Diseño e implementación de un sistema virtual de detección de la pose 3D de un objeto mediante técnicas de aprendizaje profundo para aplicaciones robóticas

Chiliquinga Chuquitarco, Richard Alexander

Departamento de Ciencias de la Energía y Mecánica

Carrera de Ingeniería Mecatrónica

Trabajo de titulación, previo a la obtención del título de Ingeniero Mecatrónico

Ing. Constante Prócel, Patricia Nataly

23 de agosto del 2020

DEPARTAMENTO DE CIENCIAS DE LA ENERGÍA Y MECÁNICA

CARRERA DE INGENIERÍA MECATRÓNICA

Certificación

Certifico que el trabajo de titulación, **"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA VIRTUAL DE DETECCIÓN DE LA POSE 3D DE UN OBJETO MEDIANTE TÉCNICAS DE APRENDIZAJE PROFUNDO PARA APLICACIONES ROBÓTICAS"** fue realizado por el señor **Chiliquinga Chuquitarco, Richard Alexander** el cual ha sido revisado y analizado en su totalidad por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Latacunga, 23 de agosto de 2021

Firma:

Firmado electrónicamente por:
PATRICIA NATALY
CONSTANTE PROCEL

……..

Ing. Constante Prócel, Patricia Nataly

DIRECTORA

C. C. 0503354029

Curiginal

Document Information

Analyzed document	Trabajo_Titulación_Chiliquinga Chuquitarco_Richard Alexander.pdf (D111720587)
Submitted	8/26/2021 10:06:00 PM
Submitted by	Lorena Ibarra
Submitter email	loretaibarra@yahoo.es
Similarity	3%
Analysis address	lorenadibarra.uta@analysis.urkund.com

Sources included in the report

W	URL: https://dspace.ups.edu.ec/bitstream/123456789/19132/1/UPS-CT008818.pdf Fetched: 8/26/2021 10:07:00 PM	2
SA	**UNIVERSIDAD TECNICA DE AMBATO / [Tesis] Bastidas Guayanay Daniela Alexandra y Gallegos Velásquez Dayana Esther.pdf** Document [Tesis] Bastidas Guayanay Daniela Alexandra y Gallegos Velásquez Dayana Esther.pdf (D111681142) Submitted by: rtoasa@uisrael.edu.ec Receiver: dguevara.uta@analysis.urkund.com	4
W	URL: https://repositorio.urp.edu.pe/bitstream/handle/URP/3523/ELEC-T030_46733086_T%20%20%20CAYA%20P%C3%89REZ%20JHAN%20CARLOS.pdf?sequence=1&isAllowed=y Fetched: 6/10/2021 1:12:41 PM	1
W	URL: https://blog.unity.com/manufacturing/toyota-makes-mixed-reality-magic-with-unity-and-microsoft-hololens-2 Fetched: 8/26/2021 10:07:00 PM	2
W	URL: https://www.usenix.org/sites/default/files/osdi16_full_proceedings.pdf Fetched: 8/26/2021 10:07:00 PM	1
W	URL: http://docplayer.es/182316720-Desarrollo-de-un-sistema-de-deteccion-y-prediccion-de-la-pose-3d-de-objetos-en-la-escena-mediante-tecnicas-de-deep-learning.html Fetched: 3/29/2021 9:44:08 AM	1
W	URL: https://labs.laan.com/blog/real-time-3d-car-pose-estimation-trained-on-synthetic-data.html Fetched: 8/26/2021 10:07:00 PM	1
W	URL: https://idus.us.es/bitstream/handle/11441/107057/TFG-3342-GUILLEN%20CANO.pdf?sequence=1&isAllowed=y Fetched: 8/20/2021 6:58:29 AM	1
W	URL: https://medium.com/analytics-vidhya/comparing-gpu-performance-for-deep-learning-between-pop-os-ubuntu-and-windows-69aa3973cc1f Fetched: 8/26/2021 10:07:00 PM	1
W	URL: https://www.researchgate.net/profile/Jose_Flores_Albino/publication/323460108_Deep_Learning_para_la_Deteccion_de_Peatones_y_Vehiculos/links/5a9747bea6fdccecff0b1bae/Deep-Learning-para-la-Deteccion-de-Peatones-y-Vehiculos.pdf	1

Firmado electrónicamente por:
PATRICIA NATALY CONSTANTE PROCEL

.....

Ing. Constante Prócel, Patricia Nataly

DIRECTORA

DEPARTAMENTO DE CIENCIAS DE LA ENERGÍA Y MECÁNICA

CARRERA DE INGENIERÍA MECATRÓNICA

Responsabilidad de autoría

Yo, **Chiliquinga Chuquitarco, Richard Alexander**, con cédula de ciudadanía n° 0550047278, declaro que el contenido, ideas y criterios del trabajo de titulación: **"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA VIRTUAL DE DETECCIÓN DE LA POSE 3D DE UN OBJETO MEDIANTE TÉCNICAS DE APRENDIZAJE PROFUNDO PARA APLICACIONES ROBÓTICAS"** es de mi autoría y responsabilidad, cumpliendo con los requisitos legales, teóricos, científicos, técnicos, y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Latacunga, 23 de agosto de 2021

Firma:

……

Chiliquinga Chuquitarco, Richard Alexander

C.C.: 0550047278

DEPARTAMENTO DE CIENCIAS DE LA ENERGÍA Y MECÁNICA

CARRERA DE INGENIERÍA MECATRÓNICA

Autorización de publicación

Yo **Chiliquinga Chuquitarco, Richard Alexander,** con cédula de ciudadanía n° 0550047278, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: **"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA VIRTUAL DE DETECCIÓN DE LA POSE 3D DE UN OBJETO MEDIANTE TÉCNICAS DE APRENDIZAJE PROFUNDO PARA APLICACIONES ROBÓTICAS"** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Latacunga, 23 de agosto de 2021

Firma:

……

Chiliquinga Chuquitarco, Richard Alexander

C.C.: 0550047278

Agradecimiento

Agradezco a Dios por permitirme llegar a donde estoy, bendecirme en cada paso y estar en los momentos más importantes de mi vida a lo largo de estos 25 años.

A la Ingeniera Patricia Constante, directora de tesis, por brindarme toda su experiencia y conocimiento científico en el desarrollo del tema; la considero una excelente docente de la universidad por su trabajo permanente y gran pasión en el aprendizaje continuo, que con su carisma me ha motivado a seguir creciendo.

A mis padres, principalmente a mi madre Rosita por acompañarme y darme fuerzas todos los días con su alegría, amor, paciencia y valores. Todo te lo debo a ti, tu me has hecho la persona que soy ahora y soy feliz con todo esto.

A mis hermanos Joseph, Jeanine por estar presentes y acompañarme en todo el camino, que con su pequeña ayuda han sido grandes cosas para mí; sobre lo demás su compañía me ha motivado mucho en tantas alegrías y tristezas.

A Evelyn, por ser el impulso cada día para no rendirme ante las adversidades; que con sus consejos me alegraba el día de continuar durante estos últimos años.

A mis amigos, en especial Jair, Sebastián, Daniel, Anderson, Diego, Yordan por confiar y creer en mí, por cada momento compartido durante el tiempo de nuestra preparación en la universidad, son buenos momentos que recordaremos toda la vida.

Dedicatoria

Quiero dedicar el presente trabajo a todas las personas que estuvieron presentes en mi vida, desde mi educación inicial hasta mi educación superior porque cada pequeña enseñanza y consejo me han permitido llegar hasta aquí.

A mis docentes, compañeros de aula y amigos que en el transcurso de la carrera hemos luchado hombro con hombre, a ellos se los dedico por todas las esperanzas puestas en mi crecimiento continuo.

A mis queridos amigos, Kari, Eli, Kathy, Dennis, Andrea, Dianita, Eli, Andrés, Bryan, Diego, Santi, Carlitos, Rus, Greco, Kimy y Jose que me han demostrado un apoyo incondicional desde que los he conocido; esta meta la comparto con ustedes por ese aprecio inmenso que les tengo y por cada vivencia juntos, en nuestro grupo de amigos.

Tabla de contenidos

Índice de tablas

Índice de figuras

Resumen

El presente proyecto realiza el diseño de un sistema virtual detector de la pose 3D de objetos mediante técnicas de aprendizaje profundo para aplicaciones robóticas, esto con el fin de proveer una herramienta académica-práctica a los estudiantes en su educación virtual. Se plantea una arquitectura del sistema virtual manejada en cuatro etapas, desde la generación de un conjunto de datos mediante el enfoque de imágenes sintéticas fotorrealistas y de aleatorización de dominio. La siguiente etapa hace referencia al método de aprendizaje profundo para el entrenamiento de redes neuronales convolucionales sobre el marco de aprendizaje Pytorch previa la selección de hiperparámetros que permiten a la red realizar una abstracción de las características del objeto. La tercera etapa integra el detector de pose 3D para conseguir los valores de posición y orientación del objeto en tiempo real, la inferencia del detector trabaja sobre imágenes publicadas por el entorno virtual o en tiempo real desde la cámara web del ordenador. Finalmente, se realiza la comunicación de estas etapas mediante sistemas operativos de robots para el envío de los resultados obtenido por el detector hacia el brazo robótico Mitsubishi RV-2SDB previo el cálculo de su configuración de cinemática inversa para la ejecución de la aplicación de recoger y colocar objetos en el entorno virtual. Los resultados obtenidos en la detección de pose indican buenos resultados en imágenes del mundo exterior con datos sintéticos como en el agarre de objetos sobre el entorno virtual, con un buen grado de satisfacción e interés por los usuarios de destino.

Palabras clave:

- **VISIÓN ARTIFICIAL**
- **REDES NEURONALES**
- **ROBÓTICA**

Abstract

This project designs a virtual system that detects the 3D pose of objects through deep learning techniques for robotic applications, to provide an academic-practical tool for students in their virtual education. We propose a virtual system architecture managed in four stages, from the generation of a dataset using the photorealistic synthetic images and domain randomization approach. The next stage refers to the deep learning method for the training of convolutional neural networks on the Pytorch learning framework before the selection of hyperparameters that allow the network to perform an abstraction of the object's characteristics. The third stage integrates the 3D pose detector to obtain the position and orientation values of the object in real-time, the inference of the detector works on images published by the virtual environment or in real-time from the computer's webcam. Finally, the communication of these stages is performed using robot operating systems to send the results obtained by the detector to the Mitsubishi RV-2SDB robotic arm before the calculation of its inverse kinematics configuration for the execution of the application of picking up and placing objects in the virtual environment. The results obtained in the pose detection indicate good results in images of the outside world with synthetic data as in the grasping of objects on the virtual environment, with a good degree of satisfaction and interest by the target users.

Keywords:

- **ARTIFICIAL VISION**
- **NEURAL NETWORKS**
- **ROBOTIC**

Capítulo I

1. Generalidades

1.1. Descripción del problema

1.1.1. Planteamiento del problema

En la actualidad, el desarrollo tecnológico en el sector industrial del Ecuador es menor en comparación a otros países sudamericanos, en el último índice de competitividad global publicado por el Foro Económico Mundial, Ecuador pasó del puesto 102 al 110 en el ámbito de innovación entre 2016 y 2017. Aunque no existen cifras exactas, la digitalización de la industria es gradual gracias al avance tecnológico; se hace mención a empresas como General Motors, Mexichem Plastigama, MivilTech Soluciones Industriales que han adoptado la digitalización de la industria como un enfoque hacia la creación de valor para el negocio en la optimización de su producción.

La manipulación de objetos es una de las aplicaciones más comunes en robótica, la mayoría de los robots industriales en empresas y laboratorios de educación superior locales carecen de percepción, realmente no tienen un sentido del mundo que los rodea, lo cual implica que no puedan manipular objetos incluso cuando esos objetos no estén en el mismo lugar cada vez. Esto lleva a buscar equipos externos alternativos, el costo de adquisición representa una inversión elevada, sumando otros gastos como la adquisición de licencias de software privativo, el pago de impuestos y aranceles correspondientes a los manejos de importación, la ausencia de repuestos como consecuencia del uso de tecnología extranjera.

La emergencia sanitaria que afectó al Ecuador como consecuencia de epidemia ha llevado a la Universidad de las Fuerzas Armadas ESPE a emitir el instructivo N.º UDED-INS-V1-2020-002 “Para la implementación del Plan de Contingencia para el desarrollo académico de los contenidos de las asignaturas de grado y tecnología, en la modalidad presencial, empleando herramientas pedagógicas y didácticas utilizadas en educación virtual”. Esto con el fin de precautelar la salud de los estudiantes, pero con una gran limitación al uso de laboratorios técnicos industriales que permitían desarrollar de forma práctica el conocimiento teórico a fin. Por lo tanto, realizar una investigación que permita el diseño de un sistema de detección de la pose 3D de un objeto mediante técnicas de aprendizaje profundo para aplicaciones robóticas validadas en un ambiente virtual, busca el desarrollo de prácticas de tipo académico-práctico, con fácil utilización a través del ordenador para ser usado por estudiantes de forma eficiente y segura dada las limitaciones al acceso de laboratorios de la Universidad de las Fuerzas Armadas ESPE.

1.1.2. Justificación e importancia

El desarrollo de este proyecto permitirá un sistema de apoyo para los estudiantes de la Universidad de las Fuerzas Armadas ESPE sede Latacunga para la ejecución de prácticas que refuercen los temas de Robótica y Visión Artificial primordialmente, desde su ordenador sin acceder a los laboratorios presencialmente los cuales cuentan con acceso restringido y limitaciones por la crisis del covid-19 dentro del país.

El desarrollo de nuevas tecnologías dentro de la visión artificial robótica permitirá percibir en la cultura laboral ecuatoriana nuevos modos de producción en base a la

innovación generada. Dentro del entorno comercial existen instituciones extranjeras dedicadas a ofrecer estos servicios con aplicaciones industriales cuyos costos son elevados, dado que este tipo de tecnología no se encuentra en desarrollo en el país lo que impulsa una comercialización de equipos o asesorías de empresas extranjeras.

La importancia del presente proyecto es desarrollar un sistema virtual donde se refleje el uso de nuevas tecnologías en visión artificial para la detección de la pose 3D de un objeto mediante técnicas de aprendizaje profundo para aplicaciones robóticas dentro de un simulador de entornos virtuales, que a nivel educativo capacite a diferentes especialidades de Ingeniería afines al tema y promueva el desarrollo de prácticas de laboratorio.

1.1.3. Objetivos

a. Objetivo General

- Diseñar e implementar un sistema virtual de detección de la pose 3D de un objeto mediante técnicas de aprendizaje profundo para aplicaciones robóticas.

b. Objetivos Específicos

- Recopilar información sobre el aprendizaje profundo aplicados a la detección de la pose 3D de un objeto y el tipo de aplicación robótica en la que se usa.
- Implementar un algoritmo de detección de pose 3D de objetos con redes neuronales convolucionales a base de datos sintéticos.

- Realizar la comunicación del simulador de entorno virtual con el sistema de detección de pose 3D para la aplicación con el robot Mitsubishi conocida su configuración de cinemática inversa.
- Realizar pruebas de funcionamiento del sistema virtual para evaluar los resultados en el campo de la robótica.

1.1.4. Hipótesis

¿El diseño e implementación de un sistema virtual basado en técnicas de aprendizaje profundo ayudará a la detección de la pose 3D de un objeto para aplicaciones robóticas?

1.1.5. Variables de la investigación

a. Variable independiente

Detección de la pose 3D de un objeto mediante técnicas de aprendizaje profundo

b. Variable dependiente

Sistema virtual para aplicaciones robóticas

1.2. Estado del arte

Los sistemas de visión artificial claves de la Industria 4.0, poseen un potencial enorme por su capacidad de percibir, comprender y actuar en tiempo real para dar solución a diversos problemas presentes en la actualidad (por ejemplo, control de calidad, seguridad, medicina, etc.). Las máquinas que integran dicho sistema analizan y decodifican la información mediante un software vinculado que responde a través de un proceso automatizado.

La importancia de la tecnología en las últimas décadas ha logrado que la rama de la robótica comparta espacio con el ser humano en el sector industrial. Por esto, Ioana (2019) menciona que “la robótica está sujeta a un constante cambio y evolución, pudiéndose desarrollar infinidad de aplicaciones con el fin de hacer que las tareas realizadas por las personas sean más fáciles” (p. 10). En las tareas de acciones repetitivas y de precisión, se caracterizan los brazos robóticos industriales que están ligados a una tarea específica de acuerdo con su actuador final.

El hecho de que los robots reciban instrucciones y actúen en consecuencia los limita a reaccionar ante las variables físicas del escenario que se encuentren. Sin embargo, con el potencial de los algoritmos de visión artificial estas máquinas serían capaces de reconocer, localizar y manejar objetos con mayor autonomía (Iglesias, 2010). De este hecho, la detección de la pose 3D de objetos en una escena es uno de los algoritmos más populares en el campo de visión artificial.

Al hablar de técnicas de aprendizaje de estos algoritmos, en Salazar (2019) menciona que el uso de algoritmos basados en técnicas de aprendizaje automatizado y

aprendizaje profundo son más populares hoy en día. Su objetivo general fue el de implementar un sistema autónomo de visión artificial en base al algoritmo YOLOv3 sobre una placa NVIDIA Jetson TX2 basado en un framework Linux-Python-OpenCVTensor-Cuda para la detección de autos y personas en video capturado a tiempo real. En las pruebas realizadas alcanza una tasa de acierto superior al noventa por ciento con YOLOv3 en versión completa, aunque la velocidad de procesamiento del detector resultó inferior.

Sobre el tema no existen investigaciones realizadas a nivel nacional de detección de pose 3D de un objeto mediante técnicas de aprendizaje profundo para aplicaciones robóticas, por ende, se ha llevado la búsqueda externa de esta. En el caso de Pavlakos, Zhou, Chan, Derpanis, & Daniilidis (2017) plantearon la identificación de la pose de un objeto a partir de una sola imagen monocular, al predecir los puntos claves semánticos en la imagen 2D junto al modelo 3D del objeto utilizando una cámara en perspectiva. El conjunto de datos PASCAL3D +, donde se demostraron resultados de vanguardia para la estimación del punto de vista. Además, el método va acompañado de una implementación eficiente con un tiempo de ejecución inferior a 0,3 segundos.

De la misma forma, en aplicaciones para teléfonos móviles inteligentes han sido partícipes estos algoritmos de visión artificial. En Laan (2018) se planteó una aplicación que guiara a capturar fotos de coches de mayor calidad a través de una guía de realidad aumentada y captura automática de fotos. Su objetivo principal el de generar una componente para la pose de los automóviles en base a la API de Apple, ARKit; su búsqueda de un conjunto de datos de automóviles requería modelos actuales en diferentes ambientes. A través de la utilización de motores gráficos se generó una base

para el entrenamiento con datos sintéticos como una forma de adaptar los datos variando el entorno de prueba.

1.3. Fundamentación teórica

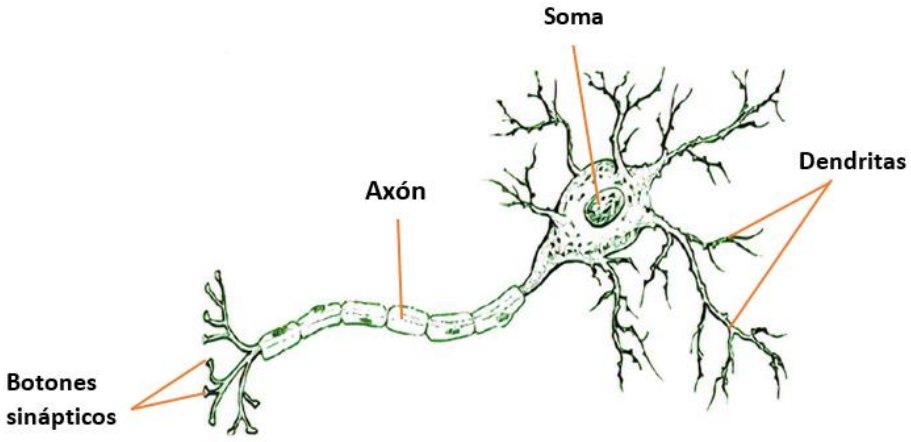
1.3.1. Redes neuronales

a. Redes neuronales biológicas

El modelo biológico de estas neuronas son la base para entender las interconexiones de la red, se observa en la Figura 1 su estructura básica forma de cuatro regiones: el soma (cuerpo celular), las dendritas, los axones y los botones sinápticos. Por las dendritas se propagan las señalas de entrada en toda la longitud del axón mediante una respuesta sináptica a otras neuronas (Hanrahan, 2011).

Figura 1

Estructura de una neurona biológica



Nota. Adaptado de *Componentes de una neurona* (p.3), por Basogain, 2001, Xabier Basogain Olabe.

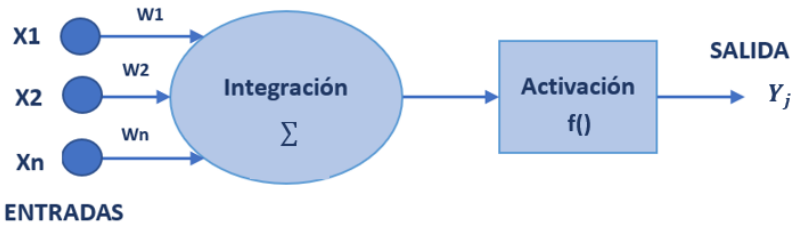
b. Redes neuronales artificiales

En el campo científico las Redes Neuronales Artificiales (RNA) imitan en esquema una red neuronal biológica, basado en el aprendizaje por la experiencia y extrayendo datos de esta (Flórez & Fernández, 2008), la Figura 2 muestra la neurona artificial de forma representativa de esta clase de redes.

El interés de estudio a las redes neuronales artificiales radica en tres conceptos fundamentales, primero la excitación simultánea de dos neuronas provoca un fortalecimiento de las conexiones; en segundo lugar, el aumento de la tasa de neuronas activadas es esencial en el aumento de la inhibición, dando lugar a una fuerte conexión y por último la función de transferencia como tasa de variación referente a las entradas que recibe la neurona (Hanrahan, 2011).

Figura 2

Partes de una neurona artificial



Nota. La descripción de las partes de la red neuronal artificial se presenta en la Tabla 1.

Las RNA son capaces de ajustar la salida en respuesta a sus entradas, su esquema básico como se puede observar en la Figura 3, está compuesto por capas que modifican los valores de entrenamiento asociados a los elementos de procesamiento; la

capacidad de la red reside en su topología(conexiones), en los pesos de sus conexiones y las funciones o métodos de aprendizaje.

Una alternativa donde los procesos pueden generar resultados imprecisos e incompletos en los métodos de conocimiento natural (Rabulan & Dorado, 2006).

Tabla 1

Definiciones de la estructura de una neurona artificial

Nombre	Símbolo	Definición
Entradas	X_i	Los datos de entrada recibida por otras neuronas
Pesos sinápticos	W_{ij}	Peso asociado de sinapsis conectando la entrada i con la salida de la neurona j
Integración	$\sum$	Integra todas las entradas con sus pesos a valores netos
Función de activación	$f(\,)$	Recoge el estímulo neto para determinar la salida, generalmente no lineal.
Salida	Y_j	Salida de la neurona sea binario o real

Nota. Tomado de (Berzal, 2018)

1.3.2. Arquitectura de las redes

Flórez & Fernández (2008) describe que la topología de las redes neuronales artificiales se encuentran divididas en tres categorías principales:

Por la estructura en capas

- **Redes monocapa:** cuentan con una única capa de neuronas de entrada y otra de salida que a su vez es capa oculta, utilizada en tareas de auto-asociación.

- **Redes multicapa:** una extensión de la red monocapa al que una o más etapas se le han añadido, ayuda a reducir el número de unidades del proceso a cambio de más esfuerzo de computo al ser entrenadas.

Por el flujo de datos dentro de la red

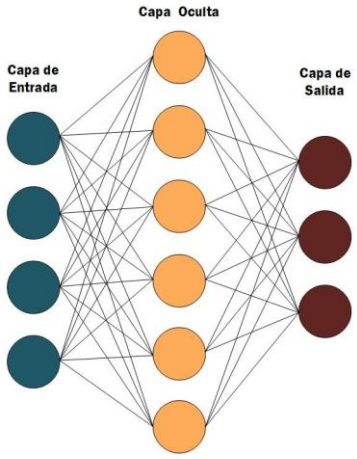
- **Redes de propagación hacia adelante:** la información viaja en un único sentido desde las capas de entrada hasta las de salida. Denominadas feedforward.
- **Redes de propagación hacia atrás:** llamados sistema recurrentes llevan los datos de capas superiores a un nivel previo o inferior. Denominadas feedback.

Por el tipo de información de la red

Se consideran las redes estáticas y dinámicas que incluyen el tiempo en los valores de entrada, suelen trabajarse en conjunto con las redes recurrentes y principalmente se desarrollan para problemas de optimización (Leonor, 2020).

Figura 3

Red neuronal simple



1.3.3. Aprendizaje Profundo

Del término traducido al español de Deep Learning es el subcampo de la Inteligencia Artificial que nombra a las arquitecturas con múltiples capas ocultas con el fin de aprender diversas características con múltiples niveles de abstracción. Estos modelos inspirados en la comprensión del cerebro biológico realizan buenas representaciones al tomar características de niveles inferiores en las capas superiores y formar conceptos más completos, resaltando una mejora en la extracción que presenta el enfoque de aprendizaje automático. Los modelos profundos han reaparecido en la última década en aplicaciones de visión artificial, texto, sonido, video, etc. (Wani, Bhat, Afzal, & Khan, 2020).

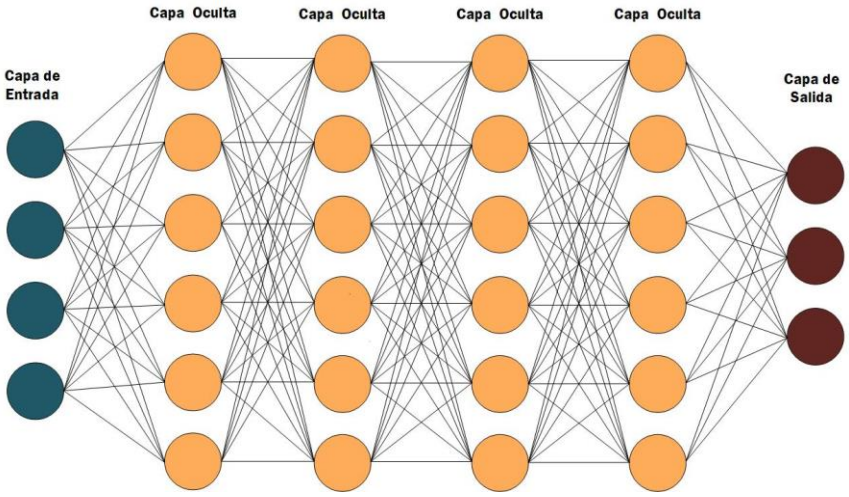
El éxito del aprendizaje profundo se ha manifestado por las mejoras en hardware como de software en la disponibilidad de GPU, más eficaz que una CPU para procesar y calcular grandes cantidades de información, sin dejar a un lado la disponibilidad de

bases de datos de entrenamiento brindado por personas, universidades y empresas; este es el caso de ImageNet representado por imágenes. (Gulli & Pal, 2017).

El perceptrón multicapa encontrado en la literatura de redes neuronales cómo se puede observar en la Figura 4, se refiere al Aprendizaje profundo dado que está compuesto por múltiples capas ocultas (Torres, First contact with Deep Learning , 2018).

Figura 4

Estructura de una red neuronal de aprendizaje profundo



1.3.4. Redes Neuronales Convolucionales

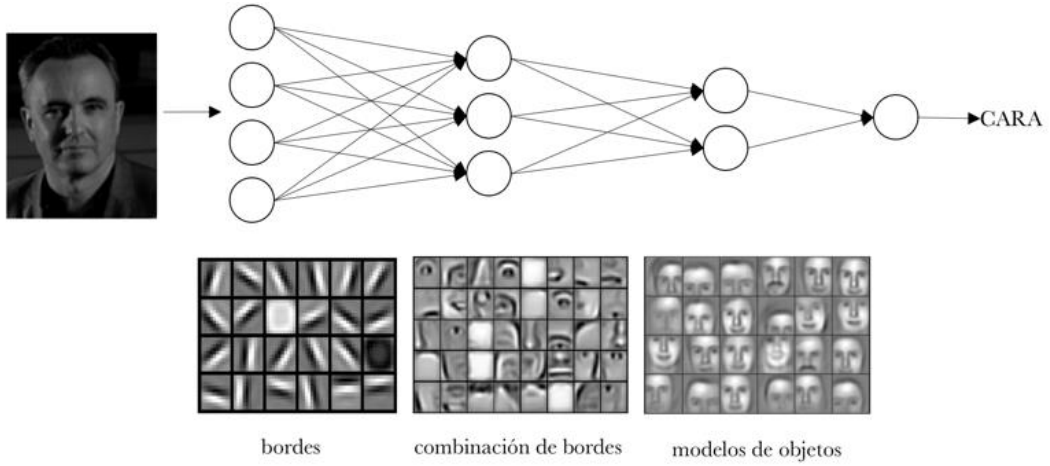
Las redes neuronales convolucionales (del acrónimo en inglés CNN o ConvNets) son redes de aprendizaje profundo que han impactado enormemente en el área de la visión por computadora (Berzal, 2018). Inspirada en la corteza visual funcionan en torno a cuatro ideas claves: conexiones locales, los pesos compartidos, la agrupación y una

alta cantidad de capas. Las CNN han sido aplicadas en amplios temas como la clasificación, detección, segmentación de imágenes. (LeCun, Bengio, & Hinton, 2015).

Como se muestra en la Figura 5 estas redes son similares a las RNA con el rasgo de poder codificar la arquitectura para encontrar características específicas en la imagen (Torres, Deep Learning Introducción Práctica con Keras, 2018).

Figura 5

Idea intuitiva de una red convolucional



Nota. El gráfico representa las características obtenidas por cada una de las capas hasta finalmente identificar que la imagen de entrada es una cara. Tomado de Deep Learning Introducción Práctica con Keras (p.148), por J.T, 2018, Jordi Torres.

En la mayoría de aplicaciones la arquitectura que presenta una red convolutiva combina tres etapas: la convolución, detector RELU y pooling (Torres, First contact with Deep Learning , 2018).

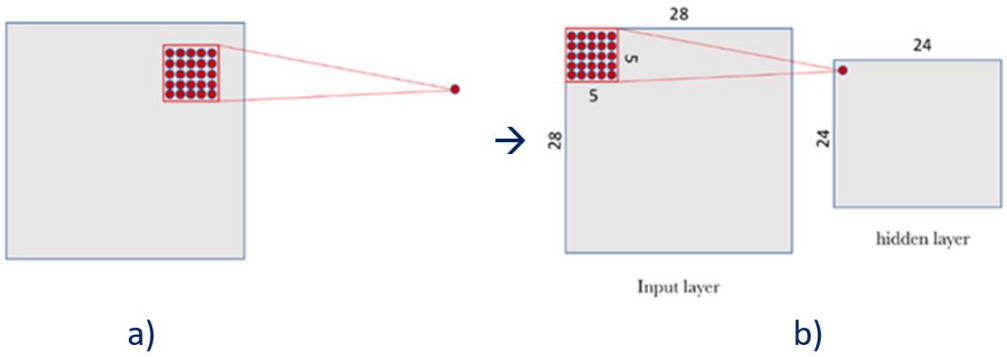
a. Capa convolucional

Para Berzal (2018) las capas convolutivas sustituyen la multiplicación de entradas por pesos de las capas densamente conectadas con una operación de convolución, realmente, una correlación cruzada entre dos funcions para obtener una función filtrada.

Estas operan sobre *feature maps* que se define como mapas de características, con dos ejes espaciales y un eje canal, altura, ancho y profundidad respectivamente. La señal de entrada de una capa convolucional está formada por un conjunto de neuronas en una pequeña región, llamada *kernel* o máscara (3x3,5x5,7x7 usadas en práctica); que va recorriendo toda la imagen en su totalidad y generando una nueva capa oculta con características únicas que sirve de entrada para la siguiente capa convolucional (Torres, Deep Learning Introducción Práctica con Keras, 2018), esta operación se muestra en la Figura 6.

Figura 6

Representación visual del filtro

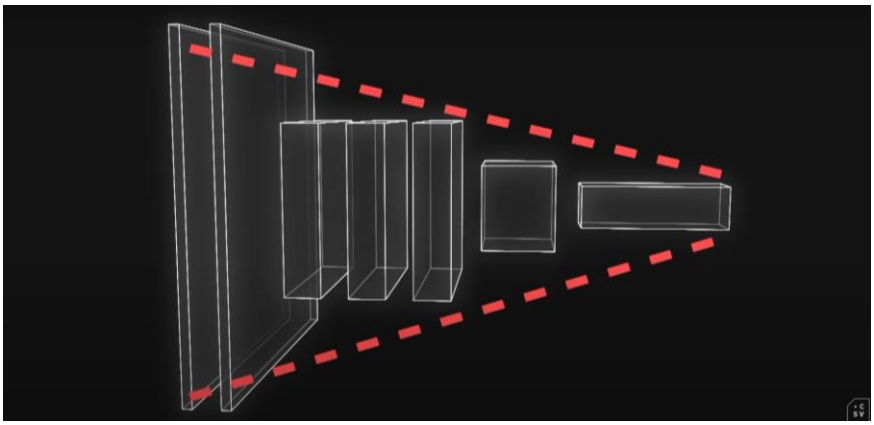


Nota. a) Conjunto de neuronas de un filtro de 5x5, b) el recorrido del filtro sobre la capa de entrada genera una capa oculta de dimensiones diferentes, esto se debe a la elección *stride* que definido como valor de paso. Tomado de Deep Learning Introducción Práctica con Keras (p.148), por J.T, 2018, Jordi Torres.

En el reconocimiento de imágenes no es suficiente usar un solo filtro por ello es factible usar varios filtros generando distintos mapas de características. La Figura 7 representa como normalmente la red neuronal convolucional es representada como un embudo donde la imagen de entrada en resolución va disminuyendo y su profundidad va aumentado.

Figura 7

Representación visual de una red neuronal convolucional



Nota. Tomado de (CSV, 2020)

Al momento de diseñar una capa de convolución hay que tomar en cuenta cuatro hiperparámetros: la profundidad, conectividad local, *padding* que se define como rellenado con ceros y el valor de paso.

- **Profundidad:** se refiere al número de filtros kernel de la capa convolutiva, esta determina la profundidad de las capas ocultas y a su vez la cantidad de características suficientes a detectar.

- **Conectividad local:** enfocada al tamaño del filtro normalmente de tamaños, p.ej. $3x3, 5x5, 7x7$ respetando la misma extensión espacial en horizontal y vertical.
- **Stride:** indica el movimiento del filtro sobre la imagen que determina cuántas muestras se desea tener a la salida, una mejor extracción de características realza el coste computacional.
- **Padding:** busca realizar ajustes en la resolución de los mapas de características al rellenar la entrada con ceros alrededor de la capa de entrada.

Aplicar una capa convolutiva genera una reducción significativa de establecer parámetros en una capa totalmente conectada, a diferencia de los hiperparámetros (Berzal, 2018).

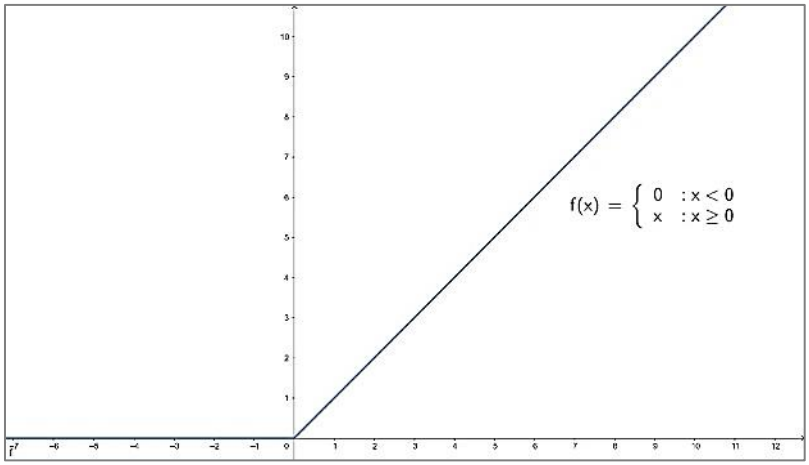
b. Función de activación

Las funciones de activación actúan en las entradas de una neurona al direccionarla hasta su salida, con el objetivo de limitar las amplitudes de salida y conseguir un modelo inherente. Cuando se trata de aprendizaje profundo en redes neuronales convolucionales la función de activación rectificada lineal (RELU) es la más popular por permitir el paso de valores positivos reales en el rango de cero hasta infinito, evitando la no linealidad en la función resultante (Valbuena, 2021)

Se puede ver en la Figura 8 la función RELU que matemáticamente es: $f(x) = máx(0, x)$

Figura 8

Función de activación RELU



c. Pooling

Las capas de pooling o submuestreo ejecutan una reducción espacial a la dimensión de profundidad y canales de las capas posteriores a la convolución. Dos de los métodos estadísticos más aplicados son en base a la relación de valores en un promedio llamado *average pooling* y el máximo de todos los valores en una región definido como *max pooling*, una de estas técnicas se representa en la Figura 9. De esa forma, se logra un trabajo con datos reducidos que se ve mejorado en el coste computacional (Berzal, 2018).

$$A = \begin{bmatrix} 4 & 2 & 5 & 3 \\ 6 & 2 & 8 & 1 \\ 1 & 9 & 5 & 7 \\ 5 & 6 & 2 & 10 \end{bmatrix} \rightarrow C = \begin{bmatrix} 6 & 8 \\ 9 & 10 \end{bmatrix} \quad (1)$$

$$A = \begin{bmatrix} 4 & 2 & 5 & 3 \\ 6 & 2 & 8 & 1 \\ 1 & 9 & 5 & 7 \\ 5 & 6 & 2 & 10 \end{bmatrix} \rightarrow C = \begin{bmatrix} 6 & 8 \\ 9 & 10 \end{bmatrix} \quad (2)$$

Como vemos en la Ecuación 1 y Ecuación 2, las matrices C y C’ son el resultante de aplicar operaciones de valor máximo y promedio a la matriz A 4x4 respectivamente.

1.3.5. Arquitecturas de aprendizaje profundo

Durante la última década los investigadores en aprendizaje profundo han desarrollado algunas arquitecturas de CNNs con resultados y características novedosas. A través del desarrollo de métodos con prueba y error, aprovechan la extracción de características en base al problema inicial que fueron entrenadas para tener un punto de partida al final de la red. Hoy en día, se estas arquitecturas se encuentran listas para su implementación en un vasto conjunto de aplicaciones de visión artificial (Pérez & Gegúndez, Deep Learning, 2021).

A partir de ImageNet la base de datos con millones de imágenes clasificadas han surgido varias arquitecturas, responsables en sí, del crecimiento exponencial del aprendizaje profundo (Berzal, 2018). A continuación, se realiza una mención de las principales arquitecturas.

LeNet-5: Reconocida como la primera CNN para el reconocimiento de dígitos escritos a mano. Está compuesta por capas de convolución y pooling intercaladas, que al final integra una red neuronal tradicional de tres capas (Pérez & Gegúndez, Deep Learning, 2021).

AlexNet: La red pionera del aprendizaje profundo y ganadora de la competencia ImageNet, se caracteriza por optimizar el entrenamiento con funciones de activación

ReLU y uso de las GPUs en paralelo. Formada por cinco capas convolutivas con max pooling, incluía medio millón de neuronas y 60 millones de parámetros (Berzal, 2018).

VGGNet: La idea de su diseño simple en bloques e identificación de profundidad de red permite que las CNNs alcancen una taza de error del 7.33 %. Una arquitectura simple pero marcada por una representación jerárquica de capas completamente conectadas, hace un manejo de kernels 3x3 y pooling 2x2 (Berzal, 2018).

GoogleNet: Con una convolución 1x1 fuerza a la red a utilizar menos parámetros al momento de comprimir la información. El diseño modular de 22 capas de la red consigue mejores rendimientos en memoria y número de parámetros a tan solo 7 millones (Pérez & Gegúndez, Deep Learning, 2021).

ResNet: Establecida como la red que superó al error humano del 5% al clasificar imágenes en la competición de ImageNet, con el uso de unidades ReLU parametrizadas para ajustar mejor la red. Desde este desarrollo se introdujeron conceptos de normalización de entradas escalables, average pooling y conexiones que saltan capas (Berzal, 2018).

1.3.6. Motores gráficos y de juego

En los últimos años las herramientas para la creación de videojuegos han crecido en el mercado, para Fernández (2020) un motor o engine es un software que usa los recursos de gráficos para el renderizado de imágenes 2D y 3D; la distinción entre un motor gráfico y de juego reside en las funciones básica que cada uno desarrolla durante la creación de un videojuego.

Por una parte, el motor gráfico se encarga del renderizado de imágenes en tiempo real mediante subprogramas para obtener gráficos pintados, iluminados y animados en la pantalla. Si hablamos de un motor de juego, provee herramientas de características físicas y colisiones de los modelos 3D o sprites; aparte de representaciones de menús, interfaz gráfica, sonidos y manejo de la inteligencia artificial. Hoy por hoy los softwares dedicados a este mercado incluyen el motor gráfico y de juego en uno solo con el propósito de ser más versátiles hacia el usuario.

Los motores de juegos líderes en el mercado, de acceso para investigación e intuitivos son:

- **Unity Game Engine:** desarrollado por Unity Technologies, permite crear objetos de manera rápida y eficaz. La pantalla de edición está orientado a un trabajo visual para arrastrar y soltar objetos, enlazar scripts, variables; el contar con un entorno scripting la capacidad de construir enlaces de nodos muestra que este potente motor alcanza capacidades de red muy altas. Por último, cabe mencionar su estructura en lenguaje de programación C# (C Sharp) orientada a objetos. (Menard & Bryan, 2015)

- **Unreal Engine:** creada por Epic Games, con un conjunto de herramientas amplio en materiales, reflejos e iluminación basados en renderizado físico en tiempo real. La interfaz de usuario que presenta, combinado con scripts visuales lo convierten en un motor accesible, además de contar con una comunidad creciente de diseñadores a nivel mundial; su gran capacidad se añade al de trabajar sobre un código fuente C++ (Shannon, 2018).

1.3.7. Robot industrial

Por la analogía con el brazo humano Reyes (2011) acepta al robot industrial como un brazo mecánico multiutilidad formado por articulaciones, actuadores, sensores y un sistema de control donde su aplicación está definida por la herramienta final que porta. Con énfasis considera a estos robots por las actividades industriales de repetitividad y en el que presentan ventaja alta sobre las personas.

1.3.8. Brazo robótico Mitsubishi RV-2SDB

Este compacto robot de alto rendimiento de la marca Mitsubishi de la serie SD, destaca por su construcción compacta y agilidad en los cambios de postura; siendo el manipulador ideal para el montaje, la manipulación de materiales, inspección, entre otras tareas (Electric Mitsubishi, 2010).

Con 6 ejes de trabajo es óptimo en alcanzar puntos cercanos al robot, con un rango de $\pm 240^{\circ}$ en el eje de base. Esto significa que es adaptable a células de trabajo pequeñas, las características técnicas del brazo robótico se presentan en la Tabla 2 (Electric Mitsubishi, 2010).

Tabla 2

Especificaciones estándar del Robot Mitsubishi RV-2SDB

Característica		Unidad	Especificaciones
Tipo			RV-2SD/RV-2SDB
Grados de libertad			6
Postura de instalación			Piso, colgado
Sistema controlador			Servo motor AC
Longitud de brazo	Brazo superior	mm	230
	Antebrazo		270
Rango de Operación	Cintura (J1)	Grados	±480
	Hombro (J2)		±240
	Codo (J3)		±160
	Giro de muñeca (J4)		±400
	Paso de muñeca (J5)		±240
	Rotación de muñeca (J6)		±720
Velocidad de movimiento	Cintura (J1)	Grados/s	225
	Hombro (J2)		150
	Codo (J3)		275
	Giro de muñeca (J4)		412
	Paso de muñeca (J5)		450
	Rotación de muñeca (J6)		720
Velocidad resultante máx.		mm/seg	4.400
Carga	Máxima	kg	3
	Índice		2
Repetibilidad		mm	±0.02
Masa		kg	19
Momento admisible de carga	Giro de muñeca (J4)		4.17
	Paso de muñeca (J5)		4.17
	Rotación de muñeca (J6)		2.45
Inercia admisible	Giro de muñeca (J4)		0.18 (0.27)
	Paso de muñeca (J5)		0.18 (0.27)

Característica		Unidad	Especificaciones
	Rotación de muñeca (J6)		0.04 (0.1)
Radio de alcance del brazo		mm	504
Cableado de herramienta			Entrada manual 4 puntos, salida manual 4 puntos
Presión de suministros		MPa	0.5±10%
Especificación de protección			IP30 (En todos los ejes)
Grado de limpieza			-
Color			Gris claro

Nota. Tomado de (Mitsubishi, s.f.)

1.3.9. Cinemática del Robot

Al analizar la arquitectura de un manipulador se entiende la cadena cinemática como una serie de eslabones unidas por articulaciones, en las cuales las coordenadas de un punto final de actuación que describe posición y orientación se pueden traspasar a coordenadas base a partir de una cadena de transformación de coordenadas. De esa forma, se obtiene una descripción de los valores articulares en un sistema coordenado independiente (Barrientos, Peñin, Balaguer, & Aracil, 2005).

a. Cinemática directa

Se refiere al estudio del movimiento del manipulador para determinar tanto la posición como orientación del extremo final del robot, conocidos las generalidades, medidas geométricas de sus articulaciones (Barrientos, Peñin, Balaguer, & Aracil, 2005).

b. Algoritmo de Denavit-Hartenberg

Con el fin de hallar una relación entre el efector final y la base, la convención Denavit-Hartenberg define una útil solución al problema de la cinemática directa; asociando dos eslabones contiguos en orden de la cadena cinemática es posible encontrar las características geométricas singulares del eslabón por medio de 4 transformaciones (Barrientos, Peñin, Balaguer, & Aracil, 2005).

a) l_i longitud del eje z_{i-1} hacia z_i , medida sobre el eje x_{i-1}
b) α_i ángulo entre los ejes z_{i-1} a z_i, medido sobre el eje x_i en sentido horario
c) d_i distancia de x_{i-1} a x_i , medido sobre el eje z_{i-1}
d) θ_i desplazamiento rotacional de x_{i-1} a x_i , medido sobre el eje z_{i-1} en sentido antihorario (Reyes, 2011).

c. Cinemática Inversa

El problema de la cinemática inversa relaciona la configuración articular que debería tomar el robot si las funciones cartesianas en el extremo final son conocidas. El detalle importante es su no linealidad por ende resulta encontrar un método fijo en la solución (Reyes, 2011).

1.3.10. Simuladores virtuales

La simulación computarizada se ha convertido en un proceso esencial para crear sistemas con todas las capacidades de Machine Learning y Deep Learning. Dan soporte a los sistemas de Inteligencia Artificial permitiendo una compresión en el desarrollo de

la cadena neuronal, tanto en el rendimiento como la puesta a prueba y comparar diferentes estrategias. Los procesos en plantas virtualizadas han de determinar los parámetros al ser evaluados en su comportamiento dinámico. Conocida como un gemelo digital, se pueden generar y analizar datos para lograr una interacción mutua con un modelo real, usando un modelo computador-asistente apoya a la construcción de sistemas con métricas de evaluación y resultados no solamente en instalaciones intralogísticas sino con una funcionalidad real. Esta opción existente representa una idea madura, latente y reconocida en la ciencia computacional para entrenar modelos de inteligencia artificial. (SSISchaeferES[Video de Youtube], 2019)

Capítulo II

2. Diseño y selección de componentes

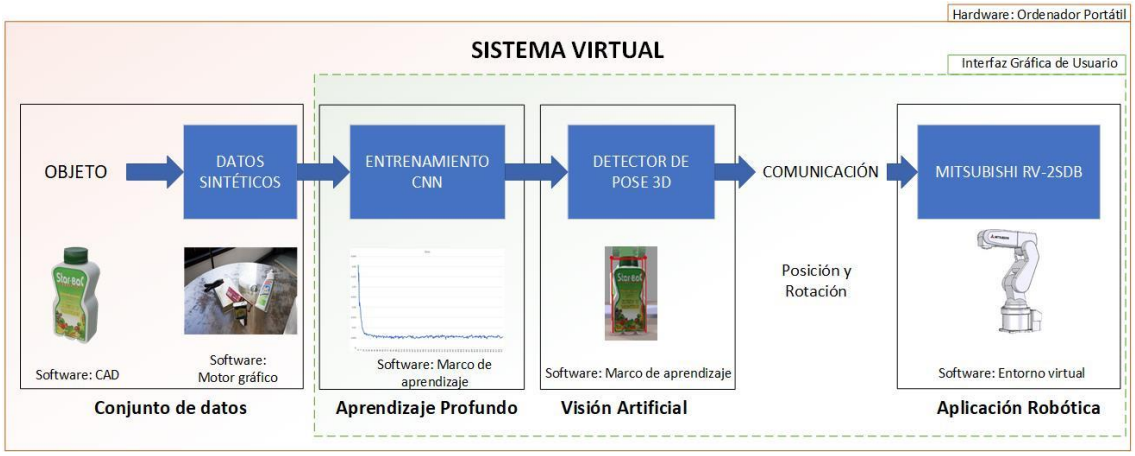
En el presente capítulo se detalla la arquitectura del sistema virtual en su componente de hardware, utilizado para el entrenamiento e inferencia del modelo de CNN. Para lo mencionado, se realiza la evaluación y selección de software en cada una de las etapas de conjunto de datos, aprendizaje profundo, visión artificial, comunicación y aplicación robótica; se consideran varios criterios de selección que puedan influir en la eficiencia computacional y técnica al momento de detectar la pose 3D del objeto.

2.1. Arquitectura del sistema virtual

En primera vista las etapas del sistema virtual inician con el modelado del objeto doméstico hasta llevar a cabo el movimiento en el efector final del manipulador. Este esquema se detalla en la Figura 9 donde se considera un objeto modelado en un software de modelado para posteriormente introducirlo en el generador de datos sintéticos con el fin de generar un conjunto de datos lo suficientemente grande para entrenar el modelo de red neuronal profunda. De modo que, extraídas las características del objeto se consiga la posición y orientación dentro de la escena, esta información se combina con el algoritmo de control que es llevado al entorno de simulación virtual donde el manipulador Mitsubishi RV-2SDB ejecute la aplicación robótica. Se incluye una interfaz gráfica de usuario (GUI) para proveer un entorno visual y de control abierto al aprendizaje en las últimas etapas que compone la aplicación robótica del sistema virtual.

Figura 9

Etapas del sistema virtual



2.1.1. Hardware

Para la implementación de este trabajo se dispone de un equipo portátil MSI con una unidad de procesamiento gráfico (GPU). Los marcos de aprendizaje profundo por GPU pueden acelerar significativamente el entrenamiento que llevarían semanas o días a tan solo días u horas (NVIDIA Developer, 2015).

Opera en los procesos de entrenamiento e inferencia de la CNN, los detalles de este ordenador se presentan en la Tabla 3.

Tabla 3

Especificaciones GP63 Leopard

Especificación	Detalle
Fabricante	Micro-Star International Co., Ltd.
Modelo	GP63 Leopard 8RF
Procesador	Intel(R) Core (TM) i7-8750H CPU @ 2.20GHz (12 CPUs), ~2.2GHz
Memoria	16GB (8G*2)

Especificación	Detalle
	DDR4 2400MHz
GPU	NVIDIA GeForce® GTX1070
Memoria de video	8G GDDR5
Webcam	720p HD Webcam
BIOS	E16P5IMS.10E (type: UEFI)

Nota. Tomado de GP63 Leopard (Intel 8th Gen), por Micro-STAR International Co., Ltd., s.f.

2.1.2. Software

En el ámbito de programas de desarrollo se considera aplicativos más recientes y con un grado de abstracción del hardware propuesto con el objetivo de acelerar el aprendizaje profundo.

2.2. Selección de componentes del sistema

Para definir los componentes óptimos orientados a la selección de software se busca metodologías que midan la información recopilada en base a criterios del evaluador, así pues, se utilizará el método de los factores ponderados que vincula tanto a consideraciones cualitativas o cuantitativas. Permite la comparación de dos o más elementos que sobre una suma de calificaciones ponderadas da como alternativa óptima la de mayor puntaje. Con una escala de importancia de cero a diez, de menor a mayor respectivamente (Córdoba, 2011).

2.2.1. Selección de software de modelado de objetos 3D

Con la finalidad de tener un objeto doméstico representativo del mundo real en escena, estos programas de modelado se direccionan a propósitos específicos en la

industria. Beane (2012) menciona que programas de animación 3D como Maya,Zbrush, 3ds Max, Blender permiten crear modelos 3D de objetos completos utilizando técnicas de sólidos paramétricos al dibujar curvas, contornos por el usuario y ser rellenadas las superficies por el software. Las componentes a destacar para que los modelos finales sean llevados a escena son: la texturización, rigging y la animación.

Al ser la primera parte de la arquitectura el programa necesita contar con un componente de texturización (relacionado al color y propiedades de superficie), complementos de exportación (para una interoperabilidad entre aplicaciones 3D), intuitivo al usuario en aspectos de interfaz gráfica, requisitos computacionales, curva de aprendizaje, contrato de licencia y soporte como criterios principales. A continuación, se muestran las especificaciones de los programas propuestos:

- **Blender 3D:** es una herramienta con diferentes técnicas de diseño que permite crear, editar, transformar y diseñar objetos y/o escenas en un campo tridimensional. La gran comunidad de usuarios que participan en su desarrollo extiende su libertad de uso y distribución. Aplicada en la creación de animaciones y películas de gran calidad por su técnica de fotogramas clave ha captado la atención en varios sectores durante los últimos años (Suau, 2011). Con su técnica de mapas UV permite una organización de textura 2D en mallas 3D y conseguir cálculos de iluminación bien definidos.

  Los requisitos computacionales del software Blender se presenta en la Tabla 4.

Tabla 4

Requisitos de hardware mínimo para la instalación de Blender

Característica	Detalle
Sistema operativo	Windows 8.1 y 10, Linux, MacOS 10.13
Tipo de CPU	64-bit dual core 2Ghz CPU
Memoria	4 GB RAM
Gráficos	1 GB RAM/GeForce 400 en adelante. Recomendado Soporte OpenGL 3.3
Pantalla	1280×768 px.
Licencia	Libre uso, copia y distribución

Nota. Tomado de (Blender, s.f.)

Los complementos disponibles del software se detallan en la Tabla 5.

Tabla 5

Complementos del programa Blender

Tipo	Detalle
Formato de importación/exportación	BVH, PLY, STL, FBX, glTF, OBJ, X3D
Renderizado	Freestyle SVG

Nota. Tomado de (Blender, s.f.)

- **ZBrush:** es una herramienta de modelado 3D con múltiples herramientas que durante estos años ha tomado gran relevancia en estudios de cine, desarrollo de videojuegos, juguetes e ilustraciones, considerándose un estándar en el arte digital. Permite la incorporación de programas de terceros para las herramientas de renderizado y materiales que incluyen: PolyPaint, Maestro UV, Texturizado, etc. (Pixologic, Inc, s.f.)

  Los requisitos de hardware del software ZBrush se presenta en la Tabla 6.

Tabla 6

Requisitos de hardware mínimo para la instalación de ZBrush

Característica	Detalle
Sistema operativo	Windows 8.1 y 10, MacOS 10.13
Tipo de CPU	64-bit Core2duo
Memoria	4 GB (6 GB rendimiento fuerte) RAM
Gráficos	Tarjetas ensambladas desde el 2008 en adelante/ Soporte OpenGL 3.3
Pantalla	1280×1024 a 32 bit de color
Licencia	$895 por 1 año

Nota. Tomado de (Pixologic, Inc, s.f.)

La capacidad de ZBrush al esculpir millones de polígonos le permite crear renders realistas de un concepto en 2D llevado a 3D por sus herramientas de iluminación y efectos de escena. Los complementos disponibles en el software se detallan en la Tabla 7.

Tabla 7

Complementos del programa ZBrush

Tipo	Detalle
Formato de importación/exportación	STL, OBJ, VRML, FBX, ZTL, to Maya, to Photoshop CC
Renderizado	Best Preview Renderer (BPR)

Nota. Tomado de (Pixologic, Inc., s.f.)

- **Fusion 360:** es un producto de Autodesk Inc., integra varias herramientas robustas e intuitivas. Combina los softwares de CAD/CAM/CAE en una sola plataforma que permite el diseño mecánico basado en características utilizando herramientas de modelado 3D sencillas, pero altamente efectivas. Enfocado a las áreas de ingeniería, arquitectura, manufactura e industrias de entretenimiento (Dogra, 2020)**.**

Los requisitos de hardware del software Fusion 360 se presenta en la Tabla 8.

Tabla 8

Requisitos de hardware mínimo para la instalación de Fusion 360

Característica	Detalle
Sistema operativo	Windows 8.1 y 10, MacOS 11
Tipo de CPU	32 y 64-bit 1.7 GHz o más
Memoria	4 GB (6 GB recomendado) RAM
Gráficos	GPU 1GB/ Gráficos integrados de 6GB
Pantalla	1366×768 px.
Licencia	De pago/ Gratis tipo estudiantil por 1 año

Nota. Tomado de (Autodesk Inc., s.f.)

La tienda oficial de Autodesk implementa soporte web para la adquisición de extensiones complementarias a Fusion 360, varios de los módulos son gratis pero un grupo mínimo requiere una suscripción de pago.

Por consiguiente, una vez analizado los programas de diseño y modelado 3D la Tabla 9 muestra la evaluación realizada con la metodología de factores ponderados.

Tabla 9

Matriz de evaluación de software de modelado

Factores	Peso (%)	Alternativas		
		Blender 3D	**ZBrush**	**Fusion 360**
Interfaz gráfica	20	7	7	10
Complementos	20	8	7	6
Requisitos computacionales	10	10	9	7
Curva de aprendizaje	20	6	7	8
Contrato de licencia	10	10	2	5
Soporte	20	9	8	6
Puntuación total		8	6.9	7.2

En base al contenido de la Tabla 9, Blender marca una diferencia en la matriz de ponderaciones entre ZBrush y Fusion 360 por la interoperabilidad entre programas 3D en su formato *.fbx, de código abierto y una gran comunidad de soporte en la red. Además, Rodriguez(2019) recomienda en su trabajo de detección de pose el uso del software Blender para realizar una corrección de la topología de mallas, sombreado y suavizado al trabajar con modelos 3D previo a la generación de datos.

2.2.2. Selección de software para la generación de datos sintéticos

Un factor importante concurrente en el rendimiento del modelo de red neuronal son la cantidad y calidad del conjunto de datos de entrenamiento (Voulodimos, Nikolaos, & Protopapadakis, 2018). En tal sentido, (Nikolenko, 2019) destaca los parámetros de un generador de datos sintéticos (SDG) por niveles:

En el nivel de construcción de escena, el SDG pueda aleatorizar el número de objetos, sus posiciones relativas y absolutas, número y forma de los objetos distractores, fondo de la escena, las texturas de los objetos que participan en la escena, etc.

A nivel de renderización, el SDG pueda aleatorizar las condiciones de iluminación, la intensidad, posición y orientación de las fuentes de luz, cambiar la calidad del renderizado en base a la resolución de imagen, tipo de renderizado, ruido aleatorio a las imágenes resultantes, etc.

A continuación, se presenta los motores gráficos con una descripción de sus herramientas a evaluar para la generación de datos sintéticos, complementos,

herramientas de posprocesamiento, requisitos computacionales, contrato de licencia y soporte:

- **Unity 3D:** es la herramienta usada por muchas organizaciones para adquirir conjuntos de datos en el entrenamiento de modelos de aprendizaje automático, utiliza efectos de posprocesamiento con el fin de simular las propiedades físicas de la cámara y la escena. El complemento destinado a este fin es Unity Computer Vision, cuenta con tres tipos de canalización de renderizado: incorporada, ligero y de alta definición (Unity Technologies, 2018), estos efectos presentados en la Tabla 10 obtienen una calidad alta de película.

Tabla 10

Descripción general de efectos de posprocesamiento en Unity

Efecto	Detalle
Anti-aliasing	Convierte los gráficos en una apariencia suave
Oclusión Ambiental	Oscurece los pliegues, los agujeros, las intersecciones y las superficies cercanas entre sí
Bloom	Luz extremadamente brillante en la cámara
Aberración Cromática	Franjas en los límites que separan las partes claras y oscuras
Gradiantes de Color	Altera o corrige el color y la luminancia de la imagen
Niebla diferida	Superpone un color a los objetos en relación con la distancia focal de la cámara
Profundidad de campo	Simula las propiedades de enfoque de la cámara
Auto-Exposición	Ajuste dinámico del lente en diferentes niveles de oscuridad
Grano	Produce pequeñas partículas en la película de la cámara, efecto de imagen gruesa
Desenfoque de movimiento	Difuminación de la imagen cuando se mueve rápida la cámara o tiempo de exposición prolongado
Reflexión del espacio de pantalla	Crea reflejos de superficies mojadas de suelo o charcos.
Viñeta	Oscurece bordes con un centro más brillante

Nota. Basado en (Unity Technologies, 2018)

Los requerimientos de hardware y complementos enfocados a datos sintéticos se detallan en la Tabla 11.

Tabla 11

Requisitos de hardware mínimo y productos de Unity 3D

Característica	Detalle
Sistema operativo	Windows 7, 8.1 y 10, Ubuntu, MacOS 11
Tipo de CPU	32 y 64-bit soporte SSE2
Memoria	8 GB RAM
Gráficos	Soporte DX10 y tarjetas gráficas
Licencia	Pago/Gratuita no comerciable
Complementos	Unity Computer Vision (Producto de pago)
Lenguaje de programación	C#

Nota. Tomado de (Unity Technologies, 2018)

- **Unreal Engine 4:** el motor gráfico con un sistema de renderizado líder en la industria para diseñar experiencias en tiempo real proporciona herramientas que permiten a los usuarios y diseñadores cambios en la escena. Con un gran editor de materiales, iluminación y sombras es categorizado como uno de los motores de juegos más populares en el mercado (Epic Games, Inc., s.f.)

Desde Unreal Engine 4.15 los posprocesos basados en física, volumen y efectos han denotado un hito importante en los estándares establecidos por la ACES (Academy Color Encoding System), la Tabla 12 expone algunos de estos efectos de posprocesamiento.

Tabla 12

Efectos de posprocesamiento en Unreal Engine 4

Efecto	Detalle
Anti-Aliasing	Suavizado de líneas
Exposición automática	Ajuste automático a la exposición de cambios de brillos
Bloom	Efectos de brillo de alta intensidad
Vendibles	Activos capaces de interpolar en el renderizado
Graduación de color y mapeado de tonos fílmicos	Mapeo de tonos y asignación de colores
Destello de lente	Simula la dispersión de la luz al ver objetos brillantes
Profundidad de campo	Control del enfoque en función de la profundidad
Aberración cromática	Simula los cambios de color en los bordes
Materiales de posproceso	Crear y combinar configuraciones pasadas de postproceso personalizadas
Proyección Panini	Proyección 3D que corrige la distorsión geométrica de la proyección en perspectiva
Viñeta	Oscurecimiento de la lente de las cámaras, en específico los bordes
Tablas de búsqueda (LUT) para clasificación de color	Emplea corrección de color usando un volumen de posproceso

Nota. Basado en (Epic Games, Inc., s.f.)

Los requerimientos de hardware y complementos enfocados a datos sintéticos se detallan en la Tabla 13.

Tabla 13

Requisitos de hardware mínimo y productos de Unreal Engine 4

Característica	Detalle
Sistema operativo	Windows 7, 8.1 y 10, Ubuntu
Tipo de CPU	32 y 64-bit Intel o AMD 4 núcleos 2.5GHz
Memoria	8 GB RAM
Gráficos	Soporte DX11 y tarjetas gráficas
Licencia	Gratuita
Complementos	NDDS (Complemento NVIDIA gratuito)
Lenguaje de programación	C++

Nota. Tomado de (Epic Games, Inc., s.f.)

Por consiguiente, una vez analizado los programas y sus herramientas para generar datos sintéticos, la Tabla 14 muestra la evaluación realizada con la metodología de factores ponderados.

Tabla 14

Matriz de evaluación de software de generación de datos sintéticos

Factores	Peso (%)	Alternativas	
		Unity 3D	**Unreal Engine 4**
Complementos	30	6	8
Herramientas de posprocesamiento	20	10	10
Requisitos computacionales	10	9	8
Curva de aprendizaje	15	9	8
Contrato de licencia	10	8	8
Soporte	15	9	9
Puntuación total		8.2	8.4

La diferencia entre los dos programas es mínima según la matriz de evaluación de la vista de la Tabla 14, sin embargo, Unreal Engine 4 es la opción elegida por varios investigadores para recolectar y crear anotaciones de entrenamiento de aprendizaje profundo, por mencionar a (Vallez, Velasco-Mata, Corroto, & Deniz, 2019) que presenta un escenario sintético del pasillo de un colegio a fin de detectar a las personas que portan armas en sus manos; (Delgado, 2019) utiliza el complemento de UnrealCV llamado Robotrix con el propósito de ubicar los datos LINEMOD en escenas interiores hiperrealistas exploradas por un robot. Por estas razones, se trabaja con el programa Unreal Engine como generador de datos sintéticos.

2.2.3. Selección de marco de aprendizaje (framework) de la red neuronal

En los inicios del aprendizaje profundo se requería un vasto conocimiento en lenguajes de programación C++ y CUDA en GPU para codificar algoritmos. Los

frameworks más dinámicos por arriba del gran número existentes son Tensorflow, Keras y PyTorch, con extensas contribuciones y proyectos en GitHub (Torres, Deep Learning Introducción Práctica con Keras, 2018).

Las características que permitieron la selección del marco de aprendizaje de acuerdo con la arquitectura del sistema virtual son: la interfaz de programación de aplicaciones(API), velocidad, depuración, arquitectura, tamaño de conjunto de datos y soporte.

- **Tensorflow**

Un sistema de aprendizaje automático a gran escala que usa flujo de datos gráficos simbólica para representar todos los cálculos y estados de los algoritmos. Múltiples niveles de ejecución donde se dan paso para compartir datos y sincronizarlos con un enfoque en el entrenamiento e inferencia de redes neuronales profundas con compatibilidad a GPU. Presenta un alto rendimiento en la industria en aplicaciones de reconocimiento de voz, traducción, etiquetado de video, detección de objetos y otras tareas; destaca entre su comunidad por la gran cantidad de usuarios, a pesar de eso el hecho de ofrecer un gran rendimiento la API no es suficientemente explícita (Abadi, y otros, 2016).

- **Keras**

La API construida sobre TensorFlow 2.0 es a grandes escalas el marco de aprendizaje fuerte en la experimentación, permitiendo probar ideas con una infraestructura pequeña y obtener resultados rápidos. Con soporte de GPU en investigación ofrece una conveniencia de alto nivel para acelerar ciclos de

experimentación. (Keras, s.f.). Es usado para pequeños conjuntos de datos con excelentes tutoriales y códigos accesibles brindados por la comunidad.

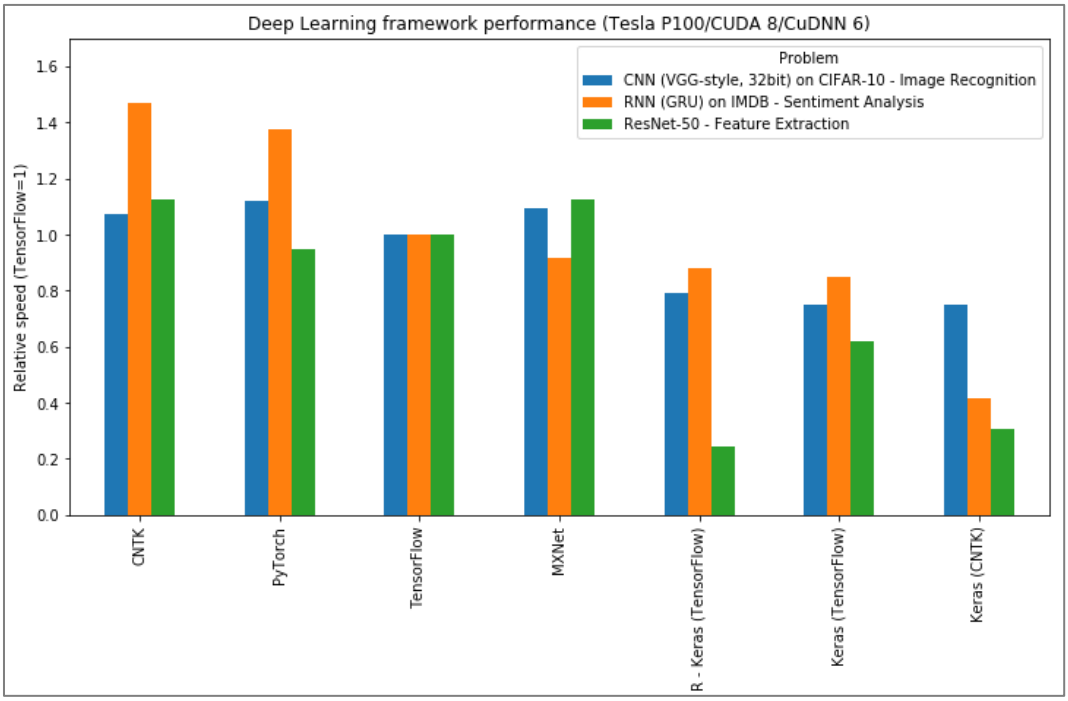
- **PyTorch**

Es una biblioteca del ecosistema de Python que proporciona un modelo de programación basado en arrays, acelerado por GPU e integrada a través de la diferenciación automática. Con la llegada del software libre fomentó a un gran número de personas a cambiar, ampliar y publicar nuevas funcionalidades de esta biblioteca. El hecho de trabajar sobre Python integra de forma natural herramientas de trazado, depuración y procesamiento de datos. Sin embargo, su implementación provee una arquitectura compleja para ofrecer un alto rendimiento al procesar grandes conjuntos de datos (Paszke, y otros, 2019).

Karmanov, Salvaris, Fierro, & Danielle (2018) realiza una evaluación de los marcos de aprendizaje profundo al comparar los tiempos de entrenamiento de modelos de CNN, RNN y ResNet 50 presentado en la Figura 10. De modo que señalan una eficiencia computacional de mayor rendimiento en las API PyTorch y Tensorflow sobre Keras.

Figura 10

Rendimiento de marcos de aprendizaje profundo



Nota. Basado en los datos de (Karmanov, Salvaris, Fierro, & Danielle, 2018)

A continuación, una vez analizado los marcos de aprendizaje profundo en la Tabla 15 muestra la evaluación realizada con la metodología de factores ponderados.

Tabla 15

Matriz de evaluación de marcos de aprendizaje

Factores	Peso (%)	Alternativas		
		Tensorflow	**Keras**	**PyTorch**
Nivel de API	10	8	10	8
Velocidad	20	9	8	10
Arquitectura	10	7	9	7
Depuración	20	6	9	9
Conjunto de datos	20	10	5	10
Soporte	20	9	8	10
Puntuación total		8.3	7.9	9.3

En base a los resultados de la matriz de evaluación de la Tabla 15, el marco de aprendizaje profundo PyTorch califica con mayor puntaje por su velocidad, integración con las bibliotecas de Python. Además, se considera el trabajo de conducción autónoma en la detección de conos de Dhall, Dai, & Van Gool, (2019) donde menciona que al trabajar con capas convolucionales los tensores que se manejan en PyTorch contienen niveles más altos de información que en nivel local, lo que significa una reducción significativa en el espacio del modelo; Delgado (2019) indica una superioridad de los tiempos de ejecución al cambio en las funciones del gradiante que ofrece Pytorch por su operación de grafos dinámicos. Por consiguiente, el framework Pytorch con soporte CUDA Nvidia es el seleccionado para diseñar el detector de pose 3D.

2.2.4. Selección de software de simulación de entornos virtuales 3D

El desarrollo de la tecnología gráfica 3D ha alcanzado una incomparable calidad por ellos los entornos virtuales simulan un mundo real de inmersión o con interacción de hardware de escritorio. Con el fin de permitir una interacción del usuario con los elementos del entorno virtual respetando restricciones físicas partículas del programa de diseño (Rincon, 2014).

Por lo tanto, se requiere un software de entorno virtual 3D que proporcione herramientas de inserción de objetos modelados, APIs de programación, simulación física y creación de interfaz de usuario que alimenten el aprendizaje al desarrollar la aplicación de pick and place del robot Mitsubishi RV-2SBD.

Los programas propuestos a evaluación son CoppeliaSim, Webots, Unity 3D y Unreal Engine 4, estos dos últimos detallados en la sección 2.2.2.

- **CoppeliaSim:** Es un software de simulación robótica de código abierto con un gran número de funciones y características en simulador de robots. Se destaca por su implementación de algoritmos con desarrollo rápido y que puede utilizarse en la validación de prototipos, enseñanza robótica, supervisión remota, seguridad de productos, entre otras. Yang, Zeng, & Jianwei (2021) describe al software por sus características de:
    - Disponibilidad para usuarios de Windows, Linus y Mac OS.
    - Utilizar entornos integrados para una arquitectura de control distribuido. Cada modelo u objetos es independiente por su script asociado, maneja complementos y módulos adicionales externos como es el caso de los nodos de ROS.
    - Escritura los lenguajes de C,C++,Lua, Python, Matlab, Java o Urbi.
    - Tener motores físicos y módulos computacionales.
    - Simulación de sensores de visión y de proximidad

Los requisitos computaciones que demanda el software no son detallados en su página de soporte.

- **Webots:** la aplicación multiplataforma de código abierto para realizar simulaciones robóticas. De uso profesional en la industria, investigación y educación es catalogado como una gran biblioteca de componentes robóticos (sensores, actuadores) para desarrollar, experimentar y validar algoritmos de IA (Cyberbotics Ltd., s.f.). Las características de este software son:
    - Disponibilidad para Windows, Linux y Mac OS.
    - Sus núcleos se basan en motores de física y motor de renderizado OpenGL 3.3.

- Programación en lenguajes C, C++, Python, Matlab, Java o nodos de ROS.
- De desarrollo continuo y aplicación en proyectos de simulación interactiva, robótica modular, automóviles, creación de entornos, entre otras.

Por lo tanto, una vez analizado los programas de desarrollo de entornos virtuales la Tabla 16 muestra la evaluación realizada con la metodología de factores ponderados.

Tabla 16

Matriz de evaluación de software para entornos virtuales

Factores	Peso (%)	Alternativas			
		CoppeliaSim	**Webots**	**Unity 3D**	**Unreal Engine 4**
Requisitos computacionales	10	10	7	7	7
Motor físico y renderizado	20	10	9	10	10
Herramientas de Posprocesamiento	20	7	8	10	10
Herramientas para creación de GUI	20	8	9	10	9
Contrato de licencia	10	10	8	8	8
Comunicación	10	10	8	7	7
Soporte	10	9	7	9	8
Puntuación total		8.9	8.2	9.1	8.8

En base a los resultados de la matriz de evaluación de la Tabla 16, el software óptimo para entornos virtuales es Unity 3D, dado que este brinda una extensa gama de herramientas en renderizado y posprocesamiento. No obstante, su desarrollo de algoritmos en “C Sharp o C#” dificultan la conexión de datos sobre el framework Pytorch basada en Python; además, de tener un alto consumo en sus procesos afecta directamente a la inferencia del detector de pose 3D.

Por tal razón, se selecciona el software CoppeliaSim Edu por su arquitectura de control distribuido en lenguajes de programación y su versatilidad en educación enfocada a la robótica. Este software ha sido de interés en proyectos nacionales tal como lo explica Gómez & Soria (2020) en la automatización de sellado y lacado para empresas carpinteras internacionales.

2.2.5. Selección de la aplicación robótica y efector final

En base a la disposición del robot articulado Mitsubishi RV-2SDB como módulo de práctica y aprendizaje técnico de procesos de automatización industrial, se busca dar un enfoque a la problemática de la pose 3D de objetos en el agarre automático basado en visión artificial. La tarea de *pick and place* descrita por la acción de seleccionar y colocar en otro lugar un objeto, describe una acción infrecuente a cambios, de ciclos repetitivos donde la posición y orientación del objeto están establecidas en el espacio de trabajo; por ende, se selecciona esta aplicación para integrar un grado de autonomía al proceso industrial y de adquirir las ventajas de un robot colaborativo.

Lien (2013) detalla que el actuador final define la operación de destino del manipulador, en el caso de la aplicación seleccionada los tipos de *grippers* o pinzas están vinculadas a la forma, textura y condiciones físicas para transferir la fuerza necesaria desde el brazo hasta el objeto para moverlo.

El objeto de interés para la detección de pose que se ha seleccionado es un envase de plástico comercial como se muestra en la Figura 11, por la forma ergonómica, texturas plásticas con rayaduras de manufactura, no transparente, con detalles de soportes y etiquetas de más de diez colores inmersos permite el enfoque de

aprendizaje profundo en encontrar la diferencia de un objeto en una mayor cantidad de carácterísticas para extraer.

Figura 11

Envase plástico

Nota. Las medidas proporcionadas se detallan en milímetros. El peso del envase es de 0.2 Kg.

Al conocer el objeto se abren las opciones en la selección del gripper para el robot Mitsubishi, en donde destacan los de presión, flexibles, penetración que reciben su potencia de la energía eléctrica, neumática e hidráulica. Se selecciona el gripper de presión Baxter eléctrico de pinzas paralelas por brindar un agarre más robusto y eficiente en objetos sólidos rectangulares, característica mencionada por Lien (2013). La Figura 13 muestra este componente en su versión de dedos cortos.

Figura 12

Gripper Baxter

Las especificaciones del producto están descritas en la Tabla 17, cabe mencionar que el gripper forma parte de la biblioteca de objetos para el desarrollo de aplicaciones en CoppeliaSim.

Tabla 17

Características Baxter Electric Gripper

Componente	Peso	Longitud
Base del gripper sin dedos	0.3 Kg	43 mm
Dedo corto	0.02 Kg	73 mm
Datos específicos de la pinza		
Carrera	n/a	44 mm
Carrera por mordaza	n/a	44 mm

Nota. Obtenido de (Rethink Robotics, 2015)

Cabe destacar que esta selección es flexible por la gran variedad de componentes que maneja CoppeliaSim en su biblioteca, abriendo un marco de exploración en los estudiantes a probar las herramientas tipo ventosas, garras de sujeción, pinzas angulares y adecuar los valores físicos como geométricos de los mismos en nuevos objetos.

2.2.6. Selección de sistema operativo base

Para la selección del sistema operativo se realizó una investigación cualitativa basado en la teoría fundamentada, Gallego, Icart, & Anna (2006) aclara en su guía que la literatura existente es un componente poderoso que se añade al estudio sin perder el criterio del investigador con el fin de contrastar ideas y resultados.

Likhith (2019) comparte su enfoque y experiencia al trabajar con blibliotecas de aprendizaje profundo, por lo que centra sus datos en sistemas operativos como Windows y Linux. Las razones principales para seleccionar Linux son: el soporte de la comunidad, seguridad, símbolo del sistema, rendimiento, compatibilidad con GPU, facilidad de uso, modularidad, contrato de licencia gratis y de código abierto. Por lo que, Linux presenta la opción adecuada para este tipo de aplicaciones en su versión de Ubuntu.

(Parmar, 2020) presenta el análisis al comparar el tiempo de entrenamiento de tarda un modelo clasificador CNN con 2400 imágenes sobre las arquitecturas de Windows 10 y Ubuntu. El tiempo de entrenamiento es 18.9 % más rápido en Ubuntu por lo que ahorraría un poco de tiempo, sin embargo, Windows comparte las mismas ventajas y se acomoda a su estilo de trabajo.

En base a la metodología aplicada se trabaja en Linux con la distribución de Ubuntu 20.04 por la mejor distribución de sus aplicaciones, instalación de paquetes por terminal que se reflejan en el rendimiento al trabajar con modelos de aprendizaje profundo; además, los programas como Blender, Unreal Engine 4 y CoppeliaSim son compatibles con el sistema operativo seleccionado.

En resumen, en este capítulo se ha realizado el diseño de la arquitectura del sistema virtual para identificar los componentes de software en cada etapa de conjunto de datos, aprendizaje profundo, visión artificial y aplicación robótica para el hardware disponible. Con el propósito de seleccionar de varias componentes disponibles en la web se ha puesto a criterio del evaluador una matriz de importancia de los factores determinantes en cada aplicación, ahora bien, la Tabla 18 agrupa las selecciones finales para el desarrollo del sistema virtual de detección de pose 3D.

Tabla 18

Resumen de softwares seleccionados

Etapa	**Software**
Modelado 3D de objeto doméstico	Blender 2.93.0
Generador de datos sintéticos	Unreal Engine 4.22
Framework de la red neuronal	PyTorch 11.1
Aplicación robótica	*Pick and place* (Recoger y colocar)
Efector final	Gripper Eléctrico
Simulador de entorno virtual	CoppeliaSim Edu v4.2.0

Capítulo III

3. Desarrollo del sistema virtual

En el presente capítulo se realiza el proceso de preparación de escena del sistema virtual, por lo que se parte con el modelado del objeto 3D para llevarlo a escenas diseñadas de fotorrealismo y aleatorización de dominios para el generador sintéticos de imágenes; finalmente la creación del entorno virtual para la aplicación robótica destinada.

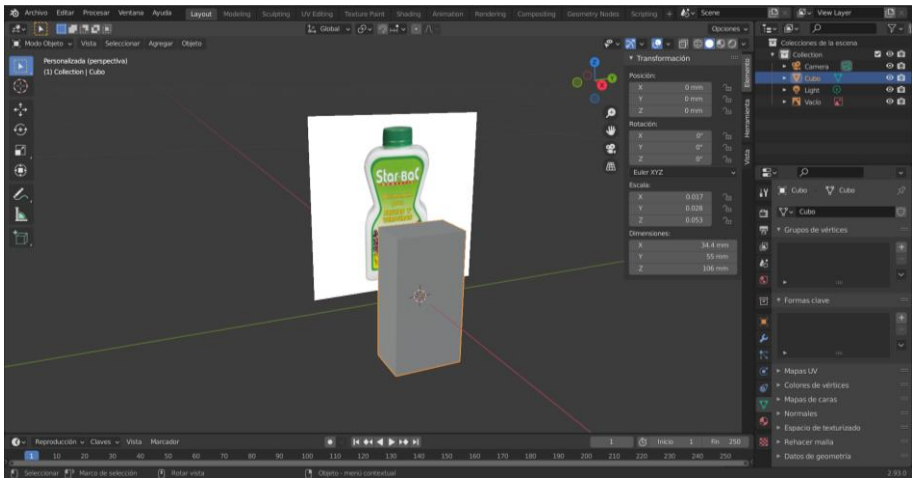
3.1. Desarrollo del modelado del objeto 3D

El modelado del objeto doméstico definido por un envase plástico es la parte inicial para el desarrollo del conjunto de datos sintéticos y de la aplicación robótica en el entorno virtual. Una de las consideraciones importantes es el replicar con la mayor cantidad de detalles en geometría, texturas y tamaño del objeto; es por eso, que en Blender v2.9 el sistema de modelado es basado en polígonos con el fin de capturar cada detalle. Blender brinda la capacidad de trabajar con millones de polígonos en base a costo computacional en el renderizado.

La técnica de modelado de caja con una imagen de referencia en escena como se muestra en la Figura 13, permite añadir geometrías a medida en forma de vértices, bordes y caras con la guía en perspectiva de la imagen del objeto real.

Figura 13

Box modeling

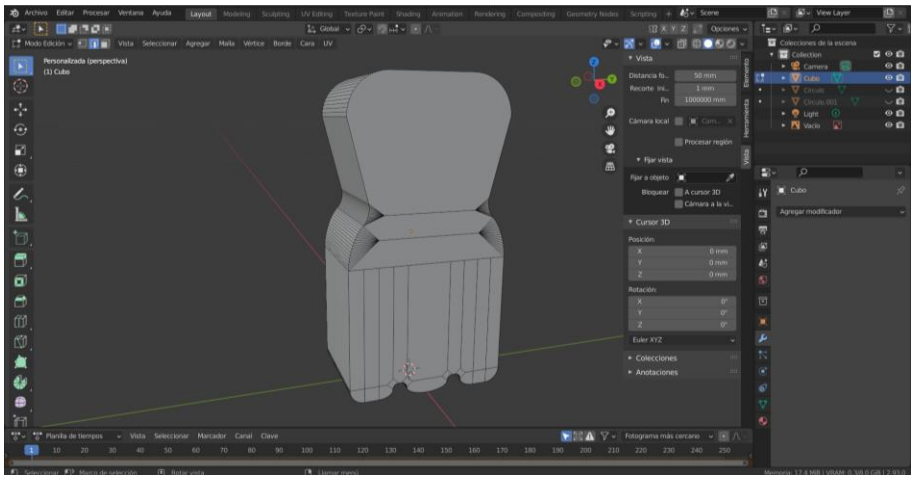


Tanto el cuerpo del envase, anillo, tapa y soportes han sido modelados por separado a pesar de que forman un mismo objeto, con el fin de hacer uso de mapas UV en la asignación de texturas, modificadores y nodos de materiales.

Las herramientas de escalado, extrusión y biselado permiten definir el cuerpo del envase como se aprecia en la Figura 14. Adicional a la guía de la imagen de referencia, el uso del calibrador de Vernier complementa la toma de medidas con una escala de 0.1mm.

Figura 14

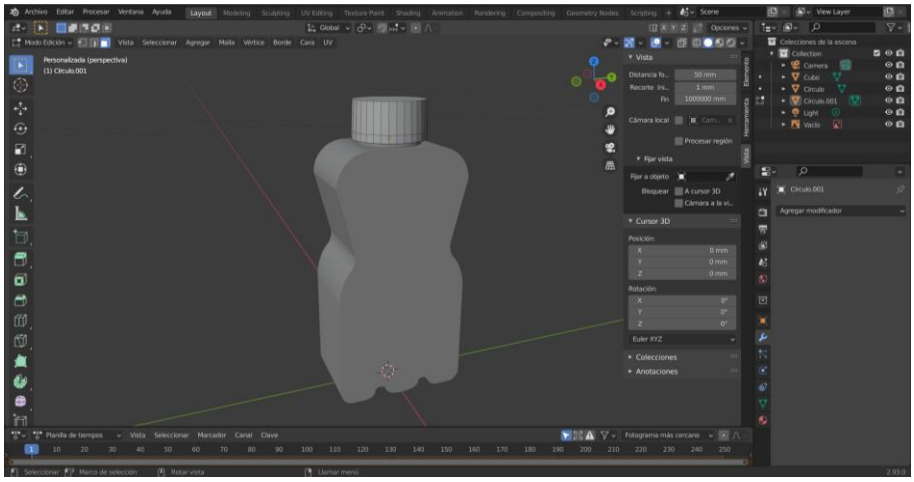
Modelado del cuerpo del envase



El anillo de seguridad parte de un círculo extruído que se une a la tapa de diámetro mayor para formar el esqueleto principal del envase detallado enla Figura 15.

Figura 15

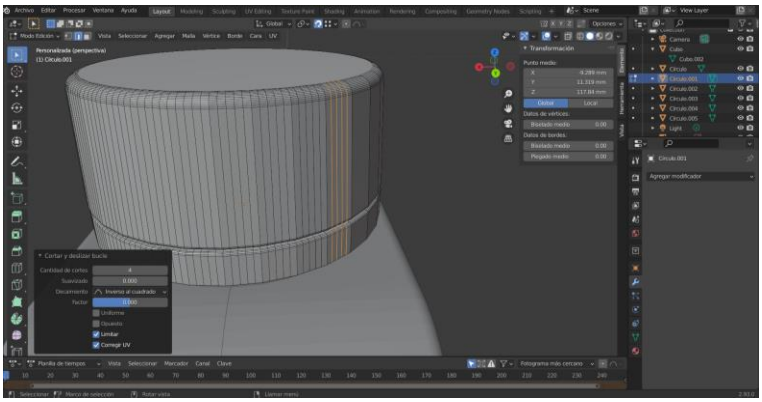
Modelado completo del envase



Los detalles de redondeo son las principales muestras de realismo, en primera instancia los bordes y texturas al tacto en la tapa están realizados con la herramienta de biselado como se observa en la Figura 16.

Figura 16

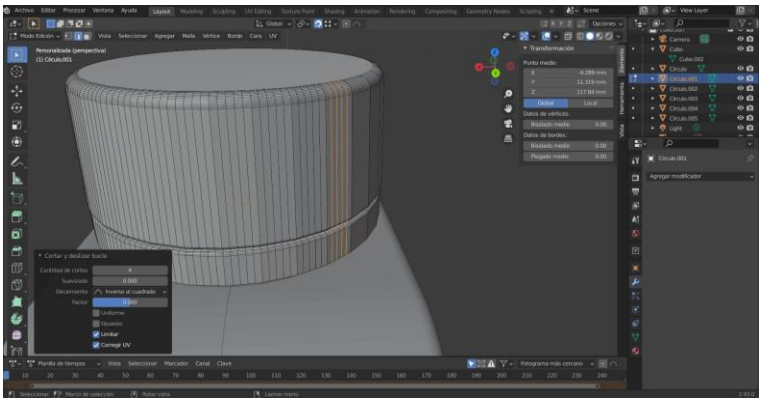
Biselado de bordes



El objetivo de este modelado es brindar realismo para la puesta en escena, mediante los nodos de composición de Blender se aplican texturas plásticas con rayones, además de brindar un color blanquecino y reflección de malla 0.5 de su valor. La Figura 17 muestra la estructura del nodo aplicado en cada parte del envase plástico.

Figura 17

Nodo de composición

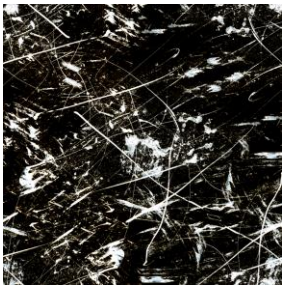


Nota. La rugosidad del material está controlada por el nodo de textura por imagen seguido de rampas de color para controlar la intensidad del relieve. Los valores son modificados en base a la parte del objeto modelado.

La Figura 18 muestra la textura rayada y de orificios aplicada sobre el plástico para conseguir el aspecto realista a cada una de las partes. Esta imagen de alta calidad en formato .tif se encuentra de libre acceso en el sitio web de desarrolladores de Blender (Aura Prods, s.f.).

Figura 18

Textura tipo pintura



Las etiquetas externas del envase están agregadas con el uso de los modificadores en la pestaña de Diseño, se aplica un modificador de Suavizado de Superficie para tener una malla poligonal de superficie curva de la imagen importada y en conjunto con el modificador de Envolver permitir a esta proyectarse al punto más cercano de la superficie, esta operación con el resultado final del envase se muestra en la Figura 19.

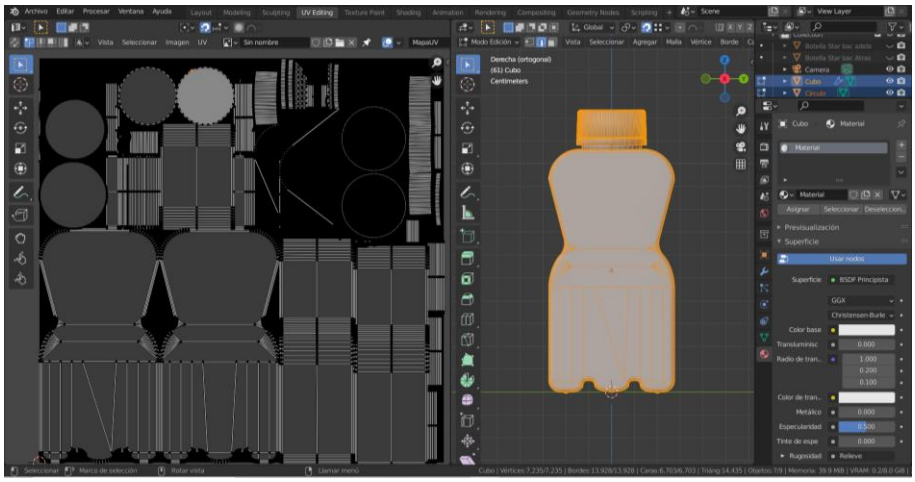
Figura 19

Envase de plástico renderizado

Dado que el objeto es exportado a los ambientes virtuales se toma los criterios de manejo de polígonos en escena, Unity Technologies (2018) explica que el rango ideal para plataformas de escritorio está entre 1500 a 4000 polígonos por malla, en Unreal Engine 4 el límite en todo tipo de malla puede ser hasta 65000 polígonos (Epic Games, Inc., s.f.). El modelado del envase plástico está formado por 6703 polígonos con alta cantidad de detalles, con el propósito que la cámara virtual en escena logre mostrar y no signifique un costo computacional alto al equipo en el renderizado de cuadro por segundo, esto se muestra en la Figura 20.

Figura 20

Mapa UV del envase modelado



Nota. Polígonos desplegados del mapa UV del envase modelado

3.2. Generador de datos sintéticos

Uno de los principales problemas que enfrenta la visión por ordenador es la segmentación de imágenes que busca separar una o varias regiones de interés de estructura similar sobre el resto de la imagen, en base al problema que se esté tratando

(Palomino & Concha, 2009). Los conjuntos de imágenes alojados en base de datos como COCO, PASCAL, Kaggle, etc., han sido procesadas manualmente por humanos para dibujar sobre las imágenes, verificar y corregir las máscaras de segmentación; un trabajo sumamente laborioso en tiempo y dinero que al final presenta polígonos toscos sin mucha definición en sus características. Además, el inconveniente de la segmentación va acompañado de pequeños conjuntos de imágenes disponibles en la red que dificultan el proceso de entrenamiento de redes de aprendizaje profundo (Nikolenko, 2019).

De estas razones nace la idea de los datos sintéticos con el objetivo de proporcionar datos artificiales (en este caso imágenes), que resuelvan la insuficiencia de datos y etiquetado manual de máscaras de segmentación para brindar datos diversos frente a circunstancias externas que algunas veces suelen llegar a ser prohibitivos (Nikolenko, 2019).

Para complementar la etapa de conjuntos de datos del sistema virtual se trabaja en el generador de datos sintéticos del software Unreal Engine v4.22.3, en base al plugin *NVIDIA Deep Learning Dataset Synthesizer* (NDDS) que permite exportar imágenes sintéticas a través de escenas aleatorias para entrenar redes neuronales profundas.

La presente etapa se enfoca en generar datos sintéticos de imágenes fotorrealistas y aleatorización de dominio. En el enfoque de fotorrealismo se busca la puesta del modelo 3D en escena con restricciones físicas en primer plano, dada la alta fidelidad del modelo a comparación de ambientes reales.

Por otra parte, Nikolenko (2019) menciona que la aleatorización de dominio o DR (de las siglas en inglés *Domain Randomization*) se considera uno de los enfoques prometedores para que los datos sintéticos funcionen y sean llevados a la realidad. Con el objetivo de ampliar el conjunto de datos para el entrenamiento de la red y tener una distribución variada de imágenes, la aleatorización de dominio complementa a los datos fotorrealistas para mejorar la robustez del modelo y evitar el sobreaprendizaje de la red a condiciones físicas que existen en la escena, perdiendo la habilidad de detección en casos nuevos.

3.2.1. Plugin NDDS

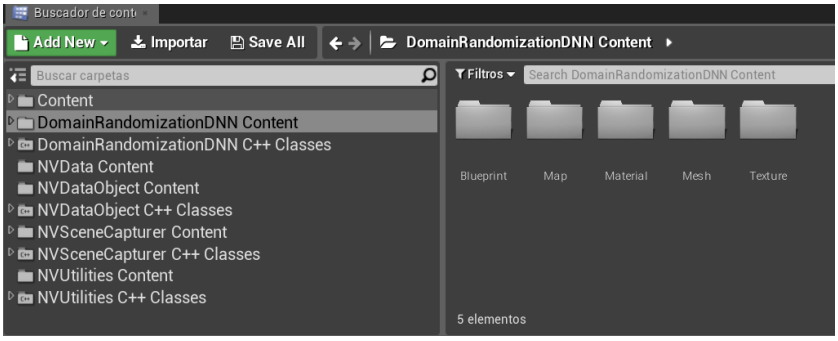
NDDS incluye componentes de segmentación de clases, instancia, profundidad, pose de objeto, cuadro delimitador y plantillas con notación de objeto de JavaScript (.json) para el intercambio de datos. En escena brinda elementos como distractores, cambios de texturas, posición, iluminación, cámara, etc., para lograr imágenes sintéticas de alta calidad (Tremblay, y otros, 2018).

En primera instancia se trabaja sobre la carpeta del plugin NDDS al generar un nuevo nivel vacío, el contenido de este se muestra en la Figura 21. Unreal Engine maneja *Blueprints* que son la manera de programar en C++ mediante objetos visuales, de la misma forma, también son contenedores de conjuntos de funciones. Es el caso de la carpeta “Content” donde se alojan los Blueprints de mallas, texturas, materiales y mapas de los ambientes realistas a crear y utilizar; “Domain RandomizationDNN” Content proporciona Blueprints para la aleatorización de dominios, aquí se incluyen los anotadores de segmentación y por último “NVSceneCapturer Content” incluye

Blueprints de capturadores simples o de gamas comerciales de la marca Intel RealSense.

Figura 21

Contenido del Plugin NDDS



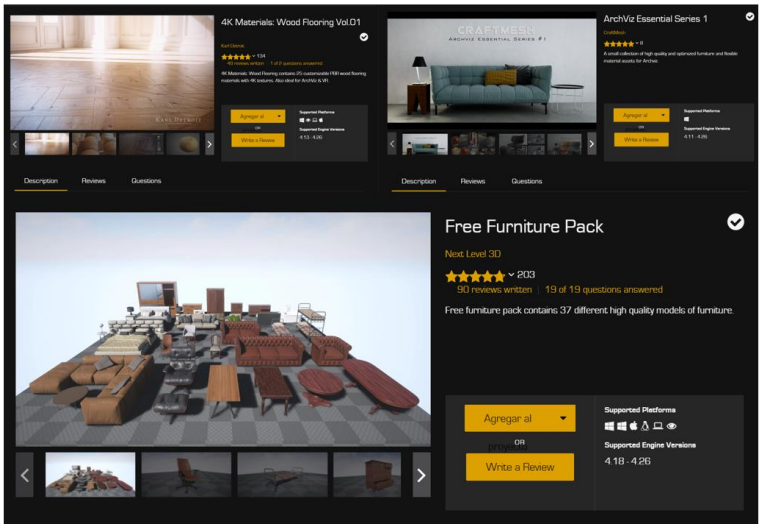
3.2.2. Creación de escenas realistas

El objetivo es crear la escena realista de una sala con objetos comunes, piso de madera e iluminación de luz del día en el interior de la habitación donde posteriormente se ubica el objeto modelado en la sección 3.1. para la generación de datos sintéticos fotorrealistas.

Gran parte de los elementos de la escena no necesitan ser modelados ya que se tardaría tiempo y esfuerzo, es por eso que la tienda de Unreal Engine de la empresa Epic Games distribuye material de mallas de objetos ya texturizados y mapas completos de uso público que se ocupan en el diseño de la escena, se observa en la Figura 22.

Figura 22

Marketplace de Epic Games

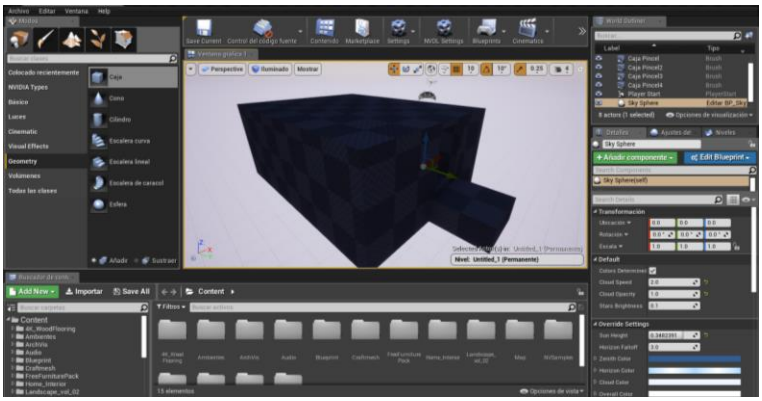


Nota. En la esquina superior izquierda se muestra materiales para pisos de madera en resolución 4K, en la esquina superior derecha la colección de decoraciones y en la parte inferior muebles del hogar.

El diseño parte de la geometría de caja de 640 x 420 x 340 cm de base para las paredes interiores y una abertura de luz estilo ventana a la derecha desde la vista frontal, se muestra en la Figura 23.

Figura 23

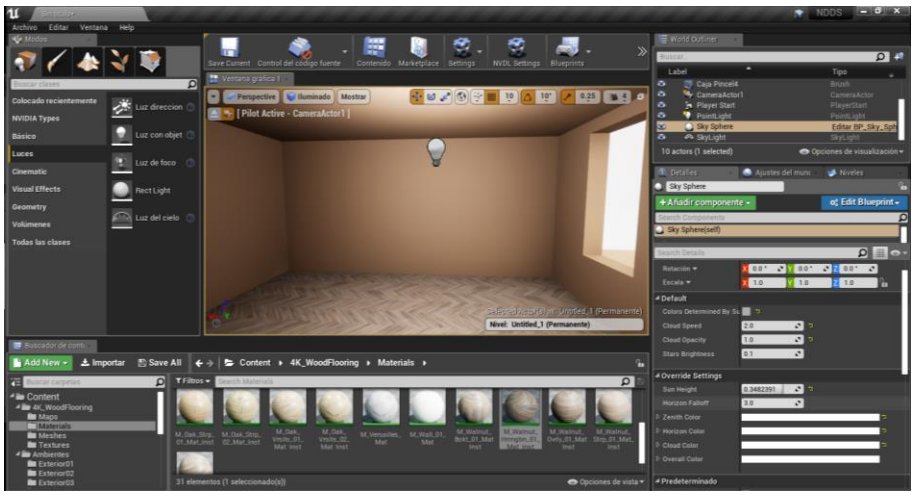
Vista en perspectiva de caja base



Para que la escena interior se vea realista se agrega el material en color café tabaco #745a43 en las paredes y textura de madera de Nogal Walnut, convirtiendo a estas geometrías en masas de luz con la capacidad de reflexión y absorción de luz como se observa en la Figura 24.

Figura 24

Materiales de escena interior



Finalmente, con la distribución de las mallas que consta de un mueble principal, lámpara, mesa, cuadros y otros elementos decorativos se consigue un diseño arquitectónico interior. Los factores importantes de renderizado son la iluminación que apoya a la escena con una luz direccional de 2.0 candelas y la esfera de reflexión para brindar reflejos de los objetos con la superficie del suelo, el renderizado final de la escena interior se expone en la Figura 25.

Figura 25

Renderizado de escena interior

Con el propósito de aumentar y diversificar la cantidad de datos sintéticos se utilizan 6 escenarios de alta calidad, distribuidas en ambientes interiores y exteriores. En la Figura 26 se muestra un comedor, una cocina y dos salas.

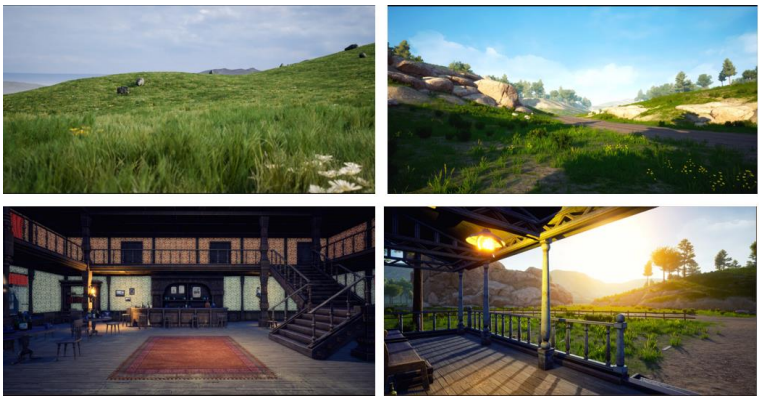
Figura 26

Ambientes realistas de interiores

Para tener características ambientales diferenciables de interiores en la Figura 27 se indica 4 ambientes realistas de exteriores: pradera, jardín, bar y corredor.

Figura 27

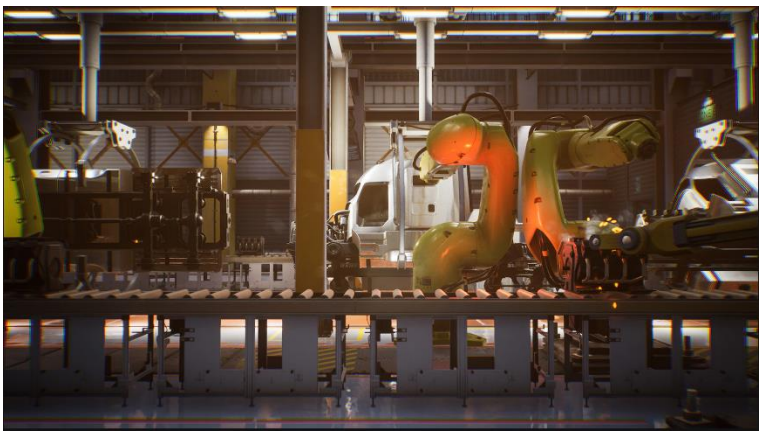
Ambientes realistas de exteriores



Al igual que este proyecto se encuentra enfocado para aplicaciones robóticas se incluye un ambiente de fábrica como se observa en el conjunto de imágenes de la Figura 28.

Figura 28

Escena de fábrica realista



3.2.3. Creación de escena de aleatorización de dominios

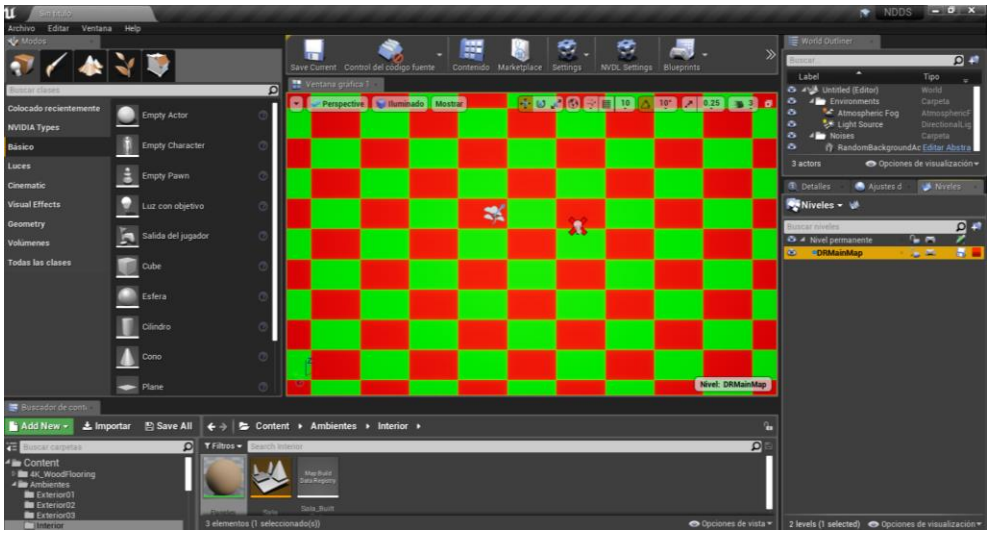
El objetivo de este apartado es la configuración del nivel de aleatorización de dominios, una escena 3D con texturas aleatorias en patrones de cuadros de fondo;

además, de poliedros cruzando toda la escena, aleatorización de parámetros de cámara y luz.

La figura 29 proyecta el mapa del nivel de Aleatorización de Dominio “DRMainMap” base de toda la escena contenido dentro de la carpeta del plugin “DomainRandomizationDNN Content\Map”.

Figura 29

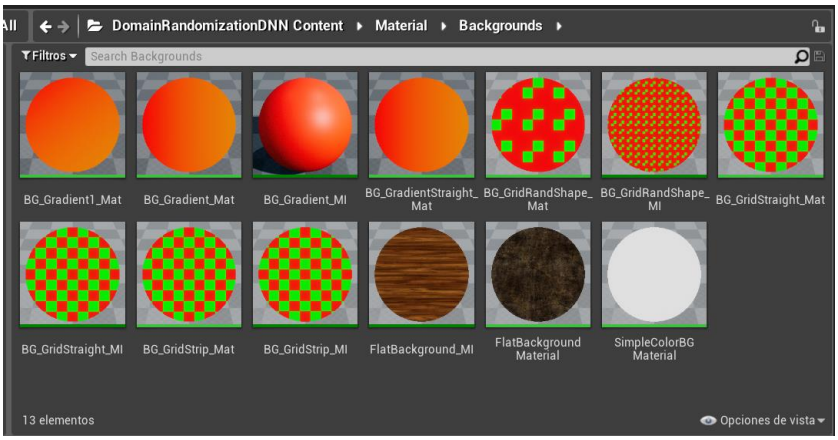
Mapa de Aleatorización de Dominio



Este contiene varios materiales de gradiante de color, mallas con patrones de cuadros de diferentes volúmenes y texturas de superficies que estarán cambiando aleatoriamente sus valores durante la ejecución, el conjunto se visualiza en la Figura 30.

Figura 30

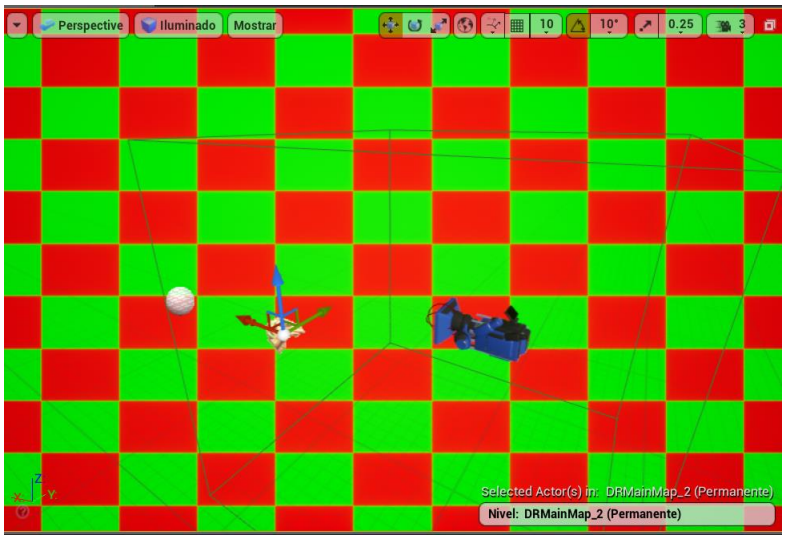
Fondos de DR



Para desarrollar los poliedros que son los distractores de la escena se ubica el Blueprint de “GroupActorManager_BP” de 40 a 60 elementos por *frame* o cuadro por segundo dentro de un volumen tipo caja de 400 uu (unidades) por lado llamado “Caja_Volumen”. Se incluye un actor de cámara para visualizar la escena DR completa, como se muestra en la Figura 31.

Figura 31

Blueprint de poliedros en escena

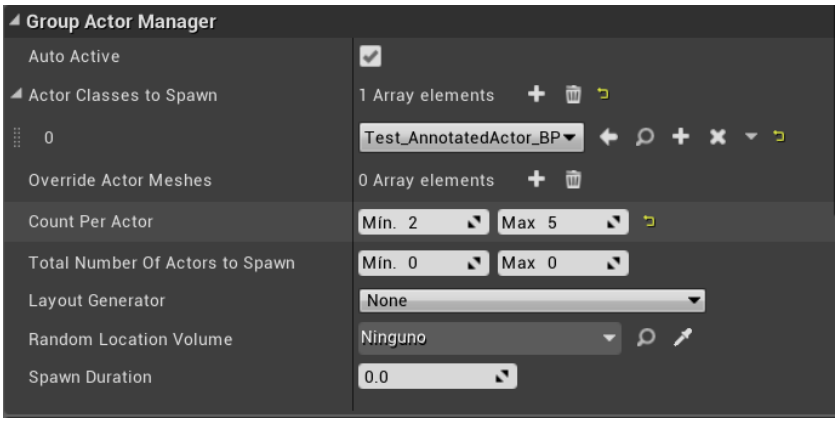


Nota. El cubo o caja que rodea a la cámara es el elemento ”Caja_Volumen”; el objeto con coordenadas es el indicador de GroupActorManager_BP.

La respectiva configuración de los Blueprints de objetos se visualiza en la Figura 32, donde inicia aleatoriamente con los rangos establecidos cada vez al momento de ejecutar el nivel del proyecto.

Figura 32

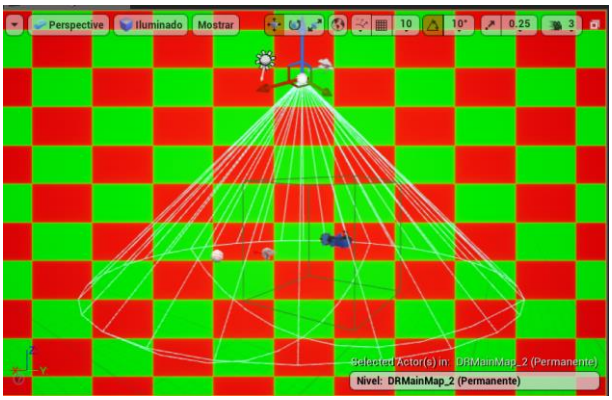
Configuración Blueprint ActorManager_BP



La Figura 33 muestra un Blueprint de luz focal aleatoria sobre la escena se incluye para abarcar una recolección mayor de datos sintéticos, desde 450 a 5800 lumens (6W a 45W), en referencia a la tabla comercial de intensidades LED para el hogar hasta lámparas de alto voltaje de Alcon Lighting (2021) para abarcar todas las condiciones límites.

Figura 33

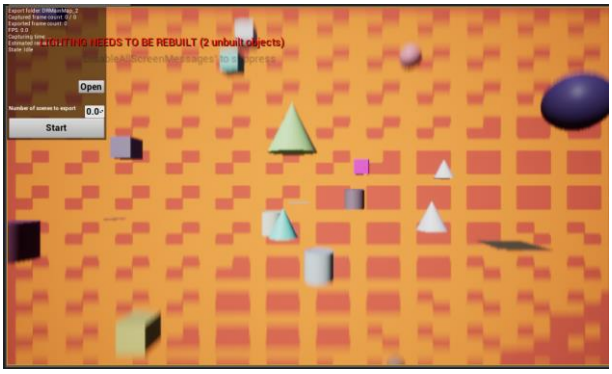
Configuración Blueprint de luz



Finalmente se proyecta en la Figura 34 el resultado de un fotograma aleatorio desde la vista de cámara sin objeto de la escena para aleatorización de dominios.

Figura 34

Escena DR



3.2.4. Generación de imágenes sintéticas

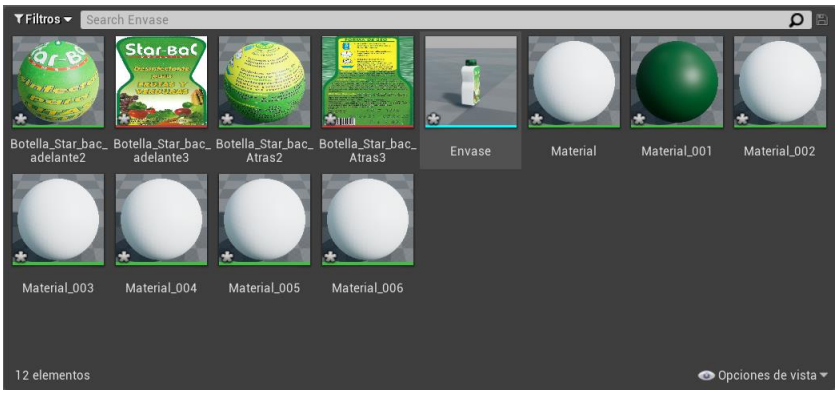
Con los escenarios virtuales tanto realistas como de aleatorización de dominio, se identifican dos etapas dentro de la generación de imágenes sintéticas: la

configuración del SDG y la obtención de imágenes sintéticas con los respectivos metadatos del envase. Las funciones integradas en el plugin NDDS permiten construir el SDG con algunas configuraciones sobre Unreal Engine.

En primer lugar, se realiza la importación del objeto a Unreal Engine v4.22.3 desde Blender v2.9 con la característica integrada de exportación FBX. La figura 35 muestra los archivos importados al directorio de la carpeta “Content” dentro del plugin NDDS, todas las partes del modelo se han integrado en una sola por lo que permite trabajar con la malla generada de nombre “Envase”.

Figura 35

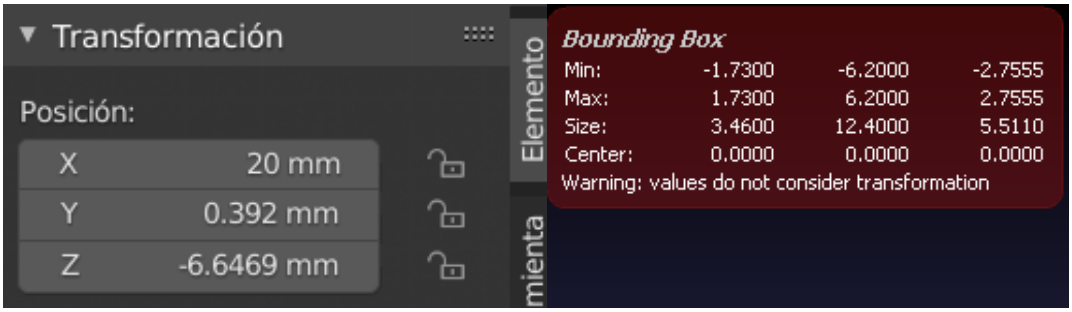
Archivo FBX del envase plástico



Dado que el envase dota de un marco de coordenadas XYZ propio en escena, se busca las coordenadas del centroide en (0,0,0). Con el programa Meshlab en “Render/Show Box Corners” se verifica las coordenadas del centroide, si estas son erróneas se debe verificar con la posición del modelado en Blender antes de la exportación. La Figura 36 muestra los valores necesarios en posición para obtener un centroide en el origen coordenado.

Figura 36

Centroide en Blender y Meshlab

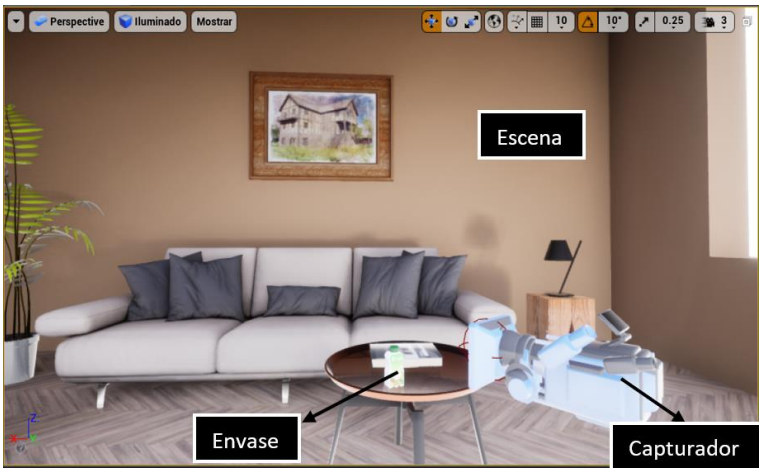


Nota. En la información de Meshlab (imagen derecha) se provee valores de centro y dimensiones en centímetros.

Para capturar imágenes se introduce a escena el Blueprint llamado “SceneCapturer_Simple” , este simula la acción de una cámara digital con extractores de características por defecto, la Figura 37 engloba los elementos en escena que intervienen en el SDG. A continuación se detalla la configuración de estos para la obtención de imágenes sintéticas.

Figura 37

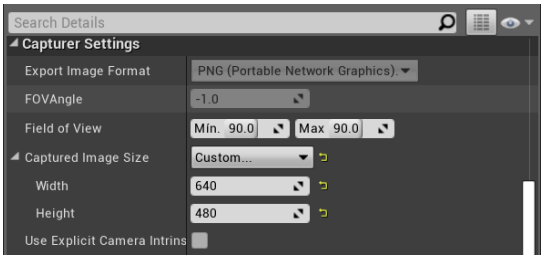
Elementos de escena



El capturador obtiene las características de (a) datos del objeto; (b) color verdadero; (c) profundidad; (d) segmentación de instancias; (e) segmentación de clases; todas necesarias para el propósito de la detección de pose 3D. El tamaño de captura de imágenes sigue el formato de 640x480 y de salida tipo PNG, ver Figura 38, el primero editable pero a mayor cantidad de tamaño más tiempo de procesamiento de pixeles para el modelo de CNN.

Figura 38

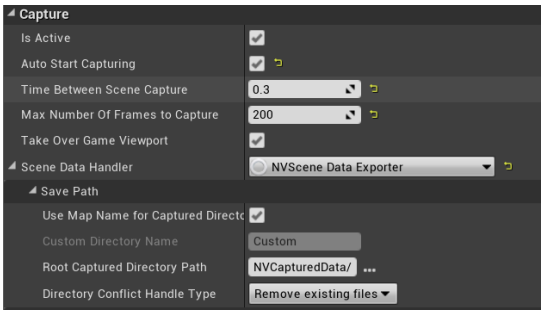
Configuración Capturador



Cada escena realista captura un total de 200 imágenes con un tiempo entre cuadro de 0.3 segundos, la Figura 39 muestra la configuración de este apartado, aparte del directorio de guardado de imágenes y metadatos en el ordenador.

Figura 39

Ubicación de guardado de imágenes y metadatos



Nota. Cada nueva captura de datos remueve archivos anteriores en el directorio

El modo de juego que maneja el proyecto es “NVSceneCaptureGameMode_BP”, este Blueprint activa las características del plugin NDDS con una ventana gráfica con botones interactivos para conocer el estado de capturas exportadas, cuadros por segundo, tiempo de captura, tiempo estimado, estado del SDG, botón de inicio, pausa y paro al momento de arranque del proyecto en Unreal Engine, ver Figura 40.

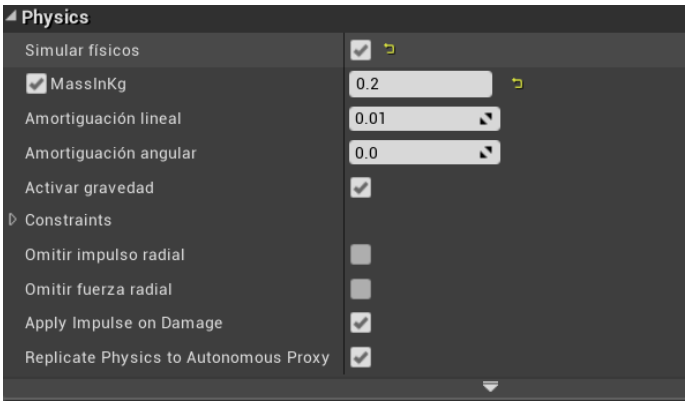
Figura 40

Ventana gráfica del SDG

Para los ambientes realistas, el envase tiene propiedades físicas para interactuar con los objetos de cada escena, la Figura 41 muestra la configuración con los valores de masa y gravedad descritos en la sección 2.2.4., esta opción identifica al objeto como movible, los problemas de cálculo de luz desaparecen ya que se esta operación se realiza por segundo en el arranque del proyecto.

Figura 41

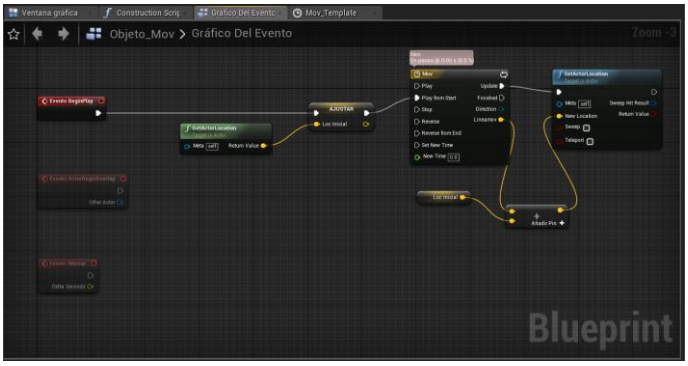
Propiedades físicas del envase



Con la intención de cubrir el envase en todas sus caras se desarrolla el componente de traslación que actúa una vez arrancado el proyecto. La serie de funciones, ver Figura 42, incluyen eventos en forma gráfica que facilitan la programación del movimiento del envase en sus coordenadas XYZ.

Figura 42

Gráfico de evento Traslación

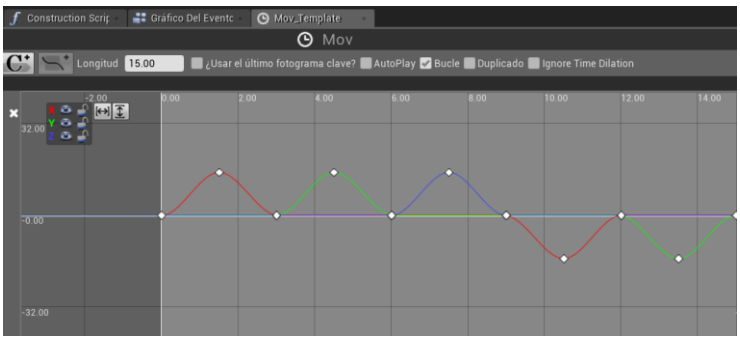


Nota. El gráfico presenta una sucesión de bloques donde se toma la posición inicial del objeto para ingresar a la línea de tiempo que forma el vector de traslación, finalmente, es sumado a la posición anterior y fijada al envase en escena.

La traslación del objeto depende de las dimensiones de la superficie donde actúa, sin embargo, se ha establecido una traslación base en X,Y $\pm 15uu$, en Z de 15uu en un tiempo de 3 segundos cada uno, por un total de 15 segundos en bucle continuo.

Figura 43

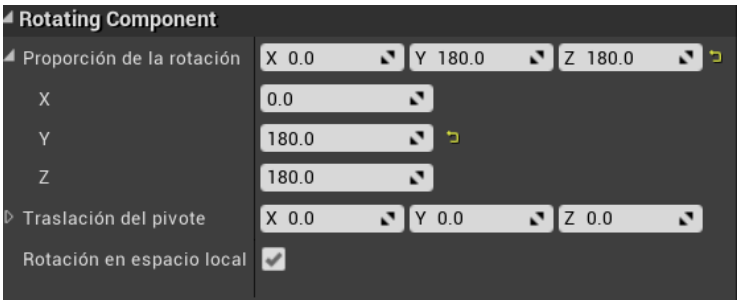
Vector de traslación del objeto



De igual forma se añade el componente de rotación al envase que se ejecuta mientras la traslación está en progreso. Se muestra la configuración en la Figura 41, donde se presenta un Blueprint propio de las herramientas de Unreal Engine que aplica bien a lo que buscamos; sin embargo, queda abierta la alternativa de crear este vector al estilo de la Figura 39.

Figura 44

Vector de rotación del objeto

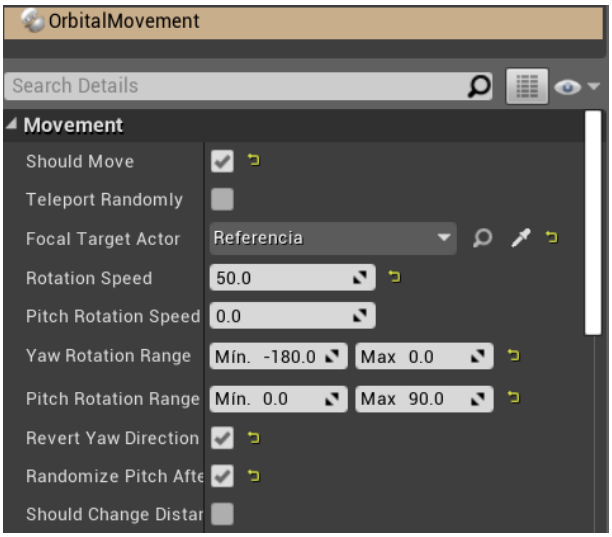


Nota. Las rotaciones combinan movimientos de 180° alrededor de los ejes Y y Z.

Por otra parte, para cubrir los alrededores de la escena el capturador presenta un movimiento orbital sobre un Blueprint de referencia. La Figura 45 muestra la configuración del movimiento orbital aleatoria con límites sobre los ejes Pitch y Yaw.

Figura 45

Rotación de capturador

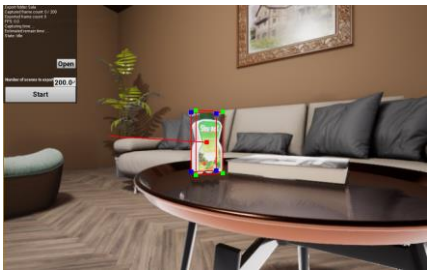


Nota. Los valores están sujetos a cambios que dependen de la escena y espacio de orbita.

Finalmente, se añade el componente “NVCapturableActorTag” que encierra al envase dentro de un cuboide delimitador 3D para que el identificador del SDG extraiga los metadatos correspondientes, es así, que definimos de nombre “envase” al Blueprint del objeto. La Figura 46 muestra la captura de imágenes sintéticas fotorrealistas.

Figura 46

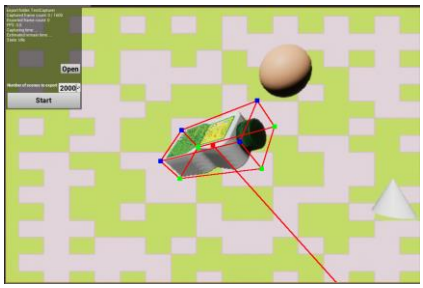
SDG en ambiente realista



Para la escena en aleatorización de dominios la configuración de cámara y objeto 3D se maneja de igual forma, excepto, que el envase es un Blueprint sin propiedades físicas que controlar. En la Figura 47 se visualiza en escena el resultado final del SDG para la captura de 2000 imágenes sintéticas de aleatorización de dominio.

Figura 47

SDG en DR

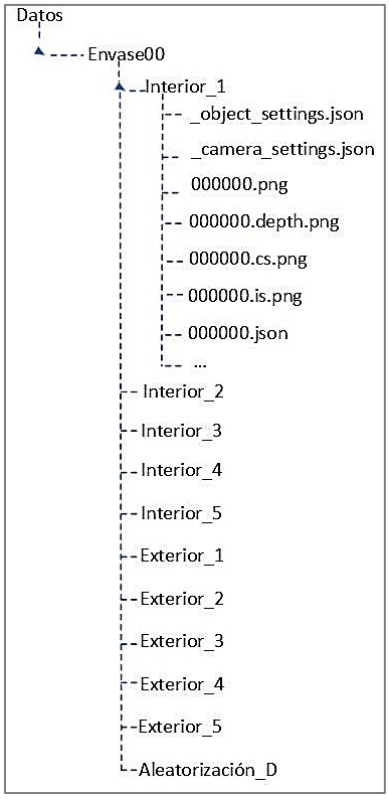


3.2.5. Metadatos de imágenes sintéticas

El plugin NDDS se ha utilizado para visualizar la superposición del modelo según la verdad del terreno, estas imágenes sintéticas vienen acompañado de metadatos que son datos acerca de los datos generados. La Figura 48 muestra el contenido del directorio por las diez escenas realistas y una de aleatorización de dominios.

Figura 48

Jerarquía de datos sintéticos



Cada carpeta de datos contiene fotogramas con dos archivos que describen la escena exportada en forma JSON. Estos son:

- "_object_settings.json", incluye la información sobre el objeto exportado (nombre de clases, ID de clase y segmentación semántica, dimensiones del cuboide 3D)
- "_camera_settings", detalla los parámetros intrínsecos del capturador (dimensiones de imagen)

Por parte de los fotogramas se genera un archivo de anotaciones de tipo JSON que incluye información del objeto, detalladas a continuación:

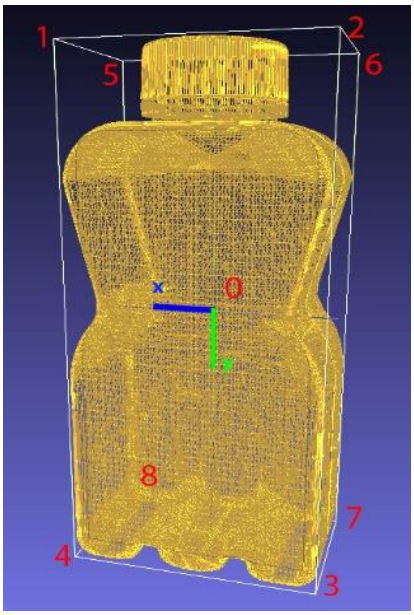
- “location_worldframe”, posición del capturador en coordenadas del mundo
- “quaternion_xyzw_worldfram”, orientación de la cámara en coordenadas del mundo
- “class”, nombre de la clase definido como envase
- “visibility”, visibilidad del objeto en la imagen (0 totalmente ocluido y 1 si es visible)
- “location”, posición XYZ en centímetros del origen
- “quaternion_xyzw”, orientación en cuaterniones en tres dimensiones.
- “pose transform”, matriz de pose del objeto
- “cuboid centroid”, posición 3D del centroide del cubo delimitador en centímetros
- “projected_cuboid_centroid”, proyección 3D del centroide del cuboide
- “bounding box”, caja delimitadora 2D del objeto sobre la imagen en pixeles
- “cuboid”, coordenadas 3D de los vértices del cuboide delimitador 3D en centímetros
- “projected_cuboid”, coordenadas 2D de la proyección del cuboide en pixeles

3.2.6. Cuboide delimitador 3D del objeto

La manera de ver a la pose 3D del objeto se define con un cuboide delimitador 3D, encierra al objeto con un marco de coordenadas propio; es así, donde cada vértice del cuboide se transforma en puntos característicos del envase. En la Figura 49 se visualiza el marco de coordenadas en el centro del envase y los 9 puntos característicos del cuboide delimitador.

Figura 49

Cuboide delimitador 3D



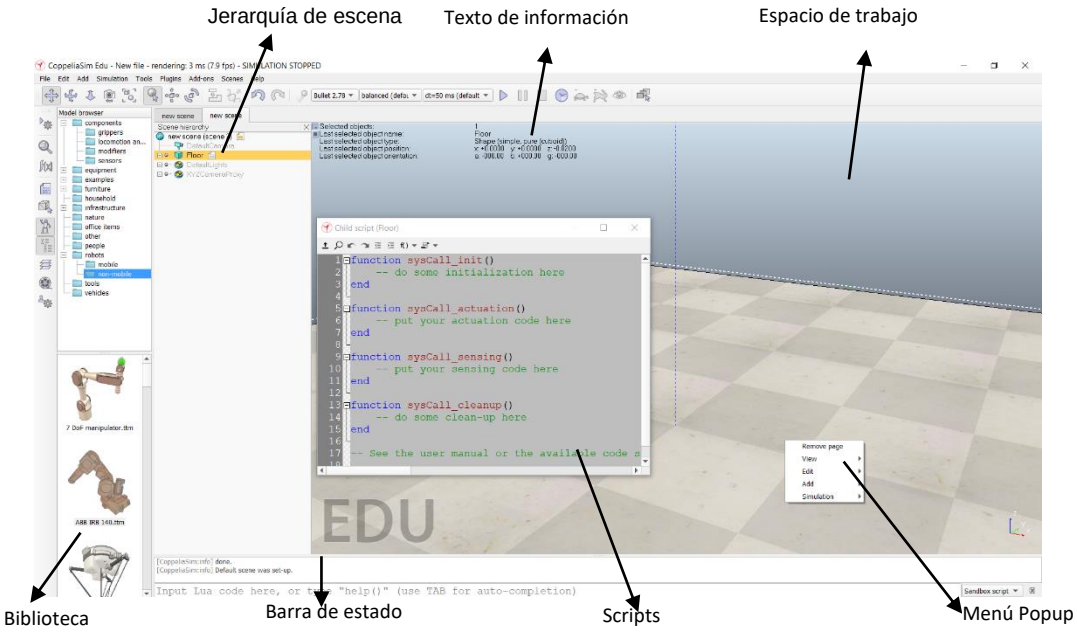
Nota. Desde el punto de visto frontal del envase desde el exterior, el eje X apunta a la izquierda, el eje Y apunta hacia abajo y el eje Z apunta hacia afuera directo al espectador.

3.3. Desarrollo del módulo de manipulación Pick and Place

CoppeliaSim es el software caracterizado por crear escenas completas de simulación en tiempo real donde su fuerte destaca en entornos con robots manipuladores fijos y móviles. En la Figura 50 se visualiza la escena por defecto al abrir el software. Los elementos destacables en la interfaz de usuario están distribuidos por el espacio de jerarquía de la escena, barra de estado, barra de herramientas, texto de información de la aplicación, menús emergentes y una biblioteca con una extensa variedad de modelos en equipamiento, efectores finales, robots, sensores, etc.

Figura 50

Interfaz de usuario CoppeliaSim

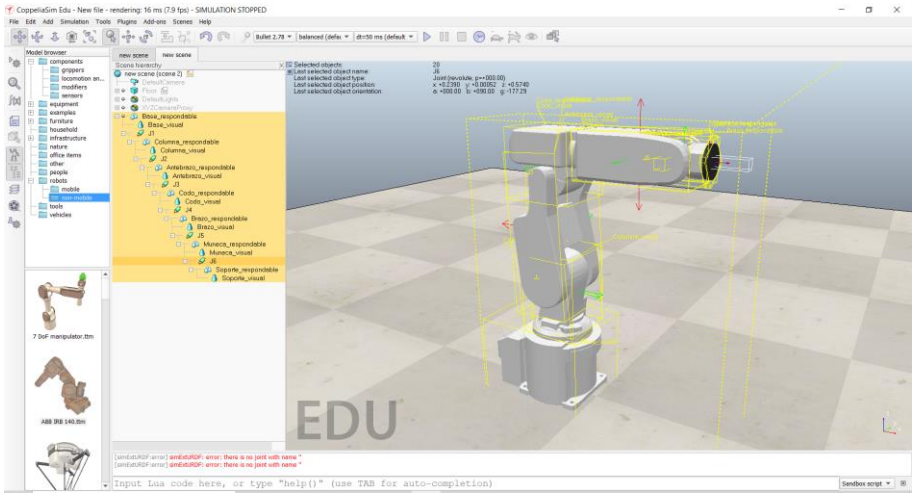


Estos modelos precargados son de libre edición gracias a la creación de *scripts* embebidos en cada uno de los objetos, estos son usados en el proyecto para tener un control de los modelos y realizar las conexiones mediante APIs remotas y conseguir la integración con el detector de pose 3D.

En el software se integra el robot Mitsubishi RV-2SDB al espacio de trabajo; las piezas del robot han sido descargadas del sitio oficial (Mitsubishi Electric Corporation, 2010). La importación del modelo se realiza en formato URDF, en el que describen las dimensiones de los eslabones, juntas y sistemas de referencias adjuntos, tal como se observa en la Figura 51.

Figura 51

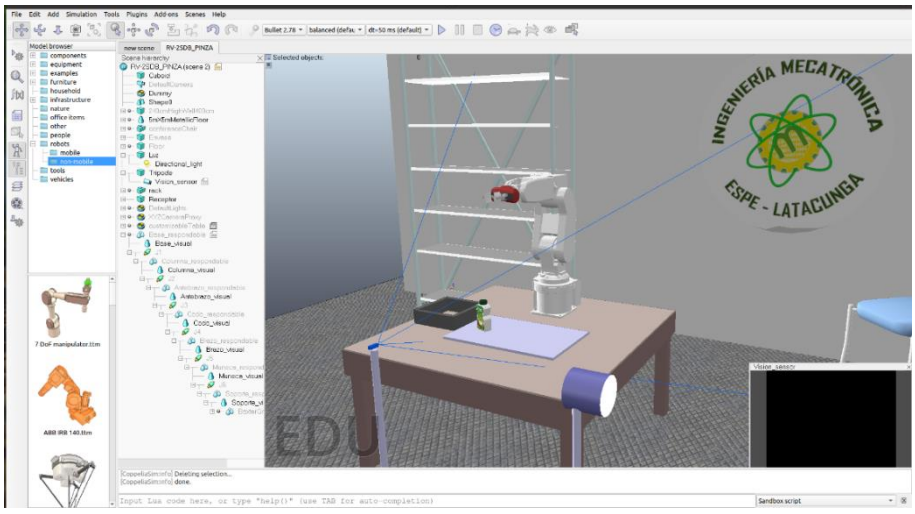
Importación de modelo CAD



El proceso de creación de la escena del entorno virtual se presenta en el Anexo A; a continuación, la Figura 52 muestra el resultado final de la aplicación de pick and place.

Figura 52

Módulo para desarrollo aplicación robótica



El manipulador se encuentra en el origen de coordenadas sobre una mesa con el objeto modelado en su radio de alcance y a la derecha el espacio para la recepción de este. El sensor de visión que actúa de cámara virtual a un metro de distancia del origen de referencia para abarcar la escena, con una renderización de gráficos basada en OpenGL3 y un componente de luz direccional para la exposición a cambios de luz ambiental.

Capítulo IV

4. Sistema de visión artificial

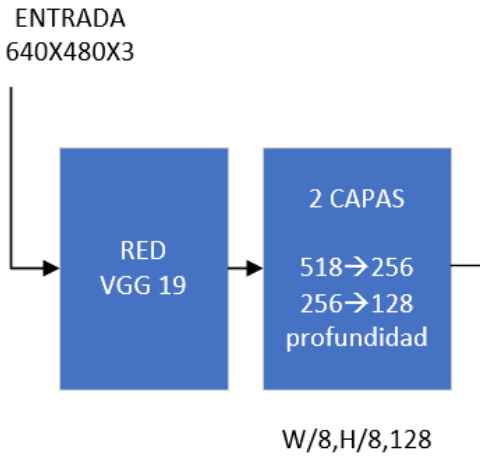
En el presente capítulo se detalla la arquitectura de la CNN para la detección de pose 3D y su respectiva configuración de hiperparámetros para el entrenamiento de la misma. Para continuar con el proceso de calibración, importante para realizar la proyección de perspectiva sobre las imágenes; finalmente se presenta el algoritmo de control que acompaña al detector para la obtención de las posiciones angulares que el robot manipulador Mitsubishi RV-2SDB necesita para realizar la aplicación de manipulación.

4.1. Arquitectura de la red neuronal convolucional

La red de propagación hacia adelante o feedforward inspirada en DOPE (Tremblay, y otros, 2018) permite detectar puntos claves por cada una de las fases en su arquitectura. En la entrada recibe imágenes de $640x480x3$ este último valor correspondiente a los canales de color RGB, mientras a la salida devuelve nueve mapas de creencia y un par de ocho mapas de afinidad correspondientes a los vértices proyectados del cuboide delimitador 3D que encierra al objeto. El Anexo B muestra la arquitectura completa de esta red CNN que toma en cuenta los resultados de características anteriores en cada capa de convolución, para comenzar se analiza la red preentrenada VGG del primer bloque mostrada en la Figura 53.

Figura 53

Bloque de entrada



A fin de extraer e identificar las características del objeto en las imágenes de entrada, se usa el método de *Transfer Learning* o llamado aprendizaje por transferencia para aprovechar su punto de partida en entrenamientos con más de mil clases de objetos. Este es el caso de la red neuronal VGG19 la cuál consta de 19 capas (16 capas convolucionales, 3 capas Fully connected, 5 capas de MaxPooling y 1 capa de SoftMax), detallado en la Figura 54 junto a las variantes entrenadas en el conjunto de datos ImageNet.

Figura 54

Configuración ConvNet

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Nota. La profundidad aumenta conforme se aumenta el número de capas. Obtenido de (Simonyan & Zisserman, 2015)

En cada capa de convolución se manejan filtros cuadrados, tal es el caso de conv3-64 que significa 64 capas en un kernel de 3x3, los filtros aumentan el doble en cada bloque (64,128,256,512). Los resultados de la etapa anterior se convierten en la entrada de la siguiente capa convolucional, esto ayuda a encontrar más información espacial sobre los mapas de características de la imagen de entrada y poder incentivar los valores cada cierto tiempo reduciendo la resolución de los mapas de características, es el trabajo de las capas maxpool.

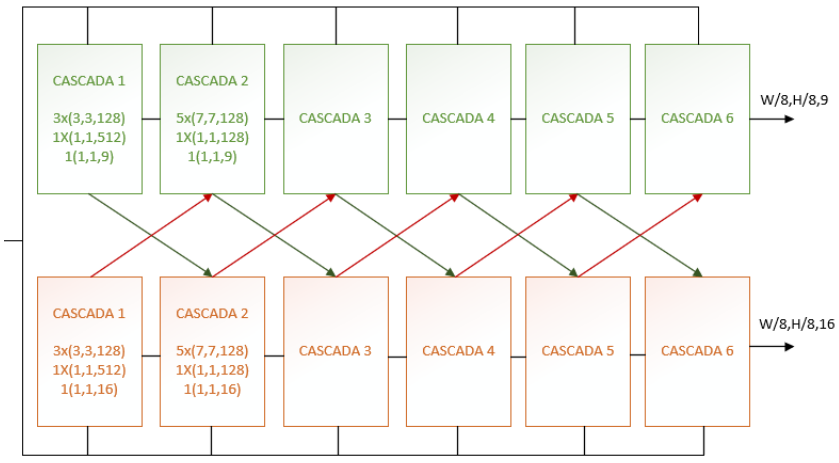
Para la arquitectura de la CNN las características de la imagen se calculan en las primeras diez capas llegando a manejar conv3-512; seguido de una capa de convolución 3x3 con el fin de reducir las dimensiones de las características de 512 a 256 y una última capa para conseguir 256 a 128. La salida de este bloque corresponde a un tamaño de $\frac{W}{8}, \frac{H}{8}$ de 128 mapas de características, que permite un trabajo más rápido de la red al generar los mapas de creencia y afinidad.

4.1.1. Mapas de creencia y afinidad

Los mapas de características son integrados a cada cascada de las cadenas de creencia y afinidad, en la Figura 55 se visualiza cada una de estas cadenas de capas de convolución para el cálculo de mapas de características vinculados al cuboide delimitador.

Figura 55

Cadenas de creencia y afinidad

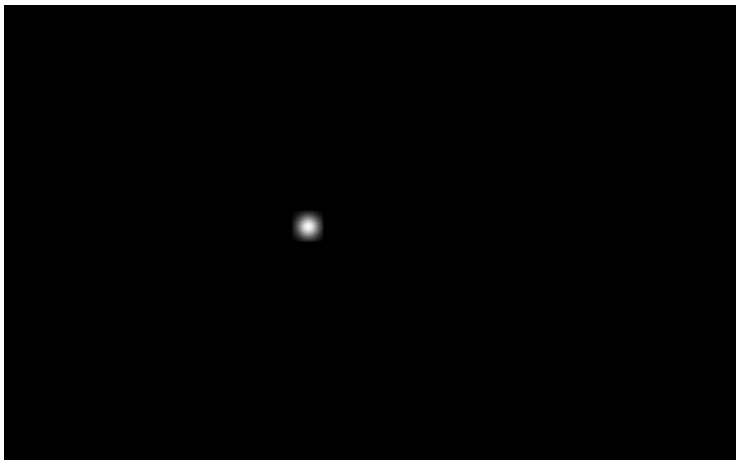


Nota. El conjunto de bloques color verde corresponde a la cadena de creencia, mientras que la cadena de afinidad por la agrupación color naranja.

Al final de las cadenas se busca el resultado de la detección en mapas de creencia que indican los puntos claves del cuboide delimitador sobre el objeto como un mapa de calor en 2D. Se definen entre 0 y 1, donde un valor alto equivale a encontrarse en aquella ubicación, la Figura 56 muestra un punto claramente definido en el mapa de creencia. Cabe mencionar, que a la salida de la cadena de creencia se espera tener nueves mapas correspondientes a los ocho puntos del cuboide delimitador y su centroide.

Figura 56

Mapa de creencia



Por otra parte, los mapas de afinidad relacionan los vértices hallados en forma de vectores normalizados con dirección hacia el centroide del cuboide. Cada punto de vértice está compuesto de vectores con componente en X y Y por eso la cadena maneja un total de 16 mapas de afinidad. Son de importancia para diferenciar en el momento de encontrar más de un objeto en la imagen.

La salida de estos mapas se encuentra al tener de entrada los mapas de dimensión 128 en el primer bloque, se realizan más operaciones de convolución para aprovechar el campo receptivo al pasar los datos por la red. Con tres capas de 3x3x128, una capa de 1x1x512 y una de 1x1x9 para creencia, alimenta a las siguientes cascadas relacionando las capas 1x1x16 de los componentes vectoriales. Las funciones de activación RELU en cada bloque permiten el paso de valores dentro de los límites, evitando generar no linealidades al mantener el tamaño de imagen en todas las etapas de $\frac{Width}{8}, \frac{Height}{8}$. Así al ingresar una imagen de 640x480x3 se ha obtenido en la salida mapas de características de 80,60,9 de creencia y 80,60,16 de afinidad.

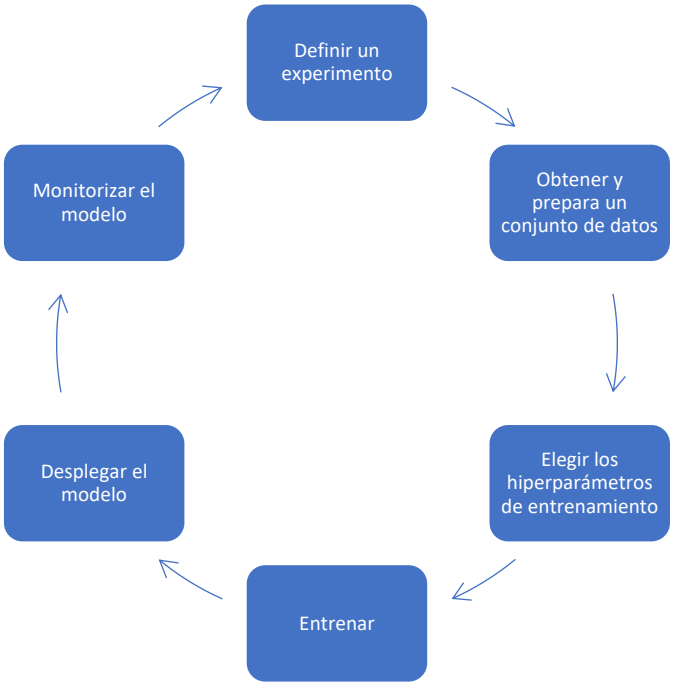
4.2. Entrenamiento del modelo de red neuronal convolucional

El entrenamiento de la CNN involucra una serie de parámetros previo al iniciar el aprendizaje. Desde la arquitectura de red, *learning rate* definido como la taza de aprendizaje*, epochs* o también llamado número de iteraciones, tamaño de imagen, *batchsize* o tamaño de lote; a estos parámetros se los definen como hiperparámetros ya que definen los pesos de la red para el propósito que ha sido diseñada. Es así, que este proceso de entrenamiento requiere varias pruebas para conseguir buenos resultados, acompañado del estudio de problemas similares o bien de criterios en base a la experiencia del usuario (Pérez & Genúndez, Deep Learning, 2021).

Se interpreta como un proceso iterativo al desarrollo de la CNN que al ingresar determinado conjunto de hiperparámetros a la salida se recibe un resultado para monitorizar el modelo. En la Figura 57 se detalla el flujo a seguir al elegir el problema a resolver en la red neuronal.

Figura 57

Flujo de experimentación de redes neuronales



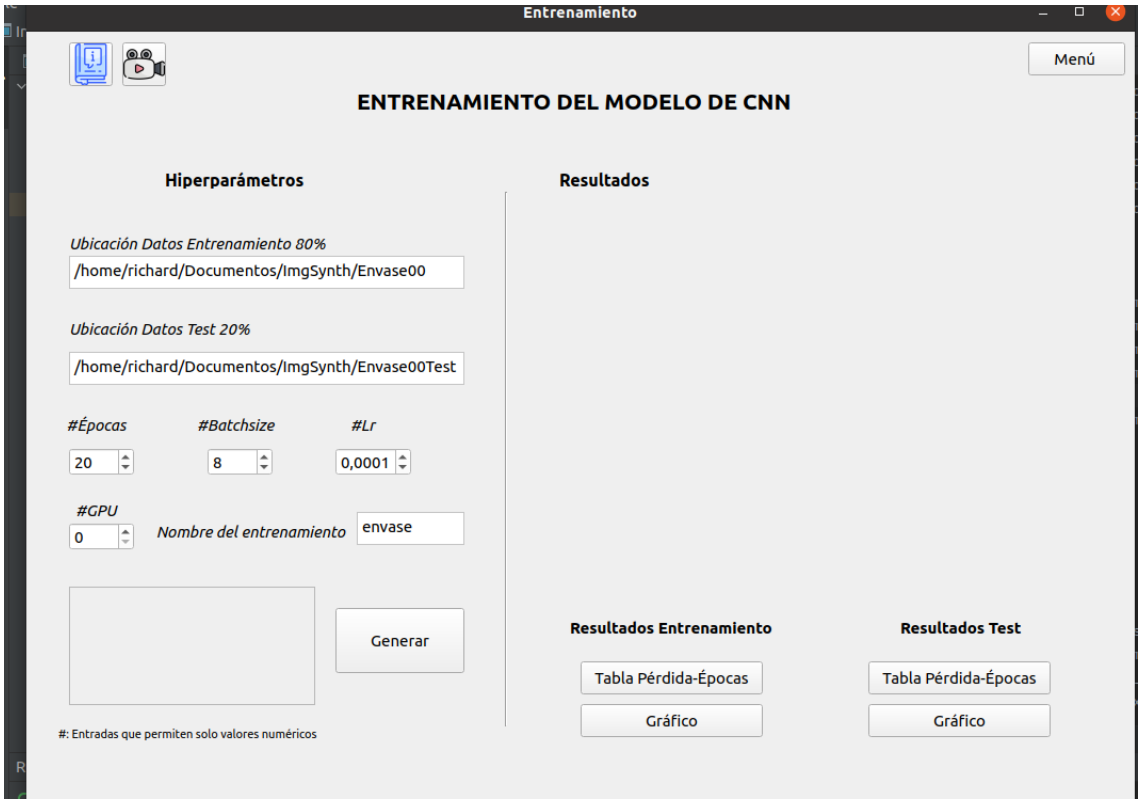
Nota. Baso en (Pérez & Genúndez, Deep Learning, 2021)

Mediante la técnica de Transfer Learning la red empieza con pesos obtenidos de ImageNet; para reducir el valor de pérdida al mínimo se necesitan varias épocas, mientras que el tamaño de lote a definir va de potencias de dos, así 2,4,8,16,32,64 y 128 los más recomendados. La capacidad de memoria de procesamiento influye directamente en la elección de estos, sin embargo, la elección de la taza de aprendizaje es primordial en gran medida para alcanzar el valor mínimo. Conectado al tamaño de lote por norma general en un lote de 128 la tasa de aprendizaje generada es de 0.1, cada vez que se divide el lote para 2 se divide entre 10 la taza de aprendizaje. Es importante considerar que si es demasiado pequeño este valor puede no terminar convergiendo y por otro lado divergir con pasos demasiados grandes. (Pérez & Gegúndez, Deep Learning, 2021).

Para el entrenamiento se dispone de un conjunto de 4000 imágenes del envase distribuidas equitativamente en fotorrealismo y aleatorización de dominio, sobre PyTorch v0.7 en la GPU GTX 1070 de 8GB de memoria. El sistema virtual de entrenamiento permite la configuración de hiperparámetros para realizar pruebas de estos conceptos e integra un visualizador de los resultados, ver Figura 58.

Figura 58

Ventana de entrenamiento del sistema virtual



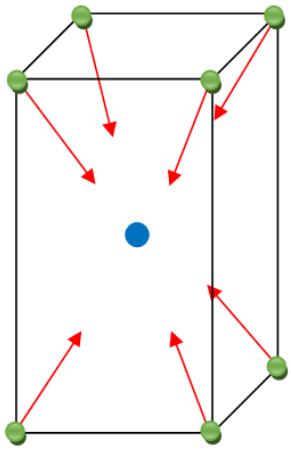
4.3. Detección de pose 3D

Considerando que se dispone de los mapas de creencia y afinidad se busca el valor más alto por encima del umbral que represente los vértices del cuboide delimitador

generado. Se realiza un procedimiento de asignación de cada vértice con los vectores X,Y hacia la dirección del centroide detectado. En particular la distancia de estos vectores a su respectivo centro (en caso de múltiples objetos). La Figura 59 muestra una representación del primer procedimiento detallado.

Figura 59

Vectores con dirección al centroide



Esta correlación facilita la transformación de los mapas de calor 2D hacia las características de texturas y color en la entrada de imagen RGB. Una vez que se han obtenido los vértices del objeto se utiliza el algoritmo de perspectiva PnP para encontrar la pose del objeto; este concepto se maneja en función del tamaño y donde se encuentra la imagen respecto a la vista de cámara.

Por ende, se dota a la red de los parámetros intrínsecos de la cámara, las dimensiones del objeto y los vértices proyectados para la obtención de la traslación y rotación final con respecto a la cámara, siempre y cuando sean visibles mínimo cuatro puntos.

4.3.1. Calibración de cámara

Dado el sistema de visión para detectar la posición y rotación, es necesario contar con las medidas de los objetos vista desde la cámara; tal es el caso de encontrar una relación de los pixeles en el plano de imagen (U, V) correspondientes a coordenadas (X, Y, Z) de la escena. Es así como nace el proceso de calibración, el resultado de transformaciones de coordenadas para obtener información del mundo real y corregir los datos que percibe el sensor de visión (Valderrama, 2000).

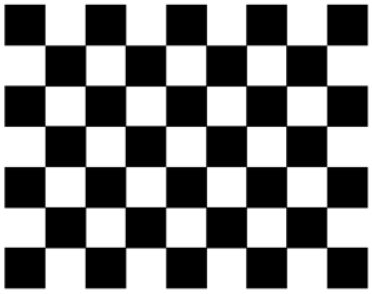
Existen dos tipos de parámetros que se pueden obtener en el proceso de la calibración de cámara: intrínsecos y extrínsecos. Este apartado se enfoca en encontrar los parámetros intrínsecos definidos por la geometría interna del conjunto cámara óptica; la información viene contenida en matrices dada por las distancias focales f_x, f_y y los centros ópticos c_x, c_y. Además de las distorsiones de la lente radial y longitud expresada por cinco coeficientes de distorsión. La Ecuación 3 expresa en forma simbólica la matriz de cámara.

$$Matriz\ de\ cámara = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

Los métodos con de calibración con patrones planos son las más versátiles a construir y devuelven buenos resultados. Es por eso, que se utiliza el método por calibración plano con patrones de matrices de cuadrados similar a un tablero de ajedrez, ver Figura 60, ya que la alternancia de colores blanco y negro define un mejor umbral a la detección de esquinas entre dos celdas.

Figura 60

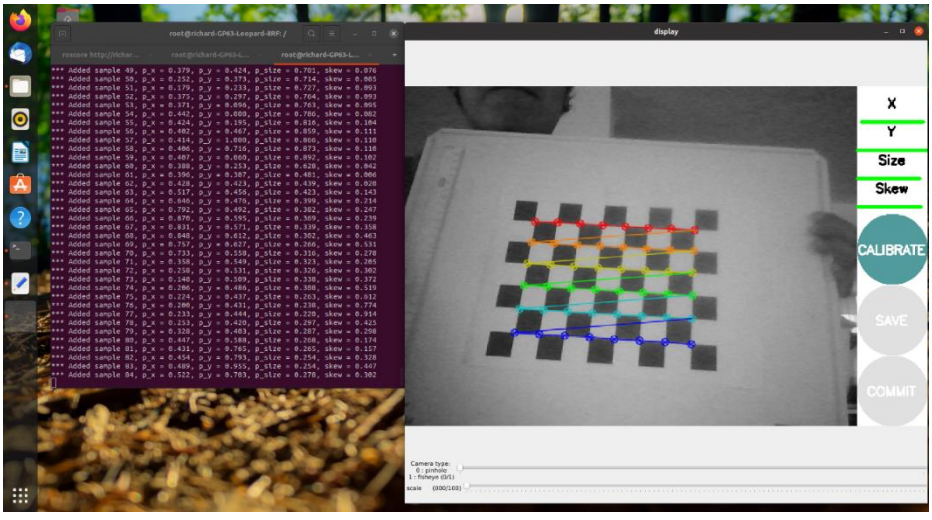
Tablero de calibración



El método se basa en la proyección de la imagen del patrón bidimensional al realizar el cálculo de homografía que se hallan en las coordenadas de cada esquina. De ahí que en este proceso se ingresa la dimensión de celda y el número de filas y columnas. En la Figura 61 se visualiza la calibración de la cámara web integrada en el hardware del ordenador, con un tamaño de celda de 0.02517 metros y tamaño 9x7. El desarrollo completo se encuentra en el Anexo C, la distorsión de cámara en forma simbólica se presenta en la Ecuación 4.

Figura 61

Proceso de calibración



Los parámetros intrínsecos calculados de la cámara web por el programa en Python, ver código de ejecución en Anexo D, son los siguientes:

$$Distorsión = (k_1 \;\; k_2 \;\; p_1 \;\; p_2 \;\; k_3) \tag{4}$$

$$Distorsión = (\,0.000899 \;\; -0.005626 \;\; 0.000480 \;\; 0.000653 \;\; 0.0000) \tag{5}$$

$$Matriz\ de\ cámara = \begin{bmatrix} 621.719233 & 0 & 329.325349 \\ 0 & 621.213129 & 263.306929 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Los resultados arrojados por la calibración en un tamaño de imagen de 640x480 se reflejan al analizar la Ecuación 5 donde los valores de p_1, p_2 son pequeños por el hecho de que las distancias al plano focal se acercan entre el eje X y Y, así mismo la Ecuación 6 muestra los centros ópticos un poco alejados del centro de imagen.

Dentro del sistema virtual no es necesario calibrar el sensor de imagen ya que se adjunta un script no anidado con los valores intrínsecos sin distorsiones; en la Figura 62 se visualizan estos valores escritos en LUA, el lenguaje que maneja CoppeliaSim para sus objetos.

Figura 62

Parámetros intrínsecos del sensor de visión

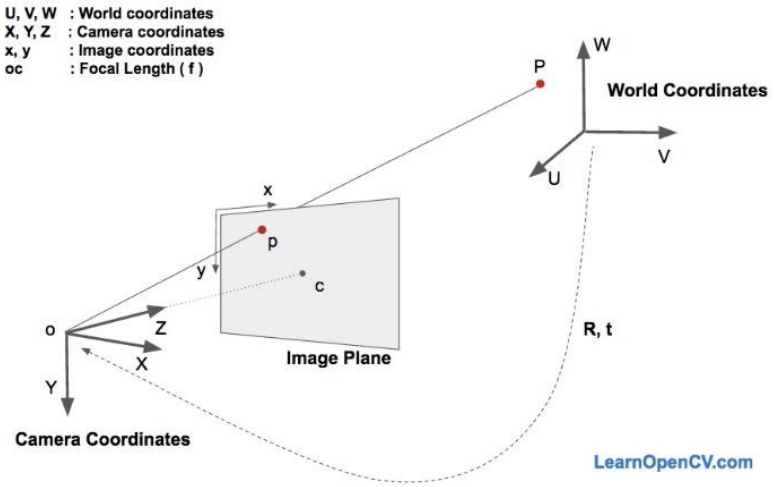
```
view_angle = 3.14/4
viewing_angle_id = 1004
sim.getObjectFloatParameter(activeVisionSensor, viewing_angle_id, view_angle)
f_x = (w/2)/math.tan(view_angle/2);
f_y = f_x;
ci={}
ci['header']={seq=0,stamp=t, frame_id="image"}
ci['height']=h
ci['width']=w
ci['distortion_model']="plumb_bob"
ci['D']={0, 0, 0, 0, 0}
ci['K']={f_x, 0, w/2, 0, f_y, h/2, 0, 0, 1}
ci['R']={1, 0, 0, 0, 1, 0, 0, 0, 1}
ci['P']={f_x, 0, w/2, 0, 0, f_y, h/2, 0, 0, 0, 1, 0}
ci['binning_x']= 1
ci['binning_y']= 1
ci['roi']= {x_offset=0, y_offset=0, width = 0, height = 0, do_rectify= false}
simROS.publish(pubCameraInfo,ci)
simROS.publish(pub,d)
```

4.3.2. Perspectiva-n-punto (PnP)

El problema de la Perspectiva-n-Point se resuelve al conocer los puntos clave del objeto en su propio sistema de coordenadas. Esta relación de pose y cámara, visualiza un concepto donde un rayo emana desde la apertura de la cámara a la ubicación del pixel. En la Figura 63 se aprecia la proyección del objeto con relación al sistema de referencia de la cámara.

Figura 63

Búsqueda del problema PnP



Nota. Obtenido de (Delgado, 2019)

La función de OpenCV3 llamada “cv::solvePnP” resuelve este problema, maneja argumentos similares a los de calibración con parámetros de entrada y salida:

- “cv::InputArray objectPoints “, localización de puntos del objeto
- “cv::InputArray imagePoints”, localización de los puntos en la imagen

- “cv::InputArray cameraMatrix”, matriz de cámara 3x3
- “cv::InputArray distCoeffs”, matriz de distorsión 1x5
- “cv::OutputArray rvec”, salida del vector resultante de rotación
- “cv::OutputArray tvec”, salida del vector resultante de traslación
- int flags = cv::ITERATIVE, tipo de método para resolver el sistema (Kaehler & Bradski, 2016).

Con los argumentos de entrada se fijan las banderas que definen el método para resolver el problema; en el primer condicional si existen 6 o más puntos válidos se aplica un *solver* iterativo (cv::ITERATIVE), este utiliza un optimizador de Levenberg-Marquardt para minimizar el error de proyección entre los imagePoints y los objectPoints. Por otro lado, en caso de encontrar 4 o más puntos válidos sin romper el primer condicional se aplica un solver P3P, que trabaja con 4 puntos de imagen; en esta situación se abre la posibilidad de implementar el solver (cv::EPNP). Al no ser iterativos, estos últimos manejan una velocidad más rápida.

Cómo resultado las matrices de traslación y rotación, esta última expresada en cuaterniones; números expresados en OpenCV por el orden WXYZ, gracias a la función “pyrr's Quaternion”. Son una representación fiable de las rotaciones sobre los ejes de desarrollo complejo que se han visto involucrados en aplicaciones de robótica, navegación, visión artificial, etc. La elección del algoritmo PnP o P3P se muestra en la Figura 64 y su desarrollo completo en el Anexo E.

Figura 64

Condición de activación PnP

```
if pnp_algorithm is None:
    if CuboidPNPSolver.cv2majorversion == 2:
        is_points_valid = True
        pnp_algorithm = cv2.CV_ITERATIVE
    elif CuboidPNPSolver.cv2majorversion > 2:
        if valid_point_count >= 6:
            is_points_valid = True
            pnp_algorithm = cv2.SOLVEPNP_ITERATIVE
        elif valid_point_count >= 4:
            is_points_valid = True
            pnp_algorithm = cv2.SOLVEPNP_P3P
```

4.4. Desarrollo del algoritmo de control

En este apartado se define las instrucciones necesarias para que el robot manipulador Mitsubishi alcance una posición y orientación en base a la detección encontrada. El desarrollo está realizado en lenguaje Python 3.8 para aprovechar las herramientas de cálculo numérico y bibliotecas de acceso remoto que proporciona CoppeliaSim.

4.4.1. Parámetros Denavit Hartenberg

En relación con el apartado 1.3.9.2 se determinan los valores de los sistemas de referencia para cada uno de los eslabones, se representa en la Tabla 19.

Tabla 19

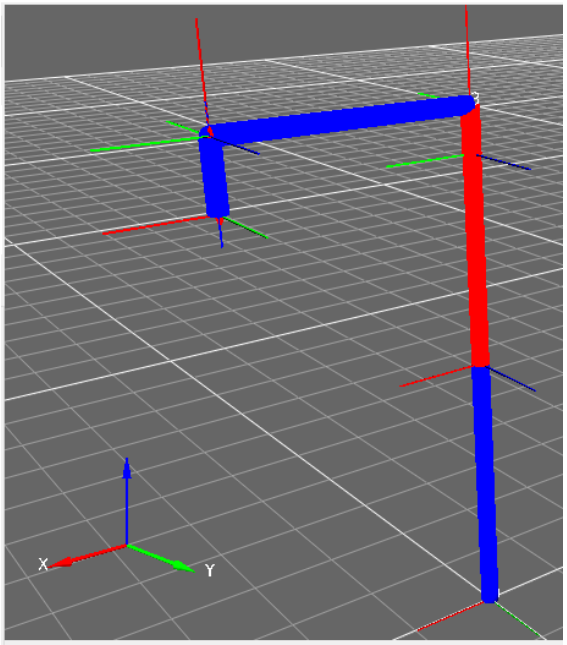
Parámetros DH brazo robótico

$\boldsymbol{\theta}$	$\boldsymbol{d}$	$\boldsymbol{l}$	$\boldsymbol{\alpha}$
$\boldsymbol{q_1}$	0.295	0	$-\pi/2$
$\boldsymbol{q_2 - \pi/2}$	0	0.230	0
$\boldsymbol{q_3}$	0	0.05	$-\pi/2$
$\boldsymbol{q_4}$	0.270	0	$\pi/2$
$\boldsymbol{q_5 + \pi/2}$	0	0	$-\pi/2$
$\boldsymbol{q_6}$	0.07	0	π

Con el conjunto de herramientas de código abierto de MRPT (Mobile Robot Programming Toolkit, 2021), en la Figura 65 se muestra la GUI “Robotic Arm Kinematic” para la interpretación visual del conjunto de parámetros DH.

Figura 65

Parámetros DH en GUI Toolkit



4.4.2. Algoritmo de cinemática inversa

El cálculo de cinemática inversa se ha realizado aprovechando la arquitectura de control distribuido de CoppeliaSim con el lenguaje de programación Python. Escrito enteramente en Python la biblioteca Sympy para matemática simbólica tiene el potencial de programas comerciales como Maple, Matlab, Mathematica, etc., manteniendo el código lo más simple y comprensible (SymPy Development Team., 2021)

Para establecer comunicación entre CoppeliaSim y Python se aloja el archivo de API junto a las librerías de comunicación “sim.py” y de cálculo simbólico “simpy.py” en el script Python del sistema virtual, la extracción de códigos en esta sección está detallado en el Anexo F. La Figura 66 muestra la instrucción al puerto 19999 de comunicación en el script no hilado del entorno virtual en CoppeliaSim.

Figura 66

Código para establecer comunicación

```
1  function sysCall_init()
2      simRemoteApi.start(19999)
3
4  end
5
```

Se obtienen los manejadores de las articulaciones y gripper como números de identificación que CoppeliaSim designa a los elementos de la escena. Con estos manejadores se obtiene el control de acciones sobre el objeto, se visualiza en la Figura 67.

Figura 67

Código de manejadores de articulaciones y gripper

```
retCode, joint1 = sim.simxGetObjectHandle(clientID, 'J1', sim.simx_opmode_blocking)
retCode, joint2 = sim.simxGetObjectHandle(clientID, 'J2', sim.simx_opmode_blocking)
retCode, joint3 = sim.simxGetObjectHandle(clientID, 'J3', sim.simx_opmode_blocking)
retCode, joint4 = sim.simxGetObjectHandle(clientID, 'J4', sim.simx_opmode_blocking)
retCode, joint5 = sim.simxGetObjectHandle(clientID, 'J5', sim.simx_opmode_blocking)
retCode, joint6 = sim.simxGetObjectHandle(clientID, 'J6', sim.simx_opmode_blocking)
retCode, signalValue2 = sim.simxGetObjectHandle(clientID, 'BaxterGripper_close', sim.simx_opmode_blocking)
```

Nota. Las articulaciones del manipulador están identificadas desde “J1” a “J6” dado por los 6 grados de libertad.

Los parámetros DH son el punto de partida para obtener las ecuaciones de cinemática directa que son la base para el cálculo de la cinemática inversa. Se inicia desde una función para construir las matrices de transformación de coordenadas homogéneas con el propósito de representar la posición y orientación del sistema de referencia. La Figura 68 muestra la función de cálculo simbólico de la matriz homogénea de rotación R_θ, traslación T_z sobre el eje Z y traslación T_l, rotación R_α sobre el eje X.

Figura 68

Código de matrices homogéneas con sympy

```
def symTfromDH(theta, d, a, alpha):
    # theta y alpha en radianes
    # d y a en metros
    Rz = sp.Matrix([[sp.cos(theta), -sp.sin(theta), 0, 0],
                    [sp.sin(theta), sp.cos(theta), 0, 0],
                    [0, 0, 1, 0],
                    [0, 0, 0, 1]])
    tz = sp.Matrix([[1, 0, 0, 0],
                    [0, 1, 0, 0],
                    [0, 0, 1, d],
                    [0, 0, 0, 1]])
    ta = sp.Matrix([[1, 0, 0, a],
                    [0, 1, 0, 0],
                    [0, 0, 1, 0],
                    [0, 0, 0, 1]])
    Ra = sp.Matrix([[1, 0, 0, 0],
                    [0, sp.cos(alpha), -sp.sin(alpha), 0],
                    [0, sp.sin(alpha), sp.cos(alpha), 0],
                    [0, 0, 0, 1]])
    T = Rz*tz*ta*Ra
    return T
```

La matriz de transformación final 4x4 resulta de la multiplicación de todos los sistemas, se expresa en la Ecuación 7. Además, este resultado se visualiza dentro de la etapa de “Aplicación Robótica” en la interfaz de usuario del sistema virtual.

$$T = T_0^1 * T_1^2 * T_2^3 * T_3^4 * T_4^5 * T_5^6 * T_e \qquad (7)$$

Donde T_e, representa el desplazamiento en Z desde la articulación J6 hasta el actuador final. El resultado es la representación de la relación entre el sistema de coordenadas de base hasta el extremo del actuador final considerando los parámetros variables de $q_1, q_2, q_3, q_4, q_5, q_6$.

Para el cálculo de la cinemática inversa la librería Sympy cuenta con un módulo solver llamado *nsolve* y permite obtener una solución por cálculo numérico. Resta definir la función de la matriz de rotación a partir de los ángulos de Euler , ya que son la representación que maneja CoppeliaSim para alpha (α), beta (β)y gamma (γ).

Las tres últimas articulaciones definen la orientación del manipulador, la Figura 69 muestra la función para construir la matriz de rotación a partir de los ángulos de Euler.

Figura 69

Código de matriz de rotación

```
# Definimos una función para construir la matriz de rotación
# a partir de los ángulos de euler
def matrixFromEuler(alpha, beta, gamma):
    # theta y alpha en radianes
    # d y a en metros
    Ra = sp.Matrix([[1, 0, 0, 0],
                    [0, sp.cos(alpha), -sp.sin(alpha), 0],
                    [0, sp.sin(alpha), sp.cos(alpha), 0],
                    [0, 0, 0, 1]])
    Rb = sp.Matrix([[sp.cos(beta), 0, sp.sin(beta), 0],
                    [0, 1, 0, 0],
                    [-sp.sin(beta), 0, sp.cos(beta), 0],
                    [0, 0, 0, 1]])
    Rc = sp.Matrix([[sp.cos(gamma), -sp.sin(gamma), 0, 0],
                    [sp.sin(gamma), sp.cos(gamma), 0, 0],
                    [0, 0, 1, 0],
                    [0, 0, 0, 1]])
    T = Ra * Rb * Rc
    return T
```

Con los valores de posición destino del detector se construye la matriz “D”, ver Figura 70, que es el producto de la matriz de traslación con la matriz de rotación.

Figura 70

Código de matriz de transformación de valores conocidos

```
t = sp.Matrix([[1, 0, 0, x],
               [0, 1, 0, y],
               [0, 0, 1, z],
               [0, 0, 0, 1]])
D = t * matrixFromEuler(alpha, beta, gamma)
```

El módulo “nsolve” de sympy busca los valores donde la expresión se iguale a cero para la resta matricial de la Ecuación 7, al buscar los ángulos $q_1, q_2, q_3, q_4, q_5, q_6$ que hacen que todos los elementos de la matriz se satisfagan. En la Figura 71 se visualiza las instrucciones ingresadas para los argumentos que acercan a la solución.

$$T - D = 0 \qquad (7)$$

Figura 71

Código del solver de cálculo numérico

```
try:
    q = sp.nsolve((T - D), (q1, q2, q3, q4, q5, q6), (0, 0, 0, 0, 0, 0), prec=6)
except:
    print('no se encontró la solución')
    q = [0, 0, 0, 0, 0, 0, 0]
print(q)
```

Nota. Los valores iniciales de búsqueda de solución están establecidos desde cero con una precisión de seis cifras significativas. En caso de no encontrar solución, devuelve cero a q

Por último, se envía los ángulos a las articulaciones por los manejadores obtenidos previamente, se muestra en la Figura 72.

Figura 72

Código de acción de articulaciones

```
retCode = sim.simxSetJointTargetPosition(clientID, joint1, q[0], sim.simx_opmode_oneshot)
retCode = sim.simxSetJointTargetPosition(clientID, joint2, q[1], sim.simx_opmode_oneshot)
retCode = sim.simxSetJointTargetPosition(clientID, joint3, q[2], sim.simx_opmode_oneshot)
retCode = sim.simxSetJointTargetPosition(clientID, joint4, q[3], sim.simx_opmode_oneshot)
retCode = sim.simxSetJointTargetPosition(clientID, joint5, q[4], sim.simx_opmode_oneshot)
retCode = sim.simxSetJointTargetPosition(clientID, joint6, q[5], sim.simx_opmode_oneshot)
```

Las instrucciones que se manejan para abrir y cerrar el gripper están definidas por 0 y 1 respectivamente, ver Figura 73.

Figura 73

Código de acción de gripper

```
retCode, signalValue = sim.simxGetObjectHandle(clientID, 'BaxterGripper', sim.simx_opmode_blocking)
retCode, signalValue2 = sim.simxGetObjectHandle(clientID, 'BaxterGripper_close', sim.simx_opmode_blocking)
retCode = sim.simxSetIntegerSignal(clientID, 'BaxterGripper_close',0, sim.simx_opmode_blocking)
```

4.5. Integración del sistema detector y entorno virtual

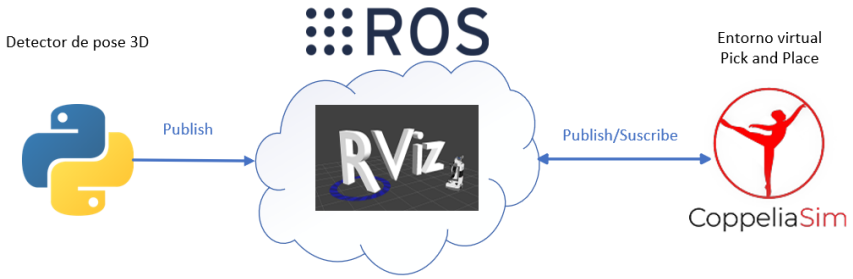
La arquitectura del sistema virtual, ver Figura 10, se completa con la comunicación entre el sistema de visión artificial y la aplicación robótica. Desde la arquitectura de control distribuido que maneja CoppeliaSim se abren las opciones para integrar el detector de pose 3D en la escena, ese es el caso del meta sistema operativo ROS que anida una gran de cantidad de herramientas 2D y 3D para integrar estas dos etapas.

ROS con su enfoque en la robótica, utiliza una distribución de procesos llamados nodos para individualizar los procesos y acoplarlos en otros, estas características permiten que los servicios sean compartidos e intercambiables. Al ser de código abierto la comunidad de ROS distribuye su documentación a la integración con OpenCV, con sus herramientas de reproducción de código y visualizadores 3D, tal es el caso de RViz. (Open Robotics, 2021)

La Figura 74 muestra la estructura empleada para integrar el sistema de detección y CoppeliaSim.

Figura 74

Estructura de comunicación entre sistemas



Los resultados del detector de pose 3D lo convierten en Publicador y dado que son proyectados en ROS por la herramienta de visualización RViz a este lo convierte en Suscriptor. La información manejada por estas conexiones llamadas *topics* son usados por los nodos para publicar mensajes sobre estos. A continuación, se detalla los canales que publica ROS al integrar el detector de pose 3D con RVIZ:

- “/dope/belief_envase”, mapas de creencia del objeto
- “/dope/dimensión_envase”, dimensiones del objeto
- “/dope/pose_envase”, pose del objeto por segundo
- “/dope/rgb_points”, muestra los cuboides detectados sobre la imagen RGB
- “/dope/markers”, superponer un cuboide o malla en el espacio 3D

En la siguiente etapa el algoritmo de control en la aplicación robótica pick and place se suscribe al canal “/dope/pose_envase” para obtener los datos de posición y traslación; sin embargo, CoppeliaSim se convierte en Publicador de los canales del sensor de visión para la suscripción de RViz. Los canales involucrados se describen a continuación:

- “/image”, imagen RGB del sensor de visión en la escena
- “/image/camera_info”, parámetros intrínsecos generados por el script

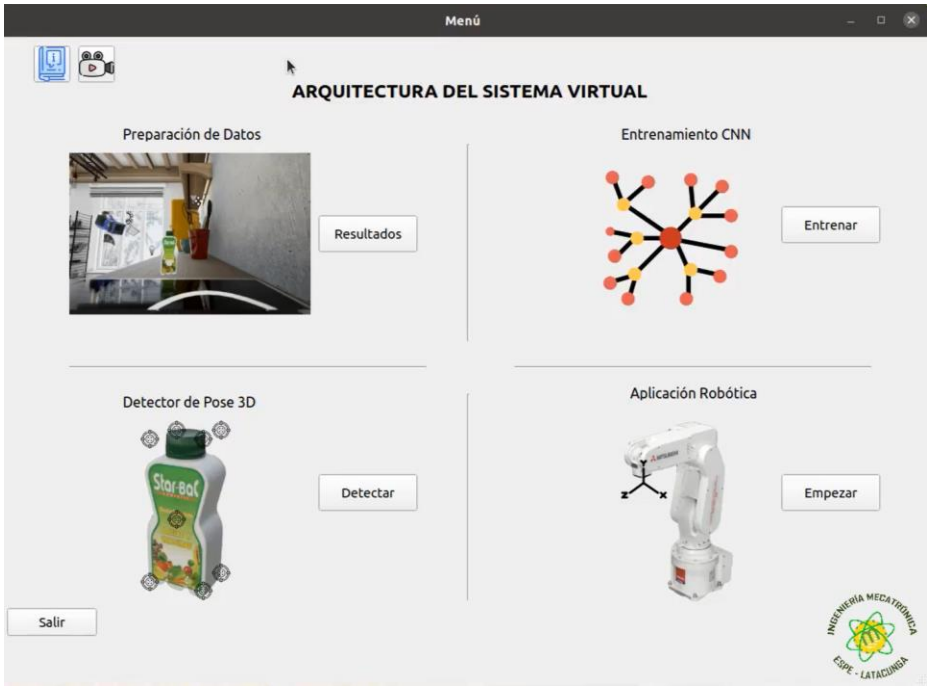
En el caso de la ejecución con webcam los canales son los siguientes:

- “/usb_cam/image_raw”, imagen RGB de la webcam
- “/usb_cam/camera_info”, parámetros intrínsecos obtenidos de la calibración de cámara

Finalmente, la arquitectura del sistema virtual se enlaza a una interfaz gráfica desarrollada en PyQt5 desde el compilador PyCharm, esta distribución se muestra en la Figura 75. La distribución completa de las ventanas se muestra en el Anexo G.

Figura 75

Interfaz gráfica de usuario



Capítulo V

5. Pruebas y resultados

En el presente capítulo se detallan las pruebas de entrenamiento y validación del conjunto de datos sintéticos, junto a la implementación dentro del entorno virtual en CoppeliaSim y dispositivo webcam, para el análisis de mapas de creencia y resultados de posición como de orientación generados por el modelo CNN; con el fin el de validar la hipótesis del proyecto.

5.1. Entrenamiento y validación del conjunto de datos

Para encontrar la convergencia de la red CNN en la detección de las características del envase y generar mapas de creencia, en estas pruebas se ha utilizado la GPU añadida en el hardware del ordenador, con capacidad de memoria de 8GB. Estos núcleos gráficos bajo la arquitectura de NVIDIA CUDA y cudNN permiten acelerar el proceso de entrenamiento bajo el framework PyTorch. Se dispone de 4000 imágenes sintéticas tanto de fotorrealismo y aleatorización de dominios, divididas en un conjunto del 80% para entrenamiento (3200 imágenes) y 20% para prueba(600 imágenes) que ingresan a la red en dimensiones de $640x480$ píxeles.

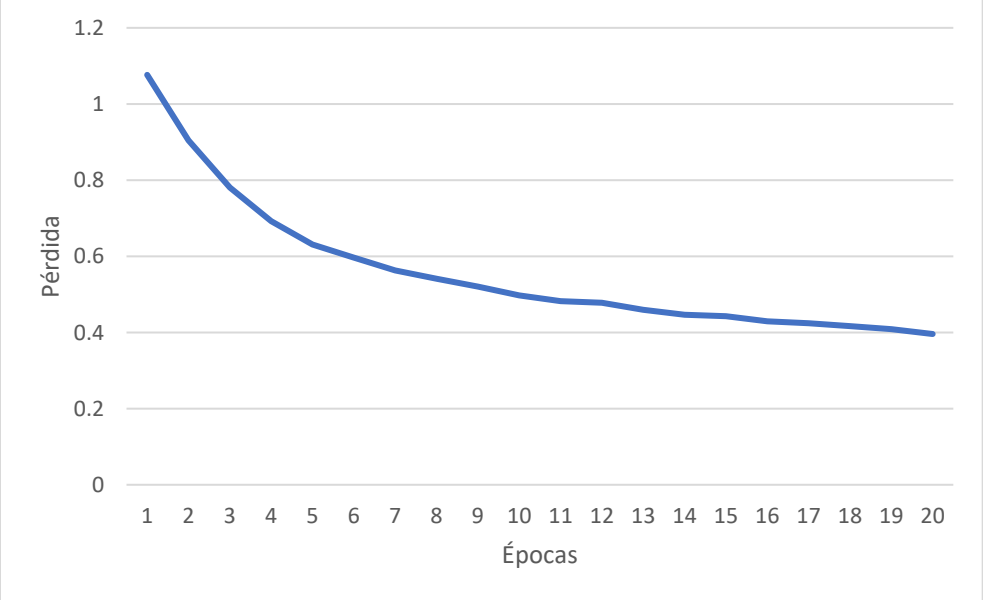
En esta prueba de entrenamiento netamente experimental parte de la generación del código en la ventana de entrenamiento del sistema virtual de la sección 4.2, donde se ubica las carpetas con las imágenes contenedoras previamente divididas, los hiperparámetros establecidos en esta prueba son de 20 épocas, tamaño de lote a 16, taza de aprendizaje 0.0001 destinada a un valor de GPU de 0 (Geforce GTX 1070

de un solo motor disponible). Los datos resultantes están expresados en forma gráfica en la Figura 76.

La prueba frente a un tamaño de lote a 16 no genera resultados de prueba dado que el proceso termina sin memoria gráfica, de finalización en 6 horas. Existe una rápida disminución de la pérdida a 0.396 pero sin datos de prueba es difícil el análisis del comportamiento de las funciones; para corroborar los pesos de salida obtenidos, los mapas de creencia proporcionan información visual e intuitiva del resultado del entrenamiento.

Figura 76

Pérdida de entrenamiento en prueba N°1

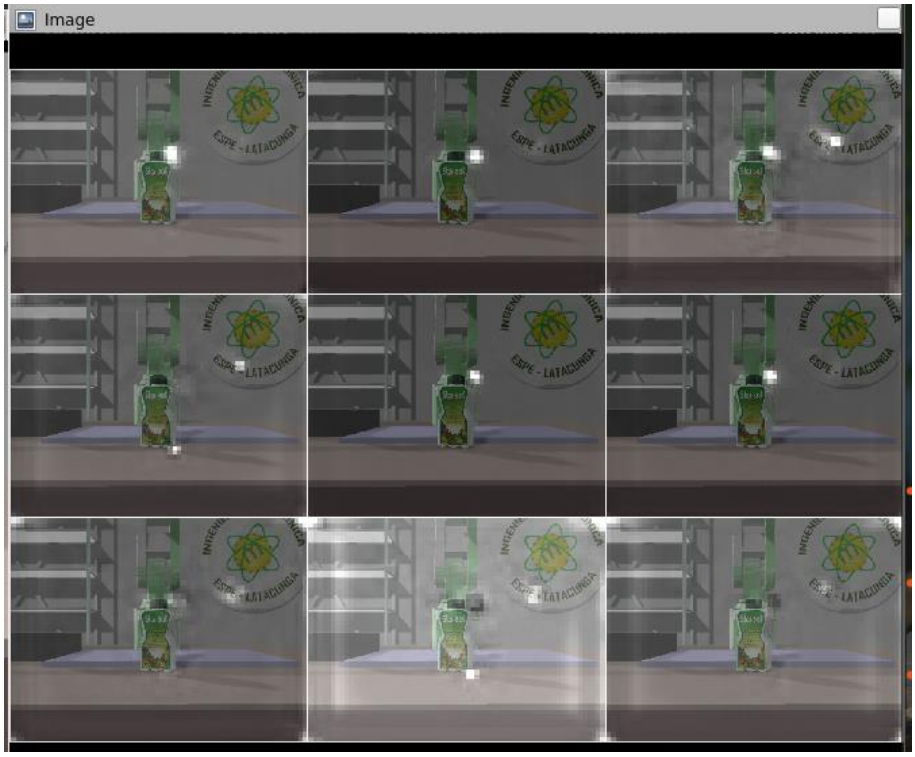


En la Figura 77 se muestra los mapas de creencia generados con el entrenamiento previo, varios de los puntos de correspondencia no están asociados al vértice y se ubican en el vértice número dos del cuboide delimitador. Cómo resultado no

hay un acercamiento a la convergencia de la red y el algoritmo de perspectiva no genera la pose.

Figura 77

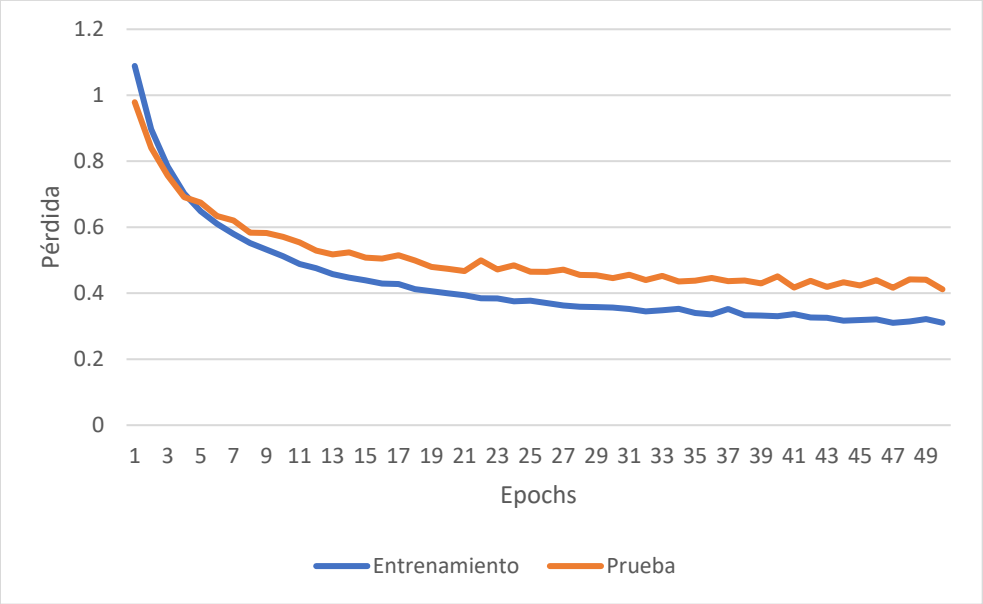
Mapas de creencia Prueba 1



Con las consideraciones de la última prueba se procede a reducir los valores de hiperparámetros con el fin de proveer al procesamiento por GPU de más memoria virtual; los datos divididos en 80-20 entrenamiento y test, tamaño de lote en 8, épocas 50 y manteniendo la taza de aprendizaje a 0.0001, en un tiempo de 12 horas se ha generado los datos que son mostrados en la Figura 78.

Figura 78

Pérdida en prueba de entrenamiento N°2



Los resultados muestran que la pérdida sigue en disminución frente a las 20 épocas del entrenamiento previo, en conjunto con los datos de prueba existe una correlación en unirse mientras continua el aprendizaje. Sin embargo, los datos de prueba están alejados de los de entrenamiento y no está lo suficientemente generalizado para trabajar con diferentes datos por lo que surgen picos en la gráfica, además de presentar una elevación de pérdida con tendencia al sobreaprendizaje.

Esta relación se puede apreciar en los mapas de creencia generados por los pesos de entrenamiento en la Figura 79, el objeto ya es detectado y genera su perspectiva.

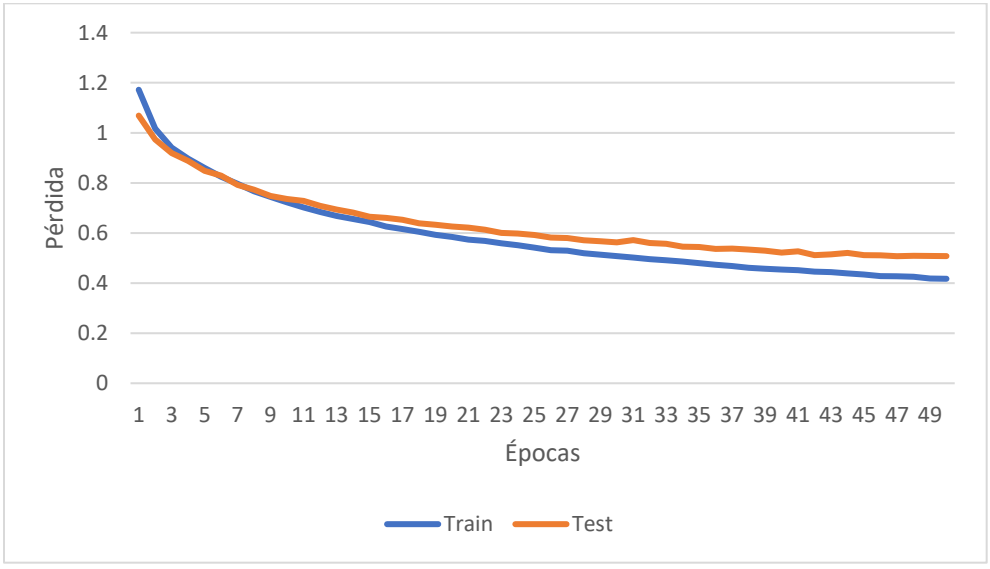
Figura 79

Mapas de creencia de entrenamiento N°2

En el siguiente entrenamiento se ha mantenido las 4000 imágenes divididas en 80-20 para entrenamiento y prueba con tamaño lote de 8, épocas 50, taza de aprendizaje a 0.00001. En busca de un suavizado de la gráfica, con duración de 28 horas los resultados se muestran en la Figura 80.

Figura 80

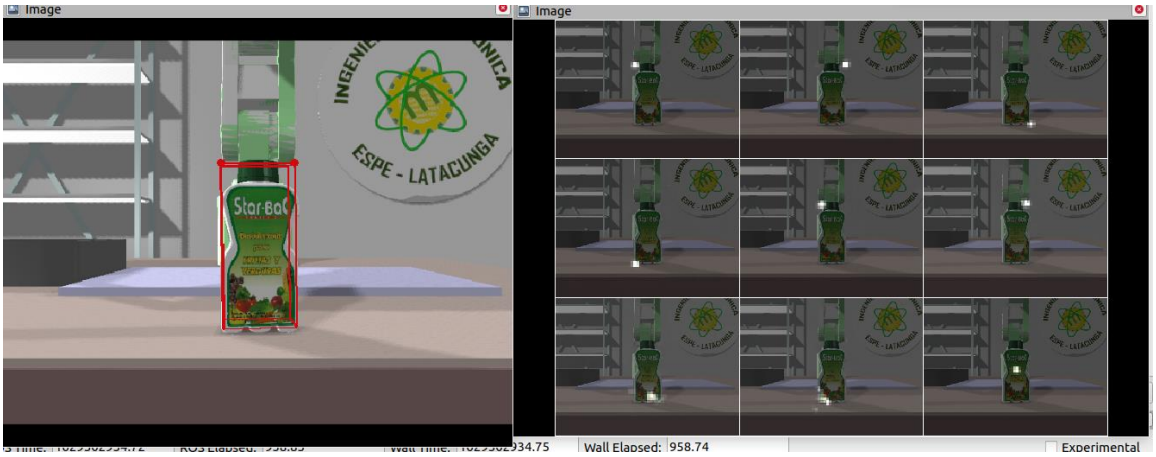
Pérdida en prueba de entrenamiento N°3



Las gráficas de entrenamiento y prueba se mantienen juntas desde el inicio eventualmente están tratando de unirse, los datos de prueba se han generalizado mejor mostrando una curva más suave durante el traspaso de épocas y hay una reducción en la pérdida de igual forma. Con respecto a los mapas de creencia, mostrados en la Figura 81, se aprecia cada uno de los puntos y genera la perspectiva sobre estos.

Figura 81

Mapas de creencia de entrenamiento N°3



5.2. Pruebas de detección de la pose 3D

Con el fin de encontrar el error generado por la estimación de traslación del objeto sobre el entorno virtual en CoppeliaSim, se realiza una toma de datos en 25 pruebas cambiando la traslación del objeto en sus posiciones X,Y,Z. Para describir la relación de estos valores se utiliza el cálculo de la distancia euclidiana, un número positivo que indica la separación que tienen dos puntos en el espacio, mostrada en la Ecuación 8.

$$Distancia\ Euclidiana = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2} \quad (7)$$

La distancia a calcular representa la separación entre el envase y el sensor de visión dentro del entorno virtual de CoppeliaSim, calculada con la Ecuación 8. Expresados estos valores en la Tabla 20 en conjunto del error generado.

$$Error\ Porcentual\ \% = \frac{|Valor\ Real - Valor\ Estimado|}{Valor\ Real} * 100 \quad (8)$$

Tabla 20

Resultado de error en detección de traslación

N° Prueba	Distancia Euclidiana Real (cm)	Ubicación Euclidiana Estimada (cm)	Error absoluto
1	46.0800575	45.79581778	0.28423972
2	53.7303536	55.28395098	1.55359737
3	56.2987646	57.88133662	1.58257198
4	56.0775534	57.16241311	1.08485967
5	56.9073106	57.53517261	0.62786201
6	43.8009326	42.96707563	0.83385701
7	51.0532242	50.43894858	0.61427561
8	51.9690456	51.61407134	0.35497425
9	52.5093487	52.40253965	0.10680904
10	51.5328216	50.97582633	0.55699526
11	53.139267	53.38849059	0.24922356
12	55.5329812	57.12593167	1.59295048
13	56.2828748	57.82027136	1.53739652
14	54.9305197	56.32038916	1.38986941
15	54.0176045	55.01153242	0.99392788
16	54.7996578	55.79085528	0.99119743
17	50.7689049	51.21914147	0.45023662
18	46.6138563	46.80091476	0.18705845
19	46.1955463	44.71118015	1.48436618
20	46.7309084	46.78646007	0.05555167
21	53.200407	54.88482669	1.68441974
22	48.5700584	49.68708223	1.11702386
23	48.6327647	50.14344047	1.5106758
24	53.2577666	53.56929673	0.31153016
25	58.0810778	59.14799413	1.06691631
	Error Porcentual	1.6947 %	

De igual manera se presenta los resultados de error en cálculos de orientación del envase en relación con el sistema de coordenadas de cámara. La Tabla 21 expresa las rotaciones en ángulos de Euler de las veinte y cinco pruebas generadas previamente.

Tabla 21

Resultado de detección de orientación vs real

	Orientación estimada			**Orientación real**		
N° Prueba	**Alpha α**	**Beta α**	**Gamma γ**	**Alpha α**	**Beta α**	**Gamma γ**
1	1.0574	1.0039	35.4751	1	1	34.58
2	1.0711	1.0033	34.9586	1	1	34.52
3	1.0095	1.0242	35.301	1	1	34.53
4	1.0398	1.0326	36.2969	1	1	34.49
5	1.0413	1.0305	36.2856	1	1	34.43
6	1.0543	1.0697	36.4105	1	1	34.36
7	1.0851	1.0671	34.5429	1	1	34.3
8	1.0262	1.0772	35.7444	1	1	34.32
9	1.0245	1.0018	35.1373	1	1	34.24
10	1.0487	1.0727	-48.8133	1	1	-55.74
11	1.0267	1.0572	-63.6816	1	1	-55.74
12	1.0269	1.0624	-64.8691	1	1	-55.76
13	1.0235	1.0631	-63.3875	1	1	-55.77
14	1.09425	1.0140	136.9141	1	1	149.41
15	1.0287	1.0457	144.74	1	1	149.4
16	1.0351	1.0321	137.658	1	1	149.4
17	1.0766	1.0424	139.6376	1	1	149.3
18	1.0845	1.0296	100.5156	1	1	109.82
19	1.0257	1.0256	15.168	1	1	16.254
20	1.0366	1.0367	63.254	1	1	65.02
21	1.0235	1.0168	45.179	1	1	50.2665
22	1.0056	1.0987	55.135	1	1	56.462
23	1.0223	1.0251	12.2122	1	1	14.35
24	1.0512	1.0068	102.21	1	1	100.89
25	1.0069	1.0201	48.264	1	1	50.26

El error porcentual generado para cada prueba se detalla en la Tabla 22, con el promedio resultante de cada ángulo de Euler.

Tabla 22

Resultado de error porcentual de orientación

N° Prueba	Alpha α	Beta α	Gamma γ
1	5.74	0.3969	2.58849
2	7.11	0.332	1.27056
3	0.951	2.421	2.23284
4	3.98	3.26	5.23890
5	4.138	3.05	5.38948
6	5.43	6.977	5.96769
7	8.51	6.7199	0.70816
8	2.625	7.726	4.15034
9	2.45	0.1854	2.62061
10	4.871	7.277	-12.4268
11	2.675	5.728	-14.2475
12	2.6995	6.244	-16.3362
13	2.35	6.314	-13.6587
14	9.425	1.408	8.36349
15	2.87	4.57	3.11914
16	3.515	3.217	7.85943
17	7.66	4.24	6.47180
18	8.45	2.961	8.47240
19	2.571	2.56	6.68143
20	3.664	3.679	2.71608
21	2.354	1.681	10.1210
22	0.568	9.87	2.35025
23	2.235	2.516	14.8975
24	5.12	0.681	1.30835
25	0.69	2.01	3.97134
Promedio %	4.106	3.8409	1.9932

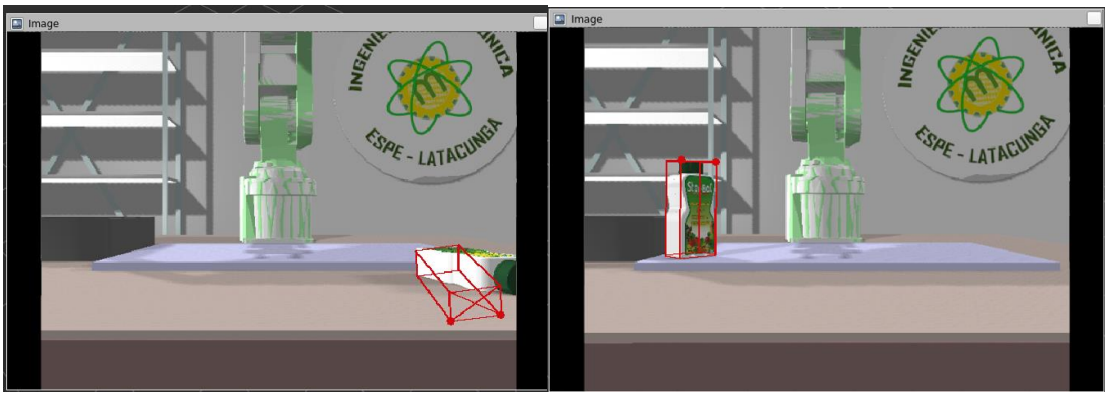
Como se evidencia en la Tabla 20 el error porcentual representa el 1.69 % en el conjunto de pruebas, este resultado varía principalmente sobre el eje Z del sistema de referencia al generar la perspectiva a distancias mayores de los 60 cm. Aun cuando el objeto se encuentre en el área de trabajo del manipulador, la acción no podrá ser completada por los datos erróneos de perspectiva, las líneas de proyección se vuelven paralelas difícilmente detectables para el algoritmo PnP.

A esto se acompaña el análisis de la Tabla 22 con un error porcentual bajo sobre gamma del 1.993%, dado que el giro del envase sobre el eje Z no presenta oclusiones

al objeto, se logra validar varios puntos bien definidos por estancia. Por otro lado, los giros sobre el ángulo beta no permite la detección de vértices suficientes y de existir las proyecciones se ven afectadas con un cuboide delimitador defectuoso, ver Figura 82.

Figura 82

Proyección sobre orientación



Nota. Proyección errónea y buena proyección continua

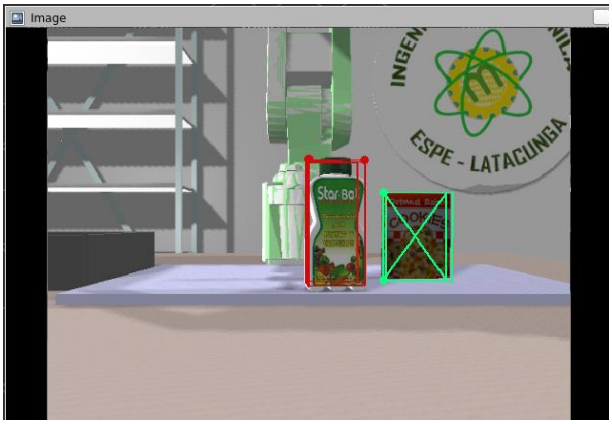
5.3. Pruebas sobre objeto pre-entrenado

En el estudio de aprendizaje profundo muchos investigadores han desarrollado base de datos con objetos domésticos para propósitos del estudio de redes neuronales en general. Es por eso, que se aplica la detección de pose 3D al objeto llamado *cracker*, al que se denomina caja de galletas en este proyecto. El modelo forma parte del grupo “YCB models” compartido en la red (Tremblay, YCB Models, 2021) de una variedad de veinte productos comerciales con los archivos de imágenes en 3D texturizados, además de incluir los pesos respectivos de entrenamiento con más de 16000 imágenes en 15 escenarios diferentes.

Con el nuevo objeto puesto en escena como se muestra en la Figura 83 se realiza la detección de pose 3D en la escena virtual compartida con el envase.

Figura 83

Caja de galleta en escena virtual



Los resultados de detección en la escena virtual están descritos por la distancia euclidiana por relacionar las posiciones de los tres ejes coordenados, la Tabla 23 resulta de diez pruebas realizadas sobre la caja de galletas.

Tabla 23

Resultado de error porcentual en caja de galleta

N° Prueba	Distancia Euclidiana Real (cm)	Ubicación Euclidiana Estimada (cm)	Error absoluto
1	54.83400222	54.78247458	0.09397024
2	53.42795242	53.52446496	0.18064054
3	51.84480109	50.84711416	1.92437218
4	57.84793341	58.6829941	1.44354455
5	46.95299245	47.34731485	0.83982379
6	64.12007018	64.8033055	1.0655561
7	57.80672193	58.31234624	0.87468083
8	50.8922106	50.67148238	0.43371711
9	59.20261987	60.03168745	1.40039001

N° Prueba	Distancia Euclidiana Real (cm)	Ubicación Euclidiana Estimada (cm)	Error absoluto
10	63.30876874	63.81435919	0.79861046
	Error Porcentual	0.924669 %	

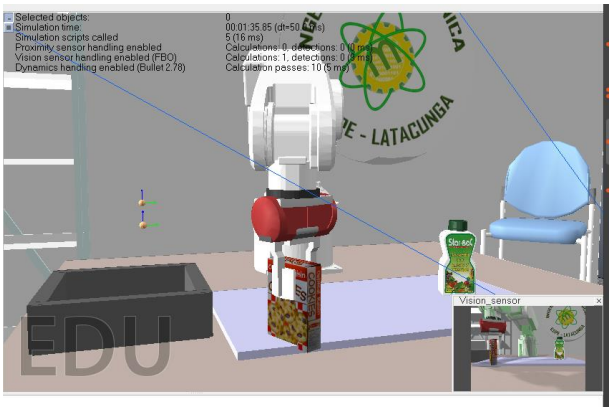
En comparación al error porcentual del envase detectado se describe un valor menor al 1%, esta prueba permite conocer que el sistema virtual puede implementar nuevos modelos pre-entrenados en la escena con buenos valores de detección.

5.4. Acción de aplicación robótica

Al realizarse la detección del objeto tanto de envase como de galleta, el algoritmo de control ejecuta la acción de recoger y colocar en el cuadrado delimitador en la escena. En la Figura 84 se muestra la acción de recoger sobre la caja de galletas, el primer movimiento lo realiza hacia una zona de aproximación de 10 cm sobre el objetivo con un giro de muñeca de 180 grados, así se orienta el lado correcto para el agarre.

Figura 84

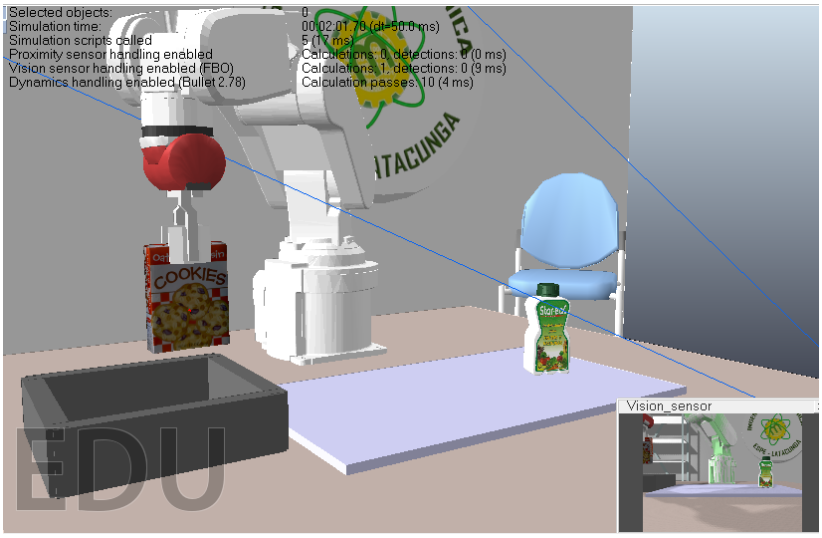
Acción de recoger del manipulador



Luego el manipulador se dirige al punto *home* donde gira el objeto para colocarlo con una vista frontal hacia la cámara y se traslada hacia la posición destino definida por los elementos “Dummy” de CoppeliaSim donde finalmente realiza la acción de colocar.

Figura 85

Acción de colocar del manipulador



La variación de error porcentual se ve reflejado en la manera de ubicar el centroide del objeto, la mayoría de veces está compensada por los 4 cm de carrera que presenta el efector final.

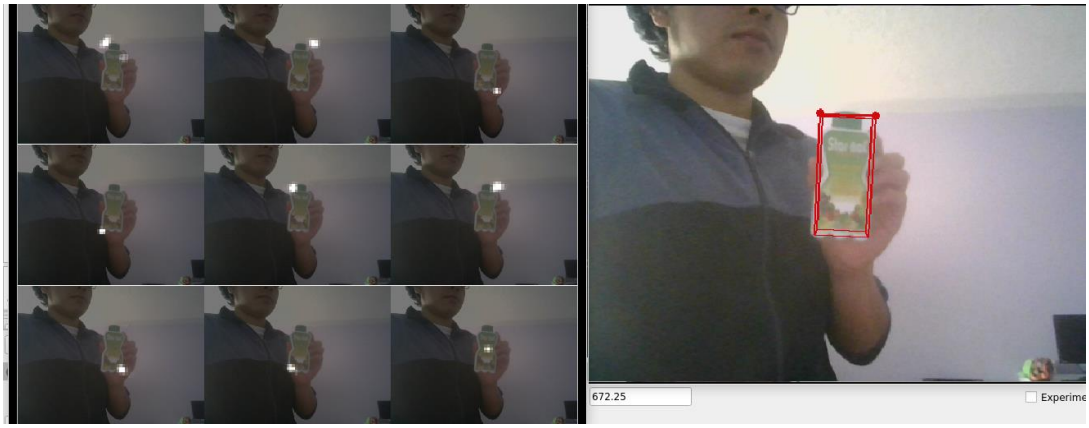
5.5. Pruebas con cámara web en tiempo real

Las imágenes capturadas en ejecución de la cámara web son el destino para esta prueba, realizadas con el envase real se procede a ubicarlo a 50 cm de la cámara para evaluar la detección de vértices y posteriormente la distancia euclidiana desde la cámara.

En la Figura 86 se muestra la detección del envase con una validación de 8 puntos encontrados desde la cara frontal del mismo.

Figura 86

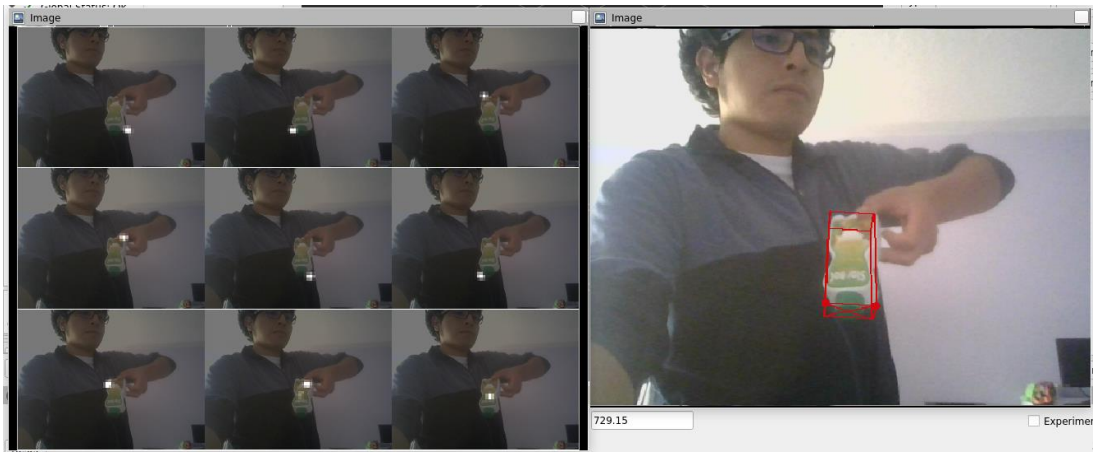
Prueba en cámara web a un envase



En la siguiente postura del envase se gira sobre sus ejes Z y X con objetivo de ver la respuesta frente a las orientaciones, con 8 puntos válidos encontrados en la Figura 87.

Figura 87

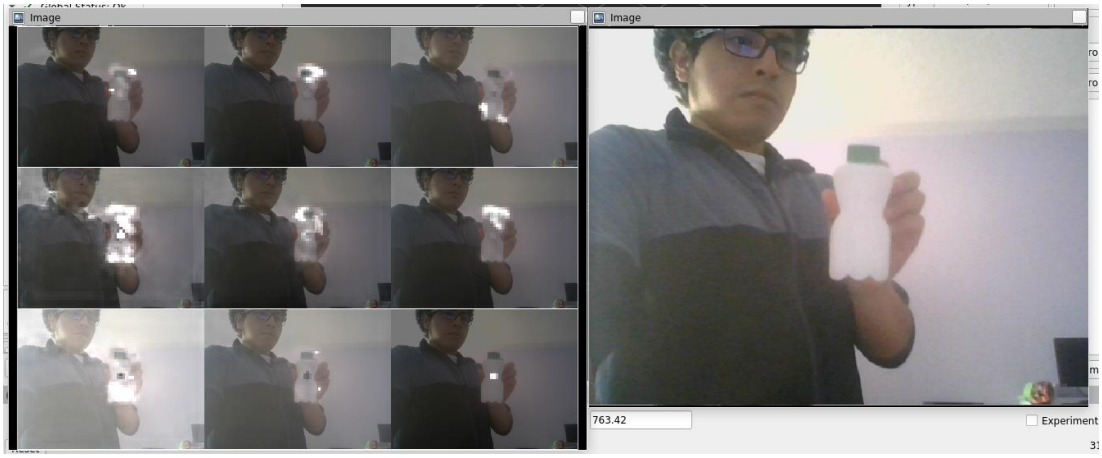
Prueba en cámara web de orientación



Frente al aprendizaje de características de la red se realiza una prueba de detección sobre el mismo objeto sin la etiqueta comercial que lleva integrada. Los resultados que se encuentran se muestran en la Figura 88, donde el detector puede discernir que no corresponde al objeto a buscar, este es el caso con 1 punto válido encontrado (centroide).

Figura 88

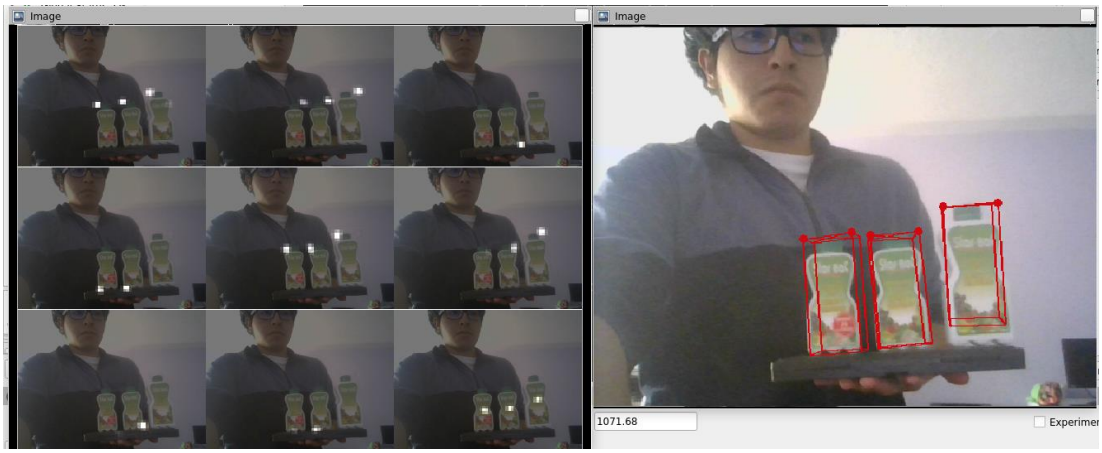
Prueba en cámara web de relación



Por último, en la Figura 89 al integrar más de un objeto en la escena del detector, lo realiza sin problema dado que al separar los cuboides delimitadores se asocian los mapas de afinidad generados. Sin embargo, el tercer objeto de escala 1.2 mayor sobre el modelo realizado en este proyecto no encaja en la proyección dada por el método, ya que el algoritmo PnP está en función de las dimensiones del objeto.

Figura 89

Detección multiclase



El error porcentual generado por la red se encuentra en relación de la distancia euclidiana desde la cámara web hacia el objeto, los resultados están detallados en la Tabla 24.

Tabla 24

Error porcentual estimación sobre cámara web

Distancia euclidiana (cm)	Error porcentual
51.243052	2.486104
51.58948	3.17896
51.959	3.918
52.41839	4.83678
50.7576	1.5152
52.0051	4.0102
51.8965	3.793
51.23265	2.4653
50.98231	1.96462
52.68035	5.3607
Error porcentual promedio	3.3528864

El error generado se produce por el error de proyección del detector y de los parámetros de lente en la calibración, a pesar de encontrar varios puntos suficientes, los centros ópticos y distancias focales están alejadas de la verdad.

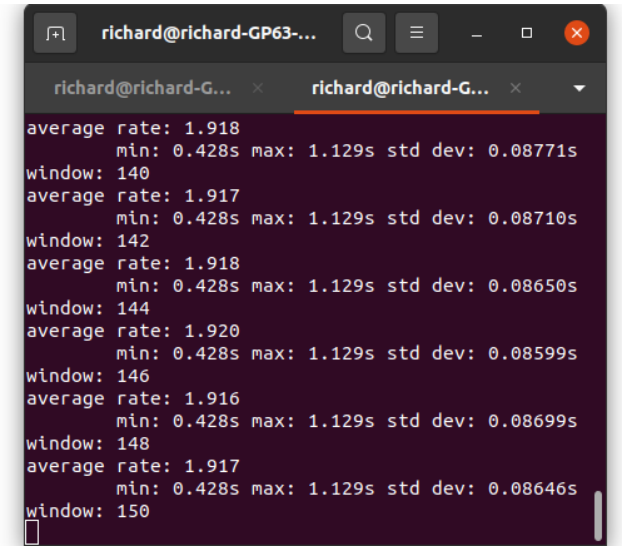
5.6. Tiempo de respuesta entre módulos

Con el propósito de encontrar el tiempo de envío de mensajes entre la imagen del sensor de visión en CoppeliaSim hacia el detector de pose, se evalúa con los comandos de ROS el informe generado por los canales de información. RVIZ

El topic “/dope/pose_envase” continuamente proyecta la imagen del entorno virtual a fin de detectar el objeto y proyectar sobre este el cuboide delimitador. Con la línea de comando de ROS “rostopic hz” sobre el topic se encuentra la taza de refresco de los fotogramas en el intercambio de información de las etapas se visualiza en la Figura 90.

Figura 90

Taza de refresco



Desde la cámara web hacia el simulador RVIZ se realiza la detección en un promedio de 1.918 Hz, es decir se recibe mensajes cada 0.5213 segundos.

5.7. Pruebas de usabilidad

Uno de los objetivos principales a resolver dentro de la problemática del proyecto es el de brindar una herramienta a los estudiantes en busca del desarrollo de prácticas de tipo académico-práctico, por lo que se realiza una prueba de usabilidad para evidenciar cuán satisfactorio es la interacción de un sistema de información al incorporarse dentro de la actividad humana.

Los participantes de esta prueba son treinta y tres alumnos de la carrera de Mecatrónica de la asignatura de “Automatización Industrial Mecatrónica”, considerados idóneos por el autor del proyecto dada una correlación del curso con los temas investigados. Dada la presente situación del país referente al Covid-19 se presenta la dificultad de realizar una interacción directa del sistema virtual con el usuario, de tal forma, se ha considerado en el modelo de SUS (Escala de Usabilidad del Sistema) un cuestionario de diez interrogantes para evaluar la interfaz del sistema virtual en la detección de pose 3D en la aplicación robótica. El cuestionario se encuentra descrito en el Anexo H.

En la Tabla 25 se muestra los resultados promedios de usabilidad con sus respectivas preguntas, es importante aclarar que la escala de números utilizados va de 1 a 5 definidos por “en desacuerdo” y “de acuerdo” respectivamente.

Tabla 25

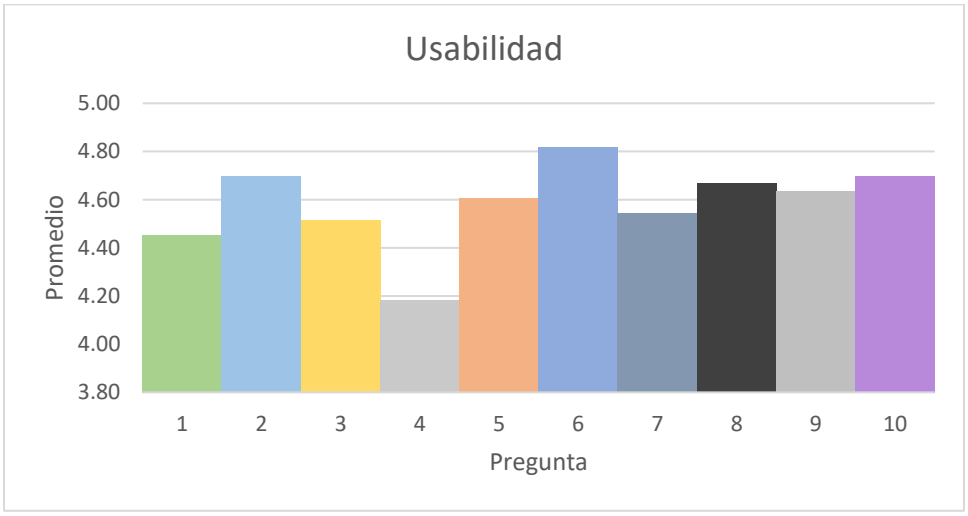
Promedio de preguntas de usabilidad

N°	Pregunta	Promedio
1	La información de las pantallas es clara y eficiente para su comprensión	4.45
2	Los colores empleados respetan su comodidad visual	4.70
3	La elección de menús es presentada con botones claros y accesibles	4.52
4	El programa permite moverse con facilidad entre etapas	4.18
5	La calidad técnica de los videos y manuales te han parecido adecuados	4.61
6	La información brindada resulta nueva y de interés	4.82
7	No se han presentado errores en la aplicación	4.55
8	El sistema virtual se adaptaría a sus necesidades en asignaturas de niveles superiores	4.67
9	Recomendaría el sistema virtual a otros estudiantes	4.64
10	Indique el grado de satisfacción con el sistema virtual en general	4.70
	Promedio total	4.58

Estos datos se expresan en el histograma de la Figura 91 para su posterior análisis del promedio de cada pregunta.

Figura 91

Histograma promedios usabilidad



Del histograma de la Figura 89 se puede obtener que de la pregunta 4 (El programa permite moverse con facilidad entre etapas) obtuvo la calificación más baja de 4,18 lo que significa una dificultad de cambio entre etapas debido a la cantidad de procesos en ejecución. Mientras en la pregunta 6 (La información brindada resulta nueva y de interés) obtuvo un valor alto de 4,82 con lo que se demuestra que existe un interés sobre los temas de aprendizaje profundo, robótica y en general del sistema virtual por el grado de satisfacción mostrada.

5.8. Validación de la hipótesis

La hipótesis planteada en la sección 1.1.5 es:

¿El diseño e implementación de un sistema virtual basado en técnicas de aprendizaje profundo ayudará a la detección de la pose 3D de un objeto para aplicaciones robóticas?

Con las que se obtiene las variables de investigación:

- **Variable dependiente**

Detección de la pose 3D de un objeto mediante técnicas de aprendizaje profundo

- **Variable independiente**

Sistema virtual para aplicaciones robóticas

Dada la situación del país en tema de salud para la presente fecha de realización del proyecto, el autor se ve limitado a realizar pruebas para poder validad o rechazar la hipótesis con estudiantes afines al tema. Por tal razón, se ve forzado a una comprobación del funcionamiento en gran medida de los datos obtenidos por pruebas dentro del sistema virtual.

La hipótesis del sistema virtual es comprobada por el método de Chi-cuadrado por lo que se define la hipótesis nula y la hipótesis alternativa.

- **Hipótesis nula (H_o):** El diseño e implementación de un sistema virtual basado en técnicas de aprendizaje profundo no ayudará a la detección de la pose 3D de un objeto para aplicaciones robóticas
- **Hipótesis alternativa (H_i):** El diseño e implementación de un sistema virtual basado en técnicas de aprendizaje profundo ayudará a la detección de la pose 3D de un objeto para aplicaciones robóticas

Los datos experimentales que conforman la Tabla 26 de frecuencias observadas relacionan si las pruebas de detección de pose 3D realizadas con el objeto personalizado realizan la acción de pick and place basado en técnicas de aprendizaje profundo en relación a la caja de galletas pre-entrenada.

Tabla 26

Frecuencia observada

Aplicación robótica ejecutada	Envase	Caja de galletas	TOTAL
Si	24	29	53
No	6	1	7
TOTAL	30	30	60

Para obtener los argumentos de frecuencias teóricas esperadas los datos de la Tabla 26 se calcula con la Ecuación 9, que corresponde a los valores en caso de que la hipótesis nula sería verdadera.

$$E_i = \frac{(Total\ de\ Columna)(Total\ de\ Fila)}{Suma\ Total} \quad (9)$$

Las frecuencias esperadas calculadas se muestran en su espacio correspondiente en la Tabla 27.

Tabla 27

Tabla de frecuencias teóricas

Aplicación robótica ejecutada	Envase	Caja de galletas
Si	26.5	26.5
No	3.5	3.5

Para determinar el grado de libertad, relacionamos el número de filas y columnas tal como se muestra en la Ecuación 10.

$$v = (columnas - 1)(filas - 1)$$
$$v = (2 - 1)(2 - 1) = 1 \quad (10)$$

Con un margen de error del 5% (0.05) se realiza el cálculo el chi cuadrado denotado por la Ecuación 11.

$$x^2 = \sum \frac{(fo - E_i)^2}{Ei} \quad (11)$$

$$x^2 = \frac{(24-26.5)^2}{26.5} + \frac{(29-26.5)^2}{26.5} + \frac{(6-3.5)^2}{3.5} + \frac{(1-3.5)^2}{3.5}$$

$$x^2 = 4.043 \qquad (12)$$

Dado el resultado de la ecuación se compara con la tabla del Anexo I sobre la distribución de Chi-cuadrado, se obtiene la Ecuación 13:

$$x_{teórico}{}^2 = 4.043 \qquad (13)$$

Se dice que la hipótesis nula se rechaza si se cumple que el valor calculado de Chi-cuadrado es mayor al Chi-cuadrado teórico.

$$x_{calculado}{}^2 > x_{teórico}{}^2 \qquad (13)$$

$$4.043 \geq 3.841 \qquad (14)$$

Dado el resultado de la Ecuación 14 se rechaza la hipótesis nula, con lo que resulta en una afirmación que el diseño e implementación de un sistema virtual basado en técnicas de aprendizaje profundo ayudará a la detección de la pose 3D de un objeto para aplicaciones robóticas.

Capítulo VI

6. Conclusiones y recomendaciones

6.1. Conclusiones

- En este trabajo se diseñó e implementó un sistema virtual para la detección de la pose 3D mediante una arquitectura basada en redes neuronales convolucionales, en un conjunto de datos sintéticos fotorrealistas y aleatorización de dominios de un objeto personalizado, hacia un entorno virtual para el desarrollo de la aplicación robótica de recoger y colocar; la etapa más importante fue establecer la comunicación entre sistemas y la generación de la interfaz gráfica de usuario de los procesos vinculados.
- Para el diseño de algoritmo de detección se partió de la recopilación de información y conceptos sobre el aprendizaje profundo el cual permite un nivel de abstracción mayor de característica por las convoluciones realizadas en las capas ocultas; además de realizarse una búsqueda de las aplicaciones robóticas que afrontan el problema de pose 3D siendo la navegación, realidad aumentada y en robots fijos la manipulación de objetos, se determinó esta acción por la versatilidad en definir las acciones de control.
- La generación de 4000 imágenes sintéticas en ambientes fotorrealistas y aleatorización de dominio, con una variedad de datos a nivel de renderizado y construcción de escena han demostrado que el entrenamiento únicamente de datos sintéticos brinda buenos resultados en la detección de pose 3D dentro del ambiente virtual de CoppeliaSim como en imágenes del mundo exterior.

- En el desarrollo del algoritmo de detección de posición y orientación se utilizó 9 mapas de creencia y 8 mapas de afinidad para correlacionar las ubicaciones de vértices en 2D hacia los proyectados por el cuboide 3D del envase; en conjunto de la biblioteca OpenCV se desarrolló el algoritmo de perspectiva PnP, conocido los parámetros intrínsecos de cámara y las dimensiones del objeto para predecir su pose.
- El algoritmo de control trabaja bajo la biblioteca Sympy de cálculo simbólico para obtener una representación de las matrices de transformación homogéneas descritas por los parámetros DH del robot Mitsubishi, en el cálculo de la cinemática inversa; esta misma se resuelve por métodos numéricos en tiempo real al recibir las configuraciones de posición y orientación final del objeto, es así que el método presentó dificultades para encontrar puntos válidos en algunos casos.
- La comunicación entre etapas del sistema virtual se realizó mediante la integración del meta sistema operativo ROS, con una taza de refresco cada 0.5213 segundos se establece la comunicación de datos a través de publicadores y suscriptores.
- Las pruebas de funcionamiento del sistema virtual se realizaron por medio de ubicaciones al azar dentro del alcance máximo del manipulador. El resultado de 25 pruebas arrojó un error porcentual de 1.6947% en traslación medida por la distancia euclidiana de la cámara hacia el envase, mientras en orientación un error promedio porcentual de 3.0136% de todos los ejes, siendo 1.99% el valor menor sobre el ángulo gamma; estos resultados frente al objeto pre-entrenado de error porcentual 0.924%. Esto se ve influenciado por la distancia que existe entre ambos elementos, obteniendo datos erróneos si la distancia es mayor a

50-60 cm o una orientación que no permita el agarre del objeto; por tal razón la aplicación robótica no se completa aun cuando se encuentre en una ubicación libre de singularidades.

- El test de usabilidad aplicado al grupo de estudiantes determinó un alto grado de satisfacción del sistema virtual sobre la interfaz gráfica desarrollada al abarcar temas nuevos y gran interés para aplicaciones en niveles superiores con una puntación general de 4.58 sobre 5 puntos; además la funcionalidad se ha realizado con el estadístico Chi-cuadrado donde el resultado calculado $4.043 \geq 3.841$ validó la hipótesis planteada sobre el valor teórico.
- La implementación del sistema virtual integra en su totalidad herramientas de software libre como Blender 3D, Unreal Engine 4, CoppeliaSim Edu, Python con bibliotecas Sympy y ROS; con el propósito de dotar de flexibilidad y libertad de estudio de los elementos integrados, este conocimiento de nuevas tecnologías ayuda a complementar el aprendizaje individual y colectivo.

6.2. Recomendaciones

- Se debe mantener en consideración al momento de la generación de modelos personalizados, que los objetos que muestran características transparentes generan confusión a la red y son difíciles de extraer características dada la pequeña reflectancia que produce la luz al llegar a su superficie.
- Dentro del generador de imágenes sintéticas es preciso que las imágenes capturadas muestren visibles los vértices del cuboide limitador, dado que la validación del modelo no relaciona oclusiones entre objetos.

- En caso de ver una limitación en hardware de cálculo computacional es recomendable el uso de plataformas de entrenamiento como Google Colaboratory, este permite la escritura y ejecución de código Python en Jupyter Notebook alojado en el navegador; además, de alojar los datos sobre Google Drive para el entrenamiento remoto en GPUs de mayor capacidad de memoria gráfica.
- Evite las orientaciones que produzcan el bloque de Cargan en ángulos de Euler, ocasionan un efecto contraproducente en la ejecución de la aplicación robótica; se recomienda mantener el trabajo de ángulos por cuaterniones entregado por el detector de pose 3D.
- En virtud del sistema desarrollado, se sugiere la exploración de otros entornos de simulación como Unreal Engine, Webots, Gazebo para brindar más herramientas de prueba y puesta en escena de estos modelos de aprendizaje profundo.
- Para un acercamiento más práctico del proyecto, se sugiere llevar una adquisición de datos personalizada e integrar la detección real de la pose 3D con el Robot Mitsubishi; enfocada a robots fijos y móviles en aplicaciones que abarcan este problema.
- Explorar el tema de contenedores e hilos en Python para unificar los procesos que intervienen en el sistema virtual, esto con el propósito de evitar el ingreso manual de líneas de código al terminal.

Bibliografía

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., . . . Zhen, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of OSDI '16: 12th USENIX Symposium on Operating Systems Design and Implementation*, 265-283. Obtenido de https://www.usenix.org/sites/default/files/osdi16_full_proceedings.pdf

Alcon Lighting. (2021). *Architectural Lighting Solutions*. Obtenido de https://www.alconlighting.com/

Aura Prods. (s.f.). *Pack de texturas para blender*. Obtenido de https://auraprods.com/texturas-gratis-blender/

Autodesk Inc. (09 de Junio de 2021). *System requirements for Autodesk Fusion 360*. Obtenido de https://knowledge.autodesk.com/support/fusion-360/troubleshooting/caas/sfdcarticles/sfdcarticles/System-requirements-for-Autodesk-Fusion-360.html

Barrientos, A., Peñin, L., Balaguer, C., & Aracil, R. (2005). *Fundamentos de Robótica.* Madrid: McGraw-Hill.

Basogain, X. (2001). *Redes Neuronales Artificiales y sus aplicaciones.* Bilbao: Xabier Basogain Olabe.

Beane, A. (2012). *3D Animation Essentials.* Indiana: John Wiley & Sons,Inc.

Berzal, F. (2018). *Redes Neuronales & Deep Learning.* Granada: Fernando Berzal.

Blender. (s.f.). *Blender 2.80 Manual*. Obtenido de https://docs.blender.org/manual/en/2.80/addons/index.html#

Blender. (s.f.). *Requirements*. Obtenido de https://www.blender.org/download/requirements/

Córdoba, M. (2011). *Formulación y Evaluación de Proyectos.* Bogotá: ECOE Ediciones.

Cyberbotics Ltd. (16 de Julio de 2021). *Webots*. Obtenido de https://cyberbotics.com/

Davis, N. (08 de Septiembre de 2020). *Unity Blog: Toyota makes mixed reality magic with Unity and Microsoft HoloLens 2.* Obtenido de https://blog.unity.com/manufacturing/toyota-makes-mixed-reality-magic-with-unity-and-microsoft-hololens-2

Delgado, M. (2019). *Desarrollo de un sistema de detección y predicción de la pose 3D de objetos en la escena mediante técnicas de Deep Learning[Tesis de Pregrado,Universidad de Alicante].* España. Obtenido de https://rua.ua.es/dspace/bitstream/10045/100915/1/Desarrollo_de_un_sistema_de_deteccion_y_prediccion_d_Delgado_Marti_Alejandro.pdf

Dhall, A., Dai, D. D., & Van Gool, L. (2019). Real-time 3D Traffic Cone Detection for Autonomous Driving. *2019 IEEE Intelligent Vehicles Symposium (IV)*, 494-501. doi:10.1109/IVS.2019.8814089

Dogra, S. (2020). *Autodesk Fusion 360: A Power Guide for Beginners and Intermediate Users (4th Edition).* CADArtifex.

Dot CSV. (12 de Noviembre de 2020). *¡APRENDE Qué son las Redes Neuronales CONVOLUCIONALES!* Obtenido de [Video de YouTube]: https://youtu.be/V8j1oENVz00

Electric Mitsubishi. (2010). *RV-2SDB/RV-2SQB Series.* Alemania: Electric Mitsubishi. Obtenido de http://suport.siriustrading.ro/02.DocArh/07.RI/03.Seria%20RV%20(Vertical)/05.RV-SD/00.Prospecte,%20cataloage/RV-2SD(Q)B%20-%20Prospect%20234738-A%20(06.10).pdf

Epic Games, Inc. (2021). *Unreal Engine 4 Documentation*. Obtenido de https://docs.unrealengine.com/4.26/en-US/

Fernández, P. (18 de Febrero de 2020). *Hyperhype*. Obtenido de https://www.hyperhype.es/motores-graficos-y-de-juego-definicion-tipos-y-modelos-de-negocio/

Flórez, R., & Fernández, M. J. (2008). *Las redes neuronales artificiales. Fundamentos teóricos y aplicaciones prácticas.* España: Netbiblo.

Gallego, C., Icart, M., & Anna, P. (2006). *Elaboración y presentación de un proyecto de investigación y una tesina.* Barcelona: Edicions Universitat Barcelona.

Gómez, P., & Soria, R. (2020). *Diseño y simulación para la automatización de los procesos de sellado y lacado, en la línea de fabricación de dormitorios de la empresa carpintería y tapicería internacional CTIN. Cia. Ltda.[Tesis de pregrado].* Cuenca: Universidad Politecnica Salesiana de Cuenca. Obtenido de https://dspace.ups.edu.ec/bitstream/123456789/19132/1/UPS-CT008818.pdf

Gulli, A., & Pal, S. (2017). *Deep Learning with Keras[Aprendizaje profundo con Keras].* Birmingham-Mumbai: Packt.

Hanrahan, G. (2011). *Artificial neural networks in biological and environmental analysis.* Boca Raton, FL: CRC Press.

Iglesias, M. (2010). *Algortimos de visión para la estimación robusta de pose 3D [Tesis de pregrado].* Madrid: Universidad Carlos III de Madrid.

Ioana, E. (2019). *Desarrollo de la teleoperación de robots industriales y colaborativos mediante técnicas avanzadas de visión artificial [Tesis de maestría].* Valencia: Universidad Politécnica de Valencia. Obtenido de https://riunet.upv.es/bitstream/handle/10251/130010/Jiva20-20Desarrollo%20de%20la%20teleoperaci%c3%b3n%20de20robots20industriales20y20colaborativos20mediante20tc3a9cnicas20av....pdf?sequence=1isAllowed=y

Kaehler, A., & Bradski, G. (2016). *Learning OpenCV3.* United States of America: O'Reilly Media, Inc.

Karmanov, I., Salvaris, M., Fierro, M., & Danielle, D. (14 de 03 de 2018). *Comparing Deep Learning Frameworks: A Rosetta Stone Approach*. Obtenido de https://docs.microsoft.com/es-es/archive/blogs/machinelearning/comparing-deep-learning-frameworks-a-rosetta-stone-approach

Keras. (s.f.). *Sobre Keras*. Obtenido de https://keras.io/about/#support

Laan, C. (28 de Mazo de 2018). *Laboratorios LAAN*. Obtenido de Real-time 3D car pose estimation trained on synthetic data: https://labs.laan.com/blog/real-time-3d-car-pose-estimation-trained-on-synthetic-data.html

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning Nature. *Nature*, 436–444. doi:https://doi.org/10.1038/nature14539

Leonor, H. (2020). *Predicción y optimización de emisiones y consumo mediante redes neuronales en motores Diesel.* Barcelona: Reverte.

Lien, T. K. (2013). Gripper technologies for food. *Robotics and automation in the food industry*, 164-191. doi:10.1533/9780857095763.1.143

Likhith, S. (23 de Septiembre de 2019). *Which OS to prefer for ML?* Obtenido de https://www.linkedin.com/pulse/which-os-prefer-ml-sailikhith-kanuparthi

López, A., Gómez, M., Sánchez, S., & Juan, M. (2019). *Tecnología de la fabricación: Apuntes de teoría.* España: Universidad de Almería.

Menard, M., & Bryan, W. (2015). *Game Development with Unity.* Boston: CENGAGE Learning.

Micro-Star International Co., Ltd. (2021). *GP63 Leopard (Intel 8th Gen)*. Obtenido de https://us.msi.com/Laptop/GP63-Leopard-Intel-8th-Gen/Specification

Mitsubishi. (2012). *RV-2SD/2SDB Standar Specifications Manual.* Alemania: Mitsubishi Electric. Obtenido de http://www.int76.ru/upload/iblock/261/261b3f0775215ba60104faa5df43dd14.pdf

Mitsubishi Electric Corporation. (13 de Abril de 2010). *CAD Data*. Obtenido de https://www.mitsubishielectric.com/fa/download/cad/index.html

Mobile Robot Programming Toolkit. (2021). *MRPT*. Obtenido de https://docs.mrpt.org/reference/latest/download-mrpt.html

Nikolenko, S. (2019). *Synthetic Data for Deep Learning.* Rusia: Springer Nature.

NVIDIA Developer. (18 de Febrero de 2015). *Deep Learning*. Obtenido de https://developer.nvidia.com/deep-learning

Open Robotics. (13 de Marzo de 2021). *Qué es ROS?* Obtenido de http://wiki.ros.org/es/ROS/Introduccion

Palomino, N. L., & Concha, U. (07 de 2009). Técnicas de Segmentación en Procesamiento Digital de. *Revista de Ingeniería de Sistemas e Informática, 6*(2). Obtenido de https://revistasinvestigacion.unmsm.edu.pe/index.php/sistem/article/download/3299/2749/

Parmar, V. (1 de 10 de 2020). *Comparing GPU performance for Deep Learning between Linux, and Windows*. Obtenido de Medium: https://medium.com/analytics-vidhya/comparing-gpu-performance-for-deep-learning-between-pop-os-ubuntu-and-windows-69aa3973cc1f

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., . . . Chintala, S. (3 de Diciembre de 2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *NeurIPS*, 12. Obtenido de https://arxiv.org/abs/1912.01703

Pavlakos, G., Zhou, X., Chan, A., Derpanis, K., & Daniilidis, K. (2017). 6-DoF Object Pose from Semantic Keypoints. *IEEE International Conference on Robotics and Automation (ICRA)*, 2011-2018.

Pérez, I., & Gegúndez, E. (2021). *Deep Learning.* España: Servicio de Publicaciones de la Universidad de Huelva.

Pérez, I., & Genúndez, E. (2021). *Deep Learning.* España: Servicio de Publicaciones de la Universidad de Huelva.

Pixologic, Inc. (2021). *ZBrush at a Glance*. Obtenido de http://pixologic.com/features/about-zbrush.php

Pixologic, Inc. (2021). *Rendering*. Obtenido de http://docs.pixologic.com/user-guide/materials-lights-rendering/rendering/

Rabulan, J., & Dorado, J. (2006). *Artificial Neural Networks in Real-Life Applications.* Coruña-España: Idea Group Publishing .

Rethink Robotics. (17 de Abril de 2015). *Baxter: User Guide*. Obtenido de mfg.rethinkrobotics.com

Reyes, F. (2011). *Robótica Control de Robots Manipuladores.* México D.F.: Alfaomega.

Rincon, J. (2014). *Desarrollo de Entornos Virtuales Inteligentes Basados en el Meta-Modelo MAM5 [Tesis de maestría].* Valencia: Universidad Politécnica de Valencia.

Rivas, W., & Mazón, B. (2018). *Redes neuronales artificiales aplicadas al reconocimiento de patrones.* Machala: UTMACH.

Rodriguez, C. (2019). *Sistema de detección y estimación de pose de objetos basado en visión por computador para Planta Piloto Industria 4.0 [Tesis de Maestría].* España: Universidad de Oviedo. Obtenido de https://digibuo.uniovi.es/dspace/bitstream/handle/10651/51673/TFM_AserCrespoRdriguez.pdf?sequence=7

Salazar, J. (2019). *Implementación de un prototipo de sistema autónomo de visión artificial para la detección de objetos en video utilizando técnicas de aprendizaje profundo.* Sangolqui: Universidad de las Fuerzas Armadas ESPE .

Shannon, T. (2018). *Unreal Engine 4 for Design Visualization: Developing Stunning Interactive Visualizations, Animations, and Renderings.* Estados Unidos: Addison-Wesley Professional.

Simonyan, K., & Zisserman, A. (10 de Abril de 2015). VERY DEEP CONVOLUTIONAL FOR LARGE-SCALE IMAGE RECOGNITION. *Computer Vision and Pattern Recognition*. Obtenido de https://arxiv.org/pdf/1409.1556.pdf

SSISchaeferES. (22 de Marzo de 2019). *Simulación | Inteligencia Artificial | Intralogística | SSI SCHAEFER*. Obtenido de [Video de Youtube]: https://youtu.be/7n76QA04h2s

Suau, P. (2011). *Manual de modelado y animación con Blender.* España: Universidad de Alicante.

SymPy Development Team. (2021). *SymPy*. Obtenido de https://www.sympy.org/en/index.html

Torres, J. (2018). *Deep Learning Introducción Práctica con Keras.* Barcelona: Jordi Torres.

Torres, J. (2018). *First contact with Deep Learning .* Barcelona: Kindle Direct Publishing.

Tremblay, J., Thang, T., McKay, D., Yamaguchi, Y., Leung, K., Balanon, A., . . . Birchifield, S. (2018). *NDDS:NVIDIA Deep Learning Dataset Synthesizer*. Obtenido de https://github.com/ NVIDIA / Dataset_Synthesizer

Tremblay, J., To, T., Sundaralingam, B., Yu, X., Fox, D., & Birchfield, S. (2018). Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects. *Conference on Robot Learning (CoRL)*. Obtenido de https://arxiv.org/abs/1809.10790

Unity Technologies. (Abril de 2018). *https://docs.unity3d.com/es/2018.4/Manual/*. Obtenido de https://docs.unity3d.com/es/2018.4/Manual/PostProcessingOverview.html

Valbuena, R. (2021). *Inteligencia Artificial: Investigación Científica Avanzada Centrada en Datos.* Venezuela: Cencal Press.

Valderrama, J. (2000). *Información Tecnológica-SUMARIO.* Chile: Editorial del Norte.

Vallez, N., Velasco-Mata, A., Corroto, J. J., & Deniz, O. (2019). Weapon Detection for Particular Scenarios Using Deep Learning. *Pattern Recognition and Image Analysis*, 371-382. doi:10.1007/978-3-030-31321-0

Voulodimos, A., Nikolaos, D., & Protopapadakis, E. (2018). Deep Learning for Computer Vision: A Brief Review. *Computational intelligence and neuroscience*, 13. doi:10.1155/2018/7068349

Wani, M. A., Bhat, F. A., Afzal, S., & Khan, A. I. (2020). *Advances in Deep Learning.* Springer.

Yang, C., Zeng, C., & Jianwei, Z. (2021). *Robot Learning Human Skills and Intelligent Control Design.* Alemania: CRC Press.

Anexos