**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**Facultad de Ingeniería en Electricidad y Computación[a]**

**Facultad de Ingeniería en Mecánica y Ciencias de la Producción[b]**

# Planificación del movimiento de un brazo robótico mediante técnicas de inteligencia artificial para mejorar la eficiencia del agarre.

**PROYECTO INTEGRADOR**

Previo la obtención del Título de:

**Ingeniero en Ciencias de la Computación[a]**
**Ingeniero en Mecatrónica[b]**

Presentado por:

**Camilo Andrés Gutiérrez Plaza[a]**
**José Johil Patino Miñán[b]**

GUAYAQUIL - ECUADOR

Año: 2022 - 2023

# DEDICATORIA

Por haberme iluminado durante todo el proceso de realización de mi proyecto integrador, dedico el mismo a Dios.

A mis tres madres, Hilda Miñán, Delia Miñán y Rosa Landívar, quienes me han guiado desde mi nacimiento y han estado presentes apoyándome en todos los momentos buenos y malos de mi vida. A mi padre, José Stalyn Patiño, quien ha forjado carácter y personalidad para afrontar las decisiones diarias y sus consecuencias. A mi hermano menor, José Andrés Patiño, un compañero cercano del cual aprendo lecciones de vida. A mis primos, Luis y Sahid Mayorga Miñán, la inspiración para ser un mejor ejemplo a seguir.

A Carolina Ormaza, quien me ha acompañado desde cuarto semestre, haciendo más amena mi vida universitaria.

**José Johil Patiño Miñán**

# DEDICATORIA

El presente proyecto se lo dedico a mi mamá Joamelly Plaza Andrade quien ha velado por mi durante toda mi trayectoria estudiantil y ha sido más que una madre una amiga en quien confiar.

A mi papá Arquelino Gutiérrez Prado quien me apoyó en mi decisión de seguir mi carrera.

A mis maestros PhD. Carmen Vaca, MsC. Eduardo Cruz, y PhD Carlos Mera quienes han creído en mí y han sido guías de mi desarrollo profesional.

A mi grupo de amigos del club TAWS que han sido un soporte en mi vida académica y personal.

**Camilo Gutiérrez Plaza**

# AGRADECIMIENTOS

Agradezco a Dios por permitirme cursar mis estudios universitarios en condiciones estables.

A mi familia por confiar en mi potencial e invertir su tiempo apoyándome incondicionalmente.

Al PhD. Carlos Saldarriaga, de quien he adquirido la mayoría de los conocimientos para realizar este proyecto.

Al PhD. Juan Hernández Vega y al PhD. Dennys Paillacho, quienes han guiado significantemente este proyecto.

**José Johil Patiño Miñán**

# AGRADECIMIENTOS

Agradezco a mi familia que invirtió tiempo y esfuerzo en mi educación universitaria.

Al PhD. Miguel Realpe y al PhD. Dennys Paillacho quienes han sido una guía en este proyecto con sus conocimientos y retroalimentación.

Al PhD. Juan Hernández Vega por su mentoría y la oportunidad de trabajar en este proyecto.

**Camilo Gutiérrez Plaza**

# DECLARACIÓN EXPRESA

"Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; *Camilo Andrés Gutiérrez Plaza* y *José Johil Patiño Miñán* 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"

Camilo Gutiérrez Plaza

José Patiño Miñán

# EVALUADORES INGENIERIA EN COMPUTACION

MIGUEL ANDRES REALPE ROBALINO

Digitally signed by MIGUEL ANDRES REALPE ROBALINO
DN: c=EC, o=BANCO CENTRAL DEL ECUADOR, ou=ENTIDAD DE CERTIFICACION DE INFORMACION-ECIBCE, l=QUITO, serialNumber=0000793814, cn=MIGUEL ANDRES REALPE ROBALINO
Date: 2023.02.13 21:11:00 -05'00'

**Miguel Realpe Ph.D.**

PROFESOR DE LA MATERIA

Firmado electrónicamente por:
DENNYS FABIAN PAILLACHO CHILUIZA

**Dennys Paillacho Ph.D.**

PROFESOR TUTOR

# EVALUADORES MECATRONICA

Firmado electrónicamente por:
CARLOS XAVIER
SALDARRIAGA MERO

**Carlos Saldarriaga Ph.D.**

PROFESOR DE LA MATERIA

Firmado electrónicamente por:
DENNYS FABIAN
PAILLACHO CHILUIZA

**Dennys Paillacho Ph.D.**

PROFESOR TUTOR

# RESUMEN

El presente proyecto tiene como objetivo mejorar la eficiencia de los sistemas de agarre robótico. Estos sistemas suelen dividirse en dos etapas: percepción del ambiente y planificación de movimiento. La percepción consiste en identificar objetos agarrables y generar posibles poses de agarre que, son evaluadas hasta encontrar una válida la cual es usada como objetivo del planificador de movimiento. Sin embargo, estas poses no consideran datos de eficiencia del planificador, como la longitud o el tiempo de procesamiento de la ruta. Por lo tanto, el proyecto propone incluir datos de planificación en el entrenamiento de una red neuronal para obtener poses más eficientes.

Para ello, se ha utilizado el algoritmo RRTConnect para la planificación y el enfoque de control por cómputo de torque para el control para minimizar errores en el seguimiento de la trayectoria. La red neuronal utiliza PointNet++ para segmentar nubes de puntos, junto a otras capas, para generar poses de agarre considerando los datos de planificación.

La simulación del sistema en CoppeliaSim ha dado resultados prometedores, con un éxito de agarre del 96.03% en escenas ordenadas y del 91% en escenas desordenadas. Además, el planificador de movimiento proyecta rutas eficientes hacia los objetivos, y el módulo de control permite al manipulador seguir trayectoria con errores menores a 1°.

**Palabras Clave:** Agarre robótico, inteligencia artificial, sistema de control, planificación de movimiento.

# *ABSTRACT*

*This project aims to improve the efficiency of robotic grasping systems. These systems are usually divided into two stages: perception of the environment and motion planning. Perception consists of identifying graspable objects and generating possible grasping poses, which are evaluated until a valid one is found and used as a target for the motion planner. However, these poses do not consider planner efficiency data, such as path length or processing time. Therefore, the project proposes to include planning data in the training of a neural network to obtain more efficient poses.*

*For this purpose, the RRTConnect algorithm has been used for planning and the torque computation control approach for control to minimize errors in trajectory tracking. The neural network uses PointNet++ to segment point clouds, along with other layers, to generate grasping poses considering the planning data.*

*Simulation of the system in CoppeliaSim has yielded promising results, with 96.03% grasping success in ordered scenes and 91% in disordered scenes. In addition, the motion planner projects efficient routes to the targets, and the control module allows the manipulator to follow trajectory with errors of less than 1°.*



***Keywords:*** *Robotic grasping, artificial intelligence, control system, motion planning.*

# ÍNDICE GENERAL

# ABREVIATURAS

ESPOL Escuela Superior Politécnica del Litoral

RG Robotic Grasping

MBA Model-Based Algorithm

MFA Model-Free Algorithm

MTH Matriz de Transformación Homogénea

# SIMBOLOGÍA

| | |
|---|---|
| m | Metro |
| deg | Grado |
| rad | Radian |
| s | Segundo |

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# CAPÍTULO 1

## 1. INTRODUCCIÓN

En la actualidad las soluciones para los problemas de robótica, como el agarre de objetos, se realizan procedimientos que consumen bastante tiempo. El uso de técnicas basadas en aprendizaje profundo ha llevado el problema a innovar en métodos de detección que tratan de predecir las poses de agarre y su calidad dada una escena.

En el proyecto a continuación se describe la problemática relacionada al agarre de objetos, enfatizando la importancia de su resolución y los objetivos definidos para lograr resolver el problema.

### 1.1 Descripción del problema

Una de las tareas más comunes de los robots manipuladores es la de agarrar objetos; ya sea para ordenar ambientes, clasificar productos o manipular herramientas. En contraste a la facilidad que tenemos los humanos para realizar tareas como el agarre de objetos, para los robots puede llegar a ser una tarea desafiante. Para un robot el agarre de un objeto no es trivial y comúnmente implica una serie de tareas, tales como: percibir el objeto de manera que pueda detectar su posición, determinar las posibles formas de agarre, planificar el movimiento del robot para determinar la trayectoria del movimiento deseado para llevarlo hasta el objetivo, y finalmente el control para conducir el movimiento del robot.

Actualmente, el estado del arte toma por separado la identificación de las poses de agarre y la planificación del movimiento-cinemática inversa. De tal manera que, si un candidato de pose de agarre no es alcanzable por el robot, el programa de control debe iterar con otros candidatos de pose de agarre, resultando en que el problema se vuelva costoso en términos de tiempo y de recursos computacionales. Para resolver este problema, actualmente se han implementado modelos de aprendizaje profundo, los cuales generalmente proveen un tiempo de ejecución

dentro de las ordenes de segundos e incluso milisegundos, otorgando una mayor factibilidad en su uso. Sin embargo, estos modelos sólo mejoran los tiempos resultantes de la detección de poses, omitiendo el tiempo que toma la planificación del movimiento.

El Centro de Inteligencia Artificial, Robótica y Sistemas Hombre-Máquina - IROHMS (por sus siglas en inglés) de la universidad de Cardiff, dentro de sus líneas de investigación, ha propuesto unificar estas dos etapas (la identificación de las poses de agarre y la planificación del movimiento-cinemática inversa), de manera que el robot aprenda qué candidatos de pose de agarre son viables utilizando no sólo la información de la percepción, sino también la de planificación del movimiento, de esta manera se tendrían como opciones solo las cinemáticamente factibles para ser llevadas a la etapa de control.

El desarrollo de la solución tiene como finalidad su implementación en un sistema manipulador redundante de 7 grados de libertad (GdL), Franka Emika Panda; con sus respectivas pinzas robóticas paralelas, Franka Emika Hand. Aprovechando el avance y aplicaciones de técnicas de aprendizaje de máquina se propone el desarrollo de un algoritmo capaz de determinar las posibles formas de agarrar un objeto con énfasis en la viabilidad cinemática de estas. Además, dichas poses presentadas como alternativas serán probadas en un planificador de movimiento y un controlador, que informe la viabilidad del agarre, es decir si las regiones o poses de agarre expuestas son factibles para el robot.

## 1.2 Justificación del problema

En robótica uno de los retos más grandes es la ejecución de tareas sofisticadas a través de robots, tales como los quehaceres domésticos. Sin embargo, esto solo puede ser posible si se logra que los robots sean experimentados en realizar tareas básicas, tal y como lo es el agarre de objetos, convirtiéndose en un área fundamental de investigación.

Hasta la fecha no se han implementado modelos de aprendizaje de máquina que incluyan en su entrenamiento datos sobre la planificación de movimiento que permitan discriminar candidatos de agarre viables para controlar el movimiento del manipulador al objetivo y su correcto agarre. De ser así, implicaría una mejora en la elección de las poses candidatas y resultando en tiempos de ejecución de la tarea de agarre más eficientes.

Por otra parte, cabe recalcar que la solución es específica para el brazo robótico Franka Emika Panda. Sin embargo, las bases y la metodología propuestas podrían ajustarse a cualquier robot.

A largo plazo, se espera integrar el proyecto a unidades quirúrgicas que manipulen utensilios para operaciones, como ayudante de laboratorio para realizar procesos repetitivos como manipulación de probetas, y en el área de la manufactura. En el caso de las cirugías con intervención de manipuladores en [1], se detalla que el 10.4% de eventos requirió reiniciar el sistema, reprogramar o cambiar la operación a técnicas no robóticas debido a una indebida planificación, fallas en el sistema de control y de percepción.

## 1.3 Objetivos

### 1.3.1 Objetivo General

Desarrollar un algoritmo de agarre de un brazo robótico mediante técnicas de aprendizaje profundo y control para mejorar su eficiencia.

### 1.3.2 Objetivos Específicos

1. Desarrollar un modelo de aprendizaje de máquina para predecir las posibles poses de agarre más eficientes utilizando los datos de planificación de movimiento del robot y percepción. (Ingeniería en Computación, Ingeniería en Mecatrónica)
2. Implementar un planificador para realizar el movimiento de agarre del robot. (Ingeniería en Mecatrónica)

3. Desarrollar el control del robot efectuando el movimiento planificado para el agarre de objetos. (Ingeniería en Mecatrónica)

## 1.4 Marco teórico

### 1.4.1 Agarre Robótico

El campo de la robótica que se encarga de implementar el agarre y la manipulación de objetos se conoce como *robotic grasping* (RG) [2].
Las etapas del RG se pueden agrupar en 4 o menos dependiendo del conocimiento del robot y de su entorno de operación. Estas 4 etapas son: localización del objeto, estimación de la pose del objeto, estimación de agarre y planificación de movimiento.

### 1.4.2 Modelos de Detección de Poses de Agarre

La identificación del objetivo de agarre puede clasificarse en dos grupos de acuerdo con la existencia de un modelo comparable: mediante algoritmos basados en modelos y mediante algoritmos libre de modelos. Si se incorpora al sistema un modelo 3D (escaneado o modelado previamente) para ser comparado con el objeto el cual se desea agarrar, se trata de un algoritmo basado en modelos (MBA por sus siglas en inglés). Por otro lado, si se obtienen directamente candidatos de poses de agarre gracias a superficies geométricas procesadas *online*, obtenidas por un sistema de percepción y es posible generalizar a la manipulación con objetos desconocidos, el sistema se conoce como algoritmo libre de modelos (MFA, por sus siglas en inglés) [3] [4] [5]. RG por medio de MBA utiliza todas las etapas; mientras que MFA solo utiliza las dos últimas: estimación de agarre y planificación de movimiento. Para la generalización del problema, y evitar el trabajo de la creación de modelos 3D, es más factible basarse en el modelo MFA [6].

Uno de los formatos de representación de objetos en el espacio más utilizados en visión por computadora son los *point clouds* o nubes de puntos. Las nubes de puntos son un conjunto de coordenadas de 3 dimensiones (X, Y, Z) que contiene información espacial del mundo medidas respecto a un sensor, estos datos

pueden ser procesados a partir de imágenes o modelos 3D obtenidos mediante escáner láser 3D (LIDAR), sensores de profundidad o cámaras RGB-D, entre otros [7].
Las nubes de puntos son de vital importancia para algunas redes neuronales que tienen estos datos como entrada, para luego ser procesados y entregar poses de agarre.

Existen diversas representaciones espaciales para una pose de agarre. Para pinzas robóticas paralelas, una de las formas más convenientes de establecer esta representación es el Espacio Euclidiano especial (SE) (3) (Espacio Euclidiano especial) o 6D [8]. Debido a la simpleza de estas pinzas robóticas, 6 variables determinan completamente su ubicación espacial. Estas 6 dimensiones se resumen en: posición 3D (x, y, z) y orientación 3D ($r_x$,$r_y$,$r_z$).

### 1.4.3 Planificación del Movimiento Robótico

Una vez que cierto candidato de agarre ha sido validado, es necesario utilizar un planificador de movimiento que permita llevar el efector final del robot hacia al objetivo. Además, este camino tiene diversas configuraciones articulares para ser seguidas por el robot. Se necesita computar torques para que el robot siga la trayectoria con el mínimo error posible, para lo cual se utilizan técnicas de control. Para robots manipuladores, es importante definir el espacio de estado, el cual consiste en cada una de las variables articulares pertenecientes al espacio de configuraciones sobre la cual se va a ejecutar la planificación. Por lo tanto, para un robot con n GdL, se tendrá un vector de estado de dimensión nx1. Los algoritmos de planificación operan interpolando entre un estado inicial y uno final, lo que los diferencia es el tipo de interpolación que se aplica a las variables de estado.

Para la etapa de planificación de movimiento, tanto para robot móviles como para articulados, una de las librerías más usadas es OMPL (Open Motion Planning Library) [9]. Esta librería contiene implementaciones de los siguientes algoritmos: PRM (*probabilistic roadmap*), RRT *(rapidly exploring random tree*), EST

(expansive spaces trees), SBL (*Single-query, Bi-directional, Lazy roadmap*) y algunas de sus variantes.

### 1.4.4 Métodos de control de movimiento para robots manipuladores

Para la parte de control, se han desarrollado algunas estrategias para obtener torques que permitan conducir a los manipuladores por una trayectoria deseada con mínimos errores [10].

El control por cómputo de torque (CCT) es una técnica de control de movimiento efectiva, que permite desacoplar y linealizar la dinámica del robot (la cual es altamente no lineal), introduciendo constantes de rapidez de estabilización y para la razón de amortiguamiento al sistema. Sin embargo, este enfoque tiene la desventaja de ser significativamente inconsistente con pequeños errores en el modelado dinámico del robot, se debe tener un modelo perfecto para utilizar esta técnica [11] [12] [13].

Por otra parte, [14] y [15] explican tanto teoría y experimentación del control proporcional-derivativo (PD) con compensación gravitatoria (PDCG), el cual es una estrategia para el control de posición de manipuladores que introducen constantes proporcionales y derivativas ($K_p$ y $K_d$) para cada una de las variables articulares. Para cualquier valor de las matrices PD se garantiza la estabilización del sistema, siempre y cuando $K_p$ y $K_d$ sean matrices positivas definidas. Esta técnica utiliza menos dinámica que el CCT, puesto a que solo compensa los términos de torques gravitacionales. Se necesita un buen modelo del vector de torques gravitacionales, además este modelo no es consistente si hay presencia de ruidos considerables o de fuerzas externas.

Similarmente, el control por impedancia es usado para este tipo de situaciones, para evitar fuerzas de impacto grandes con el ambiente [16]. En este enfoque se busca diseñar matrices de rigidez y de amortiguamiento para buscar una respuesta dinámica deseada, usando la analogía de un sistema multivariable masa-amortiguador-resorte. La matriz de rigidez proporciona las características

de fuerza-posición, mientras que la matriz de amortiguamiento se utiliza para moldear el comportamiento de la respuesta transiente.

### 1.4.5 Aprendizaje profundo en el agarre robótico

En [17] se crea una una red neuronal (DJ-GOMP) que optimiza el tiempo de ejecución de caminos factibles para realizar el agarre, incorporando candidatos de poses de agarre, datos cinemáticos y dinámicos; se registra una reducción de tiempo de 29s a 80 ms.

En [18] se realiza un sistema completo de agarre, que va desde el entrenamiento de una red neuronal convolucional (CNN), utilizando Pytorch y OpenCV, hasta realizar la planificación del movimiento, la cinemática inversa y el control por compensación de gravedad. Se registran tazas de éxito de agarre del 90% para cajas, 92% para cintas, 80% para tubos y 95% para borradores con un manipulador Barrett WAM.

Song et al. en [19] entrenan una función Q, utilizando Q-learning basado en el ejemplo humano grabado en video para discretizar la planificación de movimiento compensada por control proporcional-derivativo (PD), logrando obtener transformaciones homogéneas de 6 GdL con un 92% de éxito en el agarre con un manipulador UR5.

Respecto a los detectores de poses de agarre, Fang et al. desarrollaron Graspnet-1Billion [20], una red neuronal entrenada con 97.280 imágenes RGB-D, que contiene alrededor de 1 billón de poses generadas. Esta red toma como entrada nubes de puntos y devuelve posibles poses con su representación 6D. Esta red evalúa el éxito de cierta pose usando el concepto de cierre por fuerza.

Wang et al. en [21] crean una red neuronal profunda para RG, con una taza de identificación de poses del 98%. Para crear esta red se empleó una cámara a color y se utilizó SSD (Single Shot Multibox Detector) como modelo de identificación de

objeto. Los datos de salida también se encuentran en formato 6D, para pinzas roboticas paralelas.

Dexnet 4.0 [22] es una red basada en GQ-CNN para robots ambidiestros que devuelve una representación de poses de 6 dimensiones del objeto deseado, en este caso fue entrenada para un robot colaborativo industrial bimanual ABB YuMi. Esta red alcanza una precisión del 96% para ventosas y 98% para pinzas roboticas paralelas.
Se puede observar que todos los estudios presentes incluyen datos sobre la detección y predicción de pose de los objetivos; sin embargo, no incluyen datos sobre la planificación del movimiento en la red neuronal. Estas etapas de planificación y percepción son tomadas totalmente como separadas en la actualidad.

En el presente estudio la planificación del movimiento se considera de alta importancia, debido a que un objeto en calidad de “agarrable”, debe también debe ser alcanzable dentro de los candidatos de agarre en el espacio de trabajo del robot.

# CAPÍTULO 2

## 2. METODOLOGÍA

En este capítulo se describe el análisis del problema y la solución propuesta, la cual incluye el proceso de recolección de datos de prueba y el diseño de la arquitectura para el sistema del agarre del brazo robótico Franka Panda.

### 2.1 Análisis

Para el análisis se especificaron reuniones con el cliente para definir el alcance del proyecto, junto con el producto final, las necesidades y limitaciones del mismo. Durante las reuniones se destacaron soluciones previas de la problemática y los resultados que se habían obtenido. Proyectos previos se basan en la creación de modelos de aprendizaje por refuerzo para la generación de poses de agarre, por lo que el proyecto plantea ampliar el desarrollo de una arquitectura que permita el agarre de objetos agregando la información de la planificación de movimiento para su posterior implementación en un robot Franka Panda. Sin embargo, el cliente especificó que el alcance del presente proyecto se debe limitar a simulación.

#### 2.1.1 Requerimientos Funcionales

##### *2.1.1.1 Módulo de Percepción*

**Tabla 2.1 Requerimiento Funcional #1 – Detectar la localización del objeto**

| RF-01 | **Detección de la localización del Objeto** |
|---|---|
| Versión | 1.0 |
| Dependencias | |
| Descripción | El sistema debe detectar la localización de un objeto estático y caracterizarlo a través de una nube de puntos. |
| Prioridad | Alta |

**Tabla 2.2 Requerimiento Funcional #2 – Detectar la posición del objeto**

| RF-02 | **Detección de la posición del Objeto** |
|---|---|
| Versión | 1.0 |
| Dependencias | RF-01 |
| Descripción | El sistema debe clasificar y segmentar la información de la nube de puntos para determinar en qué posición se encuentra el objeto. |
| Prioridad | Alta |

### *2.1.1.2 Módulo de Recolección de Datos*

**Tabla 2.3 Requerimiento Funcional #3 – Generación de la escena**

| RF-03 | **Generación aleatoria de la escena** |
|---|---|
| Versión | 1.0 |
| Dependencias | |
| Descripción | El sistema debe generar de manera aleatoria una escena con objetos estáticos con formas simples tridimensionales (Cubos, Cilindros) de diferentes alturas. |
| Prioridad | Alta |

**Tabla 2.4 Requerimiento Funcional #4 – Recolección de datos**

| RF-04 | **Recolección de datos** |
|---|---|
| Versión | 1.0 |
| Dependencias | RF-01, RF02, RF-03, RF-04, RF-05, RF-06, RF-07 |
| Descripción | El sistema debe ser capaz de guardar los datos de la percepción y la planificación de movimiento del robot. |
| Prioridad | Alta |

### *2.1.1.3 Módulo de Generación de Poses*

**Tabla 2.5 Requerimiento Funcional #5 – Detectar las poses de agarre del objeto**

| RF-05 | **Detección de las poses de agarre** |
|---|---|
| Versión | 1.0 |
| Dependencias | RF-01 RF-02 |
| Descripción | El sistema debe generar candidatos de poses de agarre. |
| Prioridad | Alta |

### *2.1.1.4 Módulo de Planificación*

**Tabla 2.6 Requerimiento Funcional #6 – Solucionador de cinemática inversa**

| RF-06 | **Solución de cinemática inversa** |
|---|---|
| Versión | 1.0 |
| Dependencias | |
| Descripción | El sistema debe generar las posiciones angulares en el espacio articular para una pose del efector final dada. |
| Prioridad | Alta |

**Tabla 2.7 Requerimiento Funcional #7 – Planificador de movimiento**

| RF-07 | **Planificación de movimiento** |
|---|---|
| Versión | 1.0 |
| Dependencias | RF-06 RF-05 |
| Descripción | Dadas las posiciones articulares iniciales y finales, el planificador deberá trazar una ruta y ejecutarla en el espacio articular. |
| Prioridad | Alta |

#### *2.1.1.5 Módulo de Control*

**Tabla 2.8 Requerimiento Funcional #7 – Controlador de movimiento**

| RF-08 | **Controlador de movimiento** |
|---|---|
| Versión | 1.0 |
| Dependencias | RF-07 |
| Descripción | Dada una trayectoria angular, se deben calcular y ejercer los torques necesarios para llevar al robot por la ruta especificada. |
| Prioridad | Alta |

### 2.1.2 Requerimientos no funcionales

**Tabla 2.9 Requerimiento no Funcional #1 – Ambiente de ejecución**

| RNF-03 | **Ambiente de ejecución** |
|---|---|
| Versión | 1.0 |
| Descripción | El sistema deberá ser ejecutado en un ambiente simulado a través del simulador CoppeliaSim. |
| Prioridad | Alta |
| Criterio de validación | Se evidencia que el sistema es capaz de realizar la tarea de agarre en el ambiente simulado. |
| Clasificación | De ambiente |

### 2.1.3 Alcance y limitaciones de la solución

A partir del análisis de requerimientos se contempló que la arquitectura diseñada debe ser capaz de permitir al robot Franka Panda percibir objetos simples como cubos o cilindros y detectar su posición de manera que sea posible determinar

candidatos de poses de agarre, además debe controlar el movimiento articular del robot y mover su efector final hacia la pose de agarre seleccionada.

Entre las limitaciones se encuentra que los algoritmos actuales no cubren densamente el objeto, generando que la información de agarre no sea completa. Además, las soluciones actuales basadas en redes neuronales optan por un enfoque convolucional en el que hacen uso de millones de unidades de activación lo cual es una limitante al momento de realizar su entrenamiento, pues requiere de un alto poder computacional que permita realizar esta tarea de manera eficiente.

### 2.1.4 Riesgos y beneficios de la solución

Los beneficios de la implementación del proyecto comprenden un avance tecnológico en el área de robótica para el mejoramiento de las tareas básicas como lo es el agarre de objetos, suponiendo un inicio para llevar a cabo tareas más complejas utilizando robots.

Además, el mejorar las tareas de agarre incrementa la eficiencia de las operaciones, como lo pueden ser logísticas o de servicio, en ambos casos, el agarre de objetos disminuye los tiempos de acción del robot, mejorando su uso y la calidad del trabajo que se le delegan al robot.

Así mismo se analizaron los riesgos que suponen el desarrollo del proyecto del cual notablemente se destacan el uso indebido de la tecnología, el mal funcionamiento de la arquitectura implementada en otros robots diferentes al Franka Panda, el costo del mantenimiento del robot puesto que suponen un desgaste en el tiempo, y la falta de apertura de integración con otros sistemas.

## 2.2 Proceso de diseño

Se tuvieron que completar varios pasos para llegar al diseño final, y cada área problemática se abordó gradualmente mientras se lograba satisfacer las expectativas del cliente. Mediante el uso de un diagrama de flujo, la Figura 2.1 demuestra el enfoque adoptado a lo largo del proceso de diseño. Esta figura

muestra el orden en el que se ha realizado el proyecto y sus dependencias. Es observable que luego de la implementación del algoritmo de movimiento, es posible realizar el módulo de control independiente del módulo de programación.

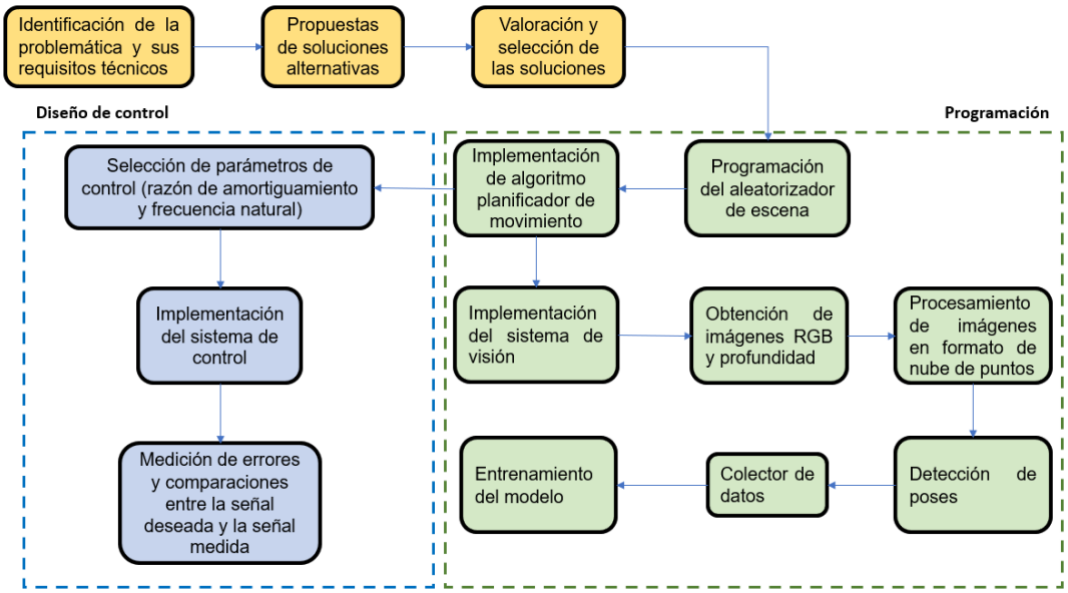


**Figura 2.1 Proceso de diseño del Sistema de Agarre del Robot Franka Panda**

## 2.3 Sistema de agarre del Robot Franka Panda

Para la realización del proyecto se optó por la construcción de 4 módulos principales. El módulo de percepción encargado de la obtención y procesamiento de los datos de entrada. El módulo de generación de poses que comprende la predicción de las mejores poses de agarre según los objetos en escena. El módulo de planificación de movimientos que recibe las mejores poses de agarre y adecua la planificación a la mejor pose seleccionada. Finalmente, el módulo de control del movimiento, que efectúa el movimiento del robot, es decir el movimiento del manipulador y de sus pinzas.

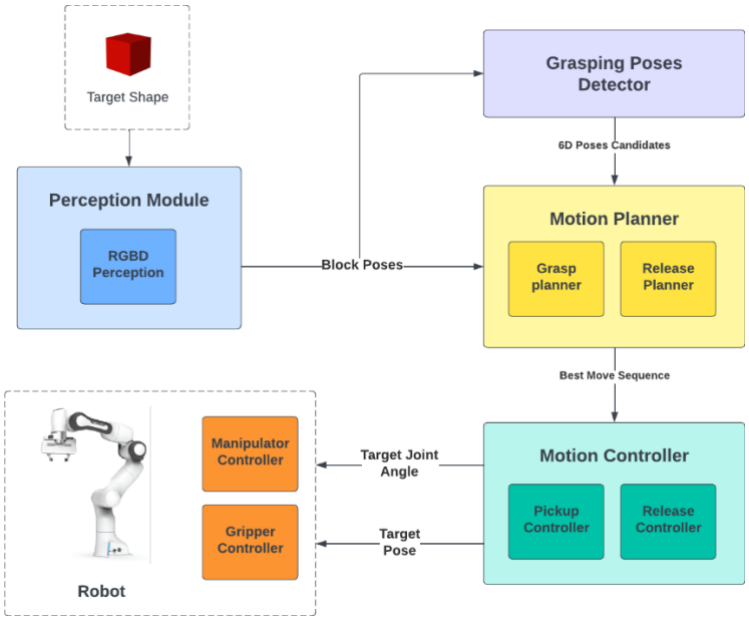


**Figura 2.2 Arquitectura del Sistema de Agarre del Robot Franka Panda**

## 2.4 Módulo de percepción

El proceso de determinar los candidatos de poses de agarre requiere la previa localización del objeto, para lograr esto se implementó un sistema de percepción que detecta y localiza objetos en una escena. Como base de la arquitectura del sistema se optó por utilizar la red PointNet++ que provee una arquitectura para la clasificación y segmentación de nube de puntos [23]. La red toma como entrada una nube de puntos de tamaño N x 3 donde N es el total de puntos en la nube, y retorna un conjunto de puntos etiquetados.

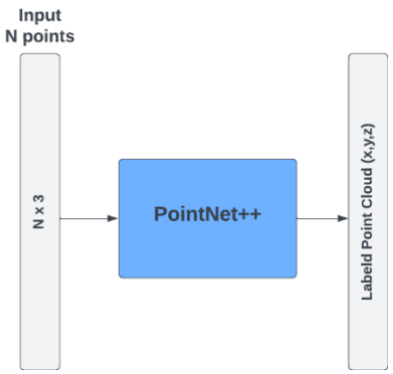


**Figura 2.3 Arquitectura de la red de percepción de objetos**

## 2.5 Detector de Poses de Agarre

Para el proceso de detección de poses de agarre se diseñó una arquitectura basada en investigaciones previas las cuales han demostrado buenos resultados en la extracción de características para la detección de poses de agarre sobre un objeto. Para este proyecto se propone la implementación de un modelo de aprendizaje profundo utilizando la información de los objetos en escena y del planificador de movimientos.

### 2.5.1 Método de recolección de datos

Se diseñó un sistema para la generación del conjunto de datos de entrenamiento que recopila la información dentro de un ambiente simulado. Para la recopilación de datos de prueba se diseñó un sistema para la generación de escenarios aleatorios con el fin de obtener información para la prueba del agarre de objetos simples.

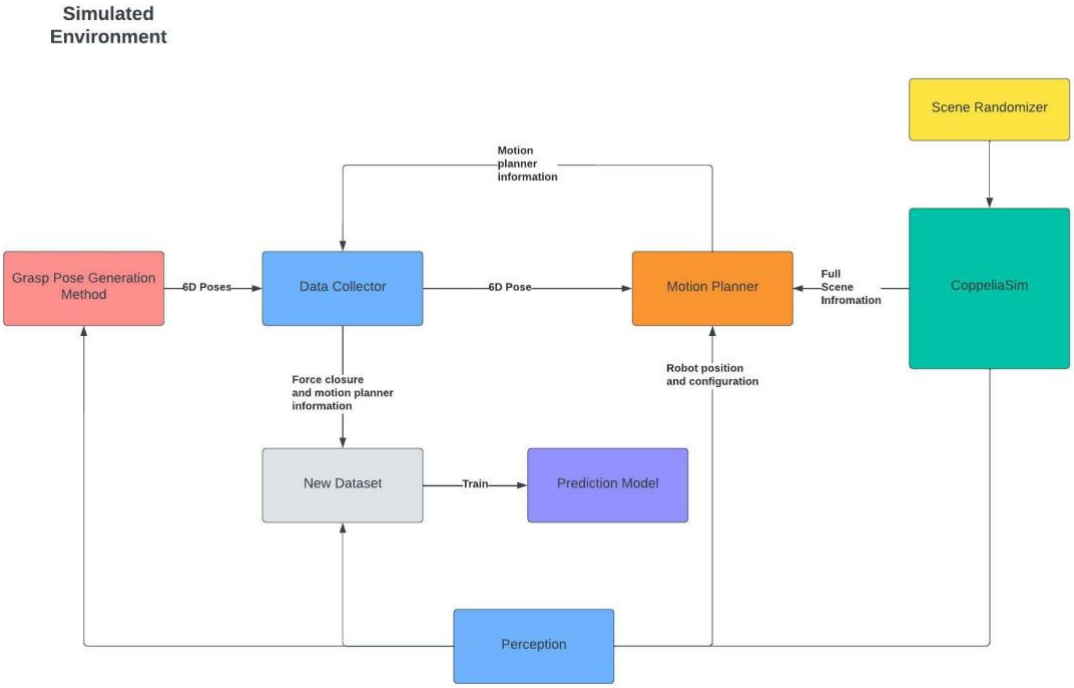


**Figura 2.4 Arquitectura del Sistema de Recolección de Datos de Prueba**

El flujo de trabajo de cada uno de los componentes involucrados es el siguiente:

1. El aleatorizador de escenas produce una descripción de escena aleatoria basada en las variables de entrada.
2. Se construye una escena basada en la descripción de la escena utilizando la interfaz de programación de Coppeliasim.
3. Se recopilan los datos de las nubes de puntos y “objectness” como entrada para el generador de poses y almacenarlos en el conjunto de datos de prueba.
4. El método de generación de agarre recibe la entrada y produce un conjunto de posibles candidatos de agarre.
5. Los planificadores de movimiento prueban las posturas de agarre y la información del planificador de movimiento se almacena, junto con el valor de cierre de fuerza en el conjunto de datos para su uso posterior.

**Método de generación de poses de prueba**

Para lograr generar el conjunto de datos de entrenamiento con las etiquetas de agarre en las escenas aleatorias se optó por un método de generación de poses de agarre que toma como referencia la geometría de los objetos de prueba, la cual estima la localización que correspondería al TCP del efector final y su orientación respecto al sensor de visión, para el agarre.

Este método realiza la estimación de las normales orientadas hacia el interior de la superficie para la obtener la orientación esperada. Para lograrlo, se utilizó el algoritmo KDTree, el cual obtiene dos puntos cercanos en las vecindades de cierto punto, luego se calcula los vectores asociados a estos dos puntos, y calcula el producto cruz para estimar la normal [24].

Teniendo en cuenta cierto vector normal unitario estimado ($\vec{B}$) y el eje z del marco de referencia de la cámara ($\vec{A}$), se debe encontrar la matriz de rotación que rote el vector $\vec{A}$ hacia el vector $\vec{B}$. Esta rotación está definida por la fórmula de rotación de Rodrigues:

$$Rot(u,\phi) = \mathrm{I}\cos\phi + uu^{\mathrm{T}}(1-\cos\phi) + \hat{u}\sin\phi \qquad \textbf{(2.1)}$$

donde $u$ es el vector del eje unitario y $\phi$ es el ángulo de rotación.
Para obtener el vector del eje:

$$u = \vec{A} \times \vec{B}$$

Además, se sabe que:

$$\sin\phi = \left|\vec{A} \times \vec{B}\right|$$

$$\cos\phi = \vec{A} \cdot \vec{B}$$

En el caso de la ubicación este método obtiene la coordenada que contenga el mayor valor para el eje X ($X_{max}$) y el mínimo ($X_{min}$), con su diferencia se obtiene el diámetro del cilindro (D). Además suponiendo que para cierto punto P(x, y, z), se tiene su matriz de rotación ($Rot(u,\phi)$), entonces la matriz de transformación homogénea (MTH) asociada al sensor de visión sería:

$$T_V = \begin{bmatrix} Rot(u,\phi) & P \\ 0\,0\,0 & 1 \end{bmatrix}$$

Sin embargo, esta MTH no representa la pose de agarre, debido a que el TCP no puede agarrar al objeto en la periferia de la superficie, por lo que falta desplazarlo el radio en el eje z. La MTH que representa a la pose de agarre sería, del objeto respecto al sensor sería:

$$T_O^S = T_V \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & D/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### 2.5.2 Selección del método de detección de poses de agarre

Para la detección de poses de agarre se optó por implementar una red neuronal utilizando métodos de aprendizaje profundo. Como backbone de nuestra red neuronal se utilizó una red base para segmentar la nube de puntos generada por el sensor de profundidad, para esto se realizó un análisis para seleccionar el método que se adaptaba mejor a las necesidades.

De entre las soluciones ya existente para la tarea de clasificación y segmentación de nubes de puntos se realizó un análisis sobre tres arquitecturas descritas a continuación:

**PointNet++**

PointNet como pionera en la utilización de métodos de aprendizaje profundo ha demostrado un gran rendimiento con respecto a las tareas de clasificación y segmentación de nubes de puntos. Sin embargo, su diseño no le permite capturar las características de áreas con patrones detallados o generalizar espacios complejos. PointNet++ como una mejora del modelo base de PointNet introduce una red neuronal jerárquica que aplica las operaciones de manera recursiva en particiones de la nube de puntos de entrada. Al realizar estas operaciones de manera jerárquica permite explotar las métricas espaciales de distancia del escenario para obtener características con mayor contexto. Dentro de los

experimentos realizados con PointNet++ se ha visto la capacidad de aprender característica de las nubes de puntos de manera eficiente y sólida [23].

**Dynamic Graph CNN**

Dentro del campo de clasificación y segmentación de nubes de puntos el uso de redes neuronales convolucionales para su análisis y aprendizaje ha sido un éxito. DG-CNN propone la implementación de una arquitectura capaz de explotar una característica que de manera inherente no se encuentran en las nubes de puntos pero que al recuperar su información puede enriquecer su representación como lo es la topología de esta. Dentro de DG-CNN se introduce EdgeConv un módulo de aprendizaje capaz de clasificar y segmentar nubes de puntos en áreas geométricas altamente dinámicas. Se ha demostrado que DG-CNN tiene una afinidad obteniendo características semánticas sobre espacios con distancias potencialmente largas [25].

**Point Transformer**

Los transformers son un método innovador en el área del aprendizaje profundo que provee una mejora en el contexto de lo que aprenden nuestras redes neuronales. Point Transformer propone un diseño de capas de autoatención para nubes de puntos y las usa para construir redes de autoatención para tareas como la segmentación semántica de escenas, la segmentación de partes de objetos y la clasificación de objetos [26].

Dentro de las soluciones descritas, PointNet++ se posiciona como un candidato sólido para tareas de clasificación y segmentación de nubes de puntos, teniendo un rendimiento similar a las demás opciones, sin embargo esta posee la ventaja de ser fácil de implementar, y no requiere de mucha capacidad de cómputo, como puede ser en el caso de DG-CNN la cual genera una representación topológica dentro del espacio visualizado, o en el uso de Point Transformer el cual aunque permita obtener características contextuales en el espacio visualizado, requiere de muchas unidades de activación para su implementación haciendo que el proceso de entrenamiento sea menos eficiente en términos de tiempo. Además,

dentro del espacio de entrenamiento de trabajo realizado sólo se cuentan con objetos estáticos por lo que arquitecturas simples como PointNet++ proveen de manera suficiente un método adecuado para realizar las tareas de clasificación y segmentación.

### 2.5.3 Arquitectura de la red

Para el desarrollo del método de generación de poses de agarre se optó por la construcción de un modelo de aprendizaje profundo utilizando como base la arquitectura de GraspNet [20]. Esta arquitectura se divide en dos etapas, la etapa de estimación de poses de agarre, y la etapa de generación de poses de agarre.

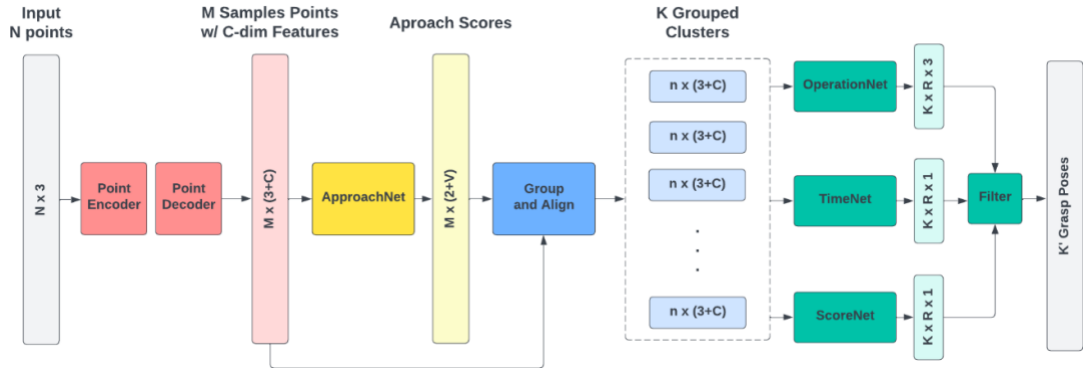


**Figura 2.5 Arquitectura de la red de detección de poses de agarre.**

#### Estimación de Poses de Agarre

#### PointNet++

En la primera etapa de la arquitectura que corresponde a la estimación de poses de agarre, se encuentra como backbone de la red neuronal el módulo de percepción anteriormente mencionado, la cual utiliza la arquitectura de PointNet++ para la segmentación de la nube de puntos.

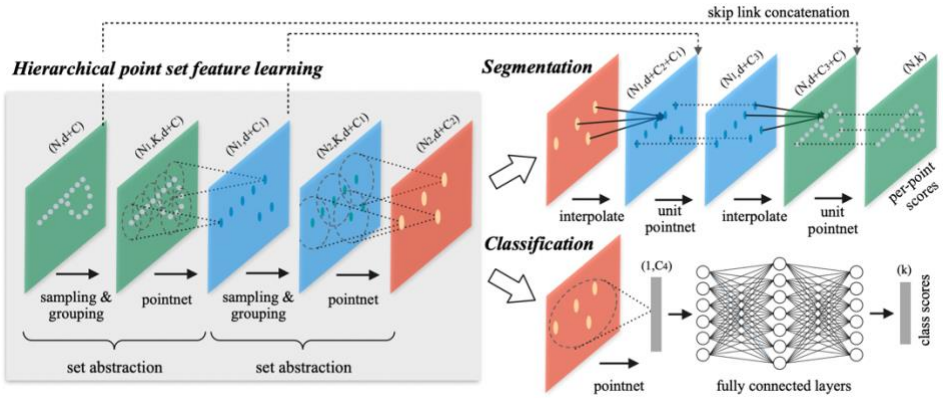


**Figura 2.6 Arquitectura de la red de PointNet++.**

Para la implementación del backbone se utilizaron 4 Módulo SA de PointNet++ que corresponden al llamado “Conjunto de Abstracción”. Estos módulos están implementados utilizando un Shared MLP de 4 capas tal y como se describen en la Tabla 2.10.

**Tabla 2.10 Detalles de los modulos SA de la red PointNet++**

| Módulo | Tipo | Tamaño de la Entrada | Función de Activación |
|---|---|---|---|
| SA1 | SharedMLP | [3, 64, 64, 128] | ReLU |
| SA2 | SharedMLP | [128, 128, 128, 256] | ReLU |
| SA3 | SharedMLP | [256, 128, 128, 256] | ReLU |
| SA4 | SharedMLP | [256, 128, 128, 256] | ReLU |

Además, se utilizaron dos capas para la propagación de características. Estos módulos están implementados utilizando un Shared MLP de 3 capas tal y como se describen en la Tabla 2.11.

**Tabla 2.11 Detalles de los modulos FP de la red PointNet++**

| Módulo | Tipo | Tamaño de la Entrada | Función de Activación |
|---|---|---|---|
| FP1 | SharedMLP | [512, 256, 256] | ReLU |
| FP2 | SharedMLP | [512, 256, 256] | ReLU |

### ApproachNet

Para estimar las poses de agarre a partir de la nube de puntos segmentada se utilizó ApproachNet, que permite clasificar los vectores de aproximación del objeto

dentro de V puntos de aproximación predefinidos. Dentro de ApproachNet se realiza el cálculo del objectness de cada punto para determinar si en los puntos de aproximación predefinidos el objeto es agarrable o no.

Para la implementación de ApproachNet se utilizaron 3 capas convolucionales de una dimensión y dos capas de normalización. Además, se utiliza como entrada el número de puntos de aproximación predefinidos en este caso 300. Los módulos utilizados están descritos en la Tabla 2.12.

**Tabla 2.12 Detalles de los modulos de la red ApproachNet**

| Módulo | Tipo | Tamaño de la Entrada | Función de Activación |
|---|---|---|---|
| Conv1 | MLP | [256,256] | ReLU |
| Conv2 | MLP | [256,302] | ReLU |
| Conv3 | MLP | [302, 302] | ReLU |
| BN1 | MLP | 256 | ReLU |
| BN2 | MLP | 302 | ReLU |

Dentro de ApproachNet se realiza el cálculo de la función de pérdida especificada en la ecuación 2.1.

$$L^A(\{c_i\},\{s_{ij}\}) = \frac{1}{N_{cls}}\sum SoftMax(c_i, c_i^*) + \lambda_1 \frac{1}{N_{reg}} \sum_i \sum_j c_i^* 1\left(|v_{ij}, v_{ij}^*| < 5^\circ\right) L_1(s_{ij}, s_{ij}^*) \quad \textbf{(2.2)}$$

En la ecuación descrita, C denota la predicción binaria de si un punto es agarrable o no, C* es 1 si el punto i es positivo y 0 es negativo, S denota el puntaje de confiabilidad del punto de aproximación j del punto i, S* corresponde al puntaje máximo de confiabilidad de agarre del punto de vista j, |V, V*| denota la diferencia de grados entre puntos de aproximación, el cual debe ser menor a 5°.

Luego de las operaciones se obtiene como salida un vector N x ( 2 + V ), donde el 2 representa los valores binarios de si el objeto es agarrable o no en dicho punto.

Además, ApproachNet añade las características de la estimación de poses a cada uno de los puntos, siendo una de estas la matriz de rotación.

### Generación de Poses de Agarre

#### CloudGrouping

Luego de estimar las poses de agarre y añadir las características de estimación al conjunto de entrada, se utilizó un método de agrupamiento para generar un área cilíndrica alrededor de los puntos utilizando como entrada la matriz de rotación generada a partir de los vectores de aproximación en la etapa de estimación de poses. Este método retorna un K grupos de vectores N x ( 3 + C ) que encierran en un radio las regiones de agarre más factibles.

Para la implementación de esta parte de la red se utiliza una SharedMLP de 4 capas, con 64, 3, 128 y 256 respectivamente como tamaño de la entrada.

#### OperationNet

Utilizando la nueva representación de las poses de agarre, estas pasan a través de la OperationNet la cual realiza una tarea de clasificación que predice el ángulo de rotación y el ancho de la pinza de agarre. Dado que la pinza de agarre es simétrica sólo se predicen rotaciones desde 0° a 180°.

Para la implementación de OperationNet se utilizó 3 capas Convolucionales y 1 capa de normalización. Se utilizó como entrada el número de clases de ángulos, en este caso 12, y también el número de clases de profundidad de la pinza de agarre, en este caso 3. Los módulos utilizados están descritos en la Tabla 2.13.

**Tabla 2.13 Detalles de los modulos de la red OperationNet**

| Módulo | Tipo | Tamaño de la Entrada | Función de Activación |
|---|---|---|---|
| Conv1 | MLP | [256, 128] | ReLU |
| Conv2 | MLP | [128, 128] | ReLU |
| Conv3 | MLP | [128, 36] | ReLU |
| BN1 | MLP | 128 | ReLU |

La función objetivo de esta parte de la red es descrita en la ecuación 2.2.

$$L^R(R_{ij}, S_{ij}, W_{ij}) = \sum_{d=1}^{K} \frac{1}{N_{cls}} \sum_{ij} SigmoidCrossEntropy(R_{ij}, R_{ij}^*) + \lambda_2 \frac{1}{N_{reg}} \sum_{ij} L_1^d(S_{ij}, S_{ij}^*) + \lambda_3 \frac{1}{N_{reg}} \sum_{ij} L_1^d(W_{ij}, W_{ij}^*) \quad \textbf{(2.3)}$$

**TimeNet**

Después de pasar por la red de operación, la red ya es capaz de predecir poses de agarre precisas. Sin embargo, para mejorar su confiabilidad se añadió otra característica, en este caso el tiempo de planificación de movimiento.

Para la implementación de TimeNet se utilizó 3 capas convolucionales y 1 capa de normalización. Se utilizó como entrada el número de clases de ángulos, en este caso 12, y también el número de clases de profundidad de la pinza de agarre, en este caso 3. Los módulos utilizados están descritos en la Tabla 2.14.

**Tabla 2.14 Detalles de los modulos de la red TimeNet**

| Módulo | Tipo | Tamaño de la Entrada | Función de Activación |
|---|---|---|---|
| Conv1 | MLP | [256, 128] | ReLU |
| Conv2 | MLP | [128, 128] | ReLU |
| Conv3 | MLP | [128, 36] | ReLU |
| BN1 | MLP | 128 | ReLU |

Dado un ground truth de las poses de agarre, utilizando el tiempo de la planificación de movimiento añadido en el método de recolección de datos se realiza una función de perdida según la ecuación.

$$L^T(A_{ij}) = \frac{1}{N_{reg}} \sum_{d=1}^{K} \sum_{ij} L_1^d(T_{ij}, T_{ij}^*) \quad \textbf{(2.4)}$$

Donde T corresponde al tiempo de la planificación de movimiento.

**ScoreNet**

Finalmente se utiliza el puntaje de fuerza de cierre que permite discriminar finalmente las poses de agarre con mayor confiabilidad con respecto al agarre del objeto. Se utilizó como entrada el número de clases de ángulos, en este caso 12, y también el número de clases de profundidad de la pinza de agarre, en este caso 3. Los módulos utilizados están descritos en la Tabla 2.15.

**Tabla 2.15 Detalles de los modulos de la red ScoreNet**

| Módulo | Tipo | Tamaño de la Entrada | Función de Activación |
|---|---|---|---|
| Conv1 | MLP | [256, 128] | ReLU |
| Conv2 | MLP | [128, 128] | ReLU |
| Conv3 | MLP | [128, 36] | ReLU |
| BN1 | MLP | 128 | ReLU |

Para el entrenamiento se utilizó la función de perdida descrita en la ecuación 2.4.

$$L^S(A_{ij}) = \frac{1}{N_{reg}} \sum_{d=1}^{K} \sum_{ij} L_1^d(F_{ij}, F_{ij}^*) \quad \textbf{(2.5)}$$

Donde F corresponde al puntaje de fuerza de cierre de la pinza de agarre.

**Entrenamiento**

**Conjunto de Datos**

Se utilizó el conjunto generado por el módulo de recolección de datos. Se generaron 50 escenas cada una con 256 imágenes RGB, de profundidad y además los metadatos de las características intrínsecas de la cámara. La organización de los archivos se describe en la Figura 2.7.

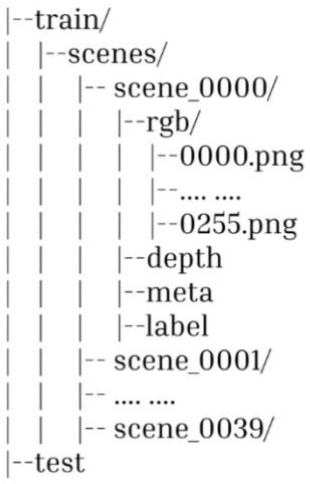


**Figura 2.7 Estructura de archivos del conjunto de datos de prueba.**

Se utilizaron 40 y 10 escenas, para el entrenamiento y las pruebas respectivamente.

**Función de Costos**

Para el entrenamiento de la red neuronal se utiliza la función de costo descrita en la ecuación 2.5 que unifica las funciones de costo de cada etapa de la red para la actualización de los parámetros:

$$L = L^A(\{c_i\},\{s_{ij}\}) + \alpha L^R(R_{ij},S_{ij},W_{ij}) + \beta L^T(A_{ij}) + \gamma L^S(A_{ij}) \quad \textbf{(2.6)}$$

## 2.6 Planificación de movimiento

### 2.6.1 Selección de algoritmo de planificación de movimiento

Para la selección del algoritmo de planificación de movimiento se realizaron pruebas experimentales simuladas para obtener datos que permitieran comparar el tiempo de cómputo del camino y la distancia angular total del camino.

Para el tiempo de cómputo se obtuvo el tiempo en el cual el algoritmo comienza a ejecutarse ($t_0$) y el tiempo en el que termina de ejecutarse ($t_f$), y se calcula la diferencia mediante la ecuación 2.6:

$$\Delta t = t_f - t_0 \quad \textbf{(2.6)}$$

Por otra parte, para la distancia angular total del camino se utilizó la ecuación 2.7:

$$|\Theta| = \sum_{i=1}^{k-1} \sum_{j=1}^{n} |q_{ji} - q_{j(i+1)}| \quad \textbf{(2.7)}$$

Donde i representa el i-ésimo punto de ruta con un total de k puntos en el camino, j representa la j-ésima junta revoluta con un total de n juntas revolutas (7 en el caso del Franka Emika Panda) y q representa la variable articular en radianes.

Para los experimentos, para cada uno de los algoritmos de la tabla 2.10a y 2.10b se inició desde una pose inicial x0= (0.5960 m, -0.00186 m, 0.5211 m, 180°, 0°, 45°)T a una pose final xf= (0.1 m, 0.1 m, 0.5 m, 180°, 0°, -100°)T , el marco de referencia utilizado para los ángulos de las poses fue Roll-Pitch-Yaw.

En la tabla Tabla 2.18 se pueden observar las medias del tiempo de cómputo y la distancia angular de los algoritmos de planificación, las cuales serán útiles para realizar la selección.

**Tabla 2.16 Tiempo de cómputo del camino y la distancia angular total del camino para algoritmos BiTRRT, BKPIECE y RRTConnect**

| | BiTRRT | | BKPIECE1 | | RRTConnect | |
|---|---|---|---|---|---|---|
| **No.** | **$\Delta t$ (s)** | **$\|\Theta\|$ (rad)** | **$\Delta t$ (s)** | **$\|\Theta\|$ (rad)** | **$\Delta t$ (s)** | **$\|\Theta\|$ (rad)** |
| **1** | 0,5640 | 2,1102 | 1,2950 | 2,8701 | 0,1230 | 2,0321 |
| **2** | 0,2590 | 1,8777 | 0,7730 | 4,2478 | 0,2170 | 2,7542 |
| **3** | 0,3930 | 1,8554 | 0,5050 | 2,4889 | 0,1870 | 1,8547 |
| **4** | 0,5100 | 2,1971 | 1,0670 | 1,8530 | 0,2159 | 1,8583 |
| **5** | 0,3119 | 1,8829 | 0,4499 | 2,7095 | 0,1069 | 1,1457 |
| **6** | 0,3159 | 1,8609 | 0,7160 | 2,1969 | 0,1760 | 1,8991 |
| **7** | 0,5770 | 2,4237 | 1,6160 | 2,6725 | 0,1779 | 2,0098 |
| **8** | 0,3670 | 1,8801 | 1,7529 | 2,4574 | 0,1749 | 1,9074 |
| **9** | 0,4269 | 1,8812 | 1,1840 | 2,1320 | 0,1739 | 1,8561 |
| **10** | 0,3549 | 1,8866 | 2,6239 | 2,0220 | 0,1490 | 2,0432 |
| **11** | 0,5370 | 1,9078 | 0,8809 | 2,4383 | 0,2300 | 1,9531 |

**Tabla 2.17 Tiempo de cómputo del camino y la distancia angular total del camino para algoritmos PRM, EST y SBL**

| | PRM | | EST | | SBL | |
|---|---|---|---|---|---|---|
| **No.** | $\Delta t$ **(s)** | $\lvert\Theta\rvert$ **(rad)** | **Tiempo (s)** | $\lvert\Theta\rvert$ **(rad)** | $\Delta t$ **(s)** | $\lvert\Theta\rvert$ **(rad)** |
| **1** | 0,1260 | 2,4915 | 0,2870 | 2,2224 | 0,3930 | 1,8530 |
| **2** | 0,2400 | 2,8019 | 0,0680 | 1,8774 | 0,3060 | 1,8809 |
| **3** | 0,3079 | 2,5007 | 0,0720 | 3,8200 | 0,5740 | 4,5569 |
| **4** | 0,3129 | 2,6415 | 0,0950 | 2,6180 | 0,3650 | 2,6257 |
| **5** | 0,3539 | 2,0187 | 0,3389 | 2,0776 | 0,6920 | 1,8625 |
| **6** | 0,4239 | 3,0060 | 0,0610 | 3,1880 | 0,6300 | 2,2576 |
| **7** | 0,3650 | 2,0625 | 0,2749 | 3,2971 | 0,6999 | 2,3373 |
| **8** | 0,3410 | 1,8724 | 0,0749 | 4,5792 | 0,3260 | 2,9289 |
| **9** | 0,4049 | 3,9040 | 0,0320 | 1,9764 | 0,3579 | 2,3143 |
| **10** | 0,3000 | 3,9027 | 0,0380 | 2,2956 | 0,6160 | 3,2740 |
| **11** | 0,3570 | 1,9944 | 0,0600 | 2,7942 | 0,7300 | 2,0552 |

**Tabla 2.18 Tiempo de cómputo del camino y la distancia angular total medios del camino de los algoritmos de planificación de movimiento**

| | BiTRRT | BKPIECE1 | RRTConnect | PRM | EST | SBL |
|---|---|---|---|---|---|---|
| $\overline{\Delta t}$ **(s)** | 0,4197 | 1,1694 | 0,1756 | 0,3211 | 0,1274 | 0,5173 |
| $\lvert\bar{\Theta}\rvert$ **(rad)** | 1,9785 | 2,5535 | 1,9376 | 2,6542 | 2,7951 | 2,5406 |

Se puede observar que entre los algoritmos más rápidos se encuentran el RRTConnect con 0.1756 segundos de cómputos y el EST con 0.1274 segundos. Mientras que con las distancias angulares óptimas se encuentran el BiTRRT con 1.9785 rad y el RRTConnect con 1.9376 rad. El RRTConnect es el algoritmo con distancia angular media más corta; mientras que por la parte del tiempo de cómputo se tiene al EST. Sin embargo, se prioriza a la distancia, debido a que una menor distancia (menos puntos de ruta) implica menor procesamiento en la parte de control.

### 2.6.2 Implementación del algoritmo de planificación de movimiento

La función del planificador de movimiento es obtener puntos de ruta que lleve al robot desde una configuración inicial a una final. Debido a que lo que se desea alcanzar con el manipulador son objetivos en el espacio de tareas, primero se deben obtener las soluciones de la cinemática inversa para la pose inicial del robot y la final (objetivo). Para la cinemática inversa se utilizó el método del Jacobiano

pseudoinverso, el cual es un método computacional bastante conocido en el campo de la robótica y que el simulador ofrece para su implementación. Dada la solución inicial y final, se puede planificar en el espacio articular mediante la utilización de diversos algoritmos ofrecidos por la librería OMPL.

Como se mencionó previamente, se necesita la pose del objetivo para realizar la planificación. El sensor de visión junto con el generador de poses, obtendrán las poses del objeto respecto a la cámara, para luego aplicarle una transformación a esta pose para obtenerla respecto a la base. Sea $T_G^B$, la transformación que va de la base del robot al punto central de la herramienta (TCP); $T_S^G$, la transformación del TCP hasta el sensor de visión y $T_O^S$, la transformación del sensor hasta el objeto; se desea conocer la transformación del objeto respecto a la base del robot ($T_O^B$), para lo cual se utilizaría la ecuación 2.8:

$$T_G^B T_S^G T_O^S = T_O^B \tag{2.8}$$

La transformación $T_S^G$, dado que el sensor de visión está sobre el efector final, es constante y está expresada por la siguiente matriz:

$$T_S^G = \begin{bmatrix} 0 & 1 & 0 & -0.0234 \\ 1 & 0 & 0 & 0.002 \\ 0 & 0 & 1 & -0.0471 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La transformación $T_G^B$ corresponde a la cinemática directa del manipulador Franka Emika Panda, esta transformación depende de la configuración. $T_G^B$ correspondería a la pose en la cual se capturen las imágenes de profundidad y RGB, mientras que $T_O^B$ sería la pose que el TCP del manipulador alcanzará (o no) en determinado momento.

Cabe recalcar que las ecuaciones 2.1 y 2.2 se utilizarán para entrenar la red, de tal manera que la red aprenda las poses que generen caminos más cortos y con un menor tiempo de cómputo.

## 2.7 Módulo de control

### 2.7.1 Selección del método de control

Para seleccionar el método de control que se utilizará, se empleará el análisis de la matriz de decisión para las alternativas: Control por Cómputo de Torque (CCT), Control PD por compensación de gravedad (PDGC) y Control por Impedancia (CI). Cada una de estas alternativas fueron evaluadas de acuerdo a los siguientes criterios, utilizando las ponderaciones de la Tabla 2.19:

**Seguimiento de trayectoria (ST).-** Posibilidad del controlador para seguir una trayectoria.

**Error en estado estable (ESE).-** Capacidad del controlador para garantizar error nulo en estado estable.

**Complejidad en la selección de parámetros (CSP).-** Facilidad para elegir los parámetros sin que estos afecten en gran medida al error en estado estable del sistema.

**Sensibilidad respecto al modelo dinámico (SMD)**.- Grado en el que el sistema de control es afectado por la precisión del modelado matemático.

**Tabla 2.19 Ponderaciones de los criterios de selección del método de control**

| **Criterio** | ST | ESE | CSP | SMD | **Σ+1** | **ponderación** |
|---|---|---|---|---|---|---|
| ST | | 1 | 1 | 1 | 4 | 0,40 |
| ESE | 0 | | 1 | 1 | 3 | 0,30 |
| CSP | 0 | 0 | | 1 | 2 | 0,20 |
| SMD | 0 | 0 | 0 | | 1 | 0,10 |
| | | | | **Suma** | 10 | 1,00 |

Tal como se puede observar en la Tabla 2.20, la solución ganadora fue el método de CCT. A pesar de que esta solución podría ser sensible al modelo dinámico, debido a que el proyecto se lleva a cabo en simulaciones, los entornos físicos simulados idealizan los sistemas y las perturbaciones, haciendo a esta opción viable. Por otra parte, la selección de parámetros es sencilla, lo cual facilitaría a los usuarios el cambio de parámetros, de acuerdo con el comportamiento deseado. Este tipo de control es apto para el seguimiento de trayecto, y además garantiza errores teóricamente nulos en estado estable.

**Tabla 2.20 Matriz de decisión de alternativas de solución**

| Conclusión | ST | ESE | CSP | SMD | Σ+1 | prioridad |
|---|---|---|---|---|---|---|
| CCT | 0,2 | 0,125 | 0,0833 | 0,025 | 0,4333 | 1 |
| PDGC | 0,1 | 0,125 | 0,0833 | 0,025 | 0,3333 | 2 |
| IC | 0,1 | 0,05 | 0,0333 | 0,05 | 0,2333 | 3 |

### 2.7.2 Implementación del módulo de control

Para el módulo de control de movimiento del manipulador, se tiene como entrada a la trayectoria generada por el planificador de movimiento. Entiéndase por trayectoria a los estados angulares de las juntas más la respectiva dimensión temporal correspondiente a cada uno de estos estados. Conociendo la trayectoria deseada es sencillo obtener la velocidad y aceleración para cualquier punto de la trayectoria. El manipulador podría tener perturbaciones iniciales o durante la trayectoria, por lo que la función del controlador sería entregar como salida los torques para reducir los errores provocados por estas perturbaciones.

El simulador proporciona funciones para obtener la posición (q) y velocidad angular ($\dot{q}$) de las juntas del manipulador; sin embargo, no existen funciones para la aceleración angular, por esta razón se aproximará la aceleración mediante diferenciación numérica centrada (Ecuación 2.9):

$$\ddot{q}(t) = \frac{\delta \dot{q}}{\delta t} \approx \frac{\dot{q}(t+1) - \dot{q}(t-1)}{2\Delta t} \quad \textbf{(2.9)}$$

Sin embargo, para el primer punto de la trayectoria no existe $\dot{q}(t-1)$; mientras que para el último no existe $\dot{q}(t+1)$. Para estos casos se utilizará diferenciación numérica hacia adelante (Ecuación 2.10) y hacia atrás (Ecuación 2.11) correspondientemente:

$$\ddot{q}(t_0) \approx \frac{\dot{q}(t+1) - \dot{q}(t)}{\Delta t} \quad \textbf{(2.10)}$$

$$\ddot{q}(t_f) \approx \frac{\dot{q}(t) - \dot{q}(t-1)}{\Delta t} \quad \textbf{(2.11)}$$

Se utilizará el enfoque del control por cómputo de torque, debido a que el simulador ofrece funciones que permiten computar elementos matriciales de la dinámica del manipulador, los cuales serán necesarios para realizar el control.

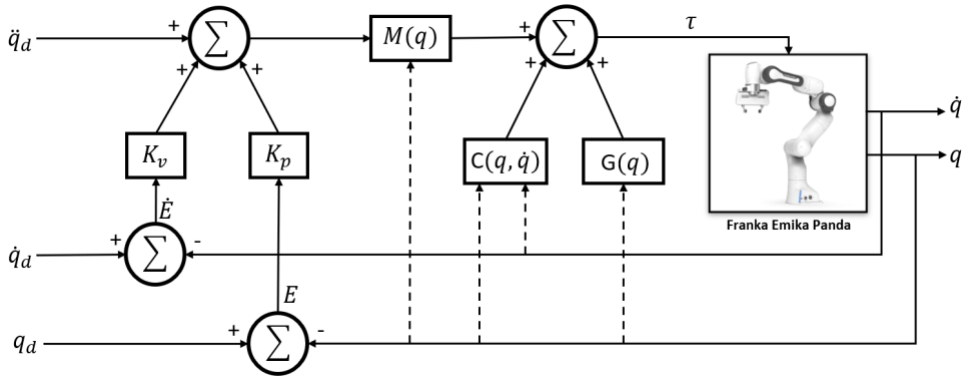


**Figura 2.8 Diagrama de bloques del control por cómputo de torque.**

En la Figura 2.8 se puede apreciar el esquema de control de movimiento que se está siguiendo. En este tipo de control, la principal característica es la cancelación efectiva de los términos no lineales gracias a la utilización de la dinámica del robot para compensar su propio masa-inercia, efectos Coriolis y gravitatorios. De esta manera, y de acuerdo con el gráfico, los torques enviados a las articulaciones del manipulador estarían determinados por la Ecuación 2.12:

$$\tau = M\left(\ddot{q}_d + K_p q_e + K_d \dot{q}_e\right) + C(q,\dot{q}) + G(q) \qquad \textbf{(2.12)}$$

El sistema en lazo cerrado está dado por la Ecuación 2.13:

$$\ddot{E} + K_d \dot{E} + K_p E = 0 \qquad \textbf{(2.13)}$$

Donde $K_p$ y $K_d$ son matrices diagonales positivas definidas, las cuales contienen en sus diagonales a cada una de las constantes proporcionales y derivativas de cada una de las juntas a las que se le aplicará el control. $E = q_d - q$ representa el error entre la configuración deseada y la configuración real.

Dada una frecuencia natural deseada y una razón de amortiguamiento, se pueden obtener los elementos de las diagonales ($k_{pi}$ y $k_{di}$) de estas matrices correspondientes para cada junta, como sigue:

$$k_{p_i} = {\omega_{n_i}}^2 \qquad \textbf{(2.14)}$$

$$k_{v_i} = 2\zeta\omega_{n_i} \quad \textbf{(2.15)}$$

No es deseable que el robot presente sobreimpulso en su respuesta dinámica, ya que esto podría provocar un impacto si, por ejemplo, una trayectoria deseada termina en la superficie de una pieza de trabajo [12]. Por lo tanto, las ganancias $k_{p_i}$ y $k_{v_i}$ generalmente se seleccionan para un amortiguamiento crítico $\zeta$ = 1.

Para la selección de la frecuencia natural se efectuaron experimentos, en los cuales se envió una señal escalón que va de 0 a 0.5 rad a cada una de las juntas, lo cual se consideraría como una desviación considerable. Con la razón de amortiguamiento fijada, se fue variando la razón de amortiguamiento para obtener tiempos de estabilización menores, teniendo como límite los torques máximos de las juntas ($\tau_{max} = [87 \quad 87 \quad 87 \quad 87 \quad 12 \quad 12 \quad 12]^T$ Nm). Las frecuencias naturales escogidas se encuentran en la Tabla 2.21.

**Tabla 2.21 Valores de frecuencias naturales escogidas**

| Junta i | $\boldsymbol{\omega_{n_i}}$ $(\boldsymbol{rad/s})$ |
|---|---|
| 1 | 30 |
| 2 | 15 |
| 3 | 30 |
| 4 | 40 |
| 5 | 15 |
| 6 | 30 |
| 7 | 65 |

# CAPÍTULO 3

## 3. RESULTADOS Y ANÁLISIS

En el presente capítulo se muestran los resultados de cada módulo y su respectivo análisis. Es importante tener en cuenta que el cliente solicitó mostrar estos resultados para dos tipos de escenas: un ambiente con objetos ordenados ubicados en una mesa y otro con objetos desordenados con restricciones de agarre dados por una estantería. En la Figura 3.1 se muestran las escenas utilizadas en los experimentos que en lo que resta del presente documento se harán referencia a ellas como “Escena O” (izquierda) y “Escena D” (ordenada) respectivamente.

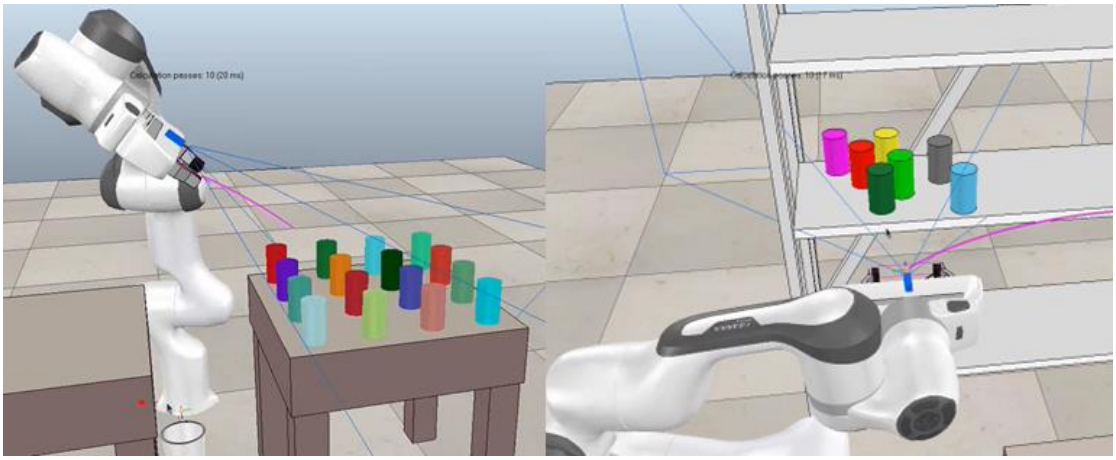

**Figura 3.1 Escenas utilizadas para los experimentos**

### 3.1 Módulo de percepción

En la Figura 3.2 y la Figura 3.3 se presentan las imágenes RGB y las imágenes de profundidad de la escena O y D respectivamente, obtenidas por el sensor de visión. Se debe tener en cuenta que, para las imágenes de profundidad presentadas, las zonas más oscuras representan zonas más cercanas al marco del sensor de visión. Por otra parte, las imágenes RGB, muestran los colores “reales” de la escena. Sin embargo, existe una zona negra, la cual es ocasionada por el límite del alcance del sensor, también conocido como plano de corte. Estas zonas pueden filtrarse fácilmente mediante segmentación en el espacio RGB, y no afectan al presente estudio.

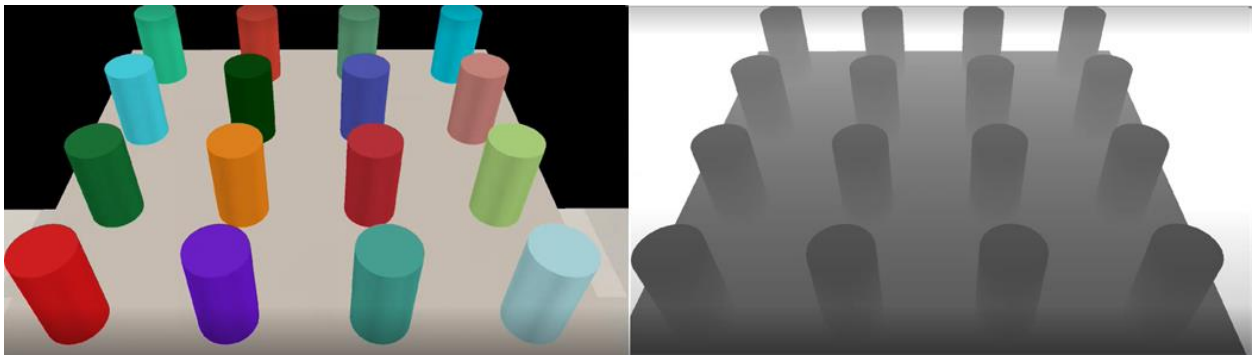

**Figura 3.2 Imágenes RGB (izquierda) y de profundidad (derecha) del escenario O**

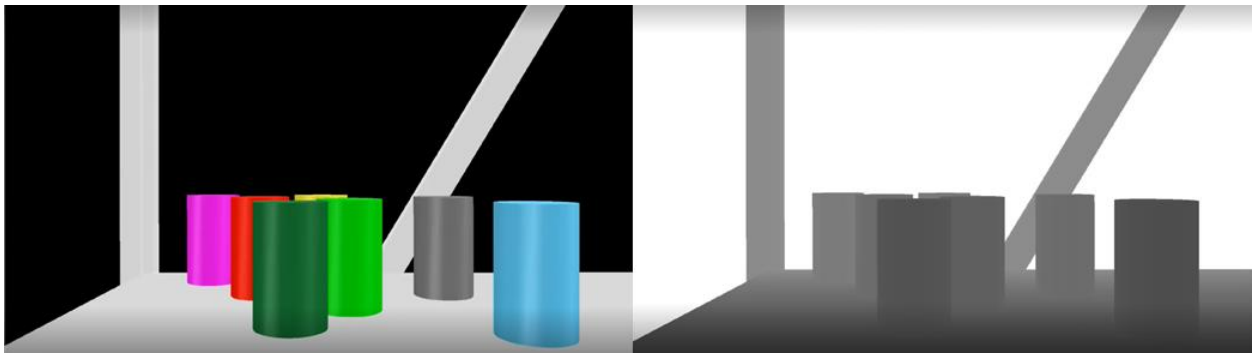

**Figura 3.3 Imágenes RGB (izquierda) y de profundidad (derecha) del escenario D**

Tanto los datos de las imágenes RGB, como los de profundidad se utilizan para producir las nubes de puntos de la Figura 3.4, procesadas mediante la librería Open3D. Las nubes de puntos presentadas tienen en total 4’194,304 puntos y se procesan en 0.25 segundos en promedio. El procesador utilizado para estas pruebas fue una Intel Core i7-8550U CPU @ 1.80GHz - 1.99 GHz.

**Figura 3.4 Nube de puntos de las escenas O (izquierda) y D (derecha)**

En la Figura 3.5 se pueden observar, tanto para la escena O como para la escena D, los diferentes grupos que se han segmentado. Para ambas nubes de puntos se muestran del mismo color los puntos pertenecientes a un mismo *cluster*. Para la escena O, debido a que existen pocas restricciones en el espacio de tareas, la mayoría de los puntos que conforman los cilindros son detectados. Para la escena D se tiene menor visibilidad debido a las restricciones de la estantería, lo cual produce que zonas significantes no se logren detectar.

**Figura 3.5 Segmentación de nube de puntos para las escenas O (izquierda) y D (derecha)**

## 3.2 Módulo de identificación de poses de agarre

En la Figura 3.6 se muestran 2 cilindros pertenecientes a las agrupaciones mencionadas anteriormente. Cada punto de este cilindro está representado por una coordenada tridimensional, la cual ayuda a colocar el TCP del efector final; sin embargo, también es necesario obtener una orientación válida para el mismo. Es por esta razón que en la figura se muestran las normales orientadas hacia el interior de las superficies estimadas mediante KDTree. Dado que el algoritmo usa una superficie discretizada, algunos vectores no son una buena estimación para ser considerados normales, esto fue tomado en cuenta al momento de elegir puntos para las poses de agarre. No obstante, la mayoría de los vectores son normales a la superficie como se percibe en la figura.

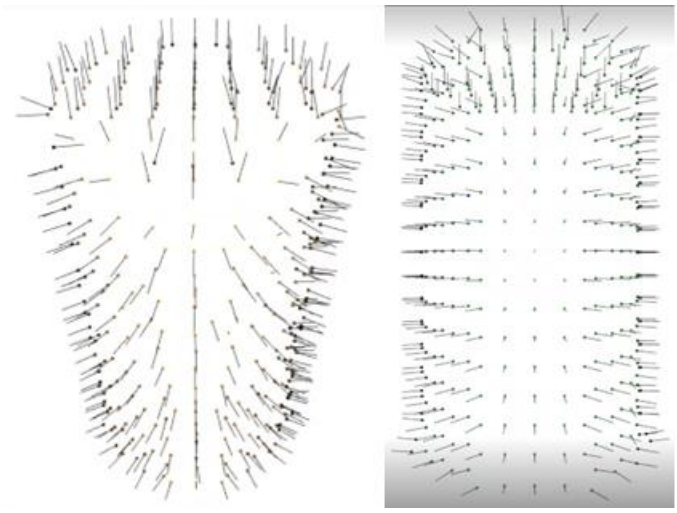

**Figura 3.6 Estimación de normales para nubes de puntos segmentadas.**

Para comprobar la validez de los puntos y orientaciones obtenidas mediante la nube de puntos, se colocaron sus correspondientes marcos de referencia (llamados *Dummies* en CoppeliaSim) en el espacio de simulación. La Figura 3.7 muestra esta verificación, los marcos se encuentran en las ubicaciones identificadas en las nubes de puntos; además, dada la alineación entre los ejes z (eje color azul en los *dummies*) y los vectores normales, no se observan estos ejes azules en abundancia debido a que se encuentran orientados hacia el interior de la superficie. Los ejes azules que se observan en la imagen corresponden a las malas estimaciones debido a la superficie discretizadas. Esto se podría mejorar con una mayor cantidad de muestras en la nube de puntos.

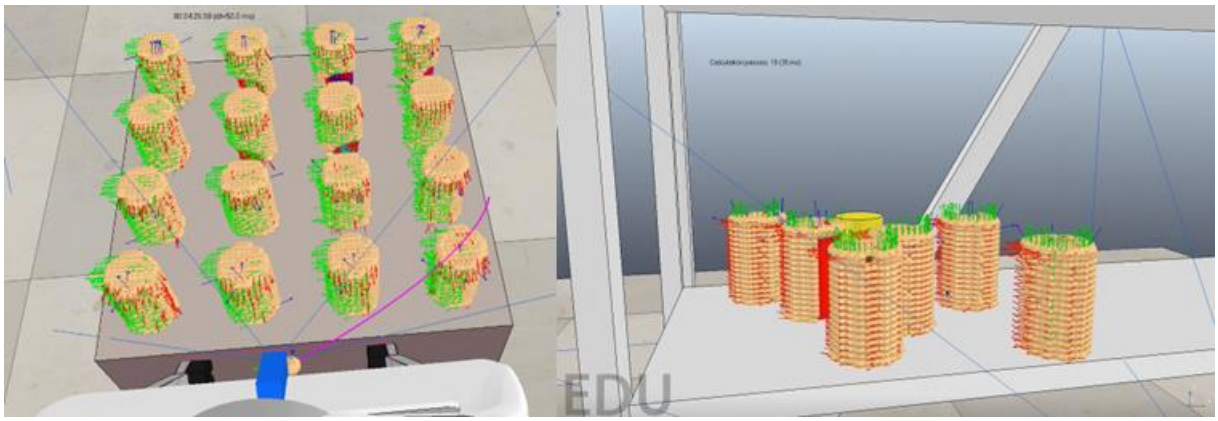

**Figura 3.7 Ubicación de marcos de referencia en las periferias de los objetos a partir de la información de la nube de puntos.**

Para que los marcos presentados en la Figura 3.7 puedan ser consideradas poses de agarre, deben ser desplazadas una distancia de tal manera que los puntos de contacto sean antipodales para cumplir con la condición del cierre por fuerza, esta distancia corresponde al radio, para obtener 2 puntos de contacto en la superficie. En la Figura 3.8 se muestra al efector final en sus poses de agarre (marco desplazado radialmente). De igual manera, la Figura 3.9 muestra al efector final en una pose de agarre para un cilindro. La diferencia radica en que, para la escena O, existen poses válidas orientadas radial y axialmente; mientras que para la escena D solo son válidas orientaciones radiales.

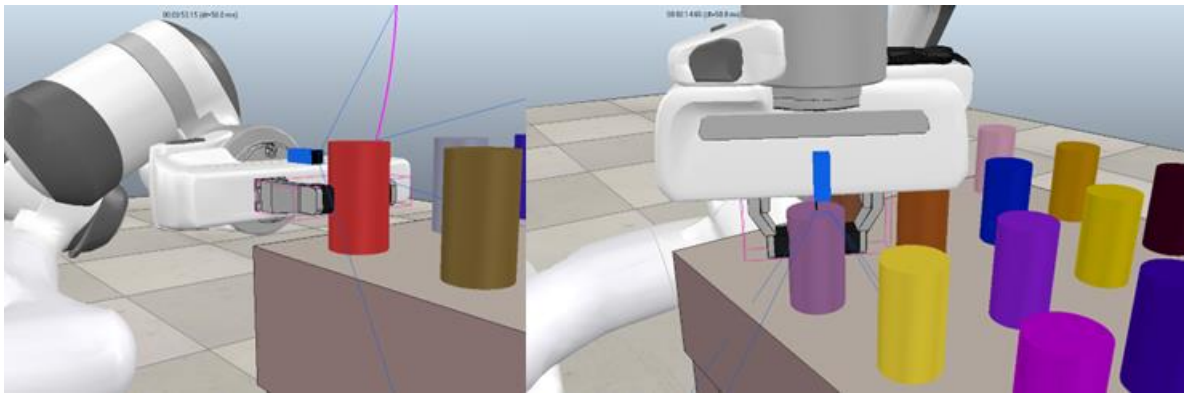

**Figura 3.8 Poses de agarre para escena O**

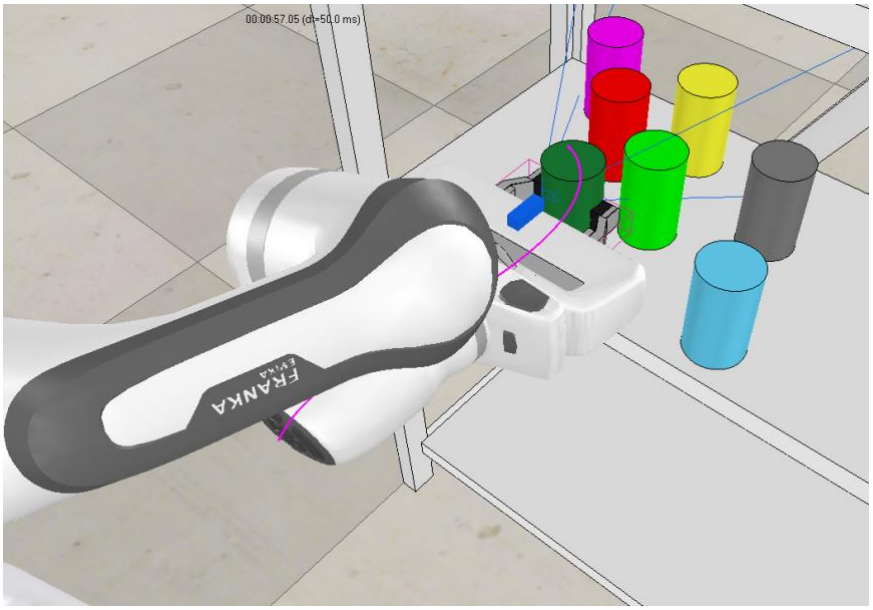


**Figura 3.9 Poses de agarre para escena D**

## 3.3 Módulo de Generación de Poses de Agarre

Para la generación de poses de agarre se utilizó el modelo de red neuronal descrito en el capítulo anterior. Para esta fase del proyecto se realizó el entrenamiento utilizando una computadora de escritorio corriendo Ubuntu 20.04 con una tarjeta gráfica Nvidia GeoForce GTX 1050Ti con 4Gb de GPU, un procesador Intel I5-12400 con 4.40 GHz de frecuencia máxima, y 16 GB de Memoria RAM DDR4 a 3200Mhz. Durante el entrenamiento se realizó un muestreo aleatorio de veinte mil puntos en cada escena del conjunto de datos de entrenamiento con un batch size de 4.

Para el entrenamiento se escogieron los hipeparámetros descritos en la Tabla 3.1 que corresponden a los parámetros de regularización usados en las funciones de pérdida de cada etapa de la red. Estos valores fueron escogidos partiendo de los valores base de Graspnet.

**Tabla 3.1 Valores de regularización de GraspNet**

| Hiperparámetro | Valor |
|---|---|
| $\lambda_1$ | 0.5 |
| $\lambda_2$ | 1.0 |
| $\lambda_3$ | 0.2 |
| $\alpha$ | 0.5 |
| $\beta$ | 0.5 |
| $LR$ | 0.0001 |

Utilizando los valores de regularización base de Graspnet se obtuvieron los resultados de entrenamiento y validación descritos en la Figura 3.10.

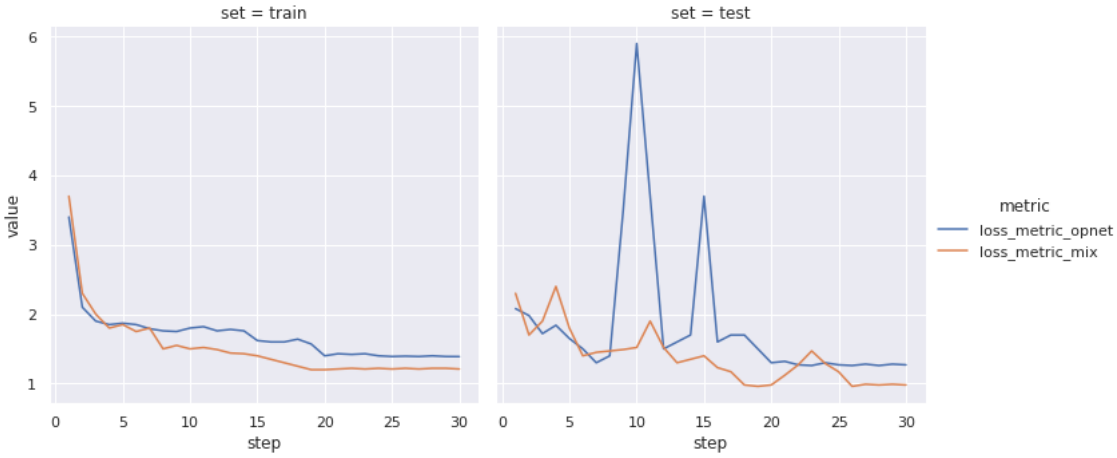


**Figura 3.10 Resultados del entrenamiento con los valores de regularización de GraspNet**

Se realizó un análisis del puntaje de pérdida donde se contrastaron los puntajes obtenidos del modelo utilizando la configuración de la red con las capas de TimeNet y ScoreNet, frente a la configuración de la red original de GraspNet.

A partir de los resultados anteriores se decidió iterar en la experimentación para donde se finalmente se utilizaron los valores de regularización descritos en la Tabla 3.2 que permitieron definir mejores resultados en la experimentación.

**Tabla 3.2 Valores de regularización finales.**

| Hiperparámetro | Valor |
|---|---|
| $\lambda_1$ | 0.7 |
| $\lambda_2$ | 1.2 |
| $\lambda_3$ | 0.3 |
| $\alpha$ | 0.5 |
| $\beta$ | 0.5 |
| $\gamma$ | 0.5 |
| $LR$ | 0.001 |

Para los nuevos valores de regularización se realizó un análisis del puntaje de pérdida en el cual se verificaron los resultados durante el entrenamiento del modelo con la configuración de la red que incluía las capas de TimeNet y ScoreNet y la configuración original de GraspNet.

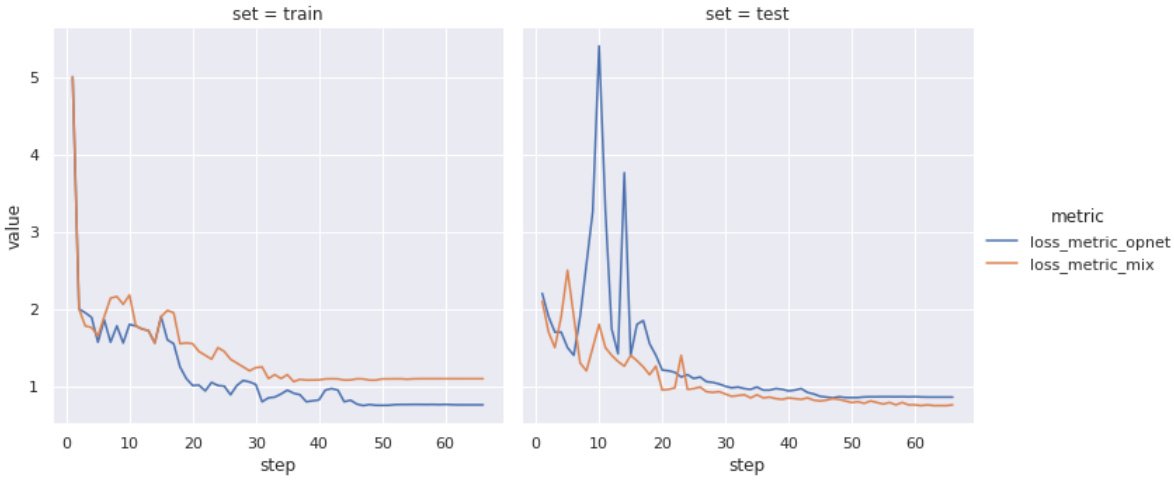


**Figura 3.11 Resultados del entrenamiento de la red**

En los resultados del entrenamiento se verificó que existía una diferencia entre ambos modelos, mientras que la configuración original de GraspNet converge su valor de pérdida en 0.76, el valor de pérdida con la información de la planificación converge en 1.09 en el caso del conjunto de datos de entrenamiento. Verificando con respecto al conjunto de validación al inicio se notaron picos en el puntaje de pérdida, y además se observó que en este caso el modelo con la configuración

original convergía en 0.86, en contraste al modelo generado por la configuración con la información de planificación que convergía en 0.73.

El análisis del puntaje de pérdida descrito en la Figura 3.11 sugiere que existe un rendimiento desfavorable con respecto al modelo entrenado con la configuración de la red que utiliza la información de planificación, dado que el puntaje de pérdida en la etapa de validación converge a un valor inferior que en la etapa de entrenamiento, lo que significa que el modelo no predice mejor los ejemplos del entrenamiento. Este fenómeno se puede deber a la cantidad de datos que se utilizan para entrenar y validar al modelo, así mismo puede corresponder a valores de regularización no optimizados.

Como siguiente análisis se realizaron pruebas con el planificador de movimientos de tal manera que se obtuvo el valor correspondiente al tiempo de planificación de cada pose de agarre según su puntaje de agarre. Para este análisis se utilizaron 4 diferentes modelos, el primero generado por la arquitectura de la red original de GraspNet, el segundo que utiliza la red de TimeNet y ScoreNet, el tercero sólo utiliza TimeNet y el cuarto sólo utiliza ScoreNet.


**Figura 3.12 Tiempo de planificación promedio y puntaje de agarre para cada pose**

En el eje de las X se visualiza el puntaje de agarre para las poses de agarre sugeridas, mientras que en el eje de las Y el tiempo de planificación para dichas poses. Se puede notar que existe una tendencia positiva con respecto al tiempo de

planificación, a mayor puntaje de agarre, mayor es el tiempo de planificación. Además, se observa que en tendencia la arquitectura que utiliza sólo tiempo de planificación tiene un mejor rendimiento en las poses de agarre con mayor puntuación de agarre, frente a las demás configuraciones, incluso a la arquitectura original de GraspNet.

Como experimentación se utilizó una escena recreada en el software de simulación Coppelia con cilindros de distintos tamaños y distintas distancias entre ellos. En la Figura 3.13 se puede visualizar las poses de agarre generadas utilizando la librería Open3D de Python.

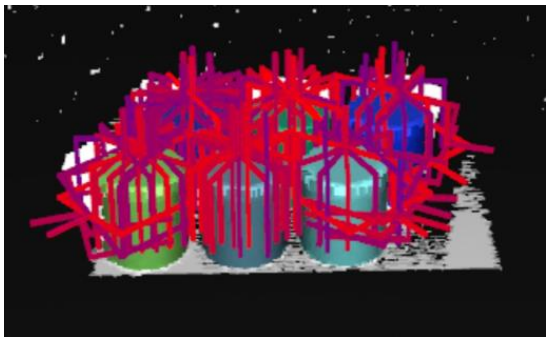

**Figura 3.13 Poses de agarre generadas por el modelo**

## 3.4 Módulo de Planificación de Movimiento

Para cada uno de los cilindros de las escenas, se planifica el movimiento hasta uno de los marcos desplazados, de tal manera que en la pose final el objetivo pueda ser agarrado. En la Figura 3.14, Figura 3.15, Figura 3.16 se pueden observar líneas rosadas, esto corresponde al camino planificado por el algoritmo RRTConnect.
Dado que el algoritmo de planificación está basado en muestreo aleatorio, no se obtienen salidas iguales para las mismas entradas en diferentes ejecuciones. Esto es lo que demuestra la Figura 3.15, en la cual para el agarre del cilindro verde se muestran diferentes caminos en el espacio cartesiano y diferentes configuraciones articulares finales, dado que la planificación está realizada en el espacio articular.

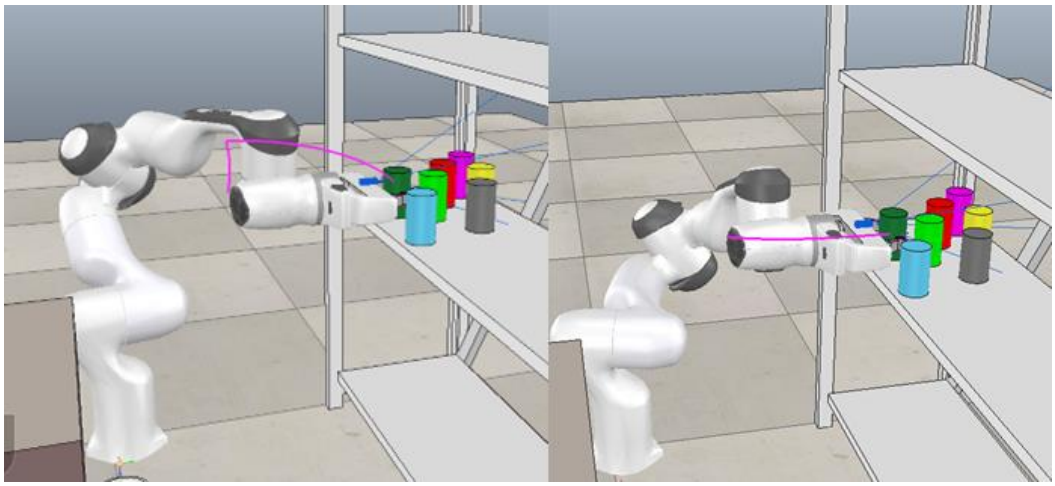

**Figura 3.14 Diferentes caminos planificados para un mismo objetivo**

Luego de agarrar los cilindros, tanto para la escena O como para D, se llevan a una mesa como se muestra en la Figura 3.15. Para la escena O, desde la inicialización de las tareas (percepción) hasta la finalización (descarga de los cilindros en la mesa), la duración de la simulación está entre 45 minutos a 1 hora; por otro lado, la simulación de la escena O dura entre 30 a 40 minutos.

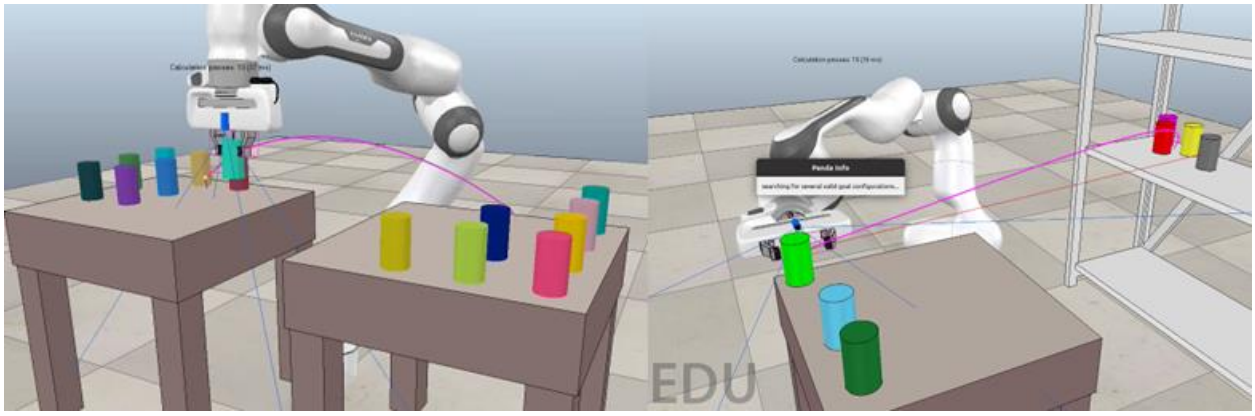

**Figura 3.15 Tarea luego del agarre**

Se realizaron 30 experimentos con escenas O y 30 con escenas D, las especificaciones de la configuración de la escena, se encuentra en el Apéndice A y Apéndice B, respectivamente. Las poses generadas por el módulo de detección de poses de agarre, luego enviadas al planificador de movimiento alcanzaron un éxito de agarre del 96.03% para escenas O y 91% para escenas D.

## 3.5 Módulo de Control

Para observar mejor el efecto del módulo de control sobre el movimiento del robot, el cliente solicitó una escena en la que el robot inicie desde una configuración inicial desviada de la configuración inicial deseada (planificada por RRTConnect).

La Figura 3.16 muestra el comportamiento del seguimiento de la trayectoria. Al inicio existe una desviación inicial (imagen izquierda), al transcurrir el tiempo, el TCP del robot se ajusta a la trayectoria (imágenes central y derecha).

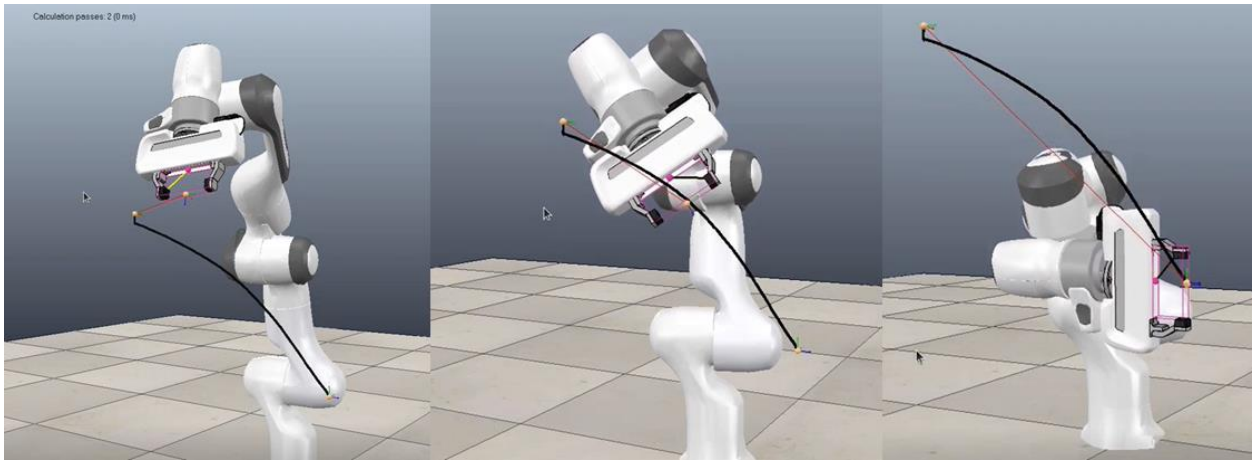

**Figura 3.16 Evolución del seguimiento de trayectoria dada una desviación inicial, controlado por CCT.**

Para poder observar mejor el comportamiento del control llevado a cabo en el espacio articular, se muestran la Figura 3.17, y Figura 3.18. Cada una de las articulaciones tienen una desviación inicial de 0.1 rad (5.729 deg), lo cual produce un efecto considerable en el espacio cartesiano, como muestra la imagen izquierda de la Figura 3.16.

En la columna izquierda de la Figura 3.18, se muestran imágenes comparativas entre las trayectorias angulares deseadas y las medidas, en esta se puede observar el comportamiento amortiguado y sin sobreimpulsos. Luego de la respuesta transitoria, tanto la señal medida como la deseada parecen traslaparse, esto debido a que los errores tienden a anularse gracias a la acción de los parámetros de control (Kv y Kp).

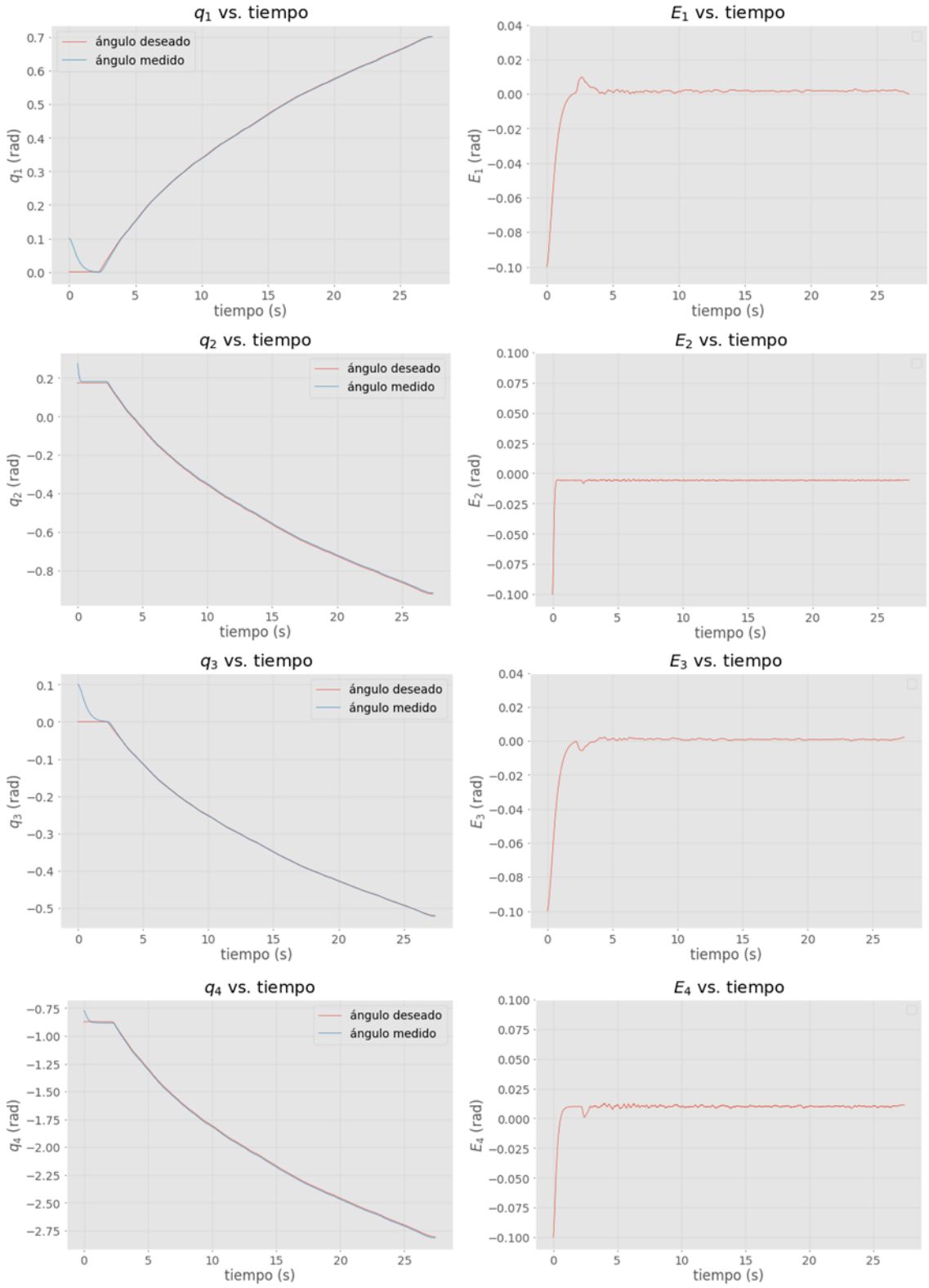


**Figura 3.17 Trayectorias medidas y deseadas de las juntas 1-4 (izquierda) con sus respectivos errores (derecha)**

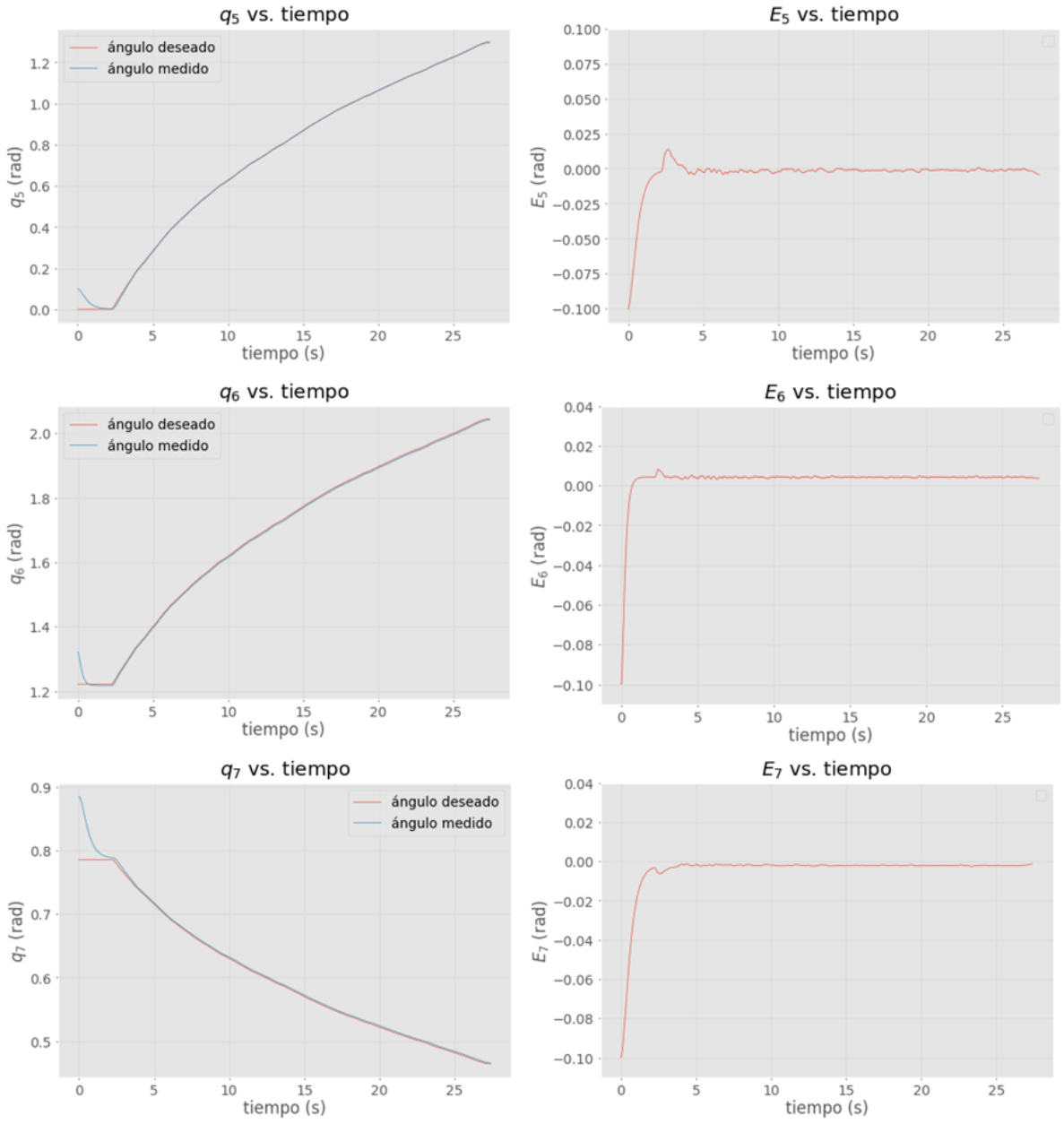


**Figura 3.18 Trayectorias medidas y deseadas de las juntas 5-7 (izquierda) con sus respectivos errores (derecha)**

A continuación, para las trayectorias de la figura Figura 3.18, se muestra un resumen de los errores máximos en tiempo estable para cada una de las articulaciones.

**Tabla 3.3 Errores máximos en tiempo estable para cada una de las juntas del manipulador.**

| Junta i | $E_{max}$ (rad) |
|---|---|
| 1 | 0.0029 |
| 2 | 0.0083 |
| 3 | 0.0022 |
| 4 | 0.0124 |
| 5 | 0.0077 |
| 6 | 0.0082 |
| 7 | 0.0044 |

Los errores mostrados en la tabla son despreciables, dado que el máximo error presentado es de 0.0124 (0.74 deg). Es posible que estos errores se den por simplificaciones en el modelo dinámicos (consideración despreciable de la fricción) y el paso de la simulación. Uno de los puntos más importantes del control es su pose final, dado que una pequeña desviación podría causar severos errores en el espacio de tareas. Para el experimento realizado se muestra que las diferencias entre la matriz de transformación homogénea (MTH) deseada como la matriz medida son insignificantes:

**Tabla 3.4 Tabla comparativa entre la MTH deseada y la medida**

| MTH deseada | MTH medida |
|---|---|
| $\begin{bmatrix} -1 & 0 & 0 & 0.34 \\ 0 & 0 & 1 & 0.2471 \\ 0 & 1 & 0 & 0.3179 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} -0.9996 & 0.0264 & -0.0087 & 0.34 \\ -0.00869 & 0.0024 & 0.9999 & 0.2471 \\ 0.02638 & 0.9996 & -0.0022 & 0.3179 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |

## 3.6 Análisis de Costos

En la sección a continuación se presenta el análisis de costos del presente proyecto que abarca desde el diseño hasta el desarrollo de la arquitectura de un sistema de agarre de objetos para el robot Franka Emika Panda.

Entre los costos de diseño y desarrollo se toman en cuenta las horas-hombre que cubren los 4 meses de desarrollo del proyecto, lo que corresponde a 80 días laborales trabajando 4 horas diarias, resultando en 320 horas-hombre invertidas en el proyecto. Se realizó la cotización de las horas-hombre según el sueldo básico del Ecuador, dando como resultado $3.75 por horas-hombre, por lo que el coste de la mano de obra del proyecto supone una inversión de $1200 por persona.

En el desarrollo del proyecto se utilizaron ambientes computacionales propios tanto para la simulación como para el entrenamiento del modelo de aprendizaje profundo. Sin embargo, esto supone un coste energético adjunto al uso de cada uno de los dispositivos.

En el caso del entrenamiento se utilizó un ambiente de cómputo según las especificaciones en la siguiente tabla:

**Tabla 3.5 Especificaciones del ambiente de cómputo de entrenamiento.**

| Componente | Especificaciones |
|---|---|
| Tarjeta Gráfica | NVIDIA GTX 1050Ti - 4GB |
| Memoria RAM | HyperX DDR4 - 16GB 3200MHz |
| Procesador | Intel Core I5-12400 – 4.40GHz |

Por lo que se cotizó un ambiente de cómputo en Amazon Web Services (AWS) comúnmente llamadas instancias con especificaciones similares. La cotización sugirió el uso servicio Amazon Sage Maker para el entrenamiento del modelo de aprendizaje Profundo. Con un total de tiempo de entrenamiento de 25 horas se tienen los valores descritos en la Tabla 3.6:

**Tabla 3.6 Especificaciones de la instancia cotizada en Amazon Sage Maker**

| Instancia | vCPU | Memoria | Precio/Hora | Total |
|---|---|---|---|---|
| ml.m5.xlarge | 4 | 16GB | $0.23 | **$5.75** |

Para el caso de simulación se cotizó una instancia de EC2 con GPU para el ambiente de pruebas. Con un promedio de tiempo de 6 horas al día durante una semana de pruebas y un total de 42 horas de uso se tienen los valores descritos en la Tabla 3.7.

**Tabla 3.7 Especificaciones de la instancia cotizada en Amazon EC2**

| Instancia | GPU | vCPU | Memoria | Precio/Hora | Total |
|---|---|---|---|---|---|
| g4dn.xlarge | 1 | 4 | 16GB | $0.526 | **$22.10** |

El precio total por el diseño y desarrollo del proyecto se describe en la Tabla 3.8.

**Tabla 3.8 Costos estimados de desarrollo y pruebas**

| Descripción | Costo |
|---|---|
| Mano de Obra | $2400 |
| Amazon Sage Maker | $5.75 |
| Amazon EC2 | $22.10 |
| **Total** | **$2427.85** |

# CAPÍTULO 4

## 4. CONCLUSIONES Y RECOMENDACIONES

### 4.1 Conclusiones

- El desarrollo del método de recolección de datos permitió la adición de la información de planificación de movimiento para cada pose de agarre enriqueciendo el conjunto de datos y mejorando la capacidad de medir su factibilidad.
- El desarrollo de la red entrenada con información proveniente del planificador de movimientos demostró la capacidad de generar un modelo que potencialmente mejora el rendimiento de detección, puesto que no mejora su confiabilidad, pero es eficiente con respecto al tiempo de planificación de las poses generadas.
- El algoritmo de planificación de movimiento (RRTConnect) junto al solucionador de cinemática inversa (método del Jacobiano pseudoinverso) permitieron proyectar rutas exactas desde la pose actual del robot redundante hacia los objetivos dados por matrices de transformación homogénea.
- Los éxitos de agarre para escenas O corresponden al 96.03% de los objetos de la escena, mientras que para escenas D, de 91%.

### 4.2 Recomendaciones

- El proyecto realizado da apertura a trabajos futuros, ya que existen otras características que se han dejado afuera como lo puede ser la topología del objeto o el grado de desorden de la escena.
- El uso de nuevos métodos de detección podría presentar una mejora en la eficiencia de la generación de poses de agarre siendo este parte crucial de la red que influye en el aprovechamiento de las características locales y globales de la escena.
- Se realizó el proyecto utilizando la simulación de un brazo robótico, en trabajos futuros se plantea la posibilidad de utilizar la información de múltiples brazos robóticos.

- Para llegar a generalizar en un mayor nivel el sistema para agarres, se recomienda el entrenamiento con diferentes tipos de objetos, además de los cilindros.
- Una de las limitaciones más significantes en el módulo de control fue el tamaño del paso temporal permitido por el simulador, el mínimo paso permitido por el simulador es de 0.01 s. En el futuro, se recomienda trabajar con simuladores que permitan un menor paso, para minimizar aún más los errores en la etapa de control.

# Bibliografía


[1] H. Alemzadeh, J. Raman, N. Leveson, Z. Kalbarczyk y R. K. Iyer, «Adverse Events in Robotic Surgery: A Retrospective Study of 14 Years of FDA Data,» *PLOS ONE,* vol. 11, pp. 1-20, 2016.

[2] A. Bicchi y V. Kumar, «Robotic grasping and contact: A review,» de *Proceedings 2000 ICRA. Millennium Conference*, San Francisco, 2000.

[3] J. Bohg, A. Morales, T. Asfour y D. Kragic, «Data-Driven Grasp Synthesis—A Survey,» *IEEE Transactions on Robotics,* vol. 30, pp. 289 - 309, 2013.

[4] R. Bormann, B. Ferreira de Brito, J. Lindermayr, M. Omainska y M. Patel, «Towards Automated Order Picking Robots for Warehouses and Retail,» de *ICVS 2019: Computer Vision Systems*, Thessaloniki, 2019.

[5] K. Kleeberger, R. Bormann, W. Kraus y M. F. Huber, «A Survey on Learning-Based Robotic Grasping.,» *Current Robotics Reports,* vol. 1, p. 239–249, 2020.

[6] G. Du, K. Wang, S. Lian y K. Zhao, «Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review,» *Artificial Intelligence Review,* vol. 54, nº 5, pp. 1677-1734, 2021.

[7] H. Duan, P. Wang, Y. Huang, G. Xu, W. Wei y X. Shen, «Robotics Dexterous Grasping: The Methods Based on Point Cloud and Deep Learning,» *Frontiers in Neurorobotics,* vol. 15, pp. 1-27, 2021.

[8] H. Zhang, J. Tang, S. Sun y X. Lan, «Robotic Grasping from Classical to Modern: A Survey,» vol. 10, pp. 1-31, 2022.

[9] I. A. Sucan, M. Moll y L. E. Kavraki, «The Open Motion Planning Library,» *IEEE Robotics & Automation Magazine,* vol. 19, pp. 72-82, 2012.

[10] B. Siciliano, L. Sciavicco, L. Villani y G. Oriolo, Robotics: Modelling, Planning and Control, London: Springer London, 2009.

[11] C. Chen, C. Zhang, T. Hu, H. Ni y W. Luo, «Model-assisted extended state observer-based computed torque control for trajectory tracking of uncertain robotic

manipulator systems,» *International Journal of Advanced Robotic Systems,* vol. 15, pp. 1-12, 2018.

[12] J. A. Shah y S. S. Rattan, «Dynamic Analysis Of Two Link Robot Manipulator For Control Design Using PID Computed Torque Control.,» *IAES International Journal of Robotics and Automation (IJRA),* vol. 5, pp. 277-283, 2016.

[13] Abdel-Nasser, P. Sharkawy y Koustoumpardis, «Dynamics and Computed-Torque Control of a 2-DOF manipulator: Mathematical Analysis,» *International Journal of Advanced Science and Technology,* vol. 28, pp. 201-212, 2019.

[14] L. Zollo, B. Siciliano, A. De Luca y E. Guglielmelli, «PD control with on-line gravity compensation for robots with flexible links,» de *2007 European Control Conference (ECC)*, Kos, 2007.

[15] R. Kelly, «PD Control with Desired Gravity Compensation,» de *Control of Robot Manipulators in Joint Space*, London, Springer London, 2005, pp. 171-199.

[16] F. J. Abu-Dakka y M. Saveriano, «Variable Impedance Control and Learning—A Review,» *Frontiers in Robotics and AI,* vol. 7, pp. 1-18, 2020.

[17] J. Ichnowski, Y. Avigal, V. Satish y K. Goldberg, «Deep learning can accelerate grasp-optimized motion planning,» *Science Robotics,* vol. 5, nº 6, pp. 1-12, 2020.

[18] M. H. Sayour, S. E. Kozhaya y S. S. Saab, «Autonomous Robotic Manipulation: Real-Time, Deep-Learning Approach for Grasping of Unknown Objects,» *Journal of Robotics,* vol. 2022, nº 7, pp. 1-14, 2022.

[19] S. Song, A. Zeng, J. Lee y T. Funkhouser, «Grasping in the wild: Learning 6DOF closed-loop grasping from low-cost demonstrations,» *IEEE ROBOTICS AND AUTOMATION LETTERS,* vol. 5, pp. 4978-4985, 2020.

[20] H.-S. Fang, C. Wang, M. Gou y C. Lu, «GraspNet-1Billion: A Large-Scale Benchmark for General Object Grasping,» de *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, 2020.

[21] Y. Wang, Y. Xu, X. Zhang, Z. Sun, Y. Zhang, G. Song y J. Wang, «3D Pose Estimation for Robotic Grasping Using Deep Convolution Neural Network,» de *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Kuala Lumpur, 2018.

[22] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley y K. Goldberg, «Learning ambidextrous robot grasping policies,» *Science Robotics,* vol. 4, pp. 1-12, 2019.

[23] C. R. Qi, L. Yi, H. Su y L. J. Guibas, «PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space,» de *Advances in neural information processing systems*, Long Beach, 2017.

[24] L. Ran, W. Wanggen, Z. Yiyuan, L. Libing y Z. Ximin, «Normal estimation algorithm for point cloud using KD-Tree,» de *IET International Conference on Smart and Sustainable City 2013 (ICSSC 2013)*, Shanghai, 2013.

[25] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein y J. M. Solomon, «Dynamic Graph CNN for Learning on Point Clouds,» de *ACM Transactions on Graphics*, New York, 2019.

[26] H. Zhao, L. Jiang, J. Jia, P. Torr y V. Koltun, «Point Transformer,» de *Computer Vision and Pattern Recognition*, 2020.

# APÉNDICES

# APÉNDICE A

## Recopilación de datos experimentales para la estimación del éxito de agarre en escenas O

### Tabla A.1 Especificaciones de los experimentos para estimar el éxito de agarre para escenas O

| Experimento | Diámetro (m) | Altura (m) | Espacio entre cilindros (m) | Número de cilindros | Número de agarres exitosos | Éxito de agarre (%) |
|---|---|---|---|---|---|---|
| 1 | 0,050 | 0,070 | 0,0300 | 16 | 15 | 93,75 |
| 2 | 0,045 | 0,080 | 0,0300 | 16 | 16 | 100,00 |
| 3 | 0,060 | 0,080 | 0,0300 | 9 | 8 | 88,89 |
| 4 | 0,030 | 0,080 | 0,0300 | 20 | 18 | 90,00 |
| 5 | 0,030 | 0,050 | 0,0150 | 49 | 47 | 95,92 |
| 6 | 0,050 | 0,050 | 0,0250 | 16 | 16 | 100,00 |
| 7 | 0,040 | 0,050 | 0,0200 | 25 | 24 | 96,00 |
| 8 | 0,035 | 0,080 | 0,0210 | 25 | 23 | 92,00 |
| 9 | 0,045 | 0,120 | 0,0360 | 9 | 9 | 100,00 |
| 10 | 0,055 | 0,100 | 0,0275 | 16 | 15 | 93,75 |
| 11 | 0,046 | 0,100 | 0,0275 | 16 | 15 | 93,75 |
| 12 | 0,042 | 0,090 | 0,0210 | 25 | 23 | 92,00 |
| 13 | 0,042 | 0,040 | 0,0210 | 25 | 25 | 100,00 |
| 14 | 0,039 | 0,040 | 0,0016 | 36 | 34 | 94,44 |
| 15 | 0,040 | 0,080 | 0,0160 | 25 | 25 | 100,00 |
| 16 | 0,060 | 0,090 | 0,0240 | 16 | 15 | 93,75 |
| 17 | 0,047 | 0,080 | 0,0235 | 16 | 16 | 100,00 |
| 18 | 0,037 | 0,085 | 0,0222 | 25 | 24 | 96,00 |
| 19 | 0,041 | 0,085 | 0,0287 | 16 | 16 | 100,00 |
| 20 | 0,051 | 0,090 | 0,0306 | 9 | 8 | 88,89 |
| 21 | 0,038 | 0,065 | 0,0228 | 25 | 24 | 96,00 |
| 22 | 0,034 | 0,065 | 0,0170 | 36 | 36 | 100,00 |
| 23 | 0,053 | 0,085 | 0,0265 | 16 | 16 | 100,00 |
| 24 | 0,054 | 0,085 | 0,0486 | 16 | 15 | 93,75 |
| 25 | 0,056 | 0,090 | 0,0504 | 9 | 8 | 88,89 |
| 26 | 0,057 | 0,080 | 0,0399 | 9 | 9 | 100,00 |
| 27 | 0,043 | 0,080 | 0,0387 | 9 | 9 | 100,00 |
| 28 | 0,043 | 0,080 | 0,0301 | 16 | 16 | 100,00 |
| 29 | 0,044 | 0,070 | 0,0176 | 25 | 24 | 96,00 |
| 30 | 0,036 | 0,060 | 0,0144 | 36 | 35 | 97,22 |
| | | | | | **Promedio** | 96,03 |

# APÉNDICE B

## Recopilación de datos experimentales para la estimación del éxito de agarre en escenas D

### Tabla B.1 Especificaciones de los experimentos para estimar el éxito de agarre para escenas D

| Experimento | Diámetro (m) | Altura (m) | Número de cilindros | Número de agarres exitosos | Éxito de agarre (%) |
|---|---|---|---|---|---|
| 1 | 0,05 | 0,070 | 6 | 5 | 83,33 |
| 2 | 0,048 | 0,080 | 6 | 6 | 100,00 |
| 3 | 0,046 | 0,080 | 6 | 6 | 100,00 |
| 4 | 0,044 | 0,080 | 6 | 4 | 66,67 |
| 5 | 0,042 | 0,050 | 6 | 6 | 100,00 |
| 6 | 0,04 | 0,050 | 6 | 6 | 100,00 |
| 7 | 0,038 | 0,050 | 6 | 6 | 100,00 |
| 8 | 0,036 | 0,080 | 6 | 5 | 83,33 |
| 9 | 0,034 | 0,120 | 6 | 6 | 100,00 |
| 10 | 0,032 | 0,100 | 6 | 4 | 66,67 |
| 11 | 0,05 | 0,100 | 5 | 5 | 100,00 |
| 12 | 0,048 | 0,090 | 5 | 5 | 100,00 |
| 13 | 0,046 | 0,040 | 5 | 5 | 100,00 |
| 14 | 0,044 | 0,040 | 5 | 4 | 80,00 |
| 15 | 0,042 | 0,080 | 5 | 4 | 80,00 |
| 16 | 0,04 | 0,090 | 5 | 4 | 80,00 |
| 17 | 0,038 | 0,080 | 5 | 5 | 100,00 |
| 18 | 0,036 | 0,085 | 5 | 5 | 100,00 |
| 19 | 0,034 | 0,085 | 5 | 3 | 60,00 |
| 20 | 0,032 | 0,090 | 5 | 4 | 80,00 |
| 21 | 0,05 | 0,065 | 4 | 4 | 100,00 |
| 22 | 0,048 | 0,065 | 4 | 4 | 100,00 |
| 23 | 0,046 | 0,085 | 4 | 4 | 100,00 |
| 24 | 0,044 | 0,085 | 4 | 4 | 100,00 |
| 25 | 0,042 | 0,090 | 4 | 3 | 75,00 |
| 26 | 0,04 | 0,080 | 4 | 3 | 75,00 |
| 27 | 0,038 | 0,080 | 4 | 4 | 100,00 |
| 28 | 0,036 | 0,080 | 4 | 4 | 100,00 |
| 29 | 0,034 | 0,070 | 4 | 4 | 100,00 |
| 30 | 0,032 | 0,060 | 4 | 4 | 100,00 |
| | | | | **Promedio** | 91,00 |

# APÉNDICE C

## Parámetros intrínsecos del sensor de visón de CoppeliaSim

**Tabla C.1 Parámetros intrínsecos y sus respectivos valores utilizados para el sensor de visón de CoppeliaSim**

| Parámetro | Valor |
|---|---|
| Ancho | 2048 px |
| Altura | 2048 px |
| Longitud focal en el eje X | -2195.975073943011 mm |
| Longitud focal en el eje Y | -2195.975073943011 mm |
| Punto principal del eje X | 1024.0 mm |
| Punto principal del eje Y | 1024.0 mm |

**Matriz intrínseca (MI)**

$$\boldsymbol{MI} = \begin{bmatrix} -2195.97 & \mathbf{0} & 1024 \\ \mathbf{0} & -2.19597 & 1024 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$