



UNIVERSIDAD DE GRANADA

Escuela Internacional de Posgrado

Master Universitario en Ciencia de
Datos e Ingeniería de Computadores

Trabajo de Fin de Master

**Análisis y estudio en frameworks
de simulación sobre la generación
de datos en problemas robótica
industrial y su aplicación en tareas
de aprendizaje automático.**

Presentado por:

Hodei Zia López

Tutores:

Alberto Luis Fernández Hilario y Mikel Galar Idoate

13/09/2022

Resumen

Actualmente la robótica nos permite automatizar una serie de tareas que simplifican la labor humana. Esto ha hecho que la sociedad avance de una manera mucho más rápida de la esperada. En particular, estas máquinas especializadas se adhieren en base al aprendizaje automático procedente del campo de la inteligencia artificial.

Todo sistema robótico, debe adaptarse al contexto de tarea para el que es destinado. Para ello debe pasar por una fase de aprendizaje en donde se incorporen los patrones que le permitan realizar con éxito la acción o acciones objetivo. En esta fase, debemos seleccionar cuidadosamente los datos de entrada para incorporar la experiencia, el modelo específico que va a realizar el aprendizaje y cualquier restricción interna que sea necesaria para adaptar el comportamiento.

Con el objetivo de ahorrar tanto en costes como en tiempo, antes de realizar cualquier prueba en el mundo real, necesitaremos hacer uso de un entorno virtual con el que simular el problema y facilitar así la puesta en marcha del caso real.

El problema se genera por la cantidad de datos que podemos obtener hoy en día en un entorno simulado dentro del mundo real en donde no solo somos capaces de obtener distintas visualizaciones de la escena desde cualquier ángulo que se nos ocurra, sino que podemos extraerlas con cámaras cotidianas, cámaras capaces de mostrar la profundidad o cámaras capaces de separar el objeto del fondo, además del tipo de acción que puede realizar el robot para trazar la trayectoria, la cual puede ser descrita mediante la posición de las articulaciones, la fuerza que ejercen o su velocidad angular por ejemplo. El inmenso volumen de datos implica la necesidad de encontrar una combinación óptima para dar mayor información de modo que el robot tenga el mejor comportamiento posible.

Para ello, se van a estudiar dos de los simuladores más utilizados dentro de este área de estudio como son MuJoCo y CoppeliaSim, analizando sus ventajas, inconvenientes y funcionamiento. Finalmente, se va a fijar el modelo con una red neuronal estándar que nos permita evaluar cuales son los datos más adecuados para cualquier tipo de problema focalizándonos en la calidad del modelo de modo que podamos ver las similitudes entre el resultado y lo que pongamos en práctica.

Palabras clave

Robótica industrial, Generación de datos, Entorno de simulación, Aprendizaje automático, Meta-Learning, Multi-view Learning, Lifelong Learning.

Summary

Currently, robotics allows us to automate a series of tasks that simplify human labor. This has made society advance much faster than expected. These specialized machines adhere based on machine Learning coming from the field of artificial intelligence.

Any robotic system must adapt to the task context for which it is intended. To do so, it must go through a Learning phase where the patterns that allow it to successfully perform the target action(s) are incorporated. In this phase, we must carefully select the input data to incorporate the experience, the specific model that will perform the Learning and any internal constraints that are necessary to adapt the behavior.

In order to save both costs and time, before performing any real-world test, we need to make use of a virtual environment to simulate the problem and thus facilitate the implementation of the real case.

The problem is generated by the amount of data we can obtain in a simulated environment within the real world where we are not only able to obtain different visualizations of the scene from any angle we can think of, but we can extract them with everyday cameras, cameras able to show depth or cameras able to segregate the object from the background, in addition to the type of action the robot can perform to trace the trajectory, which can be described by the position of the joints, the force they exert or their angular velocity for example. The immense volume of data implies the need to find an optimal combination to give more information so that the robot has the best possible behavior.

For this purpose, two of the most widely used simulators in this area of study, MuJoCo and CoppeliaSim, will be studied, analyzing their advantages, disadvantages and operation. Finally, the model will be fixed with a standard neural network that will allow us to evaluate which are the most appropriate data for any type of problem, focusing on the quality of the model so that we can see the similarities between the result and what we put into practice.

Keywords

Industrial robotics, Data generation, Simulation environment, Machine Learning, Meta-Learning, Multi-view Learning, Lifelong Learning.

Índice

Resumen	3
Palabras clave.....	4
Summary	5
Keywords.....	6
Índice.....	7
Capítulo 1.....	9
Introducción	9
Capítulo 2.....	15
Marco teórico.....	15
2.1. Fundamentos.....	15
2.1.1. Meta-Learning.....	16
2.1.2. Transfer Learning.....	17
2.1.3. Multi-task Learning.....	18
2.1.4. Few-Shot y One-Shot Learning	19
2.1.5. Lifelong Learning.....	20
2.1.6. Multi-view Learning	20
2.1.7. Redes Neuronales residuales	21
2.2. Entornos de simulación utilizados.....	22
2.2.1. CoppeliaSim	23
2.2.2. MuJoCo.....	26
2.2.3. Comparativa entre ambos entornos de simulación para robótica industrial	27
2.3. Bibliotecas utilizadas	28
2.3.1. RLBench	29
2.3.2. Metaworld	33
2.3.3. Comparativa entre ambas bibliotecas.....	37
Capítulo 3.....	39

Marco metodológico.....	39
3.1. Fase 1: Generación de los datos.....	40
3.2. Fase 2: Entrenamiento del modelo.....	44
3.3. Fase 3: Simulación y validación del modelo.....	46
Capítulo 4.....	47
Caso de estudio.....	47
4.1. El robot	47
4.2. Casos de uso.....	48
4.3. Medidas utilizadas.....	50
Capítulo 5.....	52
Resultados y análisis experimental	52
5.1. Marco experimental.....	52
5.1.1. Resultados con MuJoCo-MetaWorld.....	54
5.1.1.1. Pruebas con un único tipo de imagen.....	54
5.1.1.2. Pruebas con pares de tipos de imágenes.....	56
5.1.1.3. Pruebas con tríos de tipos de imágenes.....	58
5.1.1.4. Pruebas con otras combinatorias de tipos de imágenes	59
5.1.2. Resultados con CoppeliaSim-RLBench	61
5.1.2.1. Pruebas con un único tipo de imagen.....	62
5.1.2.2. Pruebas con pares de tipos de imágenes.....	64
5.1.2.3. Pruebas con otras combinatorias de tipos de imágenes	65
5.1.3. Comparativa para la tarea de coger una bola	66
5.2. Otras tareas.....	67
5.2.1. Tarea: Pulsar un botón.....	69
5.2.2. Tarea: Abrir una ventana.....	72
5.3. Análisis final de los resultados.....	76
Capítulo 6.....	80
Conclusiones y líneas futuras.....	80
Bibliografía	84

Capítulo 1

Introducción

La robótica industrial es una rama de la ingeniería que pretende realizar múltiples procesos industriales tales como la manipulación de objetos haciendo uso de robots con el objetivo de completar diversas tareas en cadena de forma automática de modo que necesitemos la mínima supervisión humana posible [1].

Partiendo de esta definición, podemos encontrarnos tanto con tareas que no dependan del entorno y que por tanto, se realicen aplicando los mismos movimientos sobre el robot como con otras en las que tanto el entorno como los objetos u obstáculos que se encuentren en él sean cambiantes, por lo que necesitemos hacer que nuestro robot adapte sus movimientos, decisiones o incluso fuerza en base a lo que se encuentre en cada momento.

Desde la década de los sesenta, junto con lo que la revolución industrial supuso, cada vez son más las empresas que deciden automatizar tareas o fases de su producción que antes estaban a cargo de un ser humano [2]. Es una evidencia que los robots están revolucionando tanto los pequeños como los grandes sectores realizando cada vez más y más tareas las cuales hasta hace muy poco tiempo era impensable que pudieran estar bajo la supervisión de una maquina tanto por su complejidad como por lo que esto podría suponer en un futuro. En los últimos años, incluso las interacciones más humanas han comenzado un proceso de mecanizado el cual supondrá una reestructuración de los espacios de trabajo y empleos actuales [3].

Esto último ha traído muchas negativas al cambio en ciertos sectores debido al temor de que este proceso de automatización acabe por eliminar millones de puestos de trabajo no cualificados. Sin embargo, la realidad es que se crearán más puestos de trabajo de los que se destruirán, según comentaron los expertos en el World Economic Forum de Suiza en el año 2018, en dónde predijo que para el año 2022, de los 75 millones de empleos a nivel mundial que los robots

reemplazarían, se crearían más de 133 millones de puestos de trabajo nuevos [4]. Esto es debido a que la inteligencia artificial y los robots mejorarán enormemente la productividad de los trabajos existentes provocando la creación de nuevos empleos los cuales en su mayoría supondrían de una mayor cualificación.

Todo esto es consecuencia de que cada vez son más los recursos que se invierten en automatizar las líneas de producción industriales [5] o crear nuevas máquinas autómatas capaces de adaptarse a tareas muy concretas y difíciles de realizar. En el mismo foro económico mundial que comentábamos antes, se mencionaba también que el gasto mundial en robótica alcanzaría los 242.000 millones de dólares solamente durante 2019, por lo que podemos hacernos una idea del ascenso que esto supondrá durante esta nueva década. El cambio ha llegado para quedarse y en cuanto antes lo asimilemos y nos adaptemos a él, mejor nos irá.

Como ya hemos mencionado previamente, son muchas las tareas que un robot puede realizar hoy en día, las cuales van desde poner una tuerca [6] en una cinta de producción industrial hasta conducir por ti de modo que puedas realizar cualquier viaje sin necesidad de prestar atención a la carretera [7] o mantener un local de comida rápida en donde todas las tareas, desde cocinar hasta servir la comida estén a cargo de un robot [8]. Pese a esta versatilidad, por muy distintas que parezcan estas tareas, todas necesitan de un entrenamiento previo con el que poder aprender a alcanzar su objetivo independientemente de la dificultad que éste suponga, ya que del mismo modo que ningún ser humano nace sabiendo, un robot tampoco.

Para poder entrenar un robot, debemos de mostrarle como realizar dicha tarea o en su defecto, como realizar otra tarea que tenga cierto grado de similitud. Esto se hace simulando correctamente la tarea en cuestión y almacenando los datos que se generen en cada instante de tiempo para que podamos utilizarlos posteriormente a la hora de entrenar la red con la que generaremos el modelo que sea el que devuelva las acciones que debe realizar el robot.

Sin embargo, llegados a este punto, nos surge la problemática de saber cuáles son los datos que debemos de usar para este tipo de problemas, ya que en el mundo real, de así quererlo, podríamos llenar la escena de cámaras que guarden la información visual de cada ángulo tanto con tres canales de color (en lo sucesivo RGB) como con cámaras de otro tipo como podrían ser de profundidad o capaces

de segregar los objetos del fondo. Es por este motivo por el que utilizaremos el Multi-view Learning para tratar de trabajar con distintos tipos de imágenes sobre un mismo instante de tiempo. Además de estas representaciones visuales de la escena, necesitaríamos averiguar qué clase de acción es la más adecuada, ya que podemos hacer que el robot trace una trayectoria hacia su objetivo en base a la posición de sus articulaciones, fuerza que estas ejercen o velocidad angular a la que se mueven.

Pese a poder aplicar todos los datos que quisiéramos en el entrenamiento para tratar de dotar de toda la información posible a nuestro modelo, sabemos cómo esto no sería lo más óptimo, ya que estaríamos maximizando los costes temporales, monetarios y computacionales, y esto es lo que queremos evitar a toda costa.

Es aquí donde surge la necesidad de averiguar cuál es la combinatoria de los datos que podamos extraer de la escena que aporte una mayor información a nuestro modelo evitando añadir ruido o información innecesaria, todo esto tratando de minimizar la cantidad de información que utilicemos teniéndola en cuenta a la vez que los resultados que estos aporten.

Como una de nuestras finalidades es hacer que el robot aprenda a realizar una trayectoria gracias a la información de diferentes estados, entraremos además de en el Multi-view Learning en el Lifelong Learning, con el que trataremos de aprender a realizar las acciones del estado actual gracias a los estados previos que hayan sido predichos por nuestro modelo. Estos dos conceptos que mencionamos y que vamos a poner a prueba a lo largo de nuestro proyecto se encuentran dentro de los paradigmas que más están creciendo dentro del mundo del aprendizaje automático gracias a toda la información que son capaces de aportar al modelo en la fase de entrenamiento.

Así pues, en nuestro problema, además de tratar de minimizar la cantidad de información que utilicemos, también querremos minimizar los costes que supongan la obtención de herramientas, maquinarias, patentes o incluso la electricidad y mantenimiento que puedan requerir dichos aparatos. Es por ello por lo que antes de realizar cualquier clase de tarea en el mundo real se crea su representación dentro de un mundo virtual gracias a los distintos entornos de simulación y bibliotecas asociadas a ellos que disponemos.

Este es un problema que no se ha llegado a plantear en ninguna ocasión, ya que son muchos los entornos de simulación a los que podemos acceder para realizar pequeñas simulaciones de tareas guiadas por robot los cuales presentan distintos tipos de acciones o de configuraciones para las que podrían ser las mismas tareas. Es por este motivo por el que dependiendo del entorno que dispongamos o que se nos proporcione, seremos capaces de obtener unos resultados u otros, todo esto en base a la información que nos permita extraer cada entorno de la escena.

Nosotros vamos a trabajar con dos de los entornos de simulación y sus consecuentes bibliotecas más utilizadas para este tipo de problemas en donde debamos predecir la trayectoria de un robot tras haber dotado a nuestra red de la información que veamos conveniente sobre dicha tarea. Estos entornos se tratan de CoppeliaSim [9] y MuJoCo [10], los cuales utilizaremos mediante bibliotecas diseñadas específicamente para ellos como son RLBench [11] y MetaWorld [12].

Si bien es cierto que la amplia mayoría de problemas relacionados con la robótica se encuentran dentro del campo del aprendizaje por refuerzo, nuestro problema no lo está debido a que no utilizamos ninguna función de recompensa que guie al robot hacia su objetivo. En lugar de ello, trataremos de alcanzar nuestras metas mediante el aprendizaje por imitación, de modo que debamos de enseñar previamente a nuestro sistema robótico las acciones que debe de realizar.

Al necesitar de una supervisión humana para enseñarle a nuestro robot como ha de realizar las trayectorias y para evaluar el funcionamiento de nuestro modelo, podemos decir que nos encontramos dentro del campo del aprendizaje supervisado. Es por este motivo por el que necesitaremos de distintas métricas que nos indiquen tanto el error que tienen las salidas de nuestro modelo con respecto a los valores reales como el acierto que se obtiene una vez apliquemos dicho modelo dentro de un entorno virtual. Es aquí donde entran nuestras dos medidas principales: el error cuadrático medio [13] (en lo sucesivo RMSE) y el porcentaje de acierto durante la simulación.

En definitiva, con este proyecto trataremos de sacar una conclusión sobre cual ha de ser la configuración de datos con la que debamos tratar cualquier tipo de problema que requiera de la predicción de trayectorias para la realización de todo tipo de tareas llevadas a cabo por un robot con el objetivo de minimizar los costes necesarios para ello, tanto computacionales dentro del entorno virtual como monetarios y temporales fuera de él.

De acuerdo a todo lo anterior, el objetivo principal será obtener una combinatoria general de datos con el que se caractericen distintos casos de uso en robótica industrial pudiéndola coger como punto de partida a la hora de trabajar en un proyecto de este tipo. Con la finalidad de alcanzarlo, desglosamos nuestro objetivo principal entre los siguientes subobjetivos:

1. Estudiar y analizar en profundidad tanto los distintos estudios y proyectos realizados en este ámbito como los diferentes entornos de simulación utilizados, dejando claras tanto sus diferencias como sus similitudes.
2. Generar datos para un conjunto de tareas con cada uno de los entornos de simulación que utilicemos para generar los distintos modelos en la fase de entrenamiento.
3. Analizar las mejores configuraciones posibles de datos en busca de un patrón capaz de generar una generalización para futuros problemas de modo que sepamos de donde partir inicialmente sea cual sea el problema a evaluar. Para esto trataremos de encontrar las configuraciones que maximicen tanto el RMSE como el porcentaje de acierto en simulación, teniendo en cuenta la trayectoria realizada por cada prueba.
4. Simular dentro de los entornos de simulación virtuales los modelos generados y visualizar los resultados obtenidos mediante diferentes aproximaciones y estadísticos.

Para lograr los objetivos aquí presentados vamos a estructurar la memoria de la siguiente manera:

- **Capítulo 1:** Introducción a nuestro problema en donde expondremos la motivación y objetivos que seguimos.
- **Capítulo 2:** Marco teórico en donde detallemos todos los fundamentos teóricos necesarios que creemos oportunos para la comprensión y seguimiento de nuestro problema. Además, se detallarán tanto los entornos como las bibliotecas que vamos a utilizar para nuestro proyecto.
- **Capítulo 3:** Apartado en donde expondremos como queremos resolver este problema y como lo hemos conseguido. Este capítulo estará dividido en tres fases distintas en base al desarrollo que hayamos realizado en cada una de ellas.
- **Capítulo 4:** En este capítulo expondremos nuestros casos de estudio así como los datos que hayamos obtenido para su realización y sus posibles diferencias.
- **Capítulo 5:** Marco experimental en donde hablaremos acerca de los resultados obtenidos e intentaremos alcanzar el objetivo principal que nos hemos marcado para este proyecto.
- **Capítulo 6:** Último apartado con el que concluiremos la memoria. En él intentaremos apoyarnos del capítulo 5 para llegar a una buena conclusión. Además de esto, analizaremos las posibles líneas futuras que podría abordar nuestro proyecto.

Capítulo 2

Marco teórico

En este capítulo vamos a exponer toda la parte teórica que vemos necesaria comentar tras realizar una investigación documental sobre distintos trabajos centrados en temas con cierto grado de similitud. De igual forma, una vez finalizado dicho apartado, pasaremos a hablar sobre los diferentes entornos de simulación y bibliotecas que hemos utilizado.

2.1. Fundamentos

Dada la complejidad de nuestro proyecto y de los problemas que trataremos de abordar, hemos creído conveniente dedicar un apartado para cada uno de los conceptos principales de los que hablaremos a partir de ahora de modo que facilitemos la comprensión de las decisiones tomadas más adelante. De esta manera, pasaremos a hablar de los siguientes fundamentos:

- Meta-Learning
- Transfer Learning
- Multi-task Learning
- Few-Shot y One-Shot Learning
- Lifelong Learning
- Multi-view Learning
- Redes Neuronales residuales

Con estos fundamentos, hemos tratado de agregarle cierto grado de complejidad a nuestro proyecto incorporando paradigmas emergentes del aprendizaje automático con el objetivo de trabajar con conceptos e ideas novedosas dentro de este área de conocimiento.

2.1.1. Meta-Learning

El meta-Learning o meta-aprendizaje es un paradigma del aprendizaje automático (en lo sucesivo ML) que surge con la idea de aplicar distintos algoritmos de ML a los datos obtenidos por medio de otros algoritmos de ML [14]. En otras palabras, se trata de algoritmos de ML que aprenden entre sí aprendiendo del aprendizaje que tenemos disponible por medio de otros métodos.

Este proceso de combinar las predicciones hechas por distintos algoritmos se llama apilamiento, y se refiere a tomar como entrada los datos de salida del algoritmo o método de aprendizaje anterior.

Su objetivo es conseguir realizar un buen apilamiento de predicciones creando un campo de aprendizaje conjunto en donde todos los algoritmos utilizados trabajen conjuntamente intentando realizar mejores predicciones o predicciones generalistas para un conjunto de tareas distintas.

Esto último que hemos comentado nos llevaría a la posibilidad de aprender de un conjunto de tareas en predicción relacionadas entre sí pero no iguales, lo cual nos llevaría hasta los siguientes fundamentos aquí propuestos. En la Figura 1 podemos ver un ejemplo sencillo del funcionamiento del meta-Learning:

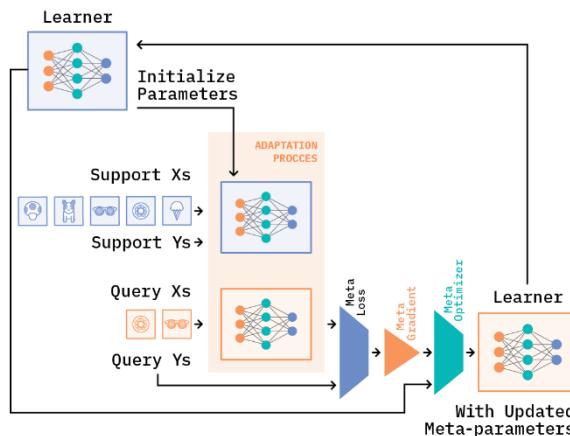


Figura 1. Funcionamiento del proceso de Meta-Learning

Vemos como en la Figura 1 existen dos redes. La primera inicializa los parámetros y aprende los pesos para su modelo. Después, utiliza su salida como datos de entrada de la siguiente red con el objetivo de optimizar los resultados gracias a la colaboración de ambas.

2.1.2. Transfer Learning

El transfer Learning o aprendizaje por transferencia se refiere al conjunto de métodos que permiten transferir conocimientos adquiridos tras la resolución de problemas para resolver otros problemas [15].

Este subcampo del ML ha sido de gran importancia con el auge del Deep Learning (en lo sucesivo DL), ya que los modelos utilizados en DL necesitan de muchos recursos y tiempo computacional. Es por esto por lo que tomando como base un modelo preentrenado, el transfer Learning permite desarrollar rápidamente modelos válidos para otro tipo de problemas. En la Figura 2 podemos ver el funcionamiento del transfer Learning:

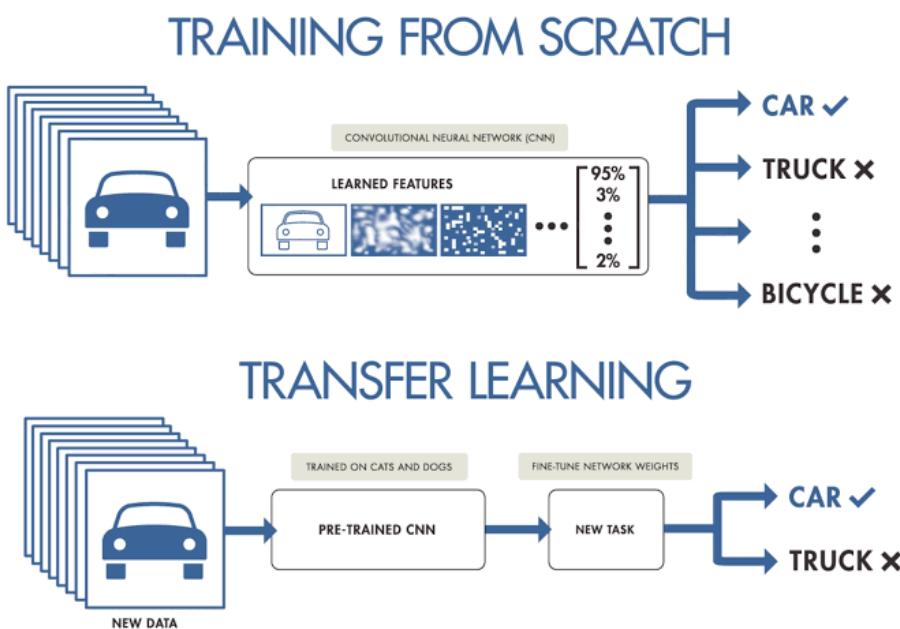


Figura 2. Funcionamiento del proceso de Transfer Learning

En la Figura 2 observamos como tras haber entrenado un modelo capaz de realizar una tarea específica, podemos aplicar dicho modelo a otro conjunto de datos y a una tarea no muy distinta para no tener que entrenar un nuevo modelo desde cero.

2.1.3. Multi-task Learning

El Multi-task Learning o aprendizaje multitarea es del mismo modo que el transfer Learning, un subcampo del ML en el que se resuelven varias tareas de aprendizaje al mismo tiempo mientras que se analizan tanto los puntos en común o patrones como las diferencias que pueden tener las tareas del conjunto [16].

El procedimiento es sencillo. Se pretende resolver una tarea principal tras haberla aprendido conjuntamente con otras tareas relacionadas. En este proceso se produce una transferencia de conocimiento entre las distintas tareas.

Sin embargo, dentro del mundo real surge el problema de encontrar tareas que estén tan relacionadas o incluso una vez después de haberlas encontrado, es muy complejo saber el grado de importancia que van a suponer a la hora de predecir la tarea principal, ya que del mismo modo que pueden obtener información ventajosa, pueden obtener información que perjudique al procedimiento o a las trayectorias de nuestra tarea principal.

Dentro del Multi-task Learning, cualquiera de las tareas que se encuentre dentro del conjunto de tareas de aprendizaje puede ser la tarea principal a predecir, ya que el objetivo es poder aplicar el modelo generado a cualquiera de estas.

Es importante destacar que existen dos paradigmas denominados como Few-Shot y One-Shot Learning los cuales se encuentran asociados al Multi-task Learning por partir de la misma idea. Sin embargo, presentan ciertas diferencias que vamos a pasar a exponer a continuación.

2.1.4. Few-Shot y One-Shot Learning

Podríamos decir que estos dos paradigmas del ML son un paso más allá del Multi-task Learning, ya que parten de la misma idea de entrenar un modelo con un amplio conjunto de tareas.

Sin embargo, es en este punto donde comienzan sus diferencias respecto al Multi-task Learning, y que en el caso del One-Shot Learning [17], la tarea principal que antes utilizábamos también dentro del conjunto de tareas de entrenamiento desaparece del conjunto, siendo a la hora de validarla la primera vez que el modelo la ve.

En cuanto al Few-Shot Learning [18], es una especie de mezcla entre ambas, ya que mantenemos la tarea principal a predecir dentro del conjunto de tareas de entrenamiento con la diferencia de tenerla muy pocas veces, de modo que el porcentaje que tiene dicha tarea dentro del conjunto respecto al de cualquier otra tarea sea minúsculo.

El objetivo de ambos paradigmas es el de generar un modelo entrenado con diversas tareas el cual sea capaz de ser aplicado para una nueva tarea o una tarea vista muy pocas veces.

Si nos fijamos bien, podemos darnos cuenta de que este concepto ya lo hemos tratado previamente al hablar del transfer Learning, y es que en ambos casos se aplican las mismas técnicas con la diferencia de los casos de estudio que se proponen.

Estos dos conceptos se exponen aquí pese a no tener mayor relevancia en la implementación del proyecto porque se consideran importantes dentro de este contexto al ser uno de los pasos que habría que dar como posibles líneas futuras para obtener una generalización no solo de los datos de entradas a utilizar sino del tipo de tareas que podemos llegar a utilizar dentro de un mismo aprendizaje.

2.1.5. Lifelong Learning

El Lifelong Learning se trata de un paradigma de ML que aprende continuamente acumulando conocimiento de estados anteriores adaptándolo y usándolo para ayudarse a predecir los resultados en el futuro del problema en cuestión [19].

El paradigma principal del ML aprende de manera aislada al tiempo, y pese a ser muy exitoso para muchos casos de uso como ya sabemos, necesita de un amplio conjunto de ejemplos de entrenamiento además de que estos ejemplos estén bien definidos y contengan la información necesaria para nuestro problema.

Sin embargo, con el Lifelong Learning, nos acercamos al comportamiento humano, ya que no solo predecimos acciones mediante datos actuales, sino que tenemos en cuenta como han sido esos datos en estados anteriores para buscar mejores patrones y prever comportamientos extraños en los datos que teniendo solo en cuenta datos del estado actual no podríamos hacer.

2.1.6. Multi-view Learning

El Multi-view Learning o aprendizaje multivista es otro de los paradigmas del ML en donde tratamos con problemas de imágenes.

En estos problemas, somos capaces de poseer por cada ejemplo imágenes o vistas del entorno desde diferentes ángulos, haciendo así que a la hora de realizar el aprendizaje, la red sea capaz de tener en cuenta como se ve el entorno desde distintos ángulos aportando una mayor dimensionalidad sobre los datos de entrada de nuestro modelo [20].

Esta metodología de aprendizaje se utiliza cuando queremos mostrar una representación más tridimensional de los datos mediante imágenes en dos dimensiones representadas como matrices.

2.1.7. Redes Neuronales residuales

Las redes neuronales residuales o ResNet [21] fueron propuestas por primera vez por el equipo de Microsoft Research en el año 2016. Estas redes son redes neuronales artificiales que utilizan atajos o conexiones de salto para moverse sobre distintas capas sin tener por qué pasar directamente por la siguiente.

El motivo por el que se saltan estas capas es para evitar el problema del desvanecimiento del gradiente, en donde por cada pasada, como el gradiente va retropropagándose entre las capas anteriores haciéndose más y más pequeño, llega un momento en el que pasa a ser completamente insignificante y por tanto, desvanece.

Con este salto, la red se simplifica eliminando así estas complicaciones al usar muy pocas capas durante la primera etapa del entrenamiento. Esto hace que al haber tan pocas capas por las que extenderse, el aprendizaje se acelere mucho minimizando la desaparición de los gradientes. Tras este inicio, la red vuelve a poner el resto de las capas a medida que va a prendiendo del espacio de características.

A medida que el entrenamiento llega al final y las capas se expanden, comienzan a aprender cosas más rápidamente. Con este tipo de redes conseguimos entrenar miles de capas con un rendimiento extremadamente bueno. A continuación se muestra en la Figura 3 la arquitectura de una de las ResNets con menor complejidad y la que nosotros utilizaremos como base, la ResNet18 [22]:

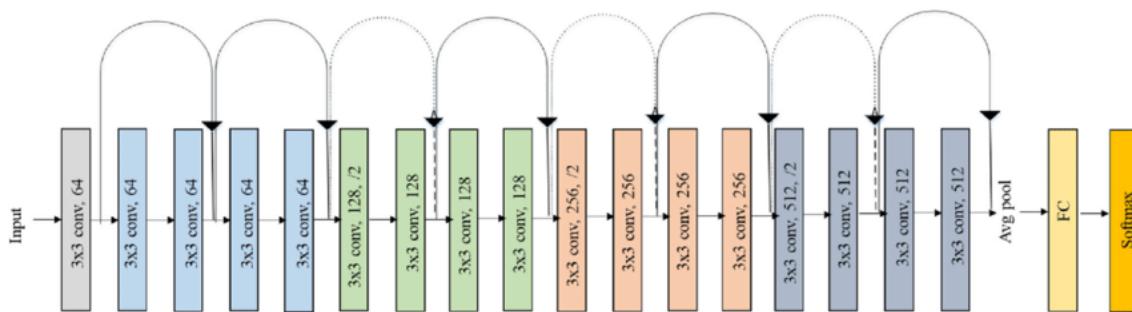


Figura 3. Arquitectura de una ResNet18

2.2. Entornos de simulación utilizados

Como ya hemos ido mencionado a lo largo del inicio de este documento, un entorno de simulación es una aplicación software que permite al usuario trabajar tanto con una amplia variedad de robots como de tareas y entornos virtuales totalmente personalizables a los gustos y necesidades de cada uno. Al ser un entorno virtual, no requiere de la disposición de ninguna clase de herramienta física, haciendo así que ahorremos tiempo y dinero en la compra y adjudicación de la puesta en marcha de todo lo necesario para simular cualquier tipo de tarea.

El objetivo principal de cualquier entorno de simulación es el de emular y validar tareas dentro de un mundo virtual antes de hacerlo en el mundo físico para así, tal y como hemos comentado antes, evitar sobrecostes y ahorrar tiempo, ya que es mucho más rápido y menos costoso simular una tarea dentro de un software que podemos modificar, iniciar o detener a nuestro antojo antes que hacerlo sobre un robot físico el cual muy posiblemente requiera de unos cuidados y puesta en marcha muy específicos. De esta manera, podemos probar distintas configuraciones de un modo muy sencillo para posteriormente evaluar cual es la que finalmente debemos llevar a cabo sobre nuestro robot o entorno físico.

Para ello, dentro de casi cualquier entorno, tenemos la libertad para crear y modificar materiales, objetos e incluso comportamientos de una manera rápida que en el mundo real no seríamos capaces de realizar tan fácilmente.

Para la realización de este proyecto, y con el objetivo principal de evitar un sesgo en los datos por culpa de las características específicas de cada entorno, hemos decidido utilizar dos de los entornos de simulación de datos más conocidos en problemas de robótica industrial: MuJoCo y CoppeliaSim. Cada uno de estos entornos de simulación es capaz de utilizar distintas bibliotecas que son las encargadas de especificar al sistema como generar, simular y trabajar con sus propias herramientas. Nosotros hemos seleccionado dos de las más utilizadas en este tipo de proyectos: RLBench y Metaworld.

2.2.1. CoppeliaSim

Este entorno de simulación y desarrollo integrado basado en una arquitectura de control distribuida, antes conocido como V-Rep, es uno de los más utilizados para la simulación de cualquier tipo de tarea dentro de la comunidad robótica. Fue desarrollado por Coppelia Robotics en Marzo de 2010 y a día de hoy, según el servicio de búsqueda de patentes *The Lens* [23], cuenta con un total de 1.282.860 citas distribuidas entre 73.775 patentes, por lo que podemos hacernos a la idea del prestigio que tiene.

Se utiliza entre muchas otras cosas para el monitoreo remoto de tareas industriales, la generación de prototipos y escenarios virtuales, el desarrollo de algoritmos, la educación relacionada con robots o la simulación de sistemas de automatización de fábricas. Es con esta última utilidad con la que vamos a trabajar en este proyecto.

Dispone de varios módulos de cálculo general como son los de cinemática inversa, detección de colisiones, cálculo de la distancia mínima o planificación de rutas entre otros con los cuales podemos ser capaces de desarrollar de una manera muy sencilla y visual algoritmos para la realización de tareas que sean capaces de ser integrados posteriormente en el mundo real.

Además de poseer todos estos módulos de cálculo propios ya integrados, tiene también un gran mecanismo de extensión que le permite recibir complementos y aplicaciones de cliente con los cuales ampliar su funcionalidad. Un ejemplo de uno de uno de estos complementos es RL Bench el cual pasaremos a explicar más adelante.

Parte de su prestigio viene dado por su amplia capa de personalización con la cual podemos ser capaces de adaptar casi cualquier escenario del mundo real dentro del mundo virtual. Para ayudarnos con ello, tenemos toda clase de objetos como pueden ser muebles, material para construir cualquier tipo de estancias desde cero, actuadores, vehículos...

A parte de todos los aquí mencionados, disponemos también de un total de cincuenta modelos distintos de robots los cuales se diferencian entre 24 robots no móviles como son los brazos robóticos y 26 robots móviles de entre los que tenemos desde robots con ruedas hasta robots humanoides o robots basados en animales e insectos.

Como ya hemos comentado previamente, está basado en una arquitectura de control distribuida. Es por este motivo por el que podemos programar escenarios de múltiples modos:

- **Scripts embebidos**
- **API remota de cliente**
- **Add-ones**
- **Plugins**
- **Nodo ROS/ROS2**
- **Nodo ZeroMQ**

Cada cual presenta ventajas y desventajas respecto al resto de métodos tal y como vamos a mostrar a continuación en la Figura 4:

	Embedded script	Add-on / sandbox script	Plugin	Remote API client	ROS / ROS2 node	ZeroMQ node
Control entity is external (i.e. can be located on a robot, different machine, etc.)	No	No	No	Yes	Yes	Yes
Difficulty to implement	Easiest	Easiest	Relatively easy	Easy	Relatively easy	Easy
Supported programming language	Lua, Python	Lua, Python	C/C++	C/C++, Python, Java, JavaScript, Matlab, Octave	Any ¹	Any
Code execution speed	Relativ. fast ²	Relativ. fast ²	Fast	Depends on programming language	Depends on programming language	Depends on programming language
Communication lag	None ³	None ³	None	Yes	Yes	Yes
Control entity can be fully contained in a scene or model, and is highly portable	Yes	No	No	No	No	No
Control entity relies on	CoppeliaSim	CoppeliaSim	CoppeliaSim	Sockets, ZeroMQ or WebSockets	ROS / ROS2 framework	ZeroMQ
Stepped operation ⁴	Yes, inherent	Yes, inherent	Yes, inherent	Yes	Yes	Yes
Non-stepped operation ⁴	Yes, via threads	Yes, via threads	No (threads available, but API access forbidden)	Yes	Yes	Yes

Figura 4. Diferencias entre los seis métodos de programación

De entre todos estos métodos de programación, nosotros vamos a trabajar en Python con scripts embebidos provenientes de la biblioteca RLBench y modificaciones que haremos sobre ellos, de modo que no necesitemos ni siquiera abrir su interfaz gráfica, ya que iremos extrayendo la información frame a frame sobre la escena en todo momento tal y como explicaremos más adelante.

En cuanto a sus demás características, podemos destacar que este entorno de simulación es tanto multiplataforma (Windows, Linux, MacOS) como multilenguaje (C/C++, Python, Java, Lua, Matlab y Octave). Además, soporta un total de cuatro motores de física (Bullet Physics, Open Dynamics Engine, Vortex Studio y Newton Dynamics) que podemos alternar ajustándonos a las necesidades que tengamos en todo momento.

En definitiva, prácticamente podemos llevar a cabo cualquier tarea que nos propongamos con él gracias a la cantidad de módulos de cálculo, capas de personalización y características que posee. Es por este motivo por el que a día de hoy es uno de los entornos de simulación más utilizados en todo el mundo.

2.2.2. MuJoCo

Este entorno de simulación y desarrollo integrado es uno de los que más está creciendo desde que fuera creado en 2018. Fue desarrollado por Roboti LLC inicialmente pese a que en 2021 fue adquirido por DeepMind y pasó a ser de código abierto.

Según el servicio de búsqueda de patentes *The Lens*, ha recibido 42 citas distribuidas en un total de 15 patentes, las cuales pese a no ser un gran número, nos muestran cómo es un entorno que se tiene en cuenta a la hora de realizar cualquier tipo de investigación dentro de este campo. Por otro lado, tenemos que destacar como la baja cantidad de citas ha podido producirse por ser hasta hace unos pocos meses una herramienta de pago.

MuJoCo surgió con el propósito principal de servir como una herramienta capaz de facilitar y ayudar en todo lo posible en el desarrollo e investigación centrado en la robótica, biomecánica, tareas de animación gráfica y otros áreas que demandaran de una simulación rápida y precisa por medio de estructuras articuladas que interactúen con el entorno.

Su utilidad es muy similar al resto de entornos de simulación que podemos encontrarnos. Sin embargo, es algo más restrictiva haciendo que no podamos desarrollar prototipos o entornos visualmente tan bien detallados como con otros entornos, ya que inicialmente, al estar pensado en ser una herramienta privada, no se tuvieron en cuenta ciertos aspectos sobre su interfaz, lo cual ha acabado derivando en que tengamos que hacer uso de complementos externos para poder añadir funcionalidades e implementar mejoras nuevas que lo hagan ponerse a la par de los grandes entornos del mercado.

Otro de los motivos por el que MuJoCo no es visualmente tan avanzado al resto de entornos es porque está centrado en la parte de código, más en concreto en la parte de desarrollo de algoritmos de ML, lo que ha hecho que se descuide un poco más el apartado visual para favorecer en temas de velocidad y diversidad al apartado de complementos y módulos matemáticos.

Hablando sobre su capa de personalización, somos capaces de utilizar hasta un total de 16 robots distintos siendo en su mayoría robots no móviles como lo son los brazos robóticos u otro tipo de pinzas. En cuanto al resto de objetos con los que el robot puede interactuar en el entorno, disponemos de muy pocos.

Al estar basado en una arquitectura distribuida, es capaz de recibir órdenes de distintas maneras soportando scripts embebidos, plugins de Unity y API remota de cliente. Para nuestro proyecto hemos decidido optar por el uso de scripts embebidos, ya que son con los que más facilidades nos hemos encontrado a la hora de analizar los distintos métodos y probar con algunos de ellos.

Entre las características más importantes de MuJoCo, destaca el que sea un entorno de simulación multiplataforma (Windows, Linux, MacOS) y multilenguaje (Swift, Java, Python, Julia) con un motor de físicas propio desarrollado por Roboti LLC.

2.2.3. Comparativa entre ambos entornos de simulación para robótica industrial

Si nos fijamos en todo lo descrito para ambos entornos de simulación, podemos ver como CoppeliaSim se encuentra a priori por encima a MuJoCo al tener no solo más funcionalidades y utilidades sino una capa de personalización más cuidada con la que emular virtualmente entornos del mundo real con el máximo añadido de detalles posible. Sin embargo, al tener una visualización tan avanzada, es ligeramente más costosa computacionalmente hablando a la hora de realizar el mismo tipo de pruebas.

Centrándonos en la capa de personalización, CoppeliaSim viene por defecto con una infinidad de objetos interactivos con el entorno y con una gran variedad de robots de todo tipo. MuJoCo, en cambio, trae muy pocos objetos y clases de robots. Sin embargo, en MuJoCo somos capaces de generar desde cero o importar objetos al entorno con mucha mayor facilidad, cosa que en CoppeliaSim es bastante difícil al ser un entorno tan restrictivo.

En cuanto al resto de características y funcionalidades, no existe una gran diferencia entre ambos. Hemos podido ver como los dos entornos pretenden ser de utilidad para los mismos casos de uso.

A priori, al ser CoppeliaSim más completo y utilizado que MuJoCo, podría parecer que con el podemos obtener mejores resultados. Pese a ello, vamos a trabajar con los dos para poder sacar unas comparativas más reales sobre el terreno aplicando los mismos casos de uso de nuestro proyecto.

2.3. Bibliotecas utilizadas

Como ya hemos ido mencionando a lo largo del apartado anterior, los entornos de simulación necesitaban de una biblioteca que les aporte todas las opciones necesarias para poder trabajar en la simulación de cualquier tipo de tarea posible en el ámbito de la robótica industrial.

No todos los entornos disponen del mismo número de bibliotecas o de las mismas adaptadas para distintos entornos, por lo que antes de escoger que dos íbamos a utilizar, hemos decidido realizar un análisis previo sobre cuáles eran las bibliotecas con las que se trabaja normalmente al utilizar estos dos entornos de simulación.

Estas bibliotecas están adaptadas al simulador en cuestión y vienen con un conjunto de escenas, objetos, texturas, materiales y funciones que hacen que podamos partir de una base sólida de dónde empezar a realizar casi cualquier tarea que se nos ocurra en el ámbito de la robótica.

Nos hemos encontrado con que en el caso de CoppeliaSim, la biblioteca que más citas recibía era RL Bench, por lo que hemos optado por escogerla. En cuanto al caso de MuJoCo, hemos tenido más dudas acerca de cuál escoger debido a que eran dos las bibliotecas que se vienen utilizando con este entorno desde que se volvió de código abierto. Tenemos tanto OpenAI Gym [24] como Metaworld. Tras evaluarlo, hemos decidido utilizar Metaworld por encima de OpenAI Gym por encontrarla más intuitiva y sencilla a la hora tanto de importarla como de trabajar con ella.

2.3.1. RLBench

RLBench, también conocida como Robot Learning Benchmark and Learning Environment, es una amplia biblioteca y entorno de aprendizaje diseñada para facilitar el trabajo de los investigadores que trabajen en el campo de la robótica dentro de las áreas del aprendizaje por refuerzo, aprendizaje por imitación, aprendizaje multitarea, visión por computador y en particular el aprendizaje por medio de pocas pruebas o Few-Shot Learning.

Fue desarrollada por Stephen James, Zicong Ma, David Rovick y Andrew J.Davison en septiembre de 2019 y ha sido desde entonces una de las bibliotecas y entornos de aprendizaje más utilizadas dentro de este área de trabajo.

Esta biblioteca engloba un total de 100 tareas diferentes realizables con la ayuda de brazos robóticos articulados. Dentro de estas tareas se encuentran algunas como recoger una pelota, pulsar un botón o incluso abrir y cerrar una ventana, tal y como se puede observar en la Figura 5:



Figura 5. Algunas de las tareas disponibles con RLBench

Podemos apreciar como la estructura de cada escena es la misma. Poseemos el mismo robot, misma mesa, y lo único que cambia es la tarea a realizar con distintos objetos además del código interno de cada una de las tareas que es el cual guía al robot a la hora de realizar pequeñas demostraciones las cuales utilizaremos más adelante para entrenar nuestro modelo.

Por otro lado, con el objetivo de dotar de una mayor complejidad a cada tarea, RLBench es capaz de generar variaciones sobre una misma tarea haciendo que cada prueba o episodio sea completamente distinto al anterior. Esto lo hace no solo inicializando los objetos en distintos lugares cada vez que llamamos a la ejecución de la escena, sino que también modificando los colores de los objetos que se encuentran en ella.

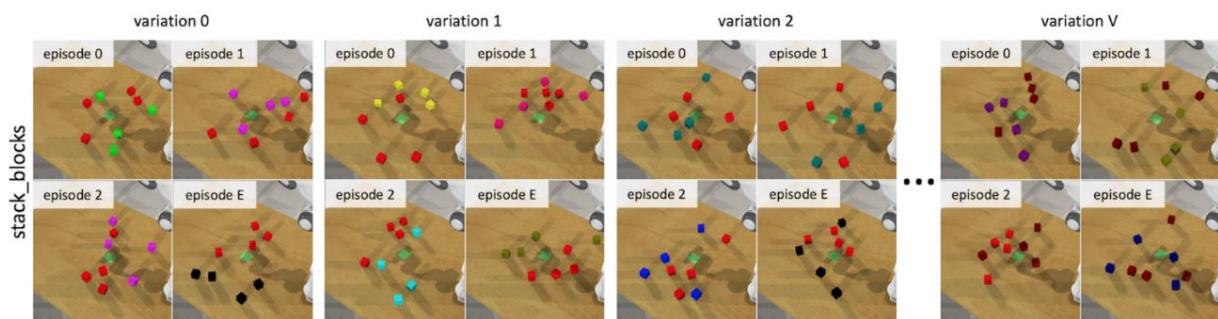


Figura 6. Ejemplo de las variaciones por prueba en cada tarea

En la Figura 6 podemos ver cómo variaría cada episodio para una misma tarea de apilado de bloques haciendo que se modifiquen no solo los colores de los objetos secundarios de la escena sino los colores del objeto que se debe tomar como objetivo, además de modificar la tarea añadiendo nuevos niveles de complejidad como podría ser apilar más bloques o apilar bloques alternando sus colores. Todo esto dependerá de la tarea en cuestión, ya que no todas disponen del mismo abanico de variaciones disponible.

El objetivo de esta biblioteca es el de dotar de las herramientas necesarias a CoppeliaSim para poder simular tareas con las que trabajar dentro del campo del aprendizaje automático y de este modo conseguir obtener y validar buenos modelos con los que trabajar en el mundo real.

No solo posee un amplio abanico de tareas, sino que estas son grabadas por un total de cinco cámaras colocadas en distintos ángulos de modo que podamos obtener toda la información posible de cada uno de los movimientos del brazo robótico. Es de estas cinco cámaras de las que extraeremos toda la información visual de la escena.

Además de estas cámaras que son capaces de mostrarnos la escena en los tres canales de color habituales, RLBench puede generar también imágenes de profundidad y máscaras desde los mismos ángulos que los mostrado en la Figura 6 tal y como podemos apreciar a continuación en la Figura 7:

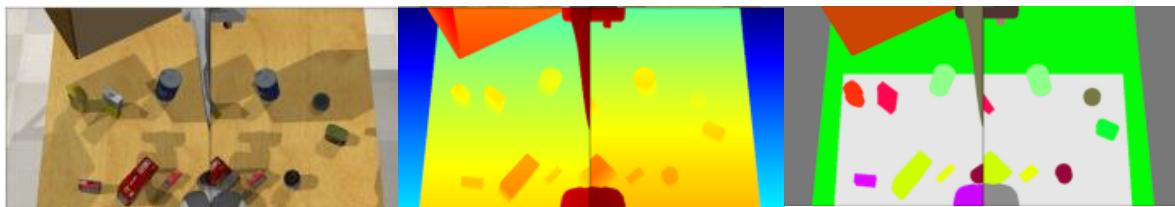


Figura 7. Imágenes en RGB, profundidad y mascara

Con el objetivo de poder realizar las pruebas lo más comparables posibles, hemos optado por usar únicamente las imágenes obtenidas mediante cámaras RGB, ya que no todos los simuladores son capaces de sacar imágenes tanto en profundidad como en forma de mascara generando una segmentación de los objetos de la escena. Por otro lado, hemos tomado esta decisión también pensando en que en el mundo real es mucho más sencillo y menos costoso obtener imágenes de tres canales de color que de los otros dos modos.

La biblioteca no solo es capaz de ofrecernos la información de forma visual por cada instante para observar y evaluar las tareas, sino que proporcionan al usuario una variedad de acciones con las que complementar sus experimentos.

Estas acciones vienen dadas como un vector de siete elementos, uno por cada una de las articulaciones, entre las cuales tenemos:

- **Velocidad absoluta:** Se trata de la velocidad con la que se mueve cada articulación en un instante de tiempo. Viene dada en radianes/segundos.
- **Velocidad relativa:** Se trata de la diferencia de velocidad entre instantes dada por cada articulación. Viene dada en radianes/segundos.
- **Posición absoluta:** Se trata de la posición en la que se encuentra cada articulación en el espacio.
- **Posición relativa:** Se trata de la diferencia de posiciones en cada instante en la que se encuentra cada articulación en el espacio.
- **End-effector pose absoluto:** Se trata únicamente de la posición de la pinza en el espacio de coordenadas xyz. Con esta acción, es como si tiráramos de la pinza haciendo que el resto de articulaciones se movieran únicamente por dicha fuerza.
- **End-effector pose relativo:** Se trata únicamente de la diferencia de la posición de la pinza en el espacio de coordenadas xyz por cada instante de tiempo.
- **Fuerza absoluta:** Se trata de la fuerza realizada por cada una de las articulaciones.
- **Fuerza relativa:** Se trata de la diferencia de fuerza realizada por cada una de las articulaciones.

En próximas actualizaciones de RLBench se comenta sobre la posibilidad de incluir nuevas métricas de acciones tales como los ángulos absolutos y relativos, los cuales serán medidos en radianes.

En definitiva, RLBench es una biblioteca muy completa capaz de dotar de una gran gama de funciones al programador con las que extraer la mayor cantidad de datos a la escena al igual que automatizar acciones o procesos que no se encontraban implementados inicialmente. Además de todo esto, facilita mucho el trabajo gracias a la cantidad de tareas y escenas que trae ya preparadas.

2.3.2. Metaworld

Metaworld es una biblioteca de simulación de tareas centrada en el aprendizaje por refuerzo y en el aprendizaje multitarea.

Fue desarrollada por Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn y Sergey Levine en el año 2019 y ha estado en constante desarrollo desde entonces implementando desde nuevas tareas a funciones capaces de facilitar el trabajo.

Esta biblioteca se centra única y exclusivamente en los dos campos ya mencionados que se pueden ver en la Figura 8:

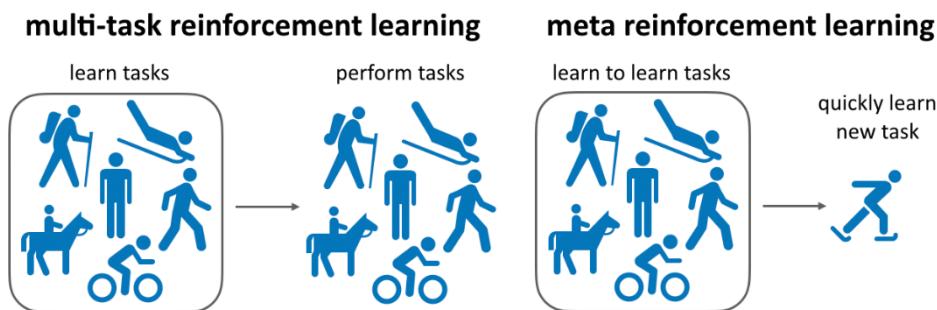


Figura 8. Explicación de lo que pretende realizar esta biblioteca

En la Figura 8, podemos ver de una manera más visual cuales son los dos objetivos principales de esta biblioteca, el aprendizaje multitarea y el meta-aprendizaje, los cuales hemos explicado previamente en el apartado de fundamentación de este mismo capítulo, por lo que no vamos a volver a entrar en detalle en ellos.

Con este fin, se han diseñado un total de 50 tareas con las que se han generado seis métodos distintos de evaluar los distintos algoritmos de aprendizaje que el usuario genere. Son estos los seis métodos con los que hemos decidido probar el simulador previamente antes de empezar a trabajar más en profundidad con él. Cada uno de estos métodos de evaluación posee una dificultad y objetivo distinto al resto tal y como vamos a ver a continuación.

- **Meta-Learning 1 (ML1):** El ML1 es el método más sencillo de evaluación de meta-aprendizaje. Se escoge una sola tarea y se entrena realizando muchas pruebas con ella en donde se modifican las inicializaciones de los objetos y de sus metas. Finalmente, se realiza una validación con otro conjunto de pruebas en donde tengamos objetos inicializados en posiciones nunca antes vistas en el conjunto de entrenamiento.



Figura 9. Ejemplo del funcionamiento del ML1

- **Multi-Task 1 (MT1):** El MT1 es bastante parecido al ML1, ya que tan solo usa una única tarea para realizar el aprendizaje. Sin embargo, realiza la evaluación sin necesidad de realizar ninguna validación previa.

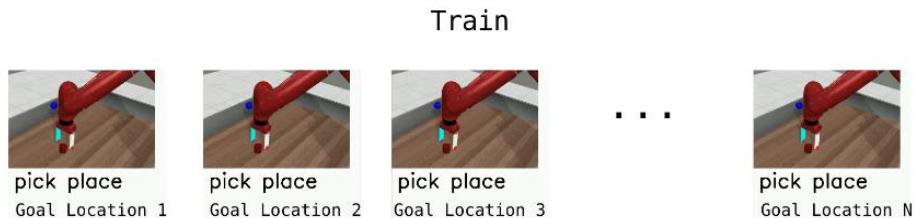


Figura 10. Ejemplo del funcionamiento del MT1

- **Meta-Learning 10 (ML10):** El ML10 es un método de meta-aprendizaje más complejo que el ML1 en donde entrenamos con un conjunto de 10 tareas distintas. Finalmente, realizamos la validación sobre 5 tareas no vistas en la fase de entrenamiento.

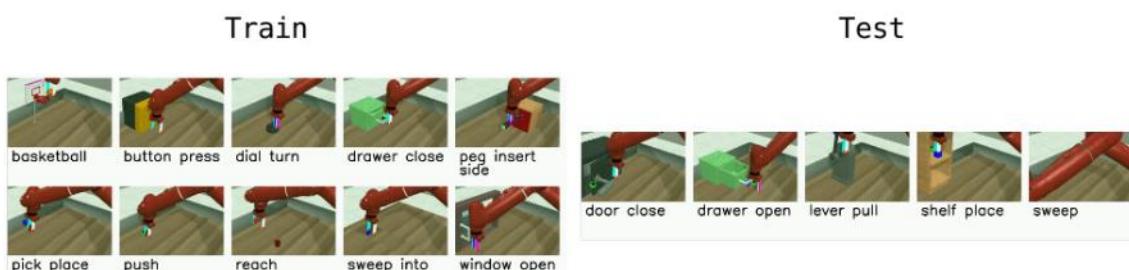


Figura 11. Ejemplo del funcionamiento del ML10

- **Multi-Task 10 (MT10):** El MT10 se asemeja al MT1 con la diferencia de que en lugar de aprender de una tarea y replicarla sin necesidad de validación, aprende de 10 tareas distintas para tratar de reproducirlas posteriormente.

Train

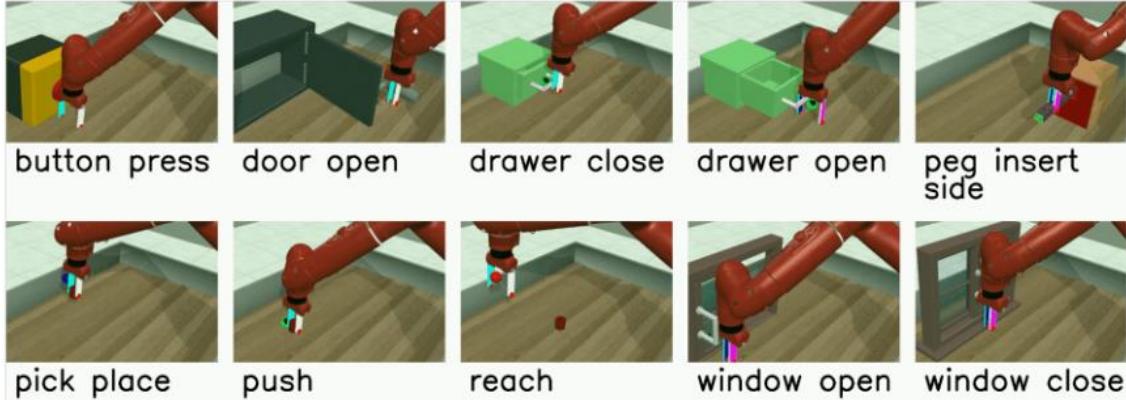


Figura 12. Ejemplo del funcionamiento del MT10

- **Meta-Learning 45 (ML45):** El ML45 es el modelo más completo de meta-aprendizaje en donde se entrena un conjunto de 45 tareas dejando de lado las 5 restantes para validación.

Train

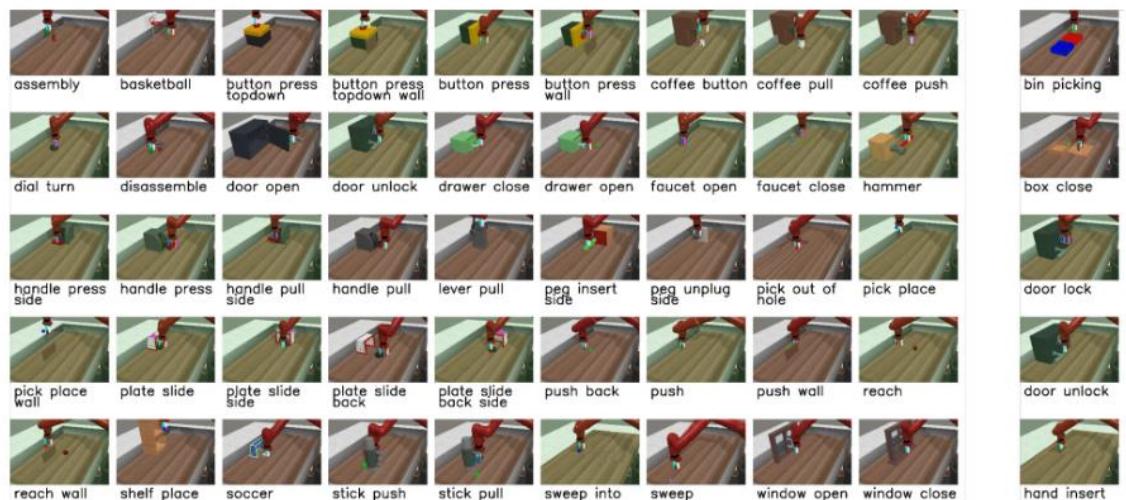


Figura 13. Ejemplo del funcionamiento del ML45

- **Multi-Task 50 (MT50):** El MT50 es el modelo más completo de multitareas, ya que usa el 100% de estas sin dejar ninguna para la validación.

Train

*Figura 14. Ejemplo del funcionamiento del MT50*

Las cincuenta tareas que se pueden observar son controladas por el robot con acciones que son pasadas instante a instante a nuestro robot.

En Metaworld, como acción, disponemos únicamente de la representación vectorial de las posiciones relativas en el espacio de coordenadas XYZ junto con el estado de la pinza, el cual puede estar representado por un cero en caso de estar abierta o por un valor decimal en caso de estar en proceso de cerrarse.

Además de a este vector de acciones, podemos acceder también a un conjunto de informaciones que nos describen la escena cada instante de tiempo dándonos datos como podrían ser la recompensa actual, el valor continuo del estado de la pinza o las posiciones absolutas en los ejes de coordenadas XYZ de la pinza por ejemplo.

Al estar realizando un proyecto en donde no queremos entrar dentro de un sistema de recompensas, vamos a considerar el uso de las dos últimas informaciones mencionadas descartando la recompensa por cada instante de tiempo.

Las imágenes utilizadas por Metaworld, tal y como podemos apreciar en la Figura 16, se encuentran situadas en distintas zonas del entorno haciendo así que podamos obtener el máximo de información posible en todo momento de una manera muy visual.

2.3.3. Comparativa entre ambas bibliotecas

Si bien es cierto que existen diferencias notables entre los datos extraíbles de las dos bibliotecas, podemos llegar a encontrar una pequeña similitud en algunos casos.

Esto podemos observarlo en los datos de tipo imagen, dado que existen tipos que son muy similares en ambas bibliotecas tal y como podemos apreciar en la Figura 15 y 16 como son las que muestran tanto la primera persona de la pinza como el entorno desde arriba y desde ambos lados, todo esto lo veremos más adelante en la fase de implementación.

En cuanto a las acciones, son completamente diferentes los datos que podemos extraer de cada una de ellas empezando por la cantidad de tipos de representaciones vectoriales de acciones que podemos conseguir, en donde RLBench es capaz de proveernos no solo de uno sino de varios tipos de acciones diferentes relacionados además de con la posición con otras fuerzas como son la velocidad o la fuerza ejercida. Por otro lado, en el caso de MetaWorld, la representación vectorial que disponemos posee solamente tres valores con los que mover el robot más uno para el estado de la pinza, mientras que en el caso de RLBench todas las acciones disponen de un total de siete valores, uno por cada grado de libertad del robot. Esta diferencia en el modo de manejar el robot nos aportara un plus a la hora de buscar la mejor generalización de datos posible, ya que existe una amplia diferencia entre las acciones extraíbles de las dos bibliotecas que hemos utilizado.

En definitiva, vemos como ambas bibliotecas, pese a tener grandes diferencias por el modo en el que están hechas y por el entorno de simulación en el que son utilizadas, sirven para lo mismo, pues podemos obtener datos e información muy parecida, aunque no igual, de tareas completamente iguales para entrenar nuestro modelo.

Además, es muy interesante ver la capa de personalización que poseen ambas, haciendo que pese a tener funciones y tareas ya programadas y funcionales, seamos capaces de modificarlas o crearlas desde cero a nuestro antojo. Esto último sí que es verdad que se nota mucho más en Metaworld, en donde no solo tenemos los scripts mejor explicados sino que al utilizar un entorno de simulación menos complejo, dependemos de menos variables para realizar nuestras modificaciones.

A continuación vamos a mostrar una tabla comparativa entre los datos que somos capaces de obtener de ambas bibliotecas:

Tabla 1. Cantidad de datos extraíble para cada biblioteca

	RLBench	MetaWorld
Cantidad de Imágenes	5	6
Cantidad de Acciones	4 (8 si diferenciamos relativas con absolutas)	1

En la Tabla 1 podemos apreciar como a simple vista, antes de comenzar a probar el funcionamiento de los datos, RLBench parece ser mucho más completa. Este será uno de nuestros objetivos también, analizar de entre las dos bibliotecas cuál de las dos, junto con que entorno de simulación, es capaz de trabajar mejor para esta problemática.

Queda para la parte del estudio experimental observar si estas diferencias a nivel teórico van a implicar a posteriori diferencias entre los modelos aprendidos así como en sus comportamientos y rendimiento.

Capítulo 3

Marco metodológico

A continuación, una vez hemos hablado tanto sobre en qué consiste nuestro proyecto como sobre los distintos entornos y bibliotecas y fundamentos que vamos a utilizar, vamos a pasar a explicar como lo hemos hecho.

Para ello, dividiremos este capítulo entre las tres fases diferenciables de nuestro proyecto: Generación de datos, entrenamiento del modelo y simulación del modelo.

Cada una de estas fases ha sido realizada tanto para distintos entornos de simulación como para distintas tareas o configuraciones, de modo que en lugar de realizar un apartado para cada una de las variaciones, vamos a explicarlas todas de la mejor manera posible intentando que todas ellas queden completamente comprensibles para que más adelante, en el capítulo de resultados podamos mostrar las salidas que hemos obtenido de cada una de ellas y lleguemos así a formular una buena conclusión para nuestro problema.

En este capítulo no entraremos en lo que a código se refiere, sino que trataremos solamente de la idea que tenemos en mente y que hemos aplicado de modo que tengamos como objetivo hacer que se entienda de una manera rápida, sencilla y lo más amena posible.

3.1. Fase 1: Generación de los datos

Esta primera fase es una de las más importantes del proyecto, ya que consiste en entender tanto los entornos de simulación como las bibliotecas y sus tareas para poder realizar pruebas con ellas y extraer sus datos, los cuales serán utilizados posteriormente a la hora de entrenar el modelo.

Como ya hemos comentado previamente, los entornos de simulación utilizados no son del todo iguales. Además, al estar utilizando bibliotecas adaptadas a cada uno de ellos, los métodos para la generación y obtención de datos tampoco son iguales pese a que hayamos intentado adecuar nuestros scripts para que sean lo más parecidos posibles, de modo que podamos estar alternando de un entorno al otro sin necesidad de volver a entender todo de nuevo.

Además, algo también a tener en cuenta es como dependiendo de la tarea que estemos ejecutando, las tres fases se van a ver afectadas, ya que no cuesta lo mismo coger una pelota que abrir una ventana, sobre todo si tenemos en cuenta la información de las observaciones del entorno a cada instante de tiempo, cosa que hace que en MuJoCo tengamos que adaptarnos a cada una de las tareas realizadas. Una vez explicado todo esto, podemos comenzar a hablar de cómo hemos obtenido los datos.

El proceso de generación de los datasets comienza replicando un amplio número de veces una misma tarea aleatorizando el entorno para que cada una de las pruebas sea completamente distinta a las demás y no lleguemos a tener nunca una prueba igual que otra. Esta aleatorización del entorno del que hablamos consiste en inicializar los objetos del entorno en distintos puntos de la escena siempre accesibles por el robot. Además de esto, en el caso de utilizar RLBench, podemos modificar los colores de los objetos también.

Cada vez que repliquemos una prueba, dependiendo del entorno de simulación que estemos utilizando, cada instante de tiempo, almacenaremos una información u otra de modo que podamos formar con ella nuestros conjuntos de datos.

En el caso de CoppeliaSim-RLBench, disponemos de un gran abanico de datos que extraer de la escena. Nosotros hemos decidido extraer tanto el conjunto de los cinco tipos de imágenes de 128x128 capturables mediante cámaras RGB como las representaciones vectoriales de las velocidades, posiciones y fuerzas de las siete articulaciones cada instante de tiempo, siendo la representación vectorial de las velocidades la acción principal con la que realizaremos las pruebas.

En la Figura 15 podemos apreciar como serían las imágenes obtenidas en un instante de tiempo con las cinco clases de cámaras:



Figura 15. Imágenes de Front, Left, Overhead, Right y Wrist de RLBench

En cuanto al caso de MuJoCo-MetaWorld, al no disponer de un conjunto de datos tan extenso, hemos tenido que extraer únicamente los que teníamos disponibles sin poder hacer distinción de cual podría ir mejor o escoger ninguno sobre otro. Estos datos han sido el conjunto de los seis tipos de imágenes de 128x128 capturables mediante cámaras RGB junto con las representaciones vectoriales tanto de las posiciones relativas como de las posiciones absolutas de la pinza del robot acompañadas del estado de la pinza representada de dos modos distintos.

En la Figura 16 podemos apreciar como serían las imágenes obtenidas en un instante de tiempo con las seis clases de cámaras:

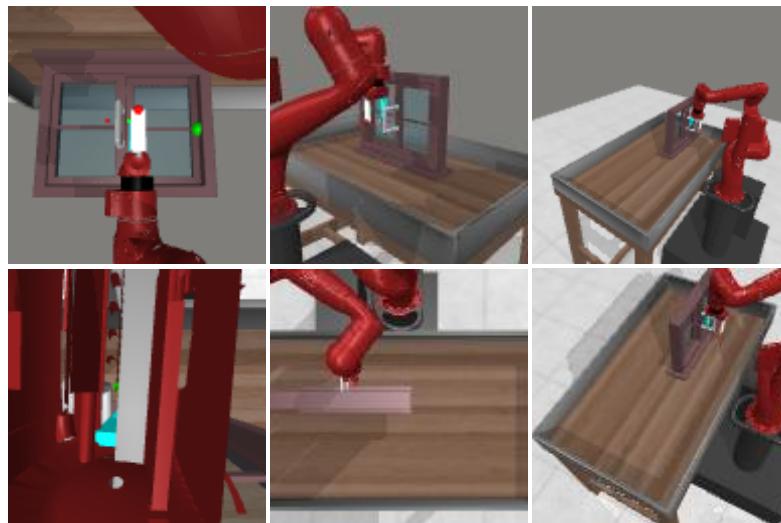


Figura 16. Imágenes de BehindGripper, Corner, Corner2, Gripper, Top y Corner3 de MetaWorld

Podemos apreciar como existen diferencias notorias entre las imágenes de ambos entornos de simulación bien por los ángulos de donde son sacadas las imágenes o bien por el propio entorno y biblioteca. Sin embargo, si nos fijamos en los ángulos de las tomas de las imágenes, vemos como hay imágenes muy parecidas por donde se encuentra la cámara situada.

Una vez extraídos los datos, tendremos una carpeta para train y otra para test en donde dispondremos de las imágenes correspondientes para cada uno de los conjuntos junto con los ficheros de extensión .csv en donde por cada uno de ellos almacenaremos las distintas representaciones vectoriales de las acciones que vayamos a probar. Por otro lado, en el caso de MuJoCo-MetaWorld, dispondremos de igual forma de un fichero para cada conjunto de extensión .csv en donde dispondremos del tamaño de cada una de las pruebas. Este fichero no será necesario en CoppeliaSim-RLBench dado a que obtenemos dichos datos durante la fase de entrenamiento del modelo.

En cuanto al volumen de los datos que podemos llegar a obtener, depende de dos factores principales: la capacidad de almacenamiento del equipo con el que estemos trabajando y el tipo de tarea con la que queramos trabajar. Estos dos motivos se encuentran fuertemente correlacionados, ya que dependiendo del tipo de tarea que vayamos a simular, tendremos pruebas de menor o mayor tamaño, por lo que necesitaremos o bien de más espacio de almacenamiento para el mismo número de pruebas o bien reducir el número de pruebas para mantener el mismo espacio de almacenamiento. Sea como sea, nosotros hemos trabajado con datasets de entre 900 y 2000 pruebas, siempre de en torno a las 15 GB de memoria las cuales equivalen a unas 550.000 imágenes de media, teniendo en cuenta que las imágenes se encuentran repartidas en 10 carpetas en el caso de RLBench y 12 en el caso de MetaWorld. A continuación, en la Tabla 2, vamos a mostrar de una manera más detallada las diferencias entre la toma de datos para ambos entornos de simulación:

Tabla 2. Diferencias entre la magnitud de datos para cada entorno de simulación

	CoppeliaSim-RLBench	MuJoCo-MetaWorld
Pruebas por dataset	1500-2000	900-2000
Imágenes por dataset	400.000-500.000	300.000-600.000

Se puede apreciar como para los dos simuladores, hemos realizado pruebas con distintas cantidades de datos. Esto se debe a que inicialmente hemos generado pruebas más pequeñas que han ido incrementándose a lo largo del proyecto. En cuanto a las diferencias por las pruebas por datasets para ambos entornos, al disponer MuJoCo-MetaWorld de una imagen extra por cada instante de tiempo y encontrarnos realizando pruebas tan extensas, hemos tenido que o bien acortar la cantidad de imágenes si queríamos ocupar el mismo espacio con los datasets de ambos entornos o bien alargar esta cantidad para obtener el mismo número de pruebas por dataset.

Finalmente, hemos acabado utilizando datasets para ambos entornos de un total de 1500 pruebas, valor que consideramos intermedio y que no requiere de tanta cantidad de espacio de almacenamiento como pueden ser los datasets de hasta 2000 pruebas. Esta cantidad sin embargo, ha sido utilizada cuando la tarea a realizar no disponía de una complejidad fuera de lo normal. En caso contrario,

al requerir cada prueba de más cantidad de imágenes por poseer una complejidad mayor, la cantidad de pruebas se ha visto reducida hasta las 1000, valor con el cual seguimos obteniendo la misma cantidad de imágenes que de normal.

Hemos comprobado además que utilizando un dataset con menos de la mitad de imágenes los resultados empeoran lo suficiente como para no funcionar bien en la fase de simulación, y que utilizando un dataset con más imágenes solamente ralentiza el proceso de entrenamiento al lograr mejoras muy poco significativas, por lo que creemos que esta medida, a priori para una sola tarea podría ser la recomendada.

3.2. Fase 2: Entrenamiento del modelo

Una vez hemos finalizado con la generación de datasets para ambos entornos de simulación, vamos a comenzar con la siguiente fase, la cual tiene como objetivo utilizar los datos recopilados para entrenar el modelo que sirva para simular los datos en la última fase.

Para ello, vamos a tratar de aplicar la red neuronal residual denominada ResNet18 que hemos comentado en el apartado de fundamentaciones sobre distintas combinatorias de nuestros datos para intentar obtener la mejor configuración posible la cual tomaremos como referencia en futuros proyectos de este tipo.

Esta red la cargaremos sin ningún preentrenamiento previo y cambiándole la primera convolución por una que generemos nosotros mismos la cual recibirá 3 canales de entrada por cada imagen que le pasemos dentro del mismo instante de tiempo y devolverá un total de 64 canales de salida. El estar trabajando con un conjunto amplio de imágenes hace que debamos tratar el problema bajo la metodología de aprendizaje del Multi-view Learning.

Tras realizar muchas pruebas, hemos decidido aplicar siempre un total de 20 épocas cogiendo un tamaño de lote de 32, ya que utilizando menos épocas obtenemos peores resultados y utilizando más, llegamos a sobreentrenamientos en el modelo. Además de esto, de las 20 épocas que entrenamos, mantenemos congeladas las capas entrenables que no apliquen una normalización del lote de nuestra red durante las cinco primeras épocas, de modo que las únicas capas que puedan verse modificadas sean las que normalicen los lotes.

El proceso comienza con la carga de los datos que hemos obtenido durante la fase anterior. Una vez los tengamos todos, tendremos que decidir la configuración con la que vamos a entrenar nuestro modelo. Dependiendo del entorno del que hemos obtenido los datos, vamos a probar una configuración u otra, ya que como hemos comentado previamente, los datos que nos devuelve la fase de entrenamientos varía dependiendo al entorno que estemos utilizando.

Dejando de lado dichas diferencias, también tendremos que probar distintas configuraciones con las combinatorias de tipos de imágenes con el objetivo de encontrar que combinación es la más optima y así evitar el uso de cámaras o tipos de imágenes innecesarias que solo añadan ruido o que no influyan en la toma de decisión del robot.

Además de probar con distintas combinaciones de los tipos de imágenes, tendremos que evaluar también que representación vectorial de acciones funciona mejor o hasta qué punto debemos de pasarle dicha representación, ya que para un instante cualquiera podemos entrenar la red con la información de las representaciones vectoriales o incluso las imágenes de instantes pasados. Es aquí donde aparecería el Lifelong Learning, con el cual vamos a intentar aportar algo más de información temporal a la red.

Una vez decidida la configuración con la que vamos a entrenar a nuestro modelo, comenzaremos obteniendo cual es la mejor tasa de aprendizaje en base a todos los parámetros que hemos decidido aplicar. De esta manera, evitaremos caer en algún mínimo local determinando así la convergencia adecuada del algoritmo.

Tras el entrenamiento, recibiremos tanto el modelo para poder utilizarlo en la siguiente fase como un conjunto de estadísticos con los cuales evaluar dicho modelo (y por tanto, dicha configuración de los datos) sobre los cuales hablaremos más adelante en el capítulo dedicado a los resultados.

Es en base a estos estadísticos sobre los que tomaremos la decisión de modificar algo en nuestra configuración o probar el modelo en el simulador. Esto último es debido a que si analizando los estadísticos nos damos cuenta de que los porcentajes de error son demasiado elevados, lo más seguro es que evitemos seguir probando por ese camino.

3.3. Fase 3: Simulación y validación del modelo

Finalmente, llegamos a la última de nuestras fases dentro del desarrollo del problema. En esta fase, una vez hemos evaluado previamente todos los modelos obtenidos, decidiremos cuales debemos de probar en su correspondiente simulador y cuales, tras analizar los estadísticos devueltos en la fase anterior, debemos de descartar sin necesidad de simular.

Una vez tengamos cargados nuestros modelos, pasaremos a realizar un procedimiento muy similar que en la primera fase, ya que tendremos que replicar nuevamente un conjunto de pruebas sobre una misma tarea.

Al contrario que en la primera fase, en esta no vamos a obtener las acciones gracias a funciones internas de cada biblioteca, sino que vamos a obtenerlas gracias a nuestro modelo. Para ello, por cada instante de tiempo, almacenaremos la misma configuración de datos que hemos utilizado para entrenar nuestro modelo y se la pasaremos de modo que este nos devuelva por salida cual es la acción predicha.

Una vez obtengamos la acción, realizaremos el movimiento y volveremos a repetir todo el proceso hasta que finalicemos la tarea correctamente o alcancemos el umbral de iteraciones máximo con el que designaremos la prueba como fallida.

Mientras realizamos todo este proceso, iremos almacenando a su vez otro conjunto de estadísticos como pueden ser el porcentaje de acierto en simulación o la media de iteraciones necesarias para que se finalice correctamente una prueba además de guardar todas las imágenes obtenidas por las cámaras para poder visualizar al finalizar los posibles errores e intentar buscar patrones en ellos.

Capítulo 4

Caso de estudio

Tras haber comentado ya a lo largo de los capítulos anteriores en que consiste nuestro problema y como lo vamos a desarrollar, vamos a pasar a este nuevo capítulo en donde expondremos todos los casos de estudio que vamos a llevar a cabo.

Para ello, comenzaremos hablando acerca del tipo de robot que hemos escogido como candidato para la realización de este problema, el cual utilizamos en ambos entornos de simulación.

Tras esto, pasaremos a exponer los distintos casos de uso en donde vamos a utilizar este robot y las diferencias que existen entre sí, ya que pese a escoger una tarea como candidata para los dos entornos, hemos decidido repetir las mismas pruebas para otra clase de tareas de igual o mayor complejidad de modo que podamos obtener unas mejores conclusiones.

Finalmente, hablaremos acerca de las medidas de calidad que vamos a aplicar para analizar los resultados, los cuales expondremos en el siguiente capítulo de una manera más detallada que la que podamos encontrar aquí.

4.1. El robot

Para la realización de nuestro proyecto, hemos decidido usar el robot articulado Panda, de la empresa alemana Franka Emika [25]. Panda es un robot colaborativo que dispone de un total de 7 articulaciones, las cuales le proporcionan hasta siete grados de libertad.

Panda es capaz de moverse de diferentes maneras gracias a la posibilidad de controlarlo de forma directa, mediante scripts o conectarlo a sensores externos que reproduzcan sus movimientos en base al entorno.

La cantidad de sensores de torsión que dispone convierten a este robot en un brazo articulado extremadamente sensible, llegando a detenerse hasta frente el contacto de un globo. Además de esto, sus sensores dotan al robot de una gran precisión y estabilidad.

Este brazo robótico puede ser usado en un amplio rango de aplicaciones, desde líneas de producción, centros de cuidados, hospitales, o incluso laboratorios.

En la Figura 17 podemos apreciar como es este robot en el mundo real y como se reparten sus siete grados de libertad:

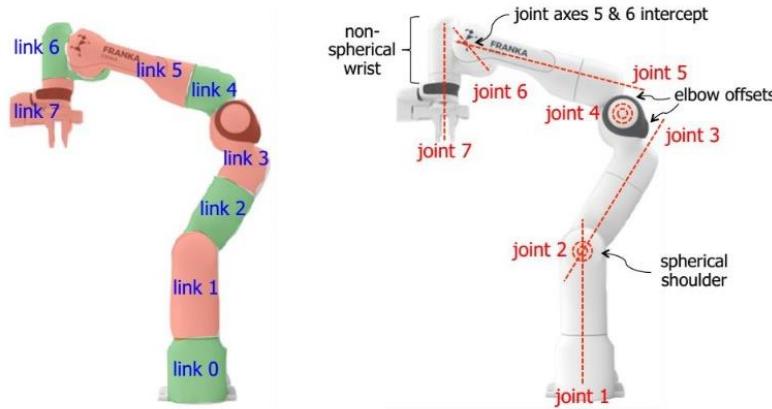


Figura 17. Apreciación de los siete grados de libertad del robot

Dentro de nuestros dos entornos de simulación, pese a parecer distintos en las imágenes proporcionadas por estar pintado de color rojo en MuJoCo, hacemos uso de este mismo robot.

4.2. Casos de uso

Tal y como ya hemos mencionado, para el estudio y realización de este problema hemos decidido escoger una tarea como candidata para ambos entornos de simulación tras ver cómo era una de las pocas tareas que se replicaban tanto en CoppeliaSim como en MuJoCo, además de resultar ser una tarea con un nivel de complejidad muy bajo con el que poder comenzar a analizar los datos extraídos de una manera más sencilla para poder probar en un futuro las conclusiones que obtengamos en otro tipo de tareas algo más complejas.

Esta tarea trata de coger una pelota de la misma base en la que se encuentra el robot. En el caso de CoppeliaSim, la tarea incluye obstáculos intermedios en el eje vertical los cuales deberá de evitar, mientras que en el caso de MuJoCo, tan solo disponemos de la pelota en la escena. En la Figura 18 podemos ver la diferencia entre ambos entornos:



Figura 18. Diferencias para la realización de la tarea entre ambos entornos de simulación

Llegados a este punto podemos ver como inicialmente, la tarea representada mediante MuJoCo puede resultar menos compleja al no disponer de más objetos en la escena que la propia pelota. Sin embargo, esto la hace más realista, ya que en una línea de producción no vamos a disponer de obstáculos aleatorios, sino que los que tengamos serán en su gran mayoría fijos, lo cual hará que el modelo aprenda a evitarlos.

En el caso de que obtengamos buenos resultados para esta tarea inicial con cada uno de los entornos, pasaremos a probar sus resultados con otro conjunto de tareas, pero inicialmente nos centraremos en esta por ser la única que se repite en ambos entornos y por tanto devuelve resultados comparables entre sí.

Para el caso de CoppeliaSim-RLBench, haremos uso de distintas combinaciones de entre los cinco tipos de imágenes que disponemos junto con una representación vectorial con tantos valores como grados de libertad posea el robot de los distintos tipos de fuerzas que ya hemos mencionado, mientras que en el caso de MuJoCo-MetaWorld usaremos las diferentes combinaciones de los seis tipos de imágenes que existen junto con la representación vectorial de los tres valores continuos que representan el diferencial de la posición de la pinza en cada instante de tiempo.

4.3. Medidas utilizadas

Para evaluar la calidad de nuestros resultados, ya hemos comentado que vamos a hacer uso de distintas métricas. La primera que tendremos en cuenta una vez finalicemos el entrenamiento de nuestros modelos será el error cuadrático medio (en lo sucesivo RMSE).

Esta medida es la desviación estándar de los valores residuales, es decir, los errores producidos en la predicción, y mide la cantidad de error que hay entre dos conjuntos de datos. En otras palabras, hace una comparativa entre el valor predicho y el valor real que tratamos de predecir.

A continuación se detalla la formula con la que se calcula esta medida, en donde n es la cantidad de valores dentro de ambos conjuntos de datos, P el conjunto de datos con los n valores predichos y R el conjunto de datos con los n valores reales:

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (P_i - R_i)^2}{n}}$$

Nótese que cada prueba va a estar formada por un dataset o conjunto de datos distinto, por lo que pese a que esta medida hace que tengamos el error normalizado por la inmensa cantidad de comparativas que realiza para su cálculo, cada una va a tener un rango distinto de error, lo cual va a afectar al RMSE y va a hacer que sea muy complicado que podamos contrastar la calidad entre distintas tareas. A continuación vamos a analizar en qué casos no va a ser comparable el valor del RMSE:

- La métrica de RMSE no puede ser comparable entre distintos tipos de tareas el ejercer cada tarea una trayectoria distinta con pequeñas peculiaridades entre si como puede ser la dirección del primer movimiento que se realice.
- La métrica de RMSE no puede ser comparable entre distintos entornos de simulación por poseer cada uno de ellos acciones en forma de representaciones vectoriales con un numero distinto de valores, el cual hará que la precisión se vea afectada y por ende, no sea comparable.

- La métrica de RMSE no puede ser comparable entre mismas tareas que utilicen distintos tipos de acciones. Esto se debe a que no es lo mismo comparar el RMSE obtenido mediante las posiciones que mediante la velocidad.

Así pues, usaremos el RMSE únicamente como medida comparativa dentro de pruebas que traten de predecir la misma tarea mediante el mismo tipo de acción. De todos modos, usaremos esta medida como referencia para saber que modelos aplicar y cuales descartar a la hora de realizar las simulaciones además de ver con que combinatoria de datos merece la pena seguir investigando dentro de la fase de entrenamiento del modelo.

Es por este motivo por el que necesitaremos hacer uso de una segunda medida independiente a esta con la que podamos comparar cualquier clase de prueba. Esta medida será el porcentaje de acierto en simulación, y como el nombre indica, calculará el porcentaje de acierto que tiene cada modelo en el entorno de simulación. Será calculado bajo el mismo conjunto de pruebas, por lo que será completamente comparable tanto para pruebas que hagan uso de la misma tarea como de pruebas que no lo hagan.

Junto con esta medida, tendremos que evaluar también otro conjunto de estadísticos como son las iteraciones máximas y mínimas o la media de iteraciones necesarias para finalizar una prueba en caso de acierto, ya que no es lo mismo finalizar correctamente 90 pruebas con una media de 150 iteraciones que finalizar correctamente tan solo 80 pero con una media de 50 iteraciones. Todo esto tendremos que tenerlo en cuenta a la hora de tomar decisiones.

Capítulo 5

Resultados y análisis experimental

En este capítulo nos centraremos en comentar todas las pruebas realizadas así como evaluarlas y analizarlas por medio de un conjunto de medidas sobre las cuales hablaremos a continuación.

Trataremos de analizar y comentar las pruebas de una manera coherente con la que poder tomar las decisiones que sean necesarias para llegar a una conclusión final que nos responda a la problemática que planteamos en este trabajo.

5.1. Marco experimental

Para la realización de nuestro análisis experimental, tal y como ya hemos mencionado previamente, hemos decidido probar con las diferentes combinatorias de los datos extraídos durante la fase de extracción y generación de datos tanto con CoppeliaSim como con MuJoCo.

En el caso de CoppeliaSim, hemos escogido como datos candidatos los cinco tipos de imágenes en RGB junto con la representación vectorial de la velocidad, posición y fuerza como acciones a tener en cuenta. De entre estas tres, ha sido la representación vectorial de la velocidad la que hemos tomado como base al ser el tipo de acción con la que más se trabajaba en proyectos similares. Una vez obtenidos los resultados con dicha acción, pasaremos a aplicarlos para el resto de acciones para poder así ver con cual funciona mejor el problema.

Para MuJoCo, en cambio, al solo tener la posición absoluta de las pinzas disponible junto con su estado, hemos decidido hacer las pruebas de la combinatoria de los seis tipos de imágenes únicamente con esta acción. Al no ser el mismo tipo de acción que la que manejamos con CoppeliaSim, nos servirá para

poder observar el funcionamiento de un nuevo tipo de acción con la que manejar las trayectorias del robot. La posición absoluta de la pinza del robot es interesante respecto al resto de acciones utilizadas en CoppeliaSim por necesitar de la mitad de valores y aportar solamente información sobre el final del robot, de modo que el resto de articulaciones se muevan con ella.

Al estar realizando un estudio sobre cuál es la mejor generalización de datos para esta clase de problemas, necesitamos probar toda la combinatoria posible de los datos que hemos obtenido con ambos entornos de simulación. Es por este motivo por el que hemos decidido comenzar el estudio utilizando cada uno de los tipos de imágenes de manera individual, de modo que en base a los resultados obtenidos por el modelo devuelto, sigamos utilizando dicha combinatoria de datos o decidamos descartarla al ver que no va a ser capaz de mejorar los datos previos obtenidos por el modelo generado. De esta manera, pasaremos de las pruebas individuales a las pruebas por pares de tipos de datos y así sucesivamente hasta que realicemos una prueba final con el conjunto total de datos. Como ya hemos comentado, uno de nuestros objetivos es intentar utilizar el mínimo número posible de datos, por lo que primaremos las pruebas con menos tipos de imágenes como datos de entrada pese a tener un resultado algo inferior a otra prueba con más cantidad de esta clase de datos de entrada. Todo esto se verá expuesto más adelante cuando analicemos cada una de las pruebas realizadas en base a sus resultados.

En cuanto a cómo vamos a estructurar el análisis de los resultados obtenidos, comenzaremos exponiendo los resultados para las pruebas que hayan sido realizadas con un único tipo de dato e iremos incrementándolas. Esto lo haremos mostrando la tabla de resultados de cada una de estas pruebas para finalmente, tras haber sido visualizadas, sacar una conclusión tanto sobre las pruebas mostradas como por lo que vamos viendo y analizando con el paso de las pruebas además de los patrones que vayamos identificando en cada una de ellas. Con el objetivo de facilitar la visualización y tener una mejor comprensión sobre las conclusiones a las que estemos llegando, por cada bloque de resultados, pintaremos las celdas de cada una de las medidas de verde o rojo en base a si es el mejor o el peor resultado de dicho bloque. Gracias a esto podremos identificar mejor tanto las mejores como las peores pruebas.

5.1.1. Resultados con MuJoCo-MetaWorld

Vamos a comenzar analizando las pruebas realizadas mediante el entorno de simulación de MuJoCo en donde para la tarea candidata de coger una pelota de una mesa hemos decidido comenzar únicamente pasándole a la red un único tipo de imagen e ir subiendo hasta probar toda la combinatoria posible siempre y cuando uno de los tipos de imágenes de dicha combinatoria devuelva como salida unos resultados prometedores o al menos dentro de unos parámetros preestablecidos, es decir, si vemos que utilizando un tipo o combinatoria de datos los resultados devuelven unos porcentajes de error muy por encima de los esperado en comparación con el resto de pruebas, no continuaremos con una mayor combinatoria que englobe dicha prueba como ítems del conjunto de datos.

Para cada prueba hemos decidido almacenar seis valores distintos, los cuales hacen referencia a las diferentes medidas que habíamos comentado previamente. Estos valores son los siguientes:

- **% Eje X:** Es el porcentaje de error de la posición de la pinza en el eje X.
- **% Eje Y:** Es el porcentaje de error de la posición de la pinza en el eje Y.
- **% Eje Z:** Es el porcentaje de error de la posición de la pinza en el eje Z.
- **RMSE:** Representa la diferencia entre las predicciones y la acción real.
- **AccSim:** Se trata del porcentaje de acierto que devuelve en simulación.
- **MediaAcierto:** Es la media de iteraciones necesarias para las pruebas acertadas en simulación.

5.1.1.1. Pruebas con un único tipo de imagen

Antes de comenzar, debemos recordar como para este entorno de simulación disponíamos de un total de 6 tipos diferentes de imágenes, las cuales vamos a denominar dentro de la primera columna de las tablas de resultados con las equivalencias mostradas en la Tabla 3 que aparece a continuación:

Tabla 3. Nomenclaturas para las cámaras de MuJoCo

Top	Corner	Corner2	Corner3	BehindGripper	Gripper
T	C	C2	C3	BG	G

Inicialmente, al no tener datos previos con los que basarnos en qué tipo de imágenes puede funcionar mejor, hemos realizado la prueba con los seis tipos de imágenes que disponemos en MuJoCo. Vamos a exponer los resultados conjuntamente dentro de la Tabla 4 para después analizarlos uno a uno:

Tabla 4. Resultados de MuJoCo para pruebas con un solo tipo de imagen

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T	1.46%	1.52%	6.72%	0.25766	35.00%	42.65
C	1.07%	0.95%	5.72%	0.20363	90.00%	41.26
C2	1.37%	1.79%	6.73%	0.25449	100.00%	37.27
C3	1.37%	4.11%	6.11%	0.29056	100.00%	38.82
BG	1.13%	1.45%	5.76%	0.21390	100.00%	46.63
G	4.40%	4.35%	8.73%	0.59091	40.00%	43.15

En cuanto a las pruebas individuales, se puede apreciar en la Tabla 4 como utilizando únicamente el tipo Corner, en la fase de entrenamiento, obtenemos unos resultados muy buenos que no se acaban de ver reflejados finalmente a la hora de realizar las simulaciones, ya que al contrario que con otras pruebas, no alcanzamos el 100% de acierto y además, de entre todas las pruebas acertadas, la media de iteraciones necesarias para alcanzar el objetivo se sitúa por encima de las 41, valor que se aleja del obtenido con la prueba realizada con Corner2, en donde no solo alcanzamos el 100% en simulación sino que obtenemos la mejor media de acierto llegando casi a un 37. Esto es curioso, ya que en el resto de métricas obtenidas durante la fase de entrenamiento, este modelo no parecía que fuera a funcionar bien viendo los resultados del resto de pruebas.

En cuanto a las demás, vemos como utilizando solamente imágenes del tipo Gripper, se obtienen unos resultados bastante malos que van a hacer que dejemos de lado este tipo de imagen y nos centremos en el resto.

Algo muy curioso es ver como la prueba realizada con el tipo Corner3 puede obtener unos resultados en entrenamiento tan alejados a los obtenidos por las mejores pruebas pero que a su vez genere el segundo mejor modelo de este conjunto de pruebas individuales siendo junto con la prueba del tipo Corner2 la única que baja de las 40 iteraciones de media obteniendo un 100% de acierto en simulación.

Pasa lo contrario con la prueba realizada con el tipo BehindGripper, en donde recibimos unos resultados muy prometedores en la fase de entrenamiento que nos hacen pensar que vamos a obtener lo mismo en simulación. Sin embargo,

y pese a llegar al 100% de acierto, vemos como la media de iteraciones necesaria sobrepasa las 46. Visto esto, podríamos llegar a la conclusión de que pese a acertar siempre con este tipo de imagen, no se realiza una trayectoria limpia u optima, lo cual penaliza el modelo pese a tener unos muy buenos resultados en el resto de medidas.

Es con esta última prueba con la que vemos la importancia de esta última medida, ya que no es lo mismo acertar todas las pruebas en 15 iteraciones que hacerlo en 150. Además, tenemos que tener en cuenta que no siempre nos será igual de válida, pues cuanto menor sea nuestro porcentaje de acierto en simulación, menos peso va a tener al estar teniendo en cuenta un número menor de pruebas.

En definitiva, tras analizar estas pruebas individuales, podemos llegar a dos conclusiones. La primera es que no debemos fijarnos únicamente en el comportamiento durante el entrenamiento, ya que hay veces que este no se ve reflejado durante la fase de simulación y por tanto, tenemos que tratar de analizarlas conjuntamente. La segunda conclusión es que las imágenes que mejor parecen funcionar son aquellas que muestran la escena desde diferentes lados (Corner, Corner2 y Corner3) seguido de las que las muestran desde arriba (Top) o enfrente (BehindGripper). Las imágenes tomadas desde la propia pinza (Gripper) aportan ruido y poca información sobre la escena si se aplican individualmente, pero puede ser que aporten información dentro de un conjunto de tipos de imágenes.

5.1.1.2. Pruebas con pares de tipos de imágenes

Una vez que hemos analizado el funcionamiento de cada uno de los tipos de imágenes de manera individual, nos toca realizar las mismas pruebas por pares de tipos de imágenes para ver si somos capaces de mejorar los resultados mostrados anteriormente. Tras ver lo mal que funcionaba el tipo Gripper, hemos decidido no aplicarlo más a partir de ahora. En la siguiente Tabla 5 podemos observar cuales han sido los resultados conseguidos para este tipo de pruebas:

Tabla 5. Resultados de MuJoCo para pruebas con pares de tipos de imágenes

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C	1.03%	0.85%	5.59%	0.19714	95.00%	40.57
T, C2	0.68%	1.01%	6.22%	0.10155	100.00%	38.75
T, C3	2.04%	5.80%	6.01%	0.38084	20.00%	43.27
C, C2	0.63%	1.23%	7.26%	0.21830	79.00%	42.02
C, C3	0.95%	1.42%	6.33%	0.21397	95.00%	39.82
C2, C3	0.94%	1.37%	6.99%	0.22775	100.00%	42.51
C, BG	0.92%	0.94%	4.02%	0.15814	100.00%	43.19

En este segundo conjunto de pruebas realizadas por pares, podemos volver a apreciar fácilmente en la Tabla 5 cuál de ellas funciona peor tanto en entrenamiento como en simulación por tener tantos resultados marcados como los peores obtenidos de entre todas. Esta prueba se trata de la realizada juntando los tipos Top y Corner3. Esto puede ser debido al mal funcionamiento durante simulación del tipo Top de manera individual. Sin embargo, vemos como nuestra mejor prueba de este conjunto incluye el tipo Top junto con el tipo Corner2, por lo que puede ser que esa combinatoria no sea capaz de aportar la suficiente información al robot para poder realizar correctamente las trayectorias.

Tal y como acabamos de mencionar, parece ser que la prueba que mejores resultados devuelve es la que junta los tipos Top y Corner2. Del mismo modo que hemos dicho ya que es curioso que el tipo Top se encuentre dentro de este mejor conjunto, no lo es que Corner2 lo esté también, ya que individualmente era la prueba que mejor ha llegado a funcionar en simulación muy por encima del resto de tipos de imágenes. Esta prueba es capaz de casi bajar el RMSE de un 0.1, lo cual si nos fijamos en el resto de pruebas ya es difícil que lo haga hasta de un 0.2.

En cuanto a las simulaciones, la media de acierto es la mejor con diferencia pese a no llegar a bajar la obtenida en la pasada prueba individual del tipo Corner2 que podemos ver en la Tabla 4.

Sorprende ver como la prueba en donde juntamos los tipos Corner2 con Corner3 empeoren a sus pruebas individuales en cuanto a la media de iteraciones necesarias. Esto podría ser debido a que al ser imágenes muy parecidas entre sí, no se lleguen a complementar del todo bien.

Así pues, la conclusión que podemos sacar de estas pruebas es que Corner2 parece afianzarse como uno de los mejores tipos con bastante diferencia. Veremos qué tal funciona al aplicársele un tercer tipo a la prueba.

5.1.1.3. Pruebas con tríos de tipos de imágenes

Tras ver el funcionamiento y los resultados de las pruebas realizadas por pares dentro de la Tabla 5, vamos a pasar a analizar los modelos que han trabajado con un total de tres tipos distintos de imágenes, de los cuales mostraremos los resultados dentro de la Tabla 6. Llegados a este punto, hemos tenido que ir descartando posibles subconjuntos de tipos de imágenes por su comportamiento en pasadas pruebas, de modo que vamos a tener un total de tres pruebas para analizar.

Tabla 6. Resultados de MuJoCo para pruebas con tríos de tipos de imágenes

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C, C2	0.71%	1.04%	6.44%	0.19980	100.00%	39.39
T, C, BG	0.63%	1.67%	4.00%	0.14997	100.00%	39.41
T, C2, BG	0.63%	1.42%	4.80%	0.16216	90.00%	56.31

Al tener menos pruebas para analizar, es más evidente destacar cual es la peor de ellas. En este caso, se trata de la prueba que incluye los tipos Top, Corner2 y BehindGripper. Vemos como esta prueba no solo no ha sido capaz de mejorar la pasada mejor en donde teníamos Top y Corner2 sino que la ha empeorado bastante en lo que al apartado de simulación se refiere, llegando a alcanzar una media de iteraciones de acierto muy por encima del total de pruebas realizadas hasta el momento. En cuanto a la mejor de las pruebas obtenidas, la decisión no es fácil, ya que tanto la primera como la segunda prueba maneja unos valores muy parejos. Sin embargo, podemos ver como la primera de ellas tiene un RMSE y un porcentaje de error para el eje Z muy alto en comparación con la segunda de las pruebas. Si nos fijamos a las medidas de simulación, ambas son un calco, por lo que decidimos seguir con la prueba que tiene los tipos Top, Corner y BehindGripper tras ver como estas funcionan mejor en la fase de entrenamiento en conjunto.

5.1.1.4. Pruebas con otras combinatorias de tipos de imágenes

Al habernos quedado de la pasada prueba solamente con Top, Corner y BehindGripper, no tenemos muchas más pruebas que realizar. Es por esto por lo que hemos decidido partir de estas tres para ir sumando de una en una hasta llegar a realizar una prueba con el conjunto total de tipos de imágenes. Comenzaremos incluyendo Corner2, la cual ha sido la que mejor ha funcionado para pruebas individuales y la que todavía arroja los mejores resultados del total de pruebas. Después, dentro de esta Tabla 7, incluiremos a esta prueba el tipo Corner3 y finalmente realizaremos un modelo con todos los tipos de imágenes que disponemos.

Tabla 7. Resultados de MuJoCo para pruebas con otras combinatorias de tipos de imágenes

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C, C2, BG	0.79%	1.47%	5.00%	0.17654	100.00%	39.64
T, C, C2, C3 BG	0.57%	0.89%	4.42%	0.14304	100.00%	41.70
ALL	0.70%	1.33%	4.84%	0.16564	75.00%	41.15

Como vemos en la Tabla 7, ocurre algo muy curioso con estas pruebas. Podemos ver como en la primera de ellas obtenemos los peores resultados de entre las tres siendo la prueba que peores resultados en la fase de entrenamiento devuelve. Esto no quiere decir que sean malos, ya que si los comparamos con el resto de pruebas realizadas hasta el momento, los resultados están bastante bien. En cuanto a la segunda prueba en donde añadimos el tipo Corner3, los resultados de entrenamiento son los mejores de entre los tres. Sin embargo, en cuanto a la simulación, pese a acertar un 100%, obtenemos una media de iteraciones bastante elevado llegados a este punto. En la última de las pruebas en donde incluimos todos los tipos de imágenes, no parece funcionar muy bien, pues nos quedamos en un 75% de acierto.

Una vez hemos realizado todas las pruebas para la tarea de coger una pelota con MuJoCo-MetaWorld y después de haberlas comparado en subgrupos, toca analizarlas tomándolas todas como pruebas candidatas de modo que podamos ver cuál de ellas nos ha devuelto unos mejores resultados y por tanto, qué configuración ha funcionado mejor para esta tarea y entorno.

Si nos fijamos únicamente en lo que a las métricas de calidad del apartado de entrenamiento se refiere, ninguna tiene las cuatro medidas como las mejores del total. Sin embargo, teniendo en cuenta la métrica más importante de entre estas cuatro, el RMSE, podemos ver como la prueba que nos devuelve el mejor RMSE es la prueba compuesta por Top y Corner2, en donde su valor es de 0.10155. El segundo mejor RMSE se nos marcha ya hasta el 0.14304. Sus otras métricas para entrenamiento son también muy buenas, salvo en el caso del porcentaje de error en el eje Z, el cual es en todas muy alto. Esta métrica nos preocupa menos, ya que al estar en un espacio tridimensional, podemos llegar al objetivo de igual forma por muchos caminos. Esto lo medimos mejor en simulación por medio de la media de iteraciones en acierto.

Pasando a las métricas de simulación, vemos como al ser una tarea relativamente sencilla, alcanzamos bastantes 100% de accuracy, por lo que debemos fijarnos mejor en la media de iteraciones de estas pruebas. En todas rondamos el 38-39 salvo en cuatro en donde sobrepasamos el 41. Sin embargo, hay una que destaca del resto por ser la única que baja de 38. Esta prueba se trata de la formada solamente por imágenes del tipo Corner2. Al hacer uso de un solo tipo de imagen, ponderamos muy positivamente este resultado, ya que tal y como explicamos en la introducción de la memoria, uno de nuestros objetivos era ahorrar en costes y tiempo, lo cual ocurre si en lugar de utilizar 6 cámaras usamos solamente una. Pese a ello, al no tener tanta diferencia en las métricas de simulación y mejorar por mucho las de entrenamiento aplicando el tipo Top a Corner2, hemos decidido que nuestro mejor resultado es el modelo que hace uso tanto de Top como de Corner2, ya que se vale de los buenos resultados de Corner2 para mejorarlos o al menos no empeorarlos apoyándose en el tipo Top.

Con el objetivo de afianzar estos resultados, vamos a replicar las pruebas con distintas tareas una vez hayamos analizado el comportamiento para la misma tarea con CoppeliaSim-RLBench.

5.1.2. Resultados con CoppeliaSim-RLBench

Tras haber analizado los resultados obtenidos por el otro entorno de simulación, vamos a mostrar cuales han sido los resultados que hemos conseguido con CoppeliaSim para la misma tarea.

El procedimiento para escoger con que pruebas debemos de seguir es el mismo que el utilizado con MuJoCo, en donde en base a los resultados de cada subconjunto de datos, decidiremos si añadirlos como parte de una nueva prueba o no. Todo esto lo haremos con dos tipos distintos de acciones, el diferencial de velocidad y el de posición, los cuales al contrario que en el caso de MuJoCo, están compuestos por un vector de siete valores, uno por cada una de las articulaciones del robot.

Esta prueba nos servirá también para ver que es mejor, si utilizar únicamente los datos de la pinza y que con ella se muevan el resto de articulaciones o tener el control total sobre todas las articulaciones del robot.

Para cada prueba hemos decidido almacenar ocho valores distintos, los cuales hacen referencia a las diferentes medidas que habíamos comentado previamente. Estos valores son los siguientes:

- **% A1:** Es el porcentaje de error de la articulación número 1.
- **% A2:** Es el porcentaje de error de la articulación número 2.
- **% A3:** Es el porcentaje de error de la articulación número 3.
- **% A4:** Es el porcentaje de error de la articulación número 4.
- **% A5:** Es el porcentaje de error de la articulación número 5.
- **% A6:** Es el porcentaje de error de la articulación número 6.
- **RMSE:** Representa la diferencia entre las predicciones y la acción real.
- **AccSim:** Se trata del porcentaje de acierto que devuelve en simulación.

Hemos decidido suprimir la medida de media de iteraciones en caso de acierto tras ver como los resultados de simulación no eran lo suficientemente buenos como para necesitar de dicha medida.

A diferencia de con el otro entorno de simulación, en este caso, al tener más de un tipo de acción, vamos a comentar las pruebas todas juntas, es decir, dentro de la primera subsección de resultados para las pruebas individuales

tendremos los resultados obtenidos tanto por un tipo de acción como por el otro, pese a tener cada tipo de acción su correspondiente explicación al finalizar con la visualización de sus tablas.

5.1.2.1. Pruebas con un único tipo de imagen

En el caso de CoppeliaSim, contamos con un total de 5 tipos diferentes de imágenes, las cuales vamos a denominar también dentro de la primera columna de las tablas de resultados con las equivalencias mostradas en la Tabla 8 que aparece a continuación:

Tabla 8. Nomenclaturas para las cámaras de CoppeliaSim

Front	Left	Right	Overhead	Wrist
F	L	R	O	W

Al principio, al igual que hacíamos con el anterior entorno de simulación, al no tener datos previos con los que basarnos en qué tipo de imágenes puede funcionar mejor, hemos realizado la prueba con los cinco tipos de imágenes que disponemos. Vamos a exponer los resultados conjuntamente para después analizarlos uno a uno. Comenzaremos mostrando en la Tabla 9 las pruebas realizadas tomando el diferencial de velocidad como acción:

Acción: Diferencial de Velocidad

Tabla 9. Resultados de CoppeliaSim para pruebas con un solo tipo de imagen

Cámaras	%A1	%A2	%A3	%A4	%A5	%A6	%A7	RMSE	AccSim
F	3.08%	1.60%	3.97%	1.49%	1.67%	2.38%	1.43%	0.02388	15.00%
O	1.63%	1.53%	1.28%	1.36%	1.37%	1.98%	1.22%	0.01753	35.00%
L	11.39%	3.56%	50.07%	2.77%	23.93%	4.12%	21.04%	0.21401	0.00%
R	1.87%	1.59%	1.76%	1.28%	1.20%	2.12%	1.28%	0.01811	18.00%
W	342.81%	93.71%	310.14%	81.34%	119.30%	100.53%	68.64%	1.70418	0.00%

Podemos observar en la Tabla 9 cómo si cogemos individualmente las imágenes de la pinza, los resultados son los peores. Además de esto, podemos ver también como en lo que a las métricas de entrenamiento se refiere, la prueba con el tipo de imagen Overhead funciona muy bien, devolviendo unos porcentajes de

error para cada una de las articulaciones por debajo del 2%. El resto de pruebas sobrepasan este porcentaje en alguna de sus articulaciones, siendo la del tipo Left la que peores resultados devuelve.

En cuanto a la simulación, los resultados son muy malos si los comparamos con el anterior entorno de simulación en donde lo normal era no bajar de un 90%. Nuestra mejor prueba tiene tan solo un 35% de acierto en simulación, lo cual nos hace pensar que este entorno puede no ser tan bueno como esperábamos inicialmente. De todos modos, vamos a seguir en la Tabla 10 con las pruebas individuales, esta vez aplicando el diferencial de posición como acción a predecir:

Acción: Diferencial de Posición

Tabla 10. Resultados de CoppeliaSim para pruebas con un solo tipo de imagen

Cámaras	%A1	%A2	%A3	%A4	%A5	%A6	%A7	RMSE	AccSim
F	1.59%	2.96%	1.55%	2.63%	2.02%	4.16%	1.31%	0.04307	5.00%
O	29.61%	41.52%	51.37%	52.74%	134.18%	35.93%	42.37%	0.71666	0.00%
L	12.30%	2.87%	2.97%	4.09%	16.56%	5.73%	3.56%	0.07665	5.00%
R	4.70%	3.21%	6.22%	7.99%	3.02%	6.66%	2.14%	0.09041	0.00%
W	68.88%	19.61%	63.45%	16.16%	82.48%	17.19%	10.49%	0.39652	0.00%

Si nos fijamos en las pruebas realizadas aplicando el diferencial de posición como acción a predecir vemos como tampoco cambia mucho la cosa, ya que seguimos teniendo unos resultados en simulación muy malos volviendo a dejar hasta tres pruebas en 0%.

La que para el diferencial de velocidad era la mejor prueba, en este caso pasa a ser la peor de todas y con mucha diferencia tal y como podemos ver, llegando a empeorar incluso los registros devueltos por el tipo Wrist que hasta el momento era el que peores resultados nos había dado siempre en ambos entornos de simulación.

En este caso es la prueba con el tipo Front la que devuelve los mejores resultados tanto de entrenamiento como de simulación. Sin embargo, estos no son nada buenos si los comparamos con la mejor de las pruebas para el otro tipo de acción y menos si la comparamos a los resultados devueltos por el otro entorno de simulación. Esto no hace más que afianzar la idea de que CoppeliaSim puede no funcionar tan bien como parecía inicialmente a la hora de simular datos. Vamos

a probar que tal evolucionan los resultados tras empezar a combinar distintos tipos de imágenes.

5.1.2.2. Pruebas con pares de tipos de imágenes

Tras haber visto como han sido los resultados de las pruebas individuales, hemos tenido que descartar unas cuantas combinatorias posibles quedándonos únicamente con tres para cada tipo de acción.

Acción: Diferencial de Velocidad

Tabla 11. Resultados de CoppeliaSim para pruebas con pares de tipos de imágenes

Cámaras	%A1	%A2	%A3	%A4	%A5	%A6	%A7	RMSE	AccSim
F, O	5.91%	2.31%	6.91%	1.96%	4.27%	2.55%	4.43%	0.04394	7.00%
F, R	17.29%	11.16%	48.91%	9.69%	19.78%	6.38%	21.08%	0.22243	0.00%
O, R	3.30%	1.63%	29.86%	1.67%	16.20%	3.23%	8.84%	0.12401	0.00%

Acción: Diferencial de Posición

Tabla 12. Resultados de CoppeliaSim para pruebas con pares de tipos de imágenes

Cámaras	%A1	%A2	%A3	%A4	%A5	%A6	%A7	RMSE	AccSim
F, L	33.37%	2.76%	7.47%	2.45%	5.71%	3.14%	2.85%	0.10357	0.00%
F, R	1.80%	2.75%	1.28%	3.30%	4.75%	3.85%	1.08%	0.04514	5.00%
L, R	1.52%	3.08%	1.98%	3.84%	1.92%	4.53%	1.48%	0.05165	0.00%

En cuanto a las pruebas por pares para el diferencial de velocidad de la Tabla 11, claramente vemos como la prueba formada por los tipos Front y Overhead son los que mejores resultados devuelven. Sin embargo, estos no mejoran en nada la pasada mejor prueba que teníamos con el tipo Overhead, llegando a empeorarla en todas las métricas.

Si nos fijamos en las pruebas por pares para el diferencial de posición de la Tabla 12, vemos como la mejor de todas es la formada por Front y Right. Sin embargo, vuelve a pasar lo mismo que antes, empeoramos los resultados obtenidos en la pasada mejor prueba que era Front.

Lo único que podemos sacar en claro hasta ahora es que el tipo de imagen Front es el que en general mejor funciona para ambos tipos de acciones. También podemos apreciar como en el caso de las pruebas de Front y Right, es muy curioso como dependiendo del tipo de acción que estemos utilizando pase de la mejor a la peor prueba.

5.1.2.3. Pruebas con otras combinatorias de tipos de imágenes

Después de ver lo mal que funcionan los modelos generados en este simulador, vamos a pasar a realizar las pruebas finales en donde vamos a ir añadiendo a los que hasta ahora han resultado ser los mejores modelos distintos tipos de imágenes hasta llegar a usar todos los tipos disponibles con ambos tipos de acciones. No vamos a diferenciar entre tríos de tipos de imágenes u otras combinatorias tras ver los resultados que nos aportan.

Acción: Diferencial de Velocidad

Tabla 13. Resultados de CoppeliaSim para pruebas con otras combinatorias de tipos de imágenes

Cámaras	%A1	%A2	%A3	%A4	%A5	%A6	%A7	RMSE	AccSim
F, O, R	2.15%	1.66%	1.77%	1.43%	1.71%	2.18%	1.71%	0.02056	5.00%
L, O, R	1.81%	1.99%	4.53%	1.45%	3.06%	2.53%	1.73%	0.02747	0.00%
F, L O, R	2.97%	5.65%	2.32%	4.20%	2.14%	4.91%	2.32%	0.04652	0.00%
ALL	1.02%	1.83%	0.83%	1.30%	0.86%	1.18%	0.87%	0.01506	56.00%

Tras analizar el resto de pruebas realizadas en la Tabla 13, podemos llegar a la conclusión de que en el caso de querer o tener que trabajar con el diferencial de velocidad como acción, la mejor combinatoria de datos posibles es el conjunto total. Este modelo nos devuelve el mejor modelo con mucha diferencia. Sin embargo, y pese a que los resultados en entrenamiento sean extremadamente buenos llegando muchos de los porcentajes de error a bajar de un 1%, en simulación no obtenemos uso resultados acorde con lo que nos dice el entrenamiento. Esto puede ser debido al entorno de simulación, ya que en el caso de MuJoCo esto no pasaba.

A continuación vamos a analizar los resultados devueltos por las pruebas finales que toman el diferencial de posición como acción a predecir. Esta información será visible dentro de la Tabla 14:

Acción: Diferencial de Posición

Tabla 14. Resultados de CoppeliaSim para pruebas con otras combinatorias de tipos de imágenes

Cámaras	%A1	%A2	%A3	%A4	%A5	%A6	%A7	RMSE	AccSim
F, L, R	3.15%	2.52%	5.35%	3.53%	3.49%	4.39%	1.90%	0.05184	5.00%
ALL	2.12%	6.96%	2.35%	11.13%	2.12%	13.00%	1.99%	0.14074	0.00%

Al tener unos resultados tan malos en previas pruebas, hemos tenido que acortar muchísimo las posibles combinaciones de datos restantes. Vemos como al contrario que para el diferencial de velocidad, al usar todas los tipos de imágenes en lugar de mejorar y obtener unos resultados aceptables, empeoramos llegando a un 0% de acierto en simulación. Respecto a la otra prueba en donde usamos tanto Front como Left y Right, los resultados no mejoran los mejores resultados que ya teníamos previamente, quedándonos con ese 5% como mejor porcentaje de acierto en simulación tras usar este tipo de acción.

5.1.3. Comparativa para la tarea de coger una bola

Una vez hemos podido analizar los resultados obtenidos por medio de ambos entornos de simulación para una misma tarea, llegamos a la conclusión de que MuJoCo no funcionaba tan mal como nos podía hacer pensar inicialmente a la hora de comparar las funcionalidades de ambos entornos de simulación. Al principio se podía llegar a pensar que CoppeliaSim, al ser el entorno de facto para este tipo de problemas, podía llegar a funcionar mucho mejor que MuJoCo, el cual llevaba relativamente poco en el mercado y había sido diseñado por particulares en lugar de por una empresa privada.

Sin embargo, tras revisar los resultados devueltos por ambos entornos para la misma tarea, vemos como MuJoCo, pese a ser más sencillo y disponer de menos funcionalidades que CoppeliaSim, es capaz de trabajar mucho mejor.

Además de esto, también podemos analizar qué clase de datos funcionan mejor para esta primera tarea básica que hemos decidido escoger como candidata para comenzar con la realización de las pruebas de nuestro proyecto. De los tres tipos de acciones que hemos utilizado, no cabe duda de que la que mejor nos ha funcionado ha sido la que disponía de la información espacial en los tres ejes de coordenadas de la posición absoluta de la pinza. Al aplicar esta acción, el resto de articulaciones del robot fluyen siguiendo el movimiento marcado por la pinza. En cuanto a los tipos de imágenes que mejor funcionan para esta tarea inicial, en el caso de MuJoCo hemos visto como la combinación de una imagen desde arriba junto con una de un costado que englobe toda la escena es la que mejor funciona. Esto ha sido afirmado nuevamente por CoppeliaSim, en donde las pruebas que mejores resultados nos han devuelto han sido aquellas que disponían de la imagen desde arriba de la escena junto con una de uno de los dos costados.

Con esto llegamos a la conclusión de que para tareas tan básicas como esta, basta con aplicar dos cámaras sobre la escena. Una que obtenga las imágenes desde arriba y otra desde uno de los costados pero con cierta altura y distancia de modo que se no quede nada tapado por la escena o el propio robot. En cuanto a la acción a utilizar, la que mejor parece funcionar con mucha diferencia es la posición espacial de la pinza.

5.2. Otras tareas

Tras haber llegado a la conclusión de que MuJoCo-MetaWorld es el entorno de simulación que mejor funciona y saber cuáles son las mejores combinatorias de datos para la tarea de coger una pelota, hemos decidido generar dos tareas nuevas con el objetivo de probar nuestra conclusión inicial en ellas.

La primera tarea se trata de pulsar un botón colocado en frente del robot. Esta tarea es muy similar en cuanto a complejidad a la de coger una pelota. Sin embargo, tiene la característica de que mientras que al coger una bola el robot ha de aprender a ir hacia abajo, en esta tarea tiene que aprender a ir hacia adelante.

La segunda de nuestras tareas añade un punto de complejidad al tratarse de una tarea que realiza dos subtareas dentro de sí misma. Nuestro objetivo es el

de abrir una ventana. Para ello, el robot deberá aprender a llegar hacia una manilla para después de esto, empujarla y lograr abrir la ventana.

En las Figuras 19 y 20 se muestra como sería la realización de estas dos nuevas tareas:



Figura 19. Inicio y final de la tarea para pulsar un botón



Figura 20. Inicio, intermedio y final de la tarea para abrir una ventana

5.2.1. Tarea: Pulsar un botón

Tanto para esta como para la siguiente tarea hemos decidido seguir las mismas combinatorias de datos, al menos al principio, con las que habíamos trabajado con la tarea de coger una pelota de modo que pudiéramos compararlas entre sí. Los resultados han sido los que se muestran en la Tabla 15:

Tipo de prueba: Imágenes individuales

Tabla 15. Resultados de MuJoCo para pruebas con un solo tipo de imagen

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T	0.71%	6.13%	1.82%	0.39989	95.00%	61.81
C	41.04%	412.56%	15.12%	26.2994	55.00%	53.05
C2	0.81%	6.96%	1.72%	0.45156	95.00%	62.61
C3	0.88%	6.15%	5.17%	0.45120	84.00%	65.67
BG	2.14%	6.57%	3.12%	0.49625	68.00%	63.37
G	4.92%	6.65%	3.10%	0.71327	0.00%	-

Como podemos observar, se repite que la prueba individual en donde usemos el tipo Gripper sea la peor en lo que a simulación se refiere. Algo extraño que ocurre es el mal resultado para el tipo Corner, el cual en la pasada tarea obtenía los mejores resultados del apartado de entrenamiento.

Algo que vemos como se repite es el buen funcionamiento de la prueba individual que hace uso de Corner2, pues pese a no haber llegado a un 100% como ocurría en la pasada tarea, llega a un 95% siendo la prueba con mejores resultados junto con la que usa el tipo Top, siendo esta última la que mejores resultados nos devuelve tanto en entrenamiento como en simulación. Estas dos pruebas son las únicas que bajan la media de iteraciones de 63. De esta manera, vemos como una de las pruebas que anteriormente funcionaba muy mal, Top, para esta tarea se comporta mucho mejor. En la Tabla 16 vamos a evaluar las pruebas realizadas por pares de tipos de imágenes.

Tipo de prueba: Pares de imágenes*Tabla 16. Resultados de MuJoCo para pruebas con pares de tipos de imágenes*

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C	0.90%	5.24%	3.88%	0.37901	100.00%	61.05
T, C2	0.57%	5.82%	1.24%	0.37383	100.00%	60.70
T, C3	0.79%	6.42%	1.65%	0.42839	95.00%	64.98
C, C2	0.65%	5.38%	1.14%	0.34870	100.00%	60.65
C, C3	0.85%	5.39%	1.03%	0.35420	95.00%	65.94
C2, C3	0.69%	6.19%	1.89%	0.40343	95.00%	67.30
C, BG	0.91%	4.60%	1.27%	0.31108	100.00%	64.75

Si analizamos los resultados, vemos como hemos pasado a un mínimo de 95% de acierto en simulación tan solo pasándole un segundo tipo de imagen a nuestro modelo. Al igual que ocurría con la tarea de coger una pelota, la combinatoria entre Top y Corner2 resulta devolver unos resultados muy buenos, los cuales junto a los obtenidos con la combinatoria de Corner y Corner2 son los mejores. Este último modelo no era de los que mejor funcionaban para la pasada tarea, por lo que vemos como aquí sí que ha cambiado algo respecto a lo antes visto. Sin embargo, se siguen repitiendo patrones, pues la prueba de Top y Corner3, pese a no ser nada mala, es de las peores de este conjunto junto con Corner2 y Corner3, la cual tampoco era muy buena en la tarea anterior. En la Tabla 17 vamos a pasar a evaluar las pruebas por tríos de tipos de imágenes.

Tipo de prueba: Tríos de imágenes*Tabla 17. Resultados de MuJoCo para pruebas con tríos de tipos de imágenes*

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C, C2	0.61%	5.73%	1.21%	0.36914	100.00%	61.25
T, C, BG	0.93%	5.03%	1.64%	0.33961	100.00%	61.80
T, C2, BG	1.02%	5.01%	1.29%	0.33977	84.00%	66.94

En cuanto a la prueba por tríos, observamos como la prueba que mejores resultados nos devuelve es la compuesta por los tipos de imágenes Top, Corner y Corner2, mientras que la peor es la última de las pruebas realizadas. En la tarea de coger una pelota sucedía lo mismo, por lo que dependiendo de las siguientes últimas pruebas podremos fortalecer la conclusión inicial dada.

A continuación, vamos a ver los resultados de los modelos para el resto de combinatorias en la Tabla 18.

Tipo de prueba: Resto de combinatorias

Tabla 18. Resultados de MuJoCo para pruebas con otras combinatorias de tipos de imágenes

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C, C2, BG	0.78%	4.38%	1.47%	0.29506	100.00%	63.80
T, C, C2, C3 BG	0.67%	4.54%	1.04%	0.29785	100.00%	60.70
ALL	1.17%	4.64%	1.23%	0.32440	100.00%	68.00

Con estos últimos resultados vemos como a medida que añadimos más y más tipos de imágenes, más probabilidad de obtener un 100% en simulación tenemos. Sin embargo, no es esto lo que buscamos.

La mejor prueba de este conjunto es la que elimina únicamente el tipo Gripper del total de imágenes, lo cual tiene sentido tras comprobar su mal funcionamiento.

Sin embargo, vuelve a pasarnos lo mismo que antes. Hemos obtenido unos resultados muy buenos pero si empezamos a eliminar tipos de imágenes del entrenamiento, podemos llegar a mejorarlo e incluso igualarlo, como es el caso de la prueba en la que usamos Top y Corner o Corner y Corner2, en donde pese a obtener un peor RMSE como es lógico por no pasarle tantos tipos de imágenes a nuestro modelo, obtenemos unas métricas en simulación calcadas e incluso mejores que la prueba que acabamos de realizar con todos los tipos de imágenes quitando el tipo Gripper.

Otra vez, volvemos a llegar a la misma conclusión y ya vamos viendo como el nombre del tipo Corner2 empieza a ser algo habitual a la hora de hablar de los mejores resultados.

5.2.2. Tarea: Abrir una ventana

Esta tarea, al contrario que con las dos anteriores, tiene una mayor complejidad a la hora de su realización al necesitar de dos acciones por parte del robot en lugar de en una sola. En la primera, alcanzara la manilla para finalmente empujarla y abrir la ventana. Hemos seguido el mismo procedimiento de pruebas que con las pasadas tareas. Los resultados han sido los que se muestran en la Tabla 19:

Tipo de prueba: Imágenes individuales

Tabla 19. Resultados de MuJoCo para pruebas con un solo tipo de imagen

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T	2.38%	1.41%	3.70%	0.50891	35.00%	49.87
C	2.28%	2.36%	3.18%	0.46259	5.00%	143.20
C2	3.38%	2.32%	4.23%	0.61607	0.00%	-
C3	2.33%	1.63%	3.79%	0.51886	25.00%	45.01
BG	2.54%	4.29%	3.36%	0.53807	95.00%	65.53
G	4.02%	4.63%	5.28%	0.79206	15.00%	186.40

Podemos observar como el hecho de estar realizando una tarea con mayor complejidad hace que los resultados comiencen siendo más distintos que en las pasadas dos tareas. Algo que nos sorprende es el mal funcionamiento del modelo compuesto por el tipo Corner2, el cual hasta la fecha arrojaba los mejores resultados en la mayoría de las pruebas. Es posible que esto se deba a la complejidad de la tarea.

Otra cosa curiosa que podemos ver respecto a las anteriores tareas es también el funcionamiento de Top, Corner y Corner3, las cuales no llegan al 50% cuando anteriormente, salvo en un caso, sobrepasaban siempre el 70% llegando a un 100% en simulación en la mayoría de veces que las aplicábamos de manera individual.

En cuanto al modelo generado con el tipo BehindGripper, vemos como es la prueba que mejores resultados nos devuelve en simulación, alcanzando un 95%. Esto es curioso porque si analizamos las medidas de entrenamiento, ninguna es la mejor de entre todas. Además, a esto le tenemos que sumar que la media de

iteraciones en caso de acierto se encuentra 20 iteraciones por encima de las dos mejores. Sin embargo, esto puede ser engañoso, ya que las dos mejores medidas para esta última métrica se han obtenido en pruebas que no han llegado a alcanzar el 35% de acierto, por lo que muy seguramente hayan acertado las pruebas más cortas fallando las que necesitaban de más iteraciones. Así pues, consideramos que para las mejores pruebas, esta métrica se mantendrá muy seguramente entorno a las 60 iteraciones.

Finalmente, nos fijamos como otra vez, el modelo en el que usamos el tipo Gripper nos devuelve los peores resultados tanto en entrenamiento como en simulación, teniendo una media de iteraciones que casi triplica la obtenida por el tipo BehindGripper. Vamos a comprobar los resultados por pares de tipos de imágenes en la Tabla 20.

Tipo de prueba: Pares de imágenes

Tabla 20. Resultados de MuJoCo para pruebas con pares de tipos de imágenes

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C	2.87%	2.02%	4.27%	0.59227	95.00%	49.82
T, C2	3.50%	2.35%	4.43%	0.64259	20.00%	33.97
T, C3	3.12%	2.64%	4.71%	0.66150	15.00%	31.33
C, C2	3.25%	2.51%	4.61%	0.65417	95.00%	55.57
C, C3	9.36%	2.55%	3.38%	0.93785	84.00%	76.62
C2, C3	2.36%	2.18%	3.86%	0.53517	5.00%	29.00
C, BG	1.84%	2.77%	2.93%	0.42997	84.00%	67.78

Al añadir un segundo tipo de imagen a la prueba, vemos como los porcentajes de acierto en simulación crecen llegando a tener cuatro pruebas con valores por encima del 84%. Estas cuatro pruebas tienen un tipo de imagen en común: Corner. Si nos fijamos, siempre que entrenamos al modelo con pares de tipos de imágenes y esta se encuentra presente, los resultados en simulación acaban siendo buenos, algo curioso si tenemos en cuenta como su prueba individual se quedaba en un 5% de acierto. Sin embargo, también es cierto como la prueba individual del tipo Corner resultaba ser a su vez la que mejores medidas de entrenamiento tenía, por lo que es posible que este tipo sea un buen complemento para el resto de tipos de imágenes a la hora de realizar pruebas más complejas.

Los siguientes tipos de imágenes que parece que mejor funcionan con Corner son Top, Corner2 y BehindGripper. En el caso de Top, no solo alcanzamos un 95% de acierto en simulación, sino que lo hacemos bajando la media de iteraciones de las 50. En el caso de BehindGripper, obtenemos las mejores medidas en la fase de entrenamiento. En cuanto a Corner2, obtenemos también unos resultados muy buenos en la fase de simulación. Analizado esto, creemos que la prueba de tríos de tipos de imágenes que incluya Top, Corner y BehindGripper debería de aportar muy buenos resultados solventando los pequeños errores que tienen en simulación actualmente por separado. Estos resultados por tríos los vamos a ver en la Tabla 21.

Tipo de prueba: Tríos de imágenes

Tabla 21. Resultados de MuJoCo para pruebas con tríos de tipos de imágenes

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C, C2	2.28%	2.09%	3.46%	0.48809	95.00%	64.87
T, C, BG	1.37%	1.94%	2.33%	0.33242	100.00%	62.17
T, C2, BG	1.38%	1.43%	2.38%	0.32859	95.00%	59.25

Una vez obtenidos los resultados de las tres combinatorias que nos interesaban tras las pruebas de pares de tipos de imágenes, vemos como estábamos en lo cierto al decir que la prueba que tuviera Top, Corner y BehindGripper iba a ser una de las mejores. Por primera vez, alcanzamos un 100% en simulación con una media de 62 iteraciones, lo cual no está nada mal si tenemos en cuenta que hasta el momento esta última medida había sido tomada de pruebas que no superaban el 95%, por lo que es posible que se estuvieran librando de tener en cuenta aquellas pruebas que más iteraciones podían llegar a costar finalizar.

Otro patrón que vemos es el uso del conjunto de Top y BehindGripper, ya que en las dos mejores pruebas se encuentra presente. Vamos a probar que tal funcionaría el modelo si juntamos los cuatro tipos de imágenes con las que estamos trabajando aquí y seguimos añadiéndole tipos de imágenes hasta llegar al total.

Finalmente, vamos a analizar las pruebas realizadas para esta tarea con el resto de combinaciones de tipos de imágenes. Esto lo veremos en la Tabla 22.

Tipo de prueba: Resto de combinaciones

Tabla 22. Resultados de MuJoCo para pruebas con otras combinaciones de tipos de imágenes

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C, C2, BG	3.59%	1.91%	3.42%	0.53963	95.00%	59.78
T, C, C2, C3 BG	1.73%	1.72%	2.89%	0.40047	95.00%	62.90
ALL	1.49%	1.33%	3.09%	0.41071	95.00%	66.09

Una vez finalizadas todas las pruebas, nos fijamos como seguimos sin ser capaces de volver a obtener un 100% de acierto, por lo que llegamos a la conclusión de que el modelo generado a partir de los tipos Top, Corner y BehindGripper es el que mejor funciona al ser el único capaz de acertar todas las pruebas en simulación.

Sin embargo, si tenemos en cuenta que nuestro objetivo es minimizar la cantidad de datos que le pasamos al modelo, podríamos decir que las pruebas que utilizan las imágenes de tipo Top-Corner y Corner-Corner2 funcionan también muy bien, llegando ambas a un 95% con un tipo de dato menos además de bajar ambas de las 60 iteraciones de media para las pruebas acertadas en simulación, llegando el modelo formado por Top y Corner a bajar de las 50, valor muy bajo si tenemos en cuenta los demás obtenidos por pruebas que tienen un 95% de acierto en simulación como mínimo.

Así pues, podemos llegar a la conclusión de que en el caso de que la tarea posea un grado de complejidad mayor, las mejores configuraciones seguirían siendo el uso de una imagen desde arriba junto con una de un costado a cierta distancia y altura, ya que si nos fijamos, hemos utilizado las mismas combinaciones de parámetros para las tres tareas.

5.3. Análisis final de los resultados

Una vez después de haber finalizado con el apartado de análisis de los resultados obtenidos mediante las distintas pruebas realizadas para ambos entornos de simulación, hemos llegado a la conclusión de que no por el hecho de que un entorno de simulación o herramienta de ayuda para la programación parezca más sofisticada o sea la más utilizada tiene por qué funcionar mejor siempre. Esto lo hemos podido comprobar tras analizar las principales diferencias con las que nos hemos encontrado a la hora de trabajar con ambos entornos.

Mientras que en el caso de CoppeliaSim la generación de datos era más costosa en lo que a tiempo se refiere y menos personalizable, MuJoCo era capaz de generar el mismo número de datos o incluso más en mucho menor tiempo además de poder modificar las escenas o tareas a nuestro antojo de una forma mucho más sencilla y didáctica. Por otro lado, si nos fijamos en los resultados devueltos por ambos entornos para la misma tarea de coger una pelota de una base sólida, vemos como en el caso de MuJoCo, la simulación es mucho mejor que en CoppeliaSim, en donde aplicando el mismo procedimiento, el robot llegaba a repetir patrones haciendo así que no fuera capaz de alcanzar su objetivo al no fijarse en la escena que lo rodeaba.

En cuanto a nuestra problemática principal de conseguir obtener la mejor generalización de datos posibles para cualquier problema basado en predicción de trayectorias dentro del ámbito de la robótica industrial, hemos llegado a la conclusión que buscábamos de que podemos predecir trayectorias completas sin necesidad de llenar la escena de cámaras que recojan cada uno de los movimientos del robot desde distintos ángulos. Esto lo hemos podido comprobar tras ver como en la mayoría de las tareas que hemos generado, una vez pasábamos a trabajar con un modelo que recibía más de dos tipos distintos de imágenes, no conseguíamos mejorar los resultados obtenidos hasta el momento o en caso de mejorarlo, no merecía la pena la mejora respecto al incremento de datos con el que pasábamos a trabajar.

De esta manera, llegamos a la conclusión de que para la realización de cualquier tipo de tarea de la misma complejidad a las tres que hemos mostrado, nos bastaría con utilizar solamente dos tipos distintos de imágenes con las que trabajar. Esta configuración de imágenes, por lo que hemos podido comprobar, debería de estar basada en una imagen tomada desde encima de la escena y otra tomada desde uno de los costados del robot con cierta altura y distancia para que pueda verse la escena al completo, tal y como se pueden apreciar en las Figuras 21 y 22:



Figura 21. Imagen tomada desde arriba

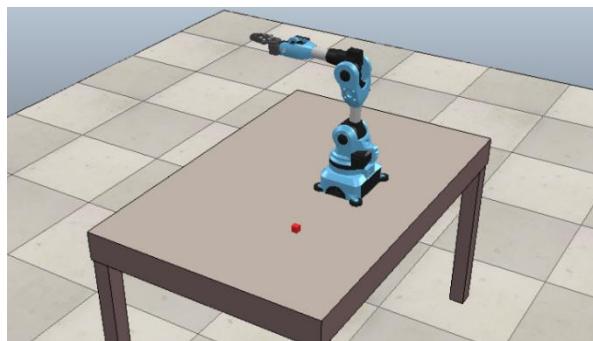


Figura 22. Imagen tomada desde un costado

Estos dos tipos de imágenes se complementan muy bien por el hecho de que con las dos podríamos tener la información de la posición espacial de todos los objetos que compongan la escena de una manera muy certera. De este modo, tendríamos la información visual necesaria para representar toda la escena y poder saber en todo momento donde se encuentran los objetos en ella.

En las Tablas 23 y 24 podemos observar la comparativa entre los resultados obtenidos para la configuración de tipos de imágenes en MuJoCo que consideramos que es la mejor y por tanto la que debería de tomarse como base para este tipo de problemas junto con la configuración obtenida con el total de los tipos de imágenes de las dos primeras tareas de menor complejidad con las que hemos trabajado en este proyecto. De este modo podremos apreciar cómo no solo por el hecho de tener más datos vamos a obtener unos mejores resultados siempre.

Tarea: Coger una pelota

Tabla 23. Comparativas entre el mejor modelo y el modelo con todas las imágenes

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C2	0.68%	1.01%	6.22%	0.10155	100.00%	38.75
ALL	0.70%	1.33%	4.84%	0.16564	75.00%	41.15

Tarea: Pulsar un botón

Tabla 24. Comparativas entre el mejor modelo y el modelo con todas las imágenes

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C2	0.57%	5.82%	1.24%	0.37383	100.00%	60.70
ALL	1.17%	4.64%	1.23%	0.32440	100.00%	68.00

En ambas tablas podemos apreciar como esta mejor configuración que hemos encontrado supera con creces los resultados obtenidos por el modelo que utiliza el total de tipos de imágenes disponibles. En la Tabla 23 no solo vemos como pasamos de un 75% a un 100% en simulación, sino que además de esto, tanto el RMSE como la media de acierto son mucho más favorables. En la Tabla 24, pese a tener un RMSE mayor y un acierto en simulación igual, vemos en la media de acierto como prácticamente nos ahorramos entre 7 y 8 iteraciones por prueba acertada, lo cual es más del 10% del total de cada prueba.

En el caso de estar manejando una tarea con mayor complejidad que las aquí vistas, por lo que hemos podido observar, otro tipo de imagen que se complementa muy bien a las dos propuestas es la que toma una imagen frontal de la escena, tal y como se muestra a continuación en la Figura 23:

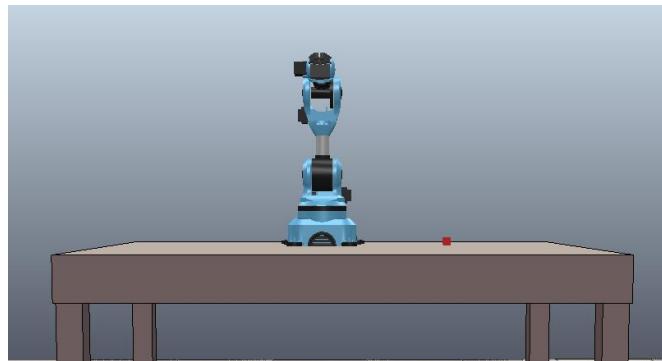


Figura 23. Imagen tomada desde en frente

Este tipo de imagen, sin embargo, sería solamente en caso de que la tarea fuera de una mayor complejidad a las que hemos presentado en este trabajo, ya que hemos podido ver como en la tarea que trata de abrir una ventana, siendo esta la más compleja de todas, no se requiere de la utilización de una imagen frontal al devolver unos resultados muy parecidos los cuales no compensan el incremento de datos. En la Tabla 25 podemos observar la comparativa entre esta mejor configuración para tareas con un punto mayor de complejidad respecto al resultado obtenido para la misma tarea con el total de tipos de imágenes.

Tarea: Abrir una ventana

Tabla 25. Comparativas entre el mejor modelo y el modelo con todas las imágenes

Cámaras	%Eje X	%Eje Y	%Eje Z	RMSE	AccSim	MediaAcierto
T, C2, BG	1.37%	1.94%	2.33%	0.33242	100.00%	62.17
ALL	1.49%	1.33%	3.09%	0.41071	95.00%	66.09

Para esta tarea, en la Tabla 25, vemos como tanto el RMSE, como el acierto en simulación, como la media de acierto son mucho mejores al aplicar esta configuración diseñada para tareas de mayor complejidad.

Finalmente, si nos centramos en el tipo de acción que debemos de utilizar, ya hemos visto como las acciones que incluyen las variaciones para todas las articulaciones del robot suelen funcionar peor al depender de demasiados factores. Además, cuantos más valores tengamos dentro de la representación vectorial de nuestra acción, más probabilidades de que alguno de los valores falle haciendo que el resto se vea afectado. Es por este motivo por el que las acciones con menos valores como es el caso de las End-Effector que tan solo disponen de la información de la última articulación del robot haciendo que el resto se muevan con ella son las más eficaces pese a tener mayor porcentajes de error en el entrenamiento. Esto último se debe a que no hay un solo camino para alcanzar el objetivo, ya que para el ejemplo de coger una pelota que se encuentre en frente sobre la mesa, el robot puede bajar en diagonal, avanzar recto y bajar o comenzar bajando para seguir recto después. Ninguno de los caminos es incorrecto, pero penalizará en las métricas de entrenamiento en el caso de que el camino no sea el esperado.

Capítulo 6

Conclusiones y líneas futuras

A lo largo de nuestro proyecto hemos intentado averiguar cuál era la mejor generalización de datos posible que podamos aplicar para cualquier tarea realizable por un sistema robótico.

Para poder lograr este objetivo, inicialmente decidimos utilizar dos de los entornos de simulación más utilizados dentro de este área. Además, sabiendo como existían distintas bibliotecas capaces de proporcionarnos todas las funcionalidades necesarias para poder trabajar, investigamos acerca de cuáles eran las más conocidas hasta la fecha. Es de este estudio inicial del que conseguimos llegar a los entornos y bibliotecas que hemos presentado y comentado a lo largo de la memoria.

Tras la correcta instalación de cada una de estas herramientas de trabajo, pasamos a la implementación de todo el apartado de generalización de datos, el

cual, al igual que todas las demás implementaciones, ha ido evolucionando con el tiempo adaptándose a las necesidades que nos iban surgiendo en todo momento y aplicando mejoras cada día. Gracias a esta primera implementación pudimos ver cuáles eran las principales diferencias entre ambos entornos de simulación a la hora de generar los datos, ya que con cada uno de los entornos éramos capaces de extraer distintos tipos de datos de distintas maneras.

Una vez tuvimos todos los datos disponibles, pasamos a la implementación de todo el apartado de entrenamiento del modelo. En este apartado vimos como pese a estar manejando distintos tipos de datasets procedentes de diferentes entornos, el tiempo de ejecución dentro del cual se encuentra el tiempo de preprocessamiento de los datos y el tiempo que le cuesta al modelo realizar la predicción, es muy parejo en todos los casos. Si bien es cierto que a medida que teníamos más datos de entrada estos tiempos se veían incrementados, no existía una diferencia como la que podíamos llegar a pensar inicialmente. Algo que pudimos apreciar dentro de este apartado, fue también ver como la métrica de entrenamiento utilizada, el RMSE, no iba a ser comparable entre todas las pruebas realizadas tal y como ya hemos mencionado a lo largo de la memoria.

Finalmente, realizamos la implementación de las simulaciones dentro de los entornos con los modelos que habíamos obtenido en la fase previa. Es en esta fase en la que pudimos ver las grandes diferencias que existían tanto entre los entornos de simulación por los resultados y tiempo de ejecución requerido para ejecutar cada prueba como por cada modelo generado gracias a las métricas de calidad que nos eran devueltas al finalizar cada simulación.

En definitiva, llegamos a una primera conclusión de que el entorno de simulación de MuJoCo, si bien es cierto que no es el más completo, es con el que mejores resultados hemos obtenido por mucha diferencia. Esto es algo que se nos hace curioso, ya que hasta la fecha, CoppeliaSim era el entorno de simulación que se utilizaba por defecto para la realización de cualquier tipo de simulación robótica.

En cuanto al análisis de los resultados obtenidos, pudimos ver como no por utilizar más datos de entrada vamos a ser capaces de obtener unos mejores resultados, ya que la configuración de datos con la que parece que se obtienen los mejores resultados para las tareas que hemos realizado disponen únicamente de tres tipos datos de entrada, de entre los cuales se encuentran dos tipos de imágenes

visibles en las Figuras 21 y 22 y una representación vectorial de la acción realizada en el estado anterior, la cual hemos podido comprobar también que cuantos menos valores tenga, funciona mejor. Esto es debido a que las pruebas en donde la representación vectorial de la acción a realizar tenía tantos valores como grados de libertad disponía el robot, los resultados eran mucho peores que en las pruebas en donde esta representación vectorial tenía solamente los valores de su última articulación por la cual se guiaban el resto de articulaciones del robot.

Así pues, llegamos a la conclusión de que para la correcta realización de cualquier tipo de tarea guiada por un brazo robótico, la generalización de datos que debemos de utilizar de aquí en adelante incluye:

- Una imagen desde encima del robot en donde se vea toda la escena.
- Una imagen de un costado con cierta altura en donde se vea toda la escena.
- Un vector de acciones con la información posicional de la pinza del robot.
- Adicionalmente, una imagen tomada de frente del robot.

En un futuro, me gustaría poder seguir trabajando en esta problemática teniendo en cuenta también tanto el tipo de red que utilicemos como los diferentes estados de datos que le pasemos a nuestra red al igual que intentar extraer otro tipo de datos del propio entorno que no sean accesibles en una primera instancia por el usuario. En resumen, quedaría dedicarle más tiempo al apartado de entrenamiento del modelo en el cual podríamos intentar profundizar más trabajando con diferentes arquitecturas de redes.

Además de todo esto, sería interesante intentar no solo generalizar el tipo de datos con el que trabajar como hemos hecho en este proyecto sino que también generar un modelo preentrenado mediante un amplio conjunto de tareas muy diferentes entre sí con el que poder predecir cualquier trayectoria a realizar tanto de una tarea ya aprendida durante el entrenamiento como de otra nunca antes vista o vista muy pocas veces. Es aquí donde entraríamos dentro del paradigma del Multi-task Learning, el cual puede ser muy interesante para un problema de estas características. Esto último requeriría de mucha más potencia computacional y de un mayor espacio de almacenamiento al necesitar generar datasets muy amplios de un gran conjunto de tareas.

Para terminar con el trabajo, otro punto que se me ocurre que podríamos mirar sería la realización del mismo problema aplicado a otra clase de robots que no tengan por qué encontrarse dentro de la categoría de brazos articulados. A parte de los robots no móviles como es el caso de estos últimos, existen también los robots móviles o bien por medio de rueda o bien por piernas al estar basados en una forma humanoide o animaloide. Sería interesante ver qué ocurriría al intentar trabajar con ellos por el hecho de disponer de una mayor escena o número de obstáculos sobre ella.

Bibliografía

- [1] “Robótica Industrial: ¿Para qué sirve y cómo funciona?”, *Escuela de Postgrado Industrial*, 2017 [En línea]. Disponible en: <https://postgradoindustrial.com/robotica-industrial-para-que-sirve-y-como-funciona/> [Accedido: 25-Jun-2022]
- [2] “Robots en la era de la revolución industrial: ¿Qué cambios sufrirán los puestos de trabajo?”, *PymeALDía*, 2017 [En línea]. Disponible en: <https://www.pimealdia.org/es/robots-revolucio-industrial-llocs-de-treball/> [Accedido: 25-Jun-2022]
- [3] “El ascenso de los robots y la automatización de los espacios de trabajo” *EuroNews*, 2022 [En línea]. Disponible en: <https://es.euronews.com/next/2022/06/22/el-ascenso-de-los-robots-y-la-automatizacion-de-los-espacios-de-trabajo> [Accedido: 26-Jun-2022]
- [4] “Los robots crearán 133 millones de puestos de trabajo a nivel mundial para 2022”, *ComputerWorld*, 2018 [En línea]. Disponible en: <https://www.computerworld.es/tendencias/los-robots-crearan-133-millones-de-puestos-de-trabajo-a-nivel-mundial-para-2022> [Accedido: 26-Jun-2022]
- [5] “El gasto mundial en TI crecerá un 3% en 2022 según Gartner”, *ComputerWorld*, 2022 [En línea]. Disponible en: <https://www.computerworld.es/tendencias/el-gasto-mundial-en-ti-crecerá-un-3-en-2022-según-gartner> [Accedido: 26-Jun-2022]
- [6] “Nuevos y más ágiles robots ejecutan tareas que antes sólo podían hacer los humanos”, *Los Angeles Times*, 2019 [En línea]. Disponible en: <https://www.latimes.com/espanol/vidayestilo/la-es-los-nuevos-y-mas-agiles-robots-aceleran-la-asuncion-de-tareas-que-antes-solo-realizaban-los-humanos-20190713-story.html> [Accedido: 27-Jun-2022]
- [7] “Así es el sistema de inteligencia artificial con el que Tesla quiere imponerse en el coche autónomo”, *BussinessInsider*, 2022 [En línea]. Disponible en: <https://www.businessinsider.es/sistema-ia-tesla-espera-liderar-coche-autonomo-1114311> [Accedido: 29-Ago-2022]

- [8] “La primera pizzería autónoma con robots pizzeros triunfa y abre su segundo local en París”, *Computerhoy*, 2021 [En línea]. Disponible en: <https://computerhoy.com/noticias/tecnologia/pazzi-pizzeria-autonoma-robot-pizzero-969261> [Accedido: 29-Jun-2022]
- [9] “Robot simulator CoppeliaSim”, *CoppeliaRobotics*, [En línea]. Disponible en: <https://www.coppeliarobotics.com/> [Accedido: 3-Oct-2021]
- [10] “MuJoCo - Advanced Physics Simulation”, MuJoCo, [En línea]. Disponible en: <https://mujoco.org/> [Accedido: 12-Nov-2021]
- [11] “RLBench: Robot Learning Benchmark”, Github, 2020 [En línea]. Disponible en: <https://github.com/stepjam/RLBench> [Accedido: 3-Oct-2021]
- [12] “Meta-World”, Github, 2019 [En línea]. Disponible en: <https://github.com/rlworkgroup/metaworld> [Accedido: 12-Nov-2021]
- [13] L.González, “Evaluando el error en los modelo de regresión”, AprendeIA, 2018 [En línea]. Disponible en: [https://aprendeia.com/evaluando-el-error-en-los-modelos-deregresion/#:~:text=Error%20cuadr%C3%A1tico%20medio%20\(RMSE\)&text=Como%20la%20ra%C3%ADz%20cuadrada%20de,RMSE%20indica%20un%20mejor%20ajuste](https://aprendeia.com/evaluando-el-error-en-los-modelos-deregresion/#:~:text=Error%20cuadr%C3%A1tico%20medio%20(RMSE)&text=Como%20la%20ra%C3%ADz%20cuadrada%20de,RMSE%20indica%20un%20mejor%20ajuste). [Accedido: 18-Jul-2022]
- [14] J.Brownlee, “What Is Meta-Learning in Machine Learning?”, MachineLearningMastery, 2021 [En línea]. Disponible en: <https://machinelearningmastery.com/meta-Learning-in-machine-Learning/> [Accedido: 24-Jul-2022]
- [15] L.Torrey, “Transfer Learning”, 2010, Google Scholar.
- [16] Y.Zhang, Q.Yang, 2017, “An overview of multi-task learning”, Google Scholar.
- [17] B.Dickson, “What is One-Shot Learning?”, TechTalks, 2020 [En línea]. Disponible en: <https://bdtechtalks.com/2020/08/12/what-is-One-Shot-Learning/> [Accedido: 24-Jul-2022]

- [18] C.Dilmegani, “What is Few-Shot Learning? Methods & Applications”, AIMultiple, 2020 [En línea]. Disponible en: <https://research.aimultiple.com/Few-Shot-Learning/> [Accedido: 24-Jul-2022]
- [19] S.Garcia-Bulle, “¿Qué es lifelong Learning y en qué consiste?”, Instituto Tecnológico de Monterrey, 2019 [En línea]. Disponible en: <https://observatorio.tec.mx/edu-news/aprendizaje-a-lo-largo-de-la-vida-lifelong-Learning> [Accedido: 24-Jul-2022]
- [20] S.Sun, 2013, “A survey of multi-view machine learning”, Google Scholar.
- [21] “RESNET18”, PyTorch, [En línea]. Disponible en: https://en.wikipedia.org/wiki/Residual_neural_network [Accedido: 26-Jul-2022]
- [22] “Residual Neural Network”, Wikipedia, [En línea]. Disponible en: <https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html> [Accedido: 26-Jul-2022]
- [23] TheLens, 2000 [En línea]. Disponible en: <https://www.lens.org/> [Accedido: 18-Jul-2022]
- [24] “OpenAI”, Github, 2016 [En línea]. Disponible en: <https://github.com/openai/gym> [Accedido: 12-Mar-2022]
- [25] “Franka Production”, Franka, 2020 [En línea]. Disponible en: <https://www.franka.de/> [Accedido: 12-Mar-2022]