

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

# RESUMEN

# ABSTRACT

ÍNDICE DE CONTENIDOS

[RESUMEN 2](#_Toc47975117)

[ABSTRACT 3](#_Toc47975118)

[1. OBJETO 5](#_Toc47975119)

[2. ALCANCE 6](#_Toc47975120)

[3. MARCO TEÓRICO 7](#_Toc47975121)

[3.1. APRENDIZAJE POR REFUERZO 7](#_Toc47975122)

[3.1.1. INTRODUCCIÓN 7](#_Toc47975123)

[3.1.2. ELEMENTOS DEL APRENDIZAJE POR REFUERZO 8](#_Toc47975124)

[3.1.3. PROCESOS DE DECISIÓN DE MARKOV 9](#_Toc47975125)

[3.1.4. BÚSQUEDA DE POLÍTICA 11](#_Toc47975126)

[3.2. OPTIMIZACIÓN BAYESIANA 11](#_Toc47975127)

[3.2.1. INTRODUCCIÓN 11](#_Toc47975128)

[3.2.2. DISTRIBUCIÓN A PRIORI 13](#_Toc47975129)

[3.2.3. FUNCIONES DE ADQUISICIÓN 15](#_Toc47975130)

[4. HERRAMIENTAS TECNOLÓGICAS 17](#_Toc47975131)

[4.1. COPPELIASIM 17](#_Toc47975132)

[4.1.1. INTRODUCCIÓN 17](#_Toc47975133)

[4.1.2. FUNCIONALIDAD DEL SIMULADOR 17](#_Toc47975134)

[4.1.3. CONTROL DE LA SIMULACIÓN 19](#_Toc47975135)

[4.2. PYREP 20](#_Toc47975136)

[4.3. RLBENCH 20](#_Toc47975137)

[4.4. PYTHON Y ANACONDA 20](#_Toc47975138)

[4.5. BAYESOPT 20](#_Toc47975139)

[4.6. GIT Y GITHUB 21](#_Toc47975140)

[5. DEFINICIÓN DEL PROBLEMA 21](#_Toc47975141)

[5.1. ENTORNO ROBÓTICO 21](#_Toc47975142)

[5.2. TAREAS 22](#_Toc47975143)

[6. SOLUCIÓN ADOPTADA 22](#_Toc47975144)

[6.1. DEFINICIÓN DE LAS TAREAS 22](#_Toc47975145)

[7. SIMULACIÓN Y RESULTADOS 22](#_Toc47975146)

[8. CONCLUSIONES 22](#_Toc47975147)

# 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

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.

El aprendizaje por refuerzo podría considerarse un tercer paradigma del aprendizaje automático, más allá del aprendizaje supervisado y del aprendizaje no supervisado. El 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 extrapolar o generalizar sus acciones para responder correctamente en situaciones no presentes en el conjunto de entrenamiento. Por ejemplo, en un conjunto de entrenamiento con fotos de perros y gatos correctamente etiquetado, el objetivo del agente sería aprender a diferenciar entre perros y gatos con la información de las imágenes y sus etiquetas, de forma que al recibir una foto de un gato que no estuviera en el conjunto de entrenamiento el agente pudiera clasificarla como un gato. El aprendizaje no supervisado se basa en aprender estructuras ocultas en conjuntos sin etiquetar.

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 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.

### 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 xt, 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.

### PROCESOS DE DECISIÓN DE MARKOV

#### Interacción agente-entorno

Los procesos de decisión de Markov (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 Markov, 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.

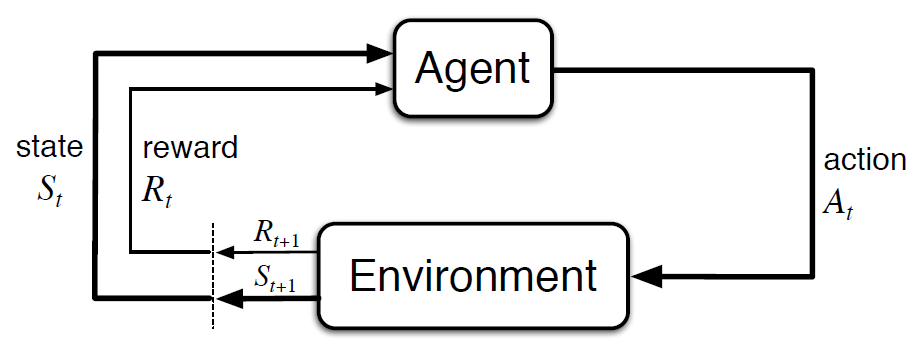


Fig. . – Interacción entre el agente y el entorno en un MDP

En un proceso de Markov finito, los conjuntos de acciones, estados y recompensas tienen un número de elementos finito. En la Fig. 3.1 se muestra la interacción entre el agente y el entorno en un proceso de decisión de Markov. En cada paso, el agente recibe una representación del estado del entorno, , y basado en esa representación ejecuta una acción . En el siguiente paso, como consecuencia de la acción, el agente recibe una recompensa y recibe la nueva representación del estado del entorno, Como consecuencia de la interacción entre el agente y el entorno se genera una secuencia de esta forma:

En el caso de procesos de Markov 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, y , 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, y .

Esta probabilidad define la dinámica del proceso de decisión de Markov. Esta función es coherente con las propiedades de la probabilidad condicional, es decir, que para unos valores concretos del estado y acción anterior, y , se cumple que:

En un proceso de decisión de Markov, 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 Markov, según la cual el estado posterior depende solo de estado actual.

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

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

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

El marco de referencia de los procesos de Markov para los problemas del aprendizaje de refuerzo es abstracto y flexible. Por ejemplo, los pasos no tienen porque 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 limites entre el agente y su entorno no tienen porque 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 Markov 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 Markov para los problemas de aprendizaje por refuerzo nos permite realizar un análisis mas amplio de los elementos del aprendizaje por refuerzo comentados en el apartado 3.1.2.

#### 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 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.

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 ultimo 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.

De esta forma, aunque la tarea sea continua, la utilidad esperada tiene un valor finito. es la tasa de descuento, donde . 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 mas alejadas en el tiempo.

#### 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 , entonces es la probabilidad de tomar la acción desde el estado .

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 bajo una política se denota y es la utilidad esperada empezando en y siguiendo la política . Para los procesos de decisión de Markov se puede definir como:

En la que denota el valor esperado de una variable aleatoria dado que el agente sigue una política . El valor de la función para el estado final es cero. se denomina la función de estado-valor para la política .

De forma similar, se puede definir el valor de tomar la acción desde el estado siguiendo la política , denotado como , como la utilidad esperada partiendo desde el estado , tomar la acción y después seguir la política . se denomina la función de acción-valor para la política .

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

De forma similar, podemos seguir el mismo procedimiento para la función acción-valor obteniendo:

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

Expandiendo los valores esperados se puede ver que:

Combinando las ecuaciones previas queda:

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

#### Política y función valor óptimas

### 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.

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.

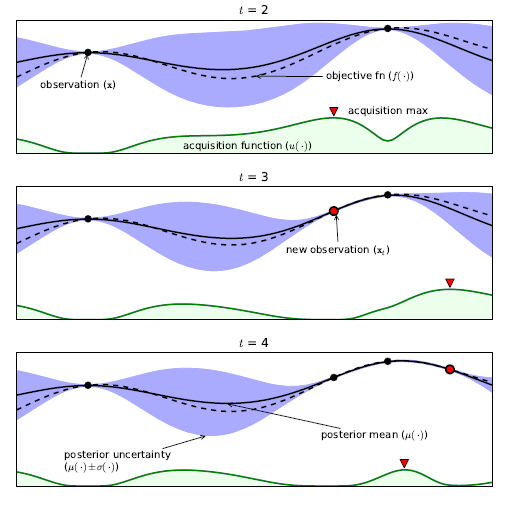


Fig. . – Ejemplo de optimización bayesiana

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 al función objetivo. La función de adquisición nos permite decidir donde muestrear.

A continuación, se va a tratar con mayor profundidad la distribución a priori y la función de adquisición.

### DISTRIBUCIÓN A PRIORI

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

1. La función de adquisición es continua y minimiza aproximadamente el riesgo, definido como la desviación esperada del mínimo global en un punto x.
2. 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 de Gauss. Para ello, es necesario especificar tres condiciones naturales adicionales:

1. La función objetivo es continua
2. La distribución a priori es homogénea
3. La optimización es independiente de la enésima derivada

Un proceso de Gauss es la extensión de la distribución de Gauss multivariante a un proceso estocástico de infinitas dimensiones, en el cualquier combinación de variables resulta en una distribución de Gauss. Un proceso gaussiano es una distribución sobre funciones, completamente especificado por una función de la media, m, y una función de la covarianza, k.

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 populares es la función exponencial al cuadrado:

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 mas probable que tengan una gran influencia entre ellos, mientras que si están mas lejos la influencia es probable que sea menor.

Al realizar la optimización, iremos guardando observaciones y el siguiente paso será decidir en que punto muestrear, , el cual tendrá un valor . Por las propiedades de los procesos de Gauss, y son conjuntamente gaussianos:

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 en ese caso cada punto esta perfectamente correlacionado consigo mismo.

Y la matriz k es:

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

Es decir, que con la media y la varianza son suficientes 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:

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:

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

### 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 los valores altos se correspondan con posibles valores altos de la función objetivo, valores con alta incertidumbre o ambos.

* Probabilidad de mejora (Probability of Improvement, PI)
* Mejora esperada (Expected Improvement, EI)
* Critero de intervalos de confianza (Confidence Bound Criteria)

En este capitulo 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, por último, posibles funciones de adquisición.

# 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.

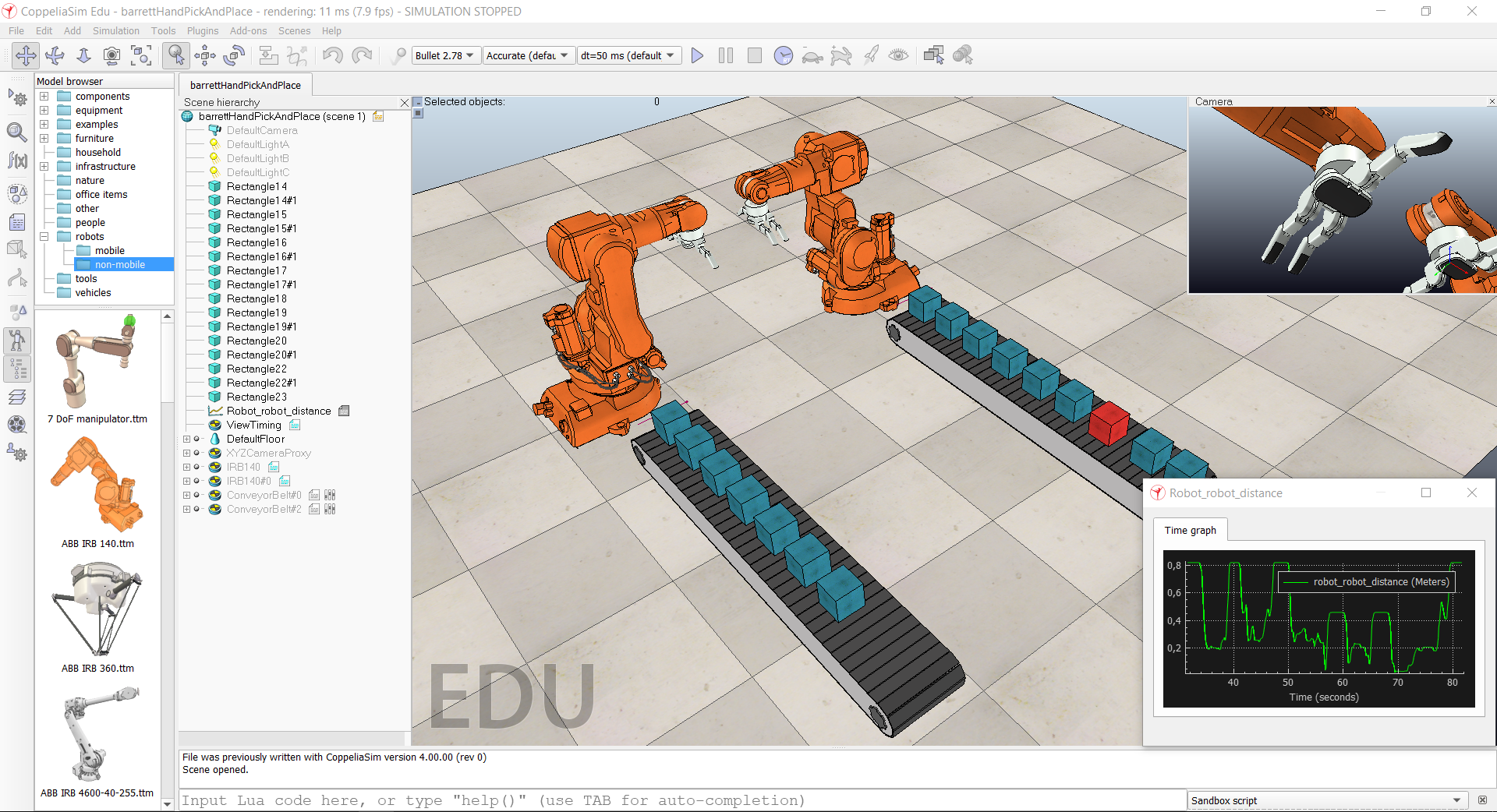


Fig. . – Interfaz de CoppeliaSim con una escena de ejemplo cargada.

### 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 decidir cuantas muestras se utiizan 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.
* 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 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 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