Grado en Ingeniería Electrónica Industrial

TRABAJO FIN DE GRADO

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

"Simulación de un sistema de clasificación robotizado de propósito general utilizando técnicas de Deep-Learning y visión artificial en Python"

2020/2021

Alumno/a:

Georgiy Kalmutskyy Kalmutskyy

Director/es:

María del Mar Castilla Nieto

José Carlos Moreno Úbeda

UNIVERSIDAD DE ALMERÍA

ESCUELA SUPERIOR DE INGENIERÍA

GRADO EN INGENIERIA ELECTRONICA INDUSTRIAL

TRABAJO FIN DE GRADO

Simulación de un sistema de clasificación robotizado de propósito general utilizando técnicas de Deep-Learning y visión artificial en Python

---

Alumno: Georgiy Kalmutskyy Kalmutskyy

Director: María del Mar Castilla Nieto

Codirector: José Carlos Moreno Úbeda

Fecha: Enero 2021

Georgiy Kalmutskyy Kalmutskyy	*María del Mar Castilla Nieto*	*José Carlos Moreno Úbeda*

Índice General

Agradecimientos

A los tutores del TFG María del Mar Castilla Nieto y José Carlos Moreno Úbeda. Por el apoyo y la atención que me han prestado.

Al departamento de ARM de la Universidad de Almería. Por demostrar una gran pasión en enseñanza.

Y a todos los compañeros que han pasado esta etapa conmigo.

Acrónimos

Acrónimo	Significado
API	Interfaz de programación de aplicaciones (del inglés *Application Programming Interface*)
GRU	Unidad recurrente cerrada (del inglés gated recurrent units)
GUI	Interfaz gráfica de usuario (del inglés Graphical User Interface)
IA	Inteligencia Artificial
LSTM	Memoria a largo – corto plazo (del inglés Long Short-Term Memory)
RELU	Unidad lineal rectificadora (del inglés REctified Linear Unit)

Índice de figuras

Índice de tablas

Resumen

En la actualidad, la inteligencia artificial es el pilar fundamental de la denominada cuarta revolución industrial. Por ello, este trabajo de fin de grado busca aplicar los algoritmos del aprendizaje profundo y visión artificial sobre un sistema de clasificación robotizado de propósito general.

El objetivo de este trabajo de fin de grado es el estudio y diseño de redes neuronales artificiales para su posible implementación en la industria. Se realiza un estudio bibliográfico con el fin de comprender el funcionamiento de este tipo de algoritmos y obtener la capacidad de diseñar modelos propios para diferentes fines.

Para la simulación del sistema de clasificación robotizado se ha hecho uso del simulador CoppeliaSim. En dicho entorno se generan los datos de entrenamiento y se realizan las validaciones de los modelos creados. También, se ha utilizado TensorFlow y Keras programando en el lenguaje de programación Python para el diseño de las redes. Se han propuesto dos problemas: i) la identificación objetos en un almacén automatizado; y ii) el segundo la obtención de la localización de un objeto, utilizando cámaras virtuales en ambas situaciones. Por último, se analiza los resultados obtenidos de las redes y se comprueba su viabilidad en la industria moderna.

Palabras clave: red neuronal artificial, visión artificial, robótica, simulación, sistema de clasificación robotizada

Abstract

Today, artificial intelligence is the cornerstone of the so-called fourth industrial revolution. For this reason, this end-of-degree project seeks to apply deep learning and computer vision algorithms to a general-purpose robotic classification system.

The objective of this final degree project is the study and design of artificial neural networks for their possible implementation in industry. A bibliographic study is carried out to understand the operation of this type of algorithms and obtain the ability to design own models for different purposes.

For the simulation of the robotic classification system, the CoppeliaSim simulator has been used. In this environment, the training data is generated, and the validations of the created models are carried out. Also, TensorFlow and Keras have been used programming in the Python programming language for the design of the networks. Two problems have been proposed: i) the identification of objects in an automated warehouse; and ii) the second, obtaining the location of an object, using virtual cameras in both situations.

Finally, the results obtained from the networks are analyzed and their viability in modern industry is verified.

1. Introducción

1.1 Motivación del proyecto

Actualmente, el ser humano se encuentra en una etapa histórica representada por un cambio tecnológico referente a una nueva manera de organizar los medios de producción. Esta etapa se ha denominado Cuarta Revolución Industrial o Industria 4.0 [1], donde la Inteligencia Artificial (IA) es señalada como el elemento central de esta transformación. La IA hace posible que las máquinas aprendan de la experiencia, se ajusten a nuevas situaciones y realicen tareas de forma similar a los humanos [2].

Dentro del avance de las técnicas implementadas en el campo de IA, el uso de redes neuronales artificiales y algoritmos de aprendizaje profundo (Deep-Learning) son elementos clave para la detección e identificación de patrones [3] como, por ejemplo, aplicaciones para el reconocimiento de caracteres, de voz, de objetos, interpretación de fotografías, predicción de terremotos, predicciones meteorológicas, conducción automática. [4].

1.2 Objetivos

El principal objetivo de este Trabajo de Fin de Grado es el desarrollo de un sistema de clasificación automático y flexible utilizando tecnologías presentes en la Industria 4.0. Más concretamente, con el desarrollo de este trabajo se pretende obtener una visión más profunda en las capacidades de la inteligencia artificial y de los algoritmos del aprendizaje profundo y su aplicación al campo de la ingeniería.

Como se ha mencionado anteriormente, el proyecto se desarrollará en simulación y, para ello, se utilizará el programa CoppeliaSim [5] , como entorno de simulación de una célula robotizada. Esta célula estará formada, al menos, por una estantería o similar para almacenar piezas de diferente forma y/o color, un brazo robotizado y una cinta transportadora para la movilización de las piezas.

Asimismo, también se hará uso de sensores simulados, entre ellos, el más importante será un cámara enfocada y centrada en la estantería. Los datos proporcionados por la cámara serán pasados por una red neuronal usando TensorFlow para la detección del objeto que hay, o si no hay ninguno, en cada célula de la estantería.

Después de obtener el estado de la estantería, el brazo robótico debe ser capaz de acceder a las diferentes piezas de esta y hacer, por ejemplo, un ordenamiento de la estantería, la extracción de una pieza solicitada o localizar un hueco para colocar otra. El programa CoppeliaSim incluye un lenguaje propio para programar el comportamiento del entorno, el lenguaje LUA. Sin embargo, en este Trabajo Fin de Grado (TFG) el objetivo es usar Python para programar el control de la célula. Para ello, se deberá hacer una conexión entre CoppeliaSim y un entorno de ejecución de código Python [6].

La segunda aplicación de las redes neuronales que se realiza en el TFG es la obtención de la localización de un objeto en el escenario, de igual manera como en el caso anterior, será mediante el uso de una cámara. El objeto para localizar es un robot móvil Summit. Además, se colocarán objetos en la posición obtenida por la red para poder ver una posible aplicación de este tipo de red.

Con este fin, se han propuesto los siguientes objetivos:

- ***Objetivo 1: Comprensión del funcionamiento de las redes neuronales.***

  El estudio del funcionamiento de las redes neuronales, la matemática interna y como se desarrolla su entrenamiento. También su evolución a lo largo de estos últimos años y las técnicas que se han ido implementando en el campo.

- ***Objetivo 2: Modelado de una escena robotizada.***

  Crear un modelo de un sistema de clasificación robotizado de propósito general en un simulador de robótica donde poder realizar ensayos con lo aprendido sobre las redes neuronales. Para ello será necesario familiarizarse con el programa de diseño de entorno robotizado.

- ***Objetivo 3: Diseño e implementación de las redes.***

  Para ello será necesario familiarizarse con las librerías que permitan diseñar redes neuronales, después realizar las arquitecturas de red y entrenarlas hasta lograr el objetivo propuesto para cada una de ellas.

1.3 Contexto

1.3.1 Definición ¿Qué es la visión artificial?

La visión artificial es una disciplina científica que aplica diferentes métodos para intentar otorgar a las máquinas una comprensión a alto nivel a partir de la capacidad de obtener imágenes del entorno mediante cámaras.

Desde el punto de vista científico, es la búsqueda de modelos capaces de extraer, comprender y analizar la información almacenada en imágenes. Desde el punto técnico es la implementación de estos modelos en computación para automatizar tareas que requieren del sistema visual humano [7].

La vista cuenta con más de dos millones de terminaciones nerviosas. La radiación externa es recibida por el ojo y se transforma en señales nerviosas que son procesadas por el cerebro. Para ello, la retina del ojo cuenta con dos células que funcionan como transductores:

- Bastones: sensibles a la intensidad lumínica

- Conos: encargados de detectar el color. Existen tres tipos de conos, cada uno sensible a un tipo de color, rojo (eritropsina 560 nm), azul (cianopsina 420 nm) y verde (cloropsina 530nm) [8].

De aquí parte la teoría tricromática, es decir, la mezcla de estos tres colores puede formar cualquier color.

Partiendo de la inspiración en la visión humana, la obtención de imagen digital se hace mediante una cámara digital. Esta cámara está formada por dispositivo sensible a una banda del especto electromagnético y produce una señal eléctrica proporcional al nivel de energía detectado. Después se convierte esta señal analógica en una digital mediante conversores A/D y se almacena en una memoria.

Por lo tanto, una imagen digital es una representación discreta de una proyección de una imagen real (continua). Una manera de almacenar la imagen digital es mediante una matriz. Esta matriz es un conjunto de valores dispuestos en columnas y filas que se denominan pixeles. Un píxel es la unidad mínima de información contenida en una imagen, y estos pueden ser monocromáticos (escala de grises) o tener una profundidad de color (como se ve en la Figura 1.1).

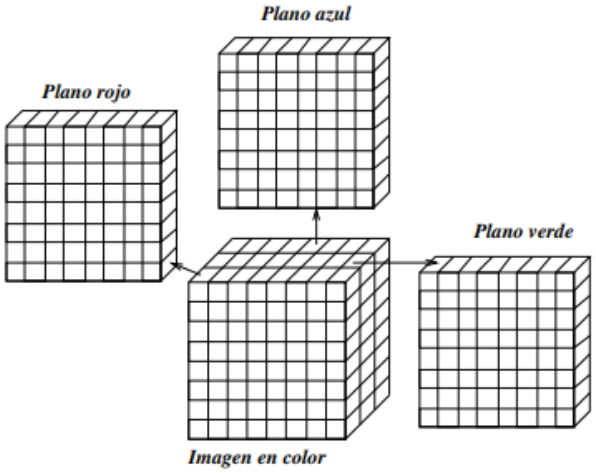


Figura 1.1 : Representación de una imagen a color

Cuando la imagen tiene profundidad de color, la imagen se puede convertir en tres matrices, cada una con un color primario, siguiendo la teoría tricromática basada en la obtención de imagen en el ser humano. A este modelo se le denomina modelo RGB (rojo, verde, azul)

RGB es un modelo basado en la síntesis aditiva, lo que significa que a partir de estos colores primarios se puede generar cualquier color visible al ojo humano mediante la suma de las intensidades de los colores primarios (Figura 1.2). La notación para la intensidad de cada color se puede representar mediante valores relativos entre 0 y 1 o 0% y 100%, véase, por ejemplo, la Figura 1.3. En la práctica se suelen usar 8 bits por color (1 byte) lo que nos

proporciona un rango de 0 a 255 representaciones por cada color primario (2^8 = 256), o 16 777 216 variaciones de color ($2^{8·3}=2^{24}$).

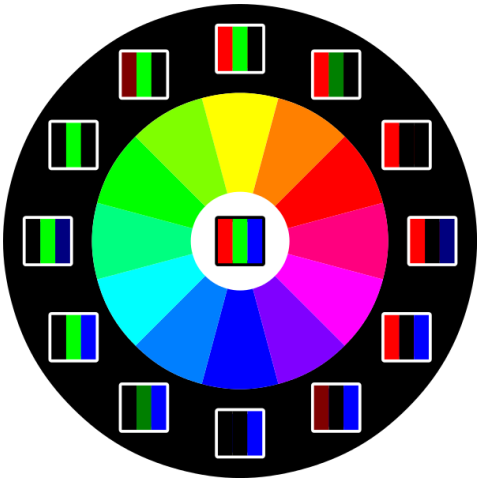


Figura 1.2: Formación de diferentes colores a partir de los colores base

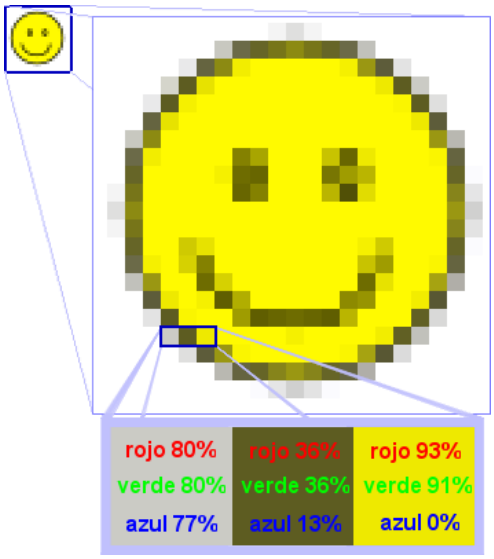


Figura 1.3: Formación de diferentes colores a partir de los colores base

Otro concepto importante es la resolución de imagen. Esto es referido al tamaño de la matriz que representa a la imagen y como las imágenes suelen ser rectangulares, su notación es la cantidad de pixeles a lo ancho x cantidad de pixeles a lo alto, véase Figura 1.4. Por ejemplo, la resolución FullHD o 1080p está formada por 1920 x 1080 pixeles, lo que implica una resolución de 2 073 600 pixeles (1920*1080) o 2.1 Megapixeles.

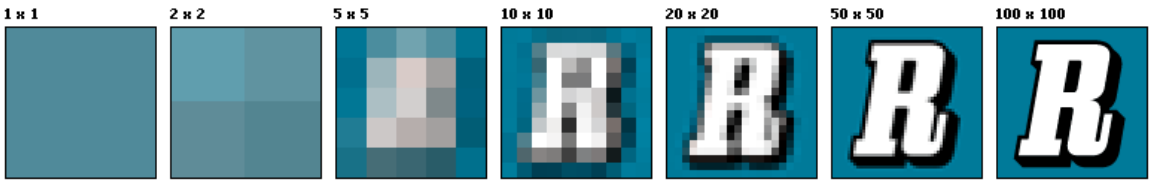


Figura 1.4: Comparativa de distintas resoluciones

1.3.2 *Historia del aprendizaje profundo y las redes neuronales convolucionales*

Cronología:

- McCullock y Walter Pitts,1943. La neurona de MCColloch-Pitts es un modelo simplificado para explicar el funcionamiento de los procesos reales en las estructuras neuronales. Esto centró el estudio en los procesos biológicos y por otro lado puso el primer pilar en la creación de redes neuronales artificiales para la IA. [9]

- Aprendizaje de Hebb, 1949. Es un intento de explicar la plasticidad sináptica que afirma que un aumento en la eficiencia sináptica surge de la estimulación repetida y persistente de una célula presináptica. Resumidamente 'Las células que se disparan juntas se conectan entre sí'. Estos estudios realizados por Hebb pusieron el fundamento de la mayoría de las funciones de aprendizaje en redes neuronales [10].

- Rosenblatt, 1958. El psicólogo Frank Rosenblatt creo el perceptrón. Es el algoritmo que estableció la base de las redes neuronales artificiales. Fue implementado en hardware que contaba con una matriz de 400 fotocélulas conectadas aleatoriamente a las neuronas (Mark I perceptrón). Los pesos se codificaban en potenciómetros y su actualización (aprendizaje) era realizado mediante motores eléctricos.

  La situación parecía prometedora para el perceptrón, pero hubo un estancamiento en su campo ya que no se pudo entrenar para detectar muchas clases de patrones [11].

- David Hubel y Torsten Wiesel, 1959. El trabajo de Hubel y Wiesel en 1959 propuso una explicación al funcionamiento de la corteza visual, basándose en el descubrimiento de dos tipos de células en esta corteza: células simples y células complejas. Estas células que se estructuraban en cascada en la capa visual primaria especializada en el procesamiento de información acerca de objetos estáticos y en movimiento [12].

- ADALINE, 1960. Una de las primeras redes neuronales aplicadas a un problema real. Su nombre viene del inglés **Ada**ptive **Li**near **Ne**uron (neurona lineal adaptativa), véase Figura 1.5. Esta red contaba con una sola capa basándose en la neurona McCulloch-Pill. Su aprendizaje se basaba en la regla Delta [13].

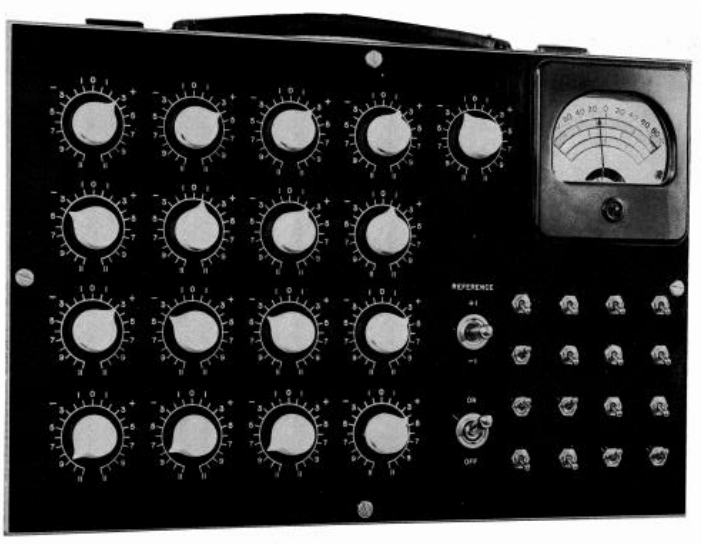


Figura 1.5: Adaline

Fuente: reporte técnico *An adaptive "ADALINE" Neuron using chemical "memistors"*

- Marvin Minsky y Seymour Papert, 1969. En 1969, Minsky y Papert publicaron Perceptrons, un libro dedicado al modelo de Rosenblatt, quien hizo predicciones sobre el perceptrón demasiado optimistas. El escrito mostraba pruebas matemáticas donde se reconocían las fortalezas de los perceptrones, pero también mostraban importantes limitaciones de estos. La más importante fue el problema de la función XOR [14].

  La crítica al perceptrón en el libro fue responsable de la época conocida como invierno de la inteligencia artificial. Un cambio en la dirección de la investigación en el campo de las redes neuronales artificiales.

- Paul Werbos, 1974. Desarrollo de la idea básica sobre el algoritmo de propagación hacia atrás (back-propagation). Este algoritmo actualmente es utilizado para el entrenamiento de las redes neuronales artificiales, ajustando sus parámetros de pesos y sesgos mediante el cálculo del gradiente de la función de error. El desarrollo de este algoritmo permitió un entrenamiento práctico de redes multicapa [15].

- Kunihiko Fukushima en 1979. Fukushima propuso una red de múltiples capas centrada en el reconocimiento de caracteres escritos a mano. Esta red llamada Neocogitron fue inspirada en el modelo propuesto de Hubel y Wiesel. El modelo de Fukushima fue el precursor de las redes convolucionales.

- Rumelhart, Hinton & Williams, 1986. El artículo titulado "*Learning representations by back-propagating errors*" donde Geoffrey Hilton y sus compañeros volvieron a popularizar las redes neuronales al aplicar el algoritmo de back-propagation a redes

multicapas. Sus experimentos demostraron que tales redes pueden aprender representaciones internas útiles de datos [16].

- Yann LeCun, 1989. LeCun junto con sus compañeros utilizaron el método de back-propagation para entrenar los coeficientes de los núcleos de convolución directamente de imágenes de números escritos a mano. Esto implicaba que todo el entrenamiento fue automático y se convirtió en la base de la visión artificial moderna [17].

1.3.3 Simuladores de sistemas robotizados

La simulación en robótica se utiliza para la creación de modelos virtuales con el fin de crear y probar diferentes aplicaciones para robots físicos sin depender de máquinas reales, lo que implica un ahorro de costes y tiempo. Los simuladores también permiten a los usuarios crear entornos de simulación con objetos rígidos, móviles, sensores, fuentes de luz, aplicación de física dinámica, etc. La principal desventaja de estos simuladores es cierta pérdida de fidelidad con el mundo real de algunos comportamientos físicos.

Algunos de los simuladores más utilizados son:

1.3.3.1 Simulador 3D Gazebo

Es una aplicación que formaba parte de Player Project, una interfaz de código abierto para dispositivos robóticos. Gazebo Simulator ahora es un proyecto independiente desarrollado por Willow Garage. Integra motor de físicas ODE (Open Dynamics Engine), Bullet, Dynamic Animation and Robotics Toolkit (DART) y Simbody. Utiliza como renderizador gráfico OpenGL. Esta aplicación se puede descargar de forma gratuita desde la página oficial y se puede trabajar con C y C++ [18].

1.3.3.2 Microsoft Robotics Studio

Es un entorno diseñado en Windows para simulación y control de robots. Usa el motor PhysX diseñado originalmente por AGEIA (ahora propiedad de NVIDIA). Puede admitir un amplio conjunto de plataformas robóticas, ya sea ejecutándose directamente en la plataforma (si se dispone de un PC integrado con Windows) o controlando el robot desde un PC con Windows a través de una conexión Wi-Fi o Bluetooth®. Microsoft Robotics Developer Studio 4 proporciona un lenguaje de programación visual (VPL) que permite a los desarrolladores crear aplicaciones simplemente arrastrando y soltando componentes en un lienzo y conectándolos [19].

1.3.3.3 AnyKode Marilou

Un entorno de simulación y modelado de robots, donde se pueden usar diferentes lenguajes de programación (C/C++, VB#, J#, C#, C++ CLI) en Windows o distribuciones Linux. No cuenta con Python [20].

1.3.3.4 CoppeliaSIM

Es el entorno elegido para este proyecto. Anteriormente denominado como V-Rep, se basa en una arquitectura de control distribuido: cada objeto o modelo se puede controlar individualmente a través de un script integrado, un Plug-in, un nodo ROS o BlueZero, un cliente API (application programming interface, interfaz de programación de aplicación) remoto o una solución personalizada. [5]

Características que lo hacen preferente a los otros mencionados:

- Se puede trabajar con lenguajes C/C++, Python, Java, Lua, MATLAB u Octave.
- Sistemas operativos Windows, MacOS y Linux
- Detección de obstáculos, cálculo de cinemáticas, cálculo de distancia mínima
- Sensores de visión, de proximidad, de fuerza…
- Creación dinámica de objetos durante la simulación.
- Cuatro motores de física: Bullet, ODE, Vortex y Newton.

1.4 Relación con las competencias

Para la realización de este proyecto han sido necesarias tener ciertas competencias básicas que se han desarrolladas. Estas competencias básicas definidas en el R.D. 1393/2007, de 29 de octubre, para todos los títulos de grado:

- Poseer y comprender conocimientos (CB1)

Que los estudiantes hayan demostrado poseer y comprender conocimientos en un área de estudio que parte de la base de la educación secundaria general, y se suele encontrar a un nivel que, si bien se apoya en libros de texto avanzados incluye también algunos aspectos que implican conocimientos procedentes de la vanguardia de su campo de estudio.

- Aplicación de conocimientos (CB2)

Que los estudiantes sepan aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro de su área de estudio.

- Capacidad de emitir juicios (CB3)

Que los estudiantes tengan la capacidad de reunir e interpretar datos relevantes (normalmente dentro de su área de estudio) para emitir juicios que incluyan una reflexión sobre temas relevantes de índole social, científica o ética.

- Capacidad de comunicar y aptitud social (CB4)

Que los estudiantes puedan transmitir información, ideas, problemas y soluciones a un público tanto especializado como no especializado.

- Habilidad para el aprendizaje (CB5)

Que los estudiantes hayan desarrollado aquellas habilidades de aprendizaje necesarias para emprender estudios posteriores con un alto grado de autonomía

Entre las competencias transversales que han sido necesarias para este proyecto:

- Conocimientos básicos de la profesión (UAL1)
- Habilidad en el uso de las TIC (UAL2)
- Capacidad para resolver problemas (UAL3)
- Comunicación oral y escrita en la propia lengua (UAL4)
- Capacidad de crítica y autocrítica (UAL5)
- Conocimiento de una segunda lengua (UAL7)
- Capacidad para aprender a trabajar de forma autónoma (UAL9)

Competencias específicas del grado de ingeniería electrónica industrial:

- Conocimiento en materias básicas y tecnológicas, que les capacite para el aprendizaje de nuevos métodos y teorías, y les dote de versatilidad para adaptarse a nuevas situaciones. (CT3)
- Capacidad de resolver problemas con iniciativa, toma de decisiones, creatividad, razonamiento crítico y de comunicar y transmitir conocimientos, habilidades y destrezas en el campo de la Ingeniería Industrial. (CT4)
- Capacidad para la resolución de los problemas matemáticos que puedan plantearse en la ingeniería. Aptitud para aplicar los conocimientos sobre: álgebra lineal; geometría; geometría diferencial; cálculo diferencial e integral; ecuaciones diferenciales y en derivadas parciales; métodos numéricos; algorítmica numérica; estadística y optimización. (CB1)
- Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería. (CB3)
- Conocimientos sobre los fundamentos de automatismos y métodos de control. (CRI6)
- Conocimiento y capacidad para el modelado y simulación de sistemas. (CTEE7)
- Conocimientos de regulación automática y técnicas de control y su aplicación a la automatización industrial. (CTEE8)
- Conocimientos de principios y aplicaciones de los sistemas robotizados. (CTEE9)
- Capacidad para diseñar sistemas de control y automatización industrial. (CTEE11)

1.5 Planificación temporal

Para este proyecto se ha desarrollado una planificación temporal donde será necesario completar las siguientes tareas:

- A - Estudio bibliográfico, primeras pruebas con las redes neuronales y primer contacto con los programas y librerías.
- B - Diseño de la célula, tanto el modelado de la célula como el control de sus elementos.
- C – Diseño de redes, creación de conjuntos de datos de entrenamiento, el propio entrenamiento y la implementación de las mencionadas redes en la célula.
- D - Ensayos y ajustes posteriores de los diseños iniciales, tanto de la célula, como del control y de las redes.
- E – Validación de los modelos, análisis de resultados y elaboración del proyecto.

Teniendo en cuenta que al día se dedica 5 horas de media para el proyecto, cada semana son 25 horas. La siguiente tabla (Tabla 1.1 : Planificación temporalTabla 1.1) muestra, organizada por los meses y semanas, la planificación seguida para conseguir completar el proyecto.

		Tarea					
Mes	Semana	A	B	C	D	E	
Junio	1	25					
	2	25					
	3	25					
	4	12.5	12.5				
Julio	1		12.5	12.5			
	2			25			
	3			12.5	12.5		
	4			12.5	12.5		
Agosto	1			12.5	12.5		
	2				12.5	12.5	
	3				12.5	12.5	
	4					25	
Septiembre	1					25	Total
Horas totales:		82.5	25	75	62.5	75	325

Tabla 1.1 : Planificación temporal

1.6 Estructura del trabajo de fin de grado

Este proyecto se divide en 5 capítulos principales:

- Primer capítulo: de manera introductoria se expone la motivación para realizar este proyecto, el contexto donde se encuentra actualmente el campo de la visión artificial y los objetivos que se pretende alcanzar durante la realización del trabajo de fin de grado.

- Segundo capítulo: se explican los métodos y algoritmos, referentes a las redes neuronales, que se van a utilizar o se han contemplado su uso en el proyecto. Se expondrán los diferentes programas que serán necesarios y las diferentes bibliotecas que permiten implementar la célula robotizada y el diseño de control. También se menciona el hardware usado.

- Tercer capítulo: se abarca el diseño de la célula robotizada junto con el diseño de control de esta. También se comenta como se han creado las redes neuronales y como ha sido el entrenamiento que han seguido. Finalmente se exponen los resultados del funcionamiento de las redes neuronales en la escena.

- Cuarto capítulo: Se exponen las conclusiones a las que se han llegado durante el desarrollo del trabajo de fin de grado.

1.7 Resumen de resultados

Revisando los objetivos propuestos, durante la realización del proyecto se ha alcanzado una comprensión de los fundamentos referidos a las redes neuronales, su funcionamiento interno y matemático, además de las posibilidades que aportan en un entorno industrial, así como, de los métodos de implementación y optimización del proceso de entrenamiento.

Como resultado, se han diseñado diferentes redes neuronales para resolver dos tipos de problemas dentro del campo de visión artificial. Para el primer tipo, la identificación de objeto en imagen se ha desarrollado una red que ha conseguido un error de 0.0132%. Para lograrlo se ha requerido de 8164 imágenes de entrenamiento y 1943 imágenes más para validar el modelo encargado de realizar la identificación. Esto implica, que teniendo 36 objetos por estantería (incluyendo los huecos), se han necesitado un total de 280 imágenes del almacén donde se hayan los objetos, cada una con una configuración distinta en cuanto a la iluminación y posición de dichos objetos. Además, cada imagen debe de ir con la información de la configuración de objetos para poder realizar un aprendizaje.

El modelo de red artificial utilizado para conseguir dicho fin, es una red neuronal convolucional. Un tipo de red que al aplicar el filtro de convolución logra un mejor funcionamiento para extraer y procesar información de forma más optima a partir de imágenes que otro tipo de redes.

El segundo tipo de problema donde se ha planteado usar redes neuronales es para la localización de un objeto dentro un escenario cuya posición varía. Para ello, se ha diseñado una red convolucional con ciertas modificaciones para intentar lograr un error mínimo.

Para el aprendizaje de la segunda red se han usado 8997 datos con la imagen y la posición del objeto. De estos datos, 5000 se han usado para entrenar la red, 2000 para validar y ajustar los hiperparámetros del aprendizaje, obteniendo un error medio relativo al área total donde

es posible hallarse el objeto de 4.9289%. El resto de los datos se ha usado para realizar una comprobación del funcionamiento de la red.

El código de las implementaciones, el archivo de CoppeliaSim, el conjunto de datos utilizado para el entrenamiento y los modelos de las redes (archivos .h5) se pueden encontrar en el enlace de YouTube: *https://youtu.be/1WjnIe4aJmA* .

2. Materiales y métodos

En este apartado se explicarán los algoritmos matemáticos que se han utilizado en la implementación de las redes neuronales. También se comentará el funcionamiento del Simulador CoppeliaSim y las librerías de Python que se van a utilizar. Por último, se especifican los componentes del computador donde se ha realizado todo el TFG.

2.1 Redes neuronales

Son modelos matemáticos y computacionales que están basados en las conexiones neuronales de un cerebro biológico. Consta de capas de neuronas artificiales llamadas perceptrones que se encuentran interconectadas entre sí. Estas interconexiones permiten la comunicación entre diferentes neuronas que van transmitiendo la información de capa en capa según el tipo de arquitectura o modelo de red. Son los modelos de *machine learning* más populares actualmente.

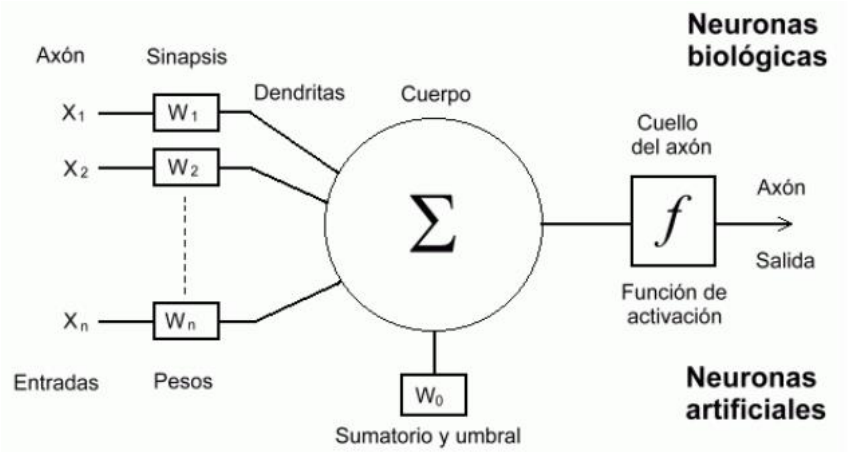


Figura 2.1: Similitudes entre neurona biológica y neurona artificial
Fuente: http://www.cs.us.es/~fsancho/?e=72

2.1.1 Neurona artificial

Una neurona artificial se puede definir como una función matemática que simula el comportamiento de una neurona biológica, es decir, es la unidad básica de procesamiento que se va a encontrar en una red neuronal (véase el símil entre una neurona biológica y una artificial en la Figura 2.1.). El cálculo interno de estas neuronas, su función matemática, consta de una suma ponderada de todos los valores de su entrada. Estos pesos les dan cierta importancia a algunas entradas y les quita a otras, subiendo o bajando la intensidad de las diferentes señales en el comportamiento de la neurona.

A esta suma se le añade un valor independiente, comúnmente llamado 'bias' o termino de sesgo(2.1). Los valores de pesos y sesgos se irán modificando durante el proceso de aprendizaje, defiendo la respuesta deseada según diferentes configuraciones de entradas. Este proceso de aprendizaje se explicará detalladamente más adelante.

En la neurona k=1:

$$s_1 = w_{11}x_1 + w_{21}x_2 + \cdots + w_{n1}x_n + b_1 \quad (2.1)$$

$x_i \rightarrow$ valor de entrada (input)

$w_{ik} \rightarrow$ peso (weight) de la conexión

$b_k \rightarrow$ sesgo, valor umbral, bias

$s_1 \rightarrow$ suma ponderada

Hasta este punto, modelo se comporta como un modelo de regresión lineal. Sin embargo, hay un último elemento que se une al modelo, la función de activación.

$$y_k = f_a(w_{1k}x_1 + w_{2k}x_2 + \cdots + w_{nk}x_n + b_k) \quad (2.2)$$

$y_k \rightarrow$ salida de neurona k (output)

$f_a \rightarrow$ función de activación, no lineal

$w_{1k}x_1 + w_{2k}x_2 + \cdots + w_{nk}x_n + b_k \rightarrow$ suma ponderada

Esta función es crucial en el funcionamiento de una red neuronal. En un ejemplo de una red con varias capas de neuronas interconectadas, el funcionamiento total equivaldría a una sola neurona, si estas funcionasen como regresiones lineales. Para no obtener un resultado lineal, habría que realizar una manipulación no lineal a cada neurona, aquí es donde entran las funciones de activación.

2.1.2 Funciones de activación

En las redes computacionales, las funciones de activación son funciones que de forma simplificada dan una respuesta binaria, al activarse como 'ON' (1) o desactivarse como 'OFF' (0). Algunas de estas funciones son:

- ***Función unidad lineal rectificada (ReLU, por sus siglas en inglés).***

Es la función de activación más utilizada. Una ventaja que tiene y por lo que es tan popular es su buen rendimiento en redes convolucionales (tratamiento de imágenes). En la Figura 2.2 se puede observar el funcionamiento de la función que viene define como:

$$f(x) = \max(0, x) \quad (2.3)$$

Rango: $[0, \inf)$

$$f'(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \quad (2.4)$$

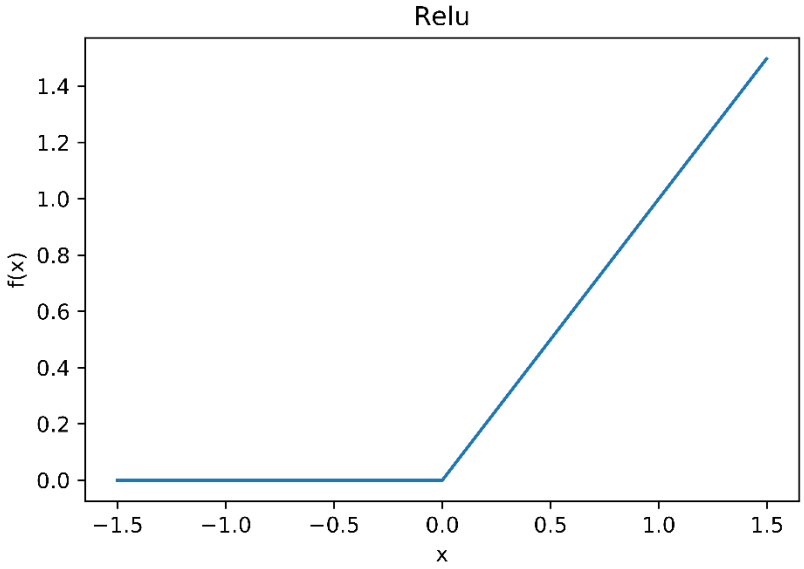


Figura 2.2 : Función Relu

- ***Función sigmoide***

Función cuya evolución se define por dos asíntotas horizontales, empezando en un valor inicial donde se mantiene hasta llegar a una región con fuerte crecimiento para acabar en una asíntota superior. Ofrece un buen rendimiento en la última capa, pero tiene una lenta convergencia. Aun así, su uso es también bastante extendido al igual que la función Relu.

La función tiene forma de 'S' (Figura 2.3) y se definiría como:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

Rango: $(0, 1)$

$$f'(x) = f(x)(1 - f(x)) \tag{2.6}$$

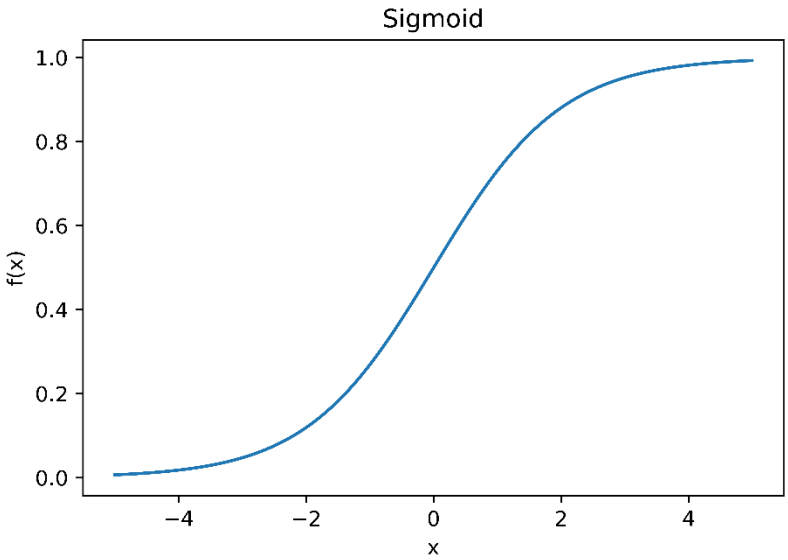


Figura 2.3: Función sigmoide

- ***Función tangente Hiperbólica (TAHN)***

Es una función muy parecida a la función sigmoide pero esta acotada entre -1 y 1, centrada en 0 (Figura 2.4). Ofrece un buen desempeño en red neuronales recurrentes para la decisión entre opciones contrarias entre sí.

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.7}$$

$$\text{Rango: } (-1, 1)$$

$$f'(x) = 1 - f(x)^2 \tag{2.8}$$

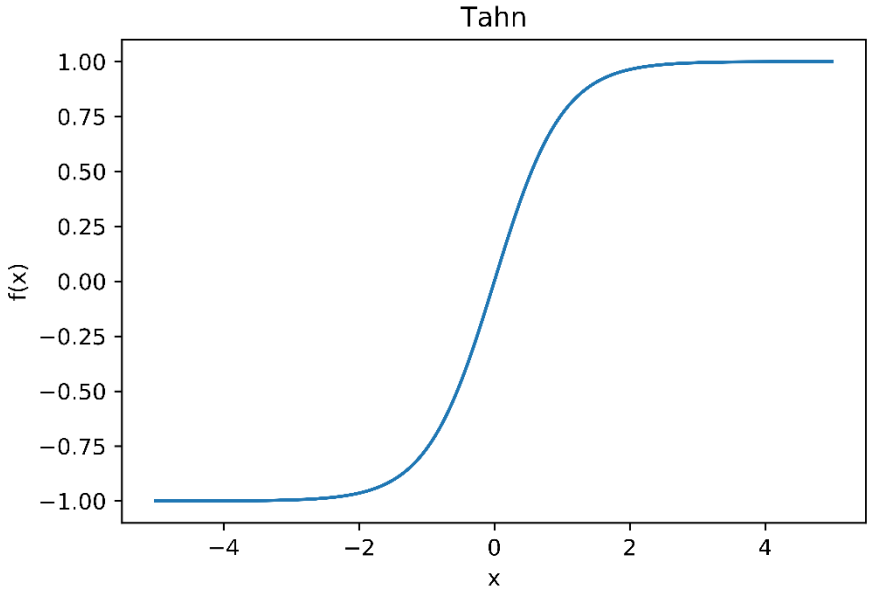


Figura 2.4 : Función Tahn

- ***Función exponencial normalizada (SOFTMAX)***

Dado un vector de entrada, devuelve una distribución de probabilidad con los valores exponenciales de la entrada. Esto deja un vector a la salida con valores reales en el rango de 0 a 1 donde todos estos suman 1. Se usa en la capa final de una red neuronal clasificadora multiclase y en el aprendizaje por refuerzo.

$$f(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \text{ para } i = 1, \dots, K \text{ y } \vec{x} = (x_1, \dots, x_K) \in \mathbb{R}^K \tag{2.9}$$

$$\frac{\partial f(\vec{x})_i}{\partial x_j} = f(\vec{x})_i \left(\delta_{ij} - f(\vec{x})_j\right) \qquad \delta_{ij} \rightarrow \text{delta de Kronecker} \tag{2.10}$$

$$\delta_{ij} = 0 \text{ si } i \neq j \text{ y } \delta_{ij} = 1 \text{ si } i = j$$

En la Figura 2.5 se observa la respuesta entre las tres primeras funciones de activación definidas anteriormente.

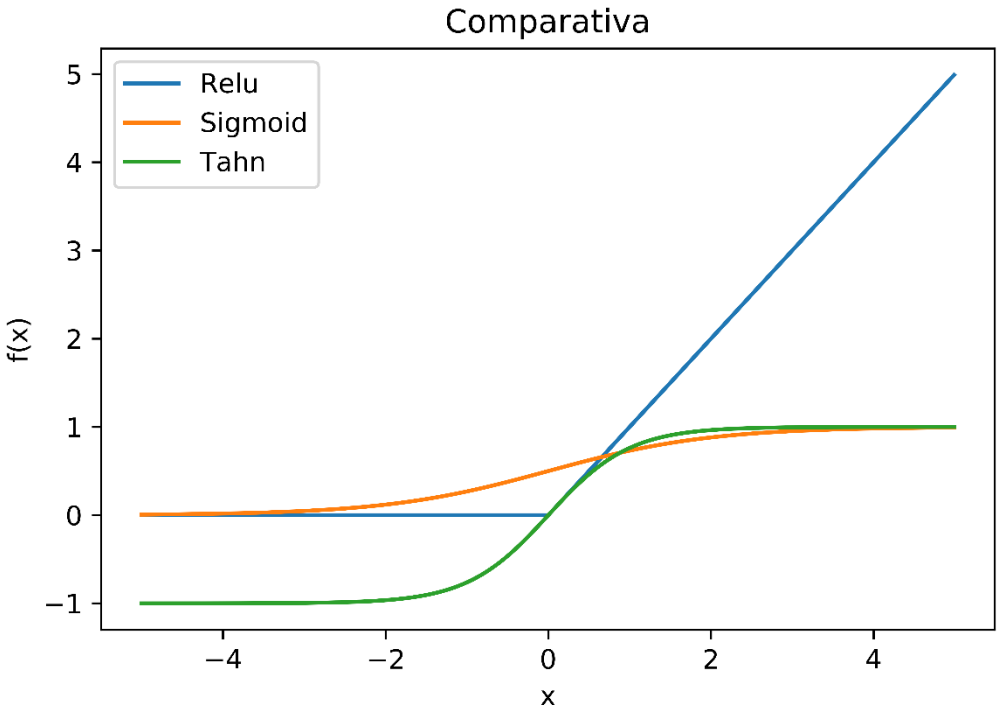


Figura 2.5 : La comparativa entre respuesta de las funciones Relu, Sigmoide, Tahn

2.1.3 Capas y estructura de red

A la hora de estructurar la red se suelen definir por un número de capas y cada capa con un numero de neuronas artificiales.

En una capa individual con *n* entradas y *m* neuronas se tiene la siguiente expresión:

$$\begin{aligned} y_1 &= f_a(w_{11}x_1 + w_{21}x_2 + \cdots + w_{n1}x_n + b_1) \\ y_2 &= f_a(w_{12}x_1 + w_{22}x_2 + \cdots + w_{n2}x_n + b_2) \\ &\cdots \\ y_m &= f_a(w_{1m}x_1 + w_{2m}x_2 + \cdots + w_{nm}x_n + b_m) \end{aligned} \tag{2.11}$$

- $n \rightarrow$ número de entradas de la capa
- $m \rightarrow$ número de neuronas y salidas de la capa

Expresado en forma matricial:

$$\begin{pmatrix} y_1 \\ \cdots \\ y_m \end{pmatrix} = f_a\left[\begin{pmatrix} w_{11} & \cdots & w_{n1} \\ \cdots & \cdots & \cdots \\ w_{1m} & \cdots & w_{nm} \end{pmatrix}\begin{pmatrix} x_1 \\ \cdots \\ x_m \end{pmatrix} + \begin{pmatrix} b_1 \\ \cdots \\ b_m \end{pmatrix}\right] \tag{2.12}$$

$$\mathrm{Y_{m\,x\,1}} = f_a(\mathrm{W_{m\,x\,n}X_{n\,x\,1}} + \mathrm{B_{m\,x\,1}}) \tag{2.13}$$

Entre las capas de una red se puede hacer una clasificación general.

- Capa de entrada: donde cada nodo es una entrada a la red.
- Capa de salida: cada nodo es una neurona cuya salida es una salida de la red.
- Capa oculta: es una capa que sitúa entre la capa de entrada y la capa de salida, cuyas neuronas reciben la información de una capa y la transmiten hacia otra capa.

Véase un ejemplo del esquema de una red simple de dos entradas, una capa oculta con dos neuronas y la capa de salida con una sola neurona en la Figura 2.6.

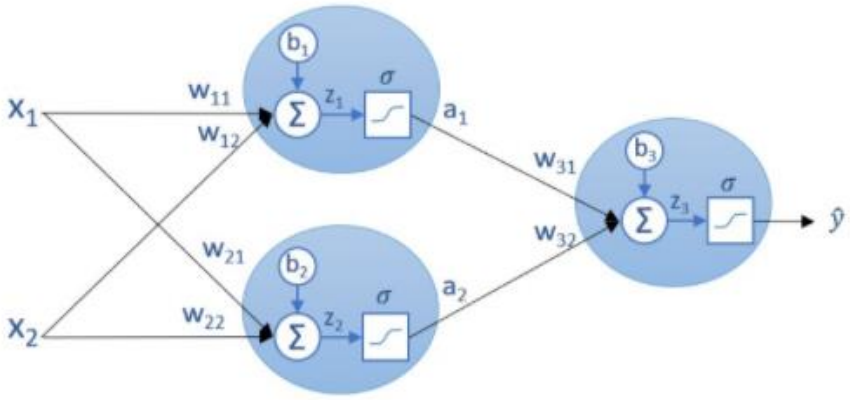


Figura 2.6: Esquema de una red neuronal artificial simple vista desde dentro

Fuente: https://www.interactivechaos.com/manual/tutorial-de-deep-learning/detalle-de-la-red-neuronal

2.1.4 Función de coste o Error

En el ámbito de aprendizaje supervisado, la función de coste sirve para determinar la diferencia entre el valor estimado por la red y el valor correcto. Existen diferentes funciones que se usan como estimadores del error:

- ***Error cuadrático medio***

Penaliza diferencias grandes, aunque es difícil de interpretar, proporciona un buen funcionamiento en optimización de regresión (2.14).

$$ECM = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2 \tag{2.14}$$

- ***Error absoluto medio***

Penaliza menos los valores grandes, más fácil de interpretar, más difícil la convergencia y diferenciación.

$$EAM = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{2.15}$$

- ***Entropía cruzada categórica***

La entropía cruzada se utiliza como función de error en el aprendizaje de redes neuronales para los problemas de clasificación de etiquetas o variables categóricas.

$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log\left(p_{ij}\right) \tag{2.16}$$

2.1.5 Optimizadores

Al inicializar los valores de parámetros de pesos (**W**) y los sesgos (**B**), la respuesta de la red dará una respuesta aleatoria. Mediante la función de error o de coste es posible calcular el propio error que comete la red y los optimizadores variarán los valores de los parámetros (**W** y **B**) de todas las capas de la red para reducir este error cometido por la red durante la propagación hacia atrás de la etapa de entrenamiento (**back-propagation**) [21].

La variación de los parámetros irá ligada al valor de la tasa de aprendizaje (**learning rate**) que tendrá que ser definida a la hora de realizar la optimización. A mayor valor, el descenso del error será más rápido, pero podrá llegar a un punto de no convergencia y a menor valor puede caer en un mínimo local como se puede ver en la Figura 2.7.

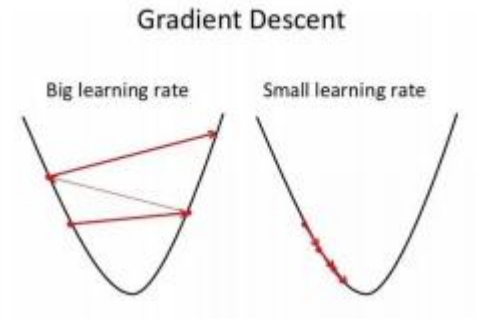


Figura 2.7 : Comparativa entre los efectos que pueden producir valores bajos y altos de la tasa de aprendizaje

Algunos de estos optimizadores son:

- ***Descenso estocástico del gradiente***

Es el método básico de optimización de redes neuronales. Este método tiene un problema muy importante, el establecer una tasa de aprendizaje (*learning rate*). Si este valor es muy pequeño puede hacer que la convergencia sea muy lenta y si es muy grande el algoritmo puede divergir mucho. Por ello existen alternativas que se mencionara más adelante.

Comúnmente se le denomina SGD por sus siglas en ingles *Stochastic Gradient Descent*. La Figura 2.8 muestra el funcionamiento del algoritmo.

$$W_t = W_{t-1} - l_R \cdot \nabla f_{Coste_i}(W_{t-1}) \tag{2.17}$$

- $\nabla f_{Coste_i}(W_{t-1}) \rightarrow$ gradiente de la función Coste o Error
- $l_R \rightarrow$ learning rate

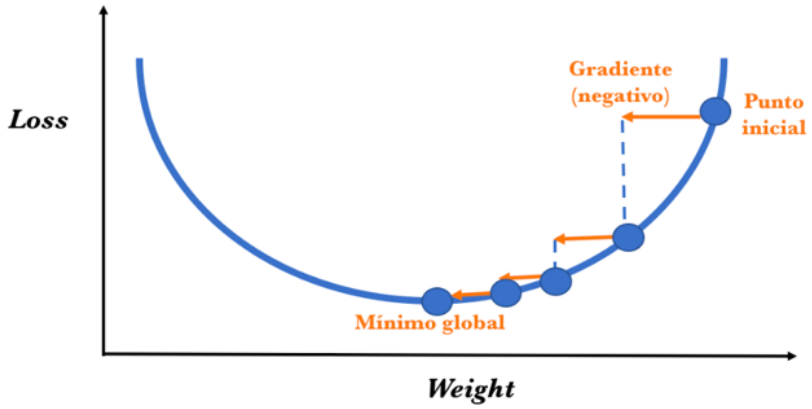


Figura 2.8: Evolución del error mediante ajuste con el algoritmo descenso estocástico del gradiente

- ***Descenso estocástico del gradiente con momentum***

El momentum acelera el descenso en direcciones similares a la anteriores. El valor α es la tasa de desvanecimiento. La Figura 2.9 muestra el funcionamiento del algoritmo.

$$\Delta W_t = \alpha \cdot \Delta W_{t-1} - l_R \cdot \nabla f_{Coste_i}(W_{t-1}) \tag{2.18}$$

$$W_t = W_{t-1} + \Delta W_t \tag{2.19}$$

O también expresado:

$$W_t = W_{t-1} + \alpha \cdot \Delta W_{t-1} - l_R \cdot \nabla f_{Coste_i}(W_{t-1}) \tag{2.20}$$

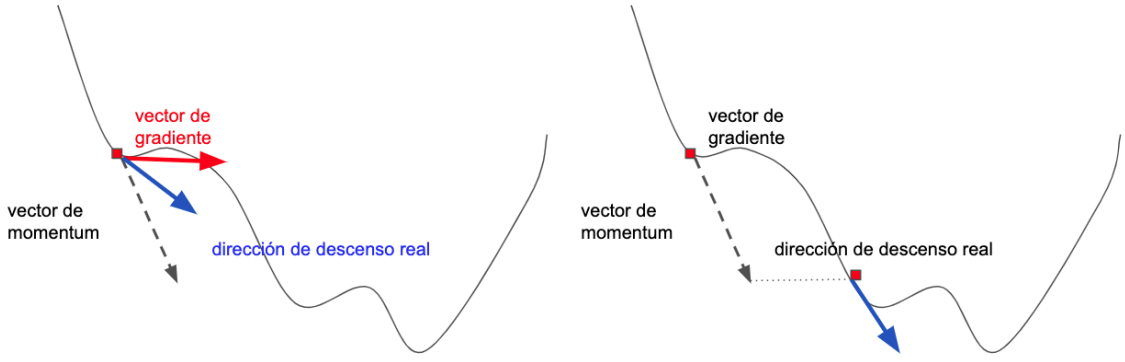


Figura 2.9 : Evolución del error mediante el optimizador con momentum

- ***Adagrad (Adaptive gradient algorithm)***

Es un método que adapta la tasa de aprendizaje según la variación que están sufriendo los parámetros. Cuando los parámetros que cambian frecuentemente tienen un cambio más bajo y los que tiene una variación menor tienen un ajuste mayor.

$$r_t = r_{t-1} + \left(\nabla f_{Coste_i}(W_{t-1})\right)^2 \tag{2.21}$$

$$W_t = W_{t-1} - \frac{l_R \cdot \nabla f_{Coste_i}(W_{t-1})}{\sqrt{r_t}} \tag{2.22}$$

Donde $\left(\nabla f_{Coste_i}(W_{t-1})\right)^2$ es referido al producto de Schur o producto Hadamard

$$\left(\nabla f_{Coste_i}(W_{t-1})\right)^2 = \nabla f_{Coste_i}(W_{t-1}) \circ \nabla f_{Coste_i}(W_{t-1}) \tag{2.23}$$

r_t es el cache con todas las variaciones de un parámetro a lo largo del proceso de aprendizaje. ϵ es un valor pequeño para impedir división entre 0 iniciando el cache $r_0 = \epsilon$.

- ***RMSProp***

Su nombre proviene del inglés *Root Mean Square Propagation* (propagación media cuadrática) y es otro método que busca corregir los efectos de la acumulación global de cache.

$$r_t = d \cdot r_{t-1} + (1-d) \cdot \left(\nabla f_{Coste_i}(W_{t-1})\right)^2 \tag{2.24}$$

$$W_t = W_{t-1} - \frac{l_R \cdot \nabla f_{Coste_i}(W_{t-1})}{\sqrt{r_t + \epsilon}} \tag{2.25}$$

d representa la tasa de descomposición/decadencia, un hiperparámetro el cual hace que las entradas anteriores en el cache influyan menos que las nuevas. El valor suele fijare en 0.9.

- ***Adadelta***

Mejora de Adagrad y RMSProp fijando el cache r_t a un pequeño número de gradientes cuadrados, los últimos n gradientes en vez todo el lote de entrenamiento.

$$r_t = d \cdot r_{t-1} + (1-d) \cdot \left(\nabla f_{Coste_i}(W_{t-1})\right)^2 \tag{2.26}$$

$$v_t = \frac{\sqrt{s_t} + \epsilon}{\sqrt{r_t} + \epsilon} \nabla f_{Coste_i}(W_{t-1}) \tag{2.27}$$

$$s_{t+1} = d s_t + (1-d) v_t^2 \tag{2.28}$$

$$W_t = W_{t-1} - v_t \tag{2.29}$$

- ***Adam (Adaptative Moment Estimator)***

Combina AdaGrad y RMSProp. Mantiene un factor de entrenamiento para cada dimensión y este factor se ve afectado por la media de momentum del gradiente [22]

$$m_t = b_1 \cdot m_{t-1} + (1 - b_1) \cdot \nabla f_{Coste_i}(W_{t-1}); m_0 = 0 \quad (2.30)$$

$$v_t = b_2 \cdot v_{t-1} + (1 - b_2) \cdot \left(\nabla f_{Coste_i}(W_{t-1})\right)^2 ; v_0 = 0 \quad (2.31)$$

$$\hat{m} = \frac{m_t}{1 - b_1^t}; \ b_1^t \rightarrow b_1 \text{elevado a t} \quad (2.32)$$

$$\hat{v} = \frac{v_t}{1 - b_2^t} \quad (2.33)$$

$$W_t = W_{t-1} - \alpha \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon} \quad (2.34)$$

Los valores b_1 y b_2 sulen fijarse 0.9 y 0.99 respectivamente, $\hat{m}$ y $\hat{v}$ son los dos momentos, el primero modela la media de los gradientes y el segundo la varianza a lo largo del tiempo.

2.1.6 Back-propagation

La salida de una red es una combinación de funciones de activación y, sumas y productos de matrices:

$$p(x) = f^L(W^L \cdot (f^{L-1}(W^{L-1} \cdot (\dots f^1(W^1 x + B^1) \dots) + B^{L-1}) + B^L) \quad (2.35)$$

Donde se va a denominar:

- L : número de capas
- W^i: matriz de pesos entre capa i-1 e i
- B^i: vector de sesgos entre capa i-1 e i
- f^i : función de activación
- x : vector de entrada a la red
- p : vector de salida de la red

Por cada entrada y salida (x_i, y_i), hay una salida predicha y por tanto un error en la predicción definido por la función de coste como $C(y_i, p(x_i))$.

$$C(y_i, p(x_i)) = C(y_i, f^L(W^L \cdot (f^{L-1}(W^{L-1} \cdot (\dots f^1(W^1 x_i + B^1) \dots) + B^{L-1}) + B^L)) \quad (2.36)$$

Se denotará con a^i la salida de cada capa y z^i como entrada a la función de activación y suma ponderada (tal como viene definido en la Figura 2.10), entonces para la capa L se tendrá la siguiente expresión:

$$a^L = f^L(z^L) \rightarrow z^L = W^L a^{L-1} + B^L \tag{2.37}$$

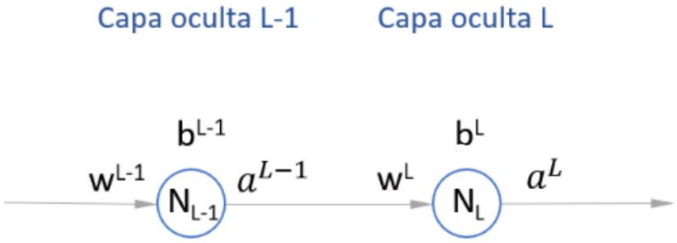


Figura 2.10 : Ejemplificación del uso de variables

Fuente: https://www.interactivechaos.com

Para obtener el gradiente de los pesos en la capa L se requiere utilizar la regla de la cadena.

$$\nabla_{W^L} f_{coste} = \frac{\partial C}{\partial W^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial W^L} \tag{2.38}$$

$$\nabla_{B^L} f_{coste} = \frac{\partial C}{\partial B^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial B^L} \tag{2.39}$$

Donde $\frac{\partial C}{\partial a^L}$ es la derivada de la función de coste respecto la función de activación de la última capa, lo que es igual a la salida de la red. Por ejemplo: función error cuadrático medio.

$$C(y_i, p_i) = \frac{1}{2} \sum_i (y_i - p_i)^2 \tag{2.40}$$

Y su derivada:

$$\frac{\partial C(y_i, p_i)}{\partial p_i} = (p_i - y_i) \tag{2.41}$$

Después, $\frac{\partial a^L}{\partial z^L}$ es la derivada de la función de activación. Siguiendo la misma metodología que antes, un ejemplo con la función sigmoide:

$$f_i(z_i) = \frac{1}{1 + e^{-z_i}} \tag{2.42}$$

$$f_i'(z_i) = f_i(z_i)(1 - f_i(z_i)) \tag{2.43}$$

Y por último $\frac{\partial z^L}{\partial W^L}$ y $\frac{\partial z^L}{\partial B^L}$, son derivadas directas de la función suma $z^L(W^L, B^L)$ (2.37) $\frac{\partial z^L}{\partial W^L} = a^{L-1}$, y $\frac{\partial z^L}{\partial B^L} = 1$.

Así que el gradiente queda:

$$\frac{\partial C}{\partial W^L} = \frac{\partial C}{\partial a^L} \cdot (f^L)' \cdot a^{L-1} = \delta^L \cdot a^{L-1} \tag{2.44}$$

$$\frac{\partial C}{\partial B^L} = \frac{\partial C}{\partial a^L} \cdot (f^L)' = \delta^L \tag{2.45}$$

- δ^i : una variable auxiliar que se utiliza en este método, llamada error de nivel o capa.

A partir de aquí, si se vuelve a aplicar la regla de la cadena en la capa siguiente, es posible ver que parte de los cálculos se ha realizado en la capa anterior y el resto es volver a aplicar derivadas simples:

$$\frac{\partial C}{\partial W^{L-1}} = \boxed{\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial W^{L-1}} = \tag{2.46}$$

$$= \delta^L \cdot W^L \cdot (f^{L-1})' \cdot a^{L-2} = \delta^{L-1} \cdot a^{L-2}$$

$$\frac{\partial C}{\partial B^{L-1}} = \boxed{\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial B^{L-1}} = \delta^L \cdot W^L \cdot (f^{L-1})' = \tag{2.47}$$

$$= \delta^{L-1}$$

Aparece la última componente que falta por definir, también una derivada directa, $\frac{\partial z^L}{\partial a^{L-1}} = W^L$ (2.37).

Después el método se repite hasta llegar a la primera capa de la red, realizando la propagación del error hacia atrás (2.48)-(2.49), capa a capa y modificando los parámetros de pesos y sesgos durante el proceso de entrenamiento:

$$\delta^{i-1} = \delta^i \cdot W^i \cdot \left(f^{i-1}\right)'; i \rightarrow \text{numero de capa} \tag{2.48}$$

$$\frac{\partial C}{\partial W^i} = \delta^i \cdot a^{i-1}; \qquad \frac{\partial C}{\partial B^1} = \delta^i \tag{2.49}$$

$$\dots$$

$$\frac{\partial C}{\partial W^1} = \delta^1 \cdot x; \qquad \frac{\partial C}{\partial B^1} = \delta^1$$

2.1.7 Aprendizaje

Resumiendo, el aprendizaje es la variación de los parámetros de pesos y sesgos de las neuronas artificiales hasta alcanzar un resultado deseado dada una entrada concreta. Para este proceso, se usa una serie de vectores con unos valores definidos. Estos vectores se introducen

en la entrada de la red pasan a través de las capas ocultas hasta llegar a la capa de salida, dando un vector de salida por cada vector de entrada.

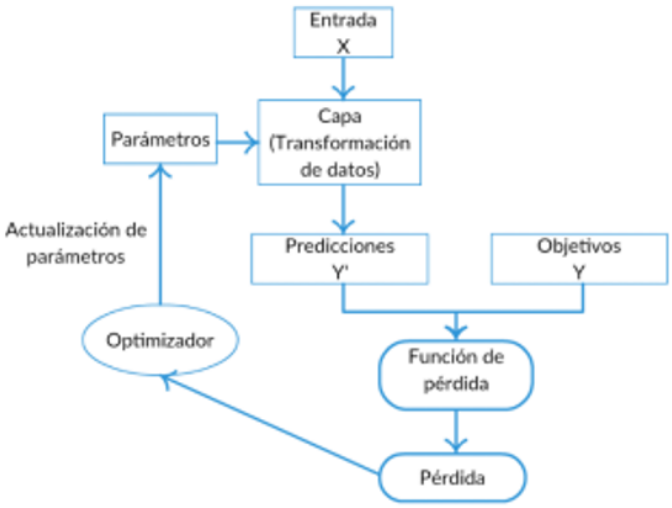


Figura 2.11 : Diagrama del proceso de aprendizaje

Después se calcula la función de coste utilizando el valor de salida obtenido con el valor de salida deseado y mediante optimizadores y la aplicación del algoritmo de propagación hacia atrás se modifican los valores de parámetros internos de la red.

Este proceso se va repitiendo hasta alcanzar un punto donde el aprendizaje se haya optimizado (Figura 2.11).

La selección de muestras influye significativamente en el proceso de aprendizaje (véase un ejemplo en la Figura 2.12). Las dos causas principales al no obtener buenos resultados en el aprendizaje son:

- **Bajo ajuste:** Es el resultado de un aprendizaje donde el algoritmo cuenta con pocas muestras para llegar a extraer características.
- **Sobreajuste:** Al contrario que el anterior, el algoritmo cuenta con suficientes muestras, pero estas muestras tienen poca variación entre ellas. Esto provoca que la red empiece a memorizar la muestra y no trabaje bien con elementos nuevos. Se suele denominar *overfit* por su traducción en inglés.

Por ello, es necesario encontrar un punto intermedio entre estas dos situaciones. En problemas de clasificación es necesario tener variedad en las muestras de las diferentes clases, además de estar equilibradas en cantidad de datos de cada clase.

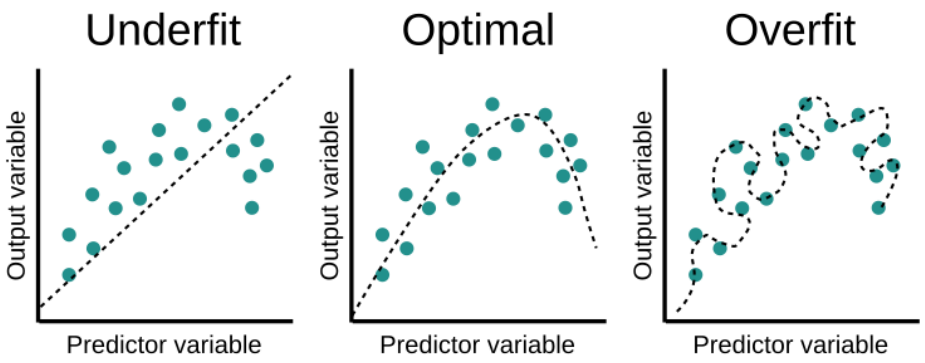


Figura 2.12 : Comparativa entre diferentes resultados de entrenamientos

Ahora se comentarán ciertos **hiperparámetro**, es decir, son variables que se definen fuera del modelo, a diferencia de los parámetros que eran referidas a los pesos y sesgo. Los hiperparámetros permiten ajustar el aprendizaje.

- ***Tasa de aprendizaje***

Este hiperparámetro fue definido anteriormente y se encarga de definir cuanto será el cambio en los parámetros de la red durante el aprendizaje. Comúnmente denominado con su traducción en inglés, *learning rate*.

- ***Momentum***

Otro hiperparámetro que fue comentado anteriormente en la etapa de optimización. Permite escapar de mínimos locales pequeños.

- ***Numero de épocas***

Es el número de veces que se van a utilizar todos los datos de entrenamiento. La métrica para definir el número de épocas es viendo cuando comienza el sobreajuste. Este momento se produce cuando la exactitud con los datos de validación empieza a bajar y el error producido empieza a subir.

- ***Tamaño de lote***

Comúnmente utilizado mediante su nombre en inglés (*batch size)*, es el número de datos que se utiliza en cada iteración de una época o ciclo.

- ***Inicialización de parámetros de la red***

Los valores de los pesos y sesgos al iniciar la red previa a ningún entrenamiento.

Conjunto de datos:

- Datos de entrenamiento: sirven para adaptar los parámetros de la red, pesos y sesgos de las capas neuronales.

- Datos de validación: sirven para comprobar cómo funciona la red con datos diferentes a los usados para entrenar, y adaptar los hiperparámetros para obtener los mejores resultados. No adaptan los pesos ni los sesgos
- Datos de prueba: se utilizarán al finalizar todo el entrenamiento y evaluar el algoritmo.

Métricas:

Las funciones métricas permiten contemplar la evolución y resultado del entrenamiento de una red. Su finalidad es similar a la función error excepto que esta no se utiliza para entrenar el modelo.

2.1.8 Tipos de aprendizaje

- **Aprendizaje supervisado**

El aprendizaje se realiza mediante el uso de datos de ejemplo. Se utiliza unos valores de entrada a la red y mediante la salida predicha por red, se compara y se ajusta la red con la respuesta correcta que debería de dar. Este es el aprendizaje que se utilizara en este TFG.

- **Aprendizaje no supervisado**

Este tipo de aprendizaje se diferencia del anterior al no disponer o no utilizar datos de salida sino solo de entrada a la red. También es el aprendizaje más relacionado con el aprendizaje profundo, ya que en estas redes el objetivo consiste en agrupar los datos de manera que se puedan catalogar en base a las descripciones de las características.

Una arquitectura muy famosa en este sector son los autocodificadores (*autoencoder*), una red que logra disminuir la señal de entrada a un vector de tamaño más limitado e intentar recuperar la señal original a la salida (Figura 2.13). Este proceso cuenta con dos redes diferentes, una especializada en codificar y la segunda en decodificar. Esto permite crear métodos potentes de compresión de datos.

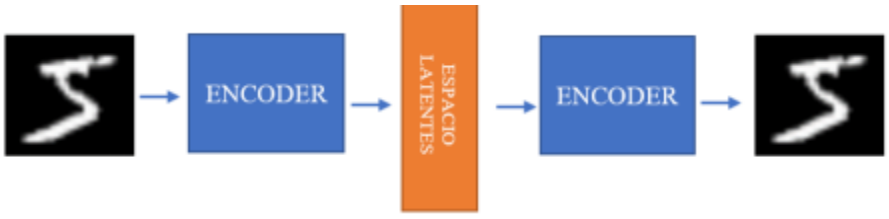


Figura 2.13: Funcionamiento de un autoencoder

- **Aprendizaje reforzado (Q-learning)**

Es un aprendizaje enfocado en la prueba y error inspirada en la psicología conductista. Se define un agente y un entorno donde el agente puede realizar una acción e influir en el entorno. Al realizar esta acción en el estado, el agente obtiene el estado del entorno y una recompensa, la cual puede ser positiva o negativa según el objetivo que se desee lograr. El agente va aprendiendo a tomar decisiones a partir del estado actual y la recompensa que

recibirá al pasar el siguiente estado. Afianzando las decisiones que le llevan a la mayor recompensa.

2.1.9 Tipos de redes

A lo largo de los últimos años se han ido diseñando estructuras diferentes de redes para solventar diversos tipos de problemas. En este apartado se comentarán algunas de las más famosas y útiles.

2.1.9.1 Red neuronal prealimentada clásica

En la práctica existe numerosas maneras de estructurar la red. La más sencilla es la red neuronal prealimentada (*feed-forward* en inglés), donde los nodos de cada capa se conectan directamente a todos los nodos de la capa siguiente, transmitiendo la información en una sola dirección, hacia delante. Este tipo de capa también se denomina *full-connected* o capa densa.

Función final:

$$Y = f_k(W_k \cdot (f_{k-1}(W_{k-1} \cdot (\dots f_1(W_1 \cdot X + B_1) \dots) + B_{k-1})) + B_k) \qquad (2.50)$$

2.1.9.2 Las redes recurrentes

Este tipo de red se caracteriza por utiliza la salida de cada neurona como entrada de esta misma en la siguiente interacción de datos. Esto le confiere a la red una especie de memoria, la cual permite analizar secuencias de datos.

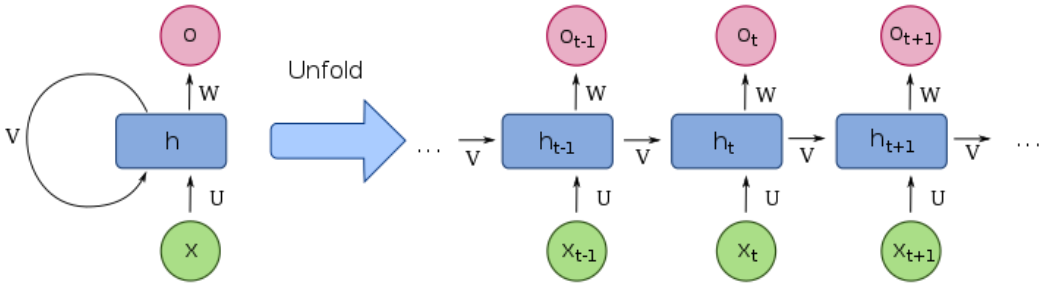


Figura 2.14 : Proceso de desenrollar una red recurrente

Un ejemplo de este tipo de red seria las redes Jordan:

$$h_t = f_h(W_h \cdot x_t + V_t y_{t-1} + b_h) \qquad (2.51)$$

$$y_t = f_y(W_y \cdot h_t + b_y) \qquad (2.52)$$

Un problema que existe con este tipo de redes es que, a mayor secuencia, mayor será el número de capas a "desenrollar" (Figura 2.14). Lo que puede llegar a producirse desvanecimiento del gradiente.

Actualmente, existen numerosas variaciones de este tipo de redes para prevenir este efecto como las redes LSTM (memorias a largo corto plazo) (Figura 2.15) o GRU (unidades recurrentes cerradas) (Figura 2.16).

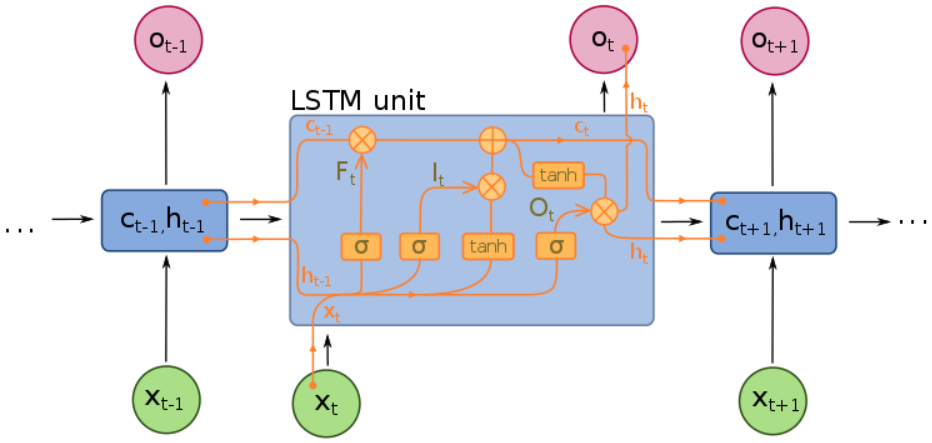


Figura 2.15: Unidad LSTM

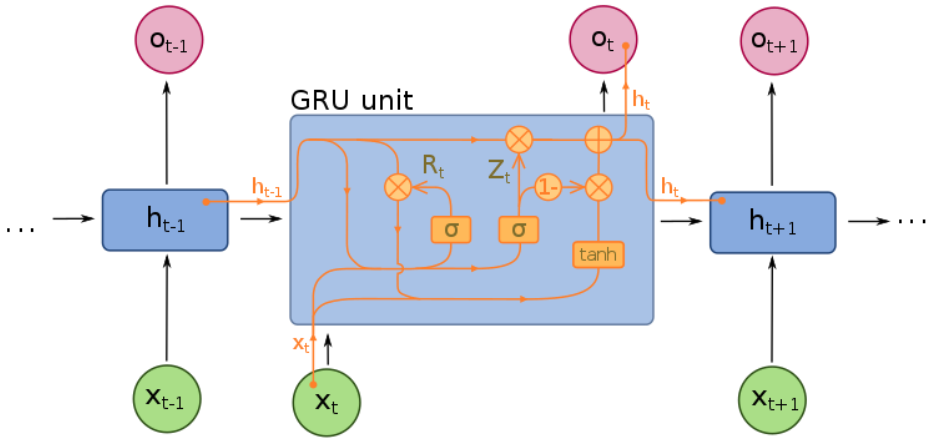


Figura 2.16 : Unidad GRU

Este tipo de red la hace aplicable en tareas como reconocimiento de voz, reconocimiento de escritura, traducción, predicciones temporales, síntesis de voz.

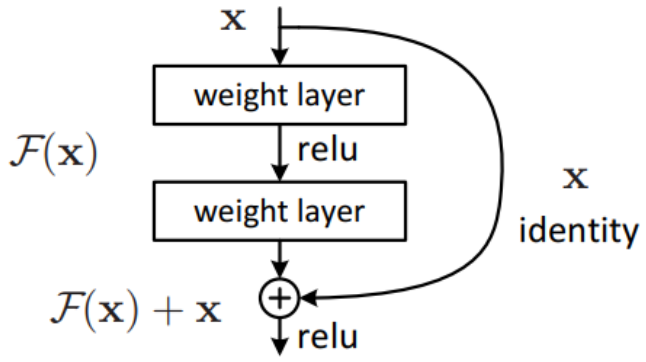


Figura 2.17: Estructura de un salto de capa

Fuente: https://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf

2.1.9.3 Las redes residuales

El problema principal de redes con gran longitud es el desvanecimiento del gradiente. Esto se produce porque el gradiente va disminuyendo capa a capa y por ello las primeras capas son más lentas de entrenar hasta a veces directamente llegar a impedir su entrenamiento. Por ello, se añadió conexiones entre capas que permitían saltar en paralelo capas intermedias (Figura 2.17), inspirado en las neuronas piramidales de la corteza cerebral.

Este simple mecanismo permitió crear y entrenar redes de profundas dimensiones.

2.1.10 *Redes neuronales convolucionales*

Este tipo de redes se caracteriza por contar con un tipo especial de capas, capas de convolución. Este tipo capa tiene un diseño pensado para sacar provecho a la estructura espacial de una imagen. En una red convencional, todas las variables de entradas son tratadas como variables independientes, pero en una convolucional se busca la relación entre variables cercanas (pixeles). Dando así una importancia a la posición que ocupa en una imagen.

Una convolución es la aplicación de una matriz, de reducido tamaño (3x3, 5x5) sobre la imagen, esta matriz operará con el valor de un píxel y sus vecinos para generar un nuevo píxel (véase como ejemplo la Figura 2.18). De esta manera, se formará una nueva imagen al aplicar el mismo filtro en todos los pixeles de la imagen. Al filtro se le suele llamar *kernel* (núcleo en inglés).

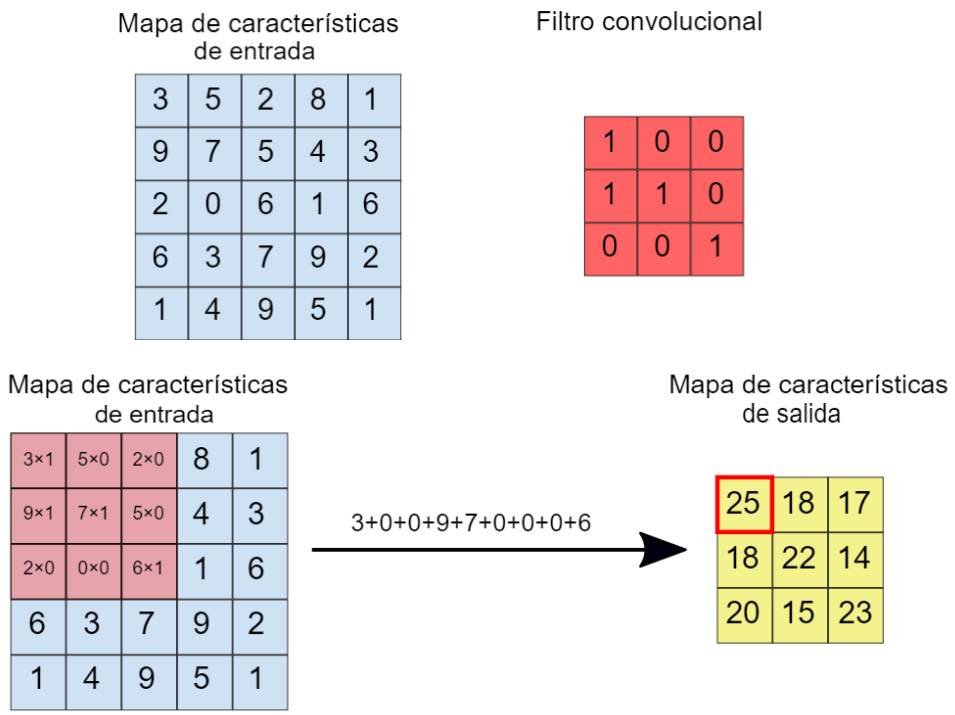


Figura 2.18 : Proceso de convolución

Fuente: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks?hl=es-419

En función de la configuración de los parámetros del filtro generará una imagen donde se puede lograr enfatizar los bordes, bordes horizontales o verticales, realizar un desenfoque o un realce. La imagen generada se llama mapa de características, puesto que la imagen obtenida indica las zonas de la imagen donde se ha detectado la característica determinada por el filtro.

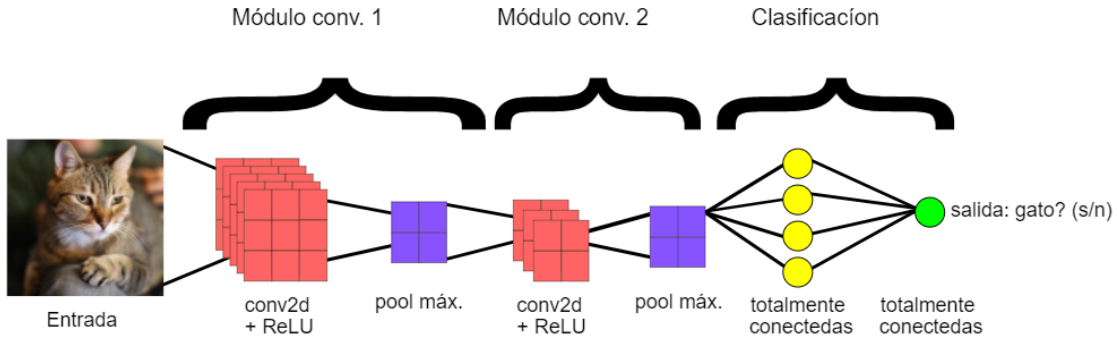


Figura 2.19 : Esquema de una red neuronal convolucional de clasificación

Fuente: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks?hl=es-419

En resumen, la imagen entra a diferentes filtros y se obtienen varios mapas de características o de patrones. Al realizar otra convolución en las nuevas imágenes se pueden encontrar patrones más complejos. Así se consiguen detectar en las primeras capas patrones básicos y generales, para después combinarlas y detectar elementos más complejos en capas posteriores.

Después de realizar las convoluciones, la información generada se introduce en una red multicapa convencional (Figura 2.19) para tomar la decisión de qué objeto es (problema de clasificación).

Los pesos y el sesgo de estas capas neuronales son los propios filtros. Esto implica que en cada capa solo hay que entrenar un vector de pesos y un solo sesgo, en lugar de que cada campo receptivo tenga su propia ponderación vectorial y sesgo.

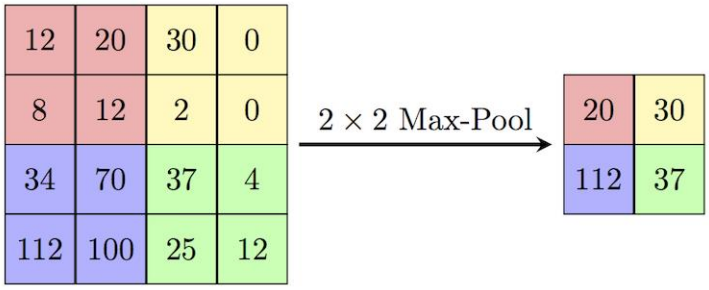


Figura 2.20: Proceso de MaxPooling

Para optimizar este proceso, a cada nueva imagen generada se le puede reducir la resolución de la imagen mediante la técnica de MaxPooling. Esta técnica permite reducir la dimensión

de una matriz mediante un proceso que aplica un filtro que toma del valor máximo de la región en la que opera (Figura 2.20). Este filtro se pasa por toda la matriz, hasta formar una nueva matriz con los valores máximos de cada región filtrada. Esto permite reducir el coste computacional al reducir el número de parámetros de la red.

2.2 CoppeliaSim

En este proyecto se usa CoppeliaSim como entorno de simulación y prueba de las redes neuronales desarrolladas aplicadas a un sistema de clasificación robotizado de propósito general.

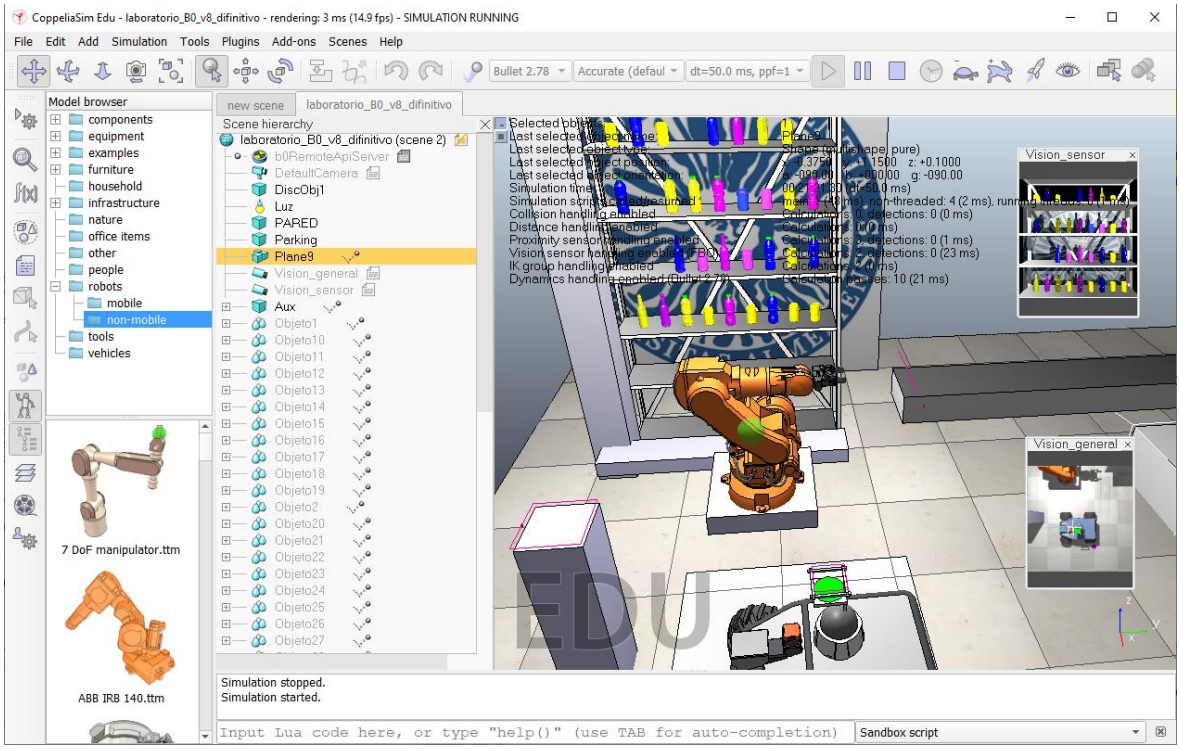


Figura 2.21 : Pantalla de CoppeliaSim con el modelo 3D desarrollado para el sistema robotizado

2.2.1 Pantalla de CoppeliaSim

En este apartado se aborda la distribución y la utilidad que ofrecen los diferentes elementos de la interfaz gráfica del simulador. La pantalla de CoppeliaSim inicia con la ventana de la escena de trabajo, la jerarquía de la escena con los diferentes elementos de escena, un explorador de modelos donde se puede seleccionar diferentes modelos predefinidos, una barra de estado y barras de herramientas como se puede observar en la Figura 2.21.

- ***Jerarquía de la escena***

Muestra todos los objetos de la escena, tales como sensores, actuadores, objetos estáticos, fuentes luz, etc. Dado que los objetos constituyen una estructura parecida a una jerarquía, es más simple de representar robots o elementos compuestos. Esta zona se muestra como un árbol con los elementos individuales de la escena (Figura 2.22).

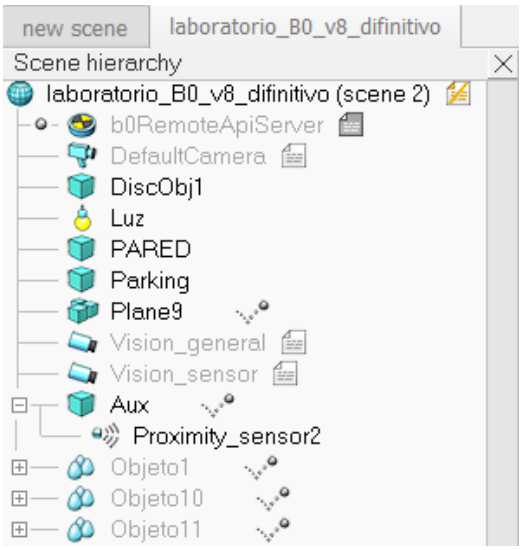


Figura 2.22 : Jerarquía de la escena

- ***Navegador de modelos***

Muestra la biblioteca de objetos disponibles ordenados por carpetas. En la versión Educativa (usada durante el desarrollo del TFG) cuenta con sensores, algunos modelos de robots reales, herramientas, muebles, etc (véase el ejemplo de la Figura 2.23). También es posible implementar modelos y guardarlos en el navegador para poder así usarlos en diferentes escenas.

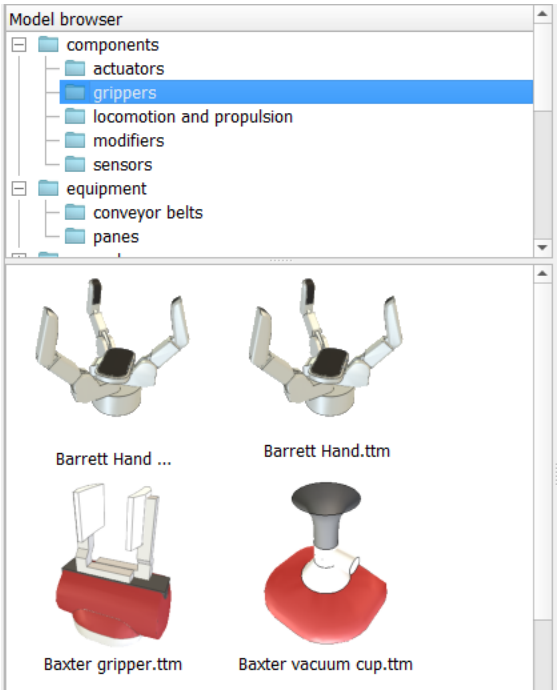


Figura 2.23 : Navegador de modelos

- ***Barras de herramientas***

Se incluyen las funciones a las que se va a recurrir con frecuencia. En la primera barra (Figura 2.24) se incluyen opciones como traslación o rotación de vista de escena, traslación o rotación de objetos, opciones de motor de física, opciones de reproducción de simulación.

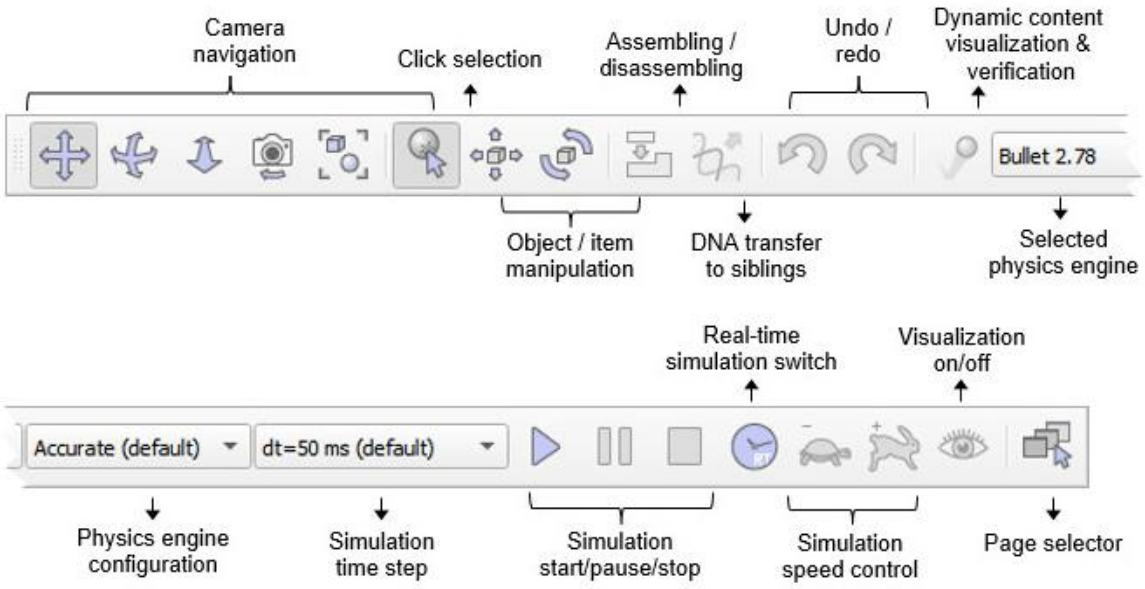


Figura 2.24 : Barra de herramientas 1

Fuente: https://www.coppeliarobotics.com/helpFiles/en/userInterface.htm

En la segunda barra (Figura 2.25) se incluyen opciones más específicas como las propiedades de simulación o las propiedades de objetos. Algunos botones abren un cuadro de dialogo como en el caso de la traslación de un objeto, donde es posible editar los parámetros de posición.

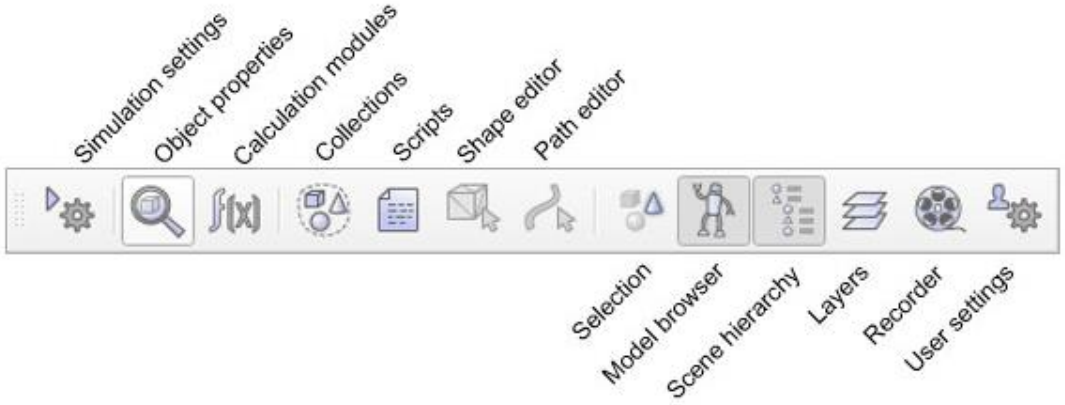


Figura 2.25 : Barra de herramientas 2

Fuente: https://www.coppeliarobotics.com/helpFiles/en/userInterface.htm

- ***Página***

Zona donde se realiza la simulación y el modelado. Es una superficie donde se realiza una visualización general de la escena. Puede contener una o varias vistas de la escena mediante una posición fija en la página o una posición flotante, como se puede ver en la Figura 2.26. Esta página puede ser personalizada por el usuario.

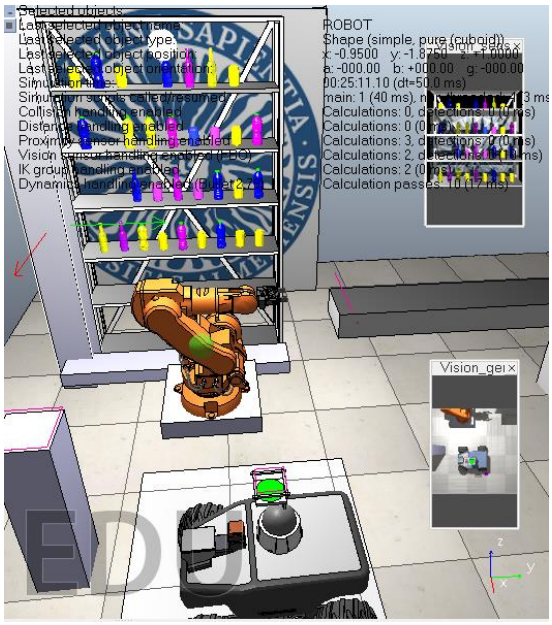


Figura 2.26 : Pagina de CoppeliaSim formada por una vista general fija y dos flotantes que representan la imagen de las camaras

- ***Barra de estado***

Aquí se muestra información de los scripts, de los errores y comandos. El usuario puede contener algún script que envíe mensajes mediante cadenas de caracteres a la barra de estado, por ejemplo, para notificar algún error.

2.2.2 Objetos de escena

Son los elementos que componen una escena de simulación de CoppeliaSim (Figura 2.27). En este apartado se explicará los que se van a usar en el proyecto.

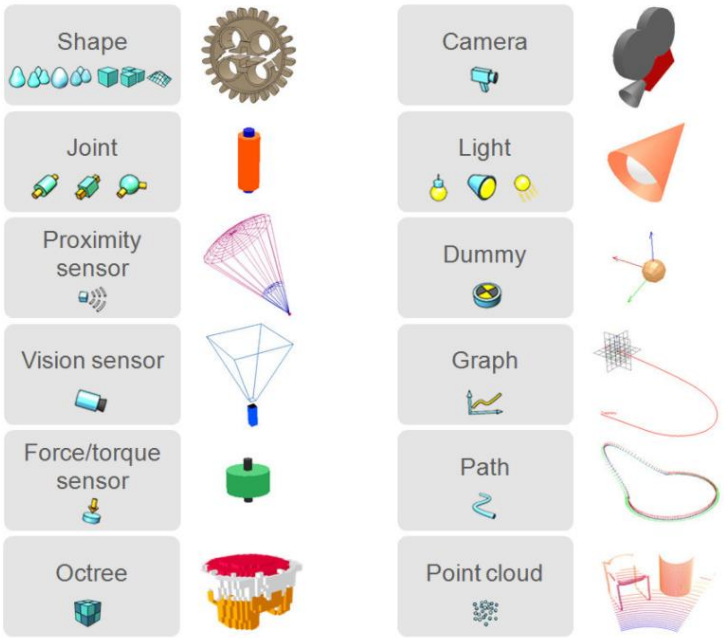


Figura 2.27 : Tipos de objetos de escena

Fuente: https://www.coppeliarobotics.com/helpFiles/en/objects.htm

- ***Formas***

Son objetos de malla rígida formada por caras triangulares, que se pueden crear, importar, exportar y editar.

Las formas que se pueden crear se hacen añadiendo formas primitivas desde [Barra de menús → Add(añadir) → Primitive Shapes]. Las formas admitidas son: plano, disco, cuboide, esfera y cilindro

Los parámetros de las formas primitivas se ajustan en el dialogo de forma primitiva como las dimensiones de la forma, en número de lados y cara (cilindros o esfera), sombreado, cono (en vez de cilindro), forma dinámica y responsable, forma pura (funciona más rápido y mejor para los cálculos dinámicos), densidad del material.

Tipos de formas:

- **Forma aleatoria simple:** representa cualquier malla que tiene un color y conjunto de atributos visuales, no está recomendado ni optimizado para el cálculo dinámico a respuesta de colisiones.
- **Forma aleatoria compuesta:** formado por varias formas aleatorias simples. Varios colores y atributos visuales.
- **Forma convexa simple:** representa una malla convexa que tiene un color y conjunto de atributos visuales, optimizado para el cálculo dinámico a respuesta de colisiones.
- **Forma convexa compuesta:** formado por varias formas convexas simples. Varios colores y atributos visuales.
- **Forma pura simple:** representa una forma primitiva, es la más adecuada para el cálculo dinámico a respuesta de colisiones.
- **Forma pura compuesta:** formado por varias formas puras simples.
- **Forma de campo de altura:** es una cuadricula regular que puede representar un terreno donde cambian las alturas. Estos campos de altura también se pueden ser considerados como formas puras simples por lo que están optimizados para el cálculo dinámico.

Todas estas formas se pueden agrupar, desagrupar, fusionar y dividir.

- ***Articulaciones***

La articulación es la forma de unir dos elementos, proporcionando un movimiento relativo entre estos objetos. El movimiento depende del tipo de articulación, que puede ser:

- **Articulación revolucionaria:** un grado de libertad y describen un movimiento de rotación. Puede actuar como junta pasiva o activa.

- **Prismático:** un grado de libertad y describen un movimiento de traslación. Puede actuar como junta pasiva o activa.
- **Tornillo:** dos grados de libertad, movimiento combinado de junta prismática y junta articulada, similar a un tornillo (fusión de los dos anteriormente mencionados). Puede actuar como junta pasiva o activa.
- **Esférica:** tres grados de libertad y movimiento de rotación en sus tres ejes. Solo puede actuar como junta pasiva.

Modo de funcionamiento:

- **Pasivo:** La articulación no se controla directamente y actuará como enlace fijo. Sin embargo, el usuario puede cambiar su posición mediante llamada a las funciones API apropiadas.
- **Cinemática inversa:** La articulación actúa como articulación pasiva, pero ajusta sus parámetros durante los cálculos de cinemática inversa.
- **Dependiente:** La posición de la articulación está ligada a la posición de otra articulación mediante una función lineal.
- **Modo de movimiento (En desuso):** Similar al modo pasivo, pero más flexible. Un script intermedio actualiza los parámetros de la posición de la articulación
- **Modo Torque o Par:** La articulación es simulada de forma dinámica. Puede ser controlada en fuerza mediante un motor o estar libre (restringida solo por sus límites). Cuenta con la posibilidad de utilizar un circuito de control PID que ajusta la velocidad de la articulación o mediante un modo resorte-amortiguador mediante una modulación fuerza/par.

- ***Cámara***

Es un objeto que permite mostrar una vista de una parte concreta de la escena. Se puede confundir con los sensores de visión, las diferencias principales son que la cámara no tiene una resolución específica, el contenido no está directamente disponible a través de una API, puede mostrar todo tipo de objeto en escena y funciona más rápido. La vista predeterminada de la escena es una cámara. Puede funcionar con OpenGL, OpenGL 3, POV-Ray u otras aplicaciones de renderizado externas.

- ***Luces***

Iluminan los objetos de la escena. Influyen en la imagen captada por la cámara o los sensores de vista. Existen tres tipos.

- *Omnidireccional*, que ilumina la escena en todas las direcciones desde una fuente de luz.

- *Foco*, que ilumina en una dirección ajustando el tamaño del cono de luz.

- *Direccional*, donde solo la dirección importa y todos los objetos de la escena se iluminará de forma similar

- ***Dummy***

El objeto más simple que puede considerarse. Se trata de un punto que puede usarse para la detección de colisiones con otros objetos, cálculo de distancias mínima y puede ser detectado por sensores de proximidad.

- ***Sensor de proximidad***

Este objeto permite la detección de objeto dentro de un línea, superficie o volumen.

2.2.3 API REMOTA

Una de las funcionalidades más potentes que integra el simulador CoppeliaSim es la API remota. Permite utilizar distintos lenguajes de programación para comunicar los modelos del simulador con aplicaciones externas (Figura 2.28).

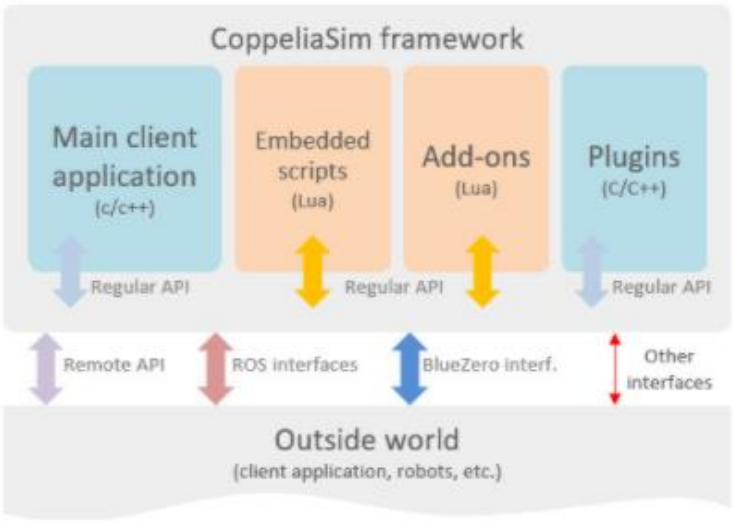


Figura 2.28 : Comunicación de CoppeliaSim con otras aplicaciones

Fuente: https://www.coppeliarobotics.com/helpFiles/en/apisOverview.htm

Existen varias versiones:

2.2.3.1 The legacy remote API:

Esta versión es más liviana y tiene menos dependencias comparativamente con la API basada en B0. Sin embargo, pierden en intuición, flexibilidad y facilidad de ampliar. Funciona en C/C++, Java, Python, MatLab, Octave y Lua.

2.2.3.2 The B0-based remote API:

La version que se utiliza en este proyecto. Basado en el *middleware* (vía que permite la conexión de dos aplicaciones y transmisión de datos entre ellas) *BlueZero* y su complemento

de interfaz para CoppeliaSim. Mas fácil y flexible de usar, y fácil de extender que la versión *legacy remote API*.

La API remota B0 se divide en dos entidades [23] :

- Servidor: el lado del simulador CoppeliaSim. La interfaz se implementa a través de un complemento de CoppeliaSim y un script Lua. Este complemento se debe cargar al inicio de CoppeliaSim.
- Cliente: el lado de la aplicación donde se desarrolla el programa principal.

Funciones usadas en el proyecto y explicación [24] :

- ***simxServiceCall***

La mayoría de las funciones de la API remota B0 requieren un argumento adicional: el canal de comunicación o elemento que se utilizara para llamar a la función. Esta función se usará a menudo como elemento de llamada. Mayormente en situaciones donde se requiere una respuesta.

Por ejemplo, en la función anterior, donde se pide el identificador de un objeto de la escena, el estado de algún sensor o la imagen de una cámara. El tema permite ejecutar la función en modo bloqueo. El comando será enviado a CoppeliaSim, se ejecutará y devolverá una respuesta al cliente.

- ***simxDefaultPublisher***

Esta es otra función de relacionada con el canal de comunicación. A diferencia de la anterior, esta función realiza la comunicación sin bloqueo. Remite la orden al servidor y devuelve el control al cliente. Sera usada cuando no será necesario una respuesta, por ejemplo, al mandar una orden de posicionamiento de una articulación.

- ***simxGetObjectHandle***

Al programar en CoppeliaSim será necesario referenciar a varios objetos. Esta función es encargada de devolver el identificador del objeto según su nombre.

Parámetros de entrada:

- **NombreObjeto (string):** nombre del objeto llamado.
- **Tema (string):** el tema o canal de comunicación que se utilizara al llamar la función. Se utilizará simxServiceCall ya que requiere esperar una respuesta.

Devuelve una lista:

- **Objeto 1 (bool):** si la llamada ha sido exitosa.
- **Objeto 2 (numero):** el identificador del objeto.

- ***simxGetJointPosition***

Función encargada de recuperar el valor de la posición intrínseca de una articulación.

Parámetros de entrada:

- **Identificador de articulación (numero):** número del identificador de la articulación llamada.
- **Tema (string):** el tema o canal de comunicación que se utilizara al llamar la función. Se utilizará simxServiceCall.

Devuelve una lista:

- **Objeto 1 (bool):** si la llamada ha sido exitosa.
- **Objeto 2 (numero):** posición de la articulación, en metros o radianes.

- ***client.simxSetJointTargetPosition***

Función encargada de establecer la posición de destino de una articulación.

Parámetros de entrada:

- **Identificador de articulación (numero):** número del identificador del actuador llamado.
- **Posición objetivo (numero):** la posición de destino.
- **Tema (string):** el tema o canal de comunicación que se utilizara al llamar la función. Se utilizará simxDefaultPublisher porque no requiere una respuesta.

Devuelve:

- **Objeto 1 (bool):** si la llamada ha sido exitosa.

- ***simxSetJointTargetVelocity***

Función encargada de establecer la velocidad objetivo de una articulación.

Parámetros de entrada:

- **Identificador de articulación (numero):** número del identificador del actuador llamado.
- **Velocidad (numero):** velocidad objetivo en rads/s o m/s.
- **Tema (string):** el tema o canal de comunicación que se utilizara al llamar la función. Se utilizará simxDefaultPublisher.

Devuelve:

- **Objeto 1 (bool):** si la llamada ha sido exitosa.

- ***simxSetObjectPosition***

Establece la posición de un objeto. La referencia de la posición puede ser absoluta o referida a otro objeto de la escena.

Parámetros de entrada:

- **Identificador del objeto (numero):** número del identificador del objeto llamado.
- **Identificador del objeto relativo (numero):** nombre del objeto al que relaciona el objeto a posicionar. Si se desea una posición en el marco de referencia absoluta se debe dejar en -1.
- **Posición(vector):** la posición deseada (x,y,z).
- **Tema (string):** el tema o canal de comunicación que se utilizara al llamar la función. Se utilizará simxDefaultPublisher.

Devuelve:

- **Objeto 1 (bool):** si la llamada ha sido exitosa.

- ***simxSetObjectOrientation***

Establece la orientación (en ángulos de Euler) de un objeto. Al igual que con la posición, la referencia de la orientación puede ser absoluta o referida a otro objeto de la escena.

Parámetros de entrada:

- **Identificador del objeto (numero):** número del identificador del objeto llamado.
- **Identificador del objeto relativo (numero):** nombre del objeto al que relaciona el objeto a orientar. Para referencia absoluta se debe dejar en -1.
- **Posición(vector):** la orientación deseada (alpha, beta, gamma).
- **Tema (string):** el tema o canal de comunicación que se utilizara al llamar la función. Se utilizará simxDefaultPublisher.

Devuelve:

- **Objeto 1 (bool):** si la llamada ha sido exitosa.

- ***simxReadProximitySensor***

Devuelve el resultado del cálculo del sensor de proximidad.

Parámetros de entrada:

- **Identificador del sensor (numero):** número del identificador del sensor llamado.
- **Tema (string):** el tema o canal de comunicación que se utilizara al llamar la función. Se utilizará simxServiceCall.

Devuelve:

- **Objeto 1 (bool):** si la llamada ha sido exitosa.
- **Objeto 2 (numero):** estado del sensor, si detecta o no.
- **Objeto 3 (numero):** distancia de detección.
- **Objeto 4 (lista):** el punto detectado relativo al sensor.
- **Objeto 5 (numero):** número de identificación del objeto detectado.
- **Objeto 6 (lista):** el vector normal de la superficie detectada, también relativo al sensor.

- ***simxCopyPasteObjects***

Para nuestra escena los objetos originales que se encuentran en la caja serán duplicados y sus copias colocadas en el escenario. Para la acción de duplicado se utilizará esta función.

Parámetros de entrada:

- **Identificadores (lista):** lista con los valores de la identificación de los objetos a duplicar.
- **Opciones(bit):** si se establece el bit0 se copiarán los modelos completos.
- **Tema (string):** el tema o canal de comunicación que se utilizara al llamar la función. Se utilizará simxServiceCall.

Devuelve:

- **Objeto 1 (bool):** si la llamada ha sido exitosa.
- **Objeto 2 (lista):** los valores de identificaciones de los objetos duplicados.

- ***simxRemoveObjects***

Esta función está diseñada para eliminar modelos u objetos de la escena.

Parámetros de entrada:

- **Identificadores (lista):** lista con los valores de la identificación de los objetos a eliminar.
- **Opciones(bit):** si se establece el bit0 se eliminaran los modelos completos y bit1 , todos los objetos y modelos.
- **Tema (string):** el tema o canal de comunicación que se utilizara al llamar la función. Se utilizará simxServiceCall.

Devuelve:

- **Objeto 1 (bool):** si la llamada ha sido exitosa.
- **Objeto 2 (lista):** número de objetos eliminados.

- ***simxCallScriptFunction()***

Función que realiza una llamada a un script dentro del simulador. Este script puede ser creado por el usuario o venir con un modelo insertado en la escena.

Parámetros de entrada:

- **Nombre de script (string):** una cadena de letras que representa el nombre de la función y el nombre del objeto del script al que está asociado.
- **Tipo de script (número o cadena):** especificado como numero o cadena.
- **Argumentos de función (lista):** valores de entrada a la funcion que se van a enviar al script.
- **Tema (string):** el tema o canal de comunicación que se utilizara al llamar la función. Se utilizará simxServiceCall.

Devuelve:

- **Objeto 1 (bool):** si la llamada ha sido exitosa.
- **Objeto 2 ():** el primer valor que devuelve el script.

- ***simxStartSimulation***

Función que permite iniciar la simulación.

- ***simxStopSimulation***

Función que permite parar la simulación.

- ***simxGetSimulationState***

Esta función devuelve el estado de la simulación. El primer valor que devuelve es si se ha podido realizar la llamada de forma exitosa. Seguidamente está el estado; 0 si esta parada, 8 si esta pausada o 16 si está corriendo.

2.3 Python

En este proyecto se utiliza principalmente el lenguaje de programación Python y sus librerías por las siguientes características que ofrece:

- Es un lenguaje de programación de alto nivel, una de sus características más notable es su claridad y legibilidad en la escritura del código. Las variables se definen de forma dinámica, es decir, no se debe especificar el tipo de variable antes de asignarle el valor.
- Compatible con muchos sistemas operativos actuales, se puede ejecutar el mismo código en diferentes plataformas sin la necesidad de volver a compilarlo.
- Como muchos lenguajes de programación modernos, Python es un lenguaje multiparadigma. Esto significa que permite adoptar varios estilos de programación: programación orientada a objetos, programación imperativa y programación funcional.

Se ha popularizado en el campo de las redes neuronales y cuenta con numerosas librerías para su fácil implementación. Actualmente es uno de los lenguajes más populares, por su gran versatilidad, que además cuenta con una comunidad muy activa que sirve de apoyo

Sin embargo, un factor importante para tener en cuenta como punto negativo es su lenta ejecución en comparación, por ejemplo, con Java o C++.

2.3.1 Anaconda

Anaconda es una distribución de código abierto que permite aplicar técnicas de datos y aprendizaje automático utilizando lenguaje Python /R. Su principal objetivo es simplificar la gestión de paquetes y su utilización. Cuenta con una interfaz gráfica de usuario (GUI) llamada Anaconda Navigator la cual permite a los usuarios usar la interfaz de comandos, iniciar aplicaciones y administrar paquetes, entornos y canales conda [25] .

Entre estas aplicaciones se incluye Jupyter Notebook, el entorno computacional donde se crea el código Python principal del programa.

2.3.2 TensorFlow

Google se introdujo en el mundo de las redes neuronales cuando un grupo de investigación llamado Google Brain crearon DistBelief, un sistema de aprendizaje automático basado en redes neuronales. Después, Google asignó mejorar la base del código de DistBelief, creando una biblioteca más robusta y rápida, llamada TensorFlow, lanzando en 2017 su versión 1.0.0 [26].

Trabaja con Python y C proporcionando APIs estables, aunque también hay compatibilidad con otros lenguajes como Java, JavaScript, C++, Go y Swift y paquete de terceros con C# o MatLab. Google la utiliza en muchos de sus productos como el reconocimiento de voz, Gmail, Google Photos y la búsqueda de Google [27].

2.3.3 Keras

Es una librería de redes neuronales de alto nivel capaz de implementar directamente redes neuronales especificando las capas, logrando un desarrollo más intuitivo y sencillo. Una herramienta perfecta para la creación rápida de prototipos. Desde TensorFlow 2.0 se incluye Keras dentro de su biblioteca [28] .

La implementación se puede realizar mediante la adición de capas utilizando las funciones **add()** y **Sequential()**.

Funciones más usadas [29] [30] :

- **tf.keras.layers.Convolution2D()** y **tf.keras.layers.Conv2D():** función destinada a la creación de una capa de convolución. El parámetro *filters* determinan el número (entero) de filtros que se van a realizar, *kernel_size* es la dimensión de los filtros.

También cuenta con la función de activación que se va a utilizar definida como *activation*.

- **tf.keras.layers.MaxPooling2D():** función destinada a la creación de una capa de *MaxPooling*. El parámetro de tamaño de la matriz encargada de la agrupación se define en *pool_size*. *Strides* define el salto que realiza durante el proceso de pooling, si no es definido o definido como *none*.
- **tf.keras.layers.Activation():** función destinada a la creación de una capa para aplicar la función de activación que define una cadena de caracteres. A la salida mantiene la misma dimensión que la entrada.
- **tf.keras.layers.Flatten():** función encargada de crear la capa de aplanamiento. Esta capa convierte una entrada de varias dimensiones en un solo vector. Se utiliza previamente a la capa densa.
- **tf.keras.layers.Dense():** función encargada de crear una capa regular conectada densamente. Implementa la operación: *output = activation(dot(input, kernel) + bias)*. Entre los parámetros a definir, contiene el tipo de función de activación, el número de neuronas (*units*), valores de inicialización de los pesos y los sesgos.
- **tf.keras.layers.Dropout():** función encargada de crear una capa de abandono. Esta capa establece aleatoriamente las unidades de entrada en valor 0 (Figura 2.29) con la frecuencia que se establezca como parámetro. Esta capa solo está en funcionamiento durante el entrenamiento y su finalidad es optimizar el entrenamiento.

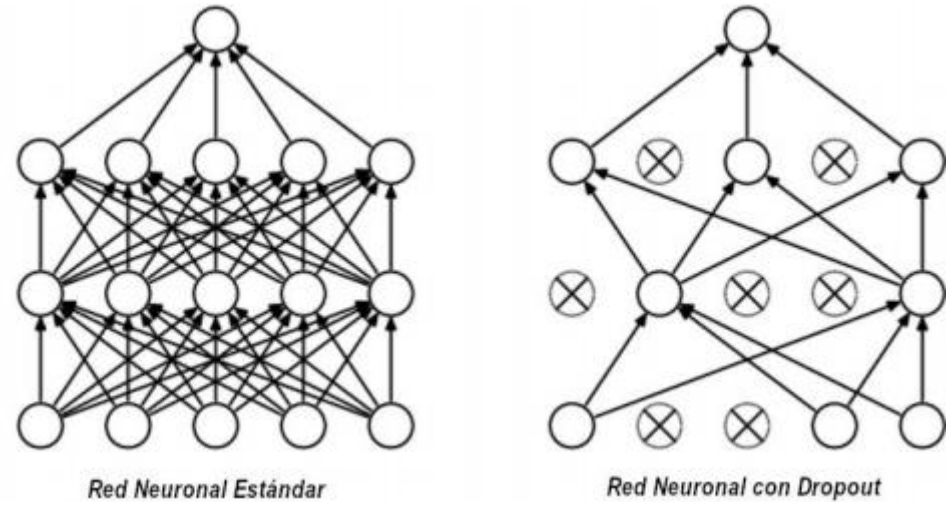


Figura 2.29 : Efecto de capa Dropout

- **tf.keras.Sequential():** sirve para crear una red neuronal mediante la agrupación de una pila de capas. Se pueden definir la capas a la hora de definir esta función, con las capas como argumento o después con la función *add().*
- **add():** función que sirve para añadir una capa (*tf.keras.layers*) a la pila de capas. Se aplica a una red previamente creada y como argumento se introduce la capa que le va a añadir.

- **compile():** función encargada de compilar el modelo después de haber sido definido su arquitectura. Se define la función de error, el optimizador y la métrica en los argumentos.
- **fit():** función encargada de realizar el ajuste de pesos y los sesgos de la red. Los valores de entrenamiento de ejemplo (*x_train, y_train*) se definen como argumento de la función como también el número de épocas y el tamaño de lote.
- **summary():** función encargada de dar el resumen de la arquitectura de la red. Esta función es meramente informativa.
- **predict():** función encargada de realizar la predicción de una entrada mediante una red neuronal. Como argumento se debe ingresar un vector de entradas donde devolverá un vector de misma longitud de salidas. Es por ello por lo que será necesario expandir con una dimensión adicional la entrada cada vez que se va a utilizar una predicción**.**

2.3.4 OpenCV

Es una biblioteca de código abierto de aprendizaje automático y visión artificial. Contiene numerosos algoritmos como por ejemplo detección de rostros, identificación de objetos, clasificación de acciones humanas, rastreo de movimientos de objetos, extracción de modelos 3D, etc . Fue originalmente desarrollada por Intel en junio de 2000. Esta biblioteca está escrita en C++ y su interfaz principal también, pero cuenta con enlaces con otros lenguajes como Python, Java y MATLAB. También es un lenguaje ejecutable en diferentes sistemas operativos de escritorio (Windows, Linux o MacOs) y móviles (Android o iOS) [31].

En este proyecto se usa como biblioteca de apoyo para pequeñas modificaciones de imagen puesto que la finalidad será utilizar Keras y TensorFlow para la creación de redes propias para analizar las imágenes.

2.3.5 NumPy

El lenguaje de Python no fue originalmente diseñado para una computación numérica. La comunidad de científicos e ingenieros, que fueron atraídos por el lenguaje, desarrollaron extensiones para el cálculo para Python [32]. Esto hizo que surgiese NumPy, originalmente denominada como Numeric, creada por Jim Hugunin. Numpy constituye la biblioteca de código abierto para funciones matemáticas que permite operar con matrices y vectores tal como MATLABo Wolfram Language.

Una de las funcionalidades más importantes es su estructura de datos ‘ndarray’ (matrices N-dimensionales).

Funciones más usadas en el proyecto [33] :

- **Numpy.Argmax():** función que devuelve el índice de los valores máximos de un vector o matriz que será definido como argumento de la función. En nuestro caso se

usa para definir el resultado más probable a la hora de realizar una clasificación utilizando redes neuronales.

- **Numpy.random.randint():** función que genera una lista de números enteros aleatorios. Como argumento se define el intervalo de valores mediante la especificación de valor bajo y el valor alto. Conjuntamente, se define el tamaño de la lista a devolver. Esta función se utiliza para generar situaciones aleatorias en la escena.
- **Numpy.expand_dims():** función que expande la matriz añadiendo una nueva dimensión o eje. Como argumento se utilizará la matriz original y la posición en el vector de ejes donde se colocará el eje adicional. Esta función se utiliza a la hora de convertir un valor de entrada a la red. Keras y TensorFlow requiere que los valores a predecir estén agrupados en una lista. Por ejemplo, si una imagen es de 28x28 la entrada a la red será 1x28x28.
- **Numpy.zeros():** función que sirve para generar una matriz con todos los valores 0 de dimensión establecida como argumento.

2.3.6 Tkinter

Para la creación de la interfaz gráfica de usuario (GUI) se utiliza Tkinter, una adaptación de la biblioteca Tk a Python [34]. Tk proporciona una serie de widgets utilizados para la creación y diseño de ventanas para programas, como botones, etiquetas para texto o barra de desplazamiento.

2.4 Hardware

Como material hardware principal, en este TFG se ha utilizado un computador personal. Éste deberá ser capaz de ejecutar los programas de simulación y la ejecución optima de la red neuronal, la cual dependerá mucho de la tarjeta gráfica de computador (por el tipo de operaciones y el número de esta que se deben ejecutar). Como también tendrá cierto peso el procesador principal y las memorias RAM instaladas. Para ello, se cuenta con un computador con un procesador AMD Ryzen 5 3600 (3.6GHz), tarjeta gráfica GeForce GTX 1660 OC 6GB GDDR5, memoria de almacenamiento SSD para todo el sistema y 16 GB RAM.

3. Resultados y discusión

Este apartado se divide en tres bloques. El primero será el diseño de la célula robotizada mediante el simulador de CoppeliaSim. El segundo será la realización del diseño de control de las diferentes partes de la célula creada mediante el código Python. El tercero se centrará en el diseño de las redes neuronales y los resultados proporcionados por las mismas.

3.1 Diseño de célula robotizada

El diseño de la célula robotizada se ha basado en parte en la célula que hay en la universidad de Almería. Sera necesario crear una zona de almacenaje donde todas deposiciones puedan ser vistas por una cámara y un sistema de manipulación que permita una entrada y salida de objetos dentro la zona de almacenaje.

3.1.1 Elementos fijos

- ***Almacén, estantería o Rack***

La célula requiere una zona de almacenaje de objetos y para ello se escogido una estantería (Figura 3.1) de los modelos que trae CoppeliaSim con la versión Edu dentro de la carpeta *furniture* y la subcarpeta *shelves-cupboardas rack* y el modelo *rack*.

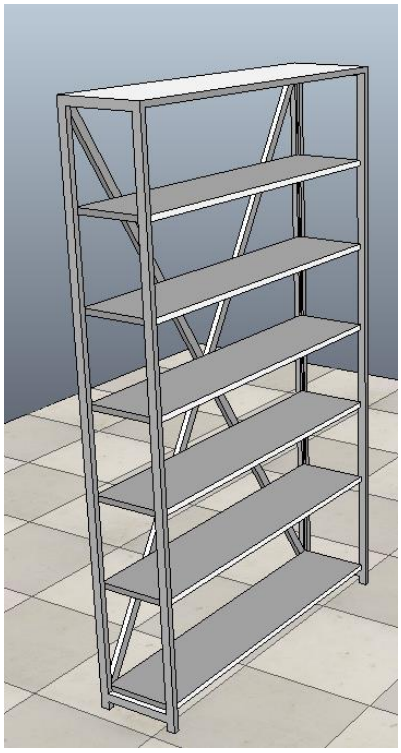


Figura 3.1 : Estantería de la escena

- ***Pared trasera***

Creada a partir de un objeto simple de forma cuboide. A esta pared se le ha puesto una textura con el logo de la universidad (Figura 3.2). La finalidad de esta pared es dar diferente fondo

para los objetos de la estantería. Así durante el aprendizaje, la red neuronal estará aprendiendo a diferenciar lo que es el objeto de interés y lo que no lo es.

Figura 3.2 : Pared con el logo de UAL de la escena

- ***Zona de aparcamiento***

Creada a partir de dos planos cuadrados cuya función es, simplemente, señalizar la zona donde el Summit puede aparcar. Un interno y otro externo (Figura 3.3). Sus límites se han definido a partir de los puntos extremos al que llega el brazo robot sin forzar sus articulaciones. La zona central (cuadrado amarillo) define la zona donde se situará el objeto correspondiente al robot Summit. Esta zona no será renderizada en la captura de la cámara.

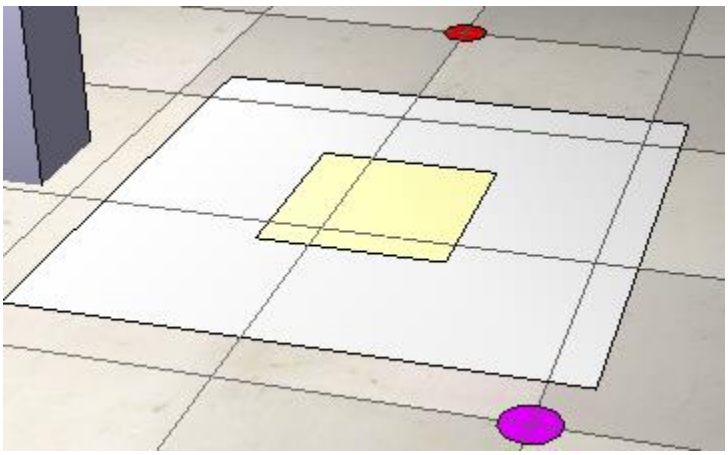


Figura 3.3 : Zona de aparcamiento del Summit

- ***Caja de objetos***

La caja está formada por planos para delimitar el movimiento de los objetos de dentro. Al duplicarse los objetos pueden salir disparados, para ello se ha diseñado esta caja para mantener los objetos originales en una zona delimitada (Figura 3.4).

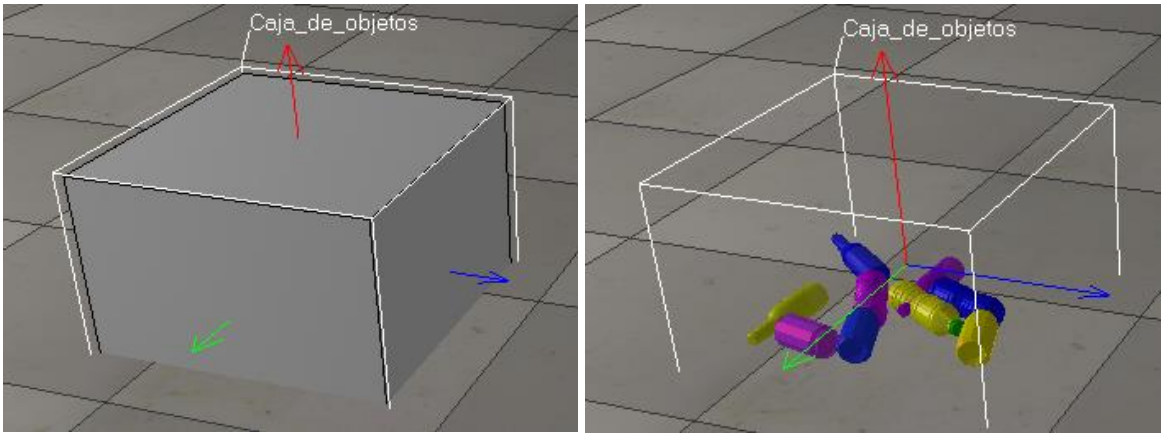


Figura 3.4 : Caja de objetos (imagen izquierda), vista interna (imagen derecha)

- ***Suelo***

El suelo viene predefinido al crear una escena. Se trata de un plano dinámico pero fijo que sirve de base de apoyo del resto de objetos de la escena (Figura 3.5).

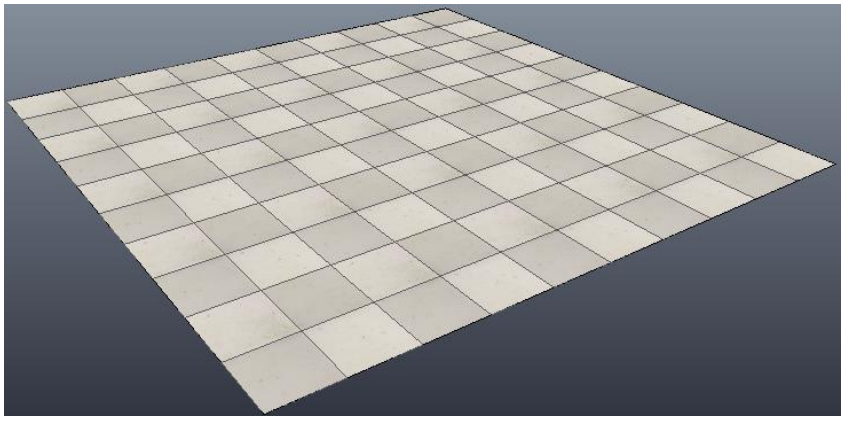


Figura 3.5 : Suelo de la escena

- ***Base de salida de objetos***

En la simulación se introducen objetos en la escena al igual que se sustraen objetos. Para esto último, se ha diseñado una base de colocación de objetos (Figura 3.6). El propósito es, una vez haya sido colocado el objeto en la base, enviar los datos de los objetos al script principal para poder eliminar dicho objeto.

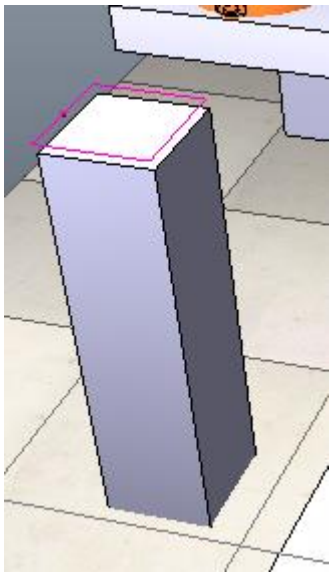


Figura 3.6 : Base de colocación de objetos con un sensor de área encima

3.1.2 Elementos móviles

- ***Brazo robot***

El robot manipulador que se utiliza en la escena será el modelo IRB140 de la casa ABB junto con una pinza modelo 85 de ROBOTIQ (Figura 3.8). Además de esto se unirá todo (Figura 3.9) sobre una estructura robotizada formada a partir de 3 formas primitivas (*primitives shapes*) cuboides unidas con 2 articulaciones lineales (Figura 3.7).

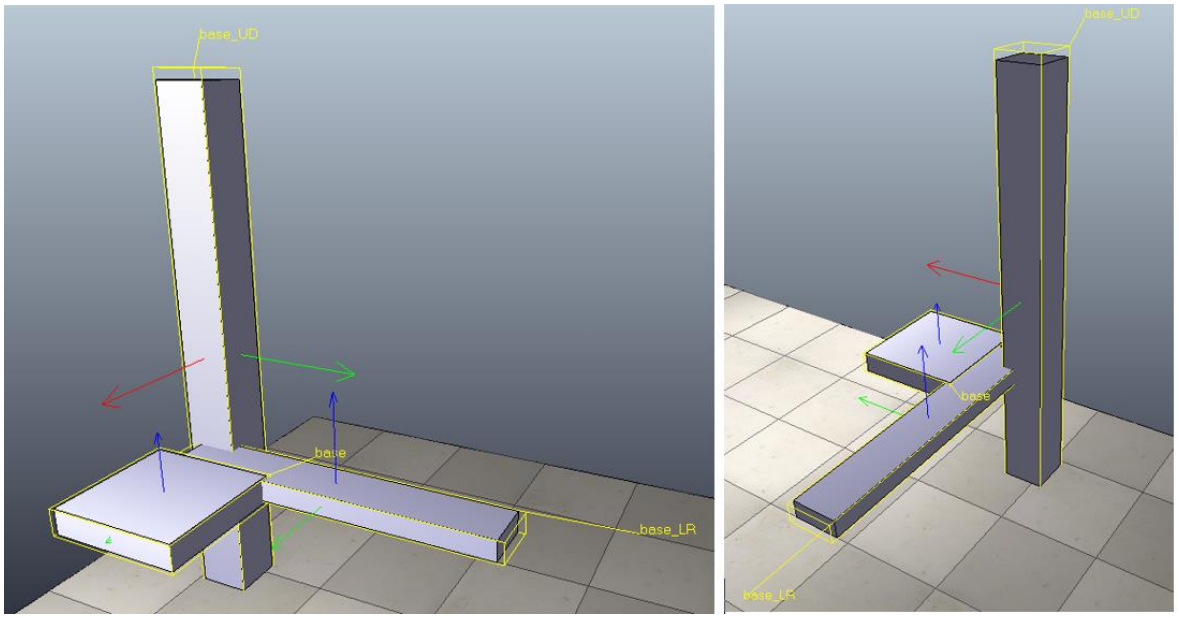


Figura 3.7 : Estructura robotizada donde será colocado el brazo robot

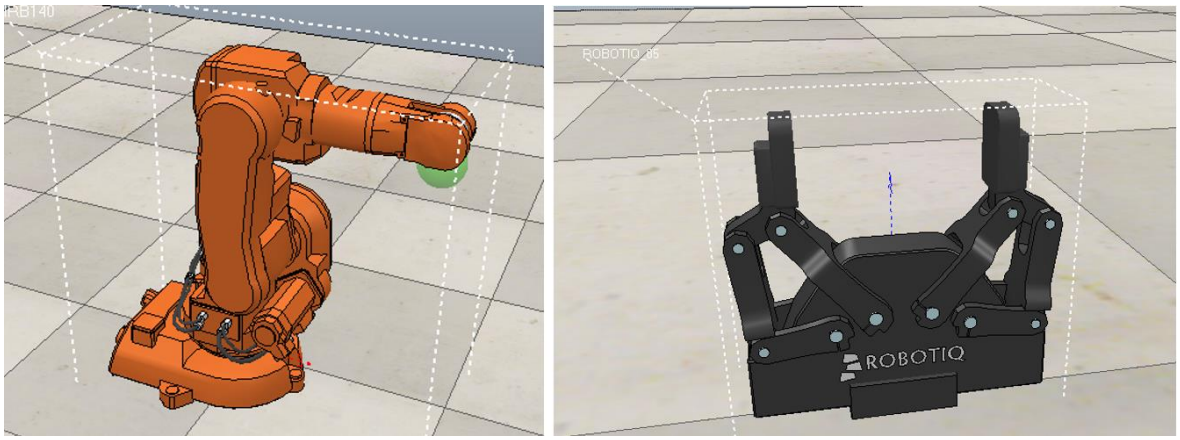


Figura 3.8 : Brazo Robot IRB140 (izquierda), pinza ROBOTIQ 85 (derecha)

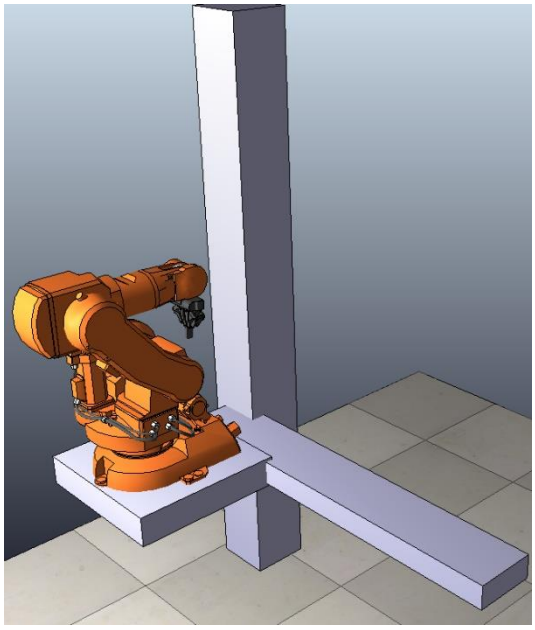


Figura 3.9 : Estructura robotiza con el brazo robot y la pinza

- ***Cinta***

Para la simulación de entrada de objetos se ha utilizado una cinta transportadora. El modelo virtual escogido es el modelo eficiente de la carpeta *coveyor belt* dentro de *equipment* entre los modelos que trae CoppeliaSim. Este modelo cuenta con un diseño de pocos polígonos y una simulación muy sencilla por lo que es ideal para evitar sobrecargar el entorno de simulación (Figura 3.10).

La entrada de objetos se basa en el duplicado de un modelo de la escena que una vez es generado, se posiciona y se orienta en la cinta donde esperara a ser recogido por el brazo robot.

Figura 3.10 : Cinta transportadora con un sensor lineal

- ***Summit***

Para la parte de diseño de la red neuronal para localizar la posición de objetos se ha usado un robot móvil con una base para la colocación de objetos.

El modelo de robot móvil es Summit XL un modelo virtual del robot la marca Robotnik. La base es una estructura (creada mediante formas primitivas cuboides) diseñada para mantener los objetos en posición vertical que se dejar con el brazo robot (Figura 3.11). La estructura también cuenta con la señal verde para facilitar la localización de su posición.

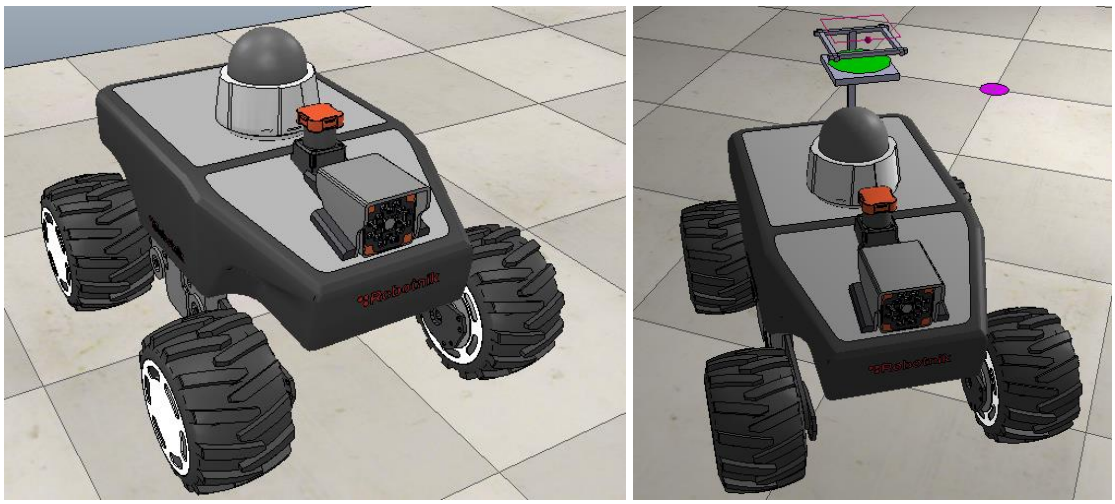


Figura 3.11 : Robot Summit (izquierda), robot modificado (derecha)

3.1.3 Sensores

Aunque es posible ver y comprobar el estado de la célula robotizada durante la simulación, los sensores son indispensables para automatizar el entorno.

- ***Sensor de cinta***

La cinta se ha programado con un sencillo control de velocidad cuyo funcionamiento se basa en la detección de objetos al final de cinta. Cuando hay objeto detectado, la velocidad de cinta se define en cero, y de forma contraria, cuando no hay objeto detectado, la cinta está en movimiento (Figura 3.12). Esta detección se realiza mediante un sensor lineal.

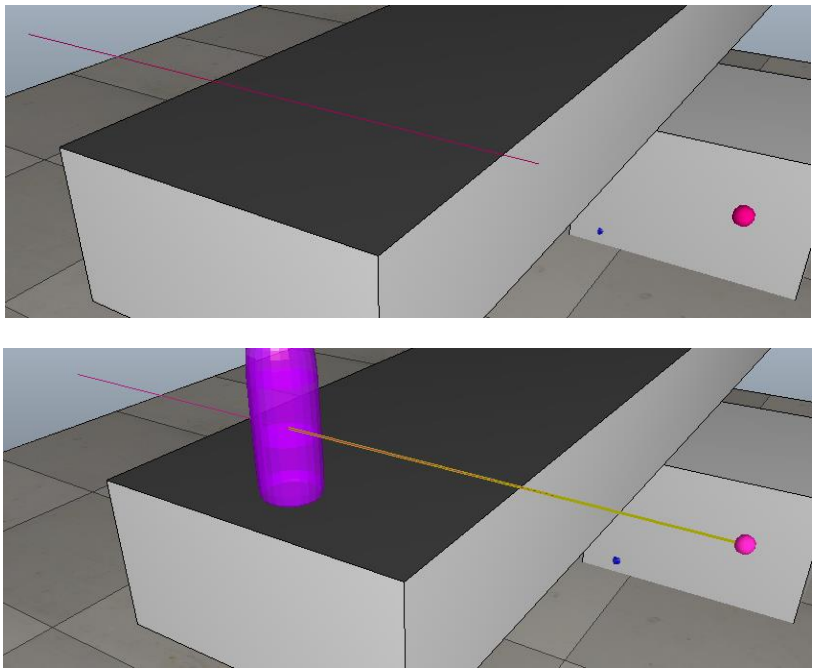
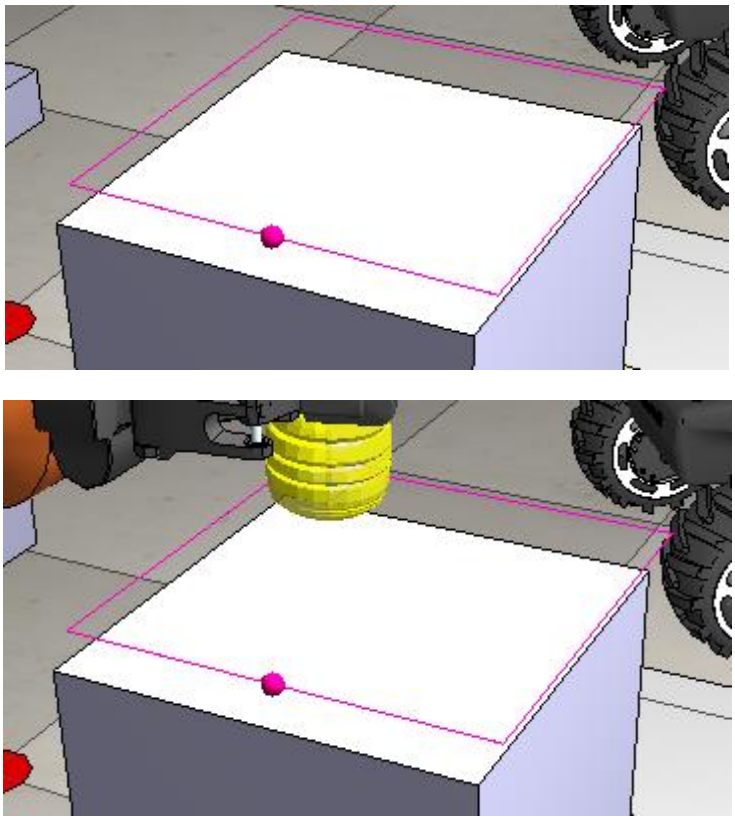


Figura 3.12 : Sensor de cinta

- ***Sensor de la base de salida de objetos***

El encargado de enviar los datos del objeto situado en la base es un sensor de área que, al llamar a la función encargada de proporcionar la información del sensor, envía el valor del identificador del objeto que detecta (Figura 3.13).



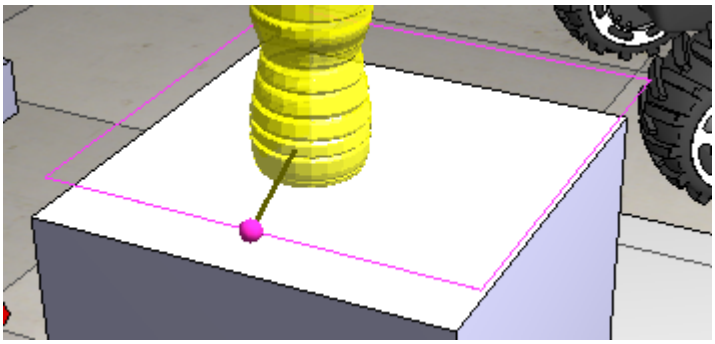


Figura 3.13 : Cinta de la base de salida de objetos

- ***Sensor del robot Summit***

De la misma manera que la base de salida, el robot Summit cuenta con un sensor de área (Figura 3.14) cuya función es obtener la información del objeto que se halla en la base del robot para su eliminación.

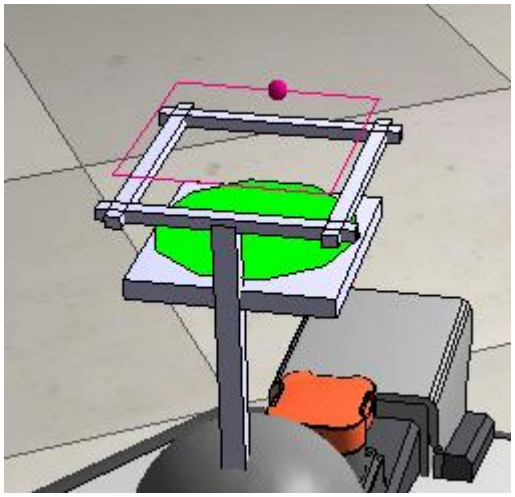


Figura 3.14 : Sensor de Summit

- ***Cámara de estantería***

La finalidad del proyecto es usar la visión artificial para la detección y clasificación de objetos de la escena. Los encargados de dar la información son las cámaras. La primera cámara está enfocada en la estantería (Figura 3.15).

La resolución es 128x128 y el renderizador OpenGL 3. La resolución es baja para no sobrecargar la red neuronal y para que el envío entre Servidor-Cliente sea de menor tamaño.

- ***Cámara de Summit***

Al igual que la anterior, esta cámara se enfoca en la zona de parada del Summit (Figura 3.16). También la resolución es de 128x128 y el renderizador OpenGL 3. Esta cámara no será fija, para simular cierto ruido, se añade cierta variación en su posición cada ver que se inicia el

programa. Esto servirá para comprobar las capacidades de las redes para enfrentarse a este tipo de problemas.

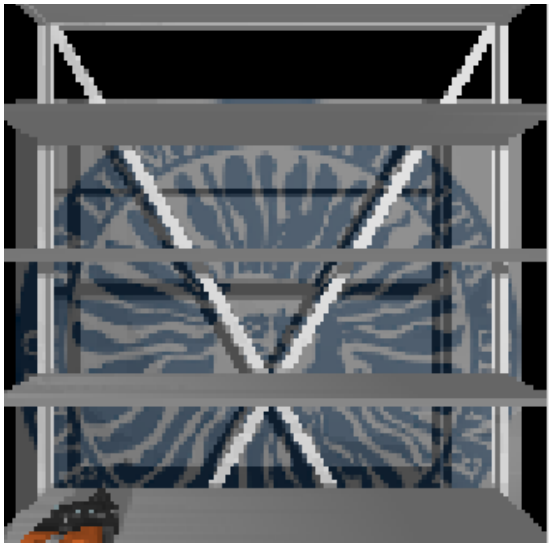


Figura 3.15 : Imagen que proyecta la cámara fijada en la estantería

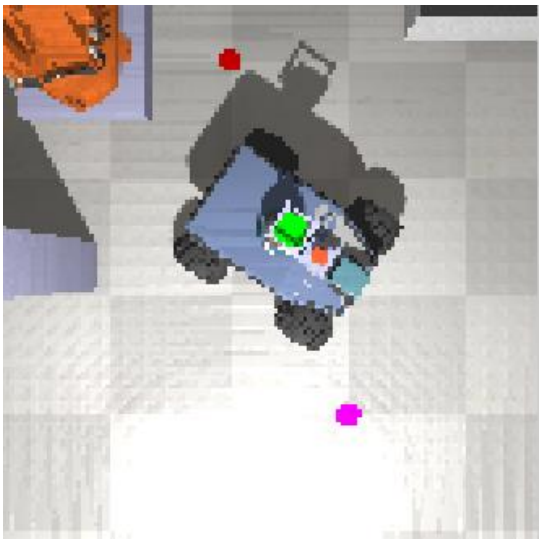


Figura 3.16 : Imagen que proyecta la cámara fijada en la zona de aparcamiento

3.1.4 Elementos mixtos

- ***Luz general***

Una de las características interesantes del simulador es su capacidad de generar sombras realistas al usar OpenGL 3 a la hora de renderizar imágenes. Para aprovechar más este efecto se utiliza un origen de luz puntual (Figura 3.17) capaz de variar su posición espacial, lo que dará diferentes imágenes para un mismo estado del escenario.

- ***Objetos de la estanteria***

Se han escogido diferentes modelos 3D ofrecidos por la página free3d.com/. Se han exportado directamente a CoppeliaSim o indirectamente, usando blender 3D para realizar algún cambio leve del modelo.

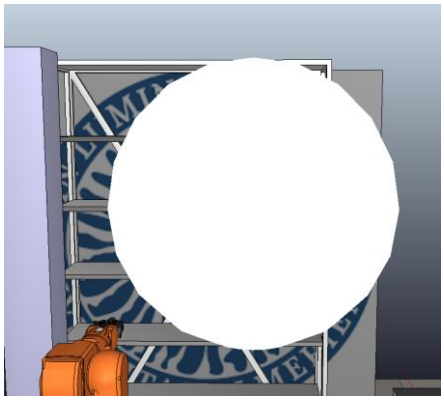


Figura 3.17 : Luz general de la escena

Después, cada objeto cuenta con una estructura dinámica que es igual en todos los modelos, un cilindro de misma dimensión y peso y por encima como forma visible la obtenido desde la página. También se ha modificado el color y la transparencia de luz.

En la Figura 3.18 se pueden ver los objetos numerados (de izquierda a derecha) de 0 a 9.

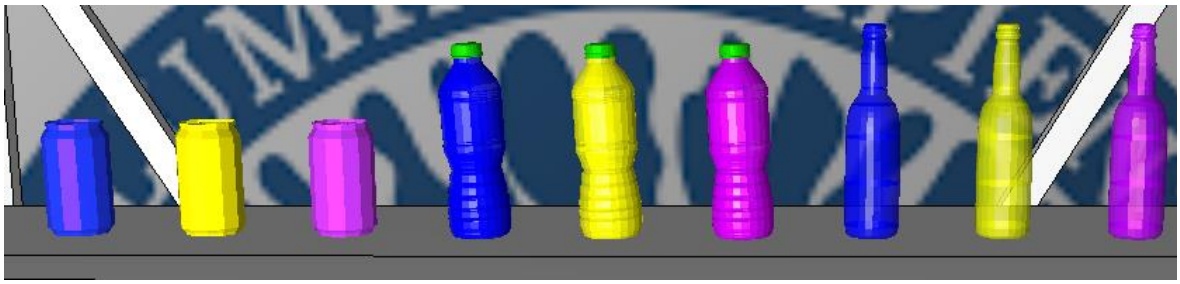


Figura 3.18 : Objetos usados en el TFG, estos formados por latas, botellas de plástico y de cristal

- ***B0RemoteApiServer***

Para utilizar la API remota basada en B0 existen dos posibilidades. La primera es iniciando el script de la barra de menú [*Complementos → b0RemoteApiServer*]. Este script activa la funcionalidad API a todas las escenas sin interrupción.

La segunda manera, y la que se utiliza en la escena, es mediante el modelo de la API. Arrastrando y soltándolo en escena como se haría con cualquier modelo del navegador de modelos. Este se encuentra en la carpeta *tools* (herramientas) como se puede ver en la Figura 3.19.

3.2 Control de la célula

El diseño se repartirá en tres fases. La primera estará enfocada a preparar los programas para la comunicación e instalación de las librerías. La segunda se centra en el diseño funciones de Python para definir el control de la célula. Y por último la tercera, el diseño de la interfaz gráfica donde se hayan los controles principales de la célula y se tendrá así, una manera más cómoda de realizar el control.

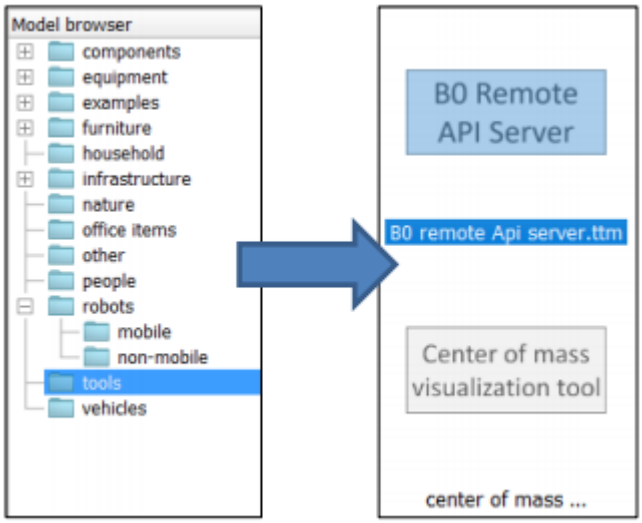


Figura 3.19 : Procedimiento para habilitar de forma manual el lado del servidor del B0-based remote API

3.2.1 Fase 1: Inicialización

Antes de iniciar con el código en *Python* se debe asegurarse de tener la habilitación de la API remota. Desde el lado del servidor, el simulador, se ha añadido el modelo B0 en la escena (Figura 3.20).

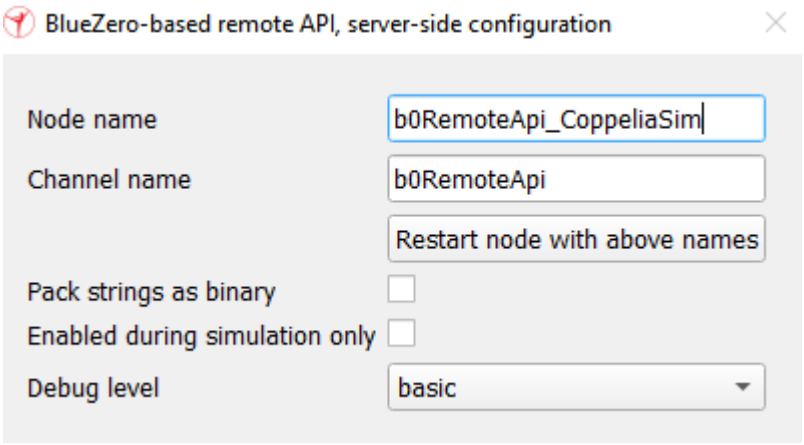


Figura 3.20 : Configuración del elemento B0 remote Api server

Desde el lado del cliente, para la funcionalidad de *Python* es necesario instalar *MessagePack* desde la consola de Anaconda Prompt. Después añadir a la carpeta del programa añadir dos archivos que trae CoppeliaSim al instalar. Ambos se encuentran en *programming /remoteApiBindings/ b0Based /Python*, son *b0RemoteApi.py* y *b0.py*.

Por último, falta la biblioteca *BlueZero* (b0.dll) y sus dependencias, que se encuentran localizados en *programming/b0RemoteApiBindings/python*:

- boost_date_time-vc141-mt-x64-1_70.dll,
- boost_filesystem-vc141-mt-x64-1_70.dll,
- boost_program_options-vc141-mt-x64-1_70.dll,
- boost_regex-vc141-mt-x64-1_70.dll,
- boost_thread-vc141-mt-x64-1_70.dll,
- libzmq-mt-4_3_2.dll,
- lz4.dll y
- zlib1.dll

También será necesario tener instaladas las librerías *TensorFlow*, *Numpy* y *OpenCV*. Las cuáles serán referidas en el programa como **tf** para *TensorFlow*, **np** para *NumPy* y **cv2** para *OpenCV*.

Ya con todo listo, se inicia la comunicación con b0RemoteApi en el código de Python importando el archivo b0RemoteApi.py y creando el cliente mediante el objeto *RemoteApiClient* con los dos primeros parámetros referidos al nombre del nodo y el nombre del canal tal como se ha definido en el modelo B0 de la escena.

```
import b0RemoteApi
client = b0RemoteApi.RemoteApiClient('b0RemoteApi_CoppeliaSim','b0RemoteApi')
```

Después toca localizar todos los objetos de interés que se va a utilizar con el código. Para ello se usará la función *simxGetObjectHandle*.

```
Brazo1=client.simxGetObjectHandle('IRB140_joint1',client.simxService-
Call())[1]
Brazo2=client.simxGetObjectHandle('IRB140_joint2',client.simxService-
Call())[1]
Brazo3=client.simxGetObjectHandle('IRB140_joint3',client.simxService-
Call())[1]
…
```

3.2.2 Fase 2: Definición de funciones

En esta fase se definen las funciones Python que se usan en el programa. Ciertas funciones estarán incluidas en una biblioteca propia llamada *funciones.py*.

- ***Función mover, moverB2y3, moverB235***

Función dedicada a mandar la posición destino de una articulación. Existe una función dedicada al control de las articulaciones, *simxGetJointPosition*, pero la siguiente función

servirá para añadir cierto margen de error tolerable para que al definir un control PID acelerado no realice ninguna oscilación peligrosa durante la ejecución.

```
def mover(client,joint,mov):
    margen=0.01
    _,MotorUD=client.simxGetObjectHandle('MotorUD',client.simxServiceCall())
    if joint==MotorUD:
        margen=0.1
    client.simxSetJointTargetPosition(joint,mov,client.simxDefault-
Publisher())
    while(client.simxGetJointPosition(joint,client.simxService-
Call())[1]<mov-margen)or(client.simxGetJointPosition(joint,client.simx-
ServiceCall())[1]>mov+margen):
        client.simxSetJointTargetPosition(joint,mov,client.simxDefault-
Publisher())
```

En ciertas ocasiones es necesario llamar a varias funciones de movimiento en el mismo instante. Para ello se ha modificado la función 'mover' para dos casos concretos. Ambas funciones se han guardado en la biblioteca *funciones.py*.

- moverB2y3: esta función permite el movimiento de los ejes IRB140_joint2 y IRB140_joint3 (nombrados Brazo2 y Brazo3 respectivamente) de forma simultánea. Se puede ver en la Figura 3.21 estas articulaciones. La posición objetivo de un eje es complementada en el otro.

```
def moverB2y3(client,mov):
    margen=0.001
    _,Brazo2=client.simxGetObjectHandle('IRB140_joint2',client.simxService-
Call())
    _,Brazo3=client.simxGetObjectHandle('IRB140_joint3',client.simxService-
Call())
    client.simxSetJointTargetPosition(Brazo2,mov,client.simxDefault-
Publisher())
    client.simxSetJointTargetPosition(Brazo3,-mov,client.simxDefault-
Publisher())
...
```

- moverB235: esta función principalmente se usa para la colocación de objetos del Summit. Previamente se pasan las dos coordenadas de la posición del Summit a coordenadas de giro del robot mediante cinemática inversa (a y b).

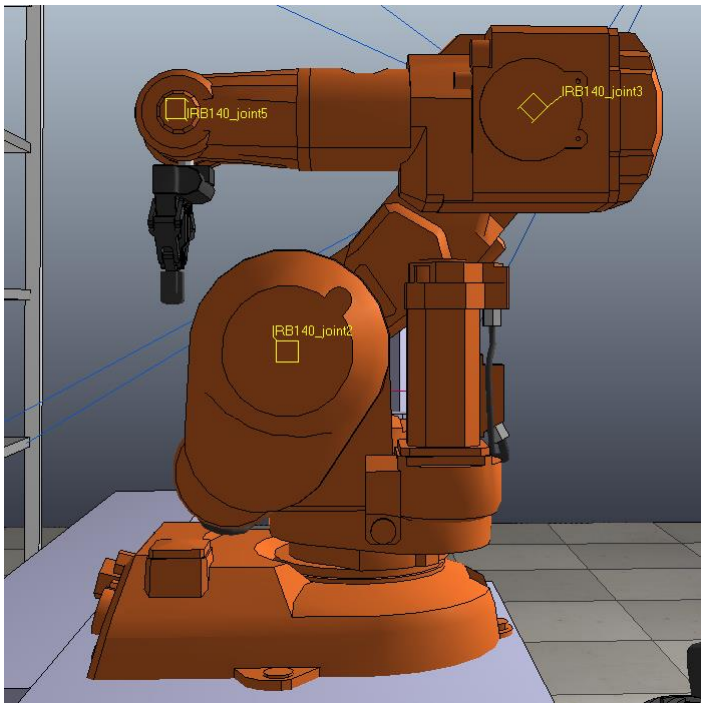


Figura 3.21 : Articulaciones del brazo robot

```
def moverB235(client,a,b):
    margen=0.001
    q2=(a-90)*(2*3.14)/360
    q3=(b+90)*(2*3.14)/360
    q5=(-(a-90)-(b+90))*(2*3.14)/360
    _,Brazo2=client.simxGetObjectHandle('IRB140_joint2',client.simxService-
Call())
    _,Brazo3=client.simxGetObjectHandle('IRB140_joint3',client.simxService-
Call())
    _,Brazo5=client.simxGetObjectHandle('IRB140_joint5',client.simxService-
Call())
...
```

- ***Grupo de funciones de posicionado:***

Se ha agrupado diferentes configuraciones del brazo robot para poder acceder a cualquiera de ellos de forma inmediata a la hora programar el robot. Estas configuraciones son funciones formadas por una secuencia de movimientos.

- p0 (posición 0): esta postura se utiliza durante la movilización del brazo robot mediante los ejes de la estructura robotizada donde este situado sin la posibilidad de chocar con ninguna parte de escena como se puede ver en la Figura 3.22.

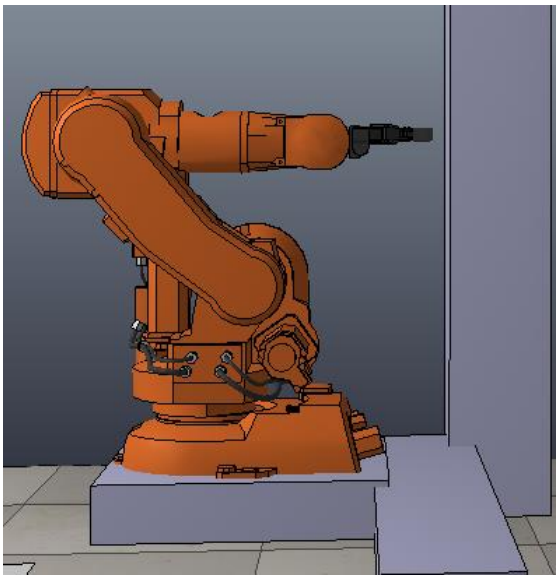


Figura 3.22 : Posición “p0”

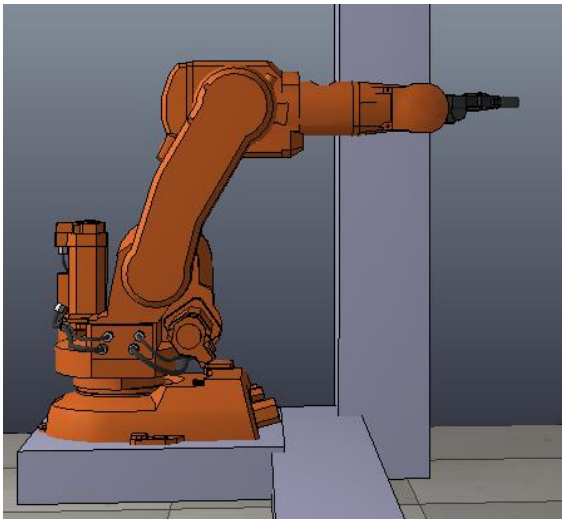


Figura 3.23 : Posición “guardar”

- guardar: esta postura será usada a la hora de acceder a la estantería. Permite llegar a la posición donde se va a recoger o guardar las piezas (Figura 3.23).
- esconder: igual que la postura p0 (Figura 3.22), pero con la posición baja en el eje de la articulación motorUD para no aparecer delante de la cámara a la hora de revisar la estantería.
- cámara: esta postura coloca la pinza justo delante de la cámara (Figura 3.24) para poder analizar el tipo de objeto que lleva el robot.

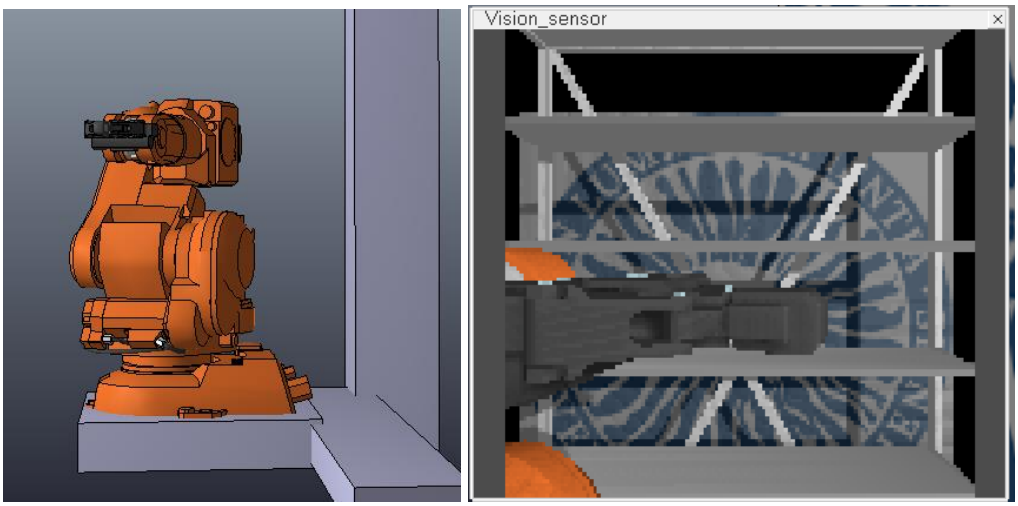


Figura 3.24 : Posición “cámara” (izquierda) y la misma posición vista por la cámara (derecha)

- recoger: se realiza la operación de recogida de la cinta, no es una postura exactamente, pero al ser una serie de movimientos que se realizan hasta tener la pieza recogida por el robot será englobada dentro de este grupo (Figura 3.25).

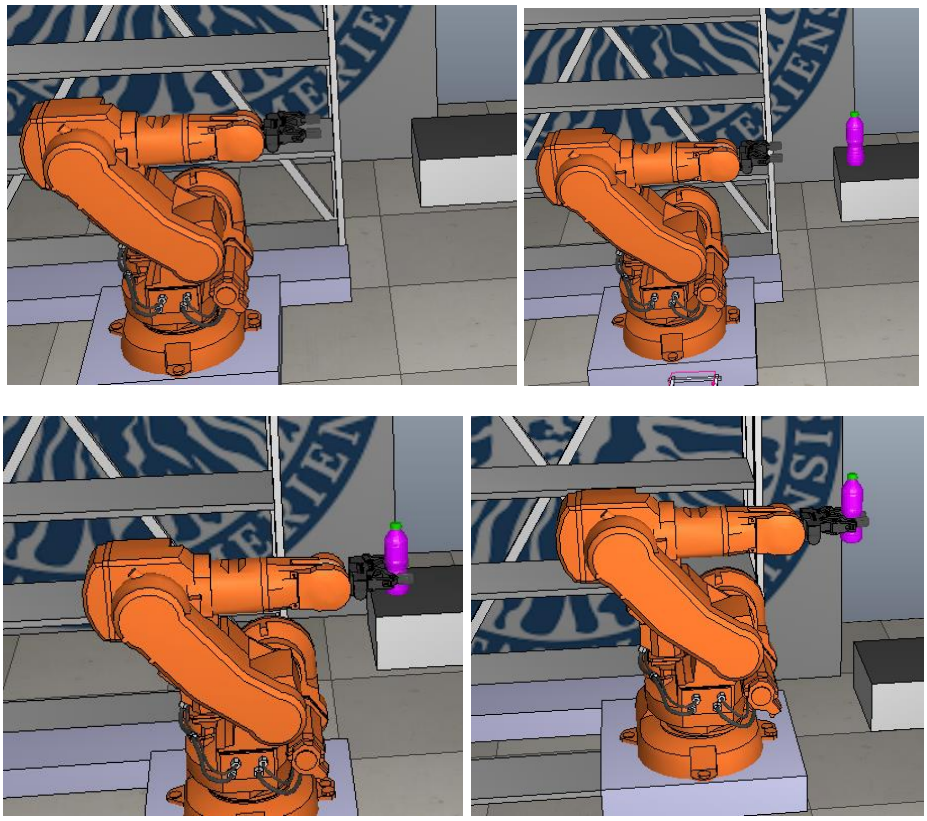


Figura 3.25 : Proceso de recoger un objeto de la cinta, posición "recoger"

```
class posicion:
    def p0():
        funciones.moverB2y3(client,45*(2*3.14)/360)
        funciones.mover(client,Brazo5,0)
        #funciones.mover(client,Brazo1,180*(2*3.14)/360)
    def guardar():
        funciones.mover(client,Brazo1,180*(2*3.14)/360)
```

```
        funciones.moverB2y3(client,0*(2*3.14)/360)
        funciones.moverB2y3(client,-25*(2*3.14)/360)
        funciones.mover(client,Brazo5,0*(2*3.14)/360)
    def esconder():
        funciones.moverB2y3(client,45*(2*3.14)/360)
        funciones.mover(client,MotorUD,-0.9)
        #funciones.mover(client,Brazo1,90*(2*3.14)/360)
    def camara():
        funciones.moverB2y3(client,45*(2*3.14)/360)
        funciones.mover(client,Brazo5,0)
        funciones.mover(client,Brazo1,70*(2*3.14)/360)
        funciones.mover(client,MotorLR,-0.3)
        funciones.mover(client,MotorUD,-0.37)
    def recoger():
        funciones.mover(client,Brazo1,0*(2*3.14)/360)
        funciones.mover(client,MotorLR,0)
        funciones.mover(client,Brazo1,90*(2*3.14)/360)
        funciones.mover(client,MotorUD,-0.83)
        funciones.moverB2y3(client,45*(2*3.14)/360)
        funciones.mover(client,Brazo5,0)
        funciones.mover(client,MotorLR,0.2)

        funciones.pinza(client,False)
        posicionado=0
        t_max=10
        t=0
        while(posicionado==0) and not(t>t_max):
            time.sleep(1)
            posicionado=client.simxReadProximitySensor(Sen-
sor1,client.simxServiceCall())[1]
        if not(t>t_max):
            funciones.mover(client,MotorLR,0.530)
            funciones.pinza(client,True)
            funciones.mover(client,MotorUD,-0.5)
            funciones.mover(client,MotorLR,0)
```

- ***Control de pinza***

Después de definir las funciones de movimiento de las articulaciones y funciones más complejas y concretas del brazo robot queda definir la apertura y cierre de la pinza. La función usa una entrada booleana para definir la acción de cierre o apertura.

El código es una adaptación del script original escrito en código LUA que venía predefinido junto al modelo de la pinza. Este script fue removido como en el caso de brazo robot, pero igualmente se ha estudiado su funcionamiento para implementar en Python un control de pinza.

```
def pinza(client,close):
    tiempo_pinza=1 #segundos
    ret,j1=client.simxGetObjectHandle('ROBOTIQ_85_active1',client.simx-
ServiceCall())
    ret,j2=client.simxGetObjectHandle('ROBOTIQ_85_active2',client.simx-
ServiceCall())
    ret,p1=client.simxGetJointPosition(j1,client.simxServiceCall())
    ret,p2=client.simxGetJointPosition(j2,client.simxServiceCall())
    if close:
        if (p1<p2-0.008):
            client.simxSetJointTargetVelocity(j1,-0.08,client.simxDefault-
Publisher()) #0.01
            client.simxSetJointTargetVelocity(j2,-0.32,client.simxDefault-
Publisher()) #0.04 y 0.02
        else:
            client.simxSetJointTargetVelocity(j1,-0.32,client.simxDefault-
Publisher())
```

```
            client.simxSetJointTargetVelocity(j2,-0.08,client.simxDefault-
Publisher())
    else:
        if (p1<p2):
            client.simxSetJointTargetVelocity(j1,0.32,client.simxDefault-
Publisher())
            client.simxSetJointTargetVelocity(j2,0.16,client.simxDefault-
Publisher())
        else:
            client.simxSetJointTargetVelocity(j1,0.16,client.simxDefault-
Publisher())
            client.simxSetJointTargetVelocity(j2,0.32,client.simxDefault-
Publisher())
    time.sleep(tiempo_pinza)
```

- ***Movimiento por la estantería***

A la hora de guardar y recoger objetos en la estantería se usará esta función. Ajusta la posición de las articulaciones de la base del brazo robot (MotorUD y MotorLR) según la posición de 'x' e 'y' de la estantería mediante una función lineal simple. Y mediante la secuencia de movimientos definidos del brazo robot se ejecuta la recogida o el almacenado de los objetos.

Primero el brazo se coloca en una postura que implique el menor esfuerzo por inercia y un nulo peligro de choque con el resto del escenario (posición p0). Después, se lleva la base del brazo hasta la posición de la estantería deseada y posteriormente, el brazo realiza el movimiento de posicionamiento del objeto abriendo la pinza para liberar el objeto o si no, el movimiento de recogida abriendo la pinza antes del movimiento. Después de ello, el brazo retorna a la posición definida como p0.

Como entrada de función serán necesarios dos valores enteros de posición (x e y), y un valor booleano que servirá para definir si la acción es almacenaje de un objeto o su recogida.

```
def estanteria(x,y,guardar):
    ymax=7
    qUD=-x*0.3-0.015+0.03
    qLR=-0.4+y*0.8/(ymax)
    posicion.p0()
    funciones.mover(client,MotorUD,qUD)
    funciones.mover(client,MotorLR,qLR)
    if guardar==False:
        qUD=-x*0.3-0.015
        funciones.mover(client,MotorUD,qUD)
        funciones.pinza(client,False)
    posicion.guardar()
    if guardar:
        qUD=-x*0.3-0.015
        funciones.mover(client,MotorUD,qUD)
        funciones.pinza(client,False)
    else:
        funciones.pinza(client,True)
    posicion.p0()
```

- ***Dejar o recoger un objeto en la base auxiliar***

Esta función está dedicada también a realizar la secuencia de recoger o dejar objetos, pero con la diferencia de que solo hay una posición, la estructura que funciona como base auxiliar para dejar temporalmente algún objeto o para la salida de escena de un objeto.

```
def dejar_pieza(guardar):
```

```
    qUD=-3*0.27-0.015
    qLR=-0.4+0*0.8/(7)
    funciones.mover(client,MotorUD,qUD)
    funciones.mover(client,MotorLR,qLR)
    if guardar==False:
        qUD=-3*0.3-0.015
        funciones.mover(client,MotorUD,qUD)
        funciones.pinza(client,False)
    funciones.mover(client,Brazo1,0*(2*3.14)/360)
    funciones.moverB2y3(client,0*(2*3.14)/360)
    funciones.moverB2y3(client,-25*(2*3.14)/360)
    funciones.mover(client,Brazo5,0*(2*3.14)/360)
    if guardar:
        qUD=-3*0.3-0.015
        funciones.mover(client,MotorUD,qUD)
        time.sleep(1)
        funciones.pinza(client,False)
    else:
        funciones.pinza(client,True)
    funciones.moverB2y3(client,0*(2*3.14)/360)
    posicion.p0()
```

- ***Grupo Pieza (Objeto)***

De la misma manera que han agrupado las funciones referentes al posicionado del brazo robot, las funciones referentes a las piezas (botellas y latas) también se agruparan en una clase.

- Colocar: Esta función coloca una pieza en la posición especificada como entrada y de forma vertical. Para ello se usan la función *simxSetObjectPosition* para posicionar el objeto en la posición específica y *simxSetObjectQuaternion* para orientarlo.

- Aleatorio: Para posicionar una pieza aleatoria en la cinta primero se elige aleatoriamente un número del 1 al 9 para elegir el objeto que se va a duplicar. Después se realiza la identificación del objeto a duplicar utilizando el nombre del objeto generado a partir del número aleatorio. Con el valor del identificador se realiza el duplicado. En este punto se generará el nuevo objeto en la misma posición que el objeto original por lo que habrá que posicionar la copia sobre la cinta.

- Aleatorio en sitio: Esta función es una variación de la anterior, pero en el proceso de aleatorización se añade la posibilidad de no duplicar ningún objeto (definido como objeto_0 o pieza_0). Además, el objeto duplicado se coloca en la posición de la estantería definiéndola en la entrada de la función.

```
class pieza:
    def colocar(objeto_n,x,y,z):
        client.simxSetObjectPosition(objeto_n,-1,[x,y,z],client.simxDefault-
Publisher())
        client.simxSetObjectQuaternion(objeto_n,-1,[0,0,0,0],client.simxDe-
faultPublisher())

    def aleatoria():
        i=np.random.randint(9, size=1)[0]
        i+=1
        shape_i='Shape'+str(i)
        objeto_i='Objeto'+str(i)
```

```
        ret0,objeto=client.simxGetObjectHandle(objeto_i,client.simxService-
Call())
        ret1,shape=client.simxGetObjectHandle(shape_i,client.simxService-
Call())
        item=client.simxCopyPasteObjects([objeto,shape],0,client.simx-
ServiceCall())[1][0]
        pieza.colocar(item,-0.4045,0.8,0.827)
        return i

    def aleatoria_en_sitio(xi,yj):
        x=-1.1705
        y=-1.6742-(1.5597-1.6742)*xi
        z=+1.7001-(1.7001-1.4001)*yj
        i=np.random.randint(10, size=1)[0]
        if not(i==0):
            shape_i='Shape'+str(i)
            objeto_i='Objeto'+str(i)
            ret0,objeto=client.simxGetObjectHandle(objeto_i,client.simx-
ServiceCall())
            ret1,shape=client.simxGetObjectHandle(shape_i,client.simx-
ServiceCall())
            item=client.simxCopyPasteObjects([objeto,shape],0,client.simx-
ServiceCall())[1][0]
            pieza.colocar(item,x,y,z)
        return i
```

- ***Generación de escena aleatoria***

El foco de luz, la cámara enfocada al Summit y el propio Summit son objetos que suelen tener una posición variable para obtener diferentes ambientes de la escena.

Para ello primero se utiliza la función Randint de Numpy, para generar 9 números aleatorios enteros entre 0 y 99. Tres de estos valores son referidos a la posición de la luz (x,y,z), otros tres para la posición de la cámara de igual manera, y los últimos tres son referidos a la posición del Summit (posición x e y) y la orientación de este en el eje z.

Esta función debe ejecutarse cuando la simulación no esté parada, ya que puede dar problemas de descomposición del robot Summit.

```
def pos_aleatorios(client,Light,Cam_2,summit):
    i=np.random.randint(100, size=3)
    j=np.random.randint(100, size=3)
    k=np.random.randint(100, size=3)
    #pos luz
    x=3*i[0]/100-1.2
    y=2*j[0]/100-2.2
    z=2*k[0]/100+1
    client.simxSetObjectPosition(Light,-1,[x,y,z],client.simxDefault-
Publisher())

    #pos cam
    x=0.1*i[1]/100+0.3#-0.5
    y=0.1*j[1]/100-0.88#-0.5
    z=0.1*k[1]/100+1.9#+0.4
    client.simxSetObjectPosition(Cam_2,-1,[x,y,z],client.simxDefault-
Publisher())

    #pos summit
    x=(0.55-0.2)*i[2]/100+0.2
    y=-(1.1-0.71)*j[2]/100-0.71
    z=360*k[2]/100-180
    client.simxSetObjectPosition(summit,-1,[x,y,0.305],client.simxDefault-
Publisher())
```

```
    client.simxSetObjectOrientation(summit,-1,[0,0,z],client.simxDefault-
Publisher())
    return x,y
```

También, será necesario tener la posibilidad de tener objetos en la estantería para no perder mucho tiempo llevando objetos de la cinta al armario durante la etapa de entrenamiento y etapa de validación de resultados. Esto será la generación de una estantería aleatoria.

```
def estanteria_aleatoria():
    for i in range(4):
        for j in range(9):
            pieza.aleatoria_en_sitio(j,i)
```

- ***Eliminación de objetos***

Para la salida de escena de un objeto primero será necesaria su identificación. Esta identificación se realiza con el sensor en la base de la plataforma mediante la función *simxReadProximitySensor,* la cual devuelve el valor de identificacion del objeto que está detectando el sensor. Después proceder a remover este objeto con la función con la función *simxRemoveObjects*. Al ser un objeto estructurado, se realizará la operación dos veces para eliminar toda la estructura.

```
def borrar_objeto(client,Sensor):
    handle_a_destruir=client.simxReadProximitySensor(Sensor,client.simxService-
Call())[4]
    client.simxRemoveObjects([handle_a_destruir],1,client.simxServiceCall())
    time.sleep(1)
    handle_a_destruir=client.simxReadProximitySensor(Sensor,client.simxService-
Call())[4]
    client.simxRemoveObjects([handle_a_destruir],1,client.simxServiceCall())
```

Para la eliminación de los objetos en la base del Summit el código es igual cambiando el sensor.

- ***Captura de imagen y transformación.***

En CoppeliaSim existe la función *simxGetVisionSensorImage*, que permite recuperar la imagen de un sensor cámara. Sin embargo, durante el diseño la función ha dado ciertos errores, por ello la captura se realiza en un script Lua dentro del simulador y cuando este es llamado, envía la imagen al cliente.

Para hacer una llamada a un script dentro de CoppeliaSim, será obligatorio la función *simxCallScriptFunction*. La imagen que se obtiene viene como una lista de valores (tupla), la cual se puede convertir en matriz de 128x128x3 utilizando la función *tuple2image*.

```
def captura(client,summit):
    if summit==True:
        sensor='Vision_general'
    else:
        sensor='Vision_sensor'
    resul=client.simxCallScriptFunction('myFunctionName@'+sen-
sor,"sim.scripttype_childscript",[], client.simxServiceCall())
    im_pre=resul[1][0]
    return tuple2image(im_pre)
```

- Tuple2image: los valores de tupla que viene como imagen vienen por tripletes de bytes con cada color por byte. Esto da valores de 0 a 255 según la intensidad de cada color por píxel.

```
def tuple2image(im_pre):
    reso=128,128
    frame2=np.zeros((reso[0], reso[1], 3))
    frame2=frame2.astype('uint8')
    #La imagen vendra en valores de 0 a 1, hay que pasarlo a 0 a 255 redondeando
a entero
    j=0
    k=0
    l=0
    for i in range(len(im_pre)):
        if j%3==0 and j!=0:
            j=0
            k=k+1
        if k%reso[0]==0 and k!=0:
            j=0
            k=0
            l=l+1
        frame2[l][reso[0]-k-1][j]=round(im_pre[len(im_pre)-i-1]*255,0)
        j=j+1
    return frame2
```

- ***Función para predecir el tipo de objeto***

La red neuronal diseñada para la identificación de objetos tiene como entrada una matriz de 30x15x3. Estos valores han sido elegidos ya que este tamaño puede contener toda información si se realiza un recorte de la imagen de un objeto de la estantería sin perder información relevante para su identificación.

Después de obtener la imagen se predice que el tipo de objeto utilizando la función *predict* con la imagen como entrada. Al finalizar la predicción se guarda el valor en un matriz que representara al estado.

```
def revisar_estanteria():
    mat_est_e=np.zeros((4,9), dtype=int)
    posicion.esconder()
    frame=funciones.captura(client,False)
    for i in range(4):
        for j in range(9):
            frame2=funciones.recorte(frame,i,j)
            x=funciones.imagen_a_red(frame2)
            mat_est_e[i][j]=np.argmax(cnn.predict(x)[0])
    return mat_est_e
```

Si el objeto a analizar es el que el brazo robot ha colocado frente a la cámara será necesario escalar la imagen capturada por la cámara a la resolución de trabajo.

```
def objeto_camara():
    frame=funciones.captura(client,False)
    frame2=cv2.resize(frame,(15,30))
    x=funciones.imagen_a_red(frame2)
    resultado=np.argmax(cnn.predict(x)[0])
    return resultado
```

- Función auxiliar de recorte:

```
def recorte(image,j,k):
```

```
    x=6+j*29 #110/4 = 27.5
    y=10+k*12 #110/9 = 12.22
    image_cutted=image[x:x+30, y:y+15]
    return image_cutted
```

- Función auxiliar imagen a red:

```
def imagen_a_red(imagen):
    frame2=cv2.cvtColor(imagen,cv2.COLOR_BGR2RGB)
    frame3=frame2.astype('float32')/255
    return np.expand_dims(frame3,axis=0)
```

- Función localizar pieza: Cuando se tenga la matriz del estado de la estantería será sencillo encontrar la posición de cualquier objeto dentro de la estantería. Se realiza un barrido a la matriz hasta localizar el valor que se está buscando.

```
def localizar_pieza(mat,pieza):
    ret=0
    n_piezas=0
    pos_i,pos_j=9,9
    for i in range(4):
        for j in range(9):
            if pieza==mat[3-i][8-j]:
                pos_i,pos_j=3-i,8-j
                ret=1
                n_piezas=n_piezas+1
    return ret, n_piezas, (pos_i,pos_j)
```

- ***Algoritmos de ordenación de la estantería***

La posibilidad de conocer el tipo de objeto, el número de estas y su posición en la estantería otorga la posibilidad de ordenar la estantería. Para ello primero se crea una matriz de estado ordenado mediante algoritmos de intercambio de valores.

Después, se replica esta operación en el simulador utilizando intercambio de objetos entre dos posiciones. La base auxiliar se utilizará como lugar donde se guardará un objeto provisionalmente, esto se debe a que el brazo robot solo puede tener un objeto simultáneamente.

```
def cambios(mat,mat_n):
    for i in range(4):
        for j in range(9):
            if mat_n[3-i][8-j]!=mat[3-i][8-j]:
                _,_,(i_n,j_n)=funciones.localizar_pieza(mat,mat_n[3-i][8-j])
                if mat[3-i][8-j]==0:
                    estanteria(i_n,j_n,False)
                    estanteria(3-i,8-j,True)
                else:
                    intercambio(3-i,8-j,i_n,j_n)
                aux=mat[3-i][8-j]
                mat[3-i][8-j]=mat[i_n][j_n]
                mat[i_n][j_n]=aux
```

- Función auxiliar intercambio: esta función utiliza dos posiciones de la estantería como entrada. Recoge el objeto de la primera posición dejándolo en la base auxiliar, después recoge el segundo objeto mediante la información de posición y lo deja en el hueco donde estaba originalmente la primera pieza. Cuando deja este objeto, recoge el que había dejado en la base auxiliar y lo coloca en la posición que ocupaba el objeto que acababa de colocar.

```
def intercambio(i1,j1,i2,j2):
    estanteria(i1,j1,False)
    dejar_pieza(True)
    estanteria(i2,j2,False)
    estanteria(i1,j1,True)
    dejar_pieza(False)
    estanteria(i2,j2,True)
```

3.2.3 Fase 3: Diseño de la interfaz

Para simplificar la ejecución de pruebas de la célula robotizada, se le añade una interfaz gráfica con la que se tiene control de funciones mencionadas en apartados anteriores. Esta interfaz será bastante sencilla y para su creación se usará la biblioteca tkinter.

3.2.3.1 Operario

El menú de operario (Figura 3.26) es el que tendrá acceso a la parte de gestión de la estantería, ordenar, generar objetos. También estarán las funciones de simulación como iniciar o parar la simulación.

Figura 3.26 : Menú Operario

- ***Generar estantería aleatoria***

Función encargada que generar objetos en la estantería. Estos objetos se escogerán aleatoriamente (con la posibilidad de generar un hueco en vez de un objeto) para ahorrar tiempo a la hora de realizar pruebas en la estantería.

Requiere que la simulación este corriendo porque los objetos duplicados se mantendrán como parte de diseño de escena y si se para la simulación no desaparecerán.

- ***Generar pieza aleatoria***

A diferencia de la anterior, si solo se desea realizar una entrada de objeto en la escena se utilizará esta función. El objeto que aparecerá también será escogido aleatoriamente, pero a diferencia con la anterior, no existe la opción de generar hueco.

Igual que antes es imperativo que la escena este ejecutándose a la hora que generar duplicados de los objetos.

- ***Ordenar estantería***

Esta función se encarga de realizar una ordenación de los objetos de la estantería. Primero se realiza se obtiene una matriz del estado de la estantería mediante el uso de la red neuronal y posteriormente, según el tipo de orden definido en la pantalla de la Figura 3.27 se realiza la operación mediante el uso del brazo robot.

Figura 3.27 : Opciones al seleccionar ordenar estantería en el menú Operario

Se han programado 2 formas de ordenar las piezas; la primera según el valor numérico de los objetos (huecos, objetos tipo 1, objetos tipo 2, …) y se selecciona como orden tipo 0 (Figura 3.28-b). La segunda configuración seria ordenar por el color y se seleccionaría con orden tipo 1 (Figura 3.28-c).

Figura 3.28-a: Estantería desordenada **Figura 3.28-b: Orden 0 (centro)** **Figura 3.28-c Orden 1 (derecha)**

Figura 3.28 : Tipos de orden de estanterías

- ***Aleatorizar***

Para añadir otro punto de variedad a la escena se ha programado esta función, encargada de establecer una posición aleatoria a la luz, cámara enfocada al Summit y al propio Summit.

- ***Acceso a la imagen de cámaras***

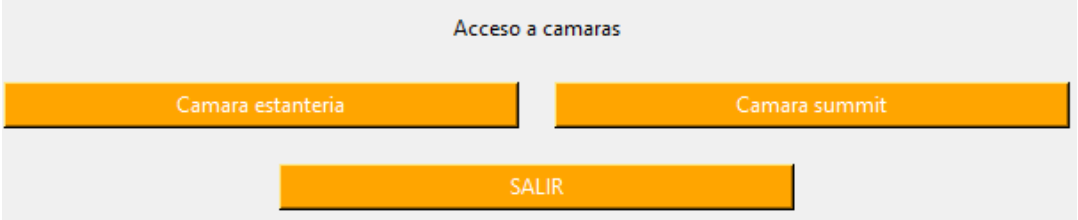


Figura 3.29 : Opciones al seleccionar acceso a cámaras

Esta función (Figura 3.29) realiza una captura mediante una de las dos cámaras y transmite la imagen a la GUI. Esta imagen esta reescalada a 256x256 a la hora de ser representada en la pantalla.

- ***Menu de Summit***

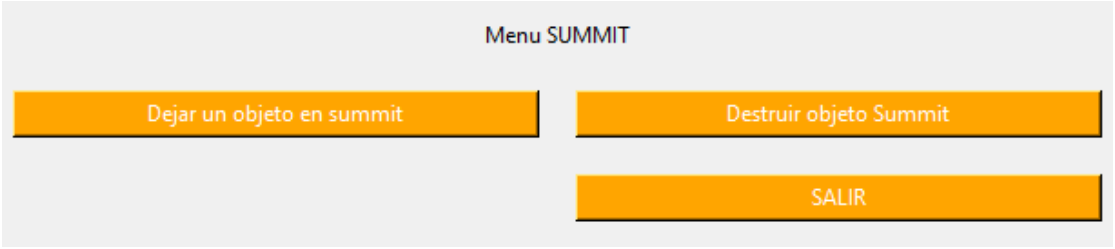


Figura 3.30 : Opciones al seleccionar menú Summit

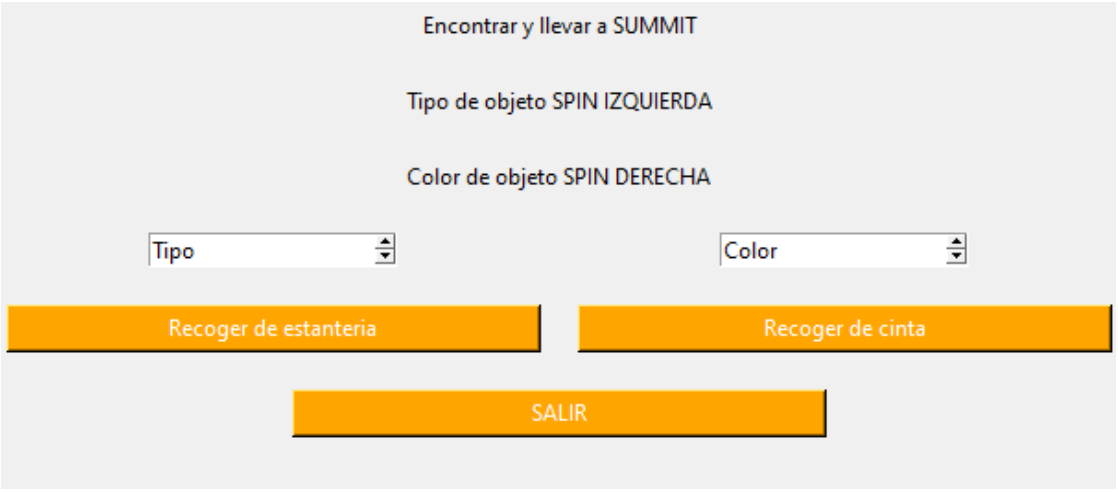


Figura 3.31 : Opciones al seleccionar dejar un objeto en Summit del menú Summit

Esta opción abre una ventana (Figura 3.30) donde se puede elegir guardar un objeto del estante o de la cinta o eliminar el objeto que este situado en la cesta del Summit. A la hora de

buscar un objeto de la estantería es necesario especificar el tipo de objeto y su color como se puede ver en la Figura 3.31.

Si no especifica ninguna pieza o esta no está, el brazo robot no realizará y se volverá al menú del operario.

- ***Inicio y parada de simulación***

Figura 3.32 : Botones que permiten el inicio (Start simulation) y la parada de la simulación (Stop simulation)

Permite el inicio y la parada de la simulación (Figura 3.32).

3.2.3.2 Cliente

En el menú del cliente (Figura 3.33) aparecen funciones relativas al estado de la estantería y la entrada y salida de objetos.

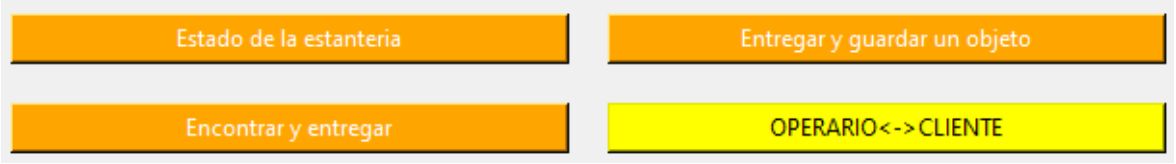


Figura 3.33 : Menú cliente

- ***Estado de estantería***

Muestra la matriz de estado de la estantería en la pantalla de programa con los valores numéricos de cada objeto y con el valor 0 para representar los huecos. Se puede ver un ejemplo en la Figura 3.34

Figura 3.34 : Representación en forma de matriz del estado de la estantería

- ***Encontrar un objeto y entregar en la base auxiliar***

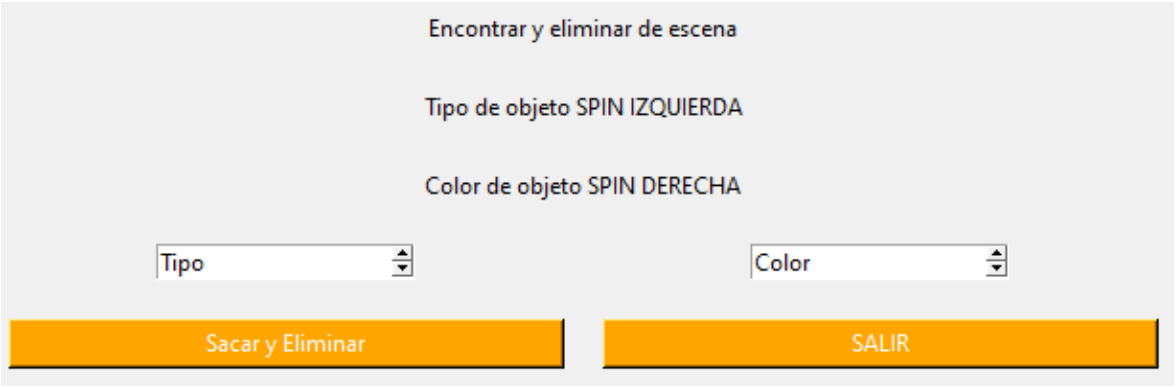


Figura 3.35 : Menú para seleccionar el objeto para recoger de estantería y llevar a la base auxiliar

Esta opción abre un selector del tipo de objeto y color (Figura 3.35), con una ventana muy parecida a de guardar en Summit (Figura 3.31) salvo que no cuenta con la opción de recoger de la cinta. Posteriormente el objeto se elimina de la escena.

- ***Entregar, recoger y guardar un objeto***

Esta función realiza una entrega de un objeto aleatorio en la estantería. Primero se genera el objeto, después lo recoge el robot siempre que haya espacio en la estantería, y lo guarda en el primer hueco que esté disponible.

3.3 Diseño de redes neuronales

3.3.1 Red de clasificación

El propósito de la primera red neuronal es la identificación del objeto que aparece en la imagen que se le ofrece como entrada. Para ello se utilizan imágenes de reducida dimensión (30x15x3) y como salida se obtiene un vector de distribución de probabilidad de 10 posiciones.

$$f_{RED}(imagen) = (p_0, p_1, p_2, \dots, p_9) \quad (3.1)$$

$p_i \rightarrow$ probabilidad del tipo de objeto que aparece en la imagen

3.3.1.1 Diseño

La red cuenta con dos etapas de capa de convolución seguida de capa de *MaxPooling*. La primera realizara 32 convoluciones y 64 la segunda. En cuanto al *MaxPooling* el tamaño del filtro es de (2,2). A esto le sigue una capa de aplanamiento que convierte en un solo vector todas las imágenes generadas en las capas previas.

Posteriormente va una capa normal de red neuronal de 256 neuronas con función RELU, seguida la última capa que da el resultado formada por 10 neuronas de salida con SOFTMAX

como función de activación. En la Figura 3.36 se puede observar un esquema general de la red.

La función de pérdida será entropía cruzada categórica y será definida como 'categorical_crossentropy' y el optimizador Adam con una tasa de aprendizaje inicial de 0.0005.

Líneas de código para definir la red en Python (resumen de la red en la Tabla 3.1 y un esquema de esta en la Figura 3.36):

```
altura, longitud=30,15
tamano_filtro1=(3,3)
tamano_filtro2=(2,2)
tamano_pool=(2,2)
clases=10
lr=0.0005

cnn=Sequential()

cnn.add(Convolution2D(filtrosConv1,tamano_filtro1,padding='same',in-
put_shape=(altura,longitud,3),activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))
cnn.add(Convolution2D(filtrosConv2,tamano_filtro2,padding='same',activa-
tion='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Flatten())
cnn.add(Dense(256,activation='relu'))
cnn.add(Dropout(0.2))
cnn.add(Dense(clases,activation='softmax'))

cnn.compile(loss='categorical_crossentropy',optimizer=tensorflow.keras.op-
timizers.Adam(lr=lr),metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 15, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 7, 32)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 15, 7, 64)         8256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 3, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1344)              0
_________________________________________________________________
dense (Dense)                (None, 256)               344320
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 356,042
Trainable params: 356,042
Non-trainable params: 0
```

Tabla 3.1 : Resumen de modelo de red 1 al aplicar la función summary()

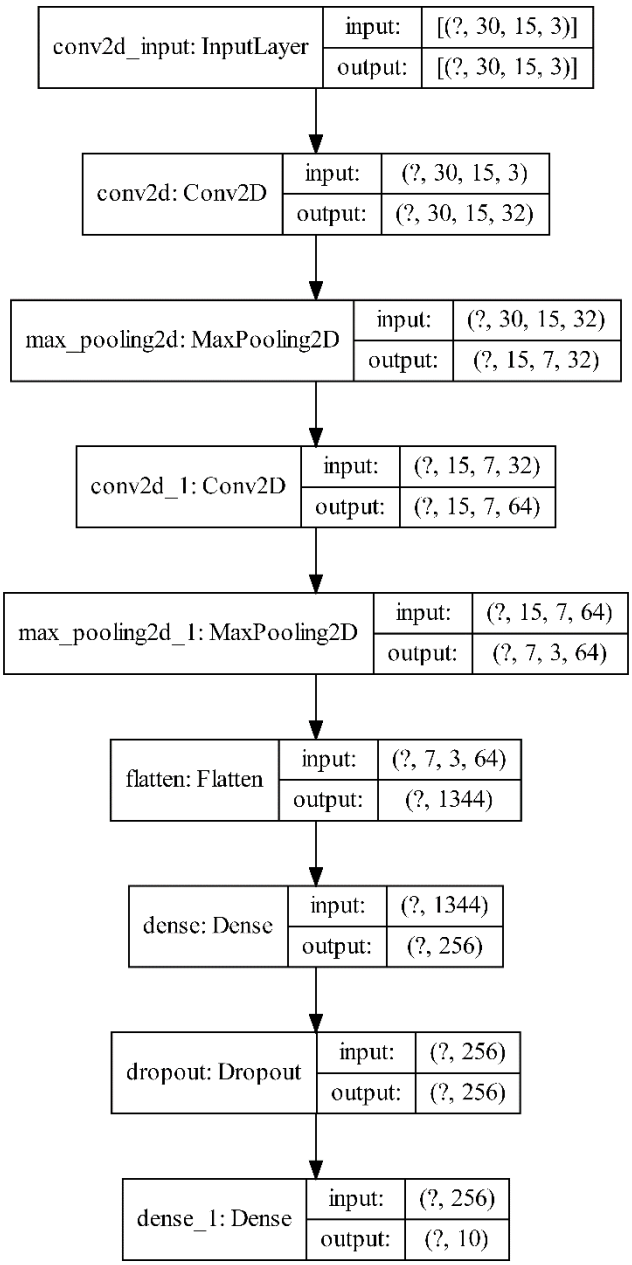


Figura 3.36 : Diagrama de la red

3.3.1.2 Entrenamiento

La primera parte del entrenamiento es la adquisición de datos. Para ello, se ha realizado un programa centrado en montar diferentes configuraciones de la estantería (mediante el posicionamiento de botellas, latas y huecos y variando la posición de luz). Después se guardaba una captura y un archivo de texto con los valores reales de la configuración de la estantería (Figura 3.37). Para parte del conjunto de imágenes se usó el brazo robot para colocar los diferentes objetos frente la cámara y generar otra perspectiva más, véase como ejemplo la Figura 3.38.

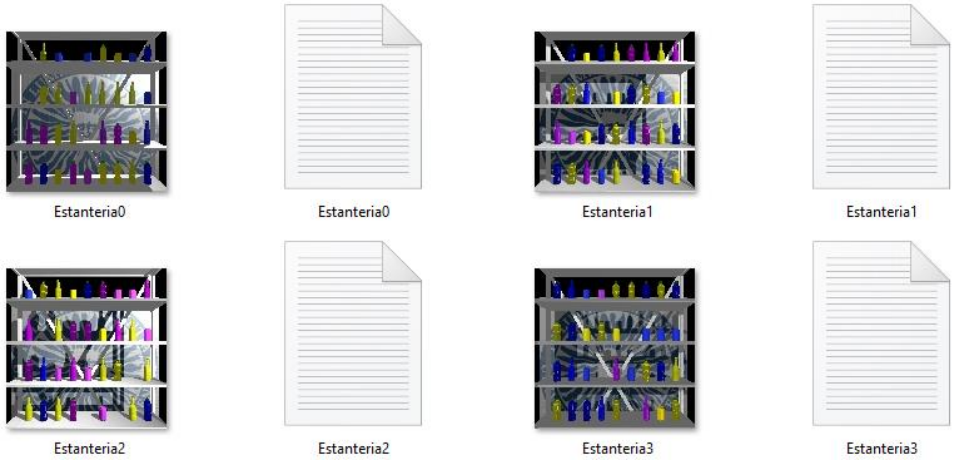


Figura 3.37 : Las imágenes de estanterías junto con los documentos de la configuración de cada imagen

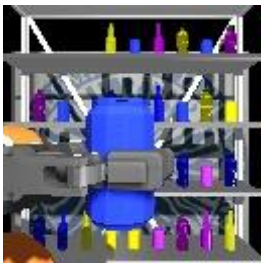


Figura 3.38 : Imagen del conjunto de datos utilizado en el entrenamiento, objeto tipo 1 frente a cámara

Con ello se generó un conjunto de datos con el que entrenar la red. Este proceso fue iterativo, se generaban datos a la vez que se entrenaba una red con los datos obtenidos hasta conseguir disminuir el error a un valor considerable.

Para aumentar la eficiencia del aprendizaje la primera capa densa (capa de 256 neuronas) desactiva aleatoriamente el 50% de sus neuronas mediante el uso de capa *Dropout*. Esto afianza diferentes enlaces entre neuronas.

Además, las imágenes pasan por algunas deformaciones como zoom y giro, lo cual también mejora el entrenamiento y hace que disminuya el tamaño del conjunto de datos.

En la Figura 3.39 se puede ver la gráfica que muestra la exactitud del modelo durante el entrenamiento (train) y la validación (test), en cuanto a la Figura 3.40, la gráfica muestra la evolución del error. Estas graficas permiten evaluar el método del entrenamiento, ver si hay sobreajuste, el número de épocas necesarios o para definir el parámetro de la tasa de aprendizaje inicial. Para evaluar cuando aparecer el sobreajuste se puede mirar el progreso del error de validación, en el momento en que comienza a subir es que se está cometiendo un sobreajuste.

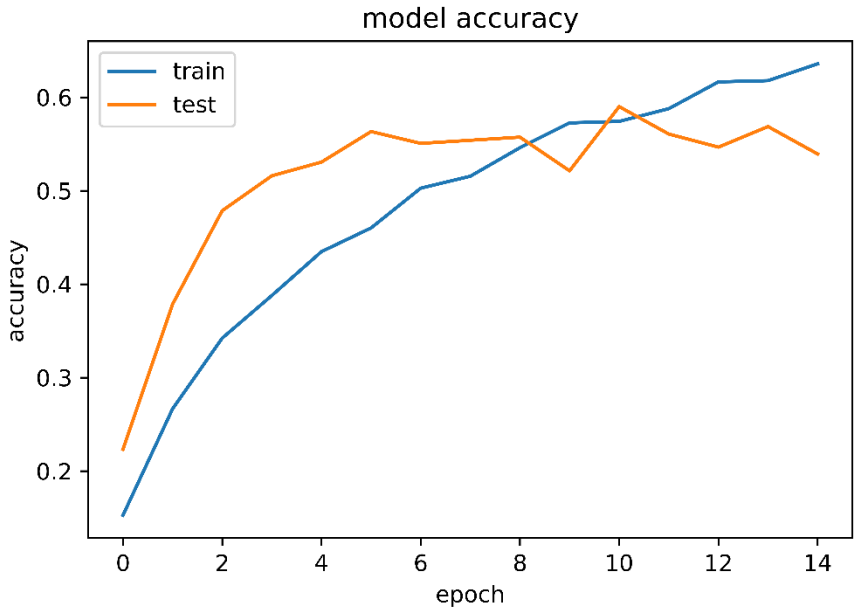


Figura 3.39 : Evolución de entrenamiento y la validación utilizando como métrica exactitud

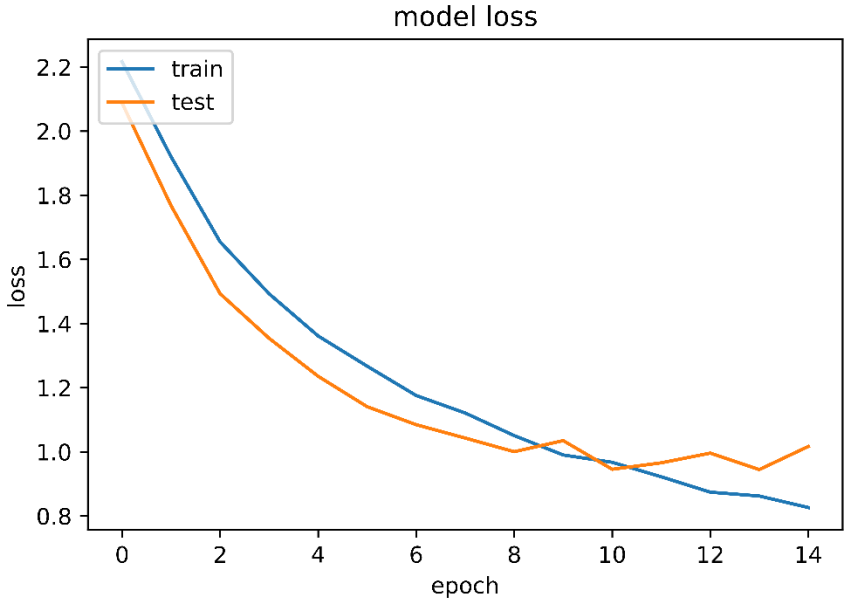


Figura 3.40 : Evolución de entrenamiento y la validación utilizando como métrica la función de coste

3.3.1.3 Resultados

A la hora de creación de ambas redes neuronales, estas pasaron por varios modelos y al igual que por varios métodos diferentes de aprendizaje (diferentes optimizadores, funciones de activación, …) hasta lograr un modelo que cumpla el cometido con un error practicable. Después del entrenamiento, estos fueron puestos a prueba en el simulador.

Se han realizado pruebas a 6 redes distintas. Todas partían del diseño original que contenían dos capas de convolución y MaxPooling, una capa de aplanamiento, una capa densa y

finalmente una capa de salida con 10 neuronas. La diferencia entre ellas radicaba en el número de neuronas en la primera capa densa.

En total se hicieron 70 ensayos a todas las redes. Donde se contó el número de veces que la red se equivocó en una identificación. Hay que tener en cuenta que cada ensayo era una estantería de 9x4 objetos (36 identificaciones por ensayo). También se midió el tiempo que tarda una red en analizar una estantería completa. Aquí no hubo mucha diferencia, dependía más de las ejecuciones en paralelo del propio ordenador.

Numero de neuronas	Fallos	Tiempo medio (s)	Número de parámetros
32	6	1.87	52 522
64	0	1.81	95 882
128	1	1.91	182 602
256	1	1.93	356 042
512	1	1.99	702 922
1024	5	1.91	1 396 682

Tabla 3.2 : Comparativa entre diferentes modelos de identificación

Como se ve en la Tabla 3.2, la red de 64 nodos es la que mejor resultado ha dado. Con esta red se realizaron 140 ensayos adicionales donde solo se cometió un fallo, dando un error de 0.0132% (1 fallo /(140 ensayos * 36 objetos por ensayo)).

3.3.2 Red de localización

A diferencia de la red anterior, esta no está diseñada para la clasificación de objetos si no que se trata de un problema de regresión. El objetivo es dar la posición del Summit mediante la imagen del robot en el escenario. Este tipo de tareas es bastante más complicado porque la cámara no es fija, los puntos de referencia están a diferente altura (hay distancia en eje z de las señales del suelo y señal de la plataforma del Summit). Además, se une el factor de que las neuronas cuentan con una función no lineal.

$$f_{RED}(imagen) = (x, y) \tag{3.2}$$

$x, y \rightarrow$ coordenadas predichas de la posicion del Summit

Por ello, hay dos cosas para tener en cuenta. La primera es el tamaño de imagen, esta se tiene que intentar mantener lo más alto posible esto significa que la entrada de red se mantendrá tal como lo da la imagen original (128x128 pixeles) y el número de capas MaxPooling mínimo con el filtro más reducido posible. Esto es debido a que no se debe realizar ninguna distorsión de la imagen original al trabajar con dimensiones de la escena.

La segunda es que los valores de salida de la red dan una posición bidimensional. Esto implica que hay que tener en cuenta cómo está acotada la salida de la red por ello a la salida directamente se utilizara una función lineal como función de activación.

3.3.2.1 Diseño

El diseño de esta red también cuenta con dos capas de convolución y MaxPooling, pero no se realiza un escalado de la imagen previo. Por ello, la imagen de entrada mantendrá la dimensión original 128x128.

Después se pasa a un aplanado y una red convencional de 6 capas donde la última no cuenta con función de activación y será la encargada de devolver el valor de posición x e y del Summit en la imagen. La función error será el error cuadrático medio y optimizador Adam.

Líneas de código para definir la red en Python (resumen de la red en la Tabla 3.3 y un esquema de esta en la Figura 3.42):

```
altura, longitud=128,128
tamano_filtro1=(3,3)
tamano_filtro2=(2,2)
tamano_pool=(2,2)
lr=0.00001
filtrosConv1=32
filtrosConv2=64

cnn=Sequential()
cnn.add(Conv2D(filtrosConv1,tamano_filtro1,padding='same',input_shape=(al-
tura,longitud,3),activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))
cnn.add(Conv2D(filtrosConv2,tamano_filtro2,padding='same',activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Flatten())
cnn.add(Dense(128,activation='relu'))
cnn.add(Dense(256,activation='relu'))
cnn.add(Dense(512,activation='relu'))
cnn.add(Dense(256,activation='relu'))
cnn.add(Dense(128,activation='relu'))
cnn.add(Dense(2,activation='linear'))

cnn.compile(loss='mean_squared_error',optimizer=tensorflow.keras.optimiz-
ers.Adam(lr=lr),metrics=['mean_absolute_error'])
```

3.3.2.2 Entrenamiento

Al igual que para el entrenamiento de red de clasificación ha sido necesario generar un conjunto de datos para entrenar la red. Este conjunto de datos se ha realizado mediante la captura del Summit utilizando la cámara enfocada en el aparcamiento. La posición de la cámara y la iluminación se ha ido seleccionando aleatoriamente en cada prueba al igual que la posición y orientación del Summit (Figura 3.41).

Por cada imagen se ha guardado la posición en la que se encontraba el robot móvil en un bloc de notas que se ha utilizado para entrenar la red.

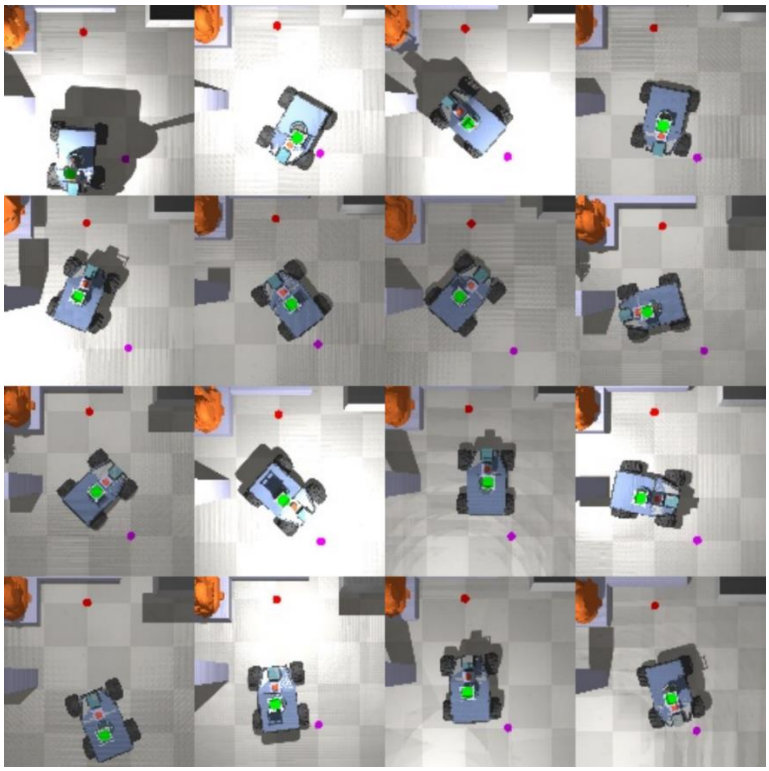


Figura 3.41 : Parte del conjunto de datos utilizados para entrenar a la red

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	896
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_1 (Conv2D)	(None, 64, 64, 64)	8256
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 64)	0
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 128)	8388736
dense_1 (Dense)	(None, 256)	33024
dense_2 (Dense)	(None, 512)	131584
dense_3 (Dense)	(None, 256)	131328
dense_4 (Dense)	(None, 128)	32896
dense_5 (Dense)	(None, 2)	258

Tabla 3.3 : Resumen de modelo de red 2 al aplicar la función summary()

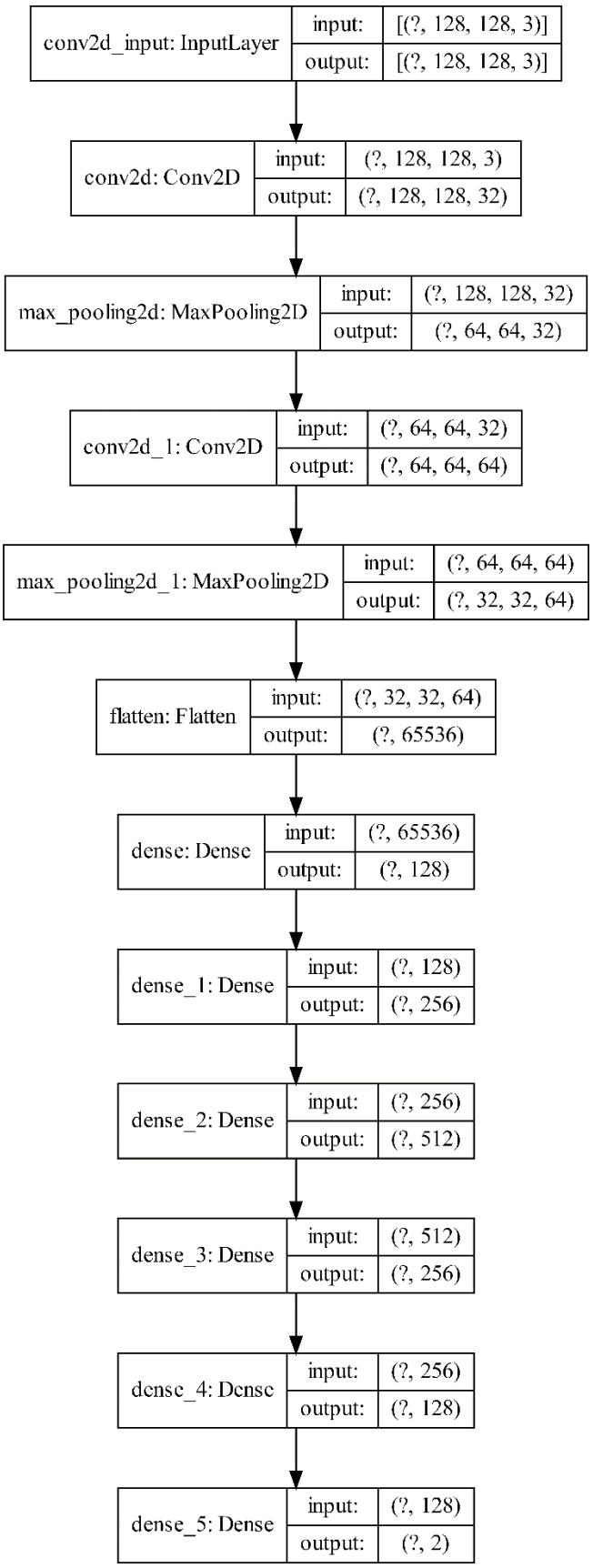


Figura 3.42 : Esquema de la red de localización

De 8997 imágenes generadas, se han utilizado 5000 imágenes de entrenamiento y 2000 imágenes de validación para el ajuste de hiperparámetros durante el aprendizaje, y el resto se ha utilizado para realizar comprobación del funcionamiento de la red.

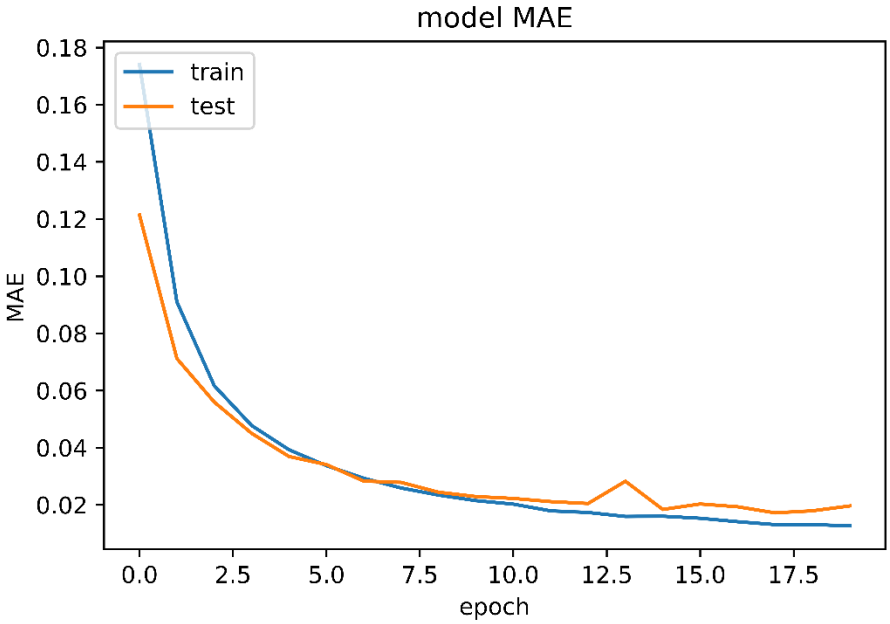


Figura 3.43 : Evolución de entrenamiento y la validación utilizando como métrica el error medio absoluto

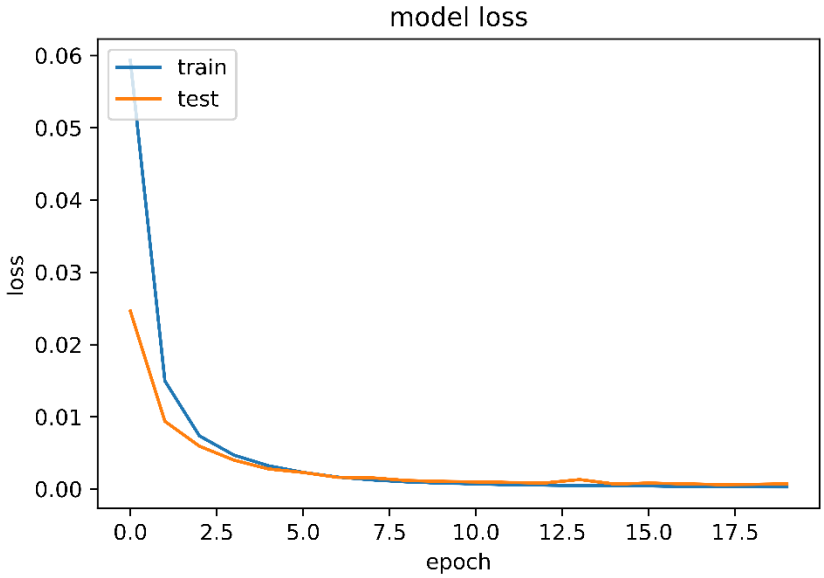


Figura 3.44 : Evolución de entrenamiento y la validación utilizando como métrica la función de coste

Al igual que en la primera red, se ha ido analizando el proceso de entrenamiento mediante la evolución de los errores cometidos a la hora de entrenar y validar. Así, realizar modificaciones en los hiperparámetros iniciales para poder optimizar el proceso de aprendizaje o si es necesario, realizar cambios en la red. La grafica de la Figura 3.43 muestra el error medio

absoluto, mientras que la gráfica de la Figura 3.44 muestra el error que se ha usado como función de coste, en este caso es el error cuadrático medio.

3.3.2.3 Resultados

En cuanto a la red de localización del Summit, el error se ha calculado con la diferencia entre el punto real y el punto predicho por la red. Entre todas las redes que se crearon, se han seleccionado las que mejor resultado han dado. Estas fueron la red original más 5 variantes de esta.

Aquí se han utilizado dos métricas (Tabla 3.4). La primera es el porcentaje de momentos de peligro, donde se anotaba el ensayo donde el error superaba el umbral de seguridad (0.05m). El segundo es el error relativo al tamaño de la zona de aparcamiento. Se ha realizado 100 ensayos.

Nombre red	Ensayo peligro (%)	Error relativo (%)	Tiempo(ms)	Nº de parámetros
3 Conv + MaxPool	4	7.17744	25.803	17,640,418
No sequencial	5	7.273	24.960	8,727,266
3 Doble Conv + concatenated	0	4.9289	41.567	46,525,154
DropOut	6	7.5239	20.219	4,565,090
Doble Conv	18	9.32306	22.580	8,728,642
Original	5	6.57448	20.937	8,726,978

Tabla 3.4 : Comparativa entre diferentes modelos de localización

La red con menos error es la red 3 ‘3 Doble Conv + concatenated’ que cuenta con una estructura de capas dobles de convolución (capa seguida de otra capa de convolución), las salidas de ambas capas se concadenan y se realiza MaxPooling Esta operación se repite dos veces más, pero añadiendo al concadenado la salida de MaxPooling anterior. Después se pasa a un aplanamiento y dos capas densas de 512 neuronas más capa de Dropout. Finalmente, la capa de salida de dos neuronas (véase el resumen de la red en la Tabla 3.5).

Esta red es la más lenta de procesar, superando casi en doble de tiempo a las demás redes por la cantidad de capas e interconexiones que contiene. Se utilizaron los saltos entre capas para conseguir un paso en paralelo de la información y así evadir el degradado del gradiente del error.

Layer (type)	Output Shape	Param #	Connected to
img (InputLayer)	[(None, 128, 128, 3)	0	
conv2d_6 (Conv2D)	(None, 128, 128, 32)	416	img[0][0]
conv2d_7 (Conv2D)	(None, 128, 128, 64)	8256	conv2d_6[0][0]
concatenate_3 (Concatenate)	(None, 128, 128, 96)	0	conv2d_6[0][0] conv2d_7[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 64, 64, 96)	0	concatenate_3[0][0]
conv2d_8 (Conv2D)	(None, 64, 64, 64)	24640	max_pooling2d_3[0][0]
conv2d_9 (Conv2D)	(None, 64, 64, 64)	16448	conv2d_8[0][0]
concatenate_4 (Concatenate)	(None, 64, 64, 224)	0	conv2d_8[0][0] conv2d_9[0][0] max_pooling2d_3[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 32, 32, 224)	0	concatenate_4[0][0]
conv2d_10 (Conv2D)	(None, 32, 32, 64)	57408	max_pooling2d_4[0][0]
conv2d_11 (Conv2D)	(None, 32, 32, 64)	16448	conv2d_10[0][0]
concatenate_5 (Concatenate)	(None, 32, 32, 352)	0	conv2d_10[0][0] conv2d 11[0][0] max_pooling2d_4[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 352)	0	concatenate_5[0][0]
flatten_1 (Flatten)	(None, 90112)	0	max_pooling2d_5[0][0]
dense_3 (Dense)	(None, 512)	46137856	flatten_1[0][0]
dropout_2 (Dropout)	(None, 512)	0	dense_3[0][0]
dense_4 (Dense)	(None, 512)	262656	dropout_2[0][0]
dropout_3 (Dropout)	(None, 512)	0	dense_4[0][0]
dense_5 (Dense)	(None, 2)	1026	dropout_3[0][0]

Tabla 3.5 : Resumen de modelo de red al aplicar la función summary()

4. Conclusiones y trabajos futuros

En esta parte del documento se exponen las conclusiones que se han extraído durante el desarrollo del TFG y trabajos futuros que podrían ampliar el estudio de las redes neuronales y encontrar posibles nuevas aplicaciones para estas en el campo industrial.

4.1 Conclusiones

Durante el desarrollo del TFG se ha experimentado con diferentes modelos de redes, como también de diferentes métodos. Consiguiendo que el modelo desarrollado para la clasificación de objetos permita una confianza aproximada del 99.9%. En cuanto a la segunda red no se han llegado a registrar ningún caso donde fallara la colocación del objeto dentro de área designada en el Summit. Los resultados obtenidos de los modelos de redes para la identificación y localización han probado un comportamiento aceptable para realizar las funciones para las que han sido diseñadas, hay que tener en cuenta los casos en los que es imprescindible un error nulo.

Una de las limitaciones que se pueden ver a primera vista es la cantidad de datos para el entrenamiento que se requieren. Durante la fase de diseño se ha buscado la forma de poder generar datos donde sea posible localizar la solución deseada (principalmente en el caso de localización, por ejemplo, mediante puntos de referencia).

También, se va a mencionar el programa CoppeliaSim. Este programa ha resultado ser muy competitivo y versátil para el campo de la robótica. La creación de la célula fue sencilla, pudiendo modificar con mucha facilidad muchos elementos sin apenas complicación. El número de funciones de B0 API remota para Python ha permitido realizar un control en casi todos los aspectos de la escena robotizada y la documentación de internet permitió un acceso a la información de todas estas funciones de una manera rápida.

4.2 Futuros trabajos

- Ampliar zona de localización de la zona de parada del Summit (o probar con otros ejemplos). Esto implicaría aumentar la resolución de imagen de localización de entrada a la red, una mayor entrada de datos y parámetros que entrenar.
- Probar preentrenamiento de redes mediante aprendizaje no supervisado para diseñar una red de clasificación. Se usaría la técnica de autocodificadores usando imágenes de la misma clase como entrada y salida. Después esta red se podría partir en un codificador y decodificador. El codificador seria la red preentrenada.
- Realizar modelos de identificación con un número mayor de objetos, perspectivas y aumentar resolución.
- Aplicar aprendizaje reforzado para el diseño de un control del robot para, por ejemplo, organizar la estantería o definir los movimientos del brazo (sin programar

directamente las secuencias de movimiento). Todo ello requerirá implementar una óptima política de recompensa.

5. Bibliografía

[1] «Demain des usines pilotées par internet,» [En línea]. Disponible en: http://www.challenges.fr/high-tech/20130305.CHA6937/demain-des-usinespilotees-par-internet.html. [Último acceso: 27 junio 2020].

[2] «Inteligencia artificial. Qué es y por qué es importante,» [En línea]. Disponible en: https://www.sas.com/es_es/insights/analytics/what-is-artificialintelligence.html. [Último acceso: 27 junio 2020].

[3] S. Behnke, "Hierarchical Neural Networks for Image Interpretation.", Springer, 2003. ISBN 978-3-540-45169-3

[4] R. O. Duda, P. E. Hart y D. G. Stork, Pattern Classification and scene analysis (2nd ed.) Part 1: Pattern clasification, John Wiley & Sons, Inc, 2001. ISBN 978-0-471-05669-0

[5] «CoppeliaSim,» [En línea]. Disponible en: https://www.coppeliarobotics.com. [Último acceso: 20 junio 2020].

[6] «Remote API functions (Python),» [En línea]. Disponible en: https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm. [Último acceso: 20 junio 2020].

[7] T.S.Huang, «Computer Vision: Evolution and Promise» *19th CERN School of Computing*, pp. 21-25, 1996. ISBN 9290830956

[8] E. R. Kandel, J. H. Schwartz y T. M. Jessell, Principles of Neural Science, McGraw-Hill, 1981.

[9] W. P. Warren McCulloch, de *A logical calculus of the ideas immanent in nervous activity*, pp. 115-133, BULLETIN OF MATHEMATICAL BIOPHYSICS vol 5, 1943.

[10] D. Hebb, de *The Organization of Behavior*, Psychology Press, 1949. ISBN 978-0805843002

[11] F. Rosenblatt, *The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain*, Psychological Review vol. 65 No 6, pp. 386-408, 1958.

[12] D. Hubel y T.N.Wiesel, «Receptive fields of single neurones in the cat's striate cortex,» *J Physiol, vol 148 No 3* pp. 574-591, 1959.

[13] B. Widrow, «An adaptive "ADALINE" neuron using chemical "memistors"» Stanford Electronics Laboratories, 1960. ISBN:9781783555130

[14] M. Minsky y S. Papert, «Perceptrons: an introduction to computational geometry,» The MIT Press, 1969. ISBN 978-0262534772

[15] P. J. Werbos, Tesis Harvard University *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, 1974.

[16] G. E. H. y. J. W. David E. Rumelhart, «Learning representations by back-propagating errors,» *Nature vol 323,* p. 533–536, 1986.

[17] B. D. H. H. H. J. LeCun, «Backpropagation applied to handwritten zip code recognition,» *Neural Computation,* pp. 541-551, 1989.

[18] «Gazebo,» [En línea]. Available: http://gazebosim.org/ . [Último acceso: 12 noviembre 2020].

[19] «Microsoft Robotics Studio 4,» Microsoft, [En línea]. Disponible en: https://www.microsoft.com/en-us/download/details.aspx?id=29081. [Último acceso: 12 noviembre 2020].

[20] «Caracteristicas clave,» anyKode, [En línea]. Disponible en: http://www.anykode.com/mariloukeyfeatures.php. [Último acceso: 12 noviembre 2020].

[21] Y. B. A. C. Ian Goodfellow, Deep Learning, MIT Press, 2016 .

[22] D. P. Kingma y J. L. Ba, «ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION,» 2017. [En línea]. Disponible en: https://arxiv.org/abs/1412.6980.

[23] «API remota basada en BØ,» Coppelia Robotics, [En línea]. Disponible en: https://www.coppeliarobotics.com/helpFiles/en/b0RemoteApiOverview.htm. [Último acceso: 20 julio 2020].

[24] «B0-based remote API, Python,» Coppelia Robotics, [En línea]. Disponible en: https://www.coppeliarobotics.com/helpFiles/en/b0RemoteApi-python.htm. [Último acceso: 13 julio 2020].

[25] «Individual Edition (pagina de descarga),» Anaconda, [En línea]. Disponible en: https://www.anaconda.com/products/individual. [Último acceso: 13 junio 2020].

[26] TensorFlow, «TensorFlow 1.0.0,» Github, [En línea]. Disponible en: https://github.com/tensorflow/tensorflow/blob/07bb8ea2379bd459832b23951fb20ec47f3fdbd4/RELEASE.md. [Último acceso: 29 agosto 2020].

[27] «¿Por que TensorFlow?,» TensorFlow, [En línea]. Disponible en: https://www.tensorflow.org/about/case-studies?hl=es-419. [Último acceso: 2 junio 2020].

[28] «Effective TensorFlow 2,» TensorFlow, [En línea]. Disponible en: https://www.tensorflow.org/guide/effective_tf2.

[29] «Keras layers API,» Keras, [En línea]. Disponible en: https://keras.io/api/layers/. [Último acceso: 20 julio 2020].

[30] «The Sequential model,» Keras, [En línea]. Disponible en: https://keras.io/guides/sequential_model/. [Último acceso: 20 julio 2020].

[31] «Acerca de OpenCV,» OpenCV, [En línea]. Disponible en: https://opencv.org/about/. [Último acceso: 21 junio 2020].

[32] Harris, CR, Millman, KJ, van der Walt, SJ y col, «Array programming with NumPy,» *Nature vol 585*, pp. 357 - 362, 2020.

[33] «Index (indice de funciones),» Numpy, [En línea]. Disponible en: https://numpy.org/doc/stable/genindex.html. [Último acceso: 20 julio 2020].

[34] «tkinter — Python interface to Tcl/Tk,» Python, [En línea]. Disponible en: https://docs.python.org/3/library/tkinter.html. [Último acceso: 25 agosto 2020].

[35] «TensorFlow,» [En línea]. Disponible en: https://www.tensorflow.org/. [Último acceso: 18 junio 2020].

6. Anexos

6.1 Scripts

6.1.1 Script funciones

```
import b0RemoteApi
import numpy as np
import time
import cv2

def mover(client,joint,mov):
    margen=0.01
    _,MotorUD=client.simxGetObjectHandle('MotorUD',client.simxServiceCall())
    if joint==MotorUD:
        margen=0.1
    client.simxSetJointTargetPosition(joint, mov,client.simxDefaultPublisher())
    while(client.simxGetJointPosition(joint,client.simxServiceCall())[1]<mov-
margen)or(client.simxGetJointPosition(joint,client.simxService-
Call())[1]>mov+margen):
        client.simxSetJointTargetPosition(joint, mov,client.simxDefault-
Publisher())

def moverB2y3(client,mov):
    margen=0.001
    _,Brazo2=client.simxGetObjectHandle('IRB140_joint2',client.simxService-
Call())
    _,Brazo3=client.simxGetObjectHandle('IRB140_joint3',client.simxService-
Call())
    client.simxSetJointTargetPosition(Brazo2, mov,client.simxDefault-
Publisher())
    client.simxSetJointTargetPosition(Brazo3, -mov,client.simxDefault-
Publisher())

    while((client.simxGetJointPosition(Brazo2,client.simxServiceCall())[1]<mov-
margen)or(client.simxGetJointPosition(Brazo2,client.simxService-
Call())[1]>mov+margen))or((client.simxGetJointPosition(Brazo3,client.simx-
ServiceCall())[1]<-mov-margen)or(client.simxGetJointPosition(Brazo3,cli-
ent.simxServiceCall())[1]>-mov+margen)):
        client.simxSetJointTargetPosition(Brazo2, mov,client.simxDefault-
Publisher())
        client.simxSetJointTargetPosition(Brazo3, -mov,client.simxDefault-
Publisher())

def moverB235(client,a,b):
    margen=0.001
    q2=(a-90)*(2*3.14)/360
    q3=(b+90)*(2*3.14)/360
    q5=(-(a-90)-(b+90))*(2*3.14)/360
    _,Brazo2=client.simxGetObjectHandle('IRB140_joint2',client.simxService-
Call())
    _,Brazo3=client.simxGetObjectHandle('IRB140_joint3',client.simxService-
Call())
    _,Brazo5=client.simxGetObjectHandle('IRB140_joint5',client.simxService-
Call())
    client.simxSetJointTargetPosition(Brazo2, q2,client.simxDefaultPublisher())
    client.simxSetJointTargetPosition(Brazo3, q3,client.simxDefaultPublisher())
    client.simxSetJointTargetPosition(Brazo5, q5,client.simxDefaultPublisher())

    p2=client.simxGetJointPosition(Brazo2,client.simxServiceCall())[1]
    p3=client.simxGetJointPosition(Brazo3,client.simxServiceCall())[1]
    p5=client.simxGetJointPosition(Brazo5,client.simxServiceCall())[1]
    while((p2<q2-margen)or(p2>q2+margen))or((p3<q3-
margen)or(p3>q3+margen))or((p5<q5-margen)or(p5>q5+margen)):
        p2=client.simxGetJointPosition(Brazo2,client.simxServiceCall())[1]
        p3=client.simxGetJointPosition(Brazo3,client.simxServiceCall())[1]
```

```
        p5=client.simxGetJointPosition(Brazo5,client.simxServiceCall())[1]

        client.simxSetJointTargetPosition(Brazo2, q2,client.simxDefault-
Publisher())
        client.simxSetJointTargetPosition(Brazo3, q3,client.simxDefault-
Publisher())
        client.simxSetJointTargetPosition(Brazo5, q5,client.simxDefault-
Publisher())

def pos_aleatorios(client,Light,Cam_2,summit):
    i=np.random.randint(100, size=3)
    j=np.random.randint(100, size=3)
    k=np.random.randint(100, size=3)
    #pos luz
    x=3*i[0]/100-1.2
    y=2*j[0]/100-2.2
    z=2*k[0]/100+1
    client.simxSetObjectPosition(Light,-1,[x,y,z],client.simxDefault-
Publisher())

    #pos cam
    x=0.1*i[1]/100+0.3#-0.5
    y=0.1*j[1]/100-0.88#-0.5
    z=0.1*k[1]/100+1.9#+0.4
    client.simxSetObjectPosition(Cam_2,-1,[x,y,z],client.simxDefault-
Publisher())

    #pos summit
    x=(0.55-0.2)*i[2]/100+0.2
    y=-(1.1-0.71)*j[2]/100-0.71
    z=360*k[2]/100-180
    client.simxSetObjectPosition(summit,-1,[x,y,0.305],client.simxDefault-
Publisher())
    client.simxSetObjectOrientation(summit,-1,[0,0,z],client.simxDefault-
Publisher())
    return x,y

def borrar_objeto(client,Sensor):
    handle_a_destruir=client.simxReadProximitySensor(Sensor,client.simxService-
Call())[4]
    client.simxRemoveObjects([handle_a_destruir],1,client.simxServiceCall())
    time.sleep(1)
    handle_a_destruir=client.simxReadProximitySensor(Sensor,client.simxService-
Call())[4]
    client.simxRemoveObjects([handle_a_destruir],1,client.simxServiceCall())

def borrar_objeto_summit(client,Sensor_summit):
    handle_a_destruir=client.simxReadProximitySensor(Sensor_summit,client.simx-
ServiceCall())[4]
    client.simxRemoveObjects([handle_a_destruir],1,client.simxServiceCall())
    time.sleep(1)
    handle_a_destruir=client.simxReadProximitySensor(Sensor_summit,client.simx-
ServiceCall())[4]
    client.simxRemoveObjects([handle_a_destruir],1,client.simxServiceCall())

def pinza(client,close):
    tiempo_pinza=1 #segundos
    ret,j1=client.simxGetObjectHandle('ROBOTIQ_85_active1',client.simxService-
Call())
    ret,j2=client.simxGetObjectHandle('ROBOTIQ_85_active2',client.simxService-
Call())
    ret,p1=client.simxGetJointPosition(j1,client.simxServiceCall())
    ret,p2=client.simxGetJointPosition(j2,client.simxServiceCall())
    if close:
        if (p1<p2-0.008):
            client.simxSetJointTargetVelocity(j1,-0.08,client.simxDefault-
Publisher()) #0.01
```

```
            client.simxSetJointTargetVelocity(j2,-0.32,client.simxDefault-
Publisher()) #0.04 y 0.02
        else:
            client.simxSetJointTargetVelocity(j1,-0.32,client.simxDefault-
Publisher())
            client.simxSetJointTargetVelocity(j2,-0.08,client.simxDefault-
Publisher())
    else:
        if (p1<p2):
            client.simxSetJointTargetVelocity(j1,0.32,client.simxDefault-
Publisher())
            client.simxSetJointTargetVelocity(j2,0.16,client.simxDefault-
Publisher())
        else:
            client.simxSetJointTargetVelocity(j1,0.16,client.simxDefault-
Publisher())
            client.simxSetJointTargetVelocity(j2,0.32,client.simxDefault-
Publisher())
    time.sleep(tiempo_pinza)

def captura(client,summit):
    if summit==True:
        sensor='Vision_general'
    else:
        sensor='Vision_sensor'
    resul=client.simxCallScriptFunction('myFunctionName@'+sen-
sor,"sim.scripttype_childscript",[], client.simxServiceCall())
    im_pre=resul[1][0]
    return tuple2image(im_pre)

def tuple2image(im_pre):
    reso=128,128
    frame2=np.zeros((reso[0], reso[1], 3))
    frame2=frame2.astype('uint8')
    #La imagen vendra en valores de 0 a 1, hay que pasarlo a 0 a 255 redon-
deando a entero
    j=0
    k=0
    l=0
    for i in range(len(im_pre)):
        if j%3==0 and j!=0:
            j=0
            k=k+1
        if k%reso[0]==0 and k!=0:
            j=0
            k=0
            l=l+1
        frame2[l][reso[0]-k-1][j]=round(im_pre[len(im_pre)-i-1]*255,0)
        j=j+1
    return frame2

def recorte(image,j,k):
    x=6+j*29 #110/4 = 27.5
    y=10+k*12 #110/9 = 12.22
    image_cutted=image[x:x+30, y:y+15]
    return image_cutted

def imagen_a_red(imagen):
    frame2=cv2.cvtColor(imagen,cv2.COLOR_BGR2RGB)
    frame3=frame2.astype('float32')/255
    return np.expand_dims(frame3,axis=0)

def ordenar_estanteria(tipo_orden,mat):
    if tipo_orden==0: #Orden de piezas(0, 1, 2, 3, ... , 8, 9)
        lista=np.zeros((1,36), dtype=int)
        mat_n=np.zeros((4,9), dtype=int)
        k=0
        for i in range(4):
```

```
            for j in range(9):
                lista[0][k]=mat[i][j]
                k=k+1
        #ordenar lista
        for i in range(36):
            for j in range(36-i-1):
                if lista[0][j]>lista[0][j+1]:
                    aux=lista[0][j]
                    lista[0][j]=lista[0][j+1]
                    lista[0][j+1]=aux
        #pasar la lista
        k=0
        for i in range(4):
            for j in range(9):
                mat_n[i][j]=lista[0][k]
                k=k+1

    if tipo_orden==1: #Orden de piezas por color (0, 1, 4, 7, 2, 5, 8, 3, 6, 9)
        lista=np.zeros((1,36), dtype=int)
        mat_n=np.zeros((4,9), dtype=int)
        k=0
        for i in range(4):
            for j in range(9):
                if mat[i][j]==4:
                    lista[0][k]=2
                else:
                    if mat[i][j]==7:
                        lista[0][k]=3
                    else:
                        if mat[i][j]==2:
                            lista[0][k]=4
                        else:
                            if mat[i][j]==8:
                                lista[0][k]=6
                            else:
                                if mat[i][j]==3:
                                    lista[0][k]=7
                                else:
                                    if mat[i][j]==6:
                                        lista[0][k]=8
                                    else:
                                        lista[0][k]=mat[i][j]
                k=k+1
        print(lista)
        for i in range(36):
            for j in range(36-i-1):
                if lista[0][j]>lista[0][j+1]:
                    aux=lista[0][j]
                    lista[0][j]=lista[0][j+1]
                    lista[0][j+1]=aux
        print(lista)
        k=0
        for i in range(4):
            for j in range(9):
                if lista[0][k]==2:
                    mat_n[i][j]=4
                else:
                    if lista[0][k]==3:
                        mat_n[i][j]=7
                    else:
                        if lista[0][k]==4:
                            mat_n[i][j]=2
                        else:
                            if lista[0][k]==6:
                                mat_n[i][j]=8
                            else:
                                if lista[0][k]==7:
```

```
                                    mat_n[i][j]=3
                                else:
                                    if lista[0][k]==8:
                                        mat_n[i][j]=6
                                    else:
                                        mat_n[i][j]=lista[0][k]
                k=k+1
    return mat_n

def localizar_pieza(mat,pieza):
    ret=0
    n_piezas=0
    pos_i,pos_j=9,9
    for i in range(4):
        for j in range(9):
            if pieza==mat[3-i][8-j]:
                pos_i,pos_j=3-i,8-j
                ret=1
                n_piezas=n_piezas+1
    return ret, n_piezas, (pos_i,pos_j)
```

6.1.2 Programa Principal

```
import b0RemoteApi
import math
import time
import numpy as np
import cv2
import tensorflow
from tensorflow.python.keras.preprocessing.image import load_img, img_to_array
from tensorflow.python.keras.models import load_model
import tkinter
from PIL import ImageTk, Image
import funciones

#INICIO:
client = b0RemoteApi.RemoteApiClient('b0RemoteApi_CoppeliaSim','b0RemoteApi')

#CARGAR REDES NEURONALES ENTRENADAS:
modelo='./modelos red 1/modelo_prueba_1.h5'
cnn=load_model(modelo)
model1=load_model('./modelos red 2/modelo_summit3.h5')

#HANDLERS IMPORTANTES Y UTILES
_,Brazo1=client.simxGetObjectHandle('IRB140_joint1',client.simxServiceCall())
_,Brazo2=client.simxGetObjectHandle('IRB140_joint2',client.simxServiceCall())
_,Brazo3=client.simxGetObjectHandle('IRB140_joint3',client.simxServiceCall())
_,Brazo4=client.simxGetObjectHandle('IRB140_joint4',client.simxServiceCall())
_,Brazo5=client.simxGetObjectHandle('IRB140_joint5',client.simxServiceCall())
_,Brazo6=client.simxGetObjectHandle('IRB140_joint6',client.simxServiceCall())
_,MotorUD=client.simxGetObjectHandle('MotorUD',client.simxServiceCall())
_,MotorLR=client.simxGetObjectHandle('MotorLR',client.simxServiceCall())
_,Sensor=client.simxGetObjectHandle('Proximity_sensor2',client.simxService-
Call())
_,Cam=client.simxGetObjectHandle('Vision_sensor',client.simxServiceCall())
_,Cam_2=client.simxGetObjectHandle('Vision_general',client.simxServiceCall())
_,Light=client.simxGetObjectHandle('Luz',client.simxServiceCall())
_,Sensor1=client.simxGetObjectHandle('Proximity_sensor',client.simxService-
Call())
_,summit=client.simxGetObjectHandle('Robotnik_Summit_XL',client.simxService-
Call())
_,Sensor_summit=client.simxGetObjectHandle('Proximity_sensor3',client.simx-
ServiceCall())

#FUNCIONES
class posicion:
```

```
    def p0():
        funciones.moverB2y3(client,45*(2*3.14)/360)
        funciones.mover(client,Brazo5,0)
        #funciones.mover(client,Brazo1,180*(2*3.14)/360)
    def guardar():
        funciones.mover(client,Brazo1,180*(2*3.14)/360)
        funciones.moverB2y3(client,0*(2*3.14)/360)
        funciones.moverB2y3(client,-25*(2*3.14)/360)
        funciones.mover(client,Brazo5,0*(2*3.14)/360)
    def esconder():
        funciones.moverB2y3(client,45*(2*3.14)/360)
        funciones.mover(client,MotorUD,-0.9)
        #funciones.mover(client,Brazo1,90*(2*3.14)/360)
    def camara():
        funciones.moverB2y3(client,45*(2*3.14)/360)
        funciones.mover(client,Brazo5,0)
        funciones.mover(client,Brazo1,70*(2*3.14)/360)
        funciones.mover(client,MotorLR,-0.3)
        funciones.mover(client,MotorUD,-0.37)
    def recoger():
        funciones.mover(client,Brazo1,0*(2*3.14)/360)
        funciones.mover(client,MotorLR,0)
        funciones.mover(client,Brazo1,90*(2*3.14)/360)
        funciones.mover(client,MotorUD,-0.83)
        funciones.moverB2y3(client,45*(2*3.14)/360)
        funciones.mover(client,Brazo5,0)
        funciones.mover(client,MotorLR,0.2)

        funciones.pinza(client,False)
        posicionado=0
        t_max=10
        t=0
        while(posicionado==0) and not(t>t_max):
            time.sleep(1)
            posicionado=client.simxReadProximitySensor(Sensor1,client.simxSer-
viceCall())[1]
        if not(t>t_max):
            #print('Time out: Recoger')
        #else:
            funciones.mover(client,MotorLR,0.530)
            funciones.pinza(client,True)
            funciones.mover(client,MotorUD,-0.5)
            funciones.mover(client,MotorLR,0)

def estanteria(x,y,guardar):
    ymax=7
    qUD=-x*0.3-0.015+0.03
    qLR=-0.4+y*0.8/(ymax)
    posicion.p0()
    funciones.mover(client,MotorUD,qUD)
    funciones.mover(client,MotorLR,qLR)
    if guardar==False:
        qUD=-x*0.3-0.015
        funciones.mover(client,MotorUD,qUD)
        funciones.pinza(client,False)
    posicion.guardar()
    if guardar:
        qUD=-x*0.3-0.015
        funciones.mover(client,MotorUD,qUD)
        funciones.pinza(client,False)
    else:
        funciones.pinza(client,True)
    posicion.p0()

class pieza:
    def colocar(objeto_n,x,y,z):
```

```
        client.simxSetObjectPosition(objeto_n,-1,[x,y,z],client.simxDefaultPu-
blisher())
        client.simxSetObjectQuaternion(objeto_n,-1,[0,0,0,0],client.simxDe-
faultPublisher())
    def aleatoria():
        i=np.random.randint(9, size=1)[0]
        i+=1
        shape_i='Shape'+str(i)
        objeto_i='Objeto'+str(i)
        ret0,objeto=client.simxGetObjectHandle(objeto_i,client.simxService-
Call())
        ret1,shape=client.simxGetObjectHandle(shape_i,client.simxServiceCall())
        item=client.simxCopyPasteObjects([objeto,shape],0,client.simxService-
Call())[1][0]
        pieza.colocar(item,-0.4045,0.8,0.827)
        return i
    def aleatoria_en_sitio(xi,yj):
        x=-1.1705
        y=-1.6742-(1.5597-1.6742)*xi
        z=+1.7001-(1.7001-1.4001)*yj
        i=np.random.randint(10, size=1)[0]
        if not(i==0):
            shape_i='Shape'+str(i)
            objeto_i='Objeto'+str(i)
            ret0,objeto=client.simxGetObjectHandle(objeto_i,client.simxService-
Call())
            ret1,shape=client.simxGetObjectHandle(shape_i,client.simxService-
Call())
            item=client.simxCopyPasteObjects([objeto,shape],0,client.simx-
ServiceCall())[1][0]
            pieza.colocar(item,x,y,z)
        return i

def dejar_pieza(guardar):
    qUD=-3*0.27-0.015
    qLR=-0.4+0*0.8/(7)
    funciones.mover(client,MotorUD,qUD)
    funciones.mover(client,MotorLR,qLR)
    if guardar==False:
        qUD=-3*0.3-0.015
        funciones.mover(client,MotorUD,qUD)
        funciones.pinza(client,False)
    funciones.mover(client,Brazo1,0*(2*3.14)/360)
    funciones.moverB2y3(client,0*(2*3.14)/360)
    funciones.moverB2y3(client,-25*(2*3.14)/360)
    funciones.mover(client,Brazo5,0*(2*3.14)/360)
    if guardar:
        qUD=-3*0.3-0.015
        funciones.mover(client,MotorUD,qUD)
        time.sleep(1)
        funciones.pinza(client,False)
    else:
        funciones.pinza(client,True)
    funciones.moverB2y3(client,0*(2*3.14)/360)
    posicion.p0()

def estanteria_aleatoria():
    for i in range(4):
        for j in range(9):
            pieza.aleatoria_en_sitio(j,i)

def objeto_camara():
    frame=funciones.captura(client,False)
    frame2=cv2.resize(frame,(15,30))
    x=funciones.imagen_a_red(frame2)
    resultado=np.argmax(cnn.predict(x)[0])
    return resultado
```

```
def revisar_estanteria():
    mat_est_e=np.zeros((4,9), dtype=int)
    posicion.esconder()
    frame=funciones.captura(client,False)
    for i in range(4):
        for j in range(9):
            frame2=funciones.recorte(frame,i,j)
            x=funciones.imagen_a_red(frame2)
            mat_est_e[i][j]=np.argmax(cnn.predict(x)[0])
    return mat_est_e

def intercambio(i1,j1,i2,j2):
    estanteria(i1,j1,False)
    dejar_pieza(True)
    estanteria(i2,j2,False)
    estanteria(i1,j1,True)
    dejar_pieza(False)
    estanteria(i2,j2,True)

def cambios(mat,mat_n):
    for i in range(4):
        for j in range(9):
            if mat_n[3-i][8-j]!=mat[3-i][8-j]:
                _,_,(i_n,j_n)=funciones.localizar_pieza(mat,mat_n[3-i][8-j])
                if mat[3-i][8-j]==0:
                    estanteria(i_n,j_n,False)
                    estanteria(3-i,8-j,True)
                else:
                    intercambio(3-i,8-j,i_n,j_n)
                aux=mat[3-i][8-j]
                mat[3-i][8-j]=mat[i_n][j_n]
                mat[i_n][j_n]=aux

def buscar_borrar(estante,tipo):
    ret, n_piezas, (pos_i,pos_j)=funciones.localizar_pieza(estante,tipo)
    #print(ret, n_piezas, (pos_i,pos_j))
    if ret==1:
        estanteria(pos_i,pos_j,False)
        dejar_pieza(True)
        time.sleep(1)
        funciones.borrar_objeto(client,Sensor)
    return ret, n_piezas

def guia_brazo(x,y):
    posicion.p0()
    qLR=1.0003*y+1.2753225
    funciones.mover(client,Brazo1,0*(2*3.14)/360)
    funciones.mover(client,MotorLR,qLR)
    funciones.mover(client,MotorUD,-0.5)
    x_extra=-0.1714
    z_extra=0
    x_obj=x-x_extra
    z_obj=0 #Para tener el maximo rango en el eje x
    L1=0.18737-x_extra
    L2=0.20664-x_extra
    #CINEMATICA INVERSA para no partir robot a>0
    q2=-math.acos((x_obj**2+z_obj**2-L1**2-L2**2)/(2*L1*L2))
    q1=math.atan(z_obj/x_obj)-math.atan((L2*math.sin(q2))/(L1+L2*math.cos(q2)))
    a=360/(2*3.14)*q1
    b=360/(2*3.14)*q2
    funciones.moverB235(client,a,b)
    funciones.mover(client,MotorUD,-0.7)
    time.sleep(1)
    funciones.pinza(client,False)
    funciones.mover(client,MotorUD,-0.5)
    posicion.p0()
```

```
#EJECUCION:

#-------------------------------------------------------------------------------
-------------
#-----------------------------------PROGRAMA------------------------------------
-------------
#-------------------------------------------------------------------------------
-------------

ventana=tkinter.Tk()
ventana.geometry("620x500")
ventana.title("CoppeliaSim NEURAL")
estante=np.zeros((4,9), dtype=int)
operario=0

def funcion_operario():
    menu(10)
    menu(1)
    global operario
    operario=1

def funcion_cliente():
    menu(10)
    menu(2)
    global operario
    operario=0

def funcion_op_cl():
    menu(10)
    global operario
    if operario==1:
        menu(1)
    else:
        menu(2)

def funcion_cambio():
    menu(10)
    global operario
    if operario==0:
        operario=1
        menu(1)
    else:
        operario=0
        menu(2)

def funcion_est_al():
    if(client.simxGetSimulationState(client.simxServiceCall())[1]==16):
        texto1.configure(text="Se ha generado una estanteria aleatoria")
        texto2.configure(text="")
        texto3.configure(text="")
        texto4.configure(text="")
        texto5.configure(text="")
        estanteria_aleatoria()
    else:
        texto1.configure(text="SIMULACIÓN NO ESTA CORRIENDO, petición dene-
gada")
        texto2.configure(text="")
        texto3.configure(text="")
        texto4.configure(text="")
        texto5.configure(text="")

def funcion_obj_al():
    if(client.simxGetSimulationState(client.simxServiceCall())[1]==16):
        texto1.configure(text="Se ha generado una pieza aleatoria")
        texto2.configure(text="")
        texto3.configure(text="")
        texto4.configure(text="")
        texto5.configure(text="")
```

```
        pieza.aleatoria()
    else:
        texto1.configure(text="SIMULACIÓN NO ESTA CORRIENDO, petición dene-
gada")
        texto2.configure(text="")
        texto3.configure(text="")
        texto4.configure(text="")
        texto5.configure(text="")

def funcion_est_ord():
    #seguir

    if(client.simxGetSimulationState(client.simxServiceCall())[1]==16):
        texto1.configure(text="Tipo de orden:")
        menu(10)
        menu(8)
    else:
        texto1.configure(text="SIMULACIÓN NO ESTA CORRIENDO, petición dene-
gada")
        texto2.configure(text="")
        texto3.configure(text="")
        texto4.configure(text="")
        texto5.configure(text="")

def funcion_orden():
    global estante
    estante=revisar_estanteria()
    texto1.configure(text="Se ha ordenado la estanteria, tipo de or-
den="+str(spin5.get()))
    texto2.configure(text="")
    texto3.configure(text="")
    texto4.configure(text="")
    texto5.configure(text="")
    estante_ordenado=funciones.ordenar_estanteria(int(spin5.get()),estante)
    cambios(estante,estante_ordenado)
    estante=revisar_estanteria()
    funcion_op_cl()


def funcion_rec_gua():
    global estante
    if(client.simxGetSimulationState(client.simxServiceCall())[1]==16):
        estante=revisar_estanteria()
        tipo=int(spin2.get())
        ret,n_piezas,(i_n,j_n)=funciones.localizar_pieza(estante,0)
        if ret==1:
            if(client.simxReadProximitySensor(Sensor1,client.simxService-
Call())[1]!=0):
                posicion.recoger()
                posicion.camara()
                time.sleep(1)
                tipo=objeto_camara()
                estanteria(i_n,j_n,True)
                text="Pieza guardada en estanteria, tipo = "+str(tipo)
            else:
                text="No hay objeto al final de la cinta"
        else:
            text="No hay hueco en estanteria"
        estante=revisar_estanteria()
        texto1.configure(text=text)
        texto2.configure(text=str(estante[0]))
        texto3.configure(text=str(estante[1]))
        texto4.configure(text=str(estante[2]))
        texto5.configure(text=str(estante[3]))
    else:
        texto1.configure(text="SIMULACION NO ESTA CORRIENDO, peticion dene-
gada")
```

```
        texto2.configure(text="")
        texto3.configure(text="")
        texto4.configure(text="")
        texto5.configure(text="")

def funcion_enc_des():
    if(client.simxGetSimulationState(client.simxServiceCall())[1]==16):
        texto1.configure(text="Encontrar y eliminar de escena")
        texto2.configure(text="Tipo de objeto SPIN IZQUIERDA")
        texto3.configure(text="Color de objeto SPIN DERECHA")
        texto4.configure(text="")
        texto5.configure(text="")
        menu(10)
        menu(3)
    else:
        texto1.configure(text="SIMULACION NO ESTA CORRIENDO, peticion dene-
gada")
        texto2.configure(text="")
        texto3.configure(text="")
        texto4.configure(text="")
        texto5.configure(text="")

def funcion_destroy():
    global estante
    estante=revisar_estanteria()
    if spin3.get()=='Tipo' or spin4.get()=='Color':
        tipo=0
    if spin3.get()=='Lata':
        if spin4.get()=='Azul':
            tipo=1
        if spin4.get()=='Amarillo':
            tipo=2
        if spin4.get()=='Rosa':
            tipo=3
    if spin3.get()=='Botella plastico':
        if spin4.get()=='Azul':
            tipo=4
        if spin4.get()=='Amarillo':
            tipo=5
        if spin4.get()=='Rosa':
            tipo=6
    if spin3.get()=='Botella cristal':
        if spin4.get()=='Azul':
            tipo=7
        if spin4.get()=='Amarillo':
            tipo=8
        if spin4.get()=='Rosa':
            tipo=9
    if tipo==0:
        print('ERROR')
    else:
        ret,n_piezas=buscar_borrar(estante,tipo)
        texto1.configure(text="Se ha sacado un objeto tipo "+str(tipo)+" de
"+str(n_piezas)+" pieza/s que habia en la estanteria")
        if(ret!=1):
            texto1.configure(text="No hay dicho objeto en la estanteria")
    texto2.configure(text="")
    texto3.configure(text="")
    texto4.configure(text="")
    texto5.configure(text="")
    funcion_op_cl()


def funcion_acc_cam():
    if(client.simxGetSimulationState(client.simxServiceCall())[1]==16):
        texto1.configure(text="Acceso a camaras")
        menu(10)
        menu(4)
```

```
    else:
        texto1.configure(text="SIMULACION PARADA, no es posible el acceso")
    texto2.configure(text="")
    texto3.configure(text="")
    texto4.configure(text="")
    texto5.configure(text="")

def funcion_cam_est():
    texto1.configure(text="Captura camara estanteria 128x128")
    texto2.configure(text="")
    texto3.configure(text="")
    texto4.configure(text="")
    texto5.configure(text="")
    image=funciones.captura(client,False)
    resize_img = cv2.resize(image,(256,256))
    image1 = cv2.cvtColor(resize_img, cv2.COLOR_BGR2RGB)
    image2 = Image.fromarray(image1)
    imgtk = ImageTk.PhotoImage(image=image2)
    imagen.imgtk = imgtk
    imagen.configure(image=imgtk)
    menu(10)
    menu(5)

def funcion_cam_sum():
    texto1.configure(text="Captura camara parada summit 128x128")
    texto2.configure(text="")
    texto3.configure(text="")
    texto4.configure(text="")
    texto5.configure(text="")
    image=funciones.captura(client,True)
    resize_img = cv2.resize(image,(256,256))
    image1 = cv2.cvtColor(resize_img, cv2.COLOR_BGR2RGB)
    image2 = Image.fromarray(image1)
    imgtk = ImageTk.PhotoImage(image=image2)
    imagen.imgtk = imgtk
    imagen.configure(image=imgtk)
    menu(10)
    menu(5)

def funcion_start():
    texto1.configure(text="Simulacion corriendo")
    texto2.configure(text="")
    texto3.configure(text="")
    texto4.configure(text="")
    texto5.configure(text="")
    client.simxStartSimulation(client.simxServiceCall())

def funcion_stop():
    texto1.configure(text="Simulacion parada")
    texto2.configure(text="")
    texto3.configure(text="")
    texto4.configure(text="")
    texto5.configure(text="")
    client.simxStopSimulation(client.simxServiceCall())

def funcion_men_sum():
    texto1.configure(text="Menu SUMMIT")
    texto2.configure(text="")
    texto3.configure(text="")
    texto4.configure(text="")
    texto5.configure(text="")
    menu(10)
    menu(6)

def funcion_des_sum():
    #borrar_objeto_summit()
    funciones.borrar_objeto(client,Sensor_summit)
```

```
def funcion_enc_sum():
    texto1.configure(text="Encontrar y llevar a SUMMIT")
    texto2.configure(text="Tipo de objeto SPIN IZQUIERDA")
    texto3.configure(text="Color de objeto SPIN DERECHA")
    texto4.configure(text="")
    texto5.configure(text="")
    menu(10)
    menu(7)

def funcion_est_sum():
    global estante
    estante=revisar_estanteria()
    if spin3.get()=='Tipo' or spin4.get()=='Color':
        tipo=0
    if spin3.get()=='Lata':
        if spin4.get()=='Azul':
            tipo=1
        if spin4.get()=='Amarillo':
            tipo=2
        if spin4.get()=='Rosa':
            tipo=3
    if spin3.get()=='Botella plastico':
        if spin4.get()=='Azul':
            tipo=4
        if spin4.get()=='Amarillo':
            tipo=5
        if spin4.get()=='Rosa':
            tipo=6
    if spin3.get()=='Botella cristal':
        if spin4.get()=='Azul':
            tipo=7
        if spin4.get()=='Amarillo':
            tipo=8
        if spin4.get()=='Rosa':
            tipo=9
    if tipo==0:
        print('ERROR')
    else:
        ret, n_piezas, (pos_i,pos_j)=funciones.localizar_pieza(estante,tipo)
        #print(ret, n_piezas, (pos_i,pos_j))
        if ret==1:
            time.sleep(1)
            imagen1=funciones.captura(client,True)
            pos_pred=model1.predict(funciones.imagen_a_red(imagen1))[0]
            estanteria(pos_i,pos_j,False)
            sum_xp,sum_yp=pos_pred[0],pos_pred[1]
            time.sleep(1)
            guia_brazo(sum_xp,sum_yp)
            texto1.configure(text="Se ha sacado un objeto tipo "+str(tipo)+" de
"+str(n_piezas)+" pieza/s que habia en la estanteria")
        else:
            texto1.configure(text="No hay dicho objeto en la estanteria")
    texto2.configure(text="")
    texto3.configure(text="")
    texto4.configure(text="")
    texto5.configure(text="")
    funcion_op_cl()

def funcion_cin_sum():
    if(client.simxReadProximitySensor(Sensor1,client.simxServiceCall())[1]!=0):
        imagen1=funciones.captura(client,True)
        pos_pred=model1.predict(funciones.imagen_a_red(imagen1))[0]
        posicion.recoger()
        sum_xp,sum_yp=pos_pred[0],pos_pred[1]
        guia_brazo(sum_xp,sum_yp)
        funcion_men_sum()
    else:
```

```
        text="No hay objeto al final de la cinta"
        texto1.configure(text=text)

def funcion_con_sum():
    #POR AHORA NO HAY CONTROL
    funcion_men_sum()

def funcion_random():
    if(client.simxGetSimulationState(client.simxServiceCall())[1]==0):
        funciones.pos_aleatorios(client,Light,Cam_2,summit)
    else:
        texto1.configure(text="SIMULACIÓN ESTA CORRIENDO, petición denegada")
        texto2.configure(text="")
        texto3.configure(text="")
        texto4.configure(text="")
        texto5.configure(text="")

def funcion_est_est():
    global estante
    if(client.simxGetSimulationState(client.simxServiceCall())[1]==16):
        estante=revisar_estanteria()
        texto1.configure(text="Estado de la estanteria")
        texto2.configure(text=str(estante[0]))
        texto3.configure(text=str(estante[1]))
        texto4.configure(text=str(estante[2]))
        texto5.configure(text=str(estante[3]))
    else:
        texto1.configure(text="SIMULACIÓN NO ESTA CORRIENDO, petición dene-
gada")
        texto2.configure(text="")
        texto3.configure(text="")
        texto4.configure(text="")
        texto5.configure(text="")

def funcion_ent_gua():
    global estante
    if(client.simxGetSimulationState(client.simxServiceCall())[1]==16):
        pieza.aleatoria()
        estante=revisar_estanteria()
        tipo=int(spin2.get())
        ret,n_piezas,(i_n,j_n)=funciones.localizar_pieza(estante,0)
        if ret==1:
            posicion.recoger()
            posicion.camara()
            time.sleep(1)
            tipo=objeto_camara()
            estanteria(i_n,j_n,True)
            text="Pieza guardada en estanteria, tipo = "+str(tipo)
        else:
            text="No hay hueco en estanteria"
        estante=revisar_estanteria()
        texto1.configure(text=text)
        texto2.configure(text=str(estante[0]))
        texto3.configure(text=str(estante[1]))
        texto4.configure(text=str(estante[2]))
        texto5.configure(text=str(estante[3]))
    else:
        texto1.configure(text="SIMULACION NO ESTA CORRIENDO, peticion dene-
gada")
        texto2.configure(text="")
        texto3.configure(text="")
        texto4.configure(text="")
        texto5.configure(text="")
```

```
def menu(i):
    if i==0:#MENU SELECCION OPERARIO/CLIENTE
        texto1.grid(column=0, row=0,columnspan=2,padx=10, pady=10)
        boton_operario.grid(column=0, row=1,padx=10, pady=10)
        boton_cliente.grid(column=1, row=1,padx=10, pady=10)

    if i==1: #MENU OPERARIO
        texto1.grid(column=0, row=0,columnspan=2,padx=10, pady=10)
        texto2.grid(column=0, row=1,columnspan=2,padx=10, pady=10)
        texto3.grid(column=0, row=2,columnspan=2,padx=10, pady=10)
        texto4.grid(column=0, row=3,columnspan=2,padx=10, pady=10)
        texto5.grid(column=0, row=4,columnspan=2,padx=10, pady=10)
        boton_est_al.grid(column=0, row=5,padx=10, pady=10)
        boton_obj_al.grid(column=1, row=5,padx=10, pady=10)
        boton_est_ord.grid(column=0, row=6,padx=10, pady=10)
        boton_rec_gua.grid(column=1, row=6,padx=10, pady=10)
        #boton_enc_des.grid(column=0, row=7,padx=10, pady=10)
        boton_summit.grid(column=0, row=7,padx=10, pady=10)
        boton_acc_cam.grid(column=1, row=7,padx=10, pady=10)
        boton_cambio.grid(column=0, row=8,padx=10, pady=10)
        boton_pos_ale.grid(column=1, row=8,padx=10, pady=10)
        boton_start.grid(column=0, row=9,padx=10, pady=10)
        boton_stop.grid(column=1, row=9,padx=10, pady=10)

    if i==2: #MENU CLIENTE
        texto1.grid(column=0, row=0,columnspan=2,padx=10, pady=10)
        texto2.grid(column=0, row=1,columnspan=2,padx=10, pady=10)
        texto3.grid(column=0, row=2,columnspan=2,padx=10, pady=10)
        texto4.grid(column=0, row=3,columnspan=2,padx=10, pady=10)
        texto5.grid(column=0, row=4,columnspan=2,padx=10, pady=10)
        boton_est_est.grid(column=0, row=5,padx=10, pady=10)
        boton_ent_gua.grid(column=1, row=5,padx=10, pady=10)
        boton_enc_des.grid(column=0, row=6,padx=10, pady=10)
        boton_cambio.grid(column=1, row=6,padx=10, pady=10)

    if i==3: #MENU SEEK AND DESTROY
        texto1.grid(column=0, row=0,columnspan=2,padx=10, pady=10)
        texto2.grid(column=0, row=1,columnspan=2,padx=10, pady=10)
        texto3.grid(column=0, row=2,columnspan=2,padx=10, pady=10)
        spin3.grid(column=0, row=3,padx=10, pady=10)
        spin4.grid(column=1, row=3,padx=10, pady=10)
        boton_destruir.grid(column=0, row=4,columnspan=1,padx=10, pady=10)
        boton_salir.grid(column=1, row=4, padx=10, pady=10)

    if i==4: #MENU CAMARAS
        texto1.grid(column=0, row=0,columnspan=2,padx=10, pady=10)
        boton_cam_est.grid(column=0, row=3,padx=10, pady=10)
        boton_cam_sum.grid(column=1, row=3,padx=10, pady=10)
        boton_salir.grid(column=0, row=4,columnspan=2,padx=100, pady=10)

    if i==5: #MENU CAPTURA
        texto1.grid(column=0, row=0,columnspan=2,padx=100, pady=10)
        imagen.grid(column=0, row=1,columnspan=2,padx=100, pady=10)
        boton_salir.grid(column=0, row=2,columnspan=2,padx=100, pady=10)

    if i==6: #MENU SUMMIT
        texto1.grid(column=0, row=0,columnspan=2,padx=10, pady=10)
        #boton_con_sum.grid(column=0, row=1,padx=10, pady=10)
        boton_des_sum.grid(column=1, row=1,padx=10, pady=10)
        boton_enc_sum.grid(column=0, row=1,padx=10, pady=10)
        boton_salir.grid(column=1, row=2,padx=10, pady=10)

    if i==7: #MENU LOCALIZAR OBJETO Y LLEVAR AL SUMMIT
        texto1.grid(column=0, row=0,columnspan=2,padx=10, pady=10)
        texto2.grid(column=0, row=1,columnspan=2,padx=10, pady=10)
        texto3.grid(column=0, row=2,columnspan=2,padx=10, pady=10)
        spin3.grid(column=0, row=3,padx=10, pady=10)
        spin4.grid(column=1, row=3,padx=10, pady=10)
```

```
        boton_est_sum.grid(column=0, row=4,padx=10, pady=10)
        boton_cin_sum.grid(column=1, row=4,padx=10, pady=10)
        boton_salir.grid(column=0, row=5,columnspan=2,padx=100, pady=10)

    if i==8:#MENU ORDEN
        texto1.grid(column=0, row=0,columnspan=1,padx=100, pady=10)
        spin5.grid(column=1, row=0,padx=10, pady=10)
        boton_orden.grid(column=0, row=1,padx=10, pady=10)
        boton_salir.grid(column=1, row=1,padx=10, pady=10)

    if i==10: #MENU FORGET ALL
        #BOTONES
        boton_operario.grid_forget()
        boton_cliente.grid_forget()
        boton_est_al.grid_forget()
        boton_obj_al.grid_forget()
        boton_est_ord.grid_forget()
        boton_rec_gua.grid_forget()
        boton_enc_des.grid_forget()
        boton_destruir.grid_forget()
        boton_cam_est.grid_forget()
        boton_cam_sum.grid_forget()
        boton_salir.grid_forget()
        boton_summit.grid_forget()
        boton_acc_cam.grid_forget()
        boton_start.grid_forget()
        boton_stop.grid_forget()
        boton_con_sum.grid_forget()
        boton_des_sum.grid_forget()
        boton_enc_sum.grid_forget()
        boton_est_sum.grid_forget()
        boton_cin_sum.grid_forget()
        boton_pos_ale.grid_forget()
        boton_cambio.grid_forget()
        boton_est_est.grid_forget()
        boton_ent_gua.grid_forget()
        boton_orden.grid_forget()

        #TEXTO
        texto1.grid_forget()
        texto2.grid_forget()
        texto3.grid_forget()
        texto4.grid_forget()
        texto5.grid_forget()
        texto6.grid_forget()
        texto7.grid_forget()

        #SPINS
        spin1.grid_forget()
        spin2.grid_forget()
        spin3.grid_forget()
        spin4.grid_forget()
        spin5.grid_forget()

        #IMAGENES
        imagen.grid_forget()


boton_operario=tkinter.Button(ventana,text="OPERARIO",bg="orange",
fg="white",command= funcion_operario, width=40)
boton_cliente=tkinter.Button(ventana,text="CLIENTE",bg="orange",
fg="white",command= funcion_cliente, width=40)
boton_est_al=tkinter.Button(ventana,text="Generar estanteria aleatoria",bg="or-
ange", fg="white",command=funcion_est_al, width=40)
boton_obj_al=tkinter.Button(ventana,text="Generar objeto aleatoria",bg="or-
ange", fg="white",command= funcion_obj_al, width=40)
```

```
boton_est_ord=tkinter.Button(ventana,text="Ordenar estanteria",bg="orange",
fg="white",command=funcion_est_ord, width=40)
boton_rec_gua=tkinter.Button(ventana,text="Recoger y guardar",bg="orange",
fg="white",command=funcion_rec_gua, width=40)
boton_enc_des=tkinter.Button(ventana,text="Encontrar y entregar",bg="orange",
fg="white",command=funcion_enc_des, width=40)
boton_destruir=tkinter.Button(ventana,text="Sacar y Eliminar",bg="orange",
fg="white",command=funcion_destroy, width=40)
boton_acc_cam=tkinter.Button(ventana,text="Acceso a la imagen de cama-
ras",bg="orange", fg="white",command=funcion_acc_cam, width=40)
boton_cam_est=tkinter.Button(ventana,text="Camara estanteria",bg="orange",
fg="white",command=funcion_cam_est, width=40)
boton_cam_sum=tkinter.Button(ventana,text="Camara summit",bg="orange",
fg="white",command=funcion_cam_sum, width=40)
boton_summit=tkinter.Button(ventana,text="Menu summit",bg="orange",
fg="white",command=funcion_men_sum, width=40)
boton_salir=tkinter.Button(ventana,text="SALIR",bg="orange", fg="white",com-
mand=funcion_op_cl, width=40)
boton_start=tkinter.Button(ventana,text="START SIMULATION",bg="green",
fg="white",command=funcion_start, width=40)
boton_stop=tkinter.Button(ventana,text="STOP SIMULATION",bg="red",
fg="white",command=funcion_stop, width=40)
boton_con_sum=tkinter.Button(ventana,text="Control manual Summit",bg="orange",
fg="white",command=funcion_con_sum, width=40)
boton_des_sum=tkinter.Button(ventana,text="Destruir objeto Summit",bg="orange",
fg="white",command=funcion_des_sum, width=40)
boton_enc_sum=tkinter.Button(ventana,text="Dejar un objeto en summit",bg="or-
ange", fg="white",command=funcion_enc_sum, width=40)
boton_est_sum=tkinter.Button(ventana,text="Recoger de estanteria",bg="orange",
fg="white",command=funcion_est_sum, width=40)
boton_cin_sum=tkinter.Button(ventana,text="Recoger de cinta",bg="orange",
fg="white",command=funcion_cin_sum, width=40)
boton_pos_ale=tkinter.Button(ventana,text="Randomizar",bg="yellow",
fg="black",command=funcion_random, width=40)
boton_cambio=tkinter.Button(ventana,text="OPERARIO<->CLIENTE",bg="yellow",
fg="black",command=funcion_cambio, width=40)
boton_est_est=tkinter.Button(ventana,text="Estado de la estanteria",bg="or-
ange", fg="white",command=funcion_est_est, width=40)
boton_ent_gua=tkinter.Button(ventana,text="Entregar y guardar un ob-
jeto",bg="orange", fg="white",command= funcion_ent_gua, width=40)
boton_orden=tkinter.Button(ventana,text="ORDENAR",bg="orange", fg="white",com-
mand= funcion_orden, width=40)

texto1=tkinter.Label(ventana, text="Bienvenido al contolador de la celula robo-
tizada")
texto2=tkinter.Label(ventana, text="")
texto3=tkinter.Label(ventana, text="")
texto4=tkinter.Label(ventana, text="")
texto5=tkinter.Label(ventana, text="")
texto6=tkinter.Label(ventana, text="spin1(posicion del eje 'z' de la estante-
ria)")
texto7=tkinter.Label(ventana, text="spin2(posicion del eje 'x' o tipo de ob-
jeto)")

spin1=tkinter.Spinbox(ventana, from_=0, to=5, width=5)
spin2=tkinter.Spinbox(ventana, from_=0, to=9, width=5)
spin3=tkinter.Spinbox(ventana, values=('Tipo','Lata','Botella plastico','Bote-
lla cristal'), width=20)
spin4=tkinter.Spinbox(ventana, values=('Color','Amarillo','Azul','Rosa'),
width=20)
spin5=tkinter.Spinbox(ventana, from_=0, to=1, width=5)

imagen=tkinter.Label(ventana,width=256, height=256,padx=40, pady=40)

#GRID INICIO
texto1.grid(column=0, row=0, columnspan=2,padx=10, pady=10)
boton_operario.grid(column=0, row=1,padx=10, pady=10)
boton_cliente.grid(column=1, row=1,padx=10, pady=10)
```

```
ventana.mainloop()
```

6.1.3 Script Entrenamiento Red de clasificacion

```
import sys
import os
import tensorflow
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras import optimizers
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dropout, Flatten, Dense, Activation,
Conv2D, MaxPooling2D
from tensorflow.python.keras import backend as K
import matplotlib.pyplot as plt
from tensorflow.keras import regularizers

#PARAMETROS:
K.clear_session()
data_entrenamiento='./data/entrenamiento'
data_validacion='./data/validacion'
epochs=30
altura, longitud=30,15
batch_size=10
steps_per_epoch=800 #n_total/batch_size=5000/30=
validation_steps=90
filtrosConv1=32 #original 32
filtrosConv2=64 #original 64
tamano_filtro1=(3,3)
tamano_filtro2=(2,2)
tamano_pool=(2,2)
clases=10
lr=0.0005
n=1024

#PREPROCESADO DE IMAGENES:

entrenamiento_datagen=ImageDataGenerator(re-
scale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
validacion_datagen=ImageDataGenerator(rescale=1./255)

imagen_entrenamiento=entrenamiento_datagen.flow_from_directory(data_entrena-
miento, target_size= (altura,longitud),batch_size=batch_size,class_mode='cate-
gorical')
imagen_validacion=validacion_datagen.flow_from_directory(data_validacion, tar-
get_size= (longitud,altura),batch_size=batch_size,class_mode='categorical')

#RED CONVOLUCIONAL:

cnn=Sequential()

cnn.add(Conv2D(filtrosConv1,tamano_filtro1,padding='same',input_shape=(al-
tura,longitud,3),activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))
cnn.add(Conv2D(filtrosConv2,tamano_filtro2,padding='same',activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Flatten())
cnn.add(Dense(n,activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(clases,activation='softmax'))

cnn.compile(loss='categorical_crossentropy',optimizer=tensorflow.keras.optimiz-
ers.Adam(lr=lr),metrics=['accuracy'])
history = cnn.fit(imagen_entrena-
miento,steps_per_epoch=steps_per_epoch,epochs=epochs,valida-
tion_data=imagen_validacion,validation_steps=validation_steps)
print(cnn.summary())
```

6.1.4 Script Entrenamiento Red de localización

```
import tensorflow
import numpy as np
import cv2
import tensorflow.python.keras as keras
import tensorflow.python.keras.layers as layers
import tensorflow.python.keras.regularizers as regularizers
from collections import namedtuple
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras import optimizers
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dropout, Flatten, Dense, Activation,
Conv2D, MaxPooling2D
from tensorflow.python.keras import backend as K
from tensorflow.python.keras.preprocessing.image import load_img, img_to_array
from tensorflow.python.keras.models import load_model
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers.experimental import preprocessing

#PARAMETROS:
n_train=5000
n_val=2000

def red_secuencial():
    altura, longitud=128,128
    tamano_filtro1=(3,3)
    tamano_filtro2=(2,2)
    tamano_pool=(2,2)
    lr=0.00001
    filtrosConv1=32
    filtrosConv2=64

    cnn=Sequential()
    cnn.add(Conv2D(filtrosConv1,tamano_filtro1,padding='same',input_shape=(al-
tura,longitud,3),activation='relu'))
    cnn.add(MaxPooling2D(pool_size=tamano_pool))
    cnn.add(Conv2D(filtrosConv2,tamano_filtro2,padding='same',activa-
tion='relu'))
    cnn.add(MaxPooling2D(pool_size=tamano_pool))

    cnn.add(Flatten())
    cnn.add(Dense(128,activation='relu'))
    cnn.add(Dense(256,activation='relu'))
    cnn.add(Dense(512,activation='relu'))
    cnn.add(Dense(256,activation='relu'))
    cnn.add(Dense(128,activation='relu'))
    cnn.add(Dense(2,activation='linear'))

    cnn.compile(loss='mean_squared_error',optimizer=tensorflow.keras.optimiz-
ers.Adam(lr=lr),metrics=['mean_absolute_error'])

    return cnn

def lectura_txt(n_is):
    array_info=np.zeros((n_is,3),dtype=np.float64)
    f = open("./data/posiciones.txt")
    for i in range(n_is):
        text=''
        t=''
        while t!=',':
            t=f.read(1)
            if t==',':
                break
            text=text+t
        array_info[i][0]=int(text)
        text=''
        t=''
```

```
        while t!=',':
            t=f.read(1)
            if t==',':
                break
            text=text+t
        array_info[i][1]=float(text)
        text=''
        t=''
        while t!='\n':
            t=f.read(1)
            if t=='\n':
                break
            text=text+t
        array_info[i][2]=float(text)
    return array_info

def sp_noise(image,prob):
    output = np.zeros(image.shape,np.uint8)
    thres = 1 - prob
    for k in range(image.shape[2]):
        for i in range(image.shape[0]):
            for j in range(image.shape[1]):
                rdn = np.random.randint(255)/255
                if rdn < prob:
                    output[i][j][k] = 0
                elif rdn > thres:
                    output[i][j][k] = np.random.randint(255)
                else:
                    output[i][j][k] = image[i][j][k]
    return output

def carga_imagen_red(i):
    path='./data/imagenes/imagen_'+str(i)+'.jpg'
    frame=cv2.imread(path)
    n=np.random.randint(6)
    if n>4:
        n=n/4
        frame2=sp_noise(frame,0.01*n)
        frame=cv2.bilateralFilter(frame2,20,20,20)
    frame4=cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
    frame5=frame4.astype('float32')/255
    return frame5

#Carga de datos:

n_prueba=n_train+n_val+1000
x_train=np.zeros((n_train,128,128,3),dtype=np.float32)
y_train=np.zeros((n_train,2),dtype=np.float32)
x_val=np.zeros((n_val,128,128,3),dtype=np.float32)
y_val=np.zeros((n_val,2),dtype=np.float32)
array_lectura=lectura_txt(n_prueba+10)

print('cargando datos de entrenamiento...')
for i in range(n_train):
    if (i+1)%100==0:
        print(i/n_train*100,'%')
    i_x=np.random.randint(n_prueba, size=1)[0]
    n_imagen=int(array_lectura[i_x][0])
    y_train[i][0],y_train[i][1]=array_lectura[i_x][1],array_lectura[i_x][2]
    x_train[i]=carga_imagen_red(n_imagen)

print('cargando datos de validacion...')
for i in range(n_val):
    if (i+1)%100==0:
        print(i/n_val*100,'%')
    i_x=np.random.randint(n_prueba, size=1)[0]
    n_imagen=int(array_lectura[i_x][0])
```

```
    y_val[i][0],y_val[i][1]=array_lectura[i_x][1],array_lectura[i_x][2]
    x_val[i]=carga_imagen_red(n_imagen)

print('datos cargados')

model=red_secuencial()
history=model.fit(x_train,y_train,batch_size=15,epochs=20,valida-
tion_data=(x_val, y_val))
print('Finish')
```

6.1.5 Script Resultados Redes

```
#Anlisis de resultados
import b0RemoteApi
import math
import time
import numpy as np
import cv2
import tensorflow
from tensorflow.python.keras.preprocessing.image import load_img, img_to_array
from tensorflow.python.keras.models import load_model
import tkinter
from PIL import ImageTk, Image

#INICIO:
client = b0RemoteApi.RemoteApiClient('b0RemoteApi_CoppeliaSim','b0RemoteApi')

#INICIALIZACION
ret=False
rett=np.zeros((19,), dtype=bool)
while(ret!=True):
    rett[0],Brazo1=client.simxGetObjectHandle('IRB140_joint1',client.simx-
ServiceCall())
    rett[1],Brazo2=client.simxGetObjectHandle('IRB140_joint2',client.simx-
ServiceCall())
    rett[2],Brazo3=client.simxGetObjectHandle('IRB140_joint3',client.simx-
ServiceCall())
    rett[3],Brazo4=client.simxGetObjectHandle('IRB140_joint4',client.simx-
ServiceCall())
    rett[4],Brazo5=client.simxGetObjectHandle('IRB140_joint5',client.simx-
ServiceCall())
    rett[5],Brazo6=client.simxGetObjectHandle('IRB140_joint6',client.simx-
ServiceCall())
    rett[6],MotorUD=client.simxGetObjectHandle('MotorUD',client.simxService-
Call())
    rett[7],MotorLR=client.simxGetObjectHandle('MotorLR',client.simxService-
Call())
    rett[8],Sensor=client.simxGetObjectHandle('Proximity_sensor2',client.simx-
ServiceCall())
    rett[9],Cam=client.simxGetObjectHandle('Vision_sensor',client.simxService-
Call())
    rett[10],Cam_2=client.simxGetObjectHandle('Vision_general',client.simx-
ServiceCall())
    rett[11],Light=client.simxGetObjectHandle('Luz',client.simxServiceCall())
    rett[12],Sensor1=client.simxGetObjectHandle('Proximity_sensor',client.simx-
ServiceCall())
    rett[13],m_mfr=client.simxGetObjectHandle('joint_front_right_wheel#0',cli-
ent.simxServiceCall())
    rett[14],m_mfl=client.simxGetObjectHandle('joint_front_left_wheel#0',cli-
ent.simxServiceCall())
    rett[15],m_mbr=client.simxGetObjectHandle('joint_back_right_wheel#0',cli-
ent.simxServiceCall())
    rett[16],m_mbl=client.simxGetObjectHandle('joint_back_left_wheel#0',cli-
ent.simxServiceCall())
    rett[17],summit=client.simxGetObjectHandle('Robotnik_Summit_XL',cli-
ent.simxServiceCall())
    rett[18],Sensor_summit=client.simxGetObjectHandle('Proximity_sensor3',cli-
ent.simxServiceCall())
```

```
    ret=True
    for i in range(len(rett)):
        ret=ret*rett[i]
        if rett[i]==False:
            print ('Error handler ',i)
    if ret==False:
        print('Hay algun error, volviendo a revisar handler')

#-------------------------------------------------------------------------
--------------
#Funciones que usaremos para la comprobacion de las redes neuronales

def pos_aleatorios():
    i=np.random.randint(100, size=3)
    j=np.random.randint(100, size=3)
    k=np.random.randint(100, size=3)
    #pos luz
    x=3*i[0]/100-1.2
    y=2*j[0]/100-2.2
    z=2*k[0]/100+1
    client.simxSetObjectPosition(Light,-1,[x,y,z],client.simxDefault-
Publisher())

    #pos cam
    x=0.1*i[1]/100+0.3#-0.5
    y=0.1*j[1]/100-0.88#-0.5
    z=0.1*k[1]/100+1.9#+0.4
    client.simxSetObjectPosition(Cam_2,-1,[x,y,z],client.simxDefault-
Publisher())

    #pos summit
    x=(0.55-0.2)*i[2]/100+0.2
    y=-(1.1-0.71)*j[2]/100-0.71
    z=360*k[2]/100-180
    client.simxSetObjectPosition(summit,-1,[x,y,0.35],client.simxDefault-
Publisher())
    client.simxSetObjectOrientation(summit,-1,[0,0,z],client.simxDefault-
Publisher())
    return x,y

def imagen_a_red(imagen):
    frame2=cv2.cvtColor(imagen,cv2.COLOR_BGR2RGB)
    frame3=frame2.astype('float32')/255
    return np.expand_dims(frame3,axis=0)

def tuple2image(im_pre):
    reso=128,128
    frame2=np.zeros((reso[0], reso[1], 3))
    frame2=frame2.astype('uint8')
    #La imagen vendra en valores de 0 a 1, hay que pasarlo a 0 a 255 redon-
deando a entero
    j=0
    k=0
    l=0
    for i in range(len(im_pre)):
        if j%3==0 and j!=0:
            j=0
            k=k+1
        if k%reso[0]==0 and k!=0:
            j=0
            k=0
            l=l+1
        frame2[l][reso[0]-k-1][j]=round(im_pre[len(im_pre)-i-1]*255,0)
        j=j+1
    return frame2
```

```
def captura():
    resul=client.simxCallScriptFunction('myFunctionName@Vision_sen-
sor',"sim.scripttype_childscript",[], client.simxServiceCall())
    im_pre=resul[1][0]
    return tuple2image(im_pre)

def captura_summit():
    resul=client.simxCallScriptFunction('myFunctionName@Vision_gen-
eral',"sim.scripttype_childscript",[], client.simxServiceCall())
    im_pre=resul[1][0]
    return tuple2image(im_pre)

def media_eliminatoria(pos_pred1,pos_pred2,pos_pred3,pos_pred4):
    error=np.zeros((4,3))
    pos_pred=np.zeros((4,2))
    pos_pred[0]=pos_pred1
    pos_pred[1]=pos_pred2
    pos_pred[2]=pos_pred3
    pos_pred[3]=pos_pred4
    pos_media=(pos_pred1+pos_pred2+pos_pred3+pos_pred4)/4
    i_max=0
    error_max=0
    for i in range(4):
        error[i][0]=pos_media[0]-pos_pred[i][0]
        error[i][1]=pos_media[1]-pos_pred[i][1]
        error[i][2]=(error[i][0]**2+error[i][1]**2)**0.5
        if error[i][2]>error_max:
            error_max=error[i][2]
            i_max=i
    #Hacer media sin i_max
    pos_media2=0
    for i in range(4):
        if i!=i_max:
            pos_media2+=pos_pred[i]
    return pos_media2/3

def predicion_pos(model1,model2):
    time.sleep(1)
    imagen1=captura_summit()
    pos_pred1=model1.predict(imagen_a_red(imagen1))
    pos_pred2=model2.predict(imagen_a_red(imagen1))
    time.sleep(1)
    imagen2=captura_summit()
    pos_pred3=model1.predict(imagen_a_red(imagen2))
    pos_pred4=model2.predict(imagen_a_red(imagen2))

    return media_eliminato-
ria(pos_pred1[0],pos_pred2[0],pos_pred3[0],pos_pred4[0])

class pieza:
    def numero():
        exist=False
        j=9
        while(exist==False):
            j=j+1
            objeto_in='Objeto'+str(j)
            ret0,objeto_n=client.simxGetObjectHandle(objeto_in,client.simx-
ServiceCall())
            if ret0==False:
                exist=True
        return j
    def colocar(objeto_n,x,y,z):
        client.simxSetObjectPosition(objeto_n,-1,[x,y,z],client.simxDefaultPu-
blisher())
        client.simxSetObjectQuaternion(objeto_n,-1,[0,0,0,0],client.simxDe-
faultPublisher())
    def aleatoria():
        i=np.random.randint(9, size=1)[0]
```

```
        i+=1
        shape_i='Shape'+str(i)
        objeto_i='Objeto'+str(i)
        objeto_in='Objeto'+str(pieza.numero())
        ret0,objeto=client.simxGetObjectHandle(objeto_i,client.simxService-
Call())
        ret1,shape=client.simxGetObjectHandle(shape_i,client.simxServiceCall())
        item1,item2=client.simxCopyPasteObjects([objeto,shape],0,client.simx-
ServiceCall())[1]
        ret0,objeto_n=client.simxGetObjectHandle(objeto_in,client.simxService-
Call())
        pieza.colocar(objeto_n,-0.4045,0.8,0.827)
        return i
    def aleatoria_en_sitio(xi,yj):
        x=-1.1705
        y=-1.6742-(1.5597-1.6742)*xi
        z=+1.7001-(1.7001-1.4001)*yj
        i=np.random.randint(10, size=1)[0]
        if not(i==0):
            shape_i='Shape'+str(i)
            objeto_i='Objeto'+str(i)
            ret0,objeto=client.simxGetObjectHandle(objeto_i,client.simxService-
Call())
            ret1,shape=client.simxGetObjectHandle(shape_i,client.simxService-
Call())
            item=client.simxCopyPasteObjects([objeto,shape],0,client.simx-
ServiceCall())[1][0]
            pieza.colocar(item,x,y,z)
        return i

def estanteria_aleatoria():
    mat_real=np.zeros((4,9), dtype=int)
    for i in range(4):
        for j in range(9):
            mat_real[i][j]=pieza.aleatoria_en_sitio(j,i)
    return mat_real

def recorte(image,j,k):
    x=6+j*29 #110/4 = 27.5
    y=10+k*12 #110/9 = 12.22
    image_cutted=image[x:x+30, y:y+15]
    return image_cutted

def imagen_a_red(imagen):
    frame2=cv2.cvtColor(imagen,cv2.COLOR_BGR2RGB)
    frame3=frame2.astype('float32')/255
    return np.expand_dims(frame3,axis=0)

def objeto_camara():
    frame=captura()
    frame2=cv2.resize(frame,(15,30))
    x=imagen_a_red(frame2)
    resultado=np.argmax(cnn.predict(x)[0])
    return resultado

def revisar_estanteria():
    mat_est_e=np.zeros((4,9), dtype=int)
    #posicion.esconder()
    frame=captura()
    for i in range(4):
        for j in range(9):
            frame2=recorte(frame,i,j)
            x=imagen_a_red(frame2)
            mat_est_e[i][j]=np.argmax(cnn.predict(x)[0])
    return mat_est_e

def mat_dif(mat_real,mat_predicha):
```

```
    dif=0
    for i in range(4):
        for j in range(9):
            if mat_real[i][j]!=mat_predicha[i][j]:
                dif+=1
                print('ERROR!!!',i,j)
    if dif>0:
        print(mat_real)
        print(mat_predicha)
    return dif

#Red analisis de estanteria
modeloRED1='modelo_prueba_1'
cnn=load_model('./modelos red 1/'+modeloRED1+'.h5')
print(cnn.summary())
errores=0
n_ensayos=100
t_medio=0
for i in range(n_ensayos):
    print('Ensayo:',i)
    pos_aleatorios()
    while(client.simxGetSimulationState(client.simxServiceCall())[1]!=16):
        client.simxStartSimulation(client.simxServiceCall())
    mat_real=estanteria_aleatoria()
    time.sleep(2)
    t_i=time.time()
    mat_pred=revisar_estanteria()
    tiempo=time.time()-t_i
    t_medio+=tiempo
    print("Tiempo analisis estanteria: %0.10f s." % tiempo)

    time.sleep(2)
    while(client.simxGetSimulationState(client.simxServiceCall())[1]!=0):
        client.simxStopSimulation(client.simxServiceCall())
    errores+=mat_dif(mat_real,mat_pred)
print(errores, t_medio/n_ensayos)

#Red Posicion Summit
model1=load_model('./modelos red 2/modelo_summit1.h5')
print(model1.summary())

x_max=0.55-0.2
y_max=1.1-0.71
n_ensayos=100

error_med1=0
error_rel_med1=0
peligros1=0
t_medio1=0


for i in range(n_ensayos):
    x_real,y_real=pos_aleatorios() #Inicio aleatorio
    client.simxStartSimulation(client.simxServiceCall())
    time.sleep(2)
    imagen1=captura_summit()

    t_i=time.time()
    pos_pred=model1.predict(imagen_a_red(imagen1))[0]
    tiempo=time.time()-t_i
    t_medio1+=tiempo
    x_pred,y_pred=pos_pred[0],pos_pred[1]
    error=((x_real-x_pred)**2+(y_real-y_pred)**2)**0.5
    error_rel=(((x_real-x_pred)/x_max)**2+((y_real-y_pred)/y_max)**2)**0.5
    error_rel_med1+=error_rel
    error_med1+=error
    if error>0.05:
        peligros1+=1
```

```
    client.simxStopSimulation(client.simxServiceCall())
    time.sleep(2)
print('1',peligros1, peligros1/n_ensayos*100,'%',round(error_rel_med1/n_ensa-
yos*100,5),'%','t=',t_medio1/n_ensayos*1000)
```