Universidad San Jorge

Escuela de Arquitectura y Tecnología

Grado en Ingeniería Informática

Proyecto Final

Visión artificial para el cálculo de trayectorias de brazos robóticos en entornos dinámicos

Autor del proyecto: Iñigo Escalante Escarda

Director del proyecto: Ana Cristina Marcén Terraza

Zaragoza, 12 de Septiembre de 2023

Este trabajo constituye parte de mi candidatura para la obtención del título de Graduado en Ingeniería Informática por la Universidad San Jorge y no ha sido entregado previamente (o simultáneamente) para la obtención de cualquier otro título.

Este documento es el resultado de mi propio trabajo, excepto donde de otra manera esté indicado y referido.

Doy mi consentimiento para que se archive este trabajo en la biblioteca universitaria de Universidad San Jorge, donde se puede facilitar su consulta.

Firma Fecha

Dedicatoria y Agradecimientos

Agradezco a todos los profesores que me han educado durante la carrera y a los amigos que me han apoyado.

Agradezco especialmente a mi familia que me ha aguantado durante estos años de estrés y que me han ayudado preparando exámenes y trabajos y han propiciado un ambiente de trabajo adecuado durante la realización del proyecto.

Dedico este trabajo a mis abuelos, a los cuales les habría encantado poder estar en este momento de mi vida. A mi abuelo por siempre animarme a seguir por el camino de la ingeniería y a mi yayo por ayudarme a realizar mis estudios universitarios.

Tabla de contenido

Resumen **1**

Abstract **2**

1. Introducción **5**

2. Antecedentes **7**

2.1. Evolución de la Automatización Industrial **7**

2.1.1. Previo a la Revolución Industrial *7*

2.1.2. Durante la Revolución Industrial *8*

2.1.3. Siglo XXI *9*

2.2. Estado del Arte en Brazos robóticos **10**

2.3. Machine Learning y Brazos robóticos **12**

3. Objetivos **15**

4. Metodología **17**

4.1. Selección de la Metodología **17**

4.2. SCRUM **18**

4.3. Planificación del proyecto **19**

5. Desarrollo del Proyecto **23**

5.1. Sprint 1 ..... **23**

5.1.1. OB0-T01. Búsqueda de proyectos sobre brazos robóticos *23*

5.1.2. OB0-T02. Búsqueda de proyectos sobre Machine Learning y OB0-T03. Búsqueda de proyectos que combinen las anteriores *23*

5.1.3. T02. Reunión sobre procedimiento con la empresa *24*

5.1.4. Resumen del Sprint 1 ..... *24*

5.2. Sprint 2 ..... **24**

5.2.1. T02. Discutir la metodología a emplear *24*

5.2.2. OB1-T01. Investigación sobre Unity *25*

5.2.3. OB1-T02. Investigación sobre MLAgents *26*

5.2.4. OB1-T03. Investigación sobre Anaconda *26*

5.2.5. OB1-T04. Investigación sobre C# *26*

5.2.6. OB1-T05. Estudio de compatibilidad de las herramientas *27*

5.2.6.1. MLAgents *27*

5.2.6.2. Anaconda *27*

5.2.6.3. Pytorch *27*

5.2.6.4. Protobuf *27*

5.2.6.5. Tensorboard *28*

5.2.7. OB1-T06. Implantación de las herramientas *28*

5.2.8. OB1-T07. Realización de Proyectos Tutoriales *28*

5.2.9. Resumen del Sprint 2 ..... *29*

5.3. Sprint 3 ..... **30**

5.3.1. OB1-T07. Realización de proyectos tutoriales *30*

5.3.2. OB2-T01. Modelado del escenario base 30
5.3.3. OB2-T02. Importación y ajustes del modelo UR10e 31
5.3.4. OB2-T03. Modelado de obstáculo 32
5.3.5. OB2-T04. Modelado de objetivos 33
5.3.6. Resumen del Sprint 3 33
5.4. Sprint 4 34
5.4.1. OB2-T05. Investigación sobre el mejor acercamiento posible 34
5.4.2. OB2-T06. Creación y configuración de las articulaciones 34
5.4.3. OB2-T07. Restricción de movimientos 37
5.4.4. OB2-T08. Programación de la lógica de colisiones 38
5.4.5. OB2-T09. Programación de la lógica del entrenamiento por refuerzo 41
5.4.5.1. Introducción al ciclo de entrenamiento por refuerzo 41
5.4.5.2. Ejemplo del Pong 44
5.4.5.3. Progreso con el modelo real 45
5.4.5.3.1. Ciclo de refuerzo: Observación 46
5.4.5.3.2. Ciclo por refuerzo: Decisión 47
5.4.5.3.3. Ciclo por refuerzo: Acción 47
5.4.5.3.4. Ciclo por refuerzo: Castigo y Recompensa 48
5.4.6. Resumen del Sprint 4 50
5.5. Sprint 5 51
5.5.1. OB2-T09. Finalización del ciclo de Castigo y Recompensa 51
5.5.2. OB3-T01. Fase de Entrenamiento por Refuerzo 52
5.5.2.1. Introducción al Entrenamiento por Refuerzo 52
5.5.2.2. Resultados del Entrenamiento por Refuerzo 57
5.5.3. OB3-T02. Fase de Entrenamiento por Imitación 61
5.5.4. Resumen del Sprint 5 62
5.6. Sprint 6 63
5.6.1. OB3-T02. Fase de Entrenamiento por Imitación 63
5.6.2. OB3-T03. Fase de Entrenamiento Combinado 64
5.6.3. OB4-T00. Entrenamiento de un robot de 3 articulaciones 65
5.6.4. OB4-T01. Analizar resultados y sacar conclusiones 67
5.6.5. Resumen del Sprint 6 67
6. Estudio Económico 69
6.1. Estudio sobre el trabajo realizado 69
6.2. Soluciones a futuro 70
6.2.1. Cloud Computing (Recomendada) 70
6.2.2. Ordenador dedicado 71
7. Resultados 73
7.1. Resultados de entrenamientos 73
7.2. Artefactos producidos 73
7.2.1. Entornos de entrenamiento 73
7.2.2. Lecciones Aprendidas 73
7.2.2.1. Enlazar entrenamientos 73
7.2.2.2. Entrenamiento combinado 73

7.2.2.3. Qué tipo de articulaciones usar 74
7.2.2.4. Complejidad exponencial en base al número de articulaciones 74
7.2.2.5. Necesidad de más poder de procesamiento 74
7.3. Desviaciones de la metodología 74
8. Conclusiones 75
9. Bibliografía 77
10. Anexos 83
10.1. Propuesta del Proyecto 84
10.2. Actas de las reuniones 85
10.3. Tablas de entrenamiento 97
10.3.1. Imágenes de los modelos y escenarios 97
10.3.2. Tablas de entrenamiento 102
10.3.2.1. Entrenamientos enlazados 128
10.3.2.2. Entrenamientos perdidos 129

Tabla de Figuras

Figura 1. Clepsidra [6] 7
Figura 2. Autómata Escritor de Jaquet-Droz [7] 7
Figura 3. Prensa de Gutenberg [9] 8
Figura 4. Dick Mourley, inventor del primer PLC [13] 9
Figura 5. Cadena de montaje de móviles con interacción humana [15] 10
Figura 6. Diversos modelos de Brazos Robóticos Industriales [16] 10
Figura 7. Brazo Robótico moviendo objetos de un lado a otro [17] 11
Figura 8. Brazo Robótico soldando piezas [18] 11
Figura 9. Brazo Robótico pegando piezas [19] 11
Figura 10. Brazo Robótico paletizando [20] 11
Figura 11 Brazo Robótico inspeccionando [21] 11
Figura 12. Product Backlog 19
Figura 13. Sprint 1 19
Figura 14. Diagrama de Gantt con Sprints 1-3 20
Figura 15. Diagrama de Gantt con Sprints 4-6 20
Figura 16. UR10e de Universal Robots [33] 25
Figura 17. Primer tutorial. Aprender a perseguir objetivo 28
Figura 18. Segundo tutorial. Aprender sobre físicas y colisiones 29
Figura 19. Tercer tutorial. Primer brazo robótico 29
Figura 20. Modelo del escenario base 31
Figura 21. Modelo completo con demasiados elementos 32
Figura 22. Ejemplo de tornillería innecesaria 32
Figura 23. Ejemplo visual de cilindro protector, suponiendo que el sombrero es el obstáculo [45] 33
Figura 24. Modelo final del obstáculo. Es el cilindro de color verdoso 33
Figura 25. Jerarquía inicial 35
Figura 26. Vista Wireframe del anclaje Base-Hombro 36
Figura 27. Jerarquía combinada 37
Figura 28. Snippet de clampeo de rotación de articulaciones 38
Figura 29. Ayuda visual para Rigid Body y Mesh Collider 1 [50] 39
Figura 30. Ayuda visual para Rigid Body y Mesh Collider 2 [50] 39
Figura 31. Ejemplo de Mesh Collider (verde) en la base 39
Figura 32. Base visible en comparación con el Mesh Collider (verde) 39
Figura 33. Componente Mesh Collider 40
Figura 34. Imagen de una partida de Pong [55] 44
Figura 35. Invocación del obstáculo en posición aleatoria 46
Figura 36. Snippet control de colisiones 48
Figura 37. Funciones de castigo 49
Figura 38. Funciones de recompensa 50
Figura 39. Snippet para mover la goal 51
Figura 40. Snippet del override de Heuristic 52
Figura 41. Escena final 53
Figura 42. Entorno de entrenamiento masivo 54
Figura 43. Carpeta config 55
Figura 44. Gráficas de Tensorboard. Cumulative Reward / Episode Length 56

Figura 45. Accumulative Reward del entrenamiento 59
Figura 46. Episode Length del entrenamiento 59
Figura 47. Gráfica ilógica 60
Figura 48. Nuevo modelo del Brazo Robótico 62
Figura 49. Cambios en el yaml para acomodar el Entrenamiento por Imitación 63
Figura 50. Cumulative Reward entrenamiento combinado 64
Figura 51. Episode Length entrenamiento combinado 64
Figura 52. Imagen del RSv1 66
Figura 53. Entorno Masivo del RSv1 66
Figura 54. Modelo de brazo robótico UR10ev1 97
Figura 55. Modelo de brazo robótico UR10ev2 98
Figura 56. Modelo de brazo robótico UR10ev3 99
Figura 57. Modelo de brazo robótico RSv1 100
Figura 58. Modelo del escenario 1 101
Figura 59. Modelo del escenario 2 101
Figura 60. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 1 102
Figura 61. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 2 103
Figura 62. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 3 104
Figura 63. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 4 105
Figura 64. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 5 106
Figura 65. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 6 107
Figura 66. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 7 108
Figura 67. . Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 8 109
Figura 68. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 9 110
Figura 69. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 10 111
Figura 70. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 11 112
Figura 71. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 12 113
Figura 72. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 13 114
Figura 73. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 14 116
Figura 74. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 30 118
Figura 75. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 31 119

Figura 76. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 32 121
Figura 77. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 33 123
Figura 78. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 34 125
Figura 79. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 35 127

Resumen

Desde el 2000 a.C., existen registros de trabajos de automatización, como la famosa Clepsidra. Actualmente existen ejemplos más complejos, como las máquinas "transfer", que están diseñadas para la fabricación de una pieza concreta. El principal objetivo de la automatización es que un trabajo se realice con la menor interacción humana posible.

En trabajos de automatización, es esencial el cálculo correcto y preciso de la trayectoria de partes móviles. Esta tarea es actualmente llevada a cabo por operarios humanos por medio de algoritmos predefinidos. El objetivo de este proyecto es preparar un entorno de entrenamiento viable para un brazo robótico. Este entorno se utilizará para estudiar si, mediante Machine Learning y el uso de Visión Artificial, se puede entrenar a un robot para realizar estas tareas.

Se ha utilizado una metodología SCRUM ajustada a las necesidades del proyecto. El resultado final es un entorno preparado para cualquier entrenamiento de trayectoria de brazo robótico de 6 articulaciones (UR10e) totalmente integrado con físicas, colisiones y modelos 3D con un obstáculo cambiante para simular el entorno dinámico.

Aunque los resultados del entrenamiento parecen prometedores, haría falta realizar un estudio más extenso para comprobar la viabilidad de la propuesta. El siguiente paso sería probar a entrenar con Cloud Computing y, si resulta viable, entrenar un robot para que realice cualquier tarea en el mismo escenario.

Palabras Clave

Machine Learning, Inteligencia Artificial, Entorno, Modelo, Articulaciones, Colisiones, Entrenamiento.

Abstract

Since 2000 B.C., there have been records of automation put into work, like the famous Clepsydra. Nowadays, there are more complex examples like "transfer" machines, that are designed to produce a specific piece of some sort. The main objective of automation is maintaining human interaction to a minimum when working.

When it comes to automation projects, it is imperative that all the calculations of the trajectories of all the mobile parts are correct and precise. This task is usually undertaken by human operators through predefined algorithms. The goal of this project is to make a training environment viable for a robot arm. This environment will be used to confirm if it's possible to train a robot to do these calculations through Machine Learning and Artificial Vision.

An adjusted version of SCRUM was the methodology used in the development of this project. The final result is a fully operational environment to train a 6 joints robot (UR10e), composed with physics, collisions and 3D models while also simulating a dynamic environment by using an ever-moving obstacle.

Even though the training results seem promising, more extensive research needs to be done to ascertain the viability of the premise. The next step would be to make use of Cloud Computing and, if it works, train a robot to do any given task in the designed scenario.

Key Words

Machine Learning, Artificial Intelligence, Environment, Model, Joints, Collisions, Training.

1. Introducción

Automatización es un término que se utiliza para aplicaciones de tecnología en las que se minimiza la interacción humana. Incluye, por ejemplo, la automatización de procesos de negocio (BPA), la automatización de TI (Tecnología de la Información) o las aplicaciones personales, como la automatización del hogar [1].

En el mundo de la automatización industrial, un brazo robótico es esencial para llevar a cabo diversas tareas. Éstas pueden ser algo tan simple como mover un objeto de un lado a otro, si el robot consta de un apéndice de agarre, o algo más complicado como realizar soldaduras con un apéndice soldador. Para preparar dicho brazo robótico, antes de realizar estas tareas, los operarios definen a mano y, con ayuda de algoritmos predefinidos, una trayectoria a seguir para que realice su tarea.

Estas trayectorias, una vez definidas, son inmutables así que no pueden alterarse si se quiere mantener un buen funcionamiento del entorno automatizado. Aunque, estos brazos robóticos pueden programarse con scripts más complejos que, con la incorporación de sensores de visión artificial, podrían esquivar obstáculos a tiempo real, como haría una puerta automática al reconocer un objeto en su campo visual artificial. El problema de esta solución es que cada obstáculo entraña problemas que requieren soluciones distintas unas a otras y que deben ser programadas a mano.

Este proyecto aborda el problema de la programación manual de brazos robóticos mediante el uso de Inteligencia Artificial (IA). En concreto, se plantea equipar a un brazo robótico con un cerebro de red neuronal entrenado mediante Machine Learning. De esta forma, el brazo robótico, equipado tanto con el cerebro de red neuronal previamente entrenado como por la anteriormente mencionada visión artificial, calculará y recalculará automáticamente las trayectorias óptimas para realizar su tarea sin importar los obstáculos salvables que se encuentren en su área de trabajo.

Este proyecto esta propuesto por la empresa ASAI [2], cuyo principal objetivo es la automatización industrial en diferentes sectores. ASAI considera este proyecto relevante para ahorrar tiempo y esfuerzo a la hora de preparar sus zonas de trabajo con brazos robóticos. Al trabajar tres meses con ellos, tuve la oportunidad de ver en acción diversos proyectos que tenían en marcha. Tras estudiar sus soluciones a estos proyectos, se aprecia claramente que el uso de Inteligencia Artificial sería una ayuda inestimable a la hora de solventar los problemas durante el desarrollo.

Existen varias maneras de resolver el problema propuesto por ASAI, desde algoritmos específicamente diseñados para cada posible escenario, hasta diversas herramientas de Machine Learning. Dado que dicha empresa planea usar ROS (Robotic Operating System) como interfaz entre la programación y el propio robot, Unity se presenta como un entorno ideal para el entrenamiento de esta Inteligencia Artificial ya que cuenta con integración oficial con ROS y librerías propias de Machine Learning.

El resultado de este proyecto es un entorno de entrenamiento capaz de enseñar a un brazo robótico UR10e a realizar distintas tareas mientras esquiva obstáculos que cambian al finalizar cada ciclo.

En esta memoria, empezaré explicando los antecedentes y estado del arte de proyectos similares (2. Antecedentes), seguiré con los objetivos del proyecto (3. Objetivos) para continuar con una explicación de la metodología empleada para desarrollar el proyecto (4. Metodología). A continuación, se explicará con detalle la fase de desarrollo del proyecto (5. Desarrollo del Proyecto), seguido del estudio económico (6. Estudio Económico), los resultados obtenidos al finalizar (7. Resultados) y las conclusiones a las que se han llegado (8. Conclusiones).

2. Antecedentes

No se puede esperar que un proyecto de Inteligencia Artificial aplicada a robots sea el primero de esa índole. Por lo tanto, para poner el problema en contexto, esta sección aborda tanto la evolución de la automatización industrial, como el estado actual de los brazos robóticos y su aplicación junto a Machine Learning.

2.1. Evolución de la Automatización Industrial

Para comprender el estado del arte de la automatización actual, es importante entender cómo hemos llegado hasta aquí, históricamente.

2.1.1. Previo a la Revolución Industrial

Podemos datar casos de automatización tan lejanos como en la antigua Grecia con la Clepsidra [3] (Figura 1) o los primeros ejemplos de puertas hidráulicas, y autómatas del siglo XVII y XVIII, como el autómata escritor de Jaquet-Droz (véase Figura 2, también fabricó uno dibujante y uno pianista) [4].

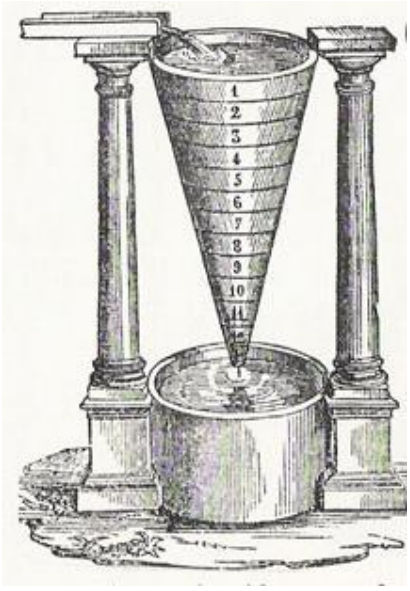

Figura 1. Clepsidra [5]

Figura 2. Autómata Escritor de Jaquet-Droz [6]

Otro de los casos más famosos históricamente, sería la prensa de Gutenberg en el siglo XV (véase Figura 3). Aunque no excluía completamente la interacción humana en el proceso de imprenta, sí la limitó enormemente, expandiendo la industria de la comunicación escrita hacia todo el mundo [7].

Figura 3. Prensa de Gutenberg [8]

Sin embargo, es importante mencionar que el gran salto en la automatización se dio en el siglo XX con la Revolución Industrial.

2.1.2. Durante la Revolución Industrial

En 1947, se fabricó el primer transistor en los laboratorios Bell. Idea de los físicos John Bardeen, Walter Brattain y William Shokkley [9] [10].

En 1952, Heinrich Grünebaum desarrolló el primer motor Alquist que, aunque no fue el primer motor de corriente continua, si fue el precursor de los motores más modernos que se usaron durante ese siglo en fábricas de todo tipo [9] [10].

En 1959, surgió la primera máquina controlada por un ordenador. Era una herramienta de máquina de maquinación, aunque todavía controlada por cables [9] [10].

En 1968, se dio un salto importante: la invención del primer PLC (Programmable Logic Controller) [9] [10], inventado por Dick Mourley (véase Figura 4). Esta herramienta permitía la programación de ciertas máquinas, lo que aseguraba un funcionamiento efectivo y correcto, a la vez que limitaba la interacción humana con dicha máquina [11].

Figura 4. Dick Mourley, inventor del primer PLC [12]

En 1987, por pura coincidencia, al instalar un disco duro en un PC, se hizo patente que un PC podría ser usado para más cosas que simplemente recoger y guardar datos. Este fue un paso decisivo a la hora de introducir ordenadores en el mundo de la industria [9] [10].

En 1997, las fábricas empiezan a integrar ordenadores a sus entornos de trabajo. Cada vez se desarrollan más aplicaciones "virtuales" para acompañar a los procesos físicos [9] [10].

2.1.3. Siglo XXI

Al llegar el siglo XXI, la incorporación de la informática en las fábricas es algo que se espera. Pensar en una fábrica no informatizada y que no haga uso de décadas de avances en el mundo de la automatización es algo inaudito.

Hasta ahora, la mayor parte de los procesos requeridos para fabricar un bien, ya sea algo simple como un vaso o complejo como un coche, han sido automatizados. Sin embargo, muchos trabajos todavía deben ser realizados por humanos. Estos trabajos son principalmente los de recogida de piezas o la recarga de materia prima. Otro trabajo muy importante en el proceso de fabricación de algo complejo es el traslado de piezas intermedias. En el caso de la fabricación de un yo-yo, por ejemplo, sería el traslado de las piezas que forman el cuerpo del yo-yo a otra cinta que las lleve a una máquina que enrolle un cordel en el centro de esas piezas.

Estas tareas son normalmente realizadas por humanos (ver Figura 5) ya que, aunque una cinta puede trasladar objetos de un lugar a otro, muchas veces las piezas no salen de la primera máquina en una posición o estado que sea óptimo para que la segunda máquina las recoja. Aquí entra el ser humano, que coloca dichas piezas de la manera correcta en cada cinta.

Claro, el ejemplo de un yo-yo puede no ser el más ilustrativo, pero imaginemos el proceso de fabricación de una pelota de baloncesto. Se utiliza goma cortada en patrones muy concretos. Para ello se utilizan máquinas de cortado, prensas y motores. Sin embargo, durante todo este proceso, seres humanos deben colocar las piezas de goma en las posiciones adecuadas para que la máquina pueda manipular dicha goma de forma eficiente y deseable. Sin esta interacción humana, simplemente pasar las gomas de un lado a otro por cintas, la pelota jamás podría ser fabricada [13]. Pero, ¿y si hubiese una forma de evitar esta interacción humana?

Aquí es donde entran al partido unos nuevos jugadores: los brazos robóticos.

Figura 5. Cadena de montaje de móviles con interacción humana [14]

2.2. Estado del Arte en Brazos robóticos

Un brazo robótico es una máquina con diversas articulaciones que permiten mover cierta parte de dicha máquina (normalmente la mano, o el actuador) por cualquier coordenada de su área de trabajo. Se pueden ver ejemplos de varios diseños en la Figura 6. Aunque no todos se parecen a un brazo humano, algunos lo hacen, sin embargo, este no es el objetivo. Lo único que le importa a un brazo robótico es poder mover su actuador (una garra, un soldador, una cámara, etc.) a cualquier parte que la tarea a realizar requiera.

Gracias a los brazos robóticos podemos evitar que humanos estén trabajando entre peligrosas máquinas y reducir el espacio de la cadena de montaje. Estos brazos pueden ser capaces de ver las piezas en la cadena de montaje, agarrarlas gracias a diferentes clases de actuadores y colocarlas de manera correcta en otra cadena de montaje para seguir con el proceso de fabricación.

Pero no sólo eso, también son capaces de realizar tareas que no tienen nada que ver con el traslado. Son capaces de realizar tareas como atornillar, soldar, pegar, mover, etc. El límite está sólo en el diseño de sus actuadores (garras, sopletes, pistolas de pegamento, etc.). Esto expande la capacidad de automatización de una cadena de montaje hasta límites sólo definidos por el dinero y el avance en tecnologías.

Figura 6. Diversos modelos de Brazos Robóticos Industriales [15]

Hoy día existen modelos de robots capaces de realizar todo tipo de tareas.

Figura 7. Brazo Robótico moviendo objetos de un lado a otro [16]

Figura 8. Brazo Robótico soldando piezas [17]

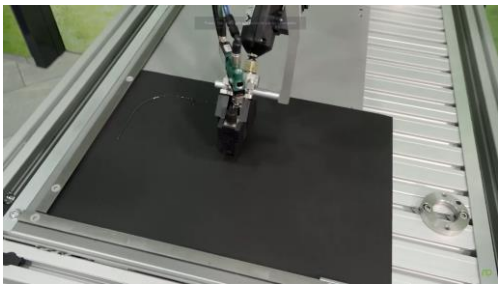

Figura 9. Brazo Robótico pegando piezas [18]

Figura 10. Brazo Robótico paletizando [19]

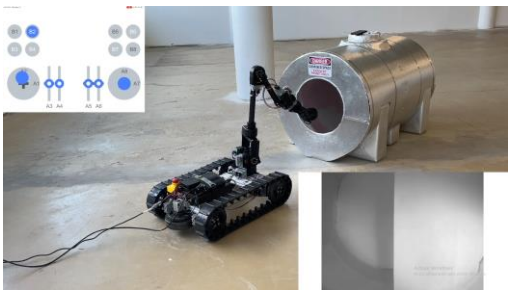

Figura 11 Brazo Robótico inspeccionando [20]

En las imágenes de arriba se pueden apreciar las distintas tareas que ocupan la mayoría de los trabajos que realiza un brazo robótico.

- Pick and Place. Que significa "coger y colocar" (véase Figura 7).
- Soldadura (véase Figura 8).
- Pegado de piezas. Puede ser con pegamento, resina o cualquier otra sustancia diseñada para pegar piezas (véase Figura 9).
- Paletizado. Consiste en el colocado de palés para almacenamiento o transporte (véase Figura 10).
- Inspección. Con la ayuda de sensores, un brazo robótico también puede usarse para realizar inspecciones automatizadas o en lugares donde un humano correría riesgo (véase Figura 11).

2.3. Machine Learning y Brazos robóticos

Ahora ya sabemos de lo que son capaces los brazos robóticos, pero muchos de ellos sólo tienen programadas tareas inmutables. Tareas como coger una caja que siempre va a estar en la misma posición y lugar en el espacio y llevarla a otro lado. Sin embargo, como hemos mencionado antes, ¿qué pasa si la caja o pieza no siempre viene de la misma manera? Ahí entra la Inteligencia Artificial y, más concretamente, el Machine Learning.

Ciertamente se podrían programar manualmente comportamientos para que el brazo robótico realice la tarea asignada independientemente de los obstáculos a afrontar. Es decir, el brazo robótico se programa para cada posible situación que se vaya a encontrar en su ámbito de trabajo (siempre suponiendo que las posibles situaciones sean un número limitado y no infinito). El problema es que en la mayoría de los casos el número de situaciones es demasiado grande, en algunos casos, infinito.

El Machine Learning es una herramienta de programación que evita precisamente que el programador tenga que contemplar todos los casos posibles de un entorno de trabajo, creando un cerebro "virtual" que hace que la máquina o aplicación sea capaz de pensar por sí misma. Existen muchos casos de proyectos de esta índole, he seleccionado algunos interesantes para contrastar sus puntos fuertes y débiles.

Unos investigadores suscritos a la plataforma **MoreLab** entrenaron un modelo de doble Agente para conseguir llevar de una cinta a otra un objeto utilizando un brazo robótico y una plataforma móvil [21] [22]. Esto es un caso de Machine Learning doble, donde el primer Agente aprende a coger y dejar un objeto, y también a esperar a estar en la posición adecuada para ello. El segundo Agente se encarga de saber dónde el primer Agente necesita estar, y moverlo acorde a ello. Es un trabajo muy interesante, pero el proceso de desarrollo ha sido pensado en un ambiente meramente virtual, con lo que no se hace uso de Visión Artificial.

Una desarrolladora del canal **SoomeemHahm Design** diseñó y entrenó un modelo de brazo robótico que intentase siempre acercarse a un objetivo móvil [23]. Esto es notable porque consigue enseñarle virtualmente a un robot a seguir un objeto dinámico. Si bien no tiene que esquivar nada, seguir un objeto concreto también es un proyecto valioso. Lamentablemente el proyecto también fue pensado puramente en un estado virtual, con lo que no hace uso de Visión Artificial.

Desarrolladores de **MM CODE** (nombre de la empresa) diseñaron y entrenaron un modelo sobre un brazo robótico de 4 articulaciones que moviese un objeto de un sitio a otro esquivando unos obstáculos fijos. Después consiguieron integrarlo en un robot real [24]. Este es quizás el más interesante desde mi punto de vista. Consiguen que un robot realice la tarea de "Pick and Place" (mover un objeto) a la vez que esquiva un entorno estático. Este proyecto sí se hizo pensando en una aplicación real, con lo que sí que hace uso de Visión Artificial. Entrenar este proyecto no debió ser fácil pero, si estudiamos el vídeo detenidamente, vemos que no utiliza más que dos articulaciones, lo que reduce el tiempo necesario de entrenamiento. No obstante, me parece un proyecto digno de mención.

También existe un proyecto Kickstarter, llamado **JetMax**, relacionado con robótica e Inteligencia Artificial empleando visión artificial para fines educativos o recreativos aplicado a un robot real [25]. Es un proyecto atractivo a nivel tecnológico y, aunque lo he calificado como proyecto de nivel recreativo/educativo, considero que las posibles aplicaciones de este

proyecto, llevadas a una escala industrial, podrían dar resultados significativos. Utiliza Visión Artificial y es muy probable que sea capaz de esquivar entornos estáticos y muy posiblemente dinámicos. Sin embargo, por la documentación que he encontrado, no puedo garantizar estas dos últimas cualidades.

En la siguiente tabla (véase Tabla 1) se comparan los proyectos anteriormente mencionados con el que nos ocupa. Se comparan las características que incumben a este proyecto en cuando a necesidades tecnológica se refiere.

	Visión Artificial	**Número de articulaciones**	**Esquivar entorno estático**	**Esquivar entorno dinámico**
MoreLab	**X**	**6**	✓	**X**
SoomeemHahm	**X**	**6**	**X**	**X**
MM CODE	✓	**3**	✓	**X**
JetMax	✓	**3**	✓*	✓*
Este TFG	✓	**6**	✓	✓

Tabla 1. Comparativa de proyectos

*No existe confirmación, pero es factible que tenga esas capacidades.

3. Objetivos

El objetivo principal de este proyecto es realizar un estudio para determinar si se puede conseguir un cerebro de red neuronal entrenado con Machine Learning y siendo capaz de integrarse en un robot real UR10e usando ROS (Robotic Operative System). Dicho robot debe ser capaz de completar la tarea especificada: mover un objeto de una mesa a otra esquivando obstáculos cambiantes en cada ciclo.

Para ello, se debe crear un entorno de entrenamiento viable en 3D. Este entorno debe integrar todas las limitaciones reales del robot (posiciones, articulaciones, colisiones…) así como un modelo preciso del mismo robot real. Cabe destacar, que dicho entorno, de modelarse correctamente, puede usarse para entrenar al robot para cualquier otra tarea de esta índole.

La empresa ASAI ha dejado claro que quiere que el robot esquive los obstáculos mediante el uso de visión artificial, por lo tanto, el entorno debe ser modelado de una manera que un robot pueda entenderlo. No puede recibir más información que la que conseguiría en un entorno real, utilizando dicha visión artificial.

Teniendo esto en cuenta, la empresa pretende conseguir resultados siguiendo estos objetivos:

OB1: Estudio del problema y de las capacidades de la visión artificial disponible en los robots disponibles.

OB2: Diseño de un sistema capaz de resolver la situación descrita.

OB3: Implementación del sistema utilizando Inteligencia Artificial y/o Machine Learning

Estudio y comparación de resultados teóricos y prácticos.

OB4: Estudio y comparación de resultados teóricos y prácticos.

Cabe destacar que al principio del proyecto vi necesario que añadir un nuevo objetivo para investigar el estado del arte e investigar sobre la industria relacionada:

OB0: Análisis del estado del arte para investigar posibles soluciones.

En este objetivo planeo investigar tanto el estado del arte como formarme en las herramientas necesarias para la realización del proyecto.

En total, estos 5 objetivos abarcan el alcance del proyecto y se realizarán con la metodología descrita a continuación.

4. Metodología

En esta sección, se justifica que metodología se ha seleccionado para planificar el proyecto, se describe cómo se ha utilizado dicha metodología, y la planificación final del proyecto.

4.1. Selección de la Metodología

Al preparar este proyecto se plantearon dos tipos de metodologías: Waterfall y Ágiles. Ambas son buenas metodologías, pero es importante elegir la que más se adecue a las exigencias del proyecto.

La metodología Waterfall, en español también conocida como Desarrollo en Cascada, se puede definir como:

"The Waterfall methodology — also known as the Waterfall model — is a sequential development process that flows like a waterfall through all phases of a project (analysis, design, development, and testing, for example), with each phase completely wrapping up before the next phase begins" [26].

Waterfall es una metodología que se usa comúnmente en proyectos donde se sabe que no va a haber ninguna desviación desde el planteamiento del proyecto hasta la resolución del mismo. En mi caso, este proyecto se realiza en colaboración con una empresa, así que se espera que ésta pueda solicitar cambios durante el desarrollo. Además, este proyecto se basa fundamentalmente en la investigación de cómo aplicar tecnologías desconocidas tanto para la empresa como para mí. Es impensable que no vaya a haber retrasos o desvíos sobre lo que estuviese planeado.

Por estas razones, se optó por el uso de metodologías ágiles. Entre sus principales ventajas, estas metodologías permiten dividir el proyecto en etapas y así centrarse en cada una de forma individual. Además, estás metodologías ágiles permiten adaptar el proyecto a medida que este avanza. Así, ante cualquier cambio que surja, es muy sencillo volver a organizar el equipo en relación con los nuevos objetivos y tareas [27].

Dentro de las metodologías ágiles, se planteó utilizar Extreme Programming (XP) o SCRUM; principalmente porque son las dos metodologías ágiles que hemos estudiado más detalladamente en clase. XP es una metodología que se centra en el tipo de prácticas a la hora de programar y seguir ciertos estándares. Mientras que SCRUM se centra más en conseguir resultados dejando en segundo plano la forma utilizada para conseguirlos [28].

Tanto SCRUM como XP tienen iteraciones (en el caso de SCRUM se llaman Sprints), tienen un Project Manager, un equipo, un Product Owner, etc. Es decir, que tanto XP como SCRUM tienen muchísimas similitudes. Sin embargo, las pocas diferencias que tienen hacen que la balanza se incline a favor de uno. En mi caso, la libertad de operación que SCRUM ofrece a sus desarrolladores ha sido decisiva.

De tal manera y, siendo un equipo de un solo desarrollador, se ha decidido utilizar SCRUM. Esto permitirá que dicho desarrollador tenga más poder de toma de decisiones al elegir las prácticas a utilizar a la hora de desarrollar el proyecto. Esto es importante ya que, por motivos de disponibilidad, tanto del Project Manager como de los Stakeholders, no se pueden realizar reuniones tan a menudo como sería indicado para XP. Además, las fases de entrenamiento en

Machine Learning no siempre salen como estaba planeado y por tanto es difícil asignarles un periodo de tiempo fijo.

4.2. SCRUM

Los cocreadores de Scrum, Ken Schwaber y Jeff Sutherland, escribieron y mantienen The Scrum Guide, que explica Scrum de forma clara y sucinta. La guía contiene la definición de Scrum, describiendo las responsabilidades, eventos, artefactos de Scrum y la guía que los une [29].

En base a la guía proporcionada, para aplicar SCRUM, se necesita un equipo formado por un propietario de producto (Product Owner), un Scrum Master y desarrolladores, cada uno de los cuales tiene responsabilidades específicas:

- El Product Owner es responsable de marcar el objetivo del producto, crear las tareas del Product Backlog, solicitar nuevas tareas en Product Backlog y garantizar que el Product Backlog sea transparente, visible y comprendido.
- Los Scrum Masters son responsables de ayudar a sus equipos a tener éxito y eso a menudo significa ofrecerles asistencia en grupos o individualmente.
- Los desarrolladores son responsables de crear un plan para cada Sprint (Sprint Backlog), inculcar calidad adhiriéndose a una definición de hecho, adaptar su plan cada día hacia el objetivo del Sprint y responsabilizarse unos a otros como profesionales.

Por supuesto, siendo un equipo de un solo desarrollador, debe haber ciertos cambios en la estructura del equipo de SCRUM. Ésta quedaría así:

- **Product Owner**: Iñigo Escalante
- **SCRUM Master**: Ana Cristina Marcén Terraza
- **Equipo**: Iñigo Escalante (todas las tareas de desarrollo)
- **Stakeholders**: ASAI. Aunque la definición del objetivo del producto es responsabilidad del Product Owner, esta responsabilidad será llevada a cabo por los stakeholders en este proyecto.

Además, SCRUM planifica el trabajo en base a Sprints. Los Sprints son períodos de trabajo de duración fija que duran un mes o menos para crear coherencia y garantizar iteraciones breves de retroalimentación con el fin de inspeccionar y adaptar tanto cómo se realiza el trabajo como en qué se está trabajando [30]. Para este proyecto, la duración de los Sprints ha sido variable debido a la disponibilidad (exámenes, vacaciones, etc.). Sin embargo, se han mantenido siempre dentro de los estándares de SCRUM (de 1-4 semanas).

Por último, cabe destacar el Product Backlog (véase Figura 12). El Product Backlog es una lista emergente y ordenada de lo que se necesita para mejorar el producto [31]. Es decir, la lista de tareas a realizar para cumplir con el objetivo u objetivos del proyecto. A continuación, se muestra el producto backlog de este proyecto en base a los objetivos fijados:

OBJETIVOS	# TAREA	TAREAS
Memoria del proyecto	T01	Redactar memoria del proyecto
Investigar estado del arte	OB0-T01	Buscar proyectos sobre brazos robóticos
	OB0-T02	Buscar proyectos sobre machine learning
	OB0-T03	Buscar proyectos que combinen las anteriores
Reunión sobre procedimiento con la empresa	T02	Discutir la metodología a emplear
Estudio del problema y de las capacidades de la visión artificial disponible en los robots disponibles	OB1-T01	Investigación sobre Unity
	OB1-T02	Investigación sobre MLAgents
	OB1-T03	Investigación sobre Anaconda
	OB1-T04	Investigación sobre C#
	OB1-T05	Estudio de compatibilidad de versiones
	OB1-T06	Implantación de las herramientas
	OB1-T07	Realización de Proyectos Tutoriales
Diseño de un sistema capaz de resolver la situación descrita	OB2-T01	Modelar escenario base
	OB2-T02	Importar y acomodar modelo UR10e
	OB2-T03	Modelar obstáculo
	OB2-T04	Modelar objetivos
	OB2-T05	Investigar mejor acercamiento posible
	OB2-T06	Crear y configurar articulaciones
	OB2-T07	Restringir movimientos
	OB2-T08	Programar lógica de colisiones
	OB2-T09	Programar lógica del entrenamiento por refuerzo
Implementación del sistema utilizando inteligencia artificial y/o machine learning	OB3-T01	Fase de entrenamiento por refuerzo
	OB3-T02	Fase de entrenamiento por imitación*
	OB3-T03	Fase de entrenamiento combinada*
Estudio y comparación de resultados teóricos y prácticos	OB4-T00	Entrenamiento de un robot de 3 articulaciones*
	OB4-T01	Analizar resultados y sacar conclusiones

Figura 12. Product Backlog

4.3. Planificación del proyecto

A partir del Product Backlog definido en la Figura 12, se definió el primer Sprint con duración de dos semanas, aunque los días a trabajar estuviesen distribuidos entre los meses de marzo y abril.

		MARZO						ABRIL							
		11	12	18	19	25	26	1	2	8	9	15	16	22	23
# TAREA	TAREAS	SPRINT 1													
T01	Redactar memoria del proyecto														
OB0-T01	Buscar proyectos sobre brazos robóticos														
OB0-T02	Buscar proyectos sobre machine learning														
OB0-T03	Buscar proyectos que combinen las anteriores														
T02	Discutir la metodología a emplear														

Figura 13. Sprint 1

Como se destaca en la Figura 13**¡Error! No se encuentra el origen de la referencia.**, el Sprint Backlog del primer Sprint estaba formado por las tareas OB0-T0, OB0-T1, OB0-T2, OB0-T3 y T02. Al final de este Sprint se inspeccionaron y redactaron los resultados del Sprint y se definió el siguiente Sprint en base a los resultados alcanzados y las tareas completadas. Este proceso se repitió para cada nuevo Sprint dentro del proyecto.

El proceso total con todos los Sprints, desviaciones, nuevas tareas incorporadas en el Product Backlog se muestra en un diagrama de Gantt. Para poder mostrar adecuadamente este diagrama se ha dividido en dos figuras (Figura 14 y Figura 15) y se ha indicado las tareas utilizando únicamente su referencia. Si se desea conocer la descripción de cada una de estas referencias se puede consultar en la Figura 12 que muestra el Product Backlog.

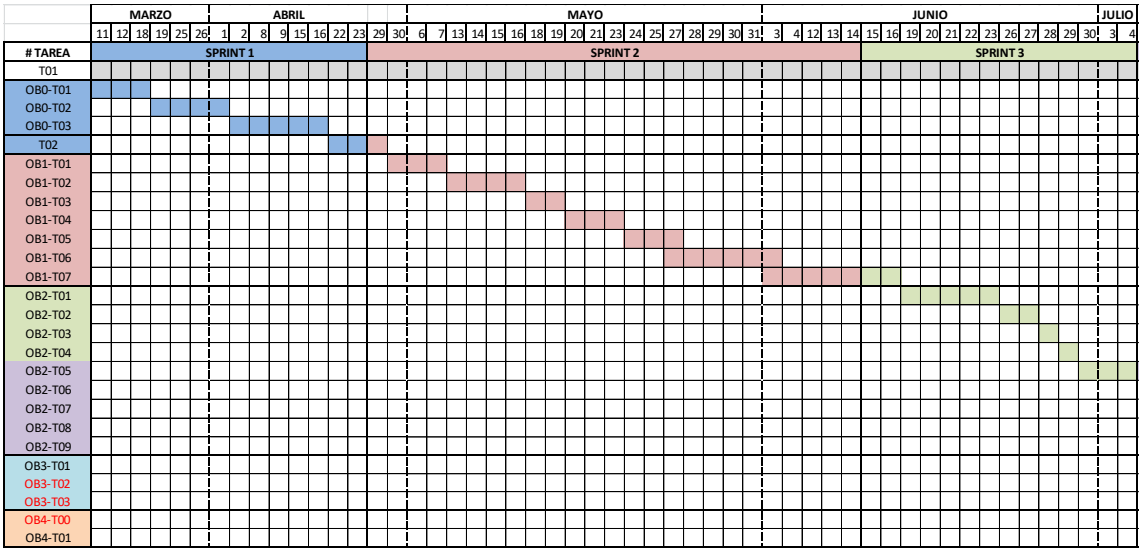

Figura 14. Diagrama de Gantt con Sprints 1-3

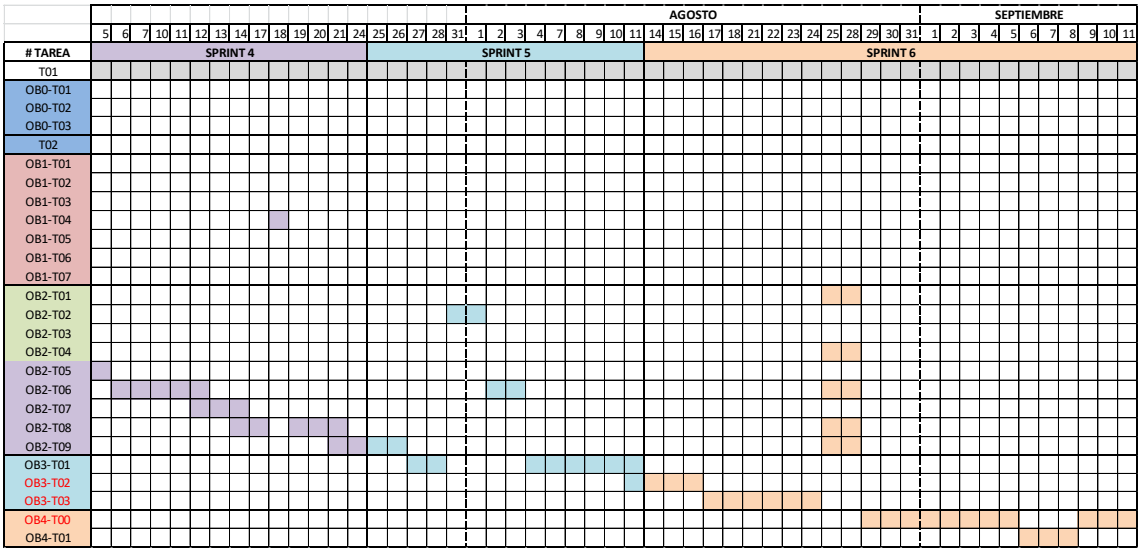

Figura 15. Diagrama de Gantt con Sprints 4-6

Como ya se ha comentado en el capítulo 3. Objetivos, al principio del proyecto, se añadió un objetivo 0 (OB0) para analizar el estado del arte de la industria. También se pueden apreciar ciertos retrasos en los Sprints debido a falta de experiencia a la hora de asignarles un tiempo de desarrollo. Por ejemplo, como se puede observar en la Figura 14, la T02 comenzó en el Sprint 1 pero fue finalizada en el Sprint 2. Además, algunas de las tareas que se daban por finalizadas, tuvieron que ser retomadas posteriormente en otros Sprints. Por ejemplo, en la Figura 15, se puede apreciar que la tarea OB2-T06 se daba por acabada en el Sprint 4 pero fue retomada en los Sprints 5 y 6 debido a cambios en los modelos de brazos robóticos.

En cualquier caso, lo más importante a destacar son las tareas OB3-T02, OB3-T03 y OB4-T00, que están destacadas con un color de fuente roja. Este color de fuente indica que estas tareas no estaban planeadas al principio del proyecto, como se puede ver si comparamos el diagrama de Gantt (Figura 14 y Figura 15) con el del Product Backlog (Figura 12Figura 12). De hecho, estas tareas fueron añadidas durante el desarrollo, debido a la duración de los entrenamientos. Ahondaré más en este tema durante el reporte del desarrollo de los Sprints 5 y 6.

5. Desarrollo del Proyecto

En esta sección, hablaré sobre el desarrollo general del proyecto, desde la fase de aprendizaje, pasando por la de modelado y acabando por las de programación y entrenamiento. Explicaré con detalle cada acción o tarea realizada durante cada Sprint así como su éxito o fracaso.

5.1. Sprint 1

Durante este primer Sprint, me concentré en investigar el mundo relacionado con el ámbito del proyecto. Para abordar el proyecto, necesitaba conocer el estado del arte sobre brazos robóticos, Machine Learning y Visión Artificial.

Este Sprint se centra en las tareas OB0-T01, OB0-T02, OB0-T03 y OB0-T04. También incluye la tarea T02 que consistía en una reunión con mis stakeholders, ASAI, para decidir de qué forma se iba a afrontar este proyecto. Por lo tanto, las tareas abordadas durante el Sprint 1 son:

- OB0-T01: Buscar proyectos sobre brazos robóticos
- OB0-T02: Buscar proyectos sobre machine learning
- OB0-T03: Buscar proyectos que combinen las anteriores
- T02: Discutir la metodología a emplear

Para cada una de estas tareas, se describirá a continuación el trabajo realizado, además de sus problemas, soluciones y resultados si procede.

5.1.1. OB0-T01. Búsqueda de proyectos sobre brazos robóticos

Los brazos robóticos son una parte crítica de este proyecto. Por lo tanto, en esta primera tarea, realicé una búsqueda entender el funcionamiento de los brazos robóticos y las tareas que realizan normalmente dentro del campo de la automatización industrial.

Gracias a ASAI, pude tener experiencia de primera mano con dichos brazos robóticos. Además, extendí este conocimiento con otras fuentes bibliográficas, sobre todo digitales.

Como resultado de la búsqueda se redactó las secciones 2.2 (Estado del Arte en Brazos robóticos) y 2.1 (Evolución de la Automatización Industrial). En esa sección, se muestra el estado del arte de los brazos robóticos en base a la información encontrada y mi experiencia en ASAI.

5.1.2. OB0-T02. Búsqueda de proyectos sobre Machine Learning y OB0-T03. Búsqueda de proyectos que combinen las anteriores

Dada la naturaleza del proyecto, estudiar sobre Machine Learning era algo imperativo. Además, conocer qué proyectos utilizan Machine Learning en brazos robóticos es algo fundamental para contextualizar el proyecto y poder compararlo.

Por eso, investigué sobre proyectos de Machine Learning en 3D así como de Visión Artificial. Los resultados de esta tarea se pueden apreciar en la sección 2.3 (Machine Learning y Brazos robóticos). Esta sección contiene no solamente la información de los proyectos encontrados si no también una tabla comparativa entre esos proyectos y este proyecto.

5.1.3. T02. Reunión sobre procedimiento con la empresa

Antes de empezar a programar o probar herramientas, debía tener una reunión con ASAI para discutir la forma de proceder. Hablé con Rubén de la Rubia, director de I+D de ASAI, por mensajes y correos sobre qué exactamente quería estudiar ASAI y cuáles eran los planes a futuro con este proyecto. Seguidamente, discutimos qué clase de herramientas usar con la ayuda de Antonio Iglesias, profesor de la USJ.

Se decidió utilizar ROS para conectar el ordenador y el futuro cerebro neuronal con los brazos robóticos de ASAI. Por ello, se decidió también emplear Unity y MLAgents para entrenar dicho cerebro ya que ROS y Unity son fácilmente integrables ya que ambos grupos de desarrolladores lo hicieron posible en un trabajo conjunto.

Se aclaró que el proyecto debía enfocarse como un proyecto de investigación por encima de todo. Producir resultados tangibles es algo secundario.

Siendo que nos comunicábamos por correos o mensajes, no pudimos completar la discusión a tiempo para el final del primer Sprint, pero sólo por un día.

5.1.4. Resumen del Sprint 1

Durante este Sprint, se completaron las tareas OB0-T01, OB0-T02, OB0-T03 y OB0-T04, y como resultado se redactó el 2. Antecedentes de esta memoria. Por otro lado, la tarea T02 sólo fue completada parcialmente. Por lo tanto, esto supuso un pequeño desvío y la tarea T02 fue incluida de nuevo en el siguiente Sprint para su finalización.

5.2. Sprint 2

En este Sprint, me centré en investigar las herramientas decididas, así como su compatibilidad e instalación, y el seguimiento de diversos tutoriales básicos para prepararme para el problema propuesto por el proyecto. En concreto, se realizaron las siguientes 7 tareas, las cuales están centradas en alcanzar el objetivo OB1 y completar la tarea T02:

- T02: Discutir la metodología a emplear
- OB1-T01: Investigación sobre Unity
- OB1-T02: Investigación sobre MLAgents
- OB1-T03: Investigación sobre Anaconda
- OB1-T04: Investigación sobre C#
- OB1-T05: Estudio de compatibilidad de versiones
- OB1-T06: Implantación de las herramientas
- OB1-T07: Realización de Proyectos Tutoriales

En las próximas secciones, se describen como se han llevado a cabo estas tareas, además de los problemas, soluciones y resultados si procede.

5.2.1. T02. Discutir la metodología a emplear

Esta tarea se comenzó en el Sprint 1, pero no dio tiempo a completarla. Por lo tanto, en este Sprint 2 se continuó y completó. En concreto, se mantuvieron más conversaciones con Rubén para determinar cómo modelar la escena, qué tipos de obstáculos se deberían esquivar, y qué clase de movimientos podría hacer el robot.

También se decidió sobre qué robot -que estuviera disponible en ASAI- se realizaría el modelado: el UR10e (véase Figura 16). Un brazo colaborativo de 6 articulaciones de tamaño medio. Por contexto, explicaré que un robot colaborativo es un robot capaz de realizar tareas en un entorno donde humanos pueden entrar en su área de trabajo. En cuanto detectase una colisión o fuerza externa, se pararía de inmediato o reduciría su velocidad a una segura.

Rubén también me facilitó el modelo exacto del robot con el que estaban estudiando otros proyectos de IA y que sería el que debería modelar, el UR10e. Gracias a esto, pude conseguir la ficha técnica de dicho robot y asegurarme de conseguir realizar un modelado preciso.

Como resultado se obtuvo todo lo necesario, por parte de la empresa, para poder continuar con el proyecto.

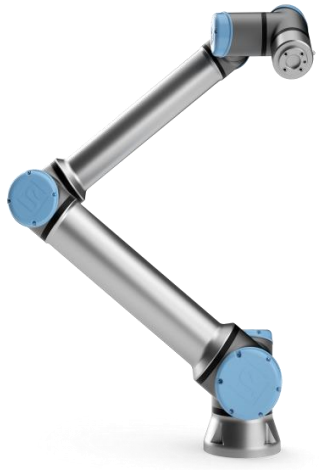

Figura 16. UR10e de Universal Robots [32]

5.2.2. OB1-T01. Investigación sobre Unity

Unity es un motor de videojuegos muy utilizado. Sirve para crear tanto videojuegos 2D como 3D y, gracias a su gran variedad de librerías mantenidas por la comunidad, puede cumplir muchas funciones en el desarrollo de un videojuego en un solo entorno de desarrollo. Gracias a estas herramientas, Unity se ha puesto a la cabeza también como entorno de desarrollo para Machine Learning en entornos gráficos.

Para aprender a usar esta tecnología, es necesario primero conocerla, instalarla, y encontrar tutoriales o material con los que aprender a usarla. Por lo tanto, la meta de esta tarea es tanto conocer un poco más de la herramienta, como su instalación y la utilización de tutoriales para aprender a usarla. Una vez aprendida, podré usarla para modelar el entorno completo y configurar el brazo robótico.

Antonio Iglesias me informó que Unity contaba con una integración nativa con ROS, con lo que se presenta como la herramienta perfecta para realizar el estudio.

Gracias a un tutorial de Unity [33] se consiguió aprender lo necesario para comenzar con el proyecto.

5.2.3. OB1-T02. Investigación sobre MLAgents

Trabajar con Unity y Machine Learning significa trabajar con MLAgents, una de las librerías de Machine Learning más conocidas de Unity. Ofrece amplias posibilidades de entrenamiento automático y es un paquete integrado en Unity por sus desarrolladores [30]. Tiene una gran cantidad de recursos y herramientas para realizar entrenamientos de Machine Learning, tanto simples como complejos.

Gracias a MLAgents, el entorno configurado que se consiga realizar con Unity podrá ser entrenado. MLAgents se aplicará para la configuración de entrenamiento y la programación.

Al investigar sobre MLAgents me enteré de que se especializa en entrenamiento por refuerzo y por imitación. Siendo que ya se planeaba usar un entrenamiento por refuerzo, se cimentó como la herramienta idónea.

Se consiguió aprender lo necesario sobre MLAgents gracias a la documentación proporcionada en su Github [34].

5.2.4. OB1-T03. Investigación sobre Anaconda

Cuando se habla de Inteligencia Artificial y Machine Learning es normal pensar en Python. Es uno de los lenguajes de programación con mayor capacidad de computación de cálculos matemáticos.

Anaconda es un entorno virtual de Python tipo CMD que permite vincular aplicaciones predeterminadas con sus algoritmos de Machine Learning e instalar programas en un entorno aislado de cualquier otro dentro del PC. Esto nos permitirá utilizar el poder de computación de Python en nuestro entrenamiento dentro de Unity.

Siendo que Anaconda es un entorno virtual, todas las demás herramientas que necesitemos para Machine Learning serán instaladas dentro de dicho entorno. Según los desarrolladores de MLAgents, esto es necesario para su correcto funcionamiento [34].

5.2.5. OB1-T04. Investigación sobre C#

C# es un lenguaje de programación orientado a componentes, orientado a objetos. C# proporciona construcciones de lenguaje para admitir directamente estos conceptos, por lo que se trata de un lenguaje natural en el que crear y usar componentes de software [35].

Unity utiliza scripts programados en C#. Estos scripts son los que manejan el comportamiento de los objetos en el escenario. Como se puede intuir, esto es imprescindible para el buen funcionamiento de nuestro entorno, ya que habrá que programar las lógicas de colisiones, configurar las articulaciones y construir el sistema de entrenamiento por refuerzo.

C# será la espina dorsal de este proyecto, ya que es la única forma en la que se podrán programar los algoritmos, como se aprendió en la asignatura de Introducción a la Informática Gráfica.

5.2.6. OB1-T05. Estudio de compatibilidad de las herramientas

Una vez seguro de mi investigación sobre las herramientas a usar, tocaba instalarlas. Se sabía que todas estas herramientas eran compatibles. Aun así, se presentaba un problema de compatibilidades ya que, para realizar Machine Learning con Unity, había que instalar unos paquetes y programas de apoyo importantes con diferentes versiones: Pytorch, Protobuf y Tensorboard. Resulta que varios de ellos no eran compatibles con versiones modernas, con lo que tuve que degradar las versiones de varios programas ya que, por ejemplo, Pytorch exigía una versión más antigua de Python en Anaconda. Sin embargo, si bajaba la versión de Python demasiado, entonces MLAgents no era compatible.

Lamentablemente, la necesidad de documentarse tan extensivamente para encontrar un equilibrio de compatibilidades hizo que la tareaOB1-T06 se retrasara considerablemente.

Primero describiré brevemente las compatibilidades entre las herramientas utilizadas.

5.2.6.1. MLAgents

Su compatibilidad con Unity depende de la versión de éste que uses. Afortunadamente MLAgents integró su paquete de Machine Learning para una versión muy moderna de Unity, la 2019.

Es importante destacar que MLAgents debe ser instalado tanto en Unity como en Anaconda para asegurar la comunicación entre ellos.

5.2.6.2. Anaconda

Como se ha comentado antes, Anaconda es un entorno virtual donde instalaremos las herramientas necesarias para conectar MLAgents con un sistema Python de entrenamiento de Machine Learning.

Afortunadamente, Anaconda no dio ningún problema en términos de compatibilidad, con lo que se pudo instalar sin ningún problema. Sin embargo, las herramientas que debían ser instaladas dentro de Anaconda no serían tan fáciles. La versión de Python a instalar dentro de Anaconda dependerá de estas herramientas.

5.2.6.3. Pytorch

Pytorch es un framework de Machine Learning basado en la librería Torch de Python. Este framework es el que permite que Anaconda realice los cálculos de Machine Learning necesarios para nuestra aplicación. Debe instalarse dentro de Anaconda

Se planteó un problema de compatibilidad con la versión de Python instalada dentro de Anaconda. No podía ser demasiado antigua ni demasiado moderna. Python 3.9 parecía ser la opción más estable.

5.2.6.4. Protobuf

En versiones más modernas de Anaconda, es necesario instalar Protobuf, que es un formato binario encargado de la anteriormente mencionada comunicación entre nuestros programas (Unity, Pytorch…).

Hace poco tiempo que es necesario instalar Protobuf en Anaconda manualmente, así que no había muchas versiones que investigar.

5.2.6.5. Tensorboard

Tensorboard es un programa que mantiene un archivo de los datos conseguidos por MLAgents y Python durante el entrenamiento del Agente. También crea automáticamente gráficas para poder seguir visualmente el progreso del entrenamiento.

Este programa parece ser compatible con todas las versiones relativamente modernas de Anaconda y Python, así que no surgieron problemas al instalarlo dentro de Anaconda.

5.2.7. OB1-T06. Implantación de las herramientas

Ahora tocaba instalar las herramientas. Se consiguió realizar una instalación exitosa de todos los programas con unas versiones que fueran todas compatibles con todas. Para ello se tuvieron que realizar decenas de instalaciones limpias y recurrir a diversos tutoriales [34] [36] [37] [38] [39] que guiaron por el camino correcto hacia un equilibrio de versiones que mantuviese un entorno estable.

5.2.8. OB1-T07. Realización de Proyectos Tutoriales

Acto seguido, se completaron unos tutoriales de MLAgents [40] (véanse Figura 17, Figura 18 y Figura 19). No era aconsejable empezar con el entrenamiento sin antes haberse documentado y haber conseguido cierta experiencia en el campo.

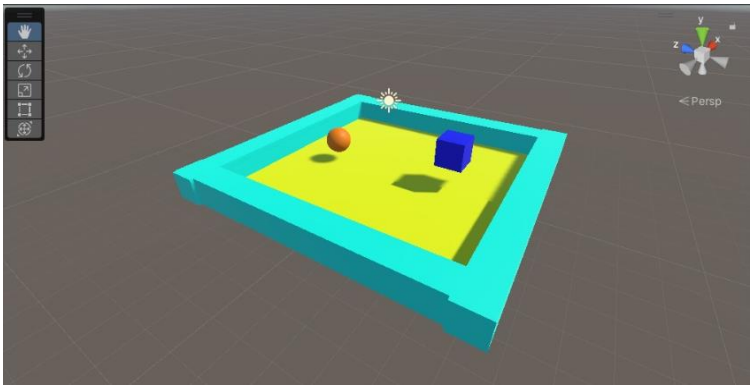

Figura 17. Primer tutorial. Aprender a perseguir objetivo

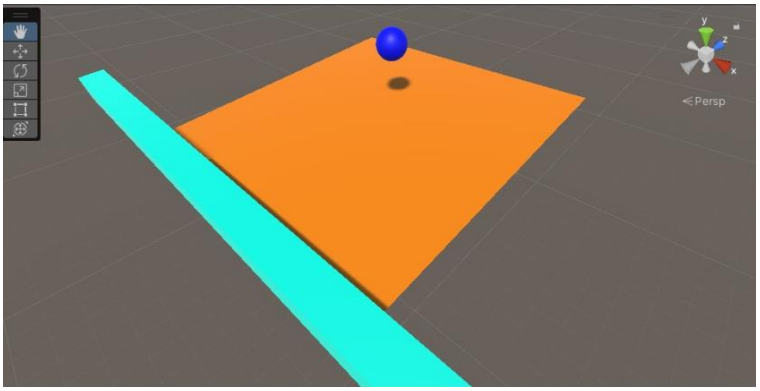

Figura 18. Segundo tutorial. Aprender sobre físicas y colisiones

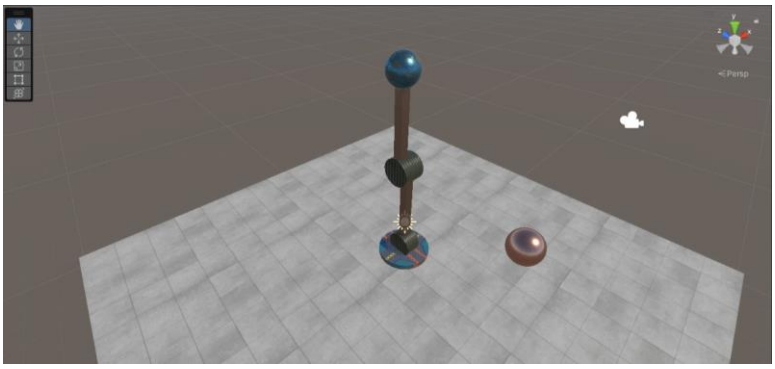

Figura 19. Tercer tutorial. Primer brazo robótico

Realicé desde el tutorial más simple sobre Machine Learning con un cubo y una bola, hasta proyectos prototipo de brazos robóticos rudimentarios. Estos tutoriales me dieron la base que necesitaba, pero no me dio tiempo a acabarlos antes del siguiente Sprint debido al retraso con las tareas antes mencionadas.

5.2.9. Resumen del Sprint 2

Durante este Sprint, me documenté sobre las herramientas a usar en el proyecto y sufrí retrasos durante la instalación debido a problemas de compatibilidad de versiones de los diferentes programas. No obstante, las herramientas estaban correctamente instaladas y operativas antes de finalizar el Sprint. Además, Aprendí un nivel básico de C# y seguí unos tutoriales sobre Unity y MLAgents para aprender desde cero. Los retrasos con las instalaciones hicieron que la tarea OB1-T07 se extendiese hasta el siguiente Sprint.

5.3. Sprint 3

Durante este Sprint, me propuse empezar y acabar el modelado 3D de toda la escena, es decir, el modelado de todos los objetos relevantes en el entorno: Brazo, obstáculo, objetivos y escenario. Pero primero debía acabar con la tarea OB1-T07. Por lo tanto, la lista de tareas a completar para este Sprint es:

- OB1-T07: Realización de proyectos tutoriales
- OB2-T01: Modelar escenario base
- OB2-T02: Importar y acomodar modelo UR10e
- OB2-T03: Modelar obstáculo
- OB2-T04: Modelar objetivos

Para cada una de estas tareas, se describirá a continuación el trabajo realizado, además de sus problemas, soluciones y resultados si procede.

5.3.1. OB1-T07. Realización de proyectos tutoriales

Antes de comenzar con las tareas propuestas para este Sprint, debía acabar de seguir los tutoriales. Concretamente, acabé de seguir el tutorial sobre el brazo robótico rudimentario (Figura 19). Tras completar los tutoriales, conseguí obtener los conocimientos necesarios para poder modelar el escenario base y tener una idea más clara de cómo integrar Machine Learning en ese escenario

5.3.2. OB2-T01. Modelado del escenario base

Antes de empezar con cualquier tipo de entrenamiento, es necesario modelar la escena correctamente. Esto incluye crear modelos 3D de todas las partes relevantes en el entrenamiento, es decir, lo que el robot se va a encontrar en la vida real en su entorno de trabajo. La calidad del modelo de la escena impacta directamente en la calidad del entrenamiento, así que cualquier fallo en la programación y modelado de la escena podría dar como resultado un cerebro neuronal que aprendiese a realizar acciones imposibles en el mundo real como, por ejemplo, realizar una rotación que el brazo real no puede realizar, o chocarse consigo mismo y "atravesarse" como si fuera un fantasma.

Igual que una casa, se empieza por los cimientos, y el entorno de trabajo del robot tiene suelo y obstáculos fijos que debe aprender a esquivar. Para modelarlos, se emplearon formas primitivas de modelado 3D de Unity, como el Plano y el Cubo. Dado que su entorno de trabajo real es una nave industrial, un suelo plano es ideal para imitarlo (no necesita obstáculos como piedras, irregularidades en el terreno, etc.). Para imitar los extremos de las cintas transportadoras y, teniendo en cuenta que tampoco se quería que el robot se pasee por debajo de ellas, un cubo sólido parecía la solución idónea para imitar esa parte.

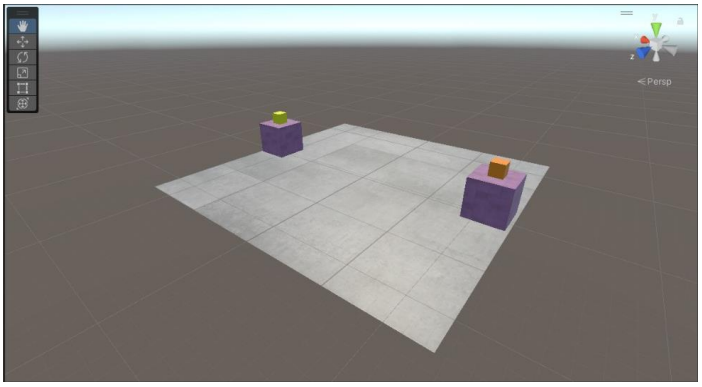

Figura 20. Modelo del escenario base

Como resultado de la tarea, se obtuvo el escenario base que se muestra en la Figura 20.

Una vez adaptados a un tamaño, forma y ubicación apropiados, podemos pasar al siguiente nivel. Sin embargo y, aunque no es obligatorio, se decidió añadir texturas a todos los objetos de la escena para mejor visualización del entrenamiento.

5.3.3. OB2-T02. Importación y ajustes del modelo UR10e

Ahora que tenemos el modelo de su entorno de trabajo, es hora de colocar virtualmente al brazo robótico. Para ello, es necesario modelarlo. Es importante que el modelo sea lo más fiel posible a la vida real ya que podría causar problemas de colisiones en la vida real si el robot entrena con un modelo que no se parezca al suyo. Sin embargo, tal tarea no es necesaria hacerla a mano ya que siendo un producto de una empresa de considerable tamaño, hay muchas posibilidades de que ya existan modelos 3D de dicho brazo. Lo más probable es que sea la misma empresa la que los ofrezca. De no ser así, existen repositorios de uso público de modelos 3D de todo tipo, no sería extraño encontrar un modelo del UR10e en uno de ellos.

El modelo fue descargado desde un repositorio gratuito de modelos 3D [41]. También se consiguieron unas texturas metálicas gratuitas, pero terminaron por no usarse en el proyecto final, sólo en los proyectos previos por aprendizaje [42]. Por supuesto, el modelo no estaba en un formato adecuado para Unity (.obj) así que se utilizó un formateador online [43].

Una vez importado el modelo, había que adecuarlo a nuestras necesidades. Lo primero a realizar era reducir el tamaño del modelo del robot a una escala manejable dentro del entorno de Unity, ya que el modelo descargado parecía ser a tamaño real y eso no es ideal para una primera toma [44] [45].

Lo segundo era quitar cualquier elemento superfluo del modelo. Cuantas menos figuras 3D tenga la escena, más eficiente será el entrenamiento. Para ello, se modificó el modelo para eliminar la tornillería, cables, embellecedores, monogramas en relieve, etc. Lo importante era evitar cualquier elemento que en la vida no suponga un impedimento para el movimiento del robot en cuanto a colisiones. Todas estas piezas eran internas o despreciables para este propósito (véanse Figura 21 y Figura 22).

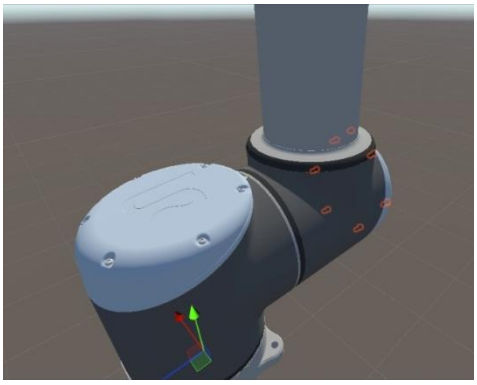

Figura 21. Modelo completo con demasiados elementos

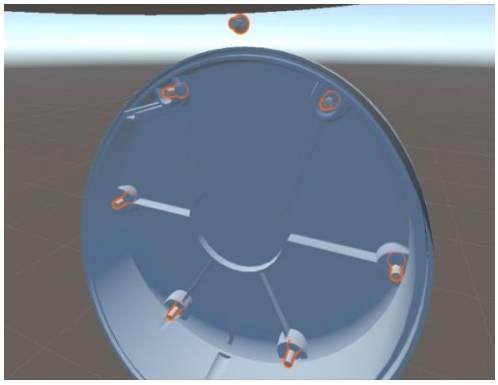

Figura 22. Ejemplo de tornillería innecesaria

Al completar esta tarea, el brazo robótico, importado y modificado, estaba situado dentro del escenario base.

5.3.4. OB2-T03. Modelado de obstáculo

Parte del proyecto es que el brazo robótico sea capaz de esquivar un obstáculo que podría estar en cualquier parte de su entorno de trabajo. Por ello, es importante diseñar un obstáculo que abarque las formas que éstos pueden tomar en la vida real.

El modelo del obstáculo fue simple de realizar, lo complicado fue pensar la forma adecuada. En la vida real el robot no sabrá la forma del obstáculo que debe esquivar. Sólo sabe la distancia y dirección a la que hay un obstáculo. En otras palabras, si fuesen coordenadas 3D, sabría la (X, Y, Z) del punto más cercano del obstáculo a su detector. Por lo tanto, se debía tomar una decisión sobre qué forma darle a dicho obstáculo.

Se optó por un cilindro de altura "infinita", como si envolviese el obstáculo en un cilindro protector del cual hay que alejarse y no tocar, como se puede apreciar en la Figura 23, que es un ejemplo real de la solución descrita. De esta forma, a menos que el obstáculo real tenga una forma más ancha por encima del detector que su propia base, no habrá problema. El diámetro del cilindro es uno escogido en base a mi experiencia, que no es mucha. Al no disponer de las dimensiones reales de los obstáculos, se decidió determinar el diámetro del obstáculo en función de las observaciones obtenidas durante los meses que trabajé en ASAI. El resultado final se puede ver en la Figura 24. Es cierto que, aunque no sea una dimensión completamente precisa, modificar el diámetro para entrenar el cerebro con otras dimensiones es un cambio sencillo que sólo debería suponer la modificación de dos variables de tipo Float.

Figura 23. Ejemplo visual de cilindro protector, suponiendo que el sombrero es el obstáculo [46]

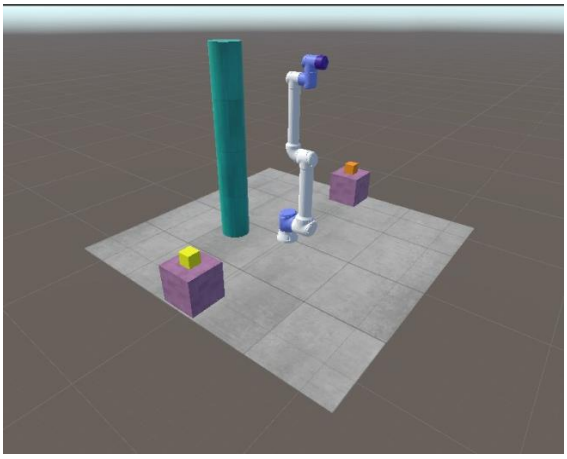

Figura 24. Modelo final del obstáculo. Es el cilindro de color verdoso

De todas maneras, de querer implementar el cerebro en un entorno real, yo recomendaría realizar un estudio exhaustivo en base a las circunstancias de cada escenario. En este caso, ASAI informó de que, de momento, se pretendían esquivar cajas apiladas en la zona de trabajo, así que envolver dichas cajas en un cilindro virtual es más que suficiente. En un futuro, sería interesante realizar entrenamientos con otras formas de obstáculo dinámico.

5.3.5. OB2-T04. Modelado de objetivos

Al igual que existe un obstáculo, existen objetivos. Estos representan los objetos que el robot debería mover en la vida real en una tarea de "pick and place".

Para esta tarea, los objetivos bastaban con ser cubos primitivos de un tamaño adecuado, puestos con precisión sobre las cintas transportadoras. La mayoría de las cosas que movería este robot son cajas cúbicas o prismáticas, así que la solución fue simple, pero efectiva. De hecho, se pueden apreciar los prototipos de los primeros objetivos en la Figura 20, siendo éstos los cubos amarillo y naranja.

5.3.6. Resumen del Sprint 3

Durante este Sprint, acabé los tutoriales de Unity que me dejé a medias en el anterior Sprint. Procedí a modelar la escena y a preparar el modelo del UR10e para posterior configuración programática. Una vez acabado el modelado de todas las figuras, me percaté de que iba con tres días de adelanto respecto al calendario propuesto. Esto hizo que planificase el siguiente Sprint y comenzase antes de tiempo con la siguiente tarea, OB2-T05.

Pude comenzar a investigar el mejor acercamiento posible para el resto del diseño de la escena (colisiones, articulaciones, restricciones, etc.). Tras aprender de varios tutoriales [46] [23] [47] [48] pude apreciar cuál sería la mejor forma de abordar el problema. Aun así, decidí dedicarle un día más al estudio de dichos tutoriales para estar seguro del todo.

5.4. Sprint 4

En este Sprint, se abordó casi toda la programación al uso. Tocaba configurar las articulaciones, restringir movimientos y programar tanto la lógica de colisiones como la de castigo y recompensa. Específicamente, la lógica de colisiones es la que ayudará al Agente a aprender qué trayectorias son correctas y cuáles no. Un Agente es como se llama al objeto que contiene los scripts relacionados con Machine Learning y, para el contexto de este proyecto, es prácticamente sinónimo de "robot". Como se ha comentado al final del Sprint anterior, este Sprint comenzó con un poco de adelanto y la primera tarea abordada fue OB2-T05. Por lo tanto, la lista de tareas a completar para este Sprint es:

- OB2-T05: Investigar mejor acercamiento posible
- OB2-T06: Crear y configurar articulaciones
- OB2-T07: Restringir movimientos
- OB2-T08: Programar lógica de colisiones
- OB2-T09: Programar lógica del entrenamiento por refuerzo

Para cada una de estas tareas, se describirá a continuación el trabajo realizado, además de sus problemas, soluciones y resultados si procede.

5.4.1. OB2-T05. Investigación sobre el mejor acercamiento posible

Tocaba pensar en la manera que se iba a abordar el tema del entrenamiento. MLAgents ofrece dos tipos distintos de entrenamiento con Machine Learning, por refuerzo o por imitación. El entrenamiento por refuerzo consiste en premiar y castigar al robot cuando realice acciones, dependiendo de si son deseables o no. El entrenamiento por imitación consiste en imitar una serie de demostraciones pregrabadas por un operario.

Tras conseguir experiencia con los tutoriales mencionados en el Sprint anterior, se decidió por un acercamiento de entrenamiento por refuerzo, ya que uno de imitación no parecía adecuado para una tarea cambiante.

5.4.2. OB2-T06. Creación y configuración de las articulaciones

Después de modelar visualmente la escena, era necesario darle directrices al robot sobre el mundo real (colisiones, posibles efectos de la gravedad…), ya que no se podía permitir que el robot se retorciera sobre sí mismo o hiciera movimientos físicamente imposibles. Para ello, se comenzó configurando las articulaciones del modelo del UR10e.

En Unity, existen muchos tipos de articulaciones programables, pero para este tipo de proyecto, consideré que una Configurable Joint es la adecuada [23] [47]. Este tipo de joint permite elegir el punto de anclaje entre piezas, limitar los ejes de rotación y limitar el movimiento lineal en el espacio.

Existen dos formas de rotar una pieza, de forma Global o de forma Pivote. Global utiliza el centro de masa de cada pieza, y Pivote utiliza el punto de origen de la pieza. Era necesario moverse en Pivote, porque en un robot no se puede hacer que un brazo se mueva por su centro de masa, ya que eso es imposible. Debe rotar en el eje de su rotor, que se imita con el punto de origen.

Se añadió un Configurable Joint para cada una de las articulaciones del robot pero, por una limitación del modelo descargado, no todas las piezas están en su posición local (0, 0, 0), y esto hizo que rotar las piezas en Pivote fuese imposible, ya que no coincidían con el eje del rotor imaginario.

Para solucionar este problema, se creó un objeto vacío, se colocó en la posición adecuada, y se hizo que el objeto modelo fuese su hijo en la jerarquía (véase Figura 25). De esta manera, pude actuar sobre el objeto vacío padre, que está en la posición adecuada. Cualquier movimiento en el objeto vacío padre afectaría a su hijo, el objeto modelo, de la forma que deseaba. Sin embargo, fue necesario ejercer extrema cautela con la jerarquía, ya que este tipo de clasificación de padres e hijos puede dar resultados inesperados si no se realiza con cuidado.

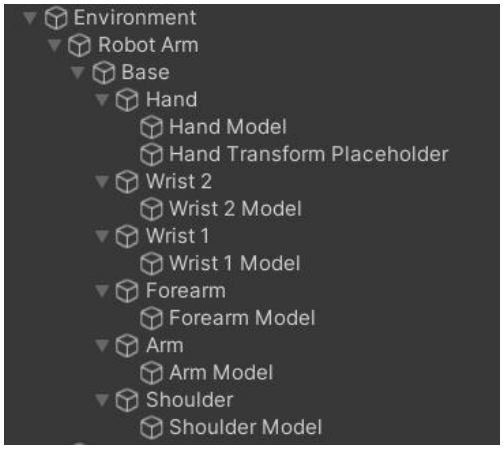

Figura 25. Jerarquía inicial

Se pudieron anclar las piezas con los Configurable Joints utilizando las posiciones de los padres vacíos y los objetos modelo. En lugar de anclar el hombro con la base, se podía anclar el padre vacío del hombro con el modelo real de la base, y así con todas las piezas. Seguidamente, tocaba calcular el centro de anclaje. Para ello, se utilizó la visión Wireframe del editor de Unity y puse la vista en isométrica, para poder calcular al micrómetro el centro exacto de cada articulación. Wireframe permite visualizar el entorno como si fuera un esquema de dibujo técnico, como se puede apreciar en la Figura 26.

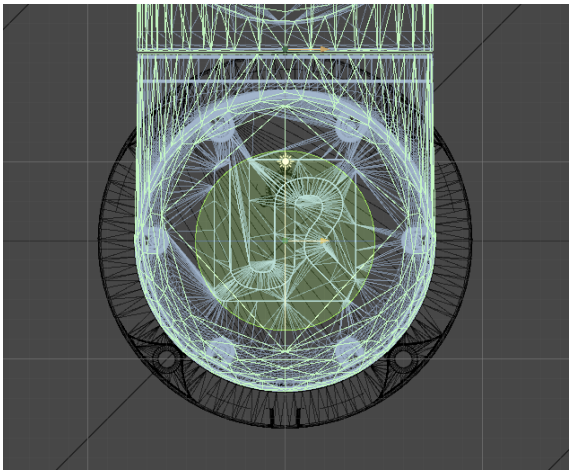

Figura 26. Vista Wireframe del anclaje Base-Hombro

El círculo amarillo en la Figura 26 simboliza el anclaje, su centro debe coincidir con el centro de giro de la articulación para un modelado correcto. Para realizar esto, realicé cálculos de vectores, mucho ojo y mucha paciencia.

Una vez acabadas las configuraciones, tocaba programar sus movimientos y utilizar el comando `addForce()` para ver si todo funcionaba correctamente. El comando `addForce()` aplica una fuerza a una pieza sobre su punto de giro, que como se ha explicado antes, depende de si estamos en modo Global o Pivote.

Lamentablemente, utilizar únicamente Configurable Joints provocó que los movimientos del robot tuviesen una inercia que no se consiguió controlar al 100%. Es cierto que aumentando la masa de cada pieza se consiguió que la inercia se volviera menos severa y los movimientos menos flexibles (cosa que deseamos en un robot), por desgracia, no se eliminó del todo. Este no es el comportamiento esperado en un robot real.

No obstante, existían otras posible soluciones. Se querían mantener las limitaciones que los Configurable Joints aportaban a los movimientos del robot (lmitaciones de movimiento en ejes, lineales…). Se decidió simplificar la forma de mover las piezas, utilizando movimientos por jerarquía en vez de movimientos por addForce(). Se combinaron los métodos de anclaje por Configurable Joints y por jerarquía con los centros de pivote vacíos que se habían creado antes. Esto resultaba en una jerarquía más compleja que la anterior (véase Figura 27), pero con una funcionalidad perfecta. Con esta configuración, era posible hacer uso del comando `rotate()` [48] que es mucho más sencillo e imita mejor los movimientos sin aceleración de un robot colaborativo.

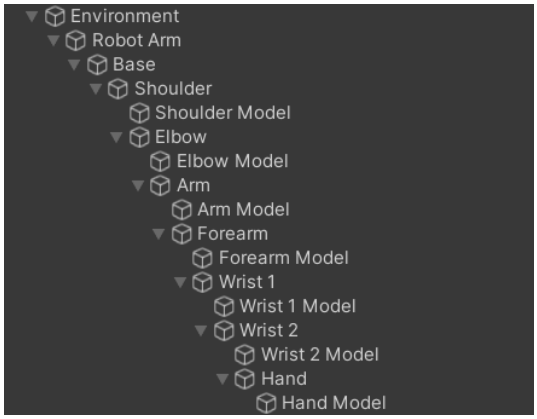

Figura 27. Jerarquía combinada

5.4.3. OB2-T07. Restricción de movimientos

Todos los robots del mundo, sean brazos, drones o perros de juguete tienen un límite en el cual pueden mover sus articulaciones. Un brazo robótico no puede desplazarse paralelo al suelo, desconectándose de su base. Este tipo de movimientos debían ser limitados.

Es importante limitar cada uno de los joints al eje en que se va a mover, bloqueando siempre todas las otras posibilidades y bloqueando por defecto cualquier movimiento lineal. No quería que el hombro se moviese tanto en el eje Z como en el eje X, este robot no tiene articulaciones esféricas que permitirían una rotación ilimitada. Esto se puede hacer de forma sencilla con Unity gracias al componente Configurable Joint. Sin embargo, lo que habría que programar a mano son las limitaciones de rotación.

Gracias a la hoja de especificaciones del brazo robótico UR10e, se puede ver que, como casi todos los brazos robóticos, las articulaciones tienen un límite de rotación, y debía ser respetado [49]. Cada articulación tiene un límite de rotación de ±360º, lo que significa que cada articulación podía moverse una vuelta completa en cada dirección y nada más.

Unity, normalmente, utiliza Quaterniones de Euler para llevar registro de las rotaciones de los objetos. No obstante, esto provocaría un problema para la programación. Los Quaterniones de Euler van desde 0 a 360, si diesen más de una vuelta, el Quaternión no mostraría un valor de 365º, por ejemplo. Volvería a 5º y, si fuese en otra dirección, no mostraría un valor negativo como -40º, si no el valor de 320º. Así no era posible llevar un registro de cuántas vueltas había dado cada articulación. Para ello, declaré una variable de tio Float que guardaba la rotación total de cada articulación, llamada `rotationSum`. Cada parte del robot tenía su propio `rotationSum`.

Gracias a la variable `rotationSum`, era posible comprobar el estado de rotación de cada articulación y limitar los movimientos si procediese. Antes siquiera de dejar que una acción del robot altere la rotación de una articulación, debía comprobar si había llegado a su límite, añadiendo el movimiento que planeaba hacer al `rotationSum`. Si éste era mayor que 360º o menor que -360º, no podía permitir que se moviera.

```
//Logic to clamp the maximum and minimum rotation of the Joint to +-360º
elbowRotationSum += moveElbow * rotateSpeed * Time.deltaTime;
if (elbowRotationSum >= 360)
{
    elbowRotationSum = 360;
}
if (elbowRotationSum <= -360)
{
    elbowRotationSum = -360;
}
if (elbowRotationSum > -360 && elbowRotationSum < 360)
{
    elbow.Rotate(0, 0, moveElbow * rotateSpeed * Time.deltaTime, Space.Self);
}
```

Figura 28. Snippet de clampeo de rotación de articulaciones

En la Figura 28 se muestra sólo un fragmento de código para el movimiento del codo, pero con ésta lógica pude limitar la rotación de cada articulación correctamente.

5.4.4. OB2-T08. Programación de la lógica de colisiones

Es importante explicar qué son las colisiones y por qué son importantes. En un entorno real, cuando una mano humana toca una mesa, se está produciendo una colisión entre la mano y la mesa. Debido a la solidez de ambos objetos, la mano no puede atravesar la mesa, se ve detenida por la fuerza de reacción que genera la mesa sobre la mano.

En robótica es lo mismo. Si una pieza del robot tocase otra pieza, o el suelo, o una persona, se habría producido una colisión. La diferencia con el ejemplo de la mano y la mesa es que el robot puede romper el objeto con el que haya colisionado o viceversa. Ambos escenarios son catastróficos en el ámbito de trabajo de un robot.

Para evitar estos escenarios, es necesario que el robot que se esté entrenando sea consciente en todo momento de cualquier posible colisión que se haya producido en su entorno. Para ello, hablaré del cálculo de colisiones y de cómo han sido manejadas.

Para el cálculo de colisiones son necesarios dos componentes: Rigid Body y un Mesh Collider [50]. Es necesario incorporar estos componentes en cada uno de los objetos del escenario.

El componente Rigid Body simplemente le da una "solidez virtual" a un objeto. Esto le permite a Unity calcular colisiones y no permite que dos objetos ocupen el mismo espacio al mismo tiempo. Un Mesh Collider es una malla de puntos que le dice a cualquier objeto de la escena (incluido él mismo) dónde están los límites de su cuerpo rígido (Rigid Body). Es decir, delimita las superficies de un objeto para cálculos de colisiones. Cuanto más fina es la malla, más precisa es la definición de su superficie. El mejor ejemplo visual sería un jamón en una malla de carnicería (véanse Figura 29 y Figura 30), donde el Rigid Body sería el jamón y el Mesh Collider la malla.

Figura 29. Ayuda visual para Rigid Body y Mesh Collider 1 [51]

Figura 30. Ayuda visual para Rigid Body y Mesh Collider 2 [51]

En la Figura 30, se puede apreciar mejor que la forma del jamón es más compleja que la forma de la malla, y que ésta no se adhiere perfectamente a la superficie del jamón (más visible cerca de la etiqueta). Sin embargo, por motivos de procesado, una malla suele ser siempre una forma simplificada de la forma real del objeto. Se delimita con el número de polígonos. Puedes tener la figura más compleja de la historia, pero, a la hora del cálculo de colisiones, necesitas una malla simplificada para no "freír" el procesador. De esta manera, sin el Mesh Collider, el Rigid Body no sabe qué forma y tamaño tiene su cuerpo para el cálculo de colisiones.

Debido a la complejidad de las piezas, no se puede realizar un Mesh Collider lo suficientemente sofisticado para emular la superficie real de las piezas, igual que en el caso del jamón. Cada Mesh Collider tiene un límite máximo de 256 polígonos. Sin embargo, tras una inspección más a fondo de cada pieza en Wireframe (un tipo de visión de entorno dentro de Unity), las formas de la mallas generadas son perfectamente válidas para el cálculo de colisiones, así que servirán cuando haya que realizar los cálculos de colisiones.

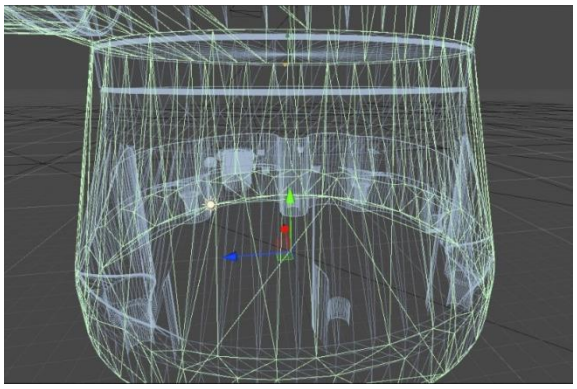

Figura 31. Ejemplo de Mesh Collider (verde) en la base

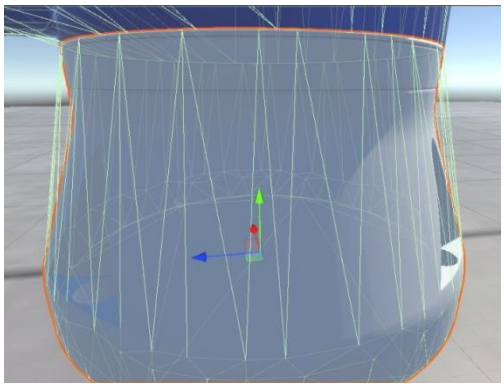

Figura 32. Base visible en comparación con el Mesh Collider (verde)

En la Figura 31 y la Figura 32 se puede apreciar que el Mesh collider no se ajusta exactamente a la superficie de la base. Sin embargo, tras una inspección minuciosa, no deja que ninguna

parte de la superficie no esté cubierta (aunque no ajustada). De tal manera, es perfectamente válido para el cálculo de colisiones.

Se asignó a cada parte del entorno un Mesh Collider y un Rigid Body, componentes que posibilitaban llevar un registro de las colisiones que ocurriesen en el entorno. Ahora tocaba programar un script para cada parte del entorno. Algunos de ellos podían estar vacíos, ya que sirven únicamente para identificar qué tipo de pieza es en el entorno (Mesa, PiezaX del robot, Carga, Suelo, etc.). Otros, debían tener programada una lógica para el manejo de colisiones.

Dada la naturaleza del entorno, las partes de la escena que debieran preocuparse de manejar las colisiones eran las partes móviles, ya que, en este escenario, una mesa nunca se va a chocar con nada, sino que se van a chocar con ella. De esta manera, sólo las partes móviles del brazo robótico tenían lógica de colisiones.

El robot tiene 6 articulaciones y consta de 8 partes importantes: Base, Hombro, Codo, Brazo, Antebrazo, Muñeca1, Muñeca2 y Mano. La base podía ser ignorada y considerada como parte del entorno, ya que nunca se movería, así que ésta no necesitaba script de colisiones.

Era imprescindible que cada pieza del entorno fuese declarada como "Trigger" en su componente Mesh Collider (véase Figura 33).

Figura 33. Componente Mesh Collider

Seguido, se programó la función interna `OnTriggerEnter()` para cada una de las 7 partes del robot. De manera que cuando la función detecta una colisión, se penalizará al modelo entrenado.

Como ya se ha comentado, se va a aplicar Machine Learning para entrenar un modelo que será el encargado de crear la trayectoria del brazo robótico. En concreto, se va a utilizar aprendizaje supervisado por refuerzo. Es decir, que el modelo entrenado (también llamado Agente) será recompensado cuando cree una trayectoria correcta y penalizado en caso de que cree una trayectoria incorrecta. Por ejemplo, será penalizado cuando alguna de las articulaciones colisione.

Por lo tanto, si existe una colisión, se espera que la función del Trigger penalice o castigue al Agente. Sin embargo, el Agente no forma parte del brazo robótico. El Agente es otro objeto dentro del entorno. Así que es necesario comunicar dos Scrpits (el script de la articulación que colisiona y el script del Agente) para poder penalizar al Agente.

Para solucionar este problema se encontraron las siguientes soluciones:

- Utilizar workers como en otros lenguajes. Sin embargo, en Unity y con C# cada script es un worker en sí mismo.

- Crear un Tag, o Etiqueta, a cada pieza del entorno [52], y entonces podía encontrar el objeto desde un script, con la función:

```
GameObject.FindGameObjectWithTag("AgentTag").Getcomponent<BrainAgent>();
```

  Sin embargo, esto impediría realizar entrenamiento masivo en el futuro, ya que habría múltiples Agentes con la misma etiqueta en la escena.

- Pasar el Agente como variable a cada objeto que detecte colisiones [53], con esta línea:

```
[SerializeField] private BrainAgent agent;
```

  En este caso, BrainAgent es el nombre de una clase personalizada de Agente creada para el proyecto. Esta clase hereda de la clase "Agente". Una vez el objeto tiene el Agente pasado como variable, puede acceder a las funciones del Agente. Además, tales funciones están programadas para afectar sólo al Agente, que es lo que me interesa.

Dadas estas tres soluciones, se implementó la tercera solución, donde el objeto tiene el Agente como variable, lo cual le permite llamar a las funciones del Agente, como `agent.collisionInEnvironment()` ó `agent.collisionWithObstacle()`.

Como resultado de la tarea, se crearon los Rigid Body y Mesh Collider necesarios, y se programaron los scripts de todas las partes móviles del brazo robótico.

5.4.5. OB2-T09. Programación de la lógica del entrenamiento por refuerzo

En esta subsección no sólo relataré el progreso y las tareas realizadas sino que también planeo explicar con toda la posible brevedad el funcionamiento básico de Machine Learning con MLAgents.

5.4.5.1. Introducción al ciclo de entrenamiento por refuerzo

Para programar con MLAgents, hay que designar un objeto del entorno como "Agente". Este Agente será el que lleve la lógica del entrenamiento de Machine Learning. MLAgents soporta dos tipos diferentes de entrenamiento: por refuerzo y por imitación [54]. Para empezar, refuerzo parecía el más indicado, ya que quería ver hasta dónde puede aprender por su cuenta sin demostraciones humanas.

El método por refuerzo se basa en un ciclo de 4 fases:

- Observar
- Decidir
- Actuar
- Recibir Recompensa o Castigo
- [Empezar de nuevo]

Todas estas acciones se definen en diferentes componentes dentro del mismo objeto que contiene al "Agente".

El Agente primero **observa** su entorno usando la función `CollectObservations(VectorSensor)`. Estas observaciones se guardan en un Vector de

Observaciones. El vector de observaciones se define en el componente Behavioral Parameters, uno de los 2 componentes necesarios para el ciclo de entrenamiento.

Estas observaciones son Floats, simples números, no pueden ser algo complicado como la forma de un objeto, el color de una flor, etc. Pero pueden traducirse a cualquier cosa. En la escena del proyecto, estos Floats indicarían la posición de los diversos elementos que la componen. Serían la posición "X", "Y" y "Z" de cada una de las piezas (mesas, cargas, partes individuales del robot…), el ángulo de rotación de cada una de las piezas del robot, e incluso indicarían si la primera tarea de la Política ha sido realizada o no. En Machine Learning la "Política" es el conjunto de tareas que el cerebro neuronal debe realizar. En este caso, esta Política era agarrar una pieza y llevarla hasta otro sitio esquivando obstáculos. Todas esas acciones juntas forman la Política.

Es importante definir el tamaño del Vector de Observaciones correctamente (número de Floats a observar) antes de iniciar el entrenamiento, o podrían perderse observaciones (en caso de definir un vector demasiado pequeño) o inventárselas (en caso de definir uno demasiado grande). En ambos casos, afectaría negativamente y, muy frecuentemente, de forma catastrófica al resultado del entrenamiento.

Para tomar **decisiones**, es necesario incorporar al Agente un componente llamado Decision Requester, el segundo de los componentes necesarios para el ciclo de entrenamiento. Este componente toma decisiones sobre las acciones disponibles en base a las observaciones que el Vector de Observaciones ha recogido durante cada "step" del entrenamiento. Lamentablemente, Python y C# (el lenguaje en el cual está programada la lógica de la escena y el ciclo de entrenamiento) tienen diferentes definiciones de lo que significa step, pero debe dejarse claro para la explicación del resto del desarrollo.

Según el glosario de MLAgents, un step "corresponde a un cambio atómico del motor que ocurre entre decisiones del Agente". [55] Sin embargo, esta definición es muy abstracta, así que se intentará simplificar.

Desde el punto de vista de C#, los steps es lo que ocurre entre ticks de la función intrínseca de los scripts de Unity `Update()`. Update es una función que el motor ejecuta en cada script en funcionamiento durante cada frame. Si el entorno se mueve a 60 fps, o frames por segundo, entonces es una función que se llama 60 veces por segundo.

Para Python, un step consiste en un intervalo de decisiones. Para ello, se utiliza en su lugar la función `FixedUpdate()`. FixedUpdate es una función que el motor de físicas utiliza para los cálculos de físicas. Se llama aproximadamente una vez cada 0.02 segundos, o 50 veces por segundo.

Por simplicidad, un step es un periodo en el cual el Agente toma una decisión sobre el vector de acciones que exploraremos más adelante. Pueden ser referidos como los "frames" en los que el Agente entrena. Usando la palabra "frame" aquí como mero símil de videojuegos para entender el funcionamiento de los steps.

Estas decisiones que toma el Agente, se basan en las observaciones que ha tomado el Vector de Observaciones, como se ha explicado antes. Según avance el entrenamiento, el Agente empezará a entender qué efecto tienen sus acciones en el Vector de Observaciones.

Un ejemplo simple sería un objeto que el Agente mueve de arriba a abajo, y que éste termina aprendiendo que sus acciones sobre el Vector de Acciones afectan a las posiciones de ese objeto, que son recogidas por el Vector de Observaciones.

A la hora de tomar **acciones**, el Agente utiliza el campo de Vector de Acciones en el componente Behavioral Parameters. Al igual que con el Vector de Observaciones, es necesario definir el tamaño de este vector correctamente antes de iniciar el entrenamiento, concurriendo en caso contrario los mismos posibles efectos adversos que en el Vector de Observaciones, sólo que en este caso, serían más severos y evidentes.

En este caso, las acciones a tomar eran 6, una por cada articulación. Cabe recordar que un Agente sólo entiende de números, así que una acción no puede ser simplemente "mueve el brazo hacia arriba". Es el momento adecuado para explicar cómo el Agente realiza acciones.

El Vector de Acciones puede ser de dos tipos: de acciones discretas o continuas. Una acción sólo puede ser un número, que luego puede traducirse en otro tipo de acciones. Es decir, el Vector de Acciones, de tamaño 6 en el caso que nos ocupa, no es más que un vector cero de tamaño 6, (0, 0, 0, 0, 0, 0). El tamaño del vector es determinado por el número de acciones que debe tomar el Agente.

Para tomar acciones, el Agente altera cada uno de esos 6 números con la función `OnActionReceived(ActionBuffers)`. Si el vector es de acciones discretas, los números serán números enteros comprendidos desde el 0 hasta el número máximo que especifiquemos. En este caso, si se pusiera todo el `vectorDeAcciones[6]` limitando el número máximo a 5, el Agente alteraría aleatoriamente los números de cada posición del vector del 0 al 4.

Sin embargo, siendo que se quieren traducir estas acciones a rotaciones de cada articulación, no es deseable usar acciones discretas, si no continuas para poder utilizar Floats. Un Vector de Acciones Continuas, en este caso un vector[6], por defecto (0, 0, 0, 0, 0, 0), altera aleatoriamente los números de cada una de las posiciones del vector desde -1 hasta 1, y todos los números entre ellos. Estos valores se pueden traducir en movimientos de cada una de las articulaciones de un robot. Siendo que éstas se mueven en grados, interesa que sean Floats, y no Integers, para poder mover una articulación unas fracciones de grado en vez de grados enteros.

Se puede coger el valor Float de la primera posición del vector y decirle al Agente, dentro de la función `OnActionReceived()`, que mueva una de las articulaciones del robot con una fórmula simple

```
robotPart.rotate.(vectorDeAcciones[0]*rotateSpeed*Time.deltaTime,    0,    0,
Space.Self);
```

Este es un ejemplo simple de cómo se puede utilizar el Vector de Acciones para realizar acciones en el entorno 3D. La variable `rotateSpeed` simplemente controla la velocidad de rotación, pero no es el único factor en ello. El Float real del vector también afecta a la velocidad final, ya que si es 1 sería la velocidad máxima que esa función podría alcanzar, o podría ser 0.5 e ir a media velocidad.

El que sea una acción continua no es sólo útil para que se pueda ir con un infinito abanico de diferentes velocidades, si no que si el número sale negativo, como -0.3687 por ejemplo, el

brazo rotará en la dirección contraria, con lo que con un solo espacio del vector de acciones, se puede girar una pieza tanto hacia un lado como para el otro. De esta manera, el Vector de Acciones Continuas puede controlar las 6 articulaciones del robot con un único vector de 6.

Hasta ahora se ha explicado cómo el robot se comunica con su entorno (Vector de Observaciones), cada cuánto toma una decisión (Decision Requester) y cómo realiza una acción (Vector de Acciones).

Pero, ¿cómo sabe el Agente si las acciones que toma son correctas o incorrectas? ¿Cómo decide si girar a la izquierda o a la derecha una determinada parte del robot? Es sencillo: al principio, no lo sabe.

En un entorno de entrenamiento por refuerzo, un Agente simplemente altera aleatoriamente los números de su Vector de Acciones y comprueba de qué manera esos cambios aleatorios han afectado al Vector de Observaciones. Según siga entrenando, inferirá con mayor precisión qué acciones o conjunto de acciones afectan de qué manera a cada una de las posiciones del Vector de Observaciones.

5.4.5.2. Ejemplo del Pong

En este **ejemplo** simple -pensemos en un juego 2D como el Pong (véase Figura 34), el primer videojuego de la historia- el Vector de Observaciones estaría compuesto por las coordenadas 2D de la pelota, y las posiciones de ambos "jugadores" (las líneas que se mueven de forma vertical para evitar que la pelota salga por los lados de la pantalla.

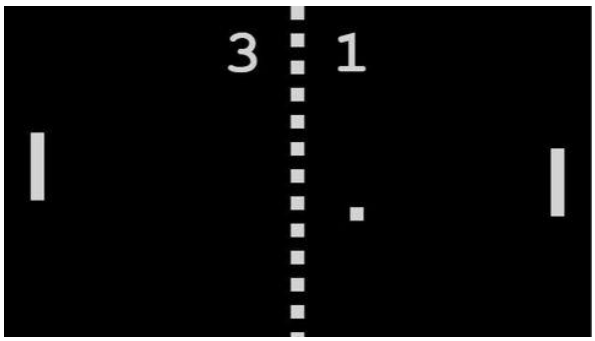

Figura 34. Imagen de una partida de Pong [56]

El Vector de Acciones estaría compuesto por dos números, cada uno controlando a uno de los jugadores (en este ejemplo estaríamos programando un juego de Pong que juegue solo, sin jugadores humanos), valores negativos harían que el jugador se moviera a la izquierda y valores positivos que se moviera a la derecha, ambas acciones con una velocidad determinada por el tamaño del Float en el Vector de Acciones, aunque podría programarse con acciones discretas para una velocidad fija.

Al principio, el Agente movería arriba y abajo a ambos jugadores de forma errática. Con el tiempo y, observando la posición de la bola en cada step, irá infiriendo que si la posición "Y" de los jugadores no corresponde con la misma que la de la pelota, la pelota termina saliendo por la pantalla, y eso no es **deseable**. Con el tiempo, el Agente aprenderá de qué manera sus acciones afectan a las posiciones de los jugadores. Con el tiempo, inferirá que es deseable que la pelota se mantenga en la pantalla, empezará a alterar los números del Vector de Acciones de

tal manera que las posiciones "Y" de los jugadores sean la misma que la de la pelota, y así se convertirán en dos jugadores que jamás podrán perder una partida, creando partidas eternas.

Con este ejemplo se entiende cómo el Agente aprende a relacionar sus acciones con las observaciones, pero se ha planteado otro asunto importante. ¿Cómo sabe el Agente que **no es deseable** que la pelota salga de la pantalla? En este momento es donde entra el último paso del ciclo: Recibir Recompensa o Castigo.

Siguiendo con el ejemplo del Pong. El Agente ha aprendido a mover los jugadores, y qué efecto tienen cuando "bloquean" la pelota (chocar las barras, o jugadores, con la pelota). Sin embargo, no sabe que lo que tiene que hacer es precisamente bloquear la pelota, así que si no se implementa un sistema de recompensas y castigos, nunca aprenderá a jugar y seguirá moviendo aleatoriamente a los jugadores a perpetuidad.

Para ello, puede indicársele al Agente que, el hecho de que pelota se "choque" con uno de los bordes horizontales de la pantalla (es decir, se salga de la pantalla) no es deseable, y se le puede indicar dándole un castigo.

Un Agente de entrenamiento por refuerzo sabe lo que es un **castigo** y una **recompensa**, ya que es un sistema interno dentro de cualquier aplicación de Machine Learning que soporte entrenamiento por refuerzo. De tal modo, cuando la pelota se choque contra el borde de la pantalla, se le puede decir al Agente que su recompensa baja, con la función `AddReward(Float)`. Cuanto mayor es la recompensa, más piensa el Agente que está realizando la tarea correctamente, con lo que bajar su recompensa es un castigo.

Si se choca con la pantalla, se le bajaría la recompensa acumulada utilizando la función `AddReward()` con un valor negativo, como `AddReward(-1f)` y, por cada segundo que la pelota esté en juego, le podemos dar una recompensa con `AddReward()` con valor positivo, como `AddReward(0.1f)`. Un Agente siempre intentará conseguir la mayor recompensa acumulada posible en cada episodio.

Un episodio es un grupo de steps que empieza desde que se llama la función `OnEpisodeBegin()` y hasta que se llama la función `EndEpisode()`. En el caso del Pong, el episodio empezaría cuando empieza la partida, y se acabaría cuando la pelota sale por la pantalla. Ahora, gracias a que el Agente infiere el efecto de sus acciones en el entorno y que sabe qué observaciones generan recompensa y castigo, aprenderá poco a poco a mantener la pelota en juego durante el mayor tiempo posible, para aumentar la recompensa acumulada todo lo que pudiese.

5.4.5.3. Progreso con el modelo real

El escenario modelado con robot, suelo, "mesas" y objetos a trasladar estaba preparado. Ya había sido definido el Vector de Observaciones, el Decision Requester y el Vector de Acciones Continuas. Si se hubiera intentado iniciar el entrenamiento ahora, el robot se habría movido de forma aleatoria, probablemente espástica y, muy poco después, casi seguro en el primer episodio, ocurriría algo que no puede pasar en la vida real: el robot atravesaría el suelo.

Para solucionar esto, es necesario implementar un sistema de recompensas y castigos coherente a la Política. Se debe castigar al Agente cuando haga algo que no puede o no debe

hacer, y premiarlo cuando haga algo que debe hacer. El primer paso, antes de programar nada, es hacerse las preguntas clave: ¿Qué **puede**, **no puede**, **debe** y **no debe** hacer el Agente?

En este caso, lo que no podía ni debía hacer el Agente era permitir que existiese ninguna colisión indeseada en el entorno. Ninguna parte del robot puede ni debe chocarse ni con el suelo, ni con las mesas, ni consigo mismo. Y, dada la naturaleza del robot y la Política del Agente, sólo la "mano" del robot puede tocar la carga a mover.

Respondiendo a estas preguntas, uno pude rápidamente entender en qué condiciones se debe premiar al Agente, y en cuáles castigarlo.

- Si ocurriera alguna colisión que no sea mano-carga, castigo.
- Si ocurriera una colisión mano-carga, premio.
- Si el robot llevara la carga al punto final designado (la otra mesa), premio extra.

Al programar el ciclo de entrenamiento por refuerzo, se decidió seguir el orden lógico establecido con anterioridad: Observación, Decisión, Acción y Recompensa.

5.4.5.3.1. Ciclo de refuerzo: Observación

`OnEpisodeBegin()` es una función interna que se lanza automáticamente cuando un episodio empieza. Dentro de esta función deben programarse todas las acciones que se deben realizar para tener el escenario en un estado de inicio, desde la posición de los objetos hasta el estado de las flags.

En primera instancia, era necesario poner todas las piezas del robot a las posiciones de inicio, que en este caso es un robot erguido. Siendo que los movimientos y la posición de las piezas son controlados por rotación, es lógico devolver el robot a su posición utilizando también el valor de los ángulos de rotación.

Hay que hacer esto para cada pieza. Es importante que cuando se altera una de las variables de `Transform` de un objeto (posición, rotación y escala) se retoquen en local y no en global. De hacerlo en global, impediría el entrenamiento masivo una vez el entorno estuviese totalmente preparado.

Tras colocar el robot en la posición de inicio, debe procederse a colocar la carga en su posición original también. Y, ya que era parte integral del proyecto, debía reiniciarse la posición del obstáculo de forma aleatoria, siempre teniendo en cuenta que la posición no podía ser una que tenga una colisión antes siquiera de empezar el episodio (como posicionar el obstáculo dentro del brazo robótico, de una mesa o atravesando el suelo).

```
//Set random position of Obstacle
float tempRandomZ = nextFloat(-53, 52);
float tempRandomX = Mathf.Sqrt((Mathf.Pow(54, 2)) - (Mathf.Pow(tempRandomZ, 2)));
if ((((nextFloat(0, 1)) * 2) - 1) < 0)
{
    tempRandomX *= -1;
}
obstacle.transform.localPosition = new Vector3(tempRandomX, 80, tempRandomZ);
```

Figura 35. Invocación del obstáculo en posición aleatoria

Con la lógica mostrada en la Figura 35, se coloca el obstáculo en un punto aleatorio de un círculo con centro en el robot y radio apropiado para no provocar colisiones.

Ahora se debía tratar con el Vector de Observaciones, utilizando la función interna `CollectObservations()`. A esta función, se le debe pasar información sobre cualquier cosa que deba saber el Agente sobre su entorno o sobre sí mismo. Sin embargo, sólo es interesante pasarle información sobre cosas que puedan cambiar. Tanto el suelo como las mesas son estáticas permanentemente, el robot aprendería su posición con el tiempo y no necesitaba observarlas. Por supuesto, se le debe pasar información sobre el objetivo (la carga), a la que se llamará "goal" de ahora en adelante. Para pasar información, se debían llamar a los sensores del Vector de Observaciones. Para ello se utiliza esta lógica:

```
sensor.AddObservation(goalTransform.localPosition);
```

En esta función, cabe destacar que `goalTransform` es la variable `Transform` del objeto `Goal` y, de ahí se podía sacar la posición con `Position`, aunque es importante notar que se está usando `localPosition` ya que simplemente `Position` cogería los valores globales en vez de locales. Además, el Agente necesita saber cómo sus acciones afectan al movimiento del robot, con lo que es importante que reciba también la información de la posición de sus articulaciones y el ángulo de rotación de cada una de ellas.

```
sensor.AddObservation(elbow.localPosition);

sensor.AddObservation(elbowRotationSum);
```

Esta lógica ha de ser declarada para cada una de las partes móviles del robot. La base no es necesaria por la misma razón que no lo son las mesas y el suelo. Por supuesto, también debe serle pasada información sobre el obstáculo. Que el Agente reciba información de la posición relativa del obstáculo es imperativo para que el Agente pueda llegar a aprender a esquivarlo, ya que su posición no es fija entre episodios.

5.4.5.3.2. Ciclo por refuerzo: Decisión

El componente Decision Requester no era necesario tocarlo. El método de toma de decisiones dentro de un entrenamiento por refuerzo es intrínseco a la aplicación de Machine Learning que se use. Es cierto que puede configurarse con un archivo .yaml que se crea al inicio del entrenamiento. En este archivo, se pueden ver y editar los hiperparámetros del entrenamiento. Intentar tocar los hiperparámetros sin un experto en el tema puede ser catastrófico para el resultado del entrenamiento.

5.4.5.3.3. Ciclo por refuerzo: Acción

El siguiente paso es tratar con las acciones del Agente y su Vector de Acciones. La función interna `OnActionReceived()` es llamada cada vez que el Agente realiza una acción, que no es nada más que alterar un número de su vector. Sin embargo, saber qué hacer con esos números es la clave para un entrenamiento exitoso.

Primero se debe llevar un registro de cada número. En este caso, el Vector de Acciones es de longitud 6 porque tiene 6 partes del robot que mover, así que se debe asignar cada posición del vector a una parte del robot. Para ello, se puede usar esta lógica:

```
Float moveShoulder = actions.ContinuousActions[0];
```

Aquí se muestra un snippet de la lógica para actuar sobre la pieza "shoulder", pero se debe realizar esto para cada pieza, obviamente cambiando el número que denota la posición del vector, del 0 al 5. Pero llevar registro del Vector de Acciones no es suficiente, se debe decidir qué hacen los números. Se puede usar cualquier función nativa de Unity que se encargue de movimientos de objetos, como transform, rotate, applyForce, etc. En este caso, interesaba rotate, así que se podía usar este comando:

```
shoulder.Rotate(0, moveShoulder * rotateSpeed * Time.deltaTime, 0, Space.Self);
```

Ya se había hablado de la función `rotate()`, pero es importante aclarar que los ceros de ahí indican los ejes "X" y "Z", ya que, en este caso, shoulder (hombro) sólo debía moverse en el eje "Y". Cabe mencionar que si hubiera habido un error de programación y se hubiera puesto el movimiento en el eje equivocado, gracias a la configuración de anclajes con Configurable Joints, el robot simplemente no se habría movido, en vez de moverse en un ángulo imposible.

5.4.5.3.4. Ciclo por refuerzo: Castigo y Recompensa

Gracias a los pasos anteriores, el Agente podía observar su entorno, tomar decisiones y realizar acciones. Sólo restaba tratar el tema de los castigos y recompensas.

Antes, se ha mencionado que cada objeto en Unity con un Mesh Collider tiene una función nativa llamada `OnTriggerEvent()`, que es llamada cada vez que un objeto con considerado Trigger choca con un objeto con Mesh Collider. Lo interesante de esta función es, que dentro de ella, se puede detectar con qué objeto se ha chocado, y tomar medidas apropiadas.

```
if (other.TryGetComponent<Environment>(out Environment environment))
{
    collisionInEnvironment();
}
```

Figura 36. Snippet control de colisiones

Con la función mostrada en la Figura 36 se puede identificar si el choque ha sido con el entorno (Environment). Es considerado entorno cualquier objeto con el script Environment dentro de él, un script usado para identificaciones, ya que las tags no sólo eran engorrosas cuando se trabaja con muchos objetos, si no que interferiría con el entrenamiento masivo.

Tras detectar que, efectivamente, algo se ha chocado con el Entorno, se puede dar un premio o castigo al Agente. En este caso, ya que cualquier choque que no sea mano con goal debe ser castigado, se programaron dos funciones para que cualquier script pudiese llamar al Agente y castigarlo:

```
35 referencias
public void collisionInEnvironment()
{
    AddReward(-1f);
    floorMeshRenderer.material = loseMaterial;
    EndEpisode();
}

5 referencias
public void collisionWithObstacle()//In case we want to give different punishment for obstacle collision
{
    AddReward(-2f);
    floorMeshRenderer.material = loseMaterial;
    EndEpisode();
}
```

Figura 37. Funciones de castigo

Las funciones mostradas en la Figura 37 podían ser llamadas por cualquier script para castigar al Agente en caso de que ocurriese cualquier colisión indeseada. Las funciones castigan al Agente, colorean el suelo de rojo, para mejor visualización durante el entrenamiento, y acaban el episodio, ya que ha ocurrido un fallo "catastrófico". Una colisión de un robot consigo mismo o el entorno es algo serio, que puede dañar el hardware considerablemente. Normalmente, se castiga también en función del tiempo para promover episodios más cortos pero, siendo que la duración del episodio no era algo preocupante en ese estadio, no se procedía a castigar por tiempo. De momento.

En este caso, hay una goal en una mesa, el objetivo del robot es tocarla con la mano, y simular que la lleva a la otra mesa. Para ello, cuando la goal es tocada por la mano del robot, ésta desaparece y vuelve a aparecer en la mesa final, incitando al robot a volver a tocar la goal con la mano otra vez, sólo que en otra posición. Es posible hacer que la mano literalmente coja el objeto y lo deje en la otra mesa, pero eso complicaría el entrenamiento. Se consideró como una posible mejora en el futuro.

Siendo que ya se habían programado funciones para el castigo, debían programarse también funciones para dar recompensa al Agente. Se diseñaron las funciones de recompensas igual que las de los castigos, con la intención de que pudiesen ser llamadas desde cualquier script al detectar una colisión. Si la colisión detectada era mano con goal, entonces se llamarían a estas funciones en vez de a las de castigo.

```
if (other.TryGetComponent<Goal1>(out Goal1 goal1))
{

    if (goal1.transform.localPosition == goal1Position)
    {
        //goal1.transform.localPosition = goal2Position;
        firstGoalReached = true;
        AddReward(1f);
        //firstGoalReached = true;
        //goal1Object.SetActive(false);
        floorMeshRenderer.material = firstTaskMaterial;
        //EndEpisode();
    }
    if (goal1.transform.localPosition == goal2Position)
    {
        AddReward(10f);
        floorMeshRenderer.material = winMaterial;
        EndEpisode();
    }
}
```

Figura 38. Funciones de recompensa

La función mostrada en la Figura 38 es la que se encarga de las recompensas, y está dentro del Agente que, a su vez, está dentro de la mano del robot. De esta manera, se comprueba si la mano ha chocado con la goal. De ser así, se comprueba si ha sido la primera vez con la flag `firstGoalReached`. Si es así, se cambia la flag a true, se le da una recompensa al Agente, y se colorea el suelo de azul para poder ver a simple vista si se ha cumplido este evento durante el entrenamiento. Si la mano ha tocado la goal y es la segunda vez (doble comprobación con la flag + comprobación de posición de la goal), se le da una recompensa mayor, se colorea el suelo de color verde para visualización y acaba el episodio.

5.4.6. Resumen del Sprint 4

Durante este Sprint dediqué la mayor parte del tiempo a crear y configurar las articulaciones del robot, así como a restringir sus movimientos a unos que pudiese realizar en el mundo real. Más tarde, programé la lógica de colisiones que permitiría que los castigos y recompensas fuesen asignados al Agente en el futuro ciclo de entrenamiento. Aunque empecé a programar dicha lógica de castigos y recompensas no dio tiempo a acabar durante el periodo del Sprint 4.

5.5. Sprint 5

En este Sprint, el desarrollo se centró principalmente en las fases de entrenamiento del modelo. Se empezó con el entrenamiento por refuerzo, que se planeaba fuese el único necesario, pero se siguió con otros tipos de entrenamientos añadidos durante la realización de este proyecto. Estos entrenamientos son las tareas en rojo dentro del Diagrama de Gantt ya que no estaban planeados en un principio pero se vieron necesarios durante el proceso. Dicho esto, la lista de tareas para este Sprint es:

- OB2-T09: Programar lógica del entrenamiento por refuerzo
- OB3-T01: Fase de entrenamiento por refuerzo
- OB3-T02: Fase de entrenamiento por imitación
- OB3-T03: Fase de entrenamiento combinada

5.5.1. OB2-T09. Finalización del ciclo de Castigo y Recompensa

El tiempo de esta tarea se dedicó a acabar la programación de la lógica de castigos y recompensas, acabando con la fase de desarrollo del entrenamiento por refuerzo.

Se puede apreciar que la lógica de recompensas depende de una bandera booleana que registra si la primera tarea (recoger el objeto) ha sido realizada o no, así que es importante recordar resetearla a false cada vez que empezase un nuevo episodio, utilizando la función `OnEpisodeBegin()`.

Las dos posiciones `goal1Position` y `goal2Position` están declaradas en las variables de entorno del Agente, con posiciones fijas. Sin embargo, en estas funciones no se puede apreciar que se haya programado que la goal se mueva tras ser tocada. Eso fue realizado dentro de la función de `CollectObservactions()`.

Una de las muchas tareas de esta función es observar el estado de la flag `firstGoalReached`. Si detecta que se vuelve `true`, le dice que se mueva a la segunda posición, `goal2Position` (véase Figura 39).

```
sensor.AddObservation(goalTransform.localPosition);
if (firstGoalReached)
{
    goalTransform.localPosition = goal2Position;
}
```

Figura 39. Snippet para mover la goal

Antes de empezar a entrenar, es siempre aconsejable testear el entorno manualmente. Para ello debía hacerse un override de la función nativa de MLAgents `Heuristic(ActionBuffers)`. Esta función permite simular un entrenamiento sin que el Agente tome decisiones, las tomaría un operario (en este caso yo mismo) con teclas o botones. Normalmente, la función Heuristic se encarga de alterar los números dentro del Vector de Acciones. Mientras esté en override, yo controlaría estas acciones.

```
0 referencias
public override void Heuristic(in ActionBuffers actionsOut)
{
    ActionSegment<float> continuosActions = actionsOut.ContinuousActions;
    //continuosActions[0] = Input.GetAxisRaw("Horizontal");
    //continuosActions[1] = Input.GetAxisRaw("Vertical");
    if (Input.GetKey("right"))
        continuosActions[0] = 1f;
    if (Input.GetKey("left"))
        continuosActions[0] = -1f;

    if (Input.GetKey("up"))
        continuosActions[1] = 1f;
    if (Input.GetKey("down"))
        continuosActions[1] = -1f;
```

Figura 40. Snippet del override de Heuristic

Esto es sólo un snippet de la función completa (véase Figura 40). Teniendo un Vector de Acciones de tamaño 6, había que hacer esto para las 6 posiciones del vector.

Una vez estuvieron los botones bien mapeados, se podía dar a Play en Unity, para empezar una sesión de testeo. Lo primero que debería verse es que nada "explote". En Unity, con Rigid Body, los objetos "explotan" y salen volando si están chocándose unos con otros al principio de la escena. No fue el caso. Acto seguido, simplemente se debe interactuar con los controles para mover el robot y comprobar la lógica. Es interesante comprobar las limitaciones y lógicas programadas para ver si funcionan bien. Para ello, se puede chocar el robot contra el suelo, contra las mesas, contra sí mismo, etc. Todas las veces debería pintarse el suelo de rojo y reiniciarse el episodio. Si no lo hiciera, algo estaría mal programado.

Tras comprobar las colisiones que generarían un castigo, se procedió a probar con las que darían recompensa. Se hizo que la mano tocase la goal. El suelo debería pintarse de azul, y la goal debería moverse a su segunda posición. Seguidamente, se procedió a tocar la goal en su segunda posición, el suelo debería pintarse verde y el episodio debería acabar, todas las piezas volviendo a su posición original. Si estas cosas no hubieran ocurrido, habría sido necesario repasar la programación del entorno.

Siendo que los castigos y las recompensas parecían funcionar correctamente, era momento de empezar el entrenamiento.

5.5.2. OB3-T01. Fase de Entrenamiento por Refuerzo

En este apartado, hare una corta explicación sobre los pormenores de un entrenamiento por refuerzo, así como de explicar mis experiencias y los resultados finales del entrenamiento por refuerzo.

5.5.2.1. Introducción al Entrenamiento por Refuerzo

Antes de poner al Agente a entrenar, es necesario preparar el entorno para un entrenamiento masivo. Si se le hubiera dado a entrenar en ese momento, se habría podido ver al escenario moviéndose y aprendiendo.

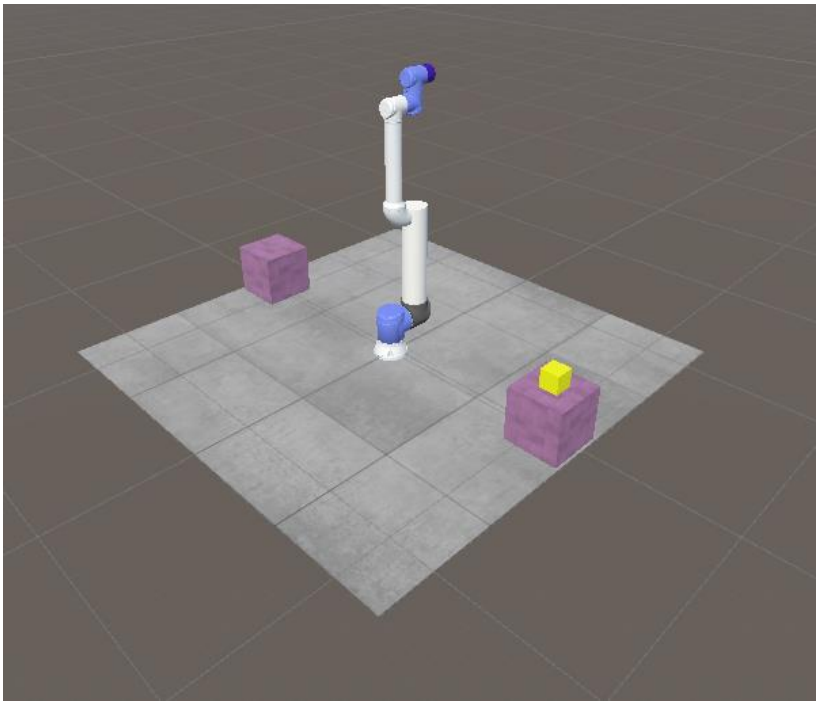

Figura 41. Escena final

Sin embargo, existe una manera de acelerar el entrenamiento para que no costase años de entrenamiento: multiplicar el escenario. En la Figura 41 se puede ver que el entorno cuenta de un solo escenario. Muchas aplicaciones de Machine Learning están preparadas para multiplicar el escenario "n" veces para reducir el tiempo de entrenamiento. No es exactamente lineal (respecto a n) la reducción de tiempo obtenida, pero es considerable. Es más pesado para la máquina que haga las computaciones, pero puede hacer que un entrenamiento que costaría meses, cueste apenas una semana.

Esta es la razón por la que se han utilizado todas las variables de posición y rotación como locales en vez de globales. Así, aunque se multiplique el número de entornos, se utilizarán las coordenadas locales de cada entorno, en vez de las globales, y se evita que puedan interferir entre ellos.

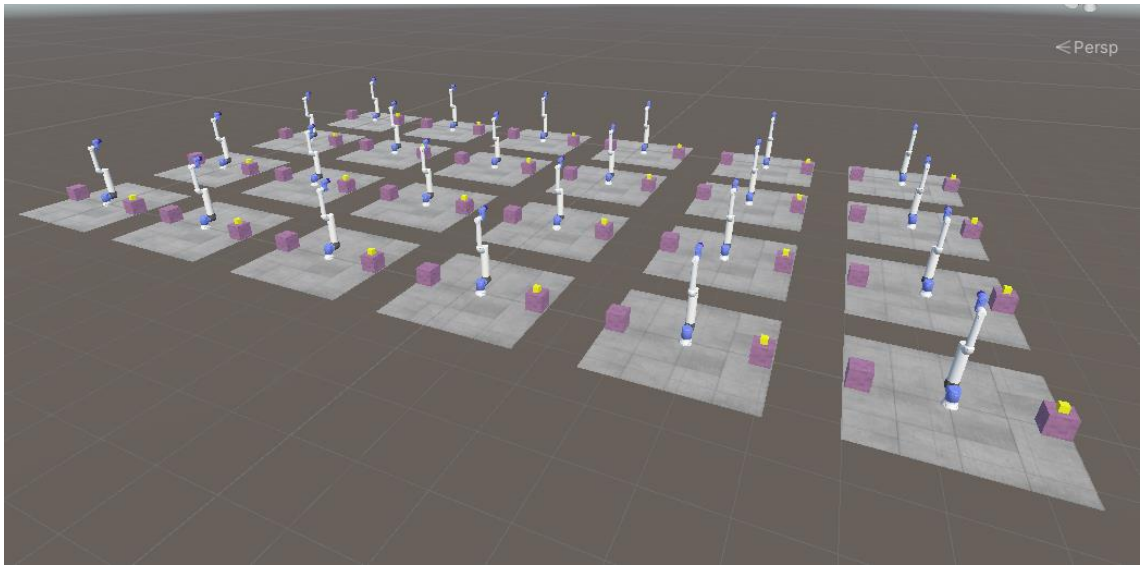

Figura 42. Entorno de entrenamiento masivo

Una vez el entorno de entrenamiento masivo estuvo preparado (véase Figura 42), se pudo empezar a entrenar. Para ello, era necesario arrancar el entorno virtual en Anaconda, con el comando:

```
conda activate tfgv1
```

Siendo `tfgv1` el nombre que se le había dado al entorno elegido para el proyecto.

Una vez estuvo el entorno arrancado, sólo se debía navegar al directorio donde se quería que se guardase cualquier cosa relacionada con el proyecto. Se navega igual que en CMD, con el comando `cd`. Una vez en el directorio, se le debe indicar que quería empezar a entrenar. Para ello, se utilizan los comandos de la librería MLAgents en Anaconda.

```
mlagents-learn --run-id=NombreDeLaID
```

`NombreDeLaID`, como su nombre indica, es el nombre que se le da a la sesión de entrenamiento. Dentro del directorio que se haya indicado al arrancar Anaconda se creará una carpeta llamada "results", donde se guardarán los resultados de los entrenamientos archivados por ID. En mi caso, el nombre de mi primera ID de entrenamiento fue "TestV1". Es importante recordar que si se cambia la ID, el Agente entrenará desde cero, ya que creerá que es un entrenamiento nuevo. Para seguir donde se haya dejado en una sesión de entrenamiento anterior, hay que añadir el parámetro `--resume` al final.

```
mlagents-learn --run-id=NombreDeLaID --resume
```

Si se ha instalado todo bien y, no hay problemas de consola en Unity (como bugs, o problemas de parser), en Anaconda saldrá el logo de Unity dibujado en ASCII y pedirá que se clique el botón de Play (dentro de Unity) para iniciar entrenamiento. Tras darle al Play, se podía observar al robot moverse como un gusano sin cabeza, pero eso era normal. Lo más probable era que se chocase contra algo en los primeros 2 segundos, pero era lo esperado. Cada episodio haría que el Agente aprendiese más y más sobre su entorno y el efecto de sus acciones.

Lamentablemente, cuando ya se llevaba un rato entrenando, el proyecto se paró y dio por finalizado el entrenamiento, aunque se podía ver claramente que el Agente no había aprendido correctamente. Esto se debía a un archivo de configuración con hiperparámetros [57] [58] que controlan diversos aspectos del entrenamiento. Si no se le pasa uno de estos archivos por comandos al empezar el entrenamiento, utiliza uno por defecto y lo mostrará por la pantalla de Anaconda.

Se procedió a alterar el archivo yaml de ese entrenamiento. Se buscó el campo "max step" y se pudo ver que, efectivamente, el máximo número de steps permitidos en el entrenamiento, de principio a fin, era 50 mil sumando los steps de todos los entornos multiplicados. Eso no era ni de lejos suficiente para un problema de esa magnitud. Por el momento, la última versión de MLAgents permite un máximo número de steps de 50 millones, así que se cambió el "50000" de "max steps" por 5.0e7, que equivale a 50 millones.

Para utilizar el archivo config.yaml (o como haya sido llamado, en mi caso AvoidObstacles.yaml) se debe añadir al comando de entrenamiento, como parámetro, la ruta del archivo yaml.

```
mlagents-learn config/config.yaml --run-id=NombreDeLaID –resume
```

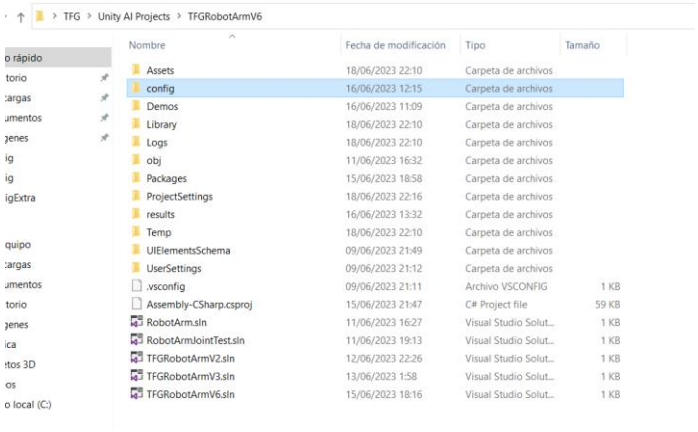

Figura 43. Carpeta config

Siendo que puede ocurrir que se quieran probar diferentes tipos de parámetros para los entrenamientos, es recomendable crear una carpeta "config" dentro de la carpeta del proyecto de Unity y guardar los yaml ahí (véase Figura 43).

Tras este cambio, el Agente podrá entrenar por un máximo de 50 millones de steps. No se puede medir la duración de un entrenamiento en tiempo a priori porque depende del ordenador, del número de entornos, y de muchos otros parámetros, por eso es más correcto medir las duraciones en steps. A partir de ahí no quedaba más que esperar a que el Agente se comportase de una manera que se considerara adecuada y parar el entrenamiento.

Para ver cómo evoluciona el Agente se puede hacer uso de TensorBoard, un programa que crea unas gráficas para, de un vistazo, ver si progresa, se ha estancado, o si ha aprendido satisfactoriamente. Para acceder a TensorBoard es necesario abrir otra instancia nueva de Anaconda, se vuelve a arrancar el mismo entorno en el que se esté entrenando, y se introduce el siguiente comando:

```
tensorboard --logdir results
```

Anaconda dirá que los resultados están siendo ploteados en localhost:6006. Lo único que se necesita hacer es abrir un explorador de internet y escribir "localhost:6006" en la barra de direcciones. Podrán verse una variedad de gráficas. Las más interesantes son las de "recompensa acumulada" y las de "duración del episodio", ambas con "número de steps" como eje X.

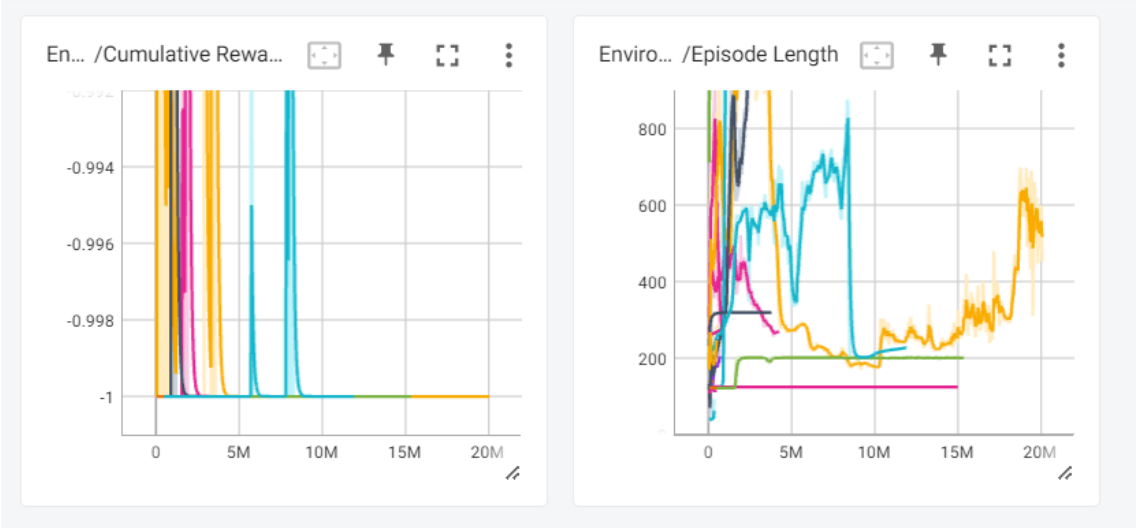

Figura 44. Gráficas de Tensorboard. Cumulative Reward / Episode Length

La Figura 44 representa un gran conjunto de entrenamientos ploteados todos en la misma gráfica para realizar comparativas. Sin embargo, es posible y recomendable ocultar los entrenamientos que no estén en marcha para poder ver la evolución del Agente.

Lo normal es que, al principio, la recompensa acumulada esté en negativo, debido a los castigos que recibirá al principio y, aunque tenga algún momento de recompensa positiva, probablemente sea casualidad. A lo largo de millones de steps, la curva debería coger una forma más uniforme y acercarse cada vez más a la recompensa esperada por una Política completa.

En cuanto a la gráfica de la duración del episodio, lo normal es que al principio sean muy cortos, ya que estará chocándose constantemente hasta aprender los límites de sus movimientos y entorno. Según fuese aprendiendo, aumentaría la duración ya que no se chocaría tan rápido. Sin embargo, cuando al final aprenda a realizar la Política, lo ideal sería que la duración del episodio fuese reduciéndose hasta estabilizarse. Llegados a ese punto, significaría que el Agente ya no consigue optimizar más la Política y que, por tanto el entrenamiento habría concluido satisfactoriamente.

Cabe destacar que un entrenamiento por refuerzo tiene dos formas de plantear el entrenamiento a bajo nivel: exploración y explotación. Estas metodologías se usan conjuntamente, pero se puede asignar un peso a cada una en los hiperparámetros del archivo yaml.

La metodología de exploración prioriza utilizar los steps para explorar su entorno hasta el último recoveco, mientras que la explotación prioriza los steps en intentar encontrar recompensa y recibir menos castigo. Si se le diera un peso de 100% a exploración y otro de 0% a explotación, el Agente sólo aprendería a explorar sin importarle la Política, la cual nunca realizaría. Si se le dieran unos pesos inversos, el Agente sólo aprendería a optimizar la recompensa acumulada tras cada episodio con lo que, si se chocase constantemente (cosa que haría ya que no habría aprendido a moverse), podría aprender a quedarse quieto para no recibir castigos, ya que llegar a la goal y recibir recompensa sería muy "arriesgado". Lo ideal es mantener una exploración relativamente alta al principio y, cuando consideremos que el Agente ha explorado y entendido su entorno lo suficiente, subir la explotación para que se centre en cumplir con la Política.

Una vez acabado el entrenamiento, se puede ir a la carpeta del proyecto de Unity>results> NombreDeLaID. Dentro habrá un archivo con una extensión "onnx" (Open Neural Network Exchange). El nombre del archivo dependerá del nombre que se le haya dado a la Política del Agente en el componente Behavioral Parameters>Behavior Name.

Este archivo es el **cerebro entrenado**. En ese estado, podría usarse en cualquier entorno de Unity+ROS metiéndolo en un objeto con un Agente de MLAgents, en el componente Behavioral Parameters>Model. Habría que cambiar Behavioral Parameters>Behavior Type a "Inference Only" para que el Agente realizase la Política sin necesidad de Anaconda ni más entrenamiento.

5.5.2.2. Resultados del Entrenamiento por Refuerzo

Tras empezar el primer entrenamiento real y, dejarlo 3 a 4 horas, no se apreció mucha mejora en el comportamiento del Agente. Al principio, debido a la falta de experiencia, pensaba que 3 o 4 horas era suficiente para este tipo de entrenamiento. Por supuesto, como se ha mencionado antes, hablar de tiempo en horas cuando se refiere a un entrenamiento es poco apropiado. Se debe emplear la medida de "steps". Es importante mencionar que, debido a la complejidad geométrica del entorno, el ordenador no podía soportar más de 24 escenarios a la vez.

Este primer entrenamiento se dejó entrenando unos 2 millones de steps, y no se vió mucha mejora, así que se decidió quitar el obstáculo temporalmente para ver si era capaz de aprender su entorno. Se reinició el entrenamiento y se dejó durante 5 millones de steps y, aunque parecía haber cierta mejora respecto al de 2 millones, no parecía estar cerca de ser entrenado. Tras repasar la programación del código, se encontraron un par de fallos (un par de funciones mal llamadas alteraban el sistema de castigos). Tras arreglarlos se inició otra sesión de entrenamiento.

Aquí es cuando se descubrió el mayor problema, al cual a día de hoy todavía no he conseguido dar solución con mis recursos: el programa crashea cuando lleva un rato entrenando. Lamentablemente, Unity no parece dejar un log de crashes. Simplemente se puede saber que ha crasheado porque ya no está encendido y Anaconda dice que ha perdido la conexión con Unity. Puede haber muchas razones para el crash, desde temperaturas altas de la CPU/GPU, bugs en Unity (el programa, no mi código) hasta problemas internos de Windows. Lamentablemente no se ha podido localizar el problema concreto.

Simplemente crashear no sería un problema en sí ya que, según el config.yaml retocado que se usaba en los entrenamientos, Anaconda hacía una copia del cerebro y los resultados del

entrenamiento automáticamente cada 500 mil steps. En teoría, simplemente resumiendo el entrenamiento, debería continuar donde lo ha dejado, tal y como indica la documentación de Unity. El verdadero problema es que, cuando crashea, el entrenamiento se **reinicia**.

Se procedió a volver a entrenar desde cero. El programa crasheó a los 11 millones de steps. Tras resumir el entrenamiento hasta los 20 millones sin ver mejora, se barajó la posibilidad de que los crasheos estuvieran borrando el entrenamiento anterior.

Para intentar solventar estos problemas, se decidió cambiar el enfoque y centrar los esfuerzos en reducir el tiempo de entrenamiento todo lo posible. Con suerte, el Agente sería entrenado antes de un crasheo. Se decidió cambiar a un entrenamiento de doble Agente lo cual, aunque complicaría el entrenamiento, aceleraría el proceso. La idea era que un Agente manejase el hombro, el brazo y el antebrazo (nótese que no se ha nombrado el codo), que son las partes que afectan a la mayor parte del movimiento de coordenadas finales de la mano del robot. Su Política cambiaría a intentar acercar la mano del robot hasta la goal. Recibiría castigos igual que antes, por colisiones, y recibiría recompensas igual que antes, si el Agente de la mano toca el goal.

Otro Agente manejaría las dos muñecas, que controlan la orientación de la mano mucho más que lo que pueden hacer los brazos. Su Política sería la de mantener la mano siempre orientada al goal. Para ello, se programó una función que realizaba un cálculo vectorial entre las coordenadas de la mano y la goal y lo compararía con el vector normal de la mano (normal a un plano imaginario que la mano estaría sujetando). Si ambos vectores resultaban paralelos, el Agente recibía una recompensa por cada fracción de segundo que mantuviese el vector paralelo. Si no, recibiría un castigo por cada fracción de segundo que no lo mantuviese paralelo, siendo la recompensa 10 veces mayor que el castigo, para propiciar un comportamiento que acumulase recompensa.

La idea era buena sobre el papel, e incluso se consiguió hacer que trabajasen juntos, pero por mucho que se investigó, no se pudo encontrar documentación que explicase cómo crear un archivo yaml para dos Agentes diferentes, ya que no es una práctica común fuera de entornos de I+D. La idea era que si el Agente de la mano conseguía mantener la mano apuntando al objetivo, sería mucho más probable que ésta colisionase con la goal cuando el brazo lo acercase, en vez de chocar con una de las muñecas.

Se decidió volver otra vez al modelo anterior, sin obstáculo y con un solo Agente. Se consideró reducir el número de articulaciones, ya que la complejidad del problema parecía muy alta para manejar con un ordenador, pero se descartó por el momento.

Llegados a este punto se decidió reducir el tamaño del Vector de Observaciones para aliviar al Agente. Se cambió el enfoque de dos goals a una que se moviera de mesa en mesa según es tocada. Esto, y otros pequeños ajustes, bajó el tamaño del vector en 13, lo cual es considerable.

Estos cambios parecieron dar resultados, tras casi 6 millones de steps, el Agente encontró la goal de casualidad (cosa normal en entrenamiento por refuerzo) pero en sólo un par de millones más de steps, parecía que ya estaba entendiendo dónde estaba la goal.

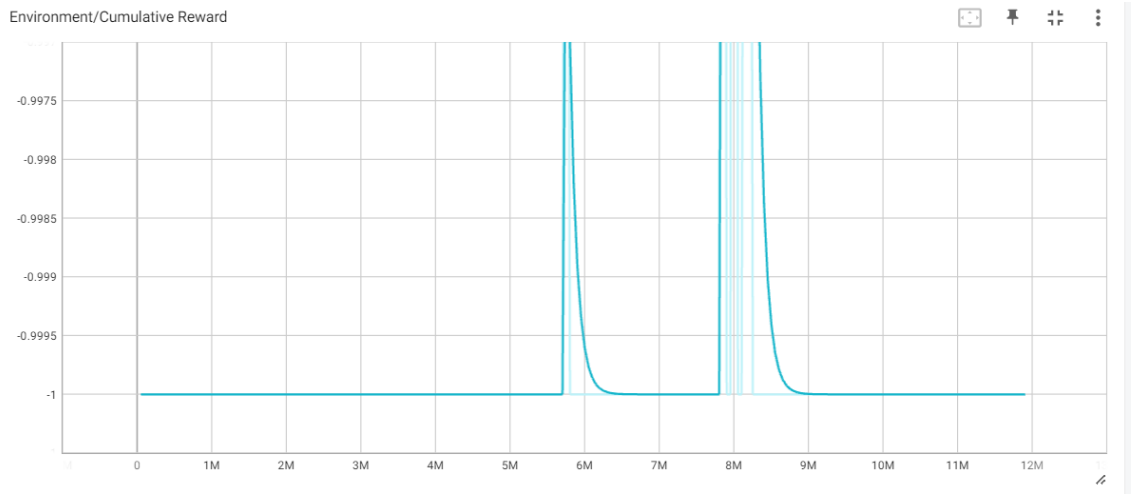

Figura 45. Accumulative Reward del entrenamiento

Se aprecia en la gráfica de la Figura 45 que a los 5.8 millones de steps descubrió la goal por casualidad en un proceso de exploración. Dedicó un par de millones de steps más en explorar su entorno antes de atreverse con la explotación, cosa que intenta sobre los 7.7 millones de steps. Ahí se puede ver cómo, durante casi un millón de steps, ha estado tocando la goal correctamente, es decir, había completado su primera tarea. Sabemos que no completaba la Política porque la suma de recompensa acumulada sigue siendo negativa, pero siendo que -1 es el suelo lógico de la recompensa acumulada (así estaba programado), estar en medias de -0.5 en algunos checkpoints (cada 500 mil pasos) era un gran avance.

Los problemas vinieron con uno de los infames crasheos, que ocurrió más o menos sobre los 9 millones de steps. Fue con este crash que se descubrió que, efectivamente, el entrenamiento se perdía al crashear. Al resumir el entrenamiento, no volvió a encontrar la goal en los próximos 3 millones de steps antes del próximo crash. Visualmente, el robot volvía a no saber ni lo que era el suelo, volvía a chocarse contra él en cuestión de dos segundos y se movía exactamente igual que cuando empezaba un entrenamiento de nuevo.

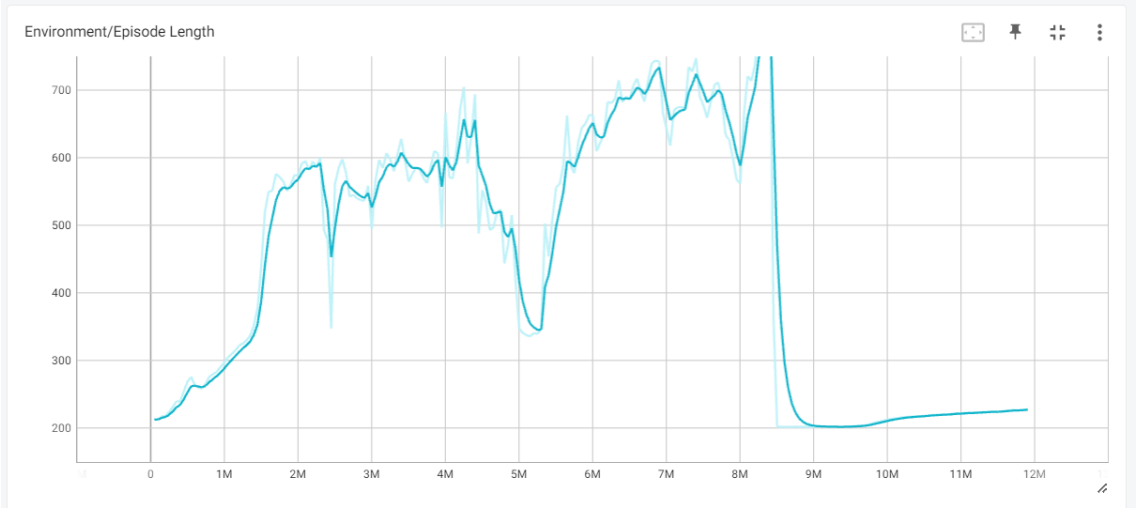

Figura 46. Episode Length del entrenamiento

La Figura 46 muestra el progreso del entrenamiento en base a la longitud de los episodios con respecto al número de steps empleados en entrenamiento.

Como es normal, al principio eran pocas centésimas de segundo. Entre los steps 6 y 8 millones, parecía haber progresado mucho en el entrenamiento. Los episodios empezaban a durar sus buenos 7-10 segundos, durante los cuales el robot exploraba el entorno en busca de la goal.

Sin embargo, se puede apreciar el crash visualmente en la gráfica. Poco antes de los 9 millones de steps el programa crasheó. Al resumir el entrenamiento, se pudo ver claramente, gracias a ambas gráficas, cómo el entrenamiento había sido reiniciado completamente. Por si acaso, se comprobaron las gráficas de varios entrenamientos donde había crasheado y, efectivamente, todas parecían seguir el mismo patrón.

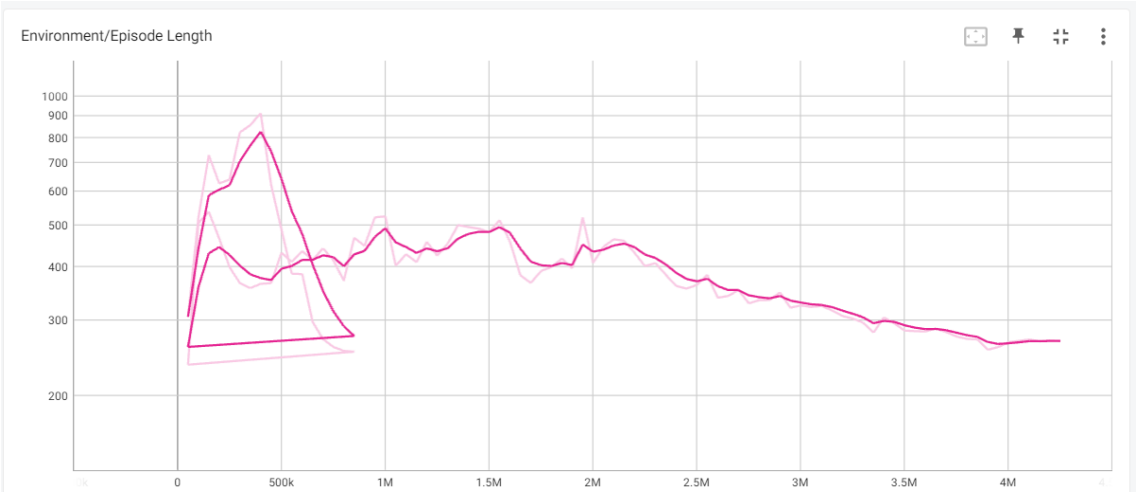

Figura 47. Gráfica ilógica

La Figura 47 muestra otro entrenamiento prometedor, antes de saber del problema del reseteo. El programa crasheó antes del medio millón de steps y consiguió resetear incluso el número de steps, dando lugar a esta gráfica bizarra con un looping, desafiando toda lógica matemática.

También es cierto que no el 100% de las veces, un crash reinicia el entrenamiento, pero siendo que es un modelo que requeriría más de 40 millones de steps, no podía ser entrenado en esas condiciones. El máximo número de steps que se habían conseguido hacer de un tirón son 11 millones.

Se investigó este problema para ver si el asunto del reseteo del entrenamiento era algo conocido. Resulta que, en ordenadores donde no se puede realizar el entrenamiento de una tirada, no es extraño que ocurra este fenómeno [59] [60] [61]. Parece ser que el equipo de MLAgents era conocedor de este problema y, según sus posts en Github, debía haber sido arreglado a finales del 2020. Parece que no consiguieron arreglarlos en todas las circunstancias posibles.

Sabiendo esto, se tomó la decisión de que se tenía que reducir, y mucho, el número de steps necesarios para completar el entrenamiento. Para conseguir esto, se optó por incluir entrenamiento por imitación, ya que esto reduce drásticamente la necesidad de entrenamiento.

5.5.3. OB3-T02. Fase de Entrenamiento por Imitación

El otro tipo de entrenamiento que soporta Unity es el entrenamiento por imitación. En este tipo de entrenamiento, un operario graba una serie de demostraciones de la Política siendo satisfecha mediante el uso del teclado para mover las partes del robot. Como grabarle un tutorial al Agente.

Esto quería ser evitado ya que, debido a la complejidad de control de un brazo con 6 articulaciones usando un teclado, iba a ser complicado grabar demostraciones válidas.

Para incluir entrenamiento por imitación, debía incluir un componente Demonstration Recorder en el mismo objeto en el que estaba el Agente. Para grabar las demostraciones solo era necesario clicar en "Record" e indicarle el nombre que se quería dar al archivo de demostración y el nombre de la carpeta donde se querían guardar.

Para grabar las demostraciones, se manejó el robot con los controles designados en `Heuristic()` que habían sido mencionados con anterioridad. El objetivo era grabar una ristra de episodios, uno detrás de otro sin descanso, para que el Agente entendiese qué es lo que había que hacer. Al hacerlo a mano, se descubrió que el modelo utilizado para el robot UR10e no era correcto desde el principio: le faltaba una articulación. Debido a la falta de experiencia con este robot en concreto, no se pudo averiguar hasta entonces.

Esta fue la razón por la cual, en el Diagrama de Gantt, se aprecian 4 días usados en tareas de Sprints pasados, ya que se tuvo que remodelar el brazo.

En un primer acercamiento, se planteó la idea de "serrar" el brazo para añadirle la articulación, pero se carecía de experiencia con modelado 3D avanzado. Tras indagar en la red, se encontró un programa llamado ProBuilder, que permitía fusionar dos objetos en uno sólo, lo cual era exactamente lo que se necesitaba para reducir el número de objetos en el escenario.

Tras seguir las instrucciones de instalación y empezar la fusión del objeto, se investigó más sobre dicho programa, aprovechando el tiempo de procesamiento de la pieza. Al parecer, aunque se fusionasen la piezas, el resultado bajaría más el rendimiento que las piezas por separado. Los usuarios de ProBuilder indicaban que esto era una alternativa pobre a retocar los modelos directamente en Blender o SolidWorks, que esta era una herramienta para los que no tenían formación en herramientas de modelado [62]. Sin embargo, aunque puede ser útil para otras aplicaciones, frustra el objetivo de la fusión.

Tras buscar en otros repositorios, se encontraron otros modelos con anatomías perfectas. Sin embargo, estos modelos eran demasiado sofisticados, hasta el punto de que aumentaban en más de 20 el número de piezas en el entorno. Se necesitaba un modelo con el mínimo número de piezas 3D, uno como el que se había usado desde el principio pero que tuviese la articulación que faltaba. Se llegó a la conclusión de que la mejor solución era mezclar dos modelos. Se mantuvo el primero casi intacto menos el brazo, el cual se sustituyó con una combinación del hombro del segundo modelo [63] más una figura primitiva (un cilindro) simulando el brazo. El resultado se puede ver en la Figura 48.

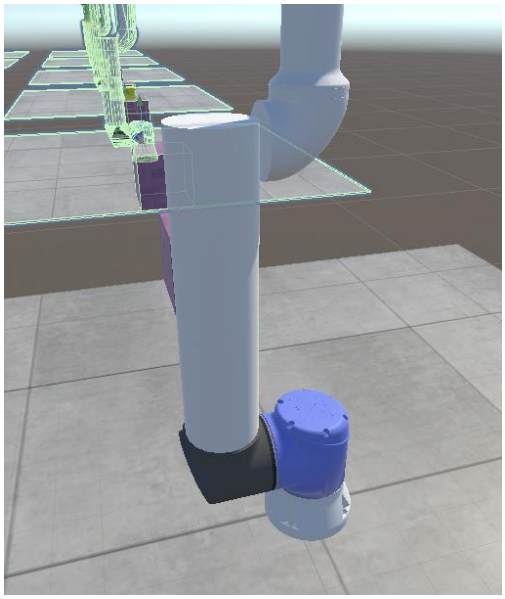

Figura 48. Nuevo modelo del Brazo Robótico

La parte negra del brazo es el codo del segundo modelo, el cilindro del centro de la imagen es la figura primitiva usada para simular el brazo real (el mesh es prácticamente igual que el anterior, aunque con menos detalle). El resto es del primer modelo.

Aunque no especialmente estético, el modelo al fin era anatómicamente correcto. Tuvo que rehacerse la jerarquía de objetos y articulaciones, pero el resultado fue impecable en cuanto a funcionalidad.

Ahora que se contaba con un modelo apropiado, se pasó a grabar las demostraciones para el entrenamiento por imitación. Se grabaron 6 demostraciones. Siendo que sólo se necesitaba una, se eligió la que tenía mejor ratio de intentos satisfactorios. Al grabar una demostración no se puede parar de realizar episodios, con lo que es normal que, a veces, se graben episodios fallidos en la demostración. La mejor demostración que se grabó fue una que contenía 32 episodios, con 30 éxitos y 2 fallos. Tras documentarse en internet, las demostraciones suelen contener de 10 a 20, así que esta demostración se consideró como apropiada. [64]

5.5.4. Resumen del Sprint 5

Durante este Sprint me centré principalmente en el entrenamiento del Agente. Tras encontrar muchos muros que superar, cambié de perspectiva varias veces; desde replantearme el tipo de Machine Learning hasta otros tipos de entrenamiento totalmente diferentes. El resultado un conjunto de entrenamientos, problemas y lecciones aprendidas que pueden servir como base para intentar mejorar futuros entrenamientos. No obstante, este proyecto no acaba aquí, sino que aún se plantearon soluciones alternativas con los otros tipos de entrenamiento antes de sacar conclusiones.

5.6. Sprint 6

Cuando se planeó el Product Backlog, no se pudo prever la complejidad del entrenamiento, ni los problemas asociados al entrenamiento. Esto llevó a que al final del Sprint 5 se planteasen nuevas tareas relacionadas con otros tipos de entrenamientos posiblemente más rápidos. En un principio, este Sprint iba a ser dedicado únicamente a la tarea OB4-T01 y completar los últimos detalles de la memoria del proyecto. Sin embargo, debido a cambios en las tareas propuestas en el Product Backlog, se le dedicó tiempo también a una tarea nueva, la tarea OB4-T00. Antes de todo esto, obviamente, debían ser acabadas las tareas OB3-T02 y OB3-T03 que se quedaron sin acabar en el anterior Sprint. A continuación se detalla la lista de tareas planeadas para este Sprint:

- OB3-T02: Fase de entrenamiento por imitación
- OB3-T03: Fase de entrenamiento combinada
- OB4-T00: Entrenamiento de un robot de 3 articulaciones
- OB4-T01: Analizar resultados y sacar conclusiones

5.6.1. OB3-T02. Fase de Entrenamiento por Imitación

En el anterior Sprints se consiguió grabar una demostración apropiada para iniciar un entrenamiento por imitación. Sin embargo, todavía había de ser implementarla en el entrenamiento. Para ello, debían retocarse unos hiperparámetros en el archivo config.yaml. Era necesario añadir varios campos, incluidas las rutas absolutas hacia el archivo de demostración.

```
reward_signals:
  extrinsic:
    gamma: 0.99
    strength: 1.0
  gail:
    strength: 0.5
    demo_path: C:\Users\icena\Desktop\TFG\Unity AI Projects\TFGRobotArmV6\Demos\Demo1.demo
behavioral_cloning:
    strength: 0.5
    demo_path: C:\Users\icena\Desktop\TFG\Unity AI Projects\TFGRobotArmV6\Demos\Demo1.demo
```

Figura 49. Cambios en el yaml para acomodar el Entrenamiento por Imitación

Dentro del entrenamiento por imitación, hay dos tipos: GAIL y Behavioral Cloning. Ambos declarados en el config.yaml como indica la Figura 49.

GAIL (Generative Adversarial Imitation Learning) es un tipo de imitación que intenta engañar a un discriminador para que se crea que las acciones del Agente vienen de la demo. Lo que consigue con esto es imitar a la demo pero intentando entre medias meter sus propias acciones que considera parecidas. Esto acelera el entrenamiento pero GAIL tiende a desviarse demasiado pasado un tiempo.

Behavioral Cloning, sin embargo, intenta seguir al pie de la letra la demostración. De esta forma, con Behavioral Cloning, el Agente aprendere al pie de la letra lo que hay que hacer, pero nunca mejorará más allá de lo que el operario haya conseguido al grabar la demo. Esto estaría bien en algunos proyectos en los que las trayectorias no cambiasen nunca y pudieras grabar una demostración en vez de programar una a una cada trayectoria.

Sin embargo, este proyecto debía poder pensar por sí mismo. Si se usase sólo uno de los dos métodos, no se llegaría a un estado de entrenamiento más rápido. La solución era usar ambas formas de imitación a la vez [65], dándoles un peso del 50% a cada una.

5.6.2. OB3-T03. Fase de Entrenamiento Combinado

El entrenamiento por imitación rara vez se usa por sí solo, así que había que combinarlo con entrenamiento por refuerzo. Combinando GAIL, Behavioral Cloning y entrenamiento por refuerzo, conseguimos un Agente con una capacidad de aprendizaje superhumana [65]. Inmediatamente se comenzó un entrenamiento desde cero y, se veía claramente que el Agente estaba aprendiendo a un ritmo impresionante.

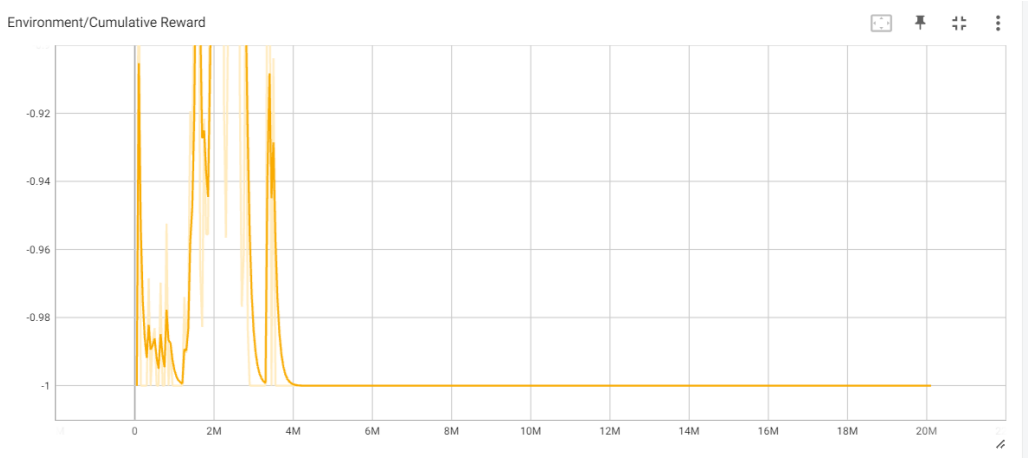

Figura 50. Cumulative Reward entrenamiento combinado

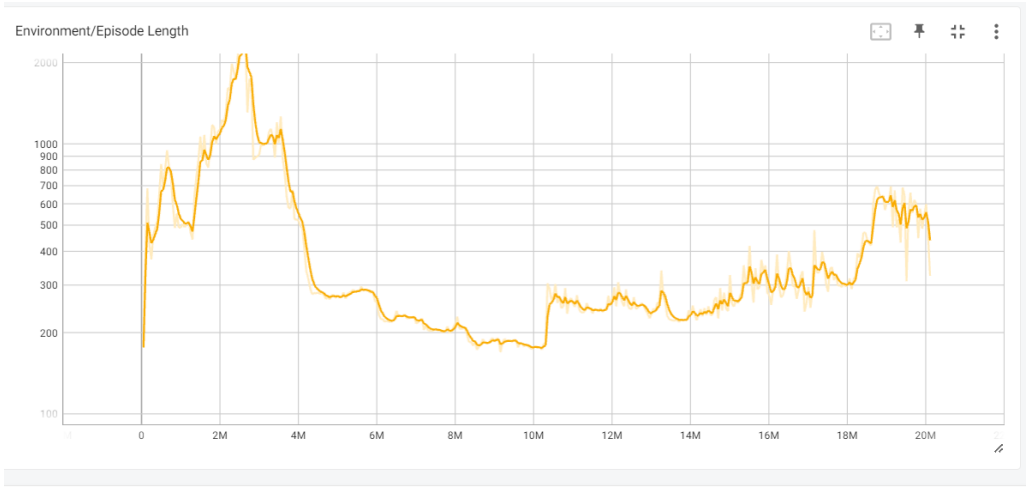

Figura 51. Episode Length entrenamiento combinado

Durante los 4 primeros millones de steps, antes de otro desafortunado crash, el Agente consiguió llegar a la goal un número record de veces. Lamentablemente, se puede ver en la Figura 50 y la Figura 51 el descenso en productividad tras el crash. Hasta el punto de que ya no volvió a encontrar la goal y se podía apreciar visualmente cómo se había reiniciado el entrenamiento. Se intentó replicar el éxito de este entrenamiento sin éxito numerosas veces,

hasta que se llegó a la conclusión de que no se disponían de los recursos necesarios para completar el entrenamiento.

5.6.3. OB4-T00. Entrenamiento de un robot de 3 articulaciones

Teniendo en cuenta los resultados de los anteriores entrenamientos, se planteó la posibilidad de entrenar un robot de 3 articulaciones para poder continuar con el estudio en el tiempo que quedaba. El mayor problema de tiempo con este estudio es la forma en la que el vector de acciones y observaciones de este proyecto interactúan entre sí.

Imaginemos que tenemos otro entorno de Machine Learning con un vector de observaciones y de acciones del mismo tamaño que el del robot de este proyecto. Ahora, imaginemos que estas acciones y observaciones sólo se afectan una a una. Es decir, la posición 1 del vector de acciones sólo afecta a la posición 1 del vector de observaciones (podría ser el 1 con el 3, o el 4 con el 6, el caso es que fuese uno a uno). Bien, pues este proyecto tardaría en entrenarse X número de steps. Siendo X el número de steps que tardaría el Agente en inferir de qué manera sus acciones afectan a sus observaciones. Probará con todas las posibles combinaciones de la posición 1 del vector de acciones para ver cómo afectan a la posición 1 del vector de observaciones. Y así con todas las posiciones de los vectores.

Ahora, imaginemos que duplicamos la cantidad de acciones y observaciones en ese entorno, pero manteniendo el hecho de que se afectan una a una (como un Pong de 4 jugadores, las acciones de cada uno no afectan a los demás). El tiempo de entrenamiento no sería exactamente 2X debido a que los procesadores hoy en día tienen más de un core, pero sería entre 1 y 2 veces más largo. Sin embargo, si triplicamos o cuadruplicamos el tamaño de los vectores, podríamos ver que el tiempo de entrenamiento sigue un crecimiento aritmético.

El problema en este proyecto es que cada una de las posibles acciones del vector de acciones afecta a todas las observaciones del vector de observaciones. El Agente tiene que aprender a inferir de qué manera las acciones de la primera posición del vector de acciones afectan a todas las observaciones. Y no sólo eso, si no que en cuanto la segunda posición del vector de acciones realice otra acción, alterará también todas las observaciones. Esto se debe a que todas las acciones y observaciones están conectadas en un brazo por sus articulaciones. De esta manera, cuantas más articulaciones añadamos a un brazo robótico, mayor es el número de posibilidades, pero con un crecimiento exponencial. Imaginemos, por un momento, que cada posición del vector de acciones tiene 100 posibilidades (no es así, tiene casi infinitas, pero es para simplificar). Si hay dos posiciones en el vector, ambas afectando a la misma observación (en el caso real afecta a todas, lo cual es aún peor en cuestión de rendimiento), serían 100^2 posibilidades. Si añadimos un tercero, 100^3, etc. En el proyecto real, cada acción afecta a entre 22-42 acciones, lo que agrava el problema.

Para poder finalizar el estudio, se decidió probar con un brazo de 3 articulaciones. El modelado y la configuración de este robot son la razón de las desviaciones en el diagrama de Gantt de los días 25 y 28 de Agosto. Se contaba con más experiencia, con lo que se hizo más rápidamente.

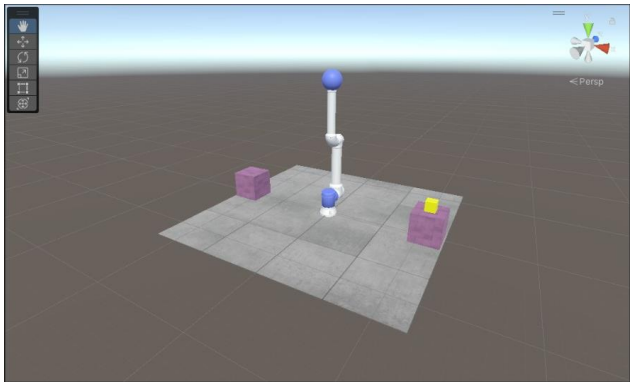

Figura 52. Imagen del RSv1

Este robot apenas contaba con una base giratoria, un hombro y un codo. Tres articulaciones. La mano inmóvil sería la esfera primitiva azul en la cima, como se puede ver en la Figura 52. Para futuras referencias, este robot fue bautizado como "RSv1".

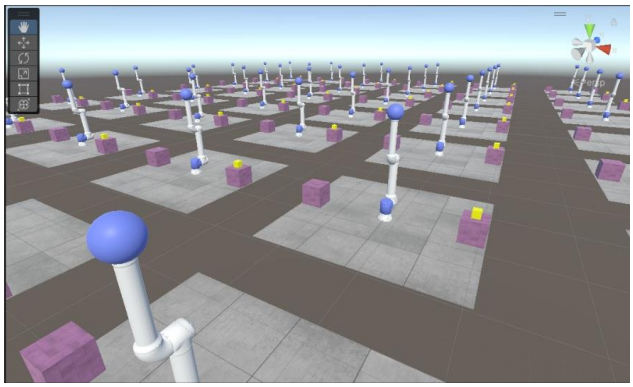

Figura 53. Entorno Masivo del RSv1

La idea era realizar el estudio con un robot más simple, para dar tiempo al Agente a entrenar y, extrapolar los resultados conseguidos con el de 3 articulaciones a un brazo robótico de 6 articulaciones. También aceleró el entrenamiento el hecho de que la simplicidad del escenario permitiese crear un entorno de entrenamiento masivo de 64 escenarios (véase Figura 53) en vez de los 24 que podía hacerse con el UR10e (véase Figura 42). Esto se debía a la simplicidad del modelo.

He continuado entrenando este robot simple hasta prácticamente el último día y, puedo decir con seguridad que ha aprendido más rápido que el UR10e. Sin embargo, aún con 3 articulaciones, mi ordenador no tiene suficiente poder de procesamiento para entrenarlo a tiempo ya que, cada hora y media, debo parar el entrenamiento para enfriar el procesador.

Es importante mencionar que, con el modelo de 3 articulaciones, el programa no ha llegado a crashear ni una vez. Puede que sea un problema de procesamiento o de memoria ya que, este entorno más simple ha aguantado sin crashear. Merece la pena mirarlo en un estudio posterior.

5.6.4. OB4-T01. Analizar resultados y sacar conclusiones

Esta tarea fue dedicada a analizar los resultados y sacar conclusiones definitivas sobre el proyecto. Se tenía que llegar a una de dos conclusiones: es o no es posible entrenar un brazo robótico para las tareas propuestas.

Los resultados de los entrenamientos con el robot de tres articulaciones, hicieron que se sacara la conclusión de que la teoría sobre el crecimiento exponencial del tiempo de entrenamiento es cierta. En menos tiempo, había aprendido más sobre su entorno, con la misma lógica de colisiones, la misma configuración de articulaciones y con el mismo sistema de recompensa y castigo. Es prometedor para una conclusión favorable.

En la sección 9.3 Tablas de entrenamiento se pueden ver las tablas sobre los entrenamientos usados en este proyecto. Sin embargo, tras realizar tantos entrenamientos, es muy posible que los resultados se vean mejor visualmente con el robot intentando explorar su entorno que con los números de una gráfica. El robot RSv1 aprendió extremadamente rápido a no chocarse contra el suelo y a esquivar gran parte de su propia superficie con sus articulaciones. Aunque, el RSv1 aún necesitaría más tiempo de entrenamiento, es prometedor para un estudio inicial.

El resto de conclusiones sacadas durante este periodo serán exhibidas en la sección 8. Conclusiones de esta memoria.

5.6.5. Resumen del Sprint 6

Durante este Sprint completé una tanda de entrenamientos con el UR10e hasta que decidí que no iba a conseguir entrenarlo por completo a tiempo debido a su complejidad. Opté por modelar un segundo robot más simple, el RSv1, para poder entrenarlo a tiempo y extrapolar los resultados. Sin embargo, aunque la idea parecía estar dando resultado, ya que el RSv1 estaba aprendiendo más rápido, tampoco ha sido tiempo suficiente para entrenarlo completamente. Gracias a estos resultados, pude sacar las conclusiones finales sobre este proyecto.

6. Estudio Económico

En esta sección, se realiza el estudio económico del proyecto. En concreto, este estudio económico abarca tanto los costes del trabajo realizado en este proyecto, como los costes que tendrían que tenerse en cuenta en caso de continuar este trabajo en un futuro.

6.1. Estudio sobre el trabajo realizado

Para realizar este proyecto en una empresa, se necesitaría un equipo. Como cabría esperar por la índole de este proyecto, yo he sido el único miembro de dicho equipo. He realizado el trabajo de un diseñador gráfico, un experto en Inteligencia Artificial, un desarrollador de C# y el trabajo equivalente a un manager para coordinar al equipo. A parte, también he utilizado unas licencias de software y un ordenador. Hay que tener en cuenta que, durante el proyecto, muchas de estas horas se han solapado debido a que yo era el único miembro del equipo.

Salarios Personas*	
Analista - 75 horas	1.381,07 €
Diseñador - 112 horas	1.811,46 €
Experto IA – 167 horas	3.511,20 €
Desarrollador - 56 horas	1.490,30 €
Manager - 6 horas	238,40 €
Total Salarios Personas	*8.432,43 €*
Gastos Recursos Materiales**	
Ordenador	160,41 €
Office365	40,82 €
Windows 10 Pro	139,98 €
Unity	0,00 €
Total Gastos Recursos Materiales	*341,21 €*
Total	**8.773,64 €**

Tabla 2. Estudio económico [66] [67] [68] [69] [70] [71] [72] [73]

*Los sueldos por hora han sido calculados con el sueldo promedio indicado en las referencias (anual) en relación a las horas anuales que, en 2023, de media, es 1750 horas.

** El coste de cada recurso está calculado en base al precio del recurso, la vida útil del mismo, y la duración del recurso. Por ejemplo, el ordenador cuesta 1100€ y tiene una vida útil de 4 años (1100€ / 4 años / 12 meses = 22,92€/mes), como el proyecto ha durado 7 meses, su coste es 160,41€ (22,92€/mes x 7 meses = 160,41€).

La Tabla 2 refleja el precio a pagar para realizar este proyecto en una empresa si se contratara a profesionales que trabajasen las mismas horas que yo (teniendo en cuenta mis solapamientos de posiciones de trabajo).

6.2. Soluciones a futuro

Aquí detallaré cuales serían las soluciones viables para continuar con este estudio para ASAI en un futuro. Cabe mencionar que en estos estudios no he puesto las licencias de Sistemas Operativos ya que Windows no es obligatorio. De quererlo, se podría hacer con Linux de forma gratuita.

6.2.1. Cloud Computing (Recomendada)

La implementación de Machine Learning requeriría de un ordenador con un procesador y una gráfica potentes. Para optimizar la velocidad de entrenamiento, debemos aumentar el número de entornos entrenando de forma paralela en el entrenamiento masivo. Para ello, cuanto más potentes sean estas piezas, mejor.

Lo ideal sería una red de ordenadores para poder entrenar con Cloud Computing. Existen varias opciones en el mercado que ofrecen servicios de Cloud Computing con tarifas por hora [74]:

- IBM Cloud
- Microsoft Azure
- Amazon Web Services
- Google Cloud Platform
- Oracle Cloud

Con esta opción, los costes serían los sueldos de los empleados, la tarifa de de Cloud Computing y el coste de la amortización del ordenador usado para preparar el entorno.

La preparación del entorno ha llevado unos 240 horas (no se cuentan ni la preparación previa ni las investigaciones, un experto no lo requeriría). El Cloud Computing no se puede estimar, ya que la duración del entrenamiento varía salvajemente dependiendo de cientos de factores. Una predicción informada pondría el entrenamiento en al menos 300 horas, suponiendo un entorno que pueda soportar más de 200 entornos para entrenamiento masivo. En base a estas estimaciones, se harán los cálculos en base a 3 meses de trabajo. También debemos añadir el precio de las licencias de software en base al tiempo que se han usado. Todo esto se ve reflejado en la Tabla 3.

Sueldo medio informático de Machine Learning	9.198,00 €
Cloud Computing	100,00 €
Coste de 3 meses de ordenador	72,00 €
Licencias (Office 365 Personal)	17,49 €
Total	**9.387,49 €**

Tabla 3. Estudio económico con Cloud Computing [75] [71] [68] [67]

Todas estas opciones ofrecen Cloud Computing para Machine Learning e Inteligencia Artificial, y tienen un abanico de tarifas para ajustarse a lo que necesitase ASAI.

Lo importante sería elegir la opción que se ajuste al tipo de entrenamiento que realicen en ASAI, ya que cada uno soporta diferentes programas, APIs y métodos de tratar los datos.

6.2.2. Ordenador dedicado

En caso de no querer depender de servicios externos, otra posibilidad es comprar un PC capaz de realizar estos entrenamientos.

Para una empresa de robótica como ASAI, recomiendo un i9 (hay varias opciones, Intel tiene el 13900K con 24 cores), 64GB o más de memoria DDR5, una o más tarjetas gráficas RTX4090 de 24GB y una placa madre Asus TUFGAMING Z790+D4. Para todo esto, se recomienda fervientemente instalar refrigeración líquida.

El presupuesto para montar este ordenador sería:

Intel13900K	647,14 €
Memoria DDR5 64GB	207,99 €
Nvidia RTX4090	1.899,90 €
Asus TUFGAMING Z790+	370,00 €
Refrigeración Líquida	240,00 €
Total	**3.365,03 €**

Tabla 4. Estudio del coste de un ordenador dedicado [76] [77] [78] [79] [68]

En la Tabla 4 se estudia el precio de montar un ordenador apropiado, aunque el Cloud Computing a menudo será más potente. Dado el uso que se planea dar al ordenador y, la vida útil media de estas piezas, para una empresa se estima que el ordenador tendría una vida útil de 4 años.

Si se elige esta opción, el precio final del proyecto, suponiendo que el empleado utilice el tiempo de entrenamiento en otro proyecto de la empresa, se ve reflejado en la Tabla 5:

Sueldo medio informático de Machine Learning	9.198,00 €
Licencias (Office 365 Personal)	17,49 €
Coste de 3 meses de ordenador	210,31 €
Total	**9.425,80 €**

Tabla 5. Estudio económico con ordenador dedicado [71] [68] [67]

Si bien los precios parecen muy similares, hay que plantearse si ese ordenador será usado para más cosas a parte de este proyecto antes de realizar la inversión.

7. Resultados

En esta sección, se hablará de los resultados conseguidos con la finalización de este proyecto así como de las desviaciones que han ocurrido al intentar seguir la metodología elegida.

7.1. Resultados de entrenamientos

Durante el proyecto, se ha realizado un gran número de entrenamientos en entornos distintos y con configuraciones variadas. Estos entrenamientos proporcionan unos datos muy valiosos para futuros entrenamientos o estudios similares.

Dichos entrenamientos serán adjuntados en la sección de Anexos debido al número de tablas.

7.2. Artefactos producidos

Aquí hablaré sobre los artefactos producidos durante el proyecto. Principalmente sobre el entorno generado pero también de las Lecciones Aprendidas.

7.2.1. Entornos de entrenamiento

El proyecto pretendía estudiar la viabilidad de enseñar a un brazo robótico a inferir las trayectorias necesarias para realizar tareas de "pick and place" (coger y dejar) en un entorno con obstáculos dinámicos.

Uno de los principales artefactos generados durante el proyecto es el entorno final para el UR10e. Este entorno cuenta con un escenario modelado, configurado y programado para poder continuar con el estudio de viabilidad para cualquier trayectoria, no sólo la que está modelada. Bastaría mover de sitio en Unity las mesas con las goals para estudiar otras trayectorias.

El proyecto también procura un entorno similar para el robot RSv1, para futuros estudios.

7.2.2. Lecciones Aprendidas

También se han recopilado lecciones aprendidas durante este proyecto que serán importantes si ASAI se plantea continuar con el estudio.

7.2.2.1. Enlazar entrenamientos

Aunque muchos usuarios de Unity y MLAgents afirmen que no se puede continuar un entrenamiento más allá del número máximo de steps, durante el proyecto se descubrió una forma de enlazar un entrenamiento "acabado" a los 50 millones de steps con uno nuevo para darle un nuevo set de steps. Se ahonda más en el tema en la sección 9.3.2.1. Entrenamientos enlazados.

7.2.2.2. Entrenamiento combinado

Durante el proyecto, se ha concluido que el entrenamiento combinado (imitación más refuerzo) es la manera más rápida y eficiente de entrenar un modelo tan complicado como el propuesto. Si fuera un modelo más simple, podría simplemente usarse entrenamiento por refuerzo y finalizar el estudio en un tiempo razonable.

7.2.2.3. Qué tipo de articulaciones usar

Después de probar varios tipos de articulaciones y formas de aplicarles fuerza, se descubrió qué tipo de articulaciones y configuraciones son las óptimas para este proyecto, como se describe en el capítulo 5.4 de esta memoria.

7.2.2.4. Complejidad exponencial en base al número de articulaciones

Tras varias pruebas con brazos de diferente número de articulaciones, se ha llegado a la conclusión de que aumentar el número de articulaciones aumenta exponencialmente la duración del entrenamiento. Un Agente, cuando sólo tiene una articulación para su vector de acciones puede inferir rápidamente, con el vector de observaciones, de qué manera sus acciones afectan a las observaciones. Cuando se aumenta el número de articulaciones, las acciones sobre primera articulación ya no son las únicas que afectan las observaciones, si no que ha de calcular todas las posibles posiciones de la primera articulación con todas las posibles de la segunda. En seguida se puede intuir de qué manera exponencial aumenta el número posible de combinaciones cuando añadimos más articulaciones, lo que da paso a la lección aprendida más importante. La necesidad de más poder de procesamiento.

7.2.2.5. Necesidad de más poder de procesamiento

Una vez finalizado el proyecto, se ha llegado a la conclusión categórica de que es necesario un mayor poder de procesamiento. Es recomendable hacer uso de las tecnologías de Cloud Computing. De no quererse, es posible utilizar un ordenador dedicado o un cluster de ellos para este tipo de estudios y entrenamientos. Esto ya ha sido detallado en la sección 6.2 en el Estudio Económico.

7.3. Desviaciones de la metodología

A parte de algunos retrasos al final de los Sprints, lo único que merece ser mencionado son las tareas OB1-T04, OB2-T02 y OB2-T05. Estas tareas se daban por finalizadas pero uno o dos Sprints más tarde se le tuvieron que dedicar más horas. Esto fue debido a falta de experiencia en el caso de la tarea OB1-T04 y a falta de planificación y error de supervisión (mío propio) para las tareas OB2-T02 y OB2-T05.

En el caso de la tarea OB4-T01, aunque parezca que vuelve a estar activa tras acabar la tarea OB4-T02, no se debe a una desviación de la metodología. Como expliqué en el capítulo 5.6 sobre el Sprint 6, se debe a que hasta el último día antes de la entrega se ha estado entrenando un modelo de menos articulaciones de la tarea OB4-T01 para intentar conseguir más datos.

8. Conclusiones

El proyecto constaba de 4 objetivos principales, estudiar el problema, diseñar un sistema que intente resolverlo, implementarlo y estudiar los datos teóricos con los reales. ASAI dejó claro en las reuniones que el objetivo principal del proyecto era una investigación de viabilidad. Para ello, me prestaron recursos y conocimientos. Sin embargo, tanto ASAI como yo estábamos faltos de experiencia en Machine Learning.

El proyecto ha cumplido con los objetivos de diseño e implantación así como del estudio sobre los datos teóricos y reales. No ha podido tocarse mucho sobre visión artificial real, pero sí sobre la virtual. Esto fue porque no llegamos a ese paso en el proyecto.

No obstante, el proyecto ha arrojado mucha luz sobre la investigación de viabilidad y, creo que es perfectamente viable, pero con más recursos computacionales. Muchas sesiones de entrenamiento fueron muy prometedoras y, de poseer los recursos apropiados, estoy convencido de que se podrán realizar.

Personalmente, me habría gustado tener más tiempo y recursos que dedicar a este proyecto. Ha sido una tarea enriquecedora y directamente relacionada con el ámbito en el que quiero trabajar en el futuro: Inteligencia Artificial aplicada a máquinas.

El momento más memorable para mí fue entrenar el robot de 3 articulaciones, el RSv1. Como disponía de los recursos para entrenar este modelo, estaba muy animado porque por fin veía resultados, aunque sabía que le harían falta varios cientos de millones de steps más para completarse. Ver al robot por fin entender su entorno fue emocionante. Me paré a ver con detalle cómo se movía el robot durante centenares de episodios y, estaba claro que el Agente ya había comprendido dónde estaba el suelo y las mesas, ya que sólo acababa el episodio cuando se chocaba contra sí mismo. Era realmente prometedor.

Todo esto fue posible gracias a los conocimientos que adquirí en el primer Sprint pero, no habría podido realizar ni eso si no fuera por lo aprendido diversas asignaturas de la carrera. Sistemas Inteligentes me dio unas bases muy sólidas sobre el funcionamiento del Machine Learning, y pude extrapolarlas a un entorno 3D. Introducción a la Informática Gráfica fue fundamental para poder manejarme en el entorno de Unity. Ingeniería del Software y las clases de Gestión de Proyectos por darme a conocer las metodologías ágiles y, en concreto SCRUM. Por supuesto, absolutamente nada habría sido posible sin todas las clases recibidas de programación.

El mayor problema que se ha encontrado este proyecto ha sido la falta de recursos personales. Lamentablemente, no pude hacer uso de todos los recursos físicos de ASAI debido a problemas de disponibilidad por mi parte. Sin embargo, las conclusiones sacadas durante este proyecto son prometedoras.

Sugiero encarecidamente que se continúe el estudio empleando tecnologías de Cloud Computing para el entrenamiento. Esto eliminará cualquier problema que se tenga de recursos y, además, permitiría continuar el estudio de forma remota desde cualquier equipo.

La razón por la que esta tarea requería tantos recursos ha sido discutida a fondo en la memoria pero, para resumir, es la complejidad del entorno y, sobre todo, la forma en la que las acciones afectan a las observaciones.

El problema no es el tamaño del vector de acciones o el de observaciones, si no el hecho de que todas las acciones del vector de acciones afectan al resto ya que todas las acciones están conectadas en un mismo cuerpo. Si el vector de acciones, aunque fuese de tamaño 20, fuese de acciones totalmente inconexas unas con otras no habría ningún problema de crecimiento exponencial. Imaginemos que son 20 acciones sobre 20 objetos con 20 observaciones que nada tienen que ver una con otra. Imaginemos que la cada una de las acciones del vector de acciones cambiase algo en el entorno que se viese reflejado en solamente una de las posiciones del vector de observaciones. Esto crearía un problema de crecimiento aritmético. Sin embargo, dado que cada una de las acciones del vector de acciones de este proyecto afecta a todas las posiciones del vector de observaciones, se crea un problema de crecimiento exponencial. Además, el proyecto entre manos siempre tuvo un Vector de Acciones de tamaño 6 y uno de Observaciones de tamaño entre 22-42, lo que agravaba más aún el problema, porque cada una de las acciones podía afectar hasta más de un 80% de las observaciones.

No obstante, con Cloud Computing estos problemas de procesamiento serían algo del pasado y, además, se podría multiplicar más veces el número de escenarios del entorno, aumentando aún más la velocidad de entrenamiento.

De conseguir entrenar este modelo con Cloud Computing, sería muy factible completar otro tipo de tareas, incluso más precisas, en el mismo entorno de entrenamiento provisto.

Bibliografía

[1] IBM, «IBM,» [En línea]. Available: https://www.ibm.com/es-es/topics/automation.

[2] ASAI, «ASAI,» [En línea]. Available: https://asai.es/.

[3] P. S. S. R. H. Alberto Brunete, «Bookdown,» [En línea]. Available: https://bookdown.org/alberto_brunete/intro_automatica/breve-historia-de-la-automatica.html.

[4] TESISMARGOT, «tesis.uson,» [En línea]. Available: http://tesis.uson.mx/digital/tesis/docs/21319/capitulo1.pdf.

[5] [En línea]. Available: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.timetoast.com%2Ftimelines%2Fel-reloj-fb349a4a-6af0-4f83-9a0a-cb9891f0817c&psig=AOvVaw3rvR-nP-LMpKugQfbx5J4F&ust=1691683964302000&source=images&cd=vfe&opi=89978449&ved=0CBEQjRxqFwoTCLDqrc_7z4ADFQAAAAAdAA.

[6] [En línea]. Available: https://www.thisiscolossal.com/wp-content/uploads/2013/11/boy-1.jpg.

[7] «britannica,» [En línea]. Available: https://www.britannica.com/technology/printing-press.

[8] [En línea]. Available: https://images.squarespace-cdn.com/content/v1/60416644b68a5453a868e856/1621450844405-WFW4V5AJA2AF3X5Y1BXX/Gutenberg.jpg.

[9] «manufactura-latam,» [En línea]. Available: https://www.manufactura-latam.com/es/noticias/evolucion-de-la-automatizacion-industrial.

[10] «automaticacionindustrial360,» [En línea]. Available: https://automatizacionindustrial360.com/historia/.

[11] «industriasgsl,» [En línea]. Available: https://industriasgsl.com/blogs/automatizacion/que-es-un-plc-y-como-funciona.

[12] [En línea]. Available: https://theplctutor.com/history.html.

[13] H. I. Made, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=K1VfdXmqjN8.

[14] [En línea]. Available: https://1.bp.blogspot.com/-9CQjtOt-UXM/UURUWK-O9iI/AAAAAAAAAEs/RnEOizEOgSc/s1600/trabajo+en+cadena.jpg.

[15] [En línea]. Available: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.eurobots.net%2F&psig=AOvVaw3eeB9RT8s_Q0Om5CpY02uz&ust=1691684113734000&source=images&cd=vfe&op

i=89978449&ved=0CBEQjRxqFwoTCLimzJb8z4ADFQAAAAAdAAAAABAE.

[16] A. Robots, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=2_1OeGjEihs.

[17] A. Robotics, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=HUU3HdxOqZs&t=1s.

[18] R. AG, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=YuXSYfMm7nY.

[19] J. Engineering, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=QfbdVboVNUM.

[20] H. Robotics, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=8MXflIcmKII.

[21] «MoreLab,» [En línea]. Available: https://morelab.deusto.es/people/former_members/.

[22] I. Vazquez, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=vmCFBdE1FQ8.

[23] S. Design, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=HOUPkBF-yv0.

[24] M. CODE, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=Vkj-trZ2NRw.

[25] J. Hiwonder, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=sDDF6Tyi4U0.

[26] Adobe, «Adobe Experience Cloud Blog,» [En línea]. Available: https://business.adobe.com/blog/basics/waterfall#:~:text=The%20Waterfall%20methodology%20%E2%80%94%20also%20known,before%20the%20next%20phase%20begins..

[27] EBF, «ebf,» [En línea]. Available: https://ebf.com.es/blog/ventajas-y-desventajas-de-las-metodologias-agiles-y-su-aplicacion-en-el-trabajo/.

[28] «coscreen,» [En línea]. Available: https://www.coscreen.co/blog/extreme-programming-vs-scrum-difference/.

[29] «scrum.org,» [En línea]. Available: https://www.scrum.org/learning-series/what-is-scrum/what-is-scrum.

[30] «scrum.org,» [En línea]. Available: https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-a-sprint.

[31] «scrum.org,» [En línea]. Available: https://www.scrum.org/learning-series/what-is-scrum/the-scrum-artifacts/what-is-a-product-backlog.

[32] «Universal Robots,» [En línea]. Available: https://a.storyblok.com/f/169662/5873x4405/a77477d600/ur10e-4x3.png.

[33] G. M. Toolkit, «youtube,» [En línea]. Available: https://www.youtube.com/watch?v=XtQMytORBmM.

[34] simoninithomas, «Github,» [En línea]. Available: https://github.com/Unity-Technologies/ml-agents.

[35] «microsoft.com,» [En línea]. Available: https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/.

[36] miguelalonsojr, «Github,» [En línea]. Available: https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Getting-Started.md.

[37] TyLindberg, «Github,» [En línea]. Available: https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Installation.md.

[38] P. D. Sielder, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=Yix4iV_io6o&t=563s.

[39] C. Monkey, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=zPFU30tbyKs&t=791s.

[40] miguelalonsojr, «Github,» [En línea]. Available: https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Learning-Environment-Examples.md.

[41] «CGTrader,» [En línea]. Available: https://www.cgtrader.com/items/3281881/download-page.

[42] Yughues, «Unity Asset Store,» [En línea]. Available: https://assetstore.unity.com/account/assets.

[43] «Aspose,» [En línea]. Available: https://products.aspose.app/3d/conversion/3mf-to-obj.

[44] Jassper-NIelsen, «Unity Answers,» [En línea]. Available: https://answers.unity.com/questions/837243/resize-an-object-without-scaling.html.

[45] Crandemolisher9, «Unity Answers,» [En línea]. Available: https://answers.unity.com/questions/1727791/resize-a-prefab-without-affecting-scale.html.

[46] «Amazon,» [En línea]. Available: https://www.amazon.es/mDesign-redonda-almacenamiento-armario-grande/dp/B01H45MMUO?th=1.

[47] Guidev, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=VdJGouwViPs&t=966s.

[48] «Unity3D Docs,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/Transform.Rotate.html.

[49] «Universal Robots,» [En línea]. Available: https://www.universal-robots.com/media/1807466/ur10e-rgb-fact-sheet-landscape-a4-125-kg.pdf.

[50] M. Bit, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=rXYjB0kdf-U.

[51] «amazon,» [En línea]. Available: https://www.amazon.es/Jam%C3%B3n-Serrano-100-Duroc-Teruel/dp/B08KJCMK4Y/ref=sr_1_241?keywords=jamon+serrano&qid=1694279706&sr=8-241.

[52] GameDevTraum, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=ZHr3v8Ewxc0&t=23s.

[53] «Unity3D Docs,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/GameObject.Find.html.

[54] C. Monkey, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=zPFU30tbyKs&t=491s.

[55] V.-P. B. J. S. d. D. P. J. W. a. Chris Elion, «GitHub,» [En línea]. Available: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Glossary.md.

[56] [En línea]. Available: https://album.mediaset.es/eimg/2022/12/03/se-cumplen-50-anos-del-arcade-pong_a858.jpg?w=480.

[57] maryamhonari, «GitHub,» [En línea]. Available: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md#common-trainer-configurations.

[58] «forum.unity,» [En línea]. Available: https://forum.unity.com/threads/ml-agents-train-an-already-trained-model.840892/.

[59] andrzej_, «Unity Forum,» [En línea]. Available: https://forum.unity.com/threads/drop-in-performance-after-resume.919535/.

[60] fquarters, «Github,» [En línea]. Available: https://github.com/Unity-Technologies/ml-agents/issues/5781.

[61] dlindmark, «Github,» [En línea]. Available: https://github.com/Unity-Technologies/ml-agents/issues/4459.

[62] ChilliCollective1, «Unity Forums,» [En línea]. Available:

https://forum.unity.com/threads/probuilder-vs-blender.462719/.

[63] «Universal Robots,» [En línea]. Available: https://www.universal-robots.com/download/mechanical-e-series/ur10e/robot-step-file-ur10e-e-series/.

[64] e. D. P. J. S. V. G. A. J. M. P. D. A. J. W. J. G. Vincent-Pierre Berges, «Github,» [En línea]. Available: https://github.com/yosider/ml-agents-1/blob/master/docs/Training-Imitation-Learning.md.

[65] C. Monkey, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=supqT7kqpEI.

[66] «PCComponentes,» [En línea]. Available: https://www.pccomponentes.com/microsoft-windows-10-pro-32-64-bit-licencia-completa-fpp-digital.

[67] «Microsoft,» [En línea]. Available: https://www.microsoft.com/en-us/microsoft-365/buy/compare-all-microsoft-365-products.

[68] A. E. Castellote, Interviewee, *Técnico Informático.* [Entrevista].

[69] «glassdoor,» [En línea]. Available: https://www.glassdoor.es/Sueldos/analista-programador-sueldo-SRCH_KO0,20.htm#:~:text=La%20remuneraci%C3%B3n%20adicional%20media%20en,de%20Analista%20Programador%20en%20Espa%C3%B1a..

[70] «Glassdoor,» [En línea]. Available: https://www.glassdoor.es/Sueldos/3d-artist-sueldo-SRCH_KO0,9.htm#:~:text=La%20remuneraci%C3%B3n%20adicional%20media%20en,de%203D%20Artist%20en%20Espa%C3%B1a..

[71] «Glassdoor,» [En línea]. Available: https://www.glassdoor.es/Sueldos/machine-learning-sueldo-SRCH_KO0,16.htm#:~:text=En%20Espa%C3%B1a%2C%20el%20sueldo%20medio,Learning%20es%20de%20%E2%82%AC36.794%20..

[72] «Glassdoor,» [En línea]. Available: https://www.glassdoor.es/Sueldos/c-net-developer-sueldo-SRCH_KO0,15.htm#:~:text=En%20Espa%C3%B1a%2C%20el%20sueldo%20medio,trabajan%20de%20C%20net%20developer..

[73] «Glassdoor,» [En línea]. Available: https://www.glassdoor.es/Sueldos/gerente-de-ti-sueldo-SRCH_KO0,13.htm#:~:text=El%20sueldo%20medio%20para%20el,4982%20%E2%82%AC%20y%2016.119%20%E2%82%AC..

[74] Sidharth, «Pycodemates,» [En línea]. Available: https://www.pycodemates.com/2022/06/top-cloud-computing-platforms-for-machine-learning.html.

[75] «AWS,» [En línea]. Available: https://aws.amazon.com/es/getting-started/projects/build-machine-learning-model/services-costs/.

[76] OCASIONA, «Amazon,» [En línea]. Available: https://www.amazon.es/Intel%C2%AE-Procesador-Escritorio-i9-13900K-n%C3%BAcleos/dp/B0BG67ZG5R/ref=sr_1_1?adgrpid=132967634593&hvadid=601334410054&hvdev=c&hvlocphy=1005548&hvnetw=g&hvqmt=e&hvrand=513063705520580257 5&hvtargid=kwd-1456652581116&hydadcr=5042_2.

[77] «PC Componentes,» [En línea]. Available: https://www.pccomponentes.com/corsair-vengeance-ddr5-5600mhz-64gb-2x32gb-cl40-negra.

[78] «PC Componentes,» [En línea]. Available: https://www.pccomponentes.com/gigabyte-geforce-rtx-4090-gaming-oc-24gb-gddr6x-dlss3.

[79] «Asus,» [En línea]. Available: https://www.asus.com/es/motherboards-components/motherboards/tuf-gaming/tuf-gaming-z790-plus-d4/.

[80] P. S. S. R. H. Alberto Brunete, «Bookdown,» [En línea]. Available: https://bookdown.org/alberto_brunete/intro_automatica/automatizacionindustrial.html.

[81] [En línea]. Available: https://st4.depositphotos.com/21607914/23626/i/1600/depositphotos_236264566-stock-photo-file-chinese-workers-produce-electronic.jpg.

[82] «github,» [En línea]. Available: https://github.com/Unity-Technologies/ml-agents.

[83] [En línea]. Available: https://m.media-amazon.com/images/I/41YivlNyhYL.jpg.

[84] «usj.es,» [En línea]. Available: https://pdu.usj.es/pluginfile.php/148375/mod_resource/content/12/Manual%20Proyecto%20Fin%20de%20Grado%2022-23_Ingenieri%CC%81a%20Informa%CC%81tica.pdf.

9. Anexos

En esta sección, se añadirán imágenes y tablas pertinentes a esta memoria para ampliar detalle si fuera necesario. Se exponen aquí porque hacerlo en las secciones explicativas haría muy farragosa la lectura. Además, se proveerá de un pen drive con los distintos proyectos de Unity con todos los entornos y modelos que se han usado para los entrenamientos.

9.1. Propuesta del Proyecto

universidad SANJORGE GRUPO SANVALERO

Propuesta de Proyecto Final

Nombre alumno: Iñigo Escalante Escarda

Titulación: Ingeniería Informática

Curso académico: 4º

1.TÍTULO DEL PROYECTO

Visión artificial para el cálculo de trayectorias de brazos robóticos en entornos dinámicos

2.DESCRIPCIÓN Y JUSTIFICACIÓN DEL TEMA A TRATAR

El proyecto consiste en el estudio y la implementación de un sistema basado en visión artificial para el cálculo de trayectorias eficientes para la evitación de obstáculos en entornos dinámicos e impredecibles. El objetivo es que un brazo robótico pueda planificar una trayectoria a lo largo de una cinta transportadora en movimiento llena de obstáculos evitando toda colisión en su desplazamiento. El proyecto está enmarcado en el ámbito real de trabajo de una empresa de robótica aragonesa.

3.OBJETIVOS DEL PROYECTO

Los objetivos del proyecto incluyen:

- Estudio del problema y de las capacidades de la visión artificial disponible en los robots disponibles
- Diseño de un sistema capaz de resolver la situación descrita.
- Implementación del sistema utilizando inteligencia artificial y/o machine learning
- Estudio y comparación de resultados teóricos y prácticos

4.METODOLOGÍA

La metodología se establecerá en las primeras fases del proyecto y variará para las partes de estudio, diseño e implementación.

5.PLANIFICACIÓN DE TAREAS

Las tareas se definirán de acuerdo a los objetivos, se fijarán en la fase de diseño y se refinarán posteriormente para adecuarse a la fase de implementación.

6.OBSERVACIONES ADICIONALES

FI-030 - 1 - Rev.0

9.2. Actas de las reuniones

En esta sección, se presentarán las actas de las diversas reuniones que se han tenido sobre este proyecto.

REUNIÓN / CONTACTO: Impresiones iniciales sobre el trabajo

Fecha	02/11/2022
Método	Reunión en la USJ
Elabora acta	Iñigo Escalante Escarda
Convocados	Ana Marcén

Asuntos a tratar

Número	Asunto
1	Valoración de la propuesta de proyecto
2	Discutir metodología de trabajo
3	Primeros pasos

Acuerdos

Número	Acuerdo
1	Elaborar un Product Backlog
2	Hablar con ASAI

REUNIÓN / CONTACTO: Primera reunión con ASAI

Fecha	09/11/2022
Método	Whatsapp y llamadas telefónicas
Elabora acta	Iñigo Escalante Escarda
Convocados	Rubén de la Rubia

Asuntos a tratar

Número	Asunto
1	Informar de la asignación del proyecto
2	Concretar detalles sobre los objetivos del proyecto
3	Hablar del proyecto de Hornero (alumni de la USJ)

Acuerdos

Número	Acuerdo
1	
2	Se mirará lo del proyecto de Hornero

REUNIÓN / CONTACTO: Comienzo del proyecto

Fecha	02/03/2023
Método	Teams
Elabora acta	Iñigo Escalante Escarda
Convocados	Ana Marcén

Asuntos a tratar

Número	Asunto
1	Valorar el Product Backlog
2	Valorar la metodología elegida (SCRUM)
3	Discutir sobre añadir el OB0
4	Comentar la reunión con ASAI

Acuerdos

Número	Acuerdo
1	Product Backlog aprobado
2	Metodología SCRUM aprobada
3	Objetivo OB0 aprobado
4	Comenzar el Sprint 1 en el día acordado
5	Programar reunión a finales del Sprint 1

REUNIÓN / CONTACTO: Segunda Reunión con ASAI

Fecha	22/04/2023-29/04/2023
Método	Whatsapp y llamadas telefónicas
Elabora acta	Iñigo Escalante Escarda
Convocados	Rubén de la Rubia

Asuntos a tratar

Número	Asunto
1	Informar de la situación del proyecto
2	Concertar cita para visitar ASAI para tomar medidas del entorno de trabajo a modelar
3	Hablar sobre el proyecto de Hornero
4	Hablar del tipo de obstáculos a esquivar
5	Hablar del robot que se debe modelar
6	Cómo enfocar el proyecto

Acuerdos

Número	Acuerdo
1	No se pudo concertar cita por problemas de disponibilidad de ambos
2	Se decide no involucrar el proyecto de Hornero. ASAI no enviará nada sobre dicho proyecto
3	Los obstáculos son principalmente cajas o palés en la zona de trabajo
4	Se proveerá a Iñigo del modelo exacto del robot a modelar para poder buscar la ficha técnica
5	Tratar el proyecto fundamentalmente como un trabajo de investigación y no como un proyecto enfocado a resultados

REUNIÓN / CONTACTO: Reunión sobre el Sprint 1

Fecha	23/04/2023
Método	Teams
Elabora acta	Iñigo Escalante Escarda
Convocados	Ana Marcén

Asuntos a tratar

Número	Asunto
1	Mostrar avances del proyecto
2	Discutir sobre el retraso con la tarea T02
3	Informar de la reunión con ASAI

Acuerdos

Número	Acuerdo
1	Continuar con la tarea T02 en el Sprint 2
2	Comenzar el Sprint 2 en el día acordado
3	Programar reunión a finales del Sprint 2

REUNIÓN / CONTACTO: Reunión sobre el Sprint 2

Fecha	13/06/2023
Método	Teams
Elabora acta	Iñigo Escalante Escarda
Convocados	Ana Marcén

Asuntos a tratar

Número	Asunto
1	Mostrar avances del proyecto
2	Discutir sobre el retraso con la tarea OB1-T07
3	Comentar dificultad de la instalación de las herramientas
4	Informar de lo decidido en la reunión con ASAI tras la anterior reunión sobre el Sprint 1

Acuerdos

Número	Acuerdo
1	Terminar la tarea OB1-T07 en el Sprint 3
2	Comenzar el Sprint 3 en el día acordado
3	Programar reunión a finales del Sprint 3
4	Reenfocar el proyecto a uno de investigación

REUNIÓN / CONTACTO: Reunión sobre el Sprint 3

Fecha	29/06/2023 (Reunión adelantada)
Método	Teams
Elabora acta	Iñigo Escalante Escarda
Convocados	Ana Marcén

Asuntos a tratar

Número	Asunto
1	Mostrar avances del proyecto
2	Hablar sobre cómo aprovechar el tiempo sobrante del Sprint 3

Acuerdos

Número	Acuerdo
1	Comenzar la tarea OB2-T05 en lo que quedaba de Sprint 3
2	Comenzar el Sprint 4 en el día acordado
3	Programar reunión a finales del Sprint 4

REUNIÓN / CONTACTO: Reunión sobre el Sprint 4

Fecha	21/07/2023 (Reunión adelantada)
Método	Teams
Elabora acta	Iñigo Escalante Escarda
Convocados	Ana Marcén

Asuntos a tratar

Número	Asunto
1	Mostrar avances del proyecto
2	Discutir sobre el retraso de la tarea OB2-T09
3	Discutir sobre el desvío en la metodología del día 18/07/2023

Acuerdos

Número	Acuerdo
1	Terminar la tarea OB2-T09 en el Sprint 5
2	Incluir desvío de la metodología en la memoria
3	Comenzar el Sprint 5 en el día acordado
4	Programar reunión a finales del Sprint 5

REUNIÓN / CONTACTO: Reunión sobre el Sprint 5

Fecha	10/08/2023
Método	Teams
Elabora acta	Iñigo Escalante Escarda
Convocados	Ana Marcén

Asuntos a tratar

Número	Asunto
1	Mostrar avances del proyecto
2	Comentar el problema del modelo defectuoso del brazo robótico
3	Discutir sobre el desvío en la metodología de las fechas 31/07/2023 – 03/08/2023
4	Hablar sobre el retraso general en este Sprint debido a problemas inesperados con el entrenamiento
5	Comentar las posibles soluciones a dichos problemas
6	Discutir la necesidad de añadir dos nuevas tareas: OB3-T02 y OB3-T03 con las posibles soluciones comentadas en el asunto 5

Acuerdos

Número	Acuerdo
1	Incluir desvío de la metodología en la memoria
2	Intentar un poco más la tarea OB3-T01 durante el Sprint 6
3	Aprobada la inclusión de las tareas OB3-T02 y OB3-T03
4	Las fechas de los Sprints no se cambiarán
5	Comenzar el Sprint 6 en el día acordado
6	Programar reunión a finales del Sprint 6

REUNIÓN / CONTACTO: Primera reunión sobre el Sprint 6

Fecha	23/08/2023 (Reunión adelantada)
Método	Teams
Elabora acta	Iñigo Escalante Escarda
Convocados	Ana Marcén

Asuntos a tratar

Número	Asunto
1	Mostrar avances del proyecto
2	Discutir la viabilidad de los nuevos entrenamientos
3	Discutir la posibilidad de añadir una tarea OB4-T00 sobre entrenar un brazo de 3 articulaciones

Acuerdos

Número	Acuerdo
1	Aprobada la inclusión de la tarea OB4-T00
2	Programar reunión a finales del Sprint 6

REUNIÓN / CONTACTO: Segunda reunión sobre el Sprint 6

Fecha	05/09/2023 (Reunión adelantada)
Método	Teams
Elabora acta	Iñigo Escalante Escarda
Convocados	Ana Marcén

Asuntos a tratar

Número	Asunto
1	Mostrar avances del proyecto
2	Discutir sobre el desvío en la metodología de las fechas 25/08/2023 – 28/08/2023
3	Comentar la probabilidad de acabar el entrenamiento antes de la fecha límite
4	Discutir sobre si realizar la tarea OB4-T01 ya

Acuerdos

Número	Acuerdo
1	Incluir desvío de la metodología en la memoria
2	Continuar entrenando el RSv1 hasta el último día
3	Empezar la tarea OB4-T01 independientemente de la tarea OB4-T00
4	Programar reunión para terminar de maquetar la memoria

REUNIÓN / CONTACTO: Reunión final

Fecha	09/09/2023 (Reunión adelantada)
Método	Teams
Elabora acta	Iñigo Escalante Escarda
Convocados	Ana Marcén

Asuntos a tratar

Número	Asunto
1	Mostrar avances de la tarea OB4-T00
2	Maquetar la memoria

Acuerdos

Número	Acuerdo
1	Proyecto listo para entrega

9.3. Tablas de entrenamiento

Aquí se mostrarán las tablas explicativas sobre las numerosas sesiones de entrenamiento con los diferentes modelos y escenarios.

9.3.1. Imágenes de los modelos y escenarios

Para contextualizar las tablas, se provee al lector de unas imágenes sobre cada uno de los modelos de brazo robótico utilizados durante los entrenamientos así como de los dos escenarios.

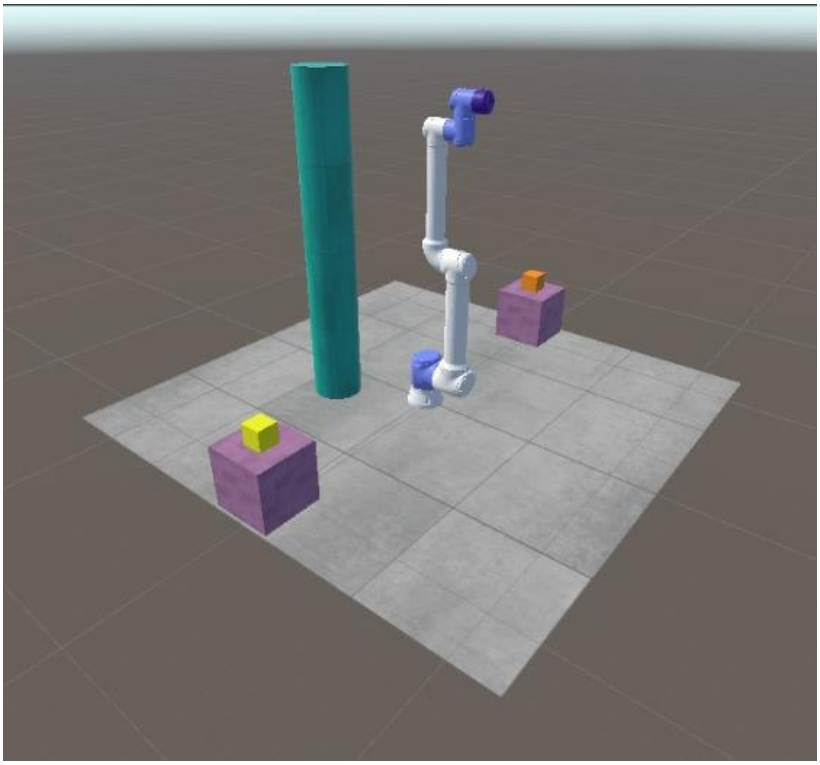

Figura 54. Modelo de brazo robótico UR10ev1

En la Figura 54, se puede observar el modelo UR10ev1 en el que se realizaron los primeros entrenamientos. Consta del modelo defectuoso de 5 articulaciones.

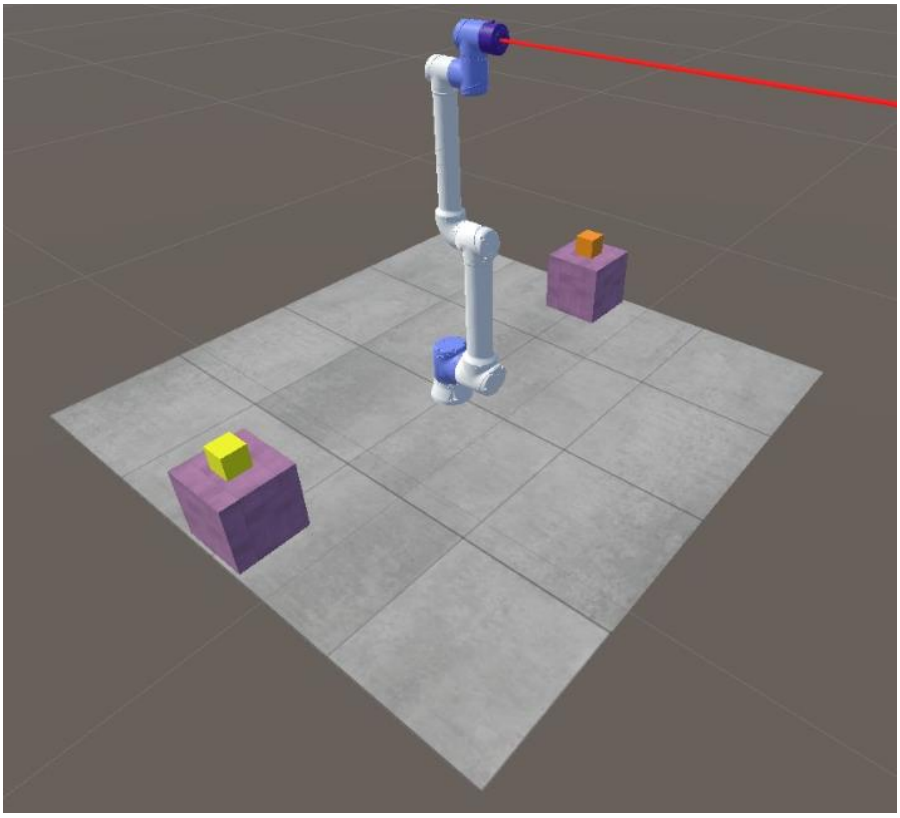

Figura 55. Modelo de brazo robótico UR10ev2

En la Figura 55, se puede observar el modelo UR10ev2 en el que se realizaron los entrenamientos de doble Agente. Consta del modelo defectuoso de 5 articulaciones, así como de un "haz láser" para visualizar mejor hacia dónde "miraba" la mano. Representa el vector normal del plano que atraviesa la mano.

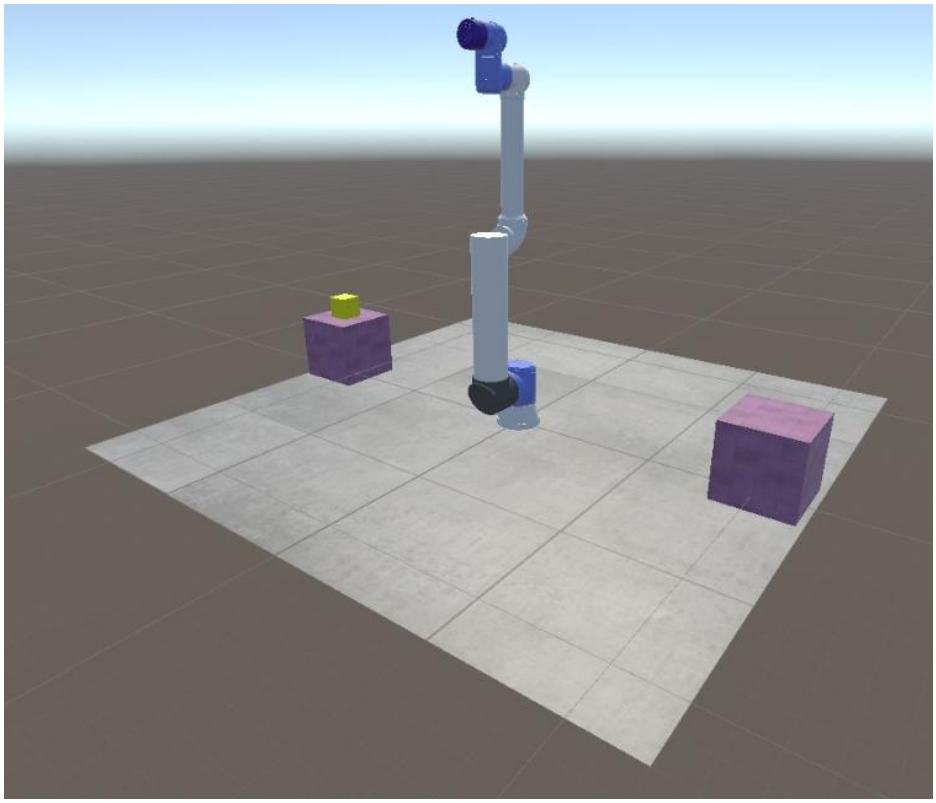

Figura 56. Modelo de brazo robótico UR10ev3

En la Figura 56, se puede observar el modelo UR10ev3 en el que se realizaron los últimos entrenamientos con el brazo robótico UR10e. Este es el modelo correcto de 6 articulaciones creado a partir de dos modelos distintos y una figura primitiva.

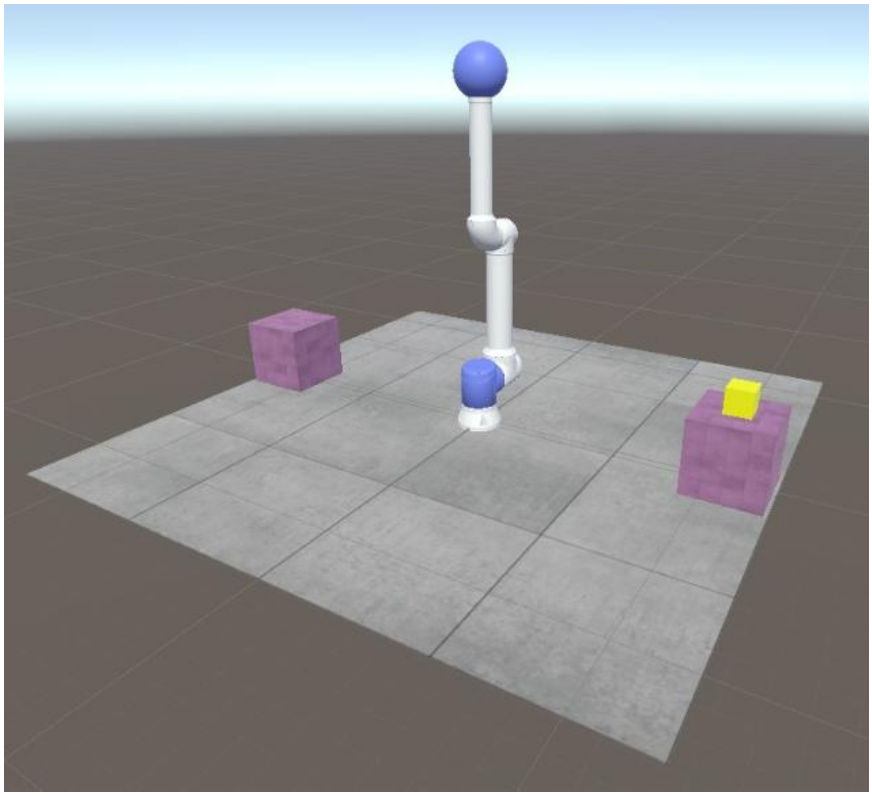

Figura 57. Modelo de brazo robótico RSv1

En la Figura 57, se puede observar el modelo RSv1 en el que se realizaron los entrenamientos sobre un brazo robótico de 3 articulaciones. La mano en este modelo es la esfera azul en la parte más alta del brazo. Consta de base rotatoria, hombro y codo.

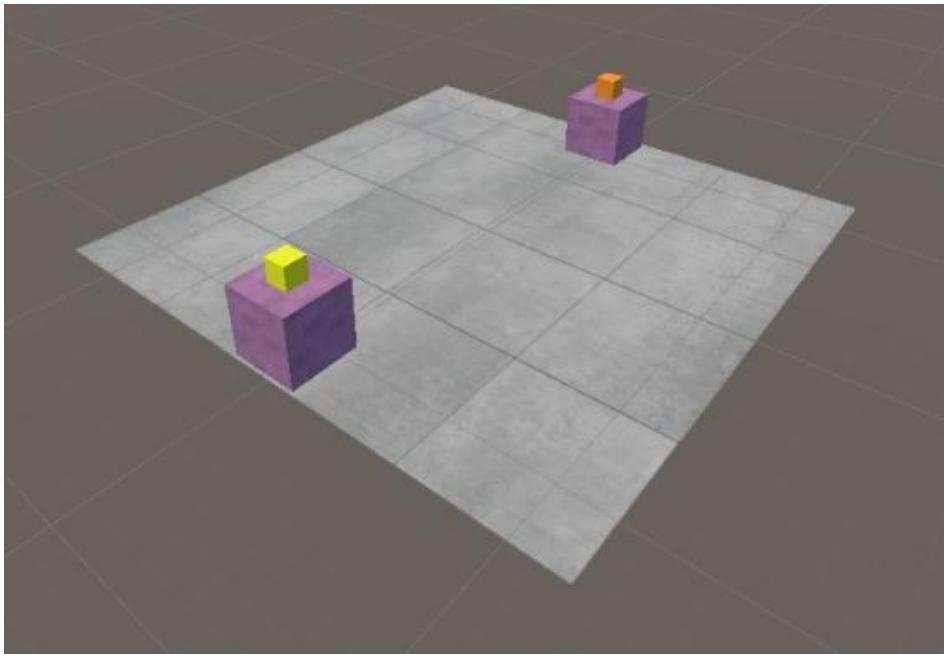

Figura 58. Modelo del escenario 1

En la Figura 58, se muestra el modelo de escenario 1. Este escenario contaba todavía con dos goals, siendo la amarilla la primera y la naranja la segunda. El Agente debía tocarlas con la mano en ese orden para que se cumpliese la Política.

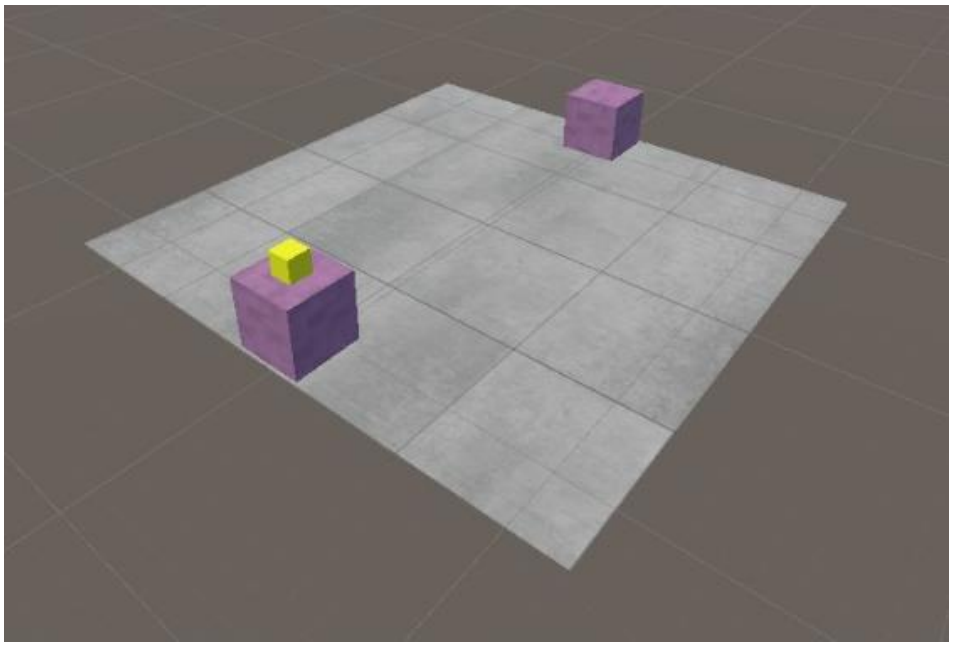

Figura 59. Modelo del escenario 2

En la Figura 59, se muestra el modelo de escenario 2. Este escenario había sido simplificado a una sola goal. Ésta se movería a la mesa opuesta tras ser tocada por la mano del brazo robótico. Esto redujo el tamaño del vector de observaciones.

9.3.2. Tablas de entrenamiento

En esta sección se incluirán las tablas sobre los entrenamientos realizados. Se indicará qué modelo de brazo robótico y escenario han sido empleados. Además, se indicarán otros datos interesantes y unas observaciones sobre cada entrenamiento.

Entrenamiento		**1**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	1
	Número de obstáculos	1
	Posición del obstáculo	Aleatoria
	Tamaño del Vector de Observaciones	42
	Nº máximo de steps	500.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Tiene una tendencia creciente pero nada más que mencionar. El entrenamiento fue muy corto
	Conclusiones	Entrenamientos iniciales. No merecen mucha mención

Tabla 6. Entrenamiento 1

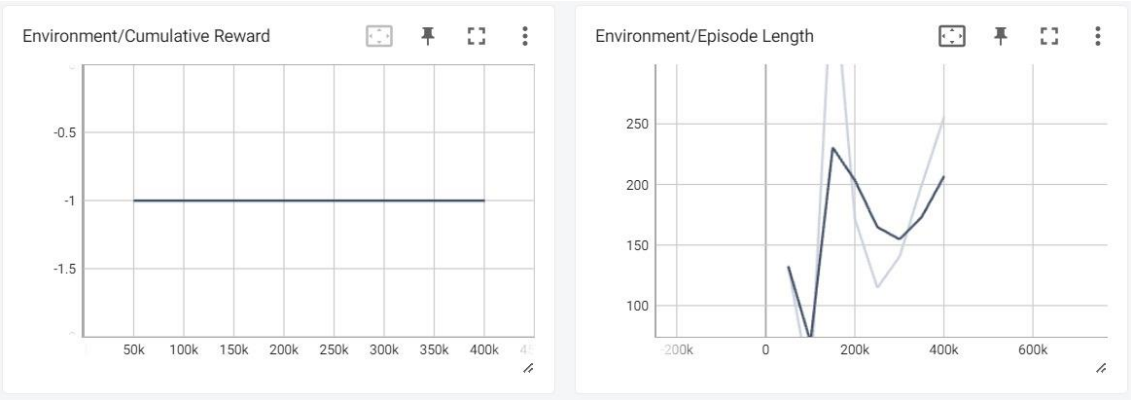

Figura 60. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 1

Entrenamiento		**2**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	1
	Número de obstáculos	1
	Posición del obstáculo	Aleatoria
	Tamaño del Vector de Observaciones	42
	Nº máximo de steps	500.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Tiene una tendencia creciente pero nada más que mencionar. El entrenamiento fue muy corto
	Conclusiones	Entrenamientos iniciales. No merecen mucha mención

Tabla 7. Entrenamiento 2

Figura 61. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 2

Entrenamiento		**3**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	1
	Número de obstáculos	1
	Posición del obstáculo	Aleatoria
	Tamaño del Vector de Observaciones	42
	Nº máximo de steps	500.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Tiene una tendencia creciente pero nada más que mencionar. El entrenamiento fue muy corto
	Conclusiones	Entrenamientos iniciales. No merecen mucha mención

Tabla 8. Entrenamiento 3

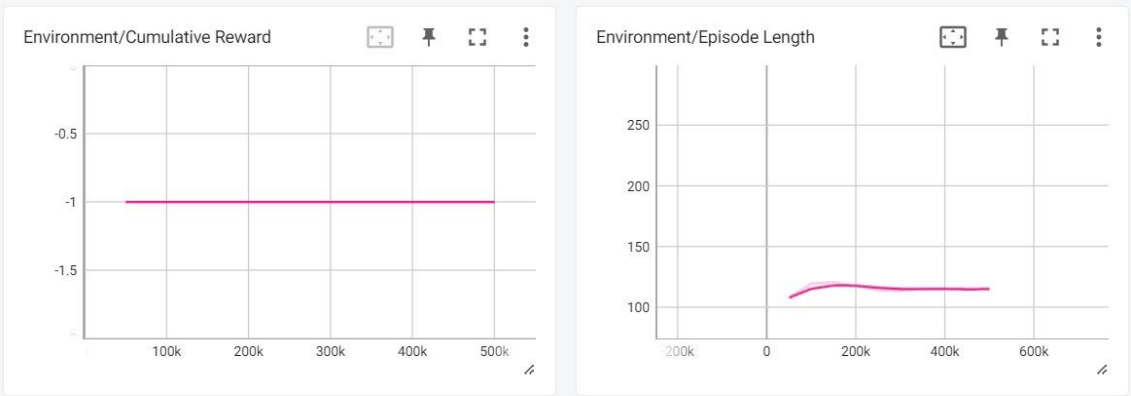

Figura 62. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 3

Entrenamiento		**4**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	1
	Número de obstáculos	1
	Posición del obstáculo	Aleatoria
	Tamaño del Vector de Observaciones	42
	Nº máximo de steps	500.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Tiene una tendencia creciente pero nada más que mencionar. El entrenamiento fue muy corto
	Conclusiones	Entrenamientos iniciales. No merecen mucha mención

Tabla 9. Entrenamiento 4

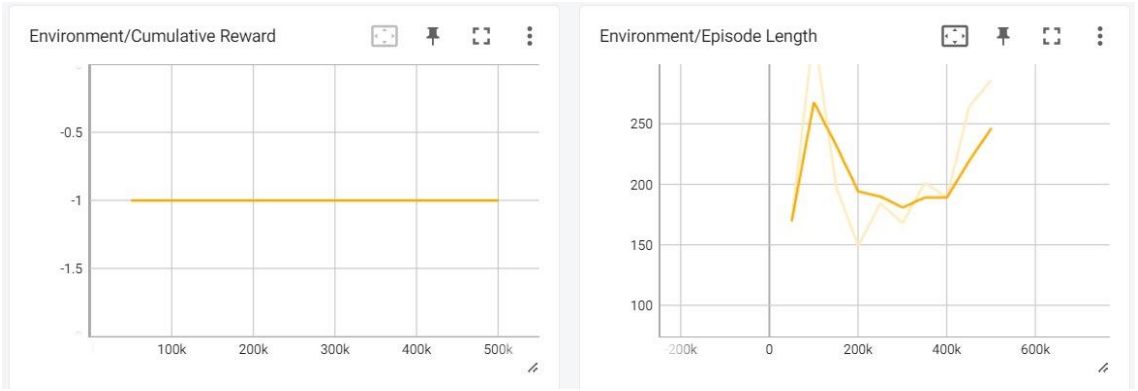

Figura 63. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 4

Entrenamiento		**5**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	1
	Número de obstáculos	1
	Posición del obstáculo	Aleatoria
	Tamaño del Vector de Observaciones	42
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Tiene una tendencia creciente pero nada más que mencionar. El entrenamiento fue muy corto
	Conclusiones	Este es el primer entrenamiento en el que la curva de la duración de los episodios es creciente durante la totalidad de los 500.000 steps

Tabla 10. Entrenamiento 5

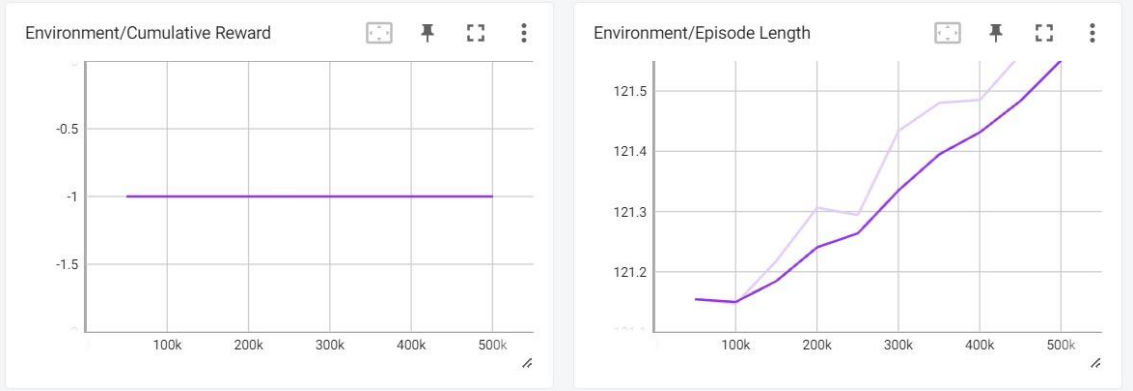

Figura 64. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 5

Entrenamiento		**6**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	1
	Número de obstáculos	1
	Posición del obstáculo	Aleatoria
	Tamaño del Vector de Observaciones	42
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Es el primer entrenamiento que supera los 1.000 steps de duración de episodio, llegando a poco más de 1.500
	Conclusiones	Aunque la programación ha mejorado la capacidad de exploración del Agente, su recompensa acumulada de -1 demuestra que no ha mejorado su capacidad de explotación

Tabla 11. Entrenamiento 6

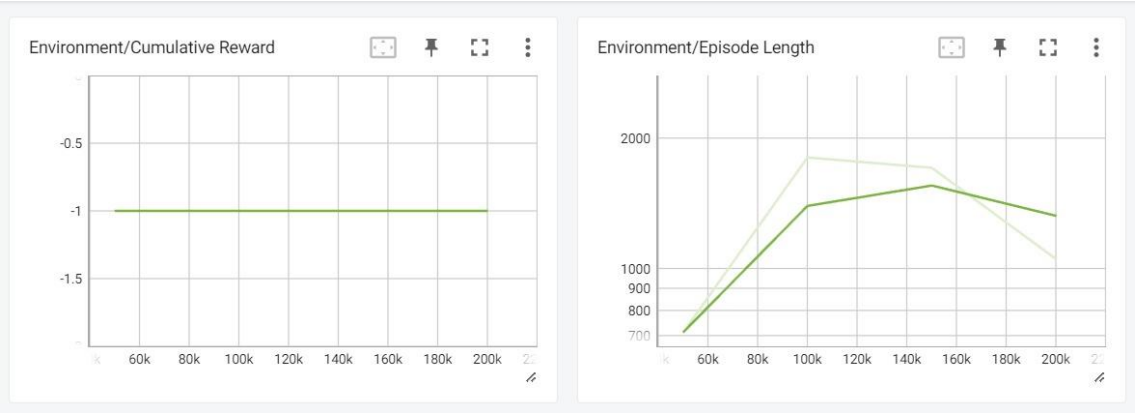

Figura 65. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 6

Entrenamiento		**7**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	1
	Número de obstáculos	1
	Posición del obstáculo	Aleatoria
	Tamaño del Vector de Observaciones	42
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	En este entrenamiento de duración más apropiada, se puede observar una tendencia creciente hasta que parece que se estabiliza
	Conclusiones	Este fue el primer entrenamiento en el que se superaron los 500.000 steps. Esto se debe a que, por fin, se tocaron los hiperparámetros

Tabla 12. Entrenamiento 7

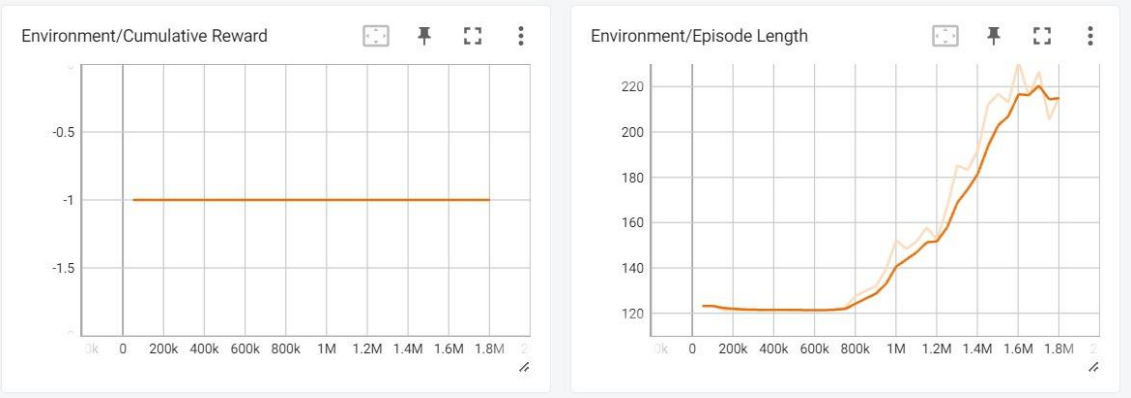

Figura 66. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 7

Entrenamiento		**8**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	29
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Justo antes de llegar al primer millón de steps y, durante aproximadamente medio millón de steps, consiguió recompensas de 0
	Duración de los episodios	Empieza por debajo de los 1.000 pero, según continua, crece hasta superar esa cota
	Conclusiones	Este fue el primer entrenamiento en el Agente consigue realizar una tarea de la política, tocar la primera goal. A partir del step 1,5M decide dedicar sus esfuerzos a la exploración. Crasheó a los 3.000.000 de steps

Tabla 13. Entrenamiento 8

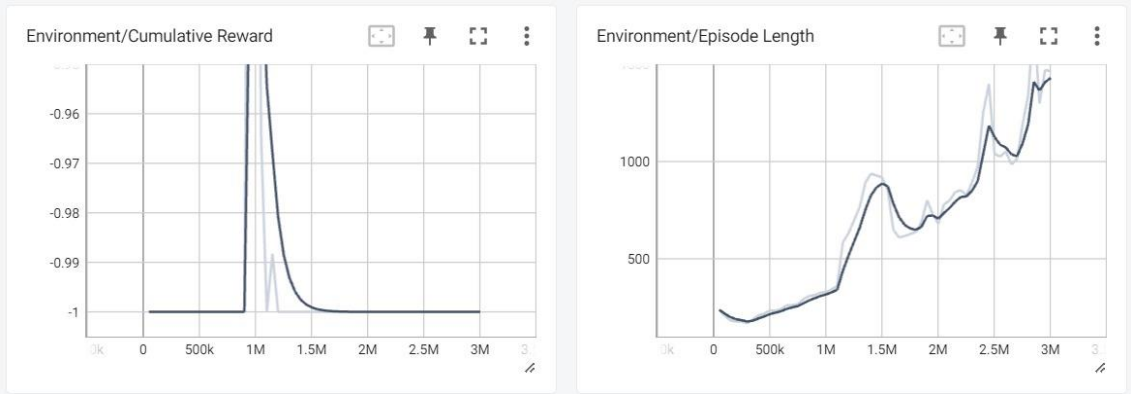

Figura 67. . Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 8

Entrenamiento		**9**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	29
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Se mantuvo en una duración de aproximadamente 125 steps durante los primeros 900.000 steps de entrenamiento. A partir de ahí creció rápidamente hasta estabilizarse, más o menos, en 1.200 steps
	Conclusiones	Estuvo chocándose inmediatamente contra el suelo durante los primeros 900.000 steps. Parece que decidió explorar en otras direcciones hasta que crasheó a los 1,4M de steps totales

Tabla 14. Entrenamiento 9

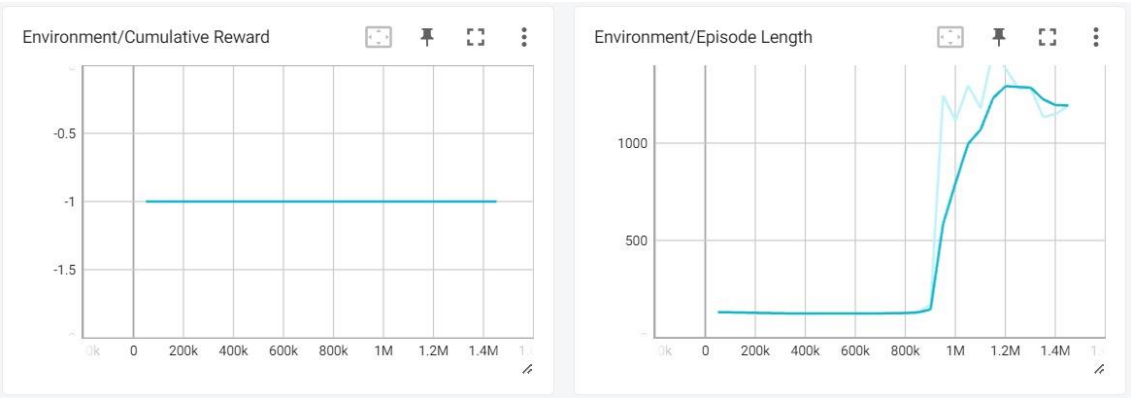

Figura 68. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 9

Entrenamiento		**10**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	29
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Se mantiene prácticamente todo el entrenamiento en 125 steps
	Conclusiones	Este fue el primer entrenamiento que se dejó toda una noche trabajando. Lamentablemente, no aprendió nada en los 15M de steps que tuvo de entrenamiento. Se paró a mano al no ver mejoras. Curiosamente, es el entrenamiento que más ha durado hasta ahora sin crashear.

Tabla 15. Entrenamiento 10

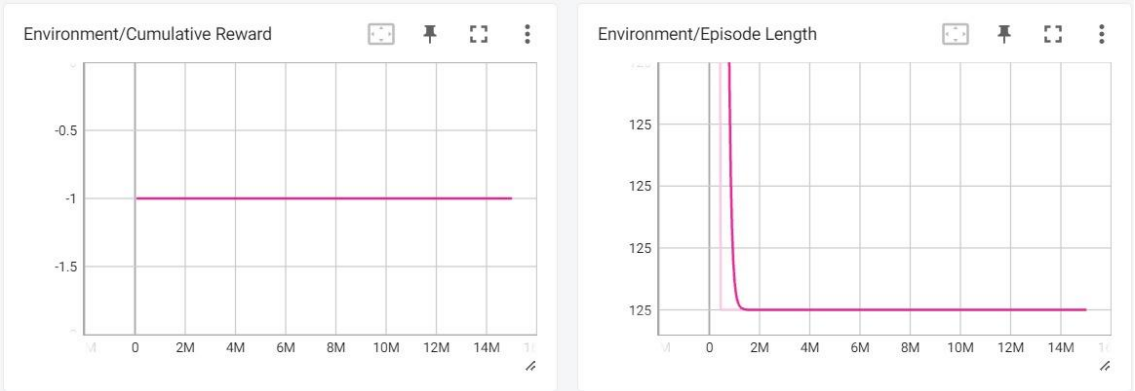

Figura 69. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 10

Entrenamiento		**11**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	29
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Durante 2M de steps se mantuvo en los 125 steps. Después, saltó directamente a los 200 steps al intentar explorar otras zonas. Se mantuvo así, prácticamente, el resto del entrenamiento
	Conclusiones	Este fue el segundo entrenamiento que llegó a los 15M de steps. Se paró manualmente al no ver mejora

Tabla 16. Entrenamiento 11

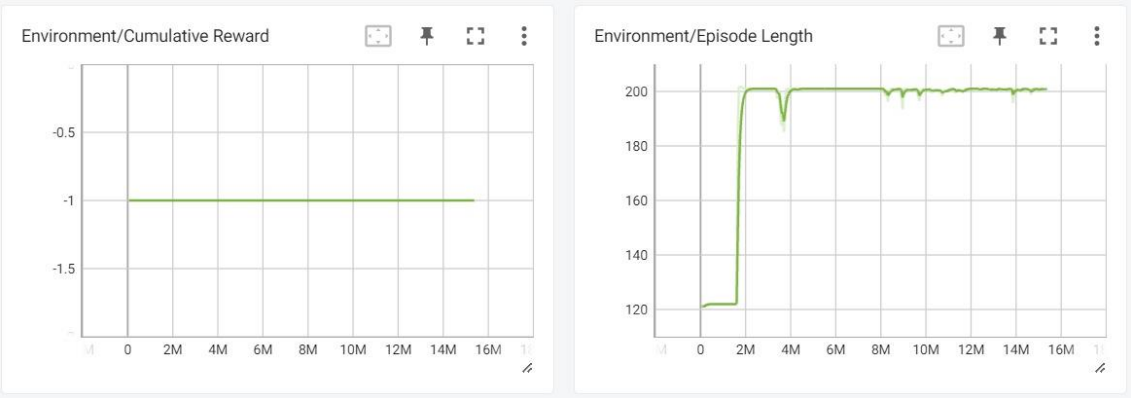

Figura 70. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 11

Entrenamiento		**12**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	29
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Tiene una tendencia creciente pero nada más que mencionar. El entrenamiento fue muy corto
	Conclusiones	Este entrenamiento se paró a mano a los 200.000 steps. Durante esta fase se estaba retocando parte de la programación, con lo que este fue un entrenamiento para probar las capacidades de la nueva programación. Al ver fallos de comportamiento, se paró manualmente y se revisó de nuevo la programación.

Tabla 17. Entrenamiento 12

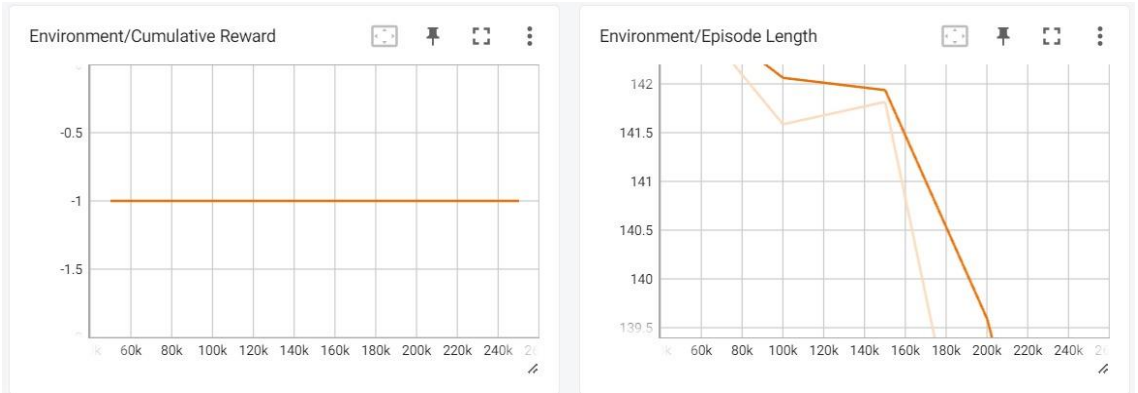

Figura 71. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 12

Entrenamiento		**13**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	29
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Se mantuvo, prácticamente todo el entrenamiento sobre los 320 steps
	Conclusiones	No aprendió nada en casi 4M de steps con lo que se retocó la programación una vez más

Tabla 18. Entrenamiento 13

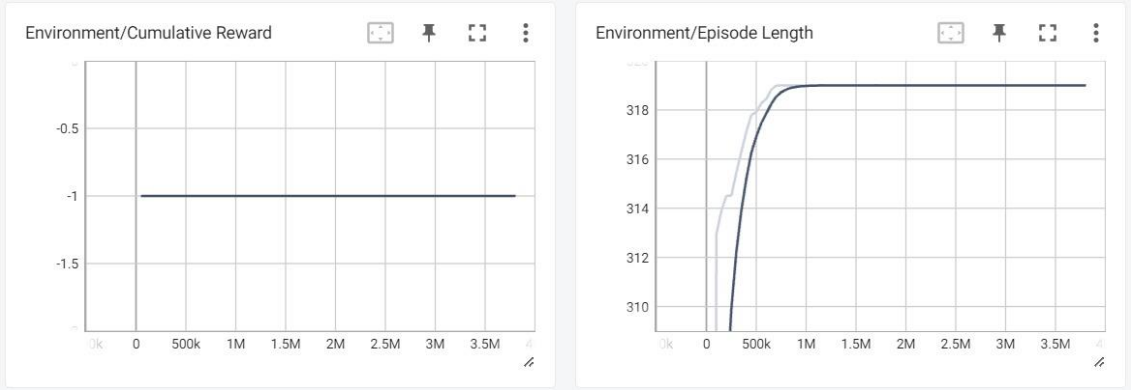

Figura 72. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 13

Entrenamiento		**14**
Configuración	*Brazo Robótico*	UR10ev1
	Nº de articulaciones	5
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	29
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se puede apreciar en la gráfica que el Agente realizó la primera tarea de la política sobre los 6M de steps. Exploró durante otros 2M y volvió a realizar esa tarea durante poco más de 1M de steps
	Duración de los episodios	Esta gráfica de duración es más acorde a la esperada en un entrenamiento satisfactorio. Tiene subidas y bajadas pero una tendencia creciente hasta el step 9M. Ahí se desploma a casi 0
	Conclusiones	Este parecía ser el primer entrenamiento que daba sus frutos. El Agente estaba realizando su tarea con una frecuencia nunca antes vista y, durante la última fase del entrenamiento, parecía hacerla de forma constante. Lamentablemente, el programa crasheó a los 9M de steps. Se retomó el entrenamiento donde se había quedado pero, como se vé en la gráfica, desde los steps 9M a 12M revirtió a un estado inicial de entrenamiento. Aquí es cuando se empezó a sospechar y se investigó sobre la posibilidad de que los crasheos reiniciaran el entrenamiento.

Tabla 19. Entrenamiento 14

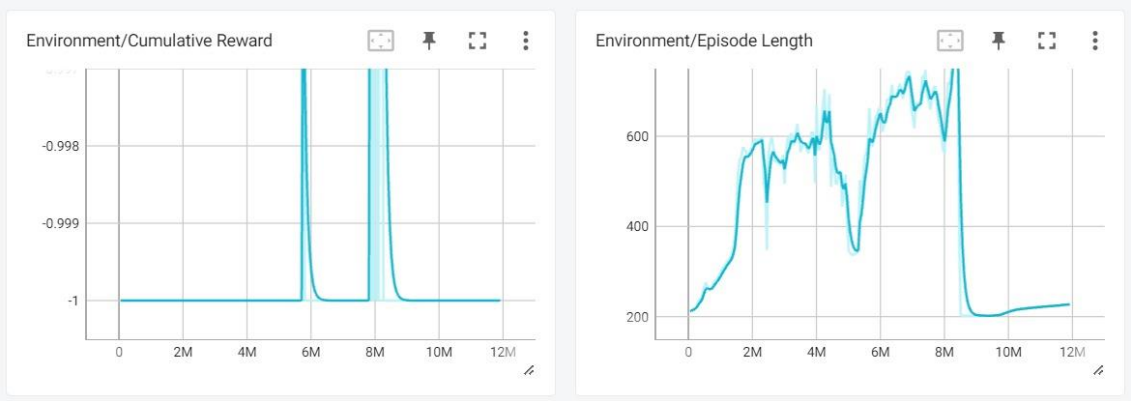

Figura 73. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 14

Entrenamiento		**30**
Configuración	*Brazo Robótico*	UR10ev3
	Nº de articulaciones	6
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	30
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Combinado
Observaciones	*Recompensa acumulada*	Se puede apreciar en la gráfica que, entre los steps 1,5M y 2,5M el Agente estaba realizando la tarea de forma consistente. No vuelve a hacer su tarea en el resto del entrenamiento
	Duración de los episodios	Es muy difícil interpretar esta gráfica ilógica si no has experimentado de primera mano por qué ha ocurrido. Durante el primer millón de steps, los episodios estaban sobre los 700-800 steps en cuyo momento la gráfica vuelve a empezar de 0 en los steps de entrenamiento, creando un looping. Esto no significa que vuelve a comenzar desde 0 sino que continúa desde donde se había quedado. Simplemente se ha reseteado el contador de steps del entrenamiento. El resto del entrenamiento no tiene una duración de steps que merezca la pena mencionar
	Conclusiones	El looping en la gráfica de duración de episodio se debe a un crasheo del programa. Irónicamente, esto hizo que el Agente, al reiniciar el entrenamiento, encontrara la goal y realizase la tarea durante 1M de steps. Esto no es algo inconcebible ya que, una misma configuración de entrenamiento puede dar resultados diametralmente distintos al inicio de una sesión de entrenamiento

Tabla 20. Entrenamiento 30

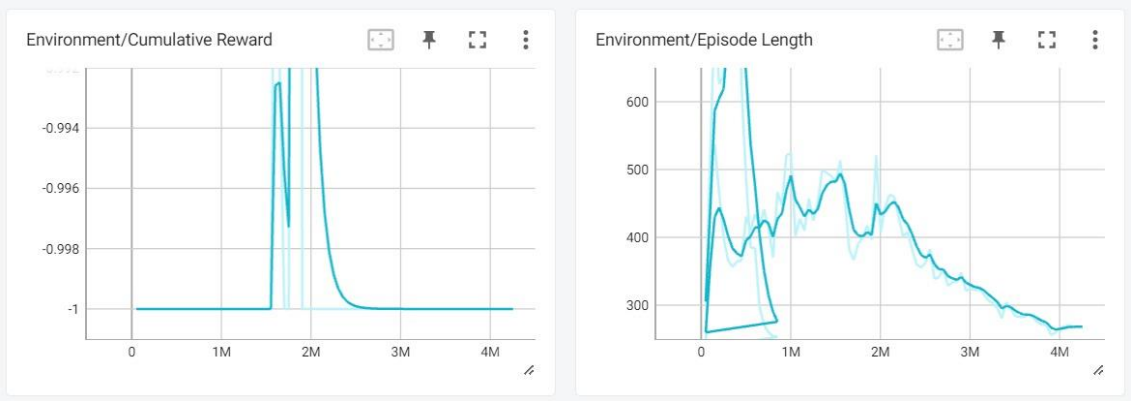

Figura 74. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 30

Entrenamiento		**31**
Configuración	*Brazo Robótico*	UR10ev3
	Nº de articulaciones	6
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	30
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Combinado
Observaciones	*Recompensa acumulada*	Es fácil apreciar, viendo la gráfica que, durante los primeros casi 4M de steps, el Agente estaba realizando la primera tarea de la política, batiendo récords de explotación hasta el momento. A partir de esos 4M su recompensa es -1 hasta el final
	Duración de los episodios	En concordancia con la gráfica de recompensa acumulada, se puede ver una tendencia creciente en los primeros 4M de steps, superando incluso los 1.000 steps de duración. Sin embargo, a los 4M parece reiniciar su entrenamiento y continúa con esa tendencia hasta el final
	Conclusiones	No es sorpresa que el step 4M se repita en ambas gráficas. Es cuando el programa crasheó. Este es, quizás, el ejemplo más claro para demostrar la teoría de que se reinicia el entrenamiento al crashear.

Tabla 21. Entrenamiento 31

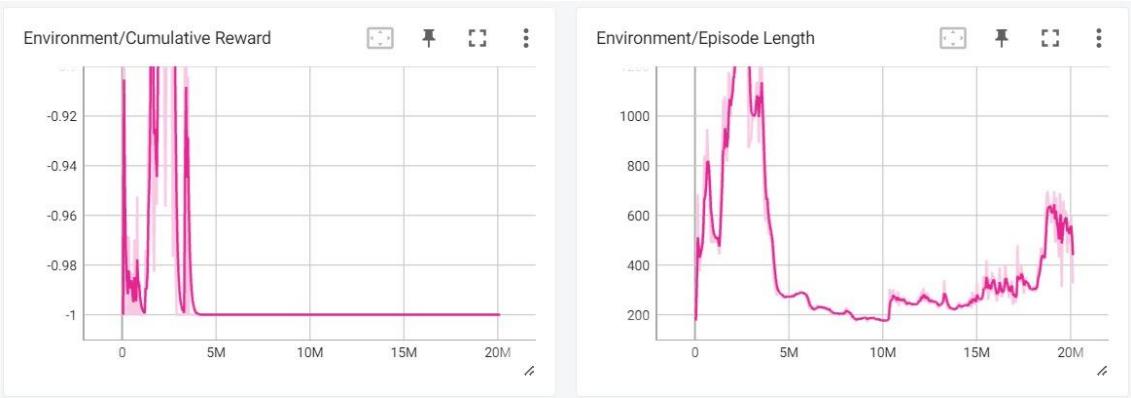

Figura 75. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 31

Entrenamiento		**32**
Configuración	*Brazo Robótico*	RSv1
	Nº de articulaciones	3
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	22
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	El Agente consigue realizar la primera tarea durante el primer 1,5M de steps. Después no volvió a realizarla
	Duración de los episodios	Aunque empezó con una duración de 70 steps, se mantuvo estable en 42 steps durante el resto del entrenamiento
	Conclusiones	Parecía un entrenamiento prometedor para ser el primero con este modelo. Sin embargo, se realizaron unos cambios necesarios en la programación y no se retomó este entrenamiento

Tabla 22. Entrenamiento 32

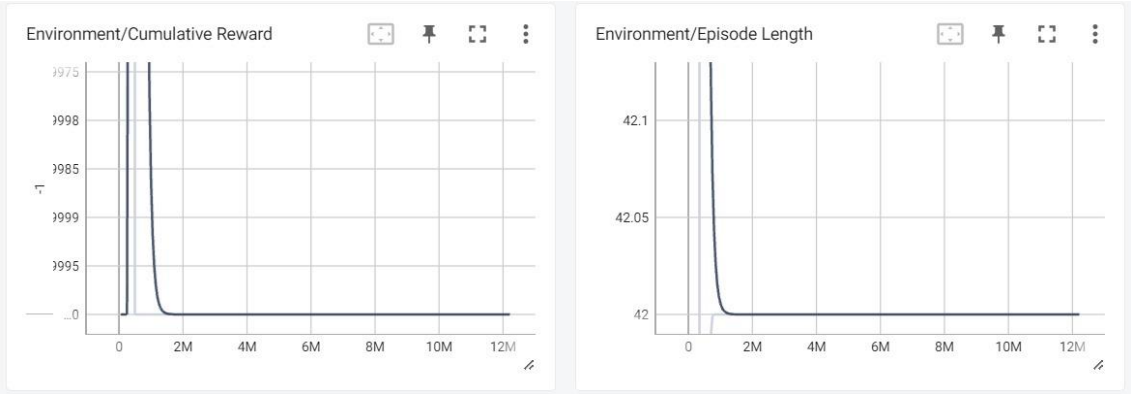

Figura 76. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 32

Entrenamiento		**33**
Configuración	*Brazo Robótico*	RSv1
	Nº de articulaciones	3
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	22
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Empieza con una clara tendencia creciente hasta los 4M de steps, donde parece estabilizarse alrededor de los 108 steps durante casi todo el resto del entrenamiento. Tiene un par de picos de menor duración debidos a un cambio en la exploración
	Conclusiones	Si bien no está realizando tarea alguna, este entrenamiento no es tan malo como lo muestran las gráficas. Éstas pueden engañar. Esta gráfica y las dos siguientes, demuestran que es más útil ver el entrenamiento con tus propios ojos que mirando sólo los resultados de un entrenamiento incompleto

Tabla 23. Entrenamiento 33

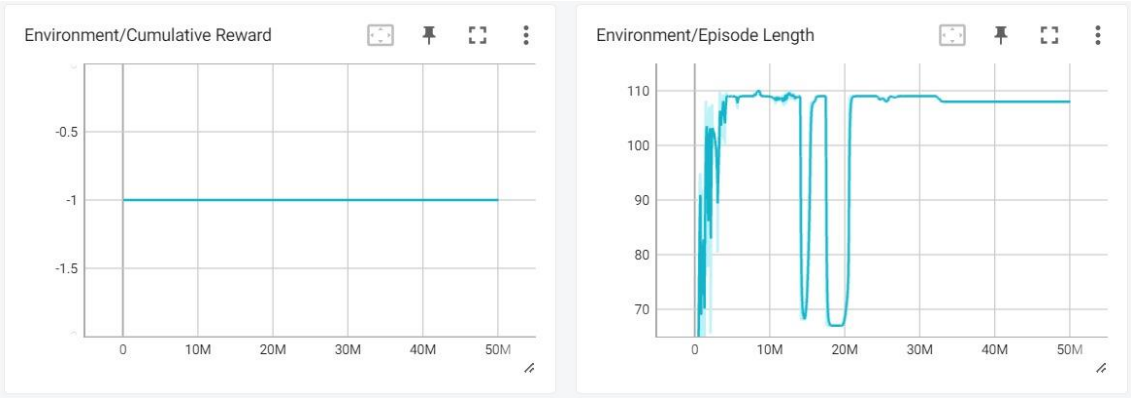

Figura 77. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 33

Entrenamiento		**34**
Configuración	*Brazo Robótico*	RSv1
	Nº de articulaciones	3
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	22
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	Salta inmediatamente de una duración de 104 steps a una estable de 67 steps durante los 50M de steps de entrenamiento
	Conclusiones	Este es el primer entrenamiento que se ha enlazado para superar el límite de 50M de steps. Aunque pueda parecer por las gráficas que el entrenamiento ha sido reiniciado, en realidad sólo ha cambiado su forma de explorar. Parece tener memoria del entrenamiento anterior (entrenamiento 33) ya que no se choca con los mismos objetos que en el anterior entrenamiento

Tabla 24. Entrenamiento 34

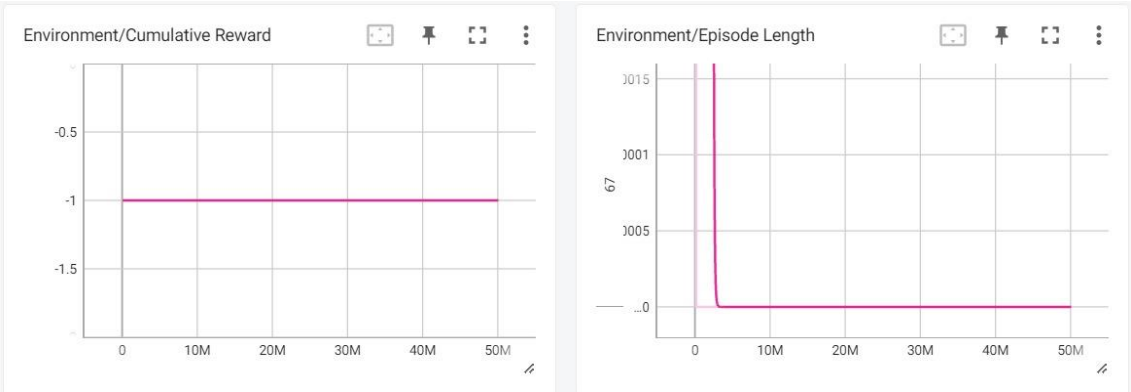

Figura 78. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 34

Entrenamiento		**35**
Configuración	*Brazo Robótico*	RSv1
	Nº de articulaciones	3
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	22
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	Se mantiene en -1 todo el entrenamiento con lo que no ha realizado ninguna tarea de la política
	Duración de los episodios	La duración de los episodios se mantiene totalmente estable en 67 steps durante todo el entrenamiento
	Conclusiones	Que se mantenga en 67 steps demuestra que ha conseguido enlazar con el anterior entrenamiento (entrenamiento 34). Si bien, gráficamente, no parece que haya habido ninguna mejora, puede ver con mis propios ojos cómo realizaba distintos ejercicios de exploración durante los 30M de steps de entrenamiento. El hecho de que todos esos ejercicios duren 67 steps, se debe probablemente, a que una de las acciones del Vector de Acciones estuviese realizando una exploración que resultara en colisión sin importar el resto de acciones

Tabla 25. Entrenamiento 35

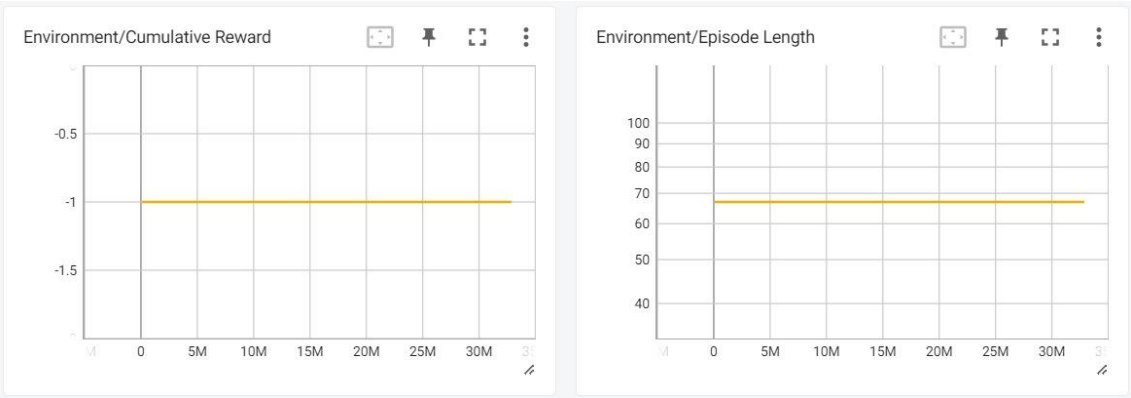

Figura 79. Gráfica de Recompensa acumulada y Gráfica de Duración de episodio del entrenamiento 35

9.3.2.1. Entrenamientos enlazados

En esta sección se hablará sobre las tablas y gráficas de los entrenamientos 33, 34 y 35 (véanse Tabla 23, Figura 77, Tabla 24, Figura 78, Tabla 25 y Figura 79).

Aunque de primeras puedan parecer unos resultados poco alentadores, es importante mencionar que, los entrenamientos con el modelo RSv1 fueron los únicos que no crashearon nunca debido a la simplicidad del entorno. Esto permitió llegar por primera vez a la cota máxima de 50M de steps.

Siendo que se quería seguir entrenando el modelo, ya que no había crasheado, se intentó enlazar el entrenamiento "terminado" con uno nuevo vacío, intentando que este segundo entrenamiento continuase donde lo había dejado el primero. Esto tuvo resultados positivos y, por tanto, los entrenamientos 33, 34 y 35 deben verse como un solo entrenamiento. Por eso las gráficas se pueden juntar una detrás de otra, conservando su continuidad.

Es cierto que en este largo entrenamiento de 130M de steps, el Agente no consiguió realizar ninguna tarea de la Política, pero debemos recordar que cada sesión de entrenamiento puede dar diferentes resultados al principio, aunque tengan los mismos parámetros.

Durante el entrenamiento, se ha seguido de cerca, visualmente, la evolución del Agente y, sus esfuerzos de exploración conllevaron una evolución clarísima. Al principio se chocaba con el suelo casi inmediatamente, como se aprecia en la gráfica sobre la duración de los episodios en la Figura 77. Sin embargo, pronto empezó a comprender qué era el suelo y, que era algo a evitar. Fue el primer modelo en aprender tan rápido dónde estaba el suelo y las consecuencias de que las coordenadas de las distintas partes del brazo robótico se acercasen a las del suelo.

Esto fue un avance prometedor. En el resto del entrenamiento, como se puede ver en la Figura 78 y en la Figura 79, se ve cómo se estabiliza la duración de los episodios. Esto se debe a que, una vez descubierto el suelo, el agente empezó a explorar su propio "cuerpo", el brazo robótico.

Empezó a estirar y contraer las articulaciones del robot para inferir sus límites. Durante los últimos, aproximadamente, 125M de episodios, se pudo ver claramente (visualmente) cómo probaba diferentes posturas y movimientos. El hecho de que los episodios durasen todos lo mismo es, probablemente, a la articulación del codo. Se podía apreciar visualmente que el hombro había aprendido rápidamente a no rotar demasiado para no chocar contra el suelo. Esto hizo que explorase otras posibles rotaciones. La base seguía probando diferentes ángulos para explorar. Sin embargo, el codo decidió explorar qué pasaba si se contraía sobre la pieza "arm" del robot (equivaldría a nuestro húmero). Parece ser que, durante el resto del entrenamiento, se centró en explorar estas posibilidades con lo que, daba igual de qué manera se comportasen la base o el hombro, siempre habría una colisión mano con brazo a los 67 steps de episodio.

Esto no es, sin embargo, no es algo intrínsecamente malo. Dado más tiempo de entrenamiento, el Agente intentaría explorar otras opciones con el codo e inferiría las dimensiones de su propio cuerpo.

9.3.2.2. Entrenamientos perdidos

Como se habrá podido fijar el lector, en la sección 9.3.2. Tablas de entrenamiento, el número de los entrenamientos salta directamente del 14 al 30. Esto se debe a que las gráficas de los entrenamientos del 15 al 29, ambos incluidos, no han sido guardadas en Tensorboard, con lo que no están disponibles. Esto se debe, muy probablemente, a la naturaleza del modelo. El modelo UR10ev2 constaba de dos Agentes, uno que manejaba las partes del brazo y otro que manejaba las partes de la muñeca, como se ha descrito en la sección 5.5.2.2. Resultados del Entrenamiento por Refuerzo. Aunque Anaconda y MLAgents están preparados para realizar entrenamientos de doble Agente, parece ser que Tensorboard no captó los datos de ambos Agentes para plotear las gráficas.

No obstante, se presente abajo, en la Tabla 26, la plantilla que seguirían estos entrenamientos. Dado que nunca se cambió de configuración durante esos 15 entrenamientos, la configuración se mantendría igual.

Entrenamiento		**15-29**
Configuración	*Brazo Robótico*	UR10ev2
	Nº de articulaciones	5
	Escenario	2
	Número de obstáculos	0
	Posición del obstáculo	Nula
	Tamaño del Vector de Observaciones	21 Hombro / 24 Muñeca
	Nº máximo de steps	50.000.000
	Tipo de entrenamiento	Refuerzo
Observaciones	*Recompensa acumulada*	
	Duración de los episodios	
	Conclusiones	

Tabla 26. Plantilla para los entrenamientos del UR10ev2

Sin embargo, esta información no es una gran pérdida, ya que se puede describir en unas pocas líneas.

Los entrenamientos con doble Agente parecían una idea prometedora y, aún creo que se deberían explorar. A pesar de ello, no dieron ningún resultado tangible ya que, como se explicó en la memoria en la sección 5.5.2.2. Resultados del Entrenamiento por Refuerzo, no se pudo encontrar la manera de configurar archivos yaml para dos Agentes con lo que, los entrenamientos con hiperpárametros por defecto, no consiguieron llegar al punto de sofisticación que se habría deseado.

Aún con todo, los archivos y resultados físicos de los entrenamientos (los archivos de proyecto y los artefactos generados por ellos) siguen estando disponibles en el pen drive que acompaña a esta memoria. Los entrenamientos que no se han podido reflejar se encuentran en los proyectos con nombre "TFGRobotArmV4" y "TFGRobotArmV5".