



UNIVERSITAT POLITÈCNICA DE CATALUNYA

BARCELONATECH

Facultat d'Informàtica de Barcelona



# DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE BALL BALANCING PARA EL ROBOT UR3 MEDIANTE TÉCNICAS DE OPTIMIZACIÓN BASADAS EN LA INTELIGENCIA ARTIFICIAL

ALEXANDRU-ILIE POPA

**Director/a:** ANTONIO CAMACHO SANTIAGO (Departamento de Ingeniería de Sistemas, Automática e Informática Industrial)

**Titulación:** Grado en Ingeniería Informática (Computación)

Memoria del trabajo de fin de grado

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

16/05/2024

# Resumen

Este Trabajo de Fin de Grado se centra en la integración de tecnologías avanzadas en robótica y control automático, con un especial énfasis en la aplicación de inteligencia artificial para abordar desafíos complejos de ingeniería. Desarrollado en el marco del programa de Grado en Ingeniería Informática de la Facultad de Informática de Barcelona, el proyecto implementa un sistema de equilibrio dinámico para una esfera sobre un cilindro, utilizando el robot UR3e de Universal Robots.

El corazón del sistema es un microcontrolador Arduino, responsable de recoger datos del sensor de distancia y comunicarse en tiempo real con el robot. Crucialmente, este microcontrolador también ejecuta la inferencia de la red neuronal entrenada con el algoritmo de aprendizaje por refuerzo Soft Actor-Critic (SAC), permitiendo aplicar directamente las técnicas de control adaptativo aprendidas en un entorno dinámico y real.

La configuración del SAC ha sido meticulosamente ajustada para adaptarse a las incertidumbres inherentes del entorno operativo. Esto incluye la optimización de parámetros como el factor de descuento y la entropía objetivo, que son esenciales para equilibrar la exploración del espacio de acciones y la explotación del aprendizaje adquirido. Estos ajustes aseguran que el sistema no solo logre mantener la esfera en equilibrio, sino que también refine continuamente su comportamiento a través del aprendizaje en tiempo real.

El proyecto desafía la transición de teorías de control y aprendizaje automático a aplicaciones prácticas, implementando estos algoritmos avanzados directamente en el hardware. La elección del Arduino para ejecutar la inferencia subraya la importancia de los sistemas embebidos en la robótica moderna y demuestra la viabilidad de soluciones inteligentes en aplicaciones reales.

# Resum

Aquest Treball de Fi de Grau es centra en la integració de tecnologies avançades en robòtica i control automàtic, amb un especial èmfasi en l'aplicació d'intel·ligència artificial per abordar desafiaments complexos d'enginyeria. Desenvolupat en el marc del programa de Grau en Enginyeria Informàtica de la Facultat d'Informàtica de Barcelona, el projecte implementa un sistema d'equilibri dinàmic per a una esfera sobre un cilindre, utilitzant el robot UR3e de Universal Robots.

El cor del sistema és un microcontrolador Arduino, responsable de recollir dades del sensor de distància i comunicar-se en temps real amb el robot. Crucialment, aquest microcontrolador també executa la inferència de la xarxa neuronal entrenada amb l'algoritme d'aprenentatge per reforç Soft Actor-Critic (SAC), permetent aplicar directament les tècniques de control adaptatiu apreses en un entorn dinàmic i real.

La configuració del SAC ha estat meticulosament ajustada per adaptar-se a les incerteses inherents de l'entorn operatiu. Això inclou l'optimització de paràmetres com el factor de descompte i l'entropia objectiu, que són essencials per equilibrar l'exploració de l'espai d'accions i l'explotació de l'aprenentatge adquirit. Aquests ajustos asseguren que el sistema no només aconsegueixi mantenir l'esfera en equilibri, sinó que també refine contínuament el seu comportament a través de l'aprenentatge en temps real.

El projecte desafia la transició de teories de control i aprenentatge automàtic a aplicacions pràctiques, implementant aquests algoritmes avançats directament en el maquinari. L'elecció de l'Arduino per executar la inferència subratlla la importància dels sistemes encastats en la robòtica moderna i demostra la viabilitat de solucions intel·ligents en aplicacions reals.

# Abstract

This Bachelor's Thesis focuses on the integration of advanced technologies in robotics and automatic control, with a special emphasis on the application of artificial intelligence to address complex engineering challenges. Developed within the Computer Engineering program at the Faculty of Computer Science in Barcelona, the project implements a dynamic balancing system for a sphere on a cylinder, using the UR3e robot from Universal Robots.

The heart of the system is an Arduino microcontroller, responsible for collecting data from the distance sensor and communicating in real time with the robot. Crucially, this microcontroller also executes the inference of the neural network trained with the Soft Actor-Critic (SAC) reinforcement learning algorithm, allowing the direct application of adaptive control techniques learned in a dynamic and real environment.

The SAC setup has been meticulously adjusted to adapt to the inherent uncertainties of the operating environment. This includes optimizing parameters such as the discount factor and the target entropy, which are essential for balancing the exploration of the action space and the exploitation of the learned knowledge. These adjustments ensure that the system not only achieves maintaining the sphere in balance, but also continuously refines its behavior through real-time learning.

The project challenges the transition from control theories and machine learning to practical applications, implementing these advanced algorithms directly on the hardware. The choice of Arduino to run the inference underscores the importance of embedded systems in modern robotics and demonstrates the feasibility of intelligent solutions in real applications.

# Índice

<b>1. Introducción y contextualización.....</b>	<b>10</b>
1.1 Introducción.....	10
1.2 Contexto.....	11
1.3 Definición de términos y conceptos relevantes.....	11
1.3.1 Brazo Robótico UR3e y la Plataforma de Equilibrio.....	11
1.3.2 Arquitectura del Robot UR3e.....	12
1.3.3 Arduino Giga Rev1 WiFi y Sensores.....	15
1.3.4 Scheduler Cíclico con posible implementación de DM (Deadline Monotonic). .....	16
1.3.5 Proceso de Decisión de Markov.....	16
1.3.6 Aprendizaje por refuerzo.....	17
1.3.7 Herramientas de Implementación.....	21
1.4 Identificación del problema a resolver.....	22
1.5 Actores implicados.....	23
<b>2. Justificación.....</b>	<b>24</b>
<b>3. Alcance.....</b>	<b>27</b>
3.1 Objetivos y subobjetivos.....	27
3.2 Identificación de requerimientos.....	27
3.3 Posibles obstáculos y riesgos.....	28
<b>4. Metodología y rigor.....</b>	<b>30</b>
4.1 Descripción de la metodología aplicada.....	30
4.2 Validación.....	30
4.3 Herramientas.....	31
<b>5. Planificación temporal.....</b>	<b>32</b>
5.1 Definición de las Tareas.....	32
5.1.1 Gestión del proyecto (GP).....	32
5.1.2 Análisis (A).....	33
5.1.3 Diseño (D).....	34
5.1.4 Implementación (I).....	34
5.1.5 Evaluación (E).....	34
5.2 Recursos.....	35
5.2.1 Recursos Humanos.....	35
5.2.2 Recursos Materiales.....	35
5.3 Modificación temporal del proyecto.....	35
<b>5.4 Estimaciones y diagrama de Gantt.....</b>	<b>36</b>
5.4.1 Estimaciones (Periodo ordinario).....	36
5.4.2 Estimaciones (Periodo extraordinario).....	37
5.4.3 Diagrama de Gantt (Periodo ordinario).....	38
5.4.4 Diagrama de Gantt (Periodo extraordinario).....	39
<b>5.5 Gestión del riesgo: Planes alternativos y obstáculos.....</b>	<b>40</b>
5.5.1 Complejidad del Aprendizaje.....	40

5.5.2 Limitaciones de Hardware.....	40
5.5.3 Incompatibilidades de Software.....	40
5.5.4 Limitaciones de Tiempo.....	41
<b>6. Presupuesto.....</b>	<b>42</b>
6.1 Identificación de los costes.....	42
6.1.1 Costes de personal.....	42
6.1.2 Costes generalmente calculados.....	43
6.1.3 Contingencias.....	45
6.1.4 Imprevistos.....	45
6.1.5 Coste final.....	46
6.2 Control de gestión.....	46
<b>7. Informe de Sostenibilidad.....</b>	<b>48</b>
7.1 Autoevaluación.....	48
7.2 Dimensión Económica.....	48
7.3 Dimensión Ambiental.....	48
7.4 Dimensión Social.....	49
<b>8. Sinergia entre Conocimientos y Cumplimiento Normativo.....</b>	<b>50</b>
8.1 Integración de conocimientos.....	50
8.2 Identificación de leyes y regulaciones.....	51
<b>9. Descripción detallada del sistema Ball and Beam.....</b>	<b>52</b>
9.1 Principios físicos y matemáticos del sistema.....	52
9.2 Modelado matemático del Ball and Beam.....	55
9.2.1 Modelo de Newton.....	55
9.2.2 Modelo de Lagrange.....	57
9.3 Integración con el robot UR3e en el entorno del laboratorio.....	59
9.3.1 Integración del Hardware.....	59
9.3.2 Sensor de distancia láser VL53L0X.....	62
9.3.3 Latencias y Funcionamiento del Código en el Arduino.....	65
9.3.4 Control del robot UR3e.....	66
<b>10. Desarrollo del algoritmo Soft Actor Critic.....</b>	<b>70</b>
10.1 Fundamentos del Soft Actor Critic.....	70
10.2 Implementación del SAC.....	76
10.2.1 Detalles del Entorno.....	76
10.2.2 Redes Neuronales en SAC.....	78
<b>11. Integración del algoritmo Soft Actor Critic con Arduino.....</b>	<b>81</b>
11.1 Preparación de la Red Neuronal.....	81
11.2 Ejecución de la inferencia.....	82
11.3 Planificador Cíclico.....	84
<b>12. Pruebas y ajustes.....</b>	<b>86</b>
12.1 Configuración Experimental.....	86
12.2 Esfera Verde.....	87
12.3 Esfera de Metacrilato.....	89
12.4 Esfera de Golf.....	91

12.5 Análisis y Conclusiones.....	93
<b>13. Conclusiones, conocimientos adquiridos y trabajo futuro.....</b>	<b>96</b>
13.1 Conclusiones.....	96
13.2 Conocimientos adquiridos y limitaciones.....	97
13.3 Trabajo Futuro.....	98
<b>Referencias.....</b>	<b>100</b>

# Índice de figuras

1. Robot UR3e.....	10
2. Configuración del robot colaborativo UR3e - Vista frontal (imagen izquierda), vista lateral (imagen derecha).....	11
3. Configuración de la viga. Fuente: Elaboración propia.....	12
4. Teach Pendant.....	12
5. Polyscope.....	13
6. Control Box.....	13
7. Sistema robótico UR3e.....	14
8. Arquitectura del sistema robótico UR3.....	15
9. Diagrama de conexión de pines del Arduino Giga Rev1 WiFi.....	15
10. Diagrama esquemático del sensor de distancia VL53L0X.....	16
11. Esquema del Proceso de Decisión de Markov.....	17
12. Esquema del aprendizaje por refuerzo.....	18
13. Interfaz de Usuario del ArduinoIDE.....	21
14. Esquema del método Kanban.....	30
15. Diagrama de Gantt (periodo ordinario).....	38
16. Diagrama de Gantt (periodo extraordinario).....	39
17. Diagrama de Fuerzas en el sistema Ball and Beam.....	54
18. Conexión del Arduino Giga Rev1 WiFi con el sensor VL53L0X.....	60
19. Sistema Ball and Beam en el entorno del laboratorio - Vista frontal (imagen izquierda), vista lateral (imagen derecha).....	60
20. Esferas: Metacrilato, Verde, Golf.....	61
21. Nivel del ruido del sensor láser para el modo de funcionamiento estándar.....	63
22. Calibración de las lecturas del sensor láser mediante un ajuste cuadrático.....	64
23. Diagrama Actor-Crítico.....	70
24. Pseudocódigo del algoritmo Soft Actor-Critic.....	74
25. Ejemplo de la red neuronal del actor.....	79
26. Arquitectura de red neuronal usada en SAC.....	80
27. Visualización de la Red de Política en formato ONNX.....	82
28. Configuración del TensorFlow Lite para el Arduino.....	83
29. Implementación de la función de inferencia de la política del agente en Arduino.....	84
30. Alpha loss esfera Verde.....	88
31. Loss criticos, actor y alpha esfera Verde.....	89
32. Ganancia acumulada, actor y alpha esfera Verde.....	89
33. Alpha loss esfera de Metacrilato.....	90
34. Loss criticos, actor y alpha esfera de Metacrilato.....	91
35. Ganancia acumulada, actor y alpha esfera de Metacrilato.....	91
36. Alpha loss esfera de Golf.....	92
37. Loss criticos, actor y alpha esfera de Golf.....	92
38. Ganancia acumulada, actor y alpha esfera de Golf.....	93

# Índice de tablas

1. Ventajas y Desafíos de Técnicas de Inteligencia Artificial (IA) .....	25
2. Tabla de tareas con la duración y las dependencias (periodo ordinario) .....	36
3. Tabla de tareas con la duración y las dependencias (periodo extraordinario) .....	37
4. Sueldos por hora para cada rol.....	42
5. Tabla de tareas con la duración, las dependencias, los roles y los costes de personal por tarea.....	43
6. Costes de Software.....	44
7. Costes de hardware.....	44
8. Costes indirectos.....	45
9. Tabla contingencia del 20 %.....	45
10. Costes de imprevistos.....	46
11. Tabla del Coste total.....	46
12. Tabla de Seguimiento y Control del Proyecto.....	47
13. Pesos de las diferentes esferas.....	61
14. Especificaciones de Modos Operativos del Sensor VL53L0X.....	62
15. Precisiones de referencia para los modos de funcionamiento estándar y largo alcance..	62
16. Tabla de las latencias.....	65
17. Tabla de principales interficies de comunicación con el robot UR3e.....	67
18. Tabla de hiperparámetros para el entrenamiento.....	86
19. Valores de Hiperparámetros para las Diferentes Esferas.....	93
20. Valores de hiperparámetros para diferentes configuraciones de la esfera Verde.....	94

# 1. Introducción y contextualización

Este proyecto se enmarca dentro del Grado en Ingeniería Informática, en la especialidad de Computación, de la Facultad de Informática de Barcelona de la Universitat Politècnica de Catalunya.

## 1.1 Introducción

En la actualidad, la integración de técnicas de aprendizaje avanzadas en sistemas de control en tiempo real aplicados a la robótica, se ha convertido en una área básica de investigación y desarrollo. La capacidad de los robots para aprender y adaptarse de manera autónoma a su entorno, en lugar de estar preprogramados para tareas específicas, promete revolucionar industrias, desde la fabricación hasta la atención médica. Una de las técnicas que ha recibido considerable atención es el aprendizaje por refuerzo (RL), un tipo de aprendizaje automático en el que un agente aprende interactuando con su entorno y recibiendo retroalimentación en forma de recompensas o penalizaciones. Un claro reflejo de la relevancia y aplicabilidad de esta técnica en la robótica se encuentra en [1].

En esta línea, el robot UR3e, un producto de Universal Robots, es un brazo robótico compacto, ligero y versátil diseñado para una multitud de aplicaciones. Este proyecto tiene como objetivo aprovechar técnicas de inteligencia artificial, como el aprendizaje por refuerzo, y combinarlas con principios de sistemas de control en tiempo real. La capacidad de un robot para equilibrar una pelota puede parecer una tarea sencilla, pero refleja muchas aplicaciones industriales donde la precisión, la estabilidad y las respuestas en tiempo real son cruciales.

Esta precisión y estabilidad no solo es esencial en tareas como equilibrar objetos, sino también en aplicaciones más complejas y delicadas. Por ejemplo, en la fabricación donde los robots necesitan manipular piezas delicadas con cuidado y en la atención sanitaria donde deben realizar movimientos precisos durante las cirugías. En la Figura 1 se puede apreciar la estructura del robot UR3e.



Figura 1: Robot UR3e. Fuente: [2]

## 1.2 Contexto

Este proyecto se centra en el uso del brazo robótico UR3e para realizar una tarea altamente específica y técnica: el equilibrio de una esfera dentro de un cilindro que el propio robot puede rotar. La finalidad es mantener la esfera en una posición estacionaria predeterminada, demostrando así la habilidad del robot para ejecutar tareas de precisión en tiempo real. Para lograr este objetivo, se utiliza un sensor de distancia que permite al robot monitorear continuamente la posición de la esfera y ajustar sus movimientos en consecuencia.

El proceso comienza con el envío de comandos de movimiento al robot a través de una conexión *WiFi*, lo que permite una interacción dinámica y adaptable con el entorno inmediato del robot. Se entrena un algoritmo de aprendizaje por refuerzo en un ordenador portátil. Este algoritmo, una vez optimizado, se transfiere a un Arduino, que actúa como el cerebro operativo, ejecutando la política de control aprendida por el algoritmo y manejando la comunicación directa con el robot para realizar ajustes en tiempo real.

## 1.3 Definición de términos y conceptos relevantes

### 1.3.1 Brazo Robótico UR3e y la Plataforma de Equilibrio

- **Brazo Robótico UR3e:** Es un robot colaborativo diseñado por Universal Robots. Está construido para ejecutar tareas de precisión y montaje ligero, y puede manejar una carga útil de 3kg. El robot es programable, con una interfaz [2] fácil de usar. En la Figura 2, se muestra la perspectiva frontal y la lateral del Brazo Robótico UR3e.

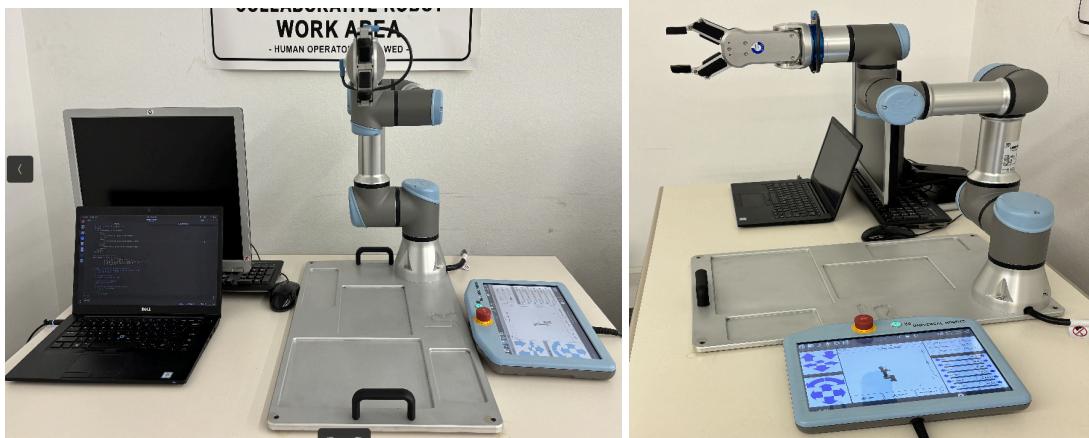


Figura 2: Configuración del robot colaborativo UR3e - Vista frontal (imagen izquierda), vista lateral (imagen derecha). Fuente: Elaboración propia

- **Ordenador portátil:** Este dispositivo se utiliza para la implementación y entrenamiento del algoritmo de aprendizaje automático que controla el brazo robótico. Se puede observar en la Figura 2 al lado del brazo robótico.

- **Plataforma de Equilibrio:** El sistema consiste en un *beam*(viga) representado por un tubo con una longitud de 1 metro y un diámetro de 50 mm, tal como se puede apreciar en la Figura 3. El *beam* tiene la capacidad de inclinarse hacia la derecha o hacia la izquierda, de esta forma se modifica la posición de la esfera situada en su interior.



Figura 3: Configuración de la viga. Fuente: Elaboración propia

### 1.3.2 Arquitectura del Robot UR3e

El control del brazo robótico *UR3e* no se realiza de manera directa, sino que implica la coordinación de varios componentes que conforman su sistema. Entre los elementos que componen el sistema robótico *UR3e*, excluyendo el brazo robótico, se encuentran los siguientes componentes:

- **Teach Pendant:** El *teach pendant*, se puede ver en la Figura 4, es un dispositivo de control manual que permite a los operadores enseñar o programar al robot directamente. Este dispositivo portátil incluye una pantalla táctil que ejecuta *Polyscope* y botones físicos para la manipulación directa y precisa del robot. Es utilizado comúnmente para la configuración inicial del robot, así como para ajustes finos o para operaciones manuales durante el mantenimiento o la programación compleja.



Figura 4: Teach Pendant. Fuente: [3]

- **Polyscope:** Es la interfaz de usuario desarrollada por *Universal Robots*, se puede apreciar en la Figura 5, para sus robots colaborativos escrita en el lenguaje de programación *Java*. Esta interfaz gráfica es intuitiva y está diseñada para simplificar la programación de robots. *Polyscope* es fundamental para facilitar la interacción entre los usuarios y los robots, permitiendo ajustes rápidos y personalización del comportamiento del robot sin necesidad de conocimientos avanzados de programación.

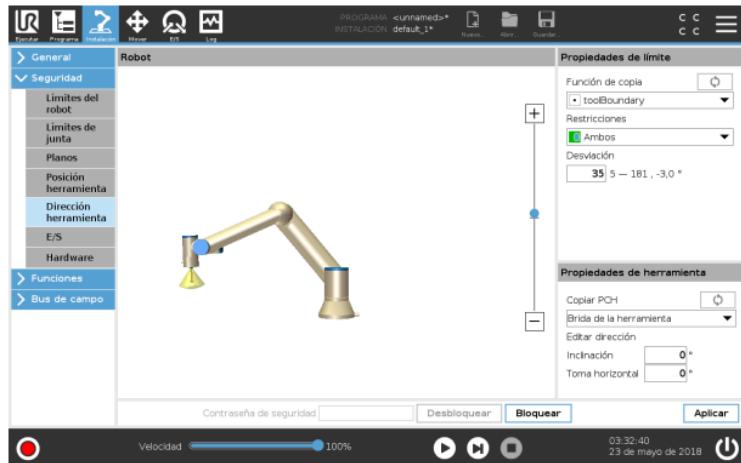


Figura 5: Polyscope. Fuente: [3]

- **Control Box:** El control box es la unidad central que aloja el *hardware* de control y los componentes electrónicos necesarios para operar el robot *UR3e*. Funciona como el cerebro del robot, procesando las entradas de los sensores y enviando comandos a los actuadores del robot. El control box contiene la fuente de alimentación, las placas de circuito, y las interfaces para la conectividad externa, asegurando que todas las operaciones del robot sean coordinadas y gestionadas eficazmente. En la Figura 6 se puede ver el *control box*:



Figura 6: Control Box. Fuente: [3]

- **URControl:** Es el *software* y los sistemas de control integrados que gestionan las operaciones del robot *UR3e*. Es parte del *control box* y es responsable de ejecutar los algoritmos de control en tiempo real, procesando los datos de los sensores y activando los actuadores según las instrucciones definidas por el usuario a través de *Polyscope* o a través de *Ethernet/IP*. *URControl* asegura que el robot responda de manera precisa y eficiente a las instrucciones programadas, garantizando la seguridad y la funcionalidad en entornos colaborativos.
- **URCAP:** Es un sistema de plugins desarrollado por *Universal Robots* para expandir y personalizar las funcionalidades de sus robots colaborativos. Los *URCaps* permiten a

los usuarios y desarrolladores integrar *software* y *hardware* adicionales que no están incluidos en el sistema estándar del robot. Esto se logra mediante la creación de aplicaciones específicas que pueden ser instaladas directamente en el *PolyScope*, proporcionando nuevas características o mejorando las existentes.

- **URScript:** Es el lenguaje de programación desarrollado por *Universal Robots* para la programación de sus robots. Permite a los usuarios escribir scripts personalizados que pueden ejecutarse directamente en el *URControl*, proporcionando una forma simple y eficiente de control sobre el comportamiento del robot. Es ideal para tareas que requieren movimientos complejos o secuencias de acciones personalizadas.

El *control box* actúa como el centro de conexión entre el brazo robótico y el *teach pendant*. El sistema *hardware* completo se puede observar en la Figura 7:



Figura 7: Sistema robótico UR3e. Fuente: [3]

La arquitectura del sistema robótico de *Universal Robots* se ilustra en la Figura 8, mostrando cómo el software *PolyScope* genera y envía *URScript* a *URControl* para la ejecución de comandos en el robot. La interacción puede incluir comunicaciones opcionales con un *Daemon* personalizado mediante *UR-Caps*, que permiten la extensión de funcionalidades mediante *XML-RPC* o sockets *TCP*, facilitando así la personalización avanzada y el control del robot.

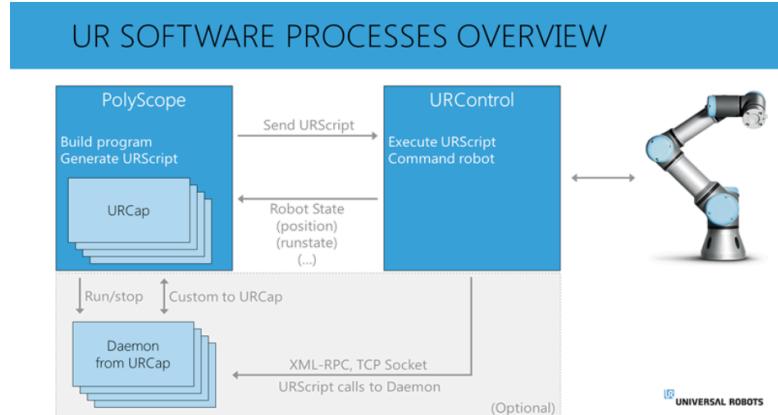


Figura 8: Arquitectura del sistema robótico UR3e. Fuente: [4]

### 1.3.3 Arduino Giga Rev1 WiFi y Sensores

En el contexto de este proyecto, los sensores son dispositivos esenciales que proporcionan información en tiempo real sobre el estado del sistema. Estos pueden incluir acelerómetros para medir la inclinación de la plataforma, sensores de distancia para determinar la posición de la esfera, y encoders para conocer la posición de los motores. El Arduino Giga Rev1[5] con su potente microcontrolador STM32H747XI[6] recopila y procesa esta información, permitiendo que el sistema tome decisiones informadas sobre cómo mover el brazo robótico y la plataforma para mantener la esfera equilibrada. Además, su conectividad *WiFi*, mediante el módulo Murata 1DX[7], permite una comunicación eficaz y segura. En la Figura 9 se puede ver el diagrama del Arduino:

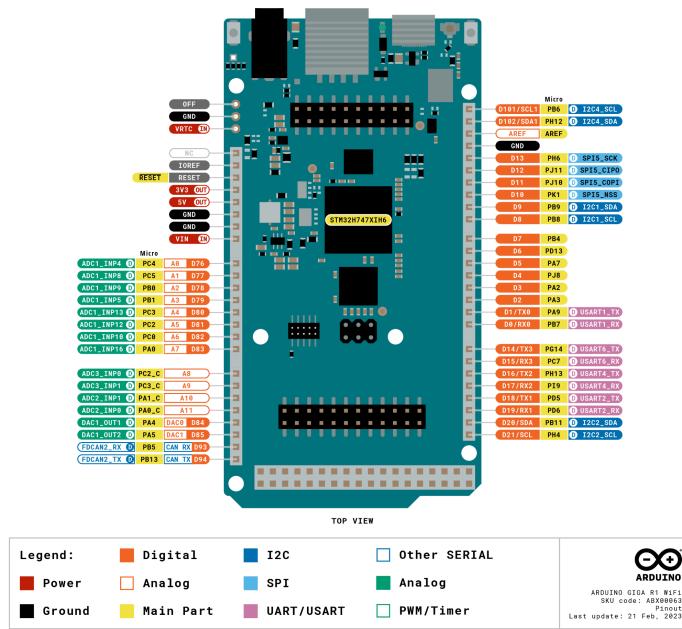


Figura 9: Diagrama de conexión de pines del Arduino Giga Rev1 WiFi. Fuente: [5]

Para medir la distancia, se utilizará el sensor VL53L0X[8], un sensor de tiempo de vuelo (*Time of Flight*, ToF) láser que mide la distancia al detectar el tiempo que tarda la luz en ir desde el sensor hasta un objeto y regresar. Este método permite mediciones precisas y

rápidas, ideal para el sistema Ball and Beam. El sensor estará situado en el extremo derecho del tubo, proporcionando datos cruciales sobre la posición de la esfera para el control del equilibrio de está. En la Figura 10 se presenta el esquemático del sensor:

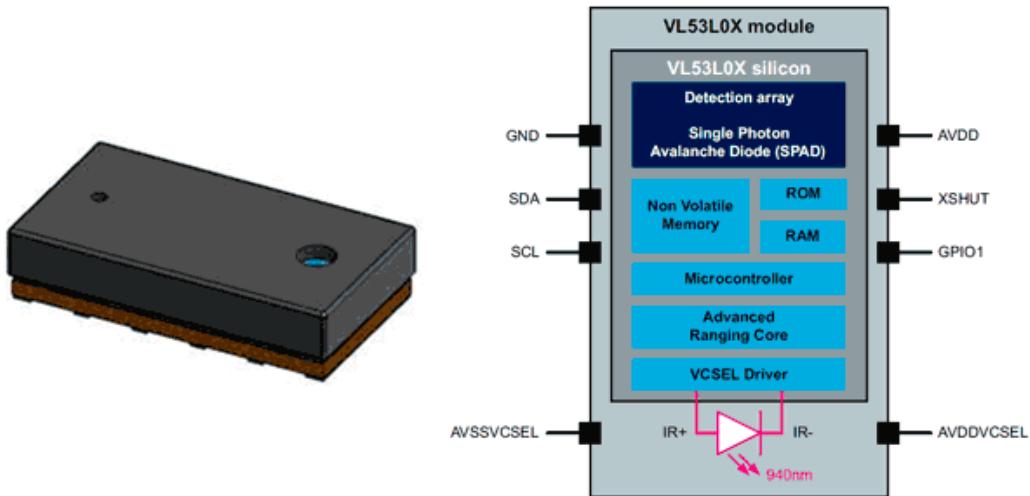


Figura 10: Diagrama esquemático del sensor de distancia VL53L0X. Fuente: [9]

#### 1.3.4 Scheduler Cíclico con posible implementación de DM (*Deadline Monotonic*)

Inicialmente, se implementará un scheduler cíclico para la gestión de tareas en tiempo real. Este enfoque es idóneo para sistemas con tareas que requieren una ejecución periódica y predecible.

Posteriormente, y en función del rendimiento del scheduler cíclico, se considerará la adopción del algoritmo Deadline Monotonic (DM). DM es un algoritmo de planificación basado en prioridades estáticas, donde las tareas tienen prioridades según sus deadlines: a menor plazo, mayor prioridad. Es especialmente adecuado para sistemas con tareas periódicas y plazos determinísticos [10].

#### 1.3.5 Proceso de Decisión de Markov

Un Proceso de Decisión de Markov (PDM) es la forma matemática de modelar la toma de decisiones en escenarios donde los resultados están influenciados tanto por factores aleatorios como por decisiones controladas. Un PDM se define por una tupla de 4 elementos:

- Conjunto de estados (**S**): Representa todos los posibles escenarios o configuraciones en las que puede estar el sistema.
- Conjunto de acciones (**A**): Enumera todas las posibles acciones que se pueden tomar en cada estado.

- Función de transición (**P**): Que define la probabilidad de transitar a un nuevo estado dado un estado actual y una acción realizada.
- Función de recompensa (**R**): Proporciona una señal inmediata de recompensa (o penalización) después de tomar una acción en un estado particular. Puede ser un valor fijo o una distribución probabilística dependiendo del estado y la acción.

En la Figura 11 se puede observar un ejemplo de un PDM:

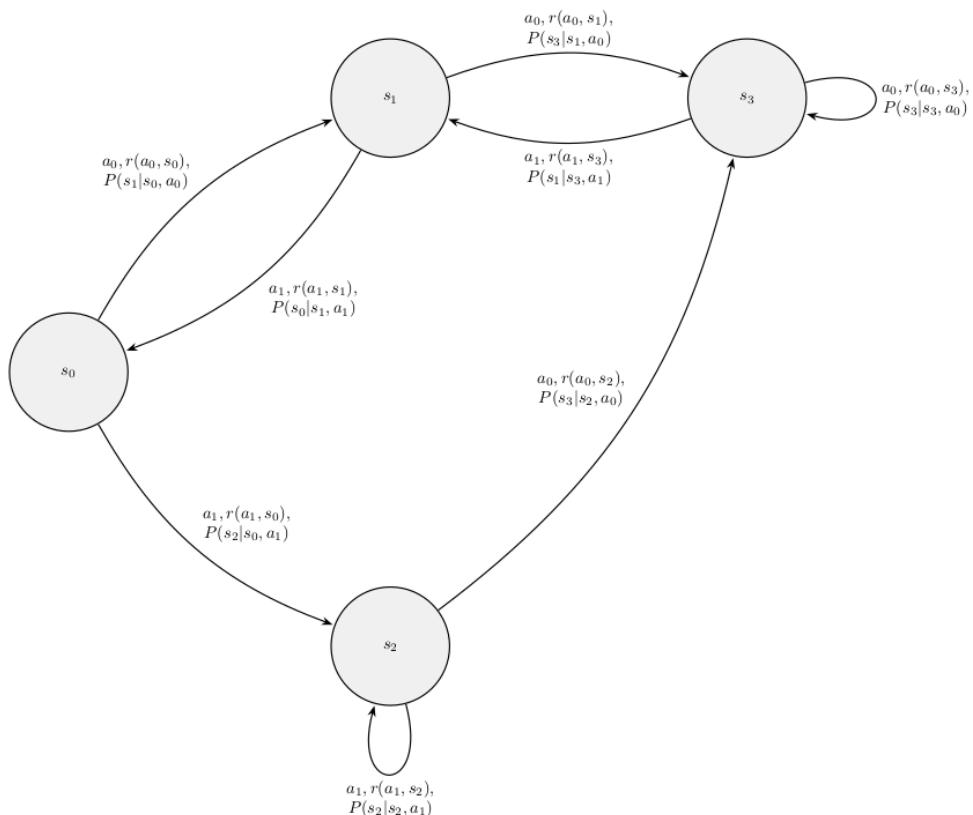


Figura 11: Esquema del Proceso de Decisión de Markov. Fuente: Elaboración propia

### Propiedad de Markov:

Para que un sistema sea considerado Markoviano, debe cumplir con la propiedad de Markov, que se define de la siguiente manera:

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, \dots, S_{t-1}, S_t) \quad (1.1)$$

Establece que el futuro es independiente del pasado, dado el presente. Esto implica que la probabilidad de moverse al siguiente estado depende únicamente del estado actual y de la acción seleccionada, sin ser influenciada por la secuencia de eventos o estados anteriores.

### 1.3.6 Aprendizaje por refuerzo

El Aprendizaje por Refuerzo (RL) es una técnica de aprendizaje automático donde un agente aprende a tomar decisiones interactuando con un entorno. A diferencia de otros métodos, en RL, el agente no recibe ejemplos explícitos de decisiones correctas, sino que descubre qué

acciones maximizan una recompensa a través del ensayo y error. En el contexto del sistema de equilibrio de la esfera, **RL** podría ser usado para entrenar al sistema a equilibrar la esfera de manera más eficiente. El proceso de **RL** se modela comúnmente como un Proceso de Decisión de Markov. En la Figura 12 se presenta el esquema general del aprendizaje por refuerzo:

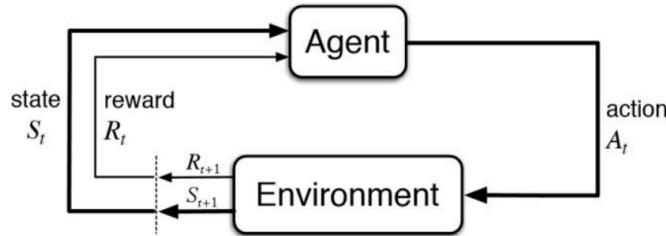


Figura 12: Esquema del aprendizaje por refuerzo. Fuente: [11]

En el ámbito del aprendizaje por refuerzo, se destacan como fundamentales los siguientes conceptos:

- **Política:** Define la manera en que el agente se comporta en un momento dado. Esencialmente, es un mapeo desde los estados percibidos del entorno hacia las acciones que el agente elige ejecutar.

$$\begin{aligned} \pi : s \in S &\rightarrow a \in A \\ a = \pi(s), \text{ Si la política es determinista} & \\ \pi(a|s) = P[A_t = a|S_t = s], \text{ Si la política es estocástica} & \end{aligned} \quad (1.2)$$

- **Modelo:** En métodos basados en modelo, representa las transiciones entre estados y las recompensas. Un modelo puede predecir qué sucederá cuando se toma una acción en un estado particular. En métodos no basados en modelo, el agente no tiene conocimiento de este.
- **Recompensa (Ganancia) Inmediata:** Es un valor escalar que el agente recibe como *feedback* y depende del estado actual. Esta recompensa,  $r_t$ , es proporcionada por la función de recompensa  $R$ , que determina la recompensa inmediata en cada paso del ciclo de vida del agente. Aunque esta recompensa evalúa directamente la acción tomada, en realidad refleja la bondad de toda la cadena de acciones, o trayectoria, hasta ese momento.

$$R(s, a) = E[r_{t+1}|S_t = s, A_t = a] \quad (1.3)$$

Esto significa que la recompensa esperada  $r_{t+1}$  depende del estado actual  $s$  y la acción tomada  $a$ , considerando todas las posibles consecuencias de esa acción.

- **Recompensa (Ganancia) de Larga Duración:** Considera no solo las recompensas inmediatas sino también las futuras. Esta ganancia se modela típicamente como un retorno descontado de horizonte infinito, donde  $R_t$ , es la recompensa total descontada a partir del paso de tiempo  $t$ :

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (1.4)$$

Aquí,  $\gamma$  (el factor de descuento) entre  $[0, 1]$ , dicta el valor presente de las recompensas futuras. Un  $\gamma$  cercano a 1 favorece una evaluación más previsora, valorando las recompensas futuras casi tan altamente como las inmediatas. Mientras que un valor de  $\gamma$  cercano a 0 conduce a una evaluación miópica, priorizando las recompensas inmediatas.

- **Función de Valor:** Estima cuánto beneficio (ganancia total futura) puede obtener el agente desde un estado o acción particular, bajo una política específica. Es decir, evalúa la bondad o maldad de los estados o acciones dentro del entorno, dependiendo de la política del agente. Esta función es crucial para decidir entre las acciones posibles en cada estado, ya que proporciona una estimación del beneficio total esperado al seguir una política desde un estado dado. En términos técnicos, se utiliza para evaluar la expectativa de la ganancia total, que incluye tanto recompensas inmediatas como futuras bajo una política específica. Usualmente hay dos tipos:

- **Función de valor estado:** Evalúa la calidad de un estado, independientemente de la acción tomada. Se define como el retorno esperado comenzando desde el estado  $s$  y luego siguiendo la política  $\pi$ :

$$V^\pi(s) = E^\pi[R_t | S_t = s] = E^\pi \left( \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| S_t = s \right) \quad (1.5)$$

- **Función Q (acción valor):** Una variante de la función de valor que además toma en cuenta la acción que se tomará bajo el estado actual. Muestra la utilidad esperada de tomar una acción particular en un estado dado, siguiendo una política determinada. Se define como el retorno esperado comenzando desde el estado  $s$ , tomando la acción  $a$ , y luego siguiendo la política  $\pi$ :

$$Q^\pi(s, a) = E^\pi[R_t | S_t = s, A_t = a] = E^\pi \left( \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| S_t = s, A_t = a \right) \quad (1.6)$$

- **Ecuación de Bellman:** La ecuación de Bellman es una herramienta fundamental en el aprendizaje por refuerzo, utilizada para describir la relación entre el valor de un estado y los valores de los estados sucesores. Se utiliza para calcular la función valor o la función acción-valor, basándose en la descomposición recursiva de las expectativas de recompensa.

- Ecuación de Bellman para la función de valor de estado ( $V$ ):

$$V^\pi(s) = E^\pi [r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s] \quad (1.7)$$

Esta expresión indica que el valor de estar en un estado  $s$  bajo una política  $\pi$ , es el valor esperado de la recompensa inmediata más el valor descontado del estado siguiente, asumiendo que el agente continúa siguiendo la política  $\pi$ .

- Ecuación de Bellman para la función de valor de acción ( $Q$ ):

$$Q^\pi(s, a) = E^\pi [r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi(s_{t+1})) \mid s_t = s, a_t = a] \quad (1.8)$$

Esta variante considera la acción a tomada en el estado  $s$ . El valor  $Q$  de tomar una acción en un estado dado y luego seguir la política  $\pi$ , es el valor esperado de la recompensa inmediata más el valor descontado de la acción en el estado sucesor según la política  $\pi$ .

Generalmente, los métodos no basados en modelos de **RL** se pueden clasificar en tres categorías principales, basadas en el enfoque que utilizan para aprender y tomar decisiones:

1. **Métodos Basados en Valor:** Estos métodos estiman el valor de cada posible acción en un estado dado. El objetivo es aprender una función de valor que indique la calidad de cada acción, facilitando la elección de la mejor acción en cada paso. Un ejemplo es el algoritmo **Q-Learning**, que aprende la función de valor  $Q$ , representando la utilidad esperada de tomar una acción en un estado particular y siguiendo una política óptima después.
2. **Métodos Basados en Política:** A diferencia de los métodos basados en valor, los métodos basados en política buscan aprender directamente la política de acción, un mapeo de estados a acciones, sin requerir una función de valor explícita. Un ejemplo serían los métodos de gradiente de política, como **Reinforce**.
3. **Métodos Actor-Crítico:** Los métodos actor-crítico representan una síntesis de los enfoques basados en valor y basados en política. En este esquema, el **actor** se encarga de seleccionar acciones conforme a una política, mientras que el **crítico** evalúa la acción tomada por el **actor** utilizando una función de valor. Este enfoque permite una mejora continua de la política del **actor** basada en las críticas, facilitando un equilibrio efectivo entre la exploración de nuevas acciones y la explotación de las ya conocidas para maximizar la recompensa. Un ejemplo sería el **Soft-Actor Critic (SAC)**.

Además de la clasificación basada en la estructura de los algoritmos, es crucial diferenciar entre las políticas on-policy y off-policy, las cuales definen cómo el agente interactúa con y aprende del entorno:

- **On-Policy:** Estos métodos requieren que la política utilizada para tomar decisiones sea la misma que se evalúa y mejora. Esto significa que la política que determina las acciones del agente es la misma que se optimiza. La ventaja es que la política siempre se adapta basándose en las experiencias directas derivadas de sus acciones, lo que puede resultar en aprendizajes más estables y predecibles.
- **Off-Policy:** En contraste, los métodos off-policy permiten que la política aprendida sea diferente de la política utilizada para generar datos. Esto significa que el aprendizaje se puede hacer desde la experiencia recogida de una política anterior o diferente. Por lo tanto posibilita que el agente explore y aprenda sobre óptimas potenciales sin el riesgo de las consecuencias directas de explorar acciones subóptimas.

### 1.3.7 Herramientas de Implementación

En el desarrollo de este proyecto, se utilizarán diversas herramientas de *software* que facilitan la implementación de los algoritmos y la programación del *hardware* necesario:

- **ArduinoIDE:** Para programar el Arduino que interactúa directamente con el *hardware*. Es un entorno de desarrollo integrado que facilita la escritura, compilación y carga de programas en placas Arduino. El lenguaje de programación es el C++, que es el soporte nativo para Arduino. En la Figura 13 se puede apreciar la interfaz del ArduinoIDE:

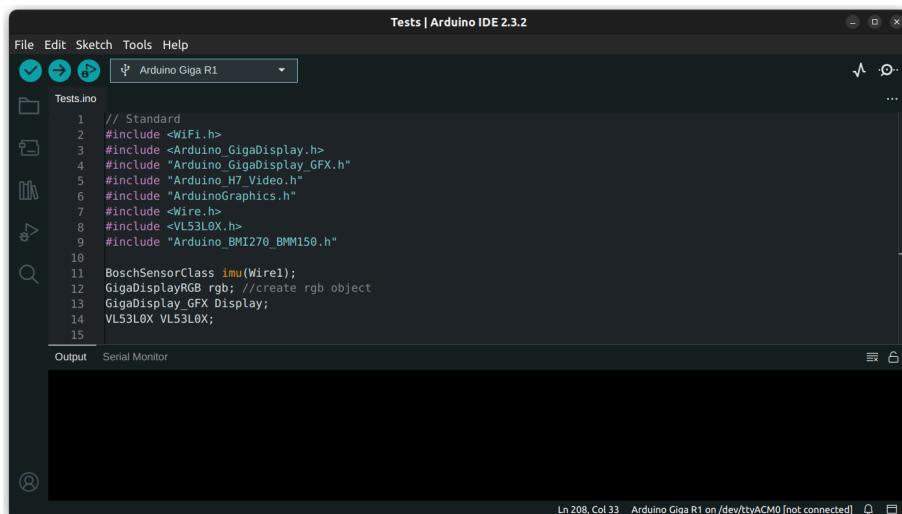


Figura 13: Interfaz de Usuario del ArduinoIDE. Fuente: Elaboracion propia

- **Python y Librerías de Aprendizaje por Refuerzo:** Para la implementación de los algoritmos de aprendizaje por refuerzo, se usará el lenguaje de programación *Python*, ya que cuenta con una poderosa colección de librerías científicas y de *machine learning*. Utilizaremos las siguientes librerías especializadas:

- **Pytorch:** PyTorch es una biblioteca de aprendizaje automático que facilita tanto el prototipado rápido como la implementación eficaz de modelos complejos, gracias a su interfaz intuitiva y capacidades de diferenciación automática.
- **Tensorflow:** TensorFlow es otra poderosa biblioteca de aprendizaje automático similar a *Pytorch*, conocida por su robusta infraestructura y escalabilidad en la producción de modelos.
- **Tensorflow Lite:** *TensorFlow Lite* es una versión más ligera de TensorFlow diseñada para dispositivos móviles y embebidos. Una vez que el modelo de aprendizaje por refuerzo esté entrenado y probado con TensorFlow, lo convertiremos a *TensorFlow Lite* para su despliegue en el arduino. Esto nos permite ejecutar inferencias de forma eficiente en el dispositivo, lo que es esencial para aplicaciones en tiempo real donde la rapidez y la eficiencia energética son cruciales.

## 1.4 Identificación del problema a resolver

El principal problema que se busca abordar en este proyecto es el de enseñar a un robot, específicamente al *UR3e*, a mantener una esfera en equilibrio sobre una viga en movimiento. Este desafío, aunque puede parecer simple a primera vista, encapsula diversas complejidades tanto en términos de control robótico como en términos de procesamiento en tiempo real. A continuación, se presentan las principales dificultades y características asociadas a este problema:

- **Complejidad del Sistema:** Mantener una esfera en equilibrio sobre una viga implica compensar de manera continua las fuerzas de la gravedad y las perturbaciones externas. Esto requiere que el robot pueda hacer movimientos precisos y rápidos en respuesta a cualquier cambio en la posición de la esfera.
- **Procesamiento en Tiempo Real:** Cualquier retraso en la respuesta del sistema puede resultar en que la esfera se caiga de la plataforma. Por ello, es esencial que el sistema pueda procesar información de los sensores y actuar sobre el brazo robótico en tiempo real.
- **Optimización:** Mientras que un enfoque de control convencional podría ser capaz de equilibrar la esfera, el uso de técnicas de aprendizaje por refuerzo podría permitir que el sistema aprenda y optimice su respuesta con el tiempo, logrando un equilibrio más estable.
- **Integración de los Componentes:** El sistema propuesto no solo depende del brazo robótico y la plataforma, sino también de una variedad de sensores, actuadores y el

microcontrolador Arduino. Estos componentes deben trabajar en conjunto de manera organizada para lograr el objetivo.

- Robustez frente a Perturbaciones: En un entorno real, es probable que el sistema tenga perturbaciones externas, como golpes o cambios en la superficie de la viga.

## 1.5 Actores implicados

Tenemos varios actores implicados en el proyecto:

- El Estudiante: Tiene el rol del principal investigador y desarrollador del proyecto. Tiene como interés adquirir habilidades y conocimientos en inteligencia artificial, robótica y sistemas de tiempo real.
- Director del TFG: Tiene el rol de guía académico y principal revisor del proyecto. Su interés es el de asegurar que el proyecto cumpla con los estándares académicos y técnicos.
- La Universidad: Tiene el rol de proveedor de recursos, infraestructura y asesoría académica. Su principal interés es promover investigaciones, fortalecer su posición en el campo de la robótica y la inteligencia artificial.
- Comunidad Educativa (maestros, alumnos): Tienen el rol de beneficiarios indirectos al ser expuestos al proyecto en contextos educativos. La idea es fomentar el interés en la robótica y la inteligencia artificial, proporcionar inspiración para futuras carreras o investigaciones en el campo.

## 2. Justificación

El desafío de equilibrar una esfera en una viga utilizando robots ha sido un tema de interés en la comunidad de robótica y control durante varios años. Esta tarea, a simple vista elemental, en realidad representa un problema complejo de control y adaptabilidad. Para abordar este problema, se han propuesto diferentes soluciones a lo largo del tiempo:

1. Aprendizaje Supervisado: Una de las aproximaciones más directas consiste en mover manualmente el robot o programarlo para que realice ciertos movimientos predefinidos, recolectando datos en el proceso. Posteriormente, estos datos son utilizados para entrenar un modelo de aprendizaje automático que permite al robot simular e imitar el comportamiento observado. Si bien esta técnica ha demostrado ser efectiva en muchos contextos, posee limitaciones, en particular, el sistema resultante puede no ser lo suficientemente adaptable o robusto frente a perturbaciones imprevistas o escenarios que no estuvieron presentes durante la fase de recolección de datos.
2. Algoritmos Genéticos: Estos se utilizan para optimizar los parámetros del controlador, tratando de encontrar la combinación óptima que permita al robot mantener el equilibrio de la esfera. Aunque los algoritmos genéticos pueden llegar a ofrecer buenos resultados, su proceso de optimización puede ser lento y no garantiza encontrar la solución óptima.
3. Redes Neuronales de Retroalimentación: Estas redes pueden modelar y predecir dinámicas complejas. Sin embargo, entrenarlas puede ser un desafío, y dependiendo de la arquitectura y configuración, pueden no ser las más adecuadas para la adaptabilidad en tiempo real requerida en este proyecto.
4. Controladores Proporcional-Integral-Derivativo (PID) convencionales: Tradicionalmente, se han utilizado en tareas de control debido a su simplicidad y efectividad en condiciones estables. Sin embargo, para aplicaciones como el equilibrio de una esfera en movimiento sobre una viga, los PID no son ideales debido a su limitada capacidad de adaptarse a situaciones cambiantes y dinámicas, lo que restringe su utilidad en entornos que requieren alta adaptabilidad.
5. Algoritmos de Aprendizaje por Refuerzo, como *Twin Delayed DDPG (TD3)*, *Soft Actor-Critic (SAC)* o *Deep Q-Network (DQN)*: Estos algoritmos permiten al robot aprender a través de la interacción con su entorno, siendo especialmente adecuados para problemas donde la solución no es trivialmente aparente o cuando se desea que el sistema sea capaz de adaptarse a cambios o perturbaciones en tiempo real.

En la Tabla 1 se presenta un resumen de las ventajas y desafíos de las técnicas presentadas:

Aspecto	Aprendizaje Supervisado	Algoritmos Genéticos	Redes Neuronales de Retroalimentación	Aprendizaje por Refuerzo	Controladores PID Convencionales
Fortalezas	<ul style="list-style-type: none"> <li>• Fácil implementación</li> <li>• Datos recopilados manualmente</li> <li>• Efectivo en contextos conocidos</li> </ul>	<ul style="list-style-type: none"> <li>• Optimización de parámetros</li> <li>• Puede ofrecer buenos resultados</li> </ul>	<ul style="list-style-type: none"> <li>• Modela dinámicas complejas</li> <li>• Preciso en ciertas configuraciones</li> </ul>	<ul style="list-style-type: none"> <li>• Adaptabilidad</li> <li>• Aprendizaje continuo</li> <li>• Bueno para problemas complejos</li> </ul>	<ul style="list-style-type: none"> <li>• Simplicidad</li> <li>• Efectividad en condiciones estables</li> </ul>
Debilidades	<ul style="list-style-type: none"> <li>• Falta de adaptabilidad</li> <li>• Sensible a perturbaciones</li> </ul>	<ul style="list-style-type: none"> <li>• Lento</li> <li>• No garantiza solución óptima</li> </ul>	<ul style="list-style-type: none"> <li>• Difícil de entrenar</li> <li>• Puede no ser adecuado para adaptabilidad en tiempo real</li> </ul>	<ul style="list-style-type: none"> <li>• Puede requerir mucho tiempo para entrenar</li> <li>• Requiere gran cantidad de datos</li> </ul>	<ul style="list-style-type: none"> <li>• Pobre adaptación a cambios dinámicos</li> <li>• Limitado en entornos que requieren alta adaptabilidad</li> </ul>
Oportunidades	<ul style="list-style-type: none"> <li>• Mejora con la adición de más datos y escenarios</li> </ul>	<ul style="list-style-type: none"> <li>• Acelera el proceso con hardware más potente</li> </ul>	<ul style="list-style-type: none"> <li>• Posible mejora con nuevas arquitecturas</li> </ul>	<ul style="list-style-type: none"> <li>• Integración con otros métodos</li> <li>• Optimización en tiempo real</li> </ul>	<ul style="list-style-type: none"> <li>• Mejoras con técnicas de control avanzadas</li> </ul>
Amenazas	<ul style="list-style-type: none"> <li>• Ineficaz en escenarios imprevistos</li> </ul>	<ul style="list-style-type: none"> <li>• Requiere muchos recursos computacionales</li> </ul>	<ul style="list-style-type: none"> <li>• Puede no ser adecuado para tiempo real</li> </ul>	<ul style="list-style-type: none"> <li>• Puede no converger</li> <li>• Sensible a la calidad de los datos</li> </ul>	<ul style="list-style-type: none"> <li>• Obsolescencia frente a nuevas tecnologías</li> </ul>

Tabla 1: Ventajas y Desafíos de Técnicas de Inteligencia Artificial (IA). Fuente: Elaboración propia

El proyecto tiene como objetivo principal abordar el reto de equilibrar una esfera en una viga con un robot UR3e utilizando técnicas de aprendizaje por refuerzo. El valor que aporta este proyecto se encuentra en la integración de sistemas de aprendizaje por refuerzo con sistemas en tiempo real, una conjunción que tiene el potencial de ofrecer una respuesta rápida y precisa al problema de equilibrio. Esta aproximación no solo se anticipa a las limitaciones de métodos previos, sino que también ofrece un sistema más robusto y adaptable, especialmente en escenarios no previstos o cambiantes. Respecto a otros enfoques, la versatilidad y la capacidad de adaptación de nuestro sistema en tiempo real lo distingue como una solución potencialmente más eficiente y efectiva.

Entre las diversas técnicas de aprendizaje por refuerzo disponibles, la elección del algoritmo *Soft Actor-Critic (SAC)* para este proyecto se debe a sus características distintivas que lo hacen particularmente adecuado para el desafío de equilibrar una esfera en una viga. A diferencia de otros métodos más simples de aprendizaje por refuerzo como *Q-Learning*, que utiliza una tabla de *Q-values* y se limita a entornos con espacios de acción discretos, SAC puede manejar de manera efectiva espacios de acciones y estados continuos. Esto es crucial para aplicaciones en robótica donde las acciones como los movimientos de un robot son inherentemente continuos y requieren un grado de precisión que los métodos discretos no pueden proporcionar.

Además, mientras que algoritmos como *Deep Q-Network (DQN)* adaptan los espacios de estado continuos a través de la aproximación de funciones, siguen limitando las acciones a un conjunto discreto, lo que puede no ser suficiente para controlar dinámicas complejas y rápidamente cambiantes como las presentes en este proyecto. SAC, por otro lado, utiliza una formulación basada en la maximización de una función de valor que tiene en cuenta tanto el retorno esperado como la entropía del espacio de acción. Esto no solo promueve la exploración eficaz durante el aprendizaje, sino que también ayuda a evitar los mínimos locales y asegura una mejor convergencia hacia políticas óptimas bajo condiciones inciertas y cambiantes.

La capacidad del SAC para manejar entornos con altos grados de libertad y su robustez ante las fluctuaciones inherentes a los escenarios en tiempo real hacen que sea la elección preferida para nuestra aplicación, superando las limitaciones observadas en técnicas de RL más tradicionales y menos flexibles.

### 3. Alcance

#### 3.1 Objetivos y subobjetivos

El objetivo principal es desarrollar un sistema *Ball and Beam (esfera y viga)* mediante el robot UR3e utilizando técnicas de aprendizaje por refuerzo, integrando principios de sistemas en tiempo real para garantizar una respuesta rápida y precisa.

Subobjetivos:

- O1 . Investigación y Selección de Algoritmos: Estudiar algoritmos relevantes de aprendizaje por refuerzo y seleccionar el más adecuado para el proyecto.
- O2 . Integración con el Robot UR3e: Asegurar que el algoritmo seleccionado pueda ser implementado y ejecutado eficientemente en el robot UR3e.
- O3 . Desarrollo de Sistemas en Tiempo Real: Integrar la lógica de aprendizaje por refuerzo con sistemas en tiempo real para asegurar respuestas precisas en tiempo real.
- O4 . Pruebas y Optimización: Realizar pruebas iterativas y ajustar el modelo para mejorar la precisión y estabilidad del sistema de equilibrio.
- O5 . Implementación del Sistema en Tiempo Real e inferencia en el Arduino: Se configurará inicialmente un sistema con un *scheduler* cíclico para la organización y ejecución de tareas periódicas. Dependiendo de los resultados y requerimientos de complejidad, se podría avanzar hacia la implementación del algoritmo *Deadline Monotonic (DM)* para una gestión de tareas basada en prioridades estáticas. También se realizará la implementación de la inferencia del modelo de aprendizaje por refuerzo para su ejecución integral en el Arduino.
- O6 . Documentación: Documentar todo el proceso de desarrollo, implementación y pruebas, asegurando que el sistema sea modular y extensible en el futuro.

#### 3.2 Identificación de requerimientos

Requerimientos funcionales:

- RF1. Integración con Sensores: El sistema debe ser capaz de recopilar datos en tiempo real de los sensores (acelerómetros, sensores de distancia, encoders).
- RF2. Control de la Plataforma: El sistema debe controlar la plataforma para mover la esfera y mantenerla en equilibrio.

RF3. Algoritmo de Aprendizaje: El sistema debe implementar un algoritmo de aprendizaje por refuerzo para adaptarse y optimizar el equilibrio de la esfera.

RF4. Registro y Monitoreo: El sistema debe registrar datos sobre cada sesión de equilibrio, incluyendo duración, cantidad de veces que la esfera se cayó, etc.

Requerimientos no funcionales:

RnF1. Tiempo Real: El sistema debe responder a los datos de los sensores en tiempo real para mantener la esfera en equilibrio.

RnF2. Adaptabilidad: El sistema debe ser capaz de adaptarse a diferentes entornos y perturbaciones externas.

RnF3. Robustez: El sistema no debe fallar ante pequeñas perturbaciones o errores en los datos de los sensores.

RnF4. Eficiencia: El sistema debe optimizar el uso de recursos computacionales y energéticos, garantizando un funcionamiento prolongado.

RnF5. Seguridad: Todas las acciones del robot deben ser seguras para evitar daños a personas, al propio robot, o al entorno.

RnF6. Interoperabilidad: El sistema debe ser compatible con diferentes versiones y así como con diferentes tipos y marcas de sensores.

RnF7. Escalabilidad: El sistema debe ser capaz de manejar la adición de más sensores o la integración con otros sistemas sin grandes modificaciones.

RnF8. Documentación: Toda la implementación del sistema debe estar bien documentada para facilitar la comprensión, modificación y expansión por parte de otros desarrolladores o investigadores.

### 3.3 Posibles obstáculos y riesgos

- Complejidad del Aprendizaje: El proceso de aprendizaje por refuerzo puede no converger o requerir más tiempo del esperado.
- Limitaciones de *Hardware*: Fallos en los sensores o en el robot UR3e pueden interrumpir el desarrollo o las pruebas.
- Incompatibilidades de *Software*: Pueden surgir problemas de compatibilidad al integrar distintos sistemas o herramientas.

- Sobrecarga del Sistema: La integración del aprendizaje por refuerzo con sistemas en tiempo real podría llevar a sobrecargas o latencias si no se maneja adecuadamente.
- Limitaciones de Tiempo: El tiempo para desarrollar, probar y optimizar podría ser insuficiente antes de la fecha de entrega del TFG.

Para abordar estos riesgos, es crucial adoptar un enfoque iterativo, realizando pruebas frecuentes, y siendo proactivo en la identificación y solución de problemas a medida que surgen. También sería útil contar con un plan de contingencia para cada posible obstáculo.

## 4. Metodología y rigor

### 4.1 Descripción de la metodología aplicada

Dada la naturaleza del proyecto y la necesidad de realizar múltiples pruebas en un período de tiempo limitado, se ha optado por implementar una metodología ágil de desarrollo de *software*. En particular, se utilizará el método *Kanban*. Una representación a alto nivel se puede observar en la Figura 14.

El proceso se dividirá en iteraciones semanales, cada una con distintas fases:

- Análisis y Diseño: Durante esta fase, se analizarán y diseñarán las soluciones que posteriormente se implementarán.
- Integración: Se integrarán las soluciones diseñadas para asegurar su correcta funcionalidad.
- Prueba: Las soluciones se probarán en el laboratorio con el robot UR3e.
- Validación: Se verificará que las soluciones cumplan con los requisitos establecidos.

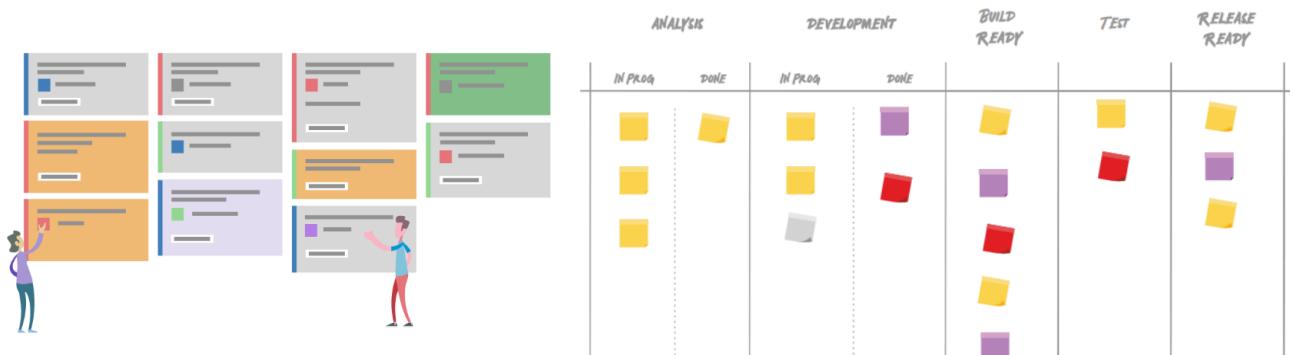


Figura 14: Esquema del método Kanban. Fuente: [12][13]

El objetivo es, al menos una vez a la semana, probar en el laboratorio las soluciones que han sido analizadas y diseñadas. Se espera que la documentación del proyecto se realice de manera concurrente con el desarrollo, asegurando que todos los pasos y decisiones queden adecuadamente registrados.

### 4.2 Validación

Para la validación de todos los elementos que conforman el proyecto se han elaborado un conjunto de procedimientos y pruebas que se describen a continuación:

1. Definición de Criterios de Aceptación: Antes de comenzar cada iteración, se definirán criterios de aceptación específicos que la solución debe cumplir. Estos criterios

funcionarán como un conjunto de condiciones que la implementación debe satisfacer para considerarse válida.

2. Pruebas de Funcionalidad: Una vez desarrollada una función o característica, se llevarán a cabo pruebas en un entorno controlado para asegurar que cumple con su objetivo. En el caso del robot UR3e, esto puede incluir pruebas de movimiento, coordinación, o cualquier otra función específica que se haya implementado.
3. Pruebas de Integración: Despues de validar la funcionalidad individual, se realizarán pruebas de integración para asegurar que las distintas piezas del *software* trabajen armoniosamente entre sí. Esto es esencial para identificar y corregir cualquier incompatibilidad o error que pueda surgir cuando diferentes partes del código interactúan.
4. Pruebas en Entorno Real: Dada la naturaleza práctica del proyecto, es fundamental que las soluciones se prueben en un entorno real con el robot UR3e. Estas pruebas buscan confirmar que el *software* se comporta como se espera en situaciones reales y no solo en un entorno de laboratorio controlado.
5. Revisión de Documentación: Asegurar que toda la documentación esté actualizada y refleje con precisión las soluciones y decisiones tomadas durante el desarrollo. Esto facilitará la comprensión y posible replicación del proyecto en el futuro.
6. Retroalimentación y Ajustes: Tras cada iteración y fase de validación, se recopilarán comentarios y observaciones. Esta retroalimentación será crucial para hacer los ajustes necesarios y mejorar continuamente el *software* desarrollado.
7. Al finalizar cada ciclo de validación, se documentarán los resultados, las observaciones y las correcciones realizadas. Esto garantizará que el proceso sea transparente y permitirá un seguimiento constante del progreso y la calidad del *software*.

### 4.3 Herramientas

- *Git*: Se empleará *Git* como sistema de control de versiones. Se establecerán dos ramas principales: *master* y *develop*. La rama *master* contendrá las versiones estables del proyecto (*releases*), mientras que *develop* albergará el desarrollo en curso. Una vez alcanzado un hito específico y cumplidos los requisitos, se fusionará (*merge*) el contenido de *develop* en *master*. Dado que el proyecto es llevado a cabo por un único estudiante, esta estructura será suficiente para gestionar el desarrollo sin complicaciones adicionales.
- *Trello*: Se utilizará *Trello* como herramienta de gestión de proyectos. Permitirá definir las tareas específicas y planificar cada iteración semanal.

## **5. Planificación temporal**

Considerando que el primer cuatrimestre del curso académico 2023-2024 comenzó el 7 de septiembre y finaliza el 22 de diciembre de 2023, la semana del 22 de enero de 2024 se ha seleccionado como el periodo más cercano para la lectura del proyecto. Este proyecto se puso en marcha efectivamente el 18 de septiembre y está previsto que concluya en enero de 2024. La semana del 15 al 21 de enero se ha elegido para llevar a cabo la revisión final de la documentación y los preparativos finales para la defensa del proyecto.

Dado que el proyecto cuenta con un total de 18 créditos y, según las normativas de la facultad, cada crédito equivale a entre 25 y 30 horas de trabajo, se espera que la duración total del proyecto oscile entre 450 y 540 horas. Desde el 18 de septiembre hasta el 21 de enero hay un total de 126 días disponibles. La estrategia es dedicar aproximadamente 5-6 horas diarias al Trabajo de Fin de Grado (TFG), con sesiones adicionales de 8 horas en los días destinados a las pruebas en el laboratorio. Esta planificación cumple con las recomendaciones establecidas por la facultad.

En esta sección, se presenta la planificación de tareas y el correspondiente diagrama de *Gantt*, junto con códigos para cada tarea y las relaciones de dependencia entre ellas.

### **5.1 Definición de las Tareas**

Las principales categorías de tareas serán: Gestión del proyecto, Análisis, Diseño, Implementación y Evaluación. La tarea de Documentación se realizará de manera concurrente con la de Implementación para documentar los componentes en tiempo real. Habrá una tarea específica dedicada a la revisión y corrección de errores en la última semana previa a la entrega del proyecto.

#### **5.1.1 Gestión del proyecto (GP)**

Esta sección se relaciona con la gestión eficaz del proyecto, con un enfoque en planificación y seguimiento.

#### **Contextualización y alcance (GP1)**

Este paso inicial implica comprender el contexto del proyecto y determinar su alcance y viabilidad. Esto incluye considerar todos los requisitos y posibles riesgos. Tiempo estimado: 21 horas.

#### **Planificación temporal (GP2)**

En esta fase se establece el cronograma del proyecto, asegurando que cada tarea tenga un período de tiempo razonable y justificado. Tiempo estimado: 7 horas.

#### **Presupuesto y sostenibilidad (GP3)**

Una vez definido el contexto, el alcance y la planificación temporal, se abordarán el presupuesto y la sostenibilidad del proyecto. Se evaluará el impacto ambiental, económico y social. Tiempo estimado: 9 horas.

#### **Definición del proyecto final (GP4)**

Aquí se realizarán los ajustes finales al documento del proyecto. Se corregirán inconsistencias y errores según los comentarios y retroalimentación recibida. Tiempo estimado: 18 horas.

#### **Reuniones (GP5)**

Para asegurar un progreso constante y recibir una retroalimentación oportuna, se realizarán reuniones semanales con el tutor del Trabajo de Fin de Grado. En caso de que no sea viable mantener este ritmo, se llevarán a cabo reuniones cada vez que:

- Se implementa una funcionalidad crucial en el proyecto.
- Surja un problema que demande una solución inmediata.

Se garantiza la realización de al menos una reunión por cada *sprint*.

#### **Defensa del proyecto (GP6)**

Al finalizar el proyecto y la documentación, es necesario preparar la presentación del proyecto.

Lo primero es crear un guión para la presentación, que debe incluir una introducción al proyecto, los problemas que resuelve, la metodología empleada y los resultados conseguidos. Al tener listo el guión, se deben hacer ensayos para perfeccionar la presentación. Estos ensayos ayudarán a ajustar el tiempo y mejorar la fluidez y claridad de la presentación.

#### **5.1.2 Análisis (A)**

En este apartado se efectúa la investigación de los requisitos y condiciones necesarias para el proyecto.

##### **Estudio de Requisitos del Robot UR3e y Sensores (A1)**

Evaluar las especificaciones del robot UR3e y los sensores que se usarán, como acelerómetros, sensores de distancia.

##### **Estudio Preliminar de Algoritmos y Herramientas (A2)**

Investigar algoritmos de aprendizaje por refuerzo y herramientas de tiempo real.

##### **Análisis de Riesgos (A3)**

Identificar posibles obstáculos y riesgos asociados al proyecto.

### 5.1.3 Diseño (D)

En esta fase, se crea un plan detallado que especifica la arquitectura y componentes del sistema.

#### **Selección de Algoritmos (D1)**

Escoger el algoritmo de aprendizaje por refuerzo más adecuado.

#### **Diseño de la Arquitectura del Sistema (D2)**

Esquematizar cómo se integrarán los diferentes componentes, como el robot UR3e, sensores y algoritmos.

#### **Diseño de Pruebas (D3)**

Definir pruebas iterativas que permitan ajustar el modelo para mejorar la precisión y estabilidad.

### 5.1.4 Implementación (I)

Se hace el desarrollo de código y funciones del proyecto basado en el diseño.

#### **Integración con el Robot UR3e (I1)**

Desarrollar e implementar el código necesario para que el algoritmo controle el robot UR3e.

#### **Desarrollo del Sistema en Tiempo Real (I2)**

Asegurar que el sistema pueda procesar los datos de los sensores en tiempo real.

#### **Configuración de Parámetros y Tolerancias (I3)**

Se abordarán aspectos como la calibración de parámetros del algoritmo de aprendizaje por refuerzo, la definición de tolerancias y umbrales para medidas correctivas.

#### **Documentación de la Implementación (I4)**

Documentar el código y la configuración del sistema para facilitar futuras modificaciones.

### 5.1.5 Evaluación (E)

Evaluación del *software* para asegurarse de que cumpla con los requisitos e interactúe como se espera con el *hardware*.

#### **Pruebas de Rendimiento (E1)**

Ejecutar las pruebas diseñadas para evaluar el rendimiento del sistema en condiciones variadas.

#### **Análisis de Resultados (E2)**

Evaluar los datos recogidos durante las pruebas para determinar si se cumplen los objetivos del proyecto.

### **Optimización (E3)**

Hacer ajustes en el algoritmo o la configuración del sistema basados en los resultados de las pruebas.

## **5.2 Recursos**

### **5.2.1 Recursos Humanos**

- Director del Proyecto: Responsable de supervisar la investigación, diseño, implementación y evaluación.
- Tutor GEP: Ofrece asesoramiento y dirección en la estructura y ejecución general del proyecto.
- Desarrollador Principal: Encargado de la implementación, pruebas y documentación del proyecto.
- Asesores Técnicos Adicionales: Expertos en inteligencia artificial y robótica que pueden ser consultados en caso de problemas complejos o imprevistos.

### **5.2.2 Recursos Materiales**

- Robot UR3e: *Hardware* central en el que se implementará el algoritmo de equilibrio.
- Sensores: Acelerómetros, sensores de distancia y encoders para el robot.
- Estación de Trabajo: Ordenador con las especificaciones necesarias para llevar a cabo el desarrollo y las pruebas.
- *Software*: Licencias para herramientas y plataformas necesarias, como sistemas de control de versiones(*git*), entornos de desarrollo y *software* de diseño asistido(*Matlab*, *Python*).
- Laboratorio: Espacio físico equipado con todo el *hardware* necesario, donde se realizarán las pruebas en el robot.

## **5.3 Modificación temporal del proyecto**

Debido a inconvenientes imprevistos relacionados con el tiempo, no ha sido posible adherirse a la planificación inicial del proyecto. Estos retrasos han hecho necesaria una extensión del trabajo hasta el turno extraordinario, programado para la semana del 13 al 17 de mayo de 2024. Los motivos principales de este ajuste incluyen la necesidad de dedicar más tiempo del previsto a las pruebas y a la implementación del algoritmo en un entorno real. Asimismo, la coincidencia temporal con otras actividades académicas y compromisos ha contribuido a esta situación. Estos cambios se han adoptado para asegurar la calidad y la exhaustividad del proyecto.

En cuanto al desarrollo del proyecto, la parte correspondiente al *GEP* (Gestión de Proyectos) se ha completado dentro del marco temporal previsto. Sin embargo, se ha experimentado un contratiempo en lo que respecta a la fase de pruebas. Específicamente, esta etapa tuvo que ser pausada durante la semana del 4 de diciembre de 2023, retomándose posteriormente la

semana del 29 de enero de 2024. Este intervalo en la continuidad de las pruebas ha requerido ajustes en la planificación global, para asegurar que el tiempo adicional contribuya efectivamente a la consolidación y al éxito del proyecto.

Tras la revisión, se ha establecido que la duración total del proyecto fluctuará entre 635 y 725 horas. El periodo comprende inicialmente desde el 18 de septiembre de 2023 hasta el 4 de diciembre de 2023, sumando un total de 78 días. Posteriormente, al retomar el proyecto desde el 29 de enero de 2024 hasta el 13 de mayo de 2024, se abarcan 106 días adicionales. En total, la duración acumulada del proyecto será de aproximadamente 184 días.

## 5.4 Estimaciones y diagrama de Gantt.

### 5.4.1 Estimaciones (Periodo ordinario)

En la Tabla 2 se presentan las estimaciones del periodo ordinario:

ID	Nombre de la Tarea	Tiempo de Dedicación (horas)	Dependencias
GP	Gestión del proyecto	90	-
GP1	Contextualización y alcance	21	-
GP2	Planificación temporal	7	GP1
GP3	Presupuesto y sostenibilidad	9	GP2
GP4	Definición del proyecto final	18	GP1, GP2, GP3
GP5	Reuniones	20	-
GP6	Defensa del proyecto	15	Todas las anteriores
A	Análisis	110	-
A1	Estudio de Requisitos del Robot UR3 y Sensores	40	GP1
A2	Estudio Preliminar de Algoritmos y Herramientas	40	GP1
A3	Ánálisis de Riesgos	30	A1, A2
D	Diseño	100	-
D1	Selección de Algoritmos	25	A3
D2	Diseño de la Arquitectura del Sistema	55	D1
D3	Diseño de Pruebas	20	D1
I	Implementación	150	-
I1	Integración con el Robot UR3	60	D3
I2	Desarrollo del Sistema en Tiempo Real	40	I1
I3	Configuración de Parámetros y Tolerancias	30	I1
I4	Documentación de la Implementación	20	I3
E	Evaluación	90	-
E1	Pruebas de Rendimiento	35	I4
E2	Ánalisis de Resultados	20	-
E3	Optimización	35	E2
<b>Total General</b>		<b>540</b>	-

Tabla 2: Tabla de tareas con la duración y las dependencias (periodo ordinario). Fuente: Elaboración propia

### 5.4.2 Estimaciones (Periodo extraordinario)

En la Tabla 3 se presentan las estimaciones del periodo extraordinario:

ID	Nombre de la Tarea	Tiempo de Dedicación (horas)	Dependencias
<b>GP</b>	<b>Gestión del proyecto</b>	<b>90</b>	-
GP1	Contextualización y alcance	21	-
GP2	Planificación temporal	7	GP1
GP3	Presupuesto y sostenibilidad	9	GP2
GP4	Definición del proyecto final	18	GP1, GP2, GP3
GP5	Reuniones	20	-
GP6	Defensa del proyecto	15	Todas las anteriores
<b>A</b>	<b>Análisis</b>	<b>110</b>	-
A1	Estudio de Requisitos del Robot UR3 y Sensores	40	GP1
A2	Estudio Preliminar de Algoritmos y Herramientas	40	GP1
A3	Ánalisis de Riesgos	30	A1, A2
<b>D</b>	<b>Diseño</b>	<b>100</b>	-
D1	Selección de Algoritmos	25	A3
D2	Diseño de la Arquitectura del Sistema	55	D1
D3	Diseño de Pruebas	20	D1
<b>I</b>	<b>Implementación</b>	<b>300</b>	-
I1	Integración con el Robot UR3	120	D3
I2	Desarrollo del Sistema en Tiempo Real	80	I1
I3	Configuración de Parámetros y Tolerancias	60	I1
I4	Documentación de la Implementación	40	I3
<b>E</b>	<b>Evaluación</b>	<b>125</b>	-
E1	Pruebas de Rendimiento	50	I4
E2	Ánalisis de Resultados	30	-
E3	Optimización	45	E2
<b>Total General</b>		<b>725</b>	-

Tabla 3: Tabla de tareas con la duración y las dependencias (periodo extraordinario). Fuente:  
Elaboración propia

### 5.4.3 Diagrama de Gantt (Periodo ordinario)

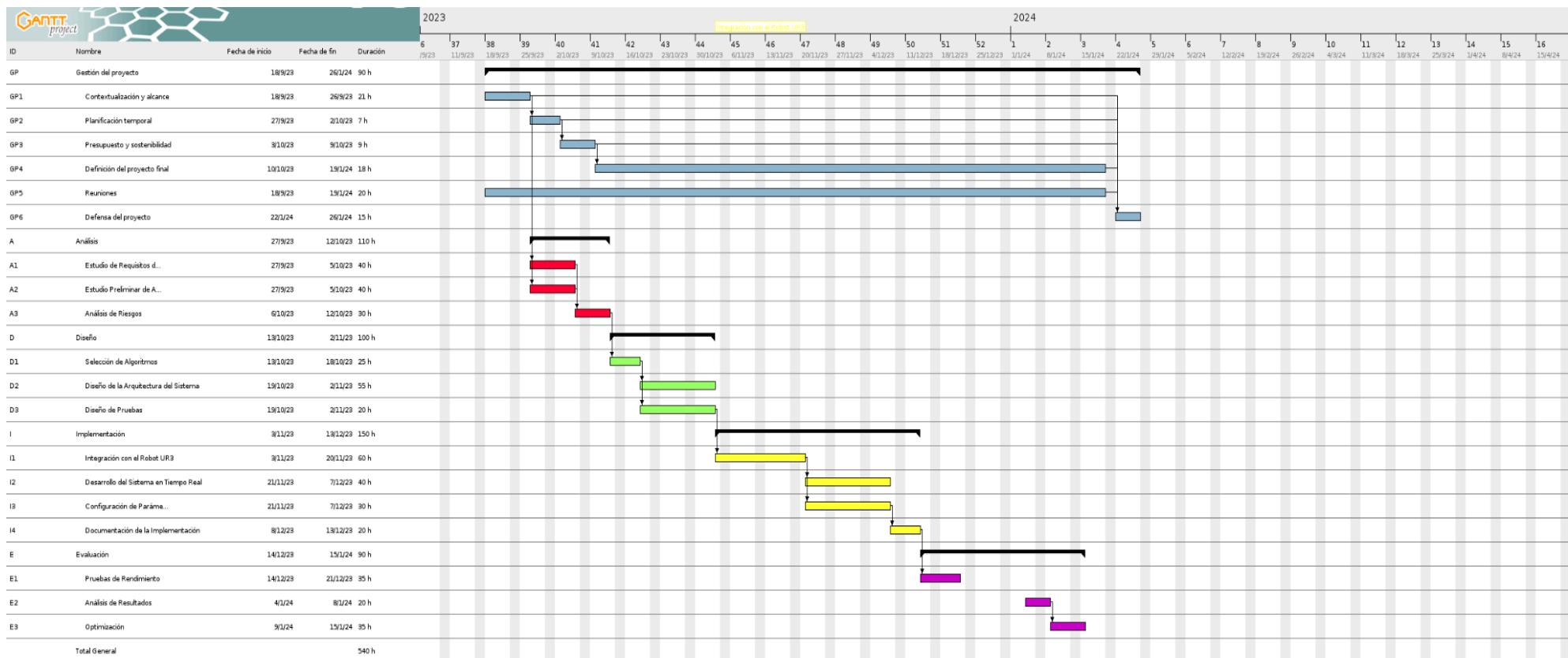


Figura 15: Diagrama de Gantt (periodo ordinario). Fuente: Elaboración propia

#### 5.4.4 Diagrama de Gantt (Periodo extraordinario)

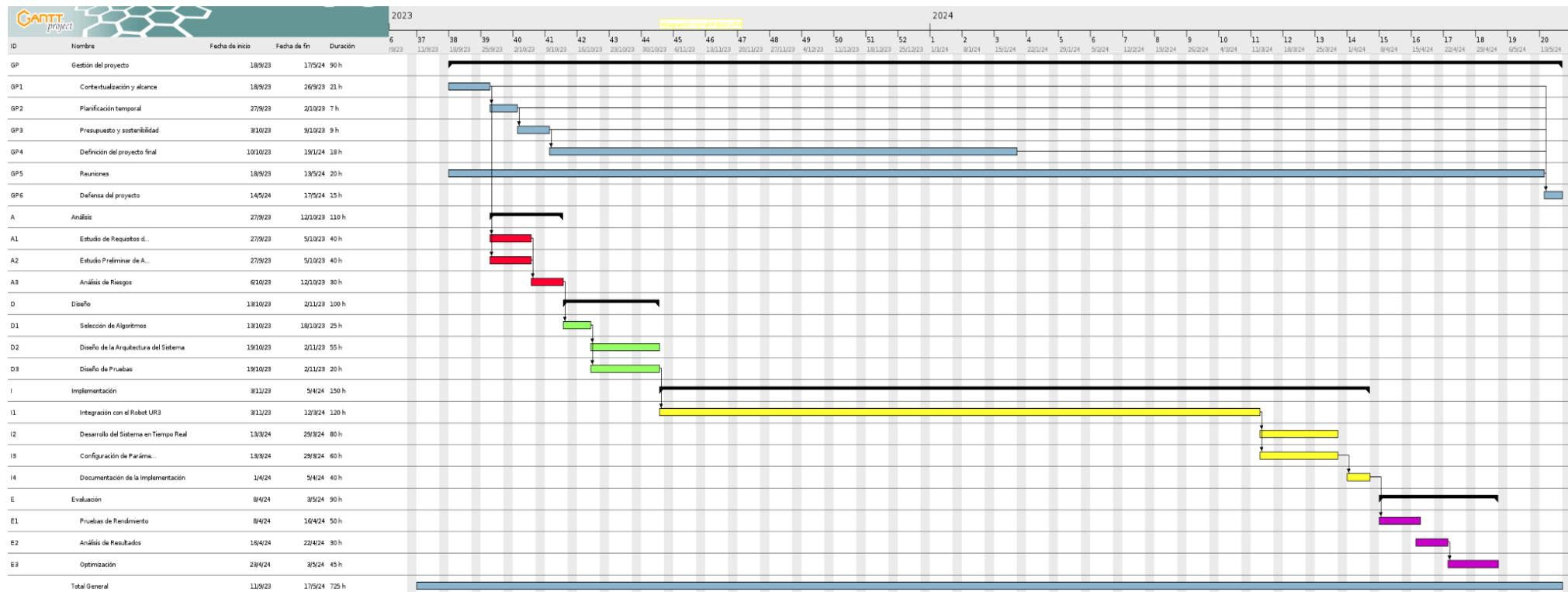


Figura 16: Diagrama de Gantt (periodo extraordinario). Fuente: Elaboración propia

## 5.5 Gestión del riesgo: Planes alternativos y obstáculos

### 5.5.1 Complejidad del Aprendizaje

Tareas Alternativas: En caso de que el aprendizaje por refuerzo no converja, se puede considerar la implementación de algoritmos de aprendizaje supervisado o no supervisado como alternativa.

Impacto en la Duración: Esta alternativa puede requerir tiempo adicional para la implementación y la optimización, extendiendo la duración total del proyecto.

Recursos Adicionales: Podría ser necesario obtener asesoramiento experto o acceder a bases de datos de entrenamiento adicionales.

Pronóstico de Probabilidad: Media

### 5.5.2 Limitaciones de *Hardware*

Tareas Alternativas: Mantener *hardware* de respaldo o considerar la posibilidad de comprar *hardware* adicional.

Impacto en la Duración: La resolución de problemas de *hardware* puede causar retrasos significativos.

Recursos Adicionales: Financiamiento adicional para *hardware* de respaldo o reparaciones.

Pronóstico de Probabilidad: Alta

### 5.5.3 Incompatibilidades de *Software*

Tareas Alternativas: Crear una arquitectura de *software* modular que permita la integración fácil de diferentes sistemas o herramientas.

Impacto en la Duración: La reestructuración del *software* puede añadir tiempo adicional al proyecto.

Recursos Adicionales: Utilizar la orientación de los supervisores del proyecto o profesores con experiencia en integración de *software* para superar desafíos específicos.

Pronóstico de Probabilidad: Baja

#### 5.5.4 Limitaciones de Tiempo

Tareas Alternativas: Reevaluar y, si es necesario, reorganizar el cronograma del proyecto para priorizar tareas críticas. También se podría buscar maneras de trabajar más eficientemente, por ejemplo, mediante la automatización de tareas repetitivas o la utilización de herramientas que puedan acelerar el desarrollo.

Impacto en la Duración: La reorganización del tiempo y la optimización del trabajo ayudarán a mantener el proyecto en el camino correcto. Las sesiones de trabajo adicionales pueden compensar algunos retrasos.

Recursos Adicionales: Hacer un uso más eficaz del tiempo disponible y buscar asesoramiento y orientación de los supervisores del proyecto u otros profesores puede proporcionar soluciones a los desafíos que puedan surgir.

Pronóstico de Probabilidad: Alta

## 6. Presupuesto

En esta sección, se presenta una estimación detallada de los costes asociados con la realización del proyecto. El presupuesto se ha dividido en varias categorías para facilitar su comprensión y seguimiento.

### 6.1 Identificación de los costes

Se describen y se incluyen todos los elementos a considerar en la estimación del presupuesto, que van desde los costes de personal hasta los costes de material, amortizaciones, contingencias e imprevistos.

#### 6.1.1 Costes de personal

Para la realización exitosa de este proyecto, se han identificado cuatro roles clave:

- Gestor del Proyecto: Encargado de la planificación, coordinación y supervisión del proyecto. El gestor es responsable de asignar tareas, gestionar el tiempo y los recursos, y asegurar que el proyecto cumplirá con los objetivos y plazos establecidos.
- Desarrollador: Este rol se encargará de la implementación técnica del proyecto. El Desarrollador será responsable de escribir el código, realizar pruebas y colaborar estrechamente con el Gestor del Proyecto y los Asesores Técnicos para asegurar que la implementación cumpla con los requisitos y expectativas.
- Técnico del Laboratorio: Se enfoca en proporcionar el apoyo técnico necesario para el proyecto. Este rol se centra más en la logística y el funcionamiento diario del laboratorio. Podría incluir tareas como la calibración de equipos, la preparación de materiales y la asistencia en la configuración de experimentos.
- Asesores Técnicos: Ofrecerán asesoramiento experto en áreas técnicas y científicas relevantes para el proyecto. Los Asesores Técnicos serán una fuente invaluable de conocimiento y orientación, especialmente en las fases de diseño e implementación. Su experiencia contribuirá a la toma de decisiones informadas y al éxito del proyecto.

Los costes de personal se han calculado en función de los roles identificados y las tarifas por hora asociadas a cada uno de ellos. A continuación, se presenta en la Tabla 4 el resumen de estos costes:

Rol	Sueldo/h (€)
Gestor del Proyecto	23
Desarrollador	18
Técnico del Laboratorio	15
Asesores Técnicos	22

Tabla 4: Sueldos por hora para cada rol. Fuente: Elaboración propia

Estas tarifas se han basado en las estimaciones salariales genéricas para roles similares en España y en la Universitat Politècnica de Catalunya (UPC). [14]-[18]

La Tabla 5 desglosa los costes de personal por tarea, según se detalla en la Tabla 4:

ID	Nombre de la Tarea	Tiempo (horas)	Dependencias	Roles	Coste (€)
GP	Gestión del proyecto	90	-	-	2.840
GP1	Contextualización y alcance	21	-	Gestor del Proyecto	483
GP2	Planificación temporal	7	GP1	Gestor del Proyecto	161
GP3	Presupuesto y sostenibilidad	9	GP2	Gestor del Proyecto	207
GP4	Definición del proyecto final	18	GP1, GP2, GP3	Gestor del Proyecto	414
GP5	Reuniones	20	-	Gestor del Proyecto, Asesores Técnicos	900
GP6	Defensa del proyecto	15	Todas las anteriores	Gestor del Proyecto, Asesores Técnicos	675
A	Análisis	110	-	-	1.980
A1	Estudio de Requisitos del Robot UR3 y Sensores	40	GP1	Desarrollador	720
A2	Estudio Preliminar de Algoritmos y Herramientas	40	GP1	Desarrollador	600
A3	Ánalisis de Riesgos	30	A1, A2	Desarrollador	660
D	Diseño	100	-	-	1.715
D1	Selección de Algoritmos	25	A3	Desarrollador	450
D2	Diseño de la Arquitectura del Sistema	55	D1	Desarrollador	825
D3	Diseño de Pruebas	20	D1	Desarrollador	440
I	Implementación	300	-	-	8.100
I1	Integración con el Robot UR3	120	D3	Desarrollador, Técnico del Laboratorio	3.960
I2	Desarrollo del Sistema en Tiempo Real	80	I1	Desarrollador	1.440
I3	Configuración de Parámetros y Tolerancias	60	I1	Desarrollador, Técnico del Laboratorio	1.980
I4	Documentación de la Implementación	40	I3	Desarrollador	720
E	Evaluación	125	-	-	5.000
E1	Pruebas de Rendimiento	50	I4	Desarrollador, Asesores Técnicos	2.000
E2	Ánalisis de Resultados	30	-	Desarrollador, Asesores Técnicos	1.200
E3	Optimización	45	E2	Desarrollador, Asesores Técnicos	1.800
	Total General	725	-	-	19.635

Tabla 5: Tabla de tareas con la duración, las dependencias, los roles y los costes de personal por tarea. Fuente: Elaboración propia

### 6.1.2 Costes generalmente calculados

Según la planificación temporal, se trabajará de lunes a viernes en el laboratorio de la universidad cuando sea posible ir y los fines de semana desde casa. Afortunadamente, el laboratorio de la universidad es gratuito para los estudiantes, lo que elimina un coste significativo que de otra manera tendríamos que considerar.

En cuanto a los recursos de *software*, que son presentados en la Tabla 6, se utilizarán varias herramientas esenciales para el desarrollo del proyecto. Primero, se usará *Git* para el control de versiones, aprovechando la versión gratuita de *GitHub*. Además, se utilizará *Matlab*, para el cual se tiene acceso a una licencia de estudiante proporcionada por la universidad. Esto también ayuda a reducir los costes. Las otras herramientas de *software*, como *Python* y *Arduino IDE*, son gratuitas.

Software	Coste (€)
Git	0
Matlab	0
Python	0
Arduino IDE	0
<b>Total</b>	<b>0</b>

Tabla 6: Costes de Software. Fuente: Elaboración propia

Para la ejecución de este proyecto, se utilizarán diversos dispositivos y componentes de *hardware*. Entre ellos se incluyen un portátil para el desarrollo y la gestión del proyecto, un *Arduino Giga Rev1 WiFi* para la implementación de la lógica de control, y un robot *UR3e* para las pruebas y demostraciones. Además, se utilizarán cuatro esferas entre 30-40 mm y una pantalla *Arduino GIGA Display Shield* como elementos adicionales en el sistema.

Para calcular la amortización de cada uno de estos dispositivos y componentes, se ha utilizado una fórmula específica que toma en cuenta la vida útil del dispositivo, el número de días disponibles para el proyecto y las horas diarias de trabajo. La fórmula es la siguiente:

$$\text{Coste por hora} = \frac{\text{Coste del dispositivo}}{\text{Vida útil en años} \times \text{Días disponibles} \times \text{Horas diarias}}$$

$$\text{Coste total de amortización} = \text{Coste por hora} \times \text{Total de horas del proyecto}$$

En este caso, se ha considerado que el proyecto se llevará a cabo en un total de 184 días, con una dedicación de aproximadamente 5-6 horas diarias, sumando un total de entre 635 y 725 horas de trabajo.

La amortización se calcula para reflejar el desgaste o la depreciación del dispositivo durante el tiempo que se utilizará en el proyecto. Este cálculo es especialmente relevante para dispositivos de alto coste como el robot UR3e, donde la amortización se convierte en un factor crítico en el presupuesto total del proyecto.

La Tabla 7 adjunta detalla estos cálculos y ofrece una visión completa de los costes de *hardware* y su amortización:

Hardware	Precio (€)	Unidades	Vida útil (años)	Horas	Amortización (€)
Portátil	800	1	4	725	131
Arduino Giga Rev1 WiFi	69	1	10	500	3
Robot UR3	27.245	1	5	400	1.974
Esferas de 40 mm	15	1	15	400	1
Arduino GIGA Display Shield	60	1	3	400	7
Viga(Tubo)	15	1	5	400	1
Sensor VL53L0X	4	1	4	400	1
<b>Total</b>	<b>28.208</b>	-	-	-	<b>2.119</b>

Tabla 7: Costes de hardware. Fuente: [19]-[22]

Los costes indirectos del proyecto se detallan en la Tabla 8 y abarcan aspectos esenciales como la electricidad, el transporte y el acceso a Internet. Estos costes, aunque no vinculados directamente con las tareas específicas del proyecto, son esenciales para que se pueda realizar:

Coste Indirecto	Coste (€)
Electricidad	17
Transporte (T-Jove)	107
Internet	250
<b>Total</b>	<b>374</b>

Tabla 8: Costes indirectos. Fuente: Elaboración propia

### 6.1.3 Contingencias

En el contexto de este proyecto, que involucra tanto *hardware* como *software* y tiene una duración específica, es crucial tener en cuenta las contingencias. Dado que estamos trabajando con tecnologías que pueden ser complejas y en un entorno de laboratorio, los riesgos de enfrentar desafíos imprevistos son altos. Podrían surgir problemas como fallos de *hardware*, necesidad de licencias de *software* adicionales, o incluso la necesidad de más tiempo para la resolución de problemas técnicos.

Por lo tanto, se ha decidido establecer una contingencia del 20% sobre el coste total del proyecto. Este porcentaje se ha elegido considerando la naturaleza experimental del proyecto y la posibilidad de que se requieran recursos adicionales, ya sea tiempo o material, para abordar problemas no previstos inicialmente. La Tabla 9 muestra estas contingencias:

Tipo	Coste (€)	Contingencia (€)
Software	0	0
Hardware	2.119	424
Genérico	374	75
Personal	14.185	3927
<b>Total</b>	<b>22.128</b>	<b>4.426</b>

Tabla 9: Tabla contingencia del 20 %. Fuente: Elaboración propia

### 6.1.4 Imprevistos

En el desarrollo de este proyecto, se han identificado varios imprevistos potenciales que podrían afectar tanto el tiempo como el coste total. Estos van desde fallos de *hardware* hasta limitaciones de tiempo. Para cuantificar estos riesgos, se ha elaborado la Tabla 10 que detalla el coste adicional estimado, la probabilidad de que ocurra cada imprevisto y el coste esperado.

<b>Imprevisto</b>	<b>Coste Adicional (€)</b>	<b>Probabilidad (%)</b>	<b>Coste Esperado (€)</b>
Aumento tiempo de desarrollo	560	20	112
Fallo del Portátil	800	15	120
Fallo del Robot UR3	27.245	5	1.362
Fallo en el Giga Display Shield	60	20	12
Fallo en sensores	250	10	25
Fallo Viga(Tubo)	15	40	6
Incompatibilidad de Software	300 (Tiempo extra)	10	30
Sobrecarga del Sistema	200 (Optimización)	5	10
Limitaciones de Tiempo	400 (Reorganización)	15	60
<b>Total</b>	<b>29.830</b>	-	<b>1.737</b>

*Tabla 10: Costes de imprevistos. Fuente: Elaboración propia*

### 6.1.5 Coste final

En la Tabla 11 se presenta el presupuesto final:

<b>Tipo</b>	<b>Coste (€)</b>
Personal	19.635
Software	0
Hardware	2.119
Genérico	374
Contingencias	4.425
Imprevistos	1.737
<b>Coste Total</b>	<b>28.290</b>

*Tabla 11: Tabla del Coste total. Fuente: Elaboración propia*

Si al final del proyecto no se han materializado los riesgos o eventualidades que justificaron la creación de partidas para contingencias e imprevistos, existen varias opciones para el uso de estos recursos. Una posibilidad es reasignarlos a otras áreas del proyecto que podrían beneficiarse de financiamiento adicional. Otra opción sería emplear el dinero en una evaluación más detallada de riesgos futuros y sus posibles mitigaciones. Finalmente, los fondos también podrían destinarse a optimizaciones o mejoras que, aunque no sean estrictamente necesarias, añadirían valor al proyecto.

## 6.2 Control de gestión

Para garantizar que el proyecto se mantenga dentro del presupuesto y el cronograma establecidos, se han implementado varios mecanismos de control de gestión. Estos mecanismos se describen a continuación:

- Se definen indicadores numéricos clave para medir las desviaciones en el presupuesto y el tiempo. Estos incluyen:
  - Índice de Desempeño del Coste (IDC): Calculado como el cociente entre el Valor Ganado (VG) y el Coste Real (CR).
  - Índice de Desempeño del Tiempo (IDT): Calculado como el cociente entre el Valor Ganado (VG) y el Valor Planeado (VP).

Si alguno de estos indicadores cae por debajo del umbral del 90%, se tomarán medidas correctivas.

- Se llevarán a cabo revisiones mensuales del presupuesto para identificar cualquier desviación y tomar medidas correctivas.
- Cualquier cambio en el alcance del proyecto que pueda afectar el presupuesto debe ser aprobado por el Gestor del Proyecto.
- Como se mencionó en la sección de contingencias, se ha establecido una reserva del 20% para abordar cualquier desviación imprevista en el presupuesto.
- Se mantendrá un registro de todos los imprevistos que ocurran durante el proyecto, junto con su impacto en el presupuesto y el tiempo. Este registro ayudará en la planificación de proyectos futuros.

Se utilizará Excel para monitorizar y controlar el progreso del proyecto, como se puede ver en la Tabla 12. Las unidades de Valor Planeado (VP), Coste Real (CR) y Valor Ganado (VG) se expresan en horas y las celdas en tonalidad roja destacan las áreas que requieren intervención, mientras que las etiquetas "#VALUE!" y "Por determinar" señalan datos pendientes de entrada:

Tarea	Fecha	Valor Planeado (VP)	Coste Real (CR)	Valor Ganado (VG)	Índice de Desempeño del Coste (IDC)	Índice de Desempeño del Tiempo (IDT)
GP1	26/09/2023	21	20	21	1,05	1
GP2	02/10/2023	7	10	7	0,7	1
GP3	09/10/2023	9	10,00	9	0,9	1
GP4	19/01/2024	18	18	18	1	1
GP5	13/05/2024	20	20	20	1	1
GP6	17/05/2024	15	Por determinar	Por determinar	#VALUE!	#VALUE!
A1	05/10/2023	40	40	40	1	1
A2	05/10/2023	40	35	35	1	0,875
A3	12/10/2023	30	25	22	0,88	0,7333333333
D1	18/10/2023	25	26	26	1	1,04
D2	02/11/2023	55	50	55	1,1	1
D3	02/11/2023	20	20	20	1	1
I1	12/03/2024	120	150	120	0,8	1
I2	29/03/2024	80	80	80	1	1
I3	29/03/2024	60	80	60	0,75	1
I4	05/04/2024	40	40	40	1	1
E1	16/04/2024	50	55	50	0,9090909091	1
E2	22/04/2024	30	30	30	1	1
E3	03/05/2024	45	45	46	1,0222222222	1,0222222222

Tabla 12: Tabla de Seguimiento y Control del Proyecto. Fuente: Elaboración propia

## **7. Informe de Sostenibilidad**

### **7.1 Autoevaluación**

Después de realizar la encuesta, mi autoevaluación se basa en los varios aspectos relacionados con la sostenibilidad y mi campo de estudio. En términos de conocimiento teórico, me siento bastante seguro. He adquirido una base sólida a través de mis estudios y me siento preparado para enfrentar desafíos en este ámbito. Sin embargo, cuando se trata de la aplicación práctica de este conocimiento, reconozco que hay un vacío. No he tenido la oportunidad de participar en proyectos de gran envergadura que me permitan aplicar lo que he aprendido en un contexto real y significativo. La encuesta me ha ayudado en este sentido, señalando claramente que necesito centrar mis esfuerzos en adquirir experiencia práctica para complementar mi formación teórica.

En cuanto a la concienciación sobre la sostenibilidad, considero que estoy alineado con lo que se espera de un estudiante en mi campo. Este tema ha sido una constante en mi educación desde etapas tempranas, y siento que tengo un buen nivel de entendimiento y compromiso con los temas de sostenibilidad.

### **7.2 Dimensión Económica**

Se ha realizado una estimación detallada de los costes asociados con la realización del proyecto, desglosando el presupuesto en varias categorías como costes de personal, material, amortizaciones, contingencias e imprevistos. En lo que respecta a los costes de personal, estos se han calculado en función de los roles identificados y las tarifas por hora asociadas a cada uno de ellos.

En términos económicos, mi proyecto ofrece una solución más optimizada en cuanto a eficiencia y robustez. Aunque los costes iniciales pueden ser altos debido a la especialización del personal y los recursos técnicos necesarios, la eficiencia del sistema propuesto podría resultar en menores costes operativos a largo plazo.

### **7.3 Dimensión Ambiental**

En cuanto al proyecto puesto en producción, se ha realizado una estimación del impacto ambiental que tendrá la realización del mismo. Se han tomado medidas para minimizar el impacto ambiental, como la reutilización de recursos y componentes. Por ejemplo, el robot UR3e y los sensores utilizados son componentes que se pueden reutilizar en futuros proyectos, lo que reduce la necesidad de nuevos materiales y disminuye el desperdicio.

En lo que respecta a la vida útil del proyecto y cómo aborda el problema en cuestión, el objetivo es enseñar a un robot a mantener una esfera en equilibrio sobre una superficie plana. Aunque este es un desafío técnico complejo, se ha optado por algoritmos que son eficientes

en términos de uso de recursos computacionales. Esto no solo tiene implicaciones económicas sino también ambientales, ya que un uso más eficiente de los recursos computacionales se traduce en un menor consumo de energía a lo largo del tiempo.

En comparación con soluciones existentes, este proyecto tiene el potencial de ser más eficiente y, por lo tanto, podría tener un menor impacto ambiental en términos de consumo de energía y generación de residuos electrónicos a largo plazo.

## 7.4 Dimensión Social

Creo que la realización de este proyecto me aportará una serie de conocimientos teóricos y prácticos relevantes a nivel personal. No solo me permitirá aplicar y profundizar mis conocimientos técnicos, sino que también me brindará la oportunidad de trabajar en un problema que tiene implicaciones sociales reales. Esto añade una capa de responsabilidad y significado a mi trabajo, lo que es altamente gratificante.

En cuanto a la vida útil del proyecto y cómo se aborda el problema actualmente, el estado del arte en mantener una esfera en equilibrio sobre una viga(tubo) generalmente implica sistemas altamente especializados que no son accesibles para el público en general. Mi proyecto, al utilizar algoritmos más eficientes y componentes reutilizables, tiene el potencial de hacer que esta tecnología sea más accesible. Esto podría tener aplicaciones en campos como la rehabilitación física o la educación, mejorando así la calidad de vida de las personas.

Finalmente, en lo que respecta a la necesidad real del proyecto, aunque puede parecer un desafío técnico específico, las habilidades y tecnologías desarrolladas tienen aplicaciones prácticas que podrían beneficiar a la sociedad en diversos campos, desde la medicina hasta la ingeniería.

## **8. Sinergia entre Conocimientos y Cumplimiento Normativo**

### **8.1 Integración de conocimientos**

Este proyecto de final de grado se destaca por un enfoque integrador y la aplicación de conocimientos interdisciplinarios. Al abordar el desafío de diseñar e implementar un sistema de ball balancing para el robot UR3e, se han empleado técnicas avanzadas de aprendizaje por refuerzo, específicamente el algoritmo Soft-Actor Critic (*SAC*), que no solo reflejan una profunda comprensión de los principios de la inteligencia artificial y la robótica, sino también una aplicación de estos en un contexto práctico y desafiante.

La integración de conocimientos de robótica aprendidos durante la asignatura de Robótica (ROB) se evidencia en la manipulación del robot UR3e. La aplicación práctica de este conocimiento ha permitido el desarrollo de un sistema que es técnicamente viable.

La implementación del algoritmo Soft-Actor Critic (*SAC*), una técnica avanzada de aprendizaje por refuerzo, es testimonio de la habilidad para adaptar y aplicar conceptos complejos de inteligencia artificial en soluciones concretas y efectivas. Este enfoque evidencia no solo una profunda comprensión teórica de técnicas no exploradas en el grado, sino también la habilidad para integrar conocimientos previos en la asimilación de nuevos conceptos y su aplicación práctica en la resolución de problemas concretos.

El proyecto también se beneficia significativamente de conocimientos adquiridos en la asignatura Sistemas de Tiempo Real (STR), una necesidad crítica dado el requisito de respuestas con restricciones de tiempo estrictas y precisas en el sistema de equilibrio.

Adicionalmente, se ha hecho uso de conocimientos especializados adquiridos en la asignatura de Probabilidad y Estadística (PE) y en la de Inteligencia Artificial (IA) para comprender y aplicar los algoritmos de aprendizaje por refuerzo, lo que refleja una capacidad excepcional para integrar conceptos matemáticos complejos en la solución de problemas de ingeniería. Esta integración es evidencia de haber adquirido una base académica sólida en esas asignaturas y también de una habilidad para aplicar estos conocimientos de manera efectiva en el desarrollo de soluciones técnicas.

En conclusión, este proyecto es un claro ejemplo de cómo se han integrado conocimientos de diversas disciplinas para la creación de soluciones efectivas. La implementación exitosa del sistema de equilibrio de bola para el robot UR3e no solo aborda un desafío técnico complejo, sino que también usa la aplicación de técnicas avanzadas de inteligencia artificial en la robótica.

## 8.2 Identificación de leyes y regulaciones

En el desarrollo de este proyecto, se ha prestado especial atención al cumplimiento de las leyes y regulaciones aplicables, garantizando así la integridad y seguridad del sistema de *Ball Balancing* diseñado para el robot UR3e. Dada la naturaleza interdisciplinaria del trabajo, que abarca áreas como la robótica, la inteligencia artificial y el procesamiento en tiempo real, se han identificado y considerado las siguientes normativas tanto a nivel de la Unión Europea (UE) como de España:

1. Seguridad del Producto y Estándares Técnicos: El diseño y funcionamiento del sistema se han desarrollado en conformidad con la Normativa de Seguridad de Productos de la UE (2001/95/CE) [23], asegurando que todos los componentes y funcionalidades cumplan con los requisitos generales de seguridad. Además, se han seguido las directrices específicas para robótica establecidas por las normas ISO 10218-1 [24] y ISO 10218-2 [25], las cuales dictan los principios de seguridad para robots industriales, incluyendo los aspectos de diseño y operaciones para robots colaborativos específicamente del robot UR3e de *Universal Robots*. Estas medidas garantizan que el robot opera dentro de parámetros seguros, minimizando cualquier riesgo para los usuarios y el entorno.
2. Protección de Datos: Aunque el proyecto no implica directamente la recopilación de datos personales, se ha establecido un marco de referencia basado en los principios del Reglamento General de Protección de Datos (RGPD) de la UE [26]. Este enfoque proactivo asegura que cualquier futura expansión o modificación del sistema que involucre datos personales se realizará respetando la privacidad y seguridad de la información.
3. Ética en IA: Se han incorporado los principios éticos de la UE para la inteligencia artificial, enfocándose en la transparencia, la equidad y la rendición de cuentas en el desarrollo del algoritmo de aprendizaje por refuerzo.

En caso de que alguna parte del proyecto no cumpla plenamente con una normativa específica, se debe principalmente a limitaciones técnicas o al estado actual de la investigación en áreas emergentes de la IA y la robótica.

## 9. Descripción detallada del sistema *Ball and Beam*

El sistema *Ball and Beam* es un problema clásico de control en ingeniería. Consiste en una barra, en nuestro caso, representada por un cilindro de un metro de longitud, que puede inclinarse alrededor de su centro de masas, mientras que una esfera se mueve a lo largo de la longitud del tubo. El objetivo es controlar la posición de la esfera en la barra inclinándola. Este sistema es un modelo interesante debido a su naturaleza inestable y no lineal, lo que presenta desafíos significativos para el diseño de un sistema de control.

### 9.1 Principios físicos y matemáticos del sistema

Los principios físicos en los cuales se basa el sistema están principalmente relacionados con la mecánica clásica. En la Figura 17 se presenta el diagrama de fuerzas del sistema en detalle. A continuación, se explican los componentes de ese diagrama y algunos principios físicos:

- Gravedad: La fuerza de gravedad actúa sobre la esfera, tirando de ella hacia abajo. Esta es la fuerza que actúa sobre la esfera debido a la masa ( $m$ ) y la aceleración debida a la gravedad ( $g$ ). La gravedad se descompone en dos componentes:  $mgsin(\alpha)$ , que impulsa la esfera cuesta abajo a lo largo del tubo, y  $mgcos(\alpha)$ , perpendicular al cilindro, que determina la magnitud de la fuerza normal.
- Fuerzas de contacto: Cuando la bola está en contacto con la barra, hay una fuerza normal (perpendicular a la superficie de contacto) que afecta el movimiento de la esfera. La fricción entre la esfera y la barra se consideran despreciables.
- Fuerza Traslacional: Se refiere al movimiento de la esfera a lo largo del tubo. La única fuerza significativa en la dirección del movimiento de la esfera es  $mgsin(\alpha)$ , que genera la aceleración traslacional de la esfera.
- Fuerza Rotacional: Relacionada con el cambio en la inclinación del tubo, es una fuerza que no actúa directamente sobre la esfera, sino sobre la barra. El robot ejerce un torque que cambia el ángulo ( $\alpha$ ) del cilindro, y este cambio afecta a la fuerza traslacional.
- Inercia: La masa de la esfera y su inercia son factores críticos que determinan cómo responde la esfera a las fuerzas aplicadas. La inercia de la esfera resiste los cambios en su estado de movimiento, lo que afecta a la dinámica del sistema.
- Momento de fuerza (torque): La actuación del robot que ajusta la inclinación del tubo. Altera el ángulo de la barra ( $\alpha$ ) y, por lo tanto, la componente de la gravedad que impulsa la esfera.
- Segunda Ley de Newton: Esta ley postula que la aceleración de un objeto es proporcional a la fuerza neta actuando sobre él. En el contexto del sistema Ball and Beam, esta ley describe la relación entre la fuerza neta y la aceleración de la esfera.

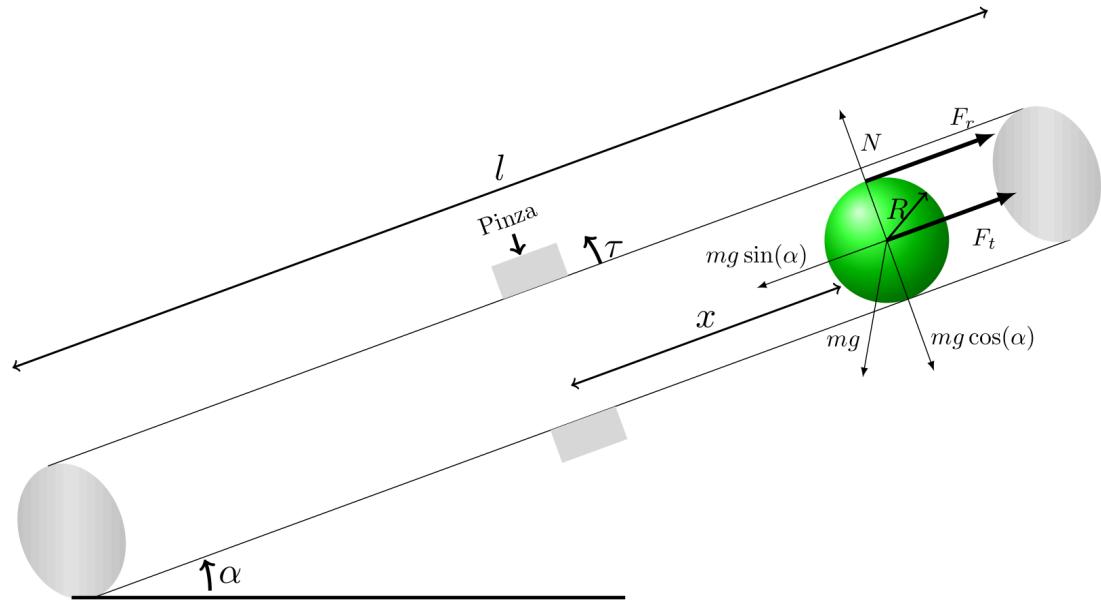
Beam, esta relación se aplicará tanto a las fuerzas translacionales como a las rotacionales para desarrollar un conjunto de ecuaciones que describan la dinámica del sistema.

- Energía Cinética(**Ec**): Para el sistema Ball and Beam, la energía cinética total se compone de la energía cinética translacional debida al movimiento de la esfera a lo largo del tubo y la energía cinética rotacional debida a cualquier rotación que pueda tener la esfera.
- Energía Potencial(**U**): La energía potencial del sistema es influenciada por la altura de la esfera en el tubo, que varía con el ángulo  $\alpha$ .
- Dinámica Rotacional: Mientras el sistema Ball and Beam se centra en el control de la posición de la esfera a lo largo del cilindro, la dinámica rotacional de la propia barra es igualmente importante. El torque aplicado para inclinar el cilindro no solo afecta la posición de la esfera, sino que también implica una consideración de la inercia angular del cilindro y cómo esta afecta la rapidez y la eficacia con la que se puede cambiar su inclinación.
- Modelo de Lagrange: El modelo de Lagrange se aplica integrando la energía cinética y la energía potencial para establecer las ecuaciones del movimiento. El Lagrangiano se define como  $L = Ec - U$ . A partir de este, se aplican las ecuaciones de Euler-Lagrange para derivar las ecuaciones diferenciales que describen la dinámica del sistema:

$$\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} = F_{q_i}$$

Donde:

- $q_i$  son las coordenadas generalizadas.
- $\dot{q}_i$  son las velocidades generalizadas.
- $F_{q_i}$  son las fuerzas externas aplicadas a cada coordenada.



Label	Descripción
$l$	Longitud total del cilindro.
$x$	Posición de la esfera dentro del cilindro.(Dada por el sensor de distancia VL53L0X)
$R$	Radio de la esfera.
$mg$	Fuerza de gravedad que actúa sobre la esfera.
$mg \cos(\alpha)$	Componente de la fuerza de gravedad paralela al plano inclinado.
$mg \sin(\alpha)$	Componente de la fuerza de gravedad perpendicular al plano inclinado.
$N$	Fuerza normal ejercida por el plano inclinado sobre la esfera.
$F_t$	Fuerza translacional actuando sobre la esfera.
$F_r$	Fuerza rotacional actuando en la superficie de la esfera.
$\tau$	Ángulo de torsión de las pinzas.
$\alpha$	Ángulo de inclinación del cilindro con respecto a la horizontal.
Pinza	Herramienta que esta conectada al actuador 6 del robot UR3e.

Figura 17: Diagrama de Fuerzas en el sistema Ball and Beam. Fuente: Elaboración propia

## 9.2 Modelado matemático del *Ball and Beam*

El comportamiento del sistema *Ball and Beam* se puede describir utilizando ecuaciones matemáticas que representan las leyes de la física mencionadas anteriormente. La dinámica del sistema se puede modelar usando un modelo lineal o un modelado no lineal para enfoques que tiene en cuenta la totalidad de la dinámica del sistema.

Dado que se quiere tener en cuenta gran parte de la complejidad del sistema, se ha optado por un modelado no lineal para que se pueda representar mejor el comportamiento real del sistema. Se utilizarán dos enfoques distintos para modelar el comportamiento dinámico del sistema *Ball and Beam*: uno basado en las ecuaciones de *Newton/Euler* y otro en el enfoque de *Lagrange*. Este enfoque dual aborda la complejidad del sistema de manera más completa, al tiempo que se mantiene la claridad conceptual en el análisis.

### 9.2.1 Modelo de Newton

- **Modelado Traslacional de la Esfera:**

Para el movimiento lineal de la esfera sobre el tubo, aplicamos la Segunda Ley de Newton:

$$\sum_{i=1}^n F_i = ma \Leftrightarrow \sum_{i=1}^n F_i = mg \sin(\alpha) \quad (9.1)$$

La fuerza  $F$  en este caso es la componente de la gravedad que impulsa la esfera a lo largo del tubo,  $mgsin(\alpha)$ , y  $a$ , es la aceleración de la esfera a lo largo del tubo. La ecuación diferencial resultante que describe este movimiento es:

$$m \frac{d^2x}{dt^2} = mg \sin(\alpha) \Rightarrow \frac{d^2x}{dt^2} = g \sin(\alpha) \quad (9.2)$$

Donde  $x$  es la posición de la esfera a lo largo del tubo y  $t$  es el tiempo.

Esta ecuación (9.2) indica que la aceleración traslacional de la esfera a lo largo del tubo es proporcional a la componente de la gravedad que actúa a lo largo de la inclinación del tubo, y está en función del ángulo de inclinación del tubo.

- **Modelado Rotacional de la Esfera:**

La esfera rueda sobre el cilindro, lo que introduce un movimiento rotacional alrededor de su propio centro de masas. Al desconsiderar el deslizamiento, la velocidad lineal de la esfera ( $v$ ) a lo largo del tubo está relacionada con su velocidad angular ( $\omega$ ) a través de la relación  $v=\omega R$ , donde  $R$  es el radio de la esfera. Aplicando la segunda ley de Newton al movimiento rotacional de la esfera obtenemos:

$$\tau_r = I \frac{d\omega}{dt} \quad (9.3)$$

Donde:

- $I$  es el momento de inercia de la esfera, que para una esfera sólida es  $\frac{2}{5}mR^2$ .
- $\frac{d\omega}{dt}$  es la aceleración angular de la esfera.
- $\tau_r$  es el torque resultante en la esfera debido a la fuerza rotacional  $F_r$ , que en el caso de deslizamiento y fricción despreciable es igual a  $F_r R$ .

Dado que la esfera rueda sin deslizar, la aceleración translacional y la aceleración angular están relacionadas por la velocidad angular:  $\frac{d\omega}{dt} = d(\frac{v}{R}) \Rightarrow \frac{d\omega}{dt} = \frac{d^2x}{dt^2 R}$ . Por lo tanto, el torque generado por la fuerza rotacional es:

$$\tau_r = \frac{2}{5}mR^2 \frac{d^2x}{dt^2 R} = \frac{2}{5}mR \frac{d^2x}{dt^2} \quad (9.4)$$

Entonces de (9.4):

$$F_r = \frac{2}{5}m \frac{d^2x}{dt^2} \quad (9.5)$$

#### • Suma de Fuerzas Actuantes:

Para obtener el modelado completo, debemos considerar tanto la fuerza translacional debida a la gravedad como la fuerza rotacional. La suma de estas fuerzas es la fuerza neta que actúa sobre la esfera:

$$\begin{aligned} F_{\text{neta}} &= F_t + F_r \Rightarrow F_{\text{neta}} = m \frac{d^2x}{dt^2} + \frac{2}{5}m \frac{d^2x}{dt^2} \Rightarrow \\ &\Rightarrow F_{\text{neta}} = \frac{7}{5}m \frac{d^2x}{dt^2} \end{aligned} \quad (9.6)$$

Por la segunda ley de *Newton* sabemos que la fuerza neta es igual a la masa de la esfera multiplicada por su aceleración lineal:

$$\frac{7}{5}m \frac{d^2x}{dt^2} = mg \sin(\alpha) \Rightarrow \frac{d^2x}{dt^2} = \frac{5}{7}g \sin(\alpha) \quad (9.7)$$

Una vez desarrollado el modelo de la ecuación (9.7) para el sistema Ball and Beam, es importante destacar que este modelo es una aproximación simplificada. A pesar de su utilidad en describir las interacciones básicas de fuerzas actuantes y movimientos resultantes, no tiene en cuenta variaciones en las masas ni la distribución del peso de la esfera. Esta simplificación puede no ser representativa de todos los escenarios operativos, especialmente en situaciones donde las variaciones de masa o las características físicas del material influyen en la dinámica del sistema. Por ello, se considerará también el enfoque de Lagrange, que proporciona un modelado más exhaustivo al incluir estos factores y ofrecer una descripción más detallada de la dinámica del sistema mediante el uso de energías cinética y potencial.

Además, en el contexto del aprendizaje por refuerzo con técnicas como el SAC, se destaca la ventaja de ser un método model-free. Esto significa que SAC no requiere un conocimiento previo detallado del modelo matemático del sistema para desarrollar y optimizar la política de control. En contraposición a métodos que dependen intensamente de modelos precisos y detallados para funcionar efectivamente, SAC puede generalizar acciones de control en una variedad de escenarios sin la necesidad de ajustar o recalibrar los parámetros del modelo a cada nueva situación. Esta característica lo hace particularmente valioso en entornos donde la capacidad de adaptación y la flexibilidad son cruciales, proporcionando una ventaja significativa frente a otras técnicas que necesitan modelos detallados para operar eficientemente.

### 9.2.2 Modelo de Lagrange

Primero de todo definimos las coordenadas que describen la configuración del sistema y algunas variables:

- $q_1 = x$ , la posición de la esfera a lo largo del cilindro.
- $q_2 = \alpha$ , el ángulo de inclinación del cilindro.
- $F_{qi}$  es  $\tau$ , el torque aplicado por el robot UR3e para  $q2$  y 0 para  $q1$ .
- $I_c$  es el momento de inercia del cilindro.
- $I_e$  es el momento de inercia de la esfera, que para una esfera sólida es  $\frac{2}{5}mR^2$ .
- $m$ , es la masa de la esfera.
- $R$ , es el radio de la esfera.
- $x$ , es la posición de la esfera a lo largo del cilindro.

- **Energía Cinética ( $E_c$ ):**

La energía cinética total del sistema es la suma de la energía cinética del cilindro y la energía cinética de la esfera.

- Energía cinética del cilindro, debido a su rotación:

$$E_{c_{\text{cilindro}}} = \frac{1}{2}I_c \left( \frac{d\alpha}{dt} \right)^2 \quad (9.8)$$

- La energía cinética traslacional de la esfera está compuesta por el movimiento radial debido a la rotación del cilindro y el movimiento propio de la esfera a lo largo del cilindro:

$$E_{c_{\text{esfera\_trans}}} = \frac{1}{2}m \left( \frac{dx}{dt} \right)^2 \quad (9.9)$$

$$E_{c_{\text{esfera\_radial}}} = \frac{1}{2}mx^2 \left( \frac{d\alpha}{dt} \right)^2 \quad (9.10)$$

- Energía cinética rotacional de la esfera alrededor de su centro de masa (sin deslizamiento,  $v=\omega R$ ):

$$\begin{aligned}
 E_{c_{\text{esfera\_rot}}} &= \frac{1}{2} I_e \left( \frac{dx}{dt} \right)^2 \left( \frac{1}{R} \right)^2 \Rightarrow \\
 \Rightarrow E_{c_{\text{esfera\_rot}}} &= \frac{1}{2} \left( \frac{2}{5} m R^2 \right) \left( \frac{dx}{dt} \right)^2 \left( \frac{1}{R} \right)^2 \Rightarrow \\
 \Rightarrow E_{c_{\text{esfera\_rot}}} &= \frac{1}{5} m \left( \frac{dx}{dt} \right)^2
 \end{aligned} \tag{9.11}$$

- Energía cinética total de la esfera:

$$\begin{aligned}
 E_{c_{\text{esfera}}} &= E_{c_{\text{esfera\_trans}}} + E_{c_{\text{esfera\_radial}}} + E_{c_{\text{esfera\_rot}}} \Rightarrow \\
 \Rightarrow E_{c_{\text{esfera}}} &= \frac{1}{2} m \left( \frac{dx}{dt} \right)^2 + \frac{1}{2} m x^2 \left( \frac{d\alpha}{dt} \right)^2 + \frac{1}{5} m \left( \frac{dx}{dt} \right)^2 \Rightarrow \\
 \Rightarrow E_{c_{\text{esfera}}} &= \frac{7}{10} m \left( \frac{dx}{dt} \right)^2 + \frac{1}{2} m x^2 \left( \frac{d\alpha}{dt} \right)^2
 \end{aligned} \tag{9.12}$$

- Energía cinética total:

$$\begin{aligned}
 E_c &= E_{c_{\text{cilindro}}} + E_{c_{\text{esfera}}} \Rightarrow \\
 \Rightarrow E_c &= \frac{1}{2} I_c \left( \frac{d\alpha}{dt} \right)^2 + \frac{7}{10} m \left( \frac{dx}{dt} \right)^2 + \frac{1}{2} m x^2 \left( \frac{d\alpha}{dt} \right)^2 \Rightarrow \\
 \Rightarrow E_c &= \frac{1}{2} \left( \frac{d\alpha}{dt} \right)^2 [I_c + m x^2] + \frac{7}{10} m \left( \frac{dx}{dt} \right)^2
 \end{aligned} \tag{9.13}$$

### ● Energía Potencial (U):

La energía potencial del sistema está relacionada con la altura de la esfera con respecto a un punto de referencia. Si el cilindro se inclina, la esfera tendrá una energía potencial que depende de  $\alpha$ :

$$U = mgx \sin(\alpha) \tag{9.14}$$

### ● Lagrangiano (L):

$$\begin{aligned}
 L &= E_c - U \Rightarrow \\
 \Rightarrow L &= \frac{1}{2} \left( \frac{d\alpha}{dt} \right)^2 [I_c + m x^2] + \frac{7}{10} m \left( \frac{dx}{dt} \right)^2 - mgx \sin(\alpha)
 \end{aligned} \tag{9.15}$$

### ● Ecuaciones de Movimiento de Lagrange:

Para describir completamente la dinámica del sistema hay que encontrar la evolución temporal de  $\alpha$  y  $x$ . Derivamos las ecuaciones de movimiento aplicando la fórmula de Lagrange, con ninguna fuerza externa para la posición de la esfera y el torque para el ángulo de inclinación del cilindro:

$$\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{x}} \right) - \frac{\partial L}{\partial x} = 0$$

(9.16)

$$\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{\alpha}} \right) - \frac{\partial L}{\partial \alpha} = \tau$$

Derivación explícita de las ecuaciones:

- Para  $\alpha$ :

$$\begin{aligned} \frac{\partial L}{\partial \alpha} &= -mgx \cos(\alpha) \\ \frac{\partial L}{\partial \dot{\alpha}} &= (I_c + mx^2) \frac{1}{2} 2 \frac{d\alpha}{dt} = (I_c + mx^2) \frac{d\alpha}{dt} \\ \frac{d}{dt} \left( (I_c + mx^2) \frac{d\alpha}{dt} \right) &= (I_c + mx^2) \frac{d^2\alpha}{dt^2} + 2mx \frac{dx}{dt} \frac{d\alpha}{dt} \end{aligned}$$

(9.17)

- Para  $x$ :

$$\begin{aligned} \frac{\partial L}{\partial x} &= mx \left( \frac{d\alpha}{dt} \right)^2 - mg \sin(\alpha) \\ \frac{\partial L}{\partial \dot{x}} &= \frac{7}{10} m 2 \frac{dx}{dt} = \frac{7}{5} m \frac{dx}{dt} \\ \frac{d}{dt} \left( \frac{7}{5} m \frac{dx}{dt} \right) &= \frac{7}{5} m \frac{d^2x}{dt^2} \end{aligned}$$

(9.18)

Ecuaciones de movimiento completas:

- Para  $\alpha$ :

$$(I_c + mx^2) \frac{d^2\alpha}{dt^2} + 2mx \frac{dx}{dt} \frac{d\alpha}{dt} + mgx \cos(\alpha) = \tau$$

(9.19)

- Para  $x$ :

$$\begin{aligned} \frac{7}{5} m \frac{d^2x}{dt^2} &= mx \left( \frac{d\alpha}{dt} \right)^2 - mg \sin(\alpha) \Rightarrow \frac{d^2x}{dt^2} = \frac{5}{7} [x \left( \frac{d\alpha}{dt} \right)^2 - g \sin(\alpha)] \Rightarrow \\ &\Rightarrow \frac{d^2x}{dt^2} + \frac{5}{7} [g \sin(\alpha) - x \left( \frac{d\alpha}{dt} \right)^2] = 0 \end{aligned}$$

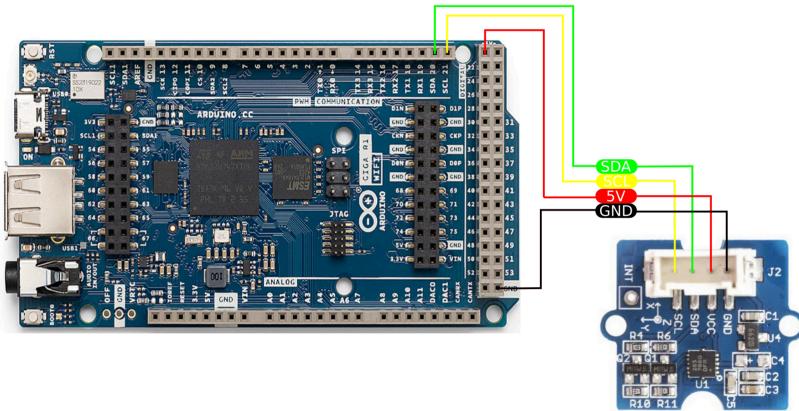
(9.20)

## 9.3 Integración con el robot UR3e en el entorno del laboratorio

La integración del robot *UR3e* en el entorno de laboratorio implica varios pasos para asegurar la comunicación y la funcionalidad efectiva entre todos los componentes del sistema. A continuación, se presenta el proceso de configuración y la dinámica operativa del conjunto.

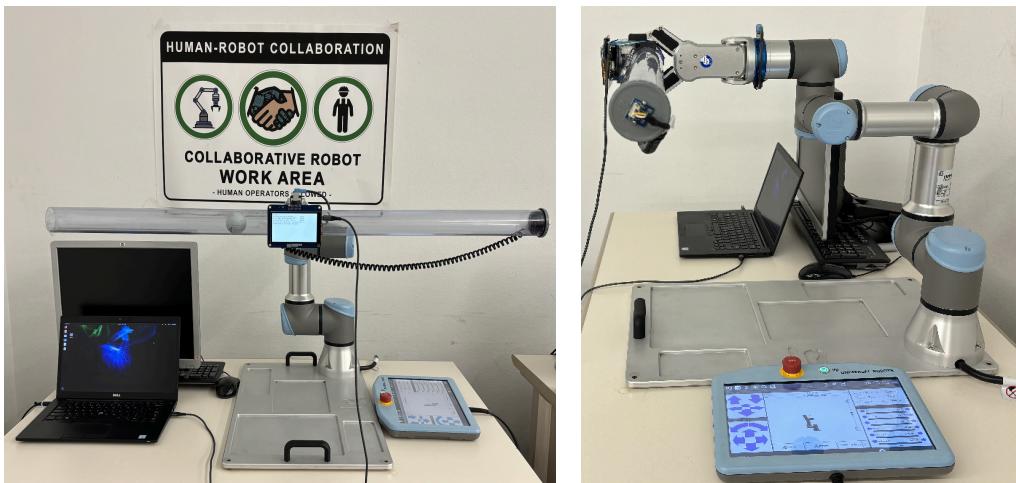
### 9.3.1 Integración del Hardware

Inicialmente, conectamos el sensor láser de distancia **VL53L0X** al **Arduino Giga Rev1 WiFi**, tal como se muestra en la Figura 18. Acto seguido añadimos el Arduino al cilindro y este a la pinza del brazo robótico.



*Figura 18: Conexión del Arduino Giga Rev1 WiFi con el sensor VL53L0X. Fuente: Elaboración propia*

El tubo, que sostiene la esfera, está fijado por la pinza *OnRobot RG2* en el centro de su masa tal como se aprecia en la Figura 19 en la vista frontal. El momento de fuerza necesario para mover el tubo hacia la derecha o hacia la izquierda y así equilibrar la esfera es proporcionado por el actuador número 6 del robot UR3e. Este actuador es justo la parte por la cual está conectada la pinza RG2, como se ve en la Figura 19 en la vista lateral.



*Figura 19: Sistema Ball and Beam en el entorno del laboratorio - Vista frontal (imagen izquierda), vista lateral (imagen derecha). Fuente: Elaboración propia*

El ordenador portátil se utiliza para el entrenamiento del agente de aprendizaje por refuerzo mediante *Python*. El cable USB que se muestra en la imagen anterior proporciona la alimentación al dispositivo Arduino y permite la visualización de información relacionada con la posición de la esfera y su velocidad. La comunicación entre el Arduino, el ordenador y el robot se realiza a través de Wi-Fi, utilizando el router Vodafone Sercomm Vox 2.5. La implementación de este método de comunicación ha sido seleccionado específicamente debido a su capacidad para ofrecer una transmisión de datos eficiente y segura. Esta elección

se fundamenta en la necesidad de asegurar la entrega oportuna de información en tiempo real con baja latencia.



Figura 20: Esferas: Metacrilato, Verde, Golf. Fuente: Elaboración propia

Para el entrenamiento del sistema de equilibrio, se emplearán tres esferas distintas, cada una con características únicas en cuanto a material y masa. Estas incluyen una esfera de metacrilato, una de golf y una esfera verde, como se observa en la Figura 20. Cada esfera ha sido seleccionada para evaluar y ajustar la adaptabilidad y precisión del sistema bajo diferentes condiciones de masa y textura, tal como muestra la Tabla 13, proporcionando así un rango amplio de datos para el refinamiento del algoritmo de control.

Material	Peso (g)
Verde	39g
Metacrilato	40g
Golf	46g

Tabla 13: Pesos de las diferentes esferas. Fuente: Elaboración propia

### Introducción a la Configuración y Operación del Sistema:

Antes de profundizar en los detalles técnicos de las siguientes secciones, es crucial comprender cómo se interconectan los diferentes componentes del sistema ball and beam y cómo fluye la información entre ellos.

Uno de los principales elementos del sistema de equilibrio, es un ordenador equipado con Python, que actúa como el cerebro de la operación entrenando el algoritmo de aprendizaje automático. Este ordenador es responsable de solicitar y recibir datos de un Arduino, que está programado para realizar tareas específicas como la lectura continua de un sensor de distancia. El Arduino, por tanto, se encarga de monitorizar la posición y la velocidad de la esfera en tiempo real, datos que son fundamentales para la operación del sistema.

Una vez que Python recibe estos datos, los procesa utilizando el algoritmo Soft Actor-Critic (SAC), un modelo de aprendizaje por refuerzo diseñado para tomar decisiones en tiempo real. El SAC evalúa la situación actual y determina la mejor acción a seguir, que en este contexto implica ajustar la posición de la esfera sobre el cilindro.

La acción determinada por el SAC se transmite luego al robot UR3e a través de una conexión WiFi. El robot, programado para entender y ejecutar comandos específicos de *URScript* como *servoj*[27], recibe estas instrucciones y ajusta sus motores para manipular la esfera hacia el estado deseado. Esta cadena de comandos y respuestas se lleva a cabo de manera continua y fluida para mantener el equilibrio de la esfera en el cilindro, ajustándose dinámicamente a las condiciones cambiantes del entorno.

Este flujo de información y control entre el ordenador, el Arduino, y el robot UR3e, es vital para la ejecución exitosa del proyecto. Cada componente cumple un rol específico que, cuando se coordina de manera efectiva, permite que el sistema opere con precisión y eficacia. A continuación, en las secciones 9.3.2, 9.3.3 y 9.3.4, se explora más a fondo cada uno de estos componentes y procesos, detallando su integración y funcionalidad dentro del entorno de laboratorio.

### 9.3.2 Sensor de distancia láser **VL53L0X**

El sensor láser **VL53L0X**, ofrece diferentes modos de configuración que ajustan su precisión y alcance, como se puede ver en la Tabla 14 y en la Tabla 15, siendo adaptable tanto para entornos interiores como exteriores. En su configuración predeterminada, el sensor puede medir distancias desde 50 mm hasta 1200 mm, contando también con un modo ampliado que permite mediciones hasta 2000 mm. La precisión de las mediciones varía en función del modo seleccionado y las condiciones del entorno, incluyendo la reflectividad del objetivo y la presencia de interferencias externas.

Modo	Timing	Alcance	Precisión
Default	30ms	1.2m	Ver tabla 15
Alta precisión	200ms	1.2m	+/- 3%
Largo alcance	33ms	2m	Ver tabla 15
Alta velocidad	20ms	1.2m	+/- 5%

Tabla 14: Especificaciones de Modos Operativos del Sensor VL53L0X. Fuente: [9]

	Indoor	Outdoor
Reflectancia objetivo	Distancia	33ms
Objetivo blanco	120cm	4%
Objetivo gris	70cm	7%

Tabla 15: Precisiones de referencia para los modos de funcionamiento estándar y largo alcance. Fuente: [9]

Basándonos en la evaluación anterior, hemos configurado el sensor en el modo estándar. Este modo ofrece un equilibrio óptimo entre el tiempo de lectura, el alcance y la precisión. Ahora procederemos a examinar y analizar el nivel de ruido generado por el sensor en esta configuración, para entender mejor su impacto en la calidad de las mediciones.

En la Figura 21, extraída de la hoja de especificación del sensor, se puede observar el ruido que presenta el sensor en el modo est\'andar:

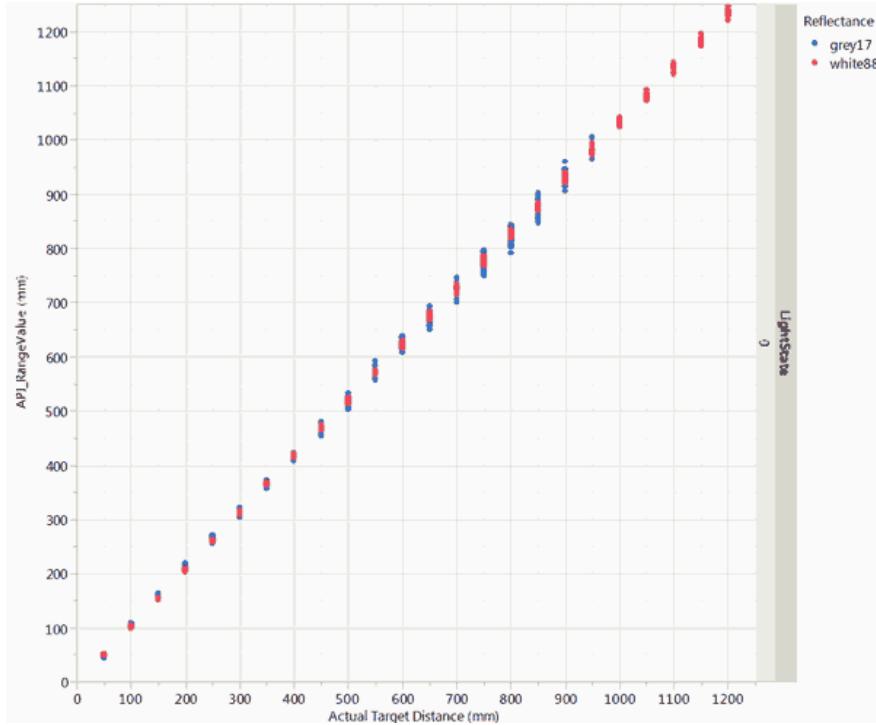


Figura 21: Nivel del ruido del sensor l\'aser para el modo de funcionamiento est\'andar. Fuente: [9]

La precisi\'on del sensor **VL53L0X** disminuye con el aumento de la distancia del objetivo, como se observa en las mediciones donde el ruido incrementa a medida que el objetivo se aleja. Adem\'as, la respuesta del sensor es no lineal, por lo cual se ha implementado una calibraci\'on utilizando un ajuste cuadr\'atico. Mediante el uso de MATLAB, se calcul\'o la siguiente ecuaci\'on de correcci\'on basada en datos experimentales:

$$y = 9.735 + 0.2274 \cdot x + 0.001235 \cdot x^2$$

Donde:

- $y$  es la distancia medida.
- $x$  es el valor bruto(*raw*) obtenido del sensor.

El proceso de calibraci\'on que se ha seguido ha sido el de obtener los datos del sensor a varias distancias conocidas, registrando tanto la distancia real (medida con un est\'andar de referencia) como la distancia le\'ida por el sensor. Despu\'es se ha usado **MATLAB** para ajustar una curva cuadr\'atica a los datos, como se aprecia en la Figura 22.

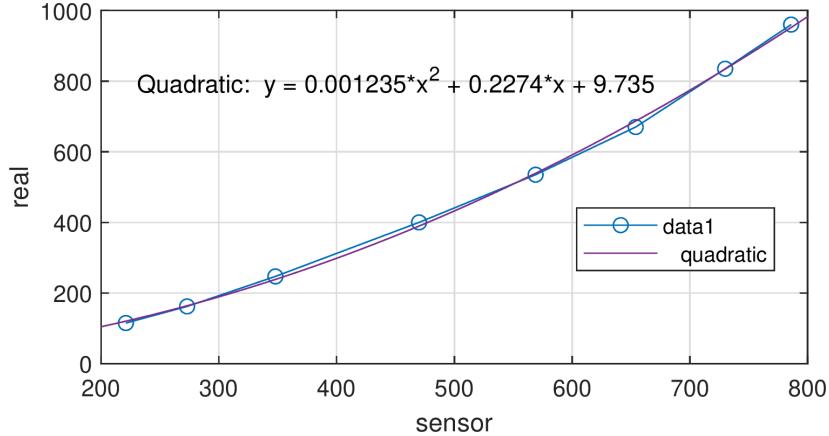


Figura 22: Calibración de las lecturas del sensor láser mediante un ajuste cuadrático. Fuente:  
Elaboración propia

Adicionalmente, también se ha aplicado un filtro pasobajos de tipo IIR (*Infinite Impulse Response*) de primer orden, también conocido como filtro de media móvil exponencial o filtro de suavizado exponencial, en el código del Arduino para suavizar las lecturas del sensor y reducir aún más el impacto del ruido del sensor de distancia. La estructura del filtro es la siguiente:

```

struct filter_data {
    float coef1 = 0.6;          // Coeficiente para la salida anterior
    float coef2 = 1 - coef1;    // Coeficiente para la entrada actual
    float in = 0;
    float out = 0;
};

float filter_function(float in, filter_data *filter_data) {
    filter_data->in = in;
    filter_data->out = filter_data->coef1 * filter_data->out + filter_data->coef2 * filter_data->in;
    return filter_data->out;
}

```

- **coef1 = 0.6:** Este coeficiente determina la influencia de la última salida del filtro (valor anterior) en la salida actual. Un valor más alto da más peso al historial, haciendo que el filtro sea más lento en responder a cambios.
- **coef2 = 0.4:** Este es el complemento de coef1, y determina la influencia del valor de entrada actual en la salida del filtro. Un valor más alto hace que el filtro responda más rápidamente a los cambios recientes.
- **Ecuación del Filtro:** La ecuación balancea entre la señal de entrada más reciente y los valores pasados, proporcionando una salida que reduce el ruido, pero sigue siendo sensible a los cambios reales en los datos de entrada.

Este filtro mejora significativamente la estabilidad de las mediciones al reducir fluctuaciones rápidas y transitorias en los datos del sensor.

### 9.3.3 Latencias y Funcionamiento del Código en el Arduino

El funcionamiento eficaz del sistema de equilibrio depende en gran medida de la capacidad para procesar y transmitir información entre los componentes del sistema en tiempo real. A continuación, se detallan los tiempos de cómputo involucrados y cómo influyen en el rendimiento general del sistema.

#### Latencias del Sistema:

- **Latencia del Sensor Láser VL53L0X:** El sensor de distancia VL53L0X, esencial para medir la posición de la esfera, presenta un tiempo de cómputo máximo de aproximadamente 30 ms para completar una medición y transmitir los datos. Este tiempo es inherente al proceso de medición del tiempo de vuelo de la luz emitida y reflejada por el objeto.
- **Latencia de la Conexión WiFi:** La conexión *Wi-Fi* entre el Arduino y el ordenador introduce un tiempo de transmisión adicional para los datos, como se muestra en la Tabla 16. En condiciones normales, esta latencia es de aproximadamente 5 ms, aunque puede llegar a 10 ms en situaciones de tráfico de red elevado o interferencia. Dado que el sistema depende de la comunicación continua para el envío de comandos de control y la recepción de datos sensoriales, el rendimiento de la red *Wi-Fi* es un componente crítico para minimizar los retrasos en el control del robot.

Elemento	Tiempo
Wifi	10ms
VL53L0X	30ms

Tabla 16: Tabla de las latencias. Fuente: Elaboración propia

Para mitigar la latencia de la red en el sistema de control del robot, se ha limitado la conexión *WiFi* únicamente al ordenador portátil, al Arduino y al brazo robótico UR3e. Esta estrategia previene la sobrecarga del router al minimizar el número de dispositivos conectados, lo cual es crucial para mantener la latencia por debajo de 10 ms. Adicionalmente, todos los dispositivos están ubicados a una distancia menor de 2 metros del router, para asegurar una señal fuerte, constante y para limitar las posibles interferencias.

#### Funcionamiento del Código en Arduino:

El Arduino desempeña un papel central en la recopilación de datos del sensor y en la comunicación con el ordenador, que ejecuta el algoritmo de control en Python. A continuación, se describe el proceso operativo:

- **Ciclo de Tarea Monocíclica en Arduino:** El Arduino está programado para ejecutar una tarea monocíclica consistente en la lectura continua del sensor VL53L0X. Este ciclo tiene un *delay* inherente de aproximadamente 5 ms, que es el tiempo dedicado a

la adquisición de datos y su preparación para la transmisión. La eficiencia de este ciclo es vital, ya que asegura que los datos más recientes estén disponibles para el siguiente paso del proceso de control.

- **Solicitud de Datos por el Ordenador:** Desde el código *Python*, el ordenador envía solicitudes periódicas al Arduino para obtener los datos más recientes sobre la posición y velocidad de la esfera. La rapidez con la que el Arduino puede responder a estas solicitudes depende de su punto en el ciclo de tarea cuando recibe la petición.
- **Comunicación y Acción del Robot:** Una vez que el ordenador procesa los datos recibidos del Arduino, genera y envía comandos de control al robot UR3e. Este proceso también está sujeto a las latencias de la comunicación *Wi-Fi* y del propio ciclo de procesamiento del controlador del robot.

Cuando el ordenador solicita datos al Arduino, éste responde con los datos más recientes de posición y velocidad de la esfera. La velocidad de esta respuesta depende del punto en que la solicitud del ordenador coincide con el ciclo de muestreo del Arduino. En el peor de los casos, la respuesta puede demorarse hasta completar el ciclo de muestreo en curso.

#### **Estimación de los Tiempos Totales de las Tareas Asociadas:**

Para calcular la latencia total del sistema, consideramos todas las etapas:

- Tiempo de lectura del sensor de distancia: 30 ms (máximo)
- Comunicación entre Arduino a ordenador: 10 ms (máximo para cada tramo, sumando un total de 20 ms en el peor caso)
- Tiempo de Procesamiento en el Ordenador: Variable, asumamos 10 ms como caso pero para un cálculo conservador.
- Comunicación entre ordenador y robot: 10 ms (máximo)

Sumando todas estas contribuciones, el tiempo total podría ser de hasta 70 ms en el peor caso. Teniendo esto en cuenta, el control del sistema se hará cada 200 ms. Es decir, se enviará una acción al brazo robótico cada 0.2 segundos.

#### **9.3.4 Control del robot UR3e**

El controlador del robot UR3e, que está implementado en un entorno operativo basado en Linux y configurado con un núcleo optimizado para operaciones en tiempo real, administra tanto *Polyscope* como la ejecución del controlador *URControl*. Este controlador proporciona datos continuos que representan el estado del robot, como posiciones, temperaturas, etc., a través de varios *sockets* de servidor en el controlador. Estos *sockets* varían en tiempo de

respuesta y en el tipo de datos que pueden transmitir. Las interfaces de comunicación del UR3e están diseñadas para recibir comandos *URScript* y se diferencian principalmente por la frecuencia de actualización y la complejidad de los datos que pueden manejar. En la Tabla 17 se resumen las características principales de cada interfaz:

e-Series							
	Primary		Secondary		Real-time		Real-time Data Exchange (RTDE)
Port no.	30001	30011	30002	30012	30003	30013	30004
Frequency [Hz]	10	10	10	10	500	500	500
Receive	URScript commands	-	URScript commands	-	URScript commands	-	Various data
Transmit	See attachment from the bottom		See attachment from the bottom		See attachment from the bottom		See RTDE Guide

Tabla 17: Tabla de principales interfaces de comunicación con el robot UR3e. Fuente: [28]

Hay tres interfaces principales consideradas para el envío de comandos *URScript*. Dos de estas interfaces operan a una frecuencia de aproximadamente 10 Hz, lo que corresponde a una respuesta cada 100 ms. La tercera interfaz opera a unos 500 Hz, es decir, una respuesta cada 2 ms, y es conocida como la interfaz *Real Time*.

Además de las interfaces mencionadas anteriormente, el UR3e dispone de la interfaz *RTDE*, una herramienta potente para el intercambio de datos en tiempo real entre el robot y aplicaciones externas. A diferencia de las interfaces que operan con comandos *URScript*, la *RTDE* se centra en el control de registros y el intercambio de datos estructurados.

En el contexto del proyecto en el entorno de laboratorio, hemos optado por usar la interfaz *Real Time* en lugar de la *RTDE* principalmente por la compatibilidad directa con el *URScript* y la facilidad de implementación. La interfaz Real Time permite la ejecución directa de comandos *URScript*, que es crucial para la integración y operación inmediata de nuestros algoritmos de control. *URScript* proporciona un medio directo y flexible para enviar instrucciones al robot, que se ajusta de manera precisa a las necesidades de control en tiempo real del robot. Aunque la *RTDE* es valorada por su capacidad para manejar intercambios de datos estructurados y control de registros, requeriría un nivel adicional de complejidad en la integración para manejar directamente comandos de control desde el Arduino. Por estas razones, y para simplificar el proceso de desarrollo e implementación, se ha elegido la interfaz *Real Time* que también ofrece un alto rendimiento en términos de tiempos de respuesta similar a la del *RTDE*.

**Control del Movimiento del Actuador 6 mediante el comando servoj:** El comando *servoj* se utiliza para controlar las posiciones de las articulaciones del robot, específicamente el actuador número 6 que es responsable de rotar el cilindro sobre el eje que permite equilibrar la esfera. Este comando ajusta la posición actual del actuador hacia la posición deseada (denotada por  $q$ ) con una dinámica definida por varios parámetros:

```
servoj(q=[0, -1.5708, -1.5708, 0, -4.7100, 1.570796 + corrección], a=0, v=0, t=0.2, lookahead_time=0.03, gain=2000)
```

- **q:** Vector que especifica las posiciones objetivo de las articulaciones en radianes. Solo ajustamos la posición de la sexta articulación (*joint6*), según las necesidades del sistema Ball and Beam.
- **t:** Tiempo de control durante el cual el comando es efectivo. Establecemos este valor en 0.2 segundos, 200 ms, para garantizar que el comando tenga suficiente tiempo para influir en el movimiento del robot mientras se adapta a los cambios de dinámica de la esfera.
- **lookahead\_time:** Configurado en 0.03 segundos, 3 ms, este parámetro permite al robot anticipar su posición futura basándose en su velocidad actual, ofreciendo una respuesta más inmediata a las instrucciones de control.
- **gain:** Ajustado a 2000, este parámetro de ganancia asegura que el robot intente alcanzar la posición deseada (q) lo más rápidamente posible, ajustando la posición actual hacia esta dirección.

El comando *servoj* ajusta la posición actual del actuador hacia la posición deseada utilizando un enfoque similar al término *P* de un controlador *PID*(Proporcional, Integral, Derivativo). Este enfoque asegura que el actuador reaccione rápidamente a las diferencias entre su posición actual y la posición deseada. La alta ganancia y el tiempo de anticipación corto son esenciales para reacciones rápidas.

### Sincronización y Envío de Comandos:

El flujo de trabajo con la interfaz *Real Time* es el siguiente:

1. **Recepción de Datos:** El Arduino, después de recoger los datos del sensor **VL53L0X**, envía la posición y la velocidad de la esfera al ordenador.
2. **Procesamiento y Comandos:** El ordenador, utilizando el modelo de aprendizaje por refuerzo, procesa esta información y envía un comando *URScript servoj* con la acción correspondiente.
3. **Envío de Comandos:** Estos comandos se envían a través de la interfaz *Real Time* al *control box* del UR3e.
4. **Acción del Robot:** El *control box* procesa estos comandos y ajusta las acciones del brazo robótico en consecuencia.

Para mantener la coherencia y la precisión, el código en *Python* está sincronizado para enviar comandos cada 200 ms al puerto 30003, que es donde la interfaz Real Time escucha. Cuando

el *control box* del *UR3e* recibe un comando nuevo y aún no ha terminado de ejecutar el anterior, detiene la acción actual y pasa a ejecutar el siguiente comando recibido. El comando *servoj* es bloqueante dentro del sistema operativo del *control box* del robot, pero solo durante un ciclo de tiempo específico ( $t$ ), que está configurado en 200 ms. Esto significa que si un nuevo comando llega al control box antes de que finalice este tiempo, el comando actualmente en ejecución se interrumpe y el nuevo comando toma prioridad, comenzando su ejecución inmediatamente.

Por esta razón hay que mantener una sincronización precisa entre:

- **La acción del robot:** Asegurando que cada movimiento enviado tenga la oportunidad de influir en el estado del sistema antes de ser potencialmente interrumpido por un nuevo comando.
- **El tiempo de espera desde el envío del comando:** Desde que se envía el comando desde el ordenador hasta que el robot comienza su ejecución.
- **La observación del sensor:** La lectura de la posición actual de la esfera, que debe corresponder al efecto del último comando ejecutado para proporcionar datos coherentes al modelo de aprendizaje.

Una desincronización en este proceso puede introducir ruido significativo en los datos observados, llevando a inferencias incorrectas sobre el estado actual del sistema por parte del agente de aprendizaje por refuerzo. Por ejemplo, si se comandan movimientos de 85 y luego 98 grados en rápida sucesión, y el comando de 85 grados es descartado antes de su ejecución completa, el agente podría recibir datos sensoriales que esperaría correspondieran a 85 grados, pero en realidad reflejan la posición resultante de 98 grados. Esto podría comprometer seriamente el proceso de aprendizaje, dado que el agente podría no estar aprendiendo de la retroalimentación correcta, afectando a la convergencia del modelo.

# 10. Desarrollo del algoritmo Soft Actor Critic

## 10.1 Fundamentos del Soft Actor Critic

El *Soft Actor Critic (SAC)*[29-32] es un algoritmo avanzado, de aprendizaje por refuerzo off-policy que optimiza una política estocástica en un entorno definido por un proceso de decisión de *Markov*. Este método es particularmente eficaz debido a su enfoque de maximización de la entropía, que no solo busca maximizar la suma total de recompensas, sino también fomentar la exploración del agente mediante la maximización de la diversidad de las acciones tomadas.

### Características importantes:

- **Maximización de la entropía:** El SAC incorpora un término de entropía,  $\mathbf{H}$ , en la función objetivo, que promueve la exploración activa por parte del agente. Este término de entropía penaliza las políticas deterministas y fomenta las políticas que mantienen una distribución más uniforme de acciones posibles.
- **Política Estocástica ( $\pi$ ):** SAC implementa una política estocástica  $\pi(a|s)$ , que define la probabilidad de tomar la acción  $a$  dado el estado  $s$ . Esta política estocástica permite que el algoritmo explore de manera más efectiva el espacio de acciones.
- **Actor-Crítico con doble Q-learning:** SAC emplea una arquitectura actor-crítico donde el **actor** actualiza la política para seguir el gradiente de la función de valor, mientras que el **crítico** estima los valores de acción utilizando dos redes Q independientes,  $Q_{\theta_1}$  y  $Q_{\theta_2}$ . La doble estimación ayuda a reducir la sobreestimación del valor. En cada paso de tiempo, el algoritmo actualiza estas funciones para reflejar mejor las recompensas esperadas de las acciones según el estado actual.

En la Figura 23 se puede apreciar la arquitectura actor-crítico:

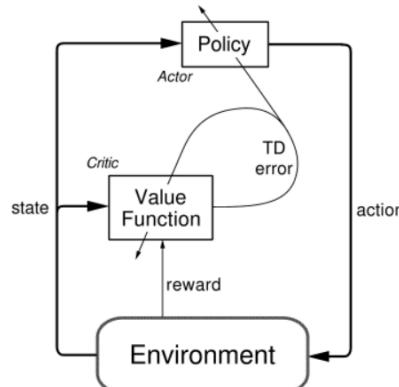


Figura 23: Diagrama Actor-Crítico. Fuente: [29]

- **Aprendizaje off-policy:** A diferencia de los métodos on-policy que requieren que las políticas para explorar y aprender sean las mismas, SAC puede aprender de experiencias antiguas almacenadas en un buffer de repetición, independientemente de la política que generó esos datos. Esto permite al SAC beneficiarse de una eficiente reutilización de datos anteriores, acelerando el aprendizaje sin comprometer la calidad del mismo.
- **Experience Replay:** El Experience Replay implica almacenar transiciones experimentadas por el agente en un buffer de repetición. Una transición típicamente incluye el estado actual, la acción tomada, la recompensa recibida, y el siguiente estado. El buffer, por lo tanto, actúa como una base de datos de experiencias pasadas que el agente puede utilizar para aprender, en lugar de depender solo de la experiencia más reciente.

Para introducir el Soft Actor-Critic (SAC), es esencial comprender el concepto de aprendizaje por refuerzo regularizado por entropía, que incorpora la entropía como parte del proceso de toma de decisiones para fomentar la exploración del agente:

- **Entropía:** La entropía, en el contexto del aprendizaje por refuerzo, es una medida que describe cuán aleatoria es una variable aleatoria.

$$H(P) = E_{x \sim P}[-\log P(x)] \quad (10.1)$$

Donde:

- $x$ , es una variable aleatoria con una función de masa o densidad de probabilidad  $P$

Esta fórmula indica que la entropía es el promedio negativo del logaritmo de las probabilidades asociadas a los posibles resultados de  $x$ .

El SAC se basa en el principio de la optimización de la política mediante el ajuste de la función de valor  $Q$  y la política  $\pi$ . El objetivo de aprendizaje se formula como la maximización del retorno esperado sumado al término de entropía, matemáticamente es la siguiente función objetivo:

$$J(\pi) = \sum_t \mathbb{E}_{(s_t, a_t) \sim p_\pi} [r(s_t, a_t) + \alpha H(\pi(\cdot | s_t))] \quad (10.2)$$

Donde:

- $p_\pi$ , es la distribución de estados bajo la política  $\pi$ .
- $r(s_t, a_t)$ , es la recompensa obtenida al tomar la acción  $a_t$  en el estado  $s_t$ .
- $\alpha$  es el coeficiente de temperatura que equilibra la importancia relativa de la entropía versus la recompensa. Controla la estocasticidad de la política óptima.

- $H(\pi(\cdot | s_t))$ , es la entropía de la política en el estado  $s_t$ , promoviendo la exploración.

### Dualidad de las Funciones Q:

La técnica de doble Q-learning se utiliza para mitigar el problema de la sobreestimación de las estimaciones de los valores de acción. El método emplea dos redes  $Q$  independientes,  $Q_{\theta_1}$  y  $Q_{\theta_2}$ , para estimar los valores de acción. En cada paso de actualización, SAC calcula el mínimo de las dos estimaciones de  $Q$  para formar el valor objetivo:

$$Q^{\text{target}}(s_t, a_t) = r(s_t, a_t) + \gamma \min_{i=1,2} Q_{\theta_i}(s_{t+1}, \pi(s_{t+1})) \quad (10.3)$$

### Formulación del Bellman Residual:

SAC aplica una versión modificada de la ecuación de *Bellman* que integra un término de entropía para promover la exploración. Este enfoque no solo busca maximizar las recompensas futuras sino también mantener la diversidad en la selección de acciones, lo que es crucial en entornos de alta dimensionalidad o desconocidos. La función objetivo de las redes  $Q$  en SAC se puede describir mediante el residual de *Bellman*:

$$J_Q(\theta_i) = \mathbb{E}_{(s_t, a_t) \sim D} \left[ \frac{1}{2} (Q_{\theta_i}(s_t, a_t) - (r(s_t, a_t) + \gamma V(s_{t+1})))^2 \right] \quad (10.4)$$

, donde  $V(s_{t+1})$  es el valor del estado calculado como:

$$V(s_{t+1}) = \mathbb{E}_{a_{t+1} \sim \pi} \left[ \min_{i=1,2} Q_{\theta_i}(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1} | s_{t+1}) \right] \quad (10.5)$$

Este término de entropía  $\alpha \log \pi(a_{t+1} | s_{t+1})$  actúa como una regularización que contrarresta la tendencia a seleccionar acciones de manera determinista, incentivando la exploración al hacer que el agente valore igualmente las acciones con recompensas esperadas similares.

### Cálculo de Gradientes:

Las actualizaciones de los parámetros de las redes en SAC implican calcular gradientes de las funciones de pérdida respectivas para las redes de actor y crítico. Los gradientes para el crítico se calculan como:

$$\nabla_{\theta_i} J_Q(\theta_i) = \mathbb{E}_{(s_t, a_t) \sim D} [(Q_{\theta_i}(s_t, a_t) - y_t) \nabla_{\theta_i} Q_{\theta_i}(s_t, a_t)] \quad (10.6)$$

Donde,  $y_t = r(s_t, a_t) + \gamma V(s_{t+1})$ .

El gradiente de la red de actor se optimiza para maximizar la suma de la función  $Q$  y el término de entropía, lo que puede formalizarse como:

$$\nabla_{\theta} J_{\pi}(\theta) = \mathbb{E}_{s_t \sim D} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (\alpha \log \pi_{\theta}(a_t | s_t) - Q_{\theta}(s_t, a_t))] \quad (10.7)$$

Estos métodos de actualización garantizan que la política converge hacia un equilibrio donde se balancea efectivamente entre exploración y explotación, lo que se refleja en actualizaciones estables y mejoras incrementales en la política a lo largo del entrenamiento.

### Ajuste Automático de la Temperatura:

Para ajustar eficazmente el parámetro  $\alpha$ , se usa un enfoque de optimización donde  $\alpha$  se ajusta automáticamente para equilibrar la exploración y la explotación. La adaptación continua de  $\alpha$  asegura que el agente no se estanke en comportamientos deterministas, lo que es crítico en entornos complejos donde múltiples acciones podrían ser igualmente viables. Al mismo tiempo, esto permite que el agente descubra y explote caminos de acción que podrían haber sido descuidados bajo una política más determinista. El objetivo es mantener la entropía de la política cerca de un umbral deseado:  $H_0$ , para asegurar una exploración del entorno suficiente. La formulación del problema de optimización es:

$$\max_{\pi} \mathbb{E} \left[ \sum_t r(s_t, a_t) \right] \quad \text{sujeto a } H(\pi_t) \geq H_0 \quad (10.8)$$

Usando la dualidad Lagrangiana, este problema se transforma en:

$$L(\pi, \alpha) = \mathbb{E} \left[ \sum_t r(s_t, a_t) \right] + \alpha (H(\pi_t) - H_0) \quad (10.9)$$

El objetivo es entonces maximizar  $L$  respecto a  $\pi$  y minimizarlo respecto a  $\alpha$ . Esto implica un ajuste fino donde  $\alpha$  se ajusta para equilibrar la exploración y explotación eficazmente, adaptándose a los cambios en el comportamiento del agente y las demandas del entorno.

El ajuste de  $\alpha$  y  $\pi$  se realiza de manera iterativa. Primero, se ajusta  $\pi$  para maximizar  $L$  para un  $\alpha$  fijo, explorando efectivamente nuevas estrategias que podrían ofrecer mejores recompensas. Luego,  $\alpha$  se ajusta para minimizar  $L$ , penalizando las estrategias que resulten en una entropía demasiado baja comparada con el umbral  $H_0$ :

$$\pi^* = \arg \max_{\pi} L(\pi, \alpha) \quad \alpha^* = \arg \min_{\alpha} L(\pi^*, \alpha) \quad (10.10)$$

El algoritmo *Soft Actor-Critic (SAC)* mostrado en la Figura 24 sigue un proceso iterativo para entrenar un agente en un entorno de aprendizaje por refuerzo. El algoritmo actualiza dinámicamente los parámetros de dos funciones de valor  $Q$  y una política  $\pi$  usando gradientes calculados a partir de transiciones muestreadas del entorno.

<b>Algorithm 1</b> Soft Actor-Critic	
<b>Input:</b> $\theta_1, \theta_2, \phi$	▷ Initial parameters
$\tilde{\theta}_1 \leftarrow \theta_1, \theta_2 \leftarrow \theta_2$	▷ Initialize target network weights
$\mathcal{D} \leftarrow \emptyset$	▷ Initialize an empty replay pool
<b>for</b> each iteration <b>do</b>	
<b>for</b> each environment step <b>do</b>	
$a_t \sim \pi_\phi(a_t   s_t)$	▷ Sample action from the policy
$s_{t+1} \sim p(s_{t+1}   s_t, a_t)$	▷ Sample transition from the environment
$\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$	▷ Store the transition in the replay pool
<b>end for</b>	
<b>for</b> each gradient step <b>do</b>	
$\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$	▷ Update the Q-function parameters
$\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$	▷ Update policy weights
$\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$	▷ Adjust temperature
$\tilde{\theta}_i \leftarrow \tau \theta_i + (1 - \tau) \bar{\theta}_i$ for $i \in \{1, 2\}$	▷ Update target network weights
<b>end for</b>	
<b>end for</b>	
<b>Output:</b> $\theta_1, \theta_2, \phi$	▷ Optimized parameters

Figura 24: Pseudocódigo del algoritmo Soft Actor-Critic. Fuente: [30]

Aquí están los pasos clave del algoritmo:

1. **Inicialización:** Comienza con la inicialización de los parámetros  $\theta$  para dos funciones  $Q$  y  $\phi$  para la política. Además, se inicializa un *experience replay* vacío  $D$  para almacenar las transiciones.
2. **Muestreo y Almacenamiento:** En cada paso de cada episodio, el algoritmo selecciona y ejecuta una acción según la política actual, observa una nueva transición del entorno, y la almacena en el *experience replay*.
3. **Actualización de Parámetros:**
  - a. **Funciones Q:** Se actualizan los parámetros  $\theta$  de las funciones  $Q$  minimizando la diferencia entre los valores  $Q$  actuales y los valores objetivo, que incorporan recompensas recibidas y el valor del próximo estado ajustado por el término de entropía.
  - b. **Política  $\pi$ :** Se actualizan los parámetros  $\phi$  de la política para maximizar la suma esperada de la función  $Q$  menos un término de entropía, fomentando así una exploración efectiva.
  - c. **Temperatura ( $\alpha$ ):** Se ajusta el parámetro  $\alpha$  para equilibrar la importancia de la entropía frente a las recompensas, permitiendo una exploración adecuada y evitando políticas deterministas excesivas.
  - d. **Redes Objetivo:** Los parámetros de las redes objetivo se actualizan suavemente para estabilizar el aprendizaje, evitando cambios drásticos que podrían resultar en divergencia.

4. **Output:** Al final de la ejecución, el algoritmo devuelve los parámetros optimizados para las funciones  $Q$  y la política, los cuales definen cómo el agente debe actuar en el entorno dado.

#### **Resumen de los parámetros principales del algoritmo SAC:**

A continuación, se explican brevemente algunos de los parámetros del algoritmo que se analizarán más en detalle en la sección de pruebas:

- **Batch Size (tamaño de lote):** Este parámetro determina el número de muestras de experiencia utilizadas en cada iteración de entrenamiento. Un tamaño de lote más grande puede proporcionar una estimación del gradiente más estable, pero también puede aumentar la carga computacional y el tiempo de entrenamiento. Un tamaño de lote adecuadamente grande asegura que cada actualización aproveche una muestra representativa de experiencias, reduciendo la varianza en las actualizaciones y mejorando la estabilidad del entrenamiento.
- **Target Entropy:** La entropía objetivo es una medida que busca mantener un nivel deseado de aleatoriedad en la selección de acciones. Al establecer una entropía objetivo, el algoritmo ajusta dinámicamente el coeficiente de temperatura ( $\alpha$ ) para balancear la exploración (probando nuevas acciones) y la explotación (optimizando las acciones conocidas). Este balance es crucial para asegurar que el agente no se quede atrapado en políticas subóptimas y pueda adaptarse a diversos aspectos del entorno.
- **Discount Factor:** Este factor, denotado como  $\gamma$ , determina la importancia relativa de las recompensas futuras. Un factor de descuento más alto hace que el agente valore más las recompensas a largo plazo, mientras que un valor más bajo lo hace enfocarse en recompensas inmediatas. En entornos con mucha incertidumbre o ruido, un factor de descuento más bajo puede ayudar a mitigar el impacto negativo de las estimaciones inexactas de las recompensas futuras.
- **Target Update Interval:** Este parámetro establece la frecuencia con la que se actualizan los parámetros de la red objetivo. Actualizar la red objetivo con regularidad, pero no demasiado frecuentemente, es vital para mantener la estabilidad del aprendizaje. Un intervalo equilibrado ayuda a prevenir los problemas de oscilaciones excesivas en las estimaciones de valor y contribuye a un aprendizaje más estable y robusto.
- **Target Smoothing Coefficient ( $\tau$ ):** Este coeficiente determina cuán rápidamente se incorporan los nuevos parámetros aprendidos en la red objetivo. Un coeficiente más alto puede acelerar el aprendizaje al adaptarse rápidamente a los nuevos datos, pero también puede llevar a inestabilidad si los cambios son demasiado abruptos. Un valor más bajo suaviza la transición, permitiendo una integración más gradual que puede

mejorar la estabilidad del aprendizaje al mantener una cierta continuidad en las estimaciones de la red.

## 10.2 Implementación del SAC

Se ha implementado el agente SAC en el lenguaje de programación *Python*, utilizando la biblioteca *PyTorch*.

La implementación del SAC se organiza en varios módulos para separar claramente las responsabilidades y facilitar la mantenibilidad del código:

- **arduino.py:** Gestiona la comunicación con el Arduino para obtener datos en tiempo real como la posición y la velocidad de la esfera.
- **buffer.py:** Implementa el *replay buffer*, una estructura de datos que permite almacenar y reutilizar experiencias pasadas. Este buffer opera como un anillo (*ring buffer*), optimizando la gestión de memoria y acceso a los datos.
- **networks.py:** Define las redes neuronales que representan tanto la política (actor) como las funciones de valor Q (crítico).
- **utils.py:** Contiene herramientas auxiliares como funciones para guardar el progreso del entrenamiento, generación de gráficos, entre otros.
- **sac.py:** Encapsula la lógica del agente SAC, gestionando la actualización de políticas y la interacción con el buffer de experiencias.
- **main.py:** Ejecuta el bucle principal de entrenamiento donde el agente interactúa con el entorno y aprende de sus acciones.

### 10.2.1 Detalles del Entorno

#### Espacio de Estados y Acciones:

- **Espacio de Estados:** El espacio de estados es tridimensional, compuesto por la posición de la esfera ( $x$ ), la velocidad de esta ( $vx$ ) y el ángulo de acción del actuador (en radianes). Estas variables son críticas para determinar la dinámica del sistema y la respuesta adecuada del actuador. Para facilitar el procesamiento y mejorar la eficiencia del modelo, tanto la posición  $x$  como la velocidad  $vx$  se normalizan a un rango de [-1, 1], centrado en 0 la parte positiva corresponde a la parte derecha del cilindro y la parte negativa a la parte izquierda del cilindro, asegurando que cada dimensión contribuya equitativamente al aprendizaje y a la decisión del modelo.

- **Espacio de Acciones:** El espacio de acciones es unidimensional, donde la acción representa el ajuste en radianes que el actuador debe realizar para mantener la esfera balanceada sobre un cilindro. El rango de acciones está limitado entre [-0.27, 0.27] radianes, aproximadamente  $\pm 15.47$  grados, para garantizar la seguridad del robot.

**Experience Replay:** Tiene capacidad para almacenar hasta  $10^6$  experiencias, cada una compuesta por una tupla de cinco elementos: estado actual, acción, recompensa, estado siguiente y una variable booleana que indica si el episodio ha terminado. Este mecanismo permite al agente aprender de experiencias pasadas, mejorando la estabilidad y la eficiencia del aprendizaje al descorrelacionar las experiencias.

**Recompensa (Reward):** El diseño de la función de recompensa en un algoritmo de aprendizaje por refuerzo es fundamental, ya que guía al agente hacia el comportamiento deseado. En el contexto del Soft Actor-Critic aplicado al sistema ball and beam, la función de recompensa es diseñada meticulosamente para penalizar las desviaciones de un estado objetivo, mientras que también penaliza velocidades altas que podrían indicar un control inestable.

La recompensa,  $r(s_t, a)$ , se calcula en cada paso temporal  $t$  y depende de dos factores principales: la posición actual de la esfera  $x$ , la velocidad  $v_x$ . La función *calculate\_reward(x, vx)* encapsula la lógica para calcular esta recompensa basada en el estado actual y la acción tomada.

La función de recompensa se estructura alrededor de un objetivo principal: mantener la esfera dentro de un rango definido cerca del centro del cilindro. Para fomentar esto, la función utiliza un sistema de penalizaciones:

- **Penalización por Distancia:** Si la esfera se encuentra fuera del rango objetivo, se impone una penalización proporcional a la distancia absoluta de la esfera al límite más cercano del rango objetivo. Esto incentiva al agente a corregir posiciones que se alejan del centro.

$$\text{distance\_penalty} = \text{distance\_penalty\_coefficient} \times |x - \text{target\_boundary}| \quad (10.11)$$

- **Penalización por Velocidad:** Independientemente de la posición, una velocidad alta de la esfera implica una mayor inestabilidad. Por ello, se penaliza la velocidad absoluta de la esfera para incentivar al agente a mantener un movimiento suave y controlado.

$$\text{velocity\_penalty} = \text{velocity\_penalty\_coefficient} \times |v_x| \quad (10.12)$$

La recompensa total en cualquier paso de tiempo se calcula sumando las penalizaciones:

$$\text{reward} = -( \text{distance\_penalty} + \text{velocity\_penalty} ) \quad (10.13)$$

Este enfoque asegura que el agente reciba una señal negativa fuerte al alejarse de los comportamientos deseados, y una penalización menor o ninguna penalización al mantenerse dentro del rango objetivo y moverse a una velocidad baja.

Además de estas penalizaciones, se incluye una penalización fija. Si la esfera toca cualquiera de los bordes del cilindro, se impone una penalización fija. Entonces el *reward* sería igual al valor de esta penalización. La penalización es importante porque evita que el agente permita que la esfera alcance y permanezca en los extremos del cilindro, lo que podría llevar a fallos en el control o a caídas de la esfera.

El diseño detallado de esta función de recompensa facilita que el agente aprenda no solo a buscar el equilibrio estático, sino también a realizar ajustes dinámicos y controlados. Esta estrategia refuerza una exploración prudente del espacio de estados y promueve un aprendizaje eficiente y robusto.

### 10.2.2 Redes Neuronales en SAC

Las redes neuronales en SAC desempeñan un papel fundamental, modelando la política y las funciones de valor  $Q$ .

Una red neuronal artificial es un modelo computacional inspirado en la estructura y funciones del cerebro humano. Está compuesta de nodos, también llamados neuronas, organizadas en capas. Cada neurona recibe entradas, ya sea de datos externos o de la salida de otras neuronas de capas anteriores. Estas entradas son procesadas para producir una salida a través de una función de activación. Las distintas capas son:

- **Capa de Entrada (*Input Layer*):** Recibe las entradas que representan los estados del entorno. En SAC, estas son las observaciones del estado actual del sistema que el agente está tratando de controlar.
- **Capas Ocultas (*Hidden Layers*):** Estas capas realizan la mayor parte del procesamiento mediante conexiones ponderadas. Las neuronas en estas capas reciben entradas de la capa anterior, las cuales son transformadas por pesos y sesgos, y finalmente son procesadas por funciones de activación.
- **Capa de Salida (*Output Layer*):** Produce la salida del modelo, que para el SAC, es crucial, pues determina las acciones a ejecutar. Diferente de otras aplicaciones, en SAC esta capa no solo emite la acción más probable, sino también parámetros de una distribución de probabilidad (media y desviación estándar) que define un espectro de acciones posibles, crucial para la exploración estocástica del espacio de acciones.

En la Figura 25 se puede ver la red neuronal del actor:

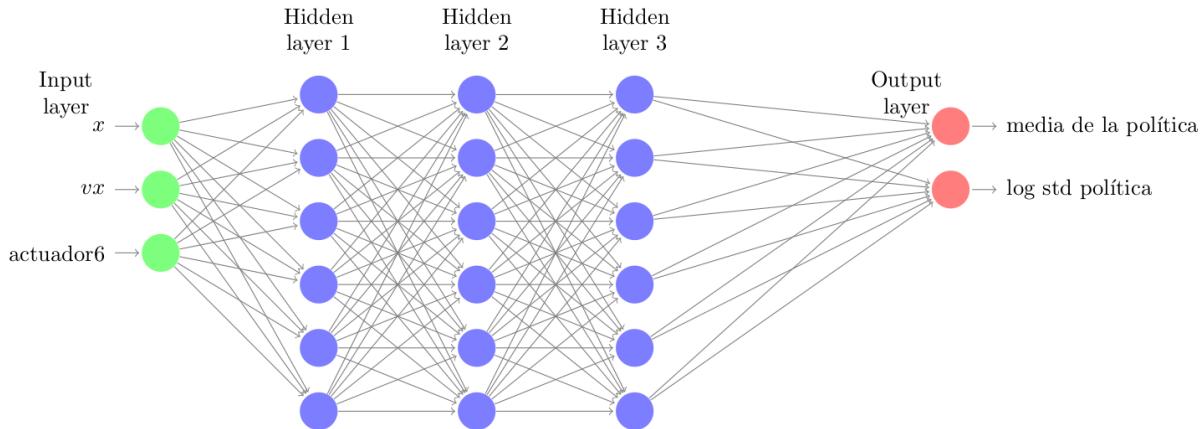


Figura 25: Ejemplo de la red neuronal del actor. Fuente: Elaboración propia

Cada neurona en la red procesa sus entradas a través de una función de activación, que introduce no linealidades en el modelo, permitiendo aprender y modelar relaciones complejas. Más específicamente, la función de activación en cada neurona define cómo se transforma la suma ponderada de las entradas. Usamos la función de activación *Rectified Linear Unit (ReLU)*, que es comúnmente usada en las redes de SAC debido a sus ventajas en términos de eficiencia computacional y capacidad para mitigar el problema del gradiente desvaneciente. *ReLU* se define como  $f(x) = \max(0, x)$ , donde  $x$  es la entrada a la neurona.

Después de describir las distintas capas de la red neuronal y cómo procesan las entradas, es crucial entender cómo la información fluye a través de la red y cómo se ajustan los parámetros de la misma para mejorar el rendimiento del modelo. Este proceso se lleva a cabo mediante dos mecanismos fundamentales conocidos como *feed-forward* y *backpropagation*:

- **Proceso de Feed-forward:** El proceso de *feed-forward* es el flujo secuencial de la información a través de la red, desde la capa de entrada hasta la capa de salida. Durante este paso, cada neurona en la capa de entrada recibe una parte del estado del entorno como entrada. Estas entradas se procesan luego a través de las capas ocultas de la red, donde cada neurona suma las entradas ponderadas por los pesos de las conexiones y aplica una función de activación, en este caso, la *ReLU*. Este proceso se repite en cada capa hasta alcanzar la capa de salida, donde se calculan los valores finales que determinarán las acciones del agente. En el contexto de SAC, la capa de salida produce no solo una acción específica sino también parámetros como la media y la desviación estándar de la distribución de probabilidad de las acciones, facilitando la exploración mediante una política estocástica.
- **Proceso de Backpropagation:** Una vez que la red ha producido una salida, el siguiente paso es evaluar el rendimiento de estas salidas mediante una función de pérdida, que compara las predicciones de la red con los valores deseados o reales. El error calculado en la capa de salida se propaga entonces hacia atrás a través de la red (de ahí el término *backpropagation*), ajustando los pesos de las conexiones en cada capa para minimizar este error en futuras predicciones. Este ajuste se realiza

utilizando el gradiente del error con respecto a cada peso, aplicando un algoritmo de optimización como el descenso del gradiente. Este proceso no solo mejora la precisión de la red al predecir acciones, sino que también refina las estimaciones de incertidumbre, esencial para el balance entre exploración y explotación que SAC busca optimizar.

Para el SAC se implementan tres redes, una para el actor y dos para el crítico, tal como se muestra en la Figura 26:

- **Política (Actor):** La red del actor genera acciones a partir de estados. Utiliza una red neuronal con varias capas ocultas donde cada capa puede tener entre 64 y 512 unidades. Esta red no solo predice la acción más probable en cada estado, sino que también estima la distribución de probabilidad de posibles acciones, facilitando la exploración gracias a su naturaleza estocástica.
- **Función de Valor Q (Crítico):** Se implementan dos redes  $Q$  para evaluar el valor de las acciones tomadas por la política. Similar a la red del actor, cada red  $Q$  contiene múltiples capas ocultas con una cantidad similar de unidades. La duplicidad de redes ayuda a mitigar el riesgo de sobreestimación del valor  $Q$ .

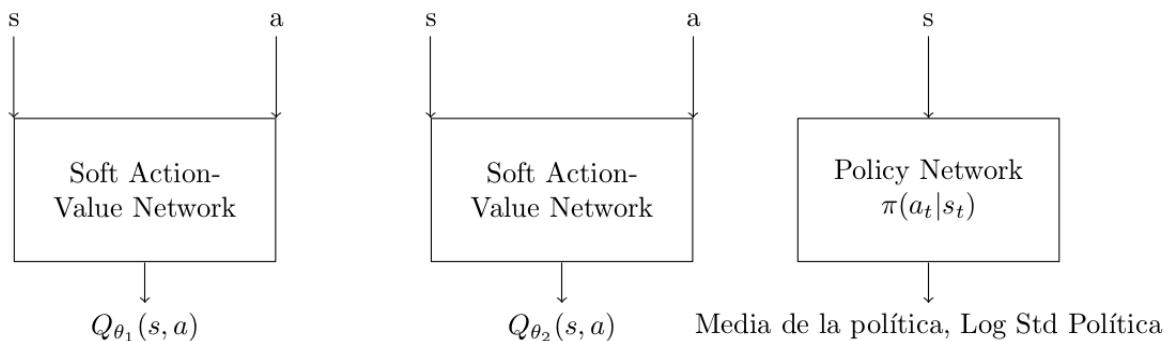


Figura 26: Arquitectura de red neuronal usada en SAC. Fuente: Elaboración propia

# 11. Integración del algoritmo Soft Actor Critic con Arduino

Para integrar el algoritmo *Soft Actor-Critic* (SAC) con un microcontrolador como el *Arduino Giga Rev1* para realizar inferencias del modelo entrenado, es necesario convertir el modelo de la política del agente a un formato compatible con el dispositivo.

## 11.1 Preparación de la Red Neuronal

Inicialmente, tenemos la red neuronal que define la política del agente, entrenada y guardada en un formato específico, en este caso en formato *PyTorch*. Dado que nos interesa realizar solo la inferencia en el dispositivo y no en continuar el entrenamiento, nos centramos en utilizar solo la parte determinista de la política, es decir, la media de la distribución de la acción sugerida por la red.

Dado que Arduino no puede ejecutar directamente modelos en formato *PyTorch*, hay que convertirlo a un formato que sea manejable por microcontroladores, específicamente usaremos la librería *TensorFlow Lite* para microcontroladores. Los pasos de conversión son los siguientes:

- 1. Convertir de PyTorch a ONNX:** Se ejecuta el código de python: `pytorch_onnx.py`
- 2. Convertir de ONNX a TensorFlow y de TensorFlow a TensorFlow Lite :** Se ejecuta el código de python: `onnx_tfLITE.py`
- 3. Convertir TensorFlow Lite a C byte array:** Se ejecuta en el terminal de linux el comando: `xxd -i sacPolicy_name.tflite > policy_model.h`

Una vez tener el modelo en formato de arreglo de bytes en C, podemos incluir el archivo `policy_model.h` en el proyecto de Arduino. Usamos la librería *TensorFlow Lite* para microcontroladores para cargar y ejecutar inferencias con este modelo.

### ¿Qué es ONNX(Open Neural Network Exchange)?

Es un formato de modelo abierto que se creó para permitir la interoperabilidad entre diferentes marcos de aprendizaje automático y herramientas. Dado que cada marco de trabajo como *TensorFlow*, *PyTorch*, *Caffe2*, y muchos otros tienen su propia forma de definir y entrenar modelos de red neuronal, *ONNX* proporciona un estándar para que estos modelos puedan ser compartidos, permitiendo la utilización de modelos entrenados en un sistema en otro sistema con diferentes tecnologías de aprendizaje automático.

Al convertir una red neuronal al formato *ONNX* se puede ver la estructura de esta en el formato *ONNX*, se muestra en la Figura 27, y se puede hacer una primera comprobación de que la conversión se ha hecho correctamente:

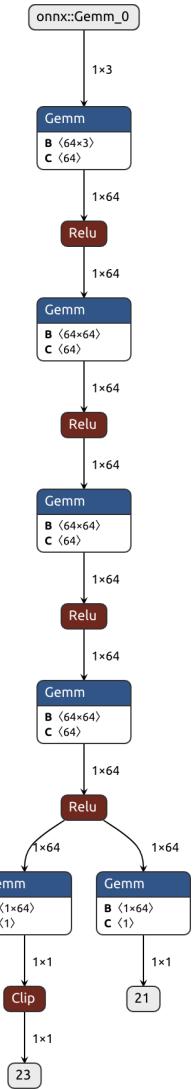


Figura 27: Visualización de la Red de Política en formato ONNX. Fuente: Elaboración propia y [32]

## 11.2 Ejecución de la inferencia

El proceso para ejecutar la inferencia del modelo comienza con la inicialización de *TensorFlow Lite* en el Arduino. La inicialización incluye la configuración de un reportador de errores, esencial para monitorear y diagnosticar problemas durante la ejecución del modelo en el dispositivo. El modelo SAC, previamente entrenado y convertido al formato *TensorFlow Lite*, se carga en el microcontrolador.

Una vez cargado el modelo, se verifica que la versión del esquema del modelo *TensorFlow Lite* coincida con la versión soportada por el intérprete de *TensorFlow Lite* en el Arduino. Esta verificación es crucial para asegurar la compatibilidad y evitar errores en la ejecución del modelo.

Para ejecutar el modelo, se configura un intérprete de *TensorFlow Lite* en el Arduino. Este intérprete utiliza un resolver de operaciones, que registra las operaciones específicas usadas

en el modelo SAC, como conexiones completamente conectadas y funciones de activación (Unidad Rectificada Uniforme,  $f(x) = \max(0, x)$ ). El intérprete gestiona la asignación de memoria para los tensores del modelo, lo cual es un paso crítico dada la memoria limitada disponible en el Arduino.

En la Figura 28 se puede ver más en detalle la configuración del modelo *TensorFlowLite*:

```
void setup_TensorFlow(){
    Serial.println("Setup TensorFlow");
    static tflite::MicroErrorReporter micro_error_reporter;
    error_reporter = &micro_error_reporter;

    model = tflite::GetModel(sacGREEN_policy_25_float32_tflite);
    if (model->version() != TFLITE_SCHEMA_VERSION) {
        TF_LITE_REPORT_ERROR(error_reporter, "Model schema version does not match TensorFlow Lite runtime.");
        return;
    }

    static tflite::MicroMutableOpResolver<4> resolver;
    resolver.AddFullyConnected();
    resolver.AddRelu();
    resolver.AddMinimum();
    resolver.AddMaximum();

    // Construymos un intérprete para ejecutar el modelo
    static tflite::MicroInterpreter *static_interpreter;
    model, resolver, tensor_arena, kTensorArenaSize);
    interpreter = &static_interpreter;

    // Asigna memoria desde tensor_arena para los tensores del modelo
    TfLiteStatus allocate_status = interpreter->AllocateTensors();
    if (allocate_status != kTfLiteOk) {
        TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors() failed.");
        return;
    }

    // Obtenemos punteros a los tensores de entrada y salida del modelo
    input = interpreter->input(0);
    output = interpreter->output(0);
}
```

Figura 28: Configuración del TensorFlow Lite para el Arduino. Fuente: Elaboración propia

La ejecución efectiva de la inferencia requiere la preparación adecuada de los datos de entrada que alimentarán al modelo. Estos datos deben ser normalizados o escalados según las especificaciones con las que fue entrenado el modelo. En este contexto los datos de entrada consisten en el espacio de estados, la posición de la esfera en el cilindro, la velocidad de la esfera y el ángulo del actuador 6 en radianes.

Con los datos de entrada preparados y cargados en los tensores de entrada del modelo, se procede a invocar el intérprete para realizar la inferencia. Esta operación computa las salidas del modelo basadas en la entrada proporcionada, aplicando la política aprendida por el algoritmo SAC.

Las salidas del modelo generalmente requieren un post-procesamiento para convertirlas en acciones concretas que pueden ser ejecutadas por el robot. En el caso de SAC, la salida es procesada a través de una función tangente hiperbólica para asegurar que las acciones están dentro de los límites aceptables y son coherentes con las capacidades físicas del sistema controlado.

La Figura 29 muestra la implementación completa de la inferencia:

```

void run_inference() {
    // Datos de entrada.
    // El modelo toma un único tensor de entrada con forma [1, 3].
    // Los tres valores son x(posición en x de la esfera), vx(velocidad en x de la esfera)
    // y acción(para la articulación 6 en radianes).
    input->data.f[0] = normalize(x, -840, 320);
    input->data.f[1] = normalize(vx, -3000, 3000);

    input->data.f[2] = action;
    Serial.println(input->data.f[0], 9);
    Serial.println(input->data.f[1], 9);
    // We run inference.
    Serial.println("Running Inference");
    TfLiteStatus invoke_status = interpreter->Invoke();
    Serial.println("After we invoke the interpreter");
    if (invoke_status != kTfLiteOk) {
        TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed.");
        return;
    }

    // Accedemos a los tensores de salida
    float meanVal = interpreter->output(0)->data.f[0];

    float sampled_action = tanh(meanVal);
    float action_range = 0.27;
    action = action_range * sampled_action;

    Serial.print("The action is: ");
    Serial.println(action);
}

```

Figura 29: Implementación de la función de inferencia de la política del agente en Arduino. Fuente:  
Elaboración propia

## 11.3 Planificador Cíclico

En la integración del algoritmo del SAC con Arduino, un aspecto crítico es la implementación de un planificador cíclico para gestionar la inferencia y la interacción con el robot UR3e y otros componentes del sistema. Este planificador se encarga de ejecutar varias funciones clave en un ciclo continuo, asegurando que todas las operaciones se realicen de manera eficiente y coordinada.

### Funciones Principales del Planificador Cíclico:

- **Ejecución de Inferencia (run\_inference):** Esta función es crucial para procesar los datos sensoriales actuales y determinar la acción adecuada del robot según el modelo de política del SAC. La inferencia se realiza cada 200 milisegundos, asegurando respuestas rápidas a las condiciones cambiantes del entorno.
- **Comunicación con UR3e (sendData2UR3):** Posterior a la inferencia, esta función envía la acción determinada al robot UR3e. Las acciones son ajustadas para asegurarse de que se mantengan dentro de los límites operativos del robot y se sincronicen con las necesidades del momento, enviando comandos cada 200 milisegundos.
- **Medición de Distancia (readToFDistance):** Utiliza un sensor láser para medir continuamente la distancia de objetos relevantes, actualizando el estado del entorno

cada 32 milisegundos. Estos datos son esenciales para alimentar el modelo de inferencia con información precisa.

- **Visualización en Giga Display Shield (plotDisplay):** Esta función controla el Giga Display Shield para mostrar animaciones que reflejan la posición de la esfera y la orientación del robot. Se actualiza cada 20 milisegundos, proporcionando una retroalimentación visual inmediata sobre la operación del sistema.

El planificador cíclico coordina meticulosamente estas funciones, asegurando que cada una se ejecute en su intervalo de tiempo designado sin interferencias. Por ejemplo, la función de inferencia y comunicación se realiza de manera que cada acción del robot esté perfectamente temporizada con las actualizaciones de los datos de inferencia.

## 12. Pruebas y ajustes

### 12.1 Configuración Experimental

Cada red neuronal, tanto del crítico como del actor, se ha configurado con dos (crítico), respectivamente tres (actor), capas ocultas para mantener un equilibrio entre capacidad de aprendizaje y eficiencia computacional. El entrenamiento se lleva a cabo utilizando diferentes esferas—verde, golf y metacrilato—para evaluar la versatilidad y adaptabilidad del algoritmo frente a variaciones en las condiciones físicas del objeto balanceado.

Cada sesión de entrenamiento está estructurada en episodios, con cada episodio representando una secuencia completa de intentos de balanceo. Al inicio de cada episodio, la esfera se posiciona aleatoriamente en un punto del cilindro, y cada episodio consiste de 200 time steps. Dado que cada acción de control se ejecuta cada 200 milisegundos, cada episodio representa un total de 40 segundos de tiempo real, permitiendo la colección de 200 experiencias por episodio.

La posición deseada de la esfera en el cilindro se establece en el rango de 0.58 a 0.60, correspondientes a la mitad de su longitud aproximadamente y un poco desplazada respecto a la ubicación del display. Este rango se selecciona para compensar el ruido inherente del sensor de distancia y permitir una tolerancia en la medición que evite penalizaciones excesivas por fluctuaciones menores alrededor del punto de equilibrio seleccionado.

Los entrenamientos se llevan a cabo de forma individual para cada tipo de esfera, registrando distintos comportamientos y ajustes necesarios para optimizar su control en cada caso. El entrenamiento se lleva a cabo en el entorno del laboratorio con el brazo robótico real UR3e. En la Tabla 18 se presentan los hiperparámetros y las penalizaciones de la función de recompensa (*reward*) que se han utilizado para el entrenamiento de cada esfera:

Hiperparámetro	Valor
Learning Rate	$3 \times 10^{-4}$
Discount ( $\gamma$ )	0.95
Number of Hidden Layers (All Networks)	3
Number of Hidden Units per Layer	128
Number of Samples per Minibatch	256
Entropy Target	-3
Target Smoothing Coefficient ( $\tau$ )	0.02
Target Update Interval	2
Gradient Steps	1
Distance Penalty Coefficient	1.0
Velocity Penalty Coefficient	0.1
Fixed Penalty	-2.0

Tabla 18: Tabla de hiperparámetros para el entrenamiento. Fuente: Elaboración propia

A continuación se hace un análisis de las pérdidas de cada red neuronal y de la ganancia acumulada:

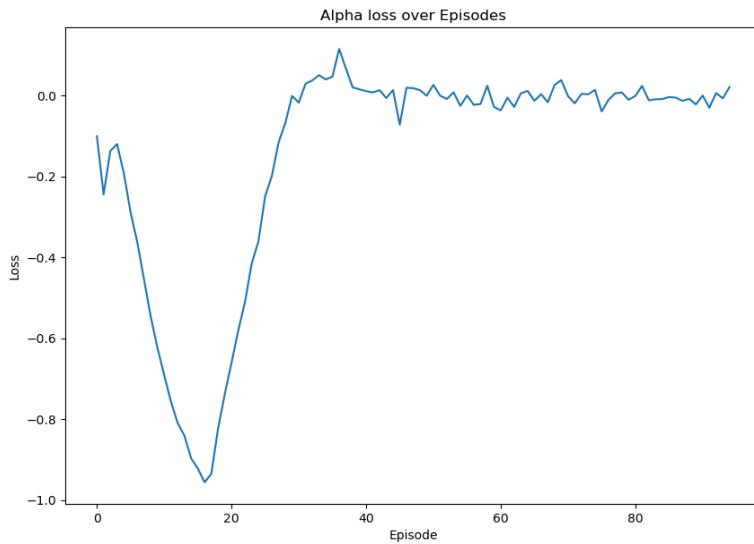
- **Pérdida Alpha:** La pérdida *alpha* está relacionada con el ajuste del parámetro de temperatura en el algoritmo SAC, que equilibra la explotación de lo aprendido contra la exploración de nuevas acciones. En términos simples, un valor alto de *alpha* impulsa al modelo a explorar más, es decir, a probar acciones variadas y potencialmente no óptimas, mientras que un valor bajo fomenta la explotación, centrando sus decisiones en las acciones que ya sabe que son efectivas. La pérdida *alpha* ayuda a ajustar dinámicamente este parámetro durante el entrenamiento para optimizar el aprendizaje. La idea es encontrar un equilibrio adecuado que permita al modelo explorar suficientemente el espacio de acción sin perder la capacidad de capitalizar sobre las estrategias efectivas que ha descubierto.
- **Pérdida del crítico (Pérdida Q):** Hace referencia a la diferencia entre los valores *Q* actuales predichos por la red neuronal y los valores *Q* objetivo que se calculan durante el entrenamiento. Los valores *Q* representan la expectativa de la recompensa total que se puede obtener, empezando en un estado dado y tomando una acción particular según una política específica. La función de pérdida *Q* busca minimizar esta diferencia, lo cual es indicativo de que el modelo está aprendiendo correctamente a estimar el valor de sus acciones en diferentes estados. Una reducción en la pérdida *Q* a lo largo del entrenamiento suele indicar que el modelo está mejorando su capacidad de predecir con precisión la recompensa futura esperada, llevando a decisiones más informadas y, en última instancia, a un comportamiento más optimizado.
- **Pérdida de la política:** La pérdida de la política, es esencialmente una medida de cuán bien la política del actor está maximizando las recompensas futuras, junto con la entropía de la política para mantener la exploración de nuevas acciones. La política del actor en el SAC se encarga de decidir qué acción tomar en un estado dado, y esta decisión debe equilibrar entre explorar nuevas acciones y explotar las que ya sabe que son buenas.
- **Recompensa Acumulada:** El *reward* acumulado es el total de recompensas obtenidas por el agente durante un episodio o a lo largo del entrenamiento. En la práctica, es una medida directa del desempeño del agente: cuanto mayor sea la recompensa acumulada, más efectivo es el agente en realizar la tarea asignada. En un gráfico de recompensas por episodios, una tendencia ascendente o una estabilización en un nivel alto indica que el modelo está actuando de manera efectiva y consistente, maximizando las recompensas y minimizando los errores o las acciones subóptimas.

## 12.2 Esfera Verde

El entrenamiento de la esfera verde se ha realizado a través de 95 episodios, acumulando un total de 63 minutos de tiempo real de experimentación. A continuación, se analizan los datos recopilados, para entender la efectividad y eficiencia del algoritmo empleado, así como su comportamiento a lo largo del tiempo.

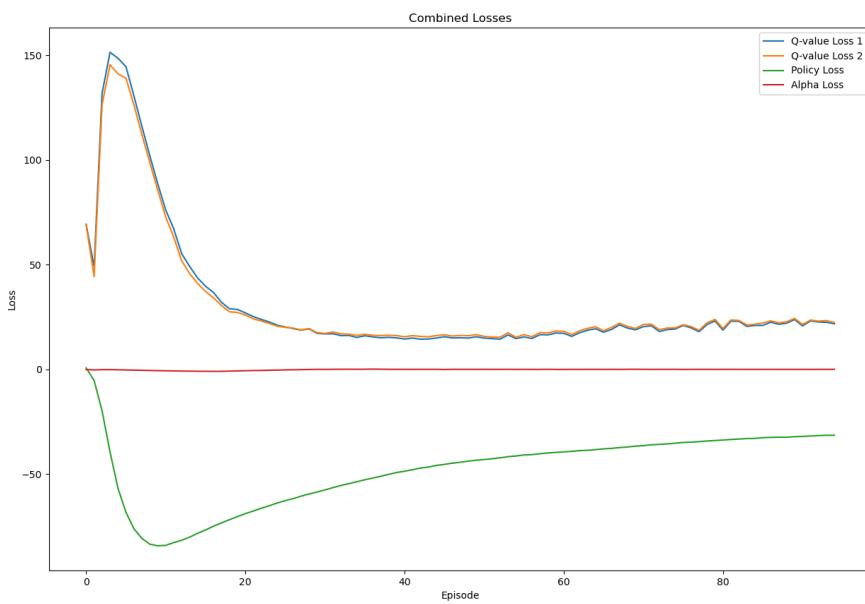
Observando la gráfica de la Figura 30, se nota un inicio volátil con un valor inicial negativo, lo que indica una preferencia inicial por la exploración. Esto es crucial al comienzo del entrenamiento, ya que permite al modelo explorar un amplio espacio de acción y no estancarse en mínimos locales.

Después de los primeros 20 episodios, se observa un aumento significativo de la pérdida, alcanzando valores cercanos a -0.6. Este pico puede interpretarse como un ajuste crítico en la política de aprendizaje del algoritmo, donde quizás se redujo temporalmente la exploración a medida que el modelo empezaba a aprender la dinámica del problema. Posteriormente, la curva se estabiliza y oscila ligeramente alrededor de -0.2, sugiriendo un equilibrio más estable entre la exploración y explotación, y una convergencia gradual hacia una política efectiva.



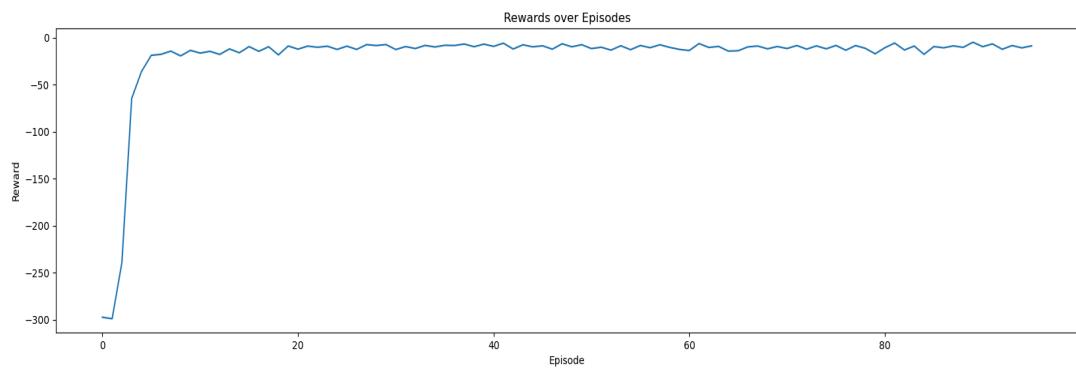
*Figura 30: Alpha loss esfera Verde. Fuente: Elaboración propia*

Las pérdidas combinadas incluyen la pérdida de valor  $Q$ , la pérdida de política, y la pérdida Alpha. La pérdida  $Q$  muestra una reducción dramática inicial, estabilizándose rápidamente, lo que sugiere una rápida convergencia del modelo en su capacidad de evaluar las acciones en términos de la maximización del retorno esperado. La pérdida de política demuestra una disminución significativa y una estabilización posterior, indicando que la estrategia de actuación se ha afinado efectivamente para el entorno específico y las acciones posibles. Todo esto se puede observar en la Figura 31:



*Figura 31: Loss críticos, actor y alpha esfera Verde. Fuente: Elaboración propia*

La gráfica de recompensas acumuladas, Figura 32, muestra una mejora significativa en las recompensas desde el inicio hasta el estabilizarse en un nivel alto y consistente. El inicial aumento en las recompensas indica que el modelo rápidamente aprendió una estrategia efectiva para mantener la esfera en equilibrio sobre el cilindro. La estabilización de las recompensas en valores altos es un buen indicador de que la política aprendida es efectiva y que el modelo es capaz de repetir su desempeño de manera consistente.



*Figura 32: Ganancia acumulada, actor y alpha esfera Verde. Fuente: Elaboración propia*

## 12.3 Esfera de Metacrilato

Para la esfera de metacrilato, se llevó a cabo el entrenamiento durante 80 episodios, totalizando aproximadamente 53 minutos en tiempo real. Este entrenamiento evalúa la capacidad del algoritmo para adaptarse a las propiedades físicas únicas del metacrilato, que son diferentes a la esfera verde e influye de manera diferente la dinámica del equilibrio, incluyendo su peso y textura superficial.

La gráfica de la pérdida *alpha*, Figura 33, en este caso muestra una volatilidad inicial significativa, que se estabiliza hacia la mitad del entrenamiento. Este patrón sugiere que el algoritmo pasa por una fase de exploración intensiva antes de encontrar un nivel de explotación más eficaz. La pérdida *alpha* negativa indica que el algoritmo ajustó efectivamente el parámetro de temperatura para optimizar la política de actuación.

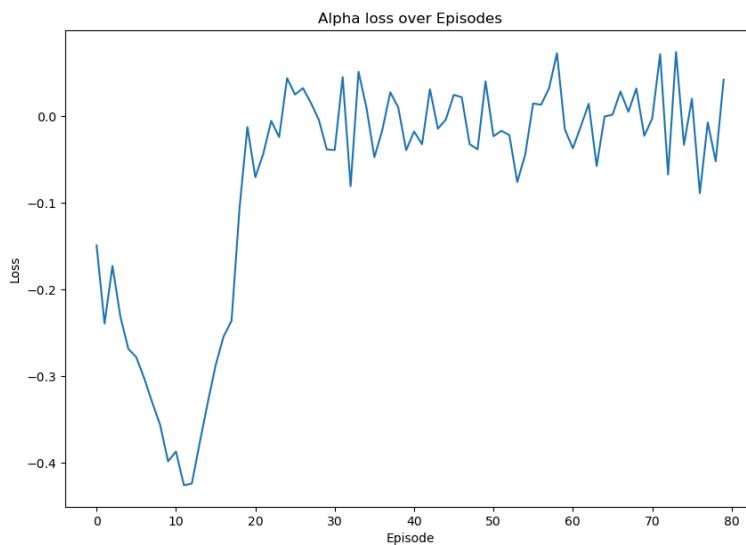


Figura 33: Alpha loss esfera de Metacrilato. Fuente: Elaboración propia

En la gráfica de pérdidas combinadas, Figura 34, la pérdida de *Q*-valor inicialmente muestra un pico significativo, lo cual puede ser indicativo de una corrección importante en las estimaciones de *Q*-valor al principio del entrenamiento. Esto es común en ambientes donde el agente necesita ajustar sus expectativas iniciales bastante imprecisas sobre el entorno. La pérdida de política también muestran una mejora continua, convergiendo hacia un valor más estable, lo que sugiere que el modelo ha comenzado a converger hacia una política óptima.

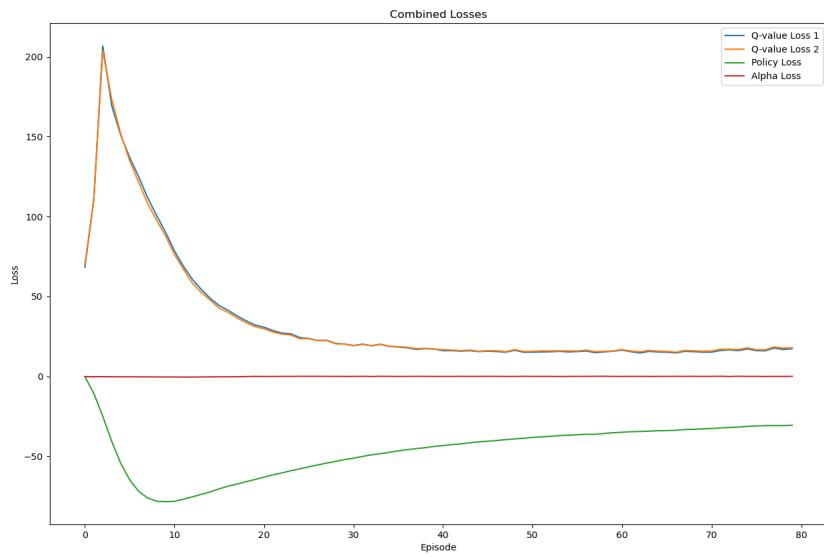


Figura 34: Loss críticos, actor y alpha esfera de Metacrilato. Fuente: Elaboración propia

Finalmente, la gráfica de recompensas, Figura 35, acumuladas muestra una mejora significativa después de los episodios iniciales, donde el agente experimenta las mayores pérdidas. Después de este periodo inicial, las recompensas se estabilizan con una ligera tendencia ascendente, indicando que el agente está logrando mantener el equilibrio de la esfera de metacrilato.

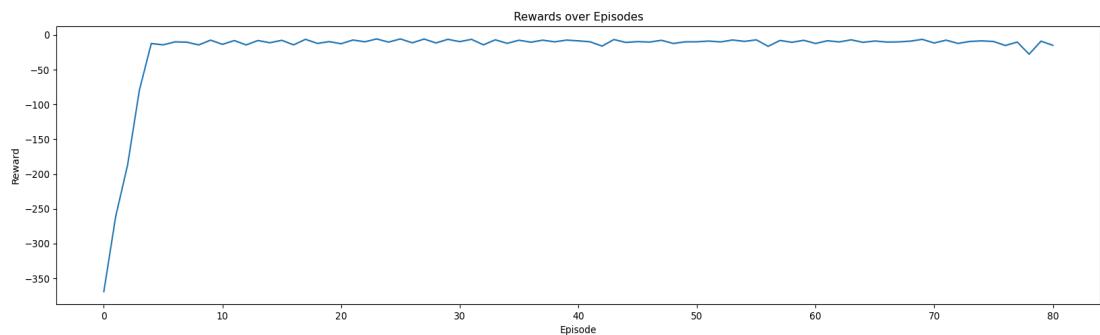


Figura 35: Ganancia acumulada, actor y alpha esfera de Metacrilato. Fuente: Elaboración propia

## 12.4 Esfera de Golf

El entrenamiento de la esfera de golf se realizó durante 85 episodios, correspondientes a 57 minutos de tiempo real. La evaluación de las gráficas de pérdida *alpha*, pérdidas combinadas y recompensas acumuladas revela las características únicas del comportamiento del agente de cómo se adapta a la nueva dinámica.

La gráfica de pérdida *alpha*, Figura 36, muestra una notable característica inicial: un descenso pronunciado seguido de una recuperación igualmente aguda, estabilizándose en un rango más bajo. Esta *V* profunda podría indicar un ajuste significativo en la política del

modelo que, después de una exploración inicial extrema, encontró un equilibrio más efectivo entre la exploración y la explotación. Posteriormente, el comportamiento oscilante pero constante en un rango menor sugiere que el modelo ha ajustado su tasa de exploración a un nivel que optimiza su rendimiento sin grandes variaciones, lo que implica un ajuste eficaz a la dinámica del entorno de la esfera de golf.

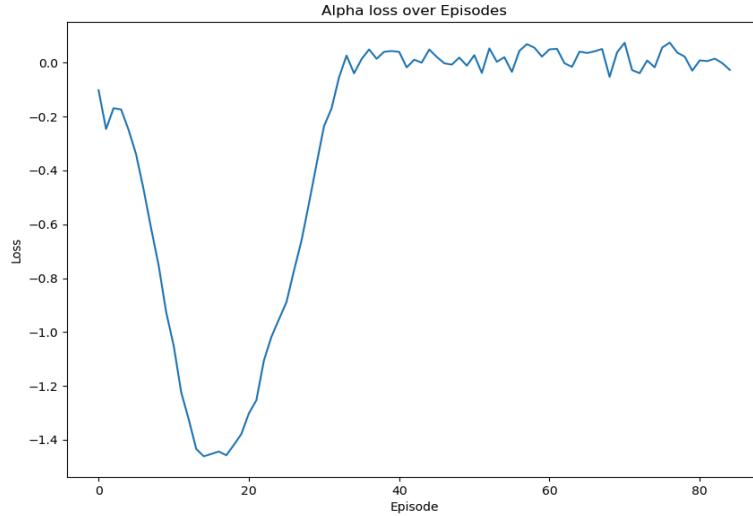


Figura 36: Alpha loss esfera de Golf. Fuente: Elaboración propia

Las pérdidas combinadas, que incluyen las pérdidas  $Q1$ ,  $Q2$ , y de política, además de la pérdida  $\alpha$ , muestran una reducción significativa en las primeras etapas del entrenamiento, estabilizándose hacia el final. Este patrón sugiere que el modelo ha aprendido a predecir los valores  $Q$  con precisión y ha optimizado su política de acciones de manera eficiente. La estabilización de las pérdidas  $Q$  y de política en valores cercanos a cero refleja que los ajustes en los pesos de la red neuronal han llegado a un punto en el cual el agente realiza evaluaciones consistentes y fiables del entorno. Todo esto se puede ver la Figura 37:

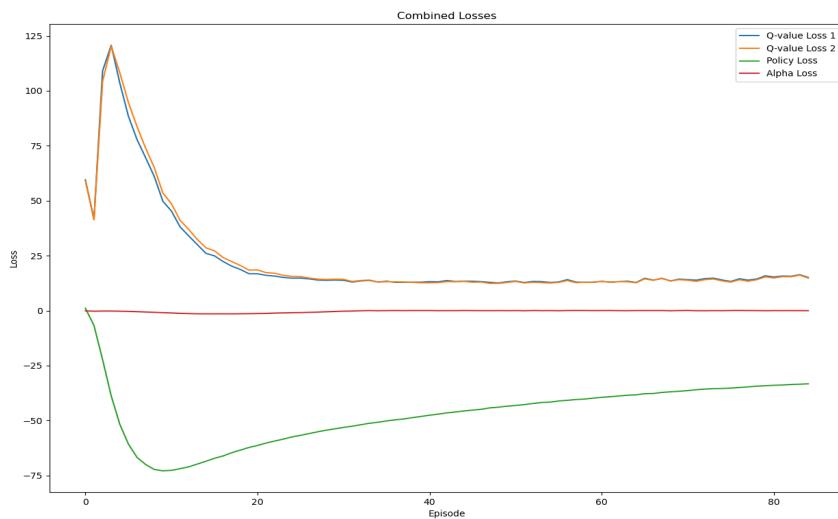


Figura 37: Loss criticos, actor y alpha esfera de Golf. Fuente: Elaboración propia

Finalmente, la gráfica de recompensa acumulada, Figura 38, muestra una recuperación rápida desde los valores iniciales negativos, alcanzando un nivel de estabilidad. Este rápido incremento y la posterior estabilidad en la recompensa acumulada son indicativos de que el agente ha aprendido a realizar las acciones necesarias para mantener la bola en equilibrio sobre la plataforma de manera efectiva. La planicie en la parte superior de la gráfica sugiere que el agente ha alcanzado un desempeño óptimo bajo las condiciones actuales del entrenamiento.

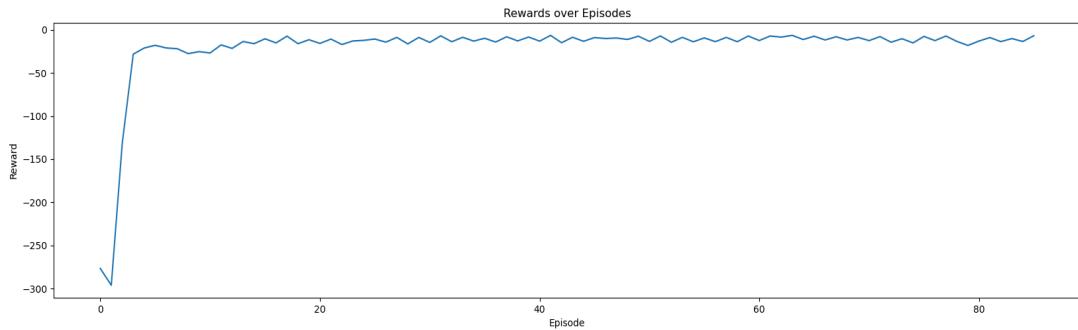


Figura 38: Ganancia acumulada, actor y alpha esfera de Golf. Fuente: Elaboración propia

## 12.5 Análisis y Conclusiones

En esta sección, se evalúan los efectos de los cambios en los hiperparámetros y en la función de recompensa para las distintas esferas utilizadas en el entrenamiento del algoritmo *Soft Actor-Critic (SAC)*. Estos ajustes son críticos, ya que pequeñas variaciones en los hiperparámetros pueden tener un impacto significativo en la eficacia del aprendizaje y en la capacidad del modelo para adaptarse a diferentes dinámicas del entorno. La Tabla 19 muestra una comparativa de los hiperparámetros modificados para cada tipo de esfera.

Hiperparámetro	Esferas		
	Configuración 1	Configuración 2	Configuración 3
Batch Size(Samples per Minibatch)	256	256	256
Hidden Dimension	128	128	128
Discount	0.99	0.99	0.99
Target Entropy	-1	-2	-4
Target Update Interval	1	2	2
Distance Penalty Coefficient	1.0	1.0	1.0
Velocity Penalty Coefficient	0.1	0.1	0.1
Target Zone Penalty	3.0	3.0	3.0
Fixed Penalty	-2.0	-2.0	-2.0

Tabla 19: Valores de Hiperparámetros para las Diferentes Esferas. Fuente: Elaboración propia

Los cambios principales en los hiperparámetros incluyen ajustes en el *Discount*, *Target Update Interval*, y la introducción del *Target Zone Penalty*.

Se ha introducido el *Target Zone Penalty*, un componente de recompensa que otorga un puntaje positivo si la esfera se mantiene dentro de un rango de equilibrio predefinido, que

como se mencionó antes, está entre 0.58 y 0.60. Este incentivo está diseñado para reforzar específicamente el comportamiento de mantener la esfera centrada, aumentando las recompensas cuando el agente logra este objetivo crítico. Fuera de este rango, la recompensa se calcula, como se ha presentado en la sección 10.2.1, una penalización por la distancia y la velocidad,  $reward = -(distance\_penalty + velocity\_penalty)$ , fomentando al agente a corregir su posición y velocidad hacia el rango objetivo. En cambio, si está en el rango de equilibrio se calcula de esta forma  $reward = target\_zone\_penalty$ . Este enfoque dual en la función de recompensa ayuda a crear un balance entre alcanzar y mantener el punto de equilibrio, y ajustar dinámicamente el comportamiento del agente cuando se desvía de este punto.

En la siguiente tabla se presentan los cambios realizados en otros parámetros, enfocados específicamente en la esfera de color verde. Este análisis permite probar variaciones adicionales de los parámetros y sus efectos correspondientes. Los cambios más relevantes se pueden consultar en la Tabla 20:

Hiperparámetro	Esfera Verde			
	Configuración 1	Configuración 2	Configuración 3	Configuración 4
Batch Size (Samples per Minibatch)	256	256	64	64
Hidden Dimension	128	128	128	64
Discount	0.99	0.99	0.99	0.99
Target Entropy	-1	-1	-1	-1
Target Update Interval	1	1	1	1
Distance Penalty Coefficient	1.0	1.0	1.0	1.0
Velocity Penalty Coefficient	1.0	0.1	0.1	0.1
Target Zone Penalty	3.0	4.0	1.0	1.0
Fixed Penalty	-2.0	-3.0	-1.0	-1.0

Tabla 20: Valores de hiperparámetros para diferentes configuraciones de la esfera Verde. Fuente:  
Elaboración propia

### Ajustes de Hiperparámetros:

- **Discount Factor:** Se ha observado en el entorno real que un valor alto del factor de descuento (0.99) resultaba en un comportamiento oscilante de la esfera alrededor del punto de equilibrio. Esto se atribuye al ruido inherente del sensor que introduce incertidumbre en la estimación del estado futuro, haciendo que un alto descuento propague este ruido a lo largo de múltiples estados futuros. Un factor de descuento reducido (menor que 0.99, por ejemplo 0.95 se ajusta muy bien) limita esta propagación, ayudando al modelo a enfocarse en recompensas a corto plazo y estabilizar más rápidamente la esfera.
- **Penalización por Velocidad:** La dinámica del sistema mostró que penalizaciones altas por velocidad exacerbaban las reacciones del controlador a variaciones menores en la velocidad de la esfera, atribuibles al ruido. Reducir esta penalización permitió al modelo operar de manera más fluida y menos reactiva, facilitando un control más estable y predecible.

- **Target Update Interval:** Actualizar la red cada episodio con dos iteraciones equilibra de manera efectiva la necesidad de adaptarse rápidamente a la dinámica del entorno y evitar el riesgo de *overfitting* por sobreexplotación de experiencias recientes. Una frecuencia de actualización más alta resultaba en convergencia rápida pero a riesgo de volatilidad en el aprendizaje, mientras que intervalos más largos retardaban el aprendizaje sin ofrecer beneficios significativos en estabilidad.
- **Target Entropy:** Ajustar la entropía objetivo según el tipo de esfera y sus características específicas (como el material y el peso) demostró ser fundamental. Para la esfera verde, una entropía objetivo demasiado baja (por ejemplo: -7.0) limitaba la exploración necesaria para escapar de mínimos locales en el aprendizaje, mientras que una entropía demasiado alta fomentaba una exploración excesiva que impedía la convergencia. Se estableció que un valor de entropía entre -1.0 y -2.0 es óptimo para balancear exploración y explotación efectivas.
- **Número de Neuronas por Capa:** Experimentos con diferentes configuraciones de la red revelaron que un número moderado de neuronas (entre 64 y 128 por capa) proporciona el mejor equilibrio entre capacidad de aprendizaje y eficiencia computacional. Valores más altos no mejoraban significativamente el aprendizaje y aumentaban la carga computacional, mientras que valores más bajos prolongaban innecesariamente el tiempo de entrenamiento.
- **Batch Size:** El tamaño del lote afecta significativamente la estabilidad del gradiente durante el entrenamiento. Un tamaño de lote mayor proporciona una estimación más estable del gradiente, lo que es crucial en entornos ruidosos. Se observó que lotes entre 64 y 256 ofrecían la mejor combinación de estabilidad y eficiencia en la actualización de la red.

# 13. Conclusiones, conocimientos adquiridos y trabajo futuro

## 13.1 Conclusiones

El desarrollo e implementación del sistema de *ball balancing* con el robot UR3e, aplicando técnicas de optimización basadas en inteligencia artificial, ha demostrado ser un proyecto complejo pero muy interesante. El sistema ha probado ser capaz de adaptarse eficazmente a las incertidumbres inherentes de un entorno operativo real y ha destacado la importancia de una configuración meticulosa y ajustada del algoritmo *Soft Actor-Critic (SAC)*.

El proyecto ha validado la capacidad del microcontrolador Arduino para ejecutar la inferencia de la red neuronal en tiempo real, facilitando así una respuesta rápida y adecuada a los cambios dinámicos del entorno. La implementación de esta tecnología en *hardware* tangible ha proporcionado una valiosa comprensión de la interacción entre los componentes de *software* y *hardware* y ha subrayado la viabilidad de soluciones inteligentes en aplicaciones de la vida real.

A través de la experimentación, se ha observado cómo los ajustes en los parámetros del algoritmo, como el factor de descuento y la entropía objetivo, son cruciales para el equilibrio entre la exploración de nuevas estrategias y la explotación de las ya aprendidas. Este balance es esencial para el aprendizaje autónomo en situaciones reales donde las condiciones pueden cambiar de manera impredecible.

Además, el proyecto ha servido como una excelente plataforma para explorar y demostrar las capacidades del robot UR3e en tareas que requieren precisión y estabilidad, elementos críticos en aplicaciones industriales y de servicio.

En resumen, se destacan varios aspectos fundamentales que demuestran el éxito y los retos del proyecto:

- **Implementación Exitosa:** El sistema ha demostrado ser efectivo en el equilibrio dinámico de una esfera sobre un cilindro, usando algoritmos avanzados de aprendizaje por refuerzo que permiten una adaptación y respuesta en tiempo real a las condiciones del entorno. La utilización del SAC ha permitido una mejora continua del comportamiento del robot mediante el ajuste de la política de control basada en el aprendizaje autónomo y la retroalimentación inmediata del entorno.
- **Optimización y Ajustes de Parámetros:** La meticulosa optimización de parámetros como el factor de descuento y la entropía objetivo ha sido crucial para lograr un equilibrio óptimo entre exploración y explotación, lo cual ha mejorado

significativamente la eficacia y eficiencia del aprendizaje automático en la aplicación real.

- **Desafíos y Soluciones Tecnológicas:** Se han enfrentado desafíos significativos, particularmente en la transición de teorías y simulaciones a la implementación práctica. La elección de utilizar un microcontrolador Arduino para ejecutar la inferencia de la red neuronal ha destacado la viabilidad y la importancia de los sistemas embebidos en aplicaciones robóticas modernas, demostrando que es posible llevar soluciones de inteligencia artificial de alto nivel a plataformas de bajo costo y recursos limitados.

## 13.2 Conocimientos adquiridos y limitaciones

### Conocimientos Adquiridos:

- Integración de sistemas robóticos e inteligencia artificial: A través del proyecto, se han adquirido habilidades significativas en la integración de componentes mecánicos y electrónicos con un algoritmo avanzado de aprendizaje por refuerzo. Esto incluye desde la configuración física del robot hasta la implementación de políticas de control en tiempo real.
- Dominio técnico del robot UR3e: Se ha profundizado a través del uso de *URScript*, el cual ha sido esencial para entender cómo funciona el robot y gestionar su operación. Este lenguaje de programación específico permite un control detallado y adaptable del robot, lo que es crucial para ejecutar tareas complejas de forma precisa. Además, la familiarización con los diversos modos de comunicación y los puertos de interfaces ha facilitado la sincronización efectiva de todas las operaciones del robot con otros sistemas, asegurando una coordinación fluida y eficiente en entornos dinámicos.
- Aplicación de técnicas de aprendizaje por refuerzo en tiempo real: Se ha profundizado en el uso de técnicas avanzadas de aprendizaje por refuerzo, específicamente el algoritmo **Soft Actor-Critic**, aplicándolas en un contexto de control en tiempo real con restricciones de tiempo estrictas y precisas.
- Desarrollo y ajuste de modelos de IA para *hardware* específico: Se han desarrollado habilidades en la adaptación de modelos de inteligencia artificial para que sean compatibles con plataformas de *hardware* limitadas, como el microcontrolador Arduino, lo cual incluyó la conversión de modelos complejos a versiones más ligeras que pueden ser ejecutadas en dispositivos con recursos limitados.
- Gestión de los tiempos de ejecución y sincronización de sistemas en tiempo real: El proyecto proporcionó experiencia práctica en la gestión del tiempo de ejecución de las tareas en sistemas en tiempo real y la importancia de la sincronización precisa entre la captura de datos sensoriales, procesamiento de decisiones y actuación mecánica.

- Análisis de datos y ajuste de parámetros: Se ha mejorado la capacidad para analizar volúmenes de datos de rendimiento del sistema, ajustar parámetros en modelos de aprendizaje por refuerzo y evaluar de manera crítica los resultados de dichos ajustes para mejorar el rendimiento del sistema.

#### **Limitaciones:**

- Desafíos de la implementación física: Se encontraron limitaciones relacionadas con la precisión y el retardo de los sensores, así como con la capacidad de actuación del robot, que afectaron la precisión del equilibrio de la esfera.
- Restricciones de *hardware*: El *hardware* disponible, especialmente en términos de capacidad de procesamiento y memoria del Arduino, impuso restricciones significativas al tamaño y complejidad de los modelos de inteligencia artificial que se podían implementar directamente en el robot.
- Complejidad del aprendizaje por refuerzo en entornos reales: La aplicación de técnicas de aprendizaje por refuerzo en entornos reales reveló complicaciones adicionales, como la necesidad de manejar variaciones impredecibles en el entorno y asegurar la estabilidad del aprendizaje frente a perturbaciones físicas y modelos dinámicos.
- Sincronización de componentes: Se enfrentaron desafíos significativos en la sincronización de los componentes del sistema para garantizar que el tiempo de respuesta fuera adecuadamente rápido para mantener el control en tiempo real, especialmente durante la transmisión de comandos y la recepción de datos del sensor.

### 13.3 Trabajo Futuro

- **Implementación de mbed RTOS:** Para optimizar la gestión de tareas y el tiempo de respuesta, se podría implementar un scheduler de tiempo real más potente, como *mbed RTOS*. Esta herramienta permitirá una mejor asignación de recursos y gestión de procesos simultáneos, lo que es crucial para aplicaciones que demandan alta precisión y eficiencia en tiempo real.
- **Uso de la Interfaz RTDE:** Implementar la *Real-Time Data Exchange (RTDE)* para mejorar la comunicación con el robot UR3e. La *RTDE* permitiría una sincronización más precisa y eficiente entre el robot y los sistemas externos, reduciendo la latencia y mejorando la precisión del control en tiempo real.
- **Integración de Múltiples Esferas:** Desarrollar y entrenar el algoritmo SAC para gestionar múltiples esferas simultáneamente. Esto implicaría añadir estados

adicionales al modelo para que pueda manejar las dinámicas y relaciones entre varias esferas, aumentando la complejidad del sistema.

- **Optimización del Control Dinámico:** Mejorar los algoritmos para permitir ajustes dinámicos más rápidos y precisos, adaptando el sistema a cambios rápidos en el entorno.
- **Uso de Múltiples Joints del Robot:** Investigar la posibilidad de utilizar más articulaciones del robot para proporcionar un control más dinámico y versátil del *beam*. Esto podría permitir movimientos más complejos y adaptativos que mejoren la capacidad del sistema para mantener el equilibrio bajo condiciones variables.
- **Movilidad del Robot:** Integrar la capacidad de mover el robot mientras mantiene el equilibrio del cilindro.
- **Inferencia en Arduino Mega o Uno:** Probar la viabilidad de realizar inferencias de SAC en microcontroladores con recursos aún más limitados que el *Arduino Giga* como en el *Arduino Mega* o en el *Arduino Uno*. Esto requerirá la adaptación y compilación de *TensorFlow Lite* para arquitecturas *AVR(Advanced RISC Machine)*, optimizando el código para asegurar que el rendimiento del sistema sea viable en plataformas de hardware con capacidades restringidas.
- **Desarrollo de Algoritmos Eficientes:** Crear versiones más ligeras y eficientes del SAC que puedan operar dentro de las limitaciones de memoria y procesamiento de dispositivos de menor capacidad.

# Referencias

- [1] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. International Journal of Robotics Research, 32(4), 1238-1274, 2013. URL: [https://www.ri.cmu.edu/pub\\_files/2013/7/Kober\\_IJRR\\_2013.pdf](https://www.ri.cmu.edu/pub_files/2013/7/Kober_IJRR_2013.pdf)
- [2] Universal Robots. Robot UR3e [en línea]. s.f. Disponible en: <https://www.universal-robots.com/es/productos/robot-ur3/>
- [3] Universal Robots. Manual del usuario del Robot UR3e. [en línea]. Disponible en: [https://s3-eu-west-1.amazonaws.com/ur-support-site/69073/99436\\_UR3e\\_User\\_Manual\\_es\\_Global.pdf](https://s3-eu-west-1.amazonaws.com/ur-support-site/69073/99436_UR3e_User_Manual_es_Global.pdf)
- [4] Universal Robots. Arquitectura de software URCap. [en línea]. Disponible en: <https://www.universal-robots.com/articles/ur/urplus-resources/urcap-ur-software-architecture/>
- [5] Arduino. Arduino Giga R1 WiFi [en línea]. Disponible en: <https://store.arduino.cc/products/giga-r1-wifi>
- [6] STMicroelectronics. STM32H747XI Microcontroller [en línea]. Disponible en: <https://www.st.com/en/microcontrollers-microprocessors/stm32h747xi.html>
- [7] Murata. Type 1DX Wi-Fi Bluetooth Connectivity Module [en línea]. Disponible en: <https://www.murata.com/products/connectivitymodule/wi-fi-bluetooth/overview/lineup/type1dx>
- [8] STMicroelectronics. VL53L0X Time-of-Flight Distance Sensor [en línea]. Disponible en: <https://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html>
- [9] Llamas, Luis. "Sensor de distancia VL53L0X con Arduino" [en línea]. Disponible en: <https://www.luisllamas.es/arduino-sensor-distancia-vl53l0x/>
- [10] Audsley, N. C., Burns, A., Davis, R., Ince, D., and Wellings, A. (1993). The deadline monotonic scheduling algorithm. Real-Time Systems, 5(2), 175-198.
- [11] Analytics Vidhya. Introduction to Reinforcement Learning for Beginners [en línia]. 21 febrero 2021. Disponible a: <https://www.analyticsvidhya.com/blog/2021/02/introduction-to-reinforcement-learning-for-beginners/>
- [12] Microsoft. What is Kanban? [en línea]. s.f. Disponible en: <https://learn.microsoft.com/en-us/devops/plan/what-is-kanban/>
- [13] Anderson, D. J. Revisando los principios y prácticas generales del método Kanban [en

línea]. s.f. Disponible en:

<https://djaa.com/revisando-los-principios-y-practicas-generales-del-metodo-kanban/>

[14] Glassdoor. Sueldo: Gestor De Proyectos en España en 2023 [en línea]. 2023. Disponible en: [https://www.glassdoor.es/Sueldos/gestor-de-proyectos-sueldo-SRCH\\_KO0,19.htm](https://www.glassdoor.es/Sueldos/gestor-de-proyectos-sueldo-SRCH_KO0,19.htm)

[15] Glassdoor. Sueldo: Desarrollador De Software en España en 2023 [en línea]. 2023. Disponible en:

[https://www.glassdoor.es/Sueldos/desarrollador-de-software-sueldo-SRCH\\_KO0,25.htm](https://www.glassdoor.es/Sueldos/desarrollador-de-software-sueldo-SRCH_KO0,25.htm)

[16] Jobted. ¿Cuánto Cobra un Técnico de Laboratorio? (Sueldo 2023) [en línea]. 2023. Disponible en:

<https://www.jobted.es/salario/t%C3%A9cnico-laboratorio#:~:text=El%20sueldo%20medio%20de%20un,de%20Laboratorio%20actualizados%20a%202023>

[17] Business Insider España. Cuánto cobra un profesor de universidad en España en 2022 [en línea]. 2022. Disponible en:

<https://www.businessinsider.es/cuanto-gana-profesor-universidad-espana-1119435>

[18] UPC Universitat Politècnica de Catalunya. Taules retributives del personal d'administració i serveis any 2023 [en línea]. 2023. Disponible en:

[https://www.upc.edu/transparencia/ca/publicitat-activa/informacio-de-personal/20230321\\_taules\\_retributives\\_del\\_personal\\_d\\_administracio\\_i\\_serveis\\_any\\_2023.pdf](https://www.upc.edu/transparencia/ca/publicitat-activa/informacio-de-personal/20230321_taules_retributives_del_personal_d_administracio_i_serveis_any_2023.pdf)

[19] Arduino. Arduino Giga R1 WiFi [en línea]. Disponible en:

<https://store.arduino.cc/products/giga-r1-wifi>

[20] QVIRO. UR3e Robot [en línea]. Disponible en:

<https://qviro.com/product/ur3#:~:text=%2427%2C245,tasks%20and%20automated%20work,bench%20scenarios>

[21] El Corte Inglés. Caja de 3 bolas de golf TP5x Taylor Made [en línea]. Disponible en:

<https://www.elcorteingles.es/deportes/A39645201-caja-de-3-bolas-de-golf-tp5x-taylor-made/>

[22] Arduino Store. GIGA Display Shield [en línea]. Disponible en:

<https://store.arduino.cc/products/giga-display-shield>

[23] Parlamento Europeo y Consejo de la Unión Europea. Directiva 2001/95/CE del Parlamento Europeo y del Consejo, de 3 de diciembre de 2001, relativa a la seguridad general de los productos [en línea]. Disponible en:

<https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX:32001L0095>

[24] Organización Internacional de Normalización. ISO 10218-1:2011, Robots y dispositivos robóticos - Requisitos de seguridad para robots industriales - Parte 1: Robots [en línea].

Disponible en: <https://www.iso.org/standard/51330.html>

[25] Organización Internacional de Normalización. ISO 10218-2:2011, Robots y dispositivos robóticos - Requisitos de seguridad para robots industriales - Parte 2: Sistemas robot [en línea]. Disponible en: <https://www.iso.org/standard/41571.html>

[26] Parlamento Europeo y Consejo de la Unión Europea. Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos (Reglamento General de Protección de Datos) [en línea]. Disponible en: <https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX%3A32016R0679>

[27] Universal Robots. Manual de script del robot UR, versión 5.12 [en línea]. Disponible en: [https://s3-eu-west-1.amazonaws.com/ur-support-site/163530/scriptmanual\\_5.12.pdf](https://s3-eu-west-1.amazonaws.com/ur-support-site/163530/scriptmanual_5.12.pdf)

[28] Universal Robots. Control remoto a través de TCP/IP en la interfaz del robot UR. [en línea]. Disponible en: <https://www.universal-robots.com/articles/ur/interface-communication/remote-control-via-tcp-ip/>

[29] Medium. (s.f.). "The Actor-Critic Reinforcement Learning Algorithm" [en línea]. Disponible en: <https://medium.com/intro-to-artificial-intelligence/the-actor-critic-reinforcement-learning-algorithm-c8095a655c14>

[30] Weng, Lilian. "Policy Gradient Algorithms" [en línea]. Publicado el 8 de abril de 2018. Disponible en: <https://lilianweng.github.io/posts/2018-04-08-policy-gradient/#sac>

[31] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., Levine, S. Soft Actor-Critic Algorithms and Applications. Arxiv preprint, 2018. Disponible en: <https://arxiv.org/pdf/1812.05905>

[32] Sutton, R. S., Barto, A. G. Reinforcement Learning: An Introduction. 2nd ed., 2020. Disponible en: <http://incompleteideas.net/book/RLbook2020.pdf>

[33] Netron. Visualizador de modelos de redes neuronales [en línea]. Disponible en: <https://netron.app/>