UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

ESTUDIO DE TÉCNICAS DE OPTIMIZACIÓN BAYESIANA APLICADAS A OPERACIONES DE PINTURA CON MANIPULADORES ROBÓTICOS

AUTORA: ELIZABET ZAKHARYAN

TUTOR: LEOPOLDO ARMESTO ÁNGEL

COTUTOR: ANTONIO SALA PIQUERAS

Curso Académico: 2020-21

RESUMEN

El presente TFM explorará el cálculo de parámetros óptimos para maniobras de robots simulados en CoppeliaSim utilizando las herramientas de optimización bayesiana de la Statistics and Machine Learning Toolbox de Matlab. Se desarrollará el entorno de simulación, la comunicación con Matlab y el análisis estadístico de resultados. Se planteará un primer caso de estudio de un robot lanzando una bola (juego de “petanca”), y se abordarán posteriormente el estudio de un manipulador robótico en operaciones de pintura que debe maximizar un cierto despeño en términos de tiempos de ejecución y cobertura de la superficie de pintado.

Palabras clave: Robótica; optimización bayesiana; control multivariable

RESUM

El present TFM explorarà el càlcul de paràmetres òptims per a maniobres de robots simulats en CoppeliaSim utilitzant les eines d'optimització bayesiana de la Statistics and Machine Learning Toolbox de Matlab. Es desenvoluparà l'entorn de simulació, la comunicació amb Matlab i l'anàlisi estadística de resultats. Es plantejarà un primer cas d'estudi d'un robot llançant una bola (joc de "petanca"), i s'abordaran posteriorment l'estudi d'un manipulador robòtic en operacions de pintura que ha de maximitzar una certa estimbada en termes de temps d'execució i cobertura de la superfície de pintat.

Paraules clau: Robòtica; optimització bayesiana; control multivariable

ABSTRACT

This Master Thesis will explore the computation of optimal parameters for simulated robot maneuvers in CoppeliaSim using the Bayesian Optimization of Matlab's Statistics and Machine Learning Toolbox. The simulation environment, communication with Matlab and the statistical analysis of results will be developed. A first case study of a robot throwing a ball will be proposed, and later on we will study the problem of a robotic manipulator in a painting task operation that must maximize the performance including execution time and overal coverage of the painted surface.

Keywords: Robotics; bayesian optimization; multivariable control

ÍNDICE

DOCUMENTOS CONTENIDOS EN EL TFM

- Memoria
- Presupuesto
- Anexos

ÍNDICE MEMORIA

ÍNDICE PRESUPUESTOS

ÍNDICE ANEXOS

ÍNDICE FIGURAS MEMORIA

ÍNDICE FIGURAS ANEXOS

ÍNDICE TABLAS MEMORIA

ÍNDICE TABLAS PRESUPUESTOS

ÍNDICE TABLAS ANEXOS

MEMORIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAPÍTULO 1. INTRODUCCIÓN

1.1. Objetivo del proyecto

Actualmente la robótica y la automatización de procesos juegan un papel imprescindible en la sociedad actual, especialmente en los procesos industriales. A medida que la sociedad avanza junto a la tecnología, las demandas de la misma son todavía mayores y, por tanto, el rendimiento de las industrias ha de aumentar para poder abastecer las necesidades del mundo actual. Obviamente, un aumento de la capacidad y el rendimiento de una industria implica un aumento en los costes y es por ello que surge la necesidad de automatizar los procesos.
Por tanto, gracias al desarrollo de nuevos sistemas automáticos se ha conseguido automatizar un porcentaje muy elevado de estos procesos, y por tanto aumentar el rendimiento del proceso con una disminución de los tiempos y los costes.
Así pues, el objetivo de este presente Trabajo Fin de Máster es demostrar que empleando la optimización bayesiana integrada en el software MATLAB, se puede controlar, con una programación sencilla, procesos industriales complejos y muy frecuentes en la industria. En este caso, se va a controlar un proceso de pintado mediante brazo robótico industrial, simulado con el software CoppeliaSim, para optimizar los parámetros más importantes del proceso con los que se va a lograr reducir los costes y el tiempo de dicho proceso.

1.2. Motivación del proyecto

La motivación principal del proyecto es implementar sistemas de control sencillos, como es la optimización bayesiana, a procesos industriales con una relativa complejidad para lograr optimizar los parámetros que intervienen en la correcta ejecución del mismo.
Sin embargo, otra de las motivaciones principales es poder aplicar los conocimientos que se han adquirido durante los estudios de Máster de Ingeniería Industrial en el segundo año de especialización en Automática, Robótica y Control de procesos.

CAPITULO 2. ESTADO DEL ARTE

2.1. Introducción al Machine Learning

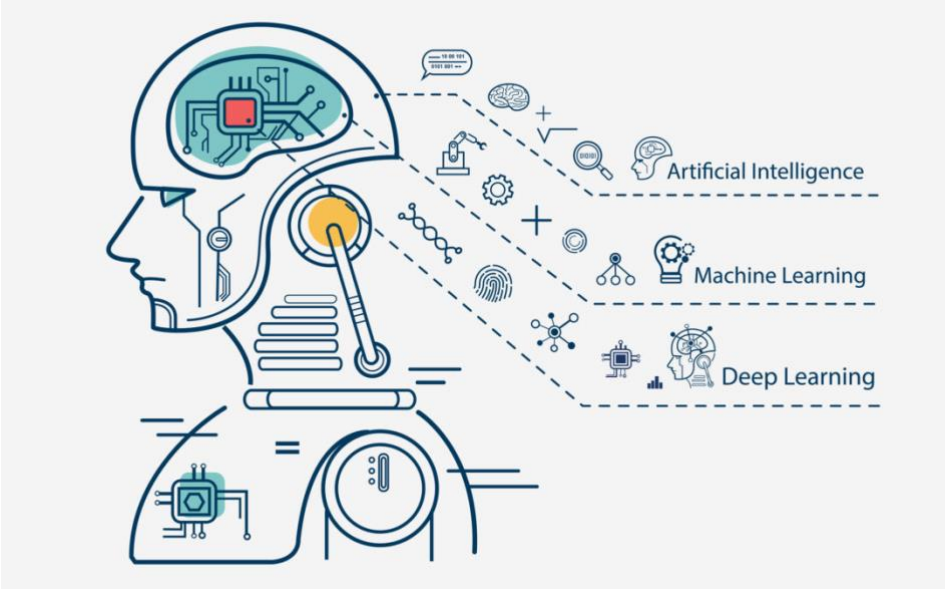

Figura 1. Machine Learning. [Fuente: https://www.atriainnovation.com/en/machine-learning-in-industry/]

El machine learning es una rama de la inteligencia artificial (IA) que actualmente juega un papel fundamental dentro de los avances de la Industria 4.0, y se trata de un proceso de aprendizaje automático de las máquinas, de manera que mediante el cual se logra que una máquina aprenda a realizar cierta acción de manera correcta y óptima, sin ser específicamente programado para ello. La máquina logra aprender a realizar una tarea de una manera óptima mediante el aprendizaje automático, que básicamente consiste en que, mediante una serie de datos transmitidos a la máquina, se construye un modelo y mediante este modelo realiza diferentes iteraciones hasta lograr el resultado óptimo.

2.1.1. Antecedentes históricos del Machine Learning

Este campo surge aproximadamente en la época de 1950, donde Alan Turning creó el Test de Turning que consistía en saber si una máquina era inteligente o no, y la máquina aprobaba el test si lograba hacer creer a un ser humano que él también era un ser humano.

Tras esto, en el año 1952 Arthur Samuel programó el primer algoritmo capaz de aprender según su experiencia. Este algoritmo consistía en el juego de damas y que mejoraba su forma de jugar después de la experiencia adquirida en cada partida.

En 1956 se organiza una conferencia mundial en donde oficialmente nace el campo de la Inteligencia Artificial y en 1958 Frank Rosenblatt diseña la primera red neuronal artificial denominada Perceptrón.

Hasta la segunda mitad de la década de los 70 no existen grandes avances en el campo y finalmente, un grupo de estudiantes de la Universidad de Stanford inventan un robot móvil denominado Stanford Cart que era capaz de moverse en un espacio cerrado evitando obstáculos.

Por otro lado, en la década de los 80 surgen grandes avances como el concepto de "Explanation Based Learning" creado por Gerald Dejong en el que un ordenador era capaz de analizar datos en base a previos entrenamientos y descartar los datos de menor relevancia.

Finalmente durante los años 90 los científicos logran crear algoritmos que son capaces de analizar grandes cantidades de datos y poder extraer conclusiones de los resultados que se obtienen. A partir de esta época, el Machine Learning ha ido creciendo, renovándose y desarrollando técnicas muy sofisticadas hasta la actualidad.

Actualmente, existen diversos tipos de algoritmos para poder implementar el Machine Learning en los que se va a profundizar en los siguientes subcapítulos.

2.2. Tipos de algoritmos

En machine learning se distinguen varías categorías, sin embargo solo se profundizarán en las tres principales: aprendizaje supervisado, aprendizaje sin supervisión y aprendizaje por refuerzo.

2.2.1. Aprendizaje supervisado

Tal y como afirma Taiwo Oladipupo Ayodele de la Universidad de Portsmouth del Reino Unido en el libro *"New Advances in Machine Learning (2016)"* :

"El aprendizaje supervisado es una técnica de aprendizaje de una función a partir de unos datos de entrenamiento. Estos datos de entrenamiento consisten en entradas (típicamente vectores) y valores de salida deseados.
La tarea del algoritmo supervisado por un recurso humano, consiste en predecir el valor de la función para cualquier tipo de entrada después de observar y aprender de los datos de entrenamiento. Para lograr esto, el algoritmo tiene que generalizar de los datos presentados y conocidos a situaciones desconocidas mediante un método razonable."

Además, según afirma Giuseppe Bonaccorso en su libro *"Machine Learning Algorithms (2018)"* los algoritmos actuales son capaces y flexibles de corregir los parámetros que crean necesarios y de reducir la magnitud de la función global de pérdida, de manera que tras cada iteración, si el algoritmo es lo suficientemente flexible y los datos son correctos, la diferencia entre los valores predecidos y esperados disminuye hasta ser cercano a 0.
El aprendizaje supervisado está compuesto de tres categorías de algoritmos, de clasificación, de redes neuronales y de regresión.

2.2.2.1. Algoritmo de clasificación

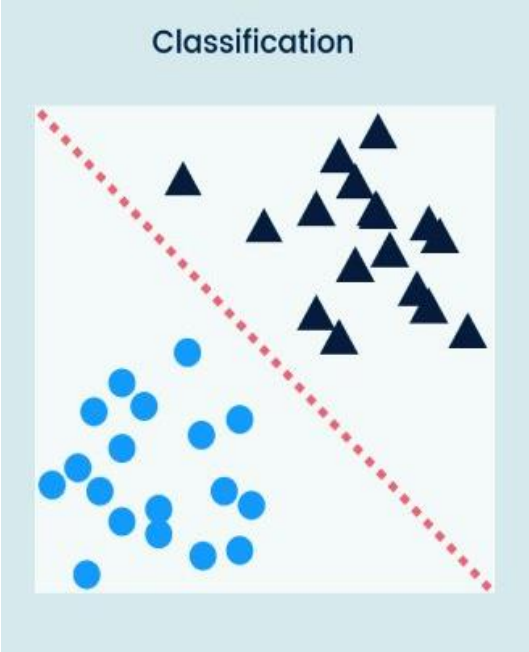

Figura 2. Algoritmo de clasificación. [Fuente: https://sooluciona.com/diferencias-entre-aprendizaje-supervisado-y-no-supervisado/]

El principal objetivo de la clasificación tal y como se observa en el ejemplo de la Figura 2, es clasificar los resultados en grupos o categorías. La clasificación puede ser binaria o multiclase, de manera que si solo existen dos tipos de clase es binaria y para más de dos tipos de clase es de tipo multiclase.

A continuación, se detalla brevemente el funcionamiento generalizado del algoritmo:

Se tiene un espacio E de representación del problema en el que existen M número de clases contenidos en dicho espacio, y un número n de muestras (x_1,x_2,…x_n) del espacio E tomadas de un conjunto de entrenamiento cuya fórmula representativa es:

$$Conjunto = \{X,\Omega\}=\{(x_1,\omega_1),(x_2,\omega_2),\ldots,(x_n,\omega_n)\} \quad (1)$$

Si se toma una nueva muestra x perteneciente a E, independiente del conjunto y que puede pertenecer a cualquiera de las M clases del espacio, entonces el algoritmo de clasificación ha de determinar a qué clase de E pertenece la muestra. Este clasificador se representa mediante:

$$\delta: E \to \Omega,\ siendo\ \delta(x) = \omega_i\ \ para\ i = 1,2, \dots M \quad (2)$$

Por tanto, el algoritmo calcula a qué clase M pertenece la muestra mediante el cálculo de la distancia métrica entre la muestra y los valores del conjunto de entrenamiento.

2.2.2.2. Algoritmo de regresión

Figura 3. Algoritmo de regresión. [Fuente: https://sooluciona.com/diferencias-entre-aprendizaje-supervisado-y-no-supervisado/]

Por otra parte, del artículo "*An Efficient Regression Classifier (2012)"* de los autores Hai Wang y Fei Hao publicado en IEEE Xplore, se ha podido comprender y resumir a continuación el funcionamiento del algoritmo de regresión.
Por tanto, la regresión es un proceso de tipo estadístico en el que se intenta predecir valores de tipo continuo a partir de la relación de variables dependientes e independientes y este valor intenta ajustarse a una recta. La regresión lineal puede ser simple o múltiple, dependiendo del número de variables que intervienen en el proceso.

Por ejemplo, la forma de una función de regresión lineal simple es la siguiente:

$$y = \beta_0 + \beta_1 x + \in \quad (3)$$

Siendo β_0 la ordenada, β_1 la pendiente de la función y $\in$ la perturbación o error aleatorio en la toma de muestras, que es la diferencia entre el valor real y el valor muestreado. La predicción en la regresión consiste en intentar minimizar el valor del error a 0 mediante el método de mínimos cuadrados, que consiste en minimizar la suma de los cuadrados de los errores.

$$\sum_{i=1} \in_i^2 \quad (4)$$

2.2.2.3. Algoritmo de redes neuronales

Finalmente, en cuanto a las redes neuronales, se puede observar el funcionamiento de una neurona en la Figura 4:

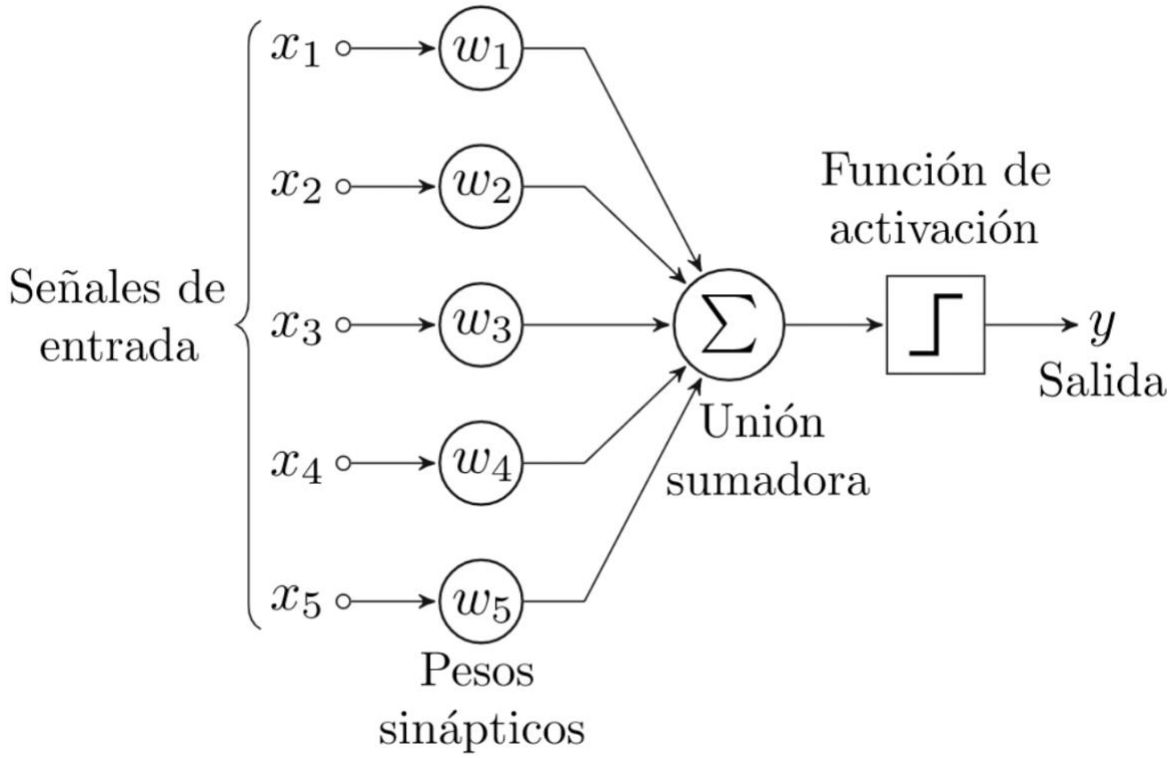

Figura 4. Esquema de funcionamiento de una neurona. [Fuente: https://empresas.blogthinkbig.com/las-matematicas-del-machine-learning-redes-neuronales-parte-i/]

En primer lugar, las señales de entrada son un número n de entradas independientes entre sí. Estas entradas posteriormente se multiplican por sus respectivos pesos sinápticos obteniendo como resultado una función denominada función de ponderación que es la suma de cada multiplicación de la variable de entrada por su respectivo peso en el proceso según la siguiente fórmula:

$$X \cdot W_t = (x_1, x_2, \dots x_n) \cdot \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \sum_{i=1} x_{i \cdot w_i} \quad (5)$$

A continuación, a esta función se le aplica una función de aplicación, que es una función mediante la cual se transmite el resultado a la salida, normalmente con modificaciones. Si no se decide modificar la información, entonces la función de activación será una función identidad, en caso contrario existen diversos tipos de funciones de aplicación que modifican la información.

$$\phi(\sum_{i=1} x_{i \cdot w_i}) \quad (6)$$

Las funciones de aplicación que modifican la información son las siguientes:

- Función escalón

$$\phi(x) = \begin{cases} 0 \; si \; x < 0 \\ 1 \; si \; x \geq 0 \end{cases} \quad (7)$$

- Función sigmoidal

$$\phi(x) = \frac{1}{1+e^{-x}} \quad (8)$$

- Función tangente hiperbólica

$$\phi(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \quad (9)$$

- Función rectificadora

$$\phi(x) = max\{0,x\}, siendo \; x \geq 0 \quad (10)$$

- Función de base radial(cuadráticas, gaussianas, etc)

Por tanto, una red neuronal está formada por un conjunto de neuronas con el funcionamiento descrito anteriormente, que van transmitiendo la información entre ellas hasta finalmente obtener una respuesta válida.

Cabe destacar que este algoritmo puede ser tanto de aprendizaje supervisado como de aprendizaje sin supervisión.

2.2.2. Aprendizaje sin supervisión

En este caso, según los autores M. Mohammed, MB. Khan y EBM. Bashier del libro *"Machine Learning: Algorithms and applications (2016*)" :
"En el aprendizaje sin supervisión no existen supervisores o datos de entrenamiento, todo lo que se tiene son datos sin etiquetas."
La máquina es la que, analizando los de datos disponibles ha de encontrar una solución óptima al proceso sin supervisión. Así pues, la máquina es capaz de analizar grandes volúmenes de datos y mejorar su capacidad de toma de decisiones conforme más datos analice.

Por tanto, es un tipo de aprendizaje contrario al aprendizaje con supervisión, ya que se trata de una red neuronal artificial no supervisada que recibe y analiza datos buscando similitudes, conexiones y patrones entre estos datos, al contrario de lo que ocurre en el aprendizaje supervisado que este tipo de similitudes y conexiones ya son dadas anteriormente al algoritmo. Por tanto, el algoritmo es el encargado de buscar conexiones y tomar decisiones en base a los datos analizados.

Las técnicas más conocidas en el aprendizaje sin supervisión son los algoritmos de análisis por grupos o clustering y los algoritmos de asociación. Sin embargo, las redes neuronales también forman parte del aprendizaje sin supervisión.

Finalmente, una de las características más importantes del aprendizaje no supervisado es que se puede combinar con algoritmos de inferencia bayesiana para analizar variables independientes entre sí y crear probabilidades condicionales, logrando un aprendizaje supervisado.

2.2.2.1. Algoritmo de clustering

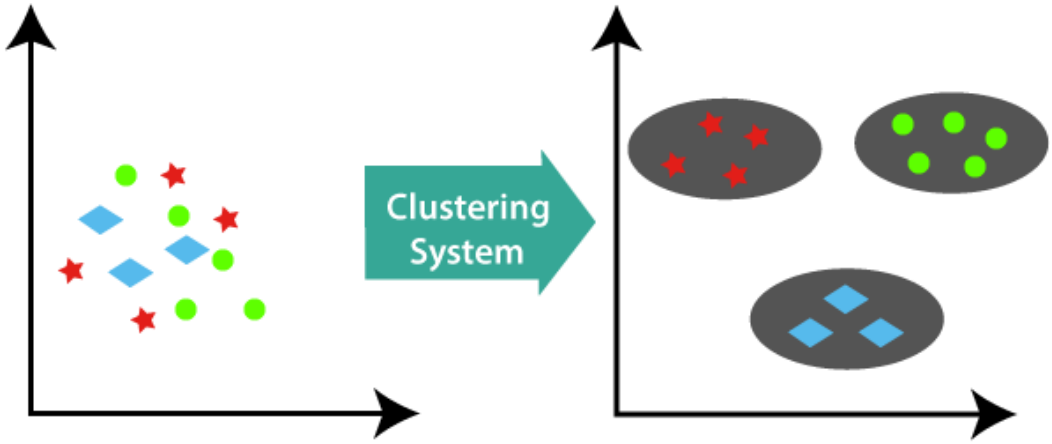

Figura 5.Clustering en Machine Learning. [Fuente: https://www.tutorialandexample.com/ml-clustering-algorithm/]

Mediante el clustering el algoritmo busca formar grupos de datos similares y por tanto, divide los datos en grupos con el mismo patrón y similitud como se puede observar en la Figura 5.

Así, el algoritmo funciona del siguiente modo según un artículo de la Universidad de Oviedo:

*"**Inicialización:** una vez escogido el número de grupos, k, se establecen k centroides en el espacio de los datos, por ejemplo, escogiéndolos aleatoriamente.*

***Asignación objetos a los centroides:** cada objeto de los datos es asignado a su centroide más cercano.*

***Actualización centroides:** se actualiza la posición del centroide de cada grupo tomando como nuevo centroide la posición del promedio de los objetos pertenecientes a dicho grupo.*

Se repiten los pasos 2 y 3 hasta que los centroides no se mueven, o se mueven por debajo de una distancia umbral en cada paso.

*El algoritmo k-means resuelve un **problema de optimización**, siendo la función a optimizar (minimizar) la suma de las distancias cuadráticas de cada objeto al centroide de su cluster.*
Los objetos se representan con vectores reales de d dimensiones ($x_1,x_2,\ldots x_n$) y el algoritmo k-means construye k grupos donde se minimiza la suma de distancias de los objetos, dentro de cada grupo $S=\{S_1,S_2,\ldots S_n\}$, a su centroide. El problema se puede formular de la siguiente forma:

$$\min E\ (\mu_i)=min\ \sum_{i=1}^{k}\sum_{x_j\in S_i}\|x_j-\mu_i\|^2 \quad (11)$$

Donde S es el conjunto de datos cuyos elementos son los objetos $\mathbf{x}j$ representados por vectores, donde cada uno de sus elementos representa una característica o atributo. Tendremos k grupos o clusters con su correspondiente centroide (μ_i) .
En cada actualización de los centroides, desde el punto de vista matemático, imponemos la condición necesaria de extremo a la función $E\ (\mu_i)$ que, para la función cuadrática (1) es:

$$\frac{\partial E}{\partial \mu_i} = 0 \rightarrow \mu_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j \quad (12)$$

y se toma el promedio de los elementos de cada grupo como nuevo centroide.
Las principales ventajas del método k-means son que es un método sencillo y rápido. Pero es necesario decidir el valor de k y el resultado final depende de la inicialización de los centroides. En principio no converge al mínimo global sino a un mínimo local."

Fuente: *Universidad de Oviedo.*
https://www.unioviedo.es/compnum/laboratorios_py/kmeans/kmeans.html

2.2.2.2. Algoritmo de asociación

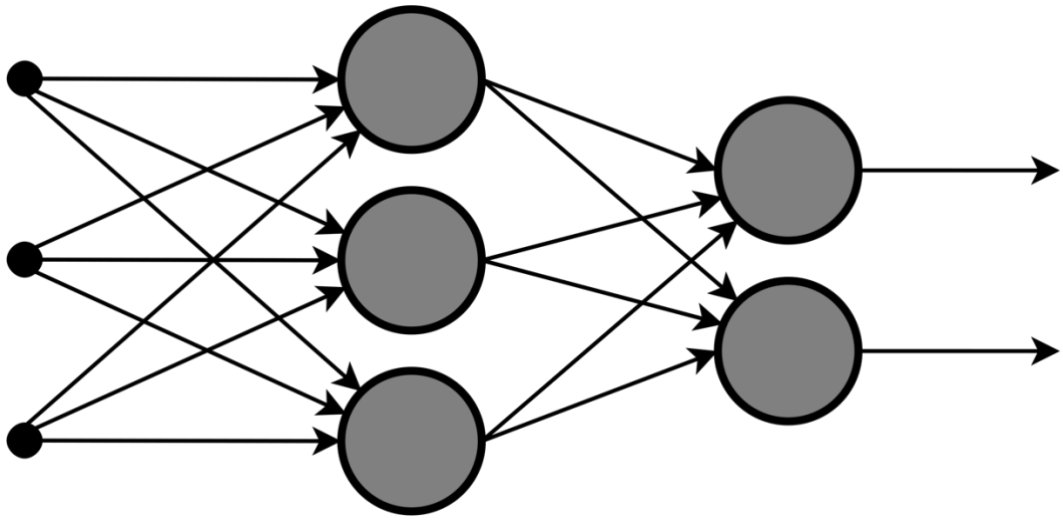

Figura 6. Funcionamiento algoritmo de asociación. [Fuente: Wikipedia]

El algoritmo de asociación ordena los datos combinándolos en base a las similitudes que comparten de manera que logra reducir la dimensión de los datos que recibe.

Tal y como se observa en la Figura 6, el funcionamiento del algoritmo es el siguiente:

Se tienen una serie de entradas que se combinan en grupos según la similitud que comparten, reduciendo la dimensión de datos. Tras esto, mediante la combinación de grupos formados anteriormente según su similitud o grado de implicación en el resultado, se obtienen los resultados de salida.

2.2.3. Aprendizaje por refuerzo

Este tipo de aprendizaje según Giuseppe Bonaccorso en su libro *"Machine Learning Algorithms (2018)"* está basado en el *feedback* que se recibe del entorno y por tanto, este tipo de aprendizaje tiene como concepto el proceso de ensayo y error. Se proporciona un algoritmo de aprendizaje con una serie de datos y la máquina realiza una serie de ensayos mediante dichos datos y comienza a aprender de los resultados obtenidos en cada ensayo, hasta lograr un resultado final óptimo.

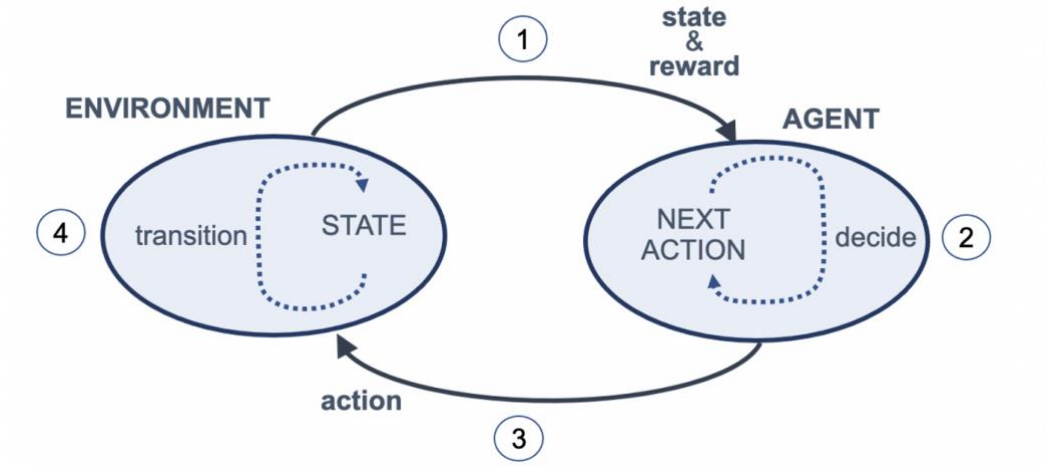

Figura 7. Ciclo de aprendizaje por refuerzo. [Fuente: https://medium.com/aprendizaje-por-refuerzo/1-introducción-al-aprendizaje-por-refuerzo-92c9239aed90]

Como se observa en la Figura 7, el ciclo de aprendizaje por refuerzo funciona del siguiente modo:

El agente observa el entorno a partir del estado y la recompensa que ha obtenido de ejecutar su anterior acción. En base a estos resultados, el agente decide cuál será su siguiente acción sobre el entorno. Cuando ejecuta la acción correspondiente, el entorno reacciona a este cambiando su estado y devolviendo nuevamente un nuevo estado. Este ciclo se repite hasta que el agente logra controlar el entorno.

En este caso uno de los algoritmos más destacados son los algoritmos de inferencia bayesiana. La inferencia bayesiana tiene como base que las evidencias u observaciones, permiten al algoritmo deducir la probabilidad de que una cierta hipótesis pueda ser cierta o no. Por tanto, cuanto mayor sea el número de observaciones, mayor capacidad de predicción tendrá el algoritmo en las probabilidades.

2.3. Elección del algoritmo para resolver el problema

El problema de este proyecto consiste en el proceso de pintura de una puerta de automóvil en el que se quieren controlar cuatro variables del proceso: la distancia entre la herramienta de pintura y la puerta, la distancia entre las pasadas que ha de realizar el robot, la velocidad del robot y el ángulo de apertura de la boquilla de pintura.

En este caso, tanto la función del modelo del proceso como los valores óptimos de los cuatro hiperparámetros son desconocidos y por tanto, se ha de seleccionar el método que más se adecue a las condiciones del problema a resolver.

Si se analizan los diversos tipos de algoritmos de Machine Learning, el algoritmo que más se adapta a este caso es el algoritmo de aprendizaje por refuerzo, que básicamente consiste en como se ha comentado anteriormente, en un proceso de ensayo y error, hasta que el algoritmo detecte una solución óptima.

Para ello, se ha decidido utilizar este tipo de algoritmo y concretamente la optimización bayesiana que ofrece el software MATLAB y cuyo funcionamiento se detallará en los próximos capítulos.

CAPÍTULO 3. SIMULACIÓN DE ROBOTS CON COPPELIASIM

3.1. Introducción a CoppeliaSim

Figura 8. Logo de CoppeliaSim. [Fuente: https://www.coppeliarobotics.com/helpFiles/en/welcome.htm]

CoppeliaSim es un software utilizado tanto a nivel industrial como a nivel académico para simulación de procesos automáticos y procesos robóticos. Es un programa muy completo y versátil en el que puedes combinar diferentes tipos de proceso industrial y controlarlos mediante diferentes software/lenguajes de programación como C/C++, Java, Python, Lua, Matlab u Octave.

En este proyecto se ha empleado la versión 4.2. de CoppeliaSim versión EDU, en el que se utiliza Lua como lenguaje de programación.

3.2. Lenguaje de programación Lua

Figura 9. Logo de Lua. [Fuente: http://elb105.com/esp8266-parte-1-programacion-en-lua-y-arduino/lua-logo-nolabel-svg/]

Lua es un lenguaje de programación creado por profesores de la Universidad Católica de Rio en los años 90. Lua está clasificado dentro de la categoría de lenguajes de programación de bajo nivel, lo que quiere decir que no requiere muchas líneas de código y no es necesaria una estructura compleja para poder programar con él.
Sus principales características son:

- Código Opensource
- Gestión de memoria automática mediante la cual se puede realizar una limpieza automática de los datos para liberar memoria
- API simple
- Gran rendimiento
- No existen limitaciones para los desarrolladores
- Las variables solo pueden ser lógicas, enteros, floats o cadenas de caracteres

3.3. Interfaz CoppeliaSim

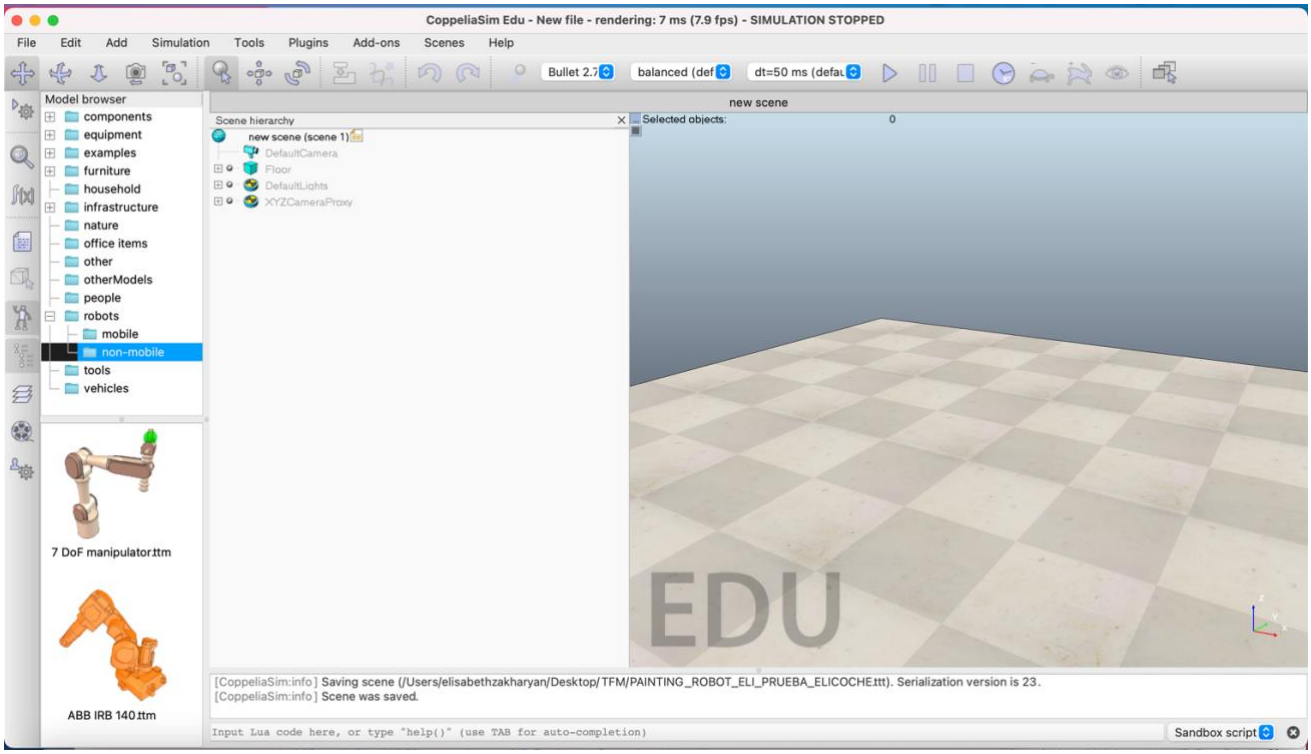

Figura 10. Interfaz CoppeliaSim

A modo de introducción al programa, se puede ver en la Figura 10 la interfaz gráfica del programa.
Como se puede observar, en la parte izquierda (Model browser) está disponibles una gran variedad de modelos para incorporar en la simulación (sensores, robots, muebles, infraestructura, etc.).
En la parte central se tiene la jerarquía de la escena, donde están ordenados jerárquicamente todos los modelos que intervienen en la simulación, y cada uno de estos modelos puede, si así se requiere, tener un script asociado al mismo. CoppeliaSim ofrece dos tipos de script: threaded y non-threaded.
El primer tipo de script es el threaded y se programa mediante corutinas que preventivamente se van interrumpiendo en intervalos regulares y reanudándose posteriormente.
Este tipo de script tiene ciertas desventajas respecto al non-threaded ya que consume mayores recursos y por tanto el coste computacional es mayor que en el otro tipo.

El segundo tipo de script, el non-threaded se basa en funciones de tipo callback que no están hiladas entre sí, por tanto, no hay intercambio de funciones y el tiempo computacional es menor que el primer tipo.
En este proyecto se han empleado los dos tipos de script y más adelante se detallarán los motivos por los que se ha escogido un tipo u otro.

3.4. Entorno programa robot pintura

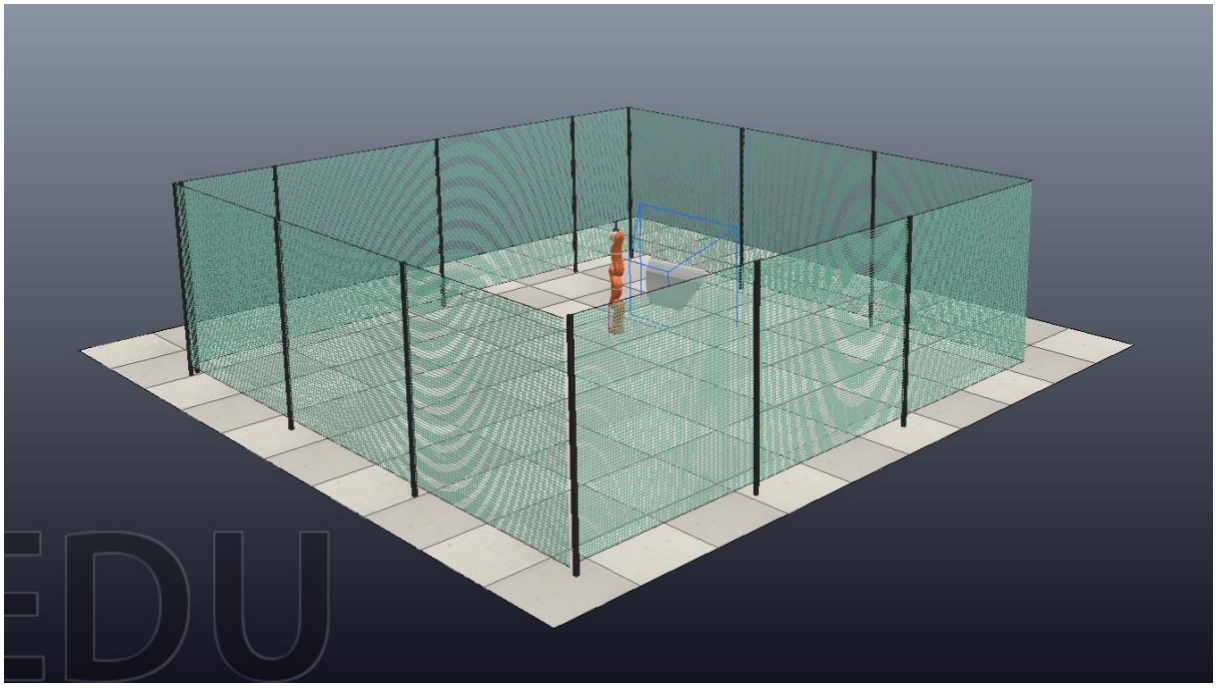

Figura 11. Entorno simulación proceso de pintado

El entorno del programa del robot de pintura se puede observar en la Figura 11. Está compuesto por una celda de seguridad, el modelo de robot LBR4p de KUKA, una herramienta de pintura, una cámara situada en la base del robot y una puerta de automóvil.

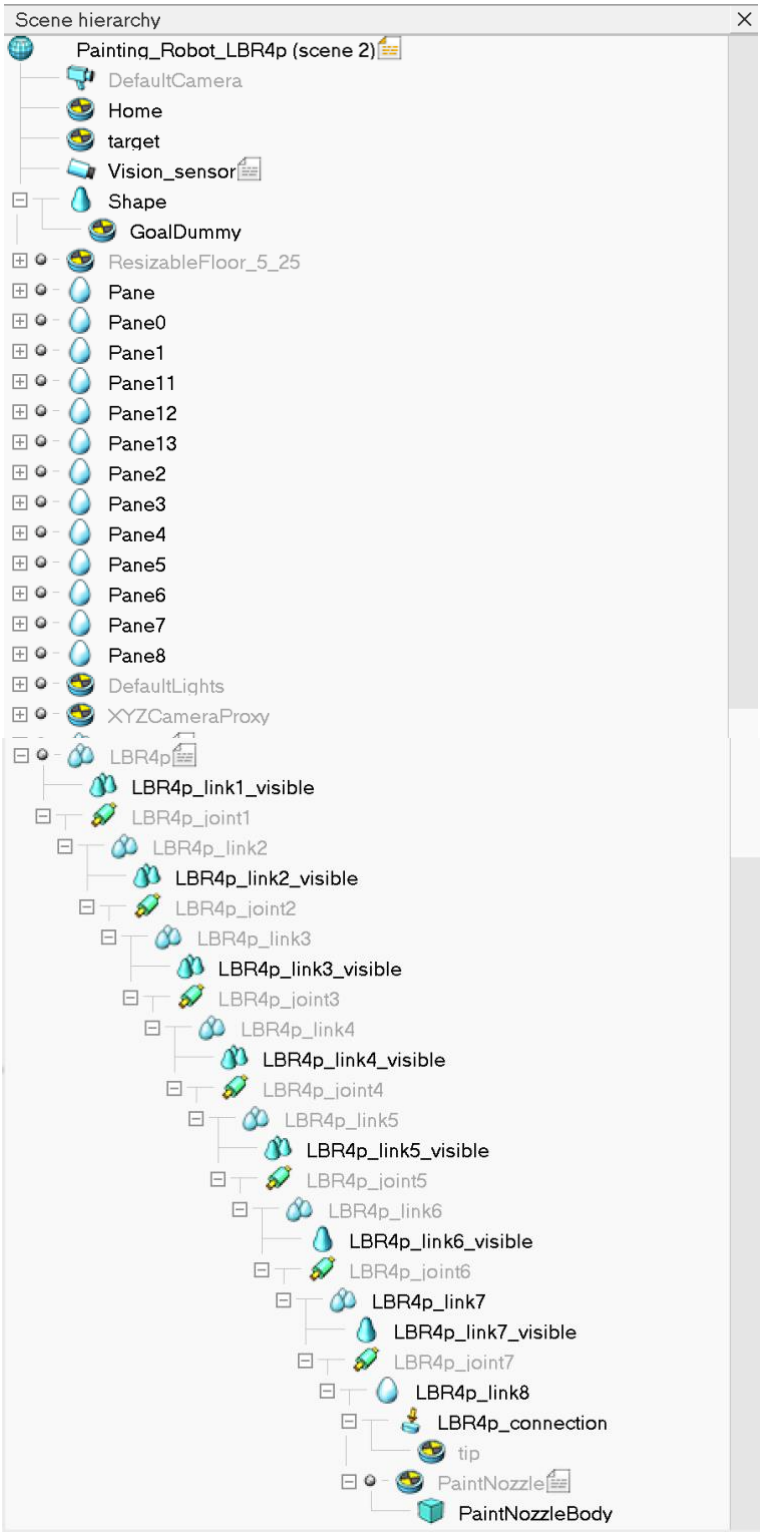

Figura 12. Jerarquía de la escena del proceso de pintura

En la Figura 12 se observa la jerarquía de la escena, siendo los objetos Home, target, GoalDummy, tip y PaintNozzle Dummies.

Un Dummy es una especie de variable indicadora que se sitúa en el espacio o se asocia a un objeto para poder calcular distancias, programar trayectorias que ha de seguir el dummy asociado al objeto, logrando que el objeto también realice esa trayectoria.

Los dummies que se han definido en la simulación tienen la siguiente función:

- Home : Define la posición de reposo del robot.
- GoalDummy: Es el dummy que va cambiando su posición para definir la trayectoria que ha de seguir el robot para realizar el pintado de la puerta del automóvil
- target: Es el dummy que sigue la trayectoria que realiza GoalDummy
- tip: Es el dummy asociado al robot y que sigue a target, de manera que el robot en su conjunto realiza la trayectoria marcada por los dos dummies detallados anteriormente.
- PaintNozzle: es el dummy asociado al PaintNozzle en el que se ha programado el algoritmo de la herramienta de pintura.

En segundo lugar, los objetos de nombre Pane, son los paneles que conforman la celda, de tamaños 2x2m y dos paneles de tamaño 1x2m siendo uno de ellos la puerta de entrada a la celda.

Por otro lado, se dispone de un sensor de visión cuyas propiedades se observan en la Figura 13:

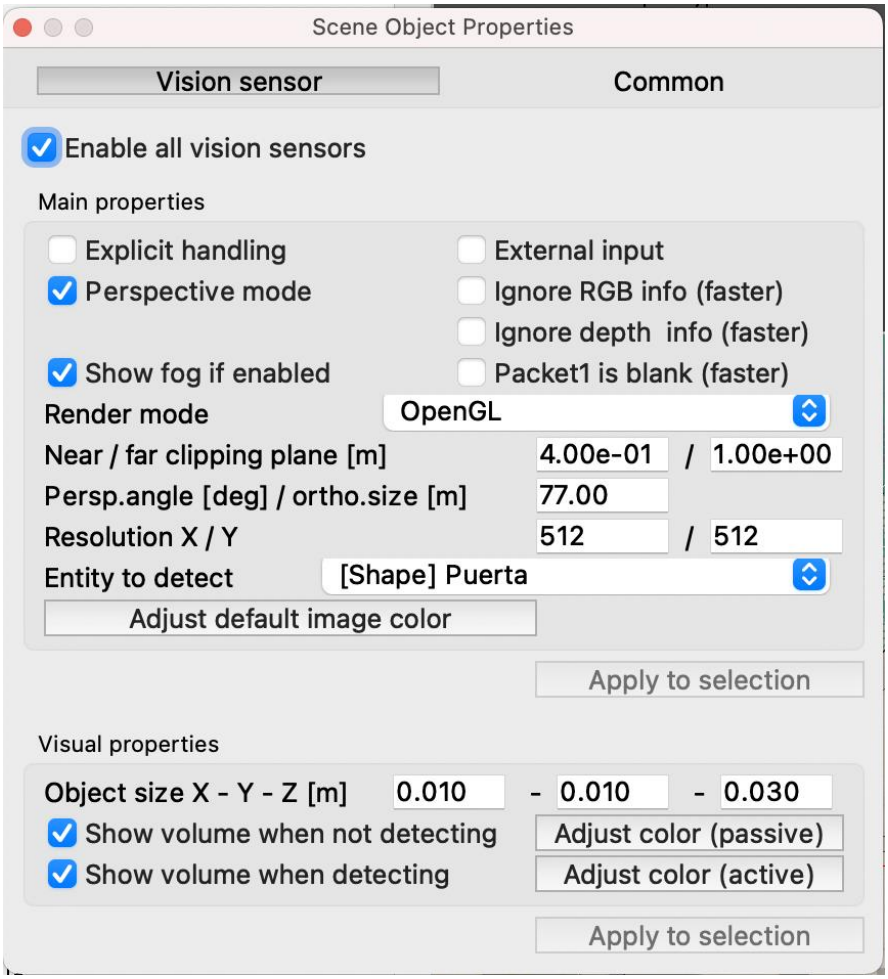

Figura 13. Propiedades de la cámara

Así, la entidad a detectar es solamente la puerta del automóvil para que se contabilicen los píxeles de pintura en la puerta y no se tengan en cuenta aquellos píxeles de pintura que durante el proceso de pintura es posible que manchen el suelo en el que está situado la puerta.

Para comprender mejor las propiedades escogidas para la cámara en la Figura 14 se puede ver el significado de cada parámetro.

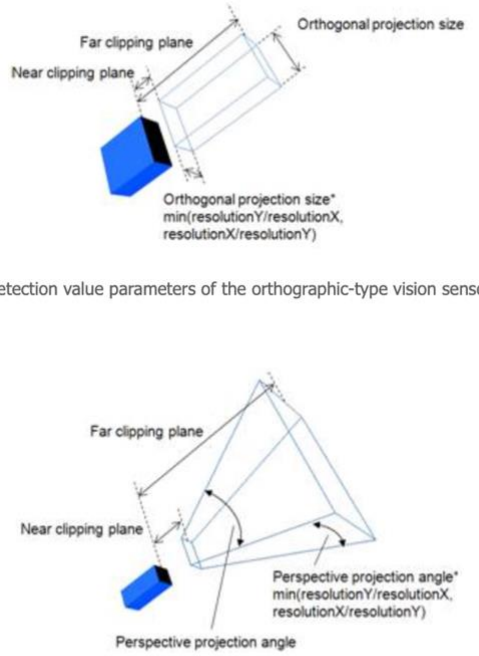

Figura 14. Descripción de los parámetros de la cámara. [Fuente: https://www.coppeliarobotics.com/helpFiles/en/visionSensorPropertiesDialog.htm]

La resolución que se ha escogido para la cámara es de 512x512 píxeles para obtener una imagen clara y precisa de la puerta y poder cuantificar los píxeles de una manera más exacta y obtener datos similares a los que se obtendrían si el proceso fuera real.

A continuación, en la Figura 15 se observa que el brazo robot industrial escogido para esta simulación es de la marca KUKA el modelo LBR4p.
Sin embargo, en este proyecto no se ha profundizado en los aspectos técnicos para escoger el robot adecuado, simplemente se ha escogido este robot, ya que su alcance y sus características han permitido realizar con éxito el proceso de pintado.

Por otra parte, se observa la cámara situada en la base del robot y la herramienta de pintura instalada en el extremo del brazo.

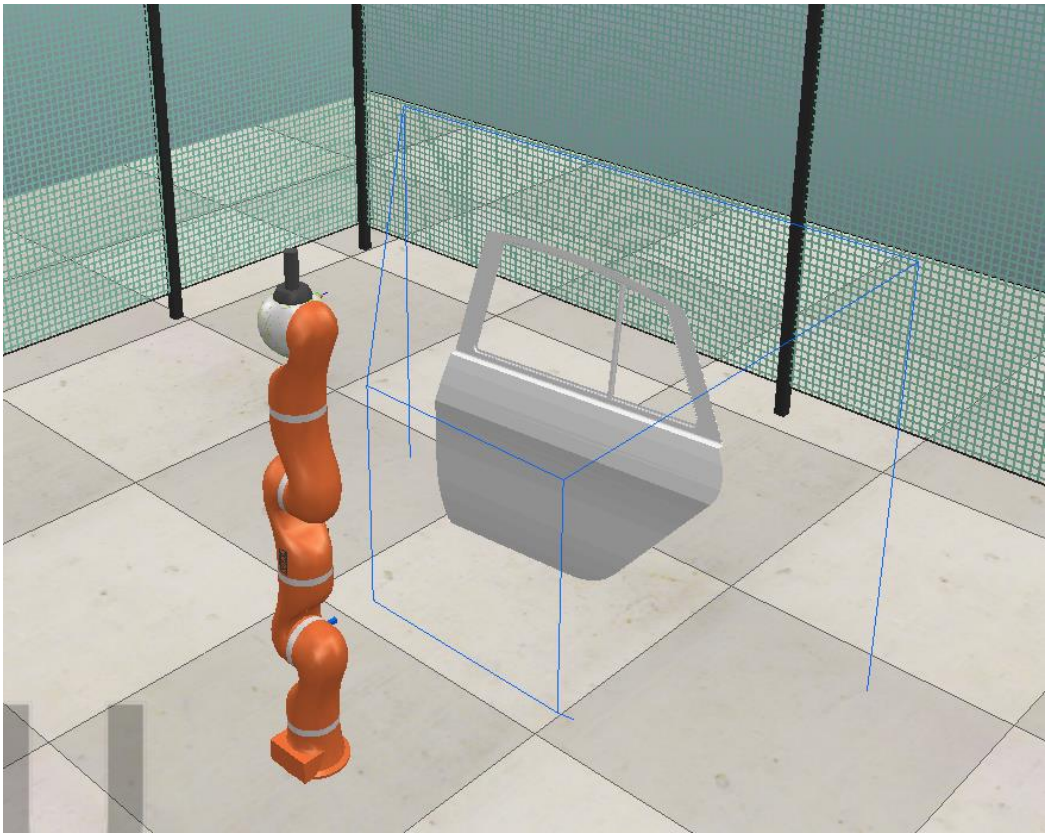

Figura 15. Robot LBR4p

Por tanto, se ha diseñado un entorno de simulación lo más similar a un entorno real del proceso dentro de una nave industrial. Para ello, se ha situado el robot dentro de una celda, por razones obvias de seguridad, ya que se trata de un robot industrial que podría dañar gravemente a los operarios si no se respeta su zona de trabajo.

Por último, la configuración de simulación de CoppeliaSim para este proceso es el siguiente:

Dynamics Engine: *Bullet V2.78* (por defecto)
dt: 50ms. (por defecto)

Por otro lado, el código de la simulación en CoppeliaSim es de tipo threaded, ya que las funciones del movimiento del robot solamente se podían implementar con un código de tipo threaded, está asociado al robot LBR4p y está compuesto por cuatro funciones resumidas a continuación:

- **Función RecibeDatos:** mediante esta función se reciben los datos enviados desde MATLAB a CoppeliaSim para poder realizar la simulación. Son tres datos de tipo float (longitud, distancia de pasadas y velocidad) y de tipo entero (ángulo de apertura de la boquilla).
  Además, se definen dos variables *suma_y* y *resta_z* que son de tipo bool y se inicializan a false y true respectivamente ya que sirven para programar al robot para que suba, baje o se mueva hacia la derecha en el proceso de pintado.

```
function RecibeDatos(inInts,inFloats,inStrings,inBuffer)
 dataFromExternalApp=inFloats
jetAngle=inInts[1]
 print("datos",dataFromExternalApp)
sim.callScriptFunction("cambiarboquilla@PaintNozzle",sim.scripttype_childscript,jetAngle)
   return {},{dataFromExternalApp,opt},{},''
end
```

- **Función sysCall_init() :** En esta función se crea la corutina, que se ha de ejecutar para el pintado y se define la cinemática inversa de la simulación para poder llevar a cabo cada uno de los movimientos de los que está compuesto el proceso.

```
function sysCall_init()
   corout=coroutine.create(coroutineMain)
   suma_y=false
   resta_z=true
   simTip=sim.getObjectHandle('tip')
   simTarget=sim.getObjectHandle('target')
   local simBase=sim.getObjectHandle('LBR4p')
   -- create an IK environment:
   ikEnv=simIK.createEnvironment()
   -- create an IK group:
   ikGroup_undamped=simIK.createIkGroup(ikEnv)
   -- set its resolution method to undamped:
simIK.setIkGroupCalculation(ikEnv,ikGroup_undamped,simIK.method_undamped_pseudo_inverse,0,
5)
   -- create an IK element based on the scene content:

simIK.addIkElementFromScene(ikEnv,ikGroup_undamped,simBase,simTip,simTarget,simIK.constraint
_x+simIK.constraint_y+simIK.constraint_z+simIK.constraint_alpha_beta)

   ikTip=simIK.createDummy(ikEnv)
   ikElement_undamped=simIK.addIkElement(ikEnv,ikGroup_undamped,ikTip)

simIK.setIkElementConstraints(ikEnv,ikGroup_undamped,ikElement_undamped,simIK.constraint_x+s
imIK.constraint_y+simIK.constraint_z+simIK.constraint_alpha_beta)

   -- create another IK group:
   ikGroup_damped=simIK.createIkGroup(ikEnv)
```

```
   -- set its resolution method to damped:

simIK.setIkGroupCalculation(ikEnv,ikGroup_damped,simIK.method_damped_least_squares,0.01,100)
   -- create an IK element based on the scene content:

simIK.addIkElementFromScene(ikEnv,ikGroup_damped,simBase,simTip,simTarget,simIK.constraint_x
+simIK.constraint_y+simIK.constraint_z+simIK.constraint_alpha_beta)
   ikTip=simIK.createDummy(ikEnv)
   --ikTarget=simIK.createDummy(ikEnv)
   ikElement_damped=simIK.addIkElement(ikEnv,ikGroup_damped,ikTip)

simIK.setIkElementConstraints(ikEnv,ikGroup_damped,ikElement_damped,simIK.constraint_x+simIK.
constraint_y+simIK.constraint_z+simIK.constraint_alpha_beta)

   tiempo=0
end
```

- Función sysCall_actuation(): En esta función se termina de aplicar la cinemática inversa.

```
function sysCall_actuation()
   if coroutine.status(corout)~='dead' then
      local ok,errorMsg=coroutine.resume(corout)
      if errorMsg then
         error(debug.traceback(corout,errorMsg),2)
      end
   end

    simIK.applyIkEnvironmentToScene(ikEnv,ikGroup_damped)

end
```

- Función coroutineMain(): En esta función se ha programado todo el proceso de pintado y se va a ir desglosando y explicando por partes.

```
function coroutineMain()
   local camera=sim.getObjectHandle('Vision_sensor')
   local flags=-1
   local currentPose=sim.getObjectPose(sim.getObjectHandle('target'),-1)
   local maxAccel={0.7,0.7,0.7,0.7}
   local maxJerk={0.4,0.4,0.4,0.4}
   local goaldummy=sim.getObjectHandle('GoalDummy')
   local redCnt=0
   local home=sim.getObjectHandle('Home')
   z_1=0.14
   z_2=0.83

local function callback(currentPose,currentVel,currentAccel,auxData)
 sim.setObjectPose(simTarget,-1,currentPose)
end
```

En las anteriores líneas se han definido los handle de la cámara y los dummy, se definen la máxima aceleración y el arranque a valores de 0.7m/s^2 y 0.4, respectivamente, ya que al realizar diversas pruebas se ha concluido que son valores adecuados para que el robot no sufra desviaciones ni tenga movimientos bruscos, se ha inicializado la variable que almacena los píxeles en rojo, y se definen las variables *z_1* y *z_2* con valores de 0.14m y 0.83m ya que son las posiciones en el eje Z que delimitan la puerta del automóvil y entre las que se mueve el robot en el proceso de pintado de la puerta.

```
while true do
if dataFromExternalApp then
longitud=dataFromExternalApp[1]
d_pasadas=dataFromExternalApp[2]
vel=dataFromExternalApp[3]
local maxVel={vel,vel,vel,vel}
posdummy=sim.getObjectPosition(goaldummy,-1)
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud-0.4,posdummy[2],posdummy[3]})
targetPose=sim.getObjectPose(goaldummy,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)

currentPose=sim.getObjectPose(simTarget,-1)
posdummy=sim.getObjectPosition(goaldummy,-1)
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud-0.2,posdummy[2],posdummy[3]})
targetPose=sim.getObjectPose(goaldummy,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)

currentPose=sim.getObjectPose(simTarget,-1)
posdummy=sim.getObjectPosition(goaldummy,-1)
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud,posdummy[2],posdummy[3]})
targetPose=sim.getObjectPose(goaldummy,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)
tiempo=sim.getSimulationTime()



while posdummy[2]>-0.05 do
posdummy=sim.getObjectPosition(goaldummy,-1)
if (suma_y==true) then
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud,posdummy[2]-
d_pasadas,posdummy[3]})
targetPose=sim.getObjectPose(goaldummy,-1)
currentPose=sim.getObjectPose(simTarget,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)
suma_y=false
else
if (resta_z==true) then
if (posdummy[3]<0.25) then
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud+0.1,posdummy[2],z_1})
```

```
else
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud,posdummy[2],z_1})
end
if (posdummy[2]<=0.05) then
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud+0.1,posdummy[2],z_1+0.1})
end

targetPose=sim.getObjectPose(goaldummy,-1)
currentPose=sim.getObjectPose(simTarget,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)
resta_z=false

else
if (posdummy[3]<0.25) then
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud+0.1,posdummy[2],z_2})
else
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud,posdummy[2],z_2})
end
if (posdummy[2]<=0.05) then
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud+0.1,posdummy[2],z_2-0.1})
end
targetPose=sim.getObjectPose(goaldummy,-1)
currentPose=sim.getObjectPose(simTarget,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)
resta_z=true
end
suma_y=true
end
end
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud-0.3,posdummy[2],z_2})
targetPose=sim.getObjectPose(goaldummy,-1)
currentPose=sim.getObjectPose(simTarget,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)
tiempo=sim.getSimulationTime()
targetPose=sim.getObjectPose(home,-1)
currentPose=sim.getObjectPose(simTarget,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)

local img=sim.getVisionSensorImage(camera)
redCnt=0
for i=0,-1+#img//3,1 do
   local rgb={img[3*i+1],img[3*i+2],img[3*i+3]}
   if rgb[1]>0.5 and rgb[2]<0.2 and rgb[3]<0.2 then -- decide what is red
   redCnt=redCnt+1
   end
   end
```

```
print("Tiempo en pintar",tiempo)
sim.setStringSignal("Tiempo",tiempo)

print("Red",redCnt)
sim.setStringSignal("Pixel",redCnt)

 end
  sim.switchThread()
end
end
```

Finalmente, se ha definido un bucle while que se cumple siempre que la simulación está en marcha y que se subdivide en dos partes: el proceso de pintado del automóvil y la toma de fotografía y recuento de píxeles de la cámara.

Cuando se inicia la simulación, el robot espera a recibir los datos desde MATLAB y una vez los recibe, comienza a acercarse a la posición inicial desde la que realizará todo el trayecto. Esta posición inicial depende de la distancia que ha de situarse la herramienta de la puerta y cuyo valor recibe desde MATLAB.
Una vez situado el robot en la posición inicial, mediante otro while que se cumple siempre que la posición del eje Y del GoalDummy sea mayor que -0.05m, ya que, si se supera este valor, el robot ya no estaría pintando la puerta.
Este while se subdivide en tres casos condicionales que funcionan del siguiente modo:

- **Caso 1:** *suma_y==true* : Si la variable *suma_y* es igual a true, entonces el *GoalDummy* cambiará su posición en el eje Y restándole la distancia entre pasadas (*d_pasadas*), y por tanto, el robot se irá moviendo hacia la derecha en el proceso de pintado.
- **Caso 2:** *resta_z==true*: Si la variable *resta_z* es igual a true, entonces el *GoalDummy* cambiará su posición en el eje Z, siendo este valor de z_1, es decir, el robot irá de arriba abajo en este caso.
- **Caso 3:** Se cumple cuando los dos casos anteriores no se cumplen, y en este caso *GoalDummy* cambia su posición en el eje Z, siendo este de valor z_2, es decir, el robot irá de abajo a arriba.

Además, en los tres casos, se han definido dos condiciones if que funcionan del siguiente modo:

- Si la posición Z del *GoalDummy* es menor que 0.25m entonces se suma 0.1m en el eje X, de manera que la herramienta se acercará más a la puerta. Esto se debe a que la puerta tiene cierta curvatura y hay zonas que el robot ha de acercarse para poder pintar de manera correcta.
- Si la posición Y del *GoalDummy* es menor o igual que 0.05 entonces al eje Z se le resta 0.1m de manera que el robot ya no realizará el recorrido completo en el eje Y, ya que a partir de ese valor en el eje Y, la superficie de la puerta empieza a ser menor, y supondría un gasto de pintura realizar todo el trayecto para pintar una pequeña superficie.

Una vez finaliza el proceso de pintado, se aleja de la posición final de pintado y vuelve a su posición de reposo.
Tras esto, toma una fotografía de la puerta pintada y comienza el recuento de los píxeles que detecta de color rojo mediante un bucle for. Se decide que el color rojo es la combinación de los valores 0.5, 0.5 y 0.2.

Una vez finaliza el recuento, aparece en pantalla el tiempo que ha tardado el robot en pintar la puerta y los píxeles pintados y estos se envían a MATLAB.

Por otro lado, la herramienta de pintura, PaintNozzle, también dispone de su código asociado por defecto. Las únicas modificaciones que se han realizado al código son las siguientes:

```
function sysCall_init()
    sim.loadModel("./paintnozzle.ttm")
    h=sim.getObjectHandle(sim.handle_self)
    jetHandle=sim.getObjectHandle("PaintNozzleJetVolume")
    PaintNozzle=sim.getObjectHandle("PaintNozzle")
    jetAngle=30*math.pi/180
    jetRange=0.4
    drawingContMap={}
    mode=sim.drawing_discpoints
    color={1,0,0}
    itemSize=18
    bufferSize=1000000
    density=65
    clearAtEnd=true
    paintingEnabled=true
end

function cambiarboquilla(jetAngle)
        p=sim.getObjectMatrix(jetHandle,-1)
        sim.removeObject(jetHandle)

jetHandle=sim.createProximitySensor(sim.proximitysensor_ray_subtype,sim.objectspecialproperty_
detectable_all,1+512,{3,3,2,2,1,1,0,0},{0,jetRange,0,0,0,0,0,0,0,jetAngle*math.pi/180,0,0,0,0,0})

sim.setObjectProperty(jetHandle,sim.objectproperty_selectable|sim.objectproperty_selectmodelbas
einstead|sim.objectproperty_dontshowasinsidemodel)
        sim.setObjectName(jetHandle,"PaintNozzleJetVolume")
        sim.setObjectParent(jetHandle,PaintNozzle,true)
        sim.setObjectMatrix(jetHandle,-1,p)
            removeAllDrawingObjects()
end
```

Se carga el modelo de la herramienta al inicio de cada simulación mediante la función sim.loadModel() y se ha elegido un valor de *itemSize* (tamaño de los puntos de pintura) de 18 en lugar de 100, un tamaño de *bufferSize* de 1000000 para que la pintura no desaparezca en la simulación y una *density* (densidad) de 65 en lugar de 100.

La última modificación es la creación de la función *cambiarboquilla(jetAngle)* que depende del valor que recibe de MATLAB de apertura de la boquilla. Una vez recibe este valor, borra la boquilla anterior y crea una nueva boquilla con este ángulo de apertura y situado en el mismo lugar que la boquilla anterior.

CAPÍTULO 4. OPTIMIZACIÓN BAYESIANA CON MATLAB

4.1. Introducción a Matlab

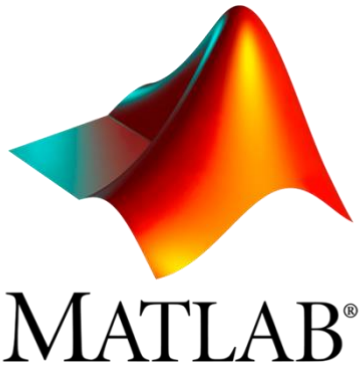

Figura 16. Logo de MATLAB. [Fuente: https://logos-marcas.com/matlab-logo/]

Matlab es un software de cálculo numérico muy versátil mediante el cual es posible realizar análisis de datos, cálculos complejos o desarrollo de algoritmos. En este caso, es posible desarrollar algoritmos de control para procesos de gran complejidad, tanto virtuales como reales.
El lenguaje de programación de Matlab es un lenguaje propio, muy similar y basado en C/C++.

4.2. Optimización bayesiana en MATLAB

4.2.1. Introducción a la optimización bayesiana

Los algoritmos bayesianos tienen su base fundamental en el teorema de Bayes, que consiste en calcular la probabilidad de que ocurra un suceso.
Es un algoritmo simple de predicción, sin embargo, en muchas ocasiones tiene una mayor empleabilidad que algoritmos más sofisticados, ya que es capaz de controlar procesos complejos.

Por tanto, la optimización bayesiana consiste en encontrar los valores adecuados de los hiperparámetros que intervienen en un proceso. Un hiperparámetro es un parámetro empleado para poder controlar el proceso de aprendizaje de una cierta máquina para realizar determinadas acciones de manera correcta.

El modo de funcionamiento de la optimización bayesiana se va a describir más detalladamente a continuación:

La optimización bayesiana basa su método de búsqueda utilizando los datos obtenidos en iteraciones anteriores, por tanto, tiene una velocidad de búsqueda mayor que otro tipo de algoritmos que basan su metodología en búsquedas aleatorias.
Podría afirmarse que la optimización bayesiana es muy similar a una búsqueda manual, ya que, por ejemplo, si se quiere optimizar un hiperparámetro que podría ser la velocidad del robot en un determinado proceso, primeramente comienza a probar una serie de valores para dicho hiperparámetro y evalúa el resultado obtenido, si este resultado se acerca al deseado, entonces es porque la optimización se está realizando en la dirección adecuada. Es por ello que los resultados de iteraciones anteriores son muy importantes para tomar decisiones futuras y por ello es un método realmente eficaz.
El funcionamiento de la optimización bayesiana se puede resumir en una serie de pasos descritos a continuación:
Primeramente, existe una función objetivo a evaluar, que, en el caso de este proyecto, es el coste del proceso que realiza el robot. Esta función objetivo es desconocida y por tanto la optimización bayesiana construye un modelo probabilístico de la función objetivo para poder evaluar los hiperparámetros que intervienen en el modelo.

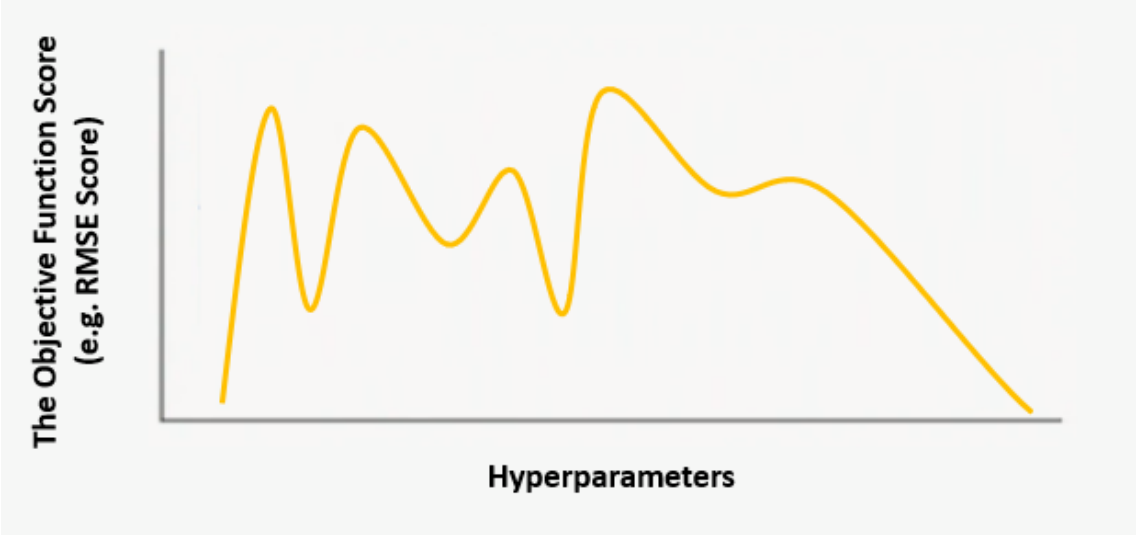

Figura 17. Función objetivo real. [Fuente: https://ichi.pro/es/concepto-de-optimizacion-bayesiana-explicado-en-terminos-simples-32073975525679]

La función objetivo verdadera es la que aparece en la Figura 17 pero esto es desconocido para la optimización bayesiana.

Entonces, está comienza a construir un modelo de superficie de respuesta, que es como una especie de modelo sustituto de la función objetivo, y se puede observar en la Figura 18 las 10 muestras que ha tomado la optimización bayesiana de la función objetivo.

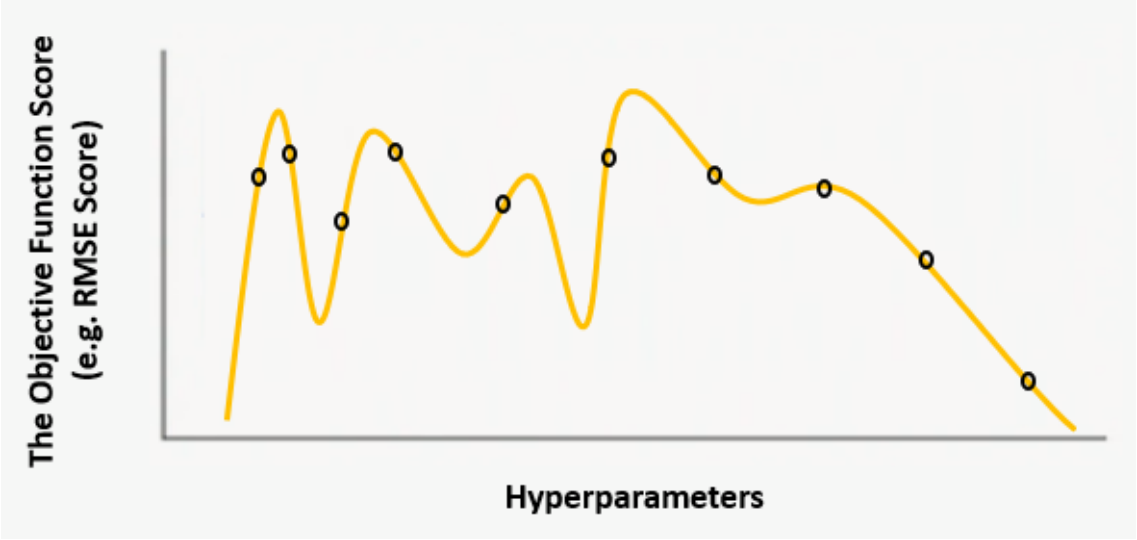

Figura 18. Puntos muestreados de la función objetivo real. [Fuente: https://ichi.pro/es/concepto-de-optimizacion-bayesiana-explicado-en-terminos-simples-32073975525679]

Una vez se obtienen las 10 muestras como se aprecia en la Figura 18, con ella se construye el modelo de superficie de respuesta para obtener una solución aproximada de la función objetivo real.

Se puede observar en la Figura 19 el modelo de superficie de respuesta representado mediante la línea azul y la región azul es la desviación del modelo real con el predecido.

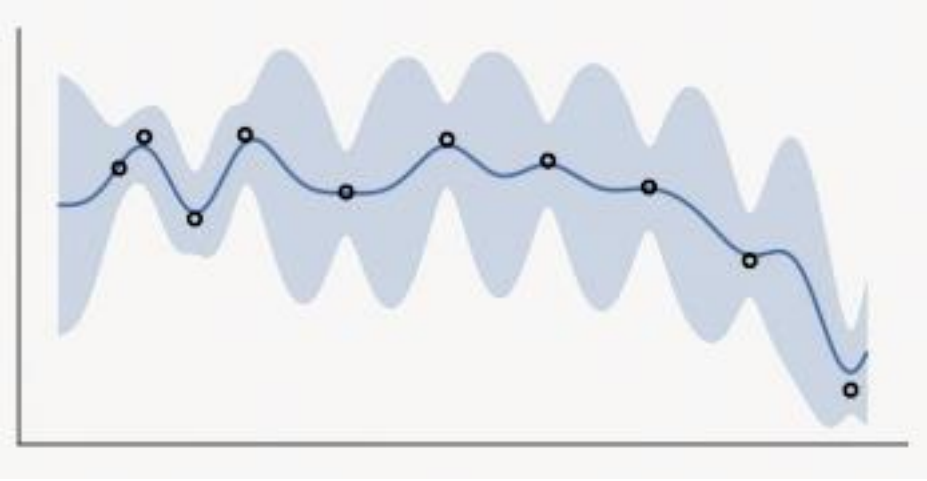

Figura 19. Función modelo de superficie de respuesta. [Fuente: https://ichi.pro/es/concepto-de-optimizacion-bayesiana-explicado-en-terminos-simples-32073975525679]

Tras obtener una aproximación de la función objetivo con 10 muestras en este caso, se ha de seguir construyendo un modelo más exacto y para esto es necesaria una función de adquisición que ayudará a la herramienta a escoger los valores adecuados de hiperparámetros.

Para ello, se ha de maximizar la función de adquisición para obtener el nuevo valor de hiperparámetro y actualizar el modelo de superficie de respuesta para aproximarlo a la función objetivo real.

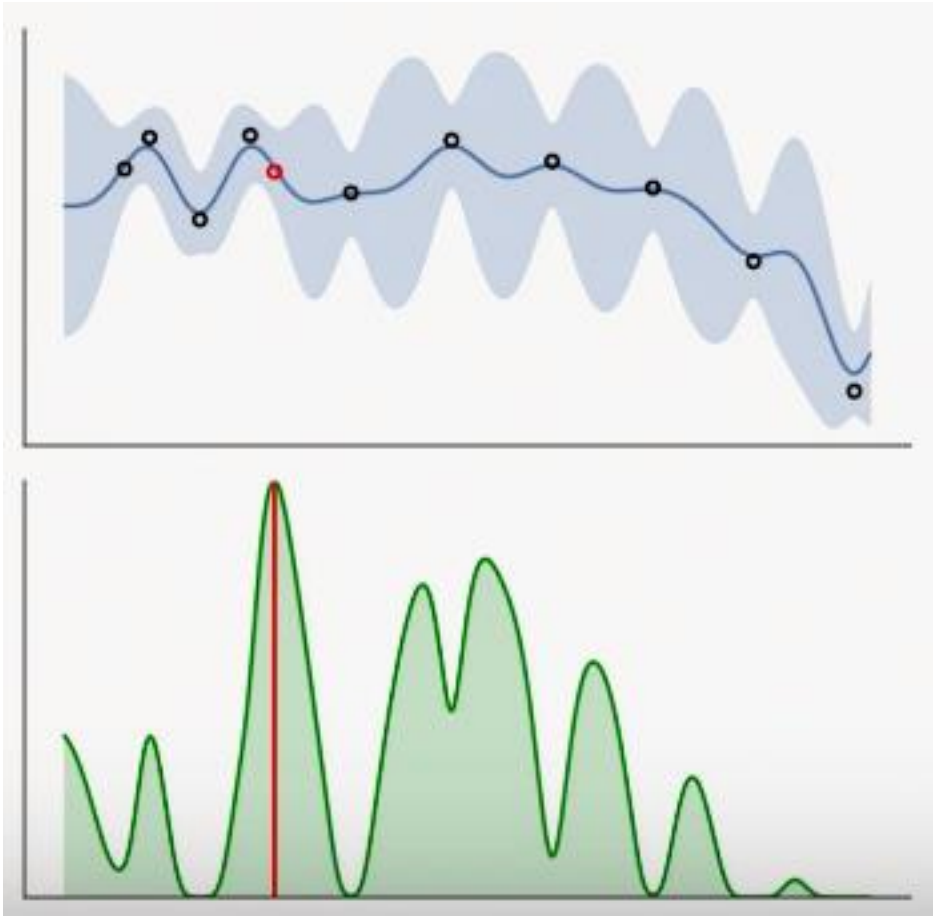

Figura 20. Función de modelo de respuesta y función de adquisición. [Fuente: https://ichi.pro/es/concepto-de-optimizacion-bayesiana-explicado-en-terminos-simples-32073975525679]

Una vez determinado el valor del hiperparámetro, se obtiene como resultado el valor de la función objetivo real para dicho hiperparámetro en ese punto en concreto.

Finalmente, si se repiten los pasos anteriores un número "n" de iteraciones se obtendrá finalmente un modelo más o menos preciso de la función objetivo real.

4.2.2. Aspectos teóricos optimización bayesiana en MATLAB

En este apartado se resumirán los aspectos teóricos de la optimización bayesiana en MATLAB según su manual de usuario, cuyo enlace está disponible en la bibliografía. Por tanto, el algoritmo de optimización bayesiana de MATLAB es una herramienta que permite minimizar el valor de una función f(x) para siendo x un hiperparámetro en un dominio acotado y el valor de x pueden ser continuos, reales, enteros, etc.

Los elementos clave de esta optimización son:

- El proceso gaussiano del modelo f(x).
- La actualización bayesiana para poder modificar el modelo en cada nueva evaluación de la función objetivo f(x).
- La función de adquisición basada en la función objetivo y la cual se maximiza para determinar el valor del siguiente punto x a evaluar.

El esquema general que sigue el algoritmo es el siguiente:

Se evalúa $y_i = f(x_i)$ para los puntos x_i, de NumSeedPoints (es una configuración de bayesopt) escogidos al azar dentro de los límites de cada hiperparámetro. La distribución de probabilidad de cada hiperparámetro es uniforme o en escala logarítmica según el valor de transformación de la optimizableVariable (variable a optimizar).
Después se repiten los siguientes pasos:

- Actualización del modelo de proceso gaussiano de f(x) para obtener una distribución posterior sobre las funciones $Q(f|x_i, y_i \text{ para } i = 1,...,t)$. A modo interno mediante fitrgp bayesopt ajusta un modelo de proceso gaussiano a los datos disponibles.
- Encontrar un nuevo punto x que maximiza la función de adquisición a(x).

Finalmente, el algoritmo se detiene si ocurre uno de los siguientes casos:

- Un número fijo de iteraciones, 30 por defecto.
- Un tiempo fijo, infinito por defecto.
- Un criterio de detención que se haya fijado con anterioridad a la simulación.

Regresión del proceso gaussiano para ajustar el modelo

El modelo probabilístico que se emplea para la función objetivo f, es un proceso gaussiano previo con ruido gaussiano agregado en las iteraciones. Por tanto, la distribución anterior en f(x) es un proceso gaussiano con media x y función de núcleo de covarianza $k(x,x';\theta)$. En este caso, θ es un vector de parámetros de kernel.

Profundizando en detalle en la regresión, si se denota un conjunto de puntos $X = x_i$ con valores de la función objetivo asociados $F = f_i$.. La distribución conjunta del mayor de los valores de la función F es una distribución normal multivariante, con media $\mu(X)$ y matriz de covarianza $K(X,X)$, donde $K_{ij} = k(x_i,x_j)$ siendo la media anterior de 0. Además, ya se ha comentado que en las iteraciones se ha añadido ruido gaussiano con varianza y, por tanto, la distribución anterior tiene covarianza.
Ajustar un modelo de regresión gaussiano a las iteraciones consiste básicamente en encontrar los valores para la varianza del ruido σ^2 y los parámetros de kernel ϑ. Dicho ajuste se logra mediante un proceso computacional intensivo realizado por fitrgp.

- Función kernel

La función del núcleo puede afectar en la calidad de un proceso de regresión gaussiano. Por ello, bayesopt utiliza el kernel ARD Matérn 5/2 que está definido en las opciones de función del kernel (covarianza).

4.2.2.1. Tipos de funciones de adquisición

Expected improvement

Evalúa el valor esperado de mejora en la función objetivo ignorando aquellos valores que causan un incremento en el objetivo. Es decir, define:

- x_{best} como la localización de la media posterior más baja
- $\mu_{Q(Xbest)}$ como el valor mínimo de la media posterior

Por tanto, la mejora esperada tiene la siguiente fórmula:

$$EI(x,Q)=E_Q[\max(0,\mu_Q(x_{\text{best}})-f(x))]. \quad (13)$$

Probability of Improvement

Realiza prácticamente el mismo cálculo que el método anterior, sin embargo, de una manera más sencilla. En ambos casos, bayesopt calcula los valores de xbest y de $\mu_{Q(Xbest)}$ pero en este caso, calcula la probabilidad de que un nuevo punto conduzca a un valor más optimo de la función objetivo modificado por un parámetro marginal "m", siendo este un ruido estándar de desviación:

$$PI(x,Q)=P_Q(f(x)<\mu_Q(x_{\text{best}})-m) \quad (14)$$

Esta probabilidad bayesopt la evalúa como:

$$PI=\Phi(v_Q(x)) \quad (15)$$

Donde:

$$v_Q = \frac{\mu_{Q(Xbest)}-M-\mu_Q(x)}{\sigma_Q(x)} \quad (16)$$

Aquí $\Phi(\cdot)$ es la unidad normal CDF y la σ_Q es la desviación estándar posterior del proceso Gaussiano en el punto evaluado, x.

Lower Confidence Bound

Esta función observa en la curva G dos desviaciones estándar por debajo de la media posterior en cada punto de evaluación

$$G(x)=\mu_Q(x)-2\sigma_Q(x) \quad (17)$$

G(x) es la envolvente de confianza más baja, $2\sigma_Q$, del modelo de la función objetivo y *bayesopt* maximiza el negativo de G:

$$LCB=2\sigma_Q(x)-\mu_Q(x) \quad (18)$$

Per second

En ocasiones el tiempo de evaluación de la función objetivo es probable que dependa de la región de estudio definida. Si es así, bayesopt puede obtener un mejor resultado empleando la ponderación de tiempo en la función de adquisición.

Las funciones de adquisición con ponderación funcionan de modo que durante las evaluaciones de la función objetivo, bayesopt mantiene otro modelo bayesiano de tiempo de evaluación de la función objetivo en función de x.

La mejora esperada por segundo que se utiliza en la función de adquisición es:

$$\text{EIpS(x)}=\frac{EI_Q(x)}{\mu_S(x)} \quad (19)$$

$\mu_S(x)$ es la media posterior de del modelo de proceso gaussiano de tiempo.

Plus

En este caso, las funciones de adquisición modifican su comportamiento si estiman que están sobreexplotando una región, evitando un mínimo de función objetivo local. Para comprender la sobreexplotación, si $\sigma_F(x)$ es la desviación estándar de la función objetivo posterior en x y σ la desviación estándar posterior del ruido aditivo de modo que:

$$\sigma_Q^2(x) = \sigma_F^2(x) + \sigma^2 \quad (20)$$

t_σ es el valor de ExplorationRatio. Las funciones de adquisición en este caso evalúan después de cada iteración si el siguiente punto x cumple:

$$\sigma_F(x) < t_\sigma \cdot \sigma \quad (21)$$

Si se cumple esta condición, entonces el algoritmo detecta que x está sobreexplotado, y la función de adquisición modifica su Función Kernel multiplicando el número de iteraciones por un valor θ. Mediante esta modificación se aumenta la varianza σ_Q para puntos entre iteraciones y se produce un nuevo punto basado en la nueva función kernel. Si se determina que el nuevo punto x también está sobreexplotado, la función de adquisición multiplica θ por un factor adicional de 10 y vuelve a intentarlo durante 5 ocasiones más para conseguir un punto no sobreexplotado.
Por tanto, *ExplorationRatio* controla la compensación de explorar nuevos puntos para una mejor solución frente a concentrar puntos cercanos que ya han sido evaluados.

Maximización de la función de adquisición

Internamente el algoritmo *bayesopt* maximiza la función de adquisición de la siguiente forma:

-Si los algoritmos son expected-improvement o probability-of-improvement , *bayesopt* estima la media más pequeña posible de la distribución posterior $\mu_{Q(Xbest)}$ muestreando miles de puntos dentro de los límites del rango del hiperparámetro, escogiendo los mejores puntos, y mejorándolos mediante la búsqueda local, para encontrar el mejor punto factible ostensible.
Factible significa que el punto satisface las restricciones y ostensible quiere decir que se perciba con facilidad.

-Para todos los algoritmos, *bayesopt* muestrea miles de puntos dentro de los límites del rango del hiperparámetro, escoge algunos de los mejores puntos factibles y los mejora mediante búsqueda local, para encontrar el mejor punto factible ostensible. Además, el valor de la función de adquisición depende de la distribución posterior modelada, no de una muestra de la función objetivo, por lo que se puede calcular de una manera rápida y eficaz.

4.2.3. Optimización bayesiana implementada en MATLAB

Para poder implementar la optimización bayesiana a cualquier proceso en MATLAB, el código genérico necesario es el siguiente:

```
var1 = optimizableVariable('x',[x1 x2]);
var2 = optimizableVariable('y',[y1 y2]);

results      =      bayesopt(@myfunct,vars,'AcquisitionFunctionName','lower-
confidence-bound','MaxObjectiveEvaluations',30);

function [Coste] = myfunct(in)
var1 = in.x;
var2 = in.y;
[var3]= functionprocess(var1,var2);
[Coste] = functionCoste(var3);
end
```

En primer lugar, se definen los hiperparámetros a optimizar, que en este caso son dos, *var1* y *var2,* cuyos valores óptimos se buscarán en un rango definido [x1 x2] y [y1 y2].

En segundo lugar, se definen las condiciones de la optimización bayesiana y se puede configurar la optimización con diversas opciones como el número de iteraciones o la función de adquisición del optimizador.

Tras esto, se define la función del proceso cuya entrada son los dos hiperparámetros y una vez se ejecuta la función del proceso, que puede ser como en este caso una simulación, o una función conocida, devuelve un parámetro o más de un parámetro, en este caso, un solo parámetro, *var3*, que se utiliza para calcular el coste que tiene la función.
Una vez la función coste devuelve el Coste, bayesopt intenta optimizar dicho coste en cada iteración.

CAPÍTULO 5. RESULTADOS

En este capítulo se van a detallar las simulaciones realizadas para implementar el control por optimización bayesiana para diferentes procesos robóticos

5.1. Optimización bayesiana en MATLAB para el robot ASTI

Previo a profundizar en el problema principal del proyecto, se ha realizado una simulación con un robot humanoide que realiza el lanzamiento de una pelota con el objetivo de que la pelota recorra una superficie con relieve y entre en un agujero situado en una mini-montaña de dicha superficie.

Todo el proceso de optimización bayesiana y la programación del robot ASTI se encuentran en el Anexo I de esta memoria.

5.2. Optimización bayesiana en MATLAB para el proceso de pintura

En los nuevo casos que a continuación se van a estudiar, se ha empleado el siguiente código para implementar el control por optimización bayesiana al proceso de pintura de una puerta de automóvil.

```
longitud = optimizableVariable('x',[0.6 0.95]);
d_pasadas = optimizableVariable('y',[0.05 0.30]);
vel = optimizableVariable('z',[0.5 1]);
id = optimizableVariable('p',[20 60],'Type','integer');
vars = [longitud,d_pasadas,vel,id];

results      =      bayesopt(@myfunct,vars,'AcquisitionFunctionName','lower-
confidence-bound','MaxObjectiveEvaluations',30);

function [Coste] = myfunct(in)
longitud = in.x;
d_pasadas = in.y;
vel=in.z;
id=in.p;
[numpixel,tiempo]= simulaCoppelia2(longitud,d_pasadas,vel,id);
[Coste] = FuncionPixel(numpixel,tiempo);
end
```

Como se puede observar en el anterior código, cuyo fichero es *bayopt2.m* hay cuatro variables como hiperparámetros a optimizar, por lo que se trata de un caso de control multivariable.

Las variables se describen a continuación:

- *longitud:* corresponde con la distancia a la que se ha de situar la herramienta de pintura de la puerta del automóvil. Se ha decidido elegir el rango [0.6-0.95]m ya que entre ese valor se encuentran valores con los que la cinemática del robot permite sin ningún inconveniente que este realice todos los movimientos pertinentes para la operación de pintado. Con valores mayores que 0.95 la cinemática del robot no consigue realizar el proceso correctamente ya que sale de su área de trabajo. Se mide en metros.
- *d_pasadas:* Es la distancia entre las pasadas que ha de realizar el robot en el proceso de pintado. El rango en este caso es entre [0.05-0.30]m ya que analizando la cinemática del robot, en estos valores puede realizar el proceso adecuadamente y además, los valores del tamaño de la boquilla, entre este rango se encuentran los valores con los que es posible pintar la puerta sin sobresalirse excesivamente la herramienta de pintura (si se sobresaliera excesivamente implicaría perder pintura). Se mide en metros.
- *vel:* Es la velocidad del robot y tras realizar diversas pruebas, se ha concluido que entre estos valores opera adecuadamente, sin ser excesivamente lento ni excesivamente rápido, y por tanto, no producir un desgaste rápido del mismo. Se mide en m/s.
- *Id:* Es el valor de la apertura que ha de tener la boquilla de la herramienta de pintura. Tras diversas pruebas se ha decidido escoger un rango de [20-60]º para la apertura de la boquilla, ya que con valores menores dificultaría la cinemática del robot para el proceso. Se mide en grados.

Por otro lado, una vez se ejecuta *simulaCoppelia2.m* con estas cuatro variables, esta devuelve dos variables, el *numpixel* que son el número de pixeles rojos que hay en la puerta del automóvil y *tiempo*, que es el tiempo que ha tardado el robot en pintar.

```
function [numpixel,tiempo]=simulaCoppelia2(longitud,d_pasadas,vel,id)
disp('Program started');
sim=remApi('remoteApi'); % using the prototype file (remoteApiProto.m)
sim.simxFinish(-1); % just in case, close all opened connections
clientID=sim.simxStart('127.0.0.1',19997,true,true,5000,5);
% start the simulation:
sim.simxStartSimulation(clientID,sim.simx_opmode_oneshot);
if (clientID>-1)
 disp('Connected to remote API server');

% Sending data to CoppeliaSim

inputInts=[id];
inputStrings=[];
inputBuffer=[];
inputFloats=[longitud,d_pasadas,vel];

pause(0.5);

[returnCode,outInts,outFloats,outStrings,outBuffer]=sim.simxCallScriptFunct
ion(clientID,'LBR4p',sim.sim_scripttype_childscript,'RecibeDatos',inputInts
,inputFloats,inputStrings,inputBuffer,sim.simx_opmode_blocking);

%Getting data from CoppeliaSim

[res1,tiempoini]=sim.simxGetStringSignal(clientID,'Tiempo',sim.simx_opmode_
streaming);
[res2,pixelini]=sim.simxGetStringSignal(clientID,'Pixel',sim.simx_opmode_st
reaming);
```

```
pause(2);

%The control loop:
while (res1~=0) %while we are connected to the server..


[res1,tiempo]=sim.simxGetStringSignal(clientID,'Tiempo',sim.simx_opmode_buf
fer);

    if(res1==sim.simx_return_ok)


       tiempo=str2double(tiempo);
    else
    end

end
sim.simxGetStringSignal(clientID,'Tiempo',sim.simx_opmode_discontinue);

while (res2~=0) %while we are connected to the server..


[res2,pixel]=sim.simxGetStringSignal(clientID,'Pixel',sim.simx_opmode_buffe
r);

    if(res2==sim.simx_return_ok)

       numpixel=str2double(pixel);
    else
    end

end

% Closing streaming conexions

sim.simxGetStringSignal(clientID,'Pixel',sim.simx_opmode_discontinue);
fprintf('N√∫mero de pixeles: %d\n',numpixel);
fprintf('Tiempo transcurrido: %f\n',tiempo);

% Now send some data to Coppelius in a non-blocking fashion:
sim.simxAddStatusbarMessage(clientID,'Hello
CoppeliaSim!',sim.simx_opmode_oneshot);

% Before closing the connection to CoppeliaSim, make sure that the last
command sent out had time to arrive. You can guarantee this with (for
example):
sim.simxGetPingTime(clientID);

% Now close the connection to CoppeliaSim:
sim.simxStopSimulation(clientID,sim.simx_opmode_blocking);

else
 disp('Failed connecting to remote API server');
end
```

```
sim.delete(); % call the destructor!
disp('Program ended');
end
```

Como se puede observar, en el anterior código primeramente se realiza la conexión con CoppeliaSim y se inicia la simulación.
Una vez iniciada la simulación, se envían los datos de los cuatro hiperparámetros de MATLAB a CoppeliaSim, y una vez el robot termina el proceso de pintado, desde CoppeliaSim se envían los datos del tiempo y del número de píxeles que se han pintado en esa simulación.

Una vez se reciben estos datos, se cierra la conexión de las transmisiones de los datos, se para la simulación y el programa devuelve los dos valores de tiempo y número de píxeles para que la FuncionPixel pueda calcular el coste del proceso.

Finalmente, en cuanto a la función coste, está se encuentra en el fichero *FuncionPixel.m* y es el siguiente código:

```
function [Coste]=FuncionPixel(numpixel,tiempo)

max_pixel=86000;
pixel_total=max_pixel-numpixel;
k=1000;
if pixel_total<3000
    Coste=tiempo;
elseif (pixel_total>3000) && (pixel_total<7000)
    Coste=k*tiempo;
else
Coste=abs(numpixel-max_pixel)*tiempo;
end
end
```

En la anterior función se ha definido la variable *max_pixel* con un valor de 86000 pixeles y esto es, el número máximo de pixeles de color rojo que pueden existir en la imagen.

La explicación de lo anterior se debe a que, es conocido que la resolución de la cámara es de 512x512 pixeles, lo que hace un total de 256.000 pixeles, pero como en la imagen, a parte de la puerta del automóvil hay un fondo negro, unos 170.000 pixeles se corresponden con los pixeles del fondo negro.

Por otra parte, se han definido la variable *pixel_total* que es la resta entre el número máximo de pixeles y el número de pixeles obtenido en la simulación y también una constante adimensional, *k*, con un valor de 1000 y a continuación se detallara su función.

Finalmente, se tiene una condición if que contempla los siguientes casos:

- **Primer caso:** Que la variable pixel_total sea menor de 3000 pixeles, lo que significa una diferencia de 3000 pixeles entre el valor máximo y el obtenido realmente. Si esto es así, entonces la función Coste es igual al tiempo de pintado y por tanto, solamente se penaliza la función de coste con el tiempo y bayesopt buscará minimizarlo en la próxima simulación.
- ***Segundo caso:*** Que la variable pixel_total sea mayor que 3000 y menor que 7000. Si se cumple esta condición, entonces la función Coste es igual al tiempo multiplicado por la constante k adimensional. Por tanto, la función coste sufre una penalización mayor y bayesopt ha de intentar en las próximas simulaciones no entrar en esta condición.

- **Tercer caso:** Si la variable pixel_total es mayor que 7000, entonces la función Coste es igual a ella misma multiplicada por el tiempo, que esto equivale al caudal de pintura expresada en pixeles por segundo. Por tanto, este es el peor caso que existe en el proceso y bayesopt ha de minimizar un valor de coste muy grande e intentar situarse en el primer o segundo caso.

Por tanto, se ha decidido evaluar 3 casos utilizando tres tipos de funciones de adquisición diferentes para observar la diferencia entre los resultados obtenidos en cada uno de los casos.

5.2.1. Simulación 1: Control bayesiano mediante lower-confidence-bound

Los resultados obtenidos en la simulación son los siguientes:

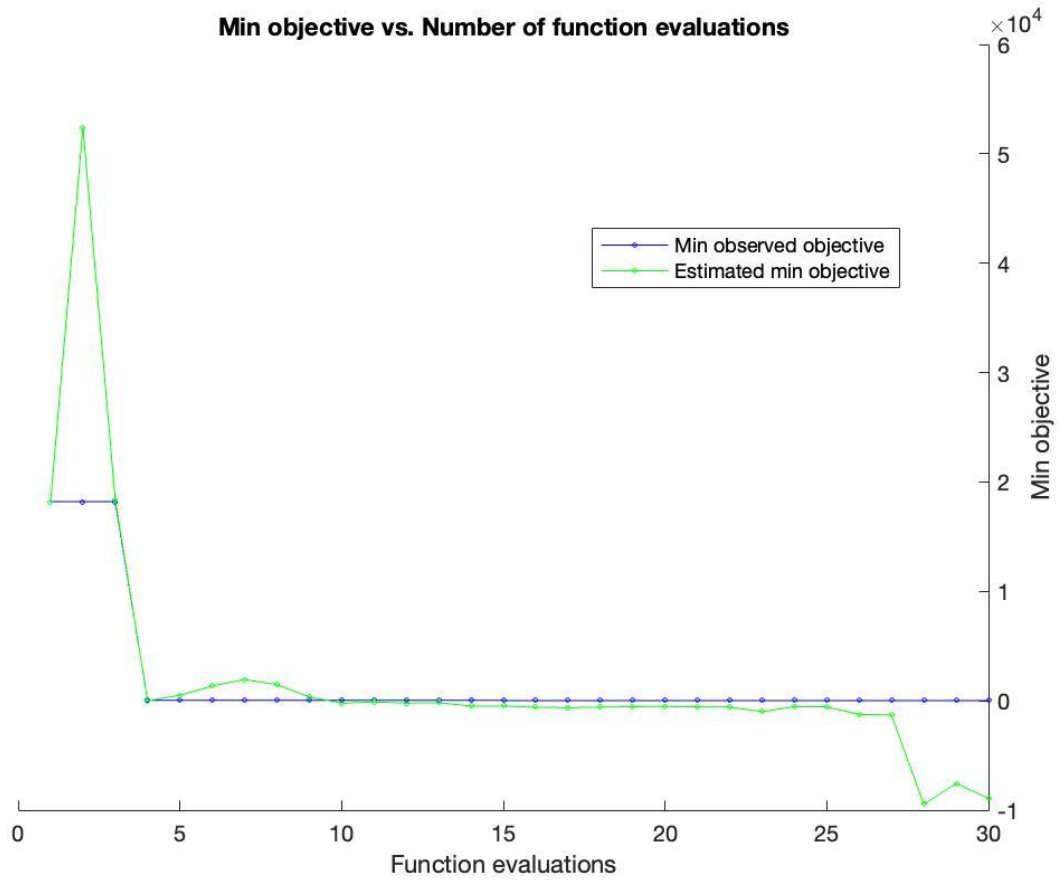

Figura 21. Mínimo función objetivo vs número de evaluaciones de la función Simulación 1

En la Figura 21 se tiene la gráfica que compara los valores estimados y observados de la función objetivo y se puede ver que excepto en 4 iteraciones, la precisión de predicción de bayesopt es muy elevada, ya que prácticamente coinciden los valores estimados con los observados.
En las últimas iteraciones se puede observar que el valor estimado alcanza valores negativos, sin embargo, como el valor mínimo de la función Coste es 0, por ello en estos casos la estimación difiere mínimamente de lo observado.

Tras estas iteraciones, la herramienta bayesopt concluye con los siguientes valor óptimos de los cuatro hiperparámetros que intervienen en el proceso:

Tabla 1. Resultados óptimos de la Simulación 1

Longitud óptima (m)	Distancia pasadas óptima (m)	Velocidad óptima (m/s)	Apertura boquilla (º)
0.83406	0.059783	0.94818	38

Por tanto, con el resultado obtenido de la Tabla 1 se observa lo siguiente:

- **Longitud óptima:** Tiene un valor de 0.834 y por tanto, se sitúa aproximadamente en un punto intermedio de su rango.
- **Distancia pasadas óptima:** Se encuentra prácticamente en el valor mínimo del rango.
- **Velocidad óptima:** Tiene un valor de 0.948, por lo que está cercano al máximo.
- **Apertura boquilla óptima:** Tiene un valor de 38º por lo que se encuentra en un punto intermedio del rango.

En la Figura 22 se observa toda la superficie de la puerta pintada correctamente con los valores óptimos anteriores.

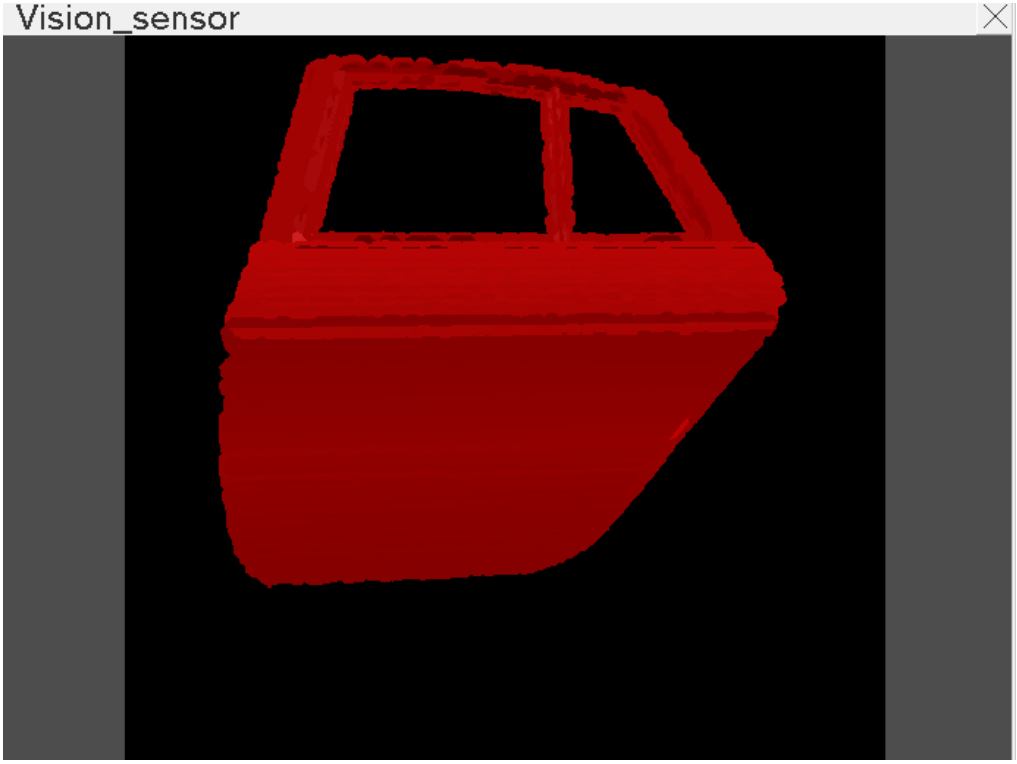

Figura 22. Resultado obtenido para la Simulación 1

Teniendo en cuenta los aspectos comentados anteriormente acerca de las variables óptimas, se puede concluir que la solución ofrecida por la optimización bayesiana en este caso no es una solución trivial ya que no se han obtenido soluciones cercanas al máximo en el valor de apertura de la boquilla, y por tanto, pintando a una mayor velocidad y menor distancia entre pasadas, se consigue una solución óptima de pintado ahorrando coste en la pintura del robot.

5.2.2. Simulación 2: Control bayesiano mediante probability-of-improvement

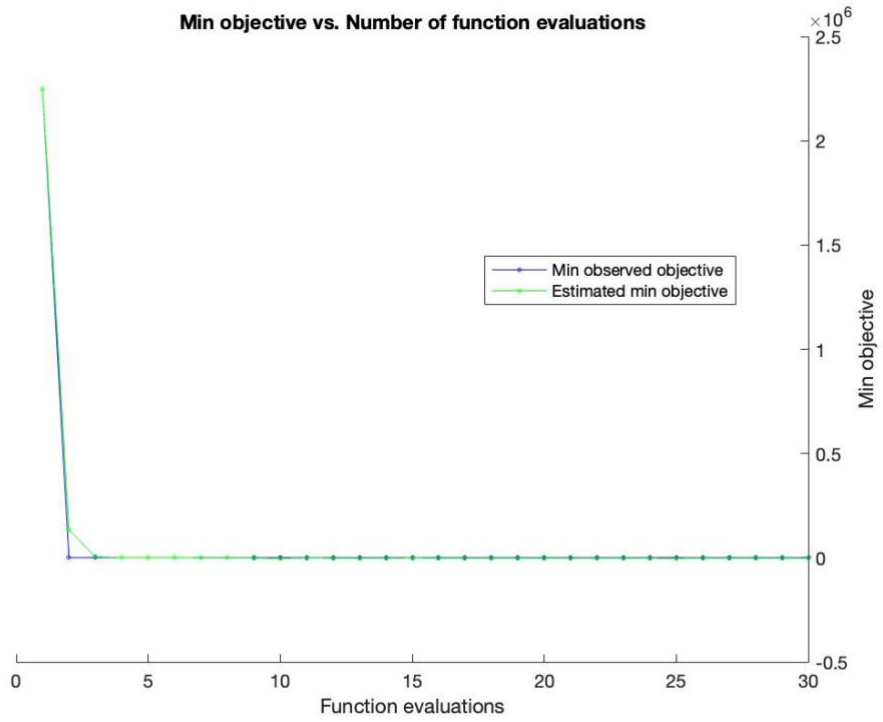

Figura 23. Mínimo función objetivo vs número de evaluaciones de la función Simulación 2

En la gráfica de la Figura 23, la precisión de bayesopt es muy elevada ya que ambos valores, estimado y observado son prácticamente iguales y desde la iteración número 2 se mantiene una línea constante en el coste y con un valor cercano al mínimo al igual que en el resto de casos.

Tabla 2. Resultados óptimos de la Simulación 2

Longitud óptima (m)	Distancia pasadas óptima (m)	Velocidad óptima (m/s)	Apertura boquilla (º)
0.88578	0.080121	0.53207	40

Tal y como se observa en la Tabla 2, los resultados obtenidos se interpretan del siguiente modo:

- **Longitud óptima:** Tiene un valor de 0.88578m y por tanto, se sitúa aproximadamente en un punto intermedio de su rango.
- **Distancia pasadas óptima:** Es un valor de 0.080121m.
- **Velocidad óptima:** Tiene un valor de 0.53207m/s lo cual es prácticamente su valor mínimo.
- **Apertura boquilla óptima:** Tiene un valor de 40º por lo que se encuentra en un punto intermedio del rango.

Se concluye que al igual que el primer caso, no es una solución trivial por el mismo motivo.

En la Figura 24 se puede ver que el robot ha sido capaz de pintar toda la superficie de la puerta con los valores óptimos que ha calculado bayesopt.

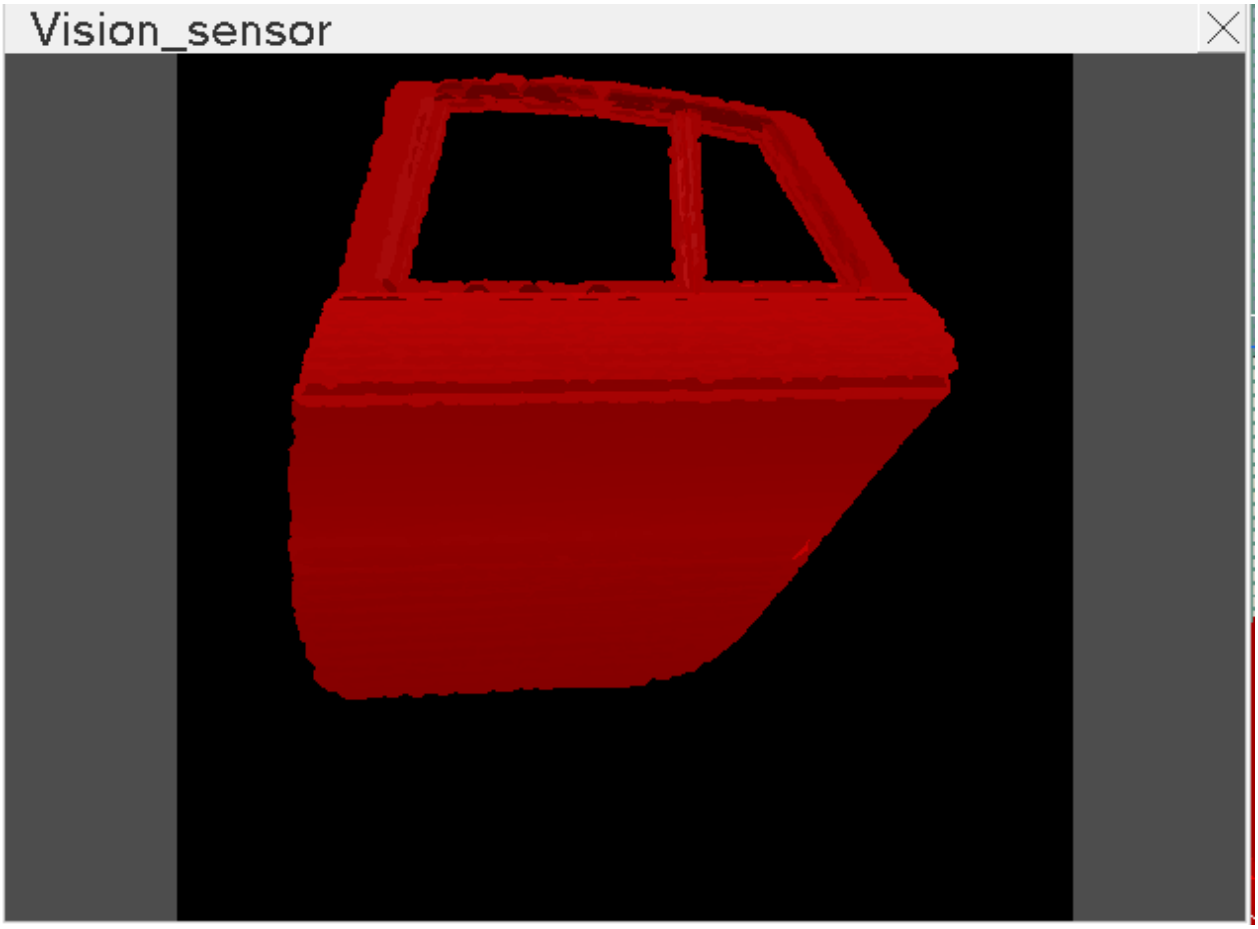

Figura 24. Resultado obtenido para la Simulación 2

5.2.3. Simulación 3: Control bayesiano mediante expected-improvement-per-second-plus

Finalmente, se va a analizar el último caso, que es la configuración por defecto que ofrece MATLAB. Para este caso, se ha cambiado el valor de la *ExplorationRatio* de 0.5 a 0.7, por tanto el ratio de exploración es mayor que en los dos casos anteriores.

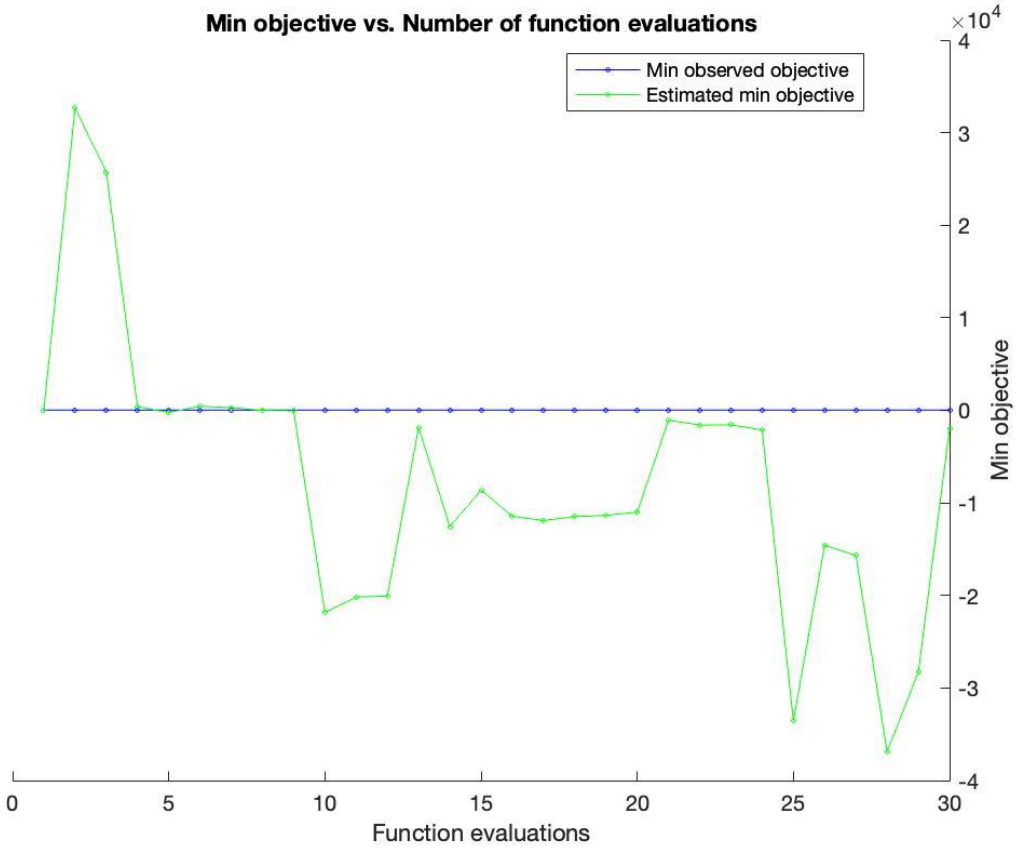

Figura 25. Mínimo función objetivo vs número de evaluaciones de la función Simulación 3

En este caso como está representado en la Figura 25 hay una gran diferencia entre los valores observados y estimados, ya que el valor observado durante toda la iteración es constante y cercano a 0, y los valores estimados se mueven en un rango de entre [3.5 y -3.5]·10^4 aproximadamente. Por tanto, *bayesopt* no ha tenido una gran precisión a la hora de realizar esta optimización.

Tabla 3. Resultados óptimos de la Simulación 3

Longitud óptima (m)	Distancia pasadas óptima (m)	Velocidad óptima (m/s)	Apertura boquilla (º)
0.9285	0.10333	0.89075	54

Se observa lo siguiente en los resultados obtenidos de la Tabla 3:

- **Longitud óptima:** Tiene un valor de 0.9285m, prácticamente en el extremo de su valor.
- **Distancia pasadas óptima:** Es un valor de 0.10333m , lo cual es aceptable.
- **Velocidad óptima:** Tiene un valor de 0.89075m/s, también aceptable
- **Apertura boquilla óptima:** Tiene un valor de 54º, lo cual es un valor muy elevado.

En la Figura 26 se observa que se ha logrado pintar toda la superficie de la puerta de una manera adecuada con los valores óptimos.

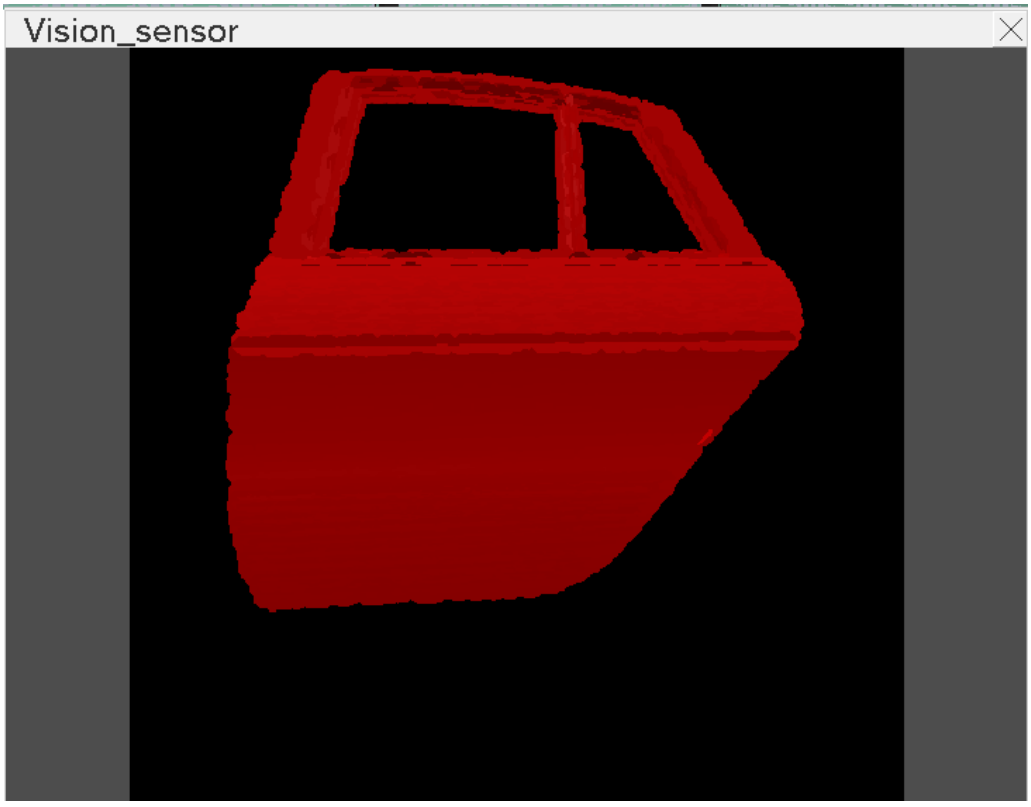

Figura 26. Resultado obtenido para la Simulación 3

5.2.4. Comparativa de los resultados obtenidos

A continuación, se va a realizar la comparativa de los resultados obtenidos en las tres simulaciones con los valores obtenidos en las simulaciones utilizando los valores extremos de los hiperparámetros del proceso.

Tabla 4. Comparativa de los resultados con los valores extremos

Longitud (m)	Distancia pasadas (m)	Velocidad (m/s)	Apertura boquilla (º)	Tiempo proceso (s)	Número pixeles
0.6	0.05	0.5	20	-	0
0.6	0.05	0.5	60	-	0
0.6	0.05	1	20	-	0
0.6	0.05	1	60	-	0
0.6	0.30	0.5	20	-	0
0.6	0.30	0.5	60	-	0
0.6	0.30	1	20	-	0
0.6	0.30	1	60	-	0
0.95	0.05	0.5	20	49.65	77062
0.95	0.05	0.5	60	49.3	85228
0.95	0.05	1	20	47.15	76562
0.95	0.05	1	60	47.15	85244
0.95	0.30	0.5	20	20.25	31848
0.95	0.30	0.5	60	20	75311
0.95	0.30	1	20	18.5	31419
0.95	0.30	1	60	18.9	73988
0.83406	0.059783	0.94818	38	42.2	85651
0.88578	0.080121	0.53207	40	36.8	85375
0.9286	0.10333	0.89075	54	30.55	85599

En primer lugar, para el primer caso se observa que los resultados obtenidos en los rangos extremos son peores para el número de píxeles que el resultado obtenido con los valores óptimos que ha calculado la optimización bayesiana, siendo este de 85651 píxeles.
Sin embargo, se puede ver que en el caso del tiempo, el menor tiempo obtenido es de 18.5 segundos y en el caso óptimo es de 42.2 segundos. No obstante, en las simulaciones que se han obtenido tiempos menores, el número de píxeles obtenidos es muy bajo, y por tanto, la función coste en estos casos tiene una penalización correspondiente al tercer caso explicado anteriormente.

Los valores que más se aproximan al óptimo son los correspondientes a las filas azules, no obstante, en ambos casos el valor del ángulo de la boquilla es el máximo, y por tanto, no son soluciones viables ya que se obtiene dicha solución a base de elevar el coste en la pintura necesaria para el proceso.

Por un lado, para el segundo caso se observa que se obtiene un resultado de 85375 píxeles en 36.8s, siendo un valor mayor que los dos casos azules y logrando este resultado en un tiempo de 10-12s menor que en dichos casos, por tanto, también es una solución válida.

Por otro lado, el tercer caso se observa que se obtiene un valor de 85599 píxeles en 30.55s. Por tanto, sí que supera a los casos azules en tiempo y píxeles y se concluye que también es una solución válida.

Finalmente, se agrupan todas las soluciones obtenidas en los tres tipos de simulación para poder decidir cuál de ellas es la solución óptima para este problema teniendo en cuenta el tamaño de la boquilla de la herramienta de pintura, de modo que se escoja una solución para ahorrar en el coste de la pintura y en el coste del tiempo del proceso.

Tabla 5. Posibles soluciones al problema

Número iteraciones	**LCB**			**POI**			**EIPSP**		
	Tamaño boquilla	**Tiempo**	**Píxeles**	**Tamaño boquilla**	**Tiempo**	**Píxeles**	**Tamaño boquilla**	**Tiempo**	**Píxeles**
30	38	42.2	85651	40	36.8	85375	54	30.55	85599

En total se disponen de 3 soluciones que se van a desglosar a continuación y elegir de cada simulación una solución candidata a ser la óptima del proceso:

Se puede observar que el mayor número de píxeles se obtiene con la simulación 1, en la que se utiliza como función de adquisición el lower-confidence-bound. No obstante, se obtiene un número grande de píxeles pero el tiempo del proceso es el mayor de los tres casos.

En cambio, el menor tiempo se obtiene con el tercer caso en el que se emplea expected-improvement-per-second-plus como función de adquisición, sin embargo, el tamaño de boquilla es cercano al máximo permitido, y por tanto, se necesitaría invertir mucho en coste de pintura, lo que a largo plazo, supondría pérdidas económicas para lograr tiempos menores.

Teniendo en cuenta de que con un tiempo de proceso alto, el robot sufrirá un desgaste conforme al tiempo ya que trabajará un número mayor de horas. Así, si se considera más importante la reducción del tiempo de proceso que el tamaño de la boquilla, lo cual quiere decir que se consigue alargar la vida útil del robot a costa de aumentar el uso de pintura, la solución más adecuada es la del segundo caso.

Así, el segundo caso en el que se ha empleado la función de adquisición de probability-of-improvement (probabilidad de mejora) se obtiene un valor de 85375 píxeles en un tiempo de 36.8s, es la solución óptima del problema.

Finalmente, si aun se requiere una precisión mayor en los resultado obtenidos y una gran mejora de los mismos, es necesario realizar simulaciones con una gran cantidad de iteraciones, ya que, cuantas más iteraciones, mayor es el campo explorado y las posibilidades de ofrecer una solución óptima aumenta. En este caso, se han realizado diversas pruebas con un número de iteraciones mayores, en un rango de 100-150 iteraciones, con una duración por simulación aproximadamente de 2h, y se han observado mejoras en los puntos óptimos, sin embargo, no han sido diferencias muy notables respecto a los puntos óptimos obtenidos con 30 iteraciones, por lo que se concluye, que para obtener una gran diferencia entre los resultados óptimos, se necesita un número muy grande de iteraciones y por tanto, se traduce en un coste computacional muy elevado para lograrlo.

CAPÍTULO 6. CONCLUSIONES

El fin de este proyecto final de máster ha sido profundizar, investigar e implementar el machine learning en un proceso tan común dentro de la industria, como es el pintado de objetos utilizando brazos robot industriales.

Se ha podido comprobar durante todo el proyecto que con una herramienta relativamente sencilla de utilizar y que no requiere ser ejecutada en ordenadores de gran capacidad, se ha podido realizar todo el proceso de simulación y control por optimización bayesiana del pintado de una puerta de automóvil. Con todo ello, se ha demostrado que un proceso de Machine Learning que aplique un autoaprendizaje del propio robot para realizar una cierta tarea de manera adecuada no es tan complejo y es muy similar a aplicar otros tipos de control existentes para procesos industriales.

Por tanto, sería interesante poder investigar y profundizar en la optimización bayesiana, de manera que pueda utilizarse a como un método de control en la industria para cualquier tipo de proceso ya que como se ha estudiado en este proyecto, posee grandes ventajas frente a otros tipos de control, destacando entre ellas su sencillez y su eficacia. Una de las grandes ventajas, a parte del ahorro en costes de proceso, es el ahorro en tiempos, recursos humanos y coste de programación del robot, ya que no es necesario programar en detalle al robot para que realice correctamente un proceso, si no que él mediante la optimización bayesiana aprende a realizar dicho proceso de manera óptima.

Sin embargo, en el caso de optimización bayesiana empleando el software MATLAB si que existe cierta limitación a la hora de realizar simulaciones y para procesos industriales relativamente sencillos y que no requieran muchos hiperparámetros como es el caso de este proyecto, es una opción valida. No obstante, si se van a realizar controles por optimización bayesiana con MATLAB de procesos con mayor complejidad y con gran cantidad de hiperparámetros, es necesario un número de iteraciones muy grande para poder obtener soluciones coherentes y óptimas.

Por tanto, se concluye que a nivel académico sí es factible realizar estudios con optimización bayesiana para procesos industriales de relativa facilidad, como es el caso de este proyecto, o a nivel industrial con procesos industriales sencillos. Sin embargo, para procesos de mayor complejidad no es posible aplicar la optimización bayesiana de MATLAB de una manera sencilla y por tanto, habría que buscar otras opciones de software o mejorar la optimización bayesiana que ofrece MATLAB para procesos industriales complejos y reales.

CAPÍTULO 7. BIBLIOGRAFÍA

Enlaces web consultados:

- https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-lua/
- https://www.cienciadedatos.net/documentos/62_optimizacion_bayesiana_hiperparametros.html
- https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/
- https://www.cs.toronto.edu/~rgrosse/courses/csc411_f18/tutorials/tut8_adams_slides.pdf
- https://www.apd.es/algoritmos-del-machine-learning/
- https://www.revelock.com/es/blog/optimizacion-de-hiperparametros
- https://ichi.pro/es/concepto-de-optimizacion-bayesiana-explicado-en-terminos-simples-32073975525679
- https://www.ciiia.mx/noticiasciiia/aprendizaje-supervisado-1
- https://www.ionos.es/digitalguide/online-marketing/marketing-para-motores-de-busqueda/unsupervised-learning/
- https://data.unimooc.com/materiales-cursos/machine-learning/Machine-Learning-5.pdf
- https://www.geeksforgeeks.org/clustering-in-machine-learning/
- http://www.cs.us.es/~fsancho/?e=77

Libros consultados:

- Bonaccorso, Giuseppe (2018). Machine Learning Algorithms
- Theodoridis, Sergios. (2015). Machine Learning: A Bayesian and Optimization perspective
- Ayodele Oladipupo, Taiwo (2016). New Advances in Machine Learning

Manuales consultados:

- CoppeliaSim: https://www.coppeliarobotics.com/helpFiles/

- MATLAB: https://www.mathworks.com/help/stats/bayesian-optimization-algorithm.html

Trabajos consultados:

- Matosevic, Antonio , 2018. *Bachelor's thesis on Bayesian optimization and its application to hyperparameter tunning*.
  Fuente: http://lnu.diva-portal.org/smash/get/diva2:1213125/FULLTEXT01.pdf

Artículos consultados:

- *Hai Wang ; Fei Hao (2012). An efficient linear regression classifier. IEEE Xplore*
  Fuente: https://ieeexplore.ieee.org/document/6224355/citations?tabFilter=papers

- *Susmita Ray (2019). A Quick Review of Machine Learning Algorithms. IEEE Xplore*
  Fuente: https://ieeexplore.ieee.org/document/8862451

PRESUPUESTOS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAPÍTULO 1. PRESUPUESTOS

1.1. Introducción

En este apartado se va a proceder a, detallar y calcular el presupuesto empleado en el proyecto. Para ello, en primer lugar se definirán todos los recursos empleados para la realización de este trabajo, que son, por un lado, recursos de mano de obra y por el otro lado, los recursos materiales utilizados.
En el presente Trabajo de Fin de Máster (TFM), se ha dedicado un tiempo aproximado de 3 meses (12 semanas) con una dedicación media de 6 horas diarias durante 6 días a la semana, por lo que el tiempo de realización del proyecto asciende a un total de 432 horas.
Finalmente, se analizarán en los próximos apartados los costes de los diferentes recursos que han intervenido en el proyecto.

1.2. Mano de obra

En este proyecto ha intervenido un tipo de recurso humano:
-Ingeniero superior Industrial Junior con un coste unitario de 60€/h

Tabla 6. Tabla coste mano de obra

Concepto	Coste unitario (€/h)
Ingeniero superior Industrial Junior	**60**

1.3. Materiales

Los materiales utilizados en la realización del proyecto se detallan en la tabla de a continuación, donde se detalla el precio, el rendimiento(o vida útil del material) y el coste unitario que suponen en el proyecto.
A continuación se van a detallar los recursos materiales empleados para la realización del proyecto, con su correspondiente precio, vida útil del material y el coste unitario en el proyecto.

- Ordenador personal (MacBook Air): Este es el recurso material principal. Se ha empleado un ordenador portátil personal para la realización de todo el proyecto, desde búsqueda de información, redacción de la memoria del proyecto hasta toda la programación de la simulación de todos los procesos que componen el mismo. Este ordenador tiene un coste de 1200€, con una vida útil estimada de 9000h. Cabe destacar que no se ha tenido en cuenta la devaluación del precio conforme la evolución del tiempo.
- MATLAB: Mediante este software se ha realizado el control mediante optimización bayesiana del proceso de pintado. El coste de su licencia es de 475€ y una amortización aproximada de 2160h.
- CoppeliaSim versión EDU: Mediante este software se ha realizado la programación de los procesos automáticos del proyecto. Esta versión es gratuita.
- Microsoft Office Professional 2016: Mediante esta herramienta informática se ha realizado la memoria y posterior presentación del proyecto. Tiene una licencia con un coste de 69€ y una amortización aproximada de 1800h.

Por tanto, para el cálculo del coste unitario se utiliza la siguiente relación:

$$\text{Coste unitario}(€/h) = \frac{Precio(€)}{Rendimiento} \quad (22)$$

Tabla 7. Tabla coste unitario materiales

Concepto	Precio(€)	Rendimiento	Coste unitario(€/h)
Ordenador personal(MacBook Air)	1200	9000	**0.133**
MATLAB	475	2160	**0.22**
CoppeliaSim versión EDU	N/A	N/A	
Microsoft Office Professional 2016	69	1800	**0.038**

1.4. Partida de mano de obra

1.4.1. Partida 1: Mano de obra

En primer lugar, la distribución del tiempo en el proyecto se ha realizado del siguiente modo:
-Realización del proyecto: El desarrollo del proyecto está compuesto por varias fases. La primera fase es la programación del proceso robótico mediante CoppeliaSim. La segunda fase, la programación del control mediante optimización bayesiana mediante MATLAB y finalmente, la fase final que es la redacción de la memoria del proyecto y la realización de la presentación del mismo. Las dos primeras fases han tenido una duración aproximada de 300h y la fase final de 132h.

Tabla 8. Partida 1: Mano de obra

Concepto	Participantes	Unidad Básica	Coste unitario(€)	Cantidad(h)	Coste total(€)
Realización del trabajo				432	25920
	Ingeniero Superior Industrial Junior	h	60	1	60
Costes indirectos				2%	518.4
TOTAL					26438.4€

1.4.2. Partida 2: Materiales

Tabla 9. Partida 2: Materiales

Concepto	Unidad Básica	Coste unitario(€)	Cantidad(h)	Coste total(€)
Ordenador personal(MacBook Air)	h	0.133	432	57.45
MATLAB	h	0.22	300	66
Microsoft Office	h	0.038	132	5.02
SUBTOTAL				128.47
Costes indirectos			2%	2.57
TOTAL				131.04 €

1.5. PRESUPUESTO DE EJECUCIÓN MATERIAL (PEM)

Es la suma de los presupuestos parciales de cada unidad de obra.

Tabla 10. Presupuesto de Ejecución Material (PEM)

Presupuestos parciales	Coste (€)
Partida 1:Mano de obra	26438.4
Partida 2: Materiales	131.04
PEM	26569.44 €

1.6. PRESUPUESTO DE EJECUCIÓN POR CONTRATA (PEC)

Para el cálculo de este valor se tienen en cuenta un 13% de gastos generales y un 6% de beneficio industrial.

Tabla 11. Presupuesto de ejecución por contrata (PEC)

Presupuesto	Coste (€)
PEM	26569.44
13% Gastos generales	3454.03
6% Beneficio industrial	1594.16
PEC	31617.63 €

1.7. PRESUPUESTO BASE DE LICITACIÓN

Finalmente, se tiene en cuenta un 21% de IVA para poder obtenerse el presupuesto base de licitación, que es el valor final del presupuesto que supone este proyecto.

Tabla 12. Presupuesto base de licitación

Presupuesto	Coste (€)
PEC	31617.63
21% IVA	6639.7
TOTAL	**38257.33€**

El coste total asciende a TREINTA Y OCHO MIL DOSCIENTOS CINCUENTA Y SIETE EUROS CON TREINTA Y TRES CÉNTIMOS.

ANEXOS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ANEXOS

I. Lanzamiento de una pelota con Robot Asti

En primer lugar, a modo de introducción y prueba del control por optimización bayesiana, se ha decidido experimentar el control de un robot humanoide lanzando una pelota a un agujero.

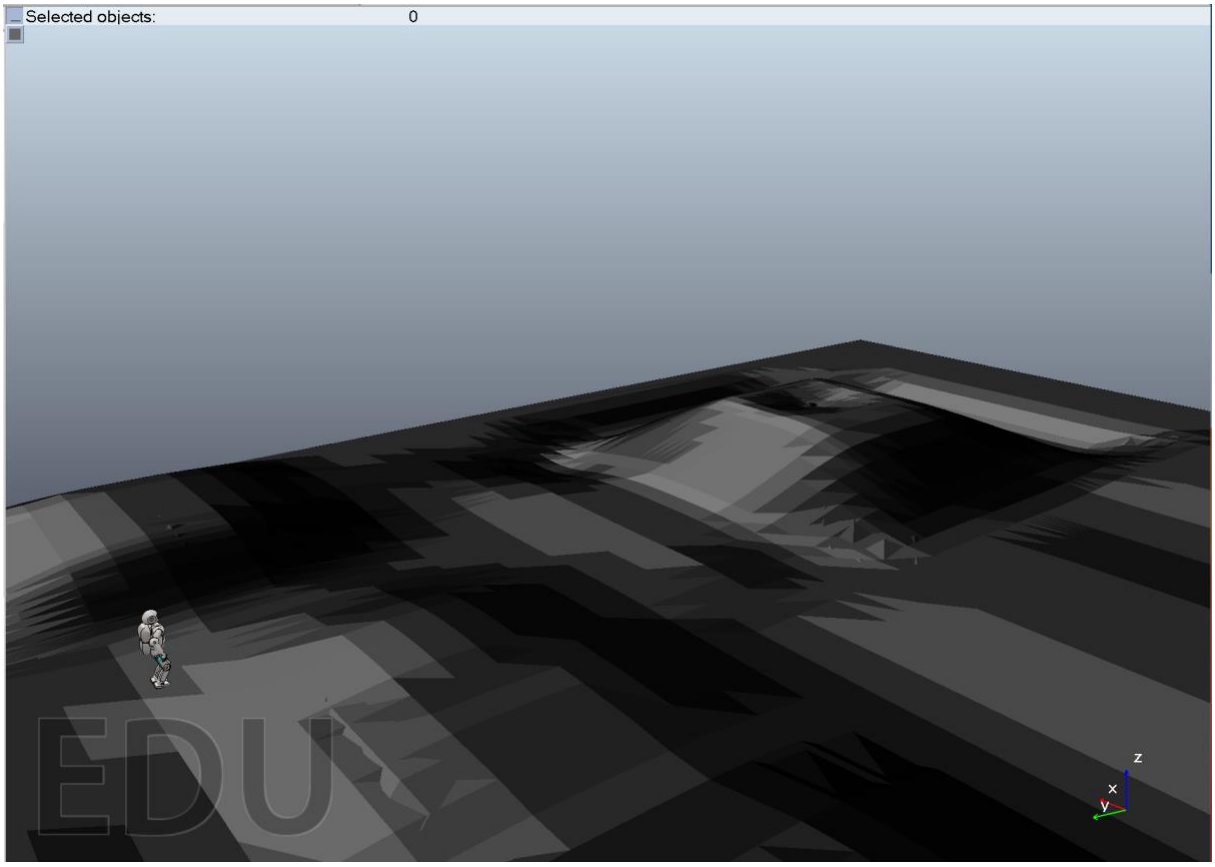

Figura 27. Entorno simulación Robot Asti

Se puede observar en la Figura 27 el entorno de simulación de este proceso. Como se puede ver, se tiene una superficie con relieves y el robot situado en un punto de dicha superficie. En el relieve más grande, como una montaña, se tiene un agujero en el que el robot ha de tirar la pelota.

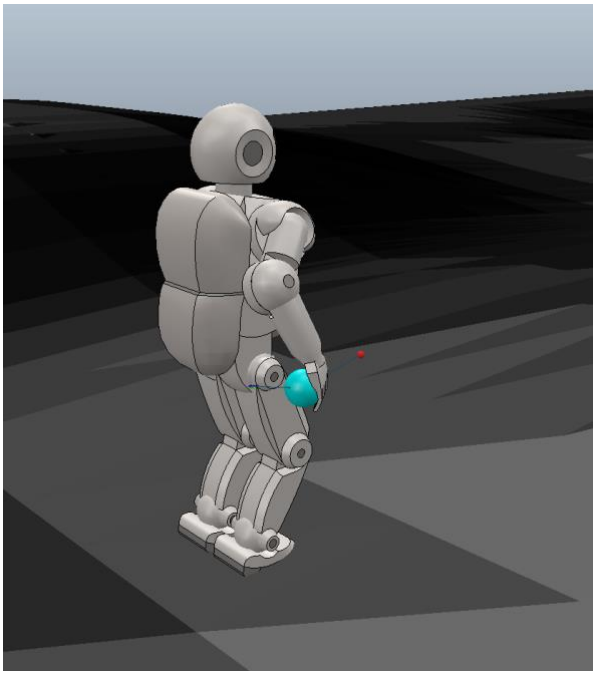

Figura 28. Robot Asti

El robot empleado se corresponde al de la Figura 28, es un robot de tipo humanoide denominado Asti.

El modelo Asti dispone de 20 articulaciones repartidas del siguiente modo:

- Cabeza: 2 articulaciones
- Brazos derecho e izquierdo: 3 articulaciones en cada brazo
- Pierna derecha e izquierda: 6 articulaciones en cada pierna

Por lo que es capaz de realizar movimientos muy similares a un ser humano.
La finalidad de realizar esta simulación ha sido poder demostrar que con control bayesiano se pueden controlar a robots humanoides en actividades cotidianas que realiza un ser humano, y por tanto, abrir una posibilidad a que se pueden programar robots humanoides que aprendan a realizar una acción de una manera correcta.

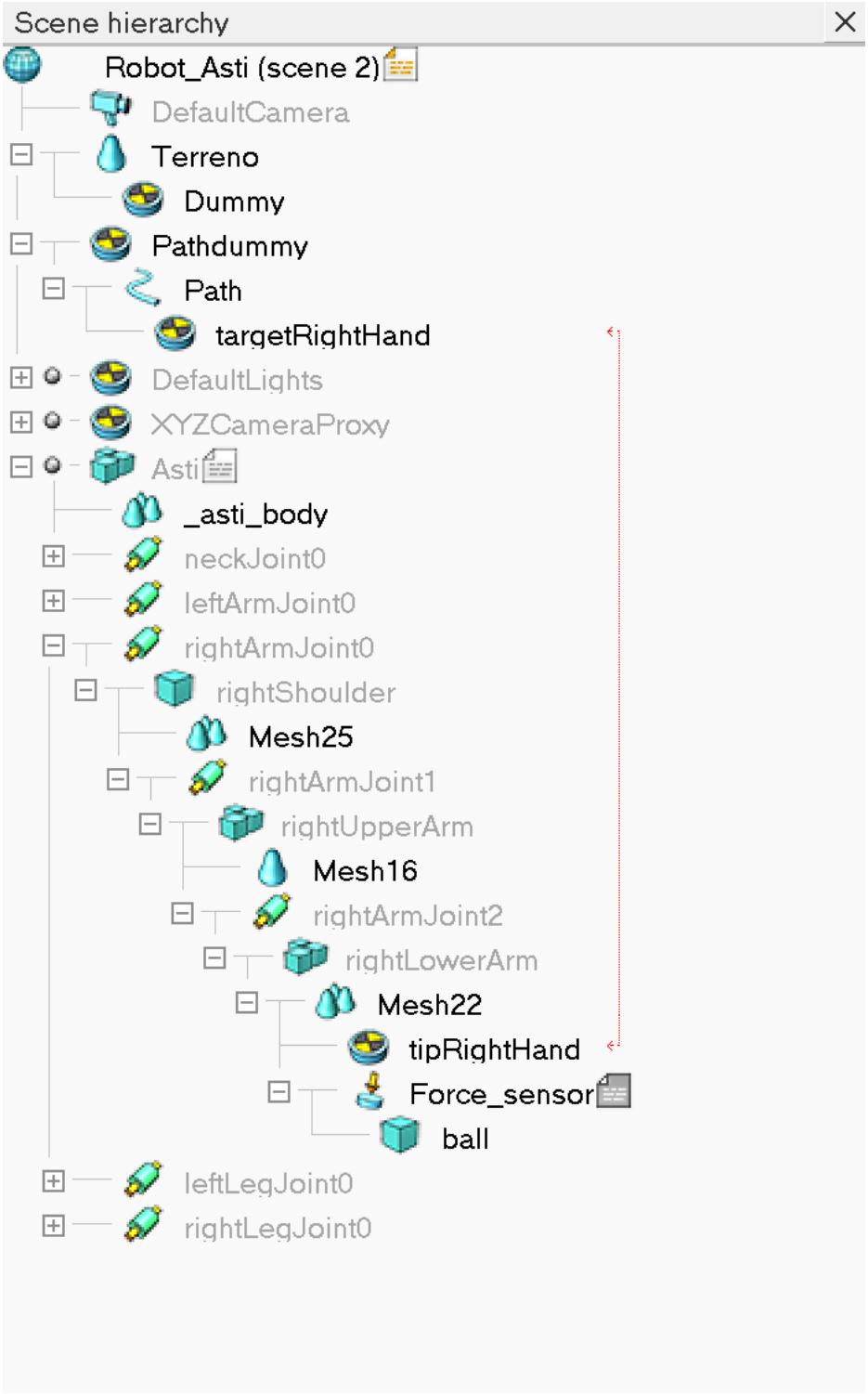

Figura 29. Jerarquía de la escena Robot Asti

En la Figura 29 está la jerarquía de la simulación. Podemos observar que el Terreno tiene un Dummy asociado a él.
Un Dummy es una especie de variable indicadora que se sitúa en el espacio o se asocia a un objeto para poder calcular distancias, programar trayectorias que ha de seguir el dummy asociado al objeto, logrando que el objeto también realice esa trayectoria.

Este Dummy se ha situado dentro del agujero para poder calcular la distancia que existe entre la pelota y el agujero y poder minimizar dicha distancia mediante la optimización bayesiana.
Por consiguiente se tiene un Path, que es un recorrido creado para simular el lanzamiento de la pelota con la mano derecha. Así pues, el Path es una trayectoria curvilínea y el dummy *targetRightHand* es el que ha de seguir la trayectoria anterior. Este dummy está asociado al dummy tipRightHand mediante un link de tipo IK (inverse kinematics), *tip-target*, esto quiere decir que el segundo dummy seguirá al primer dummy, logrando por tanto que la pelota realice el recorrido de dicha trayectoria.
Finalmente, cabe mencionar que el modelo de Asti, que tiene un código asociado, mediante el cual se ha programado toda la simulación en CoppeliaSim. Este código es de tipo non-threaded (no hilado) lo cual nos permite ejecutar el proceso a una velocidad mayor que el código de tipo threaded, por lo que implica un ahorro en el tiempo computacional.

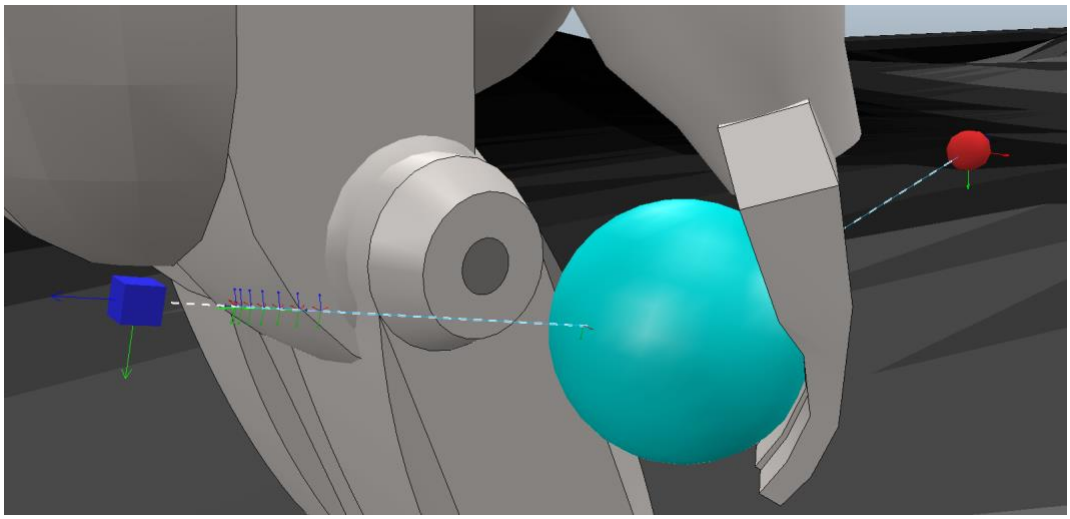

Figura 30. Trayectoria a recorrer por la bola

Para más claridad, en la Figura 30, se observa el Path, que está compuesto por dos puntos en los extremos, que definen la trayectoria curvilínea a recorrer por la pelota.

Por último, la configuración de la simulación es la siguiente:

Dynamics Engine: *Bullet V2.78* (por defecto)
dt: 10ms. En un principio, se realizaron diversas simulaciones con 50ms (por defecto), sin embargo, con un diferencial tan grande, el robot no podía alcanzar velocidades de tiro altas, por lo que se ha optado por disminuir este dt a 10ms para aumentar la capacidad del robot a unas velocidades en un rango aproximado de 0-20 m/s.
Además, para conseguir la fricción entre el terreno y la bola, se han modificado las propiedades de arrastre lineal y angular de la bola de 0 a los valores de 0.3 y 0.2 respectivamente.

La simulación se divide en dos partes: MATLAB y CoppeliaSim

La parte de MATLAB consiste en tres funciones: *bayopt.m , simulaCoppelia.m* y *FuncionBola.m*

```
vel = optimizableVariable('x',[10 16]);
ang = optimizableVariable('y',[0 8]);
vars = [vel,ang];

results = bayesopt(@myfunct,vars,'MaxObjectiveEvaluations',30);

function coste = myfunct(in)
vel = in.x;
ang = in.y;
distancia= simulaCoppelia(vel,ang);
coste = FuncionBola(distancia);
end
```

Como se puede observar, en la función *bayopt.m* se determinan las variables a optimizar, que en este caso son la velocidad y el ángulo de tiro.
Para el caso de la velocidad, se ha decidido elegir un rango entre 10-16 m/s y un rango de entre 0-8ºC para el ángulo de tiro. Los rangos se han decidido limitar a esos valores ya que se ha decidido realizar 30 iteraciones, el valor óptimo de ambos se encuentra en esos intervalos y si fueran rangos más amplios, se necesitaría un número mayor de iteraciones, lo que aumentaría el coste computacional.
Tras esto, en la función coste, se transmiten los datos de la velocidad y el ángulo de tiro a la función simulaCoppelia, que devuelve la mínima distancia entre la bola y el agujero.
La función *simulaCoppelia* cuando recibe los datos de la velocidad y el ángulo, realiza la conexión con CoppeliaSim e inicia la simulación. Una vez iniciada la simulación se transfieren los datos de las dos variables a CoppeliaSim y una vez se termina la simulación en CoppeliaSim, en esta misma función se recibe los valores de la distancia mínima entre la bola y el agujero, se finaliza la simulación en CoppeliaSim y se devuelve dicho valor a MATLAB.

Con el valor de esta distancia en la función *FuncionBola* se devuelve el coste del proceso.
La función *FuncionBola* tiene como coste la distancia, por lo que la función de la optimización bayesiana es minimizar el valor de dicho coste.

En cuanto a la programación, el código empleado en CoppeliaSim es:

```
function callme(inInts,inFloats,inStrings,inBuffer)
 dataFromExternalApp=inFloats
 print("Datos recibidos",dataFromExternalApp)
   return {},{dataFromExternalApp},{},''
end
function sysCall_init()
path=sim.getObjectHandle('Path')
ball=sim.getObjectHandle('ball')
sensor=sim.getObjectHandle('targetRightHand')
dummy=sim.getObjectHandle('Dummy')
pos_sensor=sim.getObjectPosition(sensor,-1)
pathdummy=sim.getObjectHandle('Pathdummy')
pos_ini=sim.getObjectPosition(ball,-1)
end

function sysCall_actuation()
```

```
if dataFromExternalApp then
vel=dataFromExternalApp[1]
ang=dataFromExternalApp[2]
sim.setPathTargetNominalVelocity(path,vel)
sim.setObjectOrientation(path,pathdummy,{0,0,ang*3.14/180})
simHandlePath(sim.handle_all_except_explicit,sim.getSimulationTimeStep())
pos=sim.getPathPosition(path)/sim.getPathLength(path)
if (pos==1) then
 sim.setObjectParent(ball,-1,true)
end
end
end

function sysCall_sensing()
   local threshold=smallestDistance
   if threshold==nil then
      threshold=0
   end
   local r,dist=sim.checkDistance(dummy,ball,threshold)
   if r>0 then
      smallestDistance=dist[7]
   end
   if sim.getSimulationTime()>20 and smallestDistance then
    sim.setStringSignal("Distancia",smallestDistance)
    print("La distancia es",smallestDistance)
   end
end
```

El anterior código básicamente consiste en cuatro funciones: la de conexión con Matlab, una inicialización, las acciones realizadas y una función para el cálculo y la transmisión de la distancia a MATLAB.

En primer lugar, cuando se inicia la simulación, el robot espera a recibir los datos desde MATLAB mediante la función *callme*, los recibe y se almacena en la variable dataFromExternalApp.

En la función de inicialización se realizan las inicializaciones pertinentes para poder llevar a cabo la ejecución del proceso.

Por otro lado, una vez se reciben los datos, se fija la velocidad y el ángulo de tiro y una vez que la posición, variable pos, alcanza el valor de 1, que quiere decir que la bola ya ha realizado toda la trayectoria, se desliga la pelota del sensor de fuerza implementado en la mano.

Finalmente, en la función *sysCall_sensing()* se calcula la distancia que hay entre la bola y el agujero y a los 20 segundos de simulación, se transfiere la mínima distancia mediante la variable *smallestDistance* a MATLAB a través de la función *sim.setStringSignal.*

Los resultados que se han obtenido al realizar la simulación se han desglosado en la Tabla 13:

Tabla 13. Tabla resultados Simulación 1 ASTI

Iteración	Velocidad (m/s)	Ángulo (º)	Distancia (m)
1	15.911	0.22082	6.0202
2	10.164	7.6886	5.8996
3	11.657	0.04325	3.9626
4	12.954	2.9828	3.1588
5	12.528	7.5641	2.7104
6	12.359	1.6642	3.0095
7	12.389	7.9943	3.1203
8	12.622	5.5524	2.2214
9	12.873	6.4368	4.2087
10	14.322	2.4996	3.5855
11	13.661	2.0041	1.6421
12	13.688	5.0683	0.1688
13	13.76	6.6045	0.4173
14	13.611	6.5327	0.1686
15	13.661	5.9881	0.1685
16	13.576	5.3995	0.1687
17	13.628	5.5968	0.1688
18	13.484	6.6566	0.1688
19	13.545	6.3096	0.1688
20	13.499	7.8745	0.1686
21	13.576	7.9069	0.1688
22	13.532	7.2797	2.8039
23	15.124	7.812	0.3901
24	14.977	7.931	0.4596
25	15.043	7.7417	0.4231
26	15.056	7.9757	0.3414
27	15.063	0.1035	3.0214
28	13.736	7.9016	0.6079
29	11.025	7.9898	3.8792
30	10.646	0.019281	5.7246

Como se aprecia, en las primeras iteraciones la distancia entre la bola y el agujero es muy grande, pero a medida que el control por optimización bayesiana va realizando más iteraciones y explorando más rangos de velocidad y ángulo, a partir de la iteración número 12, la distancia comienza prácticamente a ser igual a 0, (una distancia de 0.16 aproximadamente, quiere decir que la pelota está dentro del agujero y que por tanto, el robot ha acertado en su tiro).

Tras esta iteración, a rasgos generales, se observa un decrecimiento de dicha distancia.

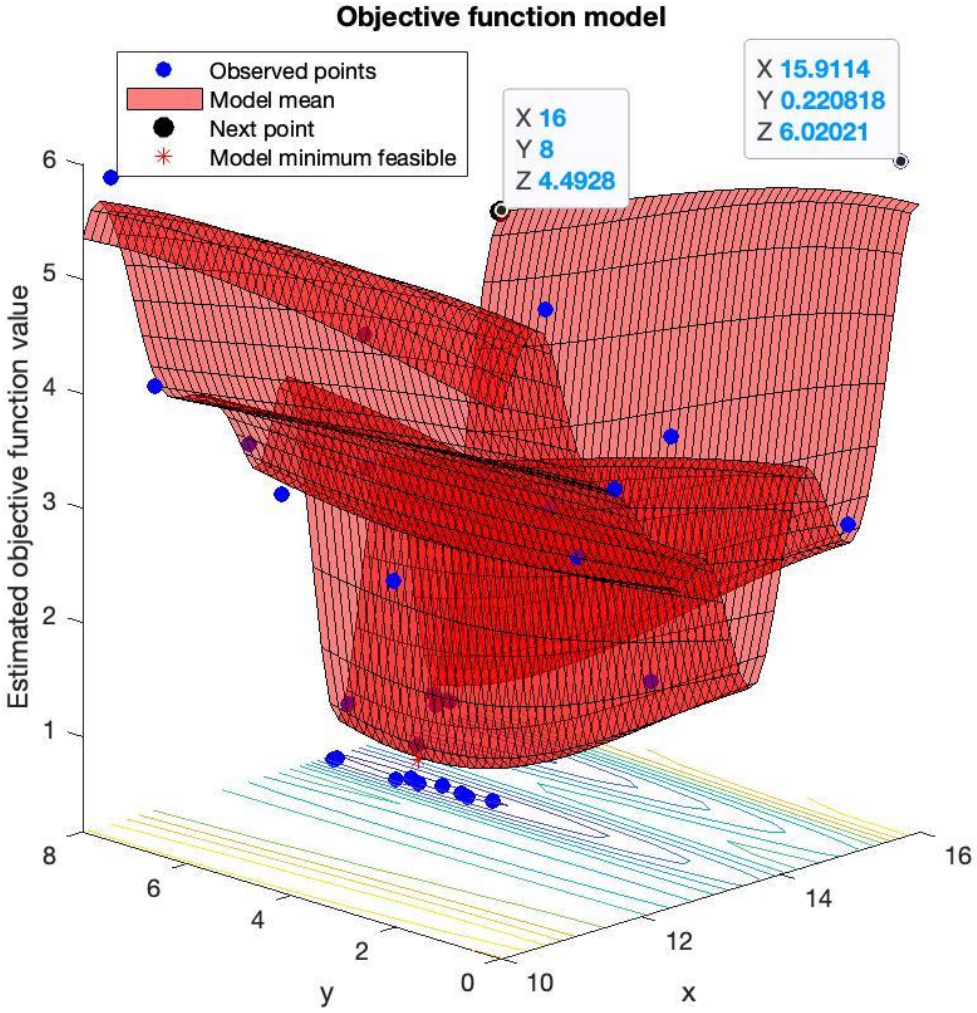

Figura 31. Gráfica del modelo de la función objetivo

En la Figura 31 se tiene el modelo de la función objetivo de esta simulación en concreto, en la que los puntos azules representan los valores de los puntos que se han analizado y la media del modelo.

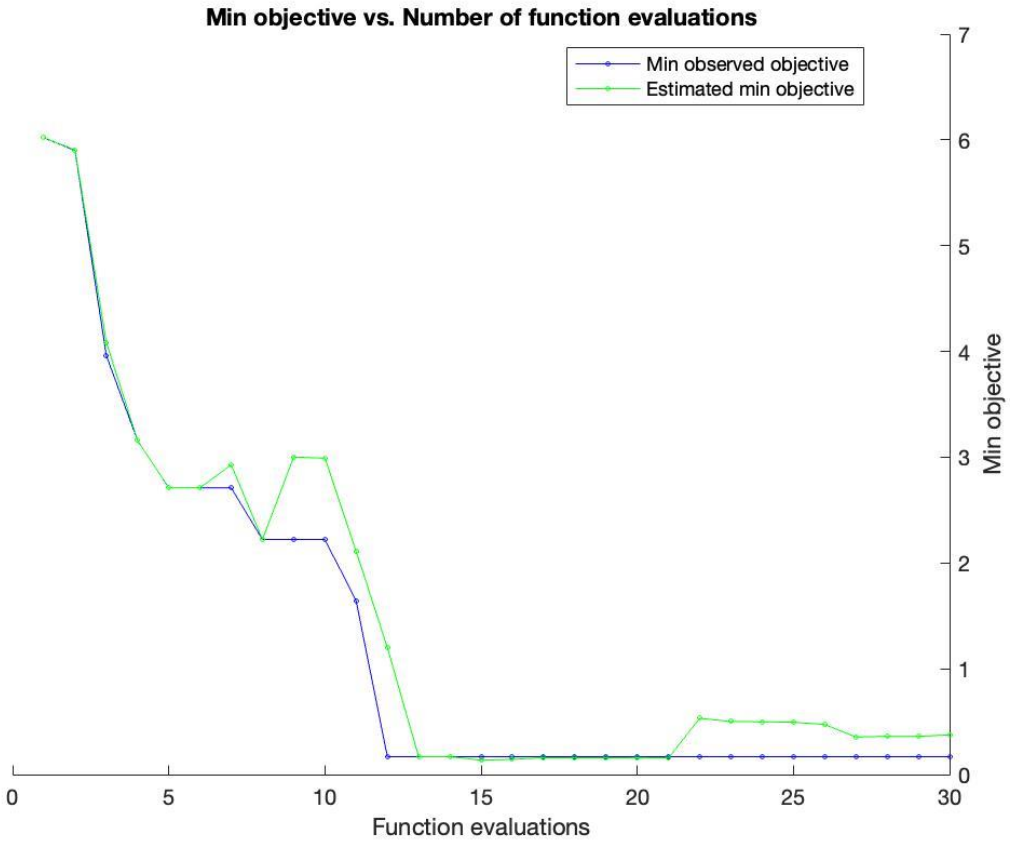

Figura 32. Gráfica de objetivo mínimo-número de evaluaciones de la función

En la Figura 32 la línea verde corresponde al valor de la distancia estimada por el control bayesiano en cada iteración y la línea azul corresponde al valor realmente observado una vez realizada la iteración. Se puede observar que en esta simulación el control bayesiano ha tenido una precisión elevada, ya que los valores estimados prácticamente son similares a los obtenidos realmente.

Tras la simulación, el valor óptimo calculado por *bayesopt,* es:

Tabla 14. Resultado óptimo Robot ASTI

Valor óptimo velocidad	13.611 m/s
Valor óptimo ángulo	6.5327 º

Para el que estima un valor de distancia de 0.37441m, sin embargo, una vez se simulan estos valores el robot acierta y la pelota entra en el agujero, y se obtiene un valor de 0.1688m de distancia.

II. Tablas de los resultados de las simulaciones del proceso de pintura

En este anexo se adjuntan las tablas con los valores obtenidos en cada una de las tres simulaciones del proceso de pintura.

Tabla 15. Valores obtenidos en cada iteración de la Simulación 1

Iteración	Longitud (m)	Distancia entre pasadas (m)	Velocidad (m/s)	Apertura boquilla (º)
1	0.83948	0.28404	0.94116	49
2	0.89084	0.24176	0.64884	26
3	0.63253	0.16108	0.6073	40
4	0.81868	0.062632	0.92052	32
5	0.76941	0.27258	0.71595	47
6	0.82787	0.24877	0.7536	57
7	0.82749	0.1404	0.99863	33
8	0.95	0.0724	0.71701	24
9	0.80357	0.10445	0.81487	55
10	0.85444	0.22171	0.60122	40
11	0.81036	0.22346	0.63169	23
12	0.94964	0.052505	0.70461	60
13	0.87215	0.05063	0.86539	60
14	0.83923	0.050743	0.64898	47
15	0.94976	0.29857	0.96107	60
16	0.88394	0.20259	0.92336	60
17	0.91378	0.05344	0.98211	48
18	0.84152	0.16799	0.99086	51
19	0.91565	0.053555	0.8154	60
20	0.95	0.062436	0.96607	47
21	0.82092	0.061518	0.7953	60
22	0.84485	0.05019	0.75357	32
23	0.8136	0.51182	0.74182	43
24	0.87193	0.29997	0.81137	60
25	0.78125	0.055353	0.65967	60
26	0.83406	0.059783	0.94818	38
27	0.94995	0.13768	0.55089	58
28	0.8696	0.25267	0.68214	52
29	0.91857	0.13499	0.85055	60
30	0.84979	0.24822	0.71521	60

Tabla 16. Valores obtenidos en cada iteración de la Simulación 2

Iteración	Longitud (m)	Distancia entre pasadas (m)	Velocidad (m/s)	Apertura boquilla (º)
1	0.6707	0.1309	0.5986	28
2	0.8969	0.0929	0.6359	50
3	0.6552	0.2611	0.8822	24
4	0.9403	0.2627	0.7472	32
5	0.8885	0.1546	0.6104	28
6	0.9051	0.1956	0.6656	29
7	0.9014	0.0505	0.5527	59
8	0.8911	0.0672	0.6825	34
9	0.9088	0.0732	0.9563	57
10	0.9495	0.0510	0.7780	43
11	0.9295	0.0502	0.6802	23
12	0.8918	0.0540	0.9857	60
13	0.9484	0.0605	0.6609	59
14	0.8769	0.0501	0.7531	42
15	0.9327	0.0509	0.9363	49
16	0.8858	0.0801	0.5321	40
17	0.9499	0.0543	0.5810	52
18	0.8982	0.0504	0.5114	41
19	0.8786	0.0558	0.9483	35
20	0.8962	0.1010	0.9802	60
21	0.9500	0.1017	0.7392	60
22	0.8858	0.0717	0.5564	54
23	0.9492	0.0735	0.5650	48
24	0.9268	0.0916	0.5059	60
25	0.8693	0.1344	0.8304	60
26	0.8698	0.2041	0.8259	60
27	0.8414	0.2993	0.6083	60
28	0.8631	0.1625	0.7793	53
29	0.8542	0.1982	0.9911	60
30	0.8706	0.1093	0.5881	47

Tabla 17. Valores obtenidos en cada iteración de la Simulación 3

Iteración	Longitud (m)	Distancia entre pasadas (m)	Velocidad (m/s)	Apertura boquilla (º)
1	0.9164	0.1265	0.7400	53
2	0.9293	0.1566	0.5464	22
3	0.8809	0.2958	0.7852	38
4	0.6847	0.1398	0.6235	35
5	0.9498	0.1627	0.8538	49
6	0.9500	0.2670	0.9548	60
7	0.9361	0.2915	0.5290	54
8	0.9430	0.0852	0.7814	60
9	0.9134	0.1600	0.9963	59
10	0.9286	0.1033	0.8908	54
11	0.9181	0.1679	0.8337	60
12	0.6008	0.2346	0.9969	60
13	0.7595	0.1625	0.9973	60
14	0.9491	0.2293	0.7450	52
15	0.6012	0.2397	0.9824	20
16	0.9499	0.2377	0.9992	41
17	0.8904	0.2937	0.9304	60
18	0.6041	0.1481	0.5039	60
19	0.7338	0.1407	0.9946	20
20	0.6032	0.1572	0.5046	20
21	0.7511	0.1582	0.5013	60
22	0.8275	0.2861	0.5217	60
23	0.8452	0.1453	0.9969	58
24	0.8547	0.2351	0.9899	55
25	0.9490	0.1789	0.9944	58
26	0.8596	0.2283	0.5626	60
27	0.8478	0.2086	0.7620	57
28	0.8599	0.1866	0.8919	60
29	0.9499	0.1395	0.5531	57
30	0.8509	0.2672	0.9949	60

III. Códigos de programación del proyecto

Los códigos de programación tanto de MATLAB como de CoppeliaSim utilizados para implementar el proceso de lanzamiento de bola y el proceso de pintado de puerta de automóvil son los siguientes:

MATLAB: *bayopt.m, bayopt2.m, FuncionBola.m, FuncionPixel.m*

- *bayopt.m*

```
vel = optimizableVariable('x',[10 16]);
ang = optimizableVariable('y',[0 8]);
vars = [vel,ang];

results = bayesopt(@myfunct,vars,'MaxObjectiveEvaluations',30);

function coste = myfunct(in)
vel = in.x;
ang = in.y;
posfinalbola= simulaCoppelia(vel,ang);
coste = FuncionBola(posfinalbola);
end
```

- *FuncionBola.m*

```
function [Coste]=FuncionBola(posfinalbola)

Coste=posfinalbola;
end
```

- *bayopt2.m*

```
longitud = optimizableVariable('x',[0.6 0.95]);
d_pasadas = optimizableVariable('y',[0.05 0.30]);
vel = optimizableVariable('z',[0.5 1]);
id = optimizableVariable('p',[20 60],'Type','integer');
vars = [longitud,d_pasadas,vel,id];

results =
bayesopt(@myfunct,vars,'ExplorationRatio',0.7,'MaxObjectiveEvaluations',30)
;

function [Coste] = myfunct(in)
longitud = in.x;
d_pasadas = in.y;
vel=in.z;
id=in.p;
[numpixel,tiempo]= simulaCoppelia2(longitud,d_pasadas,vel,id);
[Coste] = FuncionPixel(numpixel,tiempo);
end
```

- *FuncionPixel.m*

```
function [Coste]=FuncionPixel(numpixel,tiempo)

max_pixel=86000;
pixel_total=max_pixel-numpixel;
k=1000;
if pixel_total<3000
   Coste=tiempo;
elseif (pixel_total>3000) && (pixel_total<7000)
   Coste=k*tiempo;
else
   Coste=abs(numpixel-max_pixel)*tiempo;
end
end
```

CoppeliaSim: *Robot_Asti.ttt, Painting_Robot_LBR4p.ttt*

- *Robot_Asti.ttt*

```
function callme(inInts,inFloats,inStrings,inBuffer)
 dataFromExternalApp=inFloats
 print("Datos recibidos",dataFromExternalApp)

   return {},{dataFromExternalApp},{},''
end


function sysCall_init()

path=sim.getObjectHandle('Path')
ball=sim.getObjectHandle('ball')
sensor=sim.getObjectHandle('targetRightHand')
dummy=sim.getObjectHandle('Dummy')
pos_sensor=sim.getObjectPosition(sensor,-1)
pathdummy=sim.getObjectHandle('Pathdummy')
pos_ini=sim.getObjectPosition(ball,-1)

end

function sysCall_actuation()

if dataFromExternalApp then

vel=dataFromExternalApp[1]
ang=dataFromExternalApp[2]
sim.setPathTargetNominalVelocity(path,vel)
sim.setObjectOrientation(path,pathdummy,{0,0,ang*3.14/180})
simHandlePath(sim.handle_all_except_explicit,sim.getSimulationTimeStep())
pos=sim.getPathPosition(path)/sim.getPathLength(path)
```

```
if (pos==1) then
 sim.setObjectParent(ball,-1,true)
end

end
end

function sysCall_sensing()
   local threshold=smallestDistance
   if threshold==nil then
      threshold=0
   end
   local r,dist=sim.checkDistance(dummy,ball,threshold)
   if r>0 then
      smallestDistance=dist[7]
   end
   if sim.getSimulationTime()>14 and smallestDistance then
    sim.setStringSignal("Distancia",smallestDistance)
    print("La distancia es",smallestDistance)
   end

end

function sysCall_cleanup()

end
```

- *Painting_Robot_LBR4p.ttt*

Código LBR4p:

```
function RecibeDatos(inInts,inFloats,inStrings,inBuffer)
 dataFromExternalApp=inFloats
jetAngle=inInts[1]
 print("datos",dataFromExternalApp)
sim.callScriptFunction("cambiarboquilla@PaintNozzle",sim.scripttype_childscript,jetAngle)
   return {},{dataFromExternalApp,opt},{},''
end



function sysCall_init()
   corout=coroutine.create(coroutineMain)
   suma_y=false
   resta_z=true
   simTip=sim.getObjectHandle('tip')
   simTarget=sim.getObjectHandle('target')
   local simBase=sim.getObjectHandle('LBR4p')
   -- create an IK environment:
   ikEnv=simIK.createEnvironment()
```

```
   -- create an IK group:
   ikGroup_undamped=simIK.createIkGroup(ikEnv)
   -- set its resolution method to undamped:

simIK.setIkGroupCalculation(ikEnv,ikGroup_undamped,simIK.method_undamped_pseudo_inverse,0,
5)
   -- create an IK element based on the scene content:

simIK.addIkElementFromScene(ikEnv,ikGroup_undamped,simBase,simTip,simTarget,simIK.constraint
_x+simIK.constraint_y+simIK.constraint_z+simIK.constraint_alpha_beta)

   ikTip=simIK.createDummy(ikEnv)
   ikElement_undamped=simIK.addIkElement(ikEnv,ikGroup_undamped,ikTip)

simIK.setIkElementConstraints(ikEnv,ikGroup_undamped,ikElement_undamped,simIK.constraint_x+s
imIK.constraint_y+simIK.constraint_z+simIK.constraint_alpha_beta)

   -- create another IK group:
   ikGroup_damped=simIK.createIkGroup(ikEnv)
   -- set its resolution method to damped:

simIK.setIkGroupCalculation(ikEnv,ikGroup_damped,simIK.method_damped_least_squares,0.01,100)
   -- create an IK element based on the scene content:

simIK.addIkElementFromScene(ikEnv,ikGroup_damped,simBase,simTip,simTarget,simIK.constraint_x
+simIK.constraint_y+simIK.constraint_z+simIK.constraint_alpha_beta)
   ikTip=simIK.createDummy(ikEnv)
   --ikTarget=simIK.createDummy(ikEnv)
   ikElement_damped=simIK.addIkElement(ikEnv,ikGroup_damped,ikTip)

simIK.setIkElementConstraints(ikEnv,ikGroup_damped,ikElement_damped,simIK.constraint_x+simIK.
constraint_y+simIK.constraint_z+simIK.constraint_alpha_beta)

   tiempo=0
end

function sysCall_actuation()
   if coroutine.status(corout)~='dead' then
      local ok,errorMsg=coroutine.resume(corout)
      if errorMsg then
         error(debug.traceback(corout,errorMsg),2)
      end
   end

   -- Simple way:
   -- try to solve with the undamped method:
  --if simIK.applyIkEnvironmentToScene(ikEnv,ikGroup_undamped,true)==simIK.result_fail then
      -- the position/orientation could not be reached.
      -- try to solve with the damped method:
    simIK.applyIkEnvironmentToScene(ikEnv,ikGroup_damped)
```

```
      -- We display a IK failure report message:
    --   if not ikFailedReportHandle then
      --     ikFailedReportHandle=sim.displayDialog("IK failure report","IK solver failed.",
       --       sim.dlgstyle_message,false,"",nil,{1,0.7,0,0,0,0})
     --end
  -- else
      -- if ikFailedReportHandle then
           -- We close any report message about IK failure:
       --   sim.endDialog(ikFailedReportHandle)
        --  ikFailedReportHandle=nil
     -- end
  -- end
end


function coroutineMain()
  --local obj=sim.getObjectHandle('PaintNozzle')
   local camera=sim.getObjectHandle('Vision_sensor')
   local flags=-1
   local currentPose=sim.getObjectPose(sim.getObjectHandle('target'),-1)

   local maxAccel={0.7,0.7,0.7,0.7}
   local maxJerk={0.4,0.4,0.4,0.4}
   local goaldummy=sim.getObjectHandle('GoalDummy')
   local redCnt=0
   local home=sim.getObjectHandle('Home')
   z_1=0.14
   z_2=0.83

local function callback(currentPose,currentVel,currentAccel,auxData)
 sim.setObjectPose(simTarget,-1,currentPose)
end

while true do
if dataFromExternalApp then
longitud=dataFromExternalApp[1]
d_pasadas=dataFromExternalApp[2]
vel=dataFromExternalApp[3]
local maxVel={vel,vel,vel,vel}
posdummy=sim.getObjectPosition(goaldummy,-1)
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud-0.4,posdummy[2],posdummy[3]})
targetPose=sim.getObjectPose(goaldummy,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)

currentPose=sim.getObjectPose(simTarget,-1)
posdummy=sim.getObjectPosition(goaldummy,-1)
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud-0.2,posdummy[2],posdummy[3]})
targetPose=sim.getObjectPose(goaldummy,-1)
```

```
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)

currentPose=sim.getObjectPose(simTarget,-1)
posdummy=sim.getObjectPosition(goaldummy,-1)
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud,posdummy[2],posdummy[3]})
targetPose=sim.getObjectPose(goaldummy,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)
tiempo=sim.getSimulationTime()



while posdummy[2]>-0.05 do
posdummy=sim.getObjectPosition(goaldummy,-1)
if (suma_y==true) then
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud,posdummy[2]-
d_pasadas,posdummy[3]})
targetPose=sim.getObjectPose(goaldummy,-1)
currentPose=sim.getObjectPose(simTarget,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)
suma_y=false
else
if (resta_z==true) then
if (posdummy[3]<0.25) then
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud+0.1,posdummy[2],z_1})
else
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud,posdummy[2],z_1})
end
if (posdummy[2]<=0.05) then
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud+0.1,posdummy[2],z_1+0.1})
end

targetPose=sim.getObjectPose(goaldummy,-1)
currentPose=sim.getObjectPose(simTarget,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)
resta_z=false

else
if (posdummy[3]<0.25) then
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud+0.1,posdummy[2],z_2})
else
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud,posdummy[2],z_2})
end
if (posdummy[2]<=0.05) then
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud+0.1,posdummy[2],z_2-0.1})
end
targetPose=sim.getObjectPose(goaldummy,-1)
```

```
currentPose=sim.getObjectPose(simTarget,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)
resta_z=true
end
suma_y=true
end
end
cambiarPos=sim.setObjectPosition(goaldummy,-1,{longitud-0.3,posdummy[2],z_2})
targetPose=sim.getObjectPose(goaldummy,-1)
currentPose=sim.getObjectPose(simTarget,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)
tiempo=sim.getSimulationTime()
targetPose=sim.getObjectPose(home,-1)
currentPose=sim.getObjectPose(simTarget,-1)
endPose,timeLeft=sim.moveToPose(flags,currentPose,maxVel,maxAccel,maxJerk,targetPose,callback
)

local img=sim.getVisionSensorImage(camera)
redCnt=0
for i=0,-1+#img//3,1 do
   local rgb={img[3*i+1],img[3*i+2],img[3*i+3]}
   if rgb[1]>0.5 and rgb[2]<0.2 and rgb[3]<0.2 then -- decide what is red
   redCnt=redCnt+1
   end
   end

print("Tiempo en pintar",tiempo)
sim.setStringSignal("Tiempo",tiempo)

print("Red",redCnt)
sim.setStringSignal("Pixel",redCnt)

 end
  sim.switchThread()
end
end
```

Código PaintNozzle:

```
function sysCall_init()
   sim.loadModel("./paintnozzle.ttm")
   h=sim.getObjectHandle(sim.handle_self)
   jetHandle=sim.getObjectHandle("PaintNozzleJetVolume")
   PaintNozzle=sim.getObjectHandle("PaintNozzle")
   jetAngle=30*math.pi/180
   jetRange=0.4
   drawingContMap={}
   mode=sim.drawing_discpoints
```

```
   color={1,0,0}
   itemSize=18
   bufferSize=1000000
   density=65
   clearAtEnd=true
   paintingEnabled=true
end

function cambiarboquilla(jetAngle)
      p=sim.getObjectMatrix(jetHandle,-1)
      sim.removeObject(jetHandle)

jetHandle=sim.createProximitySensor(sim.proximitysensor_ray_subtype,sim.objectspecialproperty_
detectable_all,1+512,{3,3,2,2,1,1,0,0},{0,jetRange,0,0,0,0,0,0,0,jetAngle*math.pi/180,0,0,0,0,0})

sim.setObjectProperty(jetHandle,sim.objectproperty_selectable|sim.objectproperty_selectmodelbas
einstead|sim.objectproperty_dontshowasinsidemodel)
      sim.setObjectName(jetHandle,"PaintNozzleJetVolume")
      sim.setObjectParent(jetHandle,PaintNozzle,true)
      sim.setObjectMatrix(jetHandle,-1,p)
         removeAllDrawingObjects()
end



function showDlg()
   if not ui then
      local pos='position="-50,50" placement="relative"'
      if uiPos then
         pos='position="'..uiPos[1]..','..uiPos[2]..'" placement="absolute"'
      end
      local xml ='<ui title="xxxx" closeable="false" activate="false" '..pos..[[>
         <button text="Clear all paint" on-click="clear_callback"/>
         <checkbox text="Enabled" on-change="enabled_callback" id="32" />
         <checkbox text="Clear at simulation end" on-change="clearAtEnd_callback" id="31" />
         <checkbox text="Show jet volume" on-change="volume_callback" id="33" />
         <label text="Paint item" style="* {font-weight: bold;}"/>
         <group layout="vbox" flat="true" id="55">
            <radiobutton text="points" on-click="item_callback" id="1" />
            <radiobutton text="lines" on-click="item_callback" id="2" />
            <radiobutton text="triangle patch" on-click="item_callback" id="3" />
            <radiobutton text="quad patch" on-click="item_callback" id="4" />
            <radiobutton text="disc patch" on-click="item_callback" id="5" />
            <radiobutton text="cube" on-click="item_callback" id="6" />
            <radiobutton text="sphere" on-click="item_callback" id="7" />
         </group>

         <label text="Paint color" style="* {font-weight: bold;}"/>
         <group layout="form" flat="true">
            <label text="red"/>
            <hslider id="11" on-change="color_callback" minimum="0" maximum="100"/>
```

```
            <label text="green"/>
            <hslider id="12" on-change="color_callback" minimum="0" maximum="100"/>
            <label text="blue"/>
            <hslider id="13" on-change="color_callback" minimum="0" maximum="100"/>
        </group>

        <label text="Other" style="* {font-weight: bold;}"/>
        <group layout="form" flat="true">
            <label text="jet angle"/>
            <hslider id="24" on-change="other_callback" minimum="0" maximum="100"/>
            <label text="jet range"/>
            <hslider id="25" on-change="other_callback" minimum="0" maximum="100"/>
            <label text="paint item size"/>
            <hslider id="21" on-change="other_callback" minimum="0" maximum="100"/>
            <label text="buffer size"/>
            <hslider id="22" on-change="other_callback" minimum="0" maximum="100"/>
            <label text="paint density"/>
            <hslider id="23" on-change="other_callback" minimum="0" maximum="100"/>
        </group>

    </ui>]]
    ui=simUI.create(xml)
    simUI.setTitle(ui,sim.getObjectName(h))
    updateUI()
  end
end

function updateUI()
  simUI.setEnabled(ui,24,sim.getSimulationState()==sim.simulation_stopped)
  simUI.setEnabled(ui,25,sim.getSimulationState()==sim.simulation_stopped)
  simUI.setCheckboxValue(ui,31,not clearAtEnd and 0 or 2)
  simUI.setCheckboxValue(ui,32,not paintingEnabled and 0 or 2)
  local v=sim.getObjectInt32Param(jetHandle,sim.objintparam_visibility_layer)
  simUI.setCheckboxValue(ui,33,v==0 and 0 or 2)

  simUI.setRadiobuttonValue(ui,1,(mode==sim.drawing_points and 1 or 0))
  simUI.setRadiobuttonValue(ui,2,(mode==sim.drawing_lines and 1 or 0))
  simUI.setRadiobuttonValue(ui,3,(mode==sim.drawing_trianglepoints and 1 or 0))
  simUI.setRadiobuttonValue(ui,4,(mode==sim.drawing_quadpoints and 1 or 0))
  simUI.setRadiobuttonValue(ui,5,(mode==sim.drawing_discpoints and 1 or 0))
  simUI.setRadiobuttonValue(ui,6,(mode==sim.drawing_cubepoints and 1 or 0))
  simUI.setRadiobuttonValue(ui,7,(mode==sim.drawing_spherepoints and 1 or 0))

  simUI.setSliderValue(ui,11,color[1]*100)
  simUI.setSliderValue(ui,12,color[2]*100)
  simUI.setSliderValue(ui,13,color[3]*100)

  simUI.setSliderValue(ui,21,itemSize*5)
  simUI.setSliderValue(ui,22,(bufferSize/100)-1)
  simUI.setSliderValue(ui,23,density)
```

```
    simUI.setSliderValue(ui,24,100*(-5+jetAngle*180/math.pi)/85)
    simUI.setSliderValue(ui,25,100*(-0.1+jetRange)/0.9)
end

function item_callback(ui,id,v)
    if (id==1) then mode=sim.drawing_points end
    if (id==2) then mode=sim.drawing_lines end
    if (id==3) then mode=sim.drawing_trianglepoints end
    if (id==4) then mode=sim.drawing_quadpoints end
    if (id==5) then mode=sim.drawing_discpoints end
    if (id==6) then mode=sim.drawing_cubepoints end
    if (id==7) then mode=sim.drawing_spherepoints end
    removeAllDrawingObjects()
end

function color_callback(ui,id,v)
    color[id-10]=v/100
end

function other_callback(ui,id,v)
    if id==21 then
        itemSize=v/5
    end
    if id==22 then
        bufferSize=1+v*100
    end
    if id==23 then
        density=v
    end
    if id==24 then
        jetAngle=(5+85*v/100)*math.pi/180
    end
    if id==25 then
        jetRange=0.1+0.9*v/100
    end
    if id==24 or id==25 then
        local n=sim.getObjectName(jetHandle)
        local p=sim.getObjectParent(jetHandle)
        local m=sim.getObjectMatrix(jetHandle,-1)
        sim.removeObject(jetHandle)

jetHandle=sim.createProximitySensor(sim.proximitysensor_ray_subtype,sim.objectspecialproperty_
detectable_all,1+512,{3,3,2,2,1,1,0,0},{0,jetRange,0,0,0,0,0,0,0,jetAngle,0,0,0,0,0})

sim.setObjectProperty(jetHandle,sim.objectproperty_selectable|sim.objectproperty_selectmodelbas
einstead|sim.objectproperty_dontshowasinsidemodel)
        sim.setObjectName(jetHandle,n)
        sim.setObjectParent(jetHandle,p,true)
        sim.setObjectMatrix(jetHandle,-1,m)
    end
```

```
    removeAllDrawingObjects()
end

function clear_callback(ui,id,v)
    removeAllDrawingObjects()
end

function clearAtEnd_callback(ui,id,v)
    clearAtEnd=not clearAtEnd
end

function enabled_callback(ui,id,v)
    paintingEnabled=not paintingEnabled
end

function volume_callback(ui,id,v)
    local v=sim.getObjectInt32Param(jetHandle,sim.objintparam_visibility_layer)
    if v==0 then
        v=65535
    else
        v=0
    end
    sim.setObjectInt32Param(jetHandle,sim.objintparam_visibility_layer,v)
end

function hideDlg()
    if ui then
        uiPos={}
        uiPos[1],uiPos[2]=simUI.getPosition(ui)
        simUI.destroy(ui)
        ui=nil
    end
end

function sysCall_nonSimulation()
    showOrHideDlg()
end

function showOrHideDlg()
    local s=sim.getObjectSelection()
    local show= (s and #s==1 and s[1]==h)
    if show then
        showDlg()
    else
        hideDlg()
    end
end

function sysCall_beforeSimulation()
    if ui then
```

```
        simUI.setEnabled(ui,24,false)
        simUI.setEnabled(ui,25,false)
    end
    removeAllDrawingObjects()
end

function sysCall_afterSimulation()
    if clearAtEnd then
        removeAllDrawingObjects()
    end
    if ui then
        simUI.setEnabled(ui,24,true)
        simUI.setEnabled(ui,25,true)
    end
end

function sysCall_cleanup()
    removeAllDrawingObjects()
    mode=0
    itemSize=0
    hideDlg()
end

function sysCall_beforeInstanceSwitch()
    hideDlg()
end

function getDrawingObject(objectHandle)
    local retVal=drawingContMap[objectHandle]
    if retVal==nil then
        local
dr=mode+sim.drawing_cyclic+sim.drawing_painttag+sim.drawing_followparentvisibility+sim.drawing
_itemcolors
        local s=itemSize
        if (mode~=sim.drawing_points)and(mode~=sim.drawing_lines) then
            s=s/1000
        end
        retVal=sim.addDrawingObject(dr,s,0,objectHandle,bufferSize,{0,0,0})
        drawingContMap[objectHandle]=retVal
    end
    return retVal
end

function removeAllDrawingObjects()
    for key,val in pairs(drawingContMap) do
        sim.removeDrawingObject(val|sim.handleflag_silenterror)
    end
    drawingContMap={}
end
```

```
function sysCall_sensing()

   showOrHideDlg()
   if paintingEnabled then

      local ptCount=0
      local m=sim.getObjectMatrix(jetHandle,-1)
      local rot=sim.copyTable(m)
      rot[4]=0
      rot[8]=0
      rot[12]=0
      while ptCount<density do
         ptCount=ptCount+1
         local res,dist,pt,objH,n=sim.handleProximitySensor(jetHandle)
         if (res==1) then
            if
(mode==sim.drawing_trianglepoints)or(mode==sim.drawing_quadpoints)or(mode==sim.drawing_dis
cpoints) then
               -- We add a small random offset from the surface in order to nicely "mix" the paint
               local rn=0.0002+0.0003*math.random()
               pt[1]=pt[1]+n[1]*rn
               pt[2]=pt[2]+n[2]*rn
               pt[3]=pt[3]+n[3]*rn
            end
            pt=sim.multiplyVector(m,pt)
            n=sim.multiplyVector(rot,n)
            local drawingObjHandle=getDrawingObject(objH)
            if (mode==sim.drawing_lines) then
               table.insert(pt,pt[1]+n[1]*0.01)
               table.insert(pt,pt[2]+n[2]*0.01)
               table.insert(pt,pt[3]+n[3]*0.01)
            end
            if
(mode~=sim.drawing_points)and(mode~=sim.drawing_lines)and(mode~=sim.drawing_spherepoints)
then
               table.insert(pt,n[1])
               table.insert(pt,n[2])
               table.insert(pt,n[3])
            end
            table.insert(pt,color[1])
            table.insert(pt,color[2])
            table.insert(pt,color[3])
            sim.addDrawingObjectItem(drawingObjHandle,pt)
         end
      end
      sim.resetProximitySensor(jetHandle)
   end
end
showDlg=function() end
updateUI=function() end
```