

Trabajo Fin de Máster

APRENDIZAJE REFORZADO PARA AGARRE Y MANIPULACIÓN DE ROBOTS

REINFORCEMENT LEARNING FOR GRASPING AND HADLING IN ROBOTS

Autor/es

Daniel Cubel Gálvez

Director/es

Rubén Martínez Cantín

Máster en Ingeniería Industrial

Escuela de Ingeniería y Arquitectura

2020

ÍNDICE DE CONTENIDOS

[1. OBJETO 3](#_Toc45993206)

[2. ALCANCE 4](#_Toc45993207)

[3. MARCO TEÓRICO 5](#_Toc45993208)

[3.1. APRENDIZAJE POR REFUERZO 5](#_Toc45993209)

[3.1.1. INTRODUCCIÓN 5](#_Toc45993210)

[3.1.2. PROCESOS DE DECISIÓN DE MARKOV 5](#_Toc45993211)

[3.1.3. BÚSQUEDA DE POLÍTICA 5](#_Toc45993212)

[3.2. OPTIMIZACIÓN BAYESIANA 5](#_Toc45993213)

[4. HERRAMIENTAS TECNOLÓGICAS 6](#_Toc45993214)

[4.1. COPPELIASIM 6](#_Toc45993215)

[4.1.1. INTRODUCCIÓN 6](#_Toc45993216)

[4.1.2. FUNCIONALIDAD DEL SIMULADOR 6](#_Toc45993217)

[4.1.3. CONTROL DE LA SIMULACIÓN 8](#_Toc45993218)

[4.2. PYREP 9](#_Toc45993219)

[4.3. RLBENCH 9](#_Toc45993220)

[4.4. PYTHON Y ANACONDA 9](#_Toc45993221)

[4.5. BAYESOPT 9](#_Toc45993222)

[4.6. GIT Y GITHUB 10](#_Toc45993223)

[5. DEFINICIÓN DEL PROBLEMA 10](#_Toc45993224)

[5.1. ENTORNO ROBÓTICO 10](#_Toc45993225)

[5.2. TAREAS 10](#_Toc45993226)

[6. SOLUCIÓN ADOPTADA 10](#_Toc45993227)

[6.1. DEFINICIÓN DE LAS TAREAS 10](#_Toc45993228)

[7. SIMULACIÓN Y RESULTADOS 11](#_Toc45993229)

[8. CONCLUSIONES 11](#_Toc45993230)

# OBJETO

El objeto de este Trabajo Fin de Máster (TFM) es 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. 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 librería de aprendizaje por refuerzo basada en optimización bayesiana.

La justificación de este trabajo se centra sobre todo en que el problemas 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, 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.

El aprendizaje por refuerzo supone una revolución en muchos campos de la investigación y de la tecnología, como por ejemplo en este trabajo tareas robóticas de manipulación, pero también puede ser aplicado a una amplia variedad de problemas, como pueden ser 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 nuestro caso, el aprendizaje por refuerzo nos va a servir para resolver una serie de tareas en las que el robot aprenderá una política óptima, en lugar de resolver las tareas de forma tradicional.

# ALCANCE

Las actividades desarrolladas para la realización de este trabajo fin de máster han sido:

1. Estudios previos.
   1. Revisión bibliográfica y estado del arte del aprendizaje por refuerzo.
   2. Revisión bibliográfica y estado del arte de la optimización bayesiana.
2. Familiarización con las herramientas tecnológicas.
   1. Lenguaje de programación Python.
   2. Sistema operativo Ubuntu (Linux).
   3. Plataforma de simulación de robots CoppeliaSim.
   4. PyRep, que funciona por encima de CoppeliaSim, para controlar el simulador con python.
   5. Benchmark y entorno de aprendizaje RLBench, que funciona por encima de PyRep y CoppeliaSim.
   6. Librería de optimización bayesiana BayesOpt.
   7. Control de versiones con Git y Github
3. Diseño de tareas fácilmente reproducibles para los algoritmos de aprendizaje por refuerzo.
4. Diseño de políticas para resolver las tareas planteadas.
5. Escribir el código de interfaz entre el simulador y la librería de aprendizaje.
6. Conclusiones.

# MARCO TEÓRICO

## APRENDIZAJE POR REFUERZO

### INTRODUCCIÓN

### PROCESOS DE DECISIÓN DE MARKOV

### BÚSQUEDA DE POLÍTICA

## OPTIMIZACIÓN BAYESIANA

### 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 sería equivalente a la minimización de una función , y:

En la búsqueda de política, evaluar la función de recompensa con el método de Monte Carlo es muy costoso. 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 esta 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.

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 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, la distribución a priori se combina con la verosimilitud, obteniendo la probabilidad a posteriori.

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.

# HERRAMIENTAS TECNOLÓGICAS

## COPPELIASIM

### INTRODUCCIÓN

CoppeliaSim 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 que forma se pueden controlar las simulaciones.

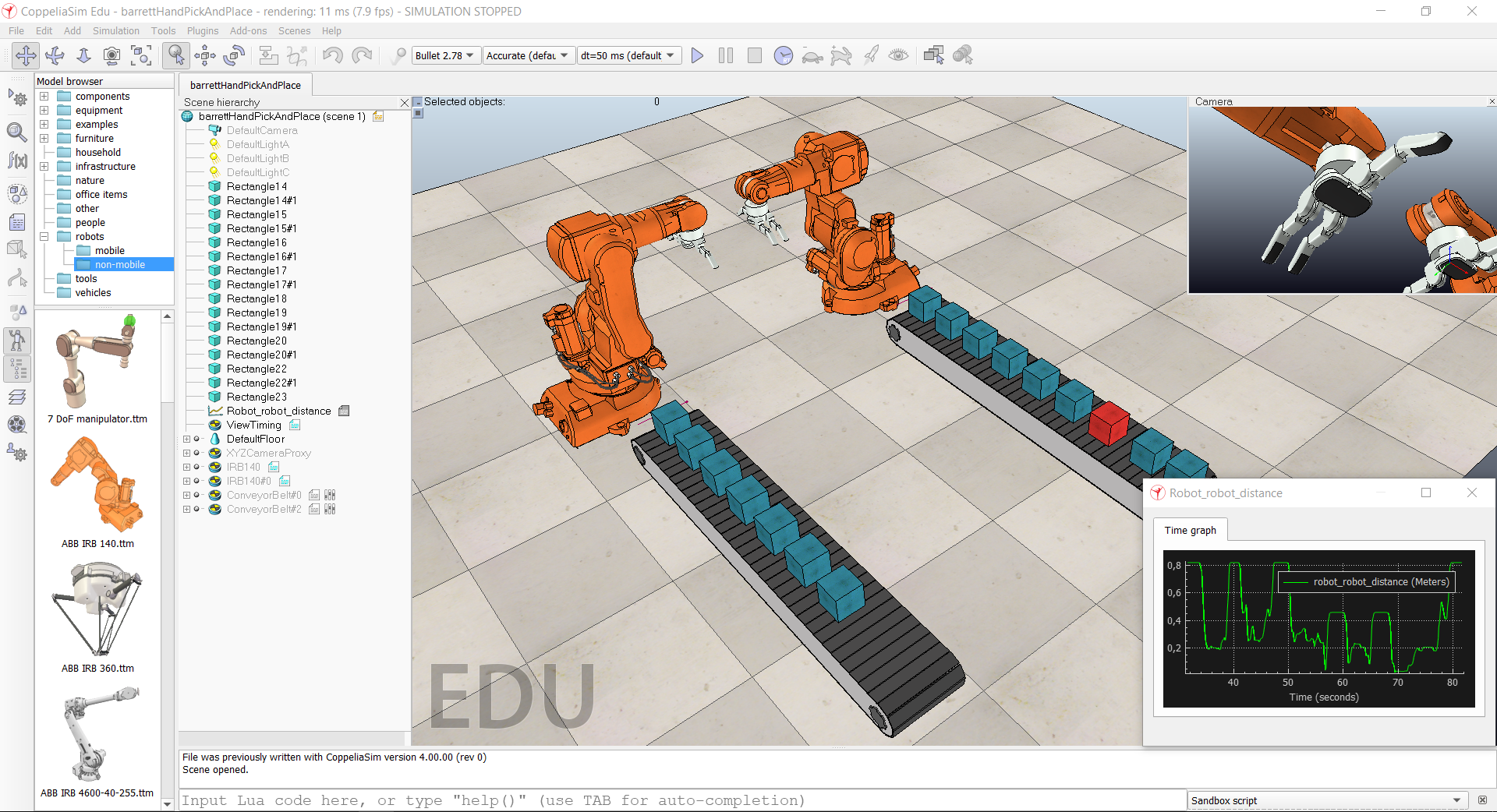


Fig. . – Interfaz de CoppeliaSim con una escena de ejemplo cargada.

### 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, los módulos de cálculo no se van a explicar.

#### 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 modulo de dinámica o el modulo de cálculo de distancia malla-malla).



Fig. . – Sensor de proximidad

* 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. (como diferencio entre paths y trajectories en español?)
* 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.

### 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:
* Nodos de ROS: CoppeliaSim implementa un nodo de ROS con un plugín 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.

## PYREP

## RLBENCH

## PYTHON Y ANACONDA

Todo el código del proyecto esta escrito en el lenguaje de programación Python, tanto los programas correspondientes a cada una de las tareas diseñadas como los programas de optimización.

Python es un lenguaje interpretado, fuertemente tipado y multiparadigma, ya que permite tanto la programación orientada a objetos como la programación imperativa y la programación funcional.

Una de las características más importante de Python es la modularidad. Python permite importar módulos, que son ficheros que contienen definiciones y funciones. Al importar un módulo, sus definiciones y funciones pueden utilizarse en el programa actual.

Otra de las características de Python es la utilización de entornos virtuales. Los entornos virtuales son una herramienta que permite instalar una versión de Python y sus dependencias de forma aislada, de forma que para cada proyecto se puede tener un entorno de Python distinto.

Para instalar Python utilizaremos Anaconda, un gestor de paquetes, de entornos y una distribución de Python con un gran número de paquetes. Anaconda viene con conda, que permite lanzar aplicaciones, gestionar paquetes y entornos mediante instrucciones de consola y con un navegador, que permite hacer lo mismo que conda pero mediante una interfaz gráfica.

## BAYESOPT

BayesOpt es una librería de optimización bayesiana para resolver problemas de optimización no lineal, diseño experimental y bandits. La librería 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 para los que obtenemos la mayor recompensa.

De los modos de utilización de la librería 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.

Además de la función, también debemos definir una serie de parámetros de optimización. A continuación, se resumen los parámetros más importantes:

* 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 numero de iteraciones se corresponde a una mayor precisión del resultado.
* 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.
* n\_init\_samples: este parámetro sirve para aprender un modelo preliminar de la función objetivo. n\_init\_samples es el número de muestras utilizadas para aprender este modelo. Cada muestra necesita una evaluación de la función objetivo.
* epsilon: este parámetro sirve para implementar una estrategia épsilon-greedy. El valor de este parámetro épsilon es la probabilidad de hacer una evaluación aletoria 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 numero de iteraciones sin ganancia antes de saltar a un punto aleatorio.
* l\_type: BayesOpt intenta aprender un modelo Bayesiano totalmente analítico para la función sustituta, pero los hiperparametros 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 Python los parámetros de BayesOpt se deben definir como un diccionario.

## GIT Y GITHUB

# DEFINICIÓN DEL PROBLEMA

## ENTORNO ROBÓTICO

Como se configura el entorno del robot: robot con mesa.

## TAREAS

En que consiste cada tarea y como se define en el simulador.

# SOLUCIÓN ADOPTADA

## DEFINICIÓN DE LAS TAREAS

Para cada tarea, como se define la política y la recompensa.

# SIMULACIÓN Y RESULTADOS

# CONCLUSIONES