UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Redes neuronales para manipulación de objetos en robots de servicio

TESIS

Que para obtener el título de

Ingeniero Eléctrico Electrónico

P R E S E N T A

Germán Alday Salazar

DIRECTOR DE TESIS

Dr. Marco Antonio Negrete Villanueva

Ciudad Universitaria, Cd. Mx., 2025

PROTESTA UNIVERSITARIA DE INTEGRIDAD Y HONESTIDAD ACADÉMICA Y PROFESIONAL
(Titulación con trabajo escrito)

De conformidad con lo dispuesto en los artículos 87, fracción V, del Estatuto General, 68, primer párrafo, del Reglamento General de Estudios Universitarios y 26, fracción I, y 35 del Reglamento General de Exámenes, me comprometo en todo tiempo a honrar a la institución y a cumplir con los principios establecidos en el Código de Ética de la Universidad Nacional Autónoma de México, especialmente con los de integridad y honestidad académica.

De acuerdo con lo anterior, manifiesto que el trabajo escrito titulado REDES NEURONALES PARA MANIPULACION DE OBJETOS EN ROBOTS DE SERVICIO que presenté para obtener el titulo de INGENIERO ELÉCTRICO ELECTRÓNICO es original, de mi autoría y lo realicé con el rigor metodológico exigido por mi Entidad Académica, citando las fuentes de ideas, textos, imágenes, gráficos u otro tipo de obras empleadas para su desarrollo.

En consecuencia, acepto que la falta de cumplimiento de las disposiciones reglamentarias y normativas de la Universidad, en particular las ya referidas en el Código de Ética, llevará a la nulidad de los actos de carácter académico administrativo del proceso de titulación.

GERMAN ALDAY SALAZAR
Número de cuenta: 420053198

Dedicado a mi familia, a mi madre Martha, a mi padre Germán, a mi hermana Salma y a mi abuela María de Jesús QEPD por apoyarme durante todo este tiempo y ser pilares fundamentales de mi vida ...

Agradecimientos

- Quiero agradecer personalmente a todas las personas que me han apoyado en mi camino para llegar hasta este punto en mi vida.
- Quiero agradecer a todos los amigos que hice durante mi desarrollo profesional. En particular a mis amigos Enrique, Rair y Kalid por apoyarme tanto profesional como emocionalmente. También quiero agradecer a mis amigos Fernando, Daniela, Denisse, David, Oscar y Carlos por compartir momentos tan bonitos de su vida conmigo.
- Quiero agradecer a todos los docentes que han forjado mi desarrollo académico y profesional.
- Quiero agradecer al Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT) por su apoyo económico durante la elaboración de este trabajo.
- Y finalmente quiero agradecer a los gigantes de Euler, Gauss, Fischer y Riemann cuyas contribuciones al mundo de las matemáticas son inconmensurables y que sin ellos este trabajo no sería posible.

Índice general

Índice de tablas

Índice de figuras

Nomenclatura

Álgebra vectorial

$\cdot; \langle , \rangle$ Producto punto

Δ Gradiente

$\mathbf{X}$ Matriz

$\vec{x}, \mathbf{x}$ Vector

Espacios topológicos

$\mathbb{E}$ Espacio euclidiano

$\mathbb{H}$ Grupo de los cuaterniones

$\mathbb{H}_1$ Grupo de los cuaterniones unitarios

$\mathbf{SO}(n)$ Grupo ortogonal especial de grado n

$\mathcal{S}^n$ Espacio de las hiperesferas de dimensión n

$SE(n)$ Grupo del movimiento rígido en n

Geometría Riemanniana

γ Geodésico

$\mathcal{M}$ Variedad

$\mathcal{T}$ Espacio tangente

$\mathcal{T}_p$ Espacio tangente sobre el punto p

g Métrica Riemanniana

Estadística

$\mathcal{K}$ Distribución de Von Mises-Fisher

$\mathcal{N}$ Distribución normal

$\mathcal{U}$	Distribución uniforme
μ	Media de una variable aleatoria
Σ	Matriz de covarianza
$P(X)$	Densidad de probabilidad de una variable aleatoria X

Acrónimos

RNC Redes neuronales convolucionales

ADAMW Estimación adaptativa de momento con regularización de pesos desacoplado. Del inglés Adaptive Moment Estimation with decoupled Weight decay regularization

VMF Von Mises-Fisher

EMV Estimación por máxima verosimilitud

NLL Negativo del logaritmo de la función de verosimilitud. Del inglés, Negative log likelihood

EMC Error medio cuadrático

EMGC Error geodésico medio cuadrático

PCA Análisis de componentes principales, por sus siglas en inglés Principal Component Analysis

PG Proceso Gaussiano

OB Optimización Bayesiana

OS-TOG Planeación de agarres orientado a tareas de una toma. Del inglés, One-Shot Task-Oriented Grasping

NVTOP Neat Videocard TOP

ROS Robot Operating System

Capítulo 1

Introducción

Debido al rápido crecimiento que ha experimentado el área de la inteligencia artificial en los últimos años, se ha popularizado el uso de las redes neuronales para la resolución de diversos problemas de ingeniería y computación. El creciente poder computacional de las GPUs modernas y la optimización en el desarrollo de algoritmos, software y herramientas como lo son PyTorch y Tensorflow en el diseño y entrenamiento de redes neuronales, ha facilitado el desarrollo e implementación de estos sistemas en la industria, la investigación y la educación.

Es por esto que se propone utilizar estos sistemas para resolver el problema de encontrar la configuración de agarre óptima para la manipulación de objetos con robots de servicio doméstico. Un problema de la robótica que involucra encontrar los puntos en el espacio de configuración para manipular objetos dentro de su alcance. Desde identificar objetos, realizar propuestas de agarre, resolver la cinemática inversa y construir trayectorias de movimiento.

Existen diversos métodos analíticos que buscan resolver este problema, sin embargo estos tienden a ser laboriosos y de alta complejidad. Entonces, se busca explorar una alternativa en las redes neuronales para encontrar esta configuración óptima de agarre. Analizando y comparando su desempeño y competitividad con los métodos ya existentes. En particular con el sistema actual que utiliza el robot de servicio del laboratorio. Esto mediante un sistema basado en arquitecturas de redes neuronales.

1.1. Motivación

Se busca realizar un trabajo exploratorio sobre el uso de las redes neuronales para la planeación de configuración de agarre en robots de servicio para analizar su viabilidad, y, de ser una propuesta lo suficientemente robusta, poder implementar estos modelos en robots reales.

1.2. Planteamiento del problema

La manipulación de objetos es una habilidad clave en los robots de servicio doméstico que se ha abordado mediante métodos analíticos y estadísticos, y más recientemente, con técnicas de aprendizaje automático. En este trabajo, se aborda el problema de proponer una pose adecuada del efector final para sujetar un objeto dada su nube de puntos parcial; es decir, se asume que no existe una vista completa del objeto.

1.3. Hipótesis

Se plantean las siguientes hipótesis sobre las que se desarrolló el trabajo de investigación:

- El uso de redes neuronales es viable para diseñar sistemas de planeación de agarres para robots de servicio.
- La optimización de los recursos de entrenamiento así como una buena selección de hiperparámetros acelerará el proceso de desarrollo y ayudará a entrenar modelos con mejor desempeño.

1.4. Objetivos

Objetivo general

Diseñar y entrenar una red neuronal para optimizar la configuración de agarre de un robot de servicio para la manipulación de objetos.

Objetivos específicos

- Diseñar una arquitectura de red neuronal que genere propuestas de configuración de agarre para robots de servicio.
- Desarrollar un ambiente simulado para generar datos de entrenamiento y realizar las pruebas de desempeño.
- Construir un dataset para entrenar la red con las especificaciones del robot de servicio del laboratorio.
- Entrenar la red utilizando el dataset construido y otros externos.
- Comparar el desempeño de la red con el sistema actual.

1.5. Descripción del documento

En el siguiente capítulo se da una breve introducción a la teoría que sustenta el trabajo de investigación, así como una exploración de la literatura y trabajos relacionados.

En el capítulo 3 se describe con más detalle la naturaleza del problema, el alcance de la investigación y la definición de las variables de estudio. En este mismo capítulo se detalla el procedimiento para el diseño e implementación de la arquitectura propuesta. Así como los obstáculos y decisiones que se tomaron durante este proceso.

En el capítulo 4 se define el entorno de pruebas y experimentos realizados para evaluar el desempeño de los sistemas implementados en el planteamiento de la arquitectura y los procedimientos para su entrenamiento, y se comparan los resultados con el sistema implementado actualmente.

Finalmente, en el capítulo 5 se realiza una discusión sobre los hallazgos del trabajo de investigación, el trabajo futuro, y se reportan las declaraciones finales de las hipótesis y los objetivos esperados definidos en este capítulo.

Capítulo 2

Marco teórico

En este capítulo se detalla una breve introducción de la teoría detrás del desarrollo de este trabajo de investigación. Se describen los fundamentos de la cinemática de robots manipuladores así como la representación de este en el espacio. También se explora un panorama general de las redes neuronales y las redes neuronales convolucionales. Se da una breve exposición a la geometría diferencial y estadística diferencial usado para los modelos estadísticos de la red propuesta en el capítulo 3.

Finalmente se presenta un breve resumen de la literatura y estado del arte respecto al problema de planeación de agarres con robots así como modelos probabilísticos de sistemas cuyas variable de respuesta yacen en espacios topológicos no euclidianos.

2.1. Cinemática de robots manipuladores

Un robot está construido como la unión entre componentes llamados cadenas mediante articulaciones, las cuales se mueven gracias al uso de actuadores eléctricos o mecánicos al suministrar fuerzas o torque.

Dado que las cadenas del robot se construyen de materiales rígidos que mantienen su forma, y la cual es conocida desde el inicio, entonces estas se pueden modelar mediante mecánica de cuerpo rígido, y para lo cual solo se necesitan unas cuantas variables para representar su configuración.

2.1.1. Movimiento de cuerpo rígido

Para poder describir el movimiento de los robots manipuladores, se utiliza el modelo del movimiento de cuerpo rígido. El cual describe el movimiento de cuerpos en los que no existe deformación. Es decir, que todas las distancias entre las partículas que conforman los cuerpos se mantienen iguales sin importar las fuerzas que actúen sobre este.

En realidad, aunque los robots de servicio se construyen con materiales resistentes estos también pueden deformarse si se les aplica la suficiente fuerza. Sin embargo, estos robots están construidos con actuadores que generalmente no tienen la potencia necesaria para deformar sus propias cadenas al moverse y por eso es posible utilizar este modelo.

2.1.2. Espacio de configuración

Lynch y Park (2017) lo definen como:

> La configuración de un robot es la especificación completa de la posición de cada punto del robot. El número mínimo de coordenadas n necesarias para representar dicha configuración es el número de grados de libertad del robot. El espacio n-dimensional que contiene todas las posibles configuraciones del robot es conocido como el **espacio de configuración (espacio-C)**. La configuración de un robot es representado como un punto en su espacio-C

Representación del espacio de configuración

Para poder representar las configuraciones del robot en el espacio, se requiere un marco de referencia base construido a partir de una representación numérica de dicho espacio. Existen muchos sistemas de coordenadas que permiten representar el espacio. Por ejemplo, para un espacio con topología euclidiana $\mathbb{E}^2$, la posición de un punto se puede describir con un sistema de coordenadas cartesianas (x, y) o coordenadas polares (r, θ).

2.2. Representación de la pose y movimiento

En cinemática de cuerpo rígido se le conoce como pose a la posición y orientación de un cuerpo en el espacio. En cinemática de robots se describe la pose, la velocidad, aceleración y otras derivadas superiores de la pose de los cuerpos que componen un sistema o mecanismo. Para el caso de la robótica, de las cadenas, articulaciones y efector final.

2.2.1. Posición

La posición de un objeto es la descripción de su ubicación en el espacio relativo a un sistema de referencia, representado mediante un marco de coordenadas.

En el espacio euclidiano $\mathbb{E}^3$, un marco de coordenadas cartesiano i consiste en un origen denominado O_i y tres vectores base mutuamente ortogonales $\hat{x_i}, \hat{y_i}, \hat{z_i}$ los cuales están fijos a un cuerpo. La pose de un cuerpo siempre se expresa relativo a otro cuerpo o marco de referencia. De esta forma, cualquier

desplazamiento se puede expresar como el desplazamiento entre dos marcos de coordenadas. En el que un marco se mueve y otro se mantiene fijo [1].

La posición de origen de un marco de coordenadas i relativo a un marco de coordenadas j se denota como un vector de $p \in \mathbb{R}^3$

$$ {}^{j}\mathbf{p}_i = \begin{pmatrix} {}^{j}p_i^x \\ {}^{j}p_i^y \\ {}^{j}p_i^z \end{pmatrix} $$

Los componentes de este vector son las coordenadas cartesianas del vector de origen O_i en el marco j, el cual es la proyección del vector ${}^{j}\mathbf{p}_i$ sobre sus ejes (base ortonormal).

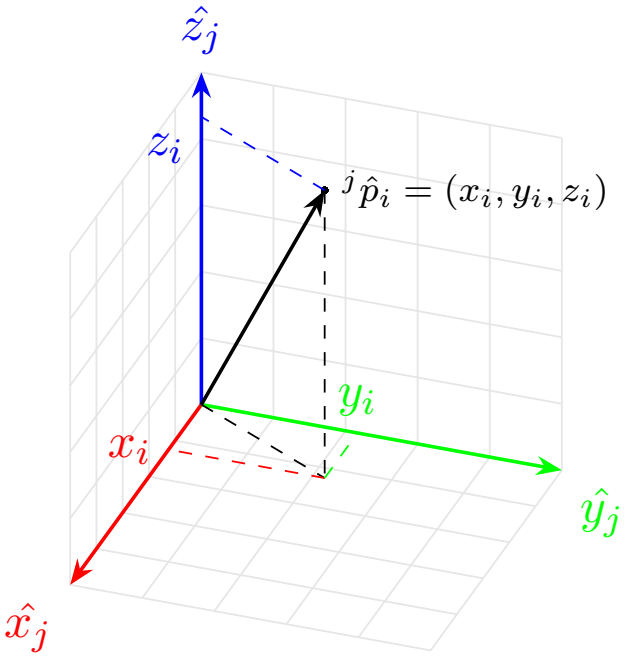


Figura 2.1: Representación de la posición en $\mathbb{E}^3$ con coordenadas cartesianas

2.2.2. Orientación

Existen varias formas de describir la orientación de un cuerpo, la cual denota la rotación de un cuerpo relativo a un eje particular. En cinemática de robots se utiliza frecuentemente los ángulos de Euler, cuaterniones y matrices de rotación.

Matrices de rotación

La orientación de un marco de coordenadas i relativo a un marco de coordenadas j puede describirse al expresar los vectores base $\hat{x}_i, \hat{y}_i, \hat{z}_i$ en

términos de los vectores base $\hat{x_j}, \hat{y_j}, \hat{z_j}$. Esto da como resultado una matriz ${}^{j}\mathbf{R}_i$ de $= 3 \times 3$ cuyos componentes son la proyección de los vectores base de un marco de coordenadas al otro. A esta matriz se le conoce como matriz de rotación

$$ {}^{j}\mathbf{R}_i = \begin{pmatrix} \hat{x_i} \cdot \hat{x_j} & \hat{y_i} \cdot \hat{x_j} & \hat{z_i} \cdot \hat{x_j} \\ \hat{x_i} \cdot \hat{y_j} & \hat{y_i} \cdot \hat{y_j} & \hat{z_i} \cdot \hat{y_j} \\ \hat{x_i} \cdot \hat{z_j} & \hat{y_i} \cdot \hat{z_j} & \hat{z_i} \cdot \hat{z_j} \end{pmatrix} $$

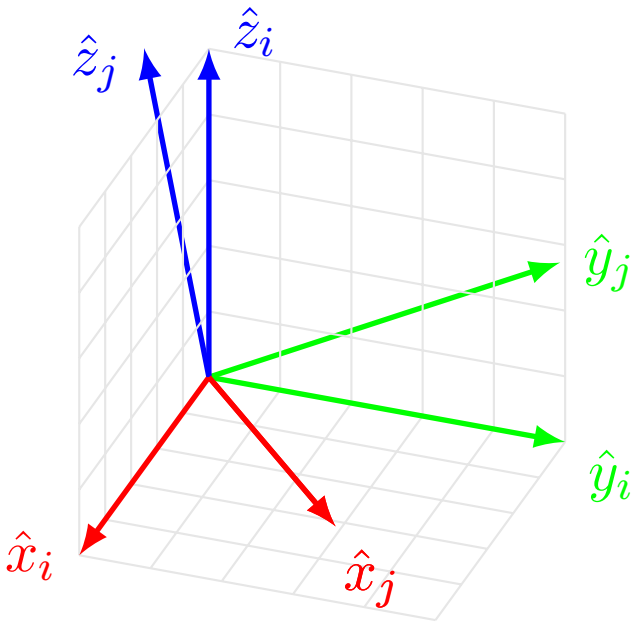


Figura 2.2: Representación de la orientación en $\mathbb{E}^3$ como la proyección de un marco de referencia a otro

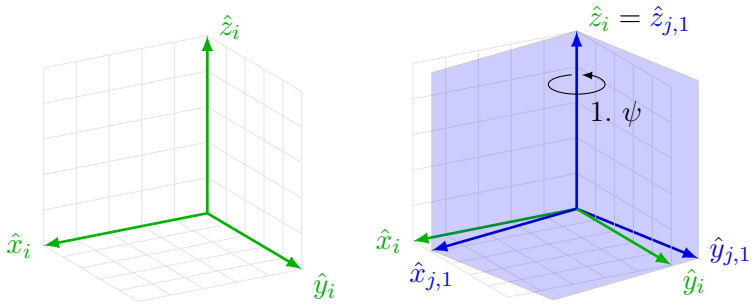
Las matrices de rotación son la representación natural de las rotaciones en 3 dimensiones y forman el grupo de rotación en $\mathbf{SO}(3) \subset \mathbb{R}^{3\times3}$ bajo el operador de composición $\circ$. Las matrices de rotación son utilizadas para describir cualquier orientación en $\mathbb{R}^3$ y también en las transformaciones homogéneas para describir el movimiento en $SE(3)$.

Ángulos de Euler

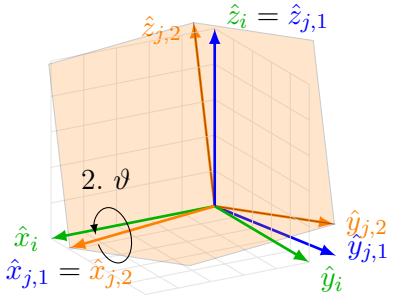
Los ángulos de Euler (*roll, pitch, yaw*) son ampliamente utilizados en la robótica y otros campos de la física para representar las rotaciones en 3 dimensiones. Se utilizan debido a que son una representación muy intuitiva y con la menor dimensionalidad entre las tres representaciones.

Para representar orientaciones con los ángulos de Euler, esto se realiza como una composición de rotaciones alrededor de ejes que son las bases ortonormales del espacio. Es decir, una rotación roll ϑ es una rotación sobre el

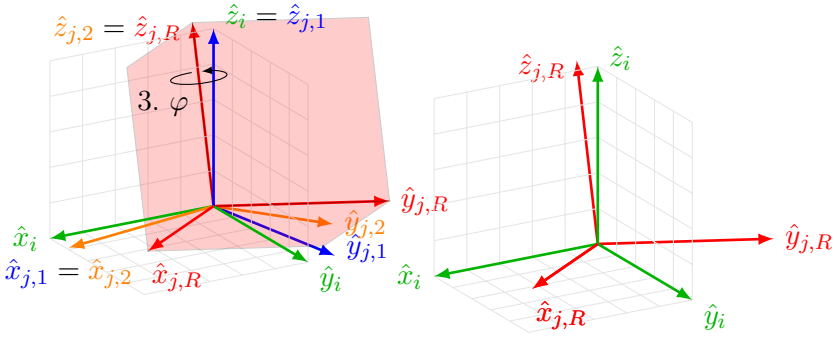
eje $\hat{x}$, pitch φ es una rotación sobre el eje $\hat{y}$ y una rotación yaw ψ es una rotación sobre el eje $\hat{z}$. Entonces, la orientación final es la composición de estas tres rotaciones.



(a) Marco de referencia inicial (b) Rotación en *yaw*



(c) Rotación en *roll*



(d) Rotación en *pitch* (e) Rotación final

Figura 2.3: Representación de rotaciones con ángulos de Euler

Sin embargo, esta representación cuenta con algunos problemas, principalmente el fenómeno de bloqueo del cardán. En el cual, se puede perder un grado de libertad cuando los ejes de rotación se alinean. Además, al ser una composición no conmutativa, no existen soluciones únicas para describir la misma orientación, pues se puede llegar a la misma rotación usando diferentes valores de ángulos con una secuencia de rotaciones diferentes.

Cuaterniones

Los cuaterniones a veces llamados versores, son utilizados ampliamente en los campos de la robótica y la computación debido a su capacidad de representar las rotaciones en tres dimensiones con una baja dimensionalidad.

Los cuaterniones forman un grupo $\mathbb{H}$ en el espacio de los vectores unitarios en $S^3 \subset \mathbb{R}^4$ bajo el operador de la multiplicación cuaterniónica. [2]

Un cuaternión está definido como $q = \omega + x\hat{i} + y\hat{j} + z\hat{k}$ donde $\hat{i}, \hat{j}, \hat{k}$ son unidades imaginarias ortogonales entre si y ortogonales al plano real. ω, x, y, z son coeficientes reales tal que su magnitud $\omega^2 + x^2 + y^2 + z^2 = 1$.

Este grupo surge como una extensión de la propiedad de la multiplicación de números complejos para modelar rotaciones en $\mathbb{R}^2$ y en este caso presentan la capacidad de representar rotaciones en $\mathbb{R}^3$. Al igual que la multiplicación de números complejos, la multiplicación cuaterniónica esta definida como una multiplicación distributiva de sus elementos tomando en consideración la restricción $i^2 = j^2 = k^2 = ijk = -1$. A continuación se muestra una tabla de Caley que describe la operación del grupo:

Tabla 2.1: Tabla de Caley para la multiplicación cuaterniónica

$\times$	**1**	**i**	**j**	**k**
1	1	i	j	k
i	i	-1	k	$-j$
j	j	$-k$	-1	i
k	k	j	$-i$	-1

2.2.3. Desplazamiento y Rotación

Para modelar el movimiento de un cuerpo rígido, este se representa como una combinación de traslaciones y rotaciones entre marcos de coordenadas.

Una traslación es un desplazamiento en el cual ningún punto del cuerpo rígido permanece en su posición original y en el que no ocurre ningún cambio en la orientación del objeto. Es decir, las líneas rectas del cuerpo permanecen paralelas a su posición original.

Una rotación es un movimiento en el cual al menos un punto del cuerpo rígido permanece en su posición original y no todas las líneas del cuerpo permanecen paralelas a su estado original.

2.2.4. Transformaciones homogéneas

Una vez teniendo las representaciones de la posición y la orientación de un cuerpo rígido en el espacio se puede describir la pose del cuerpo como

una única matriz T que encapsule ambos.

$$ {}^{j}\mathbf{T}_i = \begin{pmatrix} {}^{j}\mathbf{R}_i & \hat{p} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \hat{x_i} \cdot \hat{x_j} & \hat{y_i} \cdot \hat{x_j} & \hat{z_i} \cdot \hat{x_j} & p_x \\ \hat{x_i} \cdot \hat{y_j} & \hat{y_i} \cdot \hat{y_j} & \hat{z_i} \cdot \hat{y_j} & p_y \\ \hat{x_i} \cdot \hat{z_j} & \hat{y_i} \cdot \hat{z_j} & \hat{z_i} \cdot \hat{z_j} & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Donde ${}^{j}\mathbf{T}_i$ representa la matriz de transformación homogénea de un marco de coordenadas i que tiene una orientación ${}^{j}\mathbf{R}_i$ y una posición $\hat{p}$ respecto a un marco de referencia j.

2.3. Cinemática directa e inversa

En robótica, la cinemática directa se refiere al cálculo de la posición y orientación relativa de cualquier componente de un robot a partir de la configuración de ángulos de sus articulaciones y la relación geométrica de sus cadenas.

Un método para derivar la cinemática directa del efector final de un robot cuyas cadenas se encuentran en serie, es mediante una concatenación de transformaciones homogéneas que se realizan al asignar un marco de referencia a cada cadena del sistema. Es decir, para obtener la posición y orientación de un componente n relativo a un marco de referencia 0, este se puede resolver mediante la transformación:

$$ {}^{0}\mathbf{T}_n = {}^{0}\mathbf{T}_1 {}^{1}\mathbf{T}_2 {}^{2}\mathbf{T}_3 ... {}^{n-1}\mathbf{T}_n $$

2.3.1. Cinemática inversa

Similarmente a la cinemática directa, el objetivo de la cinemática inversa es encontrar el valor de los ángulos de las articulaciones a partir de la posición y orientación relativa de sus componentes y de la relación geométrica entre sus cadenas.

El problema de encontrar la solución de la cinemática inversa para sistemas de cadenas en serie es que se requiere resolver un conjunto de ecuaciones no lineales. Y de las cuales es posible que no exista una solución o una única solución. Para que una solución exista, la pose deseada debe encontrarse dentro del límite del espacio de trabajo. Además, en estos casos donde la solución si puede existir, frecuentemente no se pueden presentar como una solución cerrada y por lo tanto se utilizan métodos numéricos para resolverla, tales como Newton-Raphson o la regla del trapecio.

2.3.2. Método de Newton-Raphson

El método de Newton-Raphson es un método numérico para resolver ecuaciones diferenciables mediante aproximaciones iterativas de primer orden.

Para resolver una ecuación $f(x) : \mathbf{R}$ por este método, se asume x^0 como una estimación inicial. Entonces se describe la función mediante una expansión de Taylor y se trunca al primer orden.

$$f(x) = f(x^0) + \frac{\partial f}{\partial x}(x^0)(x - x^0)$$

Fijando $f(x) = 0$ y resolviendo para x se obtiene:

$$x = x^0 - \left(\frac{\partial f}{\partial x} x^0\right)^{-1} f(x)$$

Usando este valor como una nueva estimación para la solución y repitiendo los pasos anteriores obtenemos la siguiente iteración:

$$x^{k+1} = x^k - \left(\frac{\partial f}{\partial x} x^k\right)^{-1} f(x^k)$$

La iteración se repite hasta que se satisfaga algún criterio de completitud. Una condición es evaluar el error entre el valor actual y el valor anterior hasta que este sea menor a una tolerancia definida.

En la manipulación de robots se utiliza el método de Newton-Raphson en versión vectorial para resolver el problema de la cinemática inversa, en el cual se busca resolver la ecuación:

$$f(\vec{x}) = FK(\vec{q}) - \vec{p} = 0$$

Donde FK es una función vectorial de variable vectorial que representa la cinemática directa de la configuración articular q de n grados de libertad, y p representa la pose de la posición deseada. Por lo tanto se tiene que:

$$x_{\vec{k+1}} = x \vec{-} k - \left(\frac{\partial f}{\partial x} x^k\right)^{-1} f(x^k)$$

2.4. Redes neuronales artificiales

Las redes neuronales artificiales han sido inspiradas parcialmente como una biomimética de los sistemas cognitivos biológicos. Los cuales están construidos por complejas redes de neuronas interconectadas.

De manera análoga, las redes neuronales artificiales están formadas como conjuntos interconectados de unidades simples, en el que cada unidad toma un valor real discreto como entrada y produce un valor real discreto de salida, el cual puede ser las entrada de otras unidades [3].

El objetivo de estos sistemas es imitar la alta paralelización de los procesos biológicos que permiten realizar computación distribuida de una manera muy eficiente.

2.4.1. Perceptrón

El perceptrón es una unidad de red neuronal que toma como entrada un vector de valores reales $\hat{x}$ y sobre los cuales realiza una operación lineal de sumas ponderadas con un vector de pesos $\hat{\omega}$ .A su salida devuelve un 1 si la suma es mayor a un umbral, y -1 si la suma es menor al umbral. Matemáticamente se puede describir de la siguiente forma:

$$o(x_1, x_2, ..., x_n) = \begin{cases} 1 & \text{si } \omega_0 + \omega_1 x_1 + ... + \omega_n x_n > 0 \\ -1 & \text{si } \omega_0 + \omega_1 x_1 + ... + \omega_n x_n \leq 0 \end{cases}$$

Donde cada ω_i es una constante que determina el peso o contribución a la suma para cada entrada correspondiente x_i

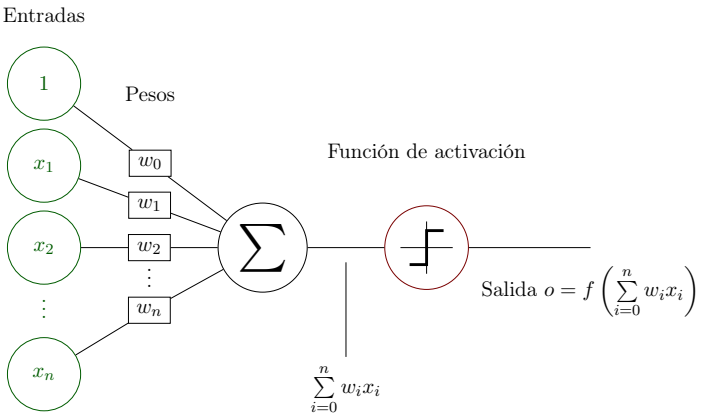


Figura 2.4: Diagrama del perceptrón

2.4.2. Redes neuronales profundas

Las redes neuronales profundas consisten en la concatenación de varias capas de red neuronal. Estas capas pueden ser de diferentes tipos, por ejemplo capas convolucionales, o capas completamente conectadas. Las cuales están formadas de múltiples neuronas interconectadas.

Esta concatenación secuencial de capas permite a los modelos aplicar transformaciones lineales y no lineales en espacios de más alta dimensionalidad. Esta expansión del espacio de hipótesis permite a las redes aprender y modelar patrones de datos muy complejos.

A continuación se muestra un ejemplo de una red completamente conectada o densa.

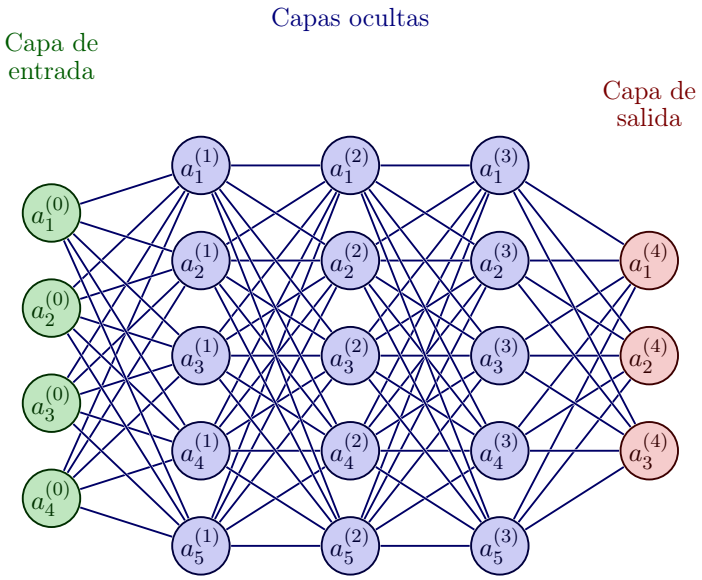
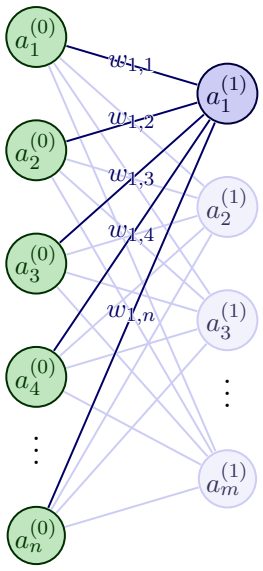


Figura 2.5: Diagrama de una red profunda de 5 capas con 3 capas ocultas

Las capas de una red completamente conectada actúan como una transformación lineal mediante el producto del vector de entrada, el cúal es la salida de una capa anterior $\mathbf{a^{n-1}}$ por la matriz de pesos $\mathbf{W}^{n-1}$ al que se le suma el vector umbral o sesgo $\mathbf{b}^{n-1}$. Para introducir no linealidades al sistema y poder aprender patrones más complejos, se le aplica una transformación no lineal en forma de una función de activación $\sigma(\mathbf{x})$. En la figura 2.6 se muestra un ejemplo de una capa de red densa.



$$= \sigma\left(w_{1,1}a_1^{(0)} + w_{1,2}a_2^{(0)} + \ldots + w_{1,n}a_n^{(0)} + b_1^{(0)}\right)$$
$$= \sigma\left(\sum_{i=1}^{n} w_{1,i}a_i^{(0)} + b_1^{(0)}\right)$$

$$\begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_m^{(1)} \end{pmatrix} = \sigma\left[\begin{pmatrix} w_{1,1} & w_{1,2} & \ldots & w_{1,n} \\ w_{2,1} & w_{2,2} & \ldots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \ldots & w_{m,n} \end{pmatrix}\begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_n^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_m^{(0)} \end{pmatrix}\right]$$

$$\mathbf{a}^{(1)} = \sigma\left(\mathbf{W}^{(0)}\mathbf{a}^{(0)} + \mathbf{b}^{(0)}\right)$$

Figura 2.6: Representación de una capa de una red neuronal profunda

2.4.3. Funciones de activación

En la aplicación de redes neuronales artificiales, las funciones de activación son funciones que transforman la suma ponderada de una señal de entrada de una neurona y su vector de pesos a una señal de salida.

Estas funciones de activación son utilizadas ya que sin ellas, las redes neuronales actuarían como modelos de regresión lineal. Esto debido a que las salidas de sus neuronas serían funciones lineales (polinomios de primer grado). Y no tendrían la capacidad de aprender patrones más complejos para procesar datos que presenten características no lineales como lo son las imágenes, el audio, el video o el lenguaje.

Existen diferentes funciones de activación, pero comúnmente se utilizan aquellas que son no lineales, continuas y diferenciables. Pues permiten utilizar el método de retropropagación y descenso del gradiente para su entrenamiento.

Las funciones de activación más utilizadas en redes neuronales son la función sigmoide, la función tangente hiperbólica, y variaciones de la función ReLU (unidad lineal rectificada). Gráficas de estas funciones se muestran en las figuras 2.7, 2.8, 2.9 y 2.10

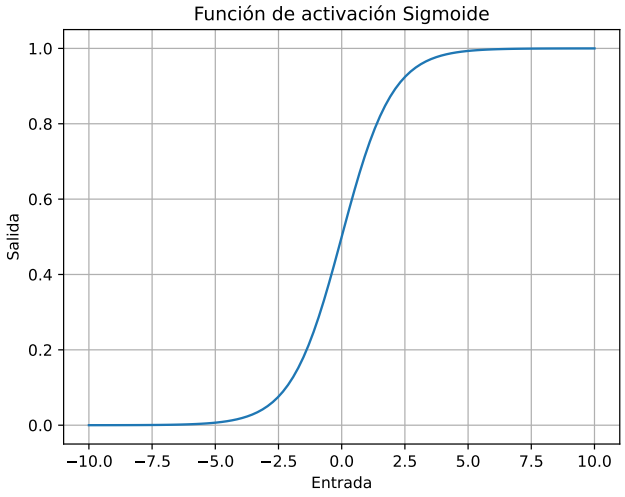


Figura 2.7: Gráfica de la función sigmoide

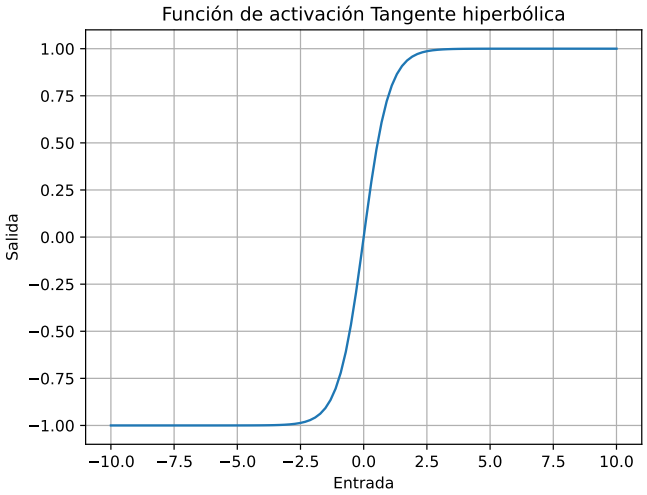
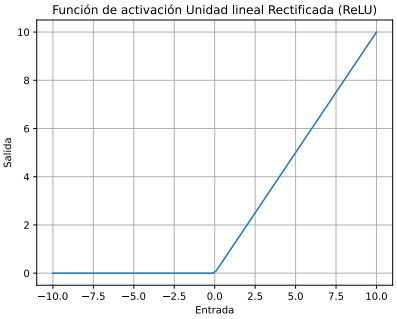
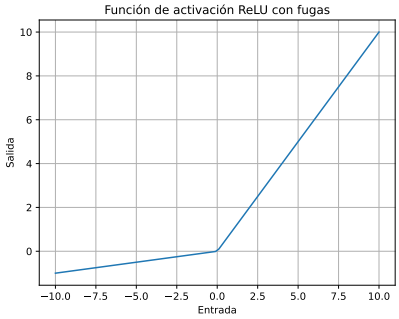


Figura 2.8: Gráfica de la función tangente hiperbólica



(a) Gráfica de la función unidad lineal rectificada



(b) Gráfica de la función unidad lineal rectificada con fugas

Figura 2.9: Variantes de la función de activación ReLU

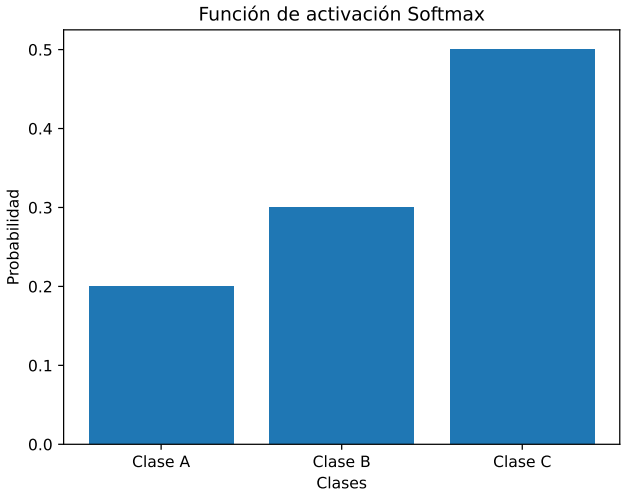


Figura 2.10: Función softmax

2.4.4. Entrenamiento de las redes

Para encontrar los valores de los pesos que permitan a las neuronas tomar decisiones o modelar funciones, se requiere de un algoritmo de aprendizaje. Existen muchos algoritmos de aprendizaje que pueden resolver el problema de entrenamiento de esta unidad como lo es la regla del perceptrón o descenso del gradiente.

La regla del perceptrón modifica los pesos ω_i asociados a una entrada x_i de la siguiente manera:

$$\omega_i \longleftarrow \omega_i + \Delta\omega_i$$
$$\Delta\omega_i = \iota(t - o)x_i$$

Donde t es la salida deseada para la entrada actual, o es la salida de la neurona y ι es una constante positiva que modifica la magnitud del cambio en los pesos entre cada paso. A esta constante se le conoce como tasa de aprendizaje.

Descenso del gradiente

La regla del perceptrón puede encontrar un vector de pesos ω cuando los ejemplos con los que se entrena son linealmente separables. Sin embargo este tiene sus limitaciones para aquellos que no lo son, ya que puede no converger a una solución. Es aquí cuando se utiliza otro algoritmo de entrenamiento, el descenso del gradiente.

Para utilizar el descenso del gradiente, primero se busca una función que mida el error entre la salida y el objetivo, y que sea continuamente diferenciable para poder utilizar este método. En este caso se utiliza la función del error cuadrático medio:

$$E(\vec{\omega}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

Donde t_d es la salida objetivo para el ejemplo de entrenamiento d en el conjunto de ejemplos de entrenamiento $\mathcal{D}$ y o_d es la salida del perceptrón para el ejemplo d. Al evaluar esta función en el espacio de hipótesis H para los pesos $\omega_0, \omega_1, ...\omega_n$ se obtiene una superficie en el espacio n-dimesional.

Entonces lo que se busca es encontrar el vector de pesos $\vec{\omega_i}$ en el espacio H que minimice el error en la función E empezando con una estimación inicial y modificándolo en pequeños incrementos hasta llegar a ese punto. Esto se realiza al ajustar el vector $\vec{\omega_i}$ levemente en dirección opuesta de la que produce el mayor cambio en la función E y por lo tanto acercándose a algún punto mínimo (aunque no necesariamente al mínimo global).

La forma de calcular esta dirección de máximo cambio es mediante el cálculo del gradiente de E respecto a $\vec{\omega}$.

$$\nabla E(\vec{\omega}) = \left[\frac{\partial E}{\partial \omega_0}, \frac{\partial E}{\partial \omega_1}, ...\frac{\partial E}{\partial \omega_n}\right]$$

A partir de esto se puede derivar la regla delta para ajustar el vector de pesos.

$$\omega_i \longleftarrow \omega_i + \Delta\omega_i$$
$$\Delta\omega_i = -\iota\nabla E(\vec{\omega_i})$$

Descenso del gradiente estocástico

Al igual que el descenso del gradiente, este es un método iterativo para modificar los parámetros de un modelo mediante el cálculo del gradiente de una función de costos respecto a los parámetros. La principal diferencia entre este método y el tradicional, es que no se utiliza el conjunto completo de ejemplos para calcular el gradiente y hacer una actualización de los parámetros. En este caso, se calcula el gradiente a partir de un pequeño subconjunto de ejemplos $\mathcal{B} \subset \mathcal{D}$ llamado lote para realizar la actualización.

Este método de optimización es muy utilizado debido a que al realizar actualizaciones de los pesos más frecuentemente, se puede alcanzar un punto óptimo más rápidamente. Y es especialmente útil cuando se trabaja con datasets muy grandes, dado que calcular el gradiente para el conjunto completo de datos puede ser muy costoso.

Una posible desventaja de utilizar este método es que al ser un proceso estocástico, la curva de optimización no es una función suave y puede presentar mucho ruido durante el entrenamiento.

2.4.5. Función de costos

El método de entrenamiento como problema de optimización con descenso del gradiente requiere de una función E llamada función de costos o pérdida, que mida el error entre la salida del modelo dada una entrada y el objetivo de entrenamiento. La elección de esta función depende del tipo de modelo que se quiere construir y la naturaleza de los datos de entrenamiento. Existen muchas funciones de costos que se pueden utilizar, pero todas ellas requieren cumplir con los siguientes requisitos:

1. Función escalar de variable vectorial. $f : \mathbb{R}^n, \mathbb{R}^n \to \mathbb{R}$
2. Diferenciable respecto a los parámetros del modelo.
3. Monótona respecto al error. Es decir, para cualquier salida x_1, x_2 con error $e_1 \leq e_2$ respecto al mismo objetivo t, $f(x_1, t) \leq f(x_2, t)$

2.4.6. Técnicas de regularización

En el aprendizaje automático, existe un fenómeno llamado sobre-ajuste que se puede presentar al entrenar un modelo. El sobre-ajuste ocurre cuando el modelo "memoriza"los datos de entrenamiento, incluyendo los patrones de ruido y los valores atípicos de los datos tal que este generaliza de manera pobre a nuevos datos. Para reducir este fenómeno, existen diversas técnicas que se emplean durante el entrenamiento para obtener modelos con mejor generalización. Entre las más utilizadas están las capas de dropout, normalización por lotes, regularización en la función de pérdida y tasas de aprendizaje altas.

Estas herramientas son de gran utilidad, pero se debe encontrar un balance al momento de utilizarlas, pues una regularización muy agresiva podría impedir al modelo encontrar patrones subyacentes importantes de los datos y disminuir su rendimiento.

Dropout

El dropout es una capa de una red neuronal que se utiliza únicamente durante su entrenamiento. Su objetivo es “apagar“ de manera aleatoria un porcentaje de las neuronas de la capa anterior al establecer su valor a 0. Así eliminando su contribución a la suma ponderada con el objetivo de evitar la dependencia de un conjunto reducido de neuronas [4].

Normalización por lotes

Es una técnica de regularización que busca reducir el desplazamiento de covariable interna. Esta capa se encarga de normalizar las salidas de la capa anterior para que tengan media cero y varianza unitaria. Y, posteriormente, se escala y desplaza una cantidad determinada. Esto se realiza al ajustar las salidas de la siguiente manera:

$$z_i = \gamma \frac{x_i - \mu_{i\mathcal{B}}}{\sigma_{i\mathcal{B}}} + \beta$$

Donde x_i representa la salida de la capa anterior, $\mu_{i\mathcal{B}}, \sigma_{i\mathcal{B}}$ la media y desviación estándar por lote de cada elemento de la salida. γ y β son las constantes de escalamiento y desplazamiento de cada elemento, y son parámetros que se aprenden durante el entrenamiento.

Al normalizar los datos se busca que los datos que se encuentran en diferentes escalas no opaquen la contribución de otras entradas con escalas más pequeñas. Porque de ser así, tendrían que compensar con valores de pesos más grandes. Y esto puede llevar a oscilaciones del gradiente durante el entrenamiento y ralentizar o incluso impedir la convergencia [5].

Regularización de la función de perdida

Esta técnica busca reducir el sobre-ajuste al agregar un elemento a la función de pérdida que penalice la magnitud del vector de pesos del modelo durante el entrenamiento de la siguiente forma:

$$E_T = E_m + E_\omega$$
$$E_\omega = \lambda \sum_{i=1}^{n} |\omega_i|^k$$

Donde el error total E_T es la suma de la función de costos del modelo E_m y la magnitud del vector de pesos E_ω. λ es la constante de regularización también conocida como constante de decaimiento de pesos. La cual controla qué tan agresiva es la penalización de los parámetros de la red. El exponente $k \in [1, 2]$ determina si se utiliza una regularización lineal o cuadrática (L1 o L2).

De esta manera se penalizan valores altos de los pesos y se fomenta una distribución más dispersa entre todas las unidades. Esto ayuda a reducir la dependencia de neuronas especificas al reducir su contribución. Y así poder utilizar todos los componentes del modelo y mejorar la generalización.

2.5. Redes Neuronales Convolucionales

Las Redes neuronales convolucionales (RNC) son un tipo de red neural *feedforward* capaces de extraer información y aprender patrones de datos con estructura de convolución [6].

Las RNC son una de las arquitecturas más utilizados en la rama del aprendizaje profundo y se presentan en numerosas aplicaciones. Desde visión computarizada, modelos de lenguaje de gran tamaño hasta sistemas de detección en medicina.

Estas redes presentan propiedades muy llamativas en comparación con sistemas completamente conectados como lo son:

- Conexiones locales: cada neurona está conectada a un pequeño grupo de neuronas relacionadas en lugar de estar conectada a cada otra neurona de la siguiente capa. Esta reducción de conexiones permite reducir el número de parámetros y acelerar el proceso de convergencia.
- Distribución de pesos: un grupo de neuronas puede compartir los mismos pesos, lo que permite reducir el número de parámetros aún más.
- Reducción de dimensión por submuestreo: se puede utilizar una capa convolucional para utilizar la técnica de submuestreo para comprimir la dimensión de datos. Lo cual permite reducir el tamaño de estos y mantener la información más importante.

2.5.1. Operador convolución 2D

La convolución 2D es una operación vectorial en la cual cada elemento de salida es la suma ponderada entre un una matriz de pesos y los elementos vecinos al elemento de entrada. Esta matriz de pesos, también llamado filtro o kernel de convolución es una matriz que se utiliza para extraer información de la matriz de entrada.

En procesamiento de imágenes esta herramienta se utiliza para aplicar filtros como desenfoque, relieve, nitidez, entre otros dependiendo del kernel que se utilice. Para el caso de las RNC, los pesos de los filtros son los parámetros que se modifican durante el entrenamiento, y estos aprenden patrones de los datos relevantes para el objetivo. La expresión general de la convolución 2D es la siguiente:

$$\mathbf{C}(x,y) = \mathbf{I} * \mathbf{K} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(i,j)K(x-i, y-j)$$

Donde $\mathbf{C}$ es es el resultado de la convolución, $\mathbf{K}$ es el kernel de convolución e $\mathbf{I}$ es la imagen original. Un ejemplo de su operación se muestra en la figura 2.11

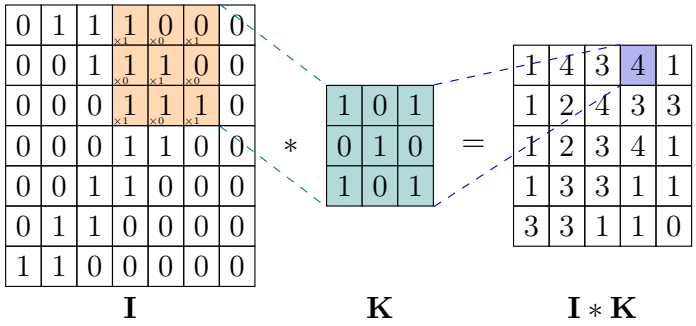


Figura 2.11: Representación de la convolución en dos dimensiones

2.5.2. Arquitectura de una RNC

Las RNC están construidas como una concatenación de capas convolucionales que aplican filtros de convolución sobre las entradas anteriores. Generalmente disminuyendo el tamaño de las matrices pero aumentando el número de canales en estos. Los cuales representan los patrones aprendidos de las capas anteriores, y que, conforme avanza el flujo de información por las capas, aprende patrones más complejos. Una representación de estas arquitecturas se muestra en la figura 2.12.

Para modelos de regresión y clasificación, la salida de la última capa convolucional pasa por un proceso de aplanamiento, el cual convierte las matrices de salida en un único vector que representa la salida final, o que pasa a otras redes densas para continuar su procesamiento.

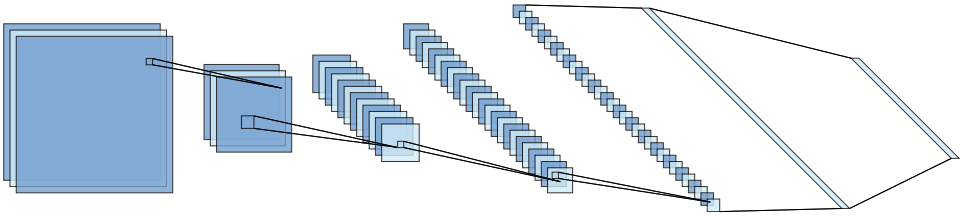


Figura 2.12: Diagrama de una red neuronal convolucional

2.5.3. Parámetros de una capa de RNC

Las capas convolucionales requieren la definición de ciertos parámetros que afectan el tamaño y forma de las matrices o imágenes de salida. La fórmula para calcular el tamaño final de la matriz de salida dados los parámetros de la capa convolucional es el siguiente:

$$W_o = \frac{W_{in} - F_W + 2P}{S + 1}$$
$$H_o = \frac{H_{in} - F_H + 2P}{S + 1}$$
$$C_o = F_m$$

Donde W_o, W_{in} son el ancho de la matriz de salida y de entrada respectivamente. H_o, H_{in} es la altura de la matriz de salida y de entrada, F_W, F_H las dimensiones del filtro de convolución y F_m el número de filtros por capa. P es el padding añadido, S el tamaño de stride y C_o es el número de canales de salida.

Filtros

Uno de los parámetros más importantes es la definición del tamaño del filtro de convolución, generalmente definido como una matriz cuadrada de $n \times n$. Con tamaños de filtro más grandes se pueden aprender patrones que abarcan más superficie de la imagen, sin embargo esto incrementa el costo computacional de realizar la convolución y tambien reduce el tamaño final de la imágen de salida. Otro parámetro importante es el número de filtros por capa, que determina el número de canales de salida.

Padding

Un problema que surge al utilizar la operación de convolución, es que se tiende a perder información en el perímetro de la imagen ya que se reduce el tamaño final de esta. Para evitar este comportamiento, se utiliza el *padding* o rellenamiento. Este método consiste en agregar celdas vacías en el perímetro de imagen original para que al realizar la convolución, la salida final de la imagen sea mayor que de no ser utilizado.

Stride

El stride determina cuántos elementos se mueve el filtro de convolución por paso. Es decir es un escalamiento de la velocidad de corrimiento del filtro. Frecuentemente se utiliza un stride unitario para mover el filtro un elemento a la vez. Sin embargo con valores mayores se puede realizar un submuestreo de la matriz de entrada. Por esto mismo incrementar el tamaño del stride disminuye considerablemente el tamaño de la matriz de salida.

2.6. Geometría diferencial

La geometría diferencial es una rama de las matemáticas cuyo enfoque de estudio son los espacios con curvatura como las esferas, superficies y otros espacios topológicos en múltiples dimensiones. Esta rama permite utilizar diferentes herramientas de cálculo diferencial para adaptarlos a problemas donde su variable de estudio se encuentra en espacios no euclidianos.

Además, es pieza fundamental para la geometría Riemanniana, que es una teoría que unifica la geometría euclidiana, la geometría analítica, la geometría proyectiva y la geometría hiperbólica [7].

2.6.1. Variedades

En geometría diferencial, las variedades $\mathcal{M}$ son espacios topológicos d-dimensionales cuya característica fundamental es que localmente son homeomórficos al espacio euclidiano $\mathbb{E}^d$. Es decir, que $\forall p \in \mathcal{M}$ existe un conjunto abierto $U \in \mathcal{M} : U \ni p$ y un mapeo biyectivo continuo $\phi : U \to \Omega$ a un conjunto abierto $\Omega \in \mathbb{R}^d$. A este homeomorfismo se le conoce como mapa o carta.

Se le conoce como atlas a la familia de cartas $\{U_a, \phi_a\}$ para el conjunto de U_a que forman una cubierta abierta de $\mathcal{M}$

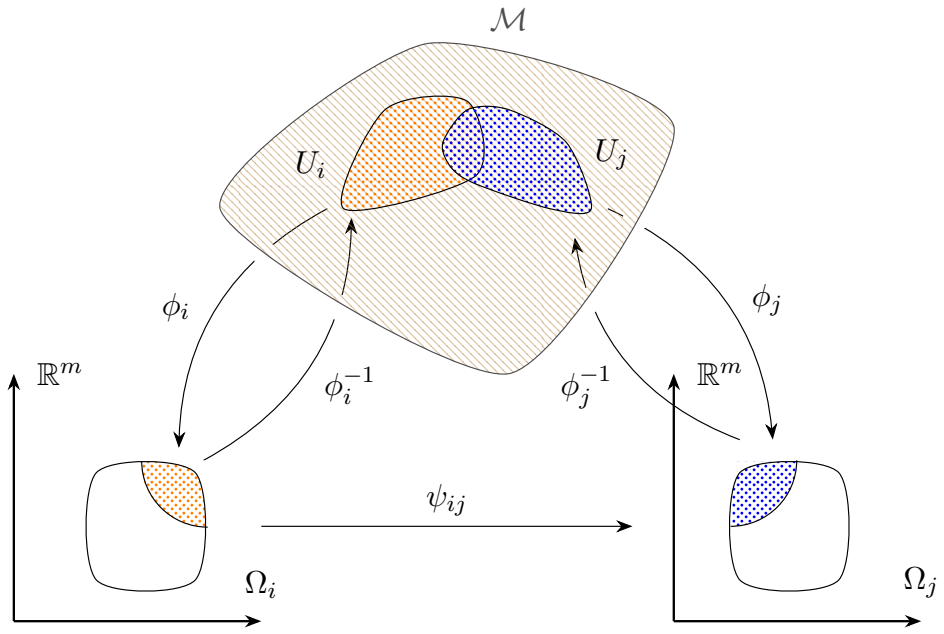


Figura 2.13: Representación de una variedad con sus cartas

Uno de los ejemplos más intuitivos de las variedades es la superficie de la tierra y esto no es casualidad. Algunos de los principios fundamentales de

las variedades surgieron de la investigación de la topología y la cartografía.

La tierra tiene una forma de elipsoide irregular, que generalmente se representa como una esfera tridimensional. Y a pesar de esto, existen mapas bidimensionales que permiten describir cualquier ubicación en la tierra. Esto se debe a que para los humanos, cuyo tamaño es órdenes de magnitud menor que el radio del planeta, la superficie de la tierra aparenta ser localmente euclidiana. Y sin embargo, es curvo. En la figura 2.14 se puede observar que al acercar la vista sobre el mapa, las delimitaciones de latitud y longitud aparentan ser líneas perpendiculares y se aproximan más a un espacio ortogonal en $\mathbb{R}^2$.

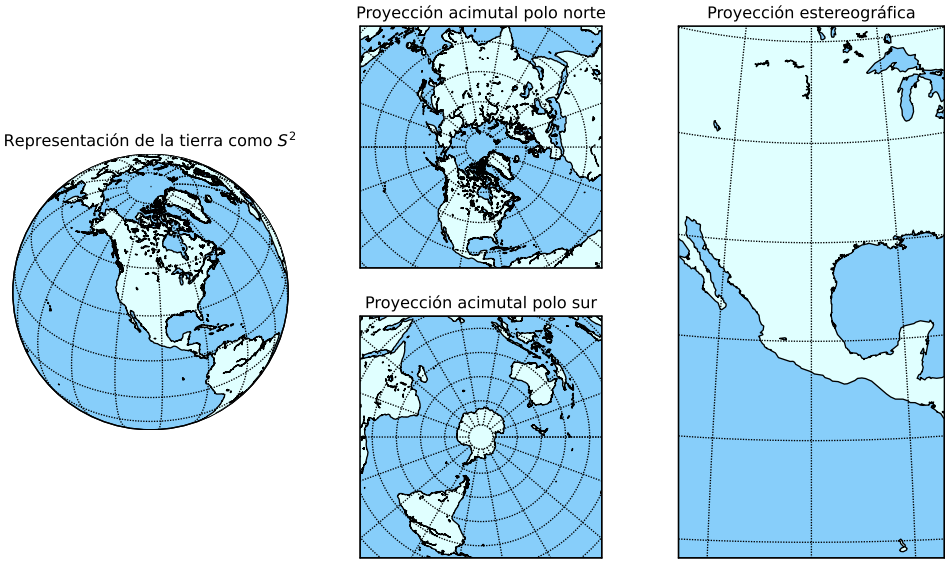


Figura 2.14: a) Representación de la tierra como una esfera en S^2, b) Proyecciones acimutales de los hemisferios centrado en los polos, los cuales representan cartas en $\mathbb{R}^2$ y su unión forma un atlas sobre la tierra, c) Proyección estereográfica alrededor de Lat N 20° W 100°

2.6.2. Variedades diferenciables

Las variedades diferenciales son un conjunto de las variedades topológicas continuas que poseen estructura diferenciable. Es decir, que se pueden definir funciones diferenciables $f' : \mathcal{M} \to \mathbb{R}$. Y por lo tanto se puede realizar cálculo en ellas. Estrictamente, la condición para ser una variedad diferenciable es que $f \circ \phi_a^{-1} \quad \forall \phi_{\mathcal{M}}$ sea diferenciable. Esta condición exige que la diferenciabilidad sea consistente incluso cuando se realice un cambio de carta.

2.6.3. Variedades de Riemann

Las variedades Riemannianas son aquellas variedades diferenciables cuyos espacios tangentes están equipados con una métrica $g\langle\cdot,\cdot\rangle$ que varia suavemente. Esta métrica permite definir distancias, ángulos y volúmenes entre puntos en $\mathcal{M}$. Así como el cálculo de la curvatura o el gradiente mediante la derivación e integración de estas.

Espacio tangente

El espacio tangente $\mathcal{T}_{\mathcal{X}}$ de una variedad está definido como el hiperplano formado por el conjunto de todos los vectores tangentes a un punto $\mathcal{X} \in \mathcal{M}$ tal que este espacio aparenta ser localmente euclidiano. Y el cual tiene la misma dimensionalidad que la variedad. En la figura 2.15 se muestra un ejemplo de espacio tangente para la esfera en S^2.

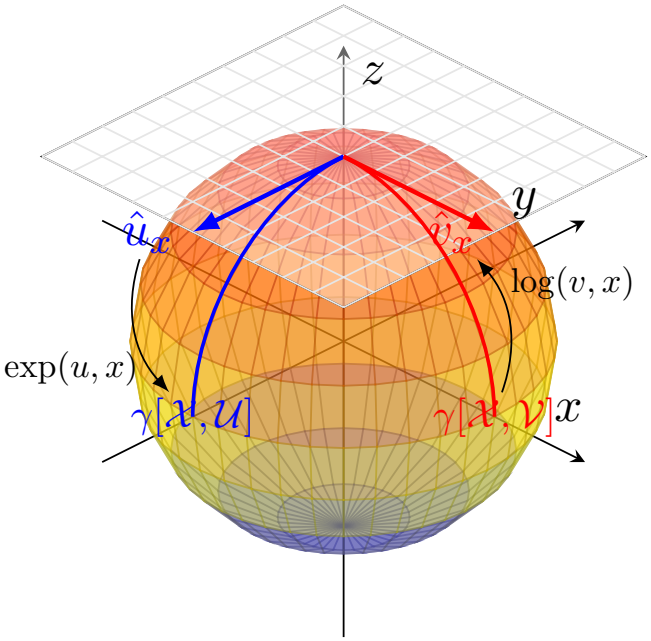


Figura 2.15: Representación del espacio tangente $\mathcal{T}_b$ de una esfera en S^2 y su mapa exponencial y logarítmico

Geodésicos

Las líneas geodésicas son una generalización del concepto de línea recta en el espacio euclidiano para las variedades Riemannianas. Estas curvas $\gamma : [a, b] \in \mathcal{M} \to \mathcal{M}$ se definen como el conjunto de puntos conectados que forman una trayectoria entre dos puntos a, b de la variedad tal que su distancia es mínima. La distancia de un geodésico se calcula como la integral de línea entre los puntos $[a, b]$ definida por la métrica de la variedad.

2.6.4. Grupos de Lie

Los grupos de Lie son aquellas variedades equipadas con una estructura de grupo (un espacio con operador binario que cumple con las propiedades asociativa, identidad, inverso y cerradura). Y además, cuentan con un álgebra de Lie denotado $\mathfrak{m}$ [8].

El álgebra de Lie $\mathfrak{m}$ está asociado al espacio tangente en el punto de identidad y tiene las siguientes propiedades:

- Es un espacio vectorial y por lo tanto sus elementos pueden ser expresados como vectores en $\mathbb{R}^m$, cuya dimensión m es el número de grados de libertad en $\mathcal{M}$.
- El mapa exponencial $Exp : \mathfrak{m} \to \mathcal{M}$ convierte elementos del álgebra de Lie en elementos del grupo. La operación inversa se define como el mapa logarítmico $\log : \mathcal{M} \to \mathfrak{m}$
- Los vectores en el espacio tangente en cualquier punto $\mathcal{X} \in \mathcal{M}$ puede ser transformado al espacio tangente de la identidad $\mathcal{E} \in \mathcal{M}$ mediante una transformación lineal y se le conoce como el adjunto.

Los grupos de Lie tienen la capacidad de transformar elementos de otros conjuntos y producir rotaciones, traslaciones, o escalamientos entre otras acciones. Es por esto que se utilizan ampliamente en la robótica y los gráficos computarizados.

Dado un grupo de Lie en $\mathcal{M}$ y un conjunto $\mathcal{V}$, se define $\mathcal{X} \cdot v$ como la acción de $\mathcal{X} \in \mathcal{M}$ en $v \in \mathcal{V}$. Y para ser una acción de grupo válida, esta debe cumplir las propiedades de identidad y compatibilidad:

$$\begin{aligned} \text{Identidad:} \quad & \exists \mathcal{E} \in \mathcal{M} : \quad \mathcal{E} \cdot v = v \\ \text{Compatibilidad:} \quad & (\mathcal{X} \circ \mathcal{Y}) \cdot v = \mathcal{X} \cdot \mathcal{Y} \cdot v \end{aligned}$$

Entre los grupos de Lie se encuentran los grupos que producen rotaciones en otros espacios, como el grupo especial ortogonal $\mathbf{SO}(n)$, los cuaterniones en $\mathbb{H}_1$, las rotaciones en los complejos unitarios S^1 y el grupo de movimiento de cuerpo rígido $SE(n)$

2.7. Estadística direccional

La estadística direccional es una rama de la estadística cuyo objeto de estudio son los datos y observaciones de variables vectoriales unitarias como las direcciones y rotaciones en $\mathbb{R}^n$ así como datos circulares o esféricos en $\mathcal{S}^d$ [9].

Esta rama tiene aplicaciones en diferentes campos de la ciencia donde se trabaja con datos cíclicos y/o direccionales como el estudio de la dirección

del viento en la meteorología, el estudio de las estructuras de proteínas en la biología, el estudio de los epicentros de terremotos en ciencias de la tierra y modelos estadísticos en el aprendizaje de máquina entre otros [9] , [10].

2.7.1. Distribución de Von Mises-Fisher

La distribución de Von Mises-Fisher es una distribución de probabilidad unimodal con covarianza isotrópica definida en el espacio de las hiperesferas $(d-1)$-dimensionales en $\mathbb{R}^d$ [11]. Su función de densidad está dada por:

$$f_x(x;\mu,\kappa) = C_d(\kappa)\exp\{(\kappa \mathbf{x}^T\mu)\} \tag{2.1}$$

Donde los vectores unitarios $\mathbf{x}, \mu \in \mathbb{R}^d$ representan los puntos en el espacio, y la media direccional de la distribución respectivamente. El parámetro de concentración κ es una medida de dispersión y define qué tanto la distribución se concentra alrededor de la media direccional. La constante de normalización para ser una distribución unitaria válida está dada por la siguiente expresión:

$$C_d(\kappa) = \frac{\kappa^{d/2-1}}{(2\pi)^{d/2} I_{d/2-1}(\kappa)} \tag{2.2}$$

Donde I es la función de Bessel modificada del primer tipo.

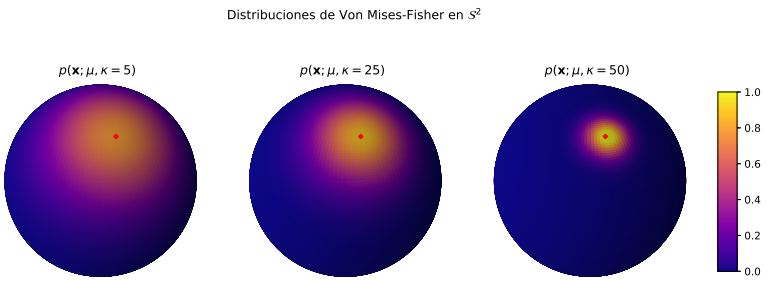


Figura 2.16: Distribuciones de Von Mises-Fisher con diferentes parámetros de concentración

2.8. Modelos de mezcla

Los modelos de mezcla son modelos probabilísticos que permiten aproximar funciones de densidad de probabilidad arbitrarias como una combinación lineal de diferentes distribuciones parametrizadas o kernels de la forma:

$$P(t) = \sum_{i=1}^{m} \alpha_i \phi_i(t) \qquad \sum_{i}^{m} \alpha_i = 1 \tag{2.3}$$

Donde ϕ_i es una distribución de probabilidad en t caracterizada por parámetros predefinidos, y α_i son los coeficientes de mezcla de sus respectivos kernels. Estos coeficientes representan la proporción que contribuye dicha distribución a la mezcla y la suma de estos debe ser unitaria para ser una función de densidad de probabilidad válida [12].

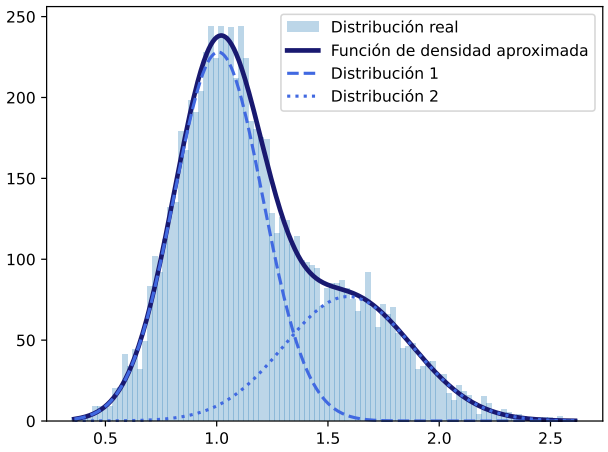


Figura 2.17: Aproximación de una función de densidad como mezcla de dos distribuciones normales

Una función de densidad condicional también puede ser modelada como una mezcla, al definir los parámetros de la mezcla como funciones de la variable condicional [13].

$$P(t|x) = \sum_{i=1}^{m} \alpha_i(x) \phi_i(t|x) \qquad \sum_{i}^{m} \alpha_i = 1 \tag{2.4}$$

2.8.1. Modelos de mezcla Gaussianos

Los modelos de mezcla gaussianos son aquellos modelos de mezcla que buscan aproximar la función de densidad $P(t|x)$ como una combinación de distribuciones normales [12].

$$\phi_i(t|x) = \mathcal{N}(\mu_i(x), \Sigma_i(x)) \tag{2.5}$$

$$P(t|x) = \sum_{i=1}^{m} \alpha_i(x)\mathcal{N}(\mu_i(x), \Sigma_i(x)) \tag{2.6}$$

2.8.2. Modelos de mezcla de Von Mises-Fisher

Estos modelos se utilizan para aproximar densidades de probabilidad definidas en la variedad de las hiperesferas. Y son modelos de mezcla en la que su kernel ϕ es la distribución de Von Mises-Fisher [14], [15], [16].

$$\phi_i(t|x) = \mathcal{K}(\mu_i(x), \kappa_i(x)) \tag{2.7}$$

$$P(t|x) = \sum_{i=1}^{m} \alpha_i(x)\mathcal{K}(\mu_i(x), \kappa_i(x)) \tag{2.8}$$

2.8.3. Estimación por máxima verosimilitud

La Estimación por máxima verosimilitud (EMV) es un método estadístico que se utiliza para encontrar los parámetros θ de una distribución o modelo paramétrico que mejor describe los datos de un conjunto de muestras $\mathcal{D}$

La verosimilitud es una medida de qué tan bueno es un modelo paramétrico para describir un conjunto de datos observados y se define como:

$$\mathcal{L}(\mathcal{D}, \theta) = \prod_{i=1}^{N} P(x_i|\theta)$$

Los parámetros de este modelo se pueden calcular al evaluar los puntos críticos de la función conociendo su derivada.

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta} = 0$$

Pero también se puede aproximar mediante proceso iterativos con el objetivo de maximizar la verosimilitud de los datos dado ciertos parámetros dentro de un conjunto de parámetros. Como objetivo de optimización este se puede obtener al definir la función de costos como el Negativo del logaritmo de la función de verosimilitud. Del inglés, Negative log likelihood (NLL) y se define como:

$$E = -\log(\mathcal{L}(\mathcal{D}, \theta)) = -\sum_{i=1}^{N} \log(P(x_i|\theta))$$

2.9. Optimización bayesiana

La Optimización Bayesiana (OB) es un método estadístico para modelar funciones en las que tomar muestras es costoso y por lo tanto, encontrar sus puntos máximos o mínimos es difícil [17].

$$x^+ = \arg\max_{x \in A} f(x)$$

Este método se deriva del teorema de Bayes, en la que se actualiza el grado de confianza de una hipótesis (a posteriori) de acuerdo al conocimiento previo (a priori) y a la nueva evidencia.

$$P(M|E) \propto P(E|M)P(M)$$

Donde $P(M|E)$ es el la probabilidad del modelo M dada una evidencia E y es proporcional a la verosimilitud de la evidencia dada el modelo multiplicado por la probabilidad previa del modelo.

El principio de este método es utilizar la información que ya se tiene de la función (previo) con las muestras que se obtienen (evidencia) para actualizar un modelo aproximado de la función objetivo (posterior), generalmente mediante un proceso gaussiano. Y posteriormente, calcular un posible máximo de acuerdo a un criterio.

El criterio es una función de utilidad a veces también llamada función de adquisición que se utiliza para determinar la región dentro del espacio de búsqueda donde se debe tomar la siguiente muestra para maximizar la utilidad. Esto al tener un balance entre la exploración de áreas donde no hay suficientes muestras y la explotación, que consiste en muestrear regiones donde ya se conoce que existe un valor alto de la función.

2.9.1. Procesos gaussianos

Un Proceso Gaussiano (PG) $\mathcal{GP}$ es un proceso estocástico el cual consiste en una familia de funciones de variable aleatoria $\{f_0(x), ..., f_n(x)\}$ tal que cualquier conjunto finito de estas funciones tenga una distribución conjunta normal multivariable. Y por lo tanto, es una distribución sobre posibles funciones [18].

Los procesos gaussianos se utilizan para modelar funciones desconocidas tomando cada muestra como una evaluación de estas funciones:

$$Y = g(X) \backsim \mathcal{GP}(m(x), k(x, x')) \tag{2.11}$$

Donde m se conoce como la función de media y k se conoce como la función de covarianza o kernel. Estos elementos se definen como:

$$m(x) = \mathbb{E}[f(x)]$$
$$k(x, x') = \mathbb{E}[(f(x) - m(x))\left(f(x') - m(x')\right)^T]$$

El kernel del proceso gaussiano define la forma de la distribución y qué tipo de funciones son las más probables para describir la función objetivo. Este kernel permite definir una medida de similitud entre dos puntos del mismo espacio x, x', bajo el principio que puntos cercanos deberían tener valores similares. Para este fin existen muchas funciones de covarianza, como la función de base radial, el kernel de Matérn, o el periódico [19].

Entonces, la distribución de la función $y^* = g(x^*)$ dado un conjunto de observaciones $\mathcal{D} = \{\mathbf{x} = \{x_0, ..., x_n\}, \mathbf{y} = \{y_0, ..., y_n\}\}$ se calcula como:

$$P(y^*|x^*, \mathcal{D}) = \mathcal{N}(\mu(x^*), \Sigma(x^*)) \tag{2.12}$$
$$\mu(x^*) = \mathbf{K}(\mathbf{x}, x^*)^T(\mathbf{K}(\mathbf{x}, \mathbf{x}) + \mathbf{K}_e)^{-1}\mathbf{y} \tag{2.13}$$
$$\Sigma(x^*) = \mathbf{K}(x^*, x^*) - \mathbf{K}(\mathbf{x}, x^*)^T(\mathbf{K}(\mathbf{x}, \mathbf{x}) + \mathbf{K}_e)^{-1}\mathbf{K}(\mathbf{x}, x^*) \tag{2.14}$$

Donde μ es la media de la función, Σ la matriz de covarianza, y las matrices $\mathbf{K}$ son el resultado de aplicar el kernel del PG a cada elemento de los vectores $x, \mathbf{x}$. En la figura 2.18 se muestra un ejemplo de un proceso gaussiano.

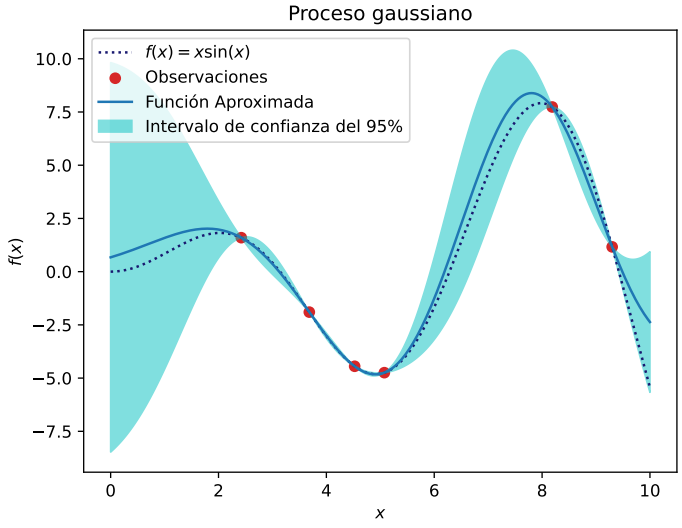


Figura 2.18: Ejemplo de un proceso gaussiano

2.9.2. Función de utilidad

La función de utilidad o adquisición de un proceso de optimización bayesiana tiene el objetivo de obtener la mejor propuesta de evaluación dentro del espacio de búsqueda para maximizar el valor esperado de la función. Es decir, se encarga de analizar el modelo aproximado de la función y proponer la región donde se debería tomar la siguiente muestra para mejorar la confianza del modelo (explorar áreas con mucha incertidumbre) o encontrar un máximo local (explotar áreas ya conocidas de alto valor).

Un ejemplo de función de utilidad es el límite de confianza superior que se define como:

$$u(X, \alpha) = \mu(X) + \alpha\sigma(X) \tag{2.15}$$

Donde u es la utilidad esperada del punto X, μ, σ la media y desviación estándar del modelo evaluado en X y α es una constante que regula el balance entre exploración y explotación del espacio de búsqueda. Cuando α es pequeño, el proceso de optimización favorecerá explotar regiones con un alto valor esperado, y si es grande favorecerá explorar regiones con mucha incertidumbre.

2.9.3. Ejemplo de optimización bayesiana

Para realizar un proceso de optimización bayesiana, es necesario contar con un modelo de la función objetivo inicial (previo) mediante una suposición de su forma y tomar algunas muestras para inicializarlo. En la figura 2.19 se muestra la inicialización de un PG [20].

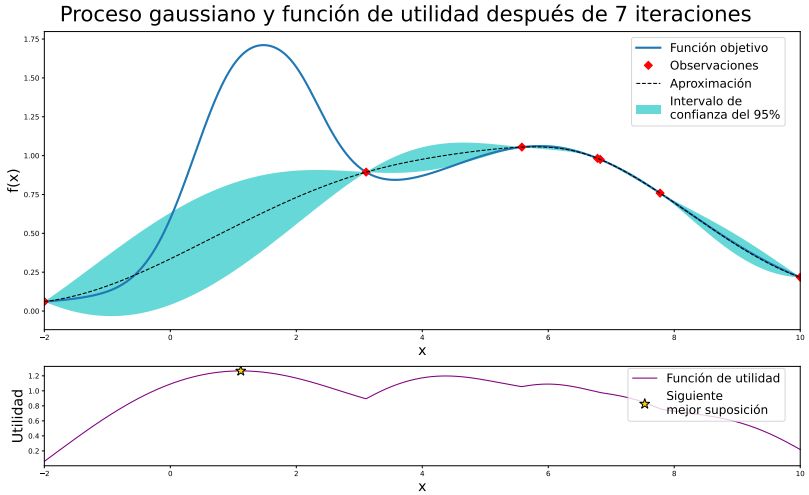


Figura 2.19: Ejemplo de un proceso gaussiano unidimensional después de 7 iteraciones

Para obtener la siguiente muestra se utiliza la función de utilidad, que en este caso propone una región poco explorada (con gran incertidumbre) del espacio de búsqueda. Al tomar la siguiente muestra, se actualiza el posterior dada dicha observación. Y su resultado se muestra en la figura 2.20

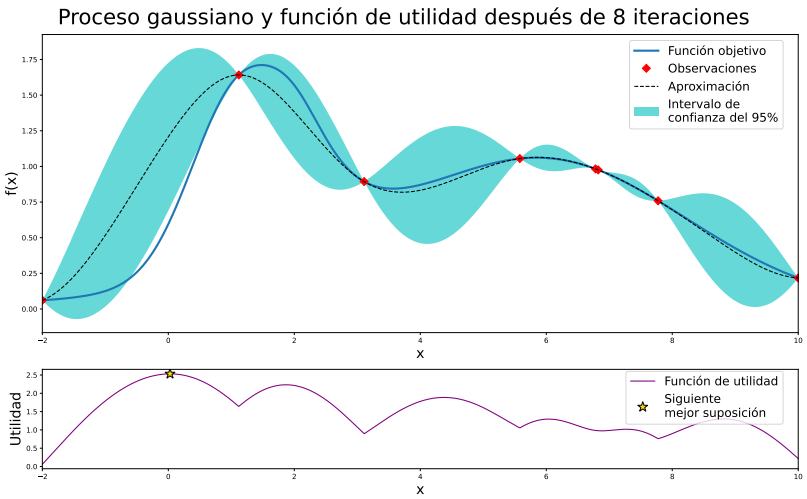


Figura 2.20: Ejemplo de un proceso gaussiano unidimensional después de 8 iteraciones

Después de repetir este proceso un número finito de iteraciones, se puede encontrar algún máximo de la función objetivo, minimizando el número de observaciones necesarias para obtenerlo. A continuación se muestra un ejemplo donde se encontró el máximo de la función objetivo después de 15 iteraciones:

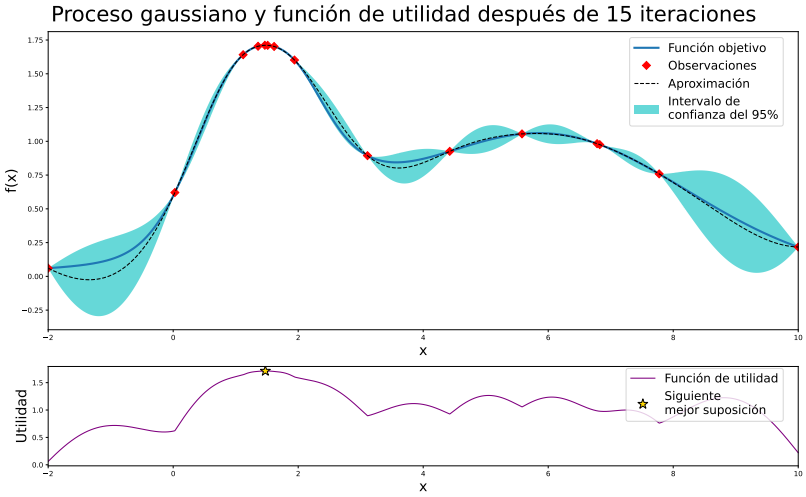


Figura 2.21: Ejemplo de un proceso gaussiano unidimensional después de 12 iteraciones

2.10. Trabajo relacionado y estado del arte

Antes de la adopción de enfoques basados en redes neuronales de los últimos años, la investigación se basaba en una combinación de métodos analíticos y estadísticos para la manipulación de objetos con robots. Un método popular se fundamentaba en el análisis de nubes de puntos e imágenes RGB-D mediante Análisis de componentes principales, por sus siglas en inglés Principal Component Analysis (PCA) para identificar el eje principal del objeto. Propuestas de agarre se derivaban de este resultado y posteriormente se aplicaban métodos geométricos para determinar y refinar el punto de agarre óptimo como en [21],[22],[23], [24].

Recientemente y debido al auge en el desarrollo de inteligencia artificial y aprendizaje automático, los enfoques basados en redes neuronales han cobrado gran relevancia. En parte gracias a la capacidad de estos modelos de proponer configuraciones de agarre con altos indices de fidelidad y exactitud. Algunos modelos basados en arquitecturas de redes neuronales profundas como Sparse-GRConvNet[25], el cuál integra visión computarizada para inferir propuestas de agarre óptimas, ha alcanzado tasas de éxito del 97.5 % con datasets como el Cornell Grasping Dataset(CGD).

Algunos trabajos implementan redes neuronales adicionales para el segmentado de objetos y su reconocimiento que auxilian otros sistemas para su funcionamiento como Mask R-CNN [26]. Arquitecturas populares como Dex-Net 2.0 proponen el uso de redes neuronales profundas para modelar la planeación de agarre directamente de una imagen de profundidad. Sus autores desarrollaron esta arquitectura al entrenar sus modelos con datasets

sintéticos debido al alto coste en tiempo y recursos de entrenar estos modelos con datos reales. Estos modelos han alcanzado tasas de éxito que superan a la mayoría de modelos basados en métodos analíticos como PCA [27].

Nuevas propuestas como Planeación de agarres orientado a tareas de una toma. Del inglés, One-Shot Task-Oriented Grasping (OS-TOG) permiten a los robots generar candidatos de agarre orientado a tareas de trabajo usando una única toma inicial por objeto, alcanzando una exactitud de 83.2 % tanto en simulación como en aplicaciones de la vida real [28].

Como alternativa, modelos basados en aprendizaje por refuerzo como Grasp-Q-Network implementan una política de planeación de agarre que no requiere un dataset anotado y predefinido de muestras de agarre. En cambio, este modelo aprende automáticamente mediante la interacción con el ambiente real usando retroalimentación visiomotriz [29]. Su función de recompensa asigna valores positivos a agarres exitosos, valores menores para contactos con el objeto y penaliza los movimientos sin interacción. De esta manera, desarrollando un sistema de planeación que mejora con el uso.

Para diseñar sistemas robustos contra ruido de fondo y datos atípicos, algunos trabajos exploran la redes residuales profundas [30] basadas en ResNet, las cuales permiten a sus sistemas ignorar las partes irrelevantes de la imagen de profundidad o nubes de puntos como lo es el fondo de la imagen, oclusión parcial u otras áreas que no aporten información relevante para el objetivo.

Las RNC han ganado amplia popularidad y adopción gracias a su gran capacidad de extraer información relevante de imágenes y otros datos con estructurados. Se han propuesto sistemas que operan en tiempo real usando modelos de regresión directa y clasificación para predecir configuraciones de agarre óptimos. Así como para la clasificación de objetos por categoría para planear dichos agarres [31]. En contraste, GR-ConvNet v2 también una arquitectura basada en RNC, cuya innovación es que no está implementada como modelo de regresión directa ni como modelo de clasificación. Este usa módulos de inferencia para generar mapas de agarres viables entre regiones de pixeles basados en la geometría local de la imagen RGB-D. De esta manera, el sistema aprende a generalizar propuestas de agarre incluso para objetos fuera de su conjunto de entrenamiento [32].

La información que proviene directamente de los sensores frecuentemente se presenta incompleta y con ruido. En la literatura esta incertidumbre se ha modelado con enfoques probabilísticos que integran modelos de mezclas e inferencia variacional o bayesiana para describir la planeación y trayectorias de agarre. Esto permite al robot tomar decisiones que tomen en cuenta la incertidumbre de sus propios sensores y facilitar la tarea de la planeación de movimientos [33].

Estos modelos probabilísticos también son utilizados para planear agarres cuando los objetos están parcialmente ocultos o deformados. Al realizar un modelo probabilístico sobre la geometría de los objetos, se puede inferir la

forma de estos a partir de esta información incompleta y planificar agarres con mejor exactitud [34].

2.10.1. Estimación de pose y aprendizaje en variedades

Existe un amplio desarrollo en el problema de la estimación de la pose y regresión de rotaciones fuera del contexto de la robótica. [35] explora la dificultad de utilizar redes neuronales profundas para realizar regresiones en rotaciones, y demuestran que todas las representaciones tradicionales presentan discontinuidades que producen errores considerables para ciertos ángulos.

PDQ-NET [36] propone un modelo probabilístico para la estimación de poses al construir una distribución de probabilidad en $SE(3)$ sobre los cuaterniones unitarios duales en $\mathbb{DH}_1$. En cambio [37] utiliza una mezcla de RNC y el teselado discreto de $SE(3)$ para estimar poses de objetos. [38] utilizan modelos de Bin & Delta con diferentes métodos (tradicional, geodésico y probabilístico) para estimar la pose de objetos a partir de imágenes 2D.

Para el aprendizaje en variedades la investigación reciente se enfoca en construir modelos probabilísticos sobre estos espacios. [39] exploran la generalización de la distribución normal sobre variedades riemannianas. [18] y [40] buscan adaptar procesos gaussianos al definirlos sobre su espacio tangente para que al aplicar el mapa exponencial, estos recubran la variedad y así poder hacer inferencia multivariable sobre estos espacios.

Un método mas reciente [41] explora construir procesos gaussianos intrínsecos a la topología de las variedades para poder realizar regresiones cuya variable de respuesta esté definida en dichas variedades. Otros como [42] buscan realizar inferencia en variedades al construir conjuntos de confianza para generar modelos predictivos.

Con esto se concluye este capítulo, dónde se explicaron las bases teóricas que fundamentan el desarrollo de las arquitecturas de red y modelos probabilísticos que fueron propuestos para la planeación de agarres. En el siguiente capítulo se usaran los conceptos aquí descritos para dar una explicación y respaldar las decisiones de diseño y entrenamiento del modelo.

Capítulo 3

Planeación de movimientos con redes neuronales

En este capítulo se detalla el análisis y definición del problema y se establecen los parámetros de diseño de acuerdo al entrono de desarrollo y el robot de servicio utilizado. Posteriormente se describe la filosofía de diseño y las arquitecturas propuestas para el sistema de planeación de agarres. También se especifica el método de recolección de datos para la creación de un dataset de nubes de puntos con anotaciones de propuestas de agarre. Finalmente se describe el procedimiento de entrenamiento de las redes, y los obstáculos y decisiones de diseño que se tomaron durante este proceso.

3.1. Parámetros de diseño

Antes de poder diseñar un sistema para la planeación de movimientos de un robot de servicio para la manipulación de objetos, primero se debe tomar en cuenta las siguientes consideraciones sobre la naturaleza del problema y el entorno de desarrollo.

3.1.1. Naturaleza del problema

El objetivo del trabajo de investigación es diseñar un sistema para la planeación de agarres de objetos basado en una arquitectura de red neuronal. En concreto, se busca diseñar un sistema capaz de proponer configuraciones del efector final del robot de servicio para tomar objetos a partir de la información proporcionada por una cámara RGB-D.

Este trabajo se limita a encontrar la pose que debería tener el efector final relativa a un marco de referencia intrínseco del robot para poder tomar un objeto. Por lo tanto, los problemas de encontrar la cinemática inversa y la planeación de trayectorias para llegar a esa configuración no son objeto de estudio en esta investigación.

Entonces, se requiere definir los parámetros de diseño y las variables de interés principales:

1. Se define un agarre g como la configuración del efector final también llamado *gripper* que permita al robot manipular un objeto arbitrario. Este agarre g está compuesto por la representación de la posición $x, y, z \in \mathbb{R}$ y la orientación del gripper relativo a la base del árbol de transformaciones del robot.

2. Existen múltiples formas de representar el grupo de rotaciones en 3 dimensiones $\mathbf{SO}(3)$ Entre ellos están las matrices de rotación, el grupo de los cuaterniones unitarios $\mathbb{H}_1$ y los ángulos de Euler. Y por lo tanto se debe elegir alguno.

3. Para cualquier nube de puntos de entrada $c \in \mathbb{R}^{H \times W \times 3}$ que contiene la información parcial de un objeto con posición y orientación respecto al robot arbitrarios, existe un conjunto de múltiples configuraciones de agarre válidos $g_1, g_2, ...g_n \in G$ dentro del espacio de configuración del robot.

Debido a estas restricciones, para este trabajo se decidió utilizar los cuaterniones sobre las matrices de rotación para representar las orientaciones del efector final. Esto porque permiten contener la información en una menor dimensionalidad ($\mathbb{R}^4 < \mathbb{R}^9$) y tienen una restricción más sencilla de mantener e implementar (vector unitario en lugar de determinante unitario).

Además, los cuaterniones no presentan el problema del bloqueo del cardán que se observa comúnmente cuando se utiliza la representación de ángulos de Euler. Sin embargo, el uso de los cuaterniones trae consigo su propio conjunto de problemas. Principalmente que este tipo de dato pertenece a la variedad de las hiperesferas en S^3 y por lo tanto, las métricas euclidianas no se pueden utilizar. Asimismo, el mapa de S^3 sobre $\mathbf{SO}(3)$ forma una doble cubierta. Lo que significa que cualquier composición de rotaciones puede ser representado por los cuaterniones $q, -q$.

3.1.2. Entorno de desarrollo

Una vez definidas las variables de interés, se necesitan delimitar las consideraciones del entorno y su aplicación, las cuales consisten en lo siguiente:

1. El sistema de planeación de agarres basado en redes neuronales a diseñar estará enfocado para **robots de servicio doméstico**. En su mayoría, este tipo de robots suelen ser altamente móviles, de menor tamaño que un humano promedio y cuya capacidad computacional no es muy elevada. Por lo tanto, la arquitectura de red propuesta no podrá ser muy profunda ni muy compleja.

2. Se diseñará un sistema para un humanoide basado en el robot de servicio del laboratorio Justina, el cuál poseé dos brazos de 7 grados de libertad con **grippers antipodales**.

3. La información de entrada se obtiene a partir de la nube de puntos parcial de un sensor RGB-D montado sobre la cabeza del robot y del cuál se conoce el árbol de transformaciones.

4. Tanto el entorno de pruebas como el de entrenamiento se realizará en un ambiente simulado. Por lo tanto, todos los datos adquiridos y generados serán sintéticos. Se usará el entorno de simulación Gazebo con integración de ROS para este fin.

3.1.3. Simulador Gazebo

El simulador Gazebo es una caja de herramientas para robótica de software libre que encapsula un entorno de simulación con motor de física, plugins de sensores, modelado de ruido, renderizado de modelos 3D, transporte de mensajes, entre otros. Este software es muy utilizado en la simulación y modelado de robots por su naturaleza abierta y extenso desarrollo y mantenimiento por la comunidad.

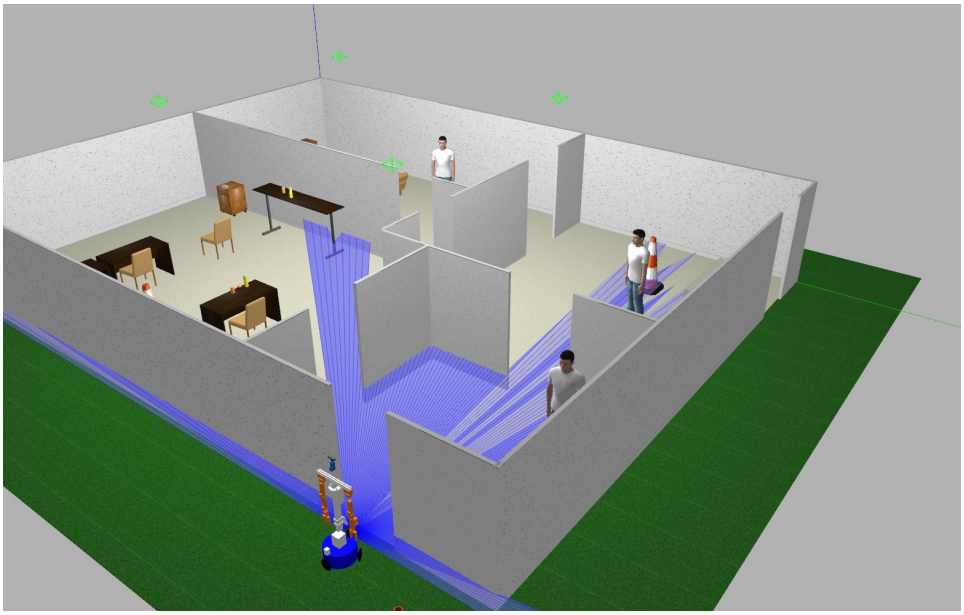


Figura 3.1: Ambiente simulado con Gazebo

3.1.4. ROS

Robot Operating System (ROS) es un software de uso libre que engloba diferentes herramientas y librerías para construir aplicaciones con robots.

Incluye paquetes de drivers, sistemas de comunicación serial entre nodos, cálculo de transformaciones homogéneas entre otros. En la fig 3.2 se muestra una captura de la integración entre ROS y Gazebo con el ambiente de visualización rviz.

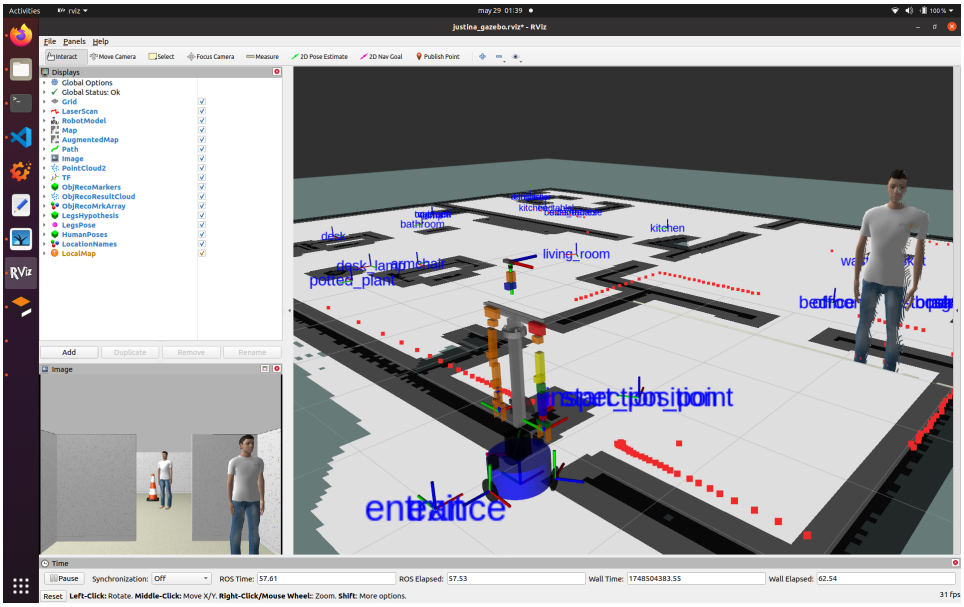


Figura 3.2: Interfaz gráfica rviz

3.2. Robot de servicio Justina

El robot de servicio que se utilizará para diseñar, entrenar y evaluar las arquitecturas de red neuronal propuestas es el robot de laboratorio Justina. A continuación se muestra una captura de la simulación de Justina y su árbol de transformaciones en rviz.

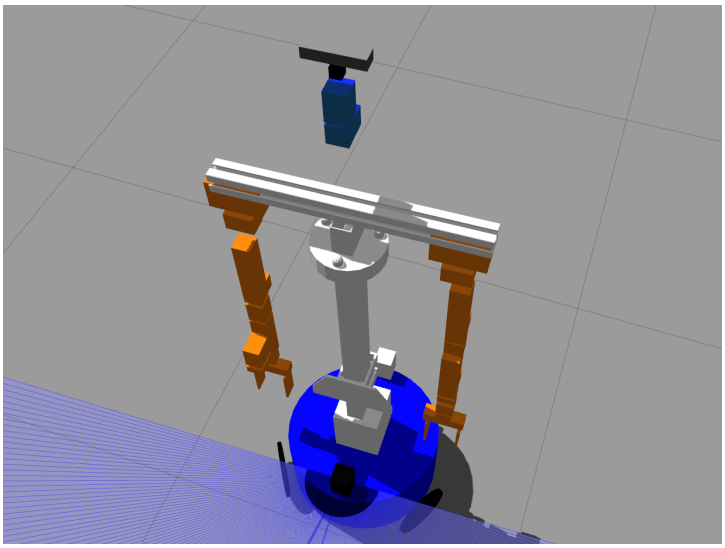


Figura 3.3: Simulación del robot de servicio Justina

3.3. Arquitectura propuesta

Teniendo en cuenta todos los parámetros de diseño y las limitaciones del entorno, se diseñó un sistema para la propuesta de agarres basado en redes neuronales. La siguiente arquitectura fue propuesta:

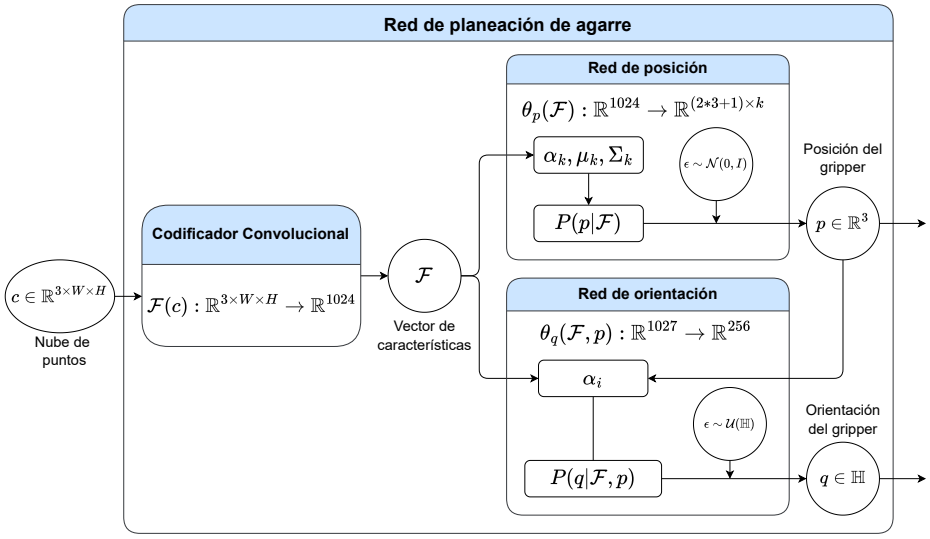


Figura 3.4: Diagrama funcional del sistema

El sistema está subdividido en tres partes principales: codificador convolucional, red de posición y red de orientación

3.3.1. Codificador convolucional

La primera parte del sistema es un codificador convolucional profundo que consiste en múltiples capas secuenciales basado en la arquitectura original de AlexNet. La red toma como entrada una imagen de profundidad de 200×200 de 3 canales (x, y, z) y pasa a través de una capa inicial de convolución con filtros de 11×11 sin padding y un stride de 1 con 32 canales de salida. El resultado se inserta en una capa de normalización por lotes y en una capa max-pool de (2,2) sin padding sobre la cual se aplica la función de activación GeLU.

Cada paso siguiente consiste en una capa de convolución con tamaños decrecientes del filtro, pero duplicando los canales de salida. Seguidos de una capa de normalización por lotes y finalmente la función de activación ReLU. Los canales de salida de la capa final se aplanan y pasan por una capa totalmente conectada hasta que la salida final devuelve un vector $\mathcal{F} \in \mathbb{R}^{1024}$, que representa las características más importantes de la nube de puntos. En la figura 3.5 se muestra un diagrama de la arquitectura.

La red se entrena como un auto-codificador mediante descenso de gradiente estocástico con regularización de pesos L2 para minimizar el Error medio cuadrático (EMC) entre los puntos de la entrada reconstruida y el objetivo.

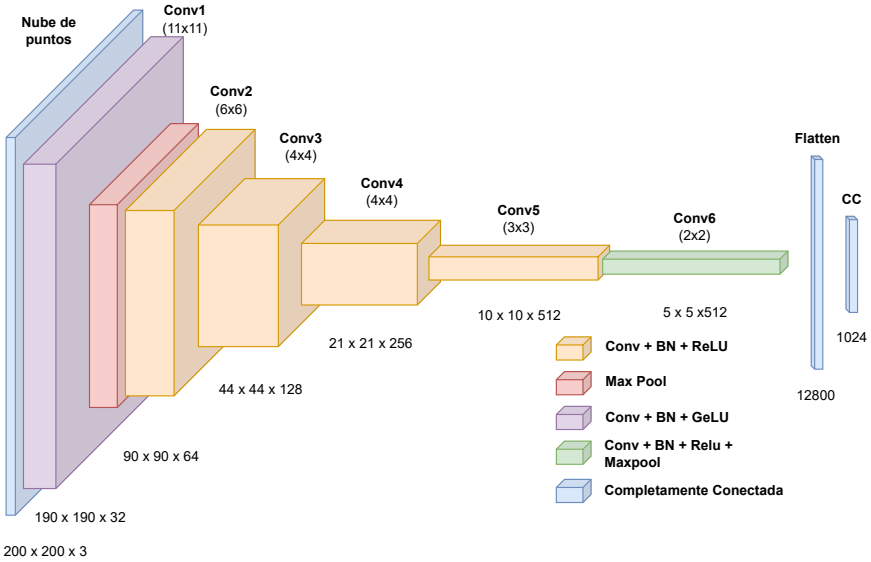


Figura 3.5: Arquitectura del codificador convolucional

3.3.2. Red de posición

Teniendo en cuenta la restricción 3, el modelo de regresión convencional no funcionará para este problema porque es multivalor (multimodal) por naturaleza. El método de optimización por minimización del EMC se reduce a encontrar la media condicional de los datos objetivo condicionados a la entrada, que no es necesariamente una solución válida. De hecho, se observó que el primer modelo entrenado de tal manera convergía a la solución trivial de proponer el centro del objeto. Ya que este es el punto que tiene el menor error respecto a todos los ejemplos de entrenamiento.

Por el mismo motivo, se optó por utilizar un modelo de mezclas para predecir la posición de agarre. En lugar de calcular los valores de la posición directamente, se entrenó la red para modelar la función de densidad de probabilidad de las posiciones del gripper $P(p)$ condicionado en el vector de entrada $\mathcal{F} \in \mathbb{R}^{1024}$. Y posteriormente muestrear de esta distribución, aquellas posiciones con mayor probabilidad.

Debido a que las posiciones se definen sobre el espacio euclidiano y se asume que las posiciones se encuentran normalmente distribuidas, se eligió el kernel de la distribución normal multivariable para el modelo de mezclas.

$$\phi_i(p|\mathcal{F}) = \mathcal{N}(\mu_i(\mathcal{F}), \Sigma_i(\mathcal{F})) \tag{3.1}$$

De este modo, las salidas de la red son los parámetros $\alpha_k, \mu_k, \Sigma_k$ de una combinación lineal de n distribuciones normales como una función del vector de características $\mathcal{F}$. Las matrices de covarianza Σ_k se simplificaron como la varianza diagonal (independiente de cada dimensión) para reducir la complejidad del modelo y el número de operaciones.

$$P(p|\mathcal{F}) = \sum_{k=1}^{n} \alpha_k(\mathcal{F}) \mathcal{N}(\mu_k(\mathcal{F}), \Sigma_k(\mathcal{F})) \tag{3.2}$$

Entonces el objetivo de entrenamiento se convierte en EMV. El cual es encontrar los parámetros de la distribución de probabilidad que mejor modelan los datos de entrenamiento al maximizar la función de verosimilitud $\mathcal{L}(\mathcal{D}, \theta)$. Donde θ son los parámetros de la mezcla Gaussianos; que al ser la salida de la red, son una función de los datos de entrenamiento y los parámetros de la red.

Dado que la función de verosimilitud es positiva y monótona, y se busca optimizar los parámetros utilizando el método de descenso del gradiente estocástico; entonces la función de costos que se elige es el NLL y se define a continuación:

$$\mathcal{L}(\mathcal{D}, \theta) = \prod_{i=1}^{N} P(p_i|\mathcal{F}_i; \theta) \qquad \forall p_1, ..., p_N \in \mathcal{D} \tag{3.3}$$

$$E = -\log \mathcal{L}(\mathcal{D}, \theta) = -\sum_{i=1}^{N} \log \left(\sum_{k=1}^{n} \alpha_k(\mathcal{F}_i; \theta) \mathcal{N}(p_i \mid \mu_k(\mathcal{F}_i; \theta), \Sigma_k(\mathcal{F}_i; \theta)) \right) \tag{3.4}$$

La arquitectura de este sistema es una red profunda de capas completamente conectadas cuya entrada es una capa de 1024 neuronas con 5 capas ocultas con 256 neuronas cada una, y cuya salida son 224 neuronas que forman los coeficientes de mezcla y sus parámetros de distribución para cada mezcla.

3.3.3. Red de orientación

Para diseñar esta red se probaron dos implementaciones diferentes. Primero se diseñó la red como un modelo de regresión geodésico. Posteriormente se diseñó una alternativa como un modelo de mezclas de Von Mises-Fisher.

Modelo de regresión geodésica

Esta red consiste en una serie de capas completamente conectadas que toman como entrada la concatenación del vector de características de la nube de puntos del codificador convolucional y la posición y devuelve un vector $u \in \mathbb{R}^4$

Debido a que se busca obtener un modelo de regresión para la orientación del agarre como cuaterniones unitarios, se debe tomar en consideración la geometría del espacio. Para entrenar esta red no se pueden usar las funciones de costo que se utilizan comúnmente para los modelos de regresión directamente. Ya que las métricas euclidianas no son adecuadas para hacer mediciones en el espacio de las hiperesferas. Además, no se puede garantizar que este vector sea unitario. Al ser una medida direccional se intentó resolver este problema simplemente normalizando el vector de salida. Sin embargo, se observó que esto condujo a un comportamiento errático de la red.

Por esta razón, se utilizaron principios de geometría diferencial para adaptar los algoritmos de entrenamiento. Para asegurarse de que la salida de la red perteneciera al espacio de cuaterniones unitarios, la hiperesfera S^3, se modificó la red para aprender un vector u proyectado al espacio tangente $\mathcal{T}_b$ en el punto base $b = (0, 0, 0, 1)$. Tal que al aplicar el mapa exponencial $Exp(b, u)$ el resultado pertenezca al grupo de los cuaterniones.

El objetivo de entrenamiento es minimizar el error de ángulo entre las orientaciones y en este caso equivale a minimizar la distancia media geodésica

entre la salida y las cuaterniones de destino. La función de pérdida elegida fue la distancia geodésica cuadrada.

$$E = d(q_o, q_t)^2 = arccos(\langle q_o, q_t \rangle)^2 \tag{3.5}$$

Donde $\langle \cdot, \cdot \rangle$ es el producto punto euclidiano en $\mathbb{R}^n$

La arquitectura de este sistema es una red profunda de capas completamente conectadas cuya entrada es una capa de 1024 neuronas con 4 capas ocultas con 256 neuronas cada una, y cuya salida son 4 neuronas que forman el vector u en el plano tangente $\mathcal{T}_b$

Modelo de mezclas de Von Mises-Fisher

De manera similar a la red de posición, esta red se definió para modelar la distribución de probabilidad de las orientaciones condicionadas en el vector de las características de la nube de puntos y la posición de agarre. $P(q|\mathcal{F}, p)$ utilizando el enfoque de estimación de densidad de mezclas.

Esto se hizo aproximando la distribución de la probabilidad condicionada como una mezcla de distribuciones Von Mises-Fisher (VMF) sobre la hiperesfera tridimensional. Así, la red está entrenada para dar a su salida los coeficientes de la combinación lineal de las distribuciones en función de los vectores de entrada $\mathcal{F}, p$.

$$P(q) \approx \sum_i^n \alpha_i \mathcal{K}(q, q_i) \qquad \sum_i^n \alpha_i = 1 \tag{3.6}$$

$$P(q|\mathcal{F}, p) \approx \sum_i^n \alpha_i(\mathcal{F}, p; \theta)\mathcal{K}(q, q_i) \qquad \sum_i^n \alpha_i(\mathcal{F}, p; \theta) = 1 \tag{3.7}$$

A diferencia de la red de posición, esta red no produce todos los parámetros de las distribuciones μ, κ para sus respectivos kernels. En cambio, estos parámetros están fijos y se eligieron a priori. La razón de esto es que reconstruir los objetos que representan las distribuciones de VMF es laborioso y es considerablemente lento durante el entrenamiento, pues para calcular la constante de normalización $C_d(\kappa)$ se requiere del cálculo de funciones de Bessel, el cuál es una aproximación costosa.

El parámetro de concentración κ es igual para todas las distribuciones y se fijó con el valor de 320 mediante optimización bayesiana. Los centros del kernel $q_j = \mu_j$ se seleccionaron como los 256 puntos en la hiperesfera que mejor describen la distribución de probabilidad para todos los cuaterniones del dataset utilizando el método de Cuantificación Riemanniana por Aprendizaje Competitivo descrito en [43].

Al igual que en la red de posición, el objetivo de entrenamiento es la estimación por máxima verosimilitud, y el Negativo del logaritmo de la función de verosimilitud. Del inglés, Negative log likelihood (NLL) elegido como función de pérdida.

$$\mathcal{L}(\mathcal{D}, \theta) = \prod_{i=1}^{N} P(q_i | \mathcal{F}_i, p_i; \theta) \qquad \forall q_1, ..., q_N \in \mathcal{D} \tag{3.8}$$

$$E = -\log \mathcal{L}(\mathcal{D}, \theta) = -\sum_{i=1}^{N} \log \left(\sum_{k=1}^{n} \alpha_k(\mathcal{F}_i, p_i; \theta) \mathcal{K}(q_i, q_k) \right) \tag{3.9}$$

La arquitectura de este sistema es una red profunda de capas completamente conectadas cuya entrada es una capa de 1027 neuronas con 5 capas ocultas con 256 neuronas cada una, y cuya salida son 256 neuronas que forman los coeficientes de mezcla.

3.4. Recolección de datos y generación del dataset

Se creó un dataset $\mathcal{D} = \{\mathcal{C} = \{c_1, c_2, ...c_n\}, G = \{g_1, g_2, ...g_m\}\}$ de datos sintéticos construido a partir de la simulación del entorno de trabajo en el motor de física Gazebo. Este consiste en un conjunto de $n = 100,000$ nubes de puntos $c \in \mathbb{R}^{H \times W \times 3}$ y $m = 1,000,000$ propuestas de agarre $g = \{p, q | p \in \mathbb{R}^3, q \in \mathbb{H}_1\}$ asociadas a dichas nubes de puntos. Donde p es la posición 3-D de la base del gripper y q, la orientación del gripper representada por un cuaternión unitario relativos al marco de referencia de la base del robot.

La escena de agarre simulada se construyó como un objeto posicionado y orientado de manera aleatoria sobre un escritorio con el robot colocado directamente frente a este. Los objetos fueron seleccionados de un total de 20 objetos del conjunto de objetos YCB.

Seguido de esto, una nube de puntos parcial es capturada del sensor RGB-D que se encuentra en la cabeza del robot. Todos los puntos son transformados al marco de referencia de la base del robot. De esta manera, no se requiere usar una red de transformación. La nube de puntos resultante se recorta a la forma de entrada de la red, la cuál es una imagen de profundidad de $200 \times 200 \times 3$ centrada en el objeto. Si la nube de puntos no contiene al objeto, este se descarta y este proceso se repite.

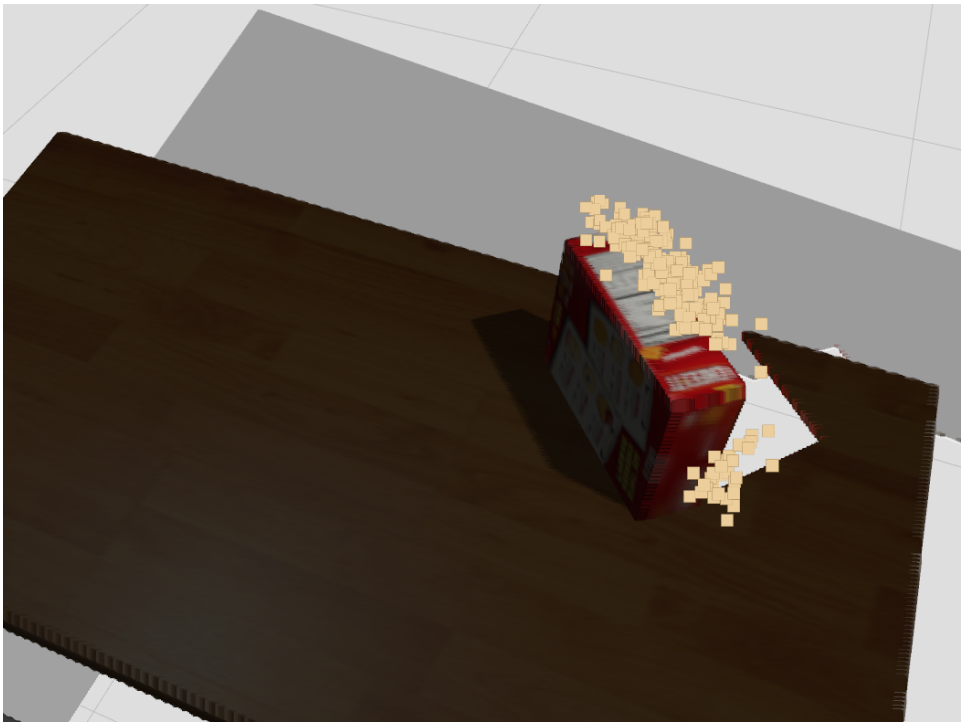


Figura 3.6: Ejemplo candidatos para muestras de agarre válidas

3.4.1. Restricciones geométricas

Se implementaron diferentes restricciones durante la captura de datos para reducir el ruido y la ambigüedad entre los ejemplos del dataset.

Debido a la condición 3.1.1, existen dos cuaterniones validos que representan la misma orientación. Por lo tanto para reducir la redundancia, la primer restricción consta de tomar las muestras cuya orientación q se encuentre en la semi-hiperesfera positiva.

$$G_1 \subset G_{\mathcal{D}} = \{g_i | q_{iw} \geq 0\}$$

La segunda restricción consta de tomar las muestras cuya altura z una vez transformada al marco de referencia de la base del robot sea mayor que la altura del escritorio más el radio del griper. Pues cualquier agarre debajo de este umbral chocará con la superficie de la tabla al intentar tomar el objeto.

$$G_2 \subset G_1 = \{g_i | z_i > z_{desk} + r_{gr}\}$$

La tercer restricción intenta disminuir el error debido a la oclusión de los objetos. Ya que solo se cuenta con la información de una nube de puntos parcial, y no se tiene una vista completa de la geometría de los objetos.

Entonces se busca evitar tomar aquellas muestras que estarían ocultas detrás de los objetos pues no se tiene suficiente información en estas áreas. Y dependiendo de orientación de la cámara y del objeto, podría ser difícil distinguir la profundidad de estos. Por ejemplo, discernir entre una lata de atún o una lata de sopa.

Entonces se intentó aproximar un modelo de eliminación por oclusión al restringir las muestras que se encuentran fuera de un cono con origen en el centro del objeto y un eje principal dado por el vector **a** entre el origen del objeto y la cámara en el marco de referencia del objeto. En la figura 3.7 se puede observar la zona de agarres válidos dentro del cono de visión. En esta imagen, los puntos no representan candidatos de agarre, simplemente es una representación de dicha zona donde los agarres se consideran válidos.

$$G_3 \subset G_2 = \{g_i | \frac{p_i}{|p_i|} \cdot \mathbf{a} > cos(70)\}$$

Figura 3.7: Ejemplo de restricción cónica

3.4.2. Dataset generado manualmente

El primer dataset utilizado para entrenar los primeros modelos de la red se creó al tomar manualmente los agarres de objetos en el simulador

del entorno. Esto se hizo al colocar cuidadosamente el gripper del robot en una posición tal que agarrase exitosamente el objeto y guardar un conjunto de poses $g = \{p, q\}$ del gripper relativa al centro del objeto en archivos comprimidos.

Una vez teniendo estos agarres previamente calculados, se colocaba el objeto sobre el escritorio con una pose aleatoria y se guardaban los agarres que cumplían las restricciones geométricas descritas anteriormente en 3.4.1 así como su nube de puntos asociada en archivos comprimidos.

A pesar de ser más rápido que tomar muestras en un ambiente real, este método seguía siendo muy laborioso y tardado. Además, al ser archivos individuales el tamaño que ocupaba en disco y el tiempo de acceso de los datos eran muy altos, por lo que se decidió no continuar con este método y buscar por alternativas.

3.4.3. Adaptación de dataset externo

Para el segundo dataset, se consideró usar o adaptar algún dataset de propuestas de agarres ya existente y públicamente accesible. Se consideraron varias opciones. Entre ellas, [27] y [44] los cuales son datasets de imágenes RGB-D con anotaciones de agarres, ambos adecuados para el entrenamiento de la red.

Al final se optó por utilizar el dataset A billion ways to grasp [45], el cuál es un dataset de millones de poses de agarre exitosos relativos al centro de masa de 21 objetos del conjunto de objetos YCB para el gripper Franka Panda 3. Se eligió este dataset porque las poses de agarre están representadas como la posición x, y, z con orientación como cuaterniones unitarios. La representación exacta de la salida del modelo. Además, al estar descritos los agarres relativos al centro de masa del objeto, adaptar el dataset al algoritmo de recolección de agarres fue relativamente sencillo.

Cabe mencionar que este dataset no cuenta con nubes de puntos asociadas, por lo que para generar el dataset se requirió pre-procesar los agarres para asociarlo a nubes de puntos. Para esto, y de manera similar al primer método de generación de datos, se tomaron de manera aleatoria 1000 agarres del dataset de A billion ways to grasp para el objeto correspondiente. Posteriormente, Se transforman las poses de agarre al marco de referencia de la base del robot y se aplican las restricciones descritas en 3.4.1. Al final de este proceso se toman 10 muestras de agarre de manera aleatoria con su nube de puntos asociada y se guardan los datos.

Para acelerar el tiempo de recolección de datos y posteriormente, el tiempo de lectura de datos en el entrenamiento, se guardaron los datos en una base de datos SQL. Pues permite mejorar el tiempo de acceso y disminuir el espacio del dataset en el disco. El tamaño final del archivo para el millón de agarres fue de 96.4 GB.

3.5. Entrenamiento de la red

Para entrenar la red se utilizó el ecosistema para aprendizaje automático PyTorch con aceleración CUDA en un sistema con un procesador AMD Ryzen 9 5900X de 12 núcleos y una tarjeta gráfica NVIDIA GeForce RTX 3070 Ti.

A continuación se detalla el proceso de entrenamiento del sistema. Así como sus limitaciones y ajustes que se realizaron para llevarlo a cabo.

3.5.1. Procesamiento de datos

Durante el proceso de entrenamiento se observó que gran parte del tiempo de entrenamiento consistía en leer y cargar los ejemplos del dataset a la RAM, convertirlo a tensores de PyTorch y luego cargar los lotes a la memoria de video de la tarjeta gráfica.

Entonces, para acelerar el proceso de entrenamiento y aprovechar mejor los recursos del sistema, se utilizó una base de datos SQL con trabajadores paralelos para pre-procesar los datos. De este modo se realiza la agrupación de ejemplos y construcción de los lotes de manera paralela para así reducir la carga del proceso principal.

Además, se utilizaron los métodos PyTorch y el API de CUDA para inicializar los tensores directamente en una sección de la memoria que se configura como no paginable para evitar la fragmentación de los tensores en la memoria virtual. Lo cual ralentiza considerablemente el proceso de carga pues se necesita reconstruir el tensor en la memoria no paginable (fijada) para poder enviarla al dispositivo CUDA.

Al cargar los lotes directamente en la memoria no paginable se reduce la sobrecarga de acceso a memoria. Además, al ser creados de manera contigua, se puede utilizar el método de copiado sin bloqueo. Esto permite al sistema utilizar la carga asíncrona de los lotes de la memoria RAM a la tarjeta gráfica sin interrumpir el proceso principal.

Es importante mencionar que el uso de la memoria fijada es un proceso intensivo que bloquea una parte considerable de la memoria RAM para ser utilizable. Por lo tanto, no se puede configurar una zona fijada demasiado grande, pues también se utiliza para el kernel del sistema y el resto de programas. Por esto mismo, tampoco se pueden utilizar tamaños de lotes muy grandes, ya que su utilización de memoria se incrementaría proporcionalmente.

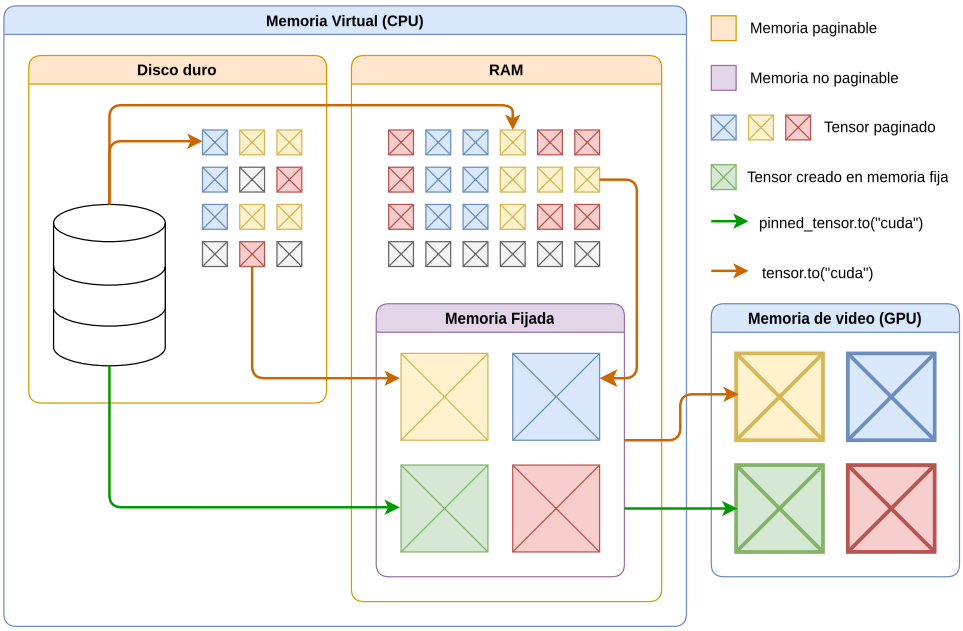


Figura 3.8: Representación de la carga de ejemplos de entrenamiento entre los dispositivos de entrenamiento

3.5.2. Parámetros de entrenamiento

Las tres secciones del sistema se entrenaron de diferente manera y con diferentes configuraciones de entrenamiento. Se utilizó el entorno de optimización bayesiana Optuna para explorar el espacio de hiperparámetros de entrenamiento para las redes de posición y orientación y buscar aquellos que mejoraran el rendimiento del modelo. A continuación se detalla el proceso de entrenamiento de cada red.

Codificador convolucional

Primero se entrenó la RNC como un auto-codificador para aprender el vector de características $\mathcal{F}$ que mejor describe las nubes de puntos de entrada. Esto se hace al entrenar la red en dos secciones consecutivas e inversas. La primera siendo un codificador que comprime la información de la nube de puntos en un vector de características $\mathbb{R}^{1024}$ y posteriormente se alimenta a un decodificador con la arquitectura inversa del codificador que intenta reconstruir la imagen de entrada a partir del vector de características.

Esta red se entrenó mediante Estimación adaptativa de momento con regularización de pesos desacoplado. Del inglés Adaptive Moment Estimation with decoupled Weight decay regularization (ADAMW) con una constante de regularización de pesos $\lambda = 0.0002$ y una taza de aprendizaje $\iota = 0.0001$ durante 80 épocas con el objetivo de minimizar el EMC entre la imagen de

entrada y la salida reconstruida del modelo. En la figura 3.9 se muestra la gráfica de entrenamiento. Donde la línea azul indica el error de entrenamiento y la línea roja indica el error de validación.

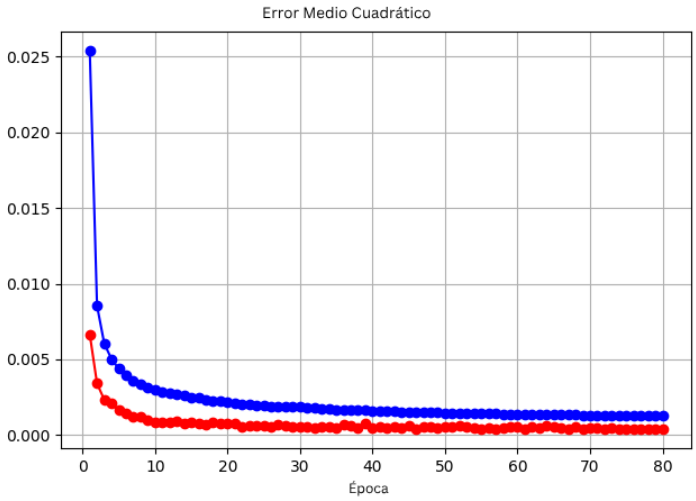


Figura 3.9: Gráfica de entrenamiento del codificador convolucional como auto-codificador

Una vez finalizado el proceso de entrenamiento, se guardó el modelo del codificador y decodificador. Sin embargo, para la implementación del sistema final sólo se utilizó el codificador.

3.5.3. Red de orientación como modelo de regresión geodésico

Esta red se entrenó con el método de optimización ADAMW con una constante de regularización de pesos $\lambda = 0.002$ y una taza de aprendizaje $\iota = 0.0001$ durante 120 épocas con el objetivo de minimizar el Error geodésico medio cuadrático (EMGC) entre la imagen de entrada y la salida reconstruida del modelo.

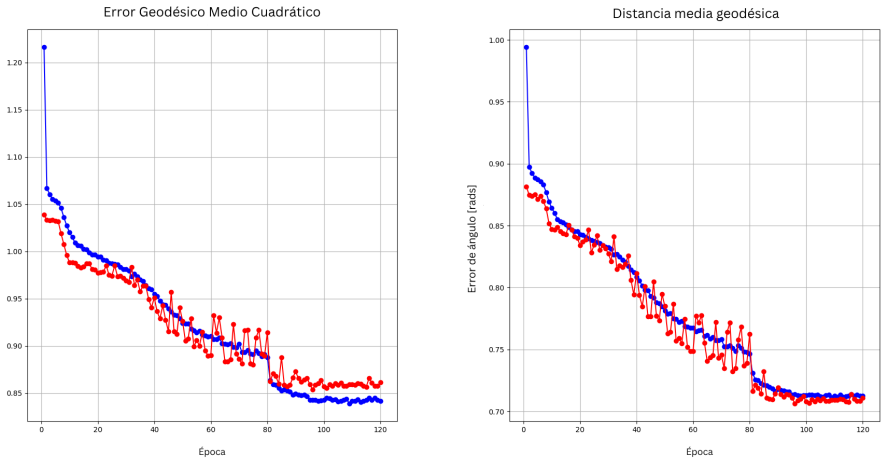


Figura 3.10: Gráfica de entrenamiento de la red de orientación como modelo de regresión geodésico

Como se puede observar en la figura 3.10, el error mínimo geodésico promedio al finalizar el entrenamiento fue de $0.7[rad]$ o aproximadamente 40°. A pesar de representar una mejora respecto al modelo entrenado por EMC tradicional, este modelo todavía presentó un error de ángulo considerable. Y lo cual se tradujo a una exactitud de agarre baja durante pruebas preliminares.

Partiendo de este resultado, la hipótesis de por qué este método se aproximó pero no pudo describir satisfactoriamente la orientación de agarre para el gripper es que, al igual que las posiciones de agarre, existen múltiples soluciones válidas para los mismos valores de entrada. Es decir, que la distribución de orientaciones no es unimodal, y por lo tanto no puede ser modelado por un único modelo de regresión geodésico.

Por este motivo ya no se continuó el desarrollo de este modelo y se buscó un modelo alternativo que pudiera describir mejor este comportamiento multimodal. Entre estos, se exploró la posibilidad de editar este diseño a un modelo Bin & Delta detallado en [46] y posteriormente un modelo de mezclas.

Bin & Delta es un modelo que junta los modelos de clasificación discretos (bin) y los modelos de regresión geodésico (delta). En este enfoque, se entrena la red para aprender una orientación discreta con mayor probabilidad q_{bin} entre un número predefinido de orientaciones y además, una corrección δ como un vector tangente para cada punto q_{bin} tal que al aplicar el mapa exponencial, se pueda aproximar cualquier orientación. De este modo, aprovechando los beneficios de ambos enfoques.

Sin embargo, este método no se utilizó porque se incrementaría la complejidad del modelo considerablemente debido a la necesidad de entrenar como mínimo, dos redes más. Estas siendo la red de clasificación y la red

delta. Y que, de acuerdo a los autores se recomienda una red delta por cada elemento entre el conjunto de valores discretos. Además, se requeriría clasificar cada orientación del dataset respecto al cuaternión discreto más próximo a este mediante algún método de agrupamiento como k-medias, lo cuál es un proceso intensivo. Finalmente se optó por utilizar un modelo de mezclas de VMF para aproximar la distribución de las orientaciones pues únicamente se necesitaría entrenar una red y no se necesita extender el dataset con nuevas anotaciones.

Red de posición y orientación como modelos de mezclas

Para entrenar estas redes se entrenaron de manera conjunta ya que su salida depende de la misma entrada, el vector de características $\mathcal{F}$. Sin embargo durante la fase entrenamiento se cortó la conexión entre la red de posición y orientación. Es decir, para poder entrenar la red no se alimentó la salida de la red de posición directamente a la red de orientación como se utilizaría en el modo de inferencia final.

En este caso la entrada p de la red de orientación se toma directamente de los ejemplos de entrenamiento. Ya que al trabajar con modelos probabilísticos, no es posible asegurar que la salida de la red de posición sea igual a la de la posición de la cual depende la orientación objetivo. Además, realizar el muestreo aleatorio podría interferir en el cálculo de la retropropagación, pues es un elemento que no depende de ningún parámetro de la red.

Para entrenar estas redes se utilizó el método de optimización ADAMW. Los valores de configuración fueron encontrados mediante optimización bayesiana realizando 200 pruebas de ensayo con diferentes propuestas de hiperparámetros $\iota_{max}, \kappa, \lambda, m$. Una vez realizadas las pruebas, se guardaron los 3 modelos con menor error de validación. A continuación se muestran los detalles de la búsqueda.

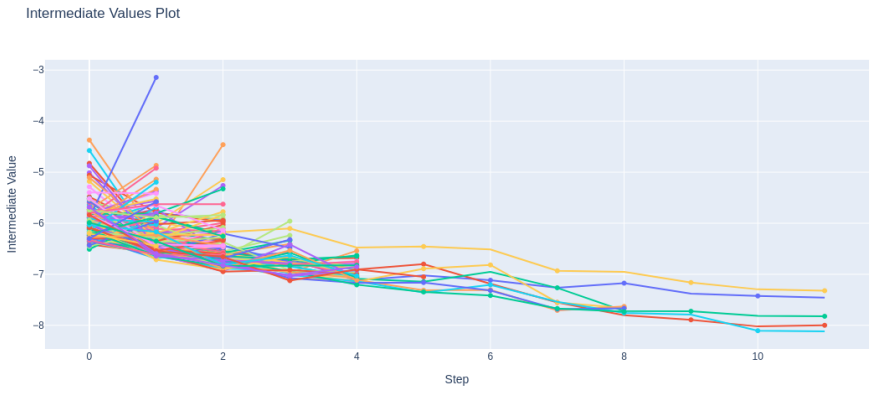


Figura 3.11: Gráfica de ensayos y su error de validación

La figura 3.11 muestra el error de validación de cada ensayo por época de entrenamiento. Entre los cientos de ensayos, estos se podan si su error de validación se encuentra dentro del percentil 75 para cada época. Así evitando desperdiciar recursos de cómputo en modelos con bajo prospecto de mejoría.

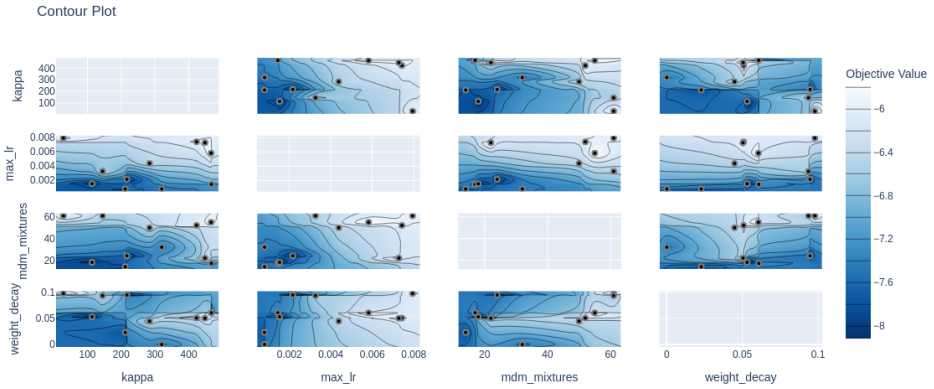


Figura 3.12: Gráfica de contornos entre los parámetros de búsqueda

La gráfica de contornos 3.12 muestra la relación entre las variables de estudio y su efecto sobre el valor objetivo. En este caso, sobre su efecto en la minimización del error de validación. Por ejemplo, analizando del mapa de contorno (0,2) se puede concluir que los mejores modelos son aquellos con parámetros de concentración κ y un número de mezclas bajos. Además, observando las columnas y renglones de la taza de aprendizaje, estas aparentan ser un gradiente pseudo-lineal que decrece proporcionalmente a medida que este aumenta, independientemente de su covariable. Esto es debido a que este parámetro tiene una importancia muy alta comparado con los otros hiperparámetros.

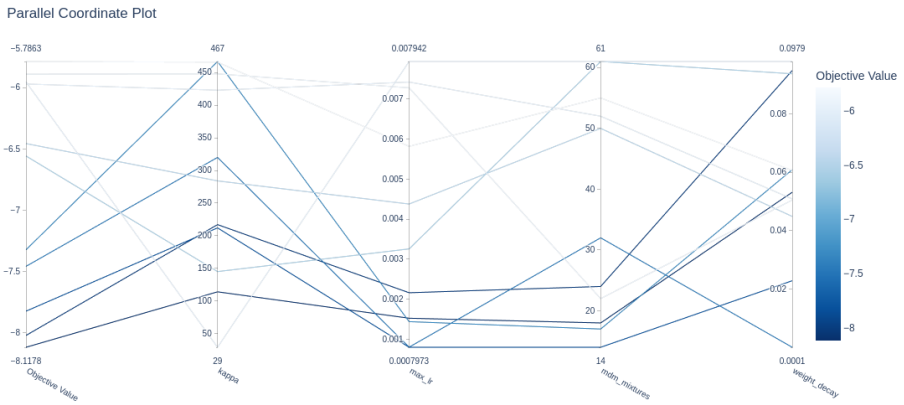


Figura 3.13: Gráfica de coordenadas paralelas entre los parámetros de búsqueda

La gráfica de coordenadas paralelas permite visualizar el panorama de hiperparámetros al mostrar el intervalo óptimo de los hiperparámetros en una escala proporcional para cada valor de interés. En esta figura 3.13 se puede observar que el intervalo óptimo para el número de mezclas m es entre 14 y 32, que la taza de aprendizaje máxima ι se encuentra entre 0.0008 y 0.002, el parámetro de concentración $120 < \kappa < 450$, y la constante de regularización de pesos $0.0001 < \lambda < 0.006$.

La configuración del modelo con mejor exactitud se entrenó con constante de regularización de pesos $\lambda = 0.0001$, taza de aprendizaje máxima $\iota = 0.0008$, número de mezclas gaussianas $m = 32$, $\kappa = 320$ utilizando el planificador de taza de aprendizaje OneCycleLR. [47]

Cerrando este capítulo se describió a detalle la filosofía de diseño y las decisiones tomadas fundamentadas con la teoría descrita en el capitulo 2. Se explicó con detenimiento las decisiones de ingeniería para desarrollar los modelos así como los obstáculos que se presentaron. En el siguiente capítulo se explorará el resultado de la implementación de todos los algoritmos de entrenamiento y de las arquitecturas propuestas. Se pondrá a prueba estos modelos para evaluar su desempeño y observar su comportamiento.

Capítulo 4

Resultados

En este capítulo se detallan los experimentos realizados y el entorno de pruebas construido para evaluar el desempeño del sistema y compararlo con el funcionamiento del sistema implementado actualmente. También se realiza una comparación del uso de recursos antes y después de la optimización paralela con base de datos SQL. Y finalmente, una breve comparación entre los modelos con optimización bayesiana y un modelo con hiperparámetros propuestos de manera heurística.

4.1. Experimentos

Para evaluar el desempeño de la red, se diseñó una prueba para medir su tiempo de respuesta y su exactitud de agarre.

En primer instancia se calcula el tiempo promedio de respuesta. Es decir, se mide cuánto tiempo le toma al sistema generar una propuesta de agarre una vez proporcionada una nube de puntos válida. En este escenario no se mide el tiempo de pre-procesamiento de la nube de puntos.

La segunda parte de la prueba está diseñada para medir la exactitud de los agarres propuestos. Donde se realizan 200 pruebas de agarre por objeto con pose aleatoria y se evalúa el porcentaje de agarres exitosos entre el total de pruebas.

4.1.1. Entorno de pruebas

Se desarrolló un entorno de pruebas simulado en el motor de física Gazebo integrado con ROS para medir el tiempo de respuesta promedio y la exactitud de las propuestas de agarre por objeto.

De manera similar al entorno de entrenamiento creado para generar el dataset de agarres, el ambiente de pruebas consiste en el robot de servicio Justina posicionado frente a un escritorio con un objeto reposando sobre la superficie de este. Y el cuál se coloca con una posición y orientación aleatorias

relativas a la base del robot.

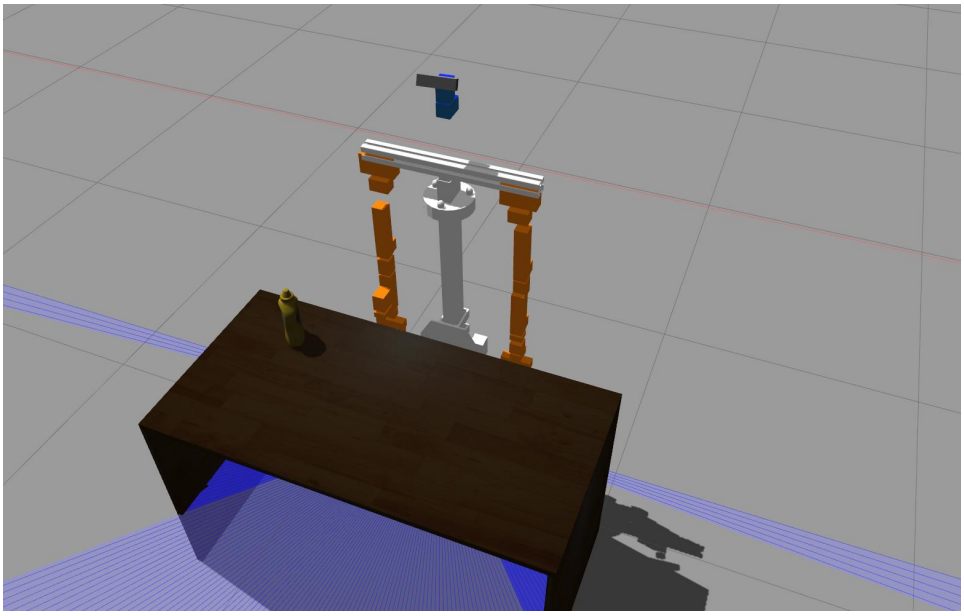


Figura 4.1: Entorno de pruebas simulado

En esta escena, el robot captura una nube de puntos centrada en el objeto y se recorta a las dimensiones de entrada de la red. Una vez que está lista, se inicia un temporizador y se entrega la nube de puntos a la red. Una vez que el sistema finaliza de procesar la nube de puntos y entrega a su salida la propuesta de configuración de agarre, se detiene el temporizador y se guarda el tiempo de respuesta.

Para medir la exactitud de las propuestas de agarre primero se posiciona el gripper con la configuración propuesta por el sistema con los dedos abiertos. Una vez que la base del gripper se encuentra en la posición y orientación deseada, lentamente se cierran los dedos del gripper.

En las terminales de los dedos del gripper se encuentran sensores de contacto que detectan colisiones entre estos y otros cuerpos. Entonces, se define como un agarre exitoso si al finalizar el movimiento de cerrar los dedos existe un contacto con el objeto con ambos dedos del gripper. Para cualquier otro caso se define como un agarre fallido. Cabe mencionar que una vez tomado el objeto no se realizan mediciones de la calidad del agarre. Simplemente se efectúa una evaluación binaria si se agarró el objeto o no.

Esta prueba se repite 200 veces para cada objeto en el conjunto de objetos de entrenamiento y para otros 4 objetos nuevos que no se encuentran en el dataset. La exactitud de agarre se define como el porcentaje de agarres exitosos entre el número de agarres totales. Los resultados de estas pruebas se muestran y discuten en la siguiente sección.

4.2. Resultados de las pruebas de agarre

Para estos experimentos sólo se realizaron pruebas con el modelo de mezclas de VMF. Durante una evaluación inicial se observó que el modelo de regresión geodésico tenía un error de ángulo medio de aproximadamente 40°, lo cual se tradujo a una exactitud de agarres baja. Posiblemente debido a que la distribución de las orientaciones de agarre no es unimodal. Por este motivo, no se incluyó este modelo para las pruebas.

Al finalizar los experimentos se observó que el sistema tiene un tiempo de respuesta promedio de 0.134s. Y una exactitud de agarre de aproximadamente 65 % para objetos con simetrías simples. A continuación se muestran las tablas de exactitud de agarre por objeto y por forma geométrica.

Tabla 4.1: Exactitud de agarre por objeto

Objeto YCB	**Forma del objeto**	**Exactitud[%]**
001_chips_can	Cilíndrico	79.5
002_master_chef_can	Cilíndrico	71
003_cracker_box	Caja grande	70.5
004_sugar_box	Caja grande	64.5
005_tomato_soup_can	Cilíndrico	69.5
006_mustard_bottle	Botella plana	69
007_tuna_fish_can	Cilíndrico	59
008_pudding_box	Caja pequeña	67.5
009_gelatin_box	Caja pequeña	72.5
010_potted_meat_can	Caja pequeña	68.5
011_banana	Otro	53
019_pitcher_base	Cilíndrico	51.5
021_bleach_cleanser	Botella plana	44
024_bowl	Otro	62.5
025_mug	Cilíndrico	58
035_power_drill	Otro	26
036_wood_block	Caja grande	78
037_scissors	Herramienta	22.5
040_large_marker	Herramienta	21.5
048_hammer	Herramienta	9
051_large_clamp	Herramienta	11.5
052_extra_large_clamp	Herramienta	24
056_tennis_ball	Esférico	60
061_foam_brick	Caja pequeña	67.5
077_rubiks_cube	Cubo	70.5

Tabla 4.2: Exactitud de agarre por forma geométrica

Forma del objeto	**Exactitud[%]**
Cilíndrico	64.75
Esférico	60
Caja grande	71
Caja pequeña	69
Cubo	70.5
Botella plana	56.5
Herramienta	17.7
Otro	47.166

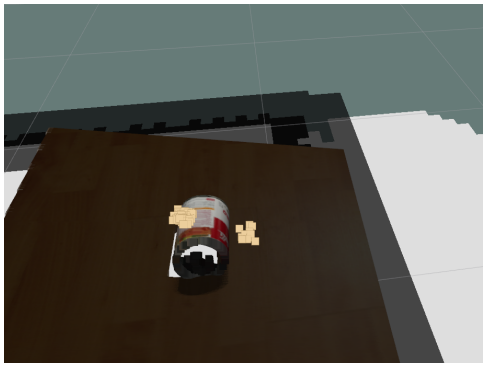


Figura 4.2: Distribución de agarres aproximada para el objeto 005_tomato_soup_can

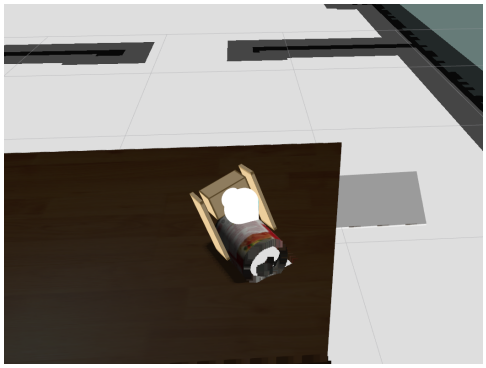


Figura 4.3: Ejemplo de agarre para el objeto 005_tomato_soup_can

4.2.1. Comparación con el sistema actual

El sistema actual está implementado como un sistema de planeación de agarres basado en PCA detallado en [24]. Este sistema utiliza métodos de procesamiento de imágenes robustos para filtrar la imágen RGB-D de la nube de puntos y segmentar los puntos que pertenecen al objeto con detección de contornos y la librería de YOLO. Se anexa la tabla 4.3 que detalla el porcentaje de agarres exitosos para algunos objetos del conjunto YCB. Todas estas figuras fueron tomados de [24].

Tipo	Objeto	Dimensiones [m]	Intentos	tipo de agarre	Agarres exitosos	tasa %
prismático	pringles	0.2x0.08x0.08	5	lateral	3	60
prismático	pringles	0.2x0.08x0.08	5	superior	5	100
caja	jugo	0.1x0.05x0.03	5	superior	4	80
cubico	taza	0.06x0.07x0.06	5	superior	3	60
prismático	lata de refresco	0.07x0.07x0.06	5	superior	4	80
esferoide	manzana	0.07x0.06x0.06	5	superior	3	60
caja	caja de cereal	0.3x0.2x0.05	5	lateral	4	80
prismático	lata de conservas	0.012x0.07x0.07	5	superior	4	80
box	jamón en lata	0.07x0.12x0.06	5	superior	4	80
prismático	copa	0.23x0.09x0.09	5	lateral	4	80
cuboide	pera	0.08x0.06x0.05	5	superior	4	80
prismático	mostaza	0.13x0.07x0.04	4	lateral	3	60
prismático	desinfectante	0.25x0.1x0.1	5	lateral	5	100
prismático	desodorante	0.1x0.05x0.05	5	superior	3	60

Tabla 4.3: Tasa de agarres para distintos objetos del dataset.

Comparando la tablas 4.2 y 4.3 para geometrías similares se puede observar que el sistema actual tiene una tasa de agarres exitosos ligeramente mayor que el sistema implementado mediante redes neuronales y modelos probabilísticos. Sin embargo, este sistema no hace uso de técnicas de procesamiento de imágenes para la detección de contornos y segmentación con modelos de visión YOLO como el sistema de PCA. Por lo tanto, a pesar de ser un sistema más simple, este es considerablemente más robusto.

4.2.2. Optimización de recursos

A continuación se muestra una comparación de la utilización de recursos antes y después de la optimización de memoria con base de datos relacional. Esto se realiza al comparar las gráficas de fase entre el porcentaje de

utilización del GPU y de memoria. Lo cuál corresponde a la fase de carga de datos y el procesamiento de estos. Este comportamiento se visualiza mediante la herramienta de comando Neat Videocard TOP (NVTOP) el cual es un monitor de tareas y recursos de software libre para tarjetas gráficas y aceleradores de cómputo.

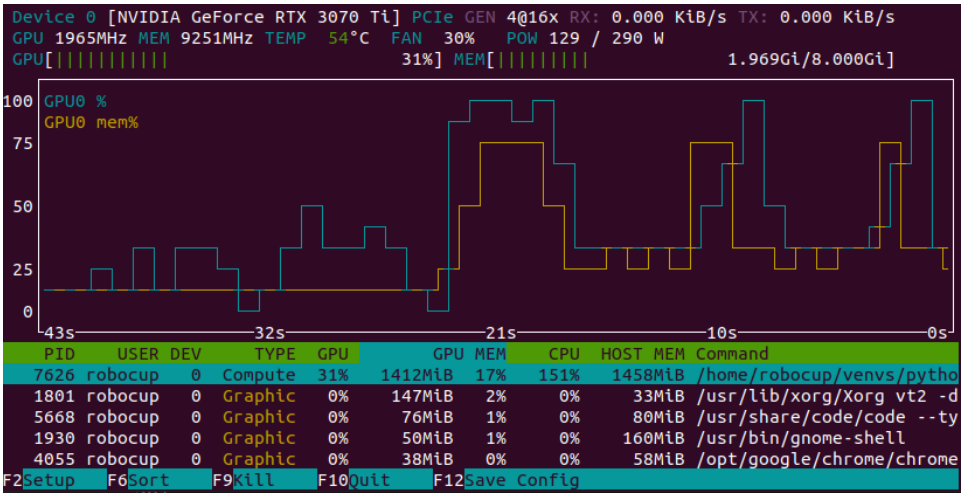


Figura 4.4: Gráfica de utilización de recursos sin optimización

En la figura 4.4 se puede observar que la utilización promedio del GPU se encuentra alrededor del 31 %. Esto debido a que la tarjeta gráfica tiene que esperar a que el procesador cargue los datos dispersos en la memoria virtual a la memoria RAM y posteriormente a la memoria de video. Como se observa en la gráfica de utilización de VRAM y GPU, los picos de utilización indican que existe un considerable retraso entre la carga y el procesamiento de los datos.

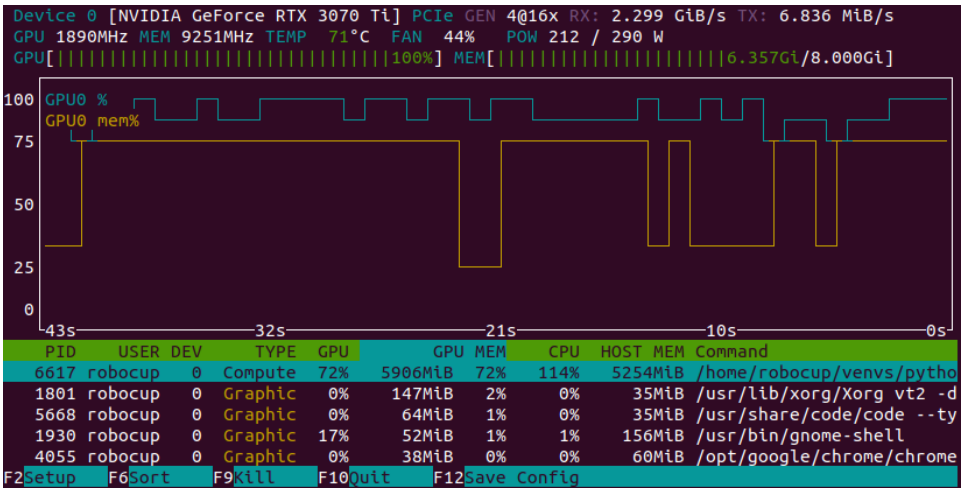


Figura 4.5: Gráfica de utilización de recursos con optimización paralela y base de datos relacional

En comparación, en la figura 4.5 se puede observar que al implementar la optimización con la base de datos relacional, y la carga directa de tensores a la memoria paginada con carga asíncrona, la utilización promedio de GPU incrementó hasta su carga máxima. Pues el transporte de datos a la memoria de video ya no representa un cuello de botella en su procesamiento. Por eso mismo se puede observar que tanto la utilización de VRAM como de GPU son constantes.

4.2.3. Optimización Bayesiana

Una vez diseñado el modelo de mezclas, inicialmente se probaron hiperparámetros de la red seleccionados de manera heurística. El primer modelo se diseño con una tasa de aprendizaje máximo de $\iota = 0.001$, coeficiente de concentración $\kappa = 100$, numero de mezclas gaussianas $m = 64$ y constante de regularización de pesos $\lambda = 0.0001$.

Sin previo conocimiento del rango óptimo de estos valores, este modelo inicial obtuvo un rendimiento bajo aunque considerablemente mejor que el modelo de regresión geodésico. Este modelo inicial obtuvo un error de validación de -3.893 después de entrenar por 30 épocas.

En comparación con el mejor modelo entrenado a partir de un proceso de optimización bayesiana, este obtuvo un error de validación mucho menor, alcanzando -8.117 y representando una mejoría de un poco más del doble. Posiblemente debido a que el parámetro de concentración κ del primer modelo era demasiado disperso, y que no se necesitaban tantas mezclas de gaussianos para aproximar sus respectivas distribuciones de probabilidad. Pero esta información se desconocía previo a su implementación.

Con esto concluye el capítulo de resultados, donde se realizó la evaluación de los modelos y se detalló su desempeño. En el siguiente capítulo se discutirán los hallazgos encontrados derivado de los resultados descritos en este capítulo. Se analizará los puntos fuertes de esta implementación de un planificador de agarres con redes neuronales así como sus desventajas y áreas de oportunidad.

Capítulo 5

Discusión

5.1. Conclusiones

Se entrenó un sistema de red neuronal el cuál es capaz de aproximar la distribución de probabilidad de los candidatos de agarre directamente de una nube de puntos parcial y tomar muestras de esta distribución con una exactitud razonable para la mayoría de los objetos.

Se logró construir un dataset de agarres para entrenar la red propio, aunque posteriormente se adaptó un dataset de agarres externos de manera satisfactoria. Encontrando que una estructuración de datos efectiva es fundamental para la aceleración del proceso de entrenamiento.

Del mismo modo se encontró que un manejo adecuado de los recursos de cómputo y una selección apropiada de hiperparámetros de entrenamiento pueden mejorar considerablemente el rendimiento de estos sistemas. Demostrando que es posible optimizar estos modelos sin necesidad de modificar la arquitectura de la red.

Se observó que la red obtuvo los mejores resultados al intentar agarrar objetos grandes y simétricos con una exactitud entre 60-70 %. Sin embargo, el sistema tuvo muchos problemas al intentar proponer agarres para objetos pequeños y con geometrías complejas como las herramientas. Una posible explicación es que las geometrías simétricas como la cilíndrica y rectangular están sobre-representadas en el dataset (casi la mitad de objetos). Además, para agarrar herramientas se requiere una mayor precisión de agarre y ser cuidadosamente calibrado a los parámetros del gripper. Y ya que se utilizó un dataset de agarres externo, este no está adecuado a la configuración del gripper del robot.

Comparando el desempeño de este sistema con el implementado actualmente, se encontró que tienen una exactitud similar aunque ligeramente menor. Cabe mencionar que el sistema actual requiere de técnicas de procesamiento de imágenes robustas como segmentación de objetos, detección de contornos y filtrado ya que este método es muy sensible al ruido y valo-

res atípicos. En ese aspecto, la arquitectura de red neuronal puede alcanzar exactitudes similares sin necesidad de estos métodos ya que solo necesita un cuadro delimitador alrededor del objeto, incluso con ruido de fondo, pues al utilizar modelos probabilísticos, se toma en cuenta la incertidumbre de los datos.

5.2. Trabajo futuro

El trabajo a futuro se enfocará en mejorar el rendimiento del sistema al entrenar la red con datos de mayor calidad. Principalmente al mejorar el rango de geometrías entre los objetos y con una distribución menos sesgada. También, buscando una manera de adaptar el dataset de agarres para las especificaciones del gripper y así disminuir el error con herramientas y objetos pequeños.

Además se busca adaptar el sistema para implementarlo y probarlo en un ambiente real. Posiblemente usando métodos de adaptación de dominio, o al trabajar con modelos probabilísticos, usar métodos de inferencia bayesiana para reducir la brecha entre el entorno simulado y el real.

Finalmente se busca completar la implementación del modelo de Bin & delta [46] para realizar una comparación con el modelo de mezclas.

Bibliografía

[1] K. M. Lynch y F. C. Park, *Modern robotics: mechanics, planning, and control*. Cambridge: Cambridge university press, 2017, ISBN: 978-1-107-15630-2.

[2] J. H. Gallier, *Geometric methods and applications: for computer science and engineering* (Texts in applied mathematics 38), 2nd ed. New York: Springer, 2011, 680 págs., ISBN: 978-1-4419-9960-3 978-1-4419-9961-0.

[3] T. M. Mitchell, *Machine learning* (McGraw-Hill series in Computer Science), Nachdr. New York: McGraw-Hill, 2013, 414 págs., ISBN: 978-0-07-042807-2 978-0-07-115467-3.

[4] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever y R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, Version Number: 1, 2012. DOI: 10.48550/ARXIV.1207.0580. dirección: https://arxiv.org/abs/1207.0580 (visitado 19-05-2025).

[5] S. Ioffe y C. Szegedy, «Batch normalization: accelerating deep network training by reducing internal covariate shift,» en *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ép. ICML'15, Lille, France: JMLR.org, 6 de jul. de 2015, págs. 448-456. (visitado 21-05-2025).

[6] Z. Li, F. Liu, W. Yang, S. Peng y J. Zhou, «A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,» *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, n.º 12, págs. 6999-7019, dic. de 2022, ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2021.3084827. dirección: https://ieeexplore.ieee.org/document/9451544/ (visitado 02-06-2025).

[7] J. W. Robbin y D. A. Salamon, *Introduction to Differential Geometry* (Springer Studium Mathematik (Master)). Berlin, Heidelberg: Springer Berlin Heidelberg, 2022, ISBN: 978-3-662-64339-6 978-3-662-64340-2. DOI: 10.1007/978-3-662-64340-2. dirección: https://link.springer.com/10.1007/978-3-662-64340-2 (visitado 18-05-2025).

[8] J. Solà, J. Deray y D. Atchuthan, *A micro Lie theory for state estimation in robotics*, Version Number: 9, 2018. DOI: 10.48550/ARXIV.1812.01537. dirección: https://arxiv.org/abs/1812.01537 (visitado 16-05-2025).

[9] C. Ley y T. Verdebout, *Modern directional statistics*. Boca Raton: Chapman & Hall CRC, 2017, ISBN: 978-1-4987-0664-3.

[10] A. SenGupta, *Directional Statistics for Innovative Applications: A Bicentennial Tribute to Florence Nightingale* (Forum for Interdisciplinary Mathematics Series), 1st ed, col. de B. C. Arnold. Singapore: Springer, 2022, 1 pág., ISBN: 978-981-19-1043-2 978-981-19-1044-9.

[11] K. V. Mardia y P. E. Jupp, eds., *Directional statistics*, Wiley series in probability and statistics, Chichester New York: J. Wiley, 2010, 429 págs., ISBN: 978-0-470-31697-9 978-0-470-31781-5.

[12] G. J. McLachlan y D. Peel, *Finite mixture models* (Wiley series in probability and statistics Applied probability and statistics section). New York: Wiley, 2005, ISBN: 978-0-471-72118-5 978-0-471-00626-8. DOI: 10.1002/0471721182.

[13] C. M. Bishop, «Mixture density networks,» Technical Report, Technical Report, Published: Technical Report, Birmingham, 1994. dirección: https://publications.aston.ac.uk/id/eprint/373/.

[14] F. Rossi y F. Barbaro, «Mixture of von Mises-Fisher distribution with sparse prototypes,» 2022, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2212.14591. dirección: https://arxiv.org/abs/2212.14591 (visitado 09-05-2025).

[15] A. Banerjee, I. S. Dhillon, J. Ghosh y S. Sra, «Clustering on the Unit Hypersphere using von Mises-Fisher Distributions,» *Journal of Machine Learning Research*, vol. 6, n.º 46, págs. 1345-1382, 2005. dirección: http://jmlr.org/papers/v6/banerjee05a.html.

[16] S. Gopal e Y. Yang, «Von mises-fisher clustering models,» en *Proceedings of the 31st International Conference on Machine Learning*, ISSN: 1938-7228, PMLR, 27 de ene. de 2014, págs. 154-162. dirección: https://proceedings.mlr.press/v32/gopal14.html (visitado 16-05-2025).

[17] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei y S.-H. Deng, «Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimizationb,» *Journal of Electronic Science and Technology*, vol. 17, n.º 1, págs. 26-40, 2019, ISSN: 1674-862X. DOI: https://doi.org/10.11989/JEST.1674-862X.80904120. dirección: https://www.sciencedirect.com/science/article/pii/S1674862X19300047.

[18] A. Mallasto y A. Feragen, «Wrapped Gaussian Process Regression on Riemannian Manifolds,» en *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, ISSN: 2575-7075, jun. de 2018, págs. 5580-5588. DOI: 10.1109/CVPR.2018.00585. dirección: https://ieeexplore.ieee.org/document/8578683 (visitado 09-05-2025).

[19] J. Görtler, R. Kehlbeck y O. Deussen, «A Visual Exploration of Gaussian Processes,» *Distill*, 2019. DOI: 10.23915/distill.00017.

[20] F. Nogueira, *Bayesian Optimization: Open source constrained global optimization tool for Python*, 2014. dirección: https://github.com/bayesian-optimization/BayesianOptimization.

[21] S. I. Ansary, S. Deb y A. K. Deb, «A novel object slicing based grasp planner for 3D object grasping using underactuated robot gripper,» *CoRR*, vol. abs/1907.09142, 2019. arXiv: 1907.09142. dirección: http://arxiv.org/abs/1907.09142.

[22] Q. Lei, G. Chen y M. Wisse, «Fast grasping of unknown objects using principal component analysis,» *AIP Advances*, vol. 7, n.º 9, pág. 095 126, sep. de 2017, _eprint: https://pubs.aip.org/aip/adv/article-pdf/doi/10.1063/1.4991996/1297964 ISSN: 2158-3226. DOI: 10.1063/1.4991996. dirección: https://doi.org/10.1063/1.4991996.

[23] B. Zapata-Impata, P. Gil, J. Pomares y F. Torres, «Fast Geometry-based Computation of Grasping Points on Three-dimensional Point Clouds,» *International Journal of Advanced Robotic Systems*, vol. 16, abr. de 2019. DOI: 10.1177/1729881419831846.

[24] B. I. González Jiménez, «Planeación de movimientos para manipulación de objetos en robots de servicio doméstico,» Tesis doct., 2024, 1 pág.

[25] G. Nandi, P. Agarwal, P. Gupta y A. Singh, «Deep Learning Based Intelligent Robot Grasping Strategy,» en *2018 IEEE 14th International Conference on Control and Automation (ICCA)*, 2018, págs. 1064-1069. DOI: 10.1109/ICCA.2018.8444265.

[26] F. H. Zunjani, S. Sen, H. Shekhar, A. Powale, D. Godnaik y G. C. Nandi, «Intent-based Object Grasping by a Robot using Deep Learning,» en *2018 IEEE 8th International Advance Computing Conference (IACC)*, 2018, págs. 246-251. DOI: 10.1109/IADCC.2018.8692134.

[27] J. Mahler, J. Liang, S. Niyaz et al., «Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics,» jul. de 2017. DOI: 10.15607/RSS.2017.XIII.058.

[28] V. Holomjova, A. J. Starkey, B. Yun y P. Meißner, «One-Shot Learning for Task-Oriented Grasping,» *IEEE Robotics and Automation Letters*, vol. 8, n.º 12, págs. 8232-8238, 2023. DOI: 10.1109/LRA.2023.3326001.

[29] S. Joshi, S. Kumra y F. Sahin, «Robotic Grasping using Deep Reinforcement Learning,» en *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020, págs. 1461-1466. DOI: 10.1109/CASE48305.2020.9216986.

[30] J. Duan, C. Ye, Q. Wang y Q. Zhang, «A Light-Weight Grasping Pose Estimation Method for Mobile Robotic Arms Based on Depthwise Separable Convolution,» *Actuators*, vol. 14, pág. 50, ene. de 2025. DOI: 10.3390/act14020050.

[31] J. Redmon y A. Angelova, «Real-time grasp detection using convolutional neural networks,» en *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, págs. 1316-1322. DOI: 10.1109/ICRA.2015.7139361.

[32] S. Revathi, M. Gupta, A. Das y M. Sambrani, «Object Grasping using Convolutional Neural Networks,» en *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, 2019, págs. 1-3. DOI: 10.1109/ViTECoN.2019.8899363.

[33] D. Song, C. H. Ek, K. Huebner y D. Kragic, «Task-Based Robot Grasp Planning Using Probabilistic Inference,» *IEEE Transactions on Robotics*, vol. 31, n.º 3, págs. 546-561, 2015. DOI: 10.1109/TRO.2015.2409912.

[34] D. Chen, V. Dietrich y G. von Wichert, «Precision grasping based on probabilistic models of unknown objects,» en *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, págs. 2044-2051. DOI: 10.1109/ICRA.2016.7487352.

[35] Y. Zhou, C. Barnes, J. Lu, J. Yang y H. Li, «On the Continuity of Rotation Representations in Neural Networks,» en *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA: IEEE, jun. de 2019, págs. 5738-5746, ISBN: 978-1-7281-3293-8. DOI: 10.1109/CVPR.2019.00589. dirección: https://ieeexplore.ieee.org/document/8953486/ (visitado 18-05-2025).

[36] W. Li, W. Naeem, J. Liu, D. Zheng, W. Hao y L. Chen, «PDQ-net: Deep probabilistic dual quaternion network for absolute pose regression on SE(3),» en *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, ISSN: 2640-3498, PMLR, 17 de ago. de 2022, págs. 1118-1127. dirección: https://proceedings.mlr.press/v180/li22b.html (visitado 01-06-2025).

[37] C. Li, J. Bai y G. D. Hager, *A Unified Framework for Multi-View Multi-Class Object Pose Estimation*, Version Number: 2, 2018. DOI: 10.48550/ARXIV.1803.08103. dirección: https://arxiv.org/abs/1803.08103 (visitado 09-05-2025).

[38] S. Mahendran, H. Ali y R. Vidal, *A Mixed Classification-Regression Framework for 3D Pose Estimation from 2D Images*, Version Number: 1, 2018. DOI: 10.48550/ARXIV.1805.03225. dirección: https://arxiv.org/abs/1805.03225 (visitado 09-05-2025).

[39] S. Heuveline, S. Said y C. Mostajeran, «Gaussian distributions on Riemannian symmetric spaces, random matrices, and planar Feynman diagrams,» 2021, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2106.08953. dirección: https://arxiv.org/abs/2106.08953 (visitado 09-05-2025).

[40] J. Liu, C. Liu, J. Q. Shi y T. Nye, *Wrapped Gaussian Process Functional Regression Model for Batch Data on Riemannian Manifolds*, 5 de sep. de 2024. DOI: 10.48550/arXiv.2409.03181. arXiv: 2409.03181[stat]. dirección: http://arxiv.org/abs/2409.03181 (visitado 09-05-2025).

[41] Z. Wang, X. Li, H. Ding y J. Q. Shi, *Intrinsic Gaussian Process Regression Modeling for Manifold-valued Response Variable*, Version Number: 2, 2024. DOI: 10.48550/ARXIV.2411.18989. dirección: https://arxiv.org/abs/2411.18989 (visitado 09-05-2025).

[42] A. Cholaquidis, F. Gamboa y L. Moreno, *Conformal inference for regression on Riemannian Manifolds*, Version Number: 1, 2023. DOI: 10.48550/ARXIV.2310.08209. dirección: https://arxiv.org/abs/2310.08209 (visitado 16-05-2025).

[43] A. L. Brigant y S. Puechmorel, *Optimal Riemannian quantization with an application to air traffic analysis*, 2018. DOI: 10.48550/ARXIV.1806.07605. dirección: https://arxiv.org/abs/1806.07605 (visitado 09-05-2025).

[44] H.-S. Fang, M. Gou, C. Wang y C. Lu, «Robust grasping across diverse sensor qualities: The GraspNet-1Billion dataset,» *The International Journal of Robotics Research*, 2023, Publisher: SAGE Publications Sage UK: London, England.

[45] C. Eppner, A. Mousavian y D. Fox, *A Billion Ways to Grasp: An Evaluation of Grasp Sampling Schemes on a Dense, Physics-based Grasp Data Set*, 11 de dic. de 2019. DOI: 10.48550/arXiv.1912.05604. arXiv: 1912.05604[cs]. dirección: http://arxiv.org/abs/1912.05604 (visitado 15-05-2025).

[46] S. Mahendran, M. Y. Lu, H. Ali y R. Vidal, *Monocular Object Orientation Estimation using Riemannian Regression and Classification Networks*, _eprint: 1807.07226, 2018. dirección: https://arxiv.org/abs/1807.07226.

[47] L. N. Smith y N. Topin, «Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates,» *CoRR*, vol. abs/1708.07120, 2017. arXiv: 1708.07120. dirección: http://arxiv.org/abs/1708.07120.