Universidad Zaragoza
1542

# Trabajo Fin de Máster

# APRENDIZAJE REFORZADO PARA AGARRE Y MANIPULACIÓN DE ROBOTS

## REINFORCEMENT LEARNING FOR GRASPING AND HANDLING IN ROBOTS


Autor/es

Daniel Cubel Gálvez

Director/es

Rubén Martínez Cantín


Máster en Ingeniería Industrial


Escuela de Ingeniería y Arquitectura

2020

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

# DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

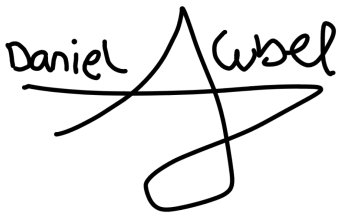
(Este documento debe entregarse en la Secretaría de la EINA, dentro del plazo de depósito del TFG/TFM para su evaluación).

D./Dª. Daniel Cubel Gálvez ,en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza, Declaro que el presente Trabajo de Fin de (Grado/Máster) Máster (Título del Trabajo)

Aprendizaje Reforzado para agarre y manipulación de robots

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, a 20 de noviembre de 2020



Fdo: Daniel Cubel Gálvez

# RESUMEN

El problema de las tareas robóticas en ambientes controlados está prácticamente resuelto, como demuestra el uso cada vez más común de robots en líneas de producción, dada la capacidad de poder controlar los parámetros del proceso de forma estricta. Sin embargo, en ambientes no controlados aparecen problemas al no poder controlar esos parámetros.

Una de las maneras de tratar los ambientes no controlados es mediante el aprendizaje automático, que permite aprender y adaptarse a cada circunstancia. Dentro del aprendizaje automático, el aprendizaje por refuerzo se basa en el aprendizaje por interacción, un agente actúa sobre el entorno y recibe una señal de recompensa que indica lo buenas o malas que son sus acciones. Aplicando el aprendizaje por refuerzo a las tareas robóticas en ambientes no controlados podemos superar las limitaciones y poder resolverlas.

En este TFM se ha diseñado, implementado y analizado una biblioteca de tareas robóticas en un simulador 3D como banco de pruebas para algoritmos de aprendizaje por refuerzo. Las tareas se han resuelto mediante el método de búsqueda de política, que consiste en parametrizar directamente el comportamiento del agente, en este caso las trayectorias. Para ello, el problema se plantea como uno de optimización, en el que la función a optimizar es la recompensa y los parámetros son los de la política. Así mismo, también se ha desarrollado una interfaz entre el simulador y los optimizadores.

Para resolver el problema de optimización es necesario un método de optimización global y que pueda funcionar con funciones cuya forma no se conoce. En este trabajo se ha implementado la optimización bayesiana, que consiste en crear un modelo de la función objetivo a partir de los datos obtenidos de la experiencia y de las suposiciones sobre las funciones objetivo que pueden ser válidas.

Las tareas se han diseñado para un robot comercial formado por un brazo antropomórfico. La simulación de las tareas se controla con PyRep, que utiliza CoppeliaSim como motor de simulación. Para la parte de optimización, la optimización bayesiana se implementa con BayesOpt y se compara con la evolución diferencial, implementada en el paquete de SciPy en Python.

# ABSTRACT

The problem of robotic tasks in controlled environments is practically solved, as it shows the increasingly common use of robots in production lines, given the ability to strictly control the process parameters. However, in uncontrolled environments, problems appear because it is not possible to control these parameters.

A way of dealing with uncontrolled environments is through machine learning, which allows to learn and adapt to each circumstance. Within machine learning, reinforcement learning is based in learning by interaction, an agent acts on the environment and receives a reward signal indicating how good or bad its actions are. By applying reinforcement learning to robotic tasks in uncontrolled environments, limitations can be overcome, and the tasks can be solved.

In this final master project, a robotic task library has been designed, implemented, and analysed in a 3D simulator as a benchmark for reinforcement learning. The proposed tasks have been resolved using the policy search method, which consists in directly parameterizing the agent's behaviour, in this case the robot's trajectory. The problem is presented as an optimization problem, in which the function to be optimized is the reward and the parameters are those of the policy. Also, an interface between the simulator and the optimizers has been developed.

To solve the optimization problem, a global optimization method and one that can work with unknown functions. In this work, Bayesian optimization has been implemented. It consists in creating a model of the objective function using data from experience and a priori assumptions about possible objective functions.

The tasks have been designed for a commercial robot formed by an anthropomorphic arm. The simulation is controlled with PyRep, which uses CoppeliaSim as a simulation engine. For the optimization part, Bayesian optimization is implemented with BayesOpt, and is compared with differential evolution, implemented in the Python library SciPy.

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS

# 1. INTRODUCCIÓN

## 1.1. MOTIVACIÓN

En la última década, el aprendizaje automático ha tenido un gran avance y los artículos académicos en los que se menciona han crecido exponencialmente. Sin embargo, el aprendizaje automático no es algo nuevo, si no que ya se hablaba de ello en la década de los 60 y desde entonces ha habido muchos avances teóricos y prácticos. Esto se debe a que los algoritmos de aprendizaje automático necesitan procesar una gran cantidad de datos y ahora disponemos de computadoras que son capaces de hacerlo.

Dentro del aprendizaje automático pueden distinguirse tres paradigmas:

- Aprendizaje supervisado: se basa en aprender a partir de un conjunto de entrenamiento etiquetado. Cada elemento del conjunto de entrenamiento tiene una etiqueta, la acción correcta que debería tomar, que normalmente es identificar a que categoría o situación corresponde el elemento. El objetivo de este agente sería generalizar sus acciones para responder correctamente en situaciones no presentes en el conjunto de entrenamiento.

- Aprendizaje no supervisado: se basa en aprender estructuras ocultas en conjuntos sin etiquetar, como por ejemplo agrupamientos.

- Aprendizaje por refuerzo: se basa en aprender que acciones debe tomar un agente mediante la interacción con el entorno con el fin de maximizar una recompensa.

El aprendizaje por refuerzo supone una revolución en muchos campos de la investigación y de la tecnología, y que puede ser utilizado para resolver multitud de problemas, como de robótica, sistemas de recomendación personalizado, diseño de estrategias financieras, juegos… en donde otros métodos fallan ya sea por la falta de estructura del entorno, la complejidad del espacio de soluciones posibles o el gran volumen de datos.

En robótica, el problema de las tareas robóticas en ambientes controlados, en los que se conoce la forma de los objetos que se quiere manipular, su posición y orientación, y la posición y orientación del elemento terminal está prácticamente resuelto. Por ejemplo, esto se puede observar en el uso de los robots en las líneas de producción. Sin embargo, en ambientes no controlados pueden aparecer problemas que antes no existían, ya que a priori no se conocen las características de los objetos que se encuentran en el entorno del robot. En nuestro caso, el aprendizaje por refuerzo nos va a servir para resolver una serie de tareas en lugar de resolver las tareas de forma tradicional.

Dentro del aprendizaje por refuerzo, las tareas robóticas pueden plantearse en forma de problemas de optimización, de forma que las dificultades que aparecen en ambientes no controlados pueden mitigarse. Como inconveniente, la función a optimizar en ese

caso suele ser desconocida, las llamadas funciones black-box, por lo que no conocemos sus características, pero si podemos obtener muestrear. Además, obtener muestras en estos casos puede ser un proceso muy costoso, ya que pueden ser procesos que produzcan daños irreversibles en los productos. En robótica, es normal la utilización de simuladores para verificar la correcta implementación de una tarea y corregir los posibles errores que aparezcan, con un coste despreciable y sin riesgo de posibles daños.

La optimización bayesiana es un método de optimización global que trata de aprender una función empleando el teorema de Bayes, estableciendo una creencia previa sobre la función a optimizar y combinándola con la experiencia para obtener una función posterior, guiando la búsqueda del óptimo con esta información. Esto permite reducir el número de iteraciones y trabajar con funciones black-box, por lo que puede aplicarse a problemas de simulación.

## 1.2. OBJETO

El objeto de este Trabajo Fin de Máster (TFM) es el diseño, implementación y análisis de una biblioteca de tareas robóticas de manipulación en un simulador 3D como banco de pruebas para algoritmos de aprendizaje por refuerzo.

Esta tarea incluye la definición y parametrización de las tareas a realizar, tanto de las acciones y políticas disponibles, como de la recompensas asociadas. El banco de pruebas diseñado se utiliza para evaluar las características y estudiar las mejoras de una biblioteca de aprendizaje por refuerzo basada en optimización bayesiana. Además, se debe programar una interfaz entre el simulador y los optimizadores.

Las tareas de la biblioteca deben ser sencillas y reproducibles, pero deberán tener la suficiente complejidad para tener más de una solución, para forzar a los optimizadores a encontrar todas las soluciones.

## 1.3. ALCANCE

En este apartado se detalla el alcance del trabajo fin de máster, desgranando las actividades realizadas y el grado de desarrollo.

En primer lugar, se llevó a cabo una serie de estudios previos, en los que se realizó tanto una revisión bibliográfica y del estado del arte del aprendizaje por refuerzo como de la optimización bayesiana.

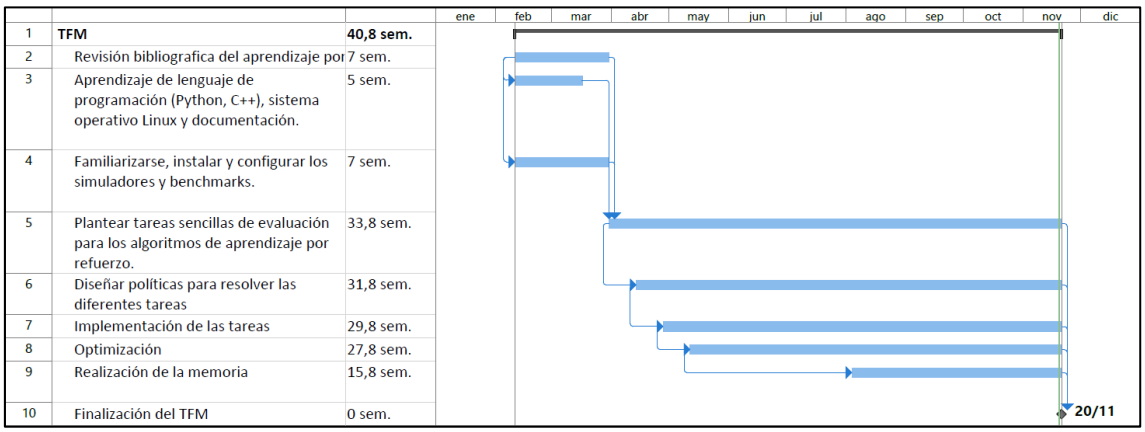
El siguiente aspecto que se trató fue la familiarización con las herramientas de programación y el sistema operativo que se utilizan a lo largo de todo el trabajo, que son el lenguaje de programación Python, el entorno de desarrollo de software PyCharm y el sistema operativo Ubuntu, una de las distribuciones de Linux. Así mismo se llevó a cabo una familiarización con la herramientas de simulación de robots, la plataforma de simulación robótica CoppeliaSim y el kit de herramientas de PyRep, que utiliza CoppeliaSim como motor de simulación. En este paso también se estudió el uso del benchmark y entorno de aprendizaje de RLBench, que funciona por encima de PyRep,

pero se terminó descartando su uso ya que con PyRep se podían cumplir las especificaciones necesarias para el desarrollo del TFM. Por último, también se estudiaron las herramientas de optimización, la biblioteca de optimización bayesiana BayesOpt y la función que implementa la evolución diferencial de SciPy.

Una vez ya se habían cubierto las necesidades teóricas y de software, el siguiente paso fue diseñar las tareas para la biblioteca. Esta tarea tenía dos fases: una primera fase de concepción, en la que se decidía que tareas se iban a implementar. Una segunda fase en la que se debía decidir como parametrizar la política y la función de recompensa. Las tareas debían ser fácilmente reproducibles para los algoritmos de aprendizaje por refuerzo.

Después del diseño, la siguiente actividad fue la implementación de las tareas, que consistía en concretar el diseño de las tareas en una escena del simulador y en un programa de Python que permitiera ejecutar la tarea. Esta actividad comprendía también la prueba de las tareas, revisando la parametrización de la política y la adecuación de la función de recompensa.

Una vez se habían diseñado e implementado las tareas, se llevó a cabo la optimización de las tareas para obtener las soluciones de las tareas. Esta actividad comprende escribir el código de interfaz entre las tareas y las biblioteca de aprendizaje, así como el análisis de los resultados obtenidos en la optimización.




| | | |
|---|---|---|
| 1 | **TFM** | **40,8 sem.** |
| 2 | Revisión bibliografica del aprendizaje por | 7 sem. |
| 3 | Aprendizaje de lenguaje de programación (Python, C++), sistema operativo Linux y documentación. | 5 sem. |
| 4 | Familiarizarse, instalar y configurar los simuladores y benchmarks. | 7 sem. |
| 5 | Plantear tareas sencillas de evaluación para los algoritmos de aprendizaje por refuerzo. | 33,8 sem. |
| 6 | Diseñar políticas para resolver las diferentes tareas | 31,8 sem. |
| 7 | Implementación de las tareas | 29,8 sem. |
| 8 | Optimización | 27,8 sem. |
| 9 | Realización de la memoria | 15,8 sem. |
| 10 | Finalización del TFM | 0 sem. |

*Fig. 1.1 – Cronograma del trabajo fin de máster*

## 1.4. METODOLOGÍA Y HERRAMIENTAS

La metodología seguida para el desarrollo del trabajo ha sido la del desarrollo ágil de software, considerando más importante el tener una tarea funcionando e ir implementado funcionalidades y características conforme se iba avanzando a tener todas la funcionalidad.

Las herramientas de simulación de robots y de optimización tienen un capítulo propio más adelante, por lo que no se van a explicar en este apartado.

Todo el trabajo se ha desarrollado en el sistema operativo Ubuntu. La mayor parte ha sido de forma remota controlando el ordenador del laboratorio con TeamViewer o en una máquina virtual en VirtualBox.

Toda la programación en Python se ha realizado en el entorno de desarrollo de software de PyCharm. Se ha decidido utilizar este entorno ya que la cuenta de estudiante de GitHub proporciona la versión profesional.

La interfaz entre el simulador y los optimizadores se ha realizado en Python, ya que PyRep está programado en Python, BayesOpt tiene una interfaz para utilizar Python y SciPy es una biblioteca de Python.

También se ha utilizado Python para generar las gráficas de las optimizaciones y analizar los resultados obtenidos mediante la biblioteca de matplotlib.

Para el control de versiones del código desarrollado se han utilizado Git y GitHub. El repositorio con el código desarrollado en el trabajo puede encontrarse en el siguiente enlace:

https://github.com/deedaniel/TFM

En el ANEXO D se adjunta un pequeño manual de la librería.

# 2. APRENDIZAJE POR REFUERZO

Para llevar a cabo la redacción de este capítulo se ha llevado a cabo una revisión del estado del arte del aprendizaje por refuerzo con el objetivo de cubrir las necesidades teóricas del trabajo fin de máster. La mayor parte del capítulo está redactado a partir del libro de aprendizaje por refuerzo de Sutton y Barto [1]. Este libro es una introducción al aprendizaje por refuerzo y cubre las ideas clave, del que se han escogido los capítulos que cubren el tema del trabajo. Se refiere esta obra al lector para una mayor profundización en la materia. Así mismo, también se han utilizado los apuntes de la asignatura de Redes Inteligentes del Máster de Ingeniería Informática de la parte de aprendizaje por refuerzo y los apuntes de un curso de aprendizaje automático de la universidad de Berkeley [2].

## 2.1. INTRODUCCIÓN

El aprendizaje por refuerzo es una forma de aprendizaje automático basado en la interacción de un agente con su entorno. El agente tiene que aprender que acciones tomar para maximizar la recompensa mediante prueba y error. En algunos casos, las acciones que se toman no solo afectan a la recompensa inmediata si no a la de situaciones futuras, por lo que aprender que acciones tomar para maximizar la recompensa no es trivial.

El aprendizaje por refuerzo es a la vez una definición de un problema, un conjunto de métodos para solucionar el problema y un subcampo que estudia el problema y el conjunto de métodos para solucionarlo. Es importante diferenciar a que concepto nos referimos al utilizar la expresión aprendizaje por refuerzo.

Para la formalización del problema de aprendizaje por refuerzo es importante tener en cuenta tres aspectos: la sensación, las acciones y los objetivos. El agente debe ser capaz de conocer el estado del entorno que le rodea y tomar acciones que le afecten. Así mismo, el agente debe tener un objetivo u objetivos en relación con el estado del entorno.

Una de las características particulares del aprendizaje por refuerzo es la necesidad de equilibrar la explotación y la exploración. Por un lado, nos interesa que el agente tome las acciones que ha probado y que mayor recompensa le han dado, y, por otro lado, para descubrir esas acciones debe probar acciones que no haya probado antes. Es decir, debe explotar las acciones que ya conoce para obtener más recompensa, pero también debe explorar para descubrir acciones que le devuelven una recompensa mayor que las acciones que ya conoce. El agente no puede elegir una de las dos, ya que si solo explotamos puede haber acciones que no hemos probado que darían una mayor recompensa y si solo exploramos no se maximiza la recompensa obtenida.

Otra característica del aprendizaje por refuerzo es que considera el problema global de un agente orientado a un objetivo que actúa en un entorno con incertidumbre, en

contra de otras aproximaciones que consideran subproblemas sin considerar como encajarán después.

El agente de un problema de aprendizaje por refuerzo es completo, interactivo y orientado a un objetivo, pero esto no quiere decir que el agente siempre sea un organismo completo, como por ejemplo un robot, sino que también puede ser un componente de un sistema más grande. En ese caso, el entorno del agente es el resto de sistema e interactúa indirectamente con su entorno.

Otro aspecto interesante del aprendizaje por refuerzo es su interacción con otros campos de la ingeniería y de la ciencia. El aprendizaje por refuerzo forma parte junto al aprendizaje automático y la inteligencia artificial de la tendencia hacia una mayor integración de la estadística, la optimización y otras cuestiones matemáticos.

## 2.2. ELEMENTOS DEL APRENDIZAJE POR REFUERZO

Más allá del agente y del entorno, se puede considerar cuatro elementos en un problema de aprendizaje por refuerzo: la política, la recompensa, la función del valor y, opcionalmente, el modelo del entorno.

- La política define como debe comportarse el agente. Es un mapeo de los estados a las acciones: si el agente está en un estado $s_t$, la política define que acción debe tomar el agente. La política puede ser determinista o estocástica, definiendo una probabilidad para cada acción.

- La recompensa es una señal escalar de realimentación que indica cómo se comporta el agente en un paso. Define el objetivo de un problema de aprendizaje por refuerzo, maximizar la recompensa acumulada. Indica como de buenas o malas son las acciones que toma el agente y se utiliza para modificar la política. Si una acción devuelve una mala recompensa, la política se puede modificar para que el agente tome otras acciones en el futuro.

  En un problema de aprendizaje por refuerzo la meta u objetivo se formula en términos de señales de recompensa. Es decir, que el objetivo del problema se debe poder traducir a una función de recompensa.

- La función de valor especifica que comportamiento es bueno en el largo plazo. El valor de un estado es la recompensa que puede esperarse obtener en el futuro desde ese estado. Mientras que la recompensa determina la conveniencia inmediata de un estado, la función de valor determina la conveniencia en el largo plazo. Un estado puede devolver una recompensa baja, pero tener un valor alto, porque los estados siguientes devuelven una recompensa alta, y viceversa.

- El modelo del entorno se utiliza para simular el comportamiento del entorno, de forma que pueden hacerse suposiciones sobre cómo se comportará el entorno al tomar una acción, como, por ejemplo, predecir la recompensa y el estado siguientes. Los modelos del entorno se utilizan para planificación.

## 2.3. PROCESOS DE DECISIÓN DE MARKOV

### 2.3.1. Interacción agente-entorno

Los procesos de decisión de Márkov (Markov Decission Process, MDP) son una formalización clásica de los problemas de decisión secuenciales y sirven como marco de referencia para los problemas de interacción con un objetivo, como el aprendizaje por refuerzo.

Como ya hemos comentado en el apartado anterior, en los problemas de aprendizaje por refuerzo y, por tanto, también en los procesos de decisión de Márkov, tenemos un agente, que aprende y toma decisiones, un entorno, que engloba todo aquello que es exterior al agente, y una señal de recompensa, que el agente busca maximizar con las acciones que toma. Desde una perspectiva ingenieril y de control, el agente sería el controlador, el entorno la planta y las acciones serían las señales de control.

Los elementos del problema de aprendizaje por refuerzo interactúan constantemente, el agente ejecuta acciones que afectan al entorno, que presenta nuevas situaciones al agente. Como consecuencia de sus acciones, el agente recibe una recompensa.

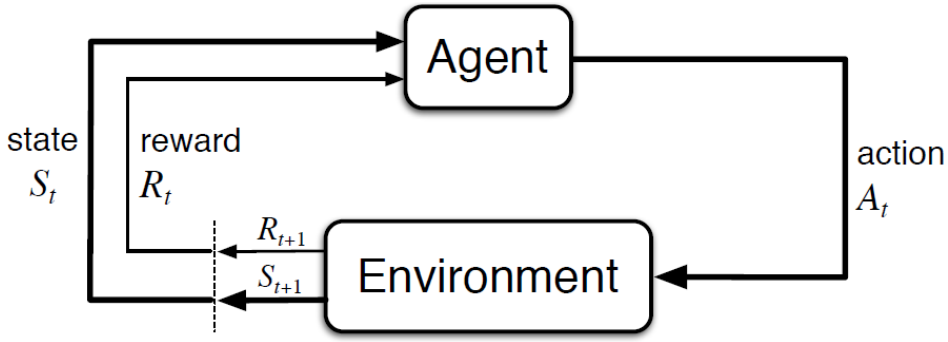



*Fig. 2.1 – Interacción entre el agente y el entorno [1]*

En un proceso de Márkov finito, los conjuntos de acciones, estados y recompensas tienen un número de elementos finito. En la Fig. 2.1 se muestra la interacción entre el agente y el entorno en un proceso de decisión de Márkov. En cada paso, el agente recibe una representación del estado del entorno, $S_t \in \mathcal{S}$, y basado en esa representación ejecuta una acción $A_t \in \mathcal{A}(s)$. En el siguiente paso, como consecuencia de la acción, el agente recibe una recompensa $R_{t+1} \in \mathcal{R}$ y recibe la nueva representación del estado del entorno, $S_{t+1}$. Como consecuencia de la interacción entre el agente y el entorno se genera una secuencia de esta forma:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$$

En el caso de procesos de Márkov finitos, el estado del entorno y la recompensa se pueden describir mediante variables aleatorias con una distribución del probabilidad aleatoria que depende únicamente del estado anterior y de la acción seleccionada. Por tanto, para un valor concreto de esas variables, $s' \in \mathcal{S}$ y $r \in \mathcal{R}$, existe una probabilidad de que el estado del entorno y la recompensa tomen esos valores en un paso t, dados unos valores concretos del estado y acción anterior, $s \in \mathcal{S}$ y $a \in \mathcal{A}$.

$$p(s', r|s, a) \stackrel{\text{def}}{=} \Pr(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a)$$

Esta probabilidad define la dinámica del proceso de decisión de Márkov. Esta función es coherente con las propiedades de la probabilidad condicional, es decir, que para unos valores concretos del estado y acción anterior, $s \in \mathcal{S}$ y $a \in \mathcal{A}$, se cumple que:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1$$

En un proceso de decisión de Márkov, la probabilidad de alcanzar un estado posterior y recibir una recompensa depende solo del estado y de la acción actual, pero no de los estados y acciones anteriores. Esto no es una restricción del proceso si no del estado. El estado debe cumplir la propiedad de Márkov, según la cual el estado posterior depende solo de estado actual.

$$p(s_{t+1}|s_t) = p(s_{t+1}|s_1, \dots, s_t)$$

A partir de la función de la dinámica del proceso de decisión de Márkov se puede calcular todas las demás propiedades del entorno, como por ejemplo la probabilidad de transición entre estados:

$$p(s'|s, a) \stackrel{\text{def}}{=} \Pr(S_t = s' \mid S_{t-1} = s, A_{t-1} = a) = \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

También se puede calcular la recompensa esperada en función del estado y la acción:

$$r(s, a) = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$$

Y también la recompensa esperada en función de estado, la acción y estado posterior:

$$r(s, a, s') = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)}$$

El marco de referencia de los procesos de Márkov para los problemas del aprendizaje de refuerzo es abstracto y flexible. Por ejemplo, los pasos no tienen por qué ser intervalos fijos, si no que pueden referirse a distintas fases de tomas de decisiones. Las acciones pueden ser tanto controles de bajo nivel (p. ej. el voltaje aplicado a los motores de un robot) como decisiones de alto nivel (p. ej. decidir qué comer hoy o qué movimiento de ajedrez hacer). De igual manera puede ocurrir con el estado y las acciones.

Es importante destacar que los límites entre el agente y su entorno no tienen por qué ser, por ejemplo, los límites físicos de un robot. Normalmente, las partes sobre las que el agente no tiene un control directo son consideradas parte del entorno. De igual manera, la recompensa se considera como externa al agente, aunque puede ser que

este conozca que se computa como resultado de su estado y de sus acciones, pero no puede actuar directamente sobre ella.

Los procesos de decisión de Márkov son una buena abstracción de los problemas de aprendizaje por refuerzo, ya que cualquier problema se reduce a tres señales que pasan de un lado a otro entre el agente y el entorno.

El utilizar los procesos de decisión de Márkov para los problemas de aprendizaje por refuerzo nos permite realizar un análisis más amplio de los elementos del aprendizaje por refuerzo comentados en el apartado 2.2.

### 2.3.2. Utilidad y episodios

Como ya hemos comentado, el objetivo de un problema de aprendizaje por refuerzo se formula en términos de la recompensa. En cada paso de tiempo de tiempo, el entorno le devuelve al agente una recompensa numérica y el objetivo es maximizar la recompensa acumulada. Formalmente, para una secuencia de recompensas $R_{t+1}, R_{t+2}, R_{t+3}$ … se puede definir la utilidad esperada como una función de la secuencia de recompensas. La más sencilla sería la suma de recompensas, en la que T sería el último paso.

$$G_T \stackrel{\text{def}}{=} R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

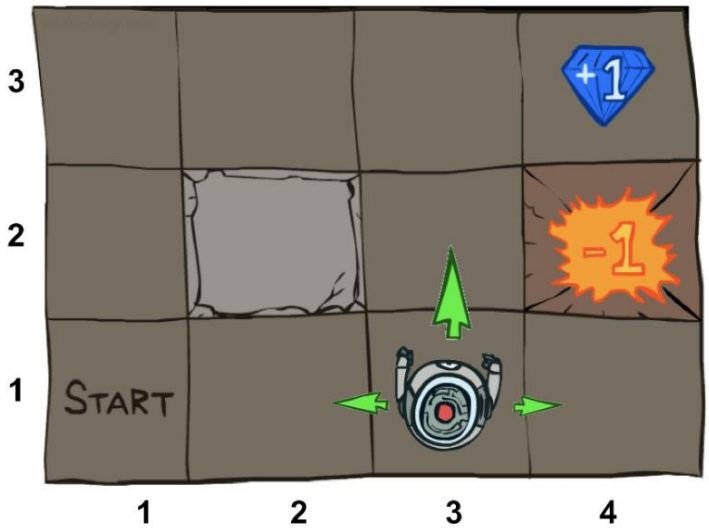



*Fig. 2.2 – Ejemplo de un problema de laberinto [2]*

Por ejemplo, en el problema de la Fig. 2.2, si el agente avanza dos casillas hacia adelante y luego una casilla a la derecha, la utilidad esperada según la ecuación de arriba sería de 1. Sin embargo, si solo avanza uno hacia adelante y luego uno a la derecha, la utilidad esperada sería de -1.

Esta aproximación supone que la tarea es episódica. Las tareas episódicas tienen un estado terminal, por lo que su duración es finita. Los episodios se definen como las

interacciones entre el agente y el entorno desde el estado inicial al estado terminal. Al terminal un episodio se produce un reinicio a un estado inicial. Todos los episodios terminan al llegar a estado terminal, y tendrán diferentes recompensa según como el agente se haya comportado durante el episodio. Por ejemplo, el ajedrez es una tarea episódica. En este caso, cada episodio sería una partida y el estado final sería su finalización.

Sin embargo, esta aproximación no es válida para las tareas continuas. Las tareas continuas son aquellas que por su naturaleza no tienen un final identificable y nunca terminan. En este caso, el último paso T estaría situado en el infinito y la utilidad esperada también podría hacerse infinita. Para el caso de tareas continuas, se plantea una utilidad descontada, de forma que el agente debe maximizar la suma de recompensas descontadas.

$$G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = R_{t+1} + \gamma G_{t+1} = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

De esta forma, aunque la tarea sea continua, la utilidad esperada tiene un valor finito. $\gamma$ es la tasa de descuento, donde $0 < \gamma < 1$. La tasa de descuento se puede ver como la preferencia sobre recompensas actuales o futuras. Si la tasa de descuento es próxima a cero, el agente solo busca maximizar las recompensas cercanas en el tiempo, mientras que si es próxima a uno el agente busca maximizar también recompensas más alejadas en el tiempo. La utilidad descontada también puede utilizarse en tareas periódicas. Por ejemplo, con una tasa de descuento de 0.8, si en el ejemplo de la Fig. 2.2 el agente avanza dos casillas hacia adelante y una casilla hacia la derecha, la utilidad esperada sería de 0.64.

### 2.3.3. Política y función valor

La política de un agente de un problema de aprendizaje por refuerzo define como su comportamiento. Formalmente, es un mapeado entre el estado y la probabilidad de tomar una de las acciones posibles. Si un agente sigue una política $\pi$, entonces $\pi(a|s)$ es la probabilidad de tomar la acción $a$ desde el estado $s$.

Las funciones de valor se definen respecto a la política que sigue el agente. Estas funciones estiman como de bueno es un estado para el agente en términos de la futura recompensa que puede obtener desde ese estado. Esta recompensa depende de las acciones que tome.

La función de valor de un estado $s$ bajo una política $\pi$ se denota $v_\pi(s)$ y es la utilidad esperada empezando en $s$ y siguiendo la política $\pi$. Para los procesos de decisión de Márkov se puede definir $v_\pi$ como:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \forall s \in \mathcal{S}$$

En la que $\mathbb{E}_\pi[\cdot]$ denota el valor esperado de una variable aleatoria dado que el agente sigue una política $\pi$. El valor de la función para el estado final es cero. $v_\pi$ se denomina la función de estado-valor para la política $\pi$.

De forma similar, se puede definir el valor de tomar la acción $a$ desde el estado $s$ siguiendo la política $\pi$, denotado como $q_\pi(s,a)$, como la utilidad esperada partiendo desde el estado $s$, tomar la acción $a$ y después seguir la política $\pi$. $q_\pi$ se denomina la función de acción-valor para la política $\pi$.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a], \forall s \in \mathcal{S}$$

Las funciones de estado-valor y acción valor también satisfacen relaciones de recursividad de forma similar a la utilidad.

$$\begin{aligned} v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(s')|S_t = s] \end{aligned}$$

Podemos seguir el mismo procedimiento para la función acción-valor obteniendo:

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(s',a')|S_t = s, A_t = a]$$

La función de estado-valor y de acción-valor pueden relacionarse mediante la propiedad de las distribuciones discretas:

$$\mathbb{E}_Z[f(z)] = \sum_{z \in Z} p(z) f(z)$$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \cdot q_\pi(s,a)$$

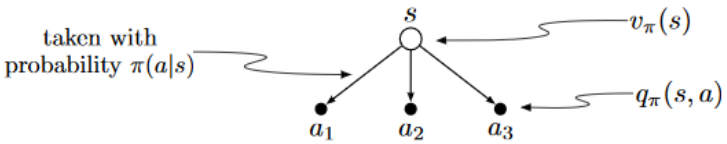



*Fig. 2.3 – Diagrama de la función de estado-valor [1]*

Expandiendo los valores esperados se puede ver que:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \cdot q_\pi(s,a)$$

$$q_\pi(s,a) = \sum_{s \in \mathcal{S}} p(s'|s,a) \cdot (r(s,a,s') + \gamma v_\pi(s'))$$

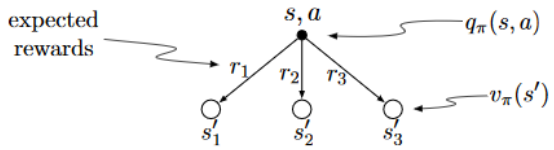



*Fig. 2.4 – Diagrama de la función de acción-valor [1]*

Combinando las ecuaciones previas queda:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \cdot \left( \sum_{s \in \mathcal{S}} p(s'|s,a) \cdot (r(s,a,s') + \gamma v_\pi(s')) \right)$$

$$q_\pi(s,a) = \sum_{s \in \mathcal{S}} p(s'|s,a) \cdot \left( r(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') \cdot q_\pi(s',a') \right)$$

Estas ecuaciones se conocen como las ecuaciones de Bellman. La primera ecuación expresa la relación entre el valor del estado actual y de los estados siguientes.

### 2.3.4. Política y función valor óptimas.

Encontrar una solución para un problema de aprendizaje por refuerzo significa encontrar una política que maximice la recompensa acumulada. La función de estado-valor define la calidad de una política. Para los procesos de decisión de Márkov finitos puede formalizarse de la siguiente forma: una política $\pi$ es igual o mejor que una política $\pi'$ si los valores de la función de estado-valor de $\pi$ son mayores o iguales a los de la función de estado-valor de $\pi'$ para todos los estados.

$$\pi \geq \pi' \Leftrightarrow v_\pi(s) \geq v_{\pi'}(s), \forall s \in \mathcal{S}$$

Por tanto, siempre hay una política que es igual o mejor que todas las políticas, la política optima. Aunque puede haber más de una política optima, estas se denotan como $\pi_*$. Las políticas optimas tienen una función de estado-valor óptima, $v_*$, así como una función de acción-valor óptima, , $q_*$, que se definen como:

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in \mathcal{S}$$

$$q_*(s,a) = \max_\pi q_\pi(s,a)\,, \forall s \in \mathcal{S}\ y\ \forall a \in \mathcal{A}$$

Entonces, la política óptima se puede considerar, según sea determinista o estocástica, como:

$$a = \pi_*(s) = \arg\max_{a \in \mathcal{A}} q_*(s,a)$$

$$\pi_*(a|s) = \begin{cases} 1\ si\ a = \arg\max_{a \in \mathcal{A}} q_*(s,a) \\ 0\ en\ cualquier\ otro\ caso \end{cases}$$

De igual forma que con $v_\pi$ y $q_\pi$, podemos operar con $v_*$ y $q_*$:

$$v_*(s) = \max_a q_*(s,a)$$

$$q_*(s,a) = \sum_{s \in \mathcal{S}} p(s'|s,a) \cdot (r(s,a,s') + \gamma v_*(s'))$$

Combinando ambas ecuaciones queda:

$$v_*(s) = \max_a \sum_{s \in \mathcal{S}} p(s'|s,a) \cdot (r(s,a,s') + \gamma v_*(s'))$$

$$q_*(s,a) = \sum_{s \in \mathcal{S}} p(s'|s,a) \cdot (r(s,a,s') + \gamma \max_a q_*(s,a))$$

Lo que se conoce como las ecuaciones de optimalidad de Bellman.

## 2.4. POSIBLES MÉTODOS DEL APRENDIZAJE POR REFUERZO

Una vez se ha introducido el aprendizaje por refuerzo y se han presentado los procesos de decisión de Márkov como el marco de referencia para plantear los problemas de RL, el siguiente paso es estudiar los posibles métodos para solucionar los problemas de aprendizaje por refuerzo.

Atendiendo a los elementos del aprendizaje por refuerzo explicados anteriormente, se podrían plantear tres métodos:

- Aprendizaje por refuerzo basado en la función valor.
- Aprendizaje por refuerzo basado en búsqueda de política.
- Aprendizaje por refuerzo actor-crítico.

En el aprendizaje basado en la función de valor se parametriza y se aprende la función de valor a partir de la experiencia, y con ella se calcula la política. En el aprendizaje basado en búsqueda de política no se aprende ninguna función de valor si no que se parametriza y se aprende directamente la política. Existe una tercera aproximación, el aprendizaje actor-critico en el que se parametriza y se aprende tanto la función de valor como la política. En este trabajo se implementa la búsqueda de política.

### 2.4.1. Búsqueda de política

El método de búsqueda de política se basa en parametrizar la política con un vector de parámetros $\theta$. De forma que se puede escribir:

$$\pi_\theta(s,a) = p(a|x,\theta)$$

Las ventajas que presenta la búsqueda de política son una mejor convergencia, efectividad en espacios de estados de grandes dimensiones o continuos y la posibilidad de poder aprender políticas estocásticas. Como desventajas presenta la posibilidad de convergencia local en lugar de global y la ineficiencia y gran variabilidad, ya que encontrar la política optima suele ir ligado a un gran número de evaluaciones.

En la búsqueda de política, el objetivo es encontrar los mejores parámetros $\theta$. Para ello es necesario definir una medida del rendimiento con respecto a los parámetros, $J(\theta)$. Para una tarea episódica y con factor de descuento 1, podemos definir $J(\theta)$ como:

$$J(\theta) = \mathbb{E}_{\tau}\left[\sum_{t=0}^{T} r(s_t, a_t, s_{t+1})\right] = \mathbb{E}_{p(x_0)\cdot\prod_{t=0}^{n}\pi(a_i,s_i)p(s_{i+1}|s_i,a_i)}\left[\sum_{t=0}^{T} r(s_t, a_t, s_{t+1})\right]$$
$$= \mathbb{E}_{\mathcal{P}}\left[\sum_{a_{0:T}}\left(\sum_{t=0}^{T} r(s_t, a_t, s_{t+1})\right)\pi(a_{0:T}|s_{0:T})\right]$$

Esta definición no es más que la utilidad esperada asociada a la política $\pi_\theta(s, a)$, es decir, el valor de la función valor-estado para el estado inicial, $v_{\pi_\theta}(s_0)$.

La búsqueda de política es un problema de optimización, ya que encontrar los mejores parámetros $\theta$ equivale a maximizar $J(\theta)$:

$$\theta = \arg\max_{\theta} J(\theta)$$

Los métodos que se utilizan para resolver el problema de optimización se suelen clasificar en dos grupos, los que están utilizan el gradiente de la política y los que no.

#### *2.4.1.1. Métodos que utilizan el gradiente.*

Los algoritmos basados en el gradiente de la política se utilizan cuando la política es diferenciable, ya que en general presentan mayor eficiencia. Un método muy utilizado es el algoritmo de ascenso de gradiente. Los algoritmos de descenso de gradiente buscan un mínimo local en $J(\theta)$ ascendiendo en el gradiente de la política:

$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\theta)$$

Donde $\nabla_\theta J(\theta)$ es el gradiente de la política y $\alpha$ es un parámetros del tamaño del paso.

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

#### *2.4.1.2. Métodos que no utilizan el gradiente*

Dentro de los métodos que no utilizan el gradiente existen muchas aproximaciones, como por ejemplo, el algoritmo hill climbing o búsqueda local, algoritmos evolutivos, optimización bayesiana, el método de entropía cruzada (CEM) o el método de estrategias de evolución con adaptación de la matriz de covarianza (CMA-ES). Nos vamos a centrar en la optimización bayesiana, que es el método que se va a implementar en el trabajo, y en los algoritmos evolutivos, con los que haremos una comparación de los resultados obtenidos.

Optimización bayesiana

La optimización bayesiana [3] es un método de optimización basado en el teorema de Bayes. La optimización bayesiana consiste en establecer una distribución de probabilidad a priori sobre la función objetivo y combinarla con la experiencia para obtener una distribución posterior. Esto permite hacer una selección de la siguiente observación de la función objetivo basada en la utilidad para la optimización. Para ello se dispone de una función de adquisición, que tiene en cuenta las áreas con mayor incertidumbre (exploración) y las áreas con mayor probabilidad de mejora de la solución (explotación). La optimización bayesiana es un método de optimización global, por lo que nos aseguramos una convergencia global. Posteriormente, este método se explicará con mayor profundidad.

Algoritmos evolutivos

Los algoritmos evolutivos [4] derivan de la teoría de la evolución de Darwin y la selección natural. Un algoritmo evolutivo actualiza de forma iterativa una población de posibles soluciones, que a menudo se codifican en estructuras llamadas cromosomas. En cada iteración, llamada generación, el algoritmos evalúa las soluciones y genera nuevas soluciones basadas en la adecuación de las soluciones anteriores. Las subestructuras de las soluciones, también llamadas genes, se modifican después mediante operadores genéticos como la recombinación y la mutación. La idea es que las estructuras asociadas a buenas soluciones puedan mutar y ser combinadas para formar mejores soluciones en las siguientes generaciones. Existe una gran variedad de algoritmos evolutivos, como por ejemplo los algoritmos genéticos, programación evolutiva, programación genética y estrategias de evolución.

En los algoritmos evolutivos para aprendizaje por refuerzo suelen utilizarse redes neuronales para aproximar la función de la política y el objetivo es encontrar los parámetros de la red. Uno de los métodos modernos con mayor relevancia es la neuroevolución de topologías aumentadas [5], en el que además de modificar los parámetros de la red neuronal, también se modifica su topología.

El método que utilizaremos para comparar con la optimización bayesiana será la evolución diferencial [6]. La evolución diferencial es un tipo de algoritmo evolutivo que se caracteriza por el uso de vectores de prueba que compiten con los individuos de la población actual por sobrevivir. Para cada generación se utiliza un número NP de

vectores de parámetros. NP es constante durante la minimización. En una generación, cada miembro de la población se compara con un vector de prueba que se genera sumando la diferencia ponderada de dos miembros de la población a un tercero. Si el vector resultante produce un valor de la función objetivo menor que el miembro con el que se compara, el vector recién generado lo reemplazará en la siguiente generación. Debido a esto, en cada iteración/generación del algoritmo hay un número de evaluaciones de la función objetivo igual al tamaño de la población.

En este capítulo se ha introducido el aprendizaje por refuerzo con una introducción sobre la materia. Se han explicado los procesos de decisión de Márkov como el principal marco de referencia para plantear y solucionar los problemas de aprendizaje por refuerzo. De los posibles métodos de soluciones, se ha incidido en la búsqueda de política, ya que es el método que se va a implementar en el trabajo. Dentro de la búsqueda de política se han explicado los métodos evolutivos y la optimización bayesiana, que va a ser explicada con mayor profundidad en el siguiente capítulo.

# 3. OPTIMIZACIÓN BAYESIANA

Para llevar a cabo la redacción de este capítulo se ha llevado a cabo una revisión del estado del arte de la optimización bayesiana con el objetivo de cubrir las necesidades teóricas del trabajo fin de máster. La mayor parte del capítulo está redactado a partir de un tutorial de optimización bayesiana [3]. En este tutorial se presentan los principios teóricos básicos de la optimización bayesiana. Además, también presenta dos extensiones de la optimización bayesiana con ejemplos prácticos, que escapan del alcance del trabajo. Así mismo, también se refiere al lector a una serie de artículos para profundizar en la materia: en el primero se aplica la optimización bayesiana a problemas de agarre en robots [7] en el que además se plantea el problema desde el aprendizaje por refuerzo y en el segundo se aplica la optimización bayesiana a sistemas autónomos [8] y se presenta otra extensión llamada optimización bayesiana espartana.

## 3.1. INTRODUCCIÓN

La optimización bayesiana es un método para obtener el máximo de una función de coste. Se aplica para aquellas funciones de las que no se conocen su forma matemática, convexidad o sus derivadas. En algunos casos solo pueden obtenerse una estimación de la función tomando muestras, lo que puede implicar procesos costosos: simulaciones, prueba de drogas, test destructivos o inversiones financieras. Por tanto, es relevante minimizar el número de muestras para obtener el valor de la función que se busca.

En nuestro caso, lo que nos interesa es aprender los valores de una política con el mínimo número de muestras posible, de forma que maximicemos la recompensa. El problema de maximización puede convertirse en uno de minimización, cambiando la función de recompensa por una de coste. En ese caso, la maximización de una función $f(x)$ sería equivalente a la minimización de una función $g(x)$, y:

$$g(x) = -f(x)$$

En la búsqueda de política, la función que queremos maximizar es $J(\theta)$. La optimización bayesiana proporciona un mecanismo de explotación-exploración que permite encontrar las regiones relevantes y ajusta la función donde es necesario.

La optimización bayesiana presenta como ventajas frente al método del gradiente de la política que no depende de las derivadas, tiene menos posibilidades de quedarse atascado en el primer mínimo local y está diseñada para minimizar el número de evaluaciones de funciones costosas.

La optimización bayesiana es uno de los métodos con mayor eficientes en términos del número de evaluaciones necesarias. Esta eficiencia proviene de la capacidad de incorporar la creencia a priori sobre el problema para tomar muestras y el equilibrio entre exploración y explotación. La creencia a priori se incorpora con el teorema de Bayes, que expone que la probabilidad a posteriori de un modelo M dados los datos E,

es proporcional a la verosimilitud de $E$ dado $M$ multiplicada por la probabilidad a priori de $M$.

$$P(M|E) \propto P(E|M) \cdot P(M)$$

Aunque la función de coste no es conocida, podemos suponer que conocemos algunas de sus propiedades, como por ejemplo la suavidad, por lo que algunas funciones son más probables que otras.

La verosimilitud $P(E|M)$ expresa como de verosímiles son los datos que hemos muestreado dado lo que sabemos a priori sobre el modelo. Por ejemplo, si consideramos que la función es suave y sin ruido, los datos con grandes oscilaciones o variaciones deberían ser considerados menos probables que datos que apenas se desvían de la media. Conforme se van acumulando muestras $D_{1:t} = \{x_{1:t}, f(x_{1:t})\}$, la distribución a priori se combina con la verosimilitud $P(D_{1:t}|f)$ , obteniendo la probabilidad a posteriori.

$$P(f|D_{1:t}) \propto D_{1:t} \cdot P(f)$$

Con la probabilidad a posteriori actualizamos la creencia sobre la función objetivo. Este paso también puede entenderse como la estimación de la función objetivo mediante una función subyacente.

Para muestrear de forma eficiente la optimización bayesiana utiliza una función de adquisición, que incorpora el equilibrio entre explotación y exploración. Es decir, esta función de adquisición tendrá mayor valor en aquellas zonas en las que haya mayor incertidumbre y en aquellas zonas en las que el valor de la función se espera que sea alto.

Además, se va a asumir que la función objetivo es una función lipschitziana y que el problema de optimización es global, en lugar de local. Para la optimización global es común considerar que la función objetivo es una caja negra: no tenemos su expresión matemática ni sus derivadas. En este caso evaluar la función se restringe a probar puntos y recoger la respuesta, que posiblemente tenga ruido.

En resumen, la optimización bayesiana es un método de optimización basado en dos componentes: la distribución posterior de la función objetivo y la función de adquisición. Al acumular observaciones, la distribución a priori se combina con la verosimilitud para obtener la distribución posterior, que actualiza la creencia sobre la función objetivo. La función de adquisición nos permite decidir donde muestrear.

En la Fig. 3.1 podemos ver un ejemplo de optimización bayesiana en 1D. La optimización comienza con dos puntos. En cada iteración se busca el máximo de la función de adquisición para determinar donde muestrear, se muestrea la función objetivo en el máximo de la función de adquisición y se actualiza el proceso, y así repetidamente.

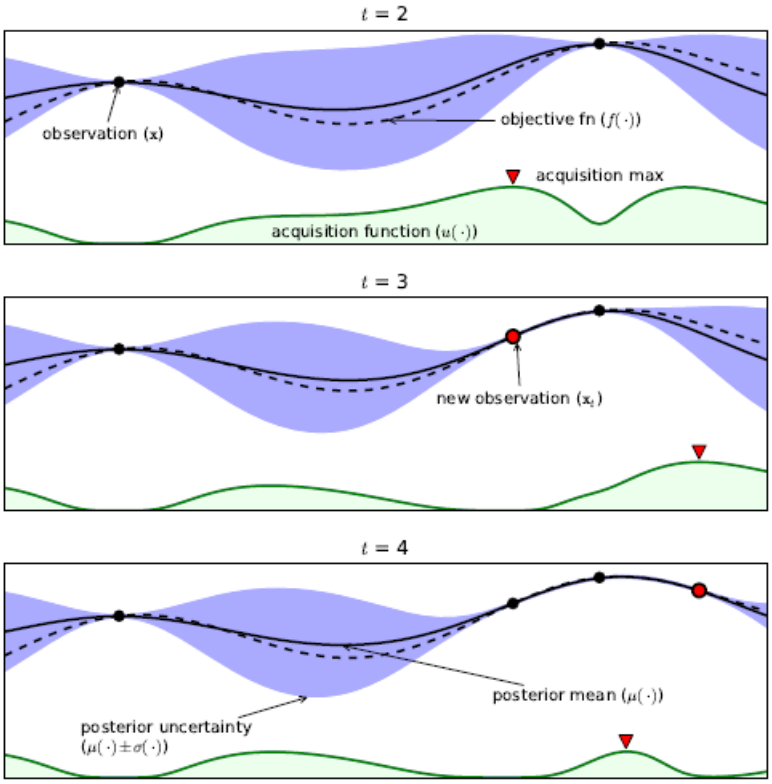



*Fig. 3.1 – Ejemplo de optimización bayesiana [3]*

A continuación se trata con mayor profundidad la distribución a priori y la función de adquisición.

## 3.2. DISTRIBUCIÓN A PRIORI

Como ya se ha explicado, la optimización bayesiana depende de una distribución a priori. La optimización bayesiana converge si [3]:

(i) La función de adquisición es continua y minimiza aproximadamente el riesgo, definido como la desviación esperada del mínimo global en un punto x.

(ii) La varianza condicional converge a cero (o un valor limite positivo si hay ruido) si y solo si la distancia a la observación más cercana es cero.

Con estas condiciones se podrían considerar multitud de modelos para la distribución a priori. Sin embargo, la distribución más utilizada en los trabajos con optimización bayesiana son los procesos gaussianos (GP). Para ello, es necesario especificar tres condiciones naturales adicionales:

(iii) La función objetivo es continua

(iv) La distribución a priori es homogénea

(v) La optimización es independiente de la enésima derivada

Un proceso gaussiano es la extensión de la distribución gaussiana o normal multivariante a un proceso estocástico de infinitas dimensiones, en el cualquier combinación de variables resulta en una distribución gaussiana. Un proceso gaussiano es una distribución sobre funciones, completamente especificado por una función de la media, $m(x)$, y una función de la covarianza, $k(x, x')$.

$$f(x) \sim GP(m(x), k(x, x'))$$

Intuitivamente, se puede pensar que un proceso gaussiano es como una función, pero al evaluarlo en un valor concreto de $x$, en lugar de obtener el valor escalar de la función lo que obtenemos es una media y una varianza.

Por conveniencia, se suele considerar que el valor de la función de la media es siempre 0. Por tanto, queda por definir la función de covarianza. Una de las opciones más utilizadas es la función exponencial cuadrática:

$$k(x_i, x_j) = exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right)$$

El valor de la función se acerca a 1 si los puntos están cerca y 0 si están lejos. Si los puntos están cerca es más probable que tengan una gran influencia entre ellos, mientras que si están más lejos la influencia es probable que sea menor.

Al realizar la optimización, iremos guardando observaciones $D_{1:t} = \{x_{1:t}, f(x_{1:t})\}$ y el siguiente paso será decidir en qué punto muestrear, $x_{t+1}$, el cual tendrá un valor $f(x_{t+1})$. Por las propiedades de los procesos gaussianos, $f(x_{1:t})$ y $f(x_{t+1})$ son conjuntamente gaussianos:

$$\begin{bmatrix} f(x_{1:t}) \\ f(x_{t+1}) \end{bmatrix} = N\left(0, \begin{bmatrix} K & k \\ k^T & k(x_{t+1}, x_{t+1}) \end{bmatrix}\right)$$

En la que $K$ es la matriz del kernel, en la que los valores de la diagonal serán 1 si no hay ruido, ya que entonces cada punto está perfectamente correlacionado consigo mismo.

$$K = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_t) \\ \vdots & \ddots & \vdots \\ k(x_t, x_1) & \cdots & k(x_t, x_t) \end{pmatrix}$$

Y la matriz $k$ es:

$$k = [k(x_{t+1}, x_1) \quad k(x_{t+1}, x_2) \quad \cdots \quad k(x_{t+1}, x_t)]$$

Utilizando la fórmula de Sherman-Morrison-Woodbury [3], se puede llegar a una expresión para la distribución posterior:

$$P(f|D_{1:t}, x_{t+1}) = N\left(\mu_t(x_{t+1}), \sigma_t^2(x_{t+1})\right)$$

$$\mu_t(x_{t+1}) = k^T K^{-1} k \cdot f(x_{t+1})$$

$$\sigma_t^2(x_{t+1}) = k(x_{t+1}, x_{t+1}) - k^T K^{-1} k$$

Es decir, que con la media y la varianza es suficiente para definir la probabilidad posterior de la función objetivo.

En la mayoría de los casos es necesario generalizar la función de covarianza añadiendo hiperparámetros. Los hiperparámetros son parámetros que se configuran antes de la instanciación del modelo y no forman parte de este, pero que si influyen en la capacidad y las características del modelo. Por ejemplo, la función de covarianza definida antes, la función exponencial al cuadrado quedaría de la siguiente forma en un modelo isotrópico, con un único hiperparámetro θ:

$$k(x_i, x_j) = exp\left(-\frac{1}{2\theta^2}\|x_i - x_j\|^2\right)$$

Para modelo anisotrópicos, se suele utilizar la función exponencial al cuadrado con un vector de determinación automática de relevancia (automatic relevance determination (ARD)) de hiperparámetros θ:

$$k(x_i, x_j) = exp\left(-\frac{1}{2}(x_i - x_j)^T diag(\theta)^{-2}(x_i - x_j)\right)$$

En la que $diag(\theta)$ es una matriz diagonal de tamaño igual al de la dimensión del problema. Si uno de los valores de $\theta$ es 0 significa que el kernel es independiente de esa dimensión.

## 3.3. FUNCIONES DE ADQUISICIÓN

En este subapartado se van a comentar de forma sintética algunas posibles funciones de adquisición. Como ya se ha comentado anteriormente, la función de adquisición sirve de guía para buscar el valor óptimo de la función objetivo. Estas funciones se definen de forma que valores altos de la función de adquisición se correspondan con posibles valores altos de la función objetivo, valores con alta incertidumbre o ambos.

- Probabilidad de mejora (Probability of Improvement, PI): el objetivo es maximizar la probabilidad de mejora sobre la mejor solución en el paso actual $f(x^+)$, $x^+ = arg\ \underset{x_i \epsilon x_{1:t}}{max}\ f(x_i)$. Es decir:

$$PI(x) = P\big(f(x) > f(x^+)\big) = \Phi\left(\frac{\mu(x) - f(x^+)}{\sigma(x)}\right)$$

  Donde $\Phi$ es la función de distribución acumulativa de la distribución normal. Sin embargo, puede verse intuitivamente que con esta formulación solo se puede explotar, ya que puntos que tengan una gran probabilidad de ser infinitesimalmente mejores que $f(x^+)$ serán escogidos en lugar de puntos que puedan ser mejores pero que tengan una gran incertidumbre. Por ello es necesaria una reformulación introduciendo un parámetro de equilibrio, $\xi \geq 0$:

$$PI(x) = P(f(x) > f(x^+) + \xi) = \Phi\left(\frac{\mu(x) - f(x^+) - \xi}{\sigma(x)}\right)$$

Lo recomendable que $\xi$ sea grande al principio de la optimización, para que la exploración tenga más peso, y vaya decreciendo hasta llegar a cero.

- Mejora esperada (Expected Improvement, EI): esta función de adquisición no solo tiene en cuenta la probabilidad de mejora si no también la magnitud de la mejora. El objetivo es que el siguiente punto de prueba se desvíe lo mínimo posible del máximo verdadero de la función objetivo. Para ello se podría utilizar la programación dinámica para tener en cuenta los pasos futuros. Debido al coste de ello, la alternativa es maximizar la mejora esperada con respecto a la mejor solución en el paso actual, $f(x^+)$. La función de mejora se define de forma que sea positiva si el nuevo punto mejora a la solución actual y sea 0 en caso contrario:

$$I(x) = max\ \{0, f_{t+1}(x) - f(x^+)\}$$

El nuevo punto se encuentra maximizando la mejora esperada:

$$x_{t+1} = arg\ \max_x \mathbb{E}(max\{0, f_{t+1}(x) - f(x^+)\}\,|D_t)$$

La probabilidad de mejora de una distribución posterior normal caracterizada por la media y la varianza se puede computar a partir de la función de densidad. La mejora esperada es la integral de esta función, y queda:

$$EI(x) = \begin{cases} \big(\mu(x) - f(x^+)\big) \cdot \Phi(Z) + \sigma(x) \cdot \Phi(Z)\ si\ \sigma(x) > 0 \\ 0\ si\ \sigma(x) = 0 \end{cases}$$

$$Z = \frac{\mu(x) - f(x^+)}{\sigma(x)}$$

Donde $\Phi(\cdot)$ es la función de distribución de probabilidad acumulativa y $\Phi(\cdot)$ es la función de densidad de la distribución normal.

Al igual que en el caso anterior, puede introducirse un parámetro de equilibrio $\xi \geq 0$, que controle el equilibrio entre exploración y explotación, de forma que queda:

$$EI(x) = \begin{cases} (\mu(x) - f(x^+) - \xi) \cdot \Phi(Z) + \sigma(x) \cdot \Phi(Z)\ si\ \sigma(x) > 0 \\ 0\ si\ \sigma(x) = 0 \end{cases}$$

$$Z = \frac{\mu(x) - f(x^+) - \xi}{\sigma(x)}$$

Para este caso, un valor de $\xi = 0.01$ funciona bien. Al contrario que en el caso anterior, comenzar la optimización con un valor alto de $\xi$ que vaya decreciendo conforme avanza la optimización no funciona correctamente.

## 3.4. ALGORITMO BÁSICO

El proceso de la optimización bayesiana podría resumirse en tres componentes:

- Inicialización: es necesario inicializar el modelo con el objetivo de evitar parcialidades y garantizar la optimalidad global. Un posible método es el muestreo de hipercubo latino (LHS, latin hypercube sample), un método cuasi aleatorio que permite una inicialización cubriendo todo el espacio manteniendo una componente aleatoria.

- Aprendizaje: actualizar el modelo y los hiperparámetros con la información obtenida de la experiencia y la creencia a priori de la función objetivo.

- Decisión: decidir donde muestrear maximizando la función de adquisición sobre el modelo.

A continuación se describe el algoritmo básico que se sigue al realizar una optimización mediante el método de optimización bayesiana:

| *Algoritmo* 1: *Optimización Bayesiana* [8] | |
|---|---|
| *Entradas*: *número máximo de iteraciones* $N$, *número de muestras iniciales* $p$, *función de adquisición* $C$ | |
| 1: $\boldsymbol{X} \leftarrow x_{1:p}, \boldsymbol{y} \leftarrow y_{1:p}$ | *Diseño inicial con LHS* |
| 2: $\boldsymbol{for}\ t = 1 \ldots N\ \boldsymbol{do}$ | |
| 3: $\quad \boldsymbol{\theta} = MuestrearHiperparametros(\boldsymbol{X}, \boldsymbol{y})$ | |
| 4: $\quad x_t = \arg\max_{x} C\ (x \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\theta})$ | |
| 5: $\quad y_t = f(x_t), \boldsymbol{X} \leftarrow \text{añadir}(x_t), \boldsymbol{y} \leftarrow \text{añadir}(y_t)$ | |

En este capítulo se ha presentado la optimización bayesiana, el método que se va a implementar en el trabajo para obtener los parámetros de la política. Primero se ha introducido el funcionamiento del método, basado en las observaciones y la creencia a priori de la función objetivo y de una función de adquisición. Después se ha explicado la distribución a priori de la función objetivo y posibles funciones de adquisición. Por último, se ha explicado el algoritmo básico de la optimización bayesiana.

# 4. HERRAMIENTAS DE SIMULACIÓN DE ROBOTS

## 4.1. COPPELIASIM

### 4.1.1. Introducción

CoppeliaSim [9] es una plataforma de simulación robótica versátil y escalable desarrollada por Coppelia Robotics. Es libre y de código abierto mientras se utilice sin fines comerciales.

En este trabajo vamos a utilizar el simulador CoppeliaSim para construir las escenas y modelos correspondientes para cada una de las tareas diseñadas y para simular las tareas. Por lo tanto, es necesario conocer la funcionalidad de CoppeliaSim para construir las escenas y de qué forma se pueden controlar las simulaciones.

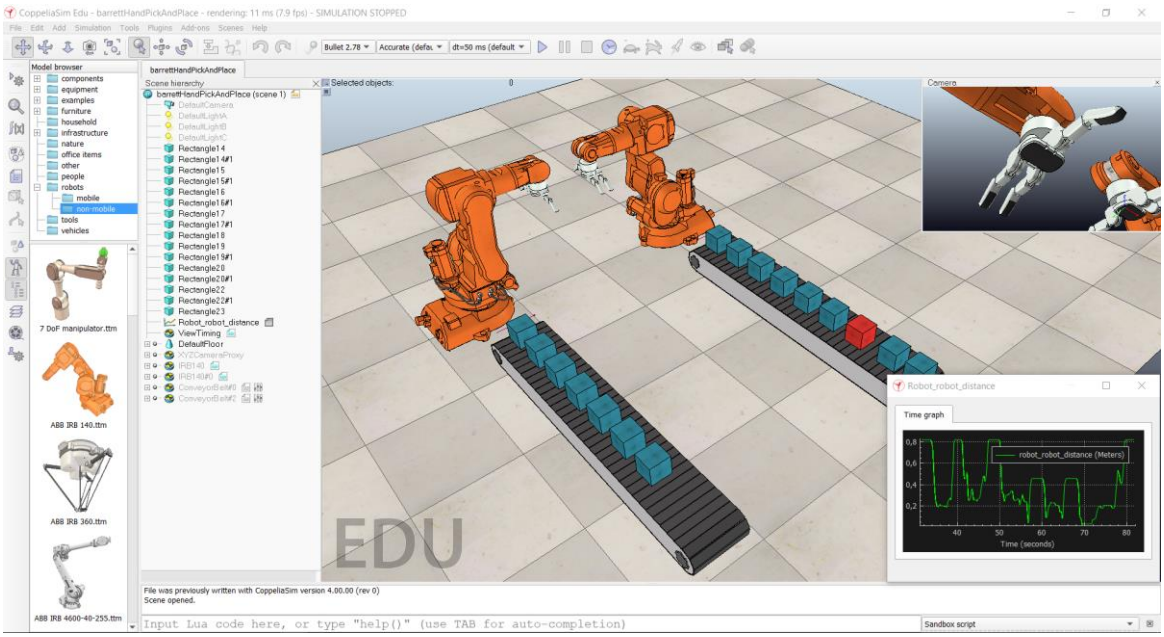



*Fig. 4.1 – Interfaz de CoppeliaSim con una escena de ejemplo cargada.*

### 4.1.2. Funcionalidad del simulador

La funcionalidad de CoppeliaSim está relacionada tanto con los objetos de la escena como con los módulos de calculo que tiene disponibles. Dado que solo vamos a necesitar los objetos de la escena para este trabajo, los módulos de cálculo no se van a explicar.

#### *4.1.2.1. Objetos de la escena*

Una escena o modelo de CoppeliaSim contiene varios objetos de la escena u objetos elementales que se pueden ensamblar según una jerarquía de árbol. Los objetos disponibles en CoppeliaSim son los siguientes:

- Articulaciones (joints): las articulaciones son elementos que permiten enlazar dos o más objetos de la escena con uno a tres grados de libertad según el tipo

de articulación (prismática, rotacional, tornillo, esférica). Tienen distintos modos de operación (modo de fuerza/momento, modo de cinemática inversa, etc.).

- Formas (shapes): las formas son mallas triangulares utilizadas para la simulación y visualización de solidos rígidos. Otros objetos de la escena o los módulos de cálculo dependen en gran medida de las formas (p. ej. los sensores de proximidad, el módulo de dinámica o el módulo de cálculo de distancia malla-malla).

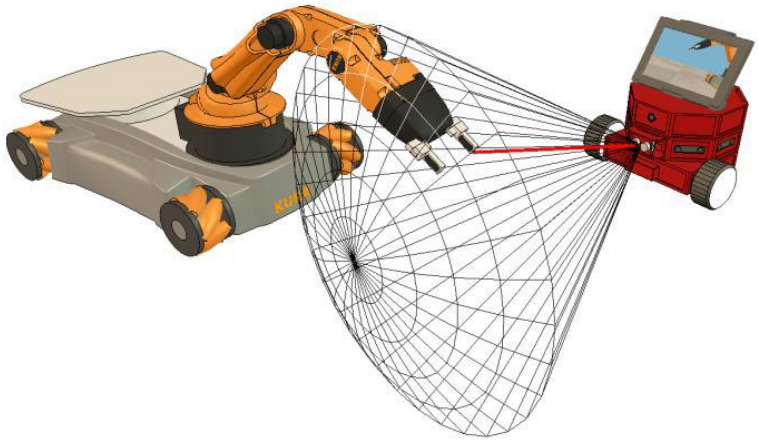


*Fig. 4.2 – Sensor de proximidad [9]*

- Sensores de proximidad (proximity sensor): los sensores de proximidad son elementos que llevan a cabo un cálculo de la distancia mínima exacto a la parte de una forma que se encuentra dentro de un volumen de detección configurable, en lugar de realizar una detección basada en rayos. Como resultado se obtiene una operación más continua y, por tanto, permite una simulación más realista.

- Sensores de visión (visión sensors): los sensores de visión son elementos que permiten extraer información compleja de la imagen (colores, tamaño de los objetos, mapas de profundidad, etc.)

- Sensores de fuerza (force sensors): los sensores de fuerza son elementos que representan enlaces rígidos entre formas, que pueden registrar fuerzas y momentos aplicados y que pueden romperse cuando sobrepasa una condición determinada.

- Gráficos (graphs): los gráficos son elementos que pueden registrar una gran variedad de flujo de datos predefinidos o personalizados. Los flujos de datos pueden representarse directamente (gráfica de un tipo de dato respecto al tiempo) o combinarse con otros para mostrar un gráfico XY o curvas 3D.

- Cámaras (cameras): las cámaras son elementos que permiten la visualización de la escena cuando están asociadas a una ventana.

- Luces (lights): las luces son elementos que iluminan la escena o elementos de la escena y que influyen directamente en las cámaras o en los sensores de visión.

- Trayectorias (paths): las trayectorias son elementos que permiten definir movimientos complejos en el espacio (p. ej. sucesión de traslaciones, rotaciones y pausas, combinadas de forma libre). Las trayectorias pueden utilizarse para guiar el soplete de un robot de soldadura a lo largo de una trayectoria predefinida o para permitir los movimientos de una cinta transportadora.

- Dummies: los dummies son sistemas de referencia, que pueden tener varias funciones y que son utilizados junto a otros objetos de la escena, por lo que pueden ser vistos como elementos de ayuda.

### 4.1.3. Control de la simulación

CoppeliaSim permite controlar la simulación con distintas técnicas, que pueden utilizarse de simultáneamente y también simbióticamente. Las técnicas de control disponibles son:

- Scripts incrustados (embedded scripts): los scripts incrustados son scripts que forman parte de una escena o de un modelo, y que se guardan y se cargan al mismo tiempo de la escena o el modelo. El lenguaje de programación es Lua. En una escena cualquiera, hay un script principal (main script) que se encarga de la funcionalidad general (p. ej. llamar a las funciones que se encargan de la cinemática o la dinámica). El script principal también se encarga de llamar a los scripts secundarios (child scripts) en forma de cascada con respecto a la jerarquía de la escena. Los scripts secundarios están unidos a objetos de la escena y se encargan de una parte concreta de la simulación.

- Add-ons: al igual que los scripts incrustados, los add-ons funcionan mediante scripts de Lua. Se pueden usar como funciones independientes (p. ej. para programar importadores/exportadores) o como código ejecutado de forma convencional (p. ej. como método ligero de customización del simulador).

- Plug-ins: los plug-ins son utilizados como una herramienta de customización. Pueden registrar comandos de Lua personalizados, permitiendo la ejecución de funciones callback desde un script incrustado. También pueden extender la funcionalidad de un modelo u objeto particular.

- Clientes de API remota: los clientes de API remota permiten el control de simulación desde una aplicación o hardware externos. La API externa de CoppeliaSim está compuesta por aproximadamente cien funciones específicas y una general, que se pueden llamar desde un programa en C/C++, Python, Java… Las funciones de la API remota interactúan con el simulador con comunicación mediante sockets.

- Nodos de ROS: CoppeliaSim implementa un nodo de ROS con un plugin que permite a ROS llamar a los comandos de CoppeliaSim desde los servicios ROS, o transmitir datos desde publicadores/suscriptores ROS. Los publicadores/ suscriptores pueden habilitarse con una llamada de servicio y también

directamente desde CoppeliaSim, a través de un comando de un script incrustado.

## 4.2. PYREP

PyRep [10] es un kit de herramientas para la investigación en el aprendizaje automático aplicado a la robótica y está desarrollado utilizando CoppeliaSim como motor de simulación.

PyRep ofrece una API en Python y una versión modificada del código abierto de CoppeliaSim para que la comunicación mediante Python sea directa y más rápida que con la API que viene por defecto, diseñados específicamente para aplicaciones de aprendizaje automático en las que es necesario recopilar datos de gran tamaño.

La API de PyRep gestiona el control de la simulación y proporciona una interfaz con la simulación mediante programación orientada a objetos. Además, permite añadir nuevos robots con unas pocas líneas de código.

```
3   from pyrep import PyRep
4   from pyrep.robots.arms.panda import Panda
5   from pyrep.objects.dummy import Dummy
6   from pyrep.objects.shape import Shape
7   from pyrep.objects.proximity_sensor import ProximitySensor
8   from pyrep.errors import ConfigurationPathError
9   from pyrep.robots.end_effectors.panda_gripper import PandaGripper
```

*Fig. 4.3 – Importación de las bibliotecas de PyRep*

En la Fig. 4.3 se puede ver un código en que se importan unas bibliotecas de PyRep. De la biblioteca pyrep se importa PyRep, que es la función que permite controlar la simulación. Las bibliotecas de dummy, shape… permiten acceder a los objetos de la escena, obtener información y actuar sobre ellos.

```
40      def __init__(self, headless_mode: bool):
41          self.pyrep = PyRep()
42          self.pyrep.launch(join(DIR_PATH, TTT_FILE), headless=headless_mode)
43          self.robot = Robot(Panda(), PandaGripper(), Dummy('Panda_tip'))
```

*Fig. 4.4 – Inicialización de PyRep en una de las tareas*

En la Fig. 4.4 se puede ver la inicialización de PyRep en una de las tareas implementadas. Se inicializa la función de PyRep y se lanza la simulación con una escena, en este caso TTT_FILE que se encuentra en DIR_PATH. El parámetro headless permite ejecutar una simulación sin la interfaz gráfica.

```
65              path = self.robot.arm.get_linear_path(position=pos.get_position(),
66                                                    euler=[0.0, np.radians(180), 0.0])
67              # Step the simulation and advance the agent along the path
68              done = False
69              while not done:
70                  done = path.step()
71                  self.pyrep.step()
```

*Fig. 4.5 – Ejemplo cálculo y ejecución de trayectoria*

En la Fig. 4.5 se puede ver un código en el que se calcula y ejecuta una trayectoria. Primero se calcula la trayectoria especificando una posición y una orientación. En este caso, pos sería un dummy y con la función get_position() se obtiene su posición.

La trayectoria se ejecuta paso a paso en un bucle while, que termina cuando se llega al final de la trayectoria.

En este capítulo se han explicado las herramientas de simulación de robots que se han utilizado en el trabajo, CoppeliaSim y PyRep, y se han explicado sus partes más importantes.

# 5. HERRAMIENTAS DE OPTIMIZACIÓN

## 5.1. BAYESOPT

BayesOpt [11] es una biblioteca de optimización bayesiana para resolver problemas de optimización no lineal, diseño experimental y bandits. La biblioteca ha sido desarrollada por Rubén Martínez Cantín y es de software libre.

En el trabajo vamos a utilizar BayesOpt para resolver las tareas diseñadas, obteniendo como resultado los parámetros de la política que maximizan la función objetivo.

BayesOpt está diseñada para ser compatible en varias plataformas y configuraciones. La biblioteca se puede utilizar en distintos sistemas operativos (Windows, Linux, Mac OS), con diferentes compiladores. El núcleo de la biblioteca está escrito en C++ pero se dispone de interfaces para poder utilizar BayesOpt en C, Python y Matlab/Octave.

De los modos de utilización de la biblioteca vamos a utilizar el método callback. En este método debemos enviar un puntero o gestor de funciones al optimizador, siguiendo un prototipo. Para Python, la función se debe definir de forma que acepte parámetros en forma de un vector de NumPy y devuelva un escalar de tipo double. En nuestro caso, los parámetros serán de entrada serán los parámetros de la política y devolverá el valor de la función objetivo.

Además de la función, también debemos definir una serie de parámetros de optimización. A continuación, se resumen los parámetros más importantes. Se va a diferenciar entre los parámetros que son comunes para BayesOpt y la evolución diferencial de SciPy y los que son específicos de cada uno. Los parámetros comunes:

- n_iterations: el número de iteraciones de BayesOpt. Cada iteración se corresponde con una evaluación de la función objetivo. Actualmente, este es el único criterio de parada. En general, un mayor número de iteraciones se corresponde a una mayor precisión del resultado. En el algoritmo del apartado 3.4, n_iterations es el parámetro de entrada N.

- bounds: los límites para los parámetros de entrada. Para BayesOpt es necesario definir los limites superiores e inferiores por separado. Los limites superiores se definen en la variable ub (upper bound) y los limites inferiores en la variable lb (lower bound).

Los parámetros específicos de BayesOpt son los siguientes:

- n_iter_relearn: el número de iteraciones entre los reaprendizajes de los parámetros del kernel. Es decir, deben ocurrir n_iter_relearn iteraciones hasta que los parámetros del kernel se reaprenden. La mejor precisión se obtiene cuando n_iter_relearn es 1 (cuando los parámetros del kernel se reaprenden en cada iteración), sin embargo, esto repercute en un mayor coste computacional

y un mayor coste por iteración. En el algoritmo del apartado 3.4, la línea 3 se ejecutará cada n_iter_relearn iteraciones.

- n_init_samples: este parámetro sirve para decidir cuantas muestras se utilizan para aprender el modelo preliminar de la función objetivo. n_init_samples es el número de muestras utilizadas. Cada muestra necesita una evaluación de la función objetivo. En el algoritmo del apartado 3.4, n_init_samples es el parámetro de entrada p.

- epsilon: este parámetro sirve para implementar una estrategia épsilon-greedy. El valor de este parámetro épsilon es la probabilidad de hacer una evaluación aleatoria de la función objetivo. Un valor alto implica una exploración forzada mientras que valor bajos implican una mayor dependencia de la política de exploración/explotación del criterio.

- force-jump: en ocasiones puede ocurrir que el modelo aprendido sea malo y la optimización se bloquee, sobre todo en casos en los que el número de muestras iniciales es pequeño. Los saltos forzados miden el número de iteraciones en las que la diferencia entre evaluaciones consecutivas es menor que el ruido esperado. Entonces, se asume que cualquier ganancia es puto ruido y que se podría obtener más información en otro punto. Este parámetro establece el número de iteraciones sin ganancia antes de saltar a un punto aleatorio.

- l_type: BayesOpt intenta aprender un modelo Bayesiano totalmente analítico para la función subyacente, pero los hiperparámetros del kernel no pueden aprenderse de forma cerrada. l_type indica el método de aprendizaje para los parámetros del kernel. Están implementado los métodos L_FIXED, L_EMPIRICAL y L_MCMC. En el algoritmo del apartado 3.4, l_type es el método que se utiliza para muestrear los hiperparámetros.

- n: indica la dimensión del problema. El valor de n tiene que ser igual al tamaño del vector de entrada de la función que se quiere optimizar.

## 5.2. DIFFERENTIAL EVOLUTION

El algoritmo de evolución diferencial se ha implementado mediante la biblioteca de código abierto para Python de SciPy. Esta biblioteca se compone de herramientas y algoritmos matemáticos. La evolución diferencial es una función que se encuentra dentro de las herramientas de optimización [12].

Al igual que en BayesOpt, la función también utiliza el método callback, siendo la función de la tarea robótica un argumento de la función de evolución diferencial. Por tanto, la función de la tarea es la misma que para BayesOpt.

Además de la función, también se deben definir una serie de parámetros de optimización. A continuación, se resumen los parámetros más importantes, y su equivalente en BayesOpt:

- Bounds: los límites para los parámetros de entrada. Se pueden definir como una instancia de la clase Bounds o como una lista de tuplas (min, máx.) para cada parámetro de entrada.

- Maxiter: el número máximo de generaciones a lo largo de las cuales evoluciona la población.

- Popsize: un multiplicador para establecer el tamaño de la población. La población tiene $\mathrm{popsize} \cdot \mathrm{len(x)}$ individuos.

El número máximo de evaluaciones de la función es $(maxiter + 1) \cdot popsize \cdot len(x)$. No hay que confundir iteraciones/generaciones con el número de evaluaciones de la función. En cada generación, la función se evalúa un número de veces igual a la población, $popsize \cdot len(x)$. Por tanto, el equivalente a n_iterations de BayesOpt sería este producto.

# 6. DEFINICIÓN DEL PROBLEMA

## 6.1. INTRODUCCIÓN

Una vez hemos introducido los principios teóricos del aprendizaje por refuerzo y la optimización bayesiana y hemos explicado las herramientas tecnológicas que se han utilizado, pasamos a definir el objeto del TFM.

El objeto de este Trabajo Fin de Máster es el diseño, implementación y análisis de una biblioteca de tareas robóticas de manipulación en un simulador 3D como banco de pruebas para algoritmos de aprendizaje por refuerzo.

Para la parte de diseño, primero es necesario definir un entorno robótico, el robot que se utiliza, el espacio en el que se sitúa y los objetos que se encuentran a su alrededor. Una vez definido el entorno del robot, se plantean las tareas que el robot tiene que resolver.

## 6.2. ENTORNO ROBÓTICO

El entorno del robot cuenta con una mesa sobre la que se sitúa el robot. El robot utilizado para diseñar las tareas es el Panda de Franka Emika.

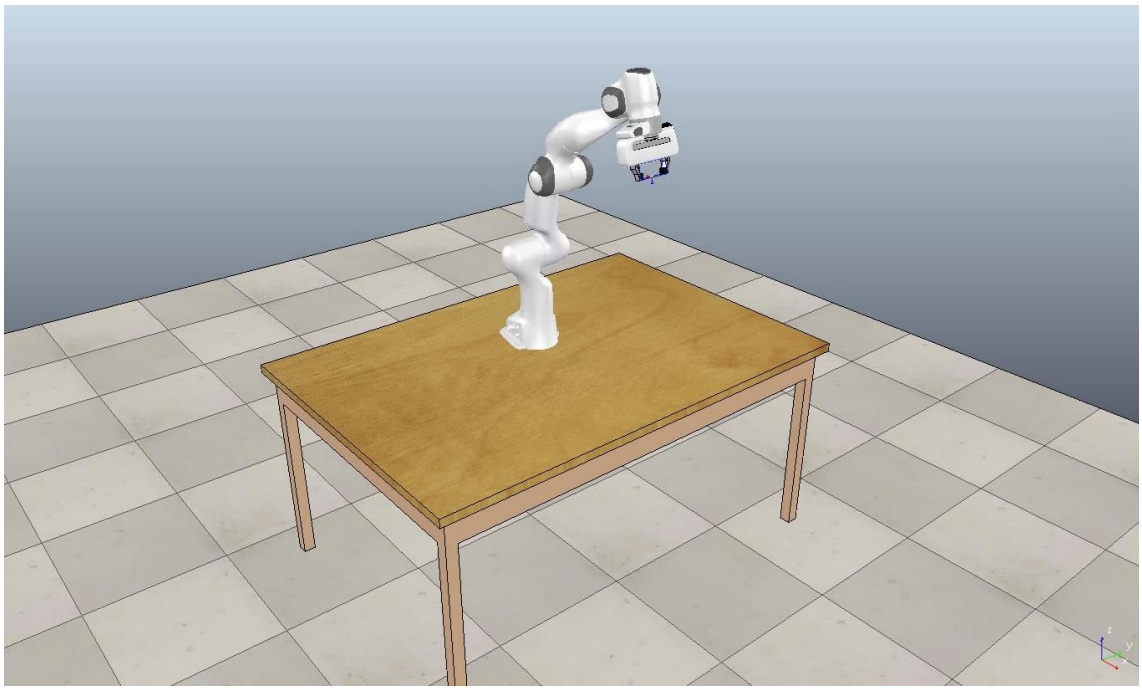


*Fig. 6.1 – Entorno robótico*

### 6.2.1. Panda

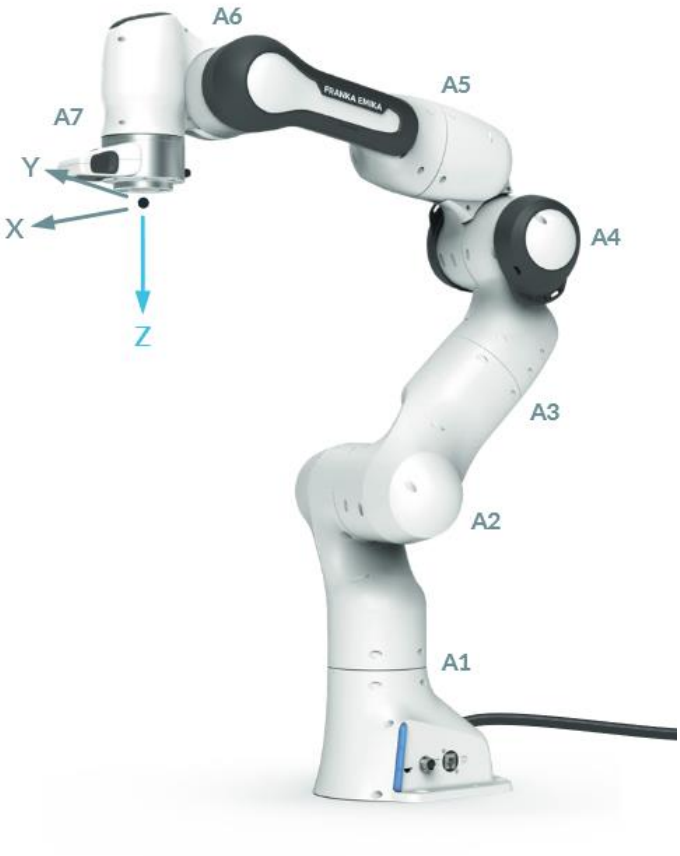



*Fig. 6.2 – Robot Panda de Franka Emika [13]*

El robot utilizado en las tareas es el Panda de Franka Emika [13], un robot industrial de 7 ejes, con una carga nominal de 3 kg, un alcance máximo de 855 mm y esta específicamente diseñado para robótica colaborativa. En la Fig. 6.3 se muestra el espacio de trabajo del robot.

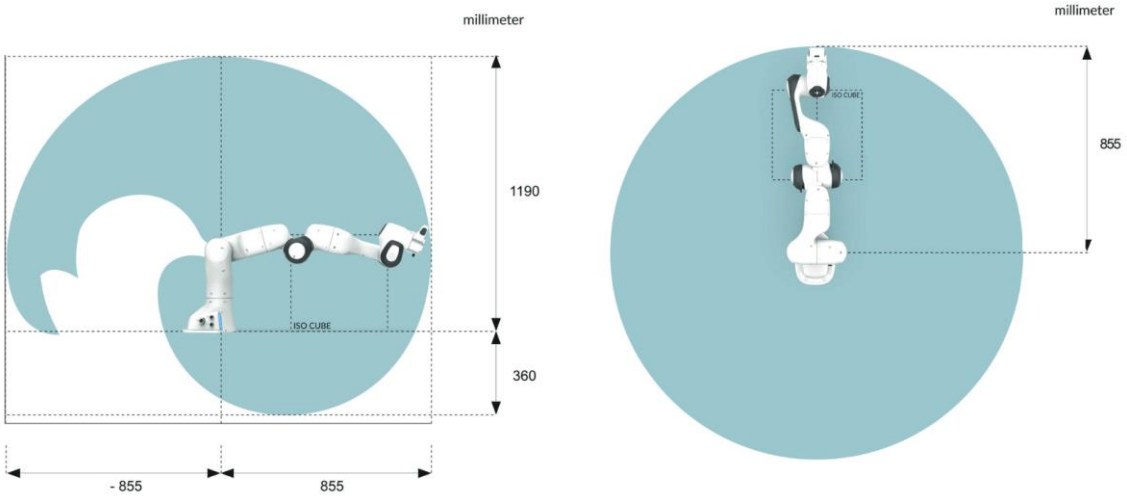



*Fig. 6.3 – Espacio de trabajo del Panda [13]*

## 6.3. PLANTEAMIENTO DE LAS TAREAS

### 6.3.1. Evitación de obstáculo

#### *6.3.1.1. Variación 1*

En esta variación delante del robot tenemos un objeto con forma esférica. El objetivo del robot consiste en ejecutar una trayectoria lo más corta posible de forma que ni el elemento terminal ni ninguno de los eslabones choque con el obstáculo.

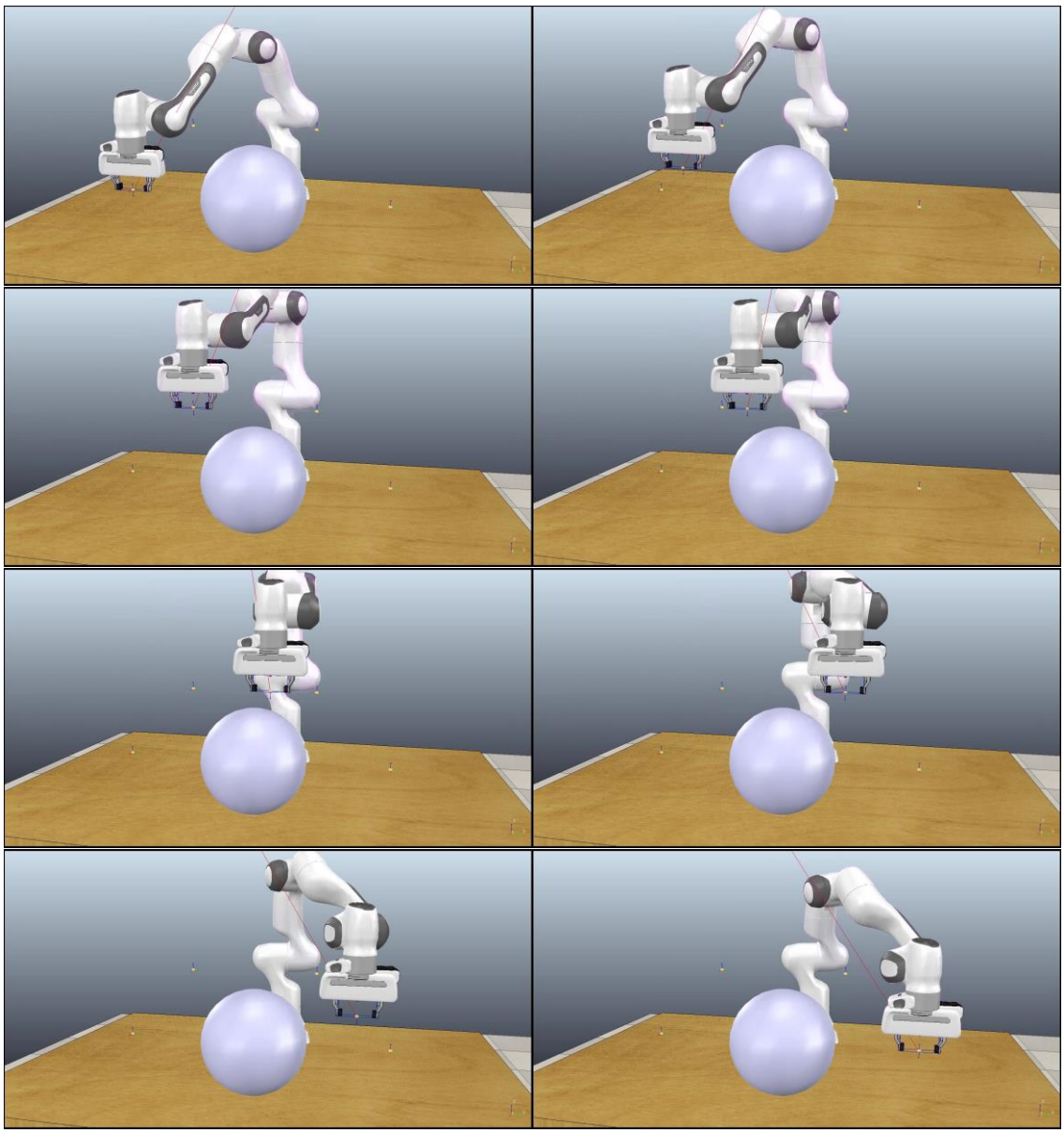


*Fig. 6.4 – Primera variación de la tarea de evitación de un obstáculo.*

#### *6.3.1.2. Variación 2*

En esta variación delante del robot tenemos tres cilindros con distinta separación entre ellos. El objetivo del robot consiste en ejecutar una trayectoria plana lo más corta posible de forma que el robot no se choque con los obstáculos. El interés de esta variación con respecto a la primera es que el robot va a tener dos posibilidades, ejecutar la trayectoria por cualquiera de las dos aberturas entre cilindros. Una de ellas será más difícil que la otra, ya que la distancia entre cilindros será es menor, pero también será más corta.

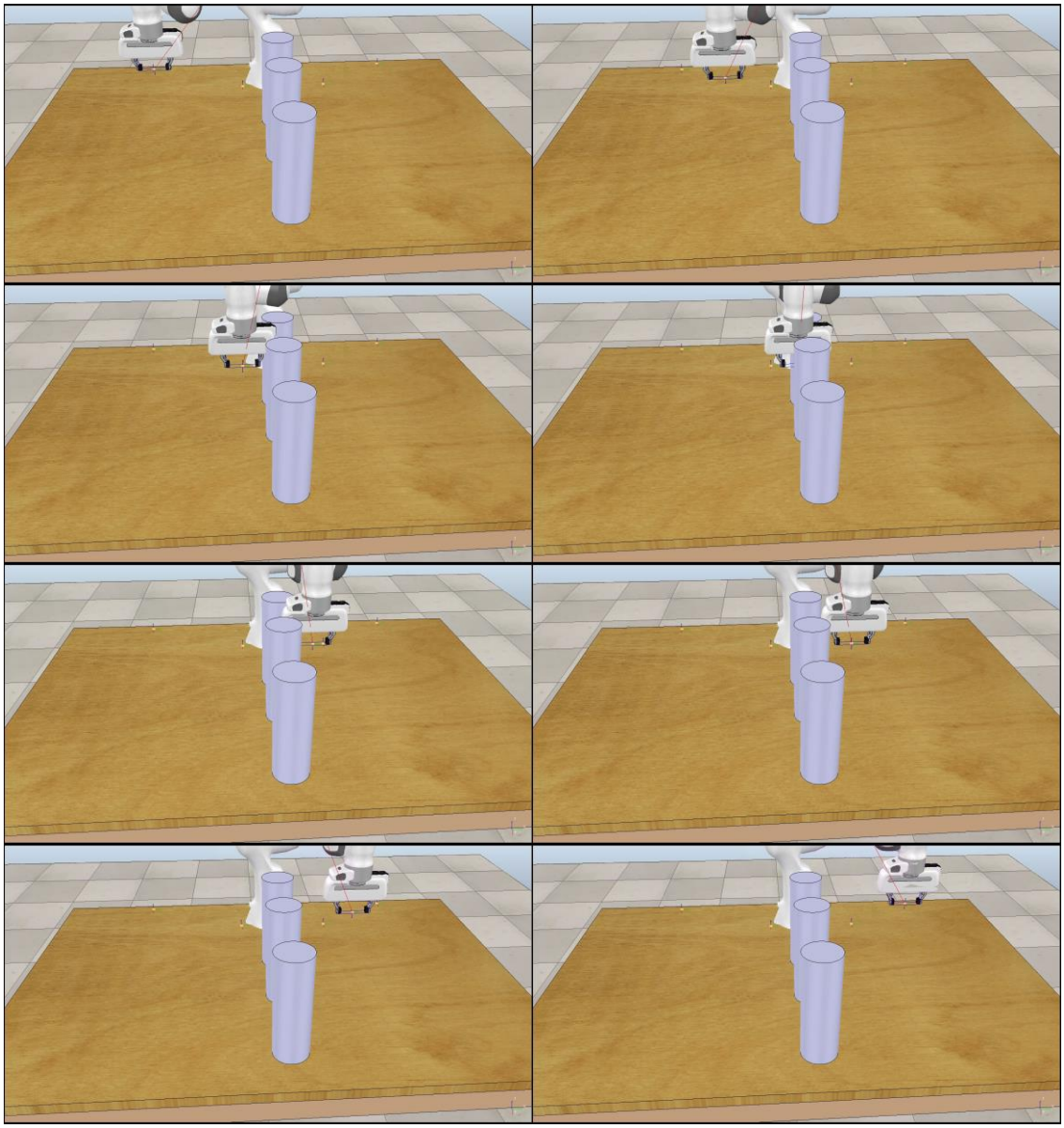


*Fig. 6.5 – Segunda variación de la tarea de evitación de un obstáculo*

### 6.3.2. Pick and place

#### *6.3.2.1. Variación 1*

Para esta variación de la tarea tenemos dos recipientes delante del robot. Uno está a la derecha, es más grande y en él se encuentra un objeto con forma cubo, y el otro está a la izquierda y es más pequeño. El objetivo del robot consiste en ejecutar una trayectoria de forma que coja el objeto del recipiente más grande y lo deposite dentro del recipiente pequeño. Para ello deberá cerrar la pinza cuando llegue a la posición inicial del cubo para cogerlo y deberá abrirla en la posición final para depositarlo.

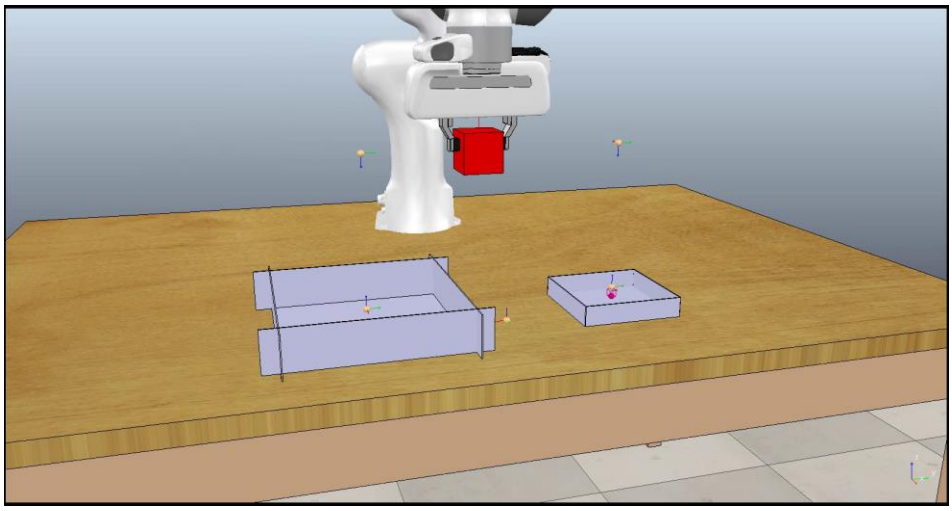


*Fig. 6.6 – Primera variación de la tarea de pick and place*

#### *6.3.2.2. Variación 2*

En esta variación delante del robot tenemos tres recipientes. El recipiente de la derecha es el mismo, pero a la izquierda tenemos dos recipiente pequeños en lugar de uno. El objetivo del robot consiste en ejecutar una trayectoria de forma que coja el objeto del recipiente más grande y lo deposite en uno de los recipientes pequeños. Lo interesante de esta variación es que, al contrario que en la variación 1, tenemos dos soluciones.

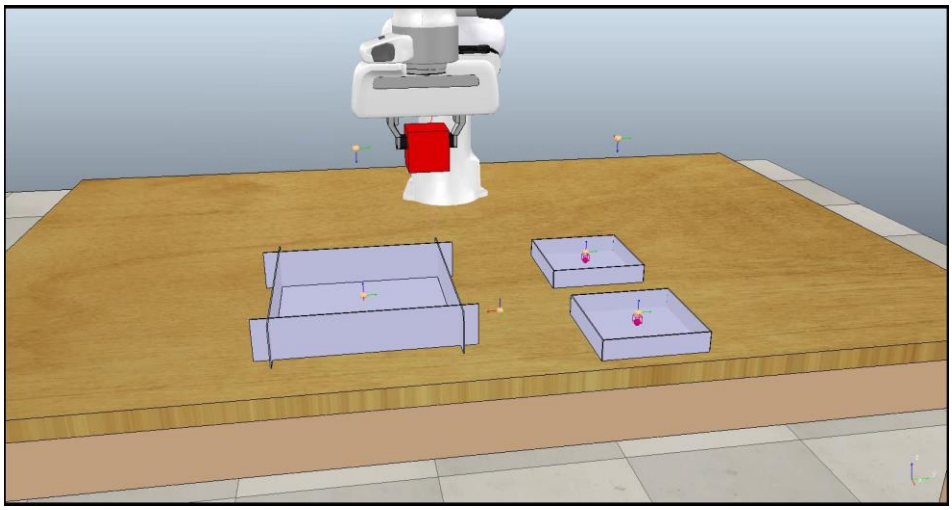


*Fig. 6.7 – Segunda variación de la tarea de pick and place*

### 6.3.3. Apretar un botón

#### *6.3.3.1. Variación 1*

En esta variación de la tarea tenemos un botón en un plano inclinado, simulando una mesa de controles. El objetivo del robot consiste en ejecutar una trayectoria de forma que accione el accione el botón de forma perpendicular a éste.

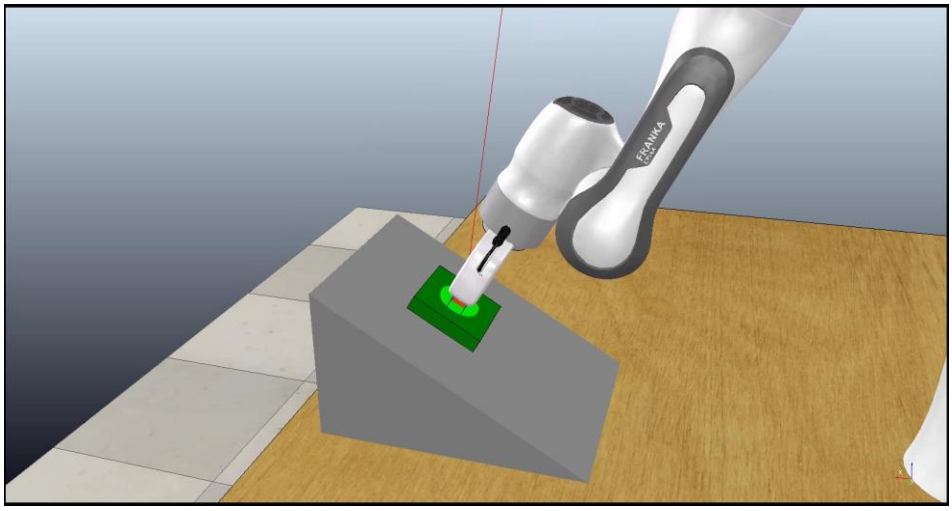


*Fig. 6.8 – Primera variación de la tarea de apretar un botón*

#### *6.3.3.2. Variación 2*

A diferencia de la primera variación, en esta ocasión tenemos dos botones. El objetivo del robot consiste en ejecutar una trayectoria que accione uno de los botones de forma perpendicular a éste. Al igual que en el caso de la tarea de pick and place, lo interesante de esta variación reside en la existencia de dos soluciones.

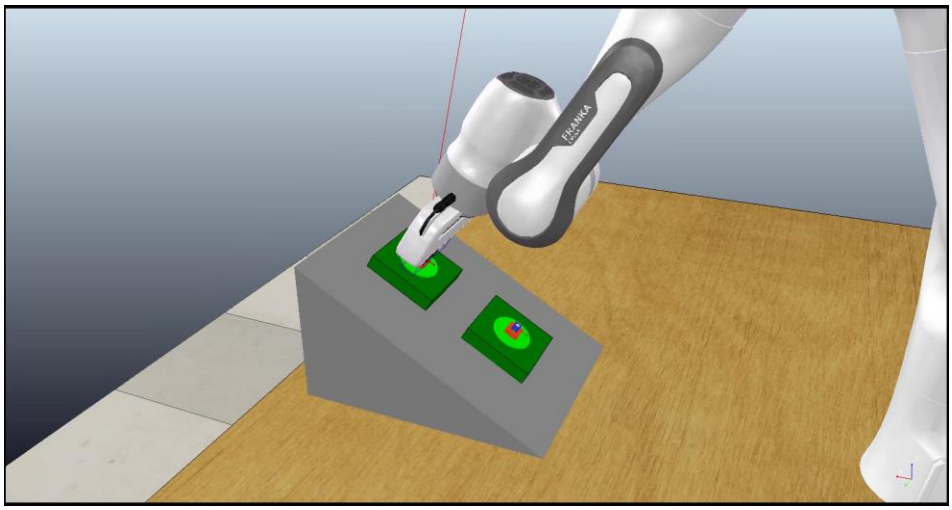


*Fig. 6.9 – Segunda variación de la tarea de apretar un botón*

### 6.3.4. Empujar objeto

#### *6.3.4.1. Variación 1*

En esta variación de la tarea tenemos un cubo delante del robot. El objetivo del robot consiste en ejecutar una trayectoria de forma que empuje el objeto desde su posición inicial a otra posición determinada.

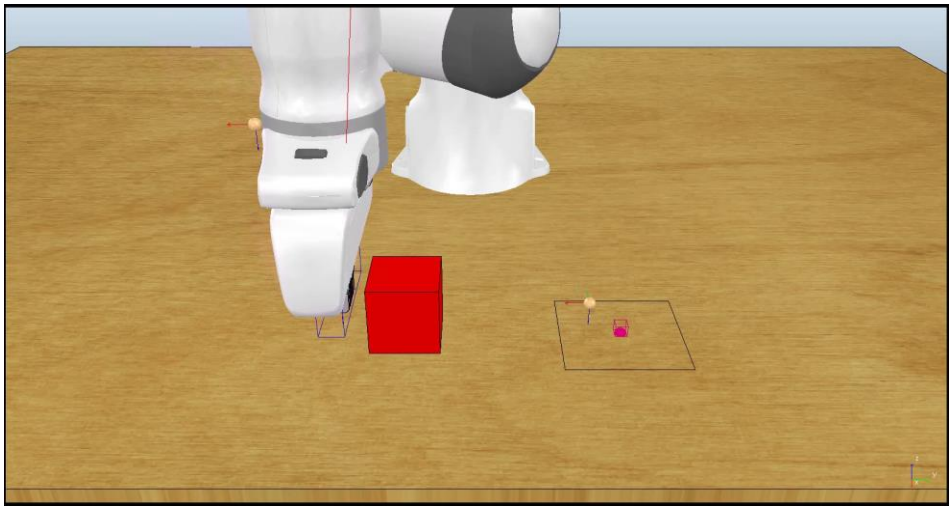


*Fig. 6.10 – Primera variación de la tarea de empujar un objeto*

#### *6.3.4.2. Variación 2*

Al igual que en la primera variación tenemos un cubo delante del robot, pero esta vez tenemos dos lugares a los que podemos empujar el cubo. Igual que en los casos anteriores, el interés de esta variación es la existencia de dos soluciones.

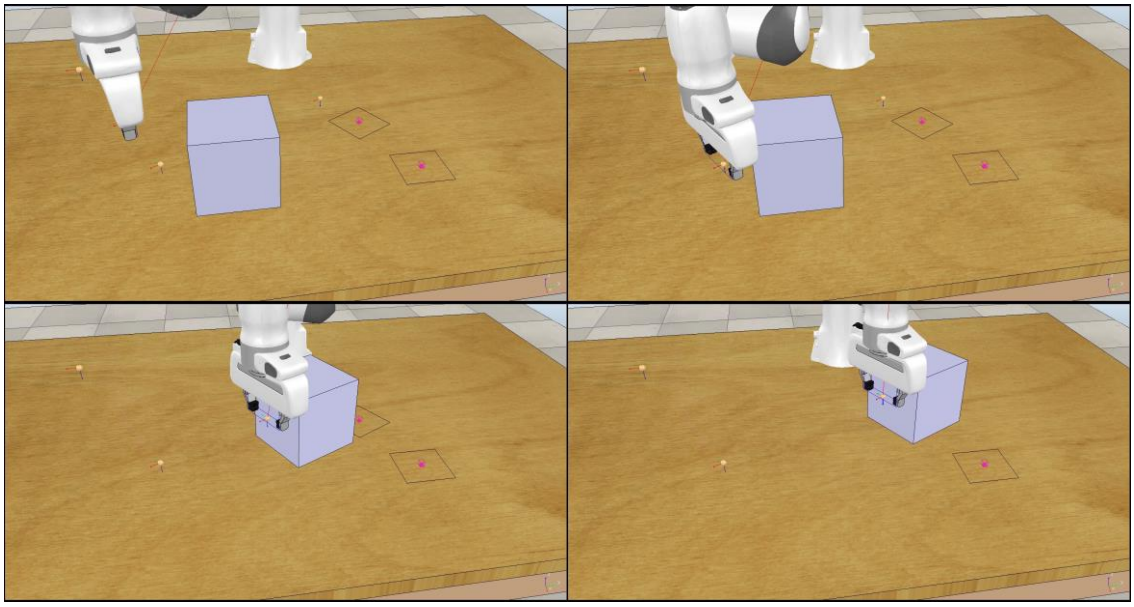


*Fig. 6.11 – Segunda variación de la tarea de empujar un objeto*

En este capítulo se han presentado el entorno robótico, el robot y se han planteado las tareas de la librería. La secuencia completa de las tareas se encuentra en el ANEXO A.

# 7. PARAMETRIZACIÓN DE LAS TAREAS

Para resolver las tareas que se han planteado mediante el aprendizaje por refuerzo, es necesario plantearlas en términos de política y recompensa. Como para resolver las tareas vamos a utilizar la técnica de búsqueda de política, tenemos que parametrizarla. En este caso, de forma intuitiva podemos entender la política del robot como la forma de moverse, es decir, la trayectoria que debe seguir el elemento terminal. Los objetivos de las tareas deben traducirse en la función de recompensa respectiva.

En este capítulo se incluye la parametrización y las funciones de recompensa de la tarea de pick and place y la tarea de empujar un objeto. La parametrización y las funciones de recompensa de todas las tareas se encuentra en el ANEXO B.

## 7.1. PICK AND PLACE

### 7.1.1. Política

Para resolver la tarea de pick and place, la política se va a parametrizar mediante cuatro waypoints en las dos variaciones.

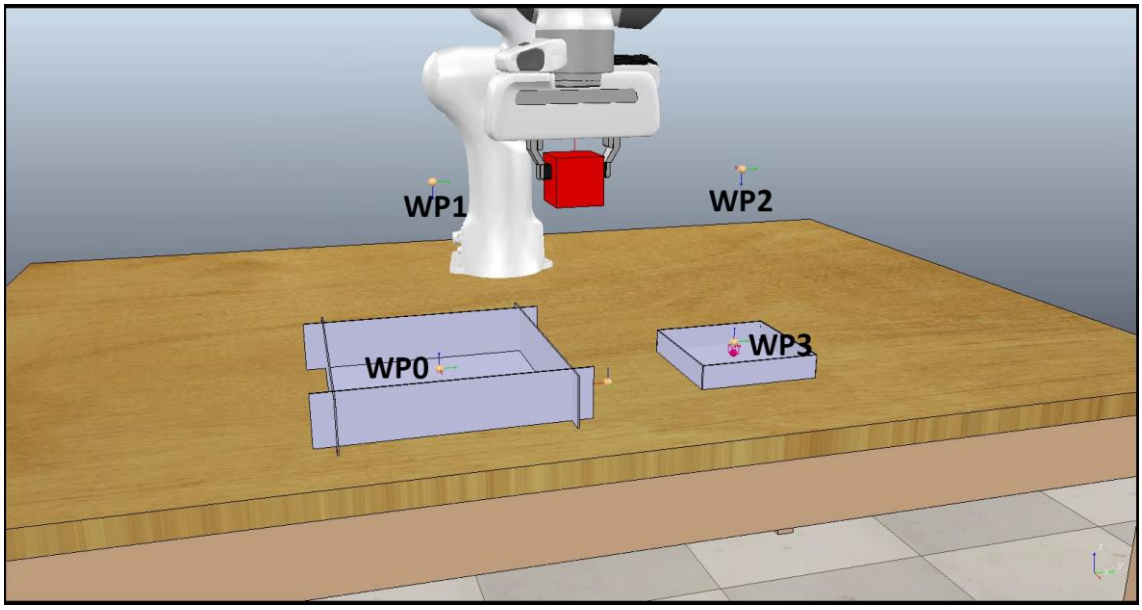



*Fig. 7.1 – Ejemplo de política para la primera variación de la tarea de pick and place*

Tenemos dos waypoints fijos, con posición y orientación conocidos, $WP1$ y $WP2$. Estos waypoints serán de aproximación al recipiente grande y el recipiente pequeño. Suponemos que son conocidos ya que, aunque no conocemos la posición exacta de los recipientes ni la del cubo, sí que podemos aproximar donde se encuentran. Los otros dos waypoints, $WP0$ y $WP3$, definen la posición en la que se coge el cubo en el recipiente grande y en la que se deposita en el recipiente pequeño, respectivamente. Para ambos waypoints, los parámetros que se tienen que aprender son solo los referentes a la posición, ya que vamos a suponer que la orientación es conocida.

Por tanto, la política será función de la posición de los dos waypoints:

$$\pi = f(P_{WP0}, P_{WP3})$$

Siendo $P_{WP0}$ y $P_{WP3}$ las posiciones de cada waypoint. La posición de $WP0$ se define respecto a la posición de $WP1$, waypoint de aproximación al recipiente grande, y la de $WP3$ se define con respecto a la posición de $WP2$, waypoint de aproximación al recipiente pequeño.

Esta parametrización es válida para las dos variaciones, lo único que cambia son los posibles valores que pueden tomar las coordenadas del waypoint $WP3$.

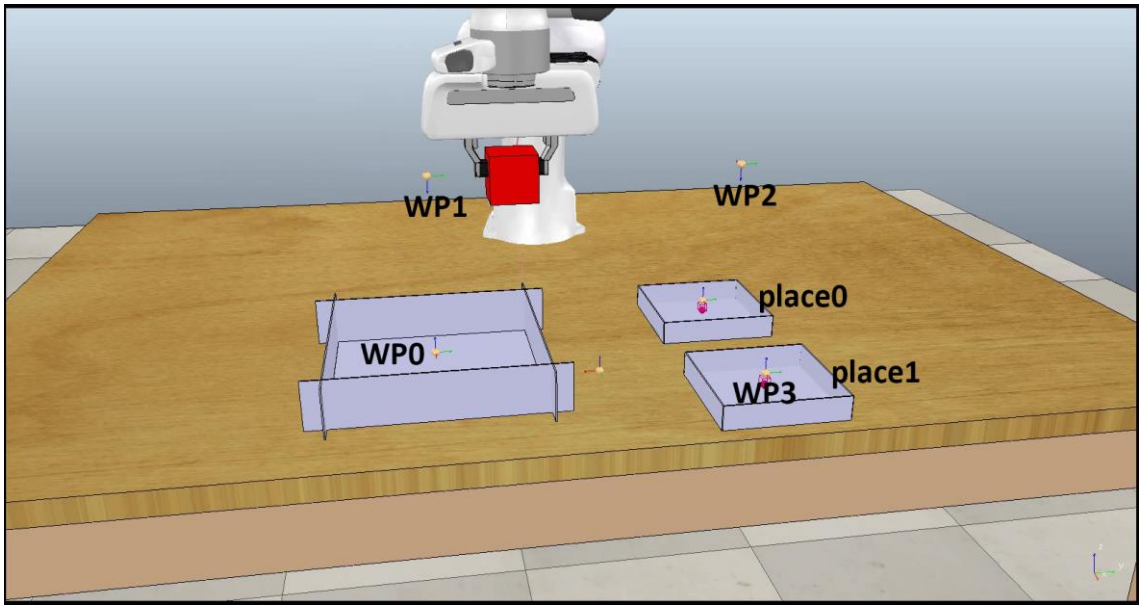



*Fig. 7.2 – Ejemplo de política para la segunda variación de la tarea de pick and place*

### 7.1.2. Recompensa

El objetivo de la tarea es coger el cubo del recipiente grande y llevarlo a un recipiente pequeño. Por tanto, la función de recompensa debe formularse en términos del éxito al coger el objeto y la posición en la que se deja el cubo. El éxito al coger el cubo se medirá como la distancia entre la pinza del robot y el cubo en su posición inicial. Para la posición final se calculará la distancia entre el cubo y la posición final deseada en el recipiente y también la distancia entre el elemento terminal del robot y la posición final deseada, ya que queremos dejar el cubo dentro del recipiente.

La distancia entre la pinza del robot y el cubo en su posición inicial será el parámetro $d_{pick}$, mientras que la distancia entre el cubo y la posición final deseada será $d_{place\_block}$ y la distancia entre el elemento terminal y la posición final deseada será $d_{place\_gripper}$.

$$R = -200 \cdot {d_{pick}}^2 - 200 \cdot {d_{place\_block}}^2 - 100 \cdot {d_{place\_gripper}}^2$$

Para la segunda variación serán necesarias $d_{pick}$, dos $d_{place\_block}$ y dos $d_{place\_gripper}$, ya que en este caso tenemos dos recipientes.

$$R = -200 \cdot {d_{pick}}^2 - 200 \cdot {d_{place0\_block}}^2 - 400 \cdot {d_{place1\_block}}^2 \\ -3500 \cdot d_{place0\_block} \cdot {d_{place1\_block}}^2 \\ -100 \cdot {d_{place0\_gripper}}^2 - 200 \cdot {d_{place1\_gripper}}^2 \\ -2500 \cdot d_{place0\_gripper} \cdot {d_{place1\_gripper}}^2$$

Esta función de recompensa supone que la posición final de un contenedor tiene el doble de recompensa que el otro. Con el producto de distancias nos aseguramos de que los mínimos de la función se encuentran en los dos recipientes.

## 7.2. EMPUJAR UN OBJETO

### 7.2.1. Política

Para resolver la tarea de empujar un objeto se ha definido la política mediante tres waypoints.

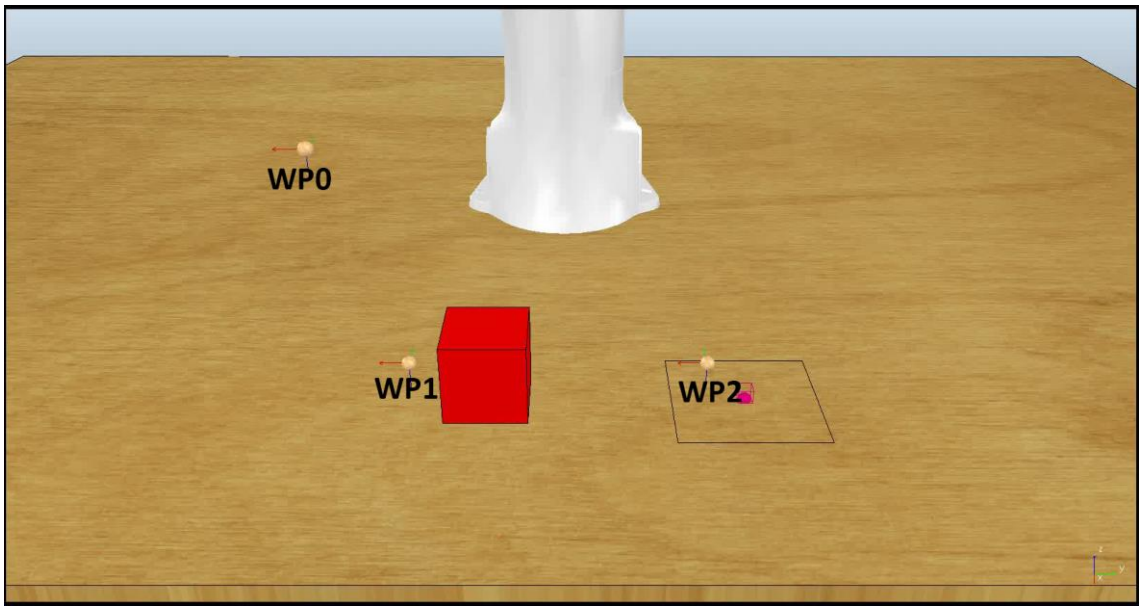



*Fig. 7.3 – Ejemplo de política para la primera variación de empujar un objeto*

Uno de los waypoints, $WP0$, es fijo, con posición y orientación conocidos, y de aproximación, ya que aunque no sepamos la posición exacta del objeto que tenemos que empujar, conocemos la zona de la mesa en la que se encuentra, y sabemos que la orientación del cubo esta entre unos límites concretos, de forma que el robot siempre va a empujar el objeto de derecha a izquierda. Otro de los waypoints, $WP1$, es el de la posición desde que el robot empieza a empujar y los parámetros que se deben aprender son la posición y la orientación. El último waypoint, $WP2$, es el de la posición final de la herramienta del robot tras empujar el objeto. Para definir la posición de este último waypoint se han utilizado dos parámetros: la distancia que debe ser empujado el objeto

y la dirección de empuje. Por tanto, la coordenada Z de $WP2$ será la misma que la de $WP1$. La orientación de empuje será también la orientación de $WP1$ y $WP2$.

Por tanto, la política depende de la posición de $WP1$ y de la distancia y orientación de empuje:

$$\pi = f(P_{WP1}, d, \varphi)$$

Siendo $\mathrm{P_{WP1}}$ la posición del waypoint $WP1$, $d$ la distancia de empuje y $\varphi$ la dirección de empuje. La posición de $WP1$ se define con respecto a la posición de $WP0$ y la posición de $WP2$ con respecto a la posición de $WP1$.

Esta parametrización es válida para las dos variaciones, lo que cambia son los posibles valores que pueden tomar la distancia y la orientación de empuje.

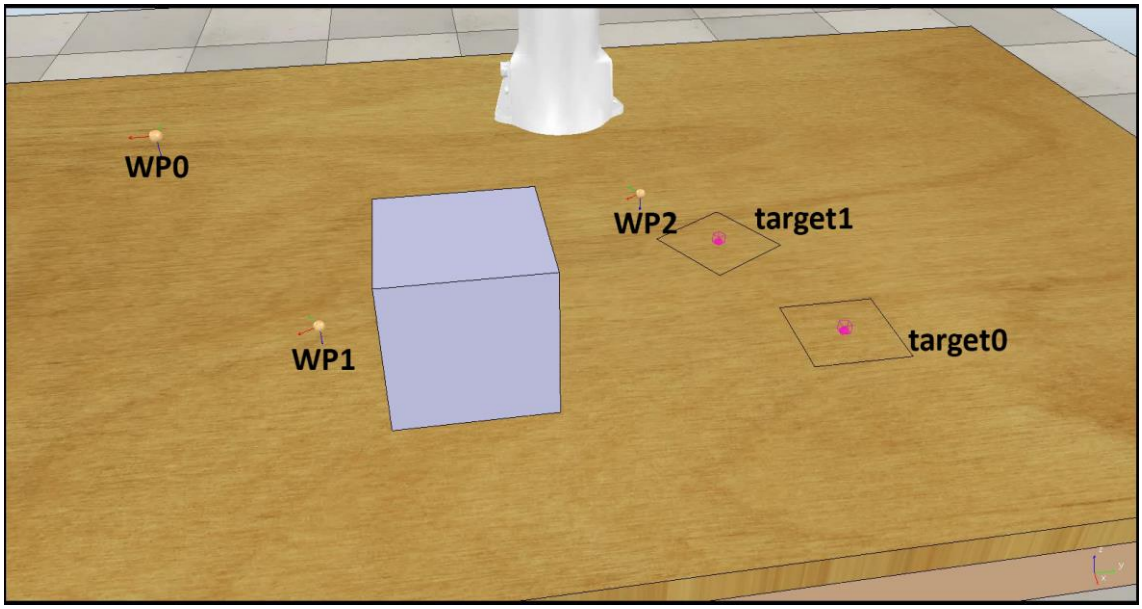



*Fig. 7.4 – Ejemplo de política para la segunda variación de empujar un objeto*

### 7.2.2. Recompensa

EL objetivo de la tarea es empujar correctamente el objeto desde una posición inicial hasta una posición final, minimizando la distancia de empuje y con una orientación correcta. Por tanto, para definir la función de recompensa se debe tener en cuenta la distancia entre la herramienta del robot y el cubo antes de empezar a empujarlo, que denominamos $d_{slide}$ , la distancia entre el cubo y la posición final deseada, que denominamos $d_{target}$ y la distancia de empuje, $d$. Para la segunda variación, también se van a tener en cuenta los errores de la orientación en el eje perpendicular a la mesa de trabajo, que denominamos $e_{or\_target}$

Para la primera variación, la función de recompensa se define como:

$$R = -10 \cdot d^2 - 200 \cdot {d_{slide}}^2 - 200 \cdot {d_{target}}^2$$

La función de recompensa para la segunda variación es:

$$R = -10 \cdot d^2 - 200 \cdot {d_{slide}}^2 - 200 \cdot {d_{target0}}^2 - 400 \cdot {d_{target1}}^2$$
$$-3500 \cdot d_{target0} \cdot d_{target1} - 200 \cdot abs(e_{or\_target0}) \cdot d_{target1}$$
$$-500 \cdot abs(e_{or\_target1}) \cdot d_{target0}$$

Al igual que en los casos anteriores, una de las soluciones tiene el doble de recompensa que la otra. Con el producto de distancias nos aseguramos de que las soluciones se encuentran en las posiciones finales deseadas. Además al multiplicar el error en la orientación en la posición final 0 con la distancia a la posición final 1 y viceversa, nos aseguramos de que si la orientación del bloque coincide con la orientación de la posición final todo ese término es cero, y es distinto de cero en el caso contrario.

En este capítulo se han parametrizado las políticas y se ha definido la función de recompensa de dos de las tareas de la biblioteca, la tarea de pick and place y la tarea de empujar un objeto. Ambas se han definido para las dos variaciones de las tareas.

# 8. OPTIMIZACIÓN Y RESULTADOS

Una vez se han diseñado, parametrizado las tareas e implementado las tareas, tanto en el simulador como en Python mediante PyRep, el siguiente paso es optimizar las tareas para encontrar los parámetros que nos dan política que menor valor de . El método de búsqueda de política es un método aproximado, por lo que la política que encontremos no tiene por qué ser la política óptima.

Para optimizar las tareas definiremos los parámetros de optimización de BayesOpt y de la evolución diferencial de SciPy, que se detallan en el capítulo 5. Esos parámetros junto a la función de la tarea se pasan al optimizador y obtenemos como resultado los parámetros de la política. Los parámetros de todas las tareas se detallan en el ANEXO C.

Para validar la solución obtenida se realizará más de un experimento. El análisis del proceso de optimización se realizará mediante gráficas de recompensa frente a iteraciones, representando para cada iteración la media y la varianza de la mejor recompensa hasta ese instante.

En este capítulo solo se presentan las optimizaciones de la tarea de pick and place y la tarea de empujar un objeto. La optimización de todas las tareas puede encontrarse en el ANEXO C.

## 8.1. PICK AND PLACE

### 8.1.1. Variación 1: un contenedor de place

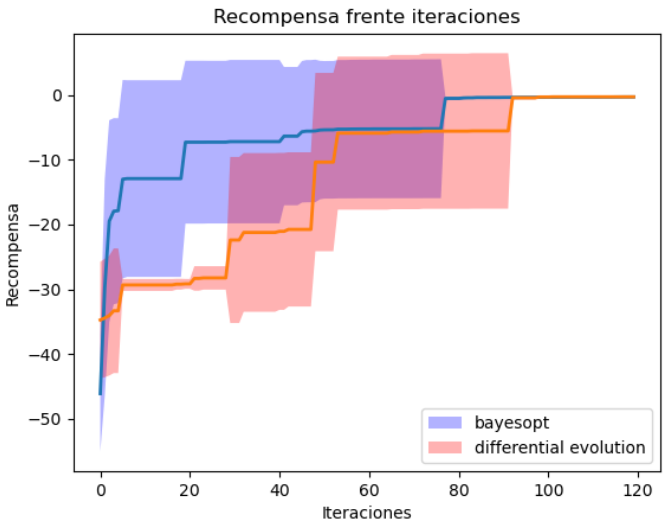



*Fig. 8.1 – Optimización de la primera variación de pick and place*

En la Fig. 8.1 podemos observar los resultados obtenidos en la optimización de la primera variación. La línea se corresponde con la media obtenida del número de experimentos y la zona sombreada es el intervalo de confianza del 95%. Desde la iteración 120 hasta el final de la optimización las dos líneas siguen rectas.

Si observamos la función de recompensa de esta tarea, definida en el apartado 7.1.2, y la forma en que se parametriza la política, en el apartado 7.1.1, podemos ver que esta tarea tiene una única solución, en la que los posición de los waypoints es tal que la distancia de pick y la distancia de place es 0, de forma que la recompensa es 0. En la gráfica podemos ver que aproximadamente a partir de la iteración 80 BayesOpt ha encontrado esta solución en todos los experimentos, ya que es valor de la media de la recompensa es 0 y no hay intervalo de confianza, mientras que a la evolución diferencial le cuesta unas iteraciones más.

### 8.1.2. Variación 2: dos contenedores de place

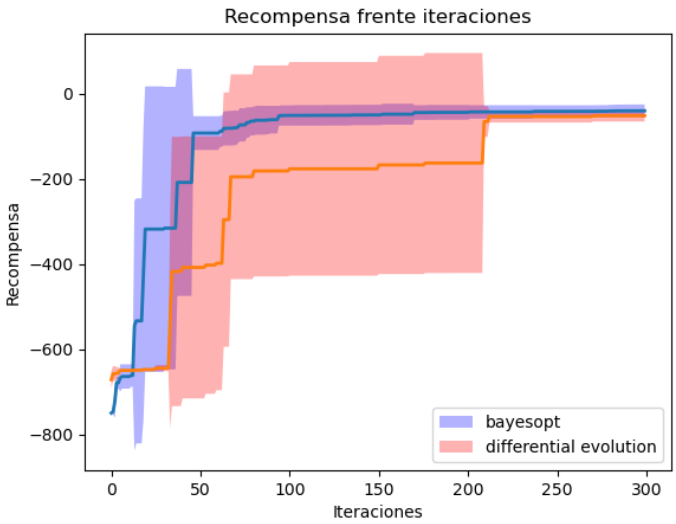



*Fig. 8.2 – Optimización de la segunda variación de pick and place*

En la Fig. 8.2 podemos observar los resultados obtenidos en la optimización de esta segunda variación. La línea se corresponde con la media obtenida del número de experimentos y la zonza sombreada es el intervalo de confianza del 95%. Desde la iteración 300 al final de la optimización ambas líneas siguen rectas.

Al igual que ocurre en la primera variación, de la forma en la que se define esta tarea, existen dos soluciones, según se lleve el cubo a un contenedor o al otro. Podemos ver que tanto BayesOpt como con la evolución diferencial encontramos las soluciones. También podemos ver que BayesOpt converge antes hacia las soluciones que la evolución diferencial.

Analizando por separado las soluciones de cada experimento, vemos que con BayesOpt encontramos el contenedor más alejado del robot en cuatro ocasiones y el más cercano en una, mientras que la evolución diferencial encuentra el contenedor más alejado en todos los experimentos.

## 8.2. EMPUJAR UN OBJETO

### 8.2.1. Variación 1: una posición final

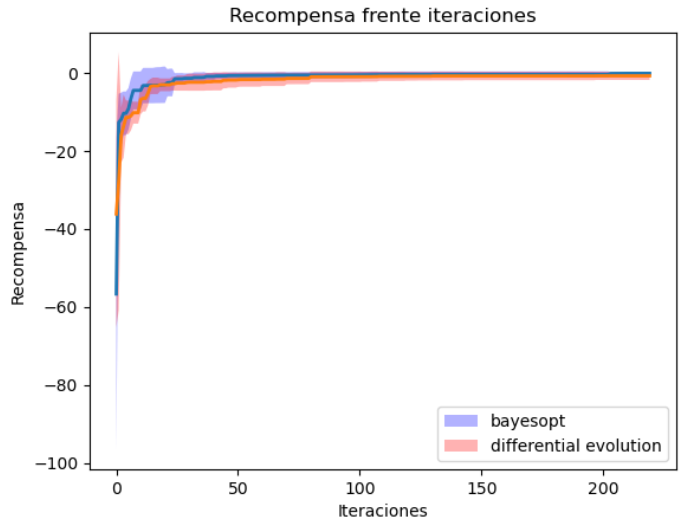



*Fig. 8.3 – Optimización para la primera variación de empujar un objeto*

En la Fig. 8.3 podemos observar los resultados obtenidos en la optimización de la primera variación. La línea se corresponde con la media obtenida del número de experimentos y la zonza sombreada es el intervalo de confianza del 95%.

Para esta variación de la tarea solo existe una solución, cuando el objeto es empujado a la posición final deseada. Viendo la función de recompensa definida en 7.2.2, en esa situación el valor es 0. Analizando la gráfica podemos ver que tanto con BayesOpt como con la evolución diferencial se encuentra la solución, aunque con BayesOpt converge unas pocas iteraciones antes.

### 8.2.2. Variación 2: dos posiciones finales

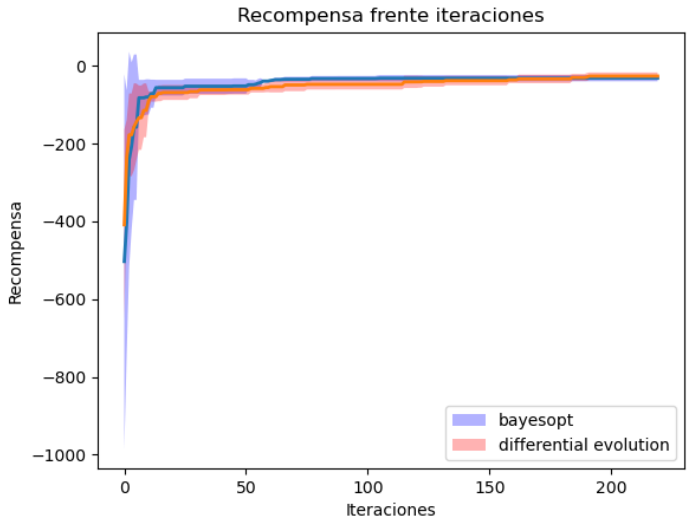



*Fig. 8.4 – Optimización para la segunda variación de empujar un objeto*

En la Fig. 8.4 podemos observar los resultados obtenidos en la optimización de la segunda variación. La línea se corresponde con la media obtenida del número de experimentos y la zonza sombreada es el intervalo de confianza del 95%. Desde la iteración 210 hasta el final de la optimización ambas líneas se mantienen igual.

En la segunda variación tenemos dos posibles soluciones ya que tenemos dos posibles posiciones finales para el bloque. Analizando la gráfica, podemos ver que tanto BayesOpt como la evolución diferencial encuentran alguna de las soluciones, y BayesOpt converge antes aunque la evolución diferencial termina encontrando una solución mejor. Esto puede deberse a que BayesOpt encuentra primero una solución, pero la solución que encuentra suele ser la que da menos recompensa y aunque la evolución diferencial tarda más en encontrar la solución, termina encontrando la solución de mayor recompensa en más ocasiones que BayesOpt. Si analizamos la solución de cada experimento por separado, vemos que BayesOpt solo ha encontrado la solución de mayor recompensa en una ocasión, mientras que con la evolución diferencial la encuentra en tres ocasiones.

## 8.3. CONCLUSIONES DE LA OPTIMIZACIÓN

En general, podemos ver que algunas de las tareas planteadas son demasiado fáciles para cualquier algoritmo de aprendizaje por refuerzo, ya sea porque la función de recompensa definida es sencilla y solo tiene un mínimo (p. ej. las primeras variaciones de las tareas de pick and place y de apretar un botón) o porque existe un gran número de posibilidades, por lo que encontrar una solución es sencillo (p. ej. la primera variación de evitación de un obstáculo, en la que para evitar la esfera con trayectorias con la misma longitud hay infinitas soluciones posibles), y hacen falta pocas iteraciones hasta encontrar una solución. De los optimizadores podemos decir que, en general, BayesOpt funciona mejor que la evolución diferencial, ya que suele necesitar menos iteraciones para encontrar una solución, sobre todo cuando hay una componente relacionada con la orientación, como se ve en el caso de la tarea de apretar un botón.

En este capítulo se han presentado los resultados de la optimización de dos de las tareas y sus variaciones. Se ha analizado el proceso de optimización en términos de la mejor recompensa obtenida en cada iteración y se han analizado las soluciones de cada experimento por separado en los casos que era necesario. También se han presentado las conclusiones generales de las optimizaciones.

# 9. CONCLUSIONES DEL TRABAJO

En este trabajo ha llevado a cabo el diseño e implementación de una biblioteca de tareas robóticas de manipulación en un simulador 3D como banco de pruebas para algoritmos de aprendizaje por refuerzo, utilizando la técnica de búsqueda de política.

En primer lugar, se ha llevado a cabo una revisión bibliográfica del aprendizaje por refuerzo, introduciendo los conceptos básicos de la materia, explicando los procesos de decisión de Márkov como el principal marco de referencia para plantear y solucionar los problemas de aprendizaje por refuerzo. De los posibles métodos de soluciones, se ha incidido en la búsqueda de política, el método implementado en el trabajo. Dentro de la búsqueda de política se han explicado los métodos evolutivos y la optimización bayesiana.

Posteriormente, se ha llevado a cabo una revisión bibliográfica de la optimización bayesiana, por ser un método de optimización global y que permite optimizar funciones cuya forma se desconoce, llamadas funciones black-box. De la optimización bayesiana se han explicado los conceptos básicos y el algoritmo tipo.

A continuación, se incluye una explicación de las herramientas de simulación de robots utilizadas en el trabajo. Se incluyen el simulador de robótica CoppeliaSim, con el que se construyen las escenas y se simulan las tareas y el kit de herramientas de PyRep, que permite controlar la simulación en Python. También se incluyen las herramientas de optimización que se utilizan en el trabajo, BayesOpt y la evolución diferencial de SciPy, detallando los parámetros de optimización más importantes.

El siguiente paso ha sido el diseño de las tareas implementadas. Se han diseñado cuatro tareas, una de evitación de obstáculos, una de pick and place, una de apretar un botón y una última de empujar un objeto. Para todas las tareas se ha diseñado una variación sencilla, en la que existen soluciones sencillas o una única solución, y otra variación más compleja, en la que existe más de una solución, para forzar al optimizador a encontrar las dos soluciones y tener dos mínimos en las funciones de recompensa.

Una vez diseñadas las tareas se han parametrizado las políticas y las funciones de recompensa de cada tarea y variación, haciendo hincapié en que la función de recompensa de las tareas debe traducir el objetivo. Después las tareas se han implementado tanto en CoppeliaSim con las escenas de simulación como en Python con PyRep.

Por último se ha llevado a cabo la optimización de las tareas para encontrar los parámetros de la política óptima. Para cada tarea se han especificado los parámetros de optimización utilizados y se han analizado los resultados obtenidos en la simulación, tanto cualitativamente, con la mejor recompensa obtenidas en cada iteración, como cuantitativamente al analizar cada solución por separado en los casos que era necesario. De manera general, se concluye que algunas tareas son demasiado fáciles para cualquier algoritmo de aprendizaje por refuerzo, ya que solo tienen una solución, y BayesOpt funciona mejor que la evolución diferencial, como se ve en los resultados obtenidos.

# 10. BIBLIOGRAFÍA


[1] R. Sutton y A. Barto, Reinforcement Learning: An Introduction, 2018.

[2] D. Klein y P. Abbeel, «UC Berkeley CS188 Intro to AI: MDPs and Reinforcement Learning lectures».

[3] E. Brochu, V. M. Cora y N. de Freitas, «A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning,» 2010.

[4] D. E. Moriarty, A. C. Schultz y J. J. Grefenstette, «Evolutionary Algorithms for Reinforcement Learning,» *Journal of Articial Intelligence Research,* vol. 11, pp. 241-276, 1999.

[5] K. O. Stanley y R. Miikkulainen, «Evolving Neural Networks through Augmenting Topologies,» *Evolutionary Computation,* vol. 10, nº 2, pp. 99-127, 2002.

[6] R. Storn y K. Price, «Differential Evolution: A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces,» *Journal of Global Optimization,* vol. 23, p. 341–359, 1995.

[7] J. Nogueira, R. Martinez-Cantin, A. Bernardino y L. Jamone, «Unscented Bayesian Optimization for Safe Robot Grasping,» de *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, 2016.

[8] R. Martinez-Cantin, «Funneled Bayesian Optimization for Design, Tuning and Control of Autonomous Systems,» *IEEE Transactions on Cybernetics,* vol. 49, nº 4, pp. 1489-1500, 2019.

[9] E. Rohmer, S. P. N. Sigh y M. Freese, «CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework,» de *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[10] S. James, M. Freese y A. J. Davison, «PyRep: Bringing V-REP to Deep Robot Learning,» 2019.

[11] R. Martínez Cantín, «BayesOpt: A Bayesian Optimization Library for Nonlinear Optimization, Experimental Design and Bandits.,» *Journal of Machine Learning Research,* 2014.

[12] «Scipy.org: differential evolution,» [En línea]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html. [Último acceso: 19 11 2020].

[13] «Robot Panda de Franka Emika,» [En línea]. Available: https://www.franka.de/technology. [Último acceso: 19 11 2020].

# ANEXO A. SECUENCIA COMPLETA DE TODAS LAS TAREAS

## A.1. EVITACIÓN DE UN OBSTÁCULO

### A.1.1. Variación 1

En esta variación delante del robot tenemos un objeto con forma esférica. El objetivo del robot consiste en ejecutar una trayectoria lo más corta posible de forma que ni el elemento terminal ni ninguno de los eslabones choque con el obstáculo.

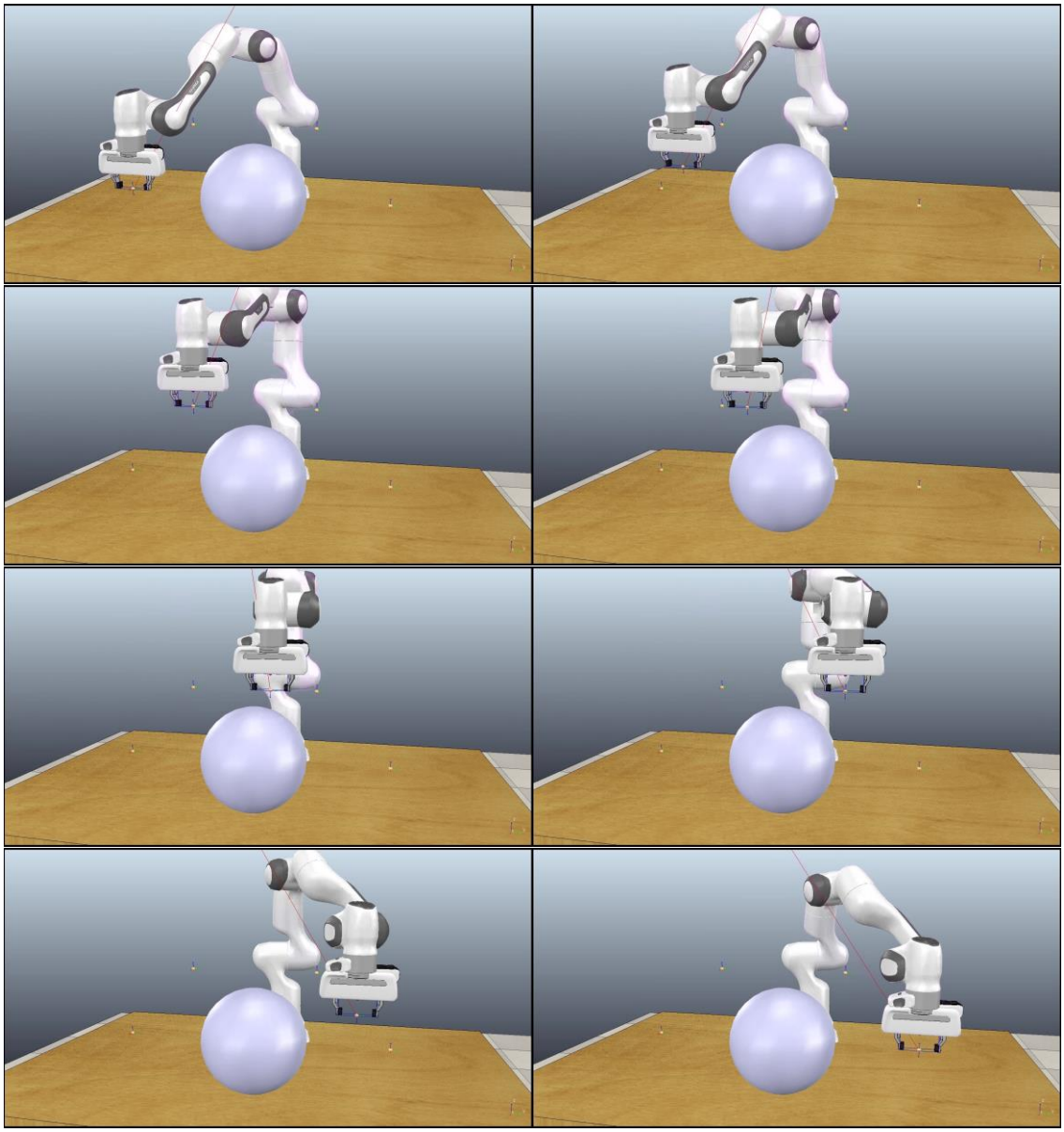


*Fig. A.1 – Secuencia de la primera variación de la tarea de evitación de un obstáculo.*

### A.1.2. Variación 2

En esta variación delante del robot tenemos tres cilindros con distinta separación entre ellos. El objetivo del robot consiste en ejecutar una trayectoria plana lo más corta posible de forma que el robot no se choque con los obstáculos. El interés de esta variación con respecto a la primera es que el robot va a tener dos posibilidades, ejecutar la trayectoria por cualquiera de las dos aberturas entre cilindros. Una de ellas será más difícil que la otra, ya que la distancia entre cilindros será es menor, pero también será más corta.

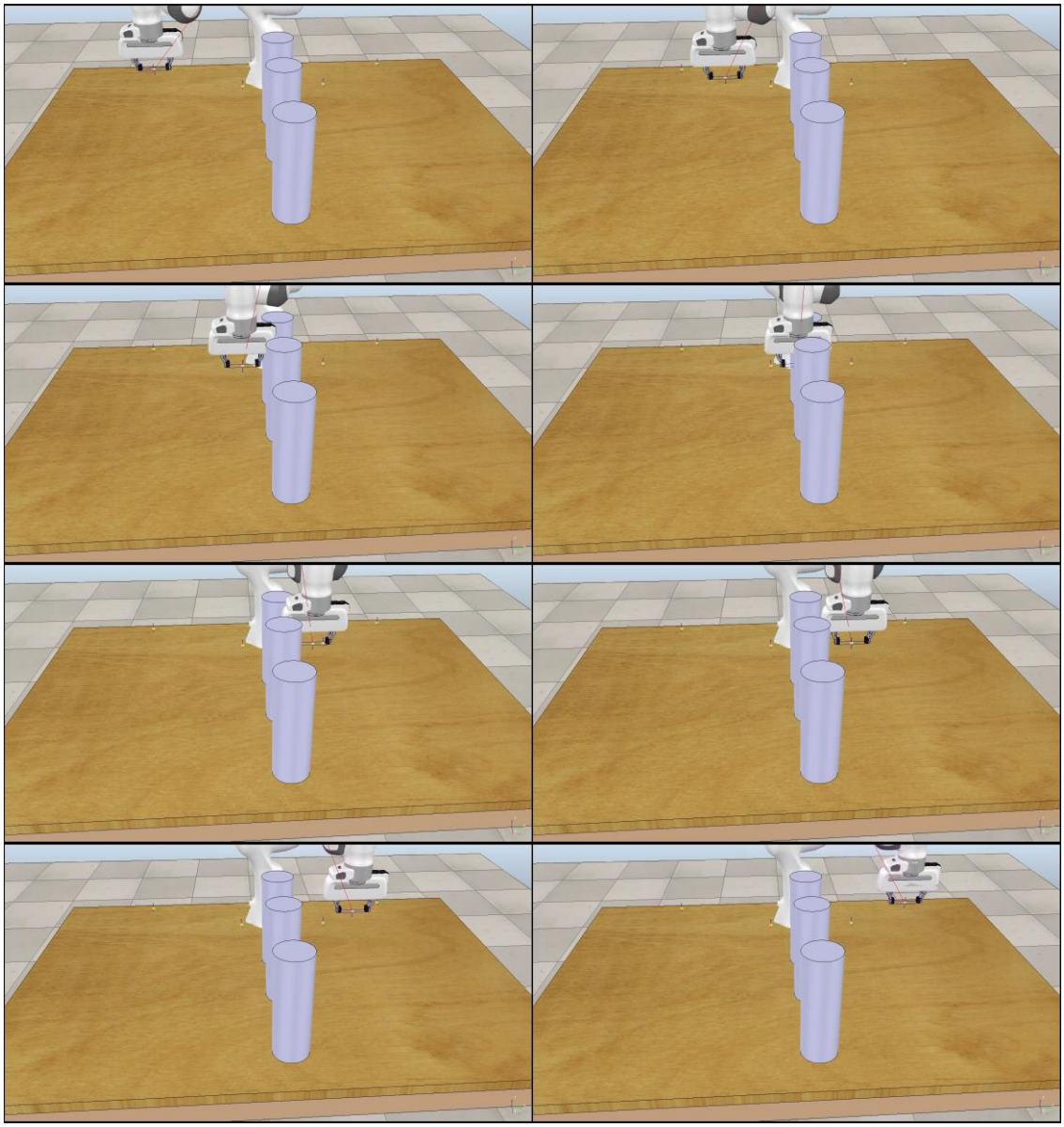


*Fig. A.2 – Secuencia de la segunda variación de la tarea de evitación de un obstáculo*

## A.2. PICK AND PLACE

### A.2.1. Variación 1

Para esta variación de la tarea tenemos dos recipientes delante del robot. Uno está a la derecha, es más grande y en él se encuentra un objeto con forma cubo, y el otro está a la izquierda y es más pequeño. El objetivo del robot consiste en ejecutar una trayectoria de forma que coja el objeto del recipiente más grande y lo deposite dentro del recipiente pequeño. Para ello deberá cerrar la pinza cuando llegue a la posición inicial del cubo para cogerlo y deberá abrirla en la posición final para depositarlo.

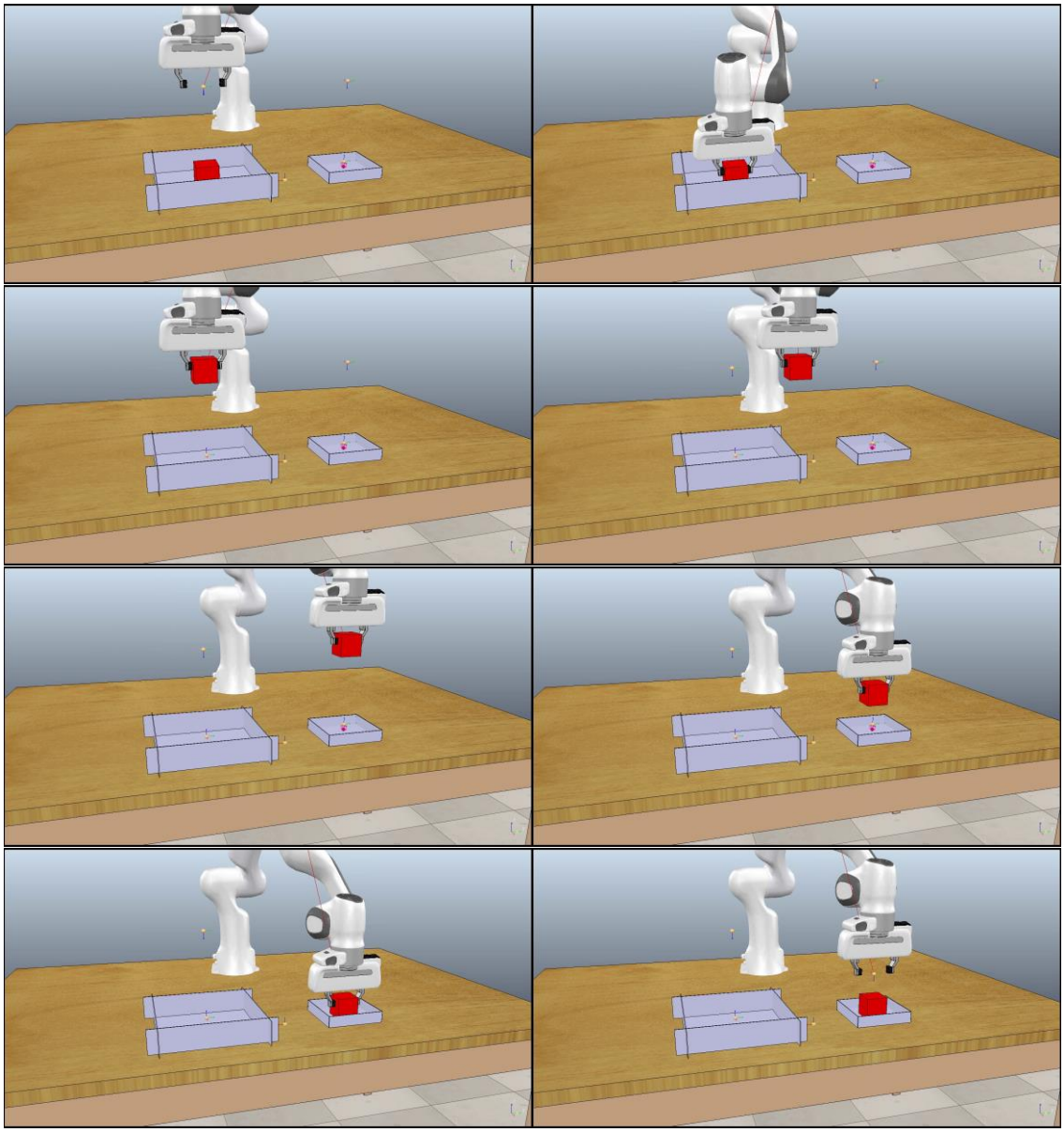


*Fig. A.3 – Secuencia de la primera variación de la tarea de pick and place*

### A.2.2. Variación 2

En esta variación delante del robot tenemos tres recipientes. El recipiente de la derecha es el misma, pero a la izquierda tenemos dos recipiente pequeños en lugar de uno. El objetivo del robot consiste en ejecutar una trayectoria de forma que coja el objeto del recipiente más grande y lo deposite en uno de los recipientes pequeños. Lo interesante de esta variación es que, al contrario que en la variación 1, tenemos dos soluciones.

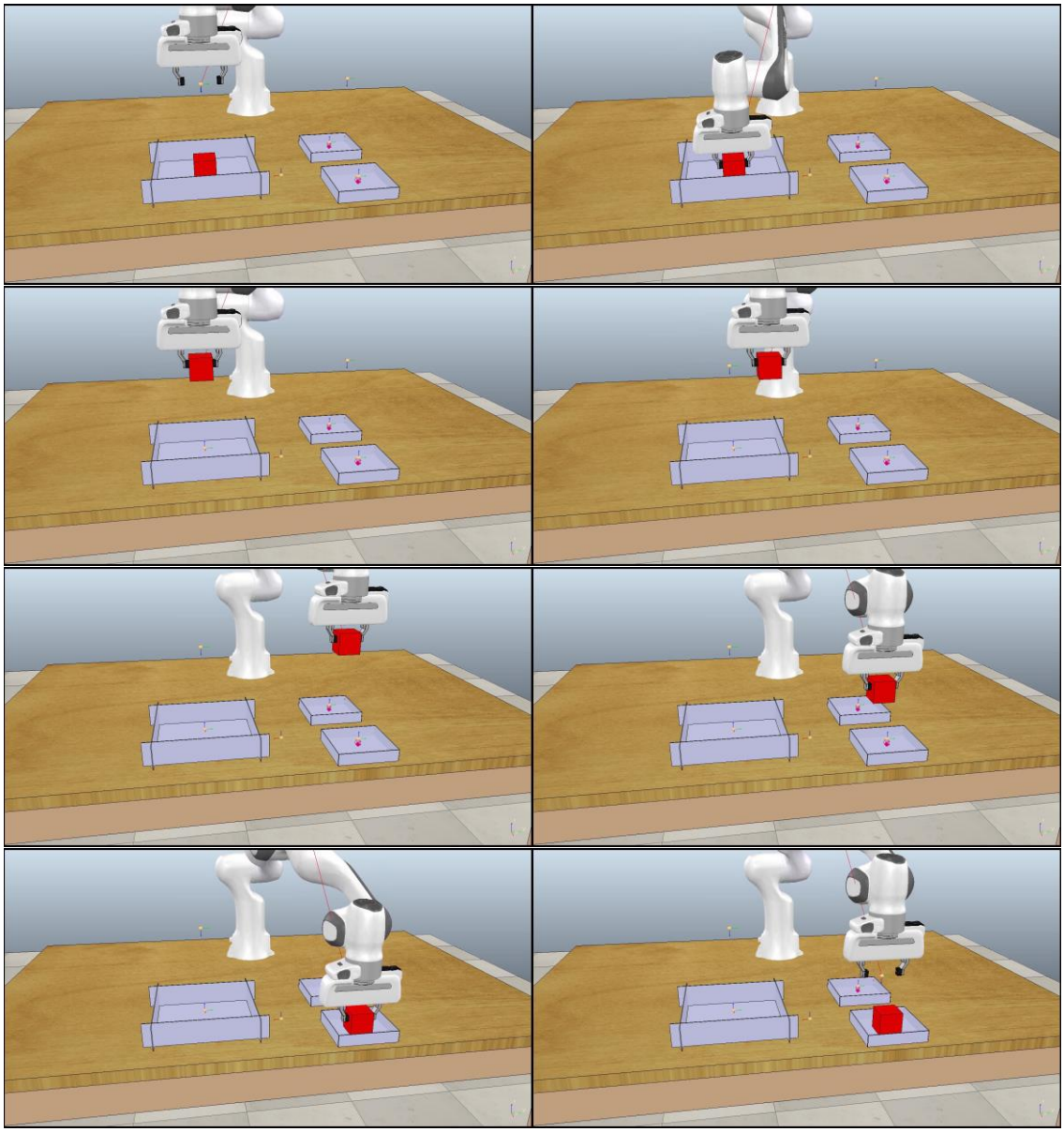


*Fig. A.4 – Secuencia de la segunda variación de la tarea de pick and place*

## A.3. APRETAR UN BOTÓN

### A.3.1. Variación 1

En esta variación de la tarea tenemos un botón en un plano inclinado, simulando una mesa de controles. El objetivo del robot consiste en ejecutar una trayectoria de forma que accione el accione el botón de forma perpendicular a éste.

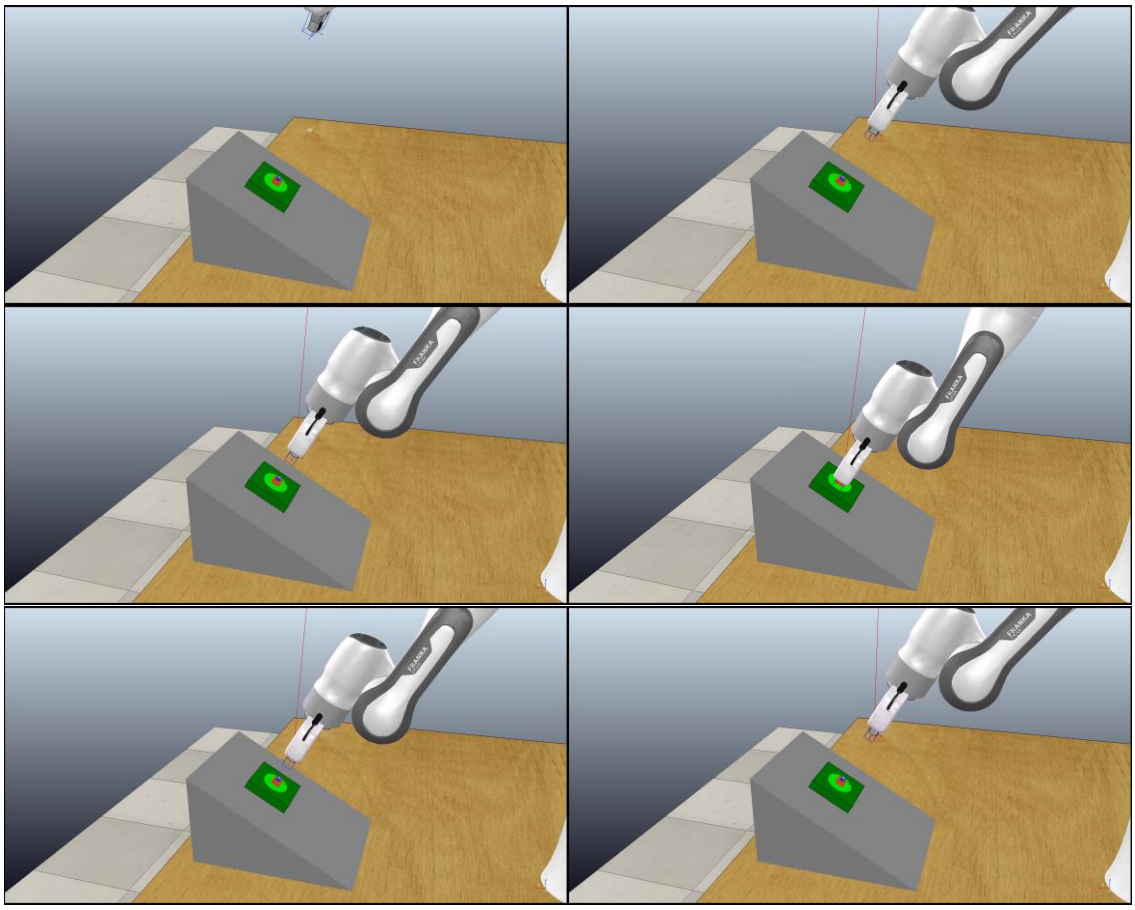


*Fig. A.5 – Secuencia de la primera variación de la tarea de apretar un botón*

### A.3.2. Variación 2

A diferencia de la primera variación, en esta ocasión tenemos dos botones. El objetivo del robot consiste en ejecutar una trayectoria que accione uno de los botones de forma perpendicular a éste. Al igual que en el caso de la tarea de pick and place, lo interesante de esta variación reside en la existencia de dos soluciones.

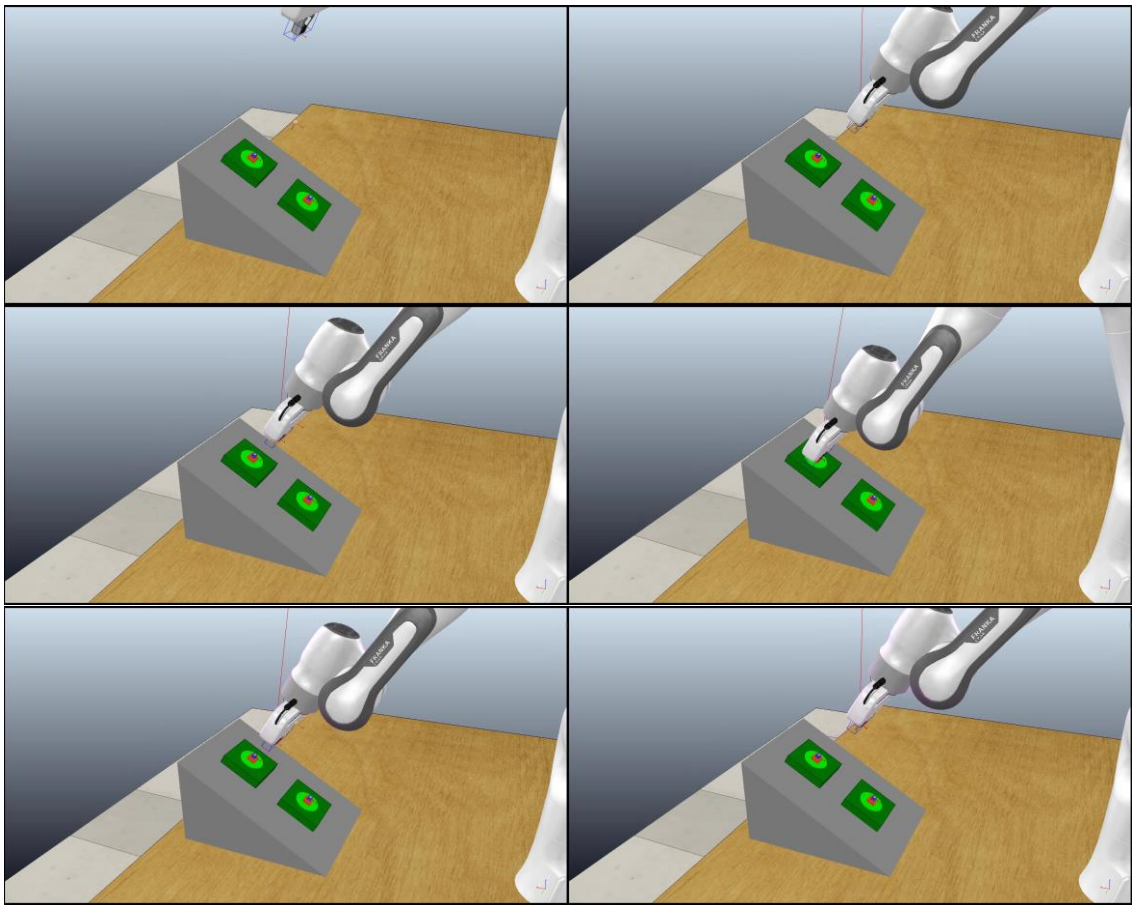


*Fig. A.6 – Secuencia de la segunda variación de la tarea de apretar un botón*

## A.4. EMPUJAR UN OBJETO

### A.4.1. Variación 1

En esta variación de la tarea tenemos un cubo delante del robot. El objetivo del robot consiste en ejecutar una trayectoria de forma que empuje el objeto desde su posición inicial a otra posición determinada.

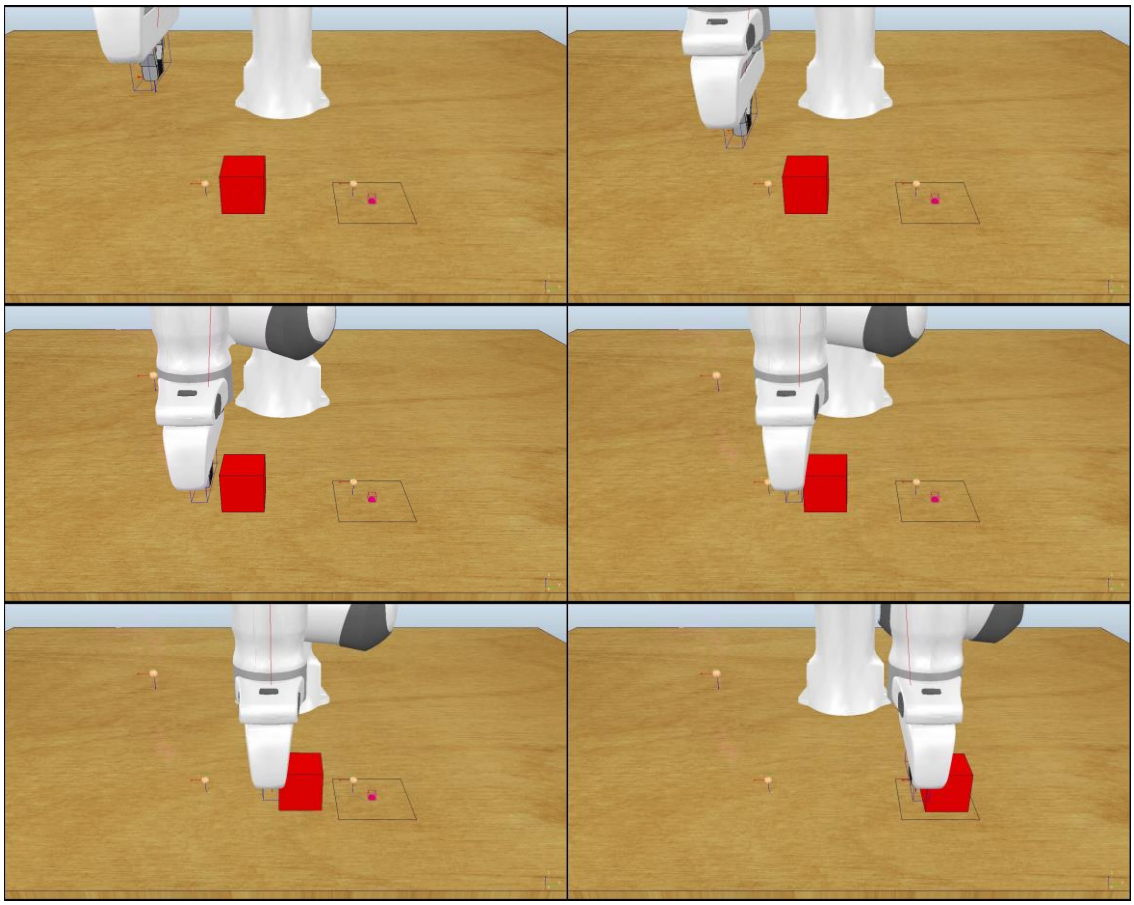


*Fig. A.7 – Secuencia de la primera variación de la tarea de empujar un objeto*

### A.4.2. Variación 2

Al igual que en la primera variación tenemos un cubo delante del robot, pero esta vez tenemos dos lugares a los que podemos empujar el cubo. El objetivo del robot consiste en ejecutar una trayectoria de forma que empuje el objeto desde su posición inicial a una de las posiciones finales. Igual que en los casos anteriores, el interés de esta variación es la existencia de dos soluciones.

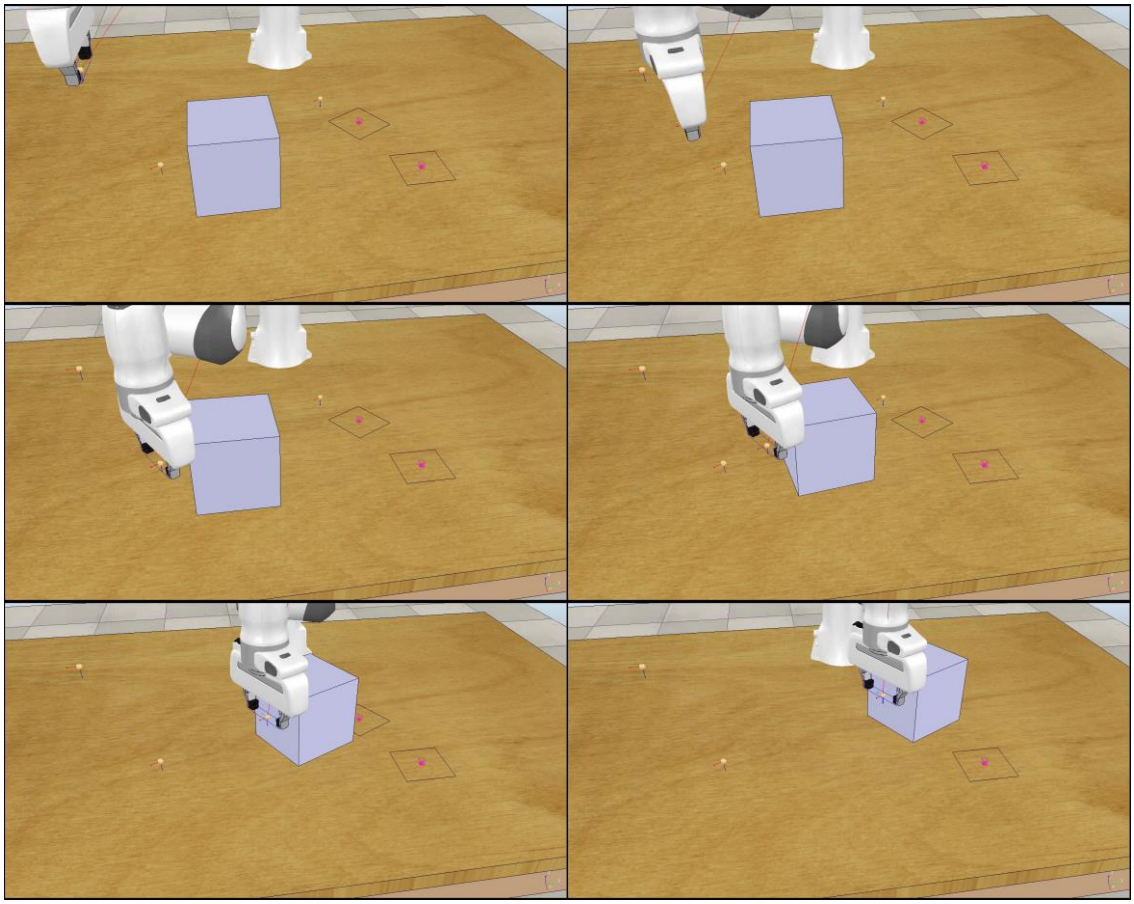


*Fig. A.8 – Secuencia de la segunda variación de la tarea de empujar un objeto*

# ANEXO B. PARAMETRIZACIONES DE LAS POLÍTICAS Y FUNCIONES DE RECOMPENSA DE TODAS LAS TAREAS

En este anexo se presentan las parametrizaciones de las políticas y las funciones de recompensa de todas las tareas con sus variaciones que han sido implementadas en la librería.

## B.1. EVITACIÓN DE UN OBSTÁCULO

### B.1.1. Política

Para resolver la tarea de evitación de obstáculo la política se va a parametrizar mediante cuatro waypoints para las dos variaciones. Los waypoints son puntos en el espacio, con posición y orientación, por los que debe pasar la trayectoria del robot.

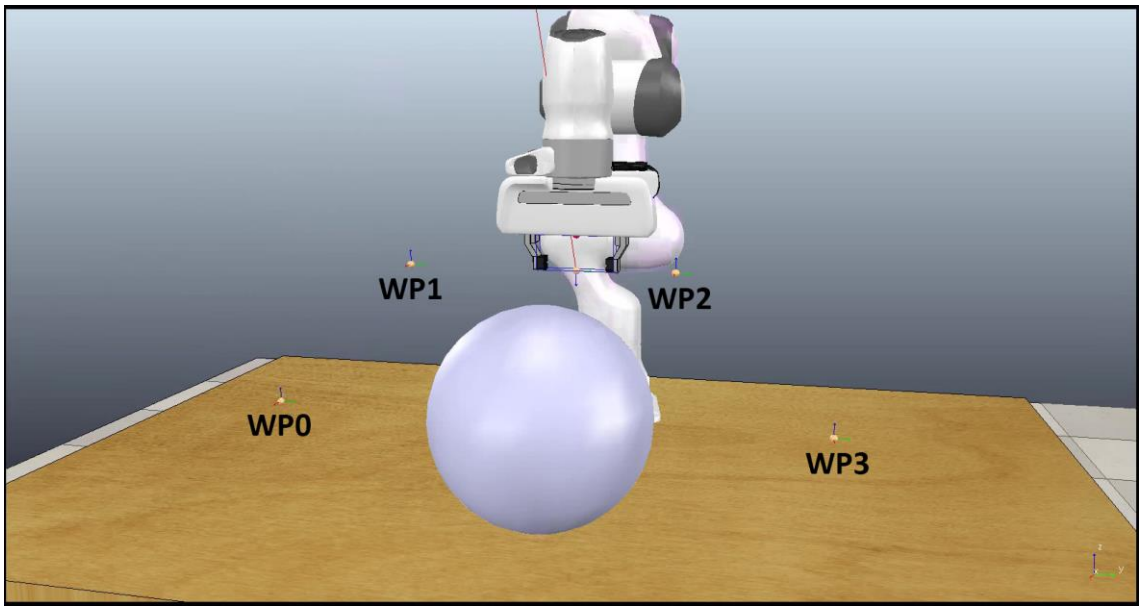



*Fig. B.1 – Ejemplo de política para la primera variación de la tarea de evitación de un obstáculo.*

Tenemos dos waypoints fijos con posición y orientación conocidos, $WP0$ y $WP3$, que serán los puntos inicial y final de la trayectoria. Vamos a suponer que son conocidos ya que se encuentran suficientemente alejados del obstáculo, por lo que no habría posibilidad de colisión. Luego habrá dos waypoints, $WP1$ y $WP2$, que serán los que definan la trayectoria entre el punto inicial y final, con orientación conocida. La posición de estos dos waypoints serán los parámetros que se tendrán que aprender.

Por tanto la política será función de la posición de estos dos waypoints:

$$\pi = f(P_{WP1}, P_{WP2})$$

Siendo $P_{WP1}$ y $P_{WP2}$ las posiciones de cada waypoint. La función $f$ viene implementada en PyRep y permite calcular un path entre dos waypoints.

Las posiciones de los dos waypoints se van a definir de dos formas distintas. En ambos casos, la posición de $WP1$ se va a definir respecto a la posición de $WP0$ y la posición de $WP2$ se va a definir respecto a la posición de $WP1$. En un caso las posiciones se definirán en coordenadas cartesianas, siendo los parámetros las coordenadas relativas XYZ para cada waypoint. En el otro caso, las posiciones se definirán en coordenadas esféricas, siendo los parámetros un radio y dos ángulos para cada waypoint.

La diferencia entra las políticas de las dos variaciones es que para la primera vamos a buscar trayectorias tridimensionales, por lo que se van a aprender las tres coordenadas XYZ de los waypoints, mientras que para la segunda variación la coordenada Z se va a mantener constante y solo se van a aprender las coordenadas XY. De esta forma, forzamos que la trayectoria sea plana y se tengan que evitar los tres obstáculos pasando por las aberturas que hay entre ellos. Para esta segunda variación solo se ha implementado la política en coordenadas polares.

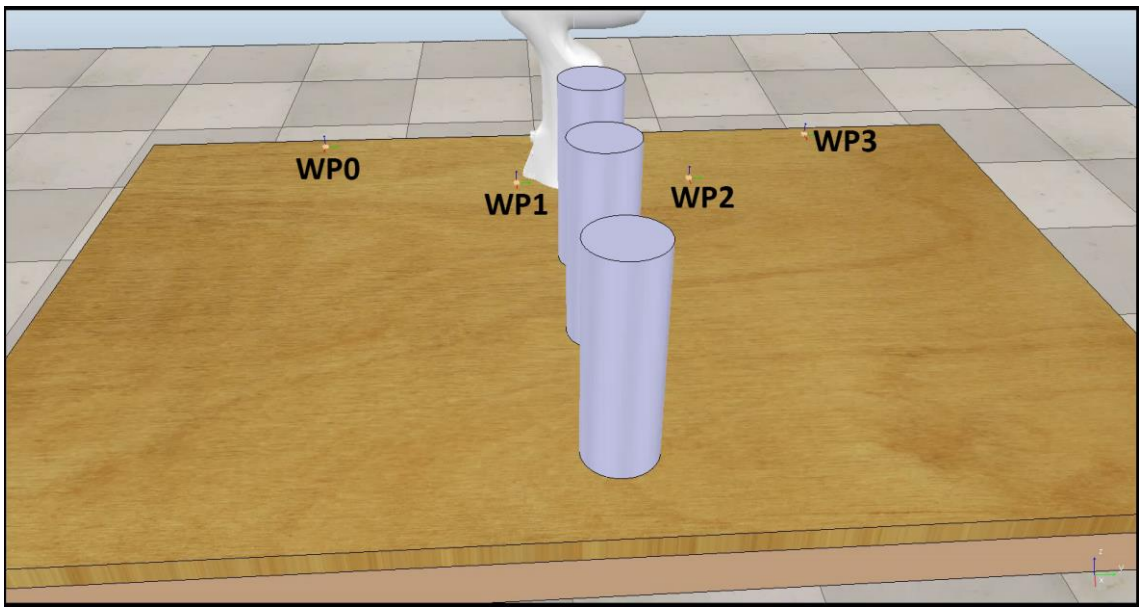



*Fig. B.2 – Ejemplo de política para la segunda variación de evitar un obstáculo*

### B.1.2. Recompensa

El objetivo de la tarea es ejecutar una trayectoria sin que el elemento terminal ni los eslabones del robot colisionen con los obstáculos y que sea lo más corta posible. Por tanto, la función de recompensa debe formularse en términos de longitud de la trayectoria y distancia del elemento terminal y del robot a los obstáculos. La distancia a los obstáculos se deberá calcular en cada paso de la trayectoria, para asegurar que el robot no colisiona en ningún instante.

Por tanto, en la primera variación tenemos como parámetros de la función la longitud de la trayectoria y la distancia al obstáculo del robot y del elemento terminal. Para la segunda variación tenemos las distancias a cada uno de los obstáculos y la longitud de la trayectoria. En el caso de la segunda variación la distancia del obstáculo al robot no es necesaria porque dada la coordenada Z de la trayectoria, el robot no puede colisionar con los obstáculos, solo el elemento terminal.

La función de recompensa para la primera variación se define de la siguiente forma:

$$R = -4 \cdot l_{tray} - \sum_{i=0}^{T} \left(20 \cdot e^{-150 \cdot d_{obstacle\_gripper}} + 20 \cdot e^{-150 \cdot d_{obstacle\_robot}}\right)$$

siendo $l_{tray}$ la longitud de la trayectoria, $d_{obstacle\_gripper}$ la distancia entre el elemento terminal y el obstáculo, y $d_{obstacle\_robot}$ la distancia entre el robot y el obstáculo.

Para la segunda variación, la función de recompensa se define de:

$$R = -4 \cdot l_{tray} - \sum_{i=0}^{T} (20 \cdot e^{-300 \cdot d_{obs0}} + 20 \cdot e^{-300 \cdot d_{obs1}} + 20 \cdot e^{-300 \cdot d_{obs2}})$$

siendo $l_{tray}$ la longitud de la trayectoria, y $d_{obs}$ la distancia a cada obstáculo.

Para la primera variación se define una recompensa de -300 para el caso en el que no se encuentra una trayectoria posible que pase por los waypoints, mientras que para la segunda variación es de -400.

## B.2. PICK AND PLACE

### B.2.1. Política

Para resolver la tarea de pick and place, la política se va a parametrizar mediante cuatro waypoints en las dos variaciones.

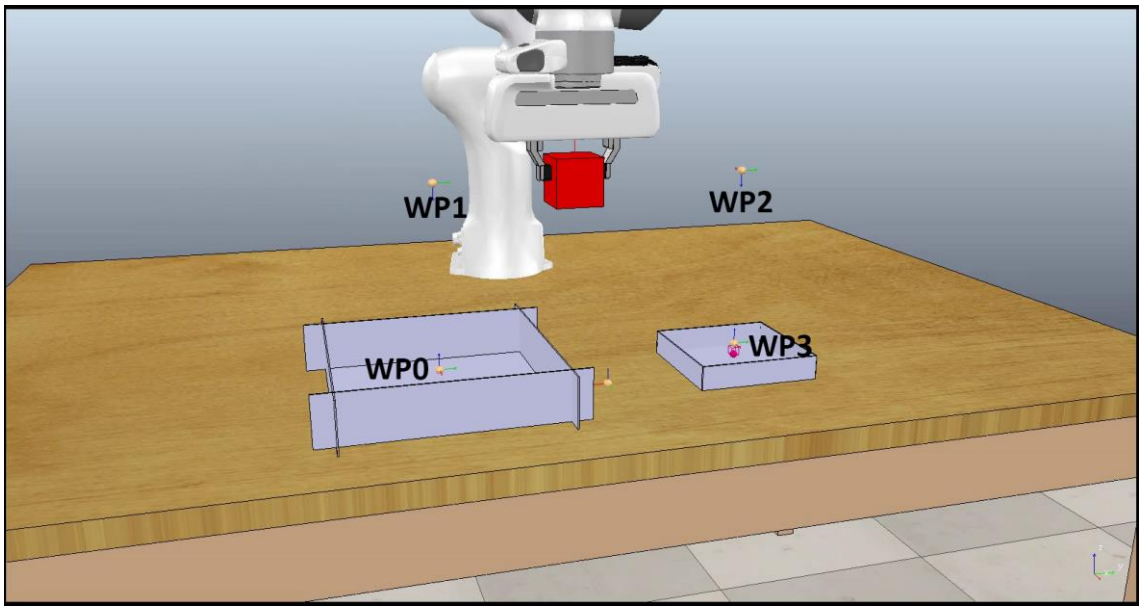



*Fig. B.3 – Ejemplo de política para la primera variación de la tarea de pick and place*

Tenemos dos waypoints fijos, con posición y orientación conocidos, $WP1$ y $WP2$. Estos waypoints serán de aproximación al recipiente grande y el recipiente pequeño. Suponemos que son conocidos ya que, aunque no conocemos la posición exacta de los

recipientes ni la del cubo, sí que podemos aproximar donde se encuentran. Los otros dos waypoints, $WP0$ y $WP3$, definen la posición en la que se coge el cubo en el recipiente grande y en la que se deposita en el recipiente pequeño, respectivamente. Para ambos waypoints, los parámetros que se tienen que aprender son solo los referentes a la posición, ya que vamos a suponer que la orientación es conocida.

Por tanto, la política será función de la posición de los dos waypoints:

$$\pi = f(P_{WP0}, P_{WP3})$$

Siendo $P_{WP0}$ y $P_{WP3}$ las posiciones de cada waypoint. La posición de $WP0$ se define respecto a la posición de $WP1$, waypoint de aproximación al recipiente grande, y la de $WP3$ se define con respecto a la posición de $WP2$, waypoint de aproximación al recipiente pequeño.

Esta parametrización es válida para las dos variaciones, lo único que cambia son los posibles valores que pueden tomar las coordenadas del waypoint $WP3$.

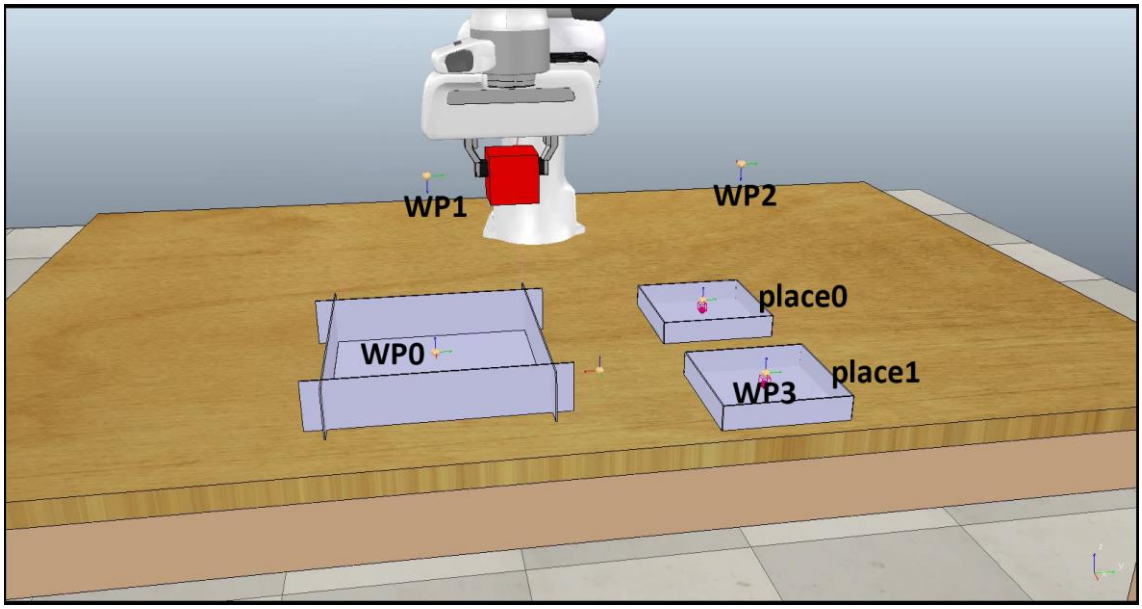



*Fig. B.4 – Ejemplo de política para la segunda variación de la tarea de pick and place*

### B.2.2. Recompensa

El objetivo de la tarea es coger el cubo del recipiente grande y llevarlo a un recipiente pequeño. Por tanto, la función de recompensa debe formularse en términos del éxito al coger el objeto y la posición en la que se deja el cubo. El éxito al coger el cubo se medirá como la distancia entre la pinza del robot y el cubo en su posición inicial. Para la posición final se calculará la distancia entre el cubo y la posición final deseada en el recipiente y también la distancia entre el elemento terminal del robot y la posición final deseada, ya que queremos dejar el cubo dentro del recipiente.

La distancia entre la pinza del robot y el cubo en su posición inicial será el parámetro $d_{pick}$, mientras que la distancia entre el cubo y la posición final deseada será $d_{place\_block}$ y la distancia entre el elemento terminal y la posición final deseada será $d_{place\_gripper}$.

$$R = -200 \cdot {d_{pick}}^2 - 200 \cdot {d_{place\_block}}^2 - 100 \cdot {d_{place\_gripper}}^2$$

Para la segunda variación serán necesarias $d_{pick}$, dos $d_{place\_block}$ y dos $d_{place\_gripper}$, ya que en este caso tenemos dos recipientes.

$$\begin{gathered} R = -200 \cdot {d_{pick}}^2 - 200 \cdot {d_{place0\_block}}^2 - 400 \cdot {d_{place1\_block}}^2 \\ -3500 \cdot d_{place0\_block} \cdot {d_{place1\_block}}^2 \\ -100 \cdot {d_{place0\_gripper}}^2 - 200 \cdot {d_{place1\_gripper}}^2 \\ -2500 \cdot d_{place0\_gripper} \cdot {d_{place1\_gripper}}^2 \end{gathered}$$

Esta función de recompensa supone que la posición final de un contenedor tiene el doble de recompensa que el otro. Con el producto de distancias nos aseguramos de que los mínimos de la función se encuentran en los dos recipientes.

## B.3. APRETAR UN BOTÓN

### B.3.1. Política

Para resolver la tarea de apretar un botón, la política se ha definido mediante dos waypoints para las dos variaciones.

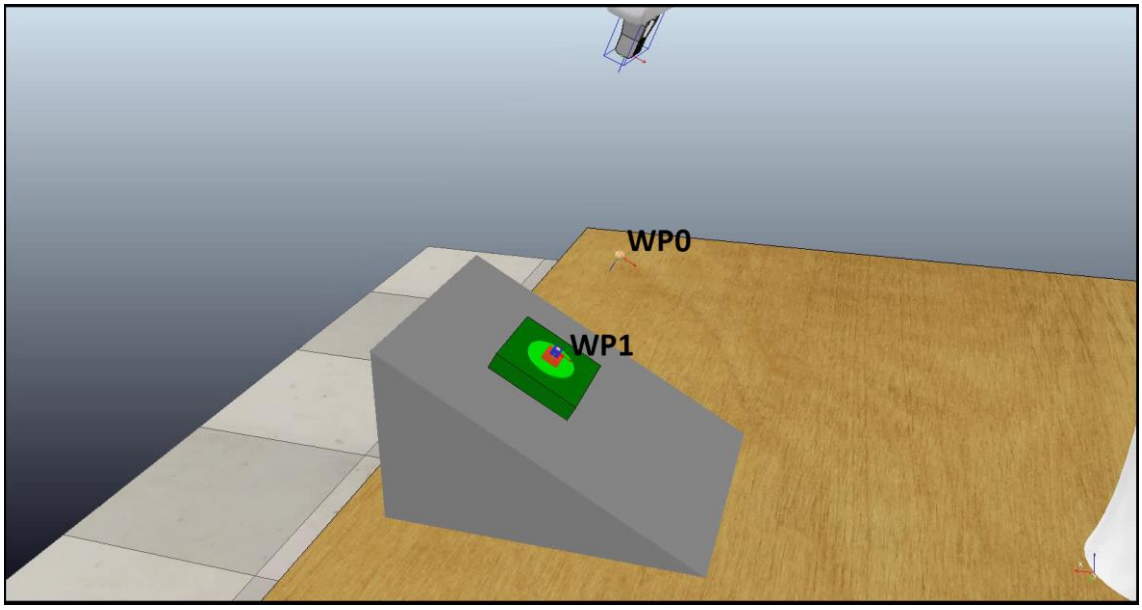



*Fig. B.5 – Ejemplo de política para la primera variación de la tarea de apretar un botón*

Uno de los waypoints, $WP0$, es de posición fija, conocida y es de aproximación, ya que igual que en el caso anterior, no conocemos exactamente la posición del botón, pero sabemos en qué zona de la mesa está situado. El otro waypoint, $WP0$, es la posición en la que se aprieta el botón y los parámetros a aprender serán tanto la posición como la orientación, ya que nos interesa apretar el botón de forma perpendicular. La orientación

es la misma para $WP1$ y para $WP0$, de forma que la trayectoria entre los dos waypoints solo sea de traslación. Por tanto, la política depende de la posición de $WP1$ y de la orientación:

$$\pi = f(P_{WP1}, \theta_{WP})$$

Siendo $P_{WP1}$ la posición de $WP1$ y $\theta_{WP}$ la orientación de los dos waypoints. La posición de $WP1$ se define con respecto a la posición del waypoint de aproximación, $WP0$. La orientación se define respecto a la orientación de la herramienta del robot, que tiene el eje Z perpendicular hacia abajo.

Esta parametrización es válida para las dos variaciones de la tarea, lo único que cambiará será los posibles valores que pueden tomar las coordenadas del waypoint $WP1$.

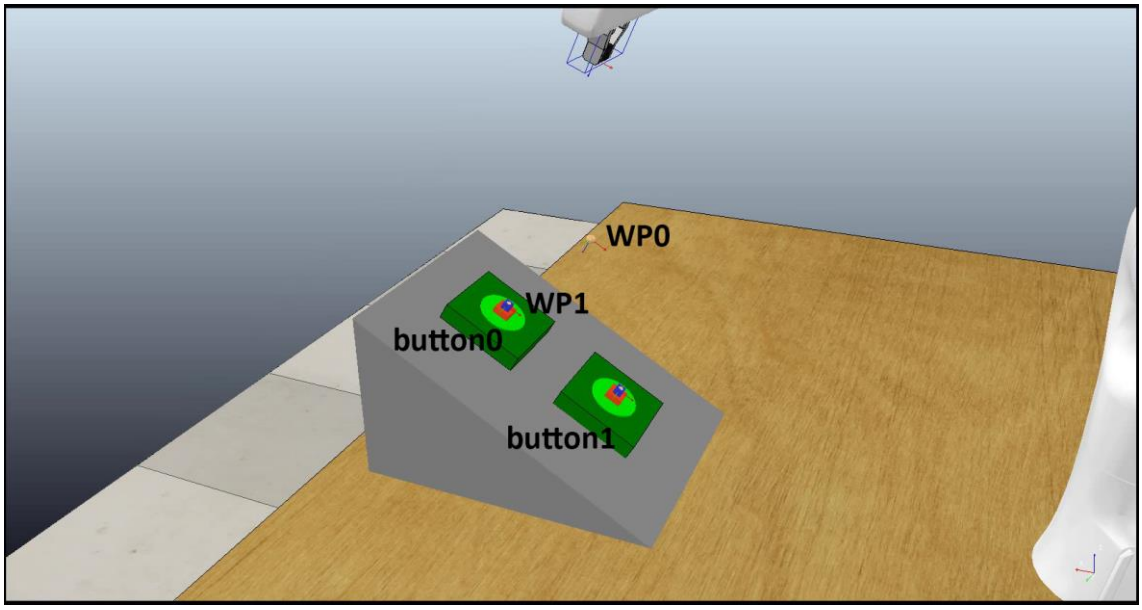



*Fig. B.6 – Ejemplo de política para la segunda variación de la tarea de apretar un botón*

### B.3.2. Recompensa

El objetivo de la tarea es apretar el botón de forma perpendicular, por lo que la función de recompensa se debe formular en términos de la distancia al botón y del error respecto a la orientación deseada. Para el error en la orientación se va a utilizar el error en cada ángulo de Euler.

Para la primera variación necesitamos la distancia al botón, $d_{button}$, y el error en la orientación.

$$R = -400 \cdot {d_{button}}^2 - 5 \cdot {e_{alpha}}^2 - 5 \cdot {e_{beta}}^2 - {e_{gamma}}^2$$

Para la segunda variación serán necesarias las distancias a ambos botones.

$$R = -400 \cdot {d_{button0}}^2 - 800 \cdot {d_{button1}}^2 - 3500 \cdot d_{button0} \cdot d_{button1} \\ -5 \cdot {e_{alpha}}^2 - 5 \cdot {e_{beta}}^2 - {e_{gamma}}^2$$

Al igual que el caso de la tarea de pick and place, la recompensa de uno de los botones es el doble que la recompensa del otro. Con el producto de distancias también nos aseguramos de que los mínimos de la función están en los botones.

## B.4. EMPUJAR UN OBJETO

### B.4.1. Política

Para resolver la tarea de empujar un objeto se ha definido la política mediante tres waypoints.

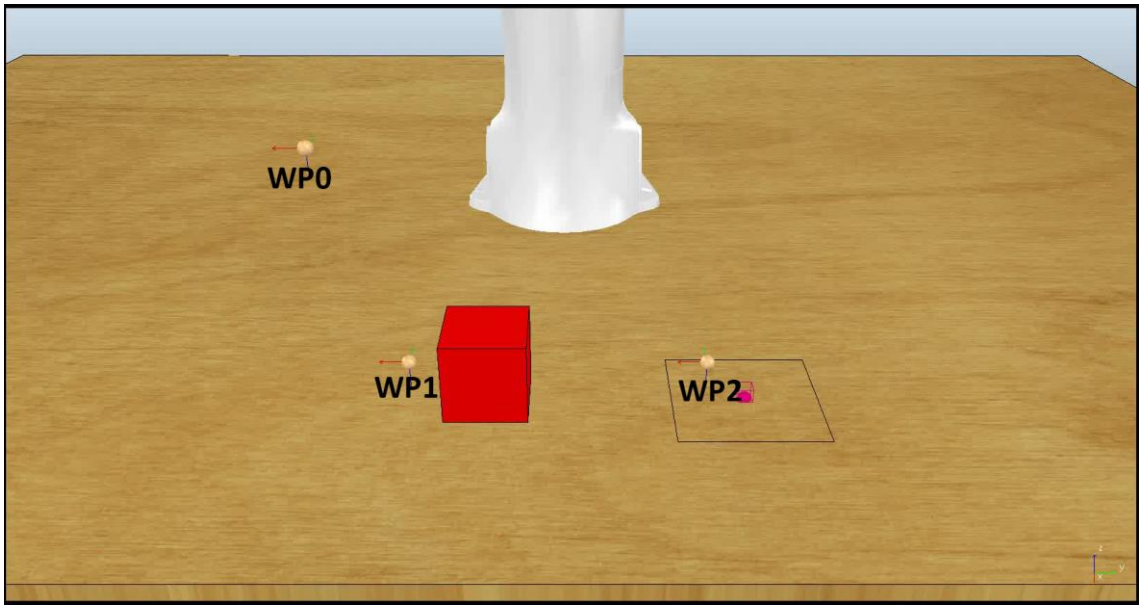



*Fig. B.7 – Ejemplo de política para la primera variación de empujar un objeto*

Uno de los waypoints, $WP0$, es fijo, con posición y orientación conocidos, y de aproximación, ya que aunque no sepamos la posición exacta del objeto que tenemos que empujar, conocemos la zona de la mesa en la que se encuentra, y sabemos que la orientación del cubo esta entre unos límites concretos, de forma que el robot siempre va a empujar el objeto de derecha a izquierda. Otro de los waypoints, $WP1$, es el de la posición desde que el robot empieza a empujar y los parámetros que se deben aprender son la posición y la orientación. El último waypoint, $WP2$, es el de la posición final de la herramienta del robot tras empujar el objeto. Para definir la posición de este último waypoint se han utilizado dos parámetros: la distancia que debe ser empujado el objeto y la dirección de empuje. Por tanto, la coordenada Z de WP2 será la misma que la de WP1. La orientación de empuje será también la orientación de WP1 y WP2

Por tanto, la política depende de la posición del primer waypoint y de la distancia y orientación de empuje:

$$\pi = f(P_{WP1}, d, \varphi)$$

Siendo $\mathrm{P}_{\mathrm{WP1}}$ la posición del waypoint $WP1$, $d$ la distancia de empuje y $\varphi$ la dirección de empuje. La posición de $WP1$ se define con respecto a la posición de $WP0$ y la posición de $WP2$ con respecto a la posición de $WP1$.

Esta parametrización es válida para las dos variaciones, lo que cambia son los posibles valores que pueden tomar la distancia y la orientación de empuje.

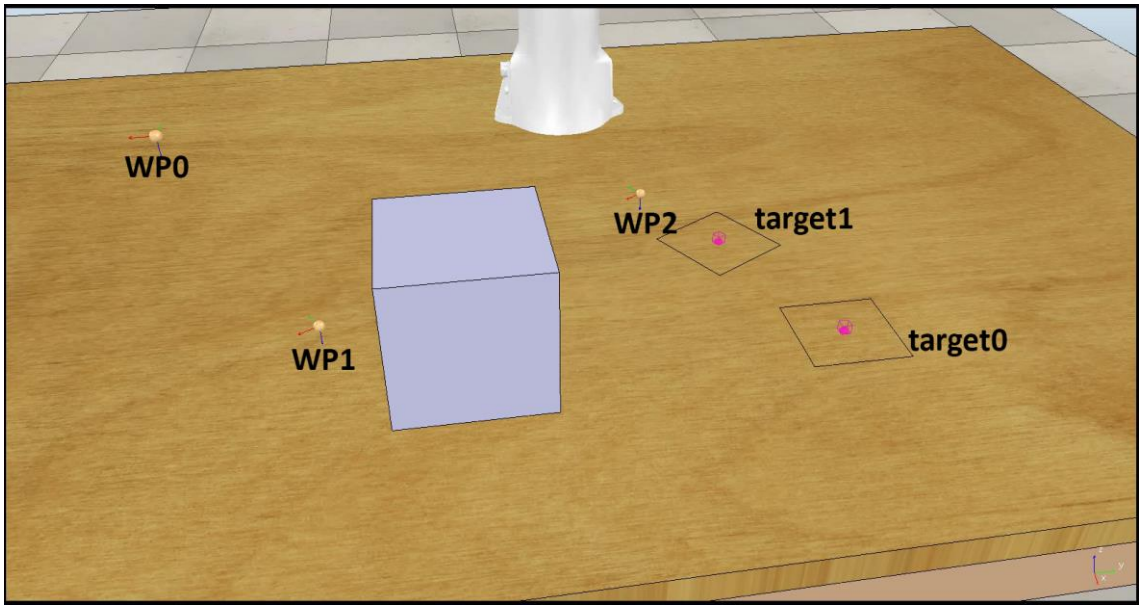



*Fig. B.8 – Ejemplo de política para la segunda variación de empujar un objeto*

### B.4.2. Recompensa

EL objetivo de la tarea es empujar correctamente el objeto desde una posición inicial hasta una posición final, minimizando la distancia de empuje y con una orientación correcta. Por tanto, para definir la función de recompensa se debe tener en cuenta la distancia entre la herramienta del robot y el cubo antes de empezar a empujarlo, que denominamos $d\_slide$, la distancia entre el cubo y la posición final deseada, que denominamos $d\_target$ y la distancia de empuje, $d$. Para la segunda variación, también se van a tener en cuenta los errores de la orientación en el eje perpendicular a la mesa de trabajo, que denominamos $e_{or\_target}$

Para la primera variación, la función de recompensa se define como:

$$R = -10 \cdot d^2 - 200 \cdot {d_{slide}}^2 - 200 \cdot {d_{target}}^2$$

La función de recompensa para la segunda variación es:

$$R = -10 \cdot d^2 - 200 \cdot {d_{slide}}^2 - 200 \cdot {d_{target0}}^2 - 400 \cdot {d_{target1}}^2$$
$$-3500 \cdot d_{target0} \cdot d_{target1} - 200 \cdot abs(e_{or\_target0}) \cdot d_{target1}$$
$$-500 \cdot abs(e_{or\_target1}) \cdot d_{target0}$$

Al igual que en los casos anteriores, una de las soluciones tiene el doble de recompensa que la otra. Con el producto de distancias nos aseguramos de que las soluciones se encuentran en las posiciones finales deseadas. Además al multiplicar el error en la orientación en la posición final 0 con la distancia a la posición final 1 y viceversa, nos aseguramos de que si la orientación del bloque coincide con la orientación de la posición final todo ese término es cero, y es distinto de cero en el caso contrario.

# ANEXO C. OPTIMIZACIÓN DE TODAS LAS TAREAS

En este anexo se presentan los resultados de las optimizaciones de todas las tareas y sus variaciones. Se incluyen los parámetros utilizados en BayesOpt y evolución diferencial.

## C.1. EVITACIÓN DE UN OBSTÁCULO

### C.1.1. Variación 1: un obstáculo

#### *C.1.1.1. Coordenadas cartesianas*

*Tabla C.1 – Parámetros de la optimización de BayesOpt para la primera variación de la tarea de evitación de un obstáculo en coordenadas cartesianas*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$X_{WP1}$, $Y_{WP1}$, $Z_{WP1}$, $X_{WP2}$, $Y_{WP2}$, $Z_{WP2}$] |
| **Límite inferior** | [-0.3, 0.1, -0.15, -0.1, 0.1, -0.1] |
| **Límite superior** | [0.3, 0.3, 0.25, 0.1, 0.5, 0.1] |
| **N.º iteraciones** | 300 |
| **N_iter_relearn** | 10 |
| **N_init_samples** | 30 |
| **N.º total evaluaciones función** | 330 |
| **N.º experimentos** | 5 |

*Tabla C.2 – Parámetros de la optimización de evolución diferencial para la primera variación de la tarea de evitación de un obstáculo en coordenadas cartesianas*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$X_{WP1}$, $Y_{WP1}$, $Z_{WP1}$, $X_{WP2}$, $Y_{WP2}$, $Z_{WP2}$] |
| **Límite inferior** | [-0.3, 0.1, -0.15, -0.1, 0.1, -0.1] |
| **Límite superior** | [0.3, 0.3, 0.25, 0.1, 0.5, 0.1] |
| **N.º iteraciones** | 10 |
| **Popsize** | 5 |
| **N.º total evaluaciones función** | 330 |
| **N.º experimentos** | 5 |

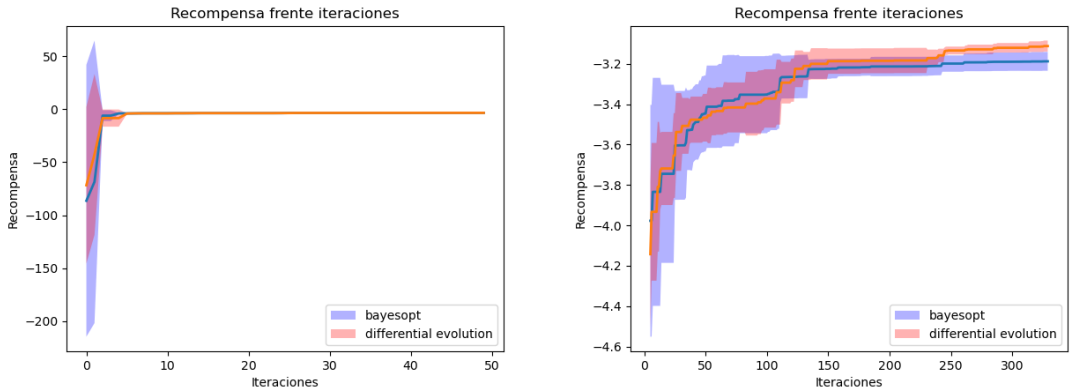



*Fig. C.1 – Optimización de la primera variación de evitación de un obstáculo con coordenadas cartesianas*

En la Fig. C.1 podemos observar los resultados obtenidos en la optimización. La línea se corresponde con la media obtenida del número de experimentos y la zona sombreada es el intervalo de confianza del 95%. En la gráfica de la izquierda se muestra la optimización en las primeras 50 iteraciones de la optimización mientras que la gráfica de la derecha muestra las iteraciones desde la iteración 5 hasta el final.

Como se puede observar, tanto para BayesOpt como para la evolución diferencial se encuentra una solución en pocas iteraciones. Viendo la tarea, se puede ver de forma intuitiva que no hay una única solución, ya que la esfera se puede evitar por diferentes zonas (por los lados, por encima…), con trayectorias de la misma longitud y en las que el robot guarda la misma distancia respecto al obstáculo. También se puede ver que con la evolución diferencial se obtienen trayectorias ligeramente mejores.

#### *C.1.1.2. Coordenadas polares*

*Tabla C.3 – Parámetros de la optimización de BayesOpt para la primera variación de la tarea de evitación de un obstáculo en coordenadas polares*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$r_{WP1}$, $\theta_{WP1}$, $\varphi_{WP1}$, $r_{WP2}$, $\theta_{WP2}$, $\varphi_{WP2}$] |
| **Límite inferior** | [0.1, π/4, π/8, 0.1, π/6, π/6] |
| **Límite superior** | [0.5, (π- π/4), (π-π/8), 0.5, (π- π/6),(π- π /6)] |
| **N.º iteraciones** | 200 |
| **N_iter_relearn** | 10 |
| **N_init_samples** | 30 |
| **N.º total evaluaciones función** | 230 |
| **N.º experimentos** | 5 |

*Tabla C.4 – Parámetros de la optimización de evolución diferencial para la primera variación de la tarea de evitación de un obstáculo en coordenadas polares*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$r_{WP1}$, $\theta_{WP1}$, $\varphi_{WP1}$, $r_{WP2}$, $\theta_{WP2}$, $\varphi_{WP2}$] |
| **Límite inferior** | [0.1, π/4, π/8, 0.1, π/6, π/6] |
| **Límite superior** | [0.5, (π- π/4), (π-π/8), 0.5, (π- π/6),(π- π /6)] |
| **N. º iteraciones** | 6 |
| **Popsize** | 5 |
| **N.º total evaluaciones función** | 210 |
| **N.º experimentos** | 5 |

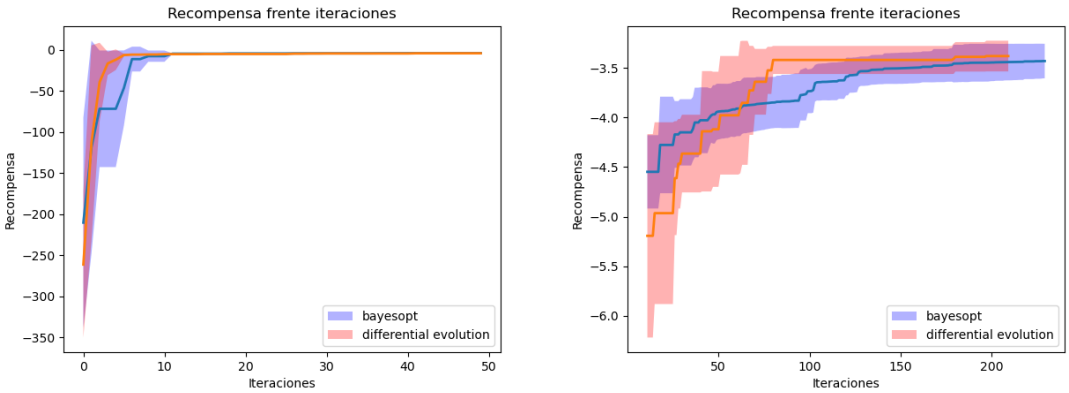



*Fig. C.2 – Optimización de la primera variación de evitación de un obstáculo con coordenadas polares*

En la Fig. C.2 podemos observar los resultados obtenidos en la optimización. La línea se corresponde con la media obtenida del número de experimentos y la zona sombreada es el intervalo de confianza del 95%. En la gráfica de la izquierda se muestras las primeras 50 iteraciones de la optimización mientras que en la gráfica de la derecha se muestra desde la iteración 10 hasta el final.

Al igual que en la optimización para coordenadas cartesianas, vemos que ambos optimizadores convergen a una solución en pocas iteraciones. Aquí BayesOpt encuentra las soluciones unas pocas iteraciones antes que la evolución diferencial, pero las soluciones que encuentra este último son ligeramente mejores.

### C.1.2. Variación 2: tres obstáculos

*Tabla C.5 – Parámetros de la optimización de BayesOpt para la segunda variación de la tarea de evitación de un obstáculo en coordenadas polares*

| | |
|---|---|
| **Dimensiones del problema** | 4 |
| **Parámetros** | [$r_{WP1}$, $\varphi_{WP1}$, $r_{WP2}$, $\varphi_{WP2}$] |
| **Límite inferior** | [0.1, 0.0, 0.1, - π/2] |
| **Límite superior** | [0.6, 70π/180, 0.5, π/2] |
| **N.º iteraciones** | 300 |
| **N_iter_relearn** | 10 |
| **N_init_samples** | 40 |
| **N.º total evaluaciones función** | 340 |
| **N.º experimentos** | 5 |

*Tabla C.6 – Parámetros de la optimización de evolución diferencial para la segunda variación de la tarea de evitación de un obstáculo en coordenadas polares*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$r_{WP1}$, $\varphi_{WP1}$, $r_{WP2}$, $\varphi_{WP2}$] |
| **Límite inferior** | [0.1, 0.0, 0.1, - π/2] |
| **Límite superior** | [0.6, 70π/180, 0.5, π/2] |
| **N.º iteraciones** | 3 |
| **Popsize** | 15 (valor por defecto) |
| **N.º total evaluaciones función** | 360 |
| **N.º experimentos** | 5 |

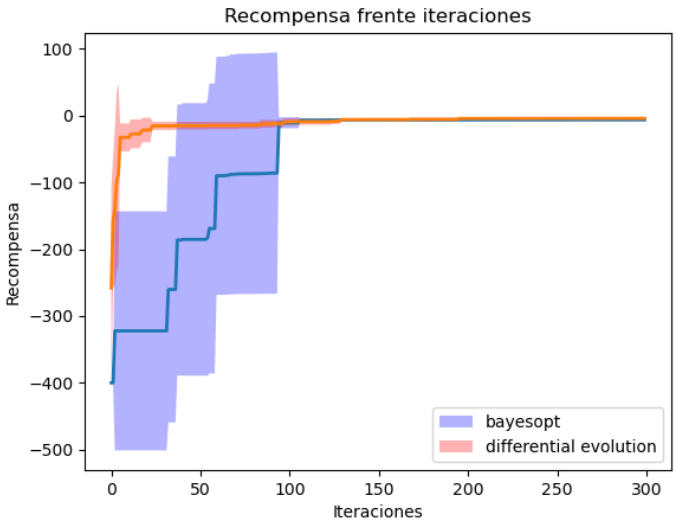



*Fig. C.3 – Optimización de la segunda variación de evitar un obstáculo*

En la Fig. C.3 podemos observar los resultados obtenidos en la optimización. La línea se corresponde con la media obtenida del número de experimentos y la zonza sombreada es el intervalo de confianza del 95%.

Como ya hemos comentado al definir esta variación de la tarea, puede haber dos posibilidades: que la trayectoria vaya por la abertura pequeña o por la abertura grande. En la gráfica podemos ver que tanto con BayesOpt como con evolución diferencial se encuentra una de las soluciones, aunque BayesOpt tarda más que la evolución diferencial en converger. Esto puede deberse a que en al menos uno de los experimentos se ha tardado mucho más en encontrar una política cercana a una solución.

Centrándonos en las soluciones de cada uno de los experimentos, las cinco soluciones obtenidas en BayesOpt son trayectorias que pasan por la abertura más grande, mientras que el caso de evolución diferencial se encuentran trayectorias que pasan por la abertura pequeña en tres experimentos, mientras que en el resto las trayectorias pasan por la abertura grande.

## C.2. PICK AND PLACE

### C.2.1. Variación 1: un contenedor de place

*Tabla C.7 – Parámetros de la optimización de BayesOpt para la primera variación de la tarea de pick and place*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$X_{WP0}$, $Y_{WP0}$, $Z_{WP0}$, $X_{WP3}$, $Y_{WP3}$, $Z_{WP3}$] |
| **Límite inferior** | [-0.1, -0.1, -0.27, -0.075, -0.075, -0.27] |
| **Límite superior** | [0.1, 0.1, -0.14, 0.075, 0.075, -0.14] |
| **N.º iteraciones** | 300 |
| **N_iter_relearn** | 10 |
| **N_init_samples** | 42 |
| **N.º total evaluaciones función** | 342 |
| **N.º experimentos** | 5 |

*Tabla C.8 – Parámetros de la optimización de evolución diferencial para la primera variación de la tarea de pick and place*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$X_{WP0}$, $Y_{WP0}$, $Z_{WP0}$, $X_{WP3}$, $Y_{WP3}$, $Z_{WP3}$] |
| **Límite inferior** | [-0.1, -0.1, -0.27, -0.075, -0.075, -0.27] |
| **Límite superior** | [0.1, 0.1, -0.14, 0.075, 0.075, -0.14] |
| **N.º iteraciones** | 7 |
| **Popsize** | 7 |
| **N.º total evaluaciones función** | 336 |
| **N.º experimentos** | 5 |

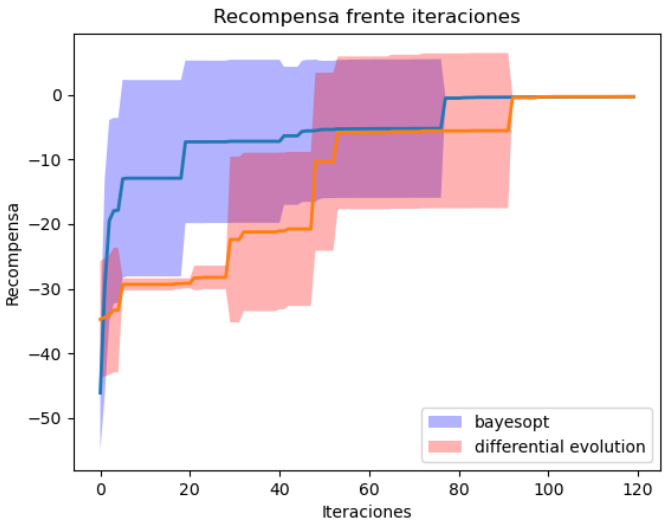



*Fig. C.4 – Optimización de la primera variación de pick and place*

En la Fig. C.4 podemos observar los resultados obtenidos en la optimización de la primera variación. La línea se corresponde con la media obtenida del número de experimentos y la zona sombreada es el intervalo de confianza del 95%. Desde la iteración 120 hasta el final de la optimización las dos líneas siguen rectas.

Si observamos la función de recompensa de esta tarea, definida en el apartado B.2.2, y la forma en que se parametriza la política, en el apartado B.2.1, podemos ver que esta tarea tiene una única solución, en la que los posición de los waypoints es tal que la distancia de pick y la distancia de place es 0, de forma que la recompensa es 0. En la gráfica podemos ver que aproximadamente a partir de la iteración 80 BayesOpt ha encontrado esta solución en todos los experimentos, ya que es valor de la media de la recompensa es 0 y no hay intervalo de confianza, mientras que a la evolución diferencial le cuesta unas iteraciones más.

### C.2.2. Variación 2: dos contenedores de place

*Tabla C.9 – Parámetros de la optimización de BayesOpt para la segunda variación de la tarea de pick and place*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$X_{WP0}$, $Y_{WP0}$, $Z_{WP0}$, $X_{WP3}$, $Y_{WP3}$, $Z_{WP3}$] |
| **Límite inferior** | [-0.1, -0.1, -0.27, -0.2, -0.075, -0.27] |
| **Límite superior** | [0.1, 0.1, -0.14, 0.2, 0.075, -0.14] |
| **N.º iteraciones** | 300 |
| **N_iter_relearn** | 10 |
| **N_init_samples** | 60 |
| **N.º total evaluaciones función** | 360 |
| **N.º experimentos** | 5 |

*Tabla C.10 – Parámetros de la optimización de evolución diferencial para la segunda variación de la tarea de pick and place*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Límite inferior** | [-0.1, -0.1, -0.27, -0.2, -0.075, -0.27] |
| **Límite superior** | [0.1, 0.1, -0.14, 0.2, 0.075, -0.14] |
| **N.º iteraciones** | 5 |
| **Popsize** | 10 |
| **N.º total evaluaciones función** | 360 |
| **N.º experimentos** | 5 |

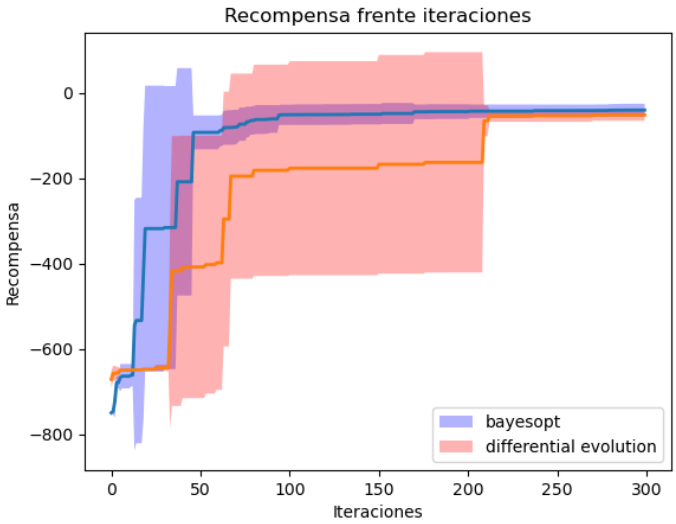



*Fig. C.5 – Optimización de la segunda variación de pick and place*

En la Fig. C.5 podemos observar los resultados obtenidos en la optimización de esta segunda variación. La línea se corresponde con la media obtenida del número de experimentos y la zonza sombreada es el intervalo de confianza del 95%. Desde la iteración 300 al final de la optimización ambas líneas siguen rectas.

Al igual que ocurre en la primera variación, de la forma en la que se define esta tarea, existen dos soluciones, según se lleve el cubo a un contenedor o al otro. Podemos ver que tanto BayesOpt como con la evolución diferencial encontramos las soluciones. También podemos ver que BayesOpt converge antes hacia las soluciones que la evolución diferencial. Analizando por separado las soluciones de cada experimento, vemos que con BayesOpt encontramos el contenedor más alejado del robot en cuatro ocasiones y el más cercano en una, mientras que la evolución diferencial encuentra el contenedor más alejado en todos los experimentos.

## C.3. APRETAR UN BOTÓN

### C.3.1. Variación 1: un botón

*Tabla C.11 – Parámetros de la optimización de BayesOpt para la primera variación de la tarea de apretar un botón*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$X_{WP1}$, $Y_{WP1}$, $Z_{WP1}$, $\alpha_{WP}$, $\beta_{WP}$, $\gamma_{WP}$] |
| **Límite inferior** | [0.0, -0.05, -0.15, -π / 4, 0.0, - π / 2] |
| **Límite superior** | [0.1, 0.05, 0.0, π / 4, π / 3, π / 2] |
| **N.º iteraciones** | 300 |
| **N_iter_relearn** | 10 |
| **N_init_samples** | 24 |
| **N.º total evaluaciones función** | 324 |
| **N.º experimentos** | 5 |

*Tabla C.12 – Parámetros de la optimización de evolución diferencial para la primera variación de la tarea de apretar un botón*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$X_{WP1}$, $Y_{WP1}$, $Z_{WP1}$, $\alpha_{WP}$, $\beta_{WP}$, $\gamma_{WP}$] |
| **Límite inferior** | [0.0, -0.05, -0.15, -π / 4, 0.0, - π / 2] |
| **Límite superior** | [0.1, 0.05, 0.0, π / 4, π / 3, π / 2] |
| **N.º iteraciones** | 5 |
| **Popsize** | 10 |
| **N.º total evaluaciones función** | 360 |
| **N.º experimentos** | 5 |

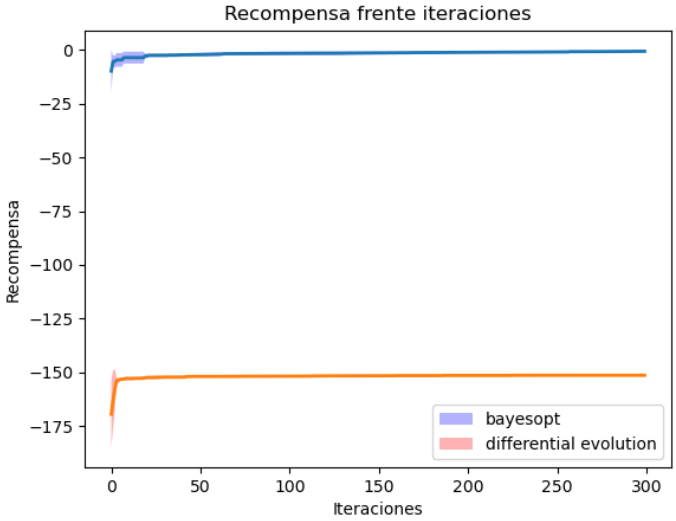



*Fig. C.6 – Optimización de la primera variación de apretar un botón*

En la Fig. C.6 podemos observar los resultados obtenidos en la optimización de la primera variación. La línea se corresponde con la media obtenida del número de experimentos y la zonza sombreada es el intervalo de confianza del 95%.

Al igual que ocurre en la tarea de pick and place para esta tarea hay una única solución, cuando el robot acciona el robot de forma perpendicular. Si observamos la función de recompensa definida en el apartado B.3.2, para ese caso la recompensa es cero, ya que todos los parámetros de la función de recompensa son también cero. Si analizamos la gráfica, se ve claramente que BayesOpt se va acercando lentamente a esa solución conforme avanzan las iteraciones, pero la evolución diferencial se queda muy lejos de encontrar la solución.

Si analizamos por separado la solución de cada experimento, podemos ver que en la evolución diferencial si se encuentra la posición del botón pero no se acciona de forma perpendicular a este, mientras que BayesOpt si es capaz de encontrar la orientación correcta.

### C.3.2. Variación 2: dos botones

*Tabla C.13 – Parámetros de la optimización de BayesOpt para la primera segunda de la tarea de apretar un botón*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$X_{WP1}$, $Y_{WP1}$, $Z_{WP1}$, $\alpha_{WP}$, $\beta_{WP}$, $\gamma_{WP}$] |
| **Límite inferior** | [-0.1, -0.05, -0.2, - π / 4, 0.0, - π / 2] |
| **Límite superior** | [0.15, 0.05, 0.0, π / 4, π / 3, π / 2] |
| **N.º iteraciones** | 300 |
| **N_iter_relearn** | 10 |
| **N_init_samples** | 24 |
| **N.º total evaluaciones función** | 324 |
| **N.º experimentos** | 5 |

*Tabla C.14 – Parámetros de la optimización de evolución diferencial para la segunda variación de la tarea de apretar un botón*

| | |
|---|---|
| **Dimensiones del problema** | 6 |
| **Parámetros** | [$X_{WP1}$, $Y_{WP1}$, $Z_{WP1}$, $\alpha_{WP}$, $\beta_{WP}$, $\gamma_{WP}$] |
| **Límite inferior** | [0.0, -0.05, -0.15, -π / 4, 0.0, - π / 2] |
| **Límite superior** | [0.1, 0.05, 0.0, π / 4, π / 3, π / 2] |
| **N.º iteraciones** | 5 |
| **Popsize** | 10 |
| **N.º total evaluaciones función** | 360 |
| **N.º experimentos** | 5 |

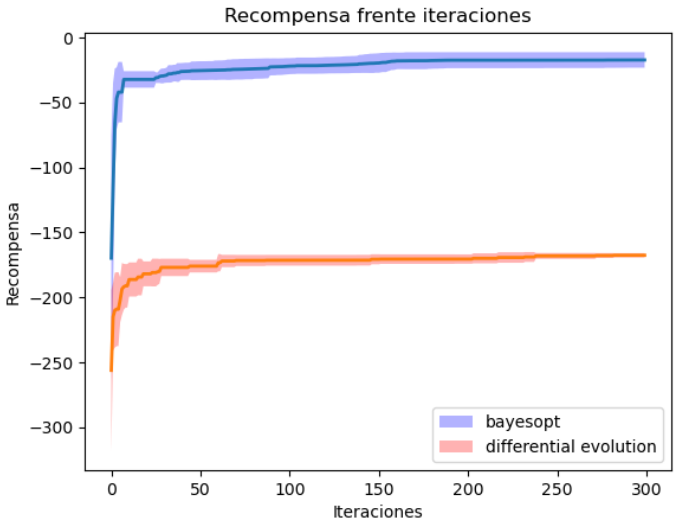



*Fig. C.7 – Optimización de la segunda variación de apretar un botón*

En la Fig. C.7 podemos observar los resultados obtenidos en la optimización de la segunda variación. La línea se corresponde con la media obtenida del número de experimentos y la zonza sombreada es el intervalo de confianza del 95%.

Al igual que ocurre en la tarea de pick and place para esta segunda variación hay dos soluciones, cuando el robot acciona uno de los dos botones. Si analizamos la gráfica, se ve claramente que BayesOpt se va acercando a la solución conforme avanzan las iteraciones, pero la evolución diferencial se queda muy lejos de encontrar la solución. El intervalo de confianza que se observa en BayesOpt se debe a que en unos experimentos se encuentra el botón de arriba y en otros se encuentra el botón de abajo.

Si analizamos la solución de cada experimento por separado ocurre lo mismo que para la primera variación, la evolución diferencial encuentra el botón pero la orientación no es la correcta.

## C.4. EMPUJAR UN OBJETO

### C.4.1. Variación 1: una posición final

*Tabla C.15 – Parámetros de la optimización de BayesOpt para la primera variación de la tarea de empujar un objeto*

| | |
|---|---|
| **Dimensiones del problema** | 5 |
| **Parámetros** | [$X_{WP1}$, $Y_{WP1}$, $Z_{WP1}$, d, φ] |
| **Límite inferior** | [-0.2, 0.0, -0.25, 0.05, - π / 4] |
| **Límite superior** | [0.2, 0.1, 0.0, 0.4, π / 4] |
| **N.º iteraciones** | 250 |
| **N_iter_relearn** | 5 |
| **N_init_samples** | 10 |
| **N.º total evaluaciones función** | 260 |
| **N.º experimentos** | 5 |

*Tabla C.16 – Parámetros de la optimización de evolución diferencial para la primera variación de la tarea de empujar un objeto*

| | |
|---|---|
| **Dimensiones del problema** | 5 |
| **Parámetros** | [$X_{WP1}$, $Y_{WP1}$, $Z_{WP1}$, d, φ] |
| **Límite inferior** | [-0.2, 0.0, -0.25, 0.05, - π / 4] |
| **Límite superior** | [0.2, 0.1, 0.0, 0.4, π / 4] |
| **N.º iteraciones** | 50 |
| **Popsize** | 1 |
| **N.º total evaluaciones función** | 300 |
| **N.º experimentos** | 5 |

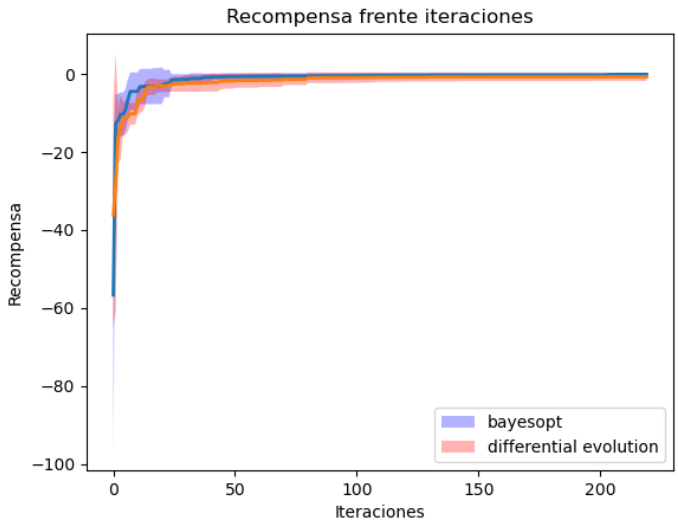



*Fig. C.8 – Optimización para la primera variación de empujar un objeto*

En la Fig. C.8 podemos observar los resultados obtenidos en la optimización de la primera variación. La línea se corresponde con la media obtenida del número de experimentos y la zonza sombreada es el intervalo de confianza del 95%.

Para esta variación de la tarea solo existe una solución, cuando el objeto es empujado a la posición final deseada. Viendo la función de recompensa definida en el apartado B.4.2, en esa situación el valor es 0. Analizando la gráfica podemos ver que tanto con BayesOpt como con la evolución diferencial se encuentra la solución, aunque con BayesOpt converge unas iteraciones antes.

### C.4.2. Variación 2: dos posiciones finales

*Tabla C.17 – Parámetros de la optimización de BayesOpt para la segunda variación de la tarea de empujar un objeto*

| | |
|---|---|
| **Dimensiones del problema** | 5 |
| **Parámetros** | [$X_{WP1}$, $Y_{WP1}$, $Z_{WP1}$, d, φ] |
| **Límite inferior** | [-0.15, 0.0, -0.25, 0.2, - π / 4] |
| **Límite superior** | [0.15, 0.2, 0, 0.5, π / 4] |
| **N.º iteraciones** | 300 |
| **N_iter_relearn** | 10 |
| **N_init_samples** | 50 |
| **l_type** | L_MCMC |
| **N.º total evaluaciones función** | 350 |
| **N.º experimentos** | 5 |

*Tabla C.18 – Parámetros de la optimización de evolución diferencial para la segunda variación de la tarea de empujar un objeto*

| | |
|---|---|
| **Dimensiones del problema** | 5 |
| **Parámetros** | [$X_{WP1}$, $Y_{WP1}$, $Z_{WP1}$, d, φ] |
| **Límite inferior** | [-0.15, 0.0, -0.25, 0.2, - π / 4] |
| **Límite superior** | [0.15, 0.2, 0, 0.5, π / 4] |
| **N.º iteraciones** | 6 |
| **Popsize** | 10 |
| **N.º total evaluaciones función** | 350 |
| **N.º experimentos** | 5 |

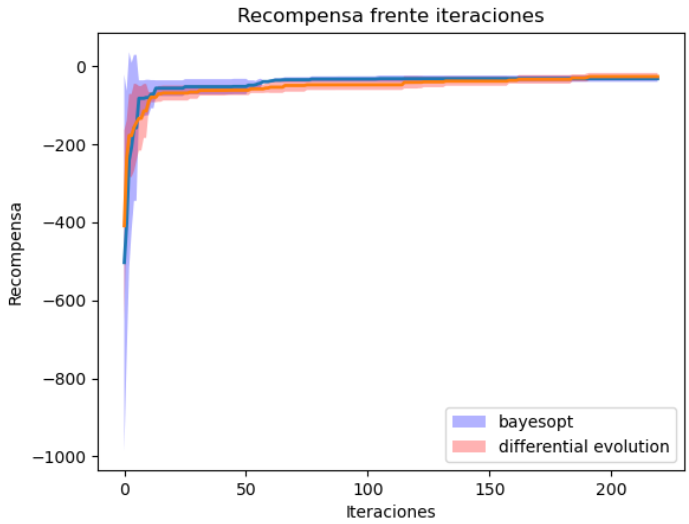



*Fig. C.9 – Optimización para la segunda variación de empujar un objeto*

En la Fig. C.9 podemos observar los resultados obtenidos en la optimización de la segunda variación. La línea se corresponde con la media obtenida del número de experimentos y la zonza sombreada es el intervalo de confianza del 95%. Desde la iteración 210 hasta el final de la optimización ambas líneas se mantienen igual.

En la segunda variación tenemos dos posibles soluciones ya que tenemos dos posibles posiciones finales para el bloque. Analizando la gráfica, podemos ver que tanto BayesOpt como la evolución diferencial encuentran alguna de las soluciones, y BayesOpt converge antes aunque la evolución diferencial termina encontrando una solución mejor. Esto puede deberse a que BayesOpt encuentra primero una solución, pero la solución que encuentra suele ser la que da menos recompensa y aunque la evolución diferencial tarda más en encontrar la solución, termina encontrando la solución de mayor recompensa en más ocasiones que BayesOpt. Si analizamos la solución de cada experimento por separado, vemos que BayesOpt solo ha encontrado la solución de mayor recompensa en una ocasión, mientras que con la evolución diferencial la encuentra en tres ocasiones.

En general, podemos ver que algunas de las tareas planteadas son demasiado fáciles para cualquier algoritmo de aprendizaje por refuerzo, ya que la función de recompensa definida solo tiene un mínimo, como por ejemplo las primeras variaciones de las tareas de pick and place y de apretar un botón, en las que hacen falta pocas iteraciones hasta encontrar una solución. De los optimizadores podemos decir que, en general, BayesOpt funciona mejor que la evolución diferencial, sobre todo cuando hay una componente relacionada con la orientación, como se ve en el caso de la tarea de apretar un botón.

# ANEXO D. MANUAL DE USUARIO DE LA LIBRERÍA

## D.1. REQUISITOS DE SOFTWARE

Se recomienda el uso de un entorno virtual de Python para instalar las dependencias de la librería, por ejemplo utilizando conda con Anaconda:

https://www.anaconda.com/products/individual

El simulador CoppeliaSim se puede descargar desde su página web. La versión de Ubuntu se descarga como una carpeta, ejecutando ./coppeliaSim.sh estando en el directorio ejecuta el simulador.

https://www.coppeliarobotics.com/downloads

PyRep se puede descargar desde el repositorio de GitHub. En el propio repositorio hay una guía de instalación.

https://github.com/stepjam/PyRep

BayesOpt también se puede descargar desde el repositorio de GitHub. La documentación de BayesOpt incluye una guía de instalación.

https://github.com/rmcantin/bayesopt

http://rmcantin.github.io/bayesopt/html/

Para instalar la evolución diferencial basta con instalar la biblioteca de SciPy en Python, por ejemplo utilizando conda.

Así mismo también es necesario instalar las bibliotecas de matplotlib y pickle para hacer gráficas y guardar y cargar datos de las optimizaciones. Así mismo también es necesario tener instalado la biblioteca de NumPy.

La biblioteca de tareas robóticas puede descargarse desde el repositorio de GitHub.

https://github.com/deedaniel/TFM

Una vez instalado y descargado todo lo anterior, la biblioteca de tareas ya puede utilizarse.

## D.2. ESTRUCTURA DE LA BIBLIOTECA

Una vez entramos en la carpeta principal de la biblioteca, vemos cuatro carpeta y cuatro archivos de Python. Dentro de las carpetas se encuentran los archivos de cada tarea: evitación de un obstáculo, pick and place, apretar un botón y empujar un objeto. Los cuatro archivos permiten optimizar las tareas con BayesOpt y con evolución diferencial, hacer gráficas con los resultados de las optimizaciones y el otro contiene los límites de los parámetros de entrada de todas las tareas.

Los archivos de optimización están programados de tal forma que cambiando unas pocas líneas de código se cambiar la tarea que se quiere optimizar, su variación y los parámetros de optimización. Los programas de optimización tienen como salida dos archivos, uno con los parámetros y la recompensa de cada iteración y el otro con los parámetros obtenidos como solución. Estos archivos se guardan en la carpeta de la tarea respectiva.

El archivo de gráficas coge el archivo con los parámetros y las recompensas de cada optimizador y pinta en cada iteración la mejor recompensa obtenida hasta ese momento, de forma que hay dos líneas, una de BayesOpt y otra de evolución diferencial. Con unas pocas líneas de código se puede cambiar la tarea y los datos que se grafican.

Dentro de cada carpeta de cada tarea hay varios archivos. Por un lado, está el archivo de cada tarea, que tiene como entrada un parámetro de modo headless y la variación. El parámetro headless permite elegir si se quiere ejecutar la interfaz del simulador. La variación permite elegir que variación de la tarea se quiere ejecutar. Dentro de este archivo está la función de la tarea, que tiene como variables de entrada los parámetros de la política y como salida la recompensa. Todas las tareas tienen un sol archivo salvo la tarea de evitación de un obstáculo, que tiene archivos separados para cada variación.

También hay un archivo de prueba, que permite ejecutar la función de la tarea. Este archivo permite cargar las soluciones obtenidas en las optimizaciones y también programar a mano la política, definiendo los parámetros de entrada.

Además, están los archivos y las gráficas obtenidas en la optimización y la escena del simulador.

## D.3. USO DE LA BIBLIOTECA

### D.3.1. OPTIMIZACIÓN

Para poder optimizar una tarea es necesario abrir uno de los archivos de optimización. Para poder optimizar una tarea, es necesario importar la función correspondiente. Después se asignan las variables correspondientes al directorio, nombre de la tarea y variación para que se guarden los datos obtenidos de manera correcta y poder cargar los límites de los parámetros de entrada.

Posteriormente se fijan los parámetros correspondientes al optimizador que corresponda, ya sea BayesOpt o evolución diferencial.

Después se inicializa la tarea que se quiere optimizar y se define la función de entrada al optimizador.

Una vez se ha configurado todo lo anterior, ya que puede ejecutar la optimización. Cuando acabe, los archivos de salida se habrán guardado en la carpeta correspondiente.

### D.3.2. EJECUCIÓN DE UNA TAREA

Para poder ejecutar una tarea es necesario abrir el archivo de prueba correspondiente. Después hay que elegir la variación de la tarea que se quiere ejecutar con su variable.

Después se definen los parámetros de la política de la tarea que se va a ejecutar. Estos parámetros pueden cargarse de las soluciones de las optimizaciones, cargándolas del archivo correspondiente o entrando a mano los valores de cada dimensión, siempre en una variable.

Después se inicializa la tarea y se introducen la variable con los parámetros de entrada en la función.

Con todo lo anterior la tarea ya estaría lista para ejecutarse. Al ejecutar el programa se ejecuta la tarea, con interfaz gráfica o no según el valor que se le dé a la variable de headless y se obtiene la recompensa de la política de entrada.

### D.3.3. HACER UNA GRÁFICA

Para hacer una gráfica solo es necesario cambiar las variables correspondientes al directorio, nombre de la tarea y variación de la tarea y cargar los archivos de optimización correspondiente. Después se ejecuta el archivo y la gráfica se guarda en la carpeta que corresponde.