Escuela Superior Politécnica del Litoral

Facultad de Ingeniería en Mecánica y Ciencias de la Producción

Manipulación Autónoma de Objetos usando un brazo robótico UR5 con Cámara RGB-D vía Aprendizaje Autónomo.

Proyecto Integrador

Previo la obtención del Título de:

Ingeniero en Mecatrónica

Presentado por:

Josué Andrés Cajamarca Contreras

Michael Antonio Quezada Zamora

Guayaquil - Ecuador

Año: 2023

Dedicatoria

El presente proyecto lo dedicamos a nuestras familias y amigos que estuvieron apoyándonos, confiando en que lograríamos llegar a este día, sin su apoyo y ayuda para seguir adelante esto no hubiera sido posible.

Agradecimientos

Nuestro más sincero agradecimiento a nuestros compañeros del Laboratorio de Bioinformática y Aprendizaje Autónomo, a nuestro tutor el PhD. Edwin Valarezo Añazco, y a nuestras familias y amigos, desde lo más profundo de nuestro corazón gracias por la ayuda, el apoyo y los ánimos de seguir adelante.

Declaración Expresa

“Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; Josué Andrés Cajamarca Contreras y Michael Antonio Quezada Zamora damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual".

Josué Cajamarca Contreras

Michael Quezada Zamora

Evaluadores

Firmado electrónicamente por:
BRYAN JOAO
PURUNCAJAS MAZA

Bryan Puruncajas Maza, M.Sc.

Profesor de materia

Firmado electrónicamente por:
EDWIN GIANNINE
VALAREZO ANAZCO

Edwin Valarezo Añazco, Ph.D.

Profesor tutor

Resumen

Dentro de las empresas almaceneras de bienes o encargadas de brindar el servicio de renta de espacios para almacenaje, existen problemas de eficiencia donde las jornadas de trabajo que llegan a exceder las 11 horas diarias les pasan factura a sus trabajadores, volviéndose propensos a cometer errores involuntarios como daños o perdida de bienes durante su reubicación, además de posibilitar casos de almacenamiento excesivo por desconcentración o fatiga. Conociendo que en la actualidad un 36% de empresas emplean activamente una solución autónoma, se propone una solución con el uso de un brazo robótico capaz de reubicar eficientemente objetos. Para lo cual se requiere implementar un sistema de visión capaz de detectar objetos, implementar el uso de algoritmos de aprendizaje reforzado basado en DDPG, y diseñar unas pinzas que se acoplen correctamente a la geometría de los objetos definidos. Este proyecto hizo uso de un software libre capaz de simular entornos con físicas reales donde se definió el espacio de trabajos con las herramientas necesarias como el robot UR5, las pinzas diseñadas, una mesa de trabajo, un estante y los objetos a reubicar. Se usó el aprendizaje profundo por refuerzo basado en experiencia, mediante la creación de funciones de recompensas que hicieron posible que el robot aprenda las acciones optimas que debe tomar para lograr una reubicación efectiva. Además de implementarse un sistema de visión robótica usando algoritmos de detección y clasificación de objetos para la localización de elementos en el espacio de trabajo. Al final contamos con una visión robótica capaz de identificar correctamente los objetos definidos inicialmente, una matriz de transformación capaz de realizar un cambio de perspectiva correctamente, el diseño de unas pinzas que se ajustan adecuadamente a los elementos seleccionados, y una red neuronal entrenada mediante algoritmos basados en experiencias que van aprendiendo a realizar las acciones necesarias.

Palabras claves: Pinzas, reubicación, aprendizaje, red.

Abstract

Within the companies that store goods or are in charge of providing storage space rental services, there are efficiency problems where working hours that exceed 11 hours a day take their toll on their workers, making them prone to involuntary errors such as damage or loss of goods during their relocation, as well as making possible cases of excessive storage due to lack of concentration or fatigue. Knowing that currently 36% of companies actively employ an autonomous solution, a solution is proposed with the use of a robotic arm capable of efficiently relocating objects. For which it is required to implement a vision system capable of detecting objects, implement the use of reinforced learning algorithms based on DDPG, and design grippers that are correctly coupled to the geometry of the defined objects. This project made use of a free software capable of simulating environments with real physics where the workspace was defined with the necessary tools such as the UR5 robot, the designed grippers, a worktable, a shelf and the objects to be relocated. Deep learning by experience-based reinforcement was used by creating reward functions that made it possible for the robot to learn the optimal actions to take to achieve an effective relocation. In addition to implementing a robotic vision system using object detection and classification algorithms for locating items in the workspace. In the end we have a robotic vision capable of correctly identifying the objects initially defined, a transformation matrix capable of correctly changing perspective, the design of grippers that properly adjust to the selected elements, and a neural network trained by algorithms based on experiences that learn to perform the necessary actions.

Keywords: Gripper, relocation, learning, network.

Índice General

Abreviaturas

DL	Deep Learning
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
DDPG	Deep Deterministic Policy Gradient
XML	Extensible Markup Language
URDF	Unified Robotics Description Format
STL	Standard Triangle Language
YOLO	You Only Look Once
RGB-D	Red Green Blue - Depth
MuJoCo	Multi-Joint dynamics with contact
DOF	Degrees of Freedom
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

Simbología

Gpa	Giga Pascales
m	Metros
cm	Centímetros
s	Segundos
rad	Radianes
k_r	Coeficiente de reducción del actuador
m_{nom}	Torque Nominal de Motor

Índice de figuras

Índice de tablas

Capítulo 1

1.1 Introducción

En la actualidad, hay varias empresas las cuales se dedican al almacenamiento de objetos o paquetes, ya sea para únicamente guardar bienes o para despachar entregas de dichos paquetes, el problema radica en la mala organización puede generar perdida de paquetes o aceptar un exceso de almacenamiento. Al incurrir en estas situaciones, las empresas llegan a tener pérdidas económicas y la confianza de los clientes. En diversos casos influye el factor humano cuando no se tiene en cuenta el espacio faltante o no le da mucha importancia donde se almacenan los objetos en la bodega.

En la actualidad existen robots para realizar tareas que requieren levantar y girar paquetes pesados en espacios reducidos, esto debido al alto riesgo para trabajadores los cuales pueden llegar a salir lastimados, es una de las razones por la cual convendría a las empresas automatizar este tipo de procesos [1].

Es importante tener una organización estricta en estos casos, de esa manera se ahorra tiempo, dinero. Así se podrá brindar un servicio más eficiente y seguro para los clientes, los cuales al estar satisfechos con el servicio dado se podrá lograr tener una recomendación y así aumentará la clientela y por ende los ingresos a la empresa.

Se sabe que, en el 2020, el 34% de los operadores de almacenes estadounidenses medianos y grandes ya contaban con robots en una o más ubicaciones. Esta cifra se estimó que subió a 42% el 2022. Además, los robots en las tiendas minoristas experimentaran un aumento similar. Lo que da a entender que las empresas buscaran aumentar el número de robots para así apoyarse en la automatización mejorando sus procedimientos internos y mostrando una imagen de vanguardia tecnología a sus clientes [2].

En este proyecto se busca crear un agente inteligente simulado para el almacenamiento de inventario accesible, el cual se implementará basado en un algoritmo de aprendizaje profundo reforzado (Deep Reinforcement Learning). El agente inteligente controlará un brazo robótico para localizar y sujetar el objeto de interés entre un grupo de posibles objetos y almacenarlo en el lugar donde corresponde. Esta solución toma en cuenta que los objetos no estarán siempre en la misma posición, la posición inicial del objeto es aleatoria en un área de trabajo.

1.2 Descripción del problema

La eficiencia en el almacenamiento de bienes o recursos dentro de bodegas (i.e., warehouse) puede ser baja en empresas de gran infraestructura, dado que actualmente aún se depende para estos procesos del factor humano, siendo susceptible a errores involuntarios del personal y reducción de la jornada efectiva de operación debido a fatiga laboral. Esta dependencia subsecuentemente puede derivar en problemas tales como el almacenamiento excesivo (i.e., overstocking), daños o perdidas en los bienes almacenados, lo que puede repercutir en empresas de préstamo de servicios de almacenamiento o depósito de inventarios (almaceneras) tanto para personas naturales como para empresas.

El coste de almacenamiento llega a ser muy variable, debido a que depende de distintos factores que definen el costo real. por ejemplo, unos de estos factores son, la cantidad de productos, dimensión de la mercadería, estacionalidad, la demanda, etc. [3] Teniendo estos factores en cuenta los errores de almacenamiento excesivo y daño de la mercadería pueden generar una perdida desde decimas hasta miles de dólares a las empresas, dependiendo de la infraestructura, es por eso que las empresas más grandes llegan a sufrir más perdidas.

1.3 Justificación del problema

En la actualidad, según una encuesta realizada por Deloitte, se revelo que el 38% de las empresas de logística utilizan activamente soluciones autónomas o robóticas dentro de sus procesos [4], trasladando así a que los operadores se concentren en tareas de alto nivel cognitivo. También se realizó un estudio por Deloitte en el que dice que un programa de mantenimiento predictivo que use Inteligencia Artificial (i.e. IA) podrá reducir los costes de mantenimiento hasta en un 10% [4]. Por lo que se busca reducir las pérdidas por errores humanos, también se reducirían los costos de mantenimiento. Por otro lado, la tecnología avanza rápidamente y las empresas deben mantenerse a la vanguardia para no perder mercado ante la competencia.

Adicionalmente, el Laboratorio de Bioinformática y Aprendizaje Autónomo (LBA2) de la Escuela Superior Politécnica del Litoral (ESPOL) tiene la necesidad de llevar este proyecto debido a la importancia de generar esta tecnología en la industria local, regional y mundial. Además, este trabajo de investigación fortalece los objetivos la institucionales, incluidos dentro de su plan quinquenal.

1.4 Objetivos

1.4.1 Objetivo General

Desarrollar un agente inteligente simulado capaz de reubicar objetos posicionados aleatoriamente en un área de trabajo dentro de un gabinete empleando algoritmos basados en Aprendizaje Autónomo.

1.4.2 Objetivos específicos

- Implementar un sistema de visión robótica basado en Inteligencia Artificial e Imágenes RGB-D para el reconocimiento y localización de objetos.
- Desarrollo de una función específica para el cálculo de la recompensa (i.e., Rewards) del proceso de reubicación mediante métodos de tipo “Model-Free”.
- Implementar un algoritmo de entrenamiento basado en Deep Reinforcement Learning (DRL), el cual controlará un brazo robótico UR5 y un actuador (Gripper).
- Diseñar un Gripper para Brazo Robótico UR5 para agarre de objetos con geometrías complejas mediante lenguaje de modelado MuJoCo Modeling XML File (MJCF) e Inventor.

1.5 Marco teórico

1.5.1 Aprendizaje profundo (DL)

El aprendizaje profundo es un subconcepto de la inteligencia artificial, la cual está centrada en crear grandes modelos de redes neuronales que son capaces de tomar decisiones acertadas basada en datos. DL es particularmente adaptada ambientes donde los datos son complejos y tiene un gran conjunto de datos disponibles. Hoy en día un sin número de tecnologías usan aprendizaje profundo, un buen ejemplo de esto son los teléfonos inteligentes los cuales usan reconocimiento facial o el reconocimiento de voz [5].

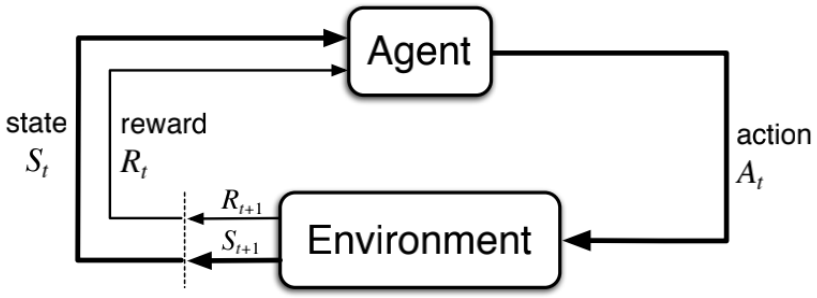
1.5.2 Aprendizaje por refuerzo (RL)

El aprendizaje por refuerzo (RL) consiste en aprender mediante acciones para maximizar una función recompensa. Se trata de realizar las acciones que proporcione la máxima recompensa. RL permite a las máquinas simuladas aprender su comportamiento en función de la recompensa recibida del entorno [6].

Entonces RL entrega un agente en base a su experiencia interactuando con un ambiente para desarrollar una tarea [7].

Figura 1.1

Sewak, M. (2019). Deep Reinforcement Learning: Frontiers of Artificial Intelligence. Springer



Como se puede ver en la figura 1.1, se considera RL al ciclo en el que un agente interactúa con un entorno ε en pasos temporales discretos. En cada paso de tiempo t, el agente observa un

estado $s_t \in \mathbb{R}^n$, realiza una acción en $a_t \in \mathbb{R}^n$,, pasa a un nuevo estado $s_{t+1} \in \mathbb{R}^n$y recibe una recompensa de retroalimentación $r_t \in \mathbb{R}$ de ε. El objetivo del RL es optimizar la política de selección de acciones del agente de forma que se consiga la máxima recompensa esperada [8].

1.5.3 Aprendizaje profundo por refuerzo (DRL)

El Deep Reinforcement Learning (DRL) es una rama del aprendizaje automático que combina dos áreas fundamentales: el aprendizaje por refuerzo (Reinforcement Learning, RL) y el aprendizaje profundo (Deep Learning). El objetivo del DRL es permitir que un agente aprenda a tomar decisiones óptimas y tomar acciones para maximizar una recompensa acumulada a lo largo del tiempo en un entorno específico y desconocido [9].

En el DRL tradicional, un agente interactúa directamente con un entorno desconocido, aprendiendo de la experiencia a través de ensayo y error. Sin embargo, en el enfoque actual para el proyecto, es decir Model-Based, el agente intenta aprender un modelo interno del entorno en lugar de interactuar directamente con él. Este modelo interno representa una aproximación del comportamiento del entorno, lo que permite al agente simular posibles interacciones y aprender de ellas sin necesidad de experimentar en el entorno real [9].

La ventaja del enfoque Model-Based DRL radica en que el agente puede aprovechar la información que ha aprendido a través de la simulación para tomar decisiones más informadas en el entorno real. Esto puede ser especialmente útil en situaciones donde la interacción directa con el entorno es costosa o peligrosa, o cuando el espacio de acciones es grande y la exploración aleatoria sería ineficiente [9].

DRL utiliza las siguientes definiciones en la composición del algoritmo:

- **Función Recompensa**

Esta función define la meta a un problema de RL. En cada instante de tiempo, el entorno envía al agente de RL un único número llamado recompensa. El objetivo de este agente es maximizar el total de la recompensa que recibe denle un episodio de tiempo determinado. La recompensa es identificada por las letras R o r [6].

- **Función de Valor**

La función de valor especifica que es bueno o útil dentro del episodio. El valor del estado es el número total de recompensas que un agente puede esperar para ir acumulando en el futuro, empezando por el estado en el que se encuentra. El valor determina la deseabilidad del estado para el final del episodio. Se identifica por el símbolo V(s) donde s es el estado actual [6].

- **Política (Policy)**

Una política es una correspondencia entre los estados percibidos del entorno y las acciones que deben emprenderse en esos estados. Una política se representa mediante el símbolo π (┣ a_t ┤| s_t) [6].

- **Espacio de Acción**

El espacio de acción es el conjunto de todas las acciones que el modelo puede realizar en cualquier momento. Se designa con el símbolo A [6].

- **Espacio de Observación**

Las observaciones muestran el efecto de la acción sobre el entorno. El espacio de observación describe el formato de las observaciones válidas [6].

- **Policy Gradient**

Los métodos de Policy Gradient aprenden una función de política a partir de trayectorias generadas por la política actual optimizando los parámetros de la política con respecto a la recompensa esperada a largo plazo mediante el descenso de gradiente. [6]

- **Actor – Critico**

Los algoritmos Actor-Crítico son un conjunto de algoritmos de RL que cambian la política antes de que se establezca el valor. En él, el Actor es responsable de mejorar la política y estimar cómo los cambios en la política afectan a la recompensa. El Crítico realiza la evaluación de la política y evalúa la recompensa dado que la política actual se utiliza tal cual. Utilizando los dos simultáneamente, se puede garantizar que cualquier cambio en la política por parte del Actor conduzca a un aumento en la recompensa predicha por el Crítico [6].

Cabe destacar que dentro de los algoritmos de DRL, Deep Q-Learning (DQN) y Deep Deterministic Policy Gradients (DDPG) fueron los primeros métodos en usar RL y Deep Learning juntos, para espacios de acción discretos y continuos, respectivamente. Desde entonces se han propuesto varios avances algorítmicos [9].

En este caso se utilizará DDPG debido a que el espacio en el que vamos a trabajar será continuo.

1.5.4 Gradiente de política determinista profunda (DDPG)

Se trata de un algoritmo de gradiente de política que evalúa una determinada política objetivo, que es mucho más fácil de aprender. Adopta un comportamiento estocástico con el fin de mejorar la exploración. Consideremos un escenario de aprendizaje por refuerzo en el que un agente se encuentra en el estado "s" en el paso de tiempo "t". El agente realiza una acción "a_t" en el paso temporal "t", recibe una recompensa "r_t" y pasa al estado "s_{t+1} ". El algoritmo DDPG consiste en una red neuronal actor parametrizada μ(s|θ^{μ}) y una red neuronal crítica Q(s, a|θ^{Q}). La red neuronal μ(s|θ^{μ}) realiza el mapeo de estados a una acción específica y la red neuronal crítica Q(s, a|θ^{Q}) aprende utilizando un algoritmo fuera de la política como Q-Learning y emite la recompensa total esperada dado un estado 's_t' y una acción 'a_t'. θ^{μ} y θ^{Q} actúan como los

parámetros de peso. Además, existen copias adicionales de ambas redes, la copia Actor μ'(s|θ^{μ}) y la copia Crítico Q'(s, a|θ^{Q}), denominadas "redes objetivo", que se encargan de realizar los cálculos. Cuando las muestras se generan mediante exploración secuencial en línea, la suposición de muestras distribuidas idénticamente ya no se cumple. Para resolver este problema, DDPG se basa en el uso de una memoria intermedia de repetición. Las transiciones resultantes de la acción "at" se almacenan como una tupla (s_t, a_t, r_t, s_{t+1}) en esta memoria intermedia. Luego, se toman muestras de las transiciones del entorno en forma de un mini lote aleatorio de N transiciones (s_i, a_i, r_i, s_{t+1}). El retorno esperado se calcula entonces mediante [6]:

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu(s_{i+1}|\theta^{\mu})\middle|\theta^{Q'}\right) \tag{1.1}$$

donde γ es el factor de descuento.

Entonces, la red crítica se actualiza minimizando la función de pérdida L observada con respecto a la recompensa total predicha utilizando Q($s_t, a_t|\theta^{Q}$) es decir [6],

$$L = \sum_i \left(y_i - Q(s_i, a_i|\theta^{Q})\right)^2/N \tag{1.2}$$

La red de actores se actualiza con la siguiente estimación de gradiente.

$$\nabla_{\theta\mu} J \approx (\sum_i \nabla_a Q(s, a|\theta^{Q})|_{\text{s=si,a=μ(si)}} \nabla_{\theta\mu} \mu(s|\theta^{\mu})|_{\text{si}})/N \tag{1.3}$$

Las actualizaciones anteriores no condujeron a la convergencia. Por lo tanto, hay una necesidad de actualizaciones suaves a los parámetros de las redes de destino que llevó a una mejor estabilidad del aprendizaje. Se siguen las siguientes ecuaciones para las actualizaciones de destino [6].

$$\theta^{Q\prime} \leftarrow \tau\theta^{Q} + (1+\tau)\theta^{Q\prime} \tag{1.4}$$

$$\theta^{\mu\prime} \leftarrow \tau\theta^{\mu} + (1+\tau)\theta^{\mu\prime} \tag{1.5}$$

donde τ es la constante de actualización suave.

1.5.4.1 MuJoCo

Multi-Joint dynamics with Contact (MuJoCo) es un programa que nos permite modelar, simular y visualizar de manera rápida y precisa la dinámica de contacto entre sistemas multi - articulados.

Una de las características fundamentales de MuJoCo es la capacidad de gestionar operaciones costosas desde el punto de vista computacional, especialmente si hablamos de sistemas dinámicos complejos con un elevado número de contactos, dichos sistemas son para lo que este simulador esta optimizado. El uso preferente del programa es la investigación y desarrollo para Machine Learning (ML) y concretamente RL, es por lo que este simulador es seleccionado por OpenAI para poder implementar entornos en 3D diseñados para el control continuo en el equipo de herramientas de Gym [10].

Sus componentes principales son:

- Módulo de simulación
- Analizador XML y Compilador de modelos
- Visor OpenGL Interactivo

1.5.5 Estado del arte

1.5.5.1 Comparación con trabajos previos

En la actualidad, no existen trabajos o publicaciones anteriores realizadas bajo los requerimientos específicos planteados por nuestro cliente (Laboratorio de Bioinformática y Aprendizaje Autónomo), sin embargo, existe semejanza con artículos de investigación publicados en conferencias de renombre como la International Conference on Intelligent Robots and Systems (IROS) [6].

Un problema popular por resolver en estos días es proponer un algoritmo tal que una extremidad robótica aprenda de forma autónoma la locomoción. Algunos de los algoritmos contemporáneos más eficientes y populares son los algoritmos de Aprendizaje Profundo por Refuerzo (DRL) [6]. A raíz de lo cual, se buscaron publicaciones científicas que reflejen el uso de estos algoritmos, así como al menos otro requerimiento del proyecto,

- **Automated Gait Generation for Simulated Bodies Using Deep Reinforcement Learning:**

En este trabajo, se estudiaron tres de algoritmos DRL, - Deep Deterministic Policy Gradient (DDPG), Advantage Actor Critic (A2C), y Proximal Policy Optimization (PPO)-. Se implementaron y compararon los algoritmos según la métrica de rendimiento de la recompensa media por episodio. Para a posterior, concluir sobre la eficiencia de los algoritmos propuestos y clasificar los algoritmos basándose en la misma eficiencia bajo un entorno de simulación de Mujoco [6].

- **Robotic Arm Control and Task Training through Deep Reinforcement Learning:**

En este artículo, se propuso una comparación detallada y extensa de la región confiable para la optimización de la política (TRPO) y la Q-Red Profunda (DQN) con funciones de ventaja normalizadas con respecto a otros algoritmos de última generación, a saber, el Gradiente de Política Determinista Profunda (DDPG) y el Gradiente de Política de Vainilla (VPG). Se presentaron experimentos simulados y reales, en donde la simulación permitió mostrar los procedimientos adoptados para estimar con precisión los hiperparámetros de los algoritmos y diseñar correctamente buenas políticas. Mientras los experimentos en el mundo real permitieron demostrar que las políticas propuestas, se entrenaron correctamente en simulación, siendo transferibles y ejecutables en un entorno real sin apenas cambios [11].

- **RGB and 3D-Segmentation Data Combination for the Autonomous Object Manipulation in Personal Care Robotics:**

En este artículo, se propuso rediseñar las funcionalidades del robot de cuidados personales (RCP) Pepper de SoftBank Robotics. Pepper está diseñado principalmente para la interacción verbal con pacientes y, a pesar de la presencia de dos brazos superiores, carece de capacidades de manipulación de objetos. En el rediseño propuesto, se utilizó una combinación de datos de la cámara RGB y el sensor de profundidad 3D de Pepper para identificar y localizar un sobre farmacéutico específico en un depósito dedicado. En este contexto, la segmentación semántica de la imagen RGB se ha confiado a un detector de objetos YOLOv3 pre-entrenado, mientras que el posicionamiento del actuador (pinza) se ha realizado mediante un algoritmo específico. Basándose en esta información posicional 3D, el PCR opera una rutina diseñada ad-hoc para agarrar el objeto y escanearlo. Una vez que el procedimiento de escaneado confirma que el agarre se ha completado con éxito y que el envase agarrado se corresponde con el fármaco necesario, el PCR debe ser capaz de desplazarse con seguridad hacia el usuario y entregar el fármaco, por ejemplo, a un médico o un paciente. El procedimiento propuesto es totalmente automático, y no se necesitó de conexión a Internet para el caso de uso nominal, preservando -de este modo- datos sensibles como mapas del hogar/hospital, datos del paciente, etc. Por último, se presenta una prueba de concepto que implementa una operación secuencial de recogida, reconocimiento y entrega de objetos, demostrando su aplicabilidad en situaciones reales [12].

1.5.5.2 Contribución

A diferencia de los trabajos previos mencionados, el presente proyecto implementará un algoritmo de visión robótica para la identificación y localización de objetos basado en una cámara RGB-D y el algoritmo de entrenamiento de imágenes YOLO. Así mismo, el agente inteligente

será basado en el algoritmo de DDPG para el control del Brazo Robótico UR5 en un ambiente de simulación de Mujoco empleando librerías como Gymnasium (OpenAI) para la recreación y manipulación del entorno. Posteriormente, se desarrollará una función de recompensas para fases de Brazo Robótico (F1: Acercamiento, F2: Levantamiento, F3: Reposicionamiento a Gabinete y F4: Reubicación en Casillas de Estante). Finalmente, se desarrollará un algoritmo de entrenamiento para la ejecución de estas fases y algoritmos de evaluación y renderización de resultados, mediante los cuales se demostrará la eficiencia del modelo entrenado (aceptable mayor al 80%) para su implementación en bodegas de almacenamiento.

Capítulo 2

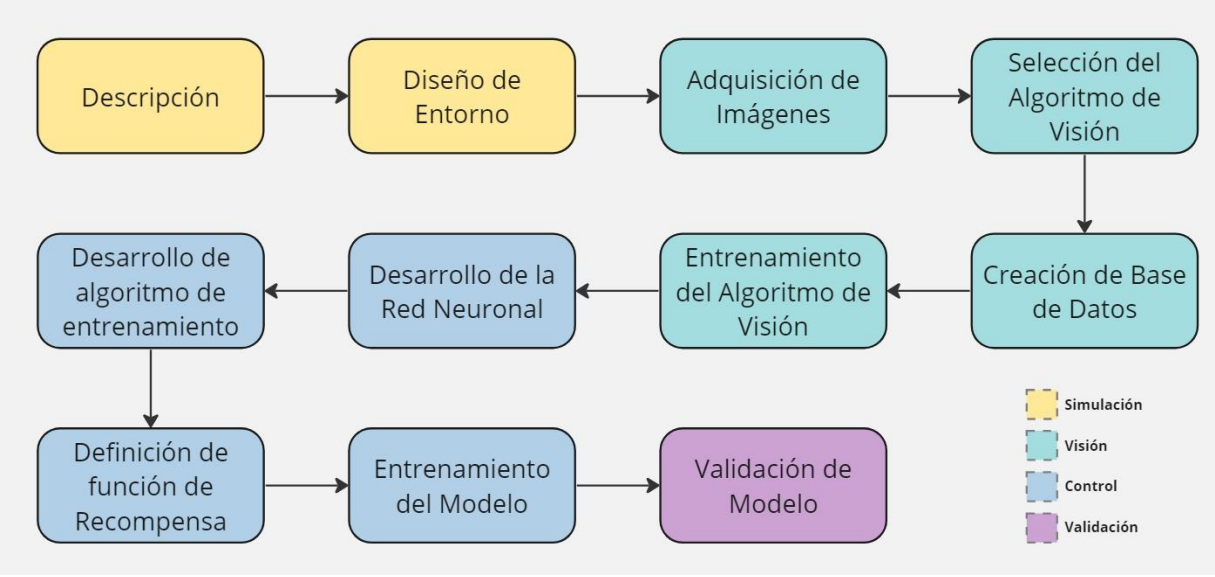
2 Metodología

2.1 Etapas

El enfoque de este capítulo se centró en la descripción de la metodología empleada para el desarrollo de un sistema de manipulación autónoma de objetos usando un brazo robótico UR5 con Cámara RGB-D vía aprendizaje autónomo. Para este propósito, se categorizaron las fases de desarrollo del proyecto, siendo la primera etapa el desarrollo de la Simulación, tal que se estableció la descripción y el diseño del Entorno; posteriormente la fase de Visión, donde se hizo la adquisición de Imágenes, se seleccionó el algoritmo de visión, se creó una base de datos de imágenes y sé entrenó el algoritmo de visión; seguido de la fase de Control, donde se desarrolló la red neuronal, el algoritmo de entrenamiento, se definió la función de recompensa y se hizo el entrenamiento del modelo; para finalmente, validar el modelo conjunto con la parte de Visión y Control.

Figura 2.1

Etapas de Diseño



2.2 Simulación

2.2.1 MuJoCo

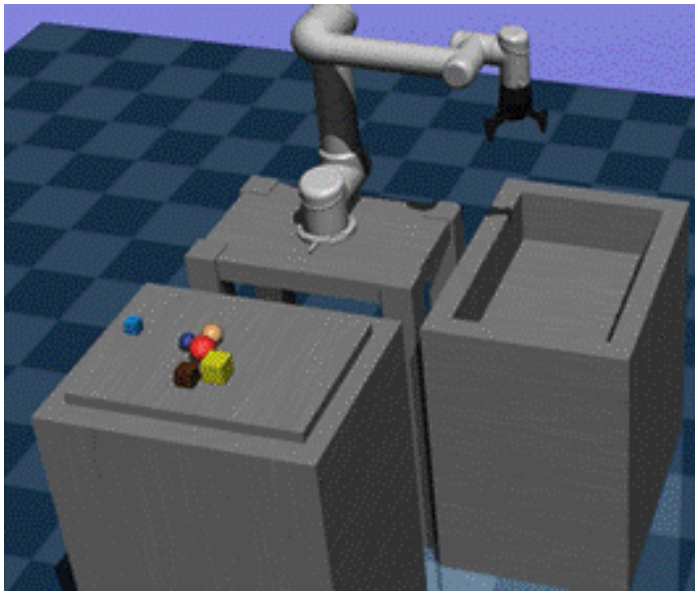
El entorno de simulación se realizó empleando el motor físico de dinámica multiarticular MuJoCo en su versión 2.3.6 distribuido por Google DeepMind, debido a que se trata de una plataforma de simulación de código abierto diseñada específicamente para simular y controlar la dinámica de sistemas mecánicos complejos (en particular, cuerpos articulados), empleada habitualmente para el desarrollo e investigación en las áreas de robótica y aprendizaje por refuerzo, donde sirve como herramienta para probar y optimizar algoritmos de control antes de desplegarlos en robots reales, su relevancia frente a otras plataformas de simulación reside en que presenta una simulación rápida y precisión en la interacción de estructuras articuladas con su entorno [18].

Dado el contexto actual del proyecto, se eligió el simulador de Mujoco por sobre otros simuladores como Gazebo, V-REP o PyBullet por las características mencionadas anteriormente y dado a que este simulador se asemeja a la realidad al incluir la definición física de los objetos y su movilidad (dinámica y cinemática), entre sus ventajas destacables, presenta un modelado de contactos realista entre objetos y superficies, flexibilidad en la definición de modelos, un control de alta fidelidad, eficiencia computacional, Interfaz de programación accesible y una amplia adopción en el campo investigativo, posibilitando la obtención de resultados que nos permitan demostrar la implementación del proyecto en un entorno lo más aproximado a la realidad para nuestra aplicación de reubicación (Todorov et al. , 2012).

Para el propósito de este proyecto, se hizo uso de la base de un modelo de entorno de simulación ya desarrollado previamente por Daniel (2020), cabe destacar que se tomó como punto de inicio este entorno, sin embargo, se fueron realizando modificaciones tanto para el espacio de simulación como en las funciones de este.

Figura 2.2

Entorno de Simulación base en MuJoCo [14]



Como se puede apreciar en la Figura 2.2, el entorno de simulación base consta de un banco de montaje para el brazo, una mesa de trabajo para piezas, un depósito, un brazo robótico modelo UR5 de Universal Robots equipado con un Gripper tipo pinza de 2 dedos y una cámara RGB-D.

Cabe destacar que el modelo del Brazo Robótico UR5 fue tomado del repositorio de Roboti LLC desarrollado por Franceschetti (2018) mientras los demás elementos son propios del repositorio de GitHub desarrollado por Daniel (2020).

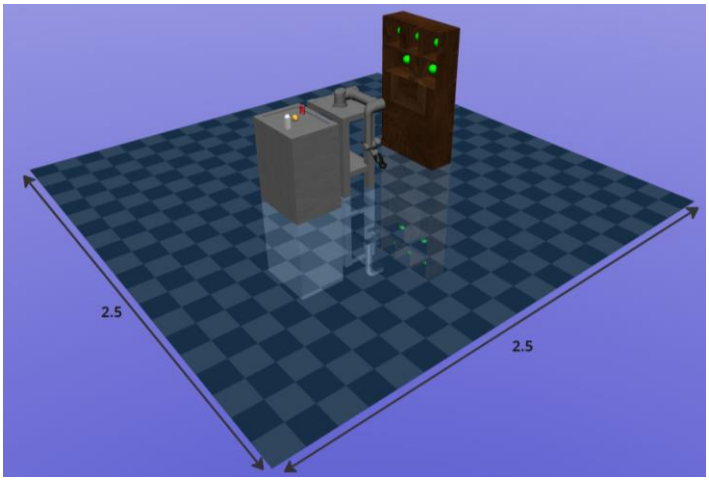
En MuJoCo, las unidades utilizadas están basadas en el Sistema Internacional de Unidades (SI), que es el sistema de unidades de medida internacional ampliamente utilizado en todo el mundo. Esto significa que las unidades en MuJoCo están basadas en las unidades estándar utilizadas en el SI, como metros (m) para longitudes, kilogramos (kg) para masas y segundos (s) para tiempo. Esto es importante, dado que el espacio de simulación a pesar de no indicar las unidades de los valores obtenidos/medidos, establecemos que se trabajará bajo el SI, donde todas

las cantidades mostradas posteriormente se regirán a este, o caso contrario se indicarán las unidades específicas [18].

Las dimensiones del entorno de simulación vienen dadas por un espacio de 2.5 m x 2.5 m:

Figura 2.3

Dimensiones de espacio de simulación

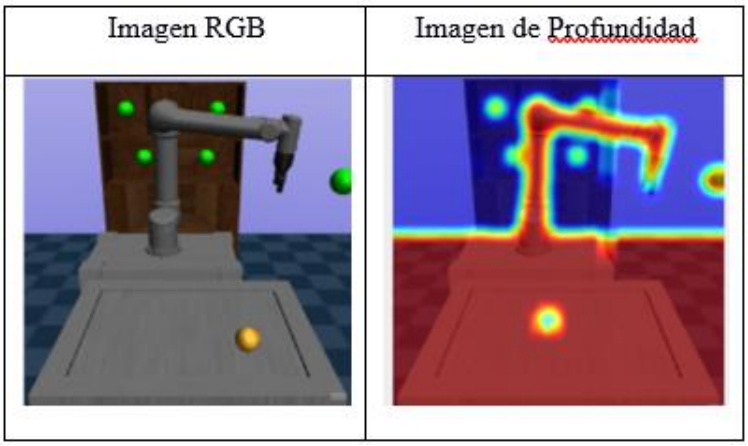


2.2.2 Cámara RGB-D

Se decidió optar por una cámara RGB-D porque se necesita una precisión a la hora de localizar los objetos los cuales se desea reubicar para esto se obtiene una captura que nos deja observar el entorno, dicha captura proporcionada por la cámara RGB-D proporciona tanto datos de profundidad (Depth), así como de color (RGB) como salida en tiempo real mediante la captura de luz en longitudes de onda rojas, verdes y azules y de luz infrarroja para la profundidad. De esa manera podemos tener la posición de los objetos con los cuales se trabajará.

Figura 2.4

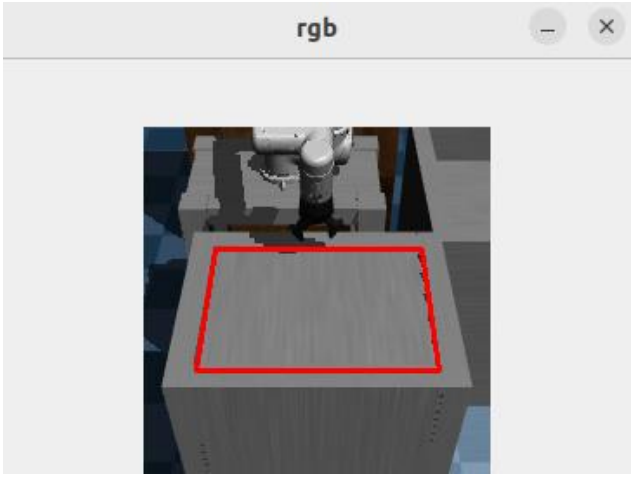
Imagen RGB e Imagen de Profundidad del Entorno de Trabajo



Para el entorno de simulación se emplearon dos tipos de renderizados definidos en la documentación de MuJoCo, uno para la obtención de la imagen RGB y otro para las imágenes de Profundidad (Depth), de lo cual, durante el entrenamiento, se estableció una ventana de renderizado para una cámara RGB-D destinada para la obtención de las imágenes de observación referenciada dentro de la programación como cámara “topdown”, la cual se ubicó en el espacio en la posición x=0, y=-1.7, z=2.0, a 45° de inclinación respecto al eje x y con un campo de visión (fovy) de 30°.

Figura 2.5

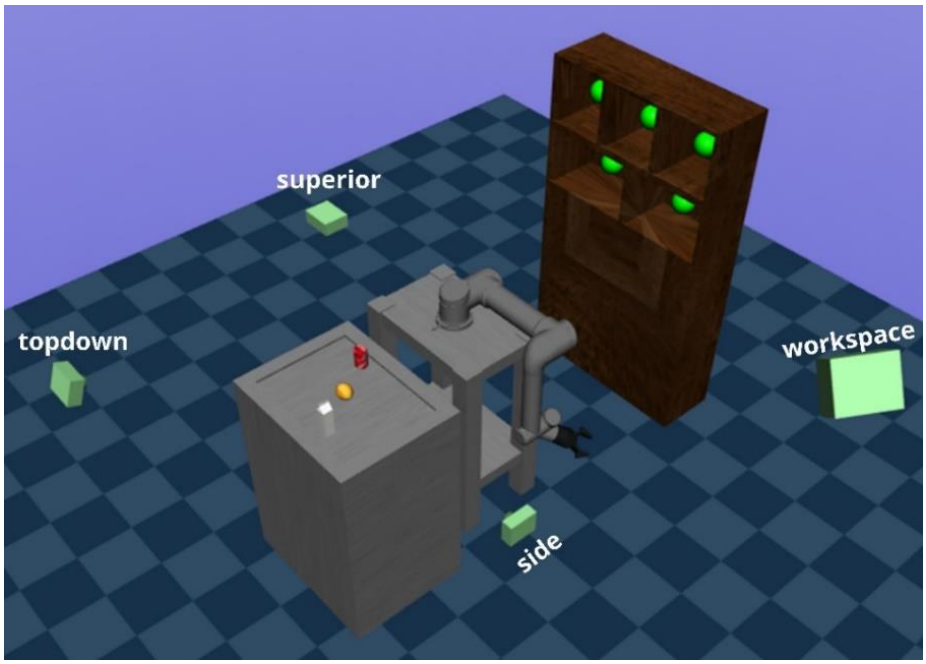
Renderizado de Imagen RGB, Cámara para Entrenamiento



Adicional a esta, se establecieron otras cámaras para la obtención de imágenes para la generación de videos, visualización de las tareas o captura del espacio de trabajo. Entre estas, se encuentran las cámaras “superior”, “side” y “workspace”, donde la primera se usó para la verificación de la transformación de coordenadas efectiva entre la cámara “topdown” a una referencia de vista superior como se verá más adelante, mientras las otras dos se emplearon para la visualización del espacio y ejecución de las tareas, tal como se aprecia en la figura 2.6.

Figura 2.6

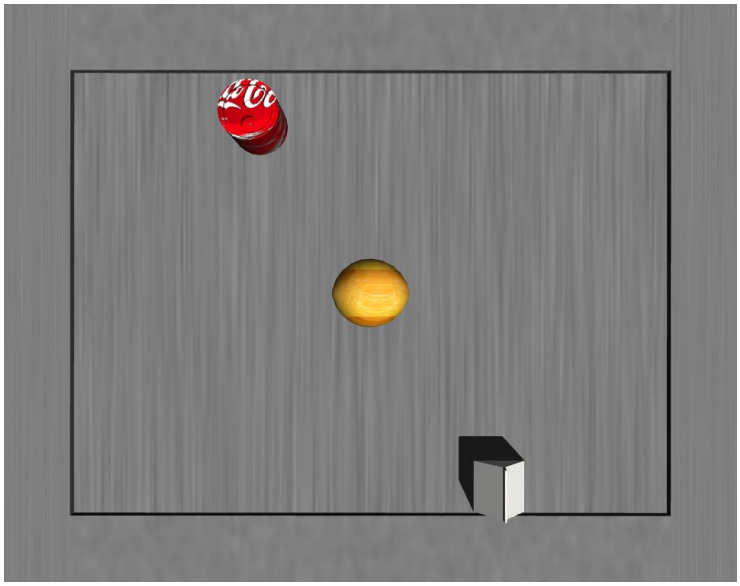
Cámaras establecidas en entorno



Respecto a la cámara fija “superior” para el renderizado de las imágenes desde una vista superior, fue útil para la generación de imágenes de muestra del correcto posicionamiento de los objetos en sus regiones de reaparición. Esta se estableció de modo fijo en la posición x = 0, y = -0.6, z = 2, con una inclinación del tipo xyaxis = "1.000 0.000 0.000 0.000 1.000 0.000" y con un campo de visión (fovy) de 30°.

Figura 2.7

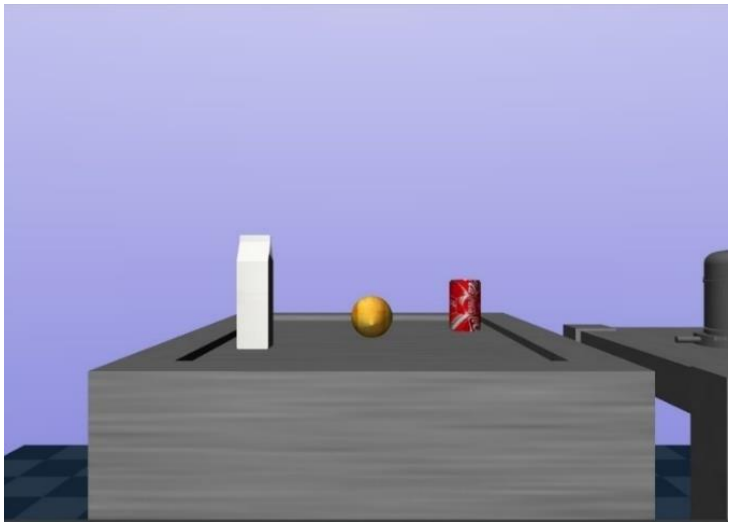
Vista superior de espacio de Trabajo



Respecto a la cámara fija “side” para la obtención de fotogramas de imágenes, fue útil para la realización de video para validación de agarre de piezas durante los episodios, se ubicó en el espacio en la posición x=0.8, y=-0.6, z=1.0, con una inclinación del tipo quaternion="0.5 0.5 0.5 0.5".

Figura 2.8

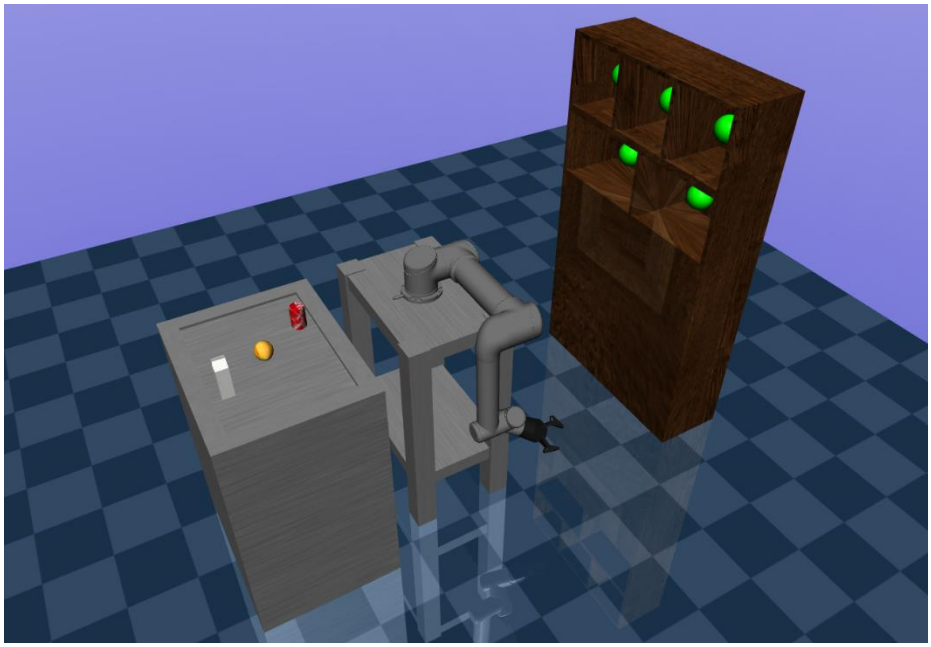
Vista lateral de espacio de Trabajo



Y, por último, respecto a la cámara fija “workspace” para la apreciación de todo el espacio de trabajo para la parte del post-entrenamiento y replicación de la simulación, de tal forma que sea apreciable la reubicación efectiva, esta camará se ubicó en la posición x = 1.9, y = -0.96, z = 2.5, con una inclinación del tipo xyaxis = "0.50 0.88 0.00 -0.55 0.30 0.78".

Figura 2.9

Cámara de Perspectiva

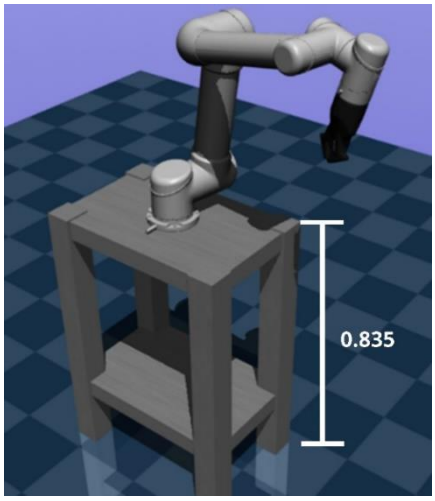


2.2.3 Brazo Robótico UR5

Para el modelado del robot manipulador UR5 de Universal Robots, contamos con 5 juntas y una base, lo cual da 6 grados de libertad al brazo robótico con una carga útil máxima de 5 kg en base a las especificaciones de diseño de Franceschetti (2018). Cabe destacar que el modelo incluye su banco de montaje para soporte del brazo robótico y una pinza basada en la pinza comercial Robotiq 2F-85 de 2 dedos acoplada en su efector final implementada por Daniel (2020), otorgando un total de 9 grados de libertad.

Figura 2.10

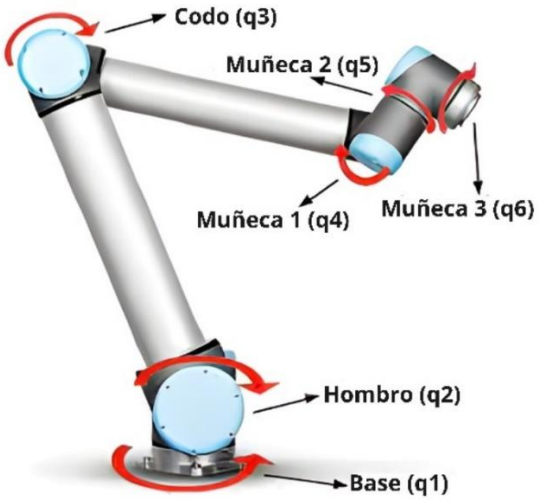
Brazo Robótico UR5 de Universal Robots



Como se mencionó anteriormente, el Brazo Robótico posee 6 juntas en su modelo estándar, sin embargo, debido a que se desea mayor libertad para el agarre y sabiendo que 2 grados de libertad no resultan suficientes para nuestro propósito, se optó por desarrollar una pinza propia debido a que se busca que la reubicación sea lo más generalizada posible, es decir para objetos de distintas geometrías o que presenten irregularidad, al menos 3 puntos de agarre necesarios.

Figura 2.11

Juntas de Brazo Robótico UR5



Dentro de las limitaciones que debe tener el objeto, se pueden ver en la tabla 2.1.

Tablas 2.1

Parámetros Básicos de Brazo Robótico UR5

Carga útil	5 kg
Alcance máximo	850 mm
Grados de libertad	6
Velocidad máxima	1 m/s

Por otro lado, en lo que concierne a los límites de las juntas para la simulación, junto a otros parámetros como el coeficiente de reducción del actuador y el torque nominal del actuador, tenemos los valores mostrados en la Tabla 2.1 de acuerdo con Franceschetti (2018):

Tablas 2.2

Parámetros para Simulación de Brazo Robótico UR5

Juntas	Límites de Juntas [rad]	k_r	m_{nom} $[Nm]$
$q1$	$[-\pi, \pi]$	101	150
$q2$	$[-\pi, 0]$	101	150
$q3$	$[-\pi, \pi]$	101	150
$q4$	$[-\pi, \pi]$	101	28
$q5$	$[-\pi, \pi]$	101	28
$q6$	$[-\pi, \pi]$	101	28

Es necesario conocer estos parámetros para la definición del archivo URDF de cinemática, tal que, dentro de este para las juntas, tendremos una variable “limit” con las especificaciones de “effort”, “lower” y “upper”, haciendo referencia para el caso de effort al valor m_{nom}, mientras los límites de las juntas se traducen en [“lower”, “upper”]. Para el caso del factor k_r, este se define dentro del archivo XML, correspondiendo a la sección donde se definen los actuadores de las juntas, se define el parámetro “gear”, el cual será 101 para todas las juntas según referencia el valor de la relación de engranaje del actuador o k_r. De esta forma, al establecer estos valores, la simulación nos permitirá en secciones

posteriores realizar comparaciones rápidas y seguras de las opciones de diseño como, en el caso de DRL (en referencia a DDPG), el ajuste de los hiperparámetros.

2.2.4 Pinzas

Las pinzas son efectores finales que se usan para realizar tareas de reubicación, específicamente de agarre, estos pueden clasificarse en 4 clases [3]:

- De impacto: estos son las mordazas o dedos (pueden ser de dos o varios dedos) que agarran los objetos por impacto directo, donde se aplica una fuerza mecánica en dos o más puntos opuestos de la superficie del objeto [3].
- Astrictivas: es cuando se aplica una fuerza de atracción sobre un punto o varios en la superficie del objeto (por ejemplo, por vacío, magneto adhesión) [3].
- Ingresivo: en esta clase se usan agujas o alfileres para penetrar, deformar o impregnar en la superficie del objeto [3].
- Contiguo: producir una fuerza de adhesión tras el contacto directo con la superficie del objeto (por ejemplo, pegamento, tensión superficial) [3].

Las pinzas más útiles para el objetivo de nuestro proyecto se deben de seleccionar de acorde a las necesidades de este, debido a las figuras irregulares que se tiene, se decidió optar por las posibles soluciones a continuación.

Posibles soluciones:

Solución A: Gripper 2 dedos

Solución B: Gripper 3 dedos

Solución C: Ventosas

Características por evaluar:

- Agarre
- Adaptabilidad
- Efectividad
- Maniobrabilidad

Una vez definimos nuestras posible soluciones y características a evaluar cada solución, realizamos la matriz de decisión para así tener un orden de prioridad en el cual se elegirá la solución con mayor prioridad.

2.2.4.1 Matriz de decisión

Tablas 2.3

Características de solución.

Agarre>Maniobrabilidad>Efectividad>Adaptabilidad						
Criterio	Agarre	Adaptabilidad	Efectividad	Maniobrabilidad	∑+1	Peso
Agarre	-	1	1	1	4	0.4
Adaptabilidad	0	-	0	0	1	0.1
Efectividad	0	1	-	0	2	0.2
Maniobrabilidad	0	1	1	-	3	0.3
				Suma	10	1

Tablas 2.4

Peso de cada característica de solución.

		B>A>C			
Agarre	Sol A	Sol B	Sol C	∑+1	Peso
Sol A	-	0	1	2	0.333333333
Sol B	1	-	1	3	0.5
Sol C	0	0	-	1	0.166666667
			Suma	6	1
		B>A>C			
Adaptablidad	Sol A	Sol B	Sol C	∑+1	Peso
Sol A	-	0	1	2	0.333333333
Sol B	1	-	1	3	0.5
Sol C	0	0	-	1	0.166666667
			Suma	6	1
		B=C>A			
Efectividad	Sol A	Sol B	Sol C	∑+1	Peso
Sol A	-	0	0	1	0.166666667
Sol B	1	-	0.5	2.5	0.416666667
Sol C	1	0.5	-	2.5	0.416666667
			Suma	6	1
		B=A>C			
Maniobrabilidad	Sol A	Sol B	Sol C	∑+1	Peso
Sol A	-	0.5	1	2.5	0.416666667
Sol B	0.5	-	1	2.5	0.416666667
Sol C	0	0	-	1	0.166666667
			Suma	6	1

Tablas 2.5

Conclusión y Prioridad de solución.

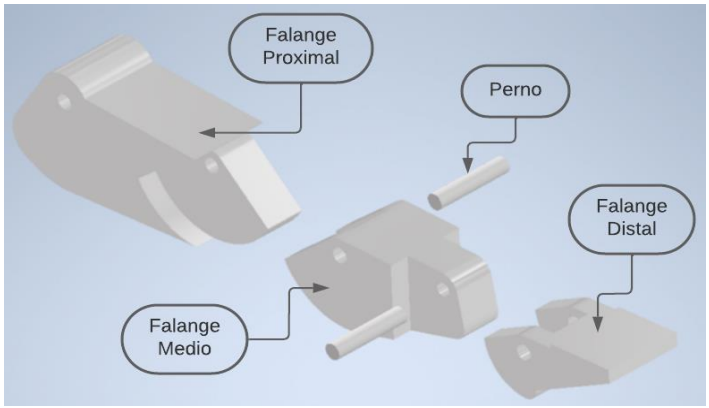
Conclusión	Agarre	Adaptabilidad	Efectividad	Maniobrabilidad	∑+1	Prioridad
Sol A	0.0833333	0.08333333	0.04166667	0.10416667	0.3125	2
Sol B	0.125	0.125	0.10416667	0.10416667	0.458333333	1
Sol C	0.0416667	0.04166667	0.10416667	0.04166667	0.229166667	3
					1	

2.2.4.2 Diseño de Pinzas

Buscamos la flexibilidad en las pinzas para poder acoplarse a los diferentes objetos que se estarán sujetando, además de tener un agarre más seguro para esto se diseñó similar a un dedo con falanges unidas por un perno para así poder dar esa flexibilidad que se necesita, además de basarnos en un modelo comercial como lo es la pinza de 3 dedos Robotiq S, lo que hace factible y versátil el diseño dado que en caso de requerirse repuestos similares o alternativas en el mercado de fácil acceso en lugar de requerirse invertir demasiado tiempo en la impresión o creación de las piezas, se puede optar por estas soluciones, eso sí, guardando las distancias con nuestro diseño dado a que posee mayor simpleza pero en esencia, la misma funcionalidad.

Figura 2.12

Partes de dedo expandidas.



En la figura 2.12 se observa la descomposición de un dedo antes del ensamblado donde consta de 3 falanges al igual que los dedos. Una vez obtenido el ensamble de los dedos, se ensambla el resto de la pinza y así obtener los planos los cuales se pueden observar en la figura 2.13.

Figura 2.13

Plano de las pinzas final.

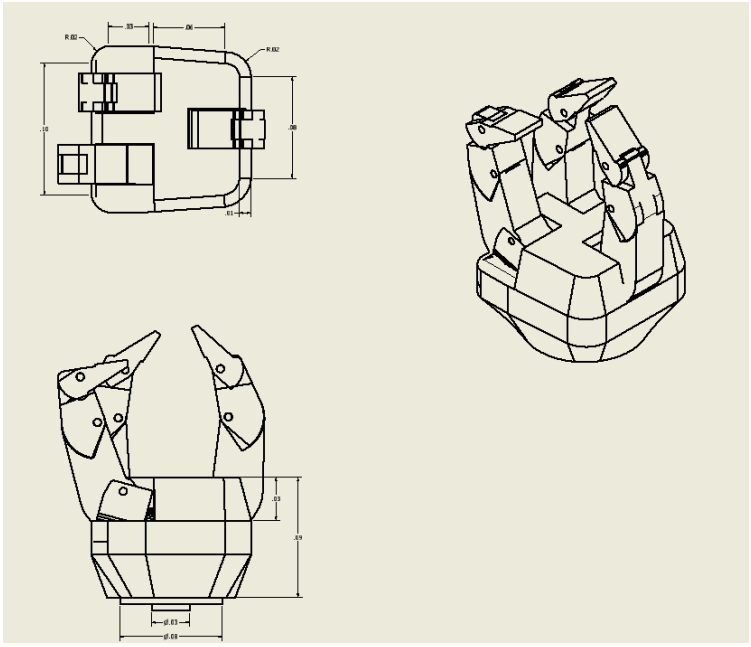
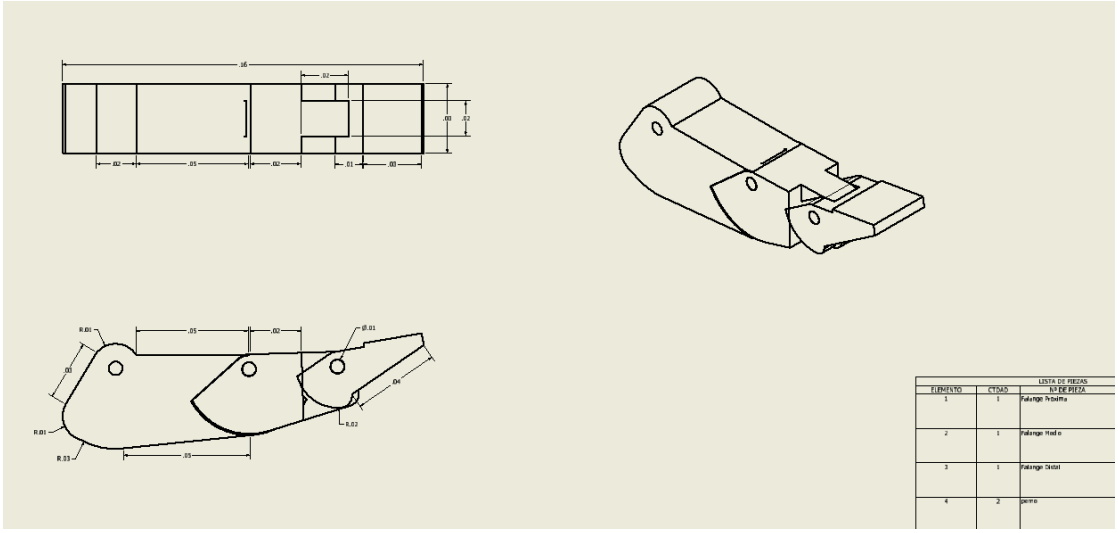


Figura 2.14

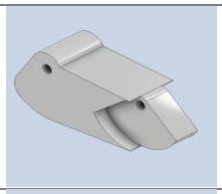
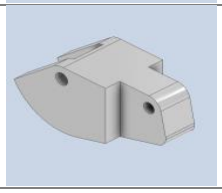
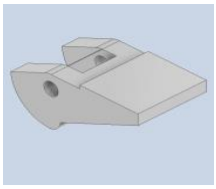
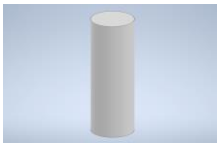
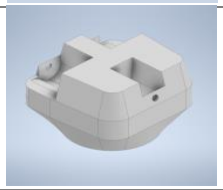
Plano dedo de las pinzas.



Para este diseño se obtuvo la tabla 2.6, que nos dice las piezas utilizadas junto con su cantidad respectiva para ser posible el ensamble final.

Tablas 2.6

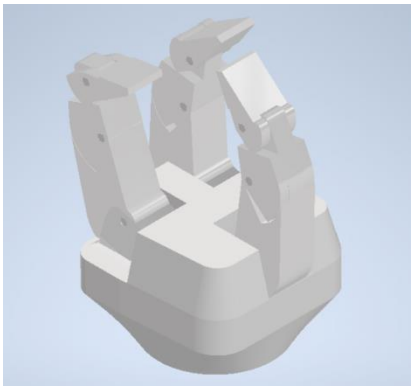
Lista de Elementos para el diseño de las pinzas.

Elemento	Nº de pieza	Miniatura	Estructura de la lista de materiales	CTDAD de unidades	CTDAD
1	Falange Proximal		Normal	Cada una	3
2	Falange Medio		Normal	Cada una	3
3	Falange Distal		Normal	Cada una	3
4	perno		Normal	Cada una	12
5	Nudillo		Normal	Cada una	3
6	Palma		Normal	Cada una	1

Finalmente podemos tener una mejor apreciación de la imagen del diseño final en la figura 16, donde se ve en una imagen 3D la pinza en su totalidad con los 3 dedos necesarios, de los cuales hemos dado prioridad según la matriz de decisión.

Figura 2.15

Diseño 3D de la pinza en inventor

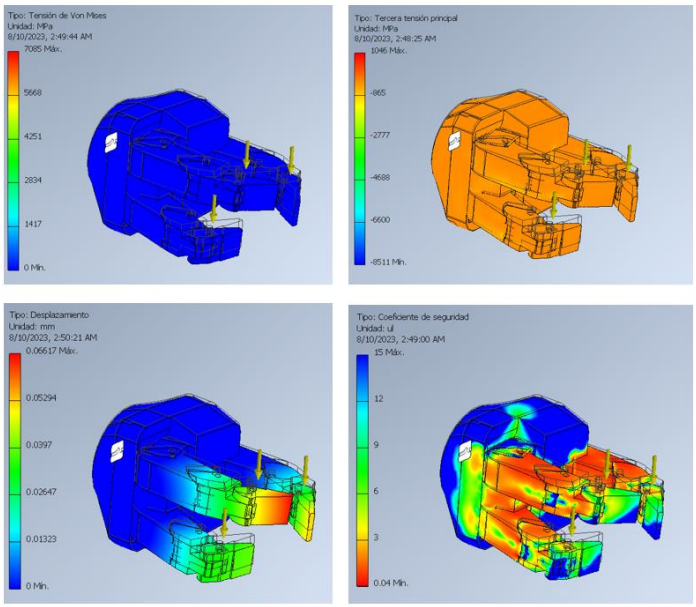


El diseño de las pinzas tiene sentido que sea de 3 dedos, debido a que al tener figuras con una geometría que se dificulta para cualquier pinza el realizar el agarre, teniendo 3 dedos pues se puede llegar a tener una mayor facilidad dado que cubre mayor superficie y nos da un agarre más seguro y óptimo de estos objetos.

Dentro de las limitaciones del UR5 se tiene que soportar un peso de 5Kg de carga, por lo que se realizó la evaluación de esfuerzos como se ve en la figura 2.16.

Figura 2.16

Análisis de tensión de la pinza



Se observa el desplazamiento en la figura 2.16. en dicho análisis se tomó en cuenta que al tener 3 superficies de contacto y teniendo una carga máxima de 5 [Kg] esta se deberá dividir entre las superficies que harían contacto con los objetos por lo cual de una fuerza de 49 [N] que inicialmente implica este peso se tendría 16.333 [N] en cada dedo de la pinza. De esa manera se obtuvo los resultados dentro de la tabla 2.7.

Tablas 2.7

Resumen de análisis de tensión de la Pinza.

Nombre	Mínimo	Máximo
Volumen	26.7477 mm^3	
Masa	0.000213982 kg	
Tensión de Von Mises	0.0124249 MPa	7085.45 MPa
Primera tensión principal	-1037.53 MPa	8576.02 MPa
Tercera tensión principal	-8511.47 MPa	1046.08 MPa
Desplazamiento	0 mm	0.0661711 mm
Coeficiente de seguridad	0.0352836 su	15 su
Tensión XX	-1068.15 MPa	1077.2 MPa
Tensión XY	-782.229 MPa	759.546 MPa
Tensión XZ	-431.218 MPa	494.004 MPa
Tensión YY	-8479.66 MPa	8542.58 MPa
Tensión YZ	-772.863 MPa	710.202 MPa
Tensión ZZ	-1993.86 MPa	2042.35 MPa
Desplazamiento X	-0.00122914 mm	0.00265929 mm
Desplazamiento Y	-0.0103229 mm	0.012455 mm
Desplazamiento Z	0 mm	0.0649545 mm
Deformación equivalente	0.0000000622523 su	0.0337958 su
Primera deformación principal	-0.00000305807 su	0.0396383 su
Tercera deformación principal	-0.0393908 su	0.0000553646 su
Deformación XX	-0.0113126 su	0.0112418 su
Deformación XY	-0.0052689 su	0.00511611 su
Deformación XZ	-0.00290458 su	0.00332749 su
Deformación YY	-0.0391764 su	0.039413 su
Deformación YZ	-0.00520581 su	0.00478374 su
Deformación ZZ	-0.00531125 su	0.00650033 su
Presión de contacto	0 MPa	3441.73 MPa
Presión de contacto X	-1495.54 MPa	1770.53 MPa
Presión de contacto Y	-2996.24 MPa	2969.41 MPa
Presión de contacto Z	-2020.04 MPa	3231.85 MPa

Dentro de la tabla 2.7. Podemos destacar valores importantes como la tensión de von mises con un máximo de 7085.45 [MPa], la primera tensión principal con máximo de 8576.02 [MPa], la

tercera tensión principal con 1046.08 [MPa] como máximo, y en cuanto a la deflexión se tiene un desplazamiento de 0.0661711 [mm].

El material más comúnmente usado es el acero inoxidable y es el que se consideró para el diseño. Un dato importante del material que hay que tener presentes es el módulo de elasticidad, el cual nos ayudara a evaluar si el diseño realizado junto con este material se romperá o aguantara. En este caso el módulo de elasticidad del acero inoxidable es de 193 [MPa].

2.2.4.3 Parámetros para Simulación

Respecto a las especificaciones de la pinza de 3 dedos, presenta los límites de las juntas de los dedos, el coeficiente de reducción del actuador y el torque nominal del actuador al igual que el brazo robótico, aunque estos últimos valores solo los tuvimos para la palma y la falange proximal, para estos valores nos valimos del modelo comercial Robotiq S, tal que tenemos los valores mostrados en la Tabla 2.8 de acuerdo con Franceschetti (2018):

Tablas 2.8

Especificaciones del motor y las juntas de la pinza de 3 dedos Robotiq S

Juntas	Límites de Juntas [rad]	k_r	m_{nom} $[Nm]$
qi_0	$[-0.2967, 0.2967]$	14	0.8
qi_1	$[0, 1.2217]$	14	0.8
qi_2	$[0, \pi/2]$	-	-
qi_3	$[-0.6632, 1.0471]$	-	-

2.2.5 Elementos del espacio simulado

2.2.5.1 Diseños en inventor

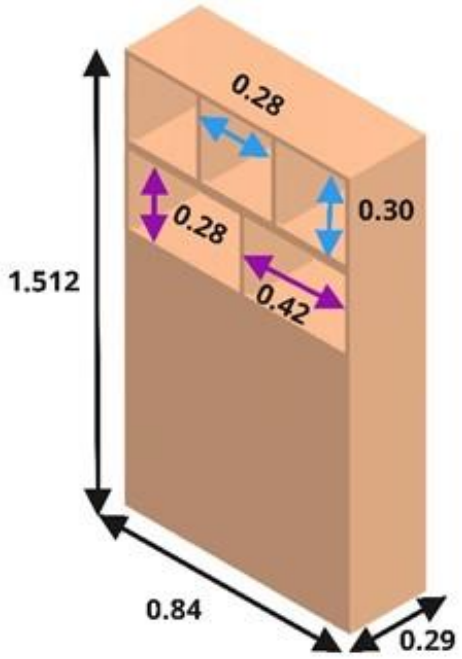
- Estante

Para el diseño del Estante de Reubicación, se emplearon diseños comerciales con modelos accesibles en la web, es así que buscando referencias en GrabCad, se optó por un diseño con 3 cabinas superiores de igual geometría y 2 cabinas inferiores de distinta geometría a las superiores,

esto dado a que se quiere verificar que la reubicación será posible independientemente de la geometría de la cabina siempre y cuando los centroides de estas se encuentren en los límites de acción del brazo robótico, es decir, que sean alcanzables y suficientemente más grandes que la pinza cerrada y los objetos a reubicar. Así, se diseñó un estante de altura 1.512 m (z), 0.84 de largo (x), ancho de 0.29 (y), considerando un espesor de 0.01 m para el espaciado entre cabinas.

Figura 2.17

Estante para Reubicación

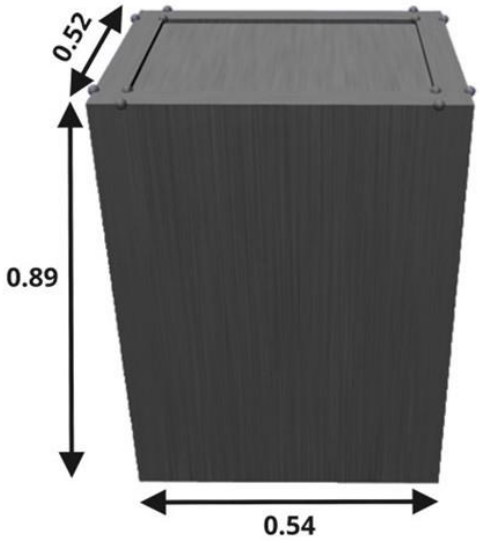


- Mesa de trabajo

Se trabajó con una mesa para reaparición de piezas de 0.89 m de altura, 0.54 de largo (en x) y 0.52 de ancho (en y).

Figura 2.18

Mesa de Trabajo para Reaparición de Piezas

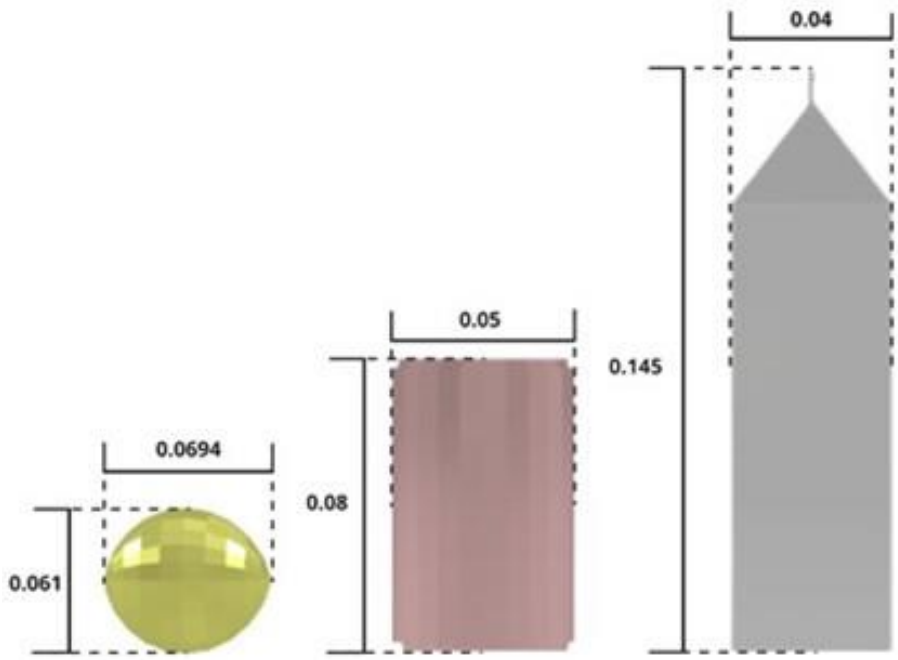


- Objetos

Para el proyecto, se contó con 3 objetos para el entrenamiento del agente inteligente de reubicación, para lo cual se diseñaron objetos de uso cotidiano en inventor, un limón, una lata y un cartón de leche, tal que se cuenten con diferentes geometrías de objetos y el agente pueda ser útil para diferentes situaciones.

Figura 2.19

Dimensiones de Objetos para Reubicación

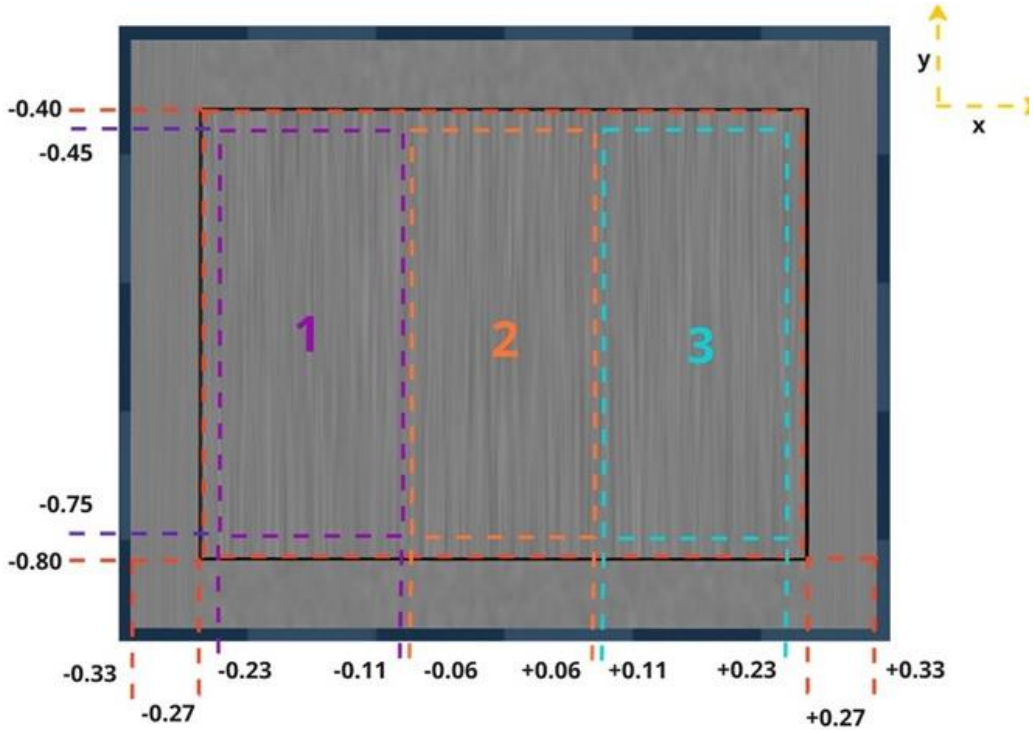


2.2.5.2 Reaparición de objetos

Los objetos se encuentran limitados a regiones de reaparición de tal forma que no se superpongan al momento de desarrollar los algoritmos. Es por ello por lo que se toman las regiones de la Figura 2.20 como consideración.

Figura 2.20

Límites de Reaparición



- Región 1: Reaparición de Limón
    - Rango: x de [-0.23, -0.11], y de [-0.75 a -0.45], z de [0.90 a 1]
- Región 2: Reaparición de Lata
    - Rango: x de [-0.06, +0.06], y de [-0.75 a -0.45], z de [0.90 a 1]
- Región 3: Reaparición de Cartón de Leche
    - Rango: x de [+0.11, +0.23], y de [-0.75 a -0.45], z de [0.90 a 1]

2.2.5.3 Lenguaje de Marcado Extensible (XML)

Como se explicó anteriormente, MuJoCo es una biblioteca de física de alta precisión para simulación dinámica y cinemática de robots y sistemas articulados, como tal, está configurada para la descripción de modelos físicos y entornos de simulación mediante lenguaje XML (Extensible Markup Language).

Lo característico de este lenguaje con relación a MuJoCo es que se pueden modificar las siguientes variables para la generación de escenarios con parámetros predeterminados e interpretables por el simulador tales como:

1. mjmodel: Elemento raíz que engloba toda la descripción del modelo.
2. compiler: Define opciones para el compilador MuJoCo.
3. default: Establece valores predeterminados para elementos específicos.
4. option: Permite configurar opciones específicas del modelo.
5. asset: Especifica archivos de recursos externos necesarios para el modelo.
6. worldbody: Define cuerpos, articulaciones, geometría y actuadores del modelo.
7. actuator: Define los actuadores, como motores o fuerzas aplicadas.
8. sensor: Define sensores que proporcionan información sobre el estado del modelo.
9. contact: Especifica propiedades de detección de contacto entre cuerpos.
10. visualization: Configura la apariencia visual de objetos en el simulador.
11. equality: Define restricciones de igualdad en el modelo.
12. tendon: Define tensores que controlan directamente las articulaciones.
13. sensor: Permite agregar sensores personalizados para obtener información específica.
14. customfield: Define campos personalizados para acceder en el código interactivo.

El archivo XML de MuJoCo es esencial para la simulación precisa de sistemas articulados y se utiliza en aplicaciones de robótica, biomecánica y aprendizaje por refuerzo, entre otras áreas. Cada elemento y atributo dentro del archivo tiene un propósito específico para definir el comportamiento y las propiedades del modelo físico que se simulará [18].

Dentro de este archivo XML, se establecieron la geometría y las relaciones del robot, de tal forma que los elementos del entorno se organicen en una jerarquía o estructura de árbol, además

de establecerse sus atributos tanto visuales como físico, para el caso del Robot y Pinza tenemos una base de la que parten a manera de árbol todas las demás componentes para desempeñar en conjunto el funcionamiento del elemento conocido como Brazo Robótico, mientras la mesa de trabajo y el estante son únicamente elementos representados por cuerpos geométricos en el espacio. Las jerarquías dentro de este archivo se organizan de la siguiente manera y con los siguientes nombres dentro del código:

- Mesa de trabajo: pick_box.
- Estantería en el entorno: shelf.
- Robot UR5 y Pinza
- Base del Robot: base_link.
- Articulación del hombro del robot: shoulder_link:
- Articulación del brazo superior: upper_arm_link.
- Antebrazo del robot: forearm_link.
- Articulaciones de la muñeca: wrist_1_link, wrist_2_link, wrist_3_link.
- Extremo del efector (End-Effector): ee_link.
- Palma de la Pinza: gripperpalm.
- Dedos de la Pinza: gripperfinger_1_link_x, 2_link_x, 3_link_x.

2.2.5.4 Formato de Descripción de Robótica Unificada (URDF)

Es un formato de descripción de robots ampliamente utilizado en el contexto de la robótica y específicamente con el framework ROS (Robot Operating System). URDF es un formato basado en XML que se utiliza para describir la estructura y la geometría de los robots, incluyendo sus articulaciones, enlaces, sensores y propiedades físicas [13].

El espacio de simulación de MuJoCo se desarrolló usando XML, sin embargo, debido a la aplicabilidad en distintos simuladores del formato URDF, el brazo robótico fue desarrollado en este formato, de esta forma, la dinámica y cinemática del robot, se definió en base al modelo desarrollado por Franceschetti (2018) y se realizaron modificaciones para la posterior implementación de la Pinza. En este, se establieron los tipos de Juntas, Momentos de Inercia, Masas de los eslabones y cinemática del Robot, de forma general tenemos lo siguiente definido dentro del código del archivo:

- Robot UR5: Brazo robótico de 6 DOF con carga útil máxima de 5 kg.
- Base de Pinza de 3 dedos Robotiq modelo S: Pinza subactuada de 4 DOF.
- Eslabones (Links): Hay 52 eslabones en total.
- Articulaciones (Joints): Hay 72 articulaciones.
- Plugins de Gazebo: gazebo_ros_control.
    - Dado que la finalidad de este archivo URDF, era su implementación en ROS y no MuJoCo, contamos con plugins nativos de compatibilidad.

De igual forma, respecto al establecimiento de los tipos de juntas y jerarquía de eslabones padre e hijo sobre las primeras 7 articulaciones del archivo URDF tenemos las siguientes:

1. shoulder_pan_joint (articulación del hombro):
    - Tipo: Revoluta (rotacional)
    - Eslabón padre: base_link
    - Eslabón hijo: shoulder_link
2. shoulder_lift_joint (articulación de elevación del hombre):
    - Tipo: Revoluta
    - Eslabón padre: shoulder_link

- Eslabón hijo: upper_arm_link

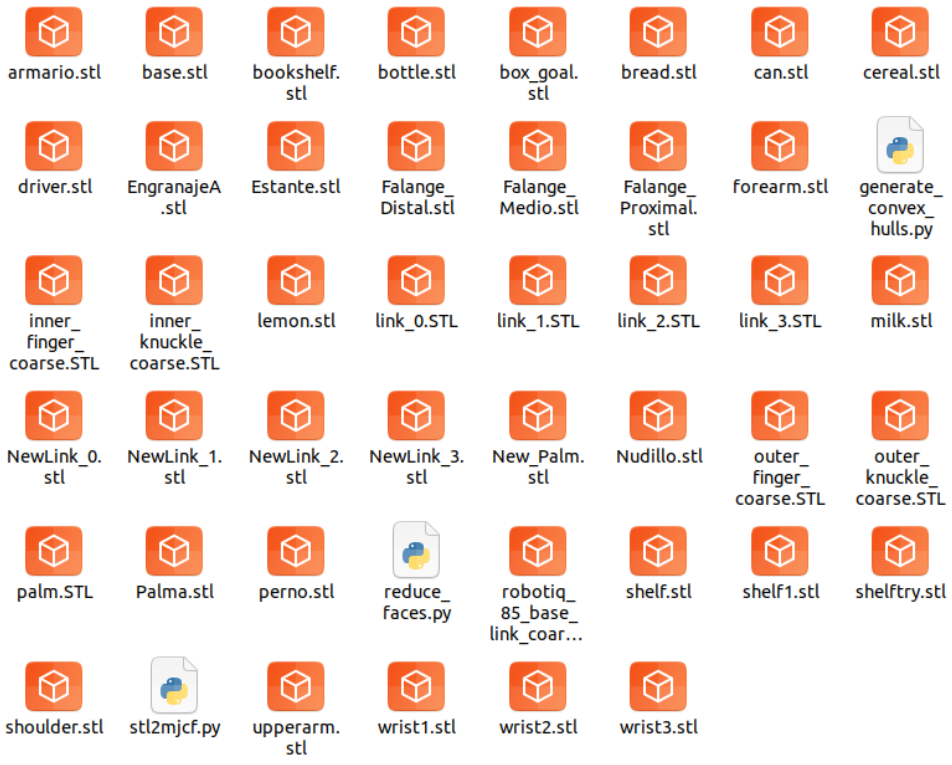
3. elbow_joint (articulación de codo):
    - Tipo: Revoluta
    - Eslabón padre: upper_arm_link
    - Eslabón hijo: forearm_link
4. wrist_1_joint (articulación de muñeca 1):
    - Tipo: Revoluta
    - Eslabón padre: forearm_link
    - Eslabón hijo: wrist_1_link
5. wrist_2_joint (articulación de muñeca 2):
    - Tipo: Revoluta
    - Eslabón padre: wrist_1_link
    - Eslabón hijo: wrist_2_link
6. wrist_3_joint (articulación de muñeca 3):
    - Tipo: Revoluta
    - Eslabón padre: wrist_2_link
    - Eslabón hijo: wrist_3_link
7. ee_fixed_joint (articulación de junta fija):
    - Tipo: Fijo
    - Eslabón padre: wrist_3_link
    - Eslabón hijo: ee_link

2.2.5.5 Lenguaje Triangular Estándar (STL)

El formato STL es comúnmente utilizado para representar modelos tridimensionales sólidos y superficies en aplicaciones de diseño, impresión 3D y gráficos por computadora. Es así como, para la aplicación actual, se crearon los objetos de geometrías complejas como los objetos para ser reubicados y el armario para reubicación, en Inventor y posteriormente se exportaron en formato. STL, con lo cual dentro de la programación del archivo .XML es posible referenciar a estos archivos de tipo malla para ser cargados dentro del entorno de MuJoCo.

Figura 2.21

Archivos STL creados para el entorno simulado



2.3 Visión Robótica

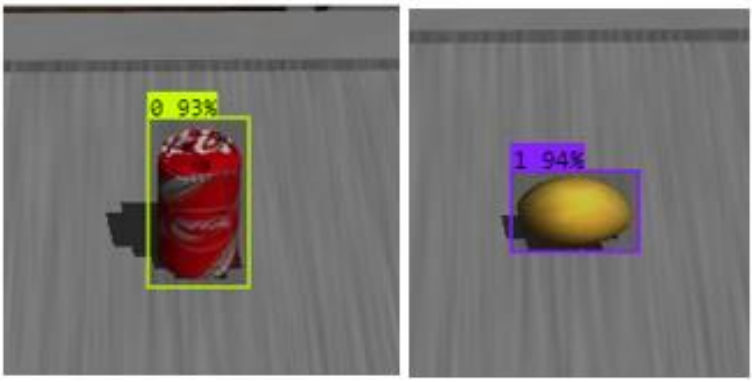
2.3.1 Base de datos

Lo primero que se realizó para llevar a cabo con la visión robótica es tener una base de datos útil para nuestro proyecto, en este caso son las imágenes de los objetos que vamos a

identificar. En total se subieron 1000 imágenes tomadas por nosotros en el entorno simulado de MuJoCo a la base de datos para identificar los objetos, los cuales son una lata de Coca cola, un cartón de leche y un limón. Se hizo toma de imágenes de estos elementos por separados de distintos ángulos y además de los 3 elementos juntos en una imagen. De esta manera podemos asegurar tener una precisión por encima de 90% dado por el algoritmo de YOLO. Dado que si los objetos presentes en la mesa de trabajo se asemejan en un 80% o mayor se identifica el objeto.

Figura 2.22

Ejemplo de detección de objetos



Se define dentro de la base de datos de roboflow el porcentaje de imágenes que serán para entrenamiento, validación y prueba, siendo estos 80%, 13% y 7% respectivamente.

2.3.2 You Only Look Once (YOLO)

Las redes convolucionales (CNNs) son usadas con mayor frecuencia para tareas de clasificación, además que son adecuadas por su rendimiento superior con entradas de imagen, voz o señales de audio. Esto con respecto a otras redes neuronales. Es por eso que se usa CNN para la clasificación, identificación y localización de objetos.

YOLO (You Only Look Once) es una serie de algoritmos que constituyen un sistema de detección de objetos en tiempo real. Puede detectar objetos en imágenes y videos, tanto seres vivos como objetos inanimados. Las principales funciones de YOLO es la clasificación, detección y

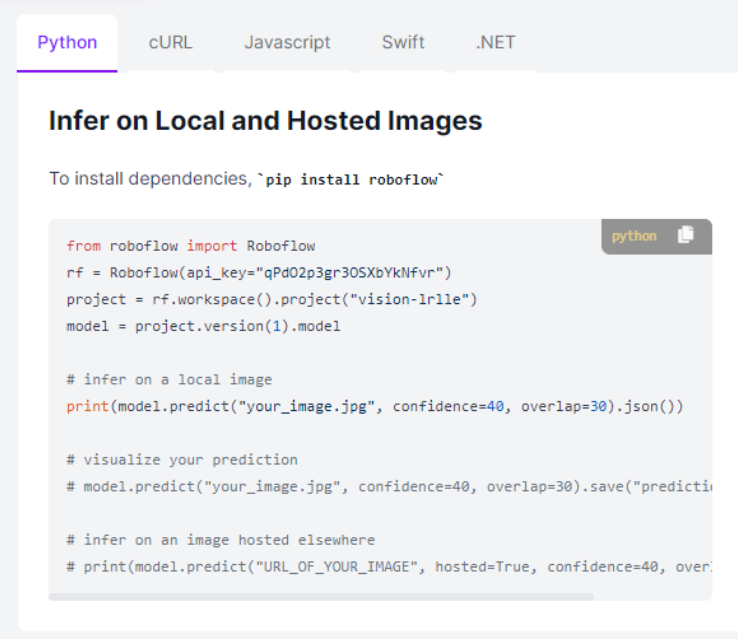
segmentación de objetos [10]. Lo que está detrás de esta serie de algoritmos llamada YOLO son precisamente las redes convolucionales, ya que son las encargadas de clasificar los objetos dentro de las imágenes que recibe el algoritmo.

2.3.3 Detalles de entrenamiento

La implementación de roboflow nos facilita su uso para la detección de imágenes ya que siendo una página la cual usa distintos algoritmos para la identificación y clasificación de imágenes, entre estos esta YOLO v8. El proceso que se siguió es primero crear una base de datos dentro de roboflow, donde se subió varias imágenes de los objetos a identificar luego anotar cada imagen identificando manualmente los objetos y creando distintas clases según el tipo de objetos que necesitamos. Luego se procede a entrenar con estas imágenes y se crea el modelo en el cual podemos subir imágenes y probar su precisión. Este modelo creado en esencia son las redes convolucionales por las cuales se procesarán las imágenes que entren al algoritmo debido que con el entrenamiento previo se programó las clases en las que se clasificaran los objetos que se encuentren en las imágenes. Una vez tengamos resultados de identificación satisfactorios se obtiene el código del modelo.

Figura 2.23

Código del modelo obtenido listo para implementar



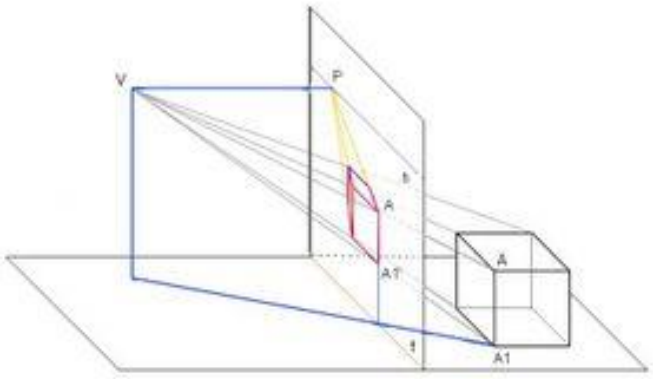
En la figura 2.23, podremos ver un pedazo de código proporcionado por roboflow, dicho código se usa dentro del código de control para hacer el llamado del modelo dentro de roboflow y así tener información de los objetos que se encuentran dentro del entorno simulado, esto debido a que los objetos reaparecen aleatoriamente como hemos determinado.

2.3.4 Matriz de transformación

Para la obtención de la localización de los objetos es necesario tomar en cuenta el cambio de perspectiva que puede ocurrir, debido a que las imágenes se obtendrán desde la posición en la que se encuentra la cámara, además de la representación de los objetos dentro de un plano es por lo que para poder guiarnos se usa un modelo de proyección central, un ejemplo está en la figura.

Figura 2.24

Ejemplo de perspectiva de un objeto dentro de un plano



La figura 2.24, nos muestra un ejemplo de la representación a la que queremos llegar donde como un objeto 3D se representa dentro de un plano dependiendo del punto de percepción, en este caso ese punto será la cámara RGB-D de donde obtendremos esa información.

Como queremos cambiar de perspectiva ya que necesitábamos una que se ajustara a la ubicación de la cámara se tuvo que utilizar la ecuación 2.1.

$$T = \begin{bmatrix} \frac{f}{\rho w} & 0 & u_o \\ 0 & \frac{f}{\rho h} & v_o \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

Donde,

f es el punto focal de la cámara.

ρw es el ancho de los pixeles.

ρh es el alto de los pixeles.

u_o, v_o es el punto central de la imagen.

Una vez conocemos la matriz de transformación, seguimos la ecuación 2.2, la cual es la representación de como cambiar la perspectiva de una coordenada tridimensional expresada en pixeles.

$$P = T^{-1}p \tag{2.2}$$

Donde,

P es la coordenada del punto en el espacio.

p es la coordenada el punto en la imagen

T^{-1} es la matriz de transformación inversa.

Para poder realizar la transformación de las coordenadas que se necesitan, se debe emplear una matriz de transformación la cual ayudara a cambiar la posición de la perspectiva mediante traslación y rotación, esta matriz a emplearse de puede ver en la ecuación 2.3.

$$T = \begin{bmatrix} R_{3x3} & P_{1x3} \\ f_{3x1} & w_{(1x1)} \end{bmatrix} \tag{2.3}$$

En este caso R es la matriz de rotación, P el vector de traslación, f el vector de perspectiva y w una constante de escalado. Juntando estas distintas matrices en una tenemos la matriz de transformación que necesitamos.

El cambio de sistema se puede expresar como se ve en la ecuación 2.4, ya que lo único que necesitábamos era la matriz de transformación y los puntos de que ya conocemos.

$$p_0 = T_{cambio}\, p_{cambio} \tag{2.4}$$

p_0 es el punto respecto al sistema de referencia del robot.

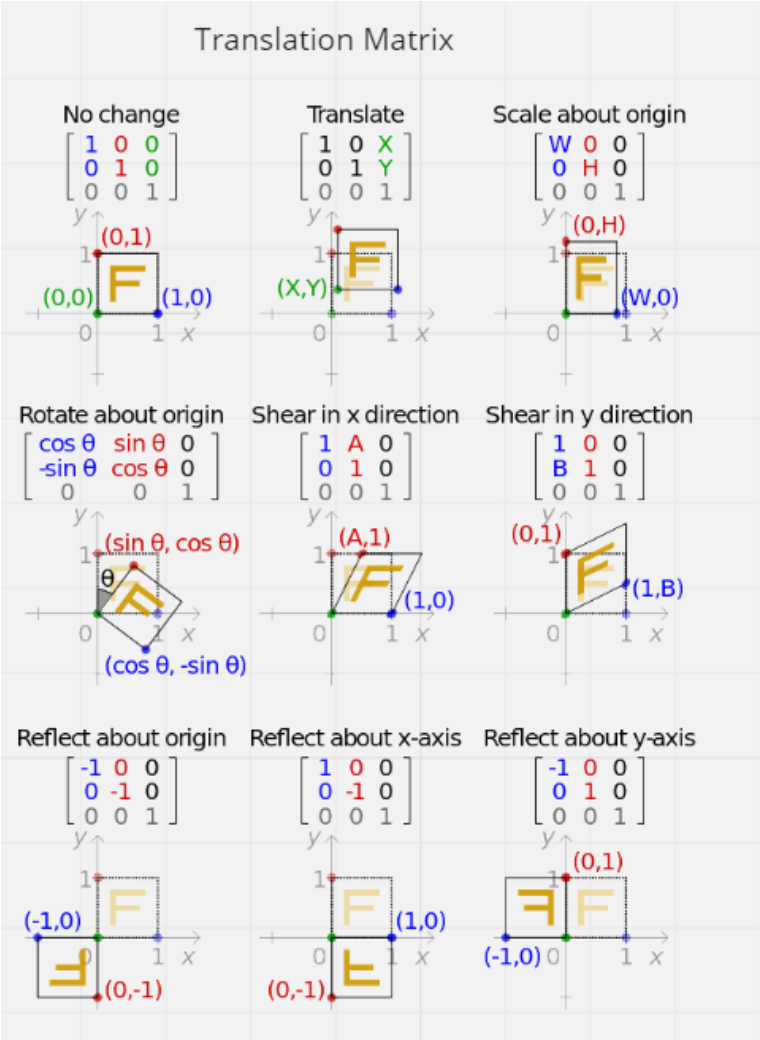
T_{cambio} es la matriz de transformación.

p_{cambio} es el punto respecto al sistema de referencia de la cámara.

Ya teniendo estos conocimientos de a lo que se quiere llegar hay que obtener las matrices que necesitamos. Las cuales son la matriz de rotación y la de traslación.

Figura 2.25

Obtención de la matriz de traslación



En la figura 2.25 podemos ver el paso a paso del concepto de lo que se tiene que hacer para poder llegar a una vector de traslación y rotación ya que habrá que mover de distintas maneras el plano.

Dentro de la matriz de rotación seguimos las ecuaciones que se logran ver en la figura 2.26. Ya que para cada posición de la matriz tiene su propia ecuación.

Figura 2.26

Estructura de la matriz de rotación

$$
\begin{aligned}
R &= R_z(\phi)R_y(\theta)R_x(\psi) \\
&= \begin{bmatrix} \cos\theta\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \cos\theta\sin\phi & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi \\ -\sin\theta & \sin\psi\cos\theta & \cos\psi\cos\theta \end{bmatrix}
\end{aligned}
$$

Una vez empleado cada uno de los pasos y usando los datos respectivos se obtuvo la siguiente matriz de transformación.

$$
T = \begin{bmatrix} 1 & 0 & 0 & 0.15 \\ 0 & -0.372 & 0.928 & -2 \\ 0 & -0.928 & -0.372 & 1.7 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

2.4 Control

2.4.1 Espacio de Acciones

El espacio de Acciones de nuestro proyecto se encuentra delimitado en 5 acciones, donde las primeras 3 representan la ubicación objetivo del centroide de la palma de la pinza del robot, la cuarta acción representa el giro de la pinza y la última acción representa la apertura de la pinza.

- Espacio de Acciones: [X, Y, Z, Giro, Apertura]

  Donde este espacio de acciones de tipo caja se encuentra delimitado de la siguiente forma:

- Límite Inferior: [-0.24, -0.77, 0.88, -1.6, -0.4]
- Límite superior: [0.24, -0.44, 1.3, 1.6, 0.4]

Figura 2.27

Acción de coordenadas X, Y, Z para movimiento de Pinza

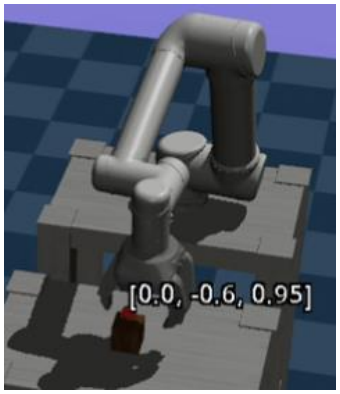
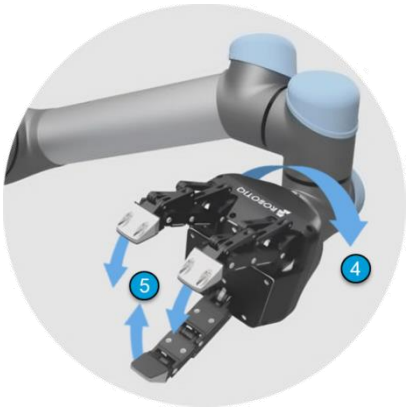


Figura 2.28

Acciones de Giro y Apertura para Pinza



2.4.2 Método Epsilon-Greedy

El método Epsilon-Greedy es una estrategia de selección de acciones utilizada en el aprendizaje por refuerzo y en otros problemas de toma de decisiones. Se utilizó para equilibrar la exploración (probar acciones desconocidas) y la explotación (seguir las acciones que parecen ser las mejores según la información actual) [7].

El proceso que se utiliza es el siguiente:

1. Etapa de Exploración vs. Explotación: El agente robótico toma decisiones basadas en dos enfoques: exploración y explotación.

2. Epsilon (ε): Se utiliza un parámetro llamado "epsilon" (ε) que es un valor entre 0 y 1, que controla la proporción de tiempo en la que el agente explora versus explota.
3. Selección de Acciones:
    a. Con probabilidad ε: El agente elige una acción al azar (exploración).
    b. Con probabilidad 1-ε: El agente elige la acción que se cree que es la mejor según la información actual recopilada por experiencias (explotación).

De esta forma permitimos al agente robótico aprender nuevas acciones al explorar, lo que es crucial para descubrir acciones óptimas en un entorno desconocido o en evolución. Además, el valor de épsilon (ε) puede variar en base al tiempo, inicialmente podemos tener un valor alto que favorezca la explotación durante la etapa de almacenamiento de experiencias y posteriormente disminuye gradualmente a medida que adquiere experiencias. Aplicado a nuestra programación, definimos una función epsilon_greedy que cuenta con las siguientes etapas:

1. Política Epsilon-Greedy: Se utiliza un enfoque epsilon-greedy para decidir entre la exploración y la explotación. Con probabilidad ϵ (“self.eps_threshold” dentro del código con un valor de 1.0), se elige una acción al azar. Con probabilidad $1-\epsilon$, se elige la mejor acción según el actor de nuestra red neuronal.
2. Adición de Ruido: En el caso de que elijas la "mejor" acción (acción greedy), se añade ruido a esta acción, considerando un ruido de exploración (“self.exploration_noise”) de 0.1, lo que se traduce en la línea de codigo: actions += torch.normal(0, self.actor[1].action_scale * self.exploration_noise). Es decir, se genera un ruido normal con una media de 0 y una desviación estándar que se escala con la constante “self.exploration_noise” y este ruido se suma a la acción seleccionada por el actor.

3. Restricción de Acción: Después de añadirse el ruido, restringuimos la acción para que esté dentro de los límites permitidos del espacio de acción, en la línea de código: actions.cpu().numpy().clip(self.env.action_space.low, self.env.action_space.high).

2.4.3 Redes Neuronales Residuales

ResNet (Residual Neural Network) es una arquitectura de red neuronal profunda que introdujo un enfoque innovador para facilitar el entrenamiento de redes extremadamente profundas.

Se caracteriza por el uso de bloques residuales, que son unidades de construcción que permiten que la red aprenda las diferencias entre las representaciones del estado actual y un estado previo (residuos). [19]

Para nuestro proyecto, se empleó este modelo de red neuronal estandarizado, como lo es Resnet, la cual surge en el año 2015, y se implementó mediante el uso de los modelos definidos dentro de la librería de torch y torchvision, para Python, al ser un modelo estandarizado, es reutilizable y escalable, dentro de las ventajas de este modelo, se pudo encontrar las siguientes según He (2015):

- Facilita el Entrenamiento de Redes Profundas: Resuelve problemas como el desvanecimiento del gradiente, permitiendo entrenar modelos más profundos.
- Conexiones Residuales: Mejoran el flujo de información y gradientes a través de la red.
- Mejora en el Rendimiento: Generalmente supera a arquitecturas más antiguas en diversas tareas.
- Modularidad: Fácil de extender con más bloques residuales.
- Versatilidad: Útil en una amplia gama de aplicaciones, no solo en visión por computadora.

- Eficiencia Computacional: Diseño eficiente en términos de cálculo y número de parámetros.
- Ampliamente Adoptado: Muchas variantes y aplicaciones en la comunidad.
- Fácil de Implementar: Disponible en varias bibliotecas de aprendizaje profundo.

Nuestra arquitectura ResNet utiliza pilas de estos bloques residuales para construir redes neuronales profundas. Además, utiliza técnicas como el uso de capas de convolución 1x1 para ajustar la dimensionalidad, el uso de capas de agrupamiento (pooling) para reducir el tamaño espacial y el uso de batch normalization para estabilizar y acelerar el entrenamiento.

De manera general, contamos con dos módulos que abarcan las capas, un módulo de percepción, el cual es el encargado de recibir las imágenes como entrada en el caso del actor, mientras en el caso del crítico, la entrada fue un arreglo de imágenes junto a acciones y generar un tensor de salida normalizado para el módulo de agarre, de donde se tuvo una salida de acción para el actor, mientras para el crítico se tuvo un valor de Acción-Estado Q.

En la clase “Basic Block”, los tensores de entrada pasan a través de dos capas de convolución 3x3, cada una seguida de una capa de Batch Normalization y una función de activación ReLU. El resultado final es el tensor de salida del bloque BasicBlock, que luego puede pasar a la siguiente capa de la red.

El módulo de agarre fue una parte importante de la arquitectura que se encarga de procesar las características extraídas por el módulo de percepción y generar la salida final del modelo. La naturaleza específica de las operaciones realizadas en este módulo depende del tipo de módulo (actor o critic) y de la tarea que el modelo está diseñado para resolver. Tuvimos dos casos:

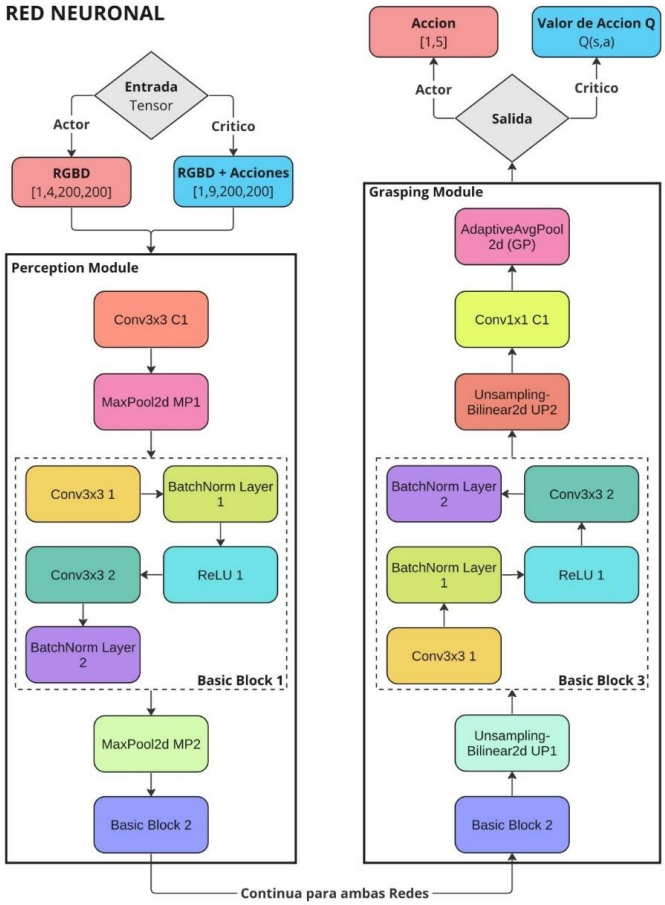
- Caso actor: Se aplica la función de activación nn.Tanh() para asegurar que los valores de salida estén en el rango entre -1 y 1, dado que es una acción. También se aplica una

rescalación (“action_scale” y “action_bias”) para ajustar la salida a un rango específico necesario para la tarea.

- Caso critic: Se aplica la función de activación nn.Sigmoid() para garantizar que los valores de salida estén en el rango entre 0 y 1, dado que se desea una salida de valor de Acción-Estado Q, los cuales siempre serán positivos.

Figura 2.29

Diseño de Red Neuronal Resnet



2.4.4 Aprendizaje por Refuerzo Profundo

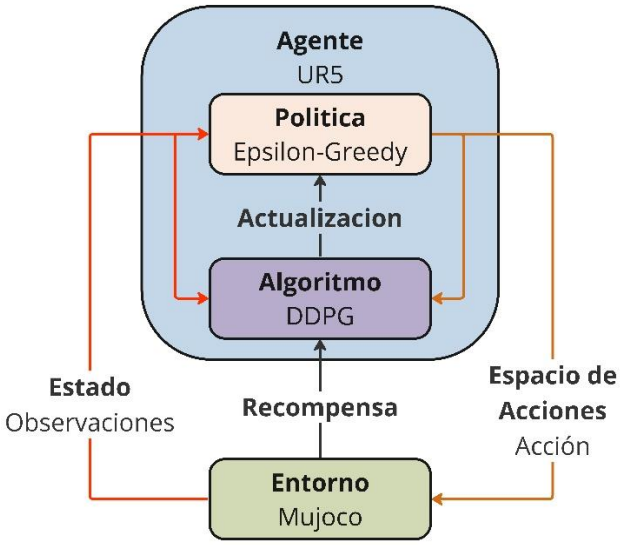
Conociendo que, el aprendizaje por refuerzo profundo es un área de la inteligencia artificial y el aprendizaje automático que combina conceptos del Aprendizaje por Reforzamiento (Reinforcement Learning, RL) con técnicas de Aprendizaje Profundo (Deep Learning). En base a

lo definido en el Capítulo 1, sabemos que un agente de RL interactúa con un entorno y toma acciones para maximizar una recompensa acumulada a lo largo del tiempo, mediante el aprendizaje por ensayo y error. Mientras, el Aprendizaje Profundo, es una técnica que utiliza redes neuronales profundas para aprender representaciones de alto nivel a partir de datos no estructurados o semiestructurados, como imágenes, texto o señales. [17]

Es así como sobre esta teoría fundamentamos el funcionamiento de nuestro proyecto, mismo que se ilustra en la Figura 31, donde iniciamos tomando la observación del estado inicialmente, es decir, lo que obtenemos de la cámara RGB-D, para posteriormente mediante este estado inicial del agente, comprobar si en el paso actual nos encontramos en una zona de exploración o explotación empleando el método de Epsilon-Greedy, en caso de encontrarnos en la exploración, se genera una acción aleatoria mediante el uso de redes neuronales, en este caso una red de Actor, en la cual ingresa un estado y sale una acción a, la cual se ejecuta en el entorno de simulación, para posteriormente ser evaluada en la red de un crítico que recibe el estado y la acción para tener como salida un valor de Acción-Estado Q, estos valores de acción a y valor de Acción-Estado Q, se almacenan en un búfer de experiencias junto al valor de recompensa del entorno y las observaciones iniciales y siguiente, una vez llenado el buffer de experiencias según se designó en los hiperparámetros, se entra a la etapa de explotación, donde el agente evalúa la calidad de la toma de decisión del actor y así entrenar tanto al actor como al crítico para que realicen mejores decisiones en el futuro.

Figura 2.30

Aprendizaje Profundo por Reforzamiento



2.4.5 Algoritmo Deep Deterministic Policy Gradient

El algoritmo de DDPG (Deep Deterministic Policy Gradients) es una técnica popular en el campo del Deep Reinforcement Learning (DRL) utilizado para resolver problemas de control y acción continua. Este es una variante del algoritmo Actor-Critic, donde "Actor" se refiere a la política que elige acciones en función de las observaciones del entorno, y "Critic" se refiere a la función de valor que estima la recompensa esperada a largo plazo a partir de un estado dado y una acción tomada [16]. Paso a paso las fases de nuestro algoritmo serían las siguientes:

1. Inicialización: Se inicializan los parámetros de la política (Actor) y la función de valor (Critico) utilizando redes neuronales profundas. También se inicializan los parámetros del ruido, que se utilizan para agregar exploración estocástica a las acciones del actor.
2. Experiencia del buffer: Se crea un buffer de experiencia para almacenar las transiciones observadas en el entorno. Cada transición consiste en un estado, una acción, la recompensa recibida, el siguiente estado y una señal que indica si la transición termina en un estado final.

3. Interacción con el entorno: Se inicia la interacción entre el agente y el entorno. En cada paso de tiempo, el agente elige una acción utilizando la política actualizada y la agrega ruido para la exploración. Luego, se toma la acción en el entorno y se observa el siguiente estado y la recompensa.
4. Actualización del buffer: Se almacenan las transiciones (estado, acción, recompensa, siguiente estado) en el buffer de experiencia.
5. Actualización de la red: Se muestrea un lote de transiciones del buffer de experiencia. Se utilizan estas muestras para calcular los objetivos para la actualización tanto de la política (Actor) como de la función de valor (Critico).
6. Actualización del Critico: El Critico se entrena para minimizar el error cuadrático medio entre las estimaciones de valor actuales y los objetivos calculados utilizando el método de Descenso de Gradiente Estocástico (SGD).
7. Actualización del Actor: El Actor se entrena para maximizar el valor esperado estimado por el Crítico para las acciones que produce el Actor en un estado determinado. Esto se realiza a través de la regla de Descenso de Gradiente Estocástico (SGD) aplicada al gradiente de la política.
8. Actualización de los pesos de las redes: Se actualizan los pesos de las redes de Actor y Critico utilizando los gradientes calculados.
9. Actualización de los parámetros del ruido: Los parámetros del ruido utilizado para agregar exploración estocástica a las acciones del Actor se actualizan de manera gradual.
10. Repetición: Se repiten los pasos 3 al 9 durante varios episodios de entrenamiento o hasta que se alcance un criterio de parada.

Tal como se puede apreciar a continuación, se utilizó como base la lógica del algoritmo DDPG en seudocódigo para su implementación dentro de Python mediante el uso de las librerías de Torch y stable_baselines3.

Figura 2.31

Algoritmo DDPG Seudocódigo tomado de "Control continuo con aprendizaje de refuerzo profundo" (Lillicrap et al, 2015)

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

El DDPG ha demostrado ser efectivo para problemas de control y acción continua, y se ha aplicado en una variedad de tareas, como control de robots, juegos y simulación. Sin embargo, el DDPG también tiene sus desafíos, como la estabilidad en el entrenamiento y la gestión de la exploración. [16]

Es por esto por lo que, usualmente al emplear algoritmos como DDPG, donde únicamente se depende de un crítico para evaluar las acciones en relación con un valor de Acción-Estado Q, puede derivar en retardos en el aprendizaje o redundancias, además de ser la causa de que se

requieran una cantidad considerable de episodios o una gran cantidad de pasos para el entrenamiento efectivo.

2.4.6 Función de recompensa

En el contexto del Deep Reinforcement Learning (DRL), la función de recompensa (reward function en inglés) es una función que asigna una puntuación o valor numérico a las diferentes situaciones o estados que el agente puede experimentar durante la interacción con el entorno. Esta función es crucial en el aprendizaje por refuerzo, ya que guía al agente para que aprenda a tomar decisiones que maximicen las recompensas acumuladas a lo largo del tiempo.

La función de recompensa define los objetivos del agente y puede variar según la tarea o el problema que se esté abordando. La elección adecuada de la función de recompensa es esencial para lograr un buen desempeño en el aprendizaje y el comportamiento deseado del agente. Una buena función de recompensa debe estar diseñada cuidadosamente para fomentar el comportamiento deseado y evitar resultados no deseados. [16]

Diseñar una función de recompensa adecuada puede requerir iteración y ajustes a medida que el agente interactúa con el entorno y se realiza el proceso de aprendizaje. Además, algunas veces el diseño de la función de recompensa puede ser un desafío, y en esos casos, se pueden utilizar técnicas como el aprendizaje de recompensas desde la demostración o el aprendizaje por imitación para obtener una función de recompensa más efectiva a partir de ejemplos proporcionados por un experto [16].

2.4.7 Diseño de funciones de recompensa

Recompensa 1: Recompensa de Distancia Euclidiana

Consideramos que $p1$ y $p2$ son los puntos en el espacio tridimensional (x,y,z) que describen la posición actual del centroide de la palma de la pinza del brazo robótico y la posición del

centroide del objeto a reubicar. La finalidad de esta función de recompensa es determinar la proximidad de la pinza del robot con el objeto a reubicar objetivo, para ello, se empleó el concepto de la distancia euclidiana, de tal forma que sea posible determinar qué tan distantes se encuentran el brazo robótico y el objeto, para así, que se recompense en mayor medida cuanto más cerca se encuentre el brazo robótico del objeto.

Para la primera función de recompensa, empleamos la siguiente formula:

$$r_1 = sensibilidad \ * \ e^{-d}$$

En donde d, es conocido como la distancia euclidiana, que viene dada por la formula:

$$d = \ \|(p1 - p2)\|$$

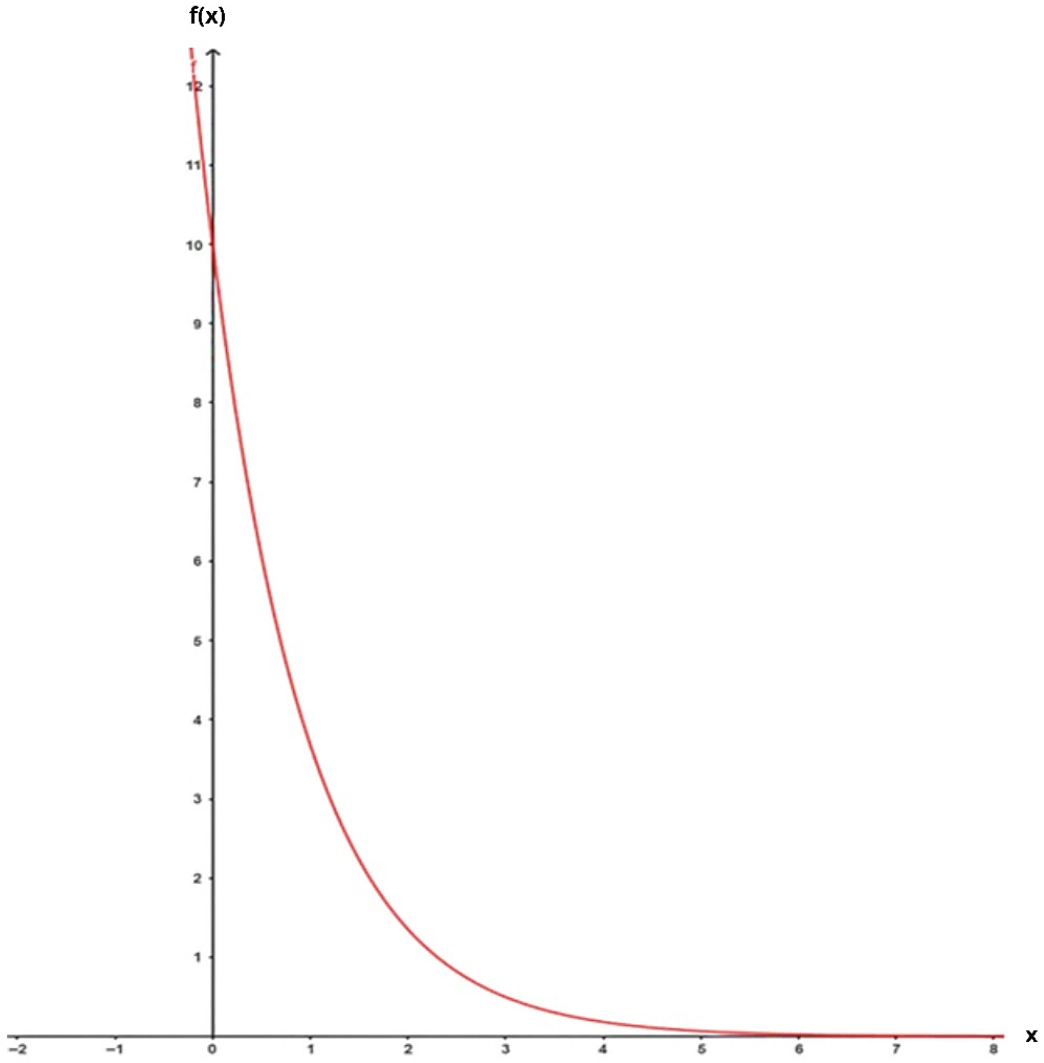
Se multiplica por un valor de sensibilidad de tal forma que nos aseguramos de que el valor máximo que pueda tomar la recompensa mediante su función sea 10, con lo que tenemos una función principal de la forma (consideración: d es lo mismo que x):

$$f(x) = 10e^{-x}$$

En la siguiente Figura 2.32, se muestra la gráfica de la función, en donde se puede apreciar la tendencia de la curva a crecer a medida nos acercamos al cero en x, es decir, cuando la distancia euclidiana sea cero, por tanto, cuando ambos centroides estén en la misma ubicación.

Figura 2.32

Grafica de Función



- Consideraciones para Recompensa 1:

Se tuvo que para acelerar la optimización de la recompensa 1 dentro del algoritmo, para ello, se multiplicaron las recompensas obtenidas por una constante en base a distintos intervalos, de tal forma que se tienda al 0 en x para la gráfica y por ende al valor mínimo de distancia al maximizar el valor de función.

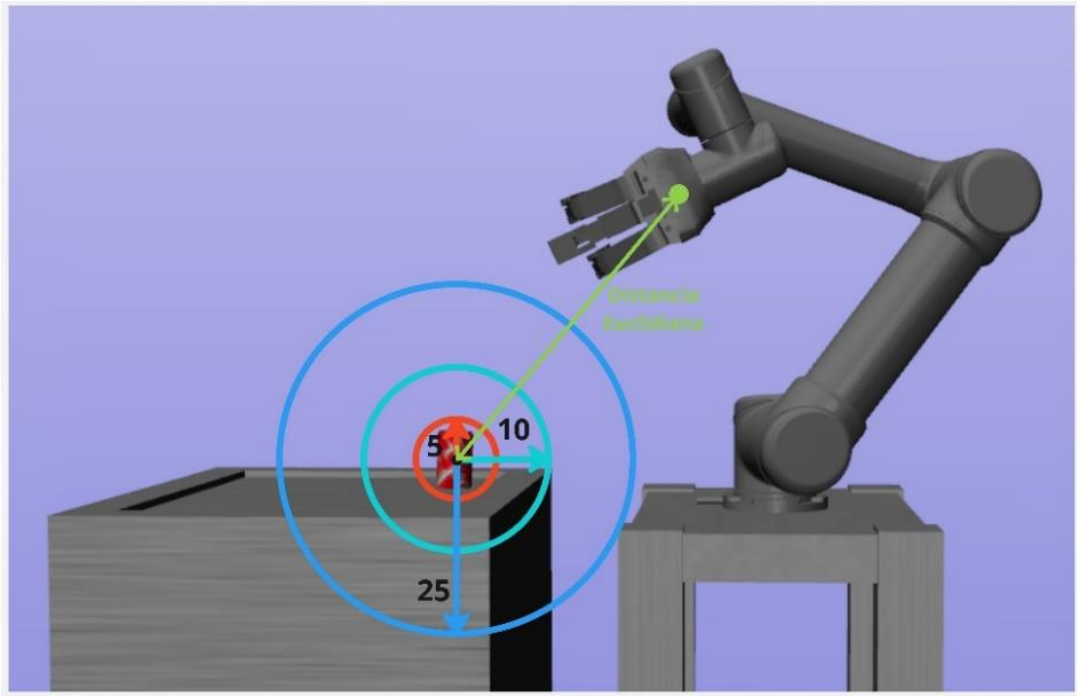
Para los intervalos de:

- $0.10 < d \leq 0.25$: La recompensa 1 se multiplica por 1.5
- $0.05 < d \leq 0.10$: La recompensa 1 se multiplica por 2.5
- $d \leq 0.05$: La recompensa 1 se establece en el valor de 250

De esta forma, se aseguró que el algoritmo intente tender a 0 para obtener la máxima recompensa, cabe recalcar que los valores de d oscilan entre 0 y 25.

Figura 2.33

Función de Recompensa 1



Recompensa 2: Recompensa de Altura

La finalidad de esta función de recompensa es determinar inicialmente si el Brazo Robótico se encuentra a la suficiente altura del objeto para que le sea posible efectuar la cinemática necesaria para descender y proceder al agarre del objeto.

$$h_{diff} = \left|\mathrm{h_{obj}} - \mathrm{h_{gp}}\right|$$

- h_{diff}: Diferencia de Altura de Centroide entre Objeto y Palma

- h_{obj}: Altura de centroide de Objeto (z)
- h_{gp}: Altura de centroide de Palma de Pinza (z)

Inicialmente, nos aseguramos de que $h_{diff} \leq 0.38$, y que la pinza tiene una posición en el eje Y inferior a -0.30, de esta forma nos aseguramos de que efectivamente la pinza se encuentra en el lado de la Mesa de Trabajo.

Una vez nos aseguramos de esto, procedemos con las condicionales para obtener la recompensa 2:

$$h_{diffobj} = \left|h_{obj} - h_{cobj}\right|$$

- $h_{diffobj}$: Desplazamiento de Centroide de Objeto
- h_{obj}: Altura de centroide de Objeto (z)
- h_{cobj}: Altura de centroide de Objeto estándar (0.93 en z)

De lo cual tenemos que la recompensa 2 será:

$$r_2 = h_{diffobj}$$

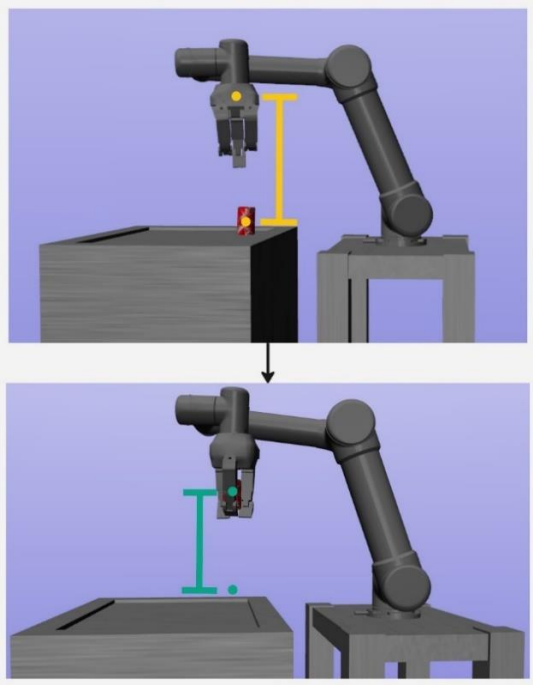
Y bajo las siguientes condiciones al final queda:

- $h_{diffobj} \geq 0.05$: La recompensa 2 se establece en un valor fijo de 500
- $h_{diffobj} < 0.05$: La recompensa 2 se multiplica por 2000

De esta forma, nos aseguramos de que efectivamente el centroide del objeto se ha desplazado más de 5 cm.

Figura 2.34

Función de Recompensa 2



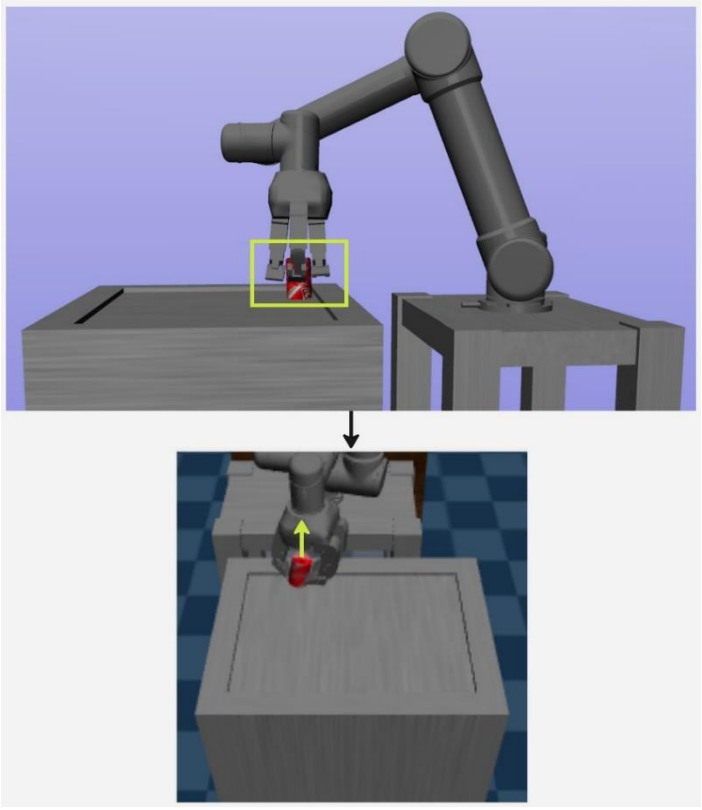
Recompensa 3: Recompensa de Agarre

La finalidad de esta función de recompensa es determinar si se agarró algo o no se agarró, y recompensar los casos de agarre. Si el agarre en la Pinza del Brazo Robótico se establece en máximo, es decir, en el intervalo de 0 a -0.4 para el agarre, adicional a esto, se verifica la condicionalidad de que el objeto posea una altura de centroide mayor a 0.93 que es la altura de centroide estándar respecto a la mesa para la lata en este caso de entrenamiento, por lo cual tal como se define dentro de los parámetros continuos para los límites del entorno, se entrega el valor de recompensa:

- h > 0.93: La recompensa 3 es r_3 = 10
- h >= 0.93: La recompensa 3 es r_3 = 0

Figura 2.35

Función de Recompensa 3



Recompensa 4: Recompensa de Colocación en Casilla (Levantamiento)

Una vez agarrado el objeto, si se coloca en algún estante, es decir, se suelta el objeto en la posición preestablecida de su casilla correspondiente, se entrega la recompensa, lo que se traduce en que el centroide del objeto (en este caso, la lata) [x_{obj}, y_{obj} , z_{obj}], cumpla las siguientes condiciones:

- -0.52 < x_{obj} < 0.32
- y_{obj} > 0
- z_{obj} > 0.756

De ser así, se entrega el valor de recompensa 4, ya que nos estamos asegurando, que el centroide de la lata se encuentra en el cuadrante indicado, así como delimitado por las dimensiones de la estantería:

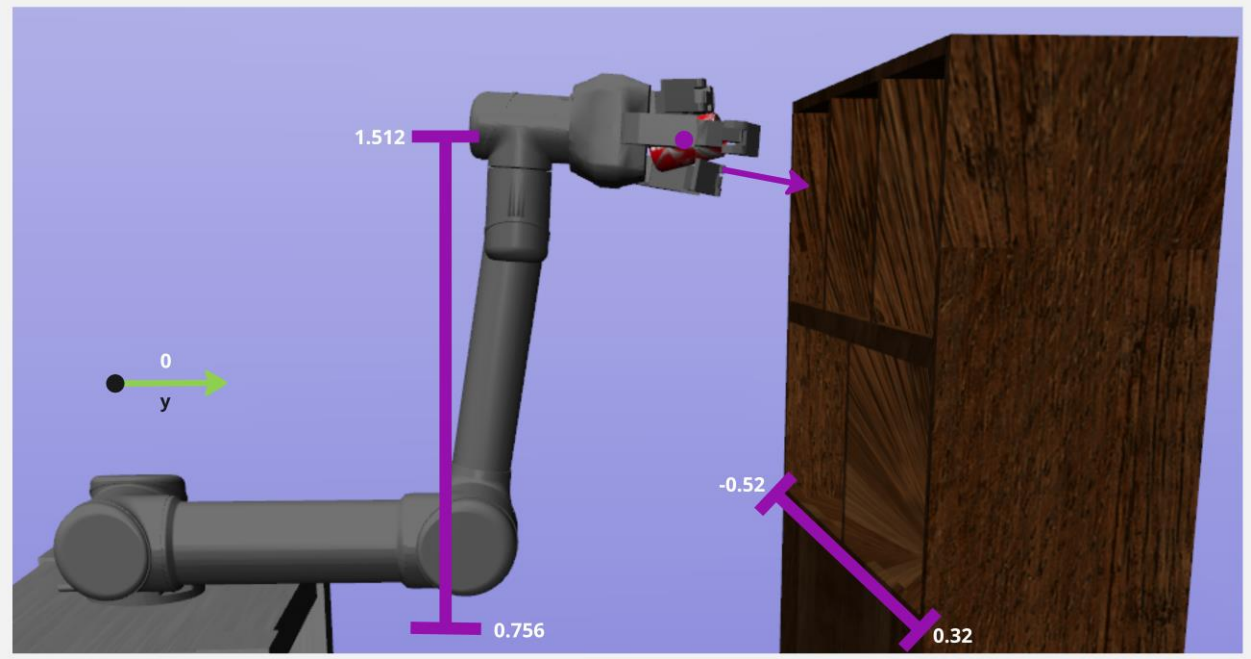
$$r_4 = 105$$

Y se suma al total de objetos agarros dentro de la impresión en consola en tiempo real durante el entrenamiento:

$$Total\ Agarrados \mathrel{+}= 1$$

Figura 2.36

Función de Recompensa 4



2.4.8 Detalles de entrenamiento

2.4.8.1 Discretización de posiciones

Cabe destacar que se establecieron ubicaciones intermedias en las casillas del armario para que sean las posiciones objetivo de reubicación.

Figura 2.37

Distancia en Y de Origen de Entorno a Estante

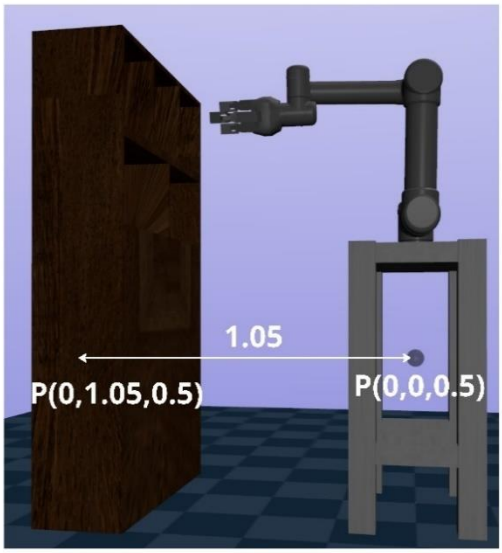
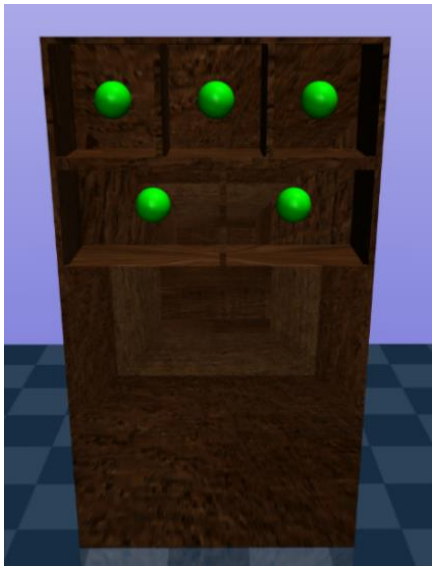


Figura 2.38

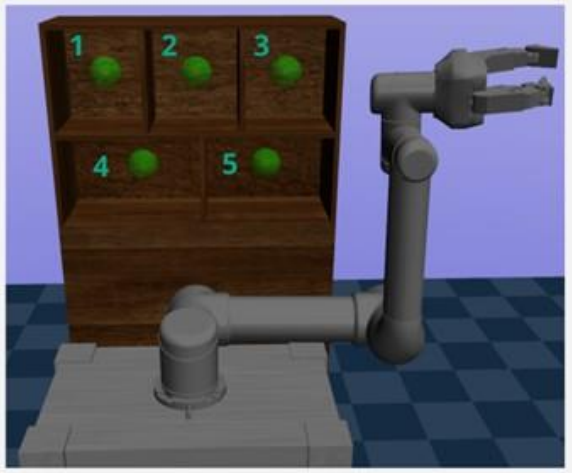
Posiciones Objetivo para Reubicación



Se estableció una etiqueta para cada una de estas casillas de tal forma que tenemos:

Figura 2.39

Etiquetas para Casillas en Armario



Donde las posiciones son:

- Pos1: x(-0.38), y(1.05), z(1.35)
- Pos2: x(-0.10), y(1.05), z(1.35)
- Pos3: x(0.18), y(1.05), z(1.35)
- Pos4: x(0.12), y(1.05), z(1.06)
- Pos5: x(-0.28), y(1.05), z(1.06)

2.4.8.2 Cinemática inversa

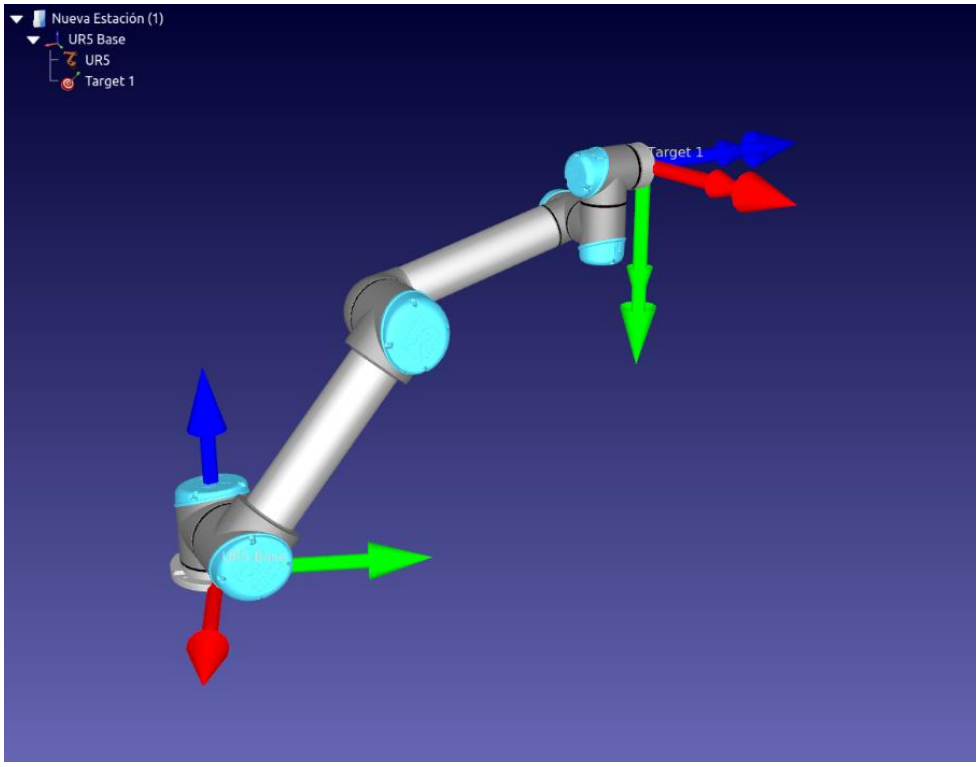
Es importante aclarar que los movimientos del brazo robótico se efectuaron empleando cinemática inversa, entendiendo que por cinemática inversa se hace referencia al proceso de determinar las configuraciones de las articulaciones de un brazo robótico necesarias para alcanzar una posición y orientación específica del extremo del brazo (la "pose objetivo"). Es decir, dado un punto en el espacio tridimensional que se desea alcanzar con el brazo robótico, la cinemática inversa calcula los ángulos o posiciones de las articulaciones que permitirán que el extremo del brazo llegue a esa ubicación.

2.4.8.3 Cinemática directa

Se empleo cinemática directa una vez ya levantado el objeto por cinemática inversa para poder efectuar los movimiento del brazo robótico hacia las posiciones discretizadas del estante debido a que por cinemática inversa presentaba problemas de resolución de la cinemática en algunas ocasiones durante la ejecución del código, es por esto que para calcular los ángulos de las juntas y establecerlas en la lista de entrada a la función de movimiento del robot para la Reubicacion, se empleó un programa de libre acceso llamado RoboDK, en cuya librería de recurso se encuentra el brazo robótico UR5, y simplemente se estableció el punto objetivo dentro del espacio.

Figura 2.40

Interfaz de RoboDK

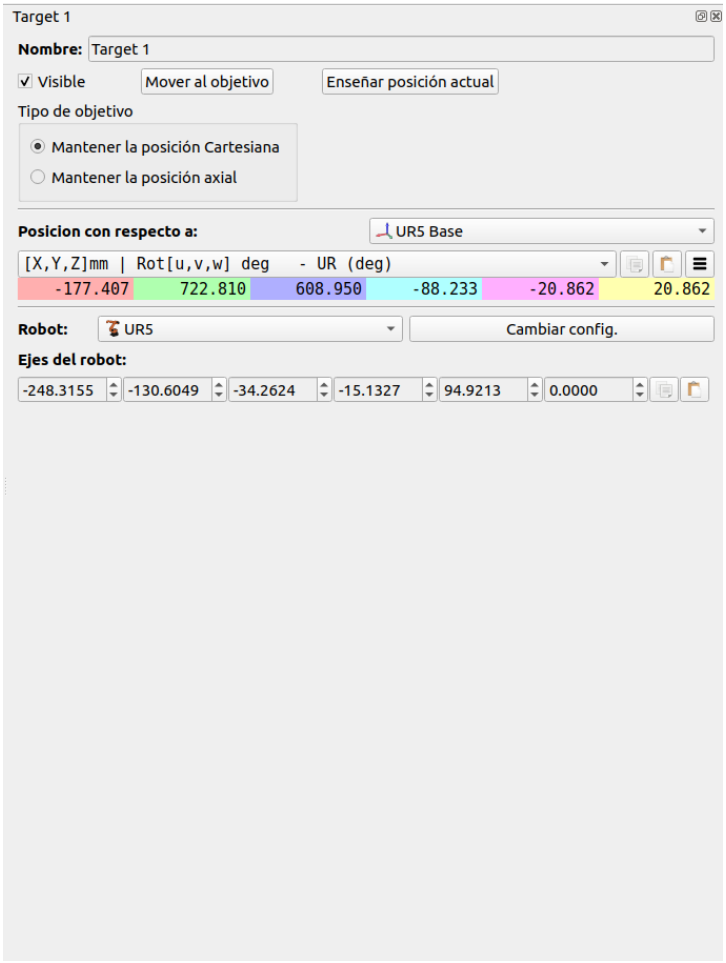


Posteriormente, una vez establecido el punto objetivo, se empleó el resultado que arrojaba la ventana de ejes del robot tal como se aprecia en la figura 2.41, cabe resaltar que estos ángulos

para los ejes se encuentran en grados, por lo cual dentro del código se debieron convertir a radianes para ser ingresados en la función de movimiento del robot y así lograr llevar el objeto a la cabina correspondiente.

Figura 2.41

Ventana de establecimiento de Posición Objetivo y Ángulos en Grados.



2.4.8.4 Parámetros de entrenamiento.

El algoritmo se está entrenando con 10000 episodios, donde cada episodio es de 200 pasos, y para efectuar la cinemática inversa del brazo robótico se estableció un límite de 1000 pasos.

De igual forma, dentro de los hiperparámetros para el algoritmo DDPG, tenemos los siguientes:

Hiperparámetros constantes:

- ENV_ID: "Grasper-v0"
- HEIGHT: 200
- WIDTH: 200
- N_EPISODES: 10000
- STEPS_PER_EPISODE: 200
- LEARNING_RATE: 0.001
- BUFFER_SIZE: 2000
- BATCH_SIZE: 12
- LEARNING_START: 2 * BATCH_SIZE
- GAMMA: 0.99
- TAU: 1×10^{-3}
- DEPTH_ONLY: False
- SEED: 20
- EPS_STEADY: 0.0
- EPS_START: 1.0
- EPS_END: 0.4
- EPS_DECAY: 10000
- SAVINGEP: 250
- WANB_PROJ_NAME: "RL-DDPG"
- WANB_ENTITY: "josuec"
- MODEL: "RL"
- ALGORITHM: "DDPG"

- OPTIMIZER: "ADAM"
- NOISE_CLIP: 0.5
- POLICY_NOISE: 0.2
- POLICY_FREQ: 2

Hiperparámetros en el constructor (__init__):

- height: Valor definido como HEIGHT (200)
- width: Valor definido como WIDTH (200)
- learning_rate: Valor definido como LEARNING_RATE (0.001)
- exploration_noise: Valor fijado a 0.1
- buffer_size: Valor definido como BUFFER_SIZE (2000)
- eps_start: Valor definido como EPS_START (1.0)
- eps_end: Valor definido como EPS_END (0.4)
- eps_decay: Valor definido como EPS_DECAY (10000)
- depth_only: Valor definido como DEPTH_ONLY (False)
- load_path: Valor fijado a None
- seed: Valor definido como SEED (20)
- optimizer: Valor definido como OPTIMIZER ("ADAM")

2.4.8.5 Repositorio de Archivos

En caso de ser deseado corroborar algun código de programación o verificar los archivos, se subió el desarrollo de la tesis a un repositorio de GitHub correspondiente al cliente, es decir, el Laboratorio Bioinformática y Aprendizaje Autónomo, a continuación, se proporcionan los enlaces:

- https://github.com/lba2/UR5-YOLO-DDPG/ (Enlace Privado)
- https://gitfront.io/r/joancaja/mXngGZP9fbd7/UR5-YOLO-DDPG/ (Enlace Publico)

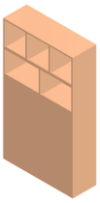
Capítulo 3

3 Resultados

3.1 Análisis de costos

Tablas 3.1

Costo de elementos necesarios

Nombre	Representación gráfica	Costo
Robot UR5		$6,300
Camara RGB		$313
Estante		$130.92
Mesa de trabajo		$87.28
Tripode		$59.95

En la tabla 3.1 podemos ver los elementos que se necesitaría para poder implementar en la vida real este proyecto, sin contar el costo de programación y diseño dado por nosotros. Tomando en cuenta lo dicho el costo de este proyecto estaría empezando desde $6,891.15. Este valor esta

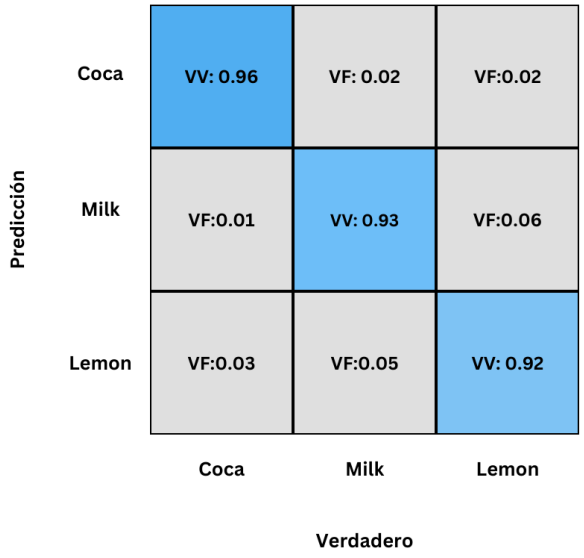
dado para uno de estos sistemas, de emplearse en empresas grandes el valor aumentara dependiendo de la cantidad de sistemas que la empresa requiera.

3.2 Visión robótica

Dentro de la visión Robótica tenemos la matriz de confusión como se ve en la figura 3.2, la cual nos muestra las clases Coca, Milk y Lemon pertenecientes a la lata de coca cola, a el cartón de leche y al limón respectivamente. Esta matriz nos deja analizar cuando tenemos resultados que pueden ser cuestionados, es decir si tenemos un verdadero falso o un falso verdadero, lo que nos indica si el objeto que se detecto era realmente otro. En nuestro caso el porcentaje que tenemos cuando se detectó el objeto correcto es de un 96% para la coca cola, un 93% para la leche y un 92% para el limón.

Figura 3.1

Matriz de confusión



Los valores para poder evaluar de forma correcta este modelo de visión robótica se pueden observar en la tabla 3.2, donde se muestran los valores que se obtuvieron luego del entrenamiento

y la prueba del modelo, cabe recalcar, que estos valores son proporcionados por la página de Roboflow tras el entrenamiento, sin embargo, la lógica detrás de estas métricas es la siguiente:

- Para la exactitud:

$$Acurracy = \frac{True\ Positive\ (TP) + True\ Negative\ (TN)}{True\ Positive\ (TP) + True\ Negative\ (TN) + False\ Positive\ (FP) + False\ Negative\ (FN)}$$

- Para la precisión:

$$Precision = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Positive\ (FP)}$$

- Para la exhaustividad:

$$Recall = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)}$$

- Para el valor-F:

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

De lo cual Roboflow nos entrega la tabla 10:

Tablas 3.2

Resultados de entrenamiento y prueba de modelo visión

Accuracy	96.7%
Precisión	97.5%
Recall	97.1%
F1 Score	97%

3.3 Matriz de transformación

Usamos un objeto en concreto en el cual trabajamos para poder realizar la transformación de la perspectiva y obtuvimos en la localización real era [-0.10000045, -0.44999865, 0.93129644] y la predicción era [-0.05376974559793245, -1.033467879018136, 1.4326527688879391].

Algo que también tuvimos en cuenta fue que el eje de referencia que se mantuvo constante en la misma dirección durante todas las rotaciones es el eje X. es la razón por la que se usaron los valores [1.859790276, 0.43542586, 0.650050187] como factores de proporcionalidad entre el eje x de la cámara y el eje x de del entorno en MuJoCo.

Para poder observar que tan eficiente es la matriz de transformación que se obtuvo se calcula el error entre la predicción y el valor real de las coordenadas de los objetos que aparecen en la mesa de trabajo. Se manejo con 100 iteraciones en donde los objetos reaparecían en lugares diferentes cada vez y se toma los datos para de esa manera obtener un promedio de los errores en cada iteración.

Tablas 3.3

Promedio de errores de la matriz de transformación

		Eje X	Eje Y	Eje Z
Lata de Coca	Error relativo porcentual (%)	5.39468547	9.30203365	6.34263342
	Error absoluto Promedio (m)	0.0093408	0.05875553	0.05906826
Carton de Leche	Error relativo porcentual (%)	26.6371555	33.7826835	19.7406582
	Error absoluto Promedio (m)	0.00499329	0.19050897	0.18793308
Limon	Error relativo porcentual (%)	22.3350209	45.5775616	26.7857371
	Error absoluto Promedio (m)	0.03964569	0.25578238	0.24666808

En esta tabla 3.3 se logra ver como los errores promedios porcentuales y los absolutos, estos últimos se usan para poder tener una ayuda visual del rango en el que se detecta a los objetos.

3.4 DRL

La cantidad de recompensas que obtenga el agente en este caso el robot dependerá de que tan bien realice las acciones que se determinan. En este caso las acciones más esenciales que debe aprender a realizar es acercarse a los objetos lo suficiente para poder sujetarlos, la segunda acción importante es el sujetar dichos objetos, ya que si no logra sujetarlos podría lanzarlos fuera de la mesa de trabajo y no lograría reubicarlos.

Figura 3.2

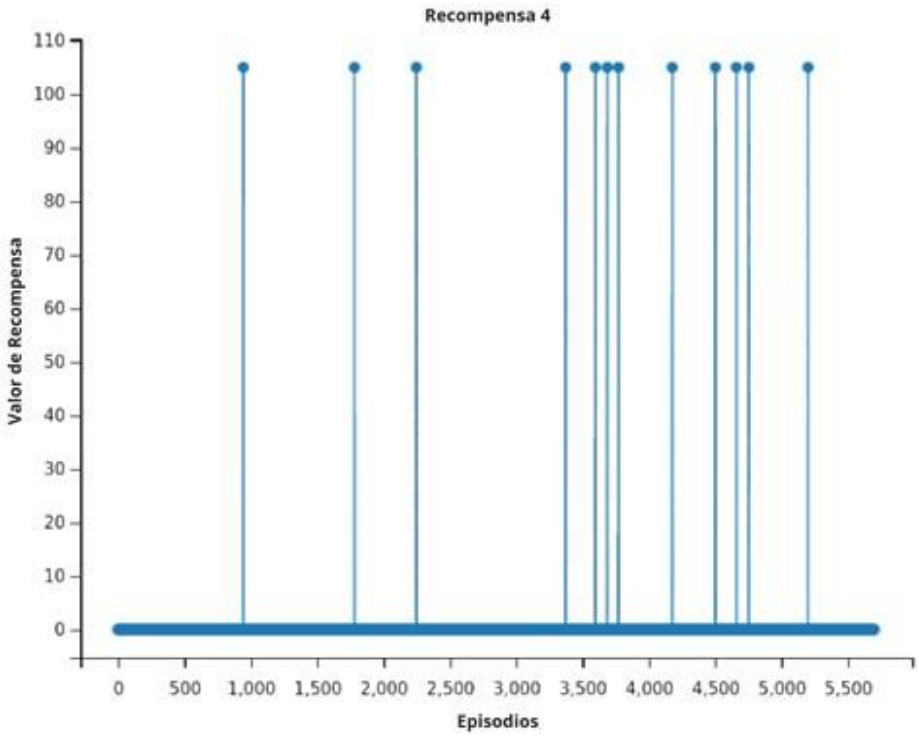
Tendencia del retorno

La figura 3.3. muestra la tendencia del retorno, el cual es la suma de las recompensas por episodio, en este caso se muestra la gráfica por pasos y cabe destacar que se tuvo que aplicar un suavizado de la gráfica mediante un método conocido como promedio móvil exponencial para poder apreciar una tendencia, dado que caso contrario, se observaría mucho ruido entre los pasos. La grafica indica que se están obteniendo mayores recompensas al realizar más episodios de

entrenamiento, con lo cual quiere decir que se está acercando cada vez más al objeto hasta llegar la distancia requerida para que inicie el agarre de los objetos debido a este aumento progresivo, además que los puntos donde existen estos dos picos aproximadamente a los 350k y 380k de pasos, son los dos últimos puntos donde se consiguió la recompensa 4 para los 5500 episodios entrenados.

Figura 3.3

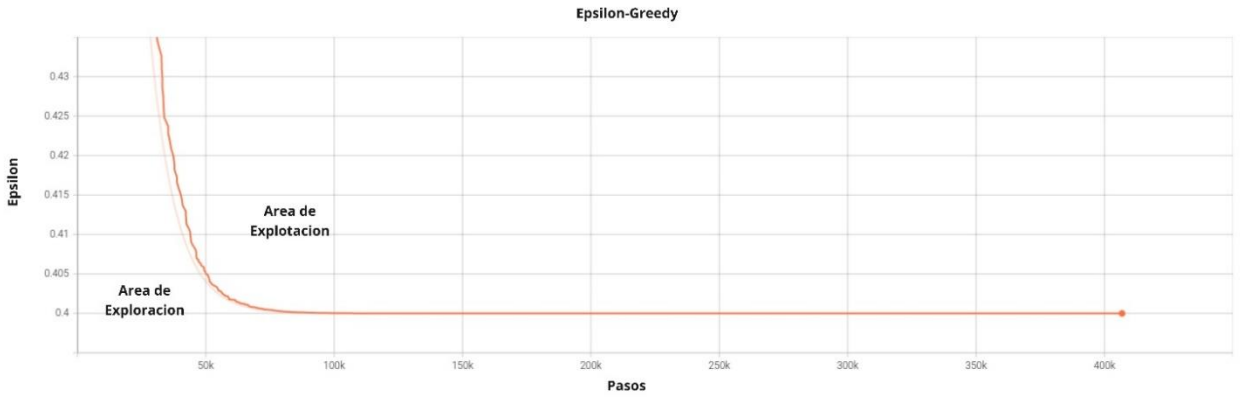
Valores obtenidos de recompensa 3.



En la figura 3.4. Se pude observar cómo se va obteniendo la máxima recompensa 4 cada vez más frecuente lo que indica que el robot está aprendiendo a levantar el objeto con mayor frecuencia. Como podemos ver a partir del episodio 3000, se puede observar cómo disminuyen la cantidad de intentos en los que no se levanta y va aumentando los episodios en los que ha tenido éxito, lo que nos sugiere que nuestro agente robótico se encuentra entrenado y ha aprendido.

Figura 3.4

Grafica de Epsilon-Greedy



En referencia a la Figura 3.5, pudimos ver que el aproximadamente después de los 60k de pasos, pasamos del área de exploración, hacia el área de explotación, lo que nos sugirió que para entonces, el agente ya ha conseguido determinar las acciones optimas y se ha llenado el búfer de experiencias del agente, tal que a medida que pasan más episodios, se mantiene en el mismo valor de épsilon, con lo cual podemos corroborar junto con el grafico anterior de recompensa 4, que el modelo se encuentra satisfactoriamente entrenado y efectúa la tarea de reubicación efectivamente.

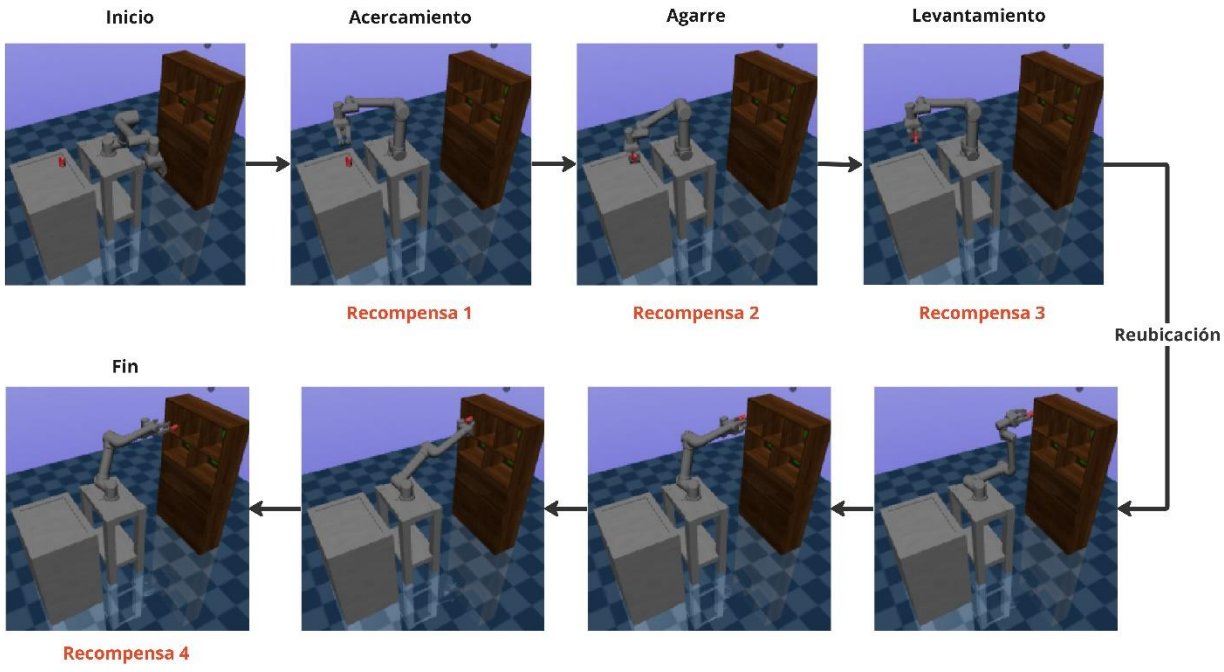
- Visualización de Tarea Efectiva de Reubicación en Espacio Simulado:

Se desarrolló un agente de prueba para corroborar la obtención de las recompensas durante el proceso de Reubicacion, de tal forma que como se muestra en la figura 3.6, para el estado inicial, aun no obtenemos ninguna recompensa máxima, a medida que se fue optimizando la recompensa de acercamiento mediante la distancia euclidiana, llegamos a obtener el valor de 250 en la función de recompensa 1, posteriormente, se entró a la fase de agarre, en donde el agente intentó acercarse en altura al centroide del objeto para proceder a agarrarlo, una vez conseguido, se fue optimizando la función de recompensa 2, hasta entrar a la fase de levantamiento, de tal forma que para esta fase, el centroide del objeto se desplazó más de 5 cm, y por ende, se obtuvo un valor de recompensa 2

de 500, al igual que un valor de recompensa 3 de 10 dado que el objeto se encuentra por encima de su centroide y se encuentra agarrado, por último se ingresó a la fase de reubicación, en donde se ejecutan los movimientos predeterminados establecidos por cinemática directa, y lleva el objeto hacia la cabina correspondiente, donde se lo suelta, y se aleja la mano, dado que el objeto al estar en la cabina cumplió las condiciones de la función de recompensa 4, se obtuvo el valor de 105 en la función de recompensa 4 y se sumó una unidad a la variable de total de objetos agarrados. De esta forma, se corroboró que el código está desempeñando bien su función y que efectivamente, la gráfica de la recompensa 4 obtenida, es un buen indicativo de la reubicación exitosa del objeto.

Figura 3.5

Proceso de Reubicación



Capítulo 4

4 Conclusiones y recomendaciones

4.1 Conclusiones

Dentro de la visión robótica se logra identificar cada objeto con éxito, teniendo una precisión de 97%, al terminar las iteraciones con las imágenes de la base de datos de testeo, y una similitud desde un 90% en cada objeto, al validar los resultados en tiempo real, usando imágenes donde los objetos aparecen en lugares aleatorios. Concluyendo así que si es capaz de identificar los objetos correctamente.

Se desarrolló una matriz de transformación la cual nos permitió cambiar de perspectiva desde el punto de vista del entorno al punto de vista de la cámara dándonos un porcentaje de error o error relativo porcentual menor a 10% para los tres ejes de la lata, objeto con el cual se entrenó el modelo, al cabo de 100 iteraciones, mientras para el cartón de leche fue menor a 34%, y para el limón menor a 46% para los tres ejes, para las mismas 100 iteraciones. Siendo así que se obtuvo una medida del error absoluto del orden de los milímetros para la lata, mientras para el cartón de leche y el limón rondaban por el orden de los centímetros. lo cual llegó a ser satisfactorio para poder tener una precisión al localizar el objeto de entrenamiento, es decir la lata, ya que al ser una medida despreciable no habrá inconveniente para que la pinza sujete el objeto en el punto correcto mediante la predicción del sistema de visión inteligente y la matriz de transformación.

Las pinzas diseñadas lograron adaptarse a las formas geométricas de los objetos que se requieren ubicar a sí mismo según el análisis de tensiones que se realizó, la tensión de von mises es mucho menor al módulo de elasticidad por lo cual podrá levantar sin problemas hasta 5 Kg lo que es el máximo del robot UR5. Claro que este módulo de elasticidad esta dado por el material que normalmente se usa para la elaboración de pinzas el cual es el acero inoxidable, para nuestro proyecto este material estaría sobredimensionado, ya que al no levantar objetos tan pesados se

puede usar un material como el plástico mediante impresiones 3D, y de igual manera podrá levantar los objetos sin problema con un numero de seguridad más adecuado.

Podemos observar una tendencia en las gráficas obtenidas del algoritmo de DDPG, tal que luego de los 60K de pasos se pasa de la etapa de exploración a la etapa de explotación donde el modelo tiene claras las acciones optimas que debe que realizar para obtener la mejor recompensa. Dentro de la siguiente grafica se ve una clara tendencia en las recompensas obtenidas, donde alrededor de los 360K pasos ya se empiezan a observar tendencias claras de un retorno cada vez más alto. Además, en la gráfica de la última recompensa o la recompensa de reubicación se ve como es cada vez más frecuente en la obtención de su valor máximo, lo cual nos indica que se reubico con éxito en varios episodios y fue aumentando la frecuencia de estos éxitos de reubicación a medida que aumentaron estos episodios, con lo cual se puede llegar a concluir que el robot está aprendiendo a recoger y reubicar los objetos en las ubicaciones determinadas.

4.2 Recomendaciones

El modelo actualmente se encuentra entrenado con un objeto únicamente, en este caso, la lata, debido a la de gran potencia computacional requerida durante el entrenamiento, no fue posible implementar el modelo para los demás objetos a pesar de ser reconocibles por el sistema inteligente de visión, por ello, sería recomendable para experimentación futura, entrenar un modelo para cada objeto bajo el mismo algoritmo, de tal forma, que al querer lograr la recolocación de un objeto teniendo más de uno en la mesa de trabajo, únicamente se requiere que se carguen los pesos a la misma base del modelo y se ejecute el modelo según el objeto que se desee por consola, de esta forma, el modelo es extrapolable a cualquier objeto siempre que se pueda reconocer por el sistema de visión y se cuente con los pesos del modelo. Adicionalmente, es recomendable contar con equipos de gran potencia computacional para el entrenamiento debido a que este supone mucha

carga para el procesador y la tarjeta gráfica del computador, además de entrenar en condiciones óptimas para el entrenamiento ya que en la aplicación actual se presentaron problemas de sobrecalentamiento de componentes.

Para el reconocimiento de otros objetos, en caso de querer incluir mayor cantidad de objetos a los actualmente trabajados en el entorno de trabajo, se recomienda primero desarrollar el modelo del objeto en STL, posteriormente cargarlo al entorno actual mediante la redefinición del archivo XML, y posteriormente tomar una serie de capturas del objeto dentro del entorno simulado, de esta forma, se puede agregar una nueva clase de objeto a la base de datos actual del sistema de visión inteligente, reentrenar el modelo de visión, y así lograr que sea reconocible dentro del entorno.

De ser requerida la puesta en marcha del proyecto en un entorno físico, es esencial emplear los parámetros establecidos y elementos específicos como lo es la cámara RGB-D, las posiciones y medidas de los elementos de simulación, dado a que toda la programación y limitaciones se establecieron en base a estos dos elementos y parámetros, en lo que respecta a la pinza, existen varias soluciones similares en el mercado con el único inconveniente de que presentan costes considerablemente superiores, pero que poseen similitudes en la geometría aunque en algunos casos con mayor cantidad de articulaciones, para lo cual se pueden agrupar articulaciones y moverlas en simultaneo, lo cual no supondría muchos inconvenientes en la programación, otro caso puede ser que se usen materiales diferentes, para lo cual sería necesario realizar cambios en las físicas del XML y el URDF del proyecto, estos cambios son pertinentes con la finalidad de mantener la coherencia de los experimentos simulados con una configuración del mundo real. Sin embargo, los algoritmos que se analizan presentan robustes, lo que los hace adaptables a cambios en los modelos dinámicos y geométricos del robot, séanse cambios en las longitudes de los enlaces,

masas e inercias, por lo cual el proyecto es extrapolable a otros robots, inclusive pinzas, pero bajo la limitación de posiciones de los elementos del entorno de simulación trabajado, con esto hacemos referencia, al tamaño de la mesa de trabajo, las cabinas del estante, y las distancias entre elementos.

5 Referencias

[1] AMAZON, “10 years of Amazon robotics: how robots help sort packages, move product, and improve safety”, 2022. [Online]. Disponible en: https://www.aboutamazon.com/news/operations/10-years-of-amazon-robotics-how-robots-help-sort-packages-move-product-and-improve-safety. [Consultado Junio 15, 2023].

[2] S. Lebow, “Robots are taking strides in retail worforces”, 2021. [Online]. Disponible en: https://www.insiderintelligence.com/content/retail-companies-introducing-robots-both-warehouse-in-store-roles. [Consultado junio 12, 2023].

[3] RACTEM, “COSTE DE ALMACENAMIENTO. CONCEPTO Y DEFINICION”, 2022. [Online]. Disponible en: Coste de almacenamiento. ¿Qué es? (ractem.es). [Consultado agosto 22, 2023].

[4] D. Griffus, “How Artificial Intelligence is Changing Warehouse Operations”, 2023. [Online]. Disponible en: https://motion2ai.com/forklift-telematics-tracking-blog/how-artificial-intelligence-is-changing-warehouse-operations. [Consultado junio 12, 2023].

[5] J. Kelleher, “DEEP LEARNING”, Cambridge, MA: MIT Press, 2019.

[6] Ananthakrishnan, A., Kanakiva, V., Ved, D., & Sharma, G. (2018). Automated Gait Generation for Simulated Bodies Using Deep Reinforcement Learning. https://doi.org/10.1109/icicct.2018.8473310

[7] Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning, second edition: An Introduction. MIT Press.

[8] Gudimella, A., Story, R., Shaker, M., Kong, R., Brown, M. A., Shnayder, V., & Campos, M. M. (2017). Deep Reinforcement Learning for Dexterous Manipulation with Concept Networks. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1709.06977

[9] Minttu Alakuijala. Self-taught Robots: Autonomous and Weakly-Supervised Learning for Robotic Manipulation. Computer Science [cs]. ENS Paris - Ecole Normale Supérieure de Paris, 2022. English.

[10] Aiello, "Robotic arm pick-and-place tasks: Implementation and comparison of approaches with and without machine learning (deep reinforcement learning) techniques", Thesis, Politécnico Di Torino, Turin, Italia, 2020.

[11] Franceschetti, A., Tosello, E., Castaman, N., & Ghidoni, S. (2020). Robotic Arm Control and Task Training through Deep Reinforcement Learning. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2005.02632

[12] Mezzina, G., & De Venuto, D. (2021). RGB and 3D-Segmentation Data Combination for the Autonomous Object Manipulation in Personal Care Robotics. https://doi.org/10.1109/dtis53253.2021.9505128

[13] E. Todorov, T. Erez and Y. Tassa, "MuJoCo: A physics engine for model-based control," 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 2012, pp. 5026-5033, doi: 10.1109/IROS.2012.6386109.

[14] P. Daniel., "GitHub - PaulDanielML/MuJoCo_RL_UR5: A MuJoCo/Gym environment for robot control using Reinforcement Learning. The task of agents in this environment is pixel-wise prediction of grasp success chances.," GitHub. https://github.com/PaulDanielML/MuJoCo_RL_UR5, Berlin, 2020.

[15] A. Franceschetti, “Universal Robots UR5 + Robotiq S Model 3 Finger Gripper 1.31.,” Roboti LLC, Mar. 17, 2018. https://roboti.us/forum/index.php?resources/universal-robots-ur5-robotiq-s-model-3-fingergripper.22

[16] T. P. Lillicrap, “Continuous control with deep reinforcement learning,” arXiv.org, Sep. 09, 2015. https://arxiv.org/abs/1509.02971.

[17] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra y M. Ried- miller, "Deterministic Policy Gradient Algorithms", en Procecdings of the 31 " International Conference on Machine Learning, Beijing, China, 2014. En Proceedings of Machine Learning Research, Vol. 32(1), pp. 387-395. Disponible en https://www.davidsilver.uk/wp-content/uploads/2020/03/deterministic-policygradients.pdf

[18] MuJoCo, “Overview - MUJOCO Documentation. (n.d.).”, 2023. [Online]. Disponible en: https://mujoco.readthedocs.io/en/latest/overview.html [Consultado agosto 14, 2023].

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” arXiv (Cornell University), Dec. 2015, doi: 10.48550/arxiv.1512.03385.