplegar lista para cambio de resolución

Caso de Uso: Desplegar lista para cambio de calidad imagen	
ID	CU-04
Precondiciones	El actor ejecutó el programa.
Descripción	El actor puede acceder a una lista para cambiar la calidad de la imagen del sistema al seleccionar la opción correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza la opción para "Cambio de Calidad de Imagen". 3.- El actor despliega la lista para cambio de calidad de imagen.

Tabla 7.4: Especificación de casos de uso: Desplegar lista para cambio de calidad imagen

Caso de Uso: Pulsar Interruptor Pantalla Completa	
ID	CU-05
Precondiciones	El actor ejecutó el programa.
Descripción	El actor puede activar o desactivar el modo de pantalla completa al pulsar el interruptor correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el interruptor para "Pantalla Completa". 3.- El actor pulsa el interruptor para activar o desactivar la pantalla completa.

Tabla 7.5: Especificación de casos de uso: Pulsar Interruptor Pantalla Completa

Caso de Uso: Pulsar Botón Salir	
ID	CU-06
Precondiciones	El actor ejecutó el programa.
Descripción	El actor puede salir del sistema al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ul style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Salir". 3.- El actor pulsa el botón de "Salir".

Tabla 7.6: Especificación de casos de uso: Pulsar Botón Salir

Caso de Uso: Pulsar Botón Pausar	
ID	CU-07
Precondiciones	El actor entró a la simulación.
Descripción	El actor puede pausar la simulación y volver al menu al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ul style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Pausar". 3.- El actor pulsa el botón de "Pausar".

Tabla 7.7: Especificación de casos de uso: Pulsar Botón Pausar

Caso de Uso: Desplegar lista para cambio de cámara	
ID	CU-08
Precondiciones	El actor entró a la simulación.
Descripción	El actor puede acceder a una lista para cambiar la cámara del sistema al seleccionar la opción correspondiente.
Actores	Usuario
Flujo principal	<ul style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza la opción para "Cambio de Cámara". 3.- El actor despliega la lista para cambio de cámara. 4.- El actor cambia la cámara

Tabla 7.8: Especificación de casos de uso: Desplegar lista para cambio de cámara

Caso de Uso: Desplegar lista para cambio de brazo robótico	
ID	CU-09
Precondiciones	El actor entró a la simulación.
Descripción	El actor puede acceder a una lista para cambiar el brazo robótico al seleccionar la opción correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza la opción para "Cambio de Brazo Robótico". 3.- El actor despliega la lista para cambio de brazo robótico. 4.- El actor cambia el brazo robótico.

Tabla 7.9: Especificación de casos de uso: Desplegar lista para cambio de brazo robótico

Caso de Uso: Pulsar Interruptor Mostrar Control	
ID	CU-10
Precondiciones	El actor entró a la simulación.
Descripción	El actor puede mostrar u ocultar el control del brazo robot al pulsar el interruptor correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el interruptor para "Mostrar Control". 3.- El actor pulsa el interruptor para mostrar u ocultar el control.

Tabla 7.10: Especificación de casos de uso: Pulsar Interruptor Mostrar Control

Caso de Uso: Pulsar Interruptor Activar Cinta	
ID	CU-11
Precondiciones	El actor entró a la simulación.
Descripción	El actor puede activar o desactivar la cinta del brazo robot al pulsar el interruptor correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el interruptor para "Activar Cinta". 3.- El actor pulsa el interruptor para activar o desactivar la cinta.

Tabla 7.11: Especificación de casos de uso: Pulsar Interruptor Activar Cinta

Caso de Uso: Pulsar Botón Rotar Base Negativa	
ID	CU-12
Precondiciones	El actor entró a la simulación. La botonera está visible
Descripción	El actor puede rotar la base del brazo robot en sentido negativo al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Rotar Base Negativa". 3.- El actor pulsa el botón de "Rotar Base Negativa".

Tabla 7.12: Especificación de casos de uso: Pulsar Botón Rotar Base Negativa

Caso de Uso: Pulsar Botón Rotar Base Positiva	
ID	CU-13
Precondiciones	El actor entró a la simulación. La botonera está visible
Descripción	El actor puede rotar la base del brazo robot en sentido positivo al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Rotar Base Positiva". 3.- El actor pulsa el botón de "Rotar Base Positiva".

Tabla 7.13: Especificación de casos de uso: Pulsar Botón Rotar Base Positiva

Caso de Uso: Pulsar Botón Rotar Shoulder Negativa	
ID	CU-14
Precondiciones	El actor entró a la simulación. La botonera está visible
Descripción	El actor puede rotar la articulación "Shoulder" en sentido negativo al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Rotar Shoulder Negativa". 3.- El actor pulsa el botón de "Rotar Shoulder Negativa".

Tabla 7.14: Especificación de casos de uso: Pulsar Botón Rotar Shoulder Negativa

Caso de Uso: Pulsar Botón Rotar Shoulder Positiva	
ID	CU-15
Precondiciones	El actor entró a la simulación. La botonera está visible
Descripción	El actor puede rotar la articulación "Shoulder" en sentido positivo al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Rotar Shoulder Positiva". 3.- El actor pulsa el botón de "Rotar Shoulder Positiva".

Tabla 7.15: Especificación de casos de uso: Pulsar Botón Rotar Shoulder Positiva

Caso de Uso: Pulsar Botón Open-Close	
ID	CU-16
Precondiciones	El actor entró a la simulación. La botonera está visible
Descripción	El actor puede abrir o cerrar la pinza del brazo robot al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón "Open-Close". 3.- El actor pulsa el botón para abrir o cerrar la pinza.

Tabla 7.16: Especificación de casos de uso: Pulsar Botón Open-Close

Caso de Uso: Pulsar Botón Rotar Elbow Negativa	
ID	CU-17
Precondiciones	El actor entró a la simulación. La botonera está visible
Descripción	El actor puede rotar la articulación "Elbow" en sentido negativo al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Rotar Elbow Negativa". 3.- El actor pulsa el botón de "Rotar Elbow Negativa".

Tabla 7.17: Especificación de casos de uso: Pulsar Botón Rotar Elbow Negativa

Caso de Uso: Pulsar Botón Rotar Elbow Positiva	
ID	CU-18
Precondiciones	El actor entró a la simulación. La botonera está visible
Descripción	El actor puede rotar la articulación "Elbow" en sentido positivo al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Rotar Elbow Positiva". 3.- El actor pulsa el botón de "Rotar Elbow Positiva".

Tabla 7.18: Especificación de casos de uso: Pulsar Botón Rotar Elbow Positiva

Caso de Uso: Pulsar Botón Rotar Pitch Negativa	
ID	CU-19
Precondiciones	El actor entró a la simulación. La botonera está visible
Descripción	El actor puede rotar la articulación "Pitch" en sentido negativo al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Rotar Pitch Negativa". 3.- El actor pulsa el botón de "Rotar Pitch Negativa".

Tabla 7.19: Especificación de casos de uso: Pulsar Botón Rotar Pitch Negativa

Caso de Uso: Pulsar Botón Rotar Pitch Positiva	
ID	CU-20
Precondiciones	El actor entró a la simulación. La botonera está visible
Descripción	El actor puede rotar la articulación "Pitch" en sentido positivo al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Rotar Pitch Positiva". 3.- El actor pulsa el botón de "Rotar Pitch Positiva".

Tabla 7.20: Especificación de casos de uso: Pulsar Botón Rotar Pitch Positiva

Caso de Uso: Pulsar Botón Rotar Roll Negativa	
ID	CU-21
Precondiciones	El actor entró a la simulación. La botonera está visible
Descripción	El actor puede rotar la articulación "Roll" en sentido negativo al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none"> 1.- El actor visualiza el control de brazo robot. 2.- El actor localiza el botón de "Rotar Roll Negativa". 3.- El actor pulsa el botón de "Rotar Roll Negativa".

Tabla 7.21: Especificación de casos de uso: Pulsar Botón Rotar Roll Negativa

Caso de Uso: Pulsar Botón Rotar Roll Positiva	
ID	CU-22
Precondiciones	El actor entró a la simulación. La botonera está visible
Descripción	El actor puede rotar la articulación "Roll" en sentido positivo al pulsar el botón correspondiente.
Actores	Usuario
Flujo principal	<ol style="list-style-type: none">1.- El actor visualiza el control de brazo robot.2.- El actor localiza el botón de "Rotar Roll Positiva".3.- El actor pulsa el botón de "Rotar Roll Positiva".

Tabla 7.22: Especificación de casos de uso: Pulsar Botón Rotar Roll Positiva

Capítulo 8

Diseño

En este capítulo, se presenta el diseño de interfaz de la aplicación desarrollada, y las diferentes opciones que se presentan.

8.1. Diseño interfaz y navegación

En la Figura 8.1 se presenta un bosquejo de la interfaz del usuario, la cual muestra la simulación principalmente, y además un control flotante. También se muestran los menús desplegables.

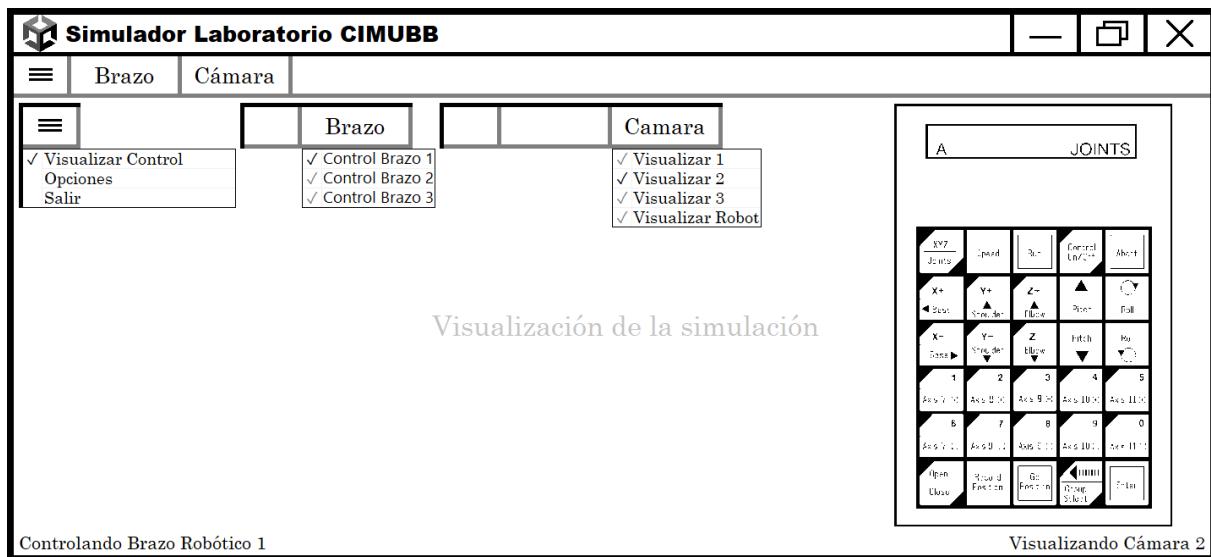


Figura 8.1: Mockup



Figura 8.2: Menu Principal Simulación

Las distintas opciones que se muestran en la Figura 8.2 se detallaran a continuación

- Entrar: Permite entrar a la simulación.
- Reiniciar: Permite reiniciar la simulación, volviendo a la posición original.
- Resolución: Permite cambiar la resolución del programa, esto permite adaptarse a diferentes resoluciones o utilizar menos recursos.
- Calidad Imagen: Permite cambiar la calidad visual de la simulación, esto permite adaptarse a cualquier equipo ya tenga buenos recursos o no.
- Pantalla Completa: Permite que el programa funcione en pantalla completa o en modo ventana,
- Salir: Permite salir de la aplicación.

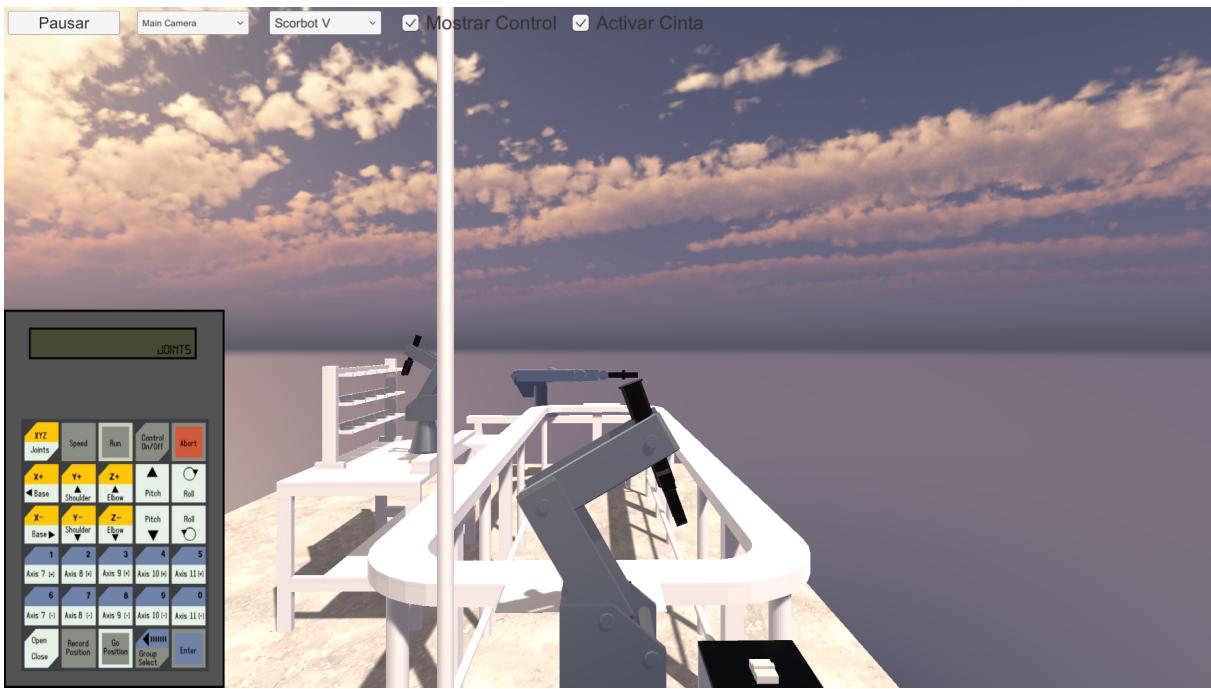


Figura 8.3: Interfaz dentro de la Simulación

Las distintas opciones que se muestran en la Figura 8.3 se detallaran a continuación

- Pausar: Permite pausar la simulación y volver al menu.
- Lista Cámaras: Permite cambiar la perspectiva del usuario.
- Lista Brazos Robóticos: Permite cambiar al robot controlado por el usuario.
- Interruptor Mostrar Control: Permite cambiar la visibilidad de la botonera.
- Interruptor Activar Cinta: Permite cambiar el estado de la cinta transportadora.
- Botonera: Realiza las operaciones replicando el control de los brazos Scorbot reales.

Capítulo 9

Código

En este capítulo, se presenta explicaciones sobre el código fuente de la aplicación desarrollada.

9.1. Lógica Cinta Transportadora

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Belt : MonoBehaviour
{
    public float speed = 5f;
    Rigidbody rBody;
    public Vector3 direccion;

    void Start()
    {
        rBody = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        Vector3 pos = rBody.position;
        rBody.position += direccion * speed * Time.fixedDeltaTime;
        rBody.MovePosition(pos);
    }
}
```

Este código proporciona la lógica simple de movimiento de objetos en la cinta transportadora, sin embargo este no llega a replicar su funcionamiento real debido a limitaciones que se explica en los detalles siguientes:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

Estas líneas son declaraciones de uso de espacio de nombres (namespace) que indican qué bibliotecas se están utilizando en el código. Están importando las bibliotecas necesarias para trabajar con Unity y otros espacios de nombres estándar de C#.

```
public class Belt : MonoBehaviour
{
    ...
}
```

Aquí comienza la definición de la clase Belt, que hereda de la clase MonoBehaviour. Esto significa que este script puede ser adjuntado a objetos en Unity y ejecutado como parte del comportamiento de esos objetos.

```
public float speed = 5f;
Rigidbody rBody;
public Vector3 direccion;
```

En esta parte se declaran las variables, se parte con una variable pública llamada speed (velocidad) que tiene un valor predeterminado de 5. Esta variable representa la velocidad a la que se moverá la cinta transportadora. Después se declara una variable llamada rBody de tipo Rigidbody. Esta variable se utilizará para almacenar una referencia al componente Rigidbody adjunto al objeto al que se adjunte este script. Un Rigidbody es un componente utilizado en el desarrollo de videojuegos y simulaciones físicas para representar objetos que tienen propiedades físicas, como masa, velocidad, y fuerzas que actúan sobre ellos. Por último se declara una variable pública llamada direccion (dirección) de tipo Vector3. Vector3 es una estructura de datos utilizada para representar vectores tridimensionales en un espacio tridimensional. Esta variable se utiliza para definir la dirección en la que se moverá la cinta transportadora. La dirección se establece desde el Inspector de Unity cuando adjuntas este script a un objeto en el juego.

```
void Start()
{
    rBody = GetComponent<Rigidbody>();
}
```

El método Start() es uno de los métodos especiales en Unity que se llama automáticamente cuando se inicia un objeto al que se adjunta un script, en este caso se obtiene una referencia al componente Rigidbody del objeto al que se adjunta este script. Esto se hace utilizando la función GetComponent<Rigidbody>(), y la referencia se almacena en la variable rBody.

```
void FixedUpdate()
{
    ...
}
```

FixedUpdate es un método especial que se encuentra en Unity y es utilizado para realizar cálculos y actualizaciones relacionadas con la física en un juego. A diferencia de Update, que se llama una vez por cada fotograma (frame) renderizado, FixedUpdate se llama en intervalos de tiempo fijos y regulares, lo que lo hace especialmente adecuado para manejar la simulación de física y movimiento en un juego.

```
Vector3 pos = rBody.position;
```

Aquí se crea una variable local llamada pos que almacena la posición actual del objeto asociado al Rigidbody.

```
rBody.position += direccion * speed * Time.fixedDeltaTime;
```

Esta línea actualiza la posición del objeto con el componente Rigidbody (rBody) al agregarle un desplazamiento basado en la dirección (direccion), la velocidad (speed), y el tiempo transcurrido (Time.fixedDeltaTime). Esto aun no simula el movimiento de la cinta transportadora y se explica con las siguientes figuras.

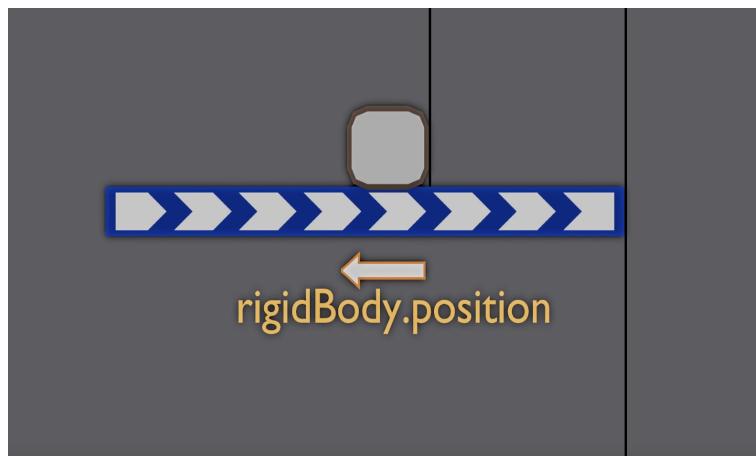


Figura 9.1: rigidBody Position 1

En la Figura 9.1 se presenta el movimiento de la cinta, este movimiento solo aplica para la cinta, a diferencia del método MovePosition que se presenta más adelante.

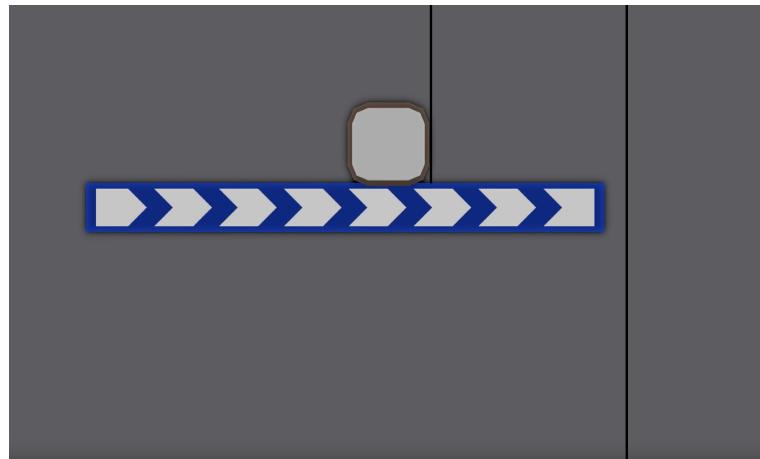


Figura 9.2: rigidBody Position 2

En la Figura 9.2 se muestra donde finaliza el movimiento, como se aprecia este solo mueve la cinta y el objeto sobre esta no posee movimiento

```
rBody.MovePosition(pos);
```

En este método, se devuelve la cinta a su posición original, la diferencia que con el método de movimiento anterior es que si se mueve el objeto, tal como se presenta en las siguientes figuras.

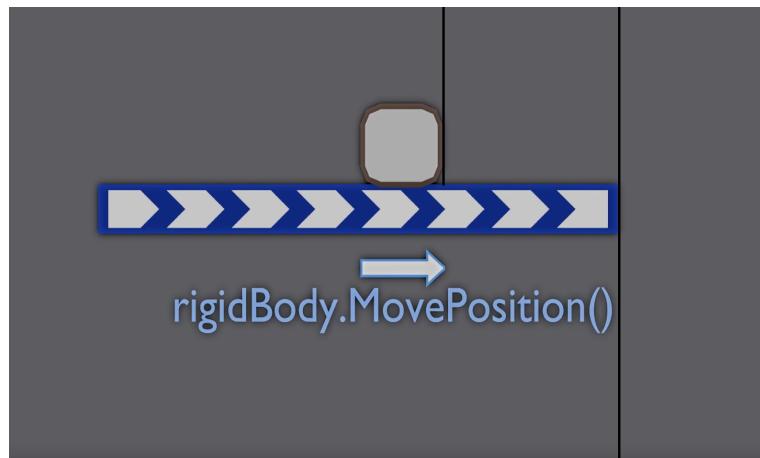


Figura 9.3: rigidBody MovePosition 1

En la Figura 9.3 se presenta el movimiento de la cinta, este movimiento aplica tanto para la cinta como para el objeto.

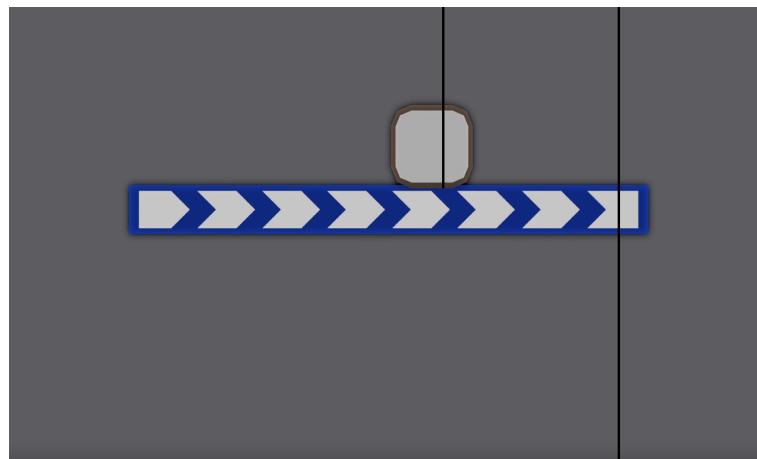


Figura 9.4: rigidBody MovePosition 2

En la Figura 9.4 se muestra donde finaliza el movimiento, como se aprecia este mueve tanto la cinta y como el objeto.

Con esta sucesión de movimientos, la cinta al "ir para atrás" sin mover el objeto, y después "volver a su posición" moviendo el objeto, repitiéndolo una y otra vez se logra el efecto de una cinta transportadora. El problema es que el objeto solo se mueve y en las esquinas no rota como en el laboratorio.

9.2. Lógica Brazos Robóticos

Para esta lógica, se explican los detalles mas importantes debido a que el código es demasiado extenso.

```
//Base
public float velocidadRotacionBase = 50.0f; // Velocidad de rotacion
public KeyCode teclaRotacionPositivaBase = KeyCode.Q; // Tecla para la ↵
    rotacion en direccion positiva
public KeyCode teclaRotacionNegativaBase = KeyCode.A; // Tecla para la ↵
    rotacion en direccion negativa
public Transform Base; // Transform del objeto que se va a rotar
private float LimitePositivoBase = 155.0f; // Angulo limite del objeto
private float LimiteNegativoBase = -155.0f; // Angulo limite del objeto
private float sumaRotacionBase = 0.0f; // Suma de la rotacion del objeto ↵
    para su uso en limites

//Shoulder
public float velocidadRotacionShoulder = 50.0f; // Velocidad de rotacion
public KeyCode teclaRotacionPositivaShoulder = KeyCode.W; // Tecla para la ↵
    rotacion en direccion positiva
public KeyCode teclaRotacionNegativaShoulder = KeyCode.S; // Tecla para la ↵
    rotacion en direccion negativa
public Transform Shoulder; // Transform del objeto a rotar
public Transform puntoFijoShoulder; // Transform del punto fijo alrededor ↵
    del cual se realizara la rotacion
private float LimitePositivoShoulder = 10.0f; // Angulo limite del objeto
private float LimiteNegativoShoulder = -130.0f; // Angulo limite del objeto
private float sumaRotacionShoulder = 0.0f; // Suma de la rotacion del objeto ↵
    para su uso en limites
```

Para explicar las declaraciones de las partes de los brazos, se toma en cuenta dos partes distintas pero aplica para cualquier otra parte similar, la base es un objeto que tiene rotación sobre su propio eje, es decir no necesita otro punto para rotar o rota sobre su propio centro, a diferencia del Shoulder o un brazo cualquiera, para rotar este tiene un punto fijo en uno de sus extremos y no en el centro. Se empieza con declarar la velocidad de la rotación del objeto en una variable float. Después se declaran variables tipo KeyCode que sirven para almacenar la información de una tecla en el teclado en particular, esto se realiza para realizar pruebas de movimiento. Al declarar una variable de tipo Transform, en Unity, un "Transform" se refiere a un componente fundamental que se encuentra en la mayoría de los objetos en un escenario 3D o 2D. El componente Transform está asociado con la posición, rotación y escala de un objeto en el espacio de juego. Básicamente, controla la ubicación y la orientación de un objeto en el mundo virtual creado en Unity. Con esta variable se puede guardar toda la información previamente descrita, con la finalidad de realizar movimientos. En casos de rotación sobre el mismo objeto como la Base, solo se pide una variable, para objetos con centro en su extremo se piden dos variables, mas adelante se detallara como funciona el movimiento y el motivo de pedir uno o dos variables. Y por ultimo se tienen los limites, consisten en dos variables que marcan el límite de operación de cada parte, para

este caso se utilizan los límites establecidos mecánicamente por la misma unidad descrita en su manual. Y también la suma de rotación, la cual almacenara como tal el movimiento realizado, esto para ser utilizado en la comprobación de límites

```
private string nombreObjeto = "";
```

En esta linea se declara una variable para identificar la parte que se moverá con la botonera.

```
private bool positivoboton = false;
private bool negativoboton = false;
```

Con estas variables sirven para saber si el botón presionado es positivo o negativo

```
private bool isControlPressed = false;
private bool isAltPressed = false;
```

Para estas lineas se utilizo para realizar pruebas para guardar posiciones y realizar el movimiento a esa posición, estas servían para saber si Control fue presionado o el Alt fue presionado

```
private bool EmpezarGuardar = false;
private bool TerminarGuardar = false;
private bool EmpezarMover = false;
private bool TerminarMover = false;
```

Acá los booleanos son para lo descrito anteriormente, sirven para marcar los inicios y finales de guardar posiciones y mover a las posiciones

```
private int NumeroUsar = -1;
```

Este numero es el numero usado en la botonera, en el cual se guardara la información.

```
private Dictionary<int, float> GuardarBase = new Dictionary<int, float>();
private Dictionary<int, float> GuardarShoulder = new Dictionary<int, float>();
private Dictionary<int, float> GuardarElbow = new Dictionary<int, float>();
private Dictionary<int, float> GuardarWrist = new Dictionary<int, float>();
private Dictionary<int, float> GuardarEndEffector = new Dictionary<int, float>();
```

Estas variables son diccionarios, los cuales con una variable devuelve una segunda variable, con esto poder guardar por ejemplo, la posición dependiendo del numero requerido, función que ocupa la botonera. Esta declarada para que con un numero entero devuelva un flotante.

```

private void Start()
{
    GuardarBase.Add(0, 0f);
    GuardarShoulder.Add(0,0f);
    GuardarElbow.Add(0,0f);
    GuardarWrist.Add(0,0f);
    GuardarEndEffector.Add(0,0f);
}

```

Al ejecutar el programa, en su inicio ejecutara esta secuencia de funciones, las cuales son para agregar la posición cero a los diccionarios.

```

private void Update()
{
    Movimiento(Base, teclaRotacionPositivaBase, teclaRotacionNegativaBase, ←
        velocidadRotacionBase, LimitePositivoBase, LimiteNegativoBase, Base, ←
        Base.up, ref sumaRotacionBase);
    Movimiento(Shoulder, teclaRotacionPositivaShoulder, ←
        teclaRotacionNegativaShoulder, velocidadRotacionShoulder, ←
        LimitePositivoShoulder, LimiteNegativoShoulder, puntoFijoShoulder, ←
        puntoFijoShoulder.forward, ref sumaRotacionShoulder);
    Movimiento(Elbow, teclaRotacionPositivaElbow, teclaRotacionNegativaElbow←
        , velocidadRotacionElbow, LimitePositivoElbow, LimiteNegativoElbow, ←
        puntoFijoElbow, puntoFijoElbow.forward, ref sumaRotacionElbow);
    Movimiento(Wrist, teclaRotacionPositivaWrist, teclaRotacionNegativaWrist←
        , velocidadRotacionWrist, LimitePositivoWrist, LimiteNegativoWrist, ←
        puntoFijoWrist, Wrist.forward, ref sumaRotacionWrist);
    Movimiento(EndEffector, teclaRotacionPositivaEndEffector, ←
        teclaRotacionNegativaEndEffector, velocidadRotacionEndEffector, ←
        LimitePositivoEndEffector, LimiteNegativoEndEffector, ←
        puntoFijoEndEffector, EndEffector.right, ref sumaRotacionEndEffector←
        );
    MovimientoBoton(nombreObjeto);
    Guardar();
    Usar();
    GuardarBoton();
    UsarBoton();
    MoverEje();
    MoverEjeBoton();
}

```

El update se encarga de que el programa este esperando una interacción por parte del usuario para realizar una función, las cuales se explicaran a continuación.

```
private void Movimiento(Transform objeto, KeyCode TeclaPositiva, KeyCode ←
    TeclaNegativa, float VelocidadRotacion, float anguloLimitePositivo, ←
    float anguloLimiteNegativo, Transform puntoFijo, Vector3 direccion, ref ←
    float Suma)
{
    float direccionRotacion = 0.0f;
    if (Input.GetKey(TeclaPositiva))
    {
        direccionRotacion = 1.0f;
    }
    else if (Input.GetKey(TeclaNegativa))
    {
        direccionRotacion = -1.0f;
    }
    if (direccionRotacion == 0.0f)
    {
        return;
    }
    float anguloRotacion = direccionRotacion * VelocidadRotacion * Time.←
        deltaTime;
    float anguloRotacionLimitado = Mathf.Clamp(Suma, anguloLimiteNegativo, ←
        anguloLimitePositivo);
    if (anguloRotacionLimitado == anguloLimiteNegativo && direccionRotacion <←
        0)
    {
        anguloRotacion = 0.0f;
    }

    if (anguloRotacionLimitado == anguloLimitePositivo && direccionRotacion >←
        0)
    {
        anguloRotacion = 0.0f;
    }
    objeto.RotateAround(puntoFijo.position, direccion, anguloRotacion);
    Suma += anguloRotacion;
    Suma = Mathf.Clamp(Suma, anguloLimiteNegativo, anguloLimitePositivo);
}
```

En la programación del simulador, lo principal fue realizar los movimientos de los brazos robóticos, los cuales principalmente se probaron mediante teclado y no por la botonera. Este código da origen al usado en la botonera, por lo que se explica el código de la botonera.

Para utilizar la botonera, primero se parte detectando que botón se está presionando

```
public void PointerUpPositivo(string nombre)
{
    nombreObjeto = nombre;
    positivoboton = false;
}

public void PointerDownPositivo(string nombre)
{
    nombreObjeto = nombre;
    positivoboton = true;
}

public void PointerUpNegativo(string nombre)
{
    nombreObjeto = nombre;
    negativoboton = false;
}

public void PointerDownNegativo(string nombre)
{
    nombreObjeto = nombre;
    negativoboton = true;
}
```

El método Pointer sirve para identificar los cambios de estado del puntero, ya sea que se haya ejecutado un clic o se mantenga presionado. Con esto se puede pasar a la función de MovimientoBoton.

```
private void MovimientoBoton(string nombre)
{
    float direccionRotacionboton = 0.0f;
    if (positivoboton)
    {
        direccionRotacionboton = 1.0f;
    }
    if (negativoboton)
    {
        direccionRotacionboton = -1.0f;
    }
    if (direccionRotacionboton == 0.0f)
    {
        return;
    }
}
```

Con el método anterior Pointer, se cambia el estado del booleano, el cual le da la dirección de movimiento.

```

float Sumar = 0;
Vector3 direccion = Vector3.up;
float anguloLimitePositivo = 0;
float anguloLimiteNegativo = 0;
float VelocidadRotacion = 0;
Transform puntoFijo = Base;
Transform objeto = Base;

```

Estos datos se pueden considerar como "basura", debido que estos se declaran solo para evitar el error nulo de Unity.

```

if (nombre == "Base")
{
    Sumar = sumaRotacionBase;
    direccion = Base.up;
    anguloLimitePositivo = LimitePositivoBase;
    anguloLimiteNegativo = LimiteNegativoBase;
    VelocidadRotacion = velocidadRotacionBase;
    puntoFijo = Base;
    objeto = Base;
}

```

En el método Pointer, este aparte de detectar un click, trae el nombre del botón presionado, con esto se reescribe los datos "basura" anteriormente declarados, para tener los datos que corresponden, esto para cada articulación del brazo.

```

float anguloRotacion = direccionRotacionboton * VelocidadRotacion * Time ←
    .deltaTime;
float anguloRotacionLimitado = Mathf.Clamp(Sumar, anguloLimiteNegativo, ←
    anguloLimitePositivo);

```

En esta parte se ejecutan los cálculos de la rotacion, primero se adapta el movimiento a una velocidad que Unity pueda interpretar. Después se realiza la comprobación de que la suma de movimiento (la cual representa el angulo que posee a partir de su punto inicial) se encuentre dentro de los límites establecidos.

```

if (anguloRotacionLimitado == anguloLimiteNegativo && ←
    direccionRotacionboton < 0)
{
    anguloRotacion = 0.0f;
}
if (anguloRotacionLimitado == anguloLimitePositivo && ←
    direccionRotacionboton > 0)
{
    anguloRotacion = 0.0f;
}

```

Aca se utiliza una segunda comprobación, pues aunque se revisara antes si la suma se encontraba dentro de los limites, este no limitaba un movimiento, por el cual la articulación sigue el movimiento. Para evitar el movimiento se agrega comprobar si la suma(el anguloRotacionLimitado es igual a la suma) es igual a los limites, y de ser asi, este elimina el anguloRotacion, eliminando cualquier movimiento calculado.

```
objeto.RotateAround(puntoFijo.position, direccion, anguloRotacion);
```

El método RotateAround se encarga de realizar el movimiento de las articulaciones, se elige este método al ser mas "universal" y el código final fue compatible con diferentes modelos sin tener que realizar modificaciones a este.

```
Sumar += anguloRotacion;
Sumar = Mathf.Clamp(Sumar, anguloLimiteNegativo, anguloLimitePositivo);
```

Para ir finalizando, se agrega el angulo recién operado a la suma, ademas de comparar si este se sale o no de su limite, esto debido que internamente en Unity se iban agregando decimales, aunque sean de muy poco valor, estos al sumar llegarían a alterar los ángulos

```
if (nombre == "Base")
{
    sumaRotacionBase = Sumar;
}
```

Y por ultimo, se guarda la suma en su variable correspondiente.

9.3. Lógica tomar objeto

```
public void OnClick()
{
    if (podertomar)
    {
        boton = true;
    }
    if (tomado)
    {
        boton = false;
    }
}
```

Aca se espera un Click en el botón, verificando si el booleano podertomar es verdadero para activar la función del botón, caso contrario se revisa si el booleano tomado es verdadero para desactivar la función del botón

```
private void OnTriggerStay(Collider other)
{
    if (other.gameObject.CompareTag("Objeto"))
    {
        if (pickedObject == null)
        {
            podertomar = true;
            if (Input.GetKey("z") || boton)
            {
                other.GetComponent<Rigidbody>().useGravity = false;
                other.GetComponent<Rigidbody>().isKinematic = true;
                other.transform.position = handPoint.transform.position;
                other.gameObject.transform.SetParent(handPoint.gameObject.transform);
                pickedObject = other.gameObject;
                tomado = true;
                podertomar = false;
                boton = true;
            }
        }
    }
}
```

El método OnTriggerStay sirve cuando dos objetos en Unity colisionan, con esto se compara que el objeto colisionado tiene la etiqueta de "Objeto". Después se revisa que no exista un objeto ya tomado, al ser nulo, este activa la función de podertomar. El objeto al estar tomado, se le desactivan la función de gravedad para poder mantenerse en la posición designada, dando el efecto de agarre. También se le activa la propiedad de kinemático, el cual hace que el objeto no se vea afectado por fuerzas externas o colisiones. Después se le asigna la posición de "mano",

la cual al desarollarlo se debe establecer, y por ultimo agrega el objeto como "hijo", con esto seguirá el movimiento del robot

```
void Update()
{
    if(pickedObject != null)
    {
        if(Input.GetKey("x") || !boton)
        {
            pickedObject.GetComponent<Rigidbody>().useGravity = true;
            pickedObject.GetComponent<Rigidbody>().isKinematic = false;
            pickedObject.gameObject.transform.SetParent(null);
            pickedObject = null;
            tomado = false;
        }
    }
}
```

En esta parte se suelta el objeto, cambiando las propiedades anteriormente modificadas, eliminando el parentesco, y nuevamente dejando el brazo listo para poder tomar otro objeto.

Capítulo 10

Pruebas

Este capítulo se enfoca en las diferentes pruebas del software desarrollado. Se destaca su importancia para garantizar la calidad y funcionalidad del producto final, asegurando que cumpla con los requisitos establecidos.

10.1. Elementos de prueba

Se realiza una prueba con 4 estudiantes de Ingeniería de Ejecución Electronica que recientemente realizaron clases utilizando los equipos del laboratorio y el profesor Luis. Estos realizan las pruebas del programa y llenan una encuesta.

10.2. Detalle de las pruebas solicitadas por el desarrollador

El usuario de prueba utiliza el programa sin algún condicionamiento, este solo conoce que el programa es para "recrear" el laboratorio. Después de utilizar el programa, rellena la siguiente encuesta

- ¿Es fácil usar la botonera del robot?
- ¿Es fácil realizar los cambios de cámara?
- ¿Es fácil cambiar de brazo robótico?
- El menu inicial ¿Es fácil de entender y sencillo de usar?
- La interfaz dentro de la simulación ¿Es fácil de entender y sencillo de usar?

Esta encuesta se realizó mediante Google Forms, las cuales también dan la opción de redactar para dar comentarios.

10.3. Respuestas

¿Es fácil usar la botonera del robot?

Nombre	Respuesta	Comentario
Luis	Si	Falta el modo XYZ, eso facilitaría poder tomar objetos, también faltan otras funciones
Tomas	Si	Fácil de usar para el operario
Carlos	Si	Si es fácil de usar ya que es interactiva
Sebastian	Si	es similar a la del laboratorio
Fabian	Si	Con el uso adecuado y entreno respectivo si se hace "fácil"

Tabla 10.1: Facilidad de uso de la botonera del robot

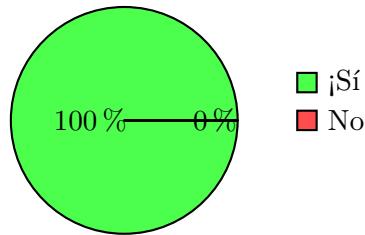


Figura 10.1: Facilidad de uso de la botonera del robot

¿Es fácil realizar los cambios de cámara?

Nombre	Respuesta	Comentario
Luis	Si	El listado de cámaras confunde. Quizás utilizar un nombre descriptivo pueda ser util. Quizás considerar utilizar el cambio de vistas como en los juegos arcade
Tomas	Si	
Carlos	No	Mejor anclaje de cámara
Sebastian	Si	
Fabian	No	

Tabla 10.2: Facilidad de uso del cambio de cámara

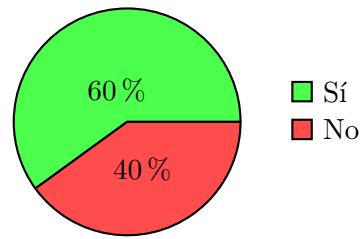


Figura 10.2: Facilidad de uso del cambio de cámara

¿Es fácil cambiar de brazo robótico?

Nombre	Respuesta	Comentario
Luis	Si	
Tomas	Si	
Carlos	Si	
Sebastian	Si	
Fabian	Si	

Tabla 10.3: Facilidad de uso del cambio de brazo robótico

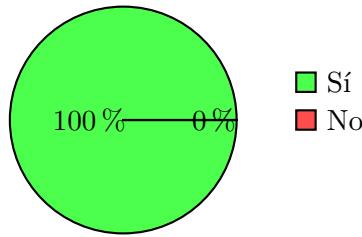


Figura 10.3: Facilidad de uso del cambio de brazo robótico

El menú inicial ¿Es fácil de entender y sencillo de usar?

Nombre	Respuesta	Comentario
Luis	Si	
Tomas	Si	
Carlos	Si	Si es fácil de acuerdo con la guía
Sebastian	Si	
Fabian	No	Hay que saber como mover el brazo, y obviamente saber lo que se está haciendo, y para ello se requiere un entreno decente

Tabla 10.4: Facilidad de uso del menu

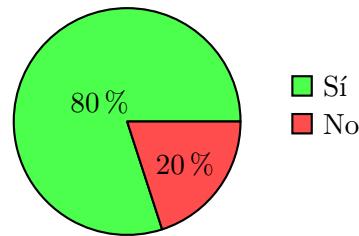


Figura 10.4: Facilidad de uso del menu

La interfaz dentro de la simulación ¿Es fácil de entender y sencillo de usar?

Nombre	Respuesta	Comentario
Luis	No	
Tomas	Si	
Carlos	Si	
Sebastian	Si	
Fabian	Si	

Tabla 10.5: Facilidad de uso del menu dentro de la simulación

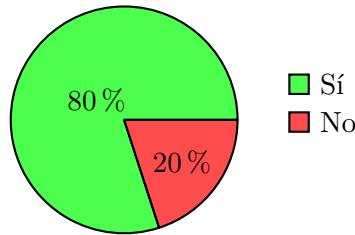


Figura 10.5: Facilidad de uso del menu dentro de la simulación

10.4. Conclusiones de prueba del desarrollador

Se puede analizar que mayormente el programa cumple la funcionalidad de ser intuitivo, básicamente cubre las expectativas que se esperaba. Por otra parte, existen comentarios de mejora como la visual de la cámara. Por ultimo, el comentario de Fabian que dice que "hay que tener conocimiento para saber lo que se hace", indica la complejidad de operación que se tiene con el brazo robótico, aunque para replicar el funcionamiento que tiene en el laboratorio, no se puede hacer de manera más simple.

10.5. Prueba del profesor

El profesor, interesado en evaluar la aplicación desarrollada, ha solicitado a uno de sus alumnos, específicamente a Juan Henríquez de la carrera Ingeniería Civil Informática ,que lleve a cabo la prueba correspondiente.

Respuesta de la prueba

La operación de la cámara resulta un tanto torpe.

Se sugiere fijar los brazos para mejorar su estabilidad.

Además, se propone incorporar un sistema visual que indique la posición de cada brazo o establecer un sistema de referencia en la mesa para clarificar la asignación de ejes a cada botón.

En relación al menú, se observa que la resolución no se ajusta a 1080p, y además, carece de utilidad al no mostrar las teclas asignadas. Sería conveniente mejorar la visualización del menú, asegurando su adaptabilidad a la resolución mencionada y proporcionando información clara sobre las teclas asignadas.

La identificación y accesibilidad al botón de abrir/cerrar la garra resulta problemática y debería ser abordada para facilitar su localización y uso eficiente.

En cuanto al sistema de selección de robots, se sugiere optimizarlo para ofrecer únicamente las opciones de cámara correspondientes a la posición del robot seleccionado. Esto ayudaría a evitar maniobrar inadvertidamente con otros robots.

Se destaca que el robot 5 eje z y 9 presentan un comportamiento inesperado al presionar el botón "1".

Capítulo 11

Plan de capacitación y entrenamiento

Este capítulo presenta el plan para capacitar y entrenar al equipo del proyecto. El objetivo es brindar las habilidades y conocimientos necesarios para lograr un uso exitoso del software.

11.1. Plan de capacitación

El plan de capacitación se centra en proporcionar a los participantes una comprensión integral del programa mediante la creación y consulta de una wiki alojada en GitHub¹. Esta plataforma servirá como un recurso centralizado y accesible para explorar todos los aspectos del programa, desde conceptos básicos hasta funciones avanzadas.

El contenido de la wiki abordará detalladamente el funcionamiento del programa, proporcionando documentación clara y ejemplos prácticos. Los participantes podrán acceder a tutoriales, guías paso a paso y recursos adicionales que facilitarán su comprensión y aplicación del programa en diversas situaciones.

Una vez que los participantes hayan adquirido una comprensión sólida a través de la wiki, el siguiente paso será una capacitación más especializada. En esta fase, un profesor capacitado asumirá la responsabilidad de guiar a los participantes a través del uso específico de los brazos robóticos asociados con el programa. Esta capacitación se enfocará en la operación práctica de los brazos robóticos, destacando las mejores prácticas, técnicas avanzadas y resolviendo cualquier pregunta o desafío que puedan enfrentar los participantes.

La combinación de la documentación en la wiki y la capacitación práctica sobre los brazos robóticos garantizará que los participantes adquieran un conocimiento completo y práctico del programa, lo que les permitirá utilizar eficazmente la tecnología en su aplicación real.

¹<https://github.com/Patricio1Labra/SimuladorCIMUBB/wiki>

Capítulo 12

Plan de implantación y puesta en marcha

En este capítulo, se detalla el plan para implementar y poner en funcionamiento el software del brazo robótico.

12.1. Plan de implantación

La aplicación se ha concebido con la premisa de ser accesible, por lo tanto:

- (a) Se debe descargar.
- (b) Se debe descomprimir.
- (c) Utilizar.

La aparente simplicidad de esta plataforma se encuentra intrínsecamente vinculada a su principal propósito: facilitar a los estudiantes la ejecución de operaciones en los equipos del laboratorio, prescindiendo de la necesidad de su presencia física. Este enfoque, busca proporcionar una experiencia de aprendizaje remoto que permita a los participantes interactuar con los recursos de manera eficaz y sin complicaciones, garantizando así un acceso fluido a las herramientas de laboratorio desde cualquier ubicación geográfica.

A continuación, se presenta un análisis de las instrucciones para la utilización de esta plataforma. Las indicaciones, abarcan aspectos clave que orientan a los usuarios en para poder acceder a la aplicación.

Instrucciones para el uso

Descarga de la Aplicación

Este paso es común para todos los sistemas, usando el navegador de internet de preferencia, se debe acceder al repositorio de GitHub¹

- 1.- Ir al repositorio, y dar click en la sección "Releases".

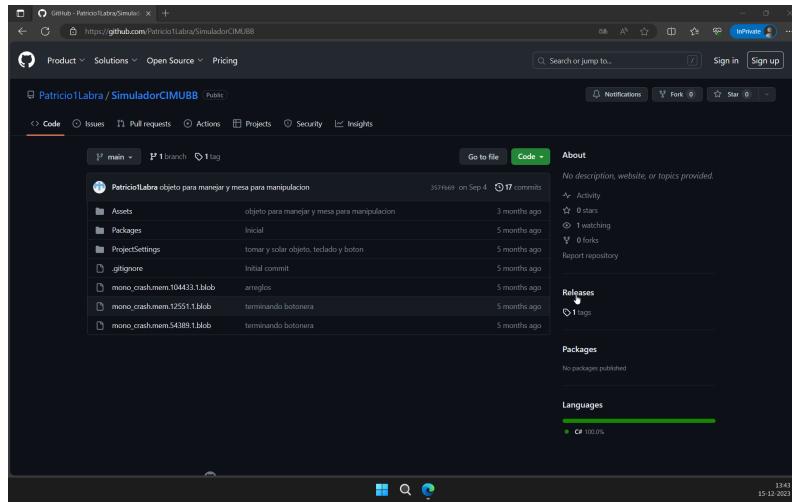


Figura 12.1: Parte 1 Tutorial

¹<https://github.com/Patricio1Labra/SimuladorCIMUBB>

2.- Estando en "Releases", buscar la ultima version, y dar click en "Assets".

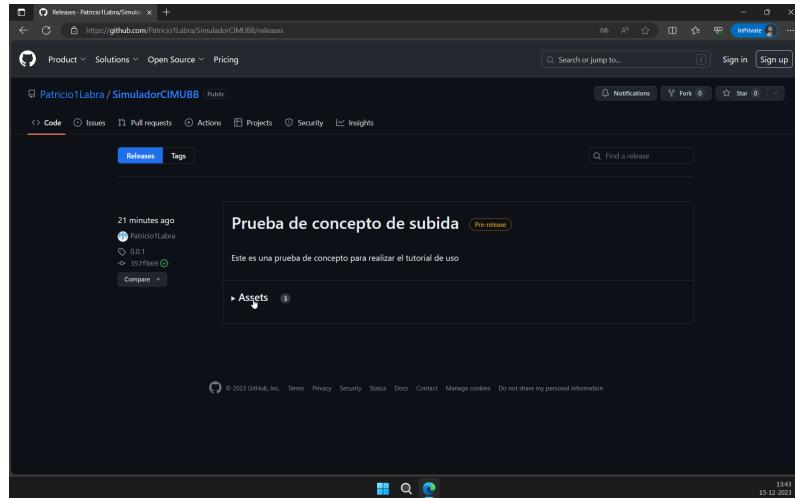


Figura 12.2: Parte 2 Tutorial

3.- Elegir el archivo correspondiente al sistema.

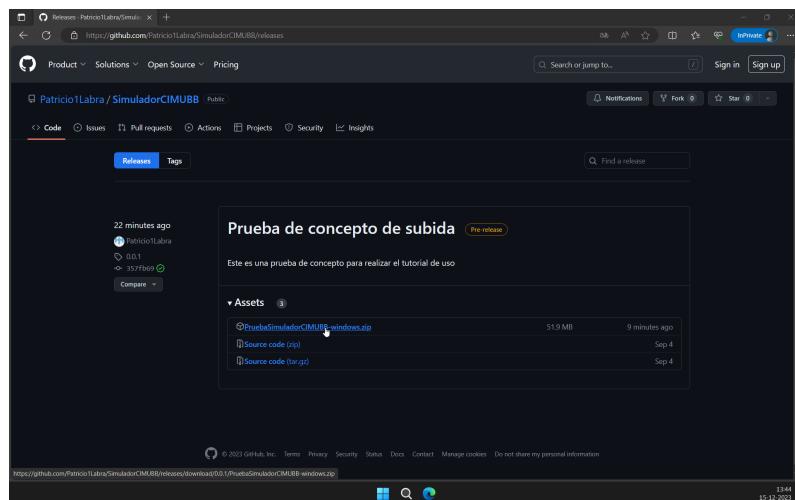


Figura 12.3: Parte 3 Tutorial

4.- Ir a la ubicación de descarga el archivo.

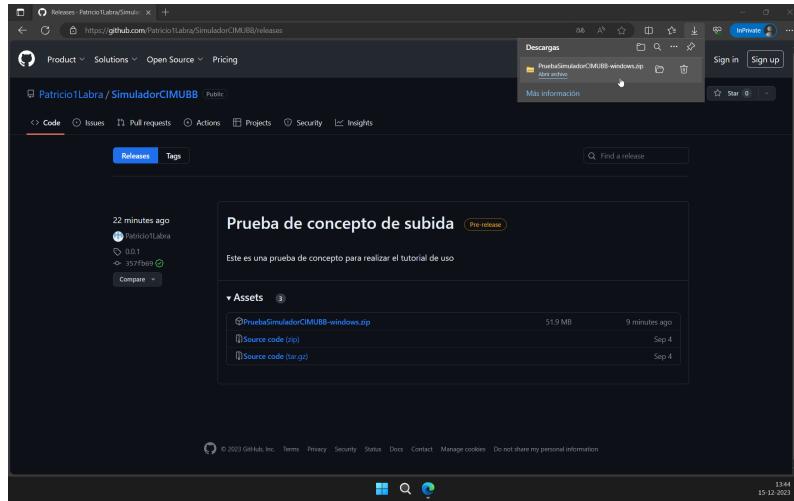


Figura 12.4: Parte 4 Tutorial

5.- Estando en esta parte, se separa dependiendo el sistema a utilizar.

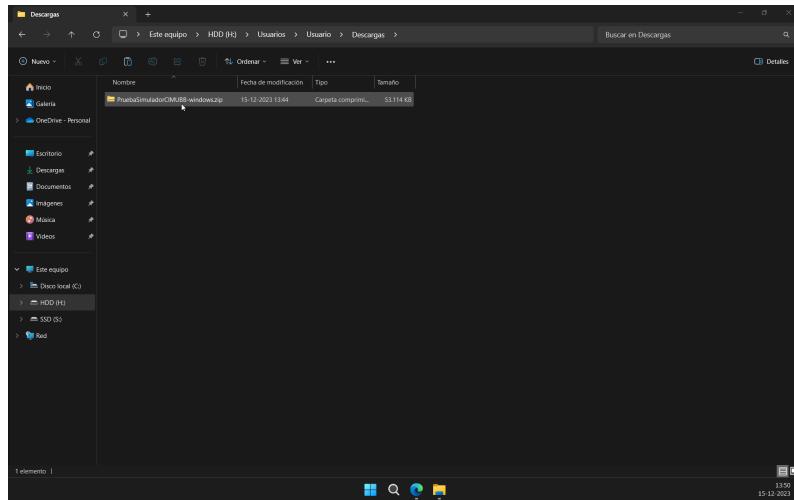


Figura 12.5: Parte 5 Tutorial

Descomprimir

Windows

- 1.- (Opcional) Descargar e instalar WinRAR u otro programa de su preferencia para descomprimir archivos ZIP. Solo si el sistema no le permite descomprimir el archivo nativamente.

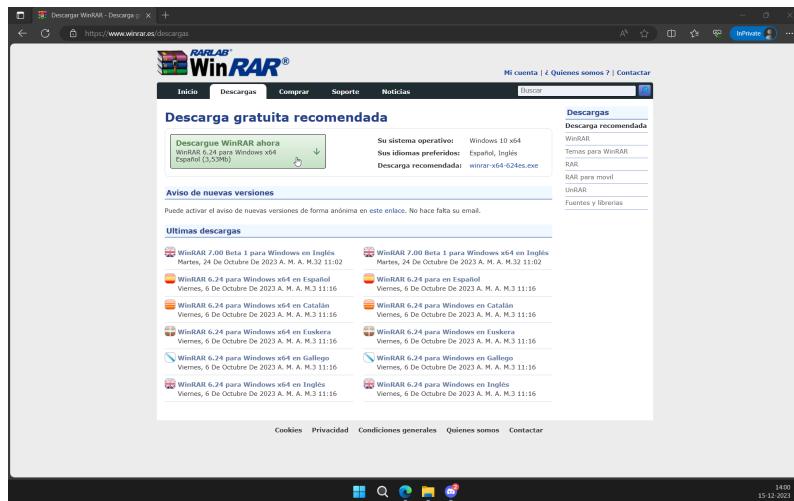


Figura 12.6: Página de descarga WinRAR

- 2.- Seleccionando el archivo descargado, elegir la opción "Extraer todo".

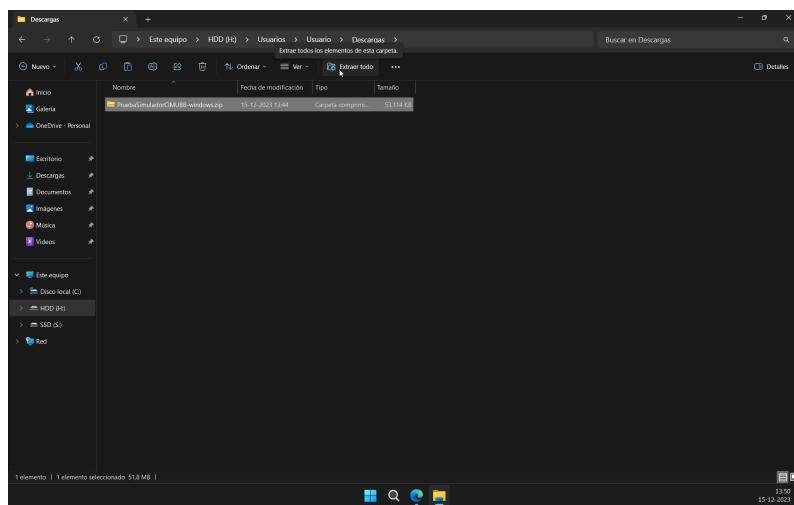


Figura 12.7: Parte 1 Tutorial Windows

3.- Darle al botón "Extraer".

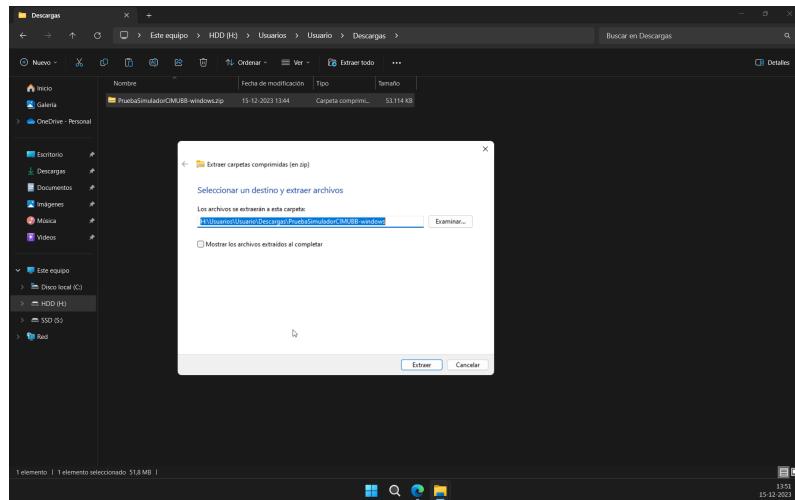


Figura 12.8: Parte 2 Tutorial Windows

4.- Abrir la nueva carpeta donde se extrajo el programa.

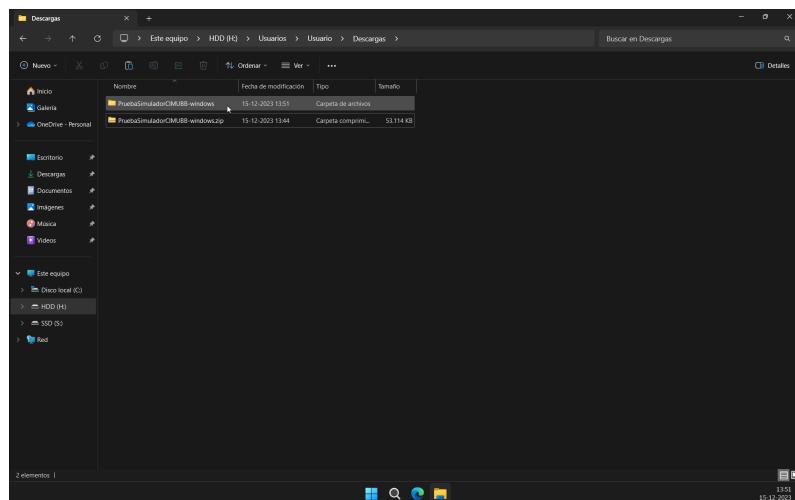


Figura 12.9: Parte 3 Tutorial Windows

5.- Ejecutar el archivo "Simulador.exe".

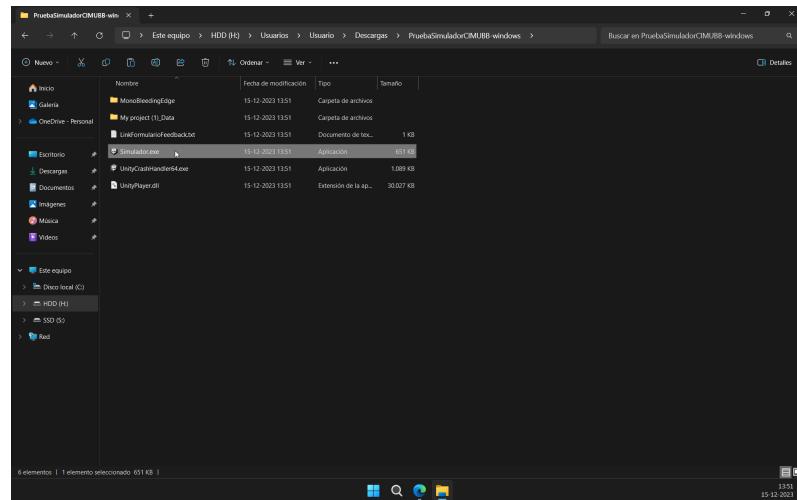


Figura 12.10: Parte 4 Tutorial Windows

6.- Algunas veces, Windows dará una advertencia, darle a "Mas información"

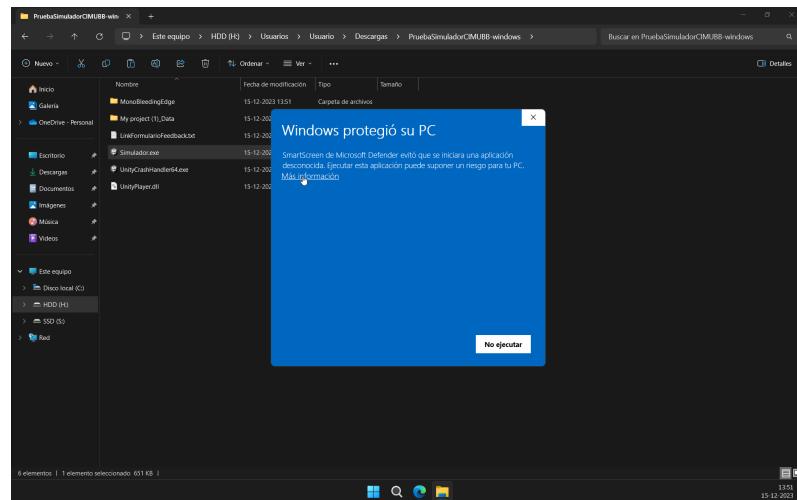


Figura 12.11: Parte 5 Tutorial Windows

7.- Dar al botón "Ejecutar de todas formas".

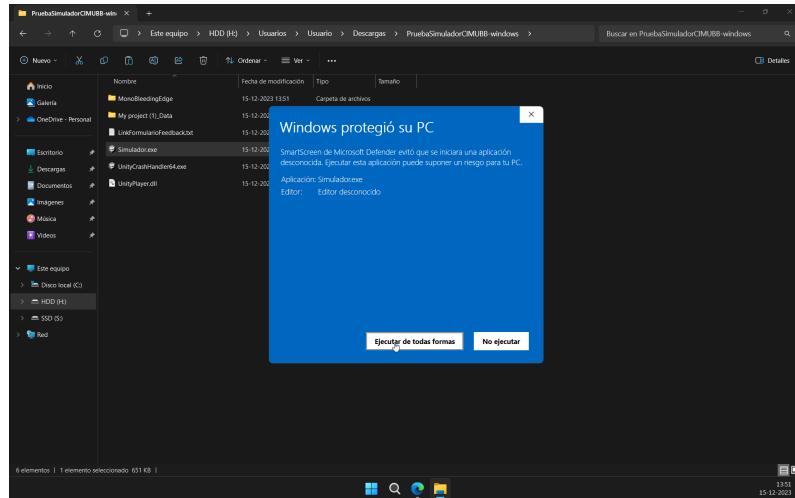


Figura 12.12: Parte 6 Tutorial Windows

8.- La aplicación iniciara.



Figura 12.13: Parte 7 Tutorial Windows

Linux

- 1.- Estando en la carpeta de descarga, se tienen dos opciones para descomprimir.

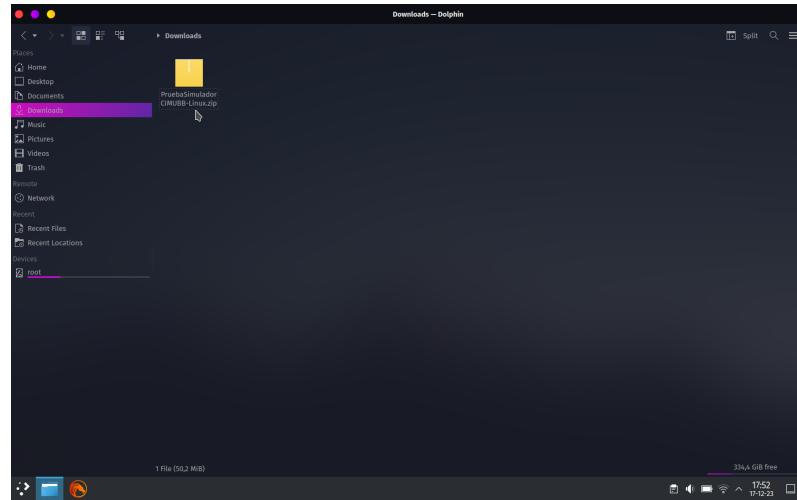


Figura 12.14: Parte 1 Tutorial Linux

- 2.- Al hacer click derecho, se da la opción de extraer.

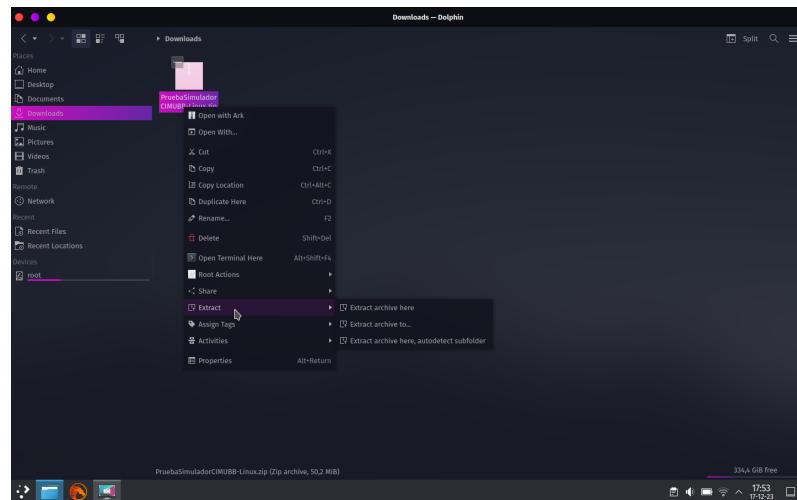


Figura 12.15: Parte 2.1 Tutorial Linux

- 3.- Si la distribución de Linux no posee esa opción, hacer click derecho para abrir la terminal en la carpeta.

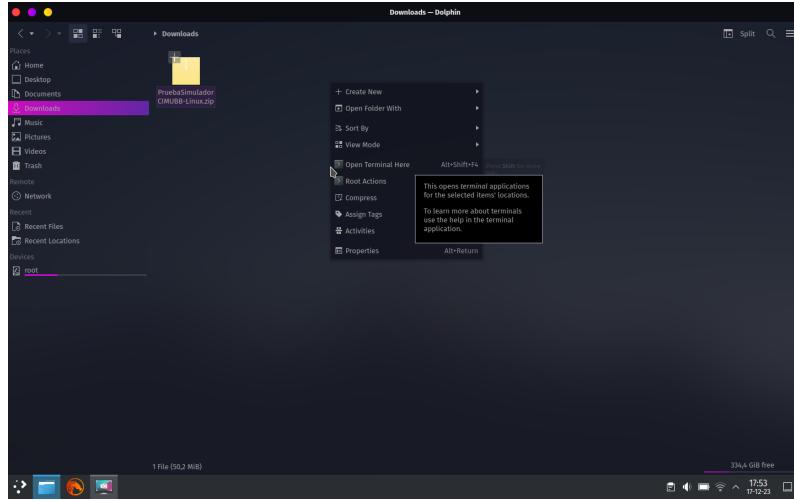


Figura 12.16: Parte 2.2.1 Tutorial Linux

- 4.- Verificar que "unzip" esta instalado usando la opción "unzip -v" (Si no se encuentra disponible, buscar el método de instalación para la distribución Linux correspondiente).

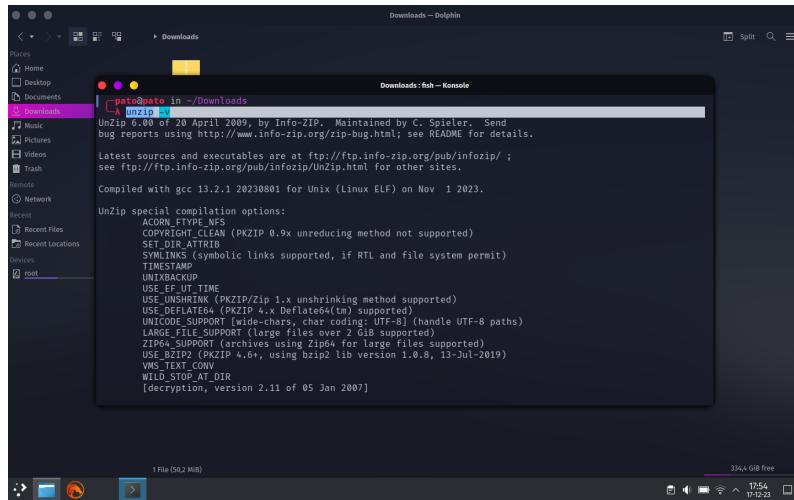


Figura 12.17: Parte 2.2.2 Tutorial Linux

5.- Ejecutar la descompresión del archivo usando ”unzip ./PruebaSimuladorCIMUBB-Linux.zip”.

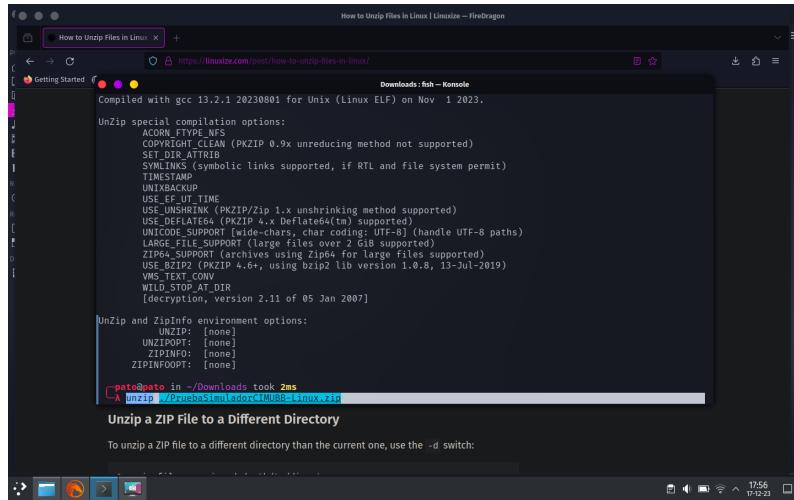


Figura 12.18: Parte 2.2.3 Tutorial Linux

6.- Abrir la carpeta extraída. De nuevo, se tienen dos opciones para realizar los siguientes pasos.

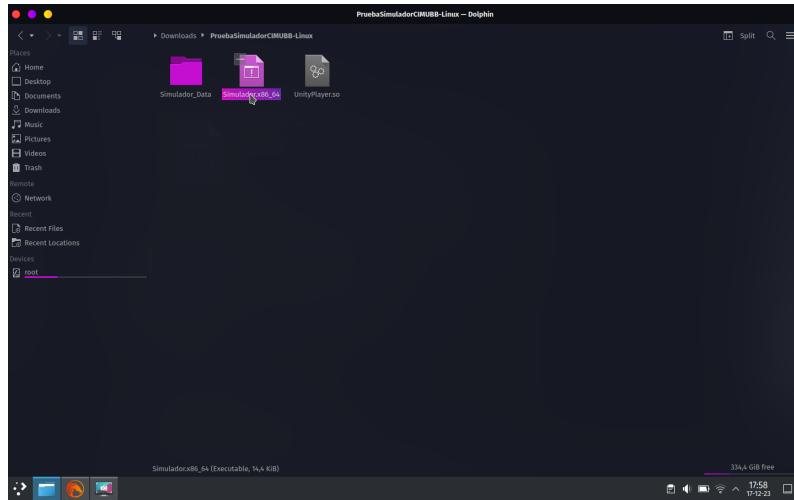


Figura 12.19: Parte 3 Tutorial Linux

7.- Hacer click derecho, ir a la opción "Propiedades".

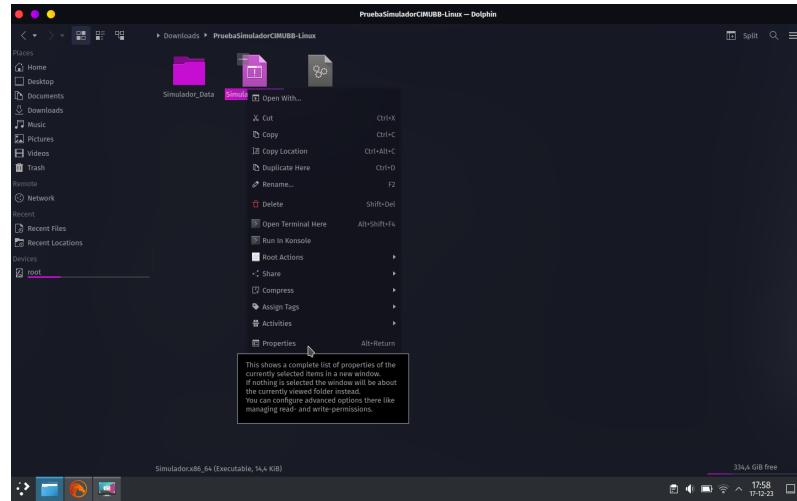


Figura 12.20: Parte 4.1.1 Tutorial Linux

8.- Ir a la pestaña "Permisos" y activar la casilla "Ejecutable".

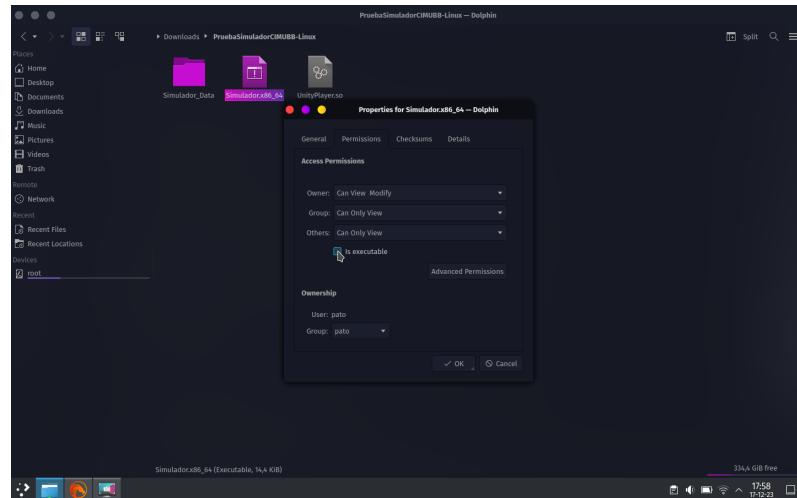


Figura 12.21: Parte 4.1.2 Tutorial Linux

9.- Como otra opción, en la consola usar "chmod +x ./Simulador.x86_64"

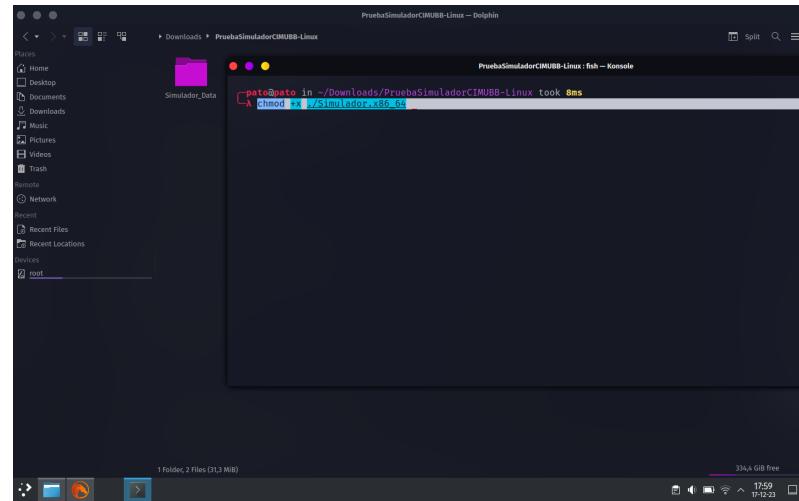


Figura 12.22: Parte 4.2 Tutorial Linux

Capítulo 13

Trabajo Futuro

En el presente capítulo, se exploran las posibles mejoras y ajustes que pueden implementarse para lograr una representación más precisa y fidedigna del programa. Se abordan aspectos que permiten perfeccionar la integridad y la coherencia de la representación, buscando así optimizar el rendimiento y la comprensión del programa en cuestión.

13.1. Cambios que se pueden realizar

- **Implementar mas funciones y equipamiento:** La principal premisa consistía en emplear los brazos robóticos sin necesidad de acudir al laboratorio ni contar con acceso directo al mismo. En consecuencia, se otorgó prioridad a la funcionalidad de dichos brazos, aunque esto ha implicado la carencia de equipo funcional dentro del laboratorio para lograr una simulación más precisa. Además, los brazos no poseen todas las funciones operativas.
- **Agregar una guía de uso:** Es viable desarrollar una guía interactiva que explique las funcionalidades básicas de los robots, con el objetivo de hacer la información más accesible y llegar a un público más amplio, especialmente a aquellos que no están familiarizados con el tema y desean aprender.
- **Mejorar modelos:** Uno de los cambios mas fundamentales que se debe realizar para mejorar la experiencia e intentar lograr ser lo mas fiel posible en la representación del laboratorio. Se puede empezar por realizar un modelado fiel y a escala real de los objetos presentes en el laboratorio, para esto se presenta el programa que provee Intelitek para su brazo Scrbot ER-4U. En la Figura 13.1 se presenta la interfaz del programa RoboCell, el cual sirve para darle instrucciones al brazo robótico, ya sea real o simulado, en este ultimo se ve el detallado del modelo que da mejor sensación de realismo.

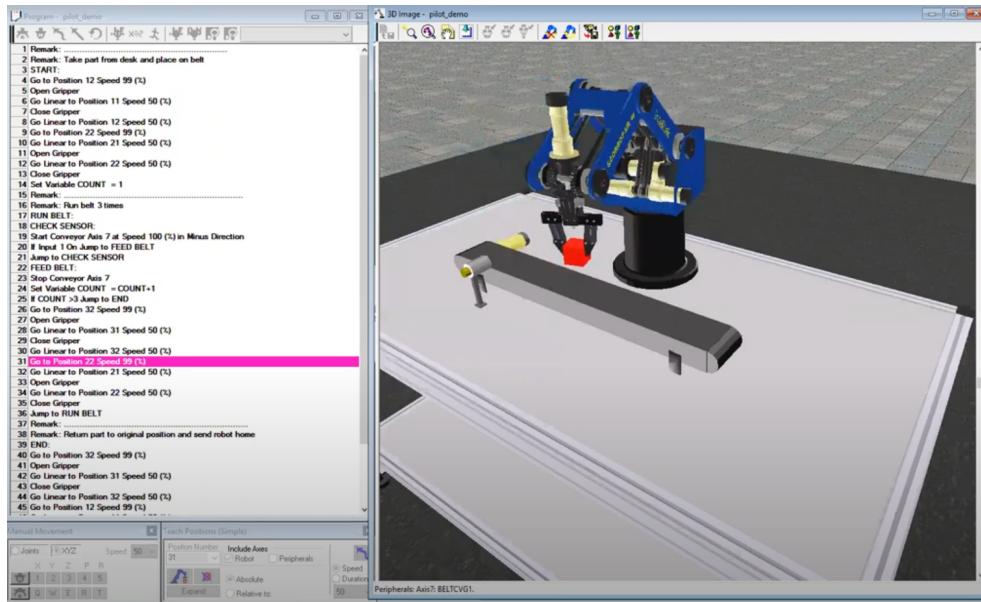


Figura 13.1: Diseño interfaz: Módulo Principal

Estas son solo algunas ideas iniciales que pueden implementarse. La simulación no tiene límites, y el programa puede adaptarse para diversas finalidades, como la integración con la realidad virtual. Esto permitiría ampliar aún más las posibilidades de aprendizaje y entrenamiento, ofreciendo una experiencia inmersiva que podría abarcar desde el manejo básico de los brazos robóticos hasta aplicaciones más avanzadas en entornos simulados. Además, se podría explorar la opción de incorporar características de interactividad, permitiendo a los usuarios realizar prácticas y experimentos virtuales para consolidar sus conocimientos de manera práctica y segura. La flexibilidad del programa ofrece un amplio espectro de oportunidades para su expansión y adaptación a diferentes necesidades y contextos.

Capítulo 14

Conclusiones

A lo largo de este proyecto, he podido comprender la importancia fundamental de mejorar la educación a través de la simulación [4]. Aunque la simulación no puede replicar completamente la experiencia real, he observado que puede acercarse lo suficiente como para tener un impacto significativo en el proceso de aprendizaje. La capacidad de crear situaciones virtualmente similares a la realidad resulta especialmente relevante para eliminar barreras educativas, proporcionando alternativas accesibles y asequibles para personas con diversas limitaciones, como problemas de movilidad, foto-sensibilidad, discapacidad y otros obstáculos.

El enfoque de la simulación va más allá de simplemente recrear situaciones en un entorno virtual. Su verdadero potencial radica en la creación de un ambiente inclusivo donde todos puedan acceder a oportunidades educativas de manera efectiva, independientemente de sus capacidades físicas o cognitivas. Al adoptar esta herramienta en el ámbito educativo, se abren puertas para aquellas personas que, de otro modo, enfrentarían dificultades para participar en experiencias de aprendizaje costosas o restringidas.

En resumen, la simulación en la educación tiene el poder de promover una sociedad más equitativa e inclusiva, permitiendo que cada individuo alcance su máximo potencial. Aunque es un campo en desarrollo, espero que esta tesis inspire a educadores, instituciones académicas y desarrolladores tecnológicos a utilizar la simulación como una herramienta transformadora en la búsqueda de una educación para todos, sin barreras ni limitaciones. Al hacerlo, construiremos un futuro en el que la adquisición de conocimientos y habilidades sea una posibilidad accesible para cada persona, contribuyendo así al avance y prosperidad de la sociedad en su conjunto.

Por otra parte, el proyecto es bastante ambicioso para ser llevado a cabo por una sola persona en el tiempo limitado asignado. Por lo tanto es posible que con un equipo más extenso o con un plazo más amplio, se puedan abordar todas las mejoras mencionadas. Este análisis subraya la importancia de contar con recursos adecuados y tiempo suficiente al abordar iniciativas de esta magnitud para garantizar resultados más satisfactorios.

El código está anexado y el proyecto se encuentra alojado en el repositorio de GitHub ¹, para que cualquiera pueda trabajar en él.

¹<https://github.com/Patricio1Labra/SimuladorCIMUBB>

Referencias

- [1] G. Salas, P. Santander, A. Precht, H. Scholten, R. Moretti, and W. López-López. Covid-19: impacto psicosocial en la escuela en chile. desigualdades y desafíos para latinoamérica. *Avances En Psicología Latinoamericana*, 38(2), 2020.
- [2] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, Boston, MA, USA, 2000.
- [3] Cámara de Diputados de Chile. El servicio de internet en chile y la implementación de las nuevas tecnologías. <https://www.camara.cl/verDoc.aspx?prmTipo=DOCASEXTERNA&prmId=1215>, 2022.
- [4] Zulma Cataldi, Fernando J Lage, and Claudio Dominighini. Fundamentos para el uso de simulaciones en la enseñanza. *Revista de informática educativa y medios audiovisuales*, 10(17):8–16, 2013.

Apéndice A

Código: Belt.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Belt : MonoBehaviour
{
    public float speed = 5f;
    Rigidbody rBody;
    public Vector3 direccion;

    void Start()
    {
        rBody = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        Vector3 pos = rBody.position;
        rBody.position += direccion * speed * Time.fixedDeltaTime;
        rBody.MovePosition(pos);
    }
}
```

Apéndice B

Código: MovimientoJoints IX.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class MovimientoJointsIX : MonoBehaviour
{
    //Base
    private float velocidadRotacionBase = 80.0f; // Velocidad de rotacion
    public KeyCode teclaRotacionPositivaBase = KeyCode.Q; // Tecla para la ←
        rotacion en direccion positiva
    public KeyCode teclaRotacionNegativaBase = KeyCode.A; // Tecla para la ←
        rotacion en direccion negativa
    public Transform Base; // Transform del objeto que se va a rotar
    private float LimitePositivoBase = 135.0f; // Angulo limite del objeto
    private float LimiteNegativoBase = -135.0f; // Angulo limite del objeto
    private float sumaRotacionBase = 0.0f; // Suma de la rotacion del objeto←
        para su uso en limites

    //Shoulder
    private float velocidadRotacionShoulder = 69.0f; // Velocidad de ←
        rotacion
    public KeyCode teclaRotacionPositivaShoulder = KeyCode.W; // Tecla para ←
        la rotacion en direccion positiva
    public KeyCode teclaRotacionNegativaShoulder = KeyCode.S; // Tecla para ←
        la rotacion en direccion negativa
    public Transform Shoulder; // Transform del objeto a rotar
    public Transform puntoFijoShoulder; // Transform del punto fijo ←
        alrededor del cual se realizara la rotacion
    private float LimitePositivoShoulder = 72.5f; // Angulo limite del ←
        objeto
    private float LimiteNegativoShoulder = -72.5f; // Angulo limite del ←
        objeto
```

```
private float sumaRotacionShoulder = 0.0f; // Suma de la rotacion del ←
    objeto para su uso en limites

//Elbow
private float velocidadRotacionElbow = 77.0f; // Velocidad de rotacion
public KeyCode teclaRotacionPositivaElbow = KeyCode.E; // Tecla para la ←
    rotacion en direccion positiva
public KeyCode teclaRotacionNegativaElbow = KeyCode.D; // Tecla para la ←
    rotacion en direccion negativa
public Transform Elbow; // Transform del objeto a rotar
public Transform puntoFijoElbow; // Transform del punto fijo alrededor ←
    del cual se realizara la rotacion
private float LimitePositivoElbow = 105.0f; // Angulo limite del objeto
private float LimiteNegativoElbow = -105.0f; // Angulo limite del objeto
private float sumaRotacionElbow = 0.0f; // Suma de la rotacion del ←
    objeto para su uso en limites

//Wrist
private float velocidadRotacionWrist = 103.0f; // Velocidad de rotacion
public KeyCode teclaRotacionPositivaWrist = KeyCode.R; // Tecla para la ←
    rotacion en direccion positiva
public KeyCode teclaRotacionNegativaWrist = KeyCode.F; // Tecla para la ←
    rotacion en direccion negativa
public Transform Wrist; // Transform del objeto a rotar
public Transform puntoFijoWrist; // Transform del punto fijo alrededor ←
    del cual se realizara la rotacion
private float LimitePositivoWrist= 98.0f; // Angulo limite del objeto
private float LimiteNegativoWrist = -98.0f; // Angulo limite del objeto
private float sumaRotacionWrist = 0.0f; // Suma de la rotacion del ←
    objeto para su uso en limites

//EndEffector
private float velocidadRotacionEndEffector = 175.0f; // Velocidad de ←
    rotacion
public KeyCode teclaRotacionPositivaEndEffector = KeyCode.T; // Tecla ←
    para la rotacion en direccion positiva
public KeyCode teclaRotacionNegativaEndEffector = KeyCode.G; // Tecla ←
    para la rotacion en direccion negativa
public Transform EndEffector; // Transform del objeto a rotar
public Transform puntoFijoEndEffector; // Transform del punto fijo ←
    alrededor del cual se realizara la rotacion
private float LimitePositivoEndEffector = 368.5f; // Angulo limite del ←
    objeto
private float LimiteNegativoEndEffector = -368.5f; // Angulo limite del ←
    objeto
private float sumaRotacionEndEffector = 0.0f; // Suma de la rotacion del←
    objeto para su uso en limites

//Botonera
private string nombreObjeto = "";
private bool positivoboton = false;
private bool negativoboton = false;
```

```
private bool positivobotonZ = false;
private bool negativobotonZ = false;

private bool isControlPressed = false;
private bool isAltPressed = false;

private bool EmpezarGuardar = false;
private bool TerminarGuardar = false;
private bool EmpezarMover = false;
private bool TerminarMover = false;
private int NumeroUsar = -1;

private Dictionary<int , float> GuardarBase = new Dictionary<int , float <-
    >();
private Dictionary<int , float> GuardarShoulder = new Dictionary<int , <-
    float >();
private Dictionary<int , float> GuardarElbow = new Dictionary<int , float <-
    >();
private Dictionary<int , float> GuardarWrist = new Dictionary<int , float <-
    >();
private Dictionary<int , float> GuardarEndEffector = new Dictionary<int , <-
    float >();

private void Start()
{
    GuardarBase.Add(0, 0f);
    GuardarShoulder.Add(0,0f);
    GuardarElbow.Add(0,0f);
    GuardarWrist.Add(0,0f);
    GuardarEndEffector.Add(0,0f);
}

private void Update()
{
    Movimiento(Base, teclaRotacionPositivaBase, <-
        teclaRotacionNegativaBase, velocidadRotacionBase, <-
        LimitePositivoBase, LimiteNegativoBase, Base, Base.up, ref <-
        sumaRotacionBase);
    Movimiento(Shoulder, teclaRotacionPositivaShoulder, <-
        teclaRotacionNegativaShoulder, velocidadRotacionShoulder, <-
        LimitePositivoShoulder, LimiteNegativoShoulder, <-
        puntoFijoShoulder, puntoFijoShoulder.forward, ref <-
        sumaRotacionShoulder);
    Movimiento(Elbow, teclaRotacionPositivaElbow, <-
        teclaRotacionNegativaElbow, velocidadRotacionElbow, <-
        LimitePositivoElbow, LimiteNegativoElbow, puntoFijoElbow, <-
        puntoFijoElbow.forward, ref sumaRotacionElbow);
    Movimiento(Wrist, teclaRotacionPositivaWrist, <-
        teclaRotacionNegativaWrist, velocidadRotacionWrist, <-
        LimitePositivoWrist, LimiteNegativoWrist, puntoFijoWrist, Wrist.<-
        forward, ref sumaRotacionWrist);
```

```
Movimiento(EndEffector, teclaRotacionPositivaEndEffector, ←
    teclaRotacionNegativaEndEffector, velocidadRotacionEndEffector, ←
    LimitePositivoEndEffector, LimiteNegativoEndEffector, ←
    puntoFijoEndEffector, EndEffector.right, ref ←
    sumaRotacionEndEffector);
MovimientoBoton(nombreObjeto);
Guardar();
Usar();
GuardarBoton();
UsarBoton();
MoverEje();
MoverEjeBoton();

}

private void Movimiento(Transform objeto, KeyCode TeclaPositiva, KeyCode←
    TeclaNegativa, float VelocidadRotacion, float anguloLimitePositivo,←
    float anguloLimiteNegativo, Transform puntoFijo, Vector3 direccion,←
    ref float Suma)
{
    // Obtener la direccion de rotacion basada en las teclas presionadas
    float direccionRotacion = 0.0f;

    if (Input.GetKey(TeclaPositiva))
    {
        direccionRotacion = 1.0f;
    }
    else if (Input.GetKey(TeclaNegativa))
    {
        direccionRotacion = -1.0f;
    }

    // Verificar si no se estan presionando teclas de rotacion
    if (direccionRotacion == 0.0f)
    {
        return;
    }
    // Calcular el angulo de rotacion
    float anguloRotacion = direccionRotacion * VelocidadRotacion * Time.←
        deltaTime;

    // Calcular el angulo de rotacion limitado
    float anguloRotacionLimitado = Mathf.Clamp(Suma, ←
        anguloLimiteNegativo, anguloLimitePositivo);
    if (anguloRotacionLimitado == anguloLimiteNegativo && ←
        direccionRotacion < 0)
    {
        anguloRotacion = 0.0f;
    }
}
```

```
if (anguloRotacionLimitado == anguloLimitePositivo && ↵
    direccionRotacion > 0)
{
    anguloRotacion = 0.0f;
}
// Realizar la rotacion alrededor del punto fijo
objeto.RotateAround(puntoFijo.position, direccion, anguloRotacion);
Suma += anguloRotacion;
Suma = Mathf.Clamp(Suma, anguloLimiteNegativo, anguloLimitePositivo)←
;
}

private void MovimientoBoton(string nombre)
{
    // Obtener la direccion de rotacion basada en las teclas presionadas
    float direccionRotacionboton = 0.0f;

    if (positivoboton)
    {
        direccionRotacionboton = 1.0f;
    }
    if (negativoboton)
    {
        direccionRotacionboton = -1.0f;
    }

    // Verificar si no se estan presionando teclas de rotacion
    if (direccionRotacionboton == 0.0f)
    {
        return;
    }
    float Sumar = 0;
    Vector3 direccion = Vector3.up;
    float anguloLimitePositivo = 0;
    float anguloLimiteNegativo = 0;
    float VelocidadRotacion = 0;
    Transform puntoFijo = Base;
    Transform objeto = Base;

    if (nombre == "Base")
    {
        Sumar = sumaRotacionBase;
        direccion = Base.up;
        anguloLimitePositivo = LimitePositivoBase;
        anguloLimiteNegativo = LimiteNegativoBase;
        VelocidadRotacion = velocidadRotacionBase;
        puntoFijo = Base;
        objeto = Base;
    }

    if (nombre == "Shoulder")
```

```
{  
    Sumar = sumaRotacionShoulder;  
    direccion = puntoFijoShoulder.forward;  
    anguloLimitePositivo = LimitePositivoShoulder;  
    anguloLimiteNegativo = LimiteNegativoShoulder;  
    VelocidadRotacion = velocidadRotacionShoulder;  
    puntoFijo = puntoFijoShoulder;  
    objeto = Shoulder;  
}  
  
if (nombre == "Elbow")  
{  
    Sumar = sumaRotacionElbow;  
    direccion = puntoFijoElbow.forward;  
    anguloLimitePositivo = LimitePositivoElbow;  
    anguloLimiteNegativo = LimiteNegativoElbow;  
    VelocidadRotacion = velocidadRotacionElbow;  
    puntoFijo = puntoFijoElbow;  
    objeto = Elbow;  
}  
  
if (nombre == "Wrist")  
{  
    Sumar = sumaRotacionWrist;  
    direccion = Wrist.up;  
    anguloLimitePositivo = LimitePositivoWrist;  
    anguloLimiteNegativo = LimiteNegativoWrist;  
    VelocidadRotacion = velocidadRotacionWrist;  
    puntoFijo = Wrist;  
    objeto = Wrist;  
}  
  
if (nombre == "EndEffector")  
{  
    Sumar = sumaRotacionEndEffector;  
    direccion = EndEffector.up;  
    anguloLimitePositivo = LimitePositivoEndEffector;  
    anguloLimiteNegativo = LimiteNegativoEndEffector;  
    VelocidadRotacion = velocidadRotacionEndEffector;  
    puntoFijo = EndEffector;  
    objeto = EndEffector;  
}  
  
// Calcular el angulo de rotacion  
float anguloRotacion = direccionRotacionboton * VelocidadRotacion * ←  
    Time.deltaTime;  
  
// Calcular el angulo de rotacion limitado  
float anguloRotacionLimitado = Mathf.Clamp(Sumar, ←  
    anguloLimiteNegativo, anguloLimitePositivo);  
if (anguloRotacionLimitado == anguloLimiteNegativo && ←  
    direccionRotacionboton < 0)
```

```
{  
    anguloRotacion = 0.0f;  
}  
  
if (anguloRotacionLimitado == anguloLimitePositivo && ↵  
    direccionRotacionboton > 0)  
{  
    anguloRotacion = 0.0f;  
}  
// Realizar la rotacion alrededor del punto fijo  
objeto.RotateAround(puntoFijo.position, direccion, anguloRotacion);  
Sumar += anguloRotacion;  
Sumar = Mathf.Clamp(Sumar, anguloLimiteNegativo, ↵  
                    anguloLimitePositivo);  
  
if (nombre == "Base")  
{  
    sumaRotacionBase = Sumar;  
}  
  
if (nombre == "Shoulder")  
{  
    sumaRotacionShoulder = Sumar;  
}  
  
if (nombre == "Elbow")  
{  
    sumaRotacionElbow = Sumar;  
}  
  
if (nombre == "Wrist")  
{  
    sumaRotacionWrist = Sumar;  
}  
if (nombre == "EndEffector")  
{  
    sumaRotacionEndEffector = Sumar;  
}  
}  
  
}  
  
public void PointerUpPositivo(string nombre)  
{  
    nombreObjeto = nombre;  
    positivoboton = false;  
}  
  
public void PointerDownPositivo(string nombre)  
{  
    nombreObjeto = nombre;  
    positivoboton = true;  
}
```

```
}

public void PointerUpNegativo(string nombre)
{
    nombreObjeto = nombre;
    negativoboton = false;
}

public void PointerDownNegativo(string nombre)
{
    nombreObjeto = nombre;
    negativoboton = true;
}

public void PointerUpPositivoZ()
{
    if (!EmpezarGuardar && !EmpezarMover)
    {
        positivobotonZ = false;
    }
}

public void PointerDownPositivoZ()
{
    if (!EmpezarGuardar && !EmpezarMover)
    {
        positivobotonZ = true;
    }
}

public void PointerUpNegativoZ()
{
    if (!EmpezarGuardar && !EmpezarMover)
    {
        negativobotonZ = false;
    }
}

public void PointerDownNegativoZ()
{
    if (!EmpezarGuardar && !EmpezarMover)
    {
        negativobotonZ = true;
    }
}

private void Guardar()
{
    // Verificar si se presiono la tecla de control (Ctrl)
    if (Input.GetKeyDown(KeyCode.LeftControl) || Input.GetKeyDown(KeyCode.RightControl))
    {
```

```
        isControlPressed = true;
    }

    // Verificar si esta habilitado el registro de teclas
    if (isControlPressed)
    {
        // Verificar las teclas numericas del 1 al 9
        for (int i = 1; i <= 9; i++)
        {
            if (Input.GetKey(i.ToString()))
            {
                SaveInformation(i);
                isControlPressed = false;
            }
        }
    }

    private void GuardarBoton()
    {
        // Verificar si se presiono el boton
        if (EmpezarGuardar)
        {
            // Verificar si se habilita el guardado
            if (TerminarGuardar)
            {
                SaveInformation(NumeroUsar);
                TerminarGuardar = false;
                EmpezarGuardar = false;
                NumeroUsar = -1;
            }
        }
    }

    private void Usar()
    {
        // Verificar si se presiono la tecla de control (Ctrl)
        if (Input.GetKey(KeyCode.LeftAlt) || Input.GetKey(KeyCode.RightAlt))
        {
            isAltPressed = true;
        }

        // Verificar si esta habilitado el registro de teclas
        if (isAltPressed)
        {
            // Verificar las teclas numericas del 0 al 9
            for (int i = 0; i <= 9; i++)
            {
                if (Input.GetKey(i.ToString()))
                {
                    GetInformation(i);
                    isAltPressed = false;
                }
            }
        }
    }
```

```
        }
    }
}

private void UsarBoton()
{
    // Verificar si se presiono el boton
    if (EmpezarMover)
    {
        // Verificar si se habilita el guardado
        if (TerminarMover)
        {
            GetInformation(Numerousar);
            TerminarMover = false;
            EmpezarMover = false;
            NumeroUsar = -1;
        }
    }
}

private void SaveInformation(int i)
{
    if (GuardarBase.ContainsKey(i))
    {
        GuardarBase[i] = sumaRotacionBase;
    }
    else
    {
        GuardarBase.Add(i, sumaRotacionBase);
    }

    if (GuardarShoulder.ContainsKey(i))
    {
        GuardarShoulder[i] = sumaRotacionShoulder;
    }
    else
    {
        GuardarShoulder.Add(i, sumaRotacionShoulder);
    }

    if (GuardarElbow.ContainsKey(i))
    {
        GuardarElbow[i] = sumaRotacionElbow;
    }
    else
    {
        GuardarElbow.Add(i, sumaRotacionElbow);
    }

    if (GuardarWrist.ContainsKey(i))
    {
```

```
        GuardarWrist[i] = sumaRotacionWrist;
    }
    else
    {
        GuardarWrist.Add(i, sumaRotacionWrist);
    }

    if (GuardarEndEffector.ContainsKey(i))
    {
        GuardarEndEffector[i] = sumaRotacionEndEffector;
    }
    else
    {
        GuardarEndEffector.Add(i, sumaRotacionEndEffector);
    }
}

private void GetInformation(int i)
{
    if (GuardarBase.ContainsKey(i))
    {
        float value = GuardarBase[i];
        Debug.Log("Base " + i + " = " + value);
        //MoverGuardado(value, sumaRotacionBase, Base, Base, ←
        //LimiteNegativoBase, LimitePositivoBase, velocidadRotacionBase ←
        , -Base.up);
    }
    else
    {
        Debug.Log("no existe");
    }

    if (GuardarShoulder.ContainsKey(i))
    {
        float value = GuardarShoulder[i];
        Debug.Log("Shoulder " + i + " = " + value);
    }
    else
    {
        Debug.Log("no existe");
    }

    if (GuardarElbow.ContainsKey(i))
    {
        float value = GuardarElbow[i];
        Debug.Log("Elbow " + i + " = " + value);
    }
    else
    {
        Debug.Log("no existe");
    }
}
```

```
if (GuardarWrist.ContainsKey(i))
{
    float value = GuardarWrist[i];
    Debug.Log("Wrist "+ i + " = " + value);
}
else
{
    Debug.Log("no existe");
}

if (GuardarEndEffector.ContainsKey(i))
{
    float value = GuardarEndEffector[i];
    Debug.Log("EndEffector "+ i + " = " + value);
}
else
{
    Debug.Log("no existe");
}

public void EmpezarAGuardar()
{
    EmpezarGuardar = true;
}

public void NumeroAUsar(int i)
{
    if(EmpezarGuardar || EmpezarMover)
    {
        NumeroUsar = i;
    }
}

public void TerminarAGuardar()
{
    if(0 < NumeroUsar && NumeroUsar < 10)
    {
        if(EmpezarGuardar)
        {
            TerminarGuardar = true;
        }
    }
}

public void EmpezarAMover()
{
    EmpezarMover = true;
}

public void TerminarAMover()
{
```

```
if(-1 < NumeroUsar && NumeroUsar < 10)
{
    if(EmpezarMover)
    {
        TerminarMover = true;
    }
}

private void MoverEje()
{
    float movimiento = 0f;

    // Verifica si se presiona la tecla positiva
    if (Input.GetKeyDown(KeyCode.Alpha1))
    {
        movimiento = 1f;
    }

    // Verifica si se presiona la tecla negativa
    if (Input.GetKeyDown(KeyCode.Alpha6))
    {
        movimiento = -1f;
    }

    // Mueve el objeto en el eje Y (o el eje que deseas) segun el valor ←
    // de movimiento
    transform.Translate(Vector3.right * movimiento * 5f * Time.deltaTime←
        );
}

private void MoverEjeBoton()
{
    float movimiento = 0f;

    if (positivobotonZ)
    {
        movimiento = 1.0f;
    }
    if (negativobotonZ)
    {
        movimiento = -1.0f;
    }

    // Mueve el objeto en el eje Y (o el eje que deseas) segun el valor ←
    // de movimiento
    transform.Translate(Vector3.right * movimiento * 5f * Time.deltaTime←
        );
}
```

```
private void MoverGuardado( float valor, float Suma, Transform puntoFijo, ←
    Transform objeto, float anguloLimiteNegativo, float ←
    anguloLimitePositivo, float VelocidadRotacion, Vector3 direccion)
{
    float direccionRotacion = 0.0f;

    if (Suma < valor)
    {
        direccionRotacion = 1.0f;
    }
    else if (Suma > valor)
    {
        direccionRotacion = -1.0f;
    }

    if (direccionRotacion == 0.0f)
    {
        return;
    }
    // Calcular el angulo de rotacion

    // Realizar la rotacion alrededor del punto fijo
    while(Suma != valor)
    {
        float anguloRotacion = direccionRotacion * VelocidadRotacion * ←
            Time.deltaTime;
        objeto.RotateAround(puntoFijo.position, direccion, ←
            anguloRotacion);
        Suma += anguloRotacion;
        if(direccionRotacion == 1.0f)
        {
            Suma = Mathf.Clamp(Suma, anguloLimiteNegativo, valor);
        }
        else
        {
            Suma = Mathf.Clamp(Suma, valor, anguloLimitePositivo);
        }
    }
}
```

Apéndice C

Código: MovimientoJoints V.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class MovimientoJointsV : MonoBehaviour
{
    //Base
    public float velocidadRotacionBase = 50.0f; // Velocidad de rotacion
    public KeyCode teclaRotacionPositivaBase = KeyCode.Q; // Tecla para la ←
        rotacion en direccion positiva
    public KeyCode teclaRotacionNegativaBase = KeyCode.A; // Tecla para la ←
        rotacion en direccion negativa
    public Transform Base; // Transform del objeto que se va a rotar
    private float LimitePositivoBase = 155.0f; // Angulo limite del objeto
    private float LimiteNegativoBase = -155.0f; // Angulo limite del objeto
    private float sumaRotacionBase = 0.0f; // Suma de la rotacion del objeto←
        para su uso en limites

    //Shoulder
    public float velocidadRotacionShoulder = 50.0f; // Velocidad de rotacion
    public KeyCode teclaRotacionPositivaShoulder = KeyCode.W; // Tecla para ←
        la rotacion en direccion positiva
    public KeyCode teclaRotacionNegativaShoulder = KeyCode.S; // Tecla para ←
        la rotacion en direccion negativa
    public Transform Shoulder; // Transform del objeto a rotar
    public Transform puntoFijoShoulder; // Transform del punto fijo ←
        alrededor del cual se realizara la rotacion
    private float LimitePositivoShoulder = 10.0f; // Angulo limite del ←
        objeto
    private float LimiteNegativoShoulder = -130.0f; // Angulo limite del ←
        objeto
    private float sumaRotacionShoulder = 0.0f; // Suma de la rotacion del ←
        objeto para su uso en limites
```

```
//Elbow
public float velocidadRotacionElbow = 50.0f; // Velocidad de rotacion
public KeyCode teclaRotacionPositivaElbow = KeyCode.E; // Tecla para la ←
    rotacion en direccion positiva
public KeyCode teclaRotacionNegativaElbow = KeyCode.D; // Tecla para la ←
    rotacion en direccion negativa
public Transform Elbow; // Transform del objeto a rotar
public Transform puntoFijoElbow; // Transform del punto fijo alrededor ←
    del cual se realizara la rotacion
private float LimitePositivoElbow = 130.0f; // Angulo limite del objeto
private float LimiteNegativoElbow = -130.0f; // Angulo limite del objeto
private float sumaRotacionElbow = 0.0f; // Suma de la rotacion del ←
    objeto para su uso en limites

//Wrist
public float velocidadRotacionWrist = 50.0f; // Velocidad de rotacion
public KeyCode teclaRotacionPositivaWrist = KeyCode.R; // Tecla para la ←
    rotacion en direccion positiva
public KeyCode teclaRotacionNegativaWrist = KeyCode.F; // Tecla para la ←
    rotacion en direccion negativa
public Transform Wrist; // Transform del objeto a rotar
private float LimitePositivoWrist= 130.0f; // Angulo limite del objeto
private float LimiteNegativoWrist = -130.0f; // Angulo limite del objeto
private float sumaRotacionWrist = 0.0f; // Suma de la rotacion del ←
    objeto para su uso en limites

//EndEffector
public float velocidadRotacionEndEffector = 50.0f; // Velocidad de ←
    rotacion
public KeyCode teclaRotacionPositivaEndEffector = KeyCode.T; // Tecla ←
    para la rotacion en direccion positiva
public KeyCode teclaRotacionNegativaEndEffector = KeyCode.G; // Tecla ←
    para la rotacion en direccion negativa
public Transform EndEffector; // Transform del objeto a rotar
private float LimitePositivoEndEffector = 570.0f; // Angulo limite del ←
    objeto
private float LimiteNegativoEndEffector = -570.0f; // Angulo limite del ←
    objeto
private float sumaRotacionEndEffector = 0.0f; // Suma de la rotacion del←
    objeto para su uso en limites

//Botonera
private string nombreObjeto = "";
private bool positivoboton = false;
private bool negativoboton = false;

private bool isControlPressed = false;
private bool isAltPressed = false;

private bool EmpezarGuardar = false;
private bool TerminarGuardar = false;
```

```
private bool EmpezarMover = false;
private bool TerminarMover = false;
private int NumeroUsar = -1;

private Dictionary<int, float> GuardarBase = new Dictionary<int, float>();
private Dictionary<int, float> GuardarShoulder = new Dictionary<int, float>();
private Dictionary<int, float> GuardarElbow = new Dictionary<int, float>();
private Dictionary<int, float> GuardarWrist = new Dictionary<int, float>();
private Dictionary<int, float> GuardarEndEffector = new Dictionary<int, float>();

private void Start()
{
    GuardarBase.Add(0, 0f);
    GuardarShoulder.Add(0, 0f);
    GuardarElbow.Add(0, 0f);
    GuardarWrist.Add(0, 0f);
    GuardarEndEffector.Add(0, 0f);
}

private void Update()
{
    Movimiento(Base, teclaRotacionPositivaBase, ←
               teclaRotacionNegativaBase, velocidadRotacionBase, ←
               LimitePositivoBase, LimiteNegativoBase, Base, -Base.up, ref ←
               sumaRotacionBase);
    Movimiento(Shoulder, teclaRotacionPositivaShoulder, ←
               teclaRotacionNegativaShoulder, velocidadRotacionShoulder, ←
               LimitePositivoShoulder, LimiteNegativoShoulder, ←
               puntoFijoShoulder, puntoFijoShoulder.forward, ref ←
               sumaRotacionShoulder);
    Movimiento(Elbow, teclaRotacionPositivaElbow, ←
               teclaRotacionNegativaElbow, velocidadRotacionElbow, ←
               LimitePositivoElbow, LimiteNegativoElbow, puntoFijoElbow, ←
               puntoFijoElbow.forward, ref sumaRotacionElbow);
    Movimiento(Wrist, teclaRotacionPositivaWrist, ←
               teclaRotacionNegativaWrist, velocidadRotacionWrist, ←
               LimitePositivoWrist, LimiteNegativoWrist, Wrist, Wrist.up, ref ←
               sumaRotacionWrist);
    Movimiento(EndEffector, teclaRotacionPositivaEndEffector, ←
               teclaRotacionNegativaEndEffector, velocidadRotacionEndEffector, ←
               LimitePositivoEndEffector, LimiteNegativoEndEffector, ←
               EndEffector, EndEffector.up, ref sumaRotacionEndEffector);
    MovimientoBoton(nombreObjeto);
    Guardar();
    Usar();
    GuardarBoton();
    UsarBoton();
}
```

```
}

private void Movimiento(Transform objeto, KeyCode TeclaPositiva, KeyCode←
    TeclaNegativa, float VelocidadRotacion, float anguloLimitePositivo,←
    float anguloLimiteNegativo, Transform puntoFijo, Vector3 direccion,←
    ref float Suma)
{
    // Obtener la direccion de rotacion basada en las teclas presionadas
    float direccionRotacion = 0.0f;

    if (Input.GetKey(TeclaPositiva))
    {
        direccionRotacion = 1.0f;
    }
    else if (Input.GetKey(TeclaNegativa))
    {
        direccionRotacion = -1.0f;
    }

    // Verificar si no se estan presionando teclas de rotacion
    if (direccionRotacion == 0.0f)
    {
        return;
    }
    // Calcular el angulo de rotacion
    float anguloRotacion = direccionRotacion * VelocidadRotacion * Time.←
        deltaTime;

    // Calcular el angulo de rotacion limitado
    float anguloRotacionLimitado = Mathf.Clamp(Suma, ←
        anguloLimiteNegativo, anguloLimitePositivo);
    if (anguloRotacionLimitado == anguloLimiteNegativo && ←
        direccionRotacion < 0)
    {
        anguloRotacion = 0.0f;
    }

    if (anguloRotacionLimitado == anguloLimitePositivo && ←
        direccionRotacion > 0)
    {
        anguloRotacion = 0.0f;
    }
    // Realizar la rotacion alrededor del punto fijo
    objeto.RotateAround(puntoFijo.position, direccion, anguloRotacion);
    Suma += anguloRotacion;
    Suma = Mathf.Clamp(Suma, anguloLimiteNegativo, anguloLimitePositivo)←
        ;
}

private void MovimientoBoton(string nombre)
```

```
{  
  
    // Obtener la dirección de rotación basada en las teclas presionadas  
    float direcciónRotacionbotón = 0.0f;  
  
    if (positivoboton)  
    {  
        direcciónRotacionbotón = 1.0f;  
    }  
    if (negativoboton)  
    {  
        direcciónRotacionbotón = -1.0f;  
    }  
  
    // Verificar si no se están presionando teclas de rotación  
    if (direcciónRotacionbotón == 0.0f)  
    {  
        return;  
    }  
    float Sumar = 0;  
    Vector3 dirección = Vector3.up;  
    float anguloLímitePositivo = 0;  
    float anguloLímiteNegativo = 0;  
    float VelocidadRotación = 0;  
    Transform puntoFijo = Base;  
    Transform objeto = Base;  
  
    if (nombre == "Base")  
    {  
        Sumar = sumaRotaciónBase;  
        dirección = Base.up;  
        anguloLímitePositivo = LímitePositivoBase;  
        anguloLímiteNegativo = LímiteNegativoBase;  
        VelocidadRotación = velocidadRotaciónBase;  
        puntoFijo = Base;  
        objeto = Base;  
    }  
  
    if (nombre == "Shoulder")  
    {  
        Sumar = sumaRotaciónShoulder;  
        dirección = puntoFijoShoulder.forward;  
        anguloLímitePositivo = LímitePositivoShoulder;  
        anguloLímiteNegativo = LímiteNegativoShoulder;  
        VelocidadRotación = velocidadRotaciónShoulder;  
        puntoFijo = puntoFijoShoulder;  
        objeto = Shoulder;  
    }  
  
    if (nombre == "Elbow")  
    {  
        Sumar = sumaRotaciónElbow;
```

```
    direccion = puntoFijoElbow.forward;
    anguloLimitePositivo = LimitePositivoElbow;
    anguloLimiteNegativo = LimiteNegativoElbow;
    VelocidadRotacion = velocidadRotacionElbow;
    puntoFijo = puntoFijoElbow;
    objeto = Elbow;
}

if (nombre == "Wrist")
{
    Sumar = sumaRotacionWrist;
    direccion = Wrist.up;
    anguloLimitePositivo = LimitePositivoWrist;
    anguloLimiteNegativo = LimiteNegativoWrist;
    VelocidadRotacion = velocidadRotacionWrist;
    puntoFijo = Wrist;
    objeto = Wrist;
}

if (nombre == "EndEffector")
{
    Sumar = sumaRotacionEndEffector;
    direccion = EndEffector.up;
    anguloLimitePositivo = LimitePositivoEndEffector;
    anguloLimiteNegativo = LimiteNegativoEndEffector;
    VelocidadRotacion = velocidadRotacionEndEffector;
    puntoFijo = EndEffector;
    objeto = EndEffector;
}

// Calcular el angulo de rotacion
float anguloRotacion = direccionRotacionboton * VelocidadRotacion * ←
    Time.deltaTime;

// Calcular el angulo de rotacion limitado
float anguloRotacionLimitado = Mathf.Clamp(Sumar, ←
    anguloLimiteNegativo, anguloLimitePositivo);
if (anguloRotacionLimitado == anguloLimiteNegativo && ←
    direccionRotacionboton < 0)
{
    anguloRotacion = 0.0f;
}

if (anguloRotacionLimitado == anguloLimitePositivo && ←
    direccionRotacionboton > 0)
{
    anguloRotacion = 0.0f;
}
// Realizar la rotacion alrededor del punto fijo
objeto.RotateAround(puntoFijo.position, direccion, anguloRotacion);
Sumar += anguloRotacion;
```

```
Sumar = Mathf.Clamp(Sumar, anguloLimiteNegativo, ←
    anguloLimitePositivo);

if (nombre == "Base")
{
    sumaRotacionBase = Sumar;
}

if (nombre == "Shoulder")
{
    sumaRotacionShoulder = Sumar;
}

if (nombre == "Elbow")
{
    sumaRotacionElbow = Sumar;
}

if (nombre == "Wrist")
{
    sumaRotacionWrist = Sumar;
}
if (nombre == "EndEffector")
{
    sumaRotacionEndEffector = Sumar;
}

}

public void PointerUpPositivo(string nombre)
{
    nombreObjeto = nombre;
    positivoboton = false;
}

public void PointerDownPositivo(string nombre)
{
    nombreObjeto = nombre;
    positivoboton = true;
}

public void PointerUpNegativo(string nombre)
{
    nombreObjeto = nombre;
    negativoboton = false;
}

public void PointerDownNegativo(string nombre)
{
    nombreObjeto = nombre;
    negativoboton = true;
}
```

```
}

private void Guardar()
{
    // Verificar si se presiono la tecla de control (Ctrl)
    if (Input.GetKeyDown(KeyCode.LeftControl) || Input.GetKeyDown(KeyCode.RightControl))
    {
        isControlPressed = true;
    }

    // Verificar si esta habilitado el registro de teclas
    if (isControlPressed)
    {
        // Verificar las teclas numericas del 1 al 9
        for (int i = 1; i <= 9; i++)
        {
            if (Input.GetKey(i.ToString()))
            {
                SaveInformation(i);
                isControlPressed = false;
            }
        }
    }
}

private void GuardarBoton()
{
    // Verificar si se presiono el boton
    if (EmpezarGuardar)
    {
        // Verificar si se habilita el guardado
        if (TerminarGuardar)
        {
            SaveInformation(NumeroUsar);
            TerminarGuardar = false;
            EmpezarGuardar = false;
            NumeroUsar = -1;
        }
    }
}

private void Usar()
{
    // Verificar si se presiono la tecla de control (Ctrl)
    if (Input.GetKeyDown(KeyCode.LeftAlt) || Input.GetKeyDown(KeyCode.RightAlt))
    {
        isAltPressed = true;
    }

    // Verificar si esta habilitado el registro de teclas
    if (isAltPressed)
```

```
{  
    // Verificar las teclas numericas del 0 al 9  
    for (int i = 0; i <= 9; i++)  
    {  
        if (Input.GetKey(i.ToString()))  
        {  
            GetInformation(i);  
            isAltPressed = false;  
        }  
    }  
}  
  
private void UsarBoton()  
{  
    // Verificar si se presiono el boton  
    if (EmpezarMover)  
    {  
        // Verificar si se habilita el guardado  
        if (TerminarMover)  
        {  
            GetInformation(NumeroUsar);  
            TerminarMover = false;  
            EmpezarMover = false;  
            NumeroUsar = -1;  
        }  
    }  
}  
  
private void SaveInformation(int i)  
{  
    if (GuardarBase.ContainsKey(i))  
    {  
        GuardarBase[i] = sumaRotacionBase;  
    }  
    else  
    {  
        GuardarBase.Add(i, sumaRotacionBase);  
    }  
  
    if (GuardarShoulder.ContainsKey(i))  
    {  
        GuardarShoulder[i] = sumaRotacionShoulder;  
    }  
    else  
    {  
        GuardarShoulder.Add(i, sumaRotacionShoulder);  
    }  
  
    if (GuardarElbow.ContainsKey(i))  
    {  
        GuardarElbow[i] = sumaRotacionElbow;  
    }  
}
```

```
        }
    else
    {
        GuardarElbow.Add(i, sumaRotacionElbow);
    }

    if (GuardarWrist.ContainsKey(i))
    {
        GuardarWrist[i] = sumaRotacionWrist;
    }
    else
    {
        GuardarWrist.Add(i, sumaRotacionWrist);
    }

    if (GuardarEndEffector.ContainsKey(i))
    {
        GuardarEndEffector[i] = sumaRotacionEndEffector;
    }
    else
    {
        GuardarEndEffector.Add(i, sumaRotacionEndEffector);
    }
}

private void GetInformation(int i)
{
    if (GuardarBase.ContainsKey(i))
    {
        float value = GuardarBase[i];
        Debug.Log("Base " + i + " = " + value);
        //MoverGuardado(value, sumaRotacionBase, Base, Base, ←
        LimiteNegativoBase, LimitePositivoBase, velocidadRotacionBase ←
        , -Base.up);
    }
    else
    {
        Debug.Log("no existe");
    }

    if (GuardarShoulder.ContainsKey(i))
    {
        float value = GuardarShoulder[i];
        Debug.Log("Shoulder " + i + " = " + value);
    }
    else
    {
        Debug.Log("no existe");
    }

    if (GuardarElbow.ContainsKey(i))
    {
```

```
        float value = GuardarElbow[i];
        Debug.Log("Elbow " + i + " = " + value);
    }
    else
    {
        Debug.Log("no existe");
    }

    if (GuardarWrist.ContainsKey(i))
    {
        float value = GuardarWrist[i];
        Debug.Log("Wrist " + i + " = " + value);
    }
    else
    {
        Debug.Log("no existe");
    }

    if (GuardarEndEffector.ContainsKey(i))
    {
        float value = GuardarEndEffector[i];
        Debug.Log("EndEffector " + i + " = " + value);
    }
    else
    {
        Debug.Log("no existe");
    }
}

public void EmpezarAGuardar()
{
    EmpezarGuardar = true;
}

public void NumeroAUsar(int i)
{
    if(EmpezarGuardar || EmpezarMover)
    {
        NumeroUsar = i;
    }
}

public void TerminarAGuardar()
{
    if(0 < NumeroUsar && NumeroUsar < 10)
    {
        if(EmpezarGuardar)
        {
            TerminarGuardar = true;
        }
    }
}
```

```
public void EmpezarAMover()
{
    EmpezarMover = true;
}

public void TerminarAMover()
{
    if(-1 < NumeroUsar && NumeroUsar < 10)
    {
        if(EmpezarMover)
        {
            TerminarMover = true;
        }
    }
}

private void MoverGuardado(float valor, float Suma, Transform puntoFijo,←
    Transform objeto, float anguloLimiteNegativo, float ←
    anguloLimitePositivo, float VelocidadRotacion, Vector3 direccion)
{
    float direccionRotacion = 0.0f;

    if (Suma < valor)
    {
        direccionRotacion = 1.0f;
    }
    else if (Suma > valor)
    {
        direccionRotacion = -1.0f;
    }

    if (direccionRotacion == 0.0f)
    {
        return;
    }
    // Calcular el angulo de rotacion

    // Realizar la rotacion alrededor del punto fijo
    while(Suma != valor)
    {
        float anguloRotacion = direccionRotacion * VelocidadRotacion * ←
            Time.deltaTime;
        objeto.RotateAround(puntoFijo.position, direccion, ←
            anguloRotacion);
        Suma += anguloRotacion;
        if(direccionRotacion == 1.0f)
        {
            Suma = Mathf.Clamp(Suma, anguloLimiteNegativo, valor);
        }
    }
}
```

```
        else
    {
        Suma = Mathf.Clamp(Suma, valor, anguloLimitePositivo);
    }

}
```

Apéndice D

Código: MovimientoJoints Vz.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class MovimientoJointsVz : MonoBehaviour
{
    //Base
    public float velocidadRotacionBase = 50.0f; // Velocidad de rotacion
    public KeyCode teclaRotacionPositivaBase = KeyCode.Q; // Tecla para la ←
        rotacion en direccion positiva
    public KeyCode teclaRotacionNegativaBase = KeyCode.A; // Tecla para la ←
        rotacion en direccion negativa
    public Transform Base; // Transform del objeto que se va a rotar
    private float LimitePositivoBase = 155.0f; // Angulo limite del objeto
    private float LimiteNegativoBase = -155.0f; // Angulo limite del objeto
    private float sumaRotacionBase = 0.0f; // Suma de la rotacion del objeto←
        para su uso en limites

    //Shoulder
    public float velocidadRotacionShoulder = 50.0f; // Velocidad de rotacion
    public KeyCode teclaRotacionPositivaShoulder = KeyCode.W; // Tecla para ←
        la rotacion en direccion positiva
    public KeyCode teclaRotacionNegativaShoulder = KeyCode.S; // Tecla para ←
        la rotacion en direccion negativa
    public Transform Shoulder; // Transform del objeto a rotar
    public Transform puntoFijoShoulder; // Transform del punto fijo ←
        alrededor del cual se realizara la rotacion
    private float LimitePositivoShoulder = 10.0f; // Angulo limite del ←
        objeto
    private float LimiteNegativoShoulder = -130.0f; // Angulo limite del ←
        objeto
    private float sumaRotacionShoulder = 0.0f; // Suma de la rotacion del ←
        objeto para su uso en limites
```

```
//Elbow
public float velocidadRotacionElbow = 50.0f; // Velocidad de rotacion
public KeyCode teclaRotacionPositivaElbow = KeyCode.E; // Tecla para la ←
    rotacion en direccion positiva
public KeyCode teclaRotacionNegativaElbow = KeyCode.D; // Tecla para la ←
    rotacion en direccion negativa
public Transform Elbow; // Transform del objeto a rotar
public Transform puntoFijoElbow; // Transform del punto fijo alrededor ←
    del cual se realizara la rotacion
private float LimitePositivoElbow = 130.0f; // Angulo limite del objeto
private float LimiteNegativoElbow = -130.0f; // Angulo limite del objeto
private float sumaRotacionElbow = 0.0f; // Suma de la rotacion del ←
    objeto para su uso en limites

//Wrist
public float velocidadRotacionWrist = 50.0f; // Velocidad de rotacion
public KeyCode teclaRotacionPositivaWrist = KeyCode.R; // Tecla para la ←
    rotacion en direccion positiva
public KeyCode teclaRotacionNegativaWrist = KeyCode.F; // Tecla para la ←
    rotacion en direccion negativa
public Transform Wrist; // Transform del objeto a rotar
private float LimitePositivoWrist= 130.0f; // Angulo limite del objeto
private float LimiteNegativoWrist = -130.0f; // Angulo limite del objeto
private float sumaRotacionWrist = 0.0f; // Suma de la rotacion del ←
    objeto para su uso en limites

//EndEffector
public float velocidadRotacionEndEffector = 50.0f; // Velocidad de ←
    rotacion
public KeyCode teclaRotacionPositivaEndEffector = KeyCode.T; // Tecla ←
    para la rotacion en direccion positiva
public KeyCode teclaRotacionNegativaEndEffector = KeyCode.G; // Tecla ←
    para la rotacion en direccion negativa
public Transform EndEffector; // Transform del objeto a rotar
private float LimitePositivoEndEffector = 570.0f; // Angulo limite del ←
    objeto
private float LimiteNegativoEndEffector = -570.0f; // Angulo limite del ←
    objeto
private float sumaRotacionEndEffector = 0.0f; // Suma de la rotacion del←
    objeto para su uso en limites

//Botonera
private string nombreObjeto = "";
private bool positivoboton = false;
private bool negativoboton = false;
private bool positivobotonZ = false;
private bool negativobotonZ = false;

private bool isControlPressed = false;
private bool isAltPressed = false;
```

```
private bool EmpezarGuardar = false;
private bool TerminarGuardar = false;
private bool EmpezarMover = false;
private bool TerminarMover = false;
private int NumeroUsar = -1;

private Dictionary<int, float> GuardarBase = new Dictionary<int, float>();
private Dictionary<int, float> GuardarShoulder = new Dictionary<int, float>();
private Dictionary<int, float> GuardarElbow = new Dictionary<int, float>();
private Dictionary<int, float> GuardarWrist = new Dictionary<int, float>();
private Dictionary<int, float> GuardarEndEffector = new Dictionary<int, float>();

private void Start()
{
    GuardarBase.Add(0, 0f);
    GuardarShoulder.Add(0, 0f);
    GuardarElbow.Add(0, 0f);
    GuardarWrist.Add(0, 0f);
    GuardarEndEffector.Add(0, 0f);
}

private void Update()
{
    Movimiento(Base, teclaRotacionPositivaBase, ←
               teclaRotacionNegativaBase, velocidadRotacionBase, ←
               LimitePositivoBase, LimiteNegativoBase, Base, -Base.up, ref ←
               sumaRotacionBase);
    Movimiento(Shoulder, teclaRotacionPositivaShoulder, ←
               teclaRotacionNegativaShoulder, velocidadRotacionShoulder, ←
               LimitePositivoShoulder, LimiteNegativoShoulder, ←
               puntoFijoShoulder, puntoFijoShoulder.forward, ref ←
               sumaRotacionShoulder);
    Movimiento(Elbow, teclaRotacionPositivaElbow, ←
               teclaRotacionNegativaElbow, velocidadRotacionElbow, ←
               LimitePositivoElbow, LimiteNegativoElbow, puntoFijoElbow, ←
               puntoFijoElbow.forward, ref sumaRotacionElbow);
    Movimiento(Wrist, teclaRotacionPositivaWrist, ←
               teclaRotacionNegativaWrist, velocidadRotacionWrist, ←
               LimitePositivoWrist, LimiteNegativoWrist, Wrist, Wrist.up, ref ←
               sumaRotacionWrist);
    Movimiento(EndEffector, teclaRotacionPositivaEndEffector, ←
               teclaRotacionNegativaEndEffector, velocidadRotacionEndEffector, ←
               LimitePositivoEndEffector, LimiteNegativoEndEffector, ←
               EndEffector, EndEffector.up, ref sumaRotacionEndEffector);
    MovimientoBoton(nombreObjeto);
    Guardar();
    Usar();
}
```

```
    GuardarBoton();
    UsarBoton();
    MoverEje();
    MoverEjeBoton();

}

private void Movimiento(Transform objeto, KeyCode TeclaPositiva, KeyCode TeclaNegativa, float VelocidadRotacion, float anguloLimitePositivo, float anguloLimiteNegativo, Transform puntoFijo, Vector3 direccion, ref float Suma)
{
    // Obtener la direccion de rotacion basada en las teclas presionadas
    float direccionRotacion = 0.0f;

    if (Input.GetKey(TeclaPositiva))
    {
        direccionRotacion = 1.0f;
    }
    else if (Input.GetKey(TeclaNegativa))
    {
        direccionRotacion = -1.0f;
    }

    // Verificar si no se estan presionando teclas de rotacion
    if (direccionRotacion == 0.0f)
    {
        return;
    }
    // Calcular el angulo de rotacion
    float anguloRotacion = direccionRotacion * VelocidadRotacion * Time.deltaTime;

    // Calcular el angulo de rotacion limitado
    float anguloRotacionLimitado = Mathf.Clamp(Suma, anguloLimiteNegativo, anguloLimitePositivo);
    if (anguloRotacionLimitado == anguloLimiteNegativo && direccionRotacion < 0)
    {
        anguloRotacion = 0.0f;
    }

    if (anguloRotacionLimitado == anguloLimitePositivo && direccionRotacion > 0)
    {
        anguloRotacion = 0.0f;
    }
    // Realizar la rotacion alrededor del punto fijo
    objeto.RotateAround(puntoFijo.position, direccion, anguloRotacion);
    Suma += anguloRotacion;
```

```
Suma = Mathf.Clamp(Suma, anguloLimiteNegativo, anguloLimitePositivo)←
;
}

private void MovimientoBoton(string nombre)
{
    // Obtener la dirección de rotación basada en las teclas presionadas
    float direccionRotacionboton = 0.0f;

    if (positivoboton)
    {
        direccionRotacionboton = 1.0f;
    }
    if (negativoboton)
    {
        direccionRotacionboton = -1.0f;
    }

    // Verificar si no se están presionando teclas de rotación
    if (direccionRotacionboton == 0.0f)
    {
        return;
    }
    float Sumar = 0;
    Vector3 direccion = Vector3.up;
    float anguloLimitePositivo = 0;
    float anguloLimiteNegativo = 0;
    float VelocidadRotacion = 0;
    Transform puntoFijo = Base;
    Transform objeto = Base;

    if (nombre == "Base")
    {
        Sumar = sumaRotacionBase;
        direccion = Base.up;
        anguloLimitePositivo = LimitePositivoBase;
        anguloLimiteNegativo = LimiteNegativoBase;
        VelocidadRotacion = velocidadRotacionBase;
        puntoFijo = Base;
        objeto = Base;
    }

    if (nombre == "Shoulder")
    {
        Sumar = sumaRotacionShoulder;
        direccion = puntoFijoShoulder.forward;
        anguloLimitePositivo = LimitePositivoShoulder;
        anguloLimiteNegativo = LimiteNegativoShoulder;
        VelocidadRotacion = velocidadRotacionShoulder;
        puntoFijo = puntoFijoShoulder;
        objeto = Shoulder;
```

```
}

if (nombre == "Elbow")
{
    Sumar = sumaRotacionElbow;
    direccion = puntoFijoElbow.forward;
    anguloLimitePositivo = LimitePositivoElbow;
    anguloLimiteNegativo = LimiteNegativoElbow;
    VelocidadRotacion = velocidadRotacionElbow;
    puntoFijo = puntoFijoElbow;
    objeto = Elbow;
}

if (nombre == "Wrist")
{
    Sumar = sumaRotacionWrist;
    direccion = Wrist.up;
    anguloLimitePositivo = LimitePositivoWrist;
    anguloLimiteNegativo = LimiteNegativoWrist;
    VelocidadRotacion = velocidadRotacionWrist;
    puntoFijo = Wrist;
    objeto = Wrist;
}

if (nombre == "EndEffector")
{
    Sumar = sumaRotacionEndEffector;
    direccion = EndEffector.up;
    anguloLimitePositivo = LimitePositivoEndEffector;
    anguloLimiteNegativo = LimiteNegativoEndEffector;
    VelocidadRotacion = velocidadRotacionEndEffector;
    puntoFijo = EndEffector;
    objeto = EndEffector;
}

// Calcular el angulo de rotacion
float anguloRotacion = direccionRotacionboton * VelocidadRotacion * ←
    Time.deltaTime;

// Calcular el angulo de rotacion limitado
float anguloRotacionLimitado = Mathf.Clamp(Sumar, ←
    anguloLimiteNegativo, anguloLimitePositivo);
if (anguloRotacionLimitado == anguloLimiteNegativo && ←
    direccionRotacionboton < 0)
{
    anguloRotacion = 0.0f;
}

if (anguloRotacionLimitado == anguloLimitePositivo && ←
    direccionRotacionboton > 0)
{
    anguloRotacion = 0.0f;
```

```
        }

        // Realizar la rotacion alrededor del punto fijo
        objeto.RotateAround(puntoFijo.position, direccion, anguloRotacion);
        Sumar += anguloRotacion;
        Sumar = Mathf.Clamp(Sumar, anguloLimiteNegativo, ←
                            anguloLimitePositivo);

        if (nombre == "Base")
        {
            sumaRotacionBase = Sumar;
        }

        if (nombre == "Shoulder")
        {
            sumaRotacionShoulder = Sumar;
        }

        if (nombre == "Elbow")
        {
            sumaRotacionElbow = Sumar;
        }

        if (nombre == "Wrist")
        {
            sumaRotacionWrist = Sumar;
        }
        if (nombre == "EndEffector")
        {
            sumaRotacionEndEffector = Sumar;
        }

    }

    public void PointerUpPositivo(string nombre)
    {
        nombreObjeto = nombre;
        positivoboton = false;
    }

    public void PointerDownPositivo(string nombre)
    {
        nombreObjeto = nombre;
        positivoboton = true;
    }

    public void PointerUpNegativo(string nombre)
    {
        nombreObjeto = nombre;
        negativoboton = false;
    }
```

```
public void PointerDownNegativo(string nombre)
{
    nombreObjeto = nombre;
    negativoboton = true;
}

public void PointerUpPositivoZ()
{
    if (!EmpezarGuardar && !EmpezarMover)
    {
        positivobotonZ = false;
    }
}

public void PointerDownPositivoZ()
{
    if (!EmpezarGuardar && !EmpezarMover)
    {
        positivobotonZ = true;
    }
}

public void PointerUpNegativoZ()
{
    if (!EmpezarGuardar && !EmpezarMover)
    {
        negativobotonZ = false;
    }
}

public void PointerDownNegativoZ()
{
    if (!EmpezarGuardar && !EmpezarMover)
    {
        negativobotonZ = true;
    }
}

private void Guardar()
{
    // Verificar si se presiono la tecla de control (Ctrl)
    if (Input.GetKeyDown(KeyCode.LeftControl) || Input.GetKeyDown(KeyCode.RightControl))
    {
        isControlPressed = true;
    }

    // Verificar si esta habilitado el registro de teclas
    if (isControlPressed)
    {
        // Verificar las teclas numericas del 1 al 9
        for (int i = 1; i <= 9; i++)
```

```
{  
    if (Input.GetKeyDown(i.ToString()))  
    {  
        SaveInformation(i);  
        isControlPressed = false;  
    }  
}  
}  
  
private void GuardarBoton()  
{  
    // Verificar si se presiono el boton  
    if (EmpezarGuardar)  
    {  
        // Verificar si se habilita el guardado  
        if (TerminarGuardar)  
        {  
            SaveInformation(NumeroUsar);  
            TerminarGuardar = false;  
            EmpezarGuardar = false;  
            NumeroUsar = -1;  
        }  
    }  
}  
  
private void Usar()  
{  
    // Verificar si se presiono la tecla de control (Ctrl)  
    if (Input.GetKey(KeyCode.LeftAlt) || Input.GetKey(KeyCode.RightAlt))  
    {  
        isAltPressed = true;  
    }  
  
    // Verificar si esta habilitado el registro de teclas  
    if (isAltPressed)  
    {  
        // Verificar las teclas numericas del 0 al 9  
        for (int i = 0; i <= 9; i++)  
        {  
            if (Input.GetKeyDown(i.ToString()))  
            {  
                GetInformation(i);  
                isAltPressed = false;  
            }  
        }  
    }  
}  
  
private void UsarBoton()  
{  
    // Verificar si se presiono el boton
```

```
    if (EmpezarMover)
    {
        // Verificar si se habilita el guardado
        if (TerminarMover)
        {
            GetInformation(Numerousar);
            TerminarMover = false;
            EmpezarMover = false;
            Numerousar = -1;
        }
    }

    private void SaveInformation(int i)
    {
        if (GuardarBase.ContainsKey(i))
        {
            GuardarBase[i] = sumaRotacionBase;
        }
        else
        {
            GuardarBase.Add(i, sumaRotacionBase);
        }

        if (GuardarShoulder.ContainsKey(i))
        {
            GuardarShoulder[i] = sumaRotacionShoulder;
        }
        else
        {
            GuardarShoulder.Add(i, sumaRotacionShoulder);
        }

        if (GuardarElbow.ContainsKey(i))
        {
            GuardarElbow[i] = sumaRotacionElbow;
        }
        else
        {
            GuardarElbow.Add(i, sumaRotacionElbow);
        }

        if (GuardarWrist.ContainsKey(i))
        {
            GuardarWrist[i] = sumaRotacionWrist;
        }
        else
        {
            GuardarWrist.Add(i, sumaRotacionWrist);
        }

        if (GuardarEndEffector.ContainsKey(i))
```

```
{  
    GuardarEndEffector[i] = sumaRotacionEndEffector;  
}  
else  
{  
    GuardarEndEffector.Add(i, sumaRotacionEndEffector);  
}  
}  
  
private void GetInformation(int i)  
{  
    if (GuardarBase.ContainsKey(i))  
    {  
        float value = GuardarBase[i];  
        Debug.Log("Base " + i + " = " + value);  
        //MoverGuardado(value ,sumaRotacionBase ,Base ,Base ,←  
        LimiteNegativoBase ,LimitePositivoBase ,velocidadRotacionBase ←  
        ,-Base .up );  
    }  
    else  
{  
        Debug.Log("no existe");  
    }  
  
    if (GuardarShoulder.ContainsKey(i))  
    {  
        float value = GuardarShoulder[i];  
        Debug.Log("Shoulder " + i + " = " + value);  
    }  
    else  
{  
        Debug.Log("no existe");  
    }  
  
    if (GuardarElbow.ContainsKey(i))  
    {  
        float value = GuardarElbow[i];  
        Debug.Log("Elbow " + i + " = " + value);  
    }  
    else  
{  
        Debug.Log("no existe");  
    }  
  
    if (GuardarWrist.ContainsKey(i))  
    {  
        float value = GuardarWrist[i];  
        Debug.Log("Wrist " + i + " = " + value);  
    }  
    else  
{  
        Debug.Log("no existe");  
    }  
}
```

```
    }

    if (GuardarEndEffector.ContainsKey(i))
    {
        float value = GuardarEndEffector[i];
        Debug.Log("EndEffector " + i + " = " + value);
    }
    else
    {
        Debug.Log("no existe");
    }
}

public void EmpezarAGuardar()
{
    EmpezarGuardar = true;
}

public void NumeroAUsar(int i)
{
    if (EmpezarGuardar || EmpezarMover)
    {
        NumeroUsar = i;
    }
}

public void TerminarAGuardar()
{
    if (0 < NumeroUsar && NumeroUsar < 10)
    {
        if (EmpezarGuardar)
        {
            TerminarGuardar = true;
        }
    }
}

public void EmpezarAMover()
{
    EmpezarMover = true;
}

public void TerminarAMover()
{
    if (-1 < NumeroUsar && NumeroUsar < 10)
    {
        if (EmpezarMover)
        {
            TerminarMover = true;
        }
    }
}
```

```
}

private void MoverEje()
{
    float movimiento = 0f;

    // Verifica si se presiona la tecla positiva
    if (Input.GetKey(KeyCode.Alpha1))
    {
        movimiento = 1f;
    }

    // Verifica si se presiona la tecla negativa
    if (Input.GetKey(KeyCode.Alpha6))
    {
        movimiento = -1f;
    }

    // Mueve el objeto en el eje Y (o el eje que deseas) segun el valor ←
    // de movimiento
    transform.Translate(Vector3.up * movimiento * 5f * Time.deltaTime);
}

private void MoverEjeBoton()
{
    float movimiento = 0f;

    if (positivobotonZ)
    {
        movimiento = 1.0f;
    }
    if (negativobotonZ)
    {
        movimiento = -1.0f;
    }

    // Mueve el objeto en el eje Y (o el eje que deseas) segun el valor ←
    // de movimiento
    transform.Translate(Vector3.up * movimiento * 5f * Time.deltaTime);
}

private void MoverGuardado(float valor, float Suma, Transform puntoFijo, ←
    Transform objeto, float anguloLimiteNegativo, float ←
    anguloLimitePositivo, float VelocidadRotacion, Vector3 direccion)
{
    float direccionRotacion = 0.0f;

    if (Suma < valor)
    {
        direccionRotacion = 1.0f;
    }
}
```

```
else if (Suma > valor)
{
    direccionRotacion = -1.0f;
}

if (direccionRotacion == 0.0f)
{
    return;
}
// Calcular el angulo de rotacion

// Realizar la rotacion alrededor del punto fijo
while(Suma != valor)
{
    float anguloRotacion = direccionRotacion * VelocidadRotacion * ←
        Time.deltaTime;
    objeto.RotateAround(puntoFijo.position, direccion, ←
        anguloRotacion);
    Suma += anguloRotacion;
    if(direccionRotacion == 1.0f)
    {
        Suma = Mathf.Clamp(Suma, anguloLimiteNegativo, valor);
    }
    else
    {
        Suma = Mathf.Clamp(Suma, valor, anguloLimitePositivo);
    }
}

}
```

Apéndice E

Código: TomarObjeto.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TomarObjeto : MonoBehaviour
{
    public GameObject handPoint;
    private GameObject pickedObject = null;
    private bool podertomar = false;
    private bool boton = false;
    private bool tomado = false;

    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if(pickedObject != null)
        {
            if(Input.GetKey("x") || !boton)
            {
                pickedObject.GetComponent<Rigidbody>().useGravity = true;
                pickedObject.GetComponent<Rigidbody>().isKinematic = false;
                pickedObject.gameObject.transform.SetParent(null);
                pickedObject = null;
                tomado = false;
            }
        }
    }
}
```

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Objeto"))
    {
        if (pickedObject == null)
        {
            podertomar = true;
            if (Input.GetKey("z") || boton)
            {
                other.GetComponent<Rigidbody>().useGravity = false;
                other.GetComponent<Rigidbody>().isKinematic = true;
                other.transform.position = handPoint.transform.position;
                other.gameObject.transform.SetParent(handPoint.←
                    gameObject.transform);
                pickedObject = other.gameObject;
                tomado = true;
                podertomar = false;
                boton = true;
            }
        }
    }
}

public void OnClick()
{
    if (podertomar)
    {
        boton = true;
    }
    if (tomado)
    {
        boton = false;
    }
}
```

Apéndice F

Código: CalidadImagen.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class CalidadImagen : MonoBehaviour
{
    public Dropdown dropdown;
    public int calidad;

    void Start()
    {
        calidad = PlayerPrefs.GetInt("numeroDeCalidad", 3);
        dropdown.value = calidad;
        AjustarCalidad();
    }

    // Update is called once per frame
    void Update()
    {

    }

    public void AjustarCalidad()
    {
        QualitySettings.SetQualityLevel(dropdown.value);
        PlayerPrefs.SetInt("numeroDeCalidad", dropdown.value);
        calidad = dropdown.value;
    }
}
```

Apéndice G

Código: Camaras.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Camaras : MonoBehaviour
{
    public Dropdown camarasDropdown;
    public List <Camera> camaras1 = new List <Camera>();
    void Start()
    {
        LlenarCamaras();
    }

    void Update()
    {

    }

    public void LlenarCamaras()
    {
        camarasDropdown.ClearOptions();

        camaras1.AddRange(FindObjectsOfType<Camera>(true));
        List<string> opciones = new List<string>();
        int j = 0;
        int activo = 0;
        foreach (var i in camaras1)
        {
            opciones.Add(i.name);
            if(i.isActiveAndEnabled)
            {
                activo = j;
            }
        }
    }
}
```

```
        j++;
    }
    camarasDropdown.AddOptions(opciones);
    camarasDropdown.value = activo;
    camarasDropdown.RefreshShownValue();
}

public void CambiarCamara()
{
    foreach (var i in camaras1)
    {
        i.gameObject.SetActive(false);
        if (string.Compare(i.name, camarasDropdown.options[camarasDropdown.value].text) == 0)
        {
            i.gameObject.SetActive(true);
        }
    }
}
```

Apéndice H

Código: ControlDropdown.cs

```
using UnityEngine;
using UnityEngine.UI;

public class ControlDropdown : MonoBehaviour
{
    public MonoBehaviour scriptBrazoUno;
    public MonoBehaviour scriptBrazoDos;
    public MonoBehaviour scriptBrazoTres;
    public MonoBehaviour scriptTomarUno;
    public MonoBehaviour scriptTomarDos;
    public MonoBehaviour scriptTomarTres;
    public GameObject brazoUnoGO;
    public GameObject brazoDosGO;
    public GameObject brazoTresGO;
    public Dropdown dropdown;

    private void Awake()
    {
        // Suscribirnos al evento OnValueChanged del Dropdown
        dropdown.onValueChanged.AddListener(OnDropdownValueChanged);
    }

    private void OnDropdownValueChanged(int index)
    {
        // Obtenemos el valor actual del Dropdown (0 es el primer item, 1 el segundo, etc.)
        if (index == 0)
        {
            // Si se selecciona el primer item, desactivamos ScriptDos y ScriptTres, y activamos ScriptUno
            scriptBrazoDos.enabled = false;
            scriptBrazoTres.enabled = false;
            scriptBrazoUno.enabled = true;
            scriptTomarDos.enabled = false;
        }
    }
}
```

```
        scriptTomarTres.enabled = false;
        scriptTomarUno.enabled = true;
        brazoDosGO.SetActive(false);
        brazoTresGO.SetActive(false);
        brazoUnoGO.SetActive(true);
    }
    else if (index == 1)
    {
        // Si se selecciona el segundo ítem, desactivamos ScriptUno y ←
        // ScriptTres, y activamos ScriptDos
        scriptBrazoUno.enabled = false;
        scriptBrazoTres.enabled = false;
        scriptBrazoDos.enabled = true;
        scriptTomarUno.enabled = false;
        scriptTomarTres.enabled = false;
        scriptTomarDos.enabled = true;
        brazoUnoGO.SetActive(false);
        brazoTresGO.SetActive(false);
        brazoDosGO.SetActive(true);
    }
    else if (index == 2)
    {
        // Si se selecciona el tercer ítem, desactivamos ScriptUno y ←
        // ScriptDos, y activamos ScriptTres
        scriptBrazoUno.enabled = false;
        scriptBrazoDos.enabled = false;
        scriptBrazoTres.enabled = true;
        scriptTomarUno.enabled = false;
        scriptTomarDos.enabled = false;
        scriptTomarTres.enabled = true;
        brazoUnoGO.SetActive(false);
        brazoDosGO.SetActive(false);
        brazoTresGO.SetActive(true);
    }
}
```

Apéndice I

Código: MenuPausa.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MenuPausa : MonoBehaviour
{
    [SerializeField] private GameObject ui;
    [SerializeField] private GameObject menuPausa;

    void Start()
    {
        Time.timeScale = 0f;
    }

    public void Pausa()
    {
        Time.timeScale = 0f;
        ui.SetActive(false);
        menuPausa.SetActive(true);
    }

    public void Reanudar()
    {
        Time.timeScale = 1f;
        ui.SetActive(true);
        menuPausa.SetActive(false);
    }

    public void Reiniciar()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

```
public void Cerrar()
{
    Application.Quit();
}
```

Apéndice J

Código: PantallaCompleta.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class PantallaCompleta : MonoBehaviour
{

    public Toggle toggle;
    public Dropdown resolucionesDropdown;
    Resolution[] resoluciones;

    void Start()
    {
        if (Screen.fullScreen)
        {
            toggle.isOn = true;
        }
        else
        {
            toggle.isOn = false;
        }

        RevisarResolucion();
    }

    // Update is called once per frame
    void Update()
    {

    }

    public void ActivarPantallaCompleta(bool pantallaCompleta)
    {
```

```
        Screen.fullScreen = pantallaCompleta;
    }

    public void RevisarResolucion()
    {
        resoluciones = Screen.resolutions;
        resolucionesDropdown.ClearOptions();
        List<string> opciones = new List<string>();
        int resolucionActual = 0;

        for (int i = 0; i < resoluciones.Length; i++)
        {
            string opcion = resoluciones[i].width + " x " + resoluciones[i].height;
            opciones.Add(opcion);

            if (Screen.fullScreen && resoluciones[i].width == Screen.currentResolution.width && resoluciones[i].height == Screen.currentResolution.height)
            {
                resolucionActual = i;
            }
        }
        resolucionesDropdown.AddOptions(opciones);
        resolucionesDropdown.value = resolucionActual;
        resolucionesDropdown.RefreshShownValue();
        resolucionesDropdown.value = PlayerPrefs.GetInt("numeroResolucion", 0);
    }

    public void CambiarResolucion(int indiceResolucion)
    {
        PlayerPrefs.SetInt("numeroResolucion", resolucionesDropdown.value);
        Resolution resolucion = resoluciones[indiceResolucion];
        Screen.SetResolution(resolucion.width, resolucion.height, Screen.fullScreen);
    }
}
```