

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Mecatrónica



Desarrollo de un Sistema Motivacional para el Descubrimiento de Needs y Drives en Robots Autónomos

Informe de Proyecto de Graduación para optar por el título de
Ingeniero/a en Mecatrónica con el grado académico de Licenciatura

Diana Cerdas Vargas

A Coruña, 4 de agosto de 2025

Licencia



Este trabajo está licenciado bajo una
Licencia Creative Commons Atribución-NoComercial-SinDerivadas 4.0 Internacional.

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

A handwritten signature in black ink, appearing to read "Diana", with several horizontal strokes crossing through it.

Diana Cerdas Vargas

A Coruña, 4 de agosto de 2025

Céd: 2-0828-0762

INSTITUTO TECNOLÓGICO DE COSTA RICA
PROGRAMA DE LICENCIATURA EN INGENIERÍA MECATRÓNICA
PROYECTO FINAL DE GRADUACIÓN
ACTA DE APROBACIÓN

El profesor asesor del presente trabajo final de graduación, indica que el documento presentado por el estudiante cumple con las normas establecidas por el programa de Licenciatura en Ingeniería Mecatrónica del Instituto Tecnológico de Costa Rica para ser defendido ante el jurado evaluador, como requisito final para aprobar el curso Proyecto Final de Graduación y optar así por el título de Ingeniero(a) en Mecatrónica, con el grado académico de Licenciatura.

Estudiante: Diana Carolina Cerdas Vargas

Proyecto: Desarrollo de un Sistema Motivacional para el Descubrimiento de Needs y Drives en Robots Autónomos



Msc. Juan Carlos Brenes Torres

Asesor

Cartago, 4 de agosto de 2025

INSTITUTO TECNOLÓGICO DE COSTA RICA
PROGRAMA DE LICENCIATURA EN INGENIERÍA MECATRÓNICA
PROYECTO FINAL DE GRADUACIÓN
ACTA DE APROBACIÓN

Proyecto final de graduación defendido ante el presente jurado evaluador como requisito para optar por el título de Ingeniero(a) en Mecatrónica con el grado académico de Licenciatura, según lo establecido por el programa de Licenciatura en Ingeniería Mecatrónica, del Instituto Tecnológico de Costa Rica.

Estudiante: Diana Carolina Cerdas Vargas

Proyecto: Desarrollo de un Sistema Motivacional para el Descubrimiento de Needs y Drives en Robots Autónomos

Miembros del jurado evaluador



Dra. Gabriela Ortiz León

Jurado



Dra. Marta Vílechez Monge

Jurado

Los miembros de este jurado dan fe de que el presente proyecto final de graduación ha sido aprobado y cumple con las normas establecidas por el programa de Licenciatura en Ingeniería Mecatrónica.

Cartago, 4 de agosto de 2025

Resumen

Este trabajo presenta un sistema motivacional para robots autónomos que permite la generación y el aprendizaje de misiones (y sus correspondientes impulsos) alineadas con propósitos humanos. El sistema se integra dentro de la arquitectura cognitiva e-MDB y utiliza Modelos de Lenguaje de Gran Tamaño (LLMs) para interpretar instrucciones en lenguaje natural. Se emplean tres instancias distintas de LLM, cada una especializada en una función específica: alineación semántica, generación de misiones y modelado de impulsos motivacionales. Los experimentos realizados en el entorno de simulación Gazebo demostraron una alineación consistente entre el comportamiento del robot y los propósitos humanos, así como una alta confiabilidad en la validez de las misiones y los impulsos generados.

Palabras clave: robots autónomos, sistema motivacional, arquitecturas cognitivas, alineación humano-robot, Modelos de Lenguaje de Gran Tamaño (LLMs).

Abstract

This work introduces a motivational system for autonomous robots that allows the generation and learning of missions (and their associated drives) aligned with human purposes. The system is integrated into the e-MDB cognitive architecture, employing Large Language Models (LLMs) to interpret natural language instructions. The system employs three distinct LLM instances, each specialized for a specific function: semantic alignment, mission generation, and modeling of motivational drives. Experiments conducted within the Gazebo simulation environment demonstrated consistent alignment between robot behavior and human purposes, as well as high reliability in mission and drive validity.

Keywords: autonomous robots, motivational system, cognitive architectures, human-robot alignment, Large Language Models (LLMs).

A mis padres, quienes con su amor, apoyo incondicional y ejemplo me guiaron durante cada paso de esta carrera. Este logro es tanto mío como de ustedes.

Agradecimientos

En primer lugar, agradezco a mi familia, por creer en mí incluso en los momentos más difíciles, por darme la oportunidad de estudiar y soñar en grande.

A mis compañeros y compañeras de carrera, por acompañarme en este proceso, en los laboratorios, proyectos, las idas al comedor. Son esos los recuerdos que más voy a atesorar del TEC.

A mis profesores y profesoras, por compartir su conocimiento con dedicación y acompañar mi formación con compromiso.

Especialmente, al profesor Juan Carlos Brenes, por siempre ser mi apoyo a lo largo de la carrera. Por permitirme trabajar con él y por ser mi asesor en este proyecto. Así mismo, a los profesores Juan Luis Crespo y Esteban Arias, que me ayudaron con el proceso para que este TFG fuera posible.

También quiero agradecer al Grupo Integrado de Ingeniería (GII), por darme la oportunidad de desarrollar este proyecto en un área de investigación tan interesante como lo es la robótica cognitiva. Por todo el apoyo que me brindaron en estos 5 meses, ha sido una experiencia increíble trabajar acá.

A mis supervisores Richard Duro y Alejandro Romero, que me guiaron a lo largo de este proceso y me enseñaron muchísimas cosas sobre esta área de investigación.

Por último, pero no menos importante, a mis compañeros y compañeras de trabajo en Ferrol, por su calidez, apoyo y amistad, que hicieron de esta experiencia una etapa inolvidable en mi vida personal y profesional. Especialmente, les agradezco a Emanuel y Sergio, que no solo me apoyaron día a día, sino que se convirtieron en amigos muy cercanos.

Diana Cerdas Vargas

A Coruña, 4 de agosto de 2025

Índice general

Lista de Figuras	iii
Lista de Tablas	vi
Lista de símbolos y abreviaciones	ix
1 Introducción	1
1.1 Antecedentes del proyecto	1
1.2 Descripción del problema	2
1.3 Justificación	2
1.4 Síntesis del problema	3
1.5 Objetivos del proyecto	3
1.6 Estructura del documento	3
2 Marco Teórico	5
2.1 Robótica autónoma	5
2.2 Arquitecturas cognitivas en robótica	6
2.2.1 ¿Qué es una arquitectura cognitiva?	7
2.2.2 Tipos de arquitecturas cognitivas	8
2.2.3 Diferentes enfoques motivacionales para el descubrimiento y aprendizaje de objetivos	9
2.3 La arquitectura e-MDB	11
2.3.1 Componentes fundamentales de la arquitectura e-MDB	11
2.3.2 Nodos cognitivos de la arquitectura e-MDB	12
2.4 Sistema motivacional: necesidades, misiones e impulsos	14
2.4.1 Necesidades	15
2.4.2 Impulsos	15
2.5 Problema de alineación en la robótica autónoma	16
2.5.1 Subproblemas del alineamiento	17
2.6 Modelos de lenguaje grandes (LLMs)	19
2.7 Similitud semántica y embedding en modelos LLM	20
2.7.1 Cálculo de la similitud semántica	20
2.7.2 Interpretación	21
3 Metodología	22

3.1	Descripción general	22
3.2	Metodología adaptada para el proyecto	24
3.2.1	Contextualización de la temática del proyecto	24
3.2.2	Identificación de necesidades	24
3.2.3	Establecimiento de especificaciones y métricas	26
3.2.4	Generación de conceptos	28
3.2.5	Selección de conceptos	34
3.2.6	Pruebas de validación del concepto	39
4	Diseño conceptual del sistema	42
4.1	Diseño general del sistema	42
4.2	Sistema motivacional: alineación, generación de misiones y funciones de impulso.	43
4.2.1	Cantidad de instancias de LLMs	44
4.2.2	Elección del modelo	46
4.2.3	Configuración del modelo	48
4.2.4	Pruebas para el diseño del sistema	50
4.3	Lógica de implementación del sistema e interfaz de interacción	55
4.4	Integración con la arquitectura e-MDB	59
4.4.1	Adaptaciones a la implementación de la arquitectura e-MDB	59
4.5	Integración del sistema motivacional y la arquitectura con el simulador robótico	64
5	Resultados y Análisis	67
5.1	Evaluación del sistema de alineación y traducción	67
5.1.1	Prueba de evaluación del sistema de alineación de propósito.	68
5.1.2	Prueba de evaluación del generador de misiones	71
5.1.3	Prueba de evaluación del generador de funciones de impulso.	74
5.2	Evaluación de la integración del sistema motivacional con la arquitectura y el simulador	77
5.2.1	Descripción de la prueba	78
5.2.2	Análisis de los resultados de la prueba	78
5.3	Evaluación del sistema con el robot	79
5.3.1	Descripción de la prueba	80
5.3.2	Resultados generales obtenidos en las repeticiones del experimento	81
5.3.3	Análisis de los resultados generales	82
5.3.4	Resultados del sistema motivacional obtenidos en los experimentos	83
5.3.5	Análisis de los resultados del sistema motivacional en los experimentos finales	86
5.4	Análisis económico	87
5.4.1	Beneficios para el GII y el proyecto PILLAR-Robots	90
6	Conclusiones	92
6.1	Recomendaciones	93
A	Posibles combinaciones de conceptos	98

B	Solicitudes iniciales para la selección de conceptos	99
C	Solución final	103
D	Propósitos utilizados para la prueba de alineación de propósitos	104
E	Resultados de pruebas	106
F	Entregables del proyecto	107

Índice de figuras

2.1	Componentes fundamentales de la arquitectura e-MDB. Tomado de [1]. . . .	13
2.2	Esquema gráfico de la arquitectura e-MDB. Tomado de [1].	14
2.3	Organización jerárquica para el marco de propósito que logra una alineación triangular. Tomado de [2].	17
3.1	Etapas propuestas por la metodología de Ulrich y Eppinger para el desarrollo de productos [3].	22
3.2	Diagrama de caja negra para representar el problema por resolver. Elaboración propia	29
3.3	Descomposición funcional del problema por resolver. Elaboración propia. . .	29
3.4	Árboles de clasificación obtenidos a partir de la búsqueda externa e interna. Elaboración propia.	33
3.5	Diagrama de conceptos para el sistema de generación de misiones y funciones de impulso. Elaboración propia.	34
4.1	Diagrama de bloques de para el sistema de alineación y traducción de propósito. Elaboración propia.	43
4.2	Comparación de rendimiento promedio para diferentes modelos de LLM con un contexto de 16 mil <i>tokens</i> . Tomado de [4].	47
4.3	Comparación de rendimiento en una serie de pruebas estándar para obtener <i>benchmarks</i> para diferentes modelos de LLM. Tomado de [5].	48
4.4	Comparación de rendimiento en pruebas estándar (<i>benchmarks</i>) para diferentes modelos de OpenAI. Tomado de [6].	49
4.5	Diagrama de flujo de la lógica general seguida en el código de Python para la interfaz de interacción entre el usuario y los modelos LLMs. Elaboración propia.	56
4.6	Diagrama de flujo de la lógica seguida en Python para los diversos procesos del sistema. Elaboración propia.	57
4.7	Interfaz estilo caja de texto para interactuar con el sistema motivacional. Elaboración propia.	58
4.8	Esquema gráfico de la arquitectura e-MDB adaptada para incorporar el sistema motivacional diseñado. Elaboración propia.	60
4.9	Parámetros utilizados para la creación de la necesidad de alineación y los métodos relacionados con la clase que representa al nodo de necesidad en ROS2 y Python. Elaboración propia.	60

4.10	Parámetros utilizados para la creación de la función de impulso relacionada con la necesidad de alineación y los métodos relacionados con la clase que representa al nodo en ROS2 y Python. Elaboración propia.	61
4.11	Parámetros utilizados para la creación del nodo de ROS2 para la política de alineación en Python. Elaboración propia.	61
4.12	Diagrama de la arquitectura cognitiva al ejecutar la política de alineación. Elaboración propia.	62
4.13	Parámetros utilizados para la creación del nodo de ROS2 para cada función de impulso y los métodos relacionados a la clase en Python. Elaboración propia.	63
4.14	Diagrama de bloques de la interfaz entre el sistema robótico y la arquitectura cognitiva inicial y las adaptaciones hechas para incorporarlo al sistema motivacional. Elaboración propia.	65
5.1	Diagrama de cajas con los valores de similitud semántica de los propósitos de entrada y las descripciones de salida generadas por el sistema LLM. Elaboración propia.	70
5.2	Porcentaje de misiones válidas e inválidas generadas para cada propósito y en paréntesis se muestra el total de misiones. Elaboración propia.	73
5.3	Distribución de la cantidad de misiones válidas generadas por el sistema para resolver cada tarea. Elaboración propia.	73
5.4	Porcentaje de funciones de impulso válidas e inválidas generadas para cada propósito y en paréntesis se muestra el total de funciones para cada propósito. Elaboración propia.	76
5.5	Esquema ilustrativo de la política ejecutada para la verificación de la integración del sistema. Elaboración propia.	78
5.6	Evaluación de la función de impulso “cilindro en mano” para diferentes posiciones ordenadas de la más lejana a la más cercana al cilindro rojo. Elaboración propia.	79
5.7	Configuración del experimento en Gazebo para la evaluación del sistema. Elaboración propia.	80
5.8	Variación en la cantidad de iteraciones necesarias para alcanzar el propósito a lo largo de los intentos. El promedio suavizado y su intervalo de confianza del 95% se muestran en azul, mientras que el área morada representa el rango ideal de 2 a 3 iteraciones. Elaboración propia.	82
5.9	Gráfica de funciones de impulso generadas para el repetición 9 ejecutado en Gazebo. Elaboración propia.	84
5.10	Evaluación de las funciones de impulso (5.10a), objetivos que dan recompensa y la satisfacción del propósito logrado (5.10b) a lo largo de las iteraciones del repetición 9. Elaboración propia.	85
5.11	Comparación entre la evaluación de funciones de impulso (arriba) y los objetivos que dieron recompensa junto con la satisfacción del propósito (abajo), en dos rangos de iteraciones de la repetición 9.	85

B.1	Prompt utilizado para la prueba con una única instancia con el modelo <i>phi4:14b</i> . Elaboración propia.	100
B.2	Prompt utilizado para la parte de alineación para la prueba con dos instancias con el modelo <i>phi4:14b</i> . Elaboración propia.	101
B.3	Prompt utilizado para la parte de traducción de misiones y funciones de im- pulso para la prueba con dos instancias, utilizando el modelo <i>phi4:14b</i> . Elab- oración propia.	102

Índice de tablas

3.1	Niveles de importancia relativos para las necesidades.	25
3.2	Necesidades del proyecto y su importancia.	25
3.3	Métricas asociadas a las necesidades del proyecto.	26
3.4	Conceptos con mayor potencial.	35
3.5	Criterios de evaluación de los conceptos de acuerdo a las necesidades.	35
3.6	Matriz de selección de conceptos para la etapa de filtrado.	36
3.7	Matriz de evaluación ponderada de conceptos del sistema propuesto.	37
3.8	Comparación entre los simuladores RoboboSim y Gazebo según distintos criterios de implementación, con calificaciones entre 0 y 5 (donde 5 es la máxima puntuación).	38
3.9	Necesidades y métricas evaluadas en la prueba 1.	39
3.10	Necesidades y métricas evaluadas en la prueba 2.	40
3.11	Necesidades y métricas evaluadas en la prueba 3.	41
4.1	Percepciones iniciales para experimento de prueba de diseño.	51
5.1	Propósitos con bajo nivel de detalle para la tarea de clasificación de cilindros por color.	69
5.2	Estadísticas obtenidas a partir de la similitud semántica de los propósitos de entrada y salida para cada tarea.	70
5.3	Ejemplo de misiones generadas por el sistema para la tarea de clasificar cilindros por color.	72
5.4	Tasa de éxito de las repeticiones del experimento con OSCAR ejecutadas en Gazebo.	81
5.5	Misiones definidas, los pesos asignados y las funciones de impulso	83
5.6	Resumen de equipos utilizados en el proyecto	88
5.7	Costos proporcionales de los equipos en el proyecto	88
5.8	Costo aproximado del servidor del GII con precios actuales.	89
5.9	Recursos asociados al desarrollo del proyecto	90
A.1	Combinaciones posibles de módulos del sistema sin considerar el simulador	98
D.1	Propósitos poco detallados para la tarea de acomodar las manzanas según su color.	104
D.2	Propósitos poco detallados para la tarea de apilar los cilindros en el centro de la mesa por orden de tamaño.	105

Lista de símbolos y abreviaciones

Abreviaciones

e-MDB	Epistemic Multilevel Darwinist Brain
LLM	Large Language Model (Modelo de lenguaje de gran tamaño)
LOLA	Lifelong Open-ended Learning Autonomy (Autonomía intencionada de aprendizaje abierto y a lo largo de la vida)
LSTM	Long Short-Term Memory (Memoria de corto y largo plazo)
NLP	Natural Language Processing (Procesamiento de lenguaje natural)
ROS	Robot Operating System

Capítulo 1

Introducción

En la presente introducción se presenta el contexto del Trabajo Final de Graduación (TFG), el cual tiene un enfoque en la robótica cognitiva. En este caso, se abordará el problema de la alineación de propósitos en un robot autónomo para lograr el aprendizaje automático de misiones e impulsos. Luego, se plantean los objetivos del proyecto y por último, se incluye una descripción general de la estructura que seguirá el documento.

1.1 Antecedentes del proyecto

El proyecto se realizó en el Grupo Integrado de Ingeniería (GII) de la Universidade da Coruña (UDC), en España. El grupo fue fundado en 1999 y se caracteriza por su enfoque interdisciplinario en el desarrollo de soluciones innovadoras aplicadas a diversos sectores. Su trayectoria incluye más de doscientos proyectos de investigación [7].

El proyecto se enmarca dentro de las áreas de robótica e inteligencia computacional, y forma parte de una línea de investigación orientada al desarrollo de arquitecturas cognitivas para robots autónomos. Específicamente, se centra en la arquitectura desarrollada por el Grupo Integrado de Ingeniería (GII), llamada e-MDB (epistemic Multilevel Darwinist Brain) [1]. Esta es una propuesta que busca brindarle a los robots mecanismos de razonamiento y toma de decisiones más robustos, para que sean capaces de adaptarse de forma abierta y continua a lo largo de su vida operativa (*lifelong open-ended learning autonomy*) [1].

En este contexto, se busca implementar una solución que permita a los robots interpretar y alinear su comportamiento autónomo con propósitos definidos por humanos. De hecho, este alcance forma parte de uno de los objetivos principales del proyecto de investigación europeo PILLAR-Robots. En el cual participan grupos de investigación de las siguientes entidades: Universidade Da Coruña, Consiglio Nazionale Delle Ricerche, Sorbonne Universite, Athina, AI2Life, PAL Robotics [8].

1.2 Descripción del problema

En el ámbito de la robótica autónoma, los sistemas motivacionales permiten a los robots descubrir y alcanzar objetivos (goals) en diversos dominios, así como aprender habilidades (skills) a través de modelos de utilidad (utility models) o políticas (policies) que optimizan su comportamiento. Estos sistemas han demostrado ser efectivos para permitir que los robots adapten su comportamiento a entornos cambiantes y complejos. Sin embargo, la autonomía sin un propósito definido tiene limitaciones, ya que no puede garantizar que las tareas realizadas por los robots sean útiles para los usuarios o los diseñadores.

Para contextualizar al lector, a continuación, se van a definir una serie de conceptos importantes para comprender la temática del proyecto y el problema a resolver. La arquitectura e-MDB necesita tener definidas una serie de necesidades (needs) y funciones de impulso (drive functions) para que el robot pueda cumplir con un propósito específico. Existen dos tipos de necesidades: las necesidades cognitivas y las necesidades operacionales. Las necesidades cognitivas son objetivos intrínsecos independientes de cualquier dominio, es decir, el robot va a tener estas necesidades solamente por existir. Por otra parte, las necesidades operacionales son objetivos generales independientes de cualquier dominio que se relacionan con el propósito del humano. También son llamadas misiones (missions).

Por su parte, cada uno de estos tipos de necesidades tiene relacionada una función de impulso. Las funciones de impulso se definen como funciones matemáticas que cuantifican cuán lejos está el robot de satisfacer dichas necesidades o misiones. Para definir las funciones de impulso, es necesario utilizar señales medibles del espacio en el que opera el robot, las cuales se obtienen de los sensores.

En la arquitectura actual, las misiones y sus impulsos son definidas uno a uno por el diseñador, de acuerdo con el propósito que se desea dar al robot. Esto implica que cada vez que se quiera agregar un propósito nuevo, se deben crear una o varias misiones nuevas y funciones de impulso que las evalúen. Este proceso no solo limita el alcance de los experimentos que se pueden ejecutar, sino que también afecta la autonomía que se puede lograr e implica un esfuerzo y conocimiento técnico elevado para traducir cada propósito humano en variables computacionales específicas. Entonces, lo que se busca es generar una solución que tome el propósito humano en lenguaje de alto nivel y lo traduzca a misiones e impulsos válidos para la arquitectura cognitiva e-MDB.

1.3 Justificación

El GII, para el proyecto PILLAR-Robots, trabaja en extraer el concepto abstracto de "propósito" (purpose) y convertirlo en elementos implementables dentro de la arquitectura cognitiva. Con el objetivo de lograr brindarle a los robots una *autonomía intencionada de aprendizaje abierto y a lo largo de su vida (purposeful lifelong open-ended autonomy)*.

Entonces, el desarrollo de un sistema que permita probar el concepto de alineación con

los propósitos de un ser humano y que sea capaz de descubrir de forma autónoma sus propias misiones y funciones de impulso representa un avance significativo para el GII y el proyecto mencionado. Además, los resultados obtenidos a partir de este sistema permitirá identificar mejoras concretas respecto a las limitaciones actuales y sentar las bases para futuras aplicaciones en robótica autónoma.

1.4 Síntesis del problema

La arquitectura cognitiva e-MDB requiere que los propósitos humanos sean traducidos manualmente en misiones e impulsos específicos. Esta dependencia limita la escalabilidad y adaptabilidad del comportamiento autónomo del robot, lo que dificulta su alineación con objetivos relevantes para los humanos.

1.5 Objetivos del proyecto

El objetivo general de este proyecto era diseñar un sistema que permita a un robot autónomo descubrir y aprender misiones e impulsos, relacionar estas misiones con objetivos concretos, y alinear su autonomía con propósitos humanos predefinidos.

Para cumplir este objetivo general se plantearon los siguientes objetivos específicos:

1. Estudio de la arquitectura actual y revisar el estado del arte sobre los sistemas motivacionales actuales en robótica y los métodos para el descubrimiento y aprendizaje de objetivos.
2. Diseñar un sistema que permita a los robots descubrir y aprender misiones e impulsos para generar efectos que se alineen con los propósitos generales definidos por los diseñadores.
3. Diseñar una interfaz entre el algoritmo motivacional y el sistema robótico para ejecutar experimentos en un robot real.
4. Validar la implementación del sistema motivacional en un entorno de simulación robótico para la ejecución de tareas que le resulten útiles para el cliente.

1.6 Estructura del documento

Este documento está conformado por seis capítulos principales.

- En el Capítulo 1 se realiza la introducción del proyecto (capítulo actual).

- En el Capítulo 2 se desarrolla el marco teórico, donde se define la robótica autónoma, arquitecturas cognitivas y otros conceptos fundamentales para el desarrollo del proyecto.
- En el Capítulo 3 se detalla la metodología de diseño seguida, basada en la propuesta de Ulrich y Eppinger[3], adaptada al contexto del proyecto.
- En el Capítulo 4 se describe el diseño conceptual del sistema motivacional, su lógica de implementación y su integración con la arquitectura e-MDB.
- El Capítulo 5 presenta los resultados obtenidos tras la validación del sistema tanto en simulación como con un robot físico y un análisis de estos resultados.
- Finalmente, el Capítulo 6 incluye las conclusiones generales del proyecto y recomendaciones para trabajos futuros.

Capítulo 2

Marco Teórico

En este capítulo se desarrollan los fundamentos conceptuales necesarios para comprender el diseño e implementación del sistema motivacional propuesto. El contenido de esta sección responde al objetivo específico 1 del proyecto, que consiste en analizar los elementos teóricos que sustentan el desarrollo del sistema.

Este capítulo inicia con la definición de autonomía en robótica y los distintos niveles que existen. Seguidamente, se incluye una definición general de las arquitecturas cognitivas y sus enfoques motivacionales. Luego, se analiza en detalle la arquitectura e-MDB y sus componentes. También se aborda el problema de alineación entre autonomía robótica y objetivos humanos. Por último, se exponen los modelos grandes de lenguaje (LLMs) como herramienta para la interpretación semántica de propósitos.

2.1 Robótica autónoma

En términos generales, la robótica autónoma se puede definir como todos aquellos sistemas robóticos que son capaces de percibir su entorno, tomar decisiones y ejecutar acciones sin ningún tipo de intervención humana directa. Es decir, son robots que fueron diseñados para operar de manera independiente en entornos dinámicos y que tienen la capacidad de adaptarse a diferentes situaciones gracias a sus mecanismos de percepción, razonamiento y aprendizaje.

Sin embargo, es importante destacar que el objetivo de la robótica autónoma no es crear robots que sean completamente autónomos, pues de ser así, ya no serían útiles para el ser humano. Es por esta razón que, por lo general, en la robótica autónoma se habla de distintos niveles de autonomía.

- **Nivel 1:** Este es el nivel más bajo de autonomía, en este el diseñador conoce previamente las tareas que debe realizar el robot, así como el espacio o dominio en el que las debe realizar. En este caso, no hay cambios en las tareas que ejecuta el robot, sino que la autonomía se enfoca en la capacidad que tiene el robot para adaptarse a pequeños

cambios en el entorno en el que opera y cómo adapta las habilidades que ya conoce [1]. Un ejemplo de esto podría ser un brazo robótico que tenga la capacidad de presionar un botón sin importar la posición del objeto en la mesa.

- **Nivel 2:** En este nivel el robot ejecuta siempre las mismas acciones, sin embargo, se enfrenta a cambios significativos en el dominio. Entonces, el robot debe aprender y adaptar estas habilidades que ya conoce para poder realizar las tareas [1]. Un ejemplo de este nivel sería, un robot aspiradora. En este caso, la tarea siempre es la misma, aspirar el suelo, no obstante, estos robots deben funcionar en diferentes habitaciones o incluso casas diferentes.
- **Nivel 3:** En este caso, en el proceso de diseño, no se conocen ni las tareas ni el dominio en el que se deben realizar. Por tanto, el robot debe ser capaz de explorar, definir sus propios objetivos y aprender las habilidades necesarias para alcanzar dichos objetivos, esto mientras que se adapta a los cambios del dominio [1]. Un ejemplo de este nivel de autonomía sería un robot que debe colocar una pelota en un recipiente. En este caso, el robot solo percibe que hay una pelota. Por medio de la exploración, puede descubrir que la puede hacer rodar o que la puede tomar. Luego, que al llevarla a la posición adecuada recibe recompensa.
- **Nivel 4:** Este nivel es similar al anterior, solo que en este caso, el robot es capaz de reutilizar el conocimiento que ha aprendido en otros dominios para realizar la nueva tarea en un dominio diferente. Esto hace que el proceso de exploración y aprendizaje sea mucho más rápido y eficiente [1]. Un ejemplo de este nivel es un robot que sabe abrir puertas y que reutilice este conocimiento para abrir gavetas.

En los últimos años, muchos investigadores se han enfocado principalmente en los niveles 3 y 4 de autonomía. Esto implica que se ha logrado dotar a los robots autónomos de capacidades más complejas, como el aprendizaje de nuevas tareas, la adaptación a diferentes contextos, y la toma de decisiones basada en necesidades u objetivos internos. Estas capacidades han sido impulsadas principalmente por los avances que se han dado recientemente en el aprendizaje por refuerzo, las arquitecturas motivacionales y el uso de modelos internos del mundo.

No obstante, al dotar a los robots de capacidades que le permitan alcanzar niveles de autonomía más elevados, surgen también importantes consideraciones éticas. Este aspecto es especialmente delicado en entornos donde este tipo de robots pueda interactuar directamente con personas, ya que se espera que su comportamiento no cause ningún tipo de daño. Por ello, resulta fundamental asegurar una adecuada alineación entre la autonomía del robot y las expectativas humanas [2] [9].

2.2 Arquitecturas cognitivas en robótica

Para lograr alcanzar estos niveles más altos de autonomía en la robótica, se debe dar a los robots una capacidad de percepción activa, toma de decisiones, adaptación continua, y

en los casos más avanzados, aprendizaje abierto y a lo largo del tiempo (*lifelong open-ended learning*, LOLA)[10]. Es acá donde surgen las arquitecturas cognitivas. En esta sección se va a profundizar sobre las arquitecturas cognitivas en la robótica, algunos tipos de arquitecturas que existen y sus enfoques para el descubrimiento y aprendizaje de los objetivos.

2.2.1 ¿Qué es una arquitectura cognitiva?

De acuerdo con [11][12], una arquitectura cognitiva se considera un marco teórico y computacional que modela mecanismos de procesamiento cognitivos, y que permiten crear comportamientos inteligentes. Estos mecanismos por lo general están inspirados en el funcionamiento del cerebro humano. Tienen como objetivo integrar componentes como la percepción, memoria, razonamiento, aprendizaje, acción y motivación.

- **Percepción:** Es un proceso que transforma la información cruda que recibe el sistema del entorno por medio de sensores, en una representación interna del sistema para poder realizar tareas cognitivas [11]. La percepción es lo que le permite al robot generar su “visión del mundo”, conocer qué objetos tiene a su alrededor, saber qué acciones puede ejecutar en cada momento, entre otras. Por lo general, las percepciones más comunes se relacionan a la visión y el tacto.
- **Memoria:** Esta es una parte fundamental en las arquitecturas cognitivas. Acá se incluye la memoria de corto y largo plazo. La memoria de corto plazo se relaciona con puntos intermedios que facilitan el proceso de aprendizaje. Mientras que, la memoria de largo plazo hace referencia al conocimiento base y algunas acciones o problemas que el robot ya logró aprender por completo [11].
- **Toma de decisiones:** Se refiere a la capacidad de la arquitectura para seleccionar las acciones que debe realizar en cada momento. Por lo general se tienen dos tipos: los deliberativos y los reactivos. La reactividad son todas aquellas acciones que realiza el robot de manera inmediata, sin “pensar”, similar a los reflejos humanos [11]. Por otro lado, los sistemas deliberativos de toma de decisiones son todos aquellos procesos cognitivos que pueden hacer que un robot seleccione una acción u otra. Cada arquitectura cognitiva puede tener un sistema deliberativo diferente, algunos ejemplos podrían ser: motivaciones, estados afectivos, emociones, estados de ánimo, impulsos, entre otros. Es importante destacar que en toda arquitectura cognitiva se debe encontrar un balance entre las acciones reactivas y deliberativas para garantizar que el robot logre su propósito y que no se quede atascado haciendo acciones reactivas o aprendiendo tareas que no son necesarias [12]
- **Aprendizaje:** Es la capacidad que tienen estos sistemas para mejorar su rendimiento con el tiempo. Al igual que en los seres humanos, el aprendizaje en las arquitecturas cognitivas se basa en la experiencia del robot. Existe una gran variedad de tipos de aprendizaje, dependiendo del enfoque que le den los diseñadores a la arquitectura cognitiva, en sus casos de aplicación, método de entrenamiento deseado y los algoritmos [11].

- **Metacognición:** De acuerdo con Flavell, citado por [11], la metacognición se puede ver como “pensar sobre el pensamiento”. Son todas aquellas habilidades que permiten realizar una introspección sobre los procesos internos de razonamiento [11]. Se podría decir que es todo lo relacionado con la explicabilidad del sistema y la capacidad de identificar, explicar y corregir las decisiones que toma el sistema.

2.2.2 Tipos de arquitecturas cognitivas

Además de su aplicación en la robótica autónoma, las arquitecturas cognitivas se pueden aplicar a una gran variedad de áreas diferentes, algunas de las más comunes de acuerdo con [11] son: para experimentos psicológicos, modelado del desempeño humano en un entorno de tareas específico, interacción humano-robot y humano-computadora (HRI/HCI), procesamiento de lenguaje natural (NLP), análisis de grandes cantidades de datos, resolver problemas complejos de visión por computadora, juegos, entre otros.

Es por esta razón, que existen diversos tipos de arquitecturas cognitivas, dependiendo de su objetivo final. En el caso de este trabajo, se va a limitar el estudio de arquitecturas cognitivas en el área de robótica autónoma. Dentro de esta área, se pueden encontrar tres tipos concretos: simbólicas, emergentes e híbridas.

Arquitecturas cognitivas simbólicas

También conocidas como arquitecturas cognitivistas [11]. Estas arquitecturas se destacan por representar conceptos por medio de símbolos que pueden ser manipulados por medio de un conjunto de reglas predefinidas. Además, este tipo de arquitecturas se caracterizan por representar de manera intuitiva el conocimiento, lo que las hace altamente usadas. Sin embargo, cabe mencionar que también tienen una menor capacidad de adaptarse a cambios en el dominio donde operan, lo que las hace menos flexibles y robustas [11].

Arquitecturas cognitivas emergentes

Se les conoce como arquitecturas conexionistas. Este tipo de arquitecturas se caracterizan por tener un aprendizaje altamente paralelo, incluso se podría decir que es similar al de las redes neuronales. Esto se debe a que, en ambos casos, el flujo de la información se va propagando en diferentes nodos desde la entrada hasta la salida. Las arquitecturas emergentes superan las limitaciones de adaptabilidad y flexibilidad de las arquitecturas simbólicas, pero al ser estructuras más complejas y que ejecutan una serie de acciones en paralelo pierden cierto grado de explicabilidad [11]. Esto no significa que sean una caja negra, como el caso de las redes neuronales, pero resulta más complicado de comprender y explicar la forma en la que aprenden.

Arquitecturas cognitivas híbridas

Este tipo de arquitecturas trata de combinar lo mejor de los dos mundos: la robustez de las arquitecturas emergentes con la simplicidad y transparencia de las arquitecturas simbólicas. No existe ningún tipo de restricciones sobre cómo se debe hacer la combinación de ambos tipos, por lo que han surgido una gran variedad de combinaciones [11]. No obstante, no todas las arquitecturas híbridas desarrolladas analizan o describen detalladamente los elementos simbólicos que utilizan y por qué lo hacen.

2.2.3 Diferentes enfoques motivacionales para el descubrimiento y aprendizaje de objetivos

Cuando se habla de los enfoques motivacionales en el contexto de arquitecturas cognitivas, se hace referencia a los mecanismos intrínsecos que impulsan al robot para explorar su entorno, que le permita adquirir conocimiento y descubrir nuevos objetivos de manera autónoma. Por lo general estos mecanismos son basados en los procesos biológicos que se dan en el cerebro del ser humano y/o en los animales. Algunos ejemplos son: impulsos internos, la curiosidad, las emociones, estados de ánimo o los estados afectivos, entre otros [11].

Actualmente, existe una amplia cantidad de arquitecturas cognitivas que se han desarrollado a lo largo de los años para tratar de dotar autonomía a los robots. En esta sección se van a destacar algunas de las arquitecturas cognitivas desarrolladas y cuál es su enfoque motivacional para el descubrimiento y aprendizaje de objetivos.

Arquitectura DNC

La arquitectura DNC (*Dynamic Neural Curiosity*) está inspirada en el funcionamiento del cerebro humano, especialmente en cómo se regula la atención y la curiosidad. Esta arquitectura permite que un robot descubra y aprenda nuevos objetivos por sí mismo, sin necesidad de instrucciones externas. El robot alterna entre dos modos: uno de exploración, donde busca cosas nuevas en su entorno, y otro de aprendizaje, donde intenta alcanzar los objetivos que ha descubierto. Para decidir cuándo cambiar entre estos modos, el sistema utiliza señales como la novedad de lo que percibe o qué tanto está mejorando al intentar una tarea. Los objetivos se descubren observando cambios en el entorno provocados por las propias acciones del robot, y se priorizan aquellos que representan un reto o que aún no han sido dominados. Gracias a este enfoque, el robot puede desarrollar habilidades cada vez más complejas de forma autónoma, guiado únicamente por su curiosidad y progreso interno [13].

Arquitectura IMGEP

La arquitectura IMGEP (*Intrinsically Motivated Goal Exploration Processes*) permite que un robot explore y aprenda por sí mismo a través de metas que él mismo genera. En lugar de recibir instrucciones externas, el robot crea sus propios objetivos y decide cuáles

intentar alcanzar según qué tanto está aprendiendo. Esta decisión se basa en su progreso de aprendizaje: si una meta le resulta cada vez más fácil, entonces ha aprendido algo útil. Así, el robot se enfoca en las metas donde nota que está mejorando, y deja de lado aquellas donde ya no progresa. Las metas se definen como funciones que evalúan resultados, por ejemplo, mover un objeto a cierta posición o lograr cierto efecto en el entorno. IMGEP también organiza el aprendizaje de forma gradual, comenzando con tareas más simples y avanzando hacia otras más complejas. Gracias a este enfoque, el robot desarrolla nuevas habilidades de forma autónoma, motivado internamente por su propia mejora y curiosidad [14].

Arquitectura GRAIL

La arquitectura cognitiva GRAIL (*Goal-discovering Robotic Architecture for Intrinsically-motivated Learning*) utiliza un enfoque motivacional basado en la competencia. En este, el robot es capaz de descubrir y seleccionar sus propias metas a partir de sus señales sensoriales (percepciones), por lo que no depende de recompensas externas. En esta arquitectura se da recompensa al robot en el progreso que hace al aprender a mejorar una habilidad, y esto impulsa al robot a aprender [15].

Para descubrir los objetivos, esta arquitectura hace que el robot explore su entorno y detecte cambios o efectos generados por sus acciones. Por ejemplo, tomar un objeto, mover un objeto, encender una luz, entre otros. Estos cambios luego se transforman en un nuevo objetivo. La arquitectura prioriza los objetivos en los que el robot tenga una mayor mejora en su desempeño, que lleva al final a que el robot ejecute primero los objetivos más simples y luego avanza a tareas más complejas [15].

Arquitectura H-GRAIL

La arquitectura H-GRAIL (*Hierarchical Goal-discovering Robotic Architecture for Intrinsically-motivated Learning*) es una versión extendida de GRAIL que permite a un robot aprender de manera más organizada y adaptarse a tareas más complejas. H-GRAIL funciona a partir de dos tipos de motivación interna: una para descubrir nuevas metas (cuando el robot nota algo interesante o nuevo en el entorno), y otra para mejorar en alcanzar metas ya conocidas. El sistema decide en cada momento si explorar (buscar nuevas metas) o explotar (practicar una meta para mejorar su habilidad). Además, organiza las metas en pasos más pequeños llamados sub-metas, lo que permite al robot aprender tareas con varios pasos interdependientes. Esta arquitectura también está diseñada para adaptarse cuando cambian las condiciones del entorno, reorganizando sus planes y prioridades. Gracias a esta combinación de curiosidad, progreso y organización jerárquica, H-GRAIL permite un aprendizaje autónomo más robusto y flexible [16].

Arquitectura ASMO

Existen varias arquitecturas cognitivas que utilizan como motivaciones intrínsecas los impulsos, que provienen del término en inglés *drives*. Estos impulsos representan necesidades fisiológicas básicas como la alimentación, la seguridad, entre otros. Por lo general, estos impulsos se van “desvaneciendo” conforme se va satisfaciendo la necesidad. Tanto en el caso de los seres vivos como en los robots, a medida que estos trabajan para alcanzar los objetivos, pueden existir varios impulsos que estén afectando su comportamiento al mismo tiempo [11].

Un ejemplo de aplicación de este enfoque motivacional es la arquitectura ASMO. (*Attentive Self-Modifying Architecture*) está inspirada en cómo los humanos prestan atención y cambian su comportamiento según lo que les interesa o necesitan en un momento dado. En este modelo, el robot tiene varios procesos mentales que “compiten” por su atención, y aquellos que son más urgentes, relevantes o motivadores son los que se activan primero. ASMO incorpora un sistema motivacional simple, compuesto por tres impulsos o *drives*: seguir objetos de color rojo, recibir elogios y alcanzar un estado interno de bienestar o “felicidad”. Estos impulsos afectan cómo se distribuye la atención del robot y qué acciones decide realizar. Además, ASMO es capaz de modificar su propio comportamiento con el tiempo, cambiando las prioridades entre procesos en función de sus experiencias. Aunque no genera metas de forma autónoma como otras arquitecturas, su diseño le permite adaptarse a diferentes situaciones guiado por sus impulsos internos y su atención dinámica [17].

2.3 La arquitectura e-MDB

La arquitectura e-MDB (*epistemic Multilevel Darwinist Brain*) fue desarrollada por el GII como la evolución de la arquitectura original MDB (*Multilevel Darwinist Brain*). Esta fue diseñada para abordar el problema de la autonomía de aprendizaje abierto a lo largo de la vida en sistemas robóticos (LOLA) como parte de la tesis doctoral expuesta en [1]. El objetivo de este sistema es darle a los robots la capacidad de operar de forma autónoma en entornos dinámicos, que no sean anticipados durante la etapa de diseño. Esto les permite a los robots realizar acciones de exploración para descubrir objetivos, aprender a lograr estos objetivos, así como reutilizar y generalizar todo su conocimiento sin ningún tipo de intervención directa del diseñador o el usuario [1].

2.3.1 Componentes fundamentales de la arquitectura e-MDB

A continuación, se van a detallar las partes fundamentales que componen esta arquitectura, estos conceptos se basan en lo expuesto en [1].

1. **Percepción:** Esta parte incluye la detección y adaptación de las percepciones del sistema. Esta parte es la que le permite al robot darle su “visión del mundo”, ya que los robots físicos perciben su dominio por medio de sensores físicos o simulados.

2. **Bloque de representación:** Este bloque se encarga de agrupar las percepciones y transformarlas en una forma más útil y fácil de procesar para el robot. Además, incluye mecanismos de atención que filtran la información más relevante y descartan lo innecesario. La salida de este bloque corresponde al estado del robot en un instante de tiempo.
3. **Memoria episódica:** La memoria episódica se encarga de almacenar los episodios que experimentados en tiempo real por el robot a lo largo de un experimento. Un episodio, sería un vector que almacena la representación del estado del robot en un instante de tiempo “ t ”, las acciones realizadas en ese instante y el estado del robot después de haber aplicado las acciones, en el momento “ $t+1$ ”.
4. **Sistema motivacional:** Esta es la parte donde se va a desarrollar este proyecto. Este es un elemento fundamental de la arquitectura, está conformado por dos procesos:
 - El primero está relacionado con las necesidades e impulsos y la relación del estado actual del robot con la satisfacción de los impulsos. Todo esto es lo que guía el comportamiento del robot.
 - El otro proceso corresponde al selector de objetivos activos en el momento actual en función de las necesidades y los impulsos activos actualmente y su satisfacción.
5. **Sistema de aprendizaje:** Este bloque incluye todos los procesos de aprendizaje, entre estos se tienen: el aprendizaje de los modelos del mundo (*Model Learner*), modelos de utilidad y políticas (*Skill Learner*), y funciones de representación (*Representation Learner*). La entrada de este bloque corresponde a episodios almacenados en la memoria episódica. Además, cabe resaltar que todos los modelos aprendidos en este bloque, se almacenan en la memoria de largo plazo.
6. **Memoria de largo plazo (LTM, por sus siglas en inglés):** Este bloque se compone de dos sistemas: la memoria contextual y el sistema de decisiones. La memoria contextual se encarga de almacenar todo el contexto de los nodos de conocimiento. Mientras que el sistema de decisiones, se encarga de decidir qué acción realizar en un momento “ t ” dado. La memoria de largo plazo recibe los objetivos que provienen del sistema motivacional y los modelos aprendidos por el sistema de aprendizaje.
7. **Actuación:** Este bloque es el encargado de enviar los comandos correspondientes a los actuadores del robot para poder ejecutar acciones en el espacio.

Por último, en la Fig. 2.1 se ilustra un diagrama de todos estos elementos y cómo se interconectan entre sí.

2.3.2 Nodos cognitivos de la arquitectura e-MDB

Cada uno de los componentes fundamentales de la arquitectura cognitiva e-MDB se compone de uno o varios tipos de nodos cognitivos o nodos de conocimiento. Estos pueden ser

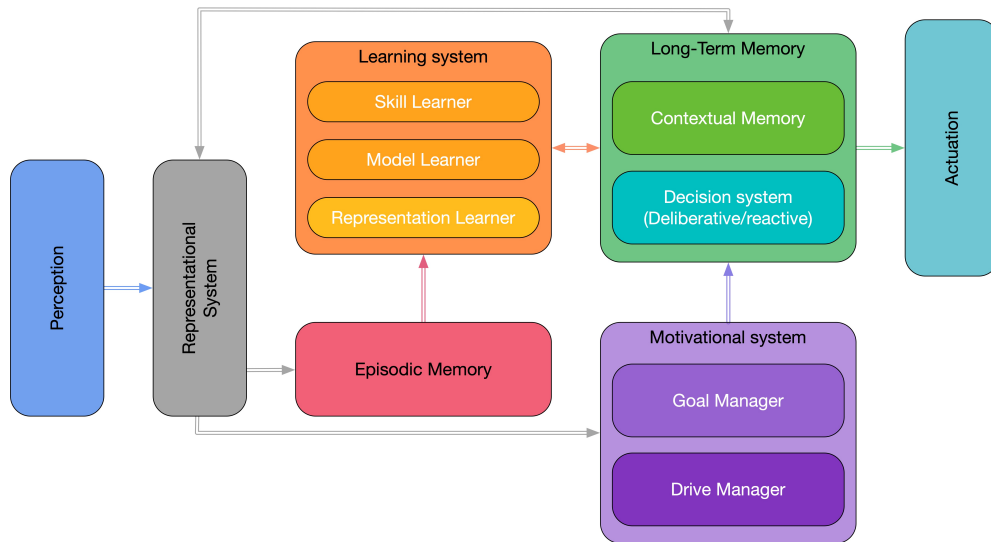


Figura 2.1: Componentes fundamentales de la arquitectura e-MDB. Tomado de [1].

aprendidos o dados por el diseñador. Todos estos nodos son almacenados en la estructura de almacenamiento de la arquitectura. En este caso, los nodos cognitivos de acuerdo con [1] serían los siguientes:

- **Percepción** (P_n): También se le conoce como observación o sensorización. En este se incluye toda la información del entorno físico y de los sensores internos del robot.
- **Redescripción de la percepción** (RP_n): Son las funciones que mapean las percepciones reales (los datos en crudo de los sensores) y la representación en la arquitectura de las percepciones.
- **Nodos-P** (pN_q): Son una manera de discretizar el espacio de estados perceptual (este es continuo y de altas dimensiones). Representan clases perceptuales a las cuales, si se les aplica la misma acción, en el mismo dominio, permite obtener la situación perceptual P_{t+1} .
- **Objetivos** (G_s): Son áreas en el espacio de estados que cuando son alcanzados llevan a una reducción del valor de al menos uno de los impulsos del sistema motivacional del robot. Dependen del dominio donde esté el robot.
- **Nodos-C** (cN_u): Estos nodos representan clases contextuales que conectan nodos-P con objetivos, el modelo de utilidad o la política ejecutada para alcanzar el nuevo estado en un dominio dado y el modelo del mundo. Su valor de activación depende de la activación de todos los nodos que contextualiza.
- **Modelo de mundo** (WM_v): Son funciones que representan el comportamiento del dominio en el que el robot está operando en ese momento específico.
- **Políticas** (π_w): Es una decisión reactiva que brinda la acción óptima que se debe aplicar para conseguir un objetivo.

- **Modelos de utilidad** (UM_t): Son funciones que dan la utilidad esperada de una percepción con respecto al objetivo que se desea alcanzar. En términos generales, los modelos de utilidad permiten determinar la probabilidad de alcanzar el objetivo a partir de una percepción.
- **Re-descripción de la actuación** (RA_x): Hace referencia a las funciones de transición entre la representación interna de las actuaciones y la actuación real en el robot.
- **Actuación** (A_y): Representan las acciones primitivas de la arquitectura. Estas incluyen tanto las acciones que afectan a los actuadores del robot como las acciones internas sobre la propia arquitectura.

En la Figura 2.2 se muestra una representación de cómo se organizan e interconectan los nodos cognitivos en la arquitectura para funcionar y comunicarse entre ellos.

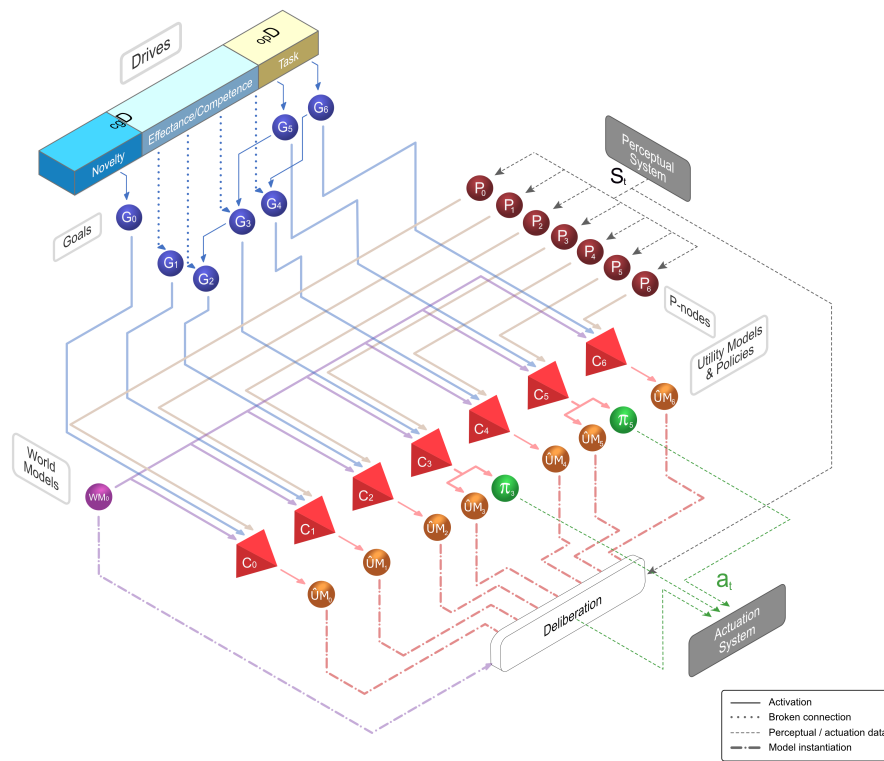


Figura 2.2: Esquema gráfico de la arquitectura e-MDB. Tomado de [1].

2.4 Sistema motivacional: necesidades, misiones e impulsos

Como se comentó en las secciones anteriores, el sistema motivacional está conformado por el administrador de objetivos y el administrador de impulsos. En esta sección se va a dar énfasis en este último, el cual se basa en los conceptos de necesidades (*Needs*), misiones

(*Missions*) e impulsos (*Drives*). Estos nodos cognitivos conforman el núcleo del mecanismo que le permite al robot descubrir y alcanzar objetivos útiles en dominios desconocidos [1].

2.4.1 Necesidades

En general, las necesidades son estados internos que el robot busca alcanzar o mantener. Por lo tanto, una necesidad puede definirse como un punto o área dentro del espacio de estados motivacionales del robot. Hasta el momento, el espacio de estados motivacionales del robot en la arquitectura cognitiva e-MDB era definido por el diseñador del robot y sus dimensiones se dan mediante variables que, generalmente, se refieren a parámetros internos del robot, por lo que son independientes del dominio [1].

2.4.2 Impulsos

Los impulsos cuantifican qué tan lejos está el sistema de satisfacer una necesidad. El valor de evaluación de un impulso está definido por una función de impulso f_j , que mide la distancia que hay entre el estado motivacional actual del robot (x_j) y el lugar donde se satisface la necesidad (n_j). Se pueden describir con la ecuación 2.1 [1].

$$D_j = f_j(x_t, n_j) \quad (2.1)$$

Existen diferentes tipos de necesidades e impulsos:

Necesidades e impulsos operacionales

Este tipo de necesidades e impulsos son todos aquellos que se relacionan con las tareas que el diseñador o usuario le desea dar al robot. Entonces, las necesidades operacionales se puede decir que reflejan el propósito que el diseñador o el usuario le quiere dar al robot, pero de manera que no dependa del dominio. Esto implica que el robot podría satisfacer el propósito humano en diferentes dominios ejecutando acciones diferentes [1]. En el contexto de este proyecto, a este tipo de necesidades se les llama misiones.

Necesidades e impulsos cognitivos

Por otro lado, las necesidades e impulsos cognitivos son todas aquellas relacionadas con la exploración y la adquisición de conocimiento. Son necesidades básicas que tiene el robot por existir. Además, este tipo de necesidades reflejan la manera en la que el diseñador quiere que el robot aprenda [1].

Diseño de impulsos

Hasta el momento, los impulsos y las necesidades son definidas por el diseñador, de acuerdo con el propósito que le quiere dar al robot. Para crear un impulso, de acuerdo con [1], basta con seguir estos dos pasos:

1. **Selección de sensores:** En esta parte se debe escoger todos aquellos sensores o percepciones que permitan medir la distancia para satisfacer la necesidad.
2. **Definir una función de impulso:** En este paso, se define una función matemática continua y monótonamente decreciente, que permita cuantificar el nivel de logro de la necesidad, es decir, qué tan lejos está de ser satisfecha la necesidad. Estas funciones pueden tener diferentes comportamientos, podrían ser lineales, cuadráticas, exponenciales, entre otras. En cuanto más se acerquen a cero, indican que la necesidad está satisfecha.

2.5 Problema de alineación en la robótica autónoma

La autonomía sin propósito no sirve de mucho al ser humano, ya que no hay forma de garantizar que las acciones que realice el robot sean útiles para el ser humano. Es aquí donde nace uno de los principales desafíos que vienen con la robótica autónoma LOLA, que es cómo garantizar que los robots puedan aprender de manera autónoma, pero que cumpla a su vez con los propósitos humanos.

El problema central está que en el aprendizaje abierto y a lo largo de la vida, los robots son capaces de aprender continuamente nuevas habilidades, sin importar el dominio en el que estén. Entonces, cómo se puede garantizar que estos aprendizajes son cosas útiles para el humano o que las acciones que el robot está aprendiendo no van a afectar negativamente a los humanos. Esto es parte del dilema planteado en [2].

Para resolver este problema, en [2], se propone un marco basado en el propósito (purpose). Específicamente, se hace referencia al propósito humano, el cual especifica lo que los humanos quieren que el robot aprenda, haga o no, dentro de un límite de autonomía e independientemente de los ámbitos en los que opera. Es a partir de esto, que nace el marco de propósito (Purpose Framework), el cual propone hacer una división jerárquica de tres niveles:

1. **Nivel humano:** Dentro de este se consideran los posibles usuarios o el diseñador del robot. Dentro de este nivel, se tienen dos partes principales:
 - **Propósito humano:** El humano tiene por sí mismo un propósito humano (human purpose). Este se expresa en lenguaje de alto nivel, es independiente del dominio y en muchos casos, puede ser abstracto. Un ejemplo de esto podría ser “ordenar una mesa”.

- **Objetivos humanos:** Este propósito se relaciona con objetivos humanos. Estos objetivos, por lo general, se relacionan con dominios concretos. Por ejemplo, “colocar los lápices en el portalápices”.
2. **Nivel robot:** En este nivel se consideran las mismas partes que en el nivel humano, pero desde la perspectiva del robot:
 - **Propósito del robot:** El propósito del robot se expresa como necesidades y misiones, dependiendo de si el propósito proviene del diseñador o del usuario. Igualmente, son independientes de cualquier dominio y tienen una relación directa con el propósito humano.
 - **Objetivos del robot:** Estos son estados dentro de dominios específicos deseados que satisfacen los propósitos.
 3. **Mundo o dominios:** Este nivel hace referencia al entorno donde se encuentra tanto el robot como el humano. En este es donde el humano espera observar que se cumpla realmente su propósito y también es el ambiente en el que el robot genera efectos.

Cuando el propósito y objetivo del humano coinciden con el propósito y objetivo del robot y además, los efectos generados por el robot en el dominio tienen concordancia con ambos propósitos, se dice que existe una alineación triangular, o simplemente alineación (alignment) [2]. En Figura 2.3, se muestra un diagrama que relaciona estos niveles jerárquicos y el concepto de alineación.

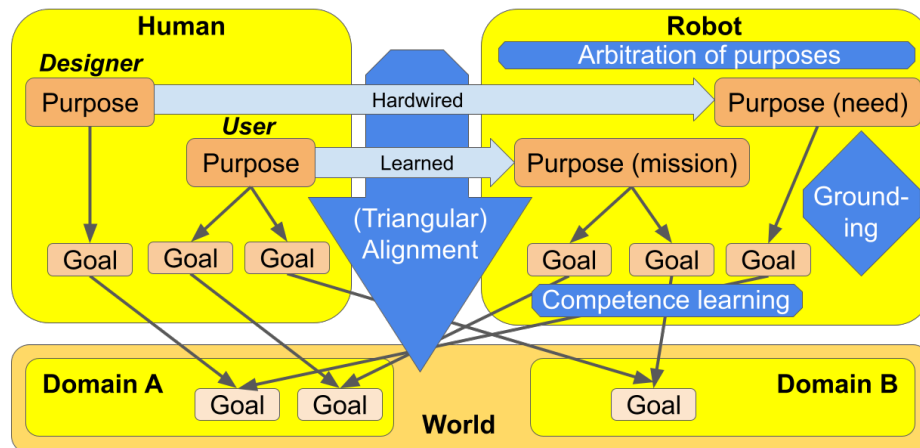


Figura 2.3: Organización jerárquica para el marco de propósito que logra una alineación triangular. Tomado de [2].

2.5.1 Subproblemas del alineamiento

Como se puede notar, el problema de alineación es bastante complejo de definir y resolver. Es por eso que hace una división de este en cuatro sub-problemas más pequeños. A continuación, se van a detallar cada sub-problema y se van a mencionar las maneras de resolver estos problemas basándose en la investigación [2].

Priorización de propósitos: funciones de utilidad motivacional

Este subproblema surge de la pregunta:

¿Cómo puede el robot priorizar entre diferentes propósitos?

De acuerdo con [1], hasta este momento, la priorización de las necesidades y misiones en la arquitectura cognitiva se ha hecho por medio de pesos definidos manualmente por el diseñador a la hora de crear las necesidades. De manera que los propósitos con un mayor peso tienen una mayor probabilidad de activar políticas o modelos de utilidad que haga que satisfagan esta necesidad o misión. Entonces, si este proceso se hace de manera autónoma se debe garantizar que se haga también esta jerarquización de las tareas para garantizar que el robot es capaz de cumplir todos los propósitos que tiene definidos.

Problema de alineación de propósitos del humano y el robot

En este caso, este subproblema nace de la pregunta:

¿Cómo se puede asegurar que los propósitos del robot, referentes a misiones, tienen una correspondencia con los propósitos del humano?

Para resolver este problema, se proponen las siguientes acciones:

1. En primer lugar, se propone que durante la etapa de diseño, se definan las necesidades y sus pesos de prioridad. En este caso, la consideración que se debe realizar es que las necesidades se apeguen al propósito humano.
2. Luego, se propone que las misiones sean transmitidas por un espacio compartido de codificación. En este caso, se debe afrontar la dificultad del lenguaje humano de alto nivel, ya que los propósitos humanos pueden ser subjetivos y ambiguos.
3. Por último, se propone que las misiones se infieran a partir de ejemplos de objetivos de dominio que generan satisfacción en el usuario, o bien, que se descubran los objetivos de manera autónoma y se le pida validación al humano. En este caso, se enfrenta el reto de que el proceso de retroalimentación puede ser extenso y no siempre se podrá generalizar a todos los casos.

Problema de puesta a tierra entre el propósito y el objetivo

En primer lugar, se debe aclarar que el concepto de puesta a tierra mencionado en el título de la sección, viene del término en inglés *grounding*. En este caso, se va a utilizar con el significado de aterrizar, anclar o poner a tierra un concepto. Para mantener una homogeneidad en el texto, se va a utilizar “poner a tierra” para referirse a este concepto.

Ahora, este sub-problema surge de la diferencia conceptual existente entre propósito y el objetivo, ya que los propósitos son independientes del dominio, mientras que los objetivos por el contrario sí dependen del dominio. Entonces, el problema radica en cómo se pueden mapear propósitos abstractos con objetivos concretos.

Para resolver este problema, se proponen tres acciones que puede realizar el robot:

- La primera es que el robot sea capaz de reconocer y segmentar los objetos del dominio.
- También se propone que se identifiquen palabras clave relevantes que pueden estar en el propósito.
- Por último, generar estados objetivo candidatos que satisfagan el propósito abstracto.

El problema de adquisición de competencia

Este subproblema se genera a partir de la pregunta:

¿Cómo puede aprender el robot a alcanzar objetivos puestos a tierra asociados con su propósito?

Este subproblema se puede afrontar de varias maneras:

- El robot puede aprender políticas por medio de funciones de recompensa, donde recibe recompensa en los casos que lo acercan al objetivo.
- Por medio de funciones de utilidad que le permita al robot definir qué tan deseable o útil es un estado o una situación.

Como se puede apreciar en esta explicación teórica, el problema de alineación es altamente complejo e involucra muchos factores que se deben considerar. Por lo que en este proyecto se propone una primera aproximación en la que se hace una prueba del concepto. Por tanto, no se espera que la solución sea perfecta.

2.6 Modelos de lenguaje grandes (LLMs)

Los modelos de lenguaje grandes (LLMs, por sus siglas en inglés), son un tipo de inteligencia artificial generativa que ha tenido un crecimiento exponencial en los últimos años. Estos modelos utilizan modelos de redes neuronales diseñados y entrenados para el procesamiento del lenguaje natural (NLP) [18]. Normalmente estos modelos utilizan texto, no obstante, recientemente se han popularizado modelos de LLMs para la generación de imágenes, videos y audios.

En la actualidad existen una gran variedad de modelos de LLMs, generados por diferentes compañías, algunas de las más destacadas están: OpenAI con sus modelos *GPT*, Google destaca con sus modelos *Gemma* y *Gemini*, Meta por su parte, creó el modelo *Llama*, entre muchos otros ejemplos.

Características importantes de los LLMs

- **Temperatura:** Controla el nivel de aleatoriedad que tienen las respuestas del modelo. Un valor más bajo de temperatura hace que las respuestas del modelo sean más deterministas. Mientras que los valores más elevados de temperatura hacen que se incremente la creatividad del modelo [19].
- **Solicitud (“*Prompting*”):** Hace referencia a la forma en la que una persona interactúa con el modelo. Consiste en un conjunto de especificaciones claras y explícitas sobre lo que el humano desea que haga el LLM [20]. El contenido de una solicitud puede variar dependiendo de la persona, el contexto en el cual se va a utilizar y la tarea que quiere realizar. Según [20], cuando se quiere utilizar un LLM para una tarea específica, por lo general, se deba iterar con diferentes solicitudes, hasta lograr que el modelo tenga el comportamiento deseado.

2.7 Similitud semántica y embedding en modelos LLM

En los modelos de NLP, obtener un valor numérico que permita cuantificar cuán similar son dos textos es un instrumento clave, especialmente cuando se trabaja con modelos de LLM. Existen diferentes formas de obtener la similitud entre dos textos. En este caso, se va a profundizar en la similitud semántica.

La similitud semántica entre dos textos, hace referencia al significado que tienen dos textos o palabras. Por ejemplo, las palabras “rey” y “reina” tienen una similitud semántica más elevada que, por ejemplo, “rey” y “hombre” [21].

No obstante, estas comparaciones son difíciles de realizar con los textos tal cual, por lo que en los modelos LLM y NLP se utilizan representaciones vectoriales numéricas. A estos se les conoce como “*embedding*” o incrustaciones. De acuerdo con [22], las incrustaciones son representaciones numéricas, mediante un vector, del significado de una palabra u oración. En este contexto, los vectores pueden concebirse como secuencias de números que codifican la información semántica de palabras o frases.

2.7.1 Cálculo de la similitud semántica

Ahora, existen diversas formas de obtener la similitud semántica en un valor numérico, en este caso, se optó por la similitud coseno. Este es un parámetro que permite obtener la desviación que existe entre los ángulos de los dos vectores, los cuales representan el significado de la

palabra o el texto [21]. Este valor se calcula con la ecuación 2.2, donde “A” y “B” son los vectores de incrustación de los textos o palabras que se desean comparar semánticamente.

$$\text{sim}(A, B) = \frac{A \cdot B}{\|A\| \|B\|} \quad (2.2)$$

2.7.2 Interpretación

Dependiendo del valor obtenido al evaluar la ecuación 2.2, se pueden inferir ciertas características semánticas de los textos a comparar. En primer lugar, cabe resaltar que el resultado de la similitud semántica coseno es un valor que se encuentra acotado entre -1 y 1. En este rango, existen los siguientes casos de interés:

- **Similitud coseno igual a 1:** Esto significa que el ángulo que existe entre los vectores de incrustación de los textos a comparar es 0° . Semánticamente, esto significa que los dos textos son idénticos, es decir, utilizan exactamente las mismas palabras en el mismo orden.
- **Similitud coseno igual a -1:** Este resultado indica que el ángulo entre los vectores de incrustación de los textos a comparar es 180° . Esto implica que los vectores de incrustación tienen direcciones opuestas, lo cual se traduce a que semánticamente, los dos textos tienen significados opuestos, por ejemplo las palabras “inicio” y “fin”.
- **Similitud coseno igual a 0:** En este caso, el resultado indica que el ángulo que existe entre los dos vectores de incrustación es de 90° . Semánticamente, esto significa que los textos no tienen ninguna relación clara entre sí.

Capítulo 3

Metodología

En este capítulo se expone la metodología seguida para el diseño propuesto en este proyecto. La metodología seguida se basa en el libro “Diseño y Desarrollo de Productos” escrito por Ulrich y Eppinger [3]. A continuación, se van a tratar todos los aspectos teóricos de esta y las adaptaciones hechas a la misma para el desarrollo de este proyecto.

3.1 Descripción general

La metodología de Ulrich y Eppinger se caracteriza porque permite hacer un estudio exhaustivo de posibles soluciones, entre las cuales al final se escoge la que mejor resuelve el problema identificado. Además, permite estructurar el desarrollo del proyecto en fases interrelacionadas, que abarcan desde la comprensión del problema hasta la validación de las soluciones. Cabe destacar que este es un proceso iterativo, lo que implica que es posible retroceder a fases anteriores en cualquier momento para ajustar o mejorar el desarrollo [3]. En la Figura 3.1 se muestran las diferentes etapas que propone el libro para el desarrollo de productos.

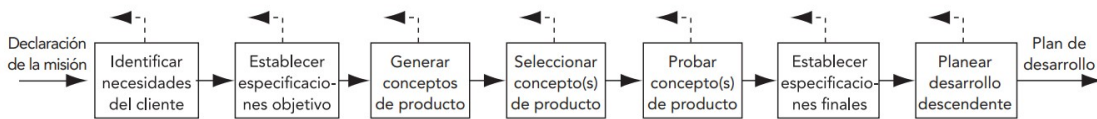


Figura 3.1: Etapas propuestas por la metodología de Ulrich y Eppinger para el desarrollo de productos [3].

A continuación se van a explicar brevemente cada una de las etapas.

- **Identificación de las necesidades:** Esta sección tiene como objetivo comprender las necesidades del cliente para considerarlas en el proceso de diseño. Para esto, es necesario obtener toda la información posible a través de entrevistas con el cliente. Cuando se redactan las necesidades, es de vital importancia que estas sean completamente independientes de cualquier posible concepto o forma de resolver el problema. Cuando

se tiene una lista de todas las necesidades, se procede a hacer una jerarquización y establecer una importancia relativa de todas estas necesidades. Para este proyecto en esta etapa se obtienen todos los requisitos que espera el cliente que tenga la solución final diseñada.

- **Establecimiento de especificaciones y métricas:** Esta etapa consiste en expresar las necesidades del cliente en una descripción precisa de lo que el producto debe realizar en lenguaje técnico. Al final de esta etapa se obtienen un conjunto de especificaciones objetivo medibles con las que se van a evaluar el cumplimiento de todos los requisitos del cliente. Para el caso del proyecto, esta es una etapa crucial, pues las métricas establecidas van a ayudar a justificar ante el cliente que la solución propuesta efectivamente cumple con los requisitos que especificó.
- **Generación de conceptos:** La etapa de generación de conceptos tiene como objetivo realizar una búsqueda exhaustiva de todas las posibles maneras en las que se puede resolver el problema. Para esto se utilizan recursos como la búsqueda interna y externa, exploración sistemática de fragmentos de solución y la solución creativa de los problemas. Para este proyecto, esta es una etapa muy importante, pues permite explorar y descubrir diversas formas de resolver el problema.
- **Selección de conceptos:** En la etapa de selección de conceptos se realiza un análisis cuidadoso de todas los conceptos generados. Por medio de pruebas puntuadas se califica cada concepto hasta quedar solamente con los conceptos más prometedores. En el proyecto realizado esta es uno de las etapas más importantes, pues permite acotar las posibles soluciones a tres mejores soluciones, de las cuales se escoge solamente una para desarrollar la prueba del concepto.
- **Prueba del concepto:** La etapa de prueba del concepto es aquella en la que se plantean pruebas para validar el concepto final seleccionado. La pruebas se diseñan de manera que se verifique que la solución propuesta cumpla con las especificaciones establecidas al inicio. En el caso de este proyecto, esta parte permite asegurar que las métricas propuestas se cumplen, pero además en caso de que alguna no se cumpla, permite también determinar qué factores hacen que no se cumpla y cuáles serían posibles puntos de mejora.
- **Generación de especificaciones finales:** En esta etapa se revisan las especificaciones objetivo establecidas al inicio del proyecto y se comparan con los resultados obtenidos de la prueba del concepto. Esto permite identificar limitaciones que hay en diseño y en el desempeño del producto. Para el proyecto desarrollado esta sección es importante pues permite obtener cuáles serán los siguientes pasos en torno al proyecto de investigación del cliente.
- **Planificación del proyecto:** En esta actividad se genera un plan detallado para el desarrollo del producto, dentro de este se incluyen: una estrategia para reducir al mínimo el tiempo de producción y todos los recursos necesarios para completar el

proyecto. En el caso de este proyecto, Esta etapa no se va a considerar, pues la solución propuesta no corresponde a un producto comercial.

3.2 Metodología adaptada para el proyecto

Para este proyecto se tuvo que realizar modificaciones sutiles a la metodología anterior, ya que esta está pensada para el desarrollo de un producto. Mientras que el proyecto desarrollado tiene un enfoque de investigación y el producto final es una prueba de concepto. Entonces, para la parte de desarrollo de conceptos se buscaron diferentes puntos de inicio y luego por medio de un proceso de prueba y error se fue mejorando el diseño. En esta sección se va a detallar sobre lo que se realizó en cada una de las etapas anteriores y cuáles fueron las adaptaciones realizadas para lograr concretar el diseño propuesto en este trabajo.

3.2.1 Contextualización de la temática del proyecto

Para poder iniciar el proyecto se tuvo que realizar un estudio técnico del proyecto de investigación. En este caso, se hizo el estudio de la arquitectura e-MDB. Dentro de este se incluye una comprensión de los conceptos teóricos de la arquitectura cognitiva y las partes concretas con las que se iba a trabajar.

Una vez se estudiaron los conceptos, se generaron preguntas al experto en la arquitectura cognitiva Dr. Alejandro Romero Montero. Este proceso fue vital para poder comprender correctamente el problema que se quería solucionar. Además, esto permite realizar las preguntas correctas durante las entrevistas con el cliente y con esto lograr aplicar el resto de la metodología correctamente.

3.2.2 Identificación de necesidades

Para esta etapa de la metodología, se plantearon dos entrevistas con Dr. Richard José Duro Fernández, director del GII y Dr. Alejandro Romero Montero, investigador. Ambos son expertos en la arquitectura cognitiva e-MDB. En la primera se trataron preguntas generales sobre sus expectativas de la solución final. Como resultado de esta primera reunión, se generó una lista inicial de necesidades.

Luego, en la segunda reunión, se analizaron las necesidades iniciales, lo cual abrió el diálogo con respecto a algunos aspectos que quedaban confusos sobre la definición del proyecto. Al final de esta sesión, se obtuvieron las necesidades finales del proyecto.

Posteriormente, estas necesidades fueron procesadas y compartidas por medio de un formulario al Dr. Alejandro Romero Montero para que estableciera una importancia relativa de las necesidades en una escala del 1 al 5. En la Tabla 3.1, se muestra una descripción del significado de los distintos niveles de prioridad.

Tabla 3.1: Niveles de importancia relativos para las necesidades.

Nivel de importancia	Descripción
1	Es indeseable
2	No es importante
3	Es deseable
4	Es importante
5	Es imprescindible

Por último, en la Tabla 3.2, se muestran las necesidades finales con su importancia relativa para el cliente.

Tabla 3.2: Necesidades del proyecto y su importancia.

ID	Enunciado de la necesidad	Importancia (1–5)
1	El sistema permite que el usuario exprese sus propósitos en lenguaje natural de alto nivel y es capaz de interpretarlo	5
2	El sistema convierte los propósitos del humano en misiones válidas	5
3	El sistema genera funciones matemáticas de impulso que permiten evaluar el nivel de satisfacción de una misión	5
4	El sistema se incorpora con el código de la arquitectura cognitiva existente en Python y ROS2 y se comunica con el robot	5
5	El sistema realiza acciones de alineación para asegurar que el propósito del humano esté completamente definido	4
6	El sistema prioriza las misiones de acuerdo con la importancia que tenga para cumplir con el propósito asignado y considerando las necesidades ya definidas	4
7	El sistema es fiel a los conceptos de la arquitectura e-MDB	4
8	El sistema proporciona retroalimentación clara al usuario sobre cuáles son las funciones de impulso en el sistema, además de cómo y cuándo se están satisfaciendo	3
9	El sistema se prueba en un robot real en un ambiente de simulación	3

3.2.3 Establecimiento de especificaciones y métricas

En esta sección se tradujeron las necesidades obtenidas en la etapa anterior a términos técnicos medibles. Para esto se redactaron las métricas y especificaciones de manera que cada necesidad tuviera al menos una especificación. Luego, para definir los porcentajes de aceptación se consultó con los investigadores del proyecto cuáles serían valores aceptables para el grupo. Igualmente, al tratarse de un proyecto de investigación, en donde se dará una primera aproximación a la barrera existente de cómo se podría dar una alineación de la autonomía de robots autónomos [2], es esperable que los porcentajes sean bajos y que el resultado final sea aplicado en una simulación y no en un robot físico. En la Tabla 3.3 se encuentran las métricas finales obtenidas.

Tabla 3.3: Métricas asociadas a las necesidades del proyecto.

ID	Necesidad	Métrica	Unidad	Valor marginal	Valor ideal
1	N1	Interpreta propósitos en lenguaje de alto nivel	Binaria	Cumple	Cumple
2	N2	Porcentaje de misiones válidas	Porcentaje	$\geq 70\%$	$\geq 80\%$
3	N3	Porcentaje de funciones de impulso válidas	Porcentaje	$\geq 70\%$	$\geq 80\%$
4	N4	La arquitectura recibe las percepciones del simulador	Binaria	Cumple	Cumple
5		El robot ejecuta políticas enviadas por la arquitectura cognitiva	Binaria	Cumple	Cumple
6	N5	Grado de convergencia entre la descripción final del propósito del usuario y la descripción ideal	Porcentaje	$\geq 70\%$	$\geq 80\%$
7	N6	Porcentaje de veces que el robot logra la misión completa en el orden adecuado	Porcentaje	$\geq 70\%$	$\geq 80\%$
8	N7	Utiliza los conceptos teóricos de la arquitectura	Binaria	Cumple	Cumple
9	N8	Brinda retroalimentación sobre las funciones de impulso evaluadas	Binaria	Cumple	Cumple
10	N9	Funciona en un robot real en un ambiente simulado	Binaria	Cumple	Cumple

Nota: Las referencias N1–N9 corresponden a las necesidades enumeradas en la Tabla 3.2.

Justificación de las métricas

- **Métrica 1:** Esta métrica hace referencia a la manera en que se van a ingresar los datos al sistema. Se espera que sea una frase en lenguaje cotidiano, de alto nivel. Esta métrica se definió como binaria, puesto que es una característica con la que el sistema cumple o no [3].
- **Métricas 2 y 3:** Estas métricas hacen referencia a una de las características más importantes que debe tener el sistema, ya que se consideran la salida, de las cuales depende la ejecución de las acciones y el aprendizaje del robot. En este caso, se propone como unidad de medida un porcentaje del total de los resultados obtenidos. Es importante destacar que los porcentajes de aceptación se les asignaron valores entre el 70 y 80%, pues ambas tareas en conjunto abarcan un problema abstracto y que de acuerdo con [2], aproximan un vacío que no se ha resuelto todavía. El cual consiste en traducir un propósito humano en una misión entendible por la arquitectura cognitiva de un robot completamente autónomo y que verdaderamente sea capaz de interpretarlo y cumplirlo. Cabe resaltar que para que este porcentaje sea válido, se deben realizar una cantidad de pruebas representativas, de manera que se compruebe que el sistema funciona correctamente ante diversos escenarios.
- **Métrica 4 y 5:** Estas métricas se utilizan para evaluar el nivel de integración que tiene el sistema con la arquitectura cognitiva y con el simulador robótico. Principalmente, lo que se necesita obtener del simulador para que la arquitectura cognitiva funcione correctamente son los valores de los sensores y observar que los impulsos se evalúen correctamente. Esto se evalúa con la métrica 4. Por su parte, para que el robot funcione, necesita recibir señales para la activación de sus actuadores. Esta comunicación se plantea evaluar con acciones concretas (políticas).
- **Métrica 6:** Esta es una métrica relacionada con las acciones de alineación realizadas por el sistema. Esta medida es bastante importante, pues para que el robot ejecute exactamente lo que espera el humano, se necesita que este le de información de la manera más completa posible. Esta métrica propone evaluar la diferencia que existe entre el propósito inicial dado y el propósito final generado después de las acciones de alineación, ambos comparados con un propósito final bien descrito. En cuanto a las unidades, al ser una comparación, es un factor escalar y los valores de aceptación se basan en los resultados expuestos en [22].
- **Métrica 7:** Esta métrica es también muy importante, pues permite evaluar la implementación final del sistema en la arquitectura cognitiva ejecutándose en el robot del simulador. Este resultado indicaría que el sistema no solo tradujo correctamente el propósito del humano en misiones y funciones de impulso válidos, sino que también hizo una correcta distribución de los pesos (importancia) que tiene cada misión. Esto afecta el orden en el que el robot cumple las misiones y por ende es un indicador de si cumple con el propósito humano esperado. Para medir este parámetro se seleccionó el porcentaje de veces que el robot logra cumplir con el propósito humano en una

cantidad de iteraciones representativas. En cuanto a los valores de aceptación, estos porcentajes también están entre el 70 y el 80%, pues es un parámetro que depende de la efectividad de las misiones y funciones de impulso generadas por el sistema integrados con la arquitectura y el robot en la simulación.

- **Métricas 8, 9 y 10:** Estas métricas es la forma de evaluar condiciones muy específicas que son deseables por el GII. En los tres casos se presentan detalles que los tiene o no el sistema, no existe un punto intermedio para evaluarlo.

Por último, cabe resaltar que en este caso, el cliente no especificó necesidades relacionadas al costo del de la solución, por lo que este no es un factor que se tome en cuenta para el diseño de la solución.

3.2.4 Generación de conceptos

Para esta sección se trató de seguir al máximo la metodología de Ulrich y Eppinger. De manera que se lograra describir al máximo el problema y que se puedan explorar diferentes formas de resolverlo.

Aclaración del problema

Para iniciar este proceso, se debe recapitular el problema general que se tiene en la actualidad. Este consiste en que el usuario debe programar manualmente las misiones y los de impulso dependiendo de lo que desee que realice el robot. Por lo que cada vez que se quiera agregar un propósito nuevo, se debe agregar una o varias misiones nuevas y funciones de impulso que las evalúen. Esto hace que el robot pierda autonomía. Entonces, lo que se busca es generar una solución, que tome el propósito humano en lenguaje de alto nivel y lo traduzca a misiones e impulsos válidos para la arquitectura cognitiva e-MDB.

Descomposición funcional del problema

Teniendo un panorama general sobre lo que la solución debe hacer, se puede hacer un diagrama de caja negra donde se muestren cuáles serían las entradas y salidas del sistema. En la Figura 3.2 se muestra este diagrama para el problema por resolver.

Como se puede apreciar en la Figura 3.2, en las entradas no solo se tiene el propósito del humano, sino que también se deben considerar: las necesidades intrínsecas definidas internamente en el sistema y las percepciones del robot. Esto se debe a que el sistema debe considerar los pesos de las necesidades intrínsecas, de manera que las misiones no se vean afectadas por ninguna de estas y viceversa. Por otra parte, las percepciones se debe considerar pues son las señales que se deben utilizar para generar las funciones de impulso, sin estas no hay manera de relacionar lo que percibe el robot con lo que se desea alcanzar.

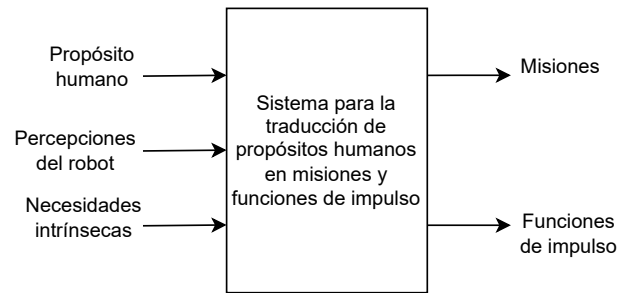


Figura 3.2: Diagrama de caja negra para representar el problema por resolver. Elaboración propia

Ahora, este es un problema bastante complejo. Por lo que se debe dividir en subproblemas más simples para poder pensar en posibles soluciones. En la Figura 3.3, se muestra el diagrama de la composición funcional para este caso.

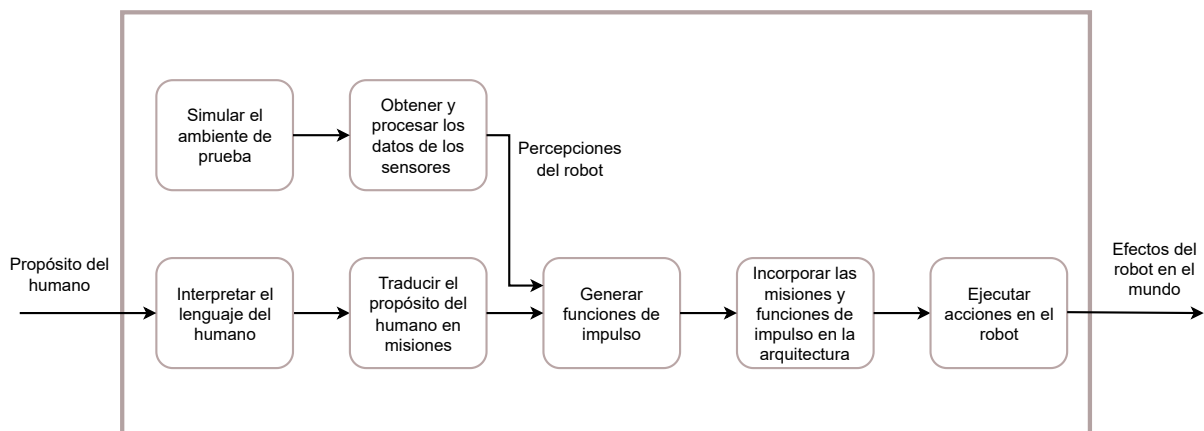


Figura 3.3: Descomposición funcional del problema por resolver. Elaboración propia.

Como se puede observar en Figura 3.3 si se considera que el sistema va a recibir un propósito en lenguaje de alto nivel, lo primero que se debe hacer es asegurarse que este esté expresado con el mayor nivel de detalle posible, para asegurar que las misiones y funciones de impulso generadas cumplan realmente con lo que el usuario desea. Una vez la descripción del propósito está completa, esta se debe traducir a misiones y funciones de impulso. Además, se tienen otros subproblemas más generales como simular el ambiente de prueba, tomar las percepciones e incorporar el sistema en la arquitectura cognitiva.

Búsqueda interna y externa de posibles soluciones

En esta sección se buscan posibles soluciones a los subproblemas identificados en Figura 3.3. Cabe resaltar que hay algunos problemas que dependen de otros factores externos y que están fijos, es decir, las maneras en que se pueden resolver están limitadas. Entonces

solamente se realizó la búsqueda de posibles soluciones para los siguientes subproblemas: simular el ambiente de prueba, interpretar el lenguaje del humano, traducir el propósito del humano en misiones y traducir el propósito humano en funciones de impulso.

Para la búsqueda externa, primero se realizó una investigación en internet para generar ideas, posteriormente, estas ideas se presentaron a investigadores del GII, para evaluar desde su criterio experto, las mejores opciones. Además, en la búsqueda interna se propusieron otras ideas que surgieron al tratar el proyecto. A continuación, se muestran los principales resultados obtenidos.

- **Simular el ambiente de prueba:** En esta área existen una gran variedad de entornos de simulación que se pueden utilizar. Al realizar una búsqueda en internet se encontraron las siguientes opciones:
 - **Webots:** Esta es una aplicación de escritorio de código abierto que permite simular diversos entornos de simulación con robots. En la aplicación se incluyen una gran variedad de robots reales comerciales equipados con sensores y actuadores. Además, permite ingresar objetos personalizados en formato CAD para crear un ambiente de simulación más cercano al real. Otra de sus características es que se puede ejecutar en Windows, Linux y macOS. Mientras que los robots pueden ser programados en C, C++, Python, Java, MATLAB o ROS (tiene nodos de ROS pre-programados). Además de que tiene una buena documentación y ejemplos de uso [23].
 - **CoppeliaSim:** Es un simulador robótico en 3D que permite simular condiciones complejas. Es una herramienta comercial utilizada principalmente para simulaciones de automatización de fábricas, creación rápida de prototipos y verificación, educación, entre otras. Tiene modelos de robots reales comerciales y sus sensores. Se puede instalar en sistemas operativos Windows, Linux y macOS. Además de que sus elementos son programables con: Python or Lua, C, C++, Java, MATLAB y Rust. Su principio de funcionamiento está basado en una arquitectura de control distribuida, lo cual lo hace altamente compatible con ROS. Sin embargo, es un software de pago y su versión gratis tiene permisos de edición limitados y está disponible solo bajo solicitud [24].
 - **Gazebo:** Es un simulador de robótica avanzado creado para la investigación, el desarrollo y diseño. Permite generar simulaciones realistas gracias a su amplia variedad de modelos de sensores comerciales, su alto nivel de precisión en su motor físico, su renderizado 3D y capacidad de simulación de ruido. Este simulador tiene un principio de funcionamiento altamente compatible con ROS, de hecho, cuenta con la herramienta *Gazebo bridge*, la cual convierte de manera automática los mensajes de Gazebo en mensajes de ROS. Sin embargo, solamente es compatible con Linux, o utilizando un subsistema de Windows para Linux (WSL) [25].
 - **MuJoCo:** Este es un motor de física que tiene como objetivo facilitar la investigación y el desarrollo en robótica, biomecánica, gráficos y animación, aprendizaje automático y otras áreas. Es una librería de C/C++. Es compatible con sistemas

operativos Linux, Windows y macOS. Se caracteriza por tener un alto nivel de realismo físico, manteniendo una buena eficiencia computacional. Sin embargo, no tiene una interfaz oficial para poder ser utilizado con ROS [26].

- **RoboboSim:** Este es un modelo de simulación realista para el robot móvil educativo Robobo. Este simulador se caracteriza por ser fiel con el modelo del robot real en sus sensores y actuadores. Esto hace que el resultado de la simulación no sea totalmente determinista. Este simulador tiene dos versiones, una gratis y de acceso libre y otra versión PRO. La diferencia entre ambas es que la versión de acceso libre tiene capacidades de edición limitadas. Otra característica importante es que el simulador es compatible con Windows, Linux y macOS, es que es programable con Python y es compatible con ROS [27].

- **Interpretar el lenguaje del humano**

- **Modelo de Inteligencia Artificial que extraiga acciones concretas en el propósito:** En este caso, se propone utilizar procesamiento de lenguaje natural, específicamente, un modelo de extracción semántica de textos, en el cual se extraiga información vital como los verbos y sustantivos. Pues de esta forma se podría dividir un propósito complejo en una serie de tareas o acciones simples. Existen herramientas como la biblioteca *spaCy* [28] o la arquitectura de redes neuronales expuesta en [29]. La ventaja que tendría un modelo como este, es que es simple de implementar. Además de que se podrían almacenar y ejecutar los modelos ya entrenados en local. Mientras que la desventaja es que el usuario debe expresar claramente el propósito para que sea interpretado correctamente.
- **Caja de texto guiado basado en reglas:** En este caso se propone un modelo de un asistente tradicional basado en reglas comúnmente utilizado para el servicio al cliente en ciertas compañías. De acuerdo con [30], este tipo de soluciones tienen un gran desempeño cuando las tareas que tienen que realizar son específicas o repetitivas. En este caso, se propone utilizar este modelo para lograr una interacción con el humano para obtener detalles de su propósito. Entre las ventajas que tiene este modelo es que son relativamente fáciles de implementar. Igualmente, esta solución se podría ejecutar en local y no debería requerir una alta capacidad computacional. La desventaja que tiene es que no tiene forma de evaluar la calidad o el nivel de detalle que recibe por parte del usuario [30].
- **Modelo grande de lenguaje (LLM):** Esta propuesta consiste en utilizar un modelo grande de lenguaje para que interactúe con el humano. En este caso, los modelos LLM ya están entrenados para comprender el lenguaje natural, por lo que serían capaces de interpretar el propósito humano y realizar preguntas para garantizar que el propósito contiene el máximo nivel de detalle posible. Entre las ventajas que presenta esta propuesta está que el sistema puede comprender el significado del texto ingresado por el humano, por tanto validar si tiene sentido, si es coherente, si está bien definido. Además, estos modelos son bastante flexibles y adaptables, por lo que se podrían adaptar mejor a trabajar con diferentes tipos de robots, en diferentes dominios. Entre sus desventajas está, que dependiendo

del tamaño y la calidad de de los datos utilizados para su entrenamiento pueden ser mejores o peores en ciertas tareas. Por lo general requieren un alto consumo computacional, por lo que es complicado ejecutarlos en local [31].

- **Traducir el propósito del humano en misiones:** Una vez que se obtiene el propósito del humano procesado, se obtienen las misiones definidas en un formato compatible con el utilizado en la arquitectura.
 - **Modelo de clasificación en misiones predefinidas:** Esta propuesta consiste en recibir como entrada el propósito dividido en tareas simples e implementar un modelo de clasificación que los relacione con etiquetas de misiones predefinidas. Estas serían acciones comunes y genéricas que puede realizar un robot. Para esto se podría entrenar un modelo desde cero o utilizar herramientas como *Scikit-Learn text classification*. La ventaja de este modelo, es que es simple de implementar y no requiere un alto consumo computacional [32]. Por ende, puede almacenar y ejecutar en local. Mientras que la desventaja es que las misiones se limitan a acciones concretas predefinidas, entonces se debe considerar una amplia variedad de acciones para que el modelo pueda ser implementado en diversos dominios o condiciones.
 - **Modelo grande de lenguaje (LLM):** En este caso se propone utilizar también un modelo grande de lenguaje que en este caso se encargue de tomar el propósito del humano y genere las misiones en formato de texto y les asigne sus pesos (prioridad). La ventaja que presenta esta propuesta es que no está limitado una serie de acciones predefinidas, por lo que se puede adaptar más fácilmente a otros propósitos. Sin embargo, esto puede considerarse una desventaja al mismo tiempo, ya que se pueden generar misiones que son imposibles de que el robot ejecute, por lo que se debe tener cuidado con este factor.
- **Generar funciones de impulso:** Una vez se tiene el propósito y las misiones definidos, se deben generar las funciones de impulso para evaluar qué tan lejos está el robot de satisfacer o lograr la misión.
 - **Modelo de predicción de sensores con funciones predefinidas:** Para esto se propone generar un modelo de red neuronal que reciba las misiones y que prediga cuáles sensores utilizar para su evaluación. Posteriormente, se generan funciones matemáticas estándar con diferente número de variables, dependiendo de la cantidad de sensores que tenga el robot. Igualmente, la ventaja de este modelo es que sería relativamente fácil de implementar y no requiere una capacidad computacional elevada. Sin embargo, se debe asegurar que los datos utilizados para el entrenamiento del modelo sean representativos. Además, el proceso de toma de datos para el entrenamiento podría tomar tiempo.
 - **Modelo grande de lenguaje (LLM):** En este caso, se propone igualmente utilizar un modelo de lenguaje grande para que genere las funciones de impulso para cada misión. En este caso, el modelo debería conocer todos los sensores que tiene el robot y su rango de posibles valores y de las misiones generadas,

crear funciones matemáticas e infiera los sensores que debe utilizar. La ventaja que tiene esta propuesta, como ya se ha dicho en los casos anteriores, es que los modelos grandes de lenguaje son flexibles y se pueden adaptar a diferentes misiones y sensores. Sin embargo, se debe considerar las capacidades del modelo que se utilice, pues en este caso se necesita que tenga conocimientos matemáticos. Además, se corre el riesgo que genere funciones extremadamente complejas, que se indefinan o que no sean válidas en el lenguaje de programación utilizado.

Exploración sistemática

En esta sección se van a analizar todas las posibles soluciones encontradas para cada subproblema analizado y se descartan las opciones que se consideren que no resuelven el problema de la manera más óptima. Para esto se recurre a la herramienta de árboles de clasificación. En Figura 3.4 se muestran los árboles de clasificación obtenidos.

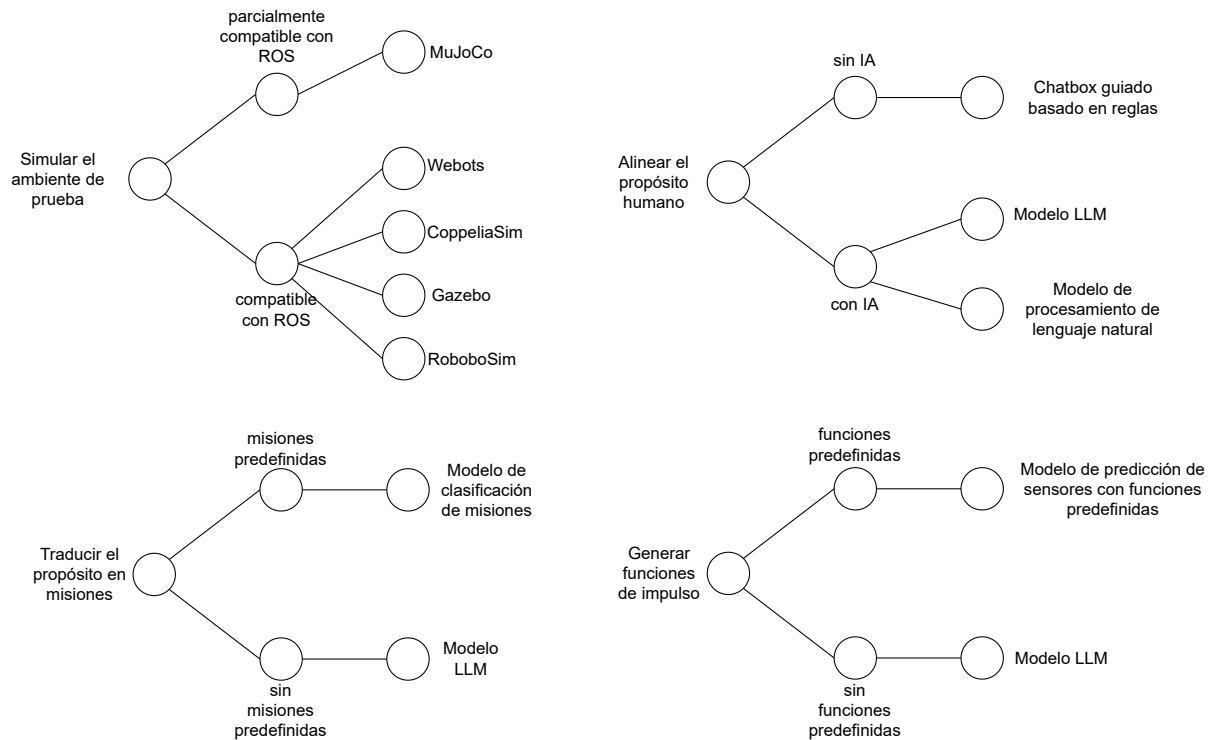


Figura 3.4: Árboles de clasificación obtenidos a partir de la búsqueda externa e interna. Elaboración propia.

De acuerdo con las soluciones propuestas, se necesita reducir el número de opciones para simplificar las posibles combinaciones. En este caso, se van a descartar opciones para el problema de simular el ambiente de prueba. Primeramente, el hecho de que la solución sea compatible con ROS2 es indispensable, por lo que MuJoCo se descartaría. Por otro lado, CoppeliaSim se descarta debido a que en la versión gratuita se tienen capacidades limitadas de edición y está solamente disponible bajo solicitud. Por último, se considera que el problema de alineación y traducción ya es bastante complejo de resolver. Por esta razón,

no es deseable tener que realizar la configuración de una nueva simulación desde cero. Esto no es un trabajo complicado, pero requiere tiempo, por lo que esta sería una razón válida para descartar Webots y de nuevo CoppeliaSim. En este caso, se sugiere utilizar Gazebo, pues el grupo ya ha realizado en el pasado otros experimentos con esta herramienta. Otra opción sería utilizar RoboSim, pues ya se había trabajado con esta herramienta previo al proyecto.

Combinación de conceptos

Por último, tomando en consideración la exploración sistemática realizada, se obtiene la combinación de concepto mostrada en Figura 3.5.

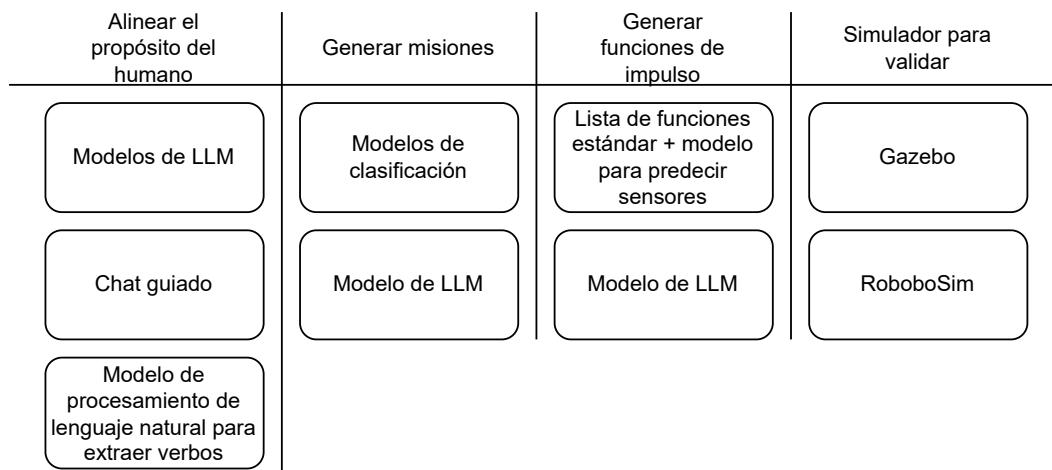


Figura 3.5: Diagrama de conceptos para el sistema de generación de misiones y funciones de impulso. Elaboración propia.

3.2.5 Selección de conceptos

De Figura 3.5, pueden surgir en total 24 combinaciones, sin embargo, estas son demasiadas. Para reducir este número y hacer un mejor análisis, se va a dejar el simulador por fuera, ya que el sistema de traducción de propósito en misiones y funciones de impulso solamente necesita del simulador las percepciones. Estas, por lo general, suelen ser sensores estándar como una cámara, sensores de distancia, entre otros. Además, se obtendrían por medio de ROS2. Por otro lado, en la salida solamente se utiliza para probar el resultado y no influye el resto del diseño. Entonces, ahora serían 12 posibles combinaciones, las cuales se muestran en el apéndice A.1.

Ahora, haciendo un análisis de las combinaciones, se destacan 6 conceptos diferentes como los que se consideran que tienen un mayor potencial. Estos se muestran en la Tabla 3.4.

Tabla 3.4: Conceptos con mayor potencial.

Concepto	Alinear el propósito del humano	Generar misiones	Generar funciones de impulso
A	Modelos de LLM	Modelos de clasificación	Lista estándar + modelo sensores
D	Modelos de LLM	Modelo de LLM	Modelo de LLM
E	Chat guiado	Modelos de clasificación	Lista estándar + modelo sensores
H	Chat guiado	Modelo de LLM	Modelo de LLM
I	Extracción de verbos	Modelos de clasificación	Modelo de LLM
K	Extracción de verbos	Modelo de LLM	Lista estándar + modelo sensores

Para la selección de los mejores conceptos, se van a aplicar dos etapas de selección. La primera de estas es un filtrado y luego la etapa de selección. Para esto, es necesario definir cuáles serán los criterios de evaluación, de acuerdo con las necesidades. En la Tabla 3.5, se muestran las necesidades que se van a utilizar para evaluar estos conceptos y su respectivo criterio de evaluación.

Tabla 3.5: Criterios de evaluación de los conceptos de acuerdo a las necesidades.

ID	Necesidad	Criterio de evaluación
1	El sistema permite que el usuario exprese sus propósitos en lenguaje de alto nivel y es capaz de interpretarlo	Facilidad de interpretación del lenguaje
2	El sistema convierte los propósitos del humano en misiones válidas	Coherencia de las misiones generadas
3	El sistema genera funciones matemáticas de impulso que permiten evaluar el nivel de satisfacción de una misión	Capacidad para representar misiones como funciones matemáticas
5	El sistema realiza acciones de alineación para asegurar que el propósito del humano esté completamente definido	Capacidad para refinar el propósito
6	El sistema prioriza las misiones de acuerdo con la importancia que tenga para cumplir con el propósito asignado y considerando las necesidades ya definidas	Capacidad de priorización de tareas

Con estos criterios definidos, se realiza una primera etapa de filtrado. En este caso, se selecciona un concepto como referencia y para cada criterio, se comparan los demás conceptos contra este: si es mejor se le coloca un “+”, si es igual un “0” y si es peor “-”. El resultado de este filtrado se muestra en la Tabla 3.6.

Tabla 3.6: Matriz de selección de conceptos para la etapa de filtrado.

Criterios de evaluación	A	D	G	H	I	K (Referencia)
Facilidad de interpretación del lenguaje	+	+	-	-	0	0
Coherencia de las misiones generadas	-	0	-	0	-	0
Capacidad para representar misiones como funciones matemáticas	0	+	0	+	+	0
Capacidad para refinar el propósito	+	+	+	+	0	0
Capacidad de priorización de tareas	-	0	-	0	-	0
Suma +	2	3	1	2	1	0
Suma 0	1	2	1	2	2	5
Suma -	2	0	3	1	2	0
Evaluación neta	0	3	-2	1	-1	0
Lugar	3	1	5	2	4	3
¿Continuar?	Combinar	Sí	No	Sí	Sí	Combinar

A partir de esta primera etapa de filtrado, se puede notar que si se combinan los conceptos A y K se puede tener un mejor resultado.

Ahora, se procede a realizar la evaluación de los conceptos, para esto se le asignan pesos a cada criterio según su importancia para la solución. Luego, a cada concepto se le asigna una calificación entre 1 y 5 de acuerdo con qué tan bueno se considera el concepto en ese rubro. Después, se obtiene una evaluación ponderada para cada rubro. Al final, se suman todas las evaluaciones ponderadas y el concepto con mayor puntuación se considera el ganador. En la Tabla 3.7, se muestra el resultado obtenido.

Tabla 3.7: Matriz de evaluación ponderada de conceptos del sistema propuesto.

Criterios de evaluación	Peso	D		H		AK	
		Calific.	Eval. ponderada	Calific.	Eval. ponderada	Califica.	Eval. ponderada
Facilidad de interpretación del lenguaje	20%	5	1	1	0.2	5	1
Coherencia de las misiones generadas	20%	3	0.6	3	0.6	3	0.6
Capacidad para representar misiones como funciones matemáticas	20%	3	0.6	3	0.6	2	0.4
Capacidad para refinar el propósito	30%	5	1.5	4	1.2	5	1.5
Capacidad de priorización de tareas	10%	3	0.3	3	0.3	3	0.3
Total puntos	100%		4		2.9		3.8
Lugar			1		3		2
¿Continuar?			Desarrollar		No		No

Selección final del entorno de simulación

Luego del proceso de búsqueda interno y externo, se consideraron dos entornos de simulación finales: RoboSim y Gazebo. Para escoger el ganador entre estos, se propone la Tabla 3.8, donde se definen factores importantes para la implementación del proyecto y se puntúan las dos opciones con valores del 0 al 5. En este caso, la puntuación se realiza con base a distintos criterios:

- **Experiencia previa:** Se utiliza el criterio personal.
- **Flexibilidad de configuración:** Se califica en base a la información en las páginas web oficiales [27] y [25].
- **Experimentos previos:** Se califica con base a la documentación del proyecto [33].
- **Variedad de robots:** Se califica con base a la información encontrada en sus páginas web oficiales [27] y [25].

Tabla 3.8: Comparación entre los simuladores RoboSim y Gazebo según distintos criterios de implementación, con calificaciones entre 0 y 5 (donde 5 es la máxima puntuación).

Criterio	RoboSim	Gazebo
Experiencia previa con el simulador	5	0
Flexibilidad de configuración del entorno de simulación	2	4
Experimentos implementados previamente con la arquitectura e-MDB	0	1
Permite variedad de robots	0	5
Total de puntos	7	10

Como se puede apreciar en la Tabla 3.8, la opción ganadora es Gazebo. Las principales razones por las que este obtuvo una mayor puntuación por su flexibilidad en la configuración del entorno de simulación, pues en este se pueden insertar objetos personalizados, se pueden colocar cámaras, entre otras características. Mientras que en RoboSim, los escenarios de prueba ya están preconfigurados y las acciones de edición que se pueden hacer están mucho más limitadas. Por otro lado, RoboSim está limitado solo al robot móvil Robo, mientras que en Gazebo se pueden incluir una amplia variedad de robots, incluido Robo. Por último, un factor muy importante es que en el GII existe un experimento que ya fue previamente implementado con Gazebo y el robot [34] [35]. Si bien, este se debe modificar para que se adapte a como el sistema diseñado define las percepciones, existen políticas pre-aprendidas que se pueden reutilizar.

Reflexión sobre el proceso

Como se puede apreciar en la Tabla 3.7 al combinar los conceptos A y K se generó una mejor solución, incluso mejor que el concepto H, que era el segundo lugar en la Tabla 3.6.

Sin embargo, su evaluación total fue levemente inferior a la del concepto D. Además de este factor, se está afrontando un problema complejo de resolver, por lo que vale la pena utilizar una herramienta tan poderosa como los LLMs para tratar de obtener los mejores resultados. Ahora, para el proceso de diseño de la solución específica también se deben considerar otros factores como: el modelo, tamaño, cantidad de LLMs que se deben utilizar y se debe entrenar con la técnica de “*prompting*” para que realice exactamente lo esperado. Todo esto se va a detallar en el capítulo 4, al igual que la selección del simulador.

3.2.6 Pruebas de validación del concepto

En esta sección se va a realizar el diseño general de las pruebas para validar el sistema. Se van a definir las métricas que se evalúan en cada prueba, así como los factores de influencia para cada prueba. Cabe destacar que esta sección se va a profundizar en el capítulo 5.

Prueba 1: Evaluación del sistema de alineación y traducción

En esta prueba se busca evaluar el rendimiento solo del sistema de traducción y alineación del propósito humano. Acá se va a evaluar la calidad de las respuestas o salidas generadas, así como la capacidad de alineación del sistema. En la Tabla 3.9, se muestran las necesidades que se evalúan en esta prueba junto con su métrica.

Tabla 3.9: Necesidades y métricas evaluadas en la prueba 1.

Enunciado de la necesidad	Métrica
El sistema permite que el usuario exprese sus propósitos en lenguaje de alto nivel y es capaz de interpretarlo	Interpreta propósitos en lenguaje de alto nivel
El sistema convierte los propósitos del humano en misiones válidas	Porcentaje de misiones válidas
El sistema genera funciones matemáticas de impulso que permiten evaluar el nivel de satisfacción de una misión	Porcentaje de misiones de impulso válidas
El sistema realiza acciones de alineación para asegurar que el propósito del humano esté completamente definido	Grado de convergencia entre la descripción final del propósito del usuario y la descripción ideal

Factores de influencia

- **Factores de diseño**

- Complejidad del propósito: Este factor está relacionado con la cantidad de movimientos o acciones que debe realizar el robot para cumplir con el propósito del humano.

- **Factores bloque**

- Nivel de detalle en el propósito: Este factor se relaciona con el nivel de detalle con el que el usuario especifique el propósito al sistema. Entre más detalle le de al sistema, será más fácil obtener las misiones y las funciones de impulso.

Prueba 2: Integración con la arquitectura y el simulador

El objetivo de esta prueba es evaluar la correcta integración del sistema con la arquitectura e-MDB y el simulador. En la Tabla 3.10, se muestran las necesidades que se evalúan en esta prueba junto con su métrica.

Tabla 3.10: Necesidades y métricas evaluadas en la prueba 2.

Enunciado de la necesidad	Métrica
El sistema se incorpora con el código de la arquitectura cognitiva existente en Python y ROS2 y esta se comunica con el robot	La arquitectura recibe las percepciones del simulador
	El robot ejecuta políticas enviadas por la arquitectura cognitiva
El sistema es fiel a los conceptos de la arquitectura e-MDB	Utiliza los conceptos teóricos de la arquitectura

Factores de influencia

- **Factores de diseño**

- Función de impulso a evaluar: Este factor hace referencia a la naturaliza de la función matemática de impulso que se desea evaluar, la y la cantidad de sensores que utiliza.
- Políticas pre-aprendidas: Este factor hace referencia a las políticas o acciones que se necesitan pre-aprender por el robot para ejecutar las pruebas finales.

Prueba 3: Evaluación del sistema con el robot

En esta prueba se busca evaluar si el robot consigue cumplir con el propósito del humano, que los conceptos de la arquitectura e-MDB fueron aplicados correctamente y que la asignación de pesos de las misiones fue correcta. En la Tabla 3.11, se muestran las necesidades que se evalúan en esta prueba junto con su métrica.

Tabla 3.11: Necesidades y métricas evaluadas en la prueba 3.

Enunciado de la necesidad	Métrica
El sistema prioriza las misiones de acuerdo con la importancia que tenga para cumplir con el propósito asignado y considerando las necesidades ya definidas	Porcentaje de veces que el robot logra la misión completa en el orden adecuado
El sistema proporciona retroalimentación clara al usuario sobre cuáles son las funciones de impulso, además de cómo y cuándo se están satisfaciendo	Brinda retroalimentación sobre las funciones de impulso evaluadas
El sistema se prueba en un robot real en un ambiente de simulación	Funciona en un robot real en un ambiente simulado

Factores de influencia

- **Factores de diseño**

- Complejidad del propósito: Este factor está relacionado con la cantidad de movimientos o acciones que debe realizar el robot para cumplir con el propósito del humano.

- **Factores bloque**

- Nivel de detalle en el propósito: Este factor se relaciona con el nivel de detalle con el que el usuario especifique el propósito al sistema. Entre más detalle le de al sistema, será más fácil obtener las misiones y las funciones de impulso.

Capítulo 4

Diseño conceptual del sistema

En este capítulo se van a describir y fundamentar todas las consideraciones de diseño realizadas que permitieron el desarrollo del sistema, con el objetivo de garantizar el cumplimiento de los requisitos funcionales y técnicos establecidos por el cliente. Este proceso refleja la adaptación de los conocimientos teóricos y las habilidades prácticas desarrolladas a lo largo de la formación académica, para lograr generar un sistema capaz de interpretar un propósito humano expresado en lenguaje de alto nivel y traducirlo en misiones y funciones de impulso compatibles con la arquitectura cognitiva e-MDB.

A continuación, se abordan de manera secuencial las diversas etapas seguidas para el proceso del diseño. Además, se refleja cómo cada elección contribuyó a obtener la solución final.

4.1 Diseño general del sistema

Una vez aplicada la metodología, se tiene una idea más clara de cómo debe ser la solución. Lo primero que se puede identificar es que el sistema propuesto se compone de tres fases fundamentales:

1. Sistema motivacional: Este diseño corresponde al objetivo específico 2 del proyecto y se incluyen la alineación, generación de misiones y funciones de impulso.
2. Integración con la arquitectura e-MDB: Esta sería una fase inicial del objetivo específico 3, donde se agrega el sistema motivacional a la arquitectura e-MDB.
3. Integración con el simulador robótico: Esta sería la fase de diseño final correspondiente al objetivo específico 3, la cual corresponde comunicar la arquitectura cognitiva y el sistema motivacional integrados con el simulador robótico.

Para facilitar el diseño y la comprensión del mismo, se va a ir resolviendo cada una de estas partes siguiendo el mismo orden.

4.2 Sistema motivacional: alineación, generación de misiones y funciones de impulso.

Esta primera fase se podría decir que es la más compleja, y el principal objetivo del proyecto. En este caso, se propone una solución como la que se muestra en el diagrama de Figura 4.1.

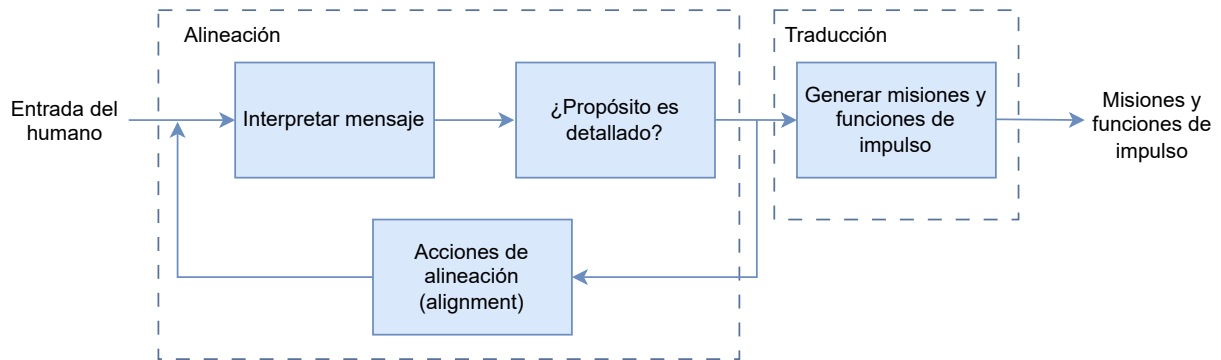


Figura 4.1: Diagrama de bloques de para el sistema de alineación y traducción de propósito. Elaboración propia.

Como se muestra en Figura 4.1, el sistema recibe la entrada del humano. En el momento inicial, sería el propósito que este tiene para el robot. Este debe ser interpretado y se verifica si está lo suficientemente detallado y completo, o si le falta especificar detalles para que quede claro lo que el humano desea. En ambos casos, el sistema aplica una serie de acciones de alineación para asegurarse de que interpretó bien el propósito. En esta etapa, el sistema debe plantear una hipótesis de lo que el humano desea y proponerla al humano. En caso de que el nivel de detalle del propósito sea muy bajo, el sistema realiza preguntas al humano para aclarar el propósito. Cuando ya tenga un buen nivel de detalle se pasa a la etapa de traducción del propósito a misiones y funciones de impulso, las cuales serán las salidas del sistema.

Para resolver cada una de las partes de este sistema, el concepto ganador fue un modelo de lenguaje grande (LLM). Esto implica que el modelo de LLM a utilizar se debe entrenar para que haga solamente las acciones que se necesitan. Para esto se entrena el modelo por medio de una solicitud (*prompt*) inicial. Acá se le da el contexto necesario para que el LLM comprenda qué debe hacer específicamente y se apegue a estas instrucciones. El objetivo de esta fase es hacer que el modelo sea lo más consistente posible, con sus respuestas y no se salga del tema.

Ahora, también, como se propone utilizar modelos LLM, se necesita conocer cuántos modelos o instancias son necesarias para resolver el problema. Por otro lado, como estos modelos se han vuelto sumamente populares en los últimos años, existe una gran variedad de modelos de diferentes compañías, así como diversos tamaños. Además, otro factor importante es la cantidad de detalle que se le ingresa en la solicitud inicial para el entrenamiento. Esto es vital, pues si se le da más contexto del que el modelo es capaz de procesar, no va a hacer lo

que se le pide; pero si, por el contrario, se le da muy poco, hace suposiciones y tampoco va a responder con lo que se espera. Todos estos son factores que se deben tomar en cuenta para generar una solución óptima. Para manejar todas estas consideraciones se va a utilizar un enfoque de pruebas preliminares, en la cual se van a hacer pequeñas pruebas con diferentes soluciones y, al final, se va a escoger la que presente mejores resultados.

4.2.1 Cantidad de instancias de LLMs

Como se va a utilizar los modelos LLMs tanto para la parte de alineación como para la parte de traducción surge la pregunta de cuántas instancias (modelos entrenados) son necesarias para resolver el problema. Esto es un factor a tomar en cuenta, porque se podría utilizar una instancia para que haga todo, se puede dividir en uno para la parte de alineación y luego otro para la traducción. Finalmente, una instancia para la parte de alineación, otro para generar misiones y un último modelo para generar las funciones de impulso.

Esto se basa en la hipótesis de que entre más se divida el problema en tareas específicas, el entrenamiento de los modelos sería concreto, por lo que existe menos riesgo de que el LLM se salga de contexto o responda de una manera indeseada. Esto generaría resultados más coherentes y controlables.

En este caso, se consideran las siguientes tres posibles divisiones, así como algunas de sus posibles ventajas y desventajas.

Un único modelo

Se propone utilizar un único modelo que se encargue de realizar la tarea de interpretación y alineación de propósito y que posteriormente lo traduzca a misiones y funciones de impulso en el formato especificado para la arquitectura.

- **Ventajas:**

1. **Estructura simplificada:** El sistema desarrollado sería muy simple y fácil de implementar en código, ya que contaría con menos componentes y no se debería procesar datos intermedios.
2. **Reducción de errores:** Al utilizar un solo modelo se elimina posibles fuentes de error a la hora de transmitir información entre distintas fases o instancias.
3. **Mayor facilidad de adaptación:** Al ser un sistema simple, que cuenta con un único servicio o *endpoint*, para hacer adaptaciones o mejoras solo se debería cambiar una única solicitud inicial *prompt*.

- **Desventajas:**

1. **Menor control sobre el proceso y el sistema:** Al ser un solo modelo de LLM el que se encarga de hacer todo el proceso se tiene un modelo de tipo caja

negra, lo cual hace que sea difícil controlar, modificar o cambiar partes internas del proceso.

2. **Mayor riesgo de respuestas inconsistentes:** El resultado puede tener mayor variabilidad entre ejecuciones, pues se le está dando una tarea sumamente compleja a un modelo. Además, dependiendo de las capacidades y el entrenamiento del modelo, este puede que le de más énfasis a una tarea en una iteración y luego en la siguiente cambie este énfasis a otra.
3. **Prompt más detallado:** Al recaer todo el proceso en un único modelo se le deben dar todas las instrucciones y restricciones de manera clara y concisa, sin hacer algún tipo de sesgo hacia alguna tarea concreta. Además se le debe dar todo el contexto necesario para que pueda realizar las tareas, pero sin desbordar las capacidades del modelo.

Dos modelos

Se propone utilizar dos modelos o instancias, en la cual la primera se encargue de realizar la tarea de interpretación y alineación de propósito, mientras que la segunda se encargue de traducirlo a misiones y funciones de impulso en el formato especificado para la arquitectura.

- **Ventajas:**

1. **Mayor capacidad para interpretar y alinear el propósito humano:** Al utilizar un modelo solamente para esta tarea, el modelo puede enfocarse en comprender y asegurar que el propósito del humano esté completo. Además se le puede dar un contexto mayor para que haga el proceso de una mejor manera.
2. **Mayor control sobre las respuestas del sistema:** Como cada modelo se enfoca en una tarea específica se puede controlar y delimitar de una mejor manera las respuestas de los modelos. Esto hace que al final se tenga un mejor control sobre lo que hace el sistema y que no se comporte como una caja negra.
3. **Reducción de respuestas desviadas de tema:** Al limitar el alcance de cada modelo, se disminuye también la probabilidad de que cada modelo se desvíe del tema o genere respuestas incoherentes.

- **Desventajas:**

1. **Mayor complejidad en la estructura:** Al utilizar dos modelos, se vuelve más complejo de implementar en el código.
2. **Se debe asegurar la compatibilidad entre respuestas:** Como se están utilizando dos modelos o instancias y la respuesta de uno va a ser una entrada para el segundo, se debe asegurar que estos sean compatibles y que el formato sea el correcto.
3. **Se deben desarrollar más solicitudes con un mayor nivel de detalle:** Como se quiere que cada modelo profundice más en cada tarea, es necesario dar

más contexto y ser más específicos en la solicitud inicial de entrenamiento para que el modelo interprete y se apegue a lo que se desea que haga.

Tres modelos

Se propone utilizar tres modelos o instancias; en este caso, la primera instancia se encargaría de realizar la tarea de interpretación y alineación de propósito. La segunda traduciría este propósito alineado a misiones. Por último, la tercera generaría las funciones de impulso para las misiones.

- **Ventajas:**

1. **Máximo nivel de especialización:** En este caso, cada uno de los modelos se dedicaría a una de las principales tareas del sistema.
2. **Mayor coherencia en las respuestas:** Al separar las fases del proceso, se pueden controlar mejor los pasos intermedios y hacer que el sistema evite ambigüedades o interpretaciones incorrectas.
3. **Mayor facilidad para corrección de errores y modificaciones:** Como cada modelo se especializa en una tarea, se puede identificar en cuál fase está el problema y es más simple hacer el cambio.

- **Desventajas:**

1. **Mayor complejidad en la estructura:** Este caso es aún más complejo de programar y desarrollar conceptualmente.
2. **Prompts más especializados:** Se requiere desarrollar y probar más *prompts* con un nivel de especificación mayor.

4.2.2 Elección del modelo

Además de definir la arquitectura en fases, se escogieron tres modelos de diferentes proveedores y de diferentes tamaños para saber qué tipo de modelo LLM emplear en cada etapa. Esto incluyó evaluar modelos de distintas compañías, con diversos tamaños.

Tamaños y tipos de modelos

Debido al auge que han tenido los modelos grandes de lenguaje en los últimos años, existe una amplia variedad de compañías que ofrecen una gran variedad de modelos. Actualmente, existen miles de modelos en el mercado y cada día lanzan uno nuevo, por lo que es imposible hacer una comparación de todos los modelos. Es por esta razón que se delimitó la comparación a tres modelos que tienen disponibles en el GII.

- **phi4:14b (tamaño pequeño):** Este es un modelo desarrollado por Microsoft, que pertenece a la familia *phi*. Estos modelos se caracterizan por ofrecer un buen rendimiento, con un costo computacional bajo y un precio accesible. Específicamente, el modelo seleccionado tiene un tamaño de 14 mil millones de parámetros (14B). Es un modelo entrenado con bases de datos sintéticas, datos filtrados de páginas web de dominio público, libros académicos y sets de preguntas y respuestas. Además, en este proceso se utilizaron técnicas de alineación y ajuste supervisado para garantizar un rendimiento consistente en las respuestas del modelo [4].

En conclusión, se podría decir que está optimizado para tareas de alineación y razonamiento de instrucciones principalmente en inglés [4]. En Figura 4.2, se muestra una gráfica del rendimiento promedio de varios modelos y se puede notar que su rendimiento es superior incluso que algunos modelos de mayor tamaño.

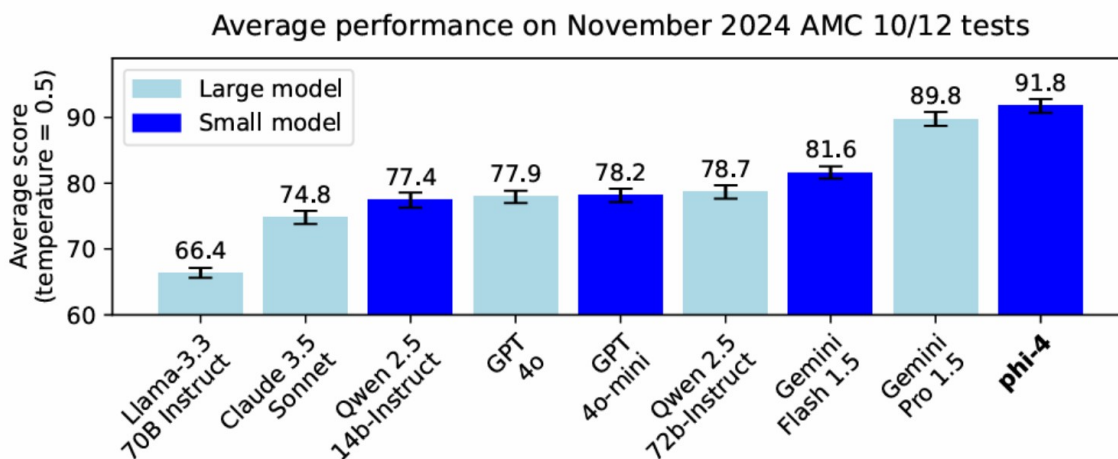


Figura 4.2: Comparación de rendimiento promedio para diferentes modelos de LLM con un contexto de 16 mil *tokens*. Tomado de [4].

- **Qwen3:30b (tamaño mediano):** Este es el último modelo de la familia *qwen*, fue lanzado al público el 28 de abril de 2025. Perteneció a la compañía Alibaba Cloud. También se caracteriza por buscar un balance entre el rendimiento y el costo computacional. Sin embargo, en comparación con el anterior, al tener un tamaño de 30 mil millones de parámetros, permite recibir y generar textos más largos y con mayor contexto. Este modelo es multilingüe y su principal objetivo es ser utilizado como una herramienta para investigadores, desarrolladores y organizaciones para crear soluciones innovadoras [5].

Estos modelos son híbridos, es decir, tienen dos modos de funcionamiento. El modo normal, que proporciona respuestas rápidas, el cual es pensado para preguntas simples donde la velocidad es más importante que la profundidad. Por otro lado, se tiene el modo de razonamiento, el cual se toma tiempo para razonar antes de dar la respuesta final. Esto es ideal para problemas complejos que requieren una reflexión más profunda. Se dice que son híbridos, ya que permite una transición simple de un modo a otro. En

Figura 4.3, se muestran los resultados obtenidos en una serie de pruebas estándar utilizadas para medir *benchmarks* de los modelos LLM. Acá se puede observar que el modelo propuesto tiene un rendimiento superior a otros modelos similares o más grandes [5].

	Qwen3-30B-A3B MoE	QwQ-32B	Qwen3-4B Dense	Qwen2.5-72B-Instruct	Gemma3-27B-IT	DeepSeek-V3	GPT-4o 2024-11-20
ArenaHard	91.0	89.5	76.6	81.2	86.8	85.5	85.3
AIME'24	80.4	79.5	73.8	18.9	32.6	39.2	11.1
AIME'25	70.9	69.5	65.6	15.0	24.0	28.8	7.6
LiveCodeBench v5, 2024.10-2025.02	62.6	62.7	54.2	30.7	26.9	33.1	32.7
CodeForces Elo Rating	1974	1982	1671	859	1063	1134	864
GPQA	65.8	65.6	55.9	49.0	42.4	59.1	46.0
LiveBench 2024-11-25	74.3	72.0	63.6	51.4	49.2	60.5	52.2
BFCL v3	69.1	66.4	65.9	63.4	59.1	57.6	72.5
MultilF 8 Languages	72.2	68.3	66.3	65.3	69.8	55.6	65.6

1. AIME 24/25: We sample 64 times for each query and report the average of the accuracy. AIME 25 consists of Part I and Part II, with a total of 30 questions.
2. Aider: We didn't activate the think mode of Qwen3 to balance efficiency and effectiveness.
3. BFCL: The Qwen3 models are evaluated using the FC format, while the baseline models are assessed using the highest scores obtained from either the FC or prompt formats.

Figura 4.3: Comparación de rendimiento en una serie de pruebas estándar para obtener *benchmarks* para diferentes modelos de LLM. Tomado de [5].

- **GPT-4.1 (tamaño grande):** Esta es una versión mejorada del modelo GPT-4.0 del 2024. Este modelo fue optimizado para seguir instrucciones avanzadas, la ingeniería de software y el razonamiento en contextos amplios. Es un modelo de pago diseñado para obtener un alto nivel de confiabilidad y rendimiento a un menor costo [6]. Este modelo fue lanzado en 2025 y hasta la fecha de redacción de este documento, no existe un dato oficial que indique el tamaño del modelo. Sin embargo, este es mucho más grande que el *phi* y el *qwen*. Permite recibir una solicitud (*prompt*) de contexto de más de 1 millón de tokens. En Figura 4.4, se puede observar que su rendimiento supera a los modelos previos GPT-4o en las pruebas de codificación (SWE-bench), comprensión multimodal (MultiChallenge accuracy) y cumplimiento de instrucciones (Internal OpenAI Instructions following eval accuracy) [6].

4.2.3 Configuración del modelo

Esta sección hace referencia a la cantidad y calidad de contexto que se le da a cada modelo o combinación de modelos para ser entrenados en la solicitud inicial. Este es un factor importante que afecta directamente la calidad de la respuesta obtenida y el rendimiento del sistema en general.

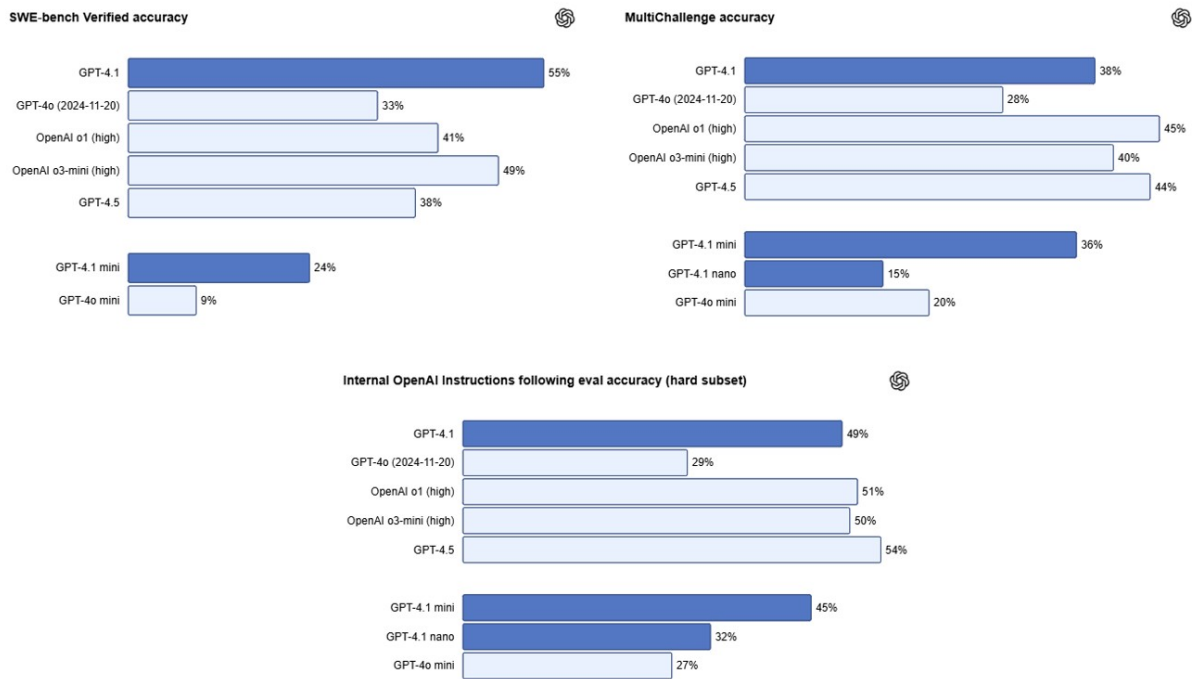


Figura 4.4: Comparación de rendimiento en pruebas estándar (*benchmarks*) para diferentes modelos de OpenAI. Tomado de [6].

Para determinar la mejor configuración de la solicitud inicial, se utilizó un enfoque iterativo basado en prueba y error. Donde se partió de un *prompt* inicial genérico. Posteriormente, se agregaron detalles de forma progresiva según se evaluaba el desempeño del modelo mediante pruebas prácticas. El proceso se detuvo cuando, a pesar de realizar cambios en la descripción inicial, la respuesta no mejoraba (lo cual descarta el modelo y la estructura escogida); o bien, se obtenía una respuesta satisfactoria (indica que el modelo y la estructura funcionan).

Consideraciones para el diseño de la solicitud inicial

En términos generales, independientemente de la cantidad, tamaño y el modelo que se vaya a utilizar, existen una serie de factores básicos que se deben considerar para diseñar la solicitud inicial del sistema.

- **Idioma:** Este es un factor importante, ya que no todos los modelos de lenguaje grande pueden comprender todos los idiomas. En este caso, por requerimiento del GII y por términos prácticos se seleccionó el idioma inglés para el entrenamiento.
- **Explicación del rol general:** Es la parte donde se le indica claramente al modelo el tipo de asistente que debe ser, cómo debe comportarse y cuál será su enfoque principal durante la interacción con el usuario. Además de cuáles son sus limitaciones y capacidades.
- **Contexto teórico sobre conceptos importantes:** Es una breve descripción o

explicación de los conceptos técnicos necesarios para que el modelo pueda comprender adecuadamente la solicitud inicial y lo que debe hacer.

- **Descripción de la tarea por hacer:** Corresponde a una o un conjunto de instrucciones específicas que se le dan al modelo sobre la tarea concreta que debe realizar. Esta parte de la solicitud debe incluir claramente qué información se espera recibir y cuál sería la salida esperada, el formato deseado, restricciones específicas, o detalles importantes para guiar al modelo hacia la respuesta esperada.

4.2.4 Pruebas para el diseño del sistema

Como se mencionó anteriormente, objetivo principal de estas pruebas es identificar el mejor diseño para el sistema de alineación, generación de misiones y funciones de impulso. Para estas pruebas se consideraron los aspectos descritos anteriormente: cantidad de instancias, el tamaño y proveedor de modelo y la configuración de la solicitud inicial. Para cada uno de estos aspectos se llevaron a cabo pruebas comparativas utilizando los mismos casos de prueba representativos.

Situación de prueba

En este caso se propone una prueba de un experimento en el cual se tiene un brazo robótico en una mesa. El objetivo es que el robot tome las frutas, las coloque en una balanza. De acuerdo con el resultado, las clasifica en válidas o inválidas. A continuación, se describe el caso inicial de prueba para los modelos de LLM.

- **Propósito humano completo:**

“The robot should classify fruits by their weight into valid or invalid. The robot should approach the fruit, grasp it and take it to the scale. Then, according to the result the fruit should be placed into the specific basket.”

- **Percepciones del robot:** Inicialmente se realizaron varias pruebas, pues se pensaba utilizar una cámara como sensor y extraer la información específica como: distancia y ángulo entre la fruta y el robot, distancia y ángulo entre la balanza y el robot, estado de la balanza, distancia y ángulo entre las canastas y el robot. Sin embargo, con esto se está resolviendo el problema, es decir, los sensores usados están limitando el dominio en el cual puede operar el robot. Por lo que poco a poco se fue emigrando a una definición más genérica de las percepciones del robot. Las percepciones finales usadas para esta etapa se muestran en la Tabla 4.1:

Resumen de resultados obtenidos

En esta sección se incluyen dos ejemplos de los resultados obtenidos en las pruebas. Pero además se van a mencionar otras combinaciones que se probaron y que incluyeron resultados

Tabla 4.1: Percepciones iniciales para experimento de prueba de diseño.

Percepción	Descripción
robot_hand.x_position	Posición en X normalizada de la mano del robot.
robot_hand.y_position	Posición en Y normalizada de la mano del robot.
robot_hand.state	Estado o configuración actual de la mano (<i>gripper</i>) del robot: abierta (0) o cerrada (1).
object1.label	Etiqueta del objeto 1. En este caso <i>“fruit”</i>
object1.x_position	Posición en X normalizada del objeto 1.
object1.y_position	Posición en Y normalizada del objeto 1.
object1.color	Color del objeto 1. En este caso sería <i>“red”</i>
object1.diameter	Diámetro normalizado entre 0 y 1 del objeto 1.
object1.state	En este caso este valor siempre es 0.
object2.label	Etiqueta del objeto 2. En este caso <i>“scale”</i>
object2.x_position	Posición en X normalizada del objeto 2.
object2.y_position	Posición en Y normalizada del objeto 2.
object2.color	Color del objeto 2. En este caso sería <i>“white”</i> .
object2.diameter	Diámetro normalizado del objeto 2.
object2.state	Estado de la balanza: 0 indica que está desactivada, 0,5 la fruta es inválida, 1 la fruta es válida.
object3.label	Etiqueta del objeto 3. En este caso <i>“valid_basket”</i> .
object3.x_position	Posición en X normalizada del objeto 3.
object3.y_position	Posición en Y normalizada del objeto 3.
object3.color	Color del objeto 3. En este caso sería <i>“green”</i> .
object3.diameter	Diámetro normalizado del objeto 3.
object3.state	Estado del objeto 3: 0 indica que no hay fruta dentro de la canasta, 1 indica que hay una fruta dentro de la canasta.
object4.label	Etiqueta del objeto 4. En este caso <i>“invalid_basket”</i> .
object4.x_position	Posición en X normalizada del objeto 4.
object4.y_position	Posición en Y normalizada del objeto 4.
object4.color	Color del objeto 4. En este caso sería <i>blue</i> .
object4.diameter	Diámetro normalizado del objeto 4.
object4.state	Estado del objeto 4: 0 indica que no hay fruta dentro de la canasta, 1 indica que hay una fruta dentro de la canasta.

vitales que permitieron descartar combinaciones de las soluciones y escoger la solución que mejor resolvía el problema.

Prueba 1: 1 instancia con el modelo phi4:14b

En este caso se utilizó la instancia mostrada en Figura B.1 para entrenar el LLM para realizar la tarea de alineación y traducción.

El propósito humano se ingresó con un bajo nivel de detalle para comprobar la capacidad del sistema para mejorar el nivel de detalle del propósito humano. A continuación, se resumen los resultados obtenidos para este caso.

- **Propósito inicial:** *“The robot should classify fruits”*

- **Descripción final:**

“The robot should classify fruits.

Here’s how I understand the task:

- *The robot should identify objects using their labels.*
- *It will classify fruits based on weight by placing them on a scale.*
- *The scale provides feedback through a perception state indicating if the fruit is valid or invalid.*
- *The robot sorts the fruits into either a valid or invalid basket.”*

- **Misiones e impulsos:**

– Misión 1:

- * Etiqueta: “identify_fruit”
- * Peso: 0.5
- * Impulso: ecuación 4.1

$$D_{M1} = \begin{cases} 0, & \text{si “fruit” in } object.label \\ 1, & \text{en otro caso} \end{cases} \quad (4.1)$$

– Misión 2:

- * Etiqueta: “grasp_and_weigh”
- * Peso: 1.0
- * Impulso: ecuación 4.2

$$D_{M2} = \begin{cases} 0, & \text{si “valid” in } object.state \text{ or “invalid” in } object.state \\ 1, & \text{en otro caso} \end{cases} \quad (4.2)$$

– Misión 3:

- * Etiqueta: “sort_fruit”
- * Peso: 0.7

- * Impulso: ecuación 4.3

$$D_{M3} = \begin{cases} 0, & \text{si } gripper == 1 \text{ and } scale.state \text{ in } ["valid", "invalid"] \\ 1, & \text{en otro caso} \end{cases} \quad (4.3)$$

Prueba 2: 2 instancias diferentes con modelo phi4:14b

En este caso se utilizaron 2 instancias diferentes, una para la parte de la alineación y la segunda para la parte de traducción. En ambos casos el modelo era el mismo, solo que el nivel de especialización de cada instancia se limita a tareas más concretas. En Figura B.2 se muestra la instancia para entrenar el LLM para realizar la tarea de alineación y en Figura B.3 la instancia usada para la traducción.

Igualmente, el propósito humano se ingresó con un bajo nivel de detalle para comprobar la capacidad del sistema para mejorar el nivel de detalle del propósito humano. A continuación, se resumen los resultados obtenidos para este caso.

- **Propósito inicial:** *“The robot should classify fruits”*
- **Descripción final:**

The robot’s task is to classify fruits based on their weight using a scale. The process involves the following steps:

- **Identify and Pick Up Fruit:** *Locate each fruit (object2) on the table.*
- **Weigh the Fruit:** *Place the fruit on the scale (object3).*
 - * *If the scale’s state is 1, the fruit is valid.*
 - * *If the scale’s state is 0.5, the fruit is invalid.*
 - * *If the scale’s state is 0, there is no fruit on it.*
- **Sort the Fruit:**
 - * *Move valid fruits to the valid basket (object4).*
 - * *Move invalid fruits to the invalid basket (object5).*

The robot will repeat this process for each fruit until all have been classified and sorted accordingly.

- **Misiones e impulsos:**
 - **Misión 1:**
 - * Etiqueta: “pick_fruit”
 - * Peso: 0.3
 - * Impulso: ecuación 4.4

$$D_{M1} = 1 - (object2.state + robot_hand.state) \quad (4.4)$$

- **Misión 2:**

- * Etiqueta: “weigh_fruit”
- * Peso: 0.4
- * Impulso: ecuación 4.5

$$D_{M_2} = (1 - \text{object3.state}) \cdot |\text{object2.x_position} - \text{object3.x_position}| \\ + (1 - \text{object2.y_position} - \text{object3.y_position}) \quad (4.5)$$

– **Misión 3:**

- * Etiqueta: “sort_valid_fruit”
- * Peso: 0.5
- * Impulso: ecuación 4.6

$$D_{M_3} = (1 - \text{object4.state}) \cdot |\text{robot_hand.x_position} - \text{object4.x_position}| \\ + (1 - \text{robot_hand.y_position} - \text{object4.y_position}) \quad (4.6)$$

– **Misión 4:**

- * Etiqueta: “sort_invalid_fruit”
- * Peso: 0.6
- * Impulso:

$$D_{M_4} = (0.5 - \text{object3.state}) \cdot |\text{robot_hand.x_position} - \text{object5.x_position}| \\ + (0.5 - \text{robot_hand.y_position} - \text{object5.y_position}) \quad (4.7)$$

En esta etapa de pruebas se notó que cuando los modelos debían hacer más de una tarea, ya sea alineación, generar misiones o generar impulsos, parecían tener un rendimiento menor. Esto se evidencia claramente en las pruebas de ejemplo. El sistema tenía mayores problemas al generar las funciones de impulso, pues en algunos casos los modelos de LLM se inventaban sensores que no estaban en la lista de percepciones dada. Por lo que el dividir las tareas fue mejorando poco a poco.

Luego, se fue probando de los modelos más pequeños a los más grandes. Sin embargo, se notó que la parte de generar funciones de impulso no tenía un buen rendimiento con los modelos *phi4:14b* y *Qwen3:30b*. Además de que los resultados no eran muy consistentes entre dos pruebas diferentes, inclusive si se utilizaban los mismos propósitos y misiones.

Luego, también se le incorporó la capacidad de recibir retroalimentación para modificar las funciones de impulso. No obstante, los resultados no eran satisfactorios. Luego, se optó por el modelo más grande *GPT-4.1* y se comenzaron a obtener mejores resultados, pero se tuvo que trabajar en la solicitud de entrenamiento para tener resultados aceptables.

Selección final

Como se mencionó anteriormente, este fue un proceso iterativo, en el cual se probaron una gran cantidad de combinaciones de modelos y diferentes solicitudes de entrenamiento. Acá

solamente se mostraron los resultados más destacables de este proceso para que el lector comprenda el procedimiento seguido y cómo se llegó a la solución final. Por tanto, tras analizar todos estos resultados, se optó por la siguiente configuración final del sistema:

- **Cantidad de instancias:** Tres instancias. Se determinó que era vital separar el problema lo máximo posible, de manera que las tareas que hiciera cada instancia fueran sumamente específicas. Con esto se logró que los resultados fueran consistentes, que las respuestas estuvieran en el formato adecuado y que el modelo no se saliera de contexto.
- **Modelos LLM seleccionados:** Se decidió utilizar **phi4:14B** para la etapa de alineación y la de generación de misiones. La razón es que logró un buen rendimiento y como es el modelo más pequeño, implica un menor consumo de recursos computacionales. Luego, para la generación de funciones de impulso se seleccionó el modelo **GPT-4.1**. Esta fue la tarea más costosa de programar, pues para garantizar que las funciones de impulso fueran correctas se necesita dar una gran cantidad de contexto. Durante las pruebas los modelos más pequeños simplemente no eran capaces de generarlas correctamente.
- **Configuración de la solicitud inicial inicial:** Luego de seleccionar la estructura y los modelos que se iban a utilizar por medio de pruebas preliminares, se procedió a mejorar las solicitudes iniciales, agregar más detalles y corregir pequeñas especificaciones de formato de la salida. Las solicitudes finales utilizadas se muestran en el Apéndice C.

4.3 Lógica de implementación del sistema e interfaz de interacción

El sistema fue implementado en Python, este fue estructurado en varios módulos que se comunican entre sí para permitir la interacción con modelos de lenguaje grande (LLMs). La lógica general consiste en generar una interfaz gráfica, de estilo “*chat*” que le permita al usuario interactuar con el sistema. En la Figura 4.5, se muestra un diagrama de flujo general sobre la lógica de programación seguida en Python.

Como se puede apreciar en la Figura 4.5, todo el código se ejecuta de manera secuencial. En este se van llamando a los diferentes procesos o etapas, con los modelos respectivos de LLMs. En la Figura 4.6 se muestra la lógica que se sigue en cada etapa del sistema. Es importante destacar que el modelo *phi4:14b* se ejecuta “localmente” en un servidor presente en el Campus de Coruña de la Universidade da Coruña. Por otro lado, el modelo *GPT-4.1* se ejecuta con el servidor de pago *OpenRouter* y es necesario obtener una *API key*. Esto implica que el sistema se debe ejecutar en un dispositivo con acceso a internet. Además, para conectarse al servidor local de la universidad, es necesario estar conectado a la red de la universidad o la red local GII en Ferrol.

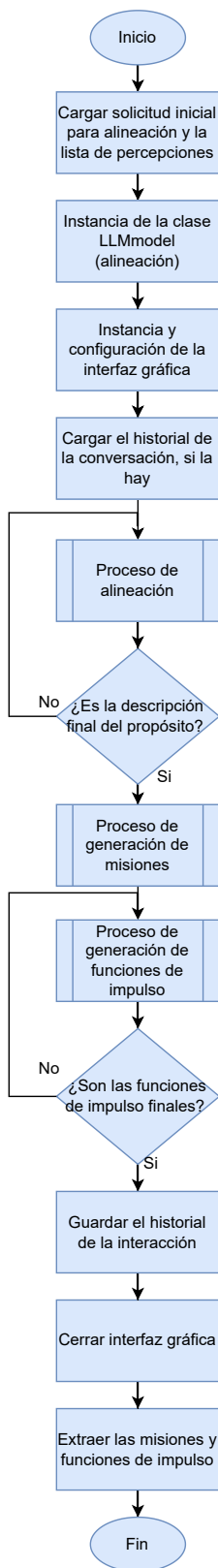


Figura 4.5: Diagrama de flujo de la lógica general seguida en el código de Python para la interfaz de interacción entre el usuario y los modelos LLMs. Elaboración propia.

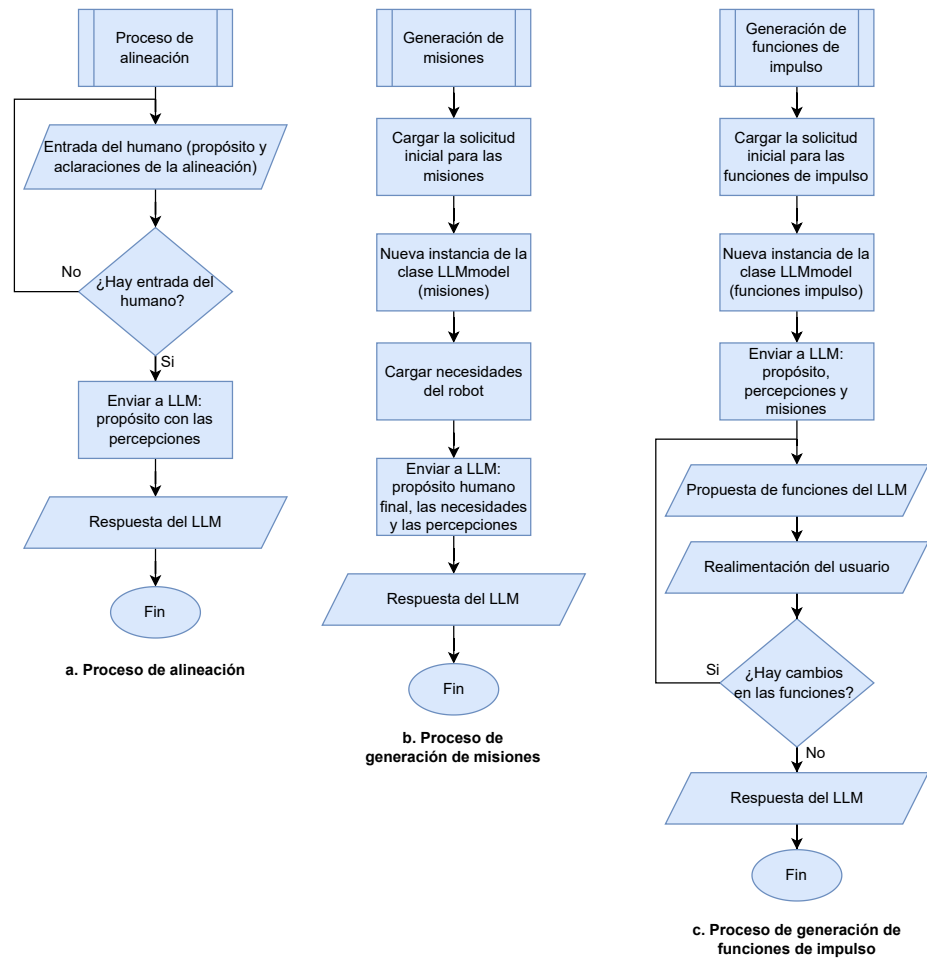


Figura 4.6: Diagrama de flujo de la lógica seguida en Python para los diversos procesos del sistema. Elaboración propia.

Como se mencionó anteriormente, este programa se diseñó para que el usuario pueda interactuar con el sistema. Para esto, se creó una interfaz muy simple, donde el usuario puede escribir en una caja texto su propósito, responder a las preguntas que genera el sistema, dar realimentación, entre otras acciones. En Figura 4.7 se muestra el diseño de la interfaz.

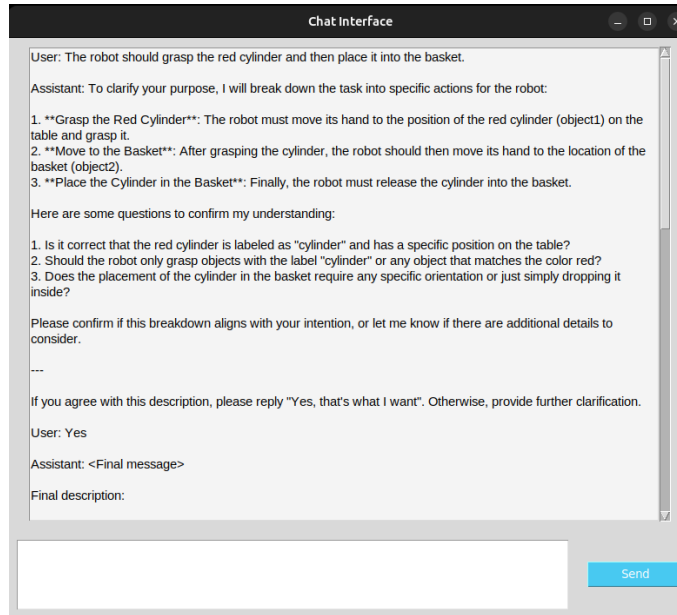


Figura 4.7: Interfaz estilo caja de texto para interactuar con el sistema motivacional. Elaboración propia.

En cuanto a la lógica seguida para el código, primeramente se instancia la interfaz gráfica y se cargan los documentos con el historial de la interacción (si lo hay), las solicitudes de entrenamiento y la lista de percepciones del sistema. Luego, se creó una clase llamada *LLMmodel*, la cual se encarga de crear cada instancia de cada modelo, enviar los mensajes al servidor y devolver las respuestas de los LLMs.

El sistema comienza con la etapa de alineación, en este caso, se espera hasta que el sistema reciba un propósito humano. Luego, inicia el proceso de alineación, en el cual el humano y el LLM interactúan hasta que se obtiene la descripción final del propósito. Posteriormente, se instancia el segundo modelo de LLM, al cual se le envía la solicitud de entrenamiento, la lista de percepciones, la descripción generada en la etapa anterior y la lista de necesidades intrínsecas que tiene el robot. La respuesta de este modelo son las misiones del robot para cumplir con el propósito humano. Seguidamente, se crea la tercera instancia de LLM, a este se le envía: la solicitud de entrenamiento, la lista de percepciones, la descripción del propósito de la primera etapa y la lista de misiones generada en la etapa anterior. Este modelo genera una propuesta de funciones de impulso, las cuales verifica el usuario y puede enviarle retroalimentación sobre estas funciones, en caso de que sea necesario. Si no hay cambios en las funciones, el modelo genera las funciones finales. Por último, se procesan las respuestas para darles el formato adecuado, se guarda la conversación en un archivo YAML y se cierra la interfaz gráfica.

4.4 Integración con la arquitectura e-MDB

La implementación de la arquitectura cognitiva e-MDB está basada en un sistema de nodos de ROS2. Cada uno de estos nodos tiene una funcionalidad diferente e independiente de los demás. Para poder formar la arquitectura, estos nodos se comunican entre ellos por medio de un *topic* (tema) o *service* (servicio). En Figura 2.2 se muestra un esquema conceptual de la arquitectura e-MDB.

En esta figura se puede apreciar cada parte fundamental de la arquitectura cognitiva: *drives* (funciones de impulso), *goals* (objetivos), *world model* (modelo del mundo), *P-nodes* (nodos de percepción), *C-nodes* (nodos de contexto), *utility models* (modelos de utilidad) y *policies* (políticas). En el código de Python, cada una de estas partes corresponde a un nodo de ROS2 independiente.

Para incorporar el sistema diseñado en la arquitectura cognitiva se tuvo que realizar adaptaciones. Primeramente, se generó una división entre las necesidades y las funciones de impulso, ya que en este momento, se consideran dos partes fundamentales que se representan en código como nodos separados de ROS2. Así mismo, se determinó que la mejor forma de adaptar también el sistema a la estructura del código de la arquitectura fue crear una necesidad de alineación (*“alignment need”*), con su respectiva función de impulso (*“alignment drive”*). Estos, junto con el nodo del modelo del mundo, un nodo perceptual y un nodo de contexto activan una política de alineación (*“alignment policy”*). En Figura 4.8 se reflejan estas adaptaciones en la arquitectura.

4.4.1 Adaptaciones a la implementación de la arquitectura e-MDB

Necesidad de alineación

En este caso, al ser una necesidad cognitiva, se utilizó el código existente en la arquitectura para crear este nodo. Para esto, al generar el archivo de configuración en formato YAML del experimento a ejecutar, se crea un nodo de tipo *Need*. Este nodo de tipo *Need* corresponde a una clase de Python, la cual tiene relacionada varios métodos que se utilizan a lo largo de la ejecución de los experimentos. En Figura 4.9, se muestra la información necesaria para la creación del nodo de ROS2.

Función de impulso (*drive*) de alineación

Para crear el nodo de función de impulso relacionado con la necesidad de alineación, se creó una clase hija de la clase *Drive* llamada *DriveAlignment*. Esta clase hereda las características de la clase original y adicionalmente, se le relaciona un método que retorna la evaluación de la necesidad de alineación. Luego, para crear el nodo, solamente se agrega en el archivo de configuración del experimento los parámetros necesarios para instanciar la clase. En Figura 4.10 se muestran los parámetros ingresados para crear este nodo.

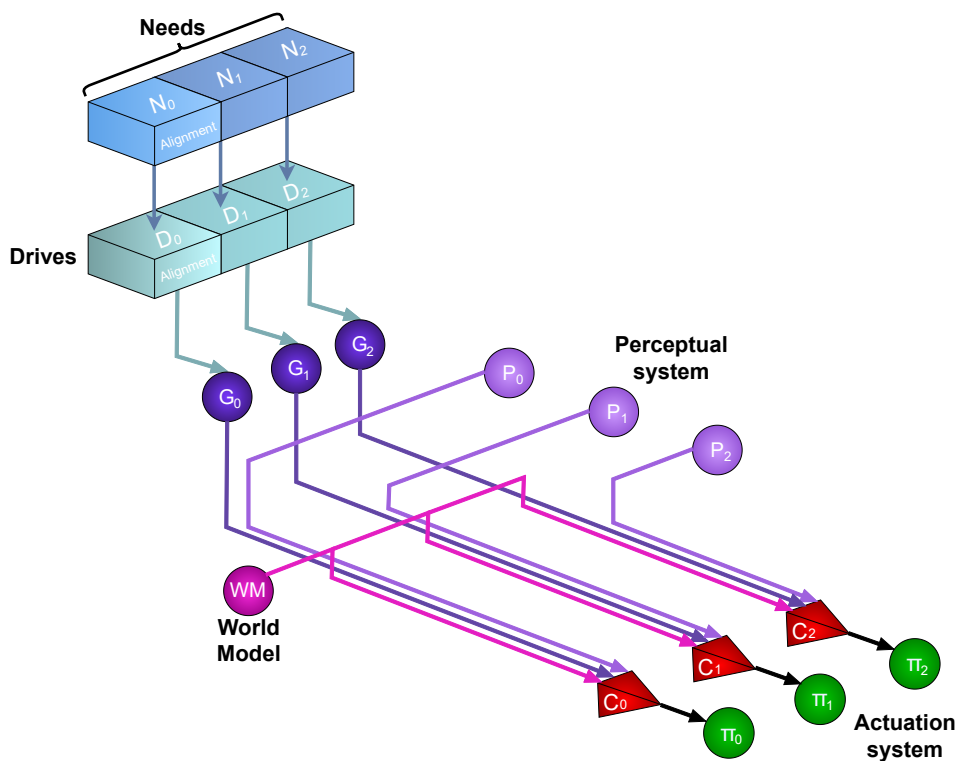


Figura 4.8: Esquema gráfico de la arquitectura e-MDB adaptada para incorporar el sistema motivacional diseñado. Elaboración propia.

Class Need
Attributes used to create "alignment_need": - name: alignment_need - class_name: cognitive_nodes.need.Need - weight: 1.0 - drive_id: 'alignment_drive' - need_type: 'Cognitive'
Methods related to the class: - read_evaluation_callback(self, msg:Evaluation) - set_activation_callback(self, request, response) - get_satisfaction_callback(self, request:IsSatisfied.Request, response:IsSatisfied.Response) - calculate_satisfaction(self) - calculate_activation(self, perception = None, activation_list=None)

Figura 4.9: Parámetros utilizados para la creación de la necesidad de alineación y los métodos relacionados con la clase que representa al nodo de necesidad en ROS2 y Python. Elaboración propia.

Class DriveAlignment
Attributes used to create "alignment_drive": - name: alignment_drive - class_name: cognitive_nodes.alignment.DriveAlignment - neighbors: [{"name": "alignment_need", "node_type": "Need"}]
Methods related to the class: - evaluate(self, perception=None)

Figura 4.10: Parámetros utilizados para la creación de la función de impulso relacionada con la necesidad de alineación y los métodos relacionados con la clase que representa al nodo en ROS2 y Python. Elaboración propia.

Política de alineación

Para implementar en Python y ROS2 el nodo relacionado con esta política, se creó una nueva clase llamada *PolicyAlignment*. Esta hereda todas las características y métodos originales de la clase original *Policy*. Adicionalmente, se crearon los métodos *execute_allback*, *userchat* y *create_missions_drives*. En Figura 4.11, se muestran los parámetros necesarios para crear el nodo de la política de alineación a la hora de ejecutar el experimento.

Class PolicyAlignment
Attributes used to create "alignment": - name: alignment - class_name: cognitive_nodes.alignment.PolicyAlignment - service_msg: cognitive_node_interfaces.srv.Policy - service_name: /mdb/oscar/executed_policy - neighbors: [{"name": "alignment_need", "node_type": "Need"}]
Methods related to the class: - execute_callback(self, request, response) - user_chat(self) - create_missions_drives(self, missions, drives)

Figura 4.11: Parámetros utilizados para la creación del nodo de ROS2 para la política de alineación en Python. Elaboración propia.

Para profundizar más en lo que realiza esta política, en el momento en que se activan todos los nodos relacionados con la alineación, se activa el nodo de contexto. Este es quien activa el nodo de la política de alineación. Esta se encarga de ejecutar el sistema motivacional diseñado, extraer y crear los diferentes nodos de las misiones y sus respectivos nodos de funciones de impulso. En Figura 4.12, se muestra un esquema de cómo se vería la arquitectura cognitiva cuando se ejecuta la política.

- **Método “*execute_callback*”:**

Este método se encarga de ejecutar toda la política. Llama a los métodos “*user_chat*” y “*create_missions_drives*”. Una vez crea las misiones y las funciones de impulso respectivas, cambia la activación de la necesidad de alineación.

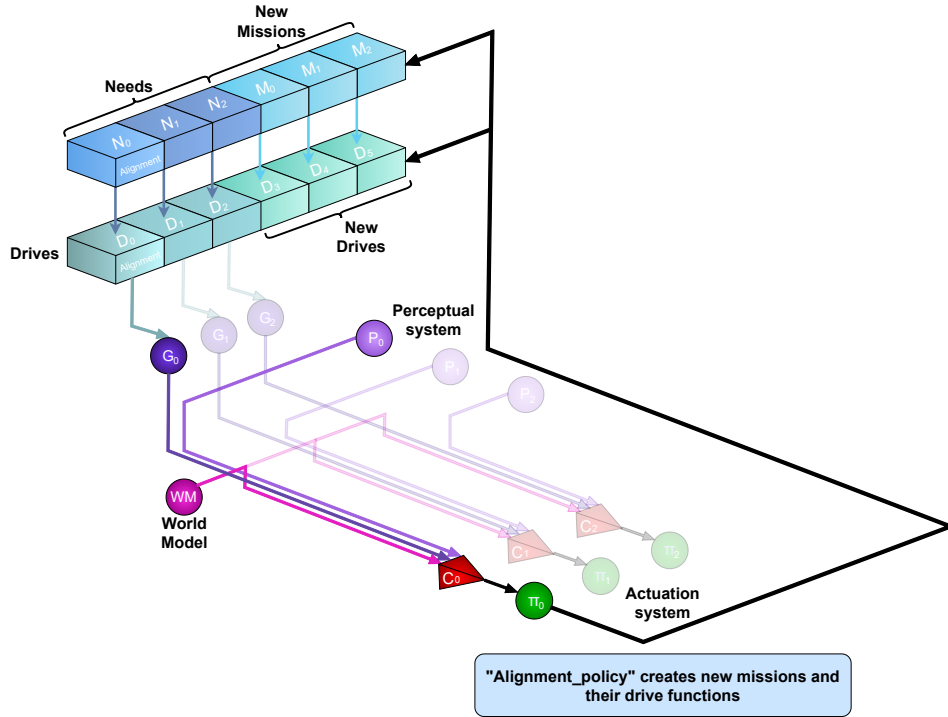


Figura 4.12: Diagrama de la arquitectura cognitiva al ejecutar la política de alineación. Elaboración propia.

- Método “*user_chat*”:

Este método ejecuta el sistema motivacional con los LLMs diseñado y extrae las misiones y las funciones de impulso para satisfacer el propósito del humano.

- Método “*create_missions_drives*”:

Por último, este método se encarga de extraer los parámetros de la respuesta del sistema motivacional y con estos datos crea los nodos de las misiones y las funciones de impulso. Las misiones en este caso son nodos de tipo *MissionLLM* y las funciones de impulso son nodos de tipo *DriveLLM*, los cuales fueron diseñados para esta implementación.

Nodo de tipo *MissionLLM*

Este tipo de nodos se representa por medio de la clase *MissionLLM*, la cual hereda todos los atributos y métodos de la clase *Need*. Fue necesario crear esta clase porque hasta el momento, la arquitectura trabajaba con funciones de impulso binarias, como la que se muestra en la ecuación 4.8. En este caso, la necesidad o misión estaba satisfecha en el momento en que el sensor tuviera un valor de 1 (objeto en la mano).

$$D_{op} = \begin{cases} 0 & \text{si hand_gripper} = 1 \\ 1 & \text{si hand_gripper} = 0 \end{cases} \quad (4.8)$$

Para este proyecto, el cliente esperaba obtener con este sistema funciones continuas, que permitan obtener más información sobre qué tan cerca está el estado actual de cumplir una misión. Entre más cercano de 0 sea la evaluación de la función, implica que está más cerca de satisfacer la misión. No obstante, en muchos casos, el resultado puede que no sea un valor exacto. Entonces, a la clase *MissionLLM* se le agregó un nuevo método llamado *calculate_satisfaction(self)*. Este método lo que hace es que cuando se tiene una misión de este tipo, se considera satisfecha cuando la evaluación de la función de impulso sea menor que 0.01.

Nodo de tipo *DriveLLM*

Como se mencionó anteriormente, este tipo de nodos son los que se utilizan para cada una de las funciones de impulso generadas por el sistema motivacional. Igualmente, se utiliza una clase para describirlo, la cual se representa en Figura 4.13.

Class DriveLLM
Attributes used to create "DriveLLM Nodes": - name: string_name - class_name: cognitive_nodes.drive.DriveLLM - drive_function: string_mathematical_function - neighbors: [{"name": need_name, "node_type": "Need"}]
Methods related to the class: - extract_variables(self, drive_function) - create_topics(self) - read_input_callback(self, msg) - extract_attributes(self) - calculate_reward(self) - get_reward(self) - publish_activation_callback(self) - evaluate(self, final_values=None, updated_perceptions=None)

Figura 4.13: Parámetros utilizados para la creación del nodo de ROS2 para cada función de impulso y los métodos relacionados a la clase en Python. Elaboración propia.

- **Método “*extract_variables*”:**

Analiza la función de impulso generada para una misión y extrae las percepciones requeridas para su evaluación.

- **Método “*create_topics*”:**

A partir de los sensores extraídos, genera una lista de *topics* de ROS2 a los que el nodo se suscribe para recibir las percepciones en tiempo real.

- **Método “*read_input_callback*”:**

Recibe los mensajes de percepción desde los *topics*, actualiza los valores de las percepciones necesarias para la evaluación y activa el proceso de evaluación y recompensa cuando todos los datos están disponibles. Esto se hace por medio de las banderas de

las percepciones (usadas para identificar cuando todas las percepciones fueron actualizadas).

- Método “*extract_attributes*”:

Transforma el formato del nombre de las percepciones finales, para facilitar la evaluación matemática de la función de impulso.

- Método “*calculate_reward*”:

Calcula la recompensa en función de la variación del valor de evaluación respecto al ciclo anterior. Si se redujo el valor de la evaluación, da recompensa, caso contrario la recompensa es 0.

- Método “*get_reward*”:

Devuelve el último valor de recompensa calculado junto con su marca temporal. Este método le permite a otros nodos o procesos obtener este valor.

- Método “*publish_activation_callback*”:

Este método publica de forma periódica el valor de activación de la función de impulso. Este valor se obtiene a partir de la evaluación actual de la función y la activación de nodos vecinos (la misión).

- Método “*evaluate*”:

Realiza la evaluación de la función de impulso. Para esto utiliza las percepciones obtenidas en el método “*read_input_callback*” una vez que todas fueron actualizadas. Además, incluye la marca de tiempo en la que se calculó esta evaluación.

4.5 Integración del sistema motivacional y la arquitectura con el simulador robótico

Como ya existía un experimento previo donde se utilizaba la arquitectura e-MDB, se utilizó como punto de partida para generar la interfaz entre la arquitectura cognitiva y el sistema robótico. En Figura 4.14 muestra un diagrama de bloques de la interfaz y los cambios realizados en cada etapa.

Extracción de señales del simulador

Este bloque representa un nodo de ROS2 en Python, el cual se encarga de extraer los datos crudos del entorno de simulación para que posteriormente sean convertidos en percepciones normalizadas útiles para la arquitectura cognitiva.

El nodo se suscribe a los temas de Gazebo donde se publica la imagen y estado de las pinzas de los brazos robóticos. Luego, procesa las imágenes mediante segmentación en el espacio

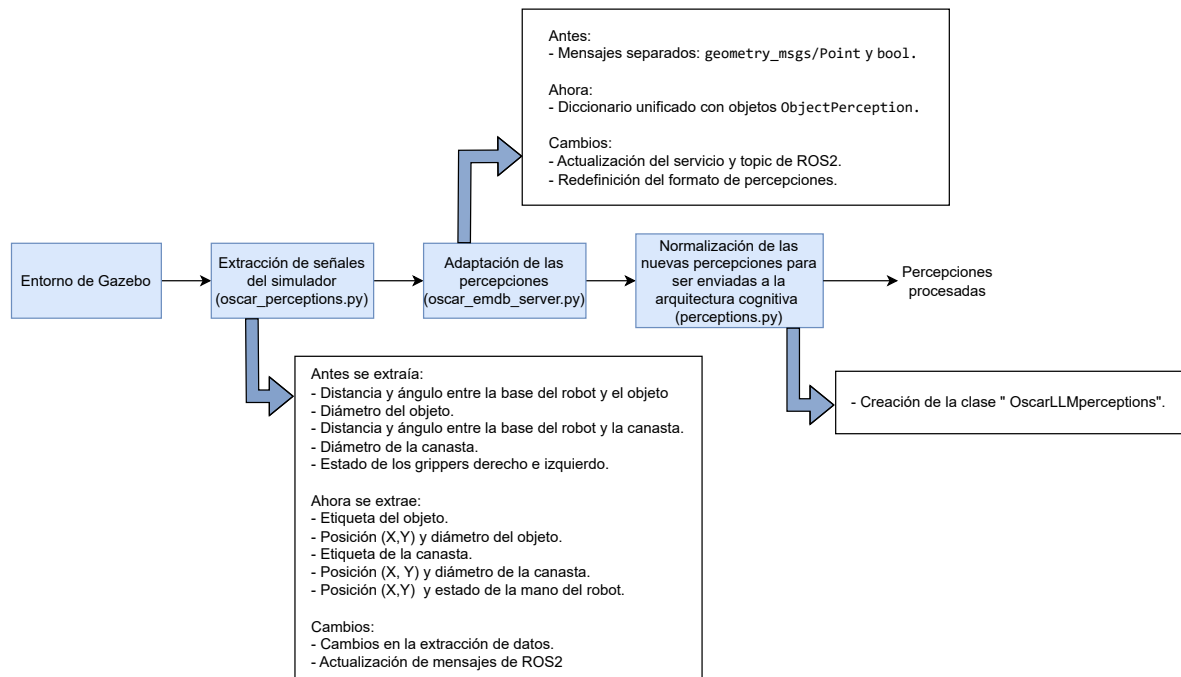


Figura 4.14: Diagrama de bloques de la interfaz entre el sistema robótico y la arquitectura cognitiva inicial y las adaptaciones hechas para incorporarlo al sistema motivacional. Elaboración propia.

HSV para detectar objetos (por ejemplo, cilindros y canastas), y evalúa el estado de las pinzas para identificar si están sosteniendo un objeto.

Como se puede observar en Figura 4.14, en esta parte principalmente se hicieron cambios en las señales que se extraen del simulador Gazebo. Por ejemplo, se cambió de un sistema polar (distancias y ángulos) a un sistema de referencia rectangular (x, y). Además, se incorporaron los métodos que hacen posible la lectura de la posición (x, y) de la mano del robot. Esto se hizo por medio del sistema de transformaciones *TF2* de ROS2, lo cual permite conocer la posición de cualquier frame publicado en ROS2.

Adicionalmente, se modificó el mensaje del *topic* y el servicio que publican los valores obtenidos del entorno de simulación para ser procesados.

Adaptación de las percepciones

En este bloque simplemente se cambió la forma en la que se empaquetaban los mensajes para ser enviados a la arquitectura cognitiva. Inicialmente, se enviaban mensajes separados, cada uno de tipo “geometry_msg”. Se modificaron para que se envíen todos los datos necesarios por cada objeto en un diccionario de tipo “ObjectPerception”. Estos cambios además implicaron una actualización de los servicios de ROS2 para publicar y recibir el mensaje.

Interfaz entre Gazebo y la arquitectura cognitiva

En esta parte se creó una clase nueva llamada *OscarLLM*, la cual hereda todos los atributos y métodos de la clase madre *Perception*. Sin embargo, se creó el método llamado *process_and_send_reading*, el cual se encarga de todo el proceso de extraer los valores del mensaje de ROS2, normalizarlos y enviarlos cada cual por su propio *topic* para que sean ahora sí, utilizados en la arquitectura cognitiva.

Capítulo 5

Resultados y Análisis

En este capítulo se presentan los resultados obtenidos y el análisis correspondiente del sistema motivacional diseñado para traducir propósitos humanos en misiones y funciones de impulso comprensibles por la arquitectura cognitiva e-MDB. El objetivo de esta sección es evidenciar la capacidad del sistema para alinear un propósito humano y robot, descomponerlo en misiones con diferentes niveles de prioridad y generar funciones de impulso matemáticas que permitan la ejecución autónoma de un experimento simulado.

Para esto, se propone realizar tres pruebas principales:

1. **Evaluación del sistema de alineación y traducción:** Esta prueba valida únicamente la efectividad del sistema motivacional diseñado; además, validará el objetivo específico 1 del proyecto.
2. **Evaluación de la integración del sistema motivacional con la arquitectura y el simulador:** Esta es una prueba de verificación que los tres elementos fueran incorporados correctamente, con la cual se validará el objetivo específico 2 del proyecto.
3. **Evaluación del sistema con el robot:** En esta prueba se incluyen los resultados simulados en Gazebo con el robot OSCAR de todo el sistema completo. Con esta prueba se pretende validar el objetivo específico 4 del proyecto.

5.1 Evaluación del sistema de alineación y traducción

En esta parte se pretende evaluar la capacidad del sistema diseñado para alinear el propósito humano, generar misiones y funciones de impulso válidas. Acá se se evalúa únicamente el sistema diseñado, sin la arquitectura cognitiva ni el simular. Además, para hacer la evaluación más sencilla, se propone hacer una división en tres pruebas:

1. Evaluación del sistema de alineación de propósito.
2. Evaluación del generador de misiones.

3. Evaluación del generador de funciones de impulso.

5.1.1 Prueba de evaluación del sistema de alineación de propósito.

En esta prueba se va a utilizar únicamente el modelo *phi4:14b*, con la solicitud de entrenamiento mostrada en el Apéndice E. El objetivo de esta prueba es validar la capacidad del sistema para interactuar con el humano y convertir un propósito general, en un propósito que describa detalladamente la tarea que debe realizar el robot.

Descripción y resultados de la prueba

Para la prueba se utilizaron tres propósitos generales diferentes. Para cada una de estas tareas se escribió un propósito completamente detallado, que sería el caso “ideal” al cual debería llegar el sistema y es el que se va a utilizar para comparar los resultados finales. A continuación, se muestran los propósitos generales con sus respectivos propósitos detallados utilizados. Cabe destacar que el propósito ideal se especificó en inglés, ya que este es el lenguaje que se utilizó en el sistema.

1. Clasificar cilindros por colores en canastas.

“The robot must arrange the cylinders by color into baskets. First it has to reach the red cylinder (object2) with its hand, grasp it and pick it up. Then, the robot should move its hand, with the red cylinder picked, to the green basket (object4) and release it into the basket. After that, the robot should move its hand to the blue cylinder and grasp it. Then, the robot should take the blue cylinder (object3) to the position of the yellow basket (object5) and place the blue cylinder into the basket.”

2. Acomodar las manzanas según su color.

“The robot should arrange the apples on the table by their color. First, the robot has to grasp the red apple (object2) and place it on the right side of the table ($y_position > 0.5$). After that, the robot has to grasp the green apple (object3) and place it on the left side of the table ($y_position < 0.5$).”

3. Apilar los cilindros en el centro de la mesa por orden de tamaño (más grande en la base y el más pequeño en la punta).

“The robot’s task is to stack the cylinders in the center of the table by their size (based on their diameters). The largest cylinder goes at the bottom and the smallest one at the top. To do this, first, the robot should grasp the object4 (largest cylinder), move it to the center of the table. After that, the robot should grasp the object2 (second largest cylinder) and place it on top of the object4. Last, it has to grasp the object3 (smallest cylinder), move it to the center of the table and stack it on top of the object2.”

Luego, se propusieron 15 maneras diferentes de expresar estos propósitos con un bajo nivel de detalle, para que el sistema tenga variabilidad. En la Tabla 5.1 se muestran los propósitos utilizados para la tarea de clasificar cilindros por color y en las Tablas D.1 y D.2 se muestran los propósitos utilizados para las otras tareas.

Tabla 5.1: Propósitos con bajo nivel de detalle para la tarea de clasificación de cilindros por color.

N°	Propósito
1	<i>Arrange the cylinders based on their color.</i>
2	<i>Classify all items on the table into their correct containers.</i>
3	<i>Organize objects on the table by color.</i>
4	<i>Put each cylinder where it belongs.</i>
5	<i>Sort all the pieces into the corresponding baskets.</i>
6	<i>Distribute the items by color into appropriate containers.</i>
7	<i>The pieces must be sorted by their color.</i>
8	<i>Place each object in its designated container.</i>
9	<i>Separate the colored items into their respective baskets.</i>
10	<i>Sort the cylinders on the table.</i>
11	<i>Assign each cylinder to the correct basket based on its color.</i>
12	<i>Organize the objects on the table by color.</i>
13	<i>Place all objects in the appropriate baskets.</i>
14	<i>Sort and place the items according to their color.</i>
15	<i>Each colored element has to end up in the right basket.</i>

Para obtener los resultados de esta prueba, se ingresó uno a uno cada propósito, en una instancia nueva y entrenada del sistema. La razón por la cual debe ser una instancia nueva es para que el LLM tenga el mismo contexto cada vez y que una iteración no afecte a la siguiente. Luego, se interactuó con el sistema hasta obtener la descripción final detallada. A continuación, se muestra un ejemplo de la respuesta final generada por el sistema para la tarea de clasificar cilindros por color.

“The robot will move its hand to the red cylinder’s position using its x -position and y -position values, grasp it by changing its state from 0 (not grasped) to 1 (grasped), transport it to the green basket by aligning with the basket’s x -position and y -position, and release it by changing its state back to 0 (not grasped). Then, the robot will move its hand to the blue cylinder’s position using its x -position and y -position values, grasp it, transport it to the yellow basket by aligning with the basket’s x -position and y -position, and release it by changing its state back to 0 (not grasped).”

Una vez obtenidas todas las descripciones finales del sistema de alineación, se calculó la similitud semántica entre el propósito inicial y la descripción final. Para esto, primero se necesita obtener el vector de *embedding* de cada una de las frases en el espacio del LLM. En

este caso, se utilizó el modelo *all-MiniLM-L6-v2*, creado por UKPLab [36]. Este vector es el que mapea el significado de un texto en una representación numérica. Luego, se obtiene la similitud semántica coseno para el propósito de entrada y salida con la ecuación 2.2. Esto se repitió para los 15 propósitos poco detallados y las descripciones finales obtenidas de las tres tareas. Los resultados completos se muestran en el documento de Excel adjunto en el Apéndice E.

A partir de estos resultados obtenidos en el Apéndice E, se generó el diagrama de cajas que se muestra en Figura 5.1 con los datos de similitud semántica para las tres tareas planteadas.

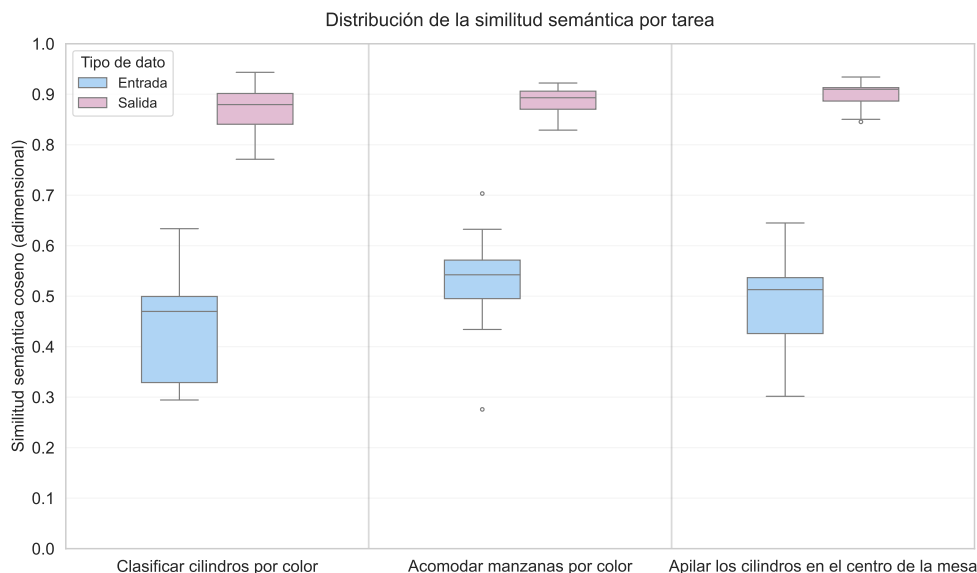


Figura 5.1: Diagrama de cajas con los valores de similitud semántica de los propósitos de entrada y las descripciones de salida generadas por el sistema LLM. Elaboración propia.

Así mismo, en la Tabla 5.2, se muestran los parámetros estadísticos obtenidos para cada tarea.

Tabla 5.2: Estadísticas obtenidas a partir de la similitud semántica de los propósitos de entrada y salida para cada tarea.

[Valores adimensionales]	Clasificar cilindros por color		Acomodar manzanas por color		Apilar cilindros en el centro de la mesa	
	Entrada	Salida	Entrada	Salida	Entrada	Salida
Promedio	0.4331	0.8698	0.5304	0.8851	0.4743	0.8996
Mediana	0.4698	0.8795	0.5424	0.8931	0.5130	0.9098
Desviación estándar	0.1034	0.0504	0.0965	0.0310	0.1017	0.0261
Coefficiente de variación	0.2388	0.0580	0.1819	0.0350	0.2144	0.0290

Análisis de los resultados de la prueba

En la Figura 5.1, se puede notar que los valores de similitud semántica al inicio se encuentran entre los 0.4 y 0.5 para las tres tareas realizadas. Además, es evidente que los datos tienen una variabilidad alta, de 0.1 aproximadamente. Por otro lado, en las salidas del sistema se evidencia que el valor promedio ronda los 0.88 aproximadamente y que la variabilidad entre los datos se redujo significativamente.

Para comprender mejor estos resultados, si se hace una comparación como humano entre el propósito 1 de la Tabla 5.1 y el propósito definido como ideal, dejando de lado todo el contexto. Se podría intuir que hablan del mismo tema, pues en ambos casos se mencionan palabras que están relacionadas como: “acomodar”, “color”, “cilindros”. Sin embargo, en el propósito 1 no se especifica quién debe acomodar los cilindros, cómo se debe hacer ni dónde se debe hacer. Por otra parte, en el ejemplo de la descripción de salida, se especifica claramente que el robot es el que debe acomodar los cilindros, que los debe tomar y colocar dentro de las canastas.

Ahora, retomando la teoría sobre la similitud semántica coseno en el contexto de LLMs, se sabe que esta es una medida de comparación entre los ángulos de dos vectores matemáticos que representan una frase en el espacio del LLM. Esta comparación entre los ángulos es la que indica el nivel de similitud del significado de las dos frases. Se puede observar que hay una concordancia con lo mencionado anteriormente. Los propósitos iniciales tienen un valor de similitud bajo. No es exactamente 0, pues, como se mencionó anteriormente, hay algunas palabras similares. Por tanto, se podría decir que hay una relación entre las palabras; pero no se puede afirmar que se está hablando de lo mismo.

Por otro lado, en la salida se puede notar que la mayoría de los valores se encuentran mucho más cerca de 1. Sin embargo, en ningún caso es exactamente 1, ya que esto solo sucede si las dos frases son exactamente iguales. Como los valores se encuentran entre 0.8 y 0.9, se podría decir que las frases no son idénticas, pero que tienen una mayor relación entre sí. Además, la reducción de la variabilidad en las respuestas es otro indicador de que el significado de las frases se encuentra más limitado y no tan abierto como en los propósitos iniciales. Por ende, se puede tener más certeza de que se está hablando de lo mismo, aunque no se utilicen exactamente las mismas palabras o el mismo orden.

Con estos resultados, se puede asegurar que el sistema es capaz de reducir la brecha semántica que puede existir entre el nivel humano y el robot, sin importar la variabilidad y el nivel de generalidad que exista en la entrada. Esto permite afirmar que existe una alineación de propósitos.

5.1.2 Prueba de evaluación del generador de misiones

En esta prueba se utiliza únicamente el modelo *phi4:14b*, con la solicitud de entrenamiento mostrada en el Apéndice E. El objetivo de esta prueba es evaluar la capacidad del sistema para generar misiones válidas a partir de una serie de propósitos completamente definidos.

Descripción y resultados de la prueba

Igual que en la prueba anterior, se utilizaron las tres tareas descritas. En este caso, se tomaron como entradas las descripciones detalladas generados por el modelo evaluado anteriormente. Esto implica que las 15 entradas para cada tarea eran diferentes entre sí.

Para obtener los resultados, se generó una instancia nueva para cada iteración, a la cual se le daba como contexto la solicitud de entrenamiento en el Apéndice E, la lista de percepciones de cada tarea y la lista de necesidades internas que tenía el sistema. En la Tabla 5.3, se muestra un ejemplo de las misiones generadas por el sistema para la tarea de clasificar cilindros por color.

Tabla 5.3: Ejemplo de misiones generadas por el sistema para la tarea de clasificar cilindros por color.

Misión	Etiqueta	Peso
1	<i>“red_cylinder_in_hand”</i>	0.3
2	<i>“green_basket_with_red_cylinder”</i>	0.5
3	<i>“blue_cylinder_in_hand”</i>	0.7
4	<i>“yellow_basket_with_blue_cylinder”</i>	1

Cabe resaltar que el sistema es capaz de recibir retroalimentación del usuario en caso de que las misiones propuestas sean incorrectas. Sin embargo, para efectos de esta prueba, se contempló únicamente la primera respuesta generada por el LLM, ya que al darle retroalimentación el sistema tiene una efectividad del 100%. Las misiones mostradas en la Tabla 5.3, serían un ejemplo de las misiones iniciales, que además se consideran válidas. Todos los resultados obtenidos se encuentran en el Apéndice E.

Ahora, para determinar la validez de las misiones se deben cumplir los siguientes criterios:

- Las misiones no son acciones demasiado simples.
- El propósito se dividió en misiones clave que permitan alcanzar el objetivo general.
- Las misiones representan un estado final significativo, como “objeto en ubicación” o “objeto en mano”.
- Los identificadores de las misiones no son solo acciones o verbos como “agarrar” o “mover hacia”.
- No se crean misiones como “concluir la tarea” o “terminar tarea”.
- Las misiones tiene una distribución de pesos que indica la priorización de las tareas. Las acciones que se realizan primero cuentan con un peso menor que las finales.

En la Figura 5.2, se muestra un resumen de los resultados obtenidos en las tres pruebas realizadas. En este caso, se contabilizaron las misiones individuales generadas con cada

propósito para cada tarea. De estas, se obtuvo cuántas correspondían a misiones válidas e inválidas.

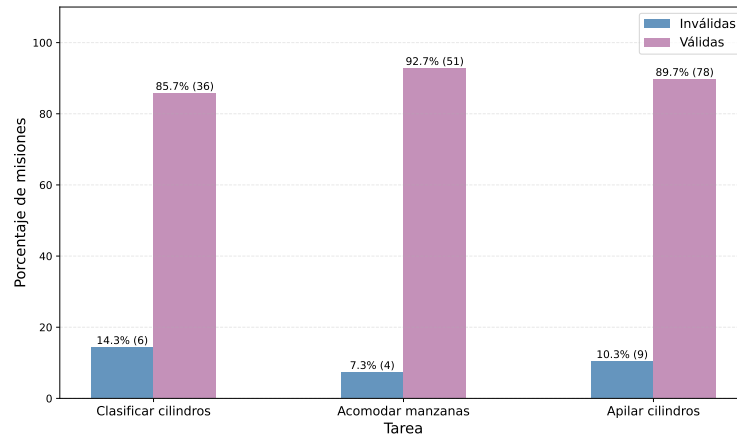


Figura 5.2: Porcentaje de misiones válidas e inválidas generadas para cada propósito y en paréntesis se muestra el total de misiones. Elaboración propia.

Por otro lado, es importante destacar que para cada una de estas tareas, el sistema puede generar diferentes cantidades de misiones que permiten alcanzar el propósito humano. En la Figura 5.3, se muestra una distribución de la cantidad de misiones válidas propuestas por el sistema que permiten lograr el propósito humano.

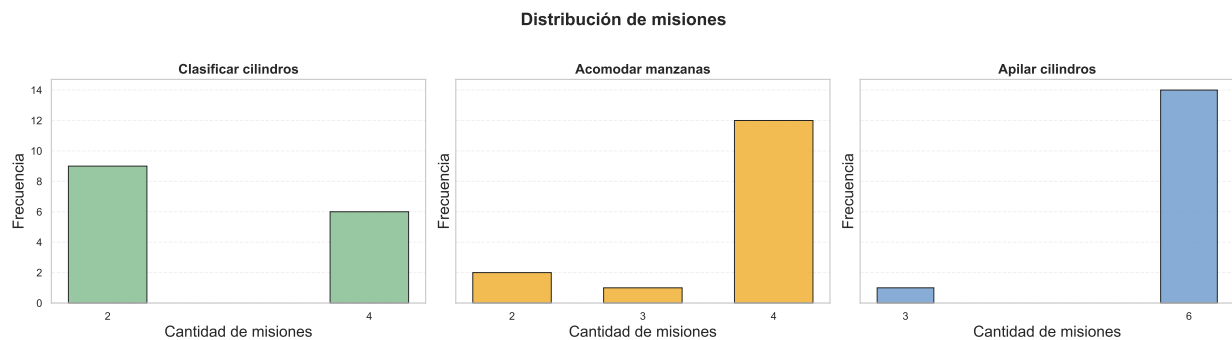


Figura 5.3: Distribución de la cantidad de misiones válidas generadas por el sistema para resolver cada tarea. Elaboración propia.

Análisis de los resultados de la prueba

En los resultados de esta prueba, lo primero que se puede notar es que en los tres propósitos, la cantidad de misiones válidas generadas supera significativamente a las misiones incorrectas. En general, se puede notar que el sistema tiene un buen rendimiento, pero se puede notar que existe una variación entre las tareas.

Por otro lado, también se observa que la cantidad de misiones que genera el LLM para resolver cada tarea es diferente. Esto se evidencia en la Figura 5.2, ya que si se suman la cantidad de misiones correctas e incorrectas para cada tarea, se puede notar que no dan lo

mismo. Se puede notar que la cantidad de misiones necesarias para satisfacer un propósito está relacionado con la complejidad de la tarea, la cantidad de acciones que debe ejecutar el robot para cumplir con el propósito humano y la forma en que esté escrita la descripción.

En la Figura 5.3 se puede observar que para cada tarea el sistema generó en algunas iteraciones 2, 3, 4 o 6 posibles misiones para cumplir con el propósito. Esto se considera válido y posible, siempre y cuando las misiones propuestas cumplan con los criterios de validez definidos y tengan sentido para la arquitectura.

Por ejemplo, para la tarea de clasificar cilindros por color, el sistema en una iteración propuso solamente 2 misiones que evalúan el estado final:

- Misión 1
 - Etiqueta: *“red_cylinder_in_green_basket”*
 - Peso: 0.6

- Misión 2: etiqueta:
 - Etiqueta: *“blue_cylinder_in_yellow_basket”*
 - Peso: 0.8

En este caso, estas misiones se consideran válidas porque no evalúan acciones demasiado simples, el propósito se dividió en misiones clave que representan un estado final significativo, que permiten alcanzar el objetivo final y tienen una distribución de pesos. Además, otro factor importante es que si bien no incluyen una misión específica para tomar el objeto rojo o azul, la arquitectura cognitiva e-MDB es capaz de crear un sub-objetivo relacionado al paso previo para cumplir la misión. Es decir, tener el objeto en la mano se convierte en un sub-objetivo.

En conclusión, se podría decir que el rendimiento promedio del sistema sin realimentación del usuario es aproximadamente 89.4%. Esto implica que el sistema tiene fallos, pero si el usuario tiene cierto nivel de conocimiento técnico podría indicarle al LLM los errores para que los corrija. Para efectos de este proyecto, se puede afirmar que el sistema cumple con el rendimiento esperado.

5.1.3 Prueba de evaluación del generador de funciones de impulso.

Para esta prueba solamente se va a utilizar el modelo *gpt-4.1*, con la solicitud de entrenamiento mostrada en el Apéndice E. El objetivo de esta prueba es evaluar la capacidad del sistema para generar funciones de impulso válidas a partir de un propósito completamente definido y una serie de misiones válidas.

Descripción y resultados de la prueba

Tal como en las dos pruebas anteriores, se utilizaron las tres tareas descritas. Sin embargo, para este caso, se tomaron como entradas las descripciones detalladas generadas por el modelo evaluado anteriormente y las misiones válidas obtenidas de la prueba anterior. Esto significa que se tuvieron también 15 entradas diferentes para cada tarea.

Igualmente, para obtener los resultados, se creó una instancia nueva del modelo en cada iteración. A esta se le ingresaba únicamente la solicitud inicial de entrenamiento (ver Apéndice E, la lista de percepciones de cada tarea, el propósito detallado y las misiones válidas. En las ecuaciones 5.1 a 5.4 se muestra un ejemplo de las funciones propuestas por el sistema. En este caso, estas funciones son para el propósito de clasificar cilindros por color y tienen una correspondencia con las misiones de la Tabla 5.3.

$$D_{M1} = (\text{robot_hand.state} - 1)^2 + (\text{robot_hand.x_position} - \text{object2.x_position})^2 + (\text{robot_hand.y_position} - \text{object2.y_position})^2 \quad (5.1)$$

$$D_{M2} = \text{robot_hand.state}^2 + (\text{robot_hand.x_position} - \text{object4.x_position})^2 + (\text{robot_hand.y_position} - \text{object4.y_position})^2 \quad (5.2)$$

$$D_{M3} = (\text{robot_hand.state} - 1)^2 + (\text{robot_hand.x_position} - \text{object3.x_position})^2 + (\text{robot_hand.y_position} - \text{object3.y_position})^2 \quad (5.3)$$

$$D_{M4} = \text{robot_hand.state}^2 + (\text{robot_hand.x_position} - \text{object5.x_position})^2 + (\text{robot_hand.y_position} - \text{object5.y_position})^2 \quad (5.4)$$

Otro factor importante es que las funciones de impulso iniciales y finales fueron las mismas, ya que desde un inicio eran todas válidas y no se necesitó realizar cambios. Sin embargo, en caso de que una o más de las funciones generadas sea inválida, se le puede indicar al sistema que la elimine o modifique. Todos los resultados obtenidos se encuentran en el Apéndice E.

Igual al caso anterior, la validez de las funciones de impulso se determinó manualmente. Para determinar la validez de una función se siguieron los siguientes criterios:

- La función de impulso debe ser una expresión válida en Python.
- Sus variables corresponden a las percepciones del robot.

- Debe ser ≥ 0 en todo momento (no puede tener valores negativos).
- Es 0 únicamente cuando la misión esté satisfecha.
- Aumenta conforme el robot se aleje del objetivo de la misión.
- No utiliza singularidades.
- Es una función continua, para que el robot pueda evaluar el progreso incremental hacia el objetivo de la misión.
- No son funciones de impulso binarias con valores únicamente de 0 o 1.

En la Figura 5.4, se muestra un resumen de los resultados obtenidos en las tres pruebas realizadas. También, es importante destacar que los casos correctos e incorrectos fueron los obtenidos en la primera respuesta generada por el sistema, sin recibir retroalimentación del usuario.

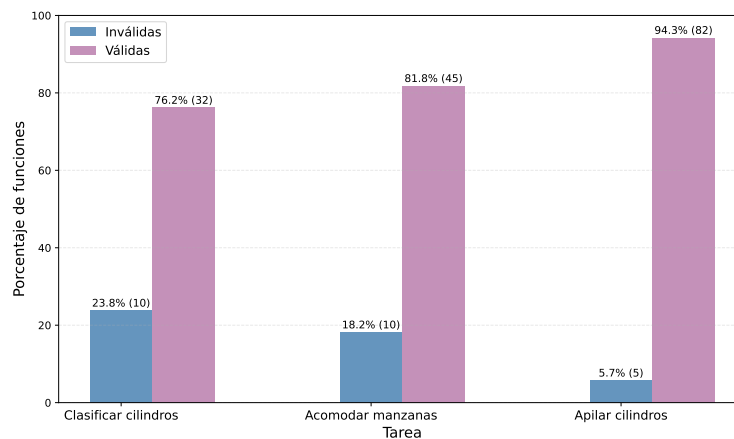


Figura 5.4: Porcentaje de funciones de impulso válidas e inválidas generadas para cada propósito y en paréntesis se muestra el total de funciones para cada propósito. Elaboración propia.

Análisis de los resultados de la prueba

Al igual que en la prueba anterior, se puede notar que la cantidad de funciones de impulso válidas generadas supera significativamente a las funciones inválidas. Esto indica que en la mayoría de los casos el LLM es capaz de crear funciones matemáticas que permiten evaluar las misiones.

Igual que en el caso de las misiones, la cantidad de funciones generadas para cada propósito es variable; es decir, no se generaron la misma cantidad de funciones de impulso para la tarea de clasificar cilindros y la tarea de apilar cilindros, a pesar de que en ambos se utilizaron los mismos 15 propósitos. La razón de esto, es que se genera una función de impulso por

cada misión. Como se explicó anteriormente, la cantidad de misiones varía dependiendo de la complejidad de la tarea y de cómo se exprese la descripción final.

En general, se puede notar que el rendimiento del sistema en las tareas de clasificar cilindros por color y acomodar manzanas parece presentar rendimientos menores. Durante la ejecución del experimento se notó que la mayoría de los casos donde fallaban las funciones de impulso era porque trataba de evaluar varias condiciones opuestas en una misma función. Esto hacía que la función fuera 0 en diferentes momentos y no solo cuando se satisfacía la misión. O bien, el caso contrario, trataba de evaluar dos acciones opuestas en una misma función, como por ejemplo, tomar un objeto y soltarlo. En el primer caso, es un problema porque al implementarlo en la arquitectura le daría recompensa al robot en momentos donde no debería. Mientras que en el segundo caso, más bien no se le daría recompensa al robot nunca. En ambos casos, esto sería un problema porque afecta las capacidades de aprendizaje del robot.

No obstante, si se obtiene el rendimiento promedio del sistema, se nota que es de aproximadamente 84.1%. Esto, para ser una primera aproximación a este problema, se podría decir que es un resultado satisfactorio. Entonces, a pesar de que el sistema no es perfecto a la primera, igualmente permite que el usuario pueda indicarle al LLM los errores para que ajuste las ecuaciones. En conclusión, se podría decir que el sistema cumple con el rendimiento esperado.

Análisis de los resultados generales del sistema motivacional

Como se pudo apreciar, en las tres pruebas propuestas, se obtuvieron resultados satisfactorios y que concuerdan con las métricas propuestas en la Tabla 3.3. En general, se puede decir que la combinación de todo el sistema no es perfecto; sin embargo, con esta prueba queda demostrado que el sistema logra el objetivo principal. El sistema es capaz de alinear el propósito humano con el del robot y es capaz de transformarlo en misiones e impulsos que son válidos y compatibles con la arquitectura motivacional. Con esto, se concluye que se cumplió el objetivo específico 2 del proyecto satisfactoriamente.

5.2 Evaluación de la integración del sistema motivacional con la arquitectura y el simulador

Esta sección tiene como objetivo verificar la comunicación entre el sistema motivacional diseñado, la arquitectura cognitiva e-MDB y el simulador de OSACR en Gazebo. Con esta prueba se pretende verificar que el sistema funciona correctamente y que se pueden ejecutar experimentos concretos.

5.2.1 Descripción de la prueba

La prueba consiste en ejecutar una política que en cada iteración acerca un poco la mano del robot al cilindro rojo y obtiene la evaluación de la función de impulso para cada estado nuevo. En total se obtuvieron 20 posiciones diferentes, que van desde la ubicación inicial del robot hasta tomar el cilindro. En Figura 5.5, se muestra un esquema del experimento.

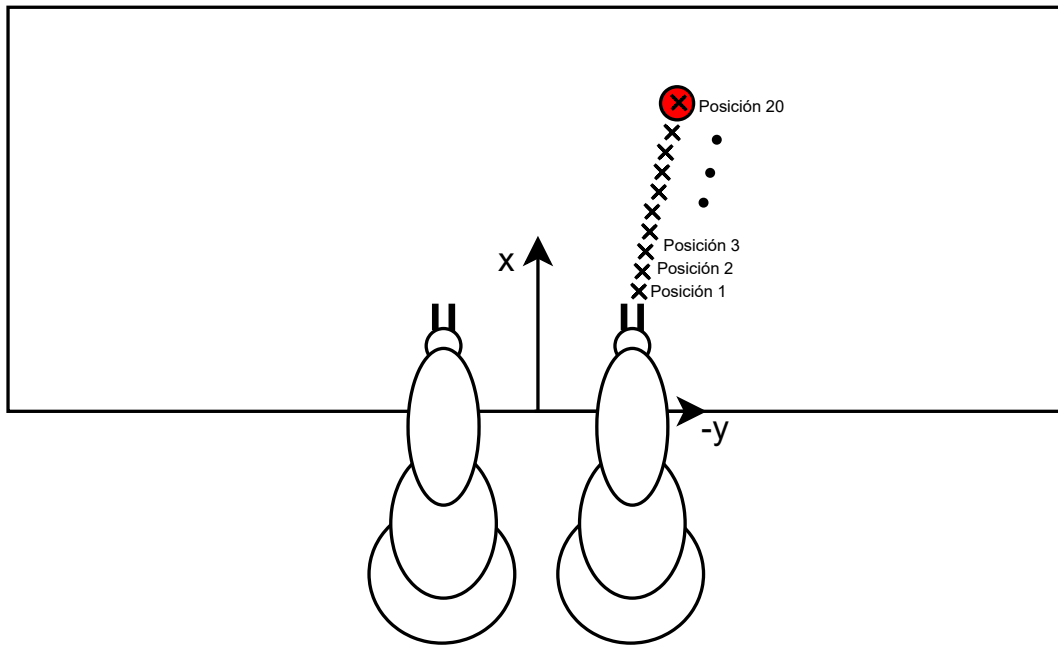


Figura 5.5: Esquema ilustrativo de la política ejecutada para la verificación de la integración del sistema. Elaboración propia.

Por su parte, la Figura 5.6 muestra la evolución del impulso promedio medido en cada iteración junto con su intervalo de confianza del 95%, lo que permite visualizar la variabilidad de los datos y la tendencia al acercar la mano al cilindro.

5.2.2 Análisis de los resultados de la prueba

En este caso, en los resultados de la Figura 5.6 se puede notar que conforme se va acercando la mano del robot al objeto la gráfica se va reduciendo poco a poco, hasta que llega a 1. Luego, en la posición 20 baja abruptamente a 0. Esto sucede porque tal como se puede apreciar en la Figura 5.5, la posición 20 corresponde al momento en el que el robot toma el cilindro. Justo en ese momento, el valor de la pinza del robot cambia de 0 a 1.

Por otro lado, algo que se notó al extraer los resultados es que el intervalo de confianza del 95% es muy estrecho y sigue el mismo comportamiento que la curva ideal. Esto se debe a que los datos provienen del simulador Gazebo, por lo que las variaciones que habían en los

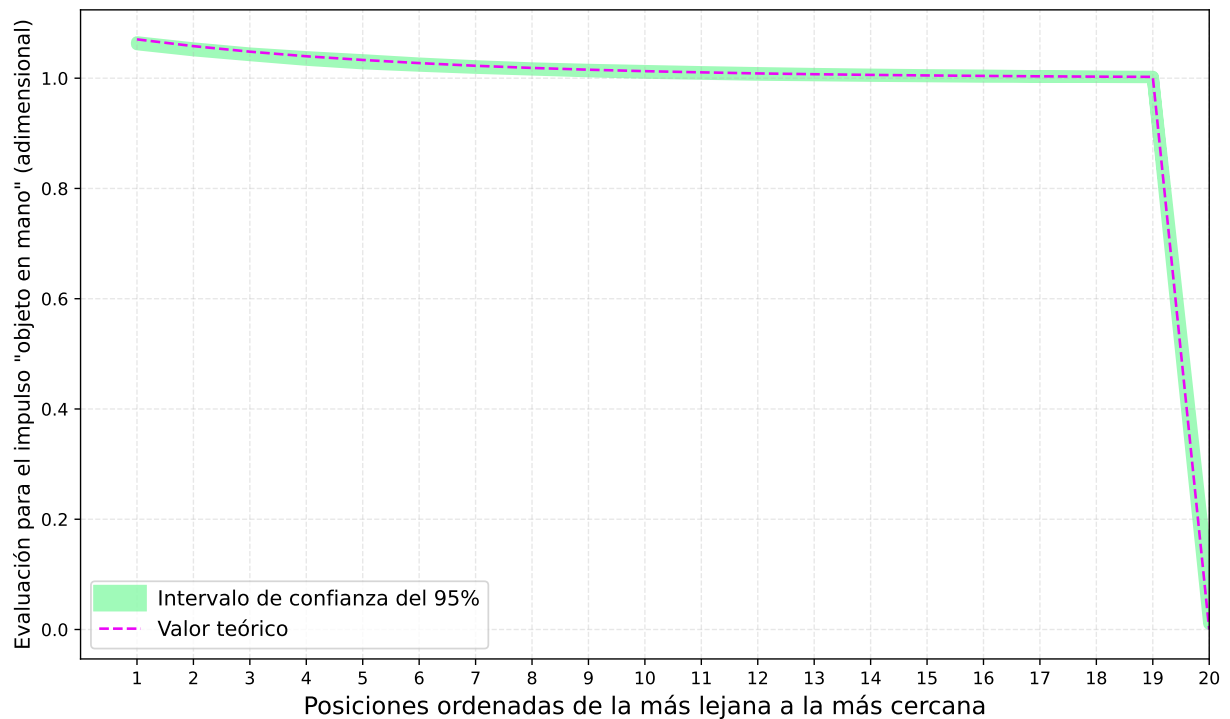


Figura 5.6: Evaluación de la función de impulso “cilindro en mano” para diferentes posiciones ordenadas de la más lejana a la más cercana al cilindro rojo. Elaboración propia.

datos entre las tres repeticiones del experimento era prácticamente nula.

Por último, esta prueba permitió no solo verificar que la función de impulso era válida, sino que también permitió comprobar que el sistema motivacional, el simulador de Gazebo y la arquitectura cognitiva e-MDB están integrados correctamente. Como se puede notar, para poder ejecutar esta prueba el robot debía realizar una política, la cual se envía desde la arquitectura cognitiva al simulador para cada iteración de la prueba. Luego, al moverse el robot ocurre un cambio en los valores de los sensores. Estos a su vez cambian las percepciones que recibe la arquitectura para poder evaluar la misión y el impulso que creó el sistema motivacional. Entonces, a partir de los resultados obtenidos, se puede asegurar que el sistema motivacional se incorporó correctamente con la arquitectura y el simulador Gazebo, lo que permite afirmar que se cumplió el objetivo específico 3 del proyecto.

5.3 Evaluación del sistema con el robot

En esta sección se va a validar todo el sistema funcionando en conjunto en un experimento concreto. En este caso, se busca evaluar la capacidad del sistema motivacional para crear misiones y funciones de impulso en la arquitectura cognitiva e-MDB. Asimismo, se pretende observar cómo, gracias a este sistema motivacional, el robot es capaz de aprender de manera autónoma para cumplir con el propósito humano.

5.3.1 Descripción de la prueba

El experimento propuesto consiste en que el robot tome el cilindro rojo y lo coloque dentro de la canasta azul. En Figura 5.7 se ilustra el entorno simulado en Gazebo con el robot OSCAR. En este caso, se parte del supuesto que el robot ya sabe realizar políticas como: tomar el cilindro rojo, colocar el cilindro dentro de la canasta, presionar el botón verde cuando el cilindro está fuera de su alcance.

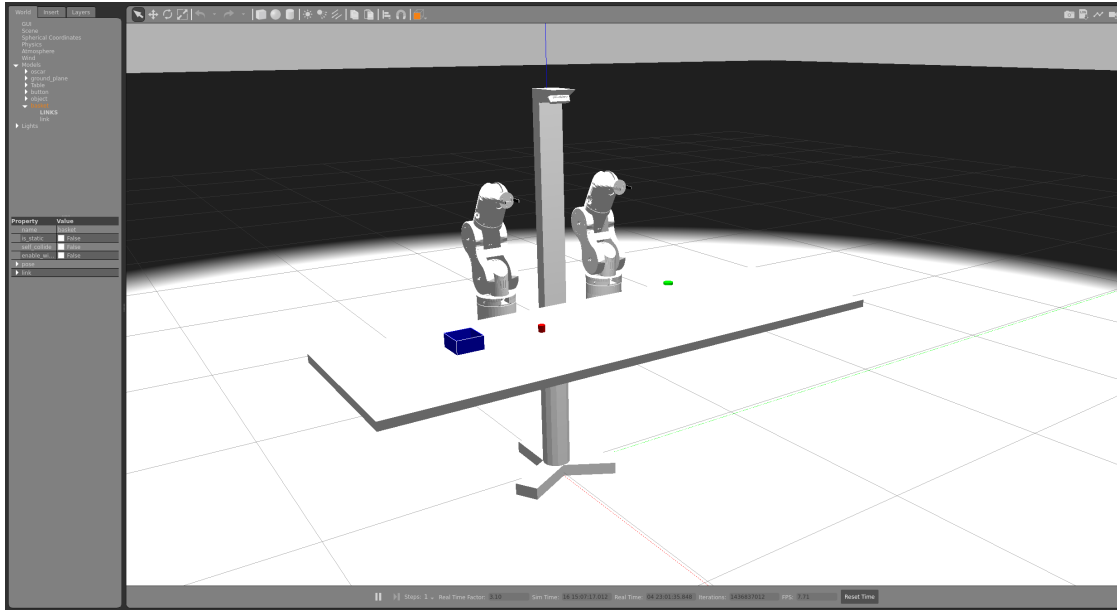


Figura 5.7: Configuración del experimento en Gazebo para la evaluación del sistema. Elaboración propia.

Antes de explicar las pruebas, se van a explicar algunos términos y parámetros importantes para poder comprender correctamente el experimento.

- **Repetición:** Corresponde a las veces que se ejecutó la simulación de Gazebo desde el inicio al final. En este caso se realizaron 10 repeticiones.
- **Iteraciones:** Corresponde a la longitud de cada repetición. En este caso, se definieron 2000 iteraciones para cada repetición. Además, cada iteración corresponde a una política ejecutada por el robot.
- **Intentos:** Corresponde a la cantidad de veces que intentó lograr el objetivo final. Cabe destacar que para conseguir el objetivo final, se necesitan idealmente 2 o 3 iteraciones.
- **Cantidad máxima de iteraciones por intento:** Se definió un límite de máximo 10 iteraciones para que el robot alcanzara el objetivo. Si han pasado 10 iteraciones y el robot no ha conseguido el propósito, el mundo se reinicia y se cambian los objetos de posición.

En este caso, cada repetición, en la iteración 0 se ejecuta la política de alineación creada. En esta, se activa el sistema motivacional diseñado, al cual se le especifica el propósito humano,

se realizan las acciones de alineación correspondientes (en el Apéndice E, se muestran las interacciones para cada caso). Luego, a partir de los resultados del sistema motivacional, la política crea los nodos cognitivos de ROS2 que representan las misiones e impulsos en la arquitectura y el robot comienza a ejecutar las políticas que ya sabe. Esta sería la etapa de aprendizaje, donde el robot intenta conseguir el propósito humano definido. La repetición termina al llegar a la iteración 2000. Esto se repite 10 veces para obtener los resultados.

5.3.2 Resultados generales obtenidos en las repeticiones del experimento

En el Apéndice E, se encuentra un enlace con el historial de conversación obtenido para las 10 repeticiones ejecutadas, así como un vídeo donde se muestra la ejecución del experimento en Gazebo.

En cada repetición se contabilizó la cantidad de intentos que el robot hizo para lograr el propósito humano y cuántas iteraciones le tomó lograrlo. Entonces, a partir de estos resultados, se generó la gráfica que se muestra en la Figura 5.8, la cual presenta la cantidad de iteraciones que le tomó al robot completar el propósito a lo largo del experimento. La curva azul representa un promedio suavizado de los intentos en las 10 repeticiones, mientras que el sombreado celeste indica la variabilidad de los resultados con un intervalo de confianza del 95%. Por su parte, el sombreado de color morado indica el intervalo de iteraciones ideal para lograr el propósito. Por último, cabe destacar que el eje de intentos se limitó a los primeros 50 para facilitar la interpretación del comportamiento.

Ahora, en la Tabla 5.4, se muestra un resumen de los resultados generales obtenidos en las 10 repeticiones ejecutadas.

Tabla 5.4: Tasa de éxito de las repeticiones del experimento con OSCAR ejecutadas en Gazebo.

Experimento	Intentos totales	Éxitos	Fallos	Tasa de éxito (%)
0	603	597	6	99.00
1	681	676	5	99.27
2	674	670	4	99.41
3	680	676	4	99.41
4	680	675	5	99.26
5	686	682	4	99.42
6	570	568	2	99.65
7	675	669	6	99.11
8	688	684	4	99.42
9	685	681	4	99.42

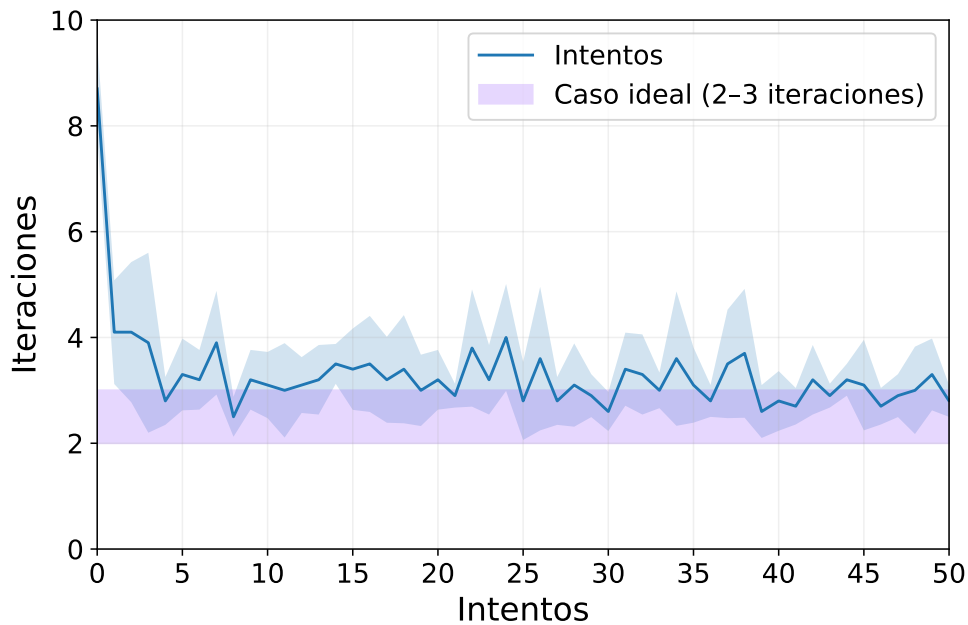


Figura 5.8: Variación en la cantidad de iteraciones necesarias para alcanzar el propósito a lo largo de los intentos. El promedio suavizado y su intervalo de confianza del 95% se muestran en azul, mientras que el área morada representa el rango ideal de 2 a 3 iteraciones. Elaboración propia.

5.3.3 Análisis de los resultados generales

En primer lugar, en la gráfica mostrada en la Figura 5.8, se puede ver que al inicio del experimento en todas las repeticiones le tomaba más iteraciones para lograr el objetivo final (colocar el cilindro rojo en la canasta azul). Esto es esperable, pues como se mencionó anteriormente, al inicio la arquitectura cognitiva no sabe qué es lo que debe hacer para satisfacer sus misiones. Entonces comienza a ejecutar políticas al azar, hasta que ejecuta la secuencia correcta. Conforme el robot se va acercando a cumplir el propósito, se le da recompensa y así es como el robot va aprendiendo.

Luego, conforme avanza el experimento se puede notar que la cantidad de iteraciones que necesita el robot para lograr el propósito humano va disminuyendo hasta estabilizarse entre 2 y 4. En este caso, se puede decir que la arquitectura cognitiva aprende bastante rápido a cumplir el objetivo, con aproximadamente 10 intentos ya la arquitectura cognitiva aprende lo que debe hacer para cumplir el propósito y recibir recompensa. Esto está relacionado a la naturaleza del propósito utilizado, ya que solamente se necesitan 2 o 3 acciones para cumplirlo.

Por otro lado, en los resultados de la Tabla 5.4 se puede notar que el sistema tiene un alto rendimiento, pues en general tiene una tasa de éxito mayor o igual al 99%. En el peor de los casos, falló 6 veces de 603 intentos. La razón por la que no pudo lograr el propósito 6 veces puede deberse a varios motivos. El primero, es que algunas veces el robot mueve los objetos al ejecutar las políticas. Esto hace que los objetos salgan del campo visual de la cámara y fuera del alcance del robot. Por ende, aunque el robot presione el botón para indicar que

el cilindro está fuera de su alcance, el cilindro no puede ser movido. No obstante, como se puede apreciar en 5.4, estos casos son mínimos. Por lo tanto se puede concluir que el sistema funciona correctamente.

Otro factor a mencionar es que el hecho de que el robot logre cumplir el propósito tantas veces no solo indica que la creación de las misiones y los impulsos fue satisfactoria, sino que también la distribución de los pesos de las misiones fue correcta. Esto es un factor muy importante, ya que si esta distribución no es adecuada, el robot se queda en un ciclo ejecutando las primeras acciones que le dan recompensa y nunca ejecuta la última, que es la que cumple el propósito.

Por ultimo, se puede observar en el área celeste de la Figura 5.8, que en general, las 10 repeticiones que se ejecutaron tienen un comportamiento similar y al final convergen al caso ideal.

5.3.4 Resultados del sistema motivacional obtenidos en los experimentos

A continuación, se muestran los resultados relacionados con el sistema motivacional. En este caso, solamente se va a representar los resultados de la repetición 9, pues aunque los experimentos, las misiones y las funciones de impulso sean diferentes, el comportamiento que sigue el sistema es el mismo.

En la Tabla 5.5 se presentan las misiones definidas junto con sus pesos asignados. Las ecuaciones 5.5 y 5.6 describen las funciones de impulso asociadas, las cuales cuantifican el grado de alineamiento del robot con el propósito establecido en cada misión.

Tabla 5.5: Misiones definidas, los pesos asignados y las funciones de impulso

Misión	Peso	Función de impulso
<i>“hand_at_red_cylinder_and_grasped”</i>	0.45	Ecuación 5.5
<i>“red_cylinder_at_blue_basket_and_released”</i>	0.65	Ecuación 5.6

A continuación, se muestran las funciones de impulso correspondientes:

$$D_1 = (\text{robot_hand.x_position} - \text{object1.x_position})^2 + (\text{robot_hand.y_position} - \text{object1.y_position})^2 + (1 - \text{robot_hand.state})^2 \quad (5.5)$$

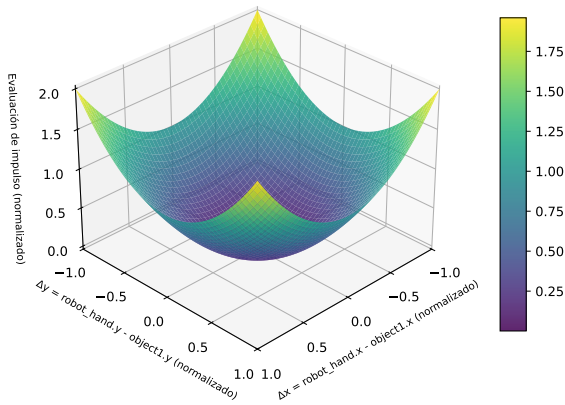
$$D_2 = (\text{object1.x_position} - \text{object2.x_position})^2 + (\text{object1.y_position} - \text{object2.y_position})^2 + (\text{robot_hand.state})^2 \quad (5.6)$$

Para validar que estas funciones de impulso son correctas, se realizó una gráfica en tres dimensiones, donde se pueda verificar que la función es continua, positiva y que es cero solamente cuando se satisface la misión. Como se puede notar, las funciones dependen de 5 variables cada una. Entonces, para poder graficar su comportamiento, se tuvo que simplificar las funciones. En estos casos se tomó la resta de las variables en X y Y como 2 variables únicas y se definió un valor estratégico para el estado de la mano del robot. En las ecuaciones se muestran las versiones simplificadas y en la Figura 5.9 se muestran las gráficas correspondientes.

$$D_1 = (\Delta x)^2 + (\Delta y)^2 + (1 - \text{robot_hand.state})^2 \text{ con } \text{robot_hand.state} = 1 \quad (5.7)$$

$$D_2 = (\Delta x)^2 + (\Delta y)^2 + (\text{robot_hand.state})^2 \text{ con } \text{robot_hand.state} = 0 \quad (5.8)$$

Función de impulso: cilindro en mano (hand_state=1)



Función de impulso: cilindro en canasta (hand_state=0)

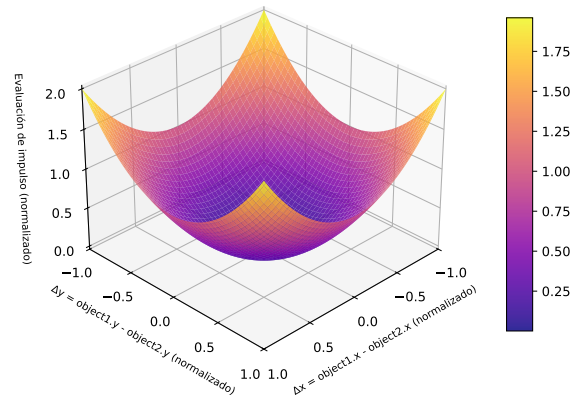
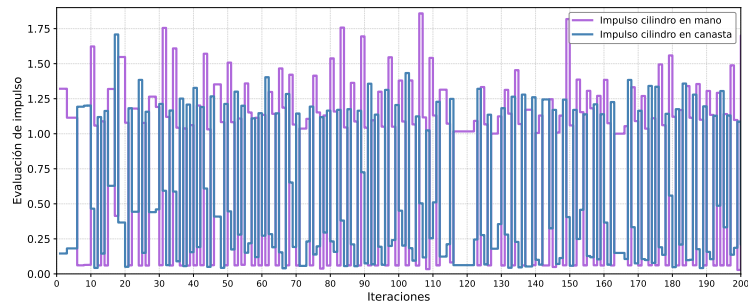
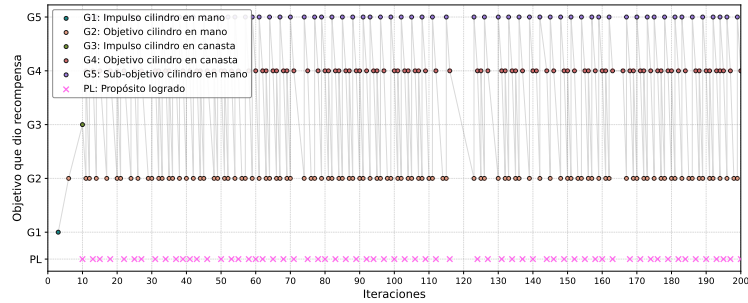


Figura 5.9: Gráfica de funciones de impulso generadas para el repetición 9 ejecutado en Gazebo. Elaboración propia.

Por otro lado, durante la ejecución del experimento en la arquitectura las funciones de impulso se evalúan y actualizan únicamente antes y después de cada iteración o política ejecutada. En este caso, en la Figura 5.10, se muestra la evaluación de los impulsos para las primeras 200 iteraciones de la repetición 9. Así como los objetivos que se fueron creando y dando recompensa a lo largo de estas iteraciones. Así mismo, para poder apreciar de mejor manera el comportamiento del sistema se generaron dos gráficas adicionales. La primera va de la iteración 0 a la 25, donde se muestra el proceso de aprendizaje del robot (Figuras 5.11a y 5.11c) y la segunda va de la iteración 100 a la 125, donde ya el robot ha aprendido a satisfacer el propósito humano (Figuras 5.11b y 5.11d).

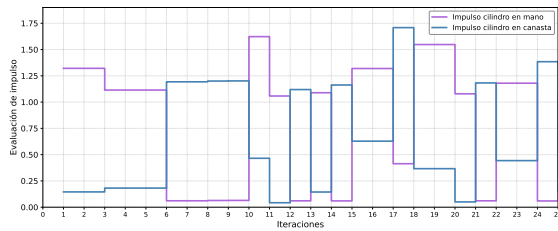


(a) Evaluación de impulso en las iteraciones

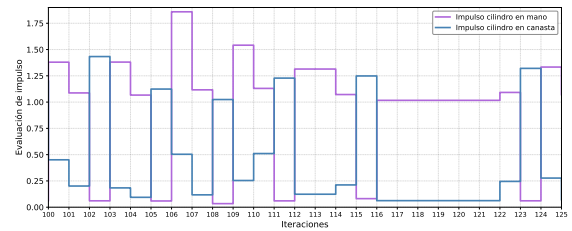


(b) Objetivos que dan recompensa en las iteraciones

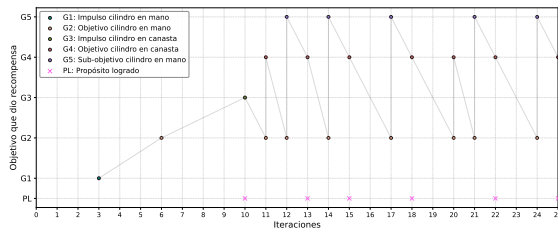
Figura 5.10: Evaluación de las funciones de impulso (5.10a), objetivos que dan recompensa y la satisfacción del propósito logrado (5.10b) a lo largo de las iteraciones del repetición 9. Elaboración propia.



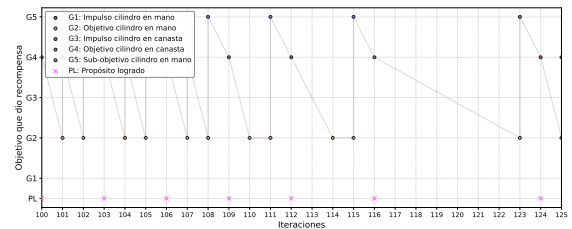
(a) Evaluación de impulso en las iteraciones (Iteraciones 1–25)



(b) Evaluación de impulso en las iteraciones (Iteraciones 100–125)



(c) Objetivos que dan recompensa en las iteraciones (Iteraciones 1–25)



(d) Objetivos que dan recompensa en las iteraciones (Iteraciones 100–125)

Figura 5.11: Comparación entre la evaluación de funciones de impulso (arriba) y los objetivos que dieron recompensa junto con la satisfacción del propósito (abajo), en dos rangos de iteraciones de la repetición 9.

5.3.5 Análisis de los resultados del sistema motivacional en los experimentos finales

Primeramente, se puede observar que las misiones propuestas cumplen con los criterios definidos en la sección 5.1.2 de los resultados. Dividen el propósito en misiones clave que representan un estado final significativo y permiten alcanzar el objetivo final. Además, no son acciones demasiado simples y tienen una distribución de pesos.

Por otro lado, las funciones de impulso propuestas (ecuaciones 5.5 y 5.6), en este caso se validaron gráficamente (ver Figura 5.9). Como se puede apreciar en esta figura, ambos tienen un comportamiento decreciente. En el caso de la función del impulso “cilindro en mano”, la función es únicamente 0 cuando la distancia en (x, y) entre la mano del robot y el cilindro rojo es 0 y el estado de la pinza de la mano es 1. Esto implica que la mano está en la misma ubicación que el cilindro y tiene la pinza cerrada.

En cuanto al impulso “cilindro en canasta”, se puede notar que tiene el mismo comportamiento que el impulso “cilindro en mano”, pero en lugar de la distancia entre la mano y el cilindro, sería la distancia (x, y) entre el cilindro y la canasta. Entonces, la misión se satisfará cuando la distancia entre el cilindro y la canasta es 0 y además el estado de la pinza del robot es 0 (no tiene el objeto en la mano).

Ahora, cuando ya se ejecutó el experimento en conjunto se puede notar en la Figura 5.10 cómo estos dos impulsos van cambiando sus valores a lo largo de las iteraciones. Además, cuando se reduce el valor de alguno de los dos, o de ambos, se activa un objetivo específico que le da recompensa al robot.

En el inicio del experimento (Figuras 5.11a y 5.11c), se puede notar que las gráficas de los impulsos comienzan luego de la iteración 1. Esto es porque en la primera iteración todavía no existen las misiones ni los impulsos, en esta iteración es cuando se ejecuta la política de alineación, que es la que ejecuta el sistema motivacional. Una vez termina la política, se crean los nodos de ROS2 que corresponden a las misiones e impulsos de acuerdo con el propósito humano ingresado.

Luego, se puede notar que en la iteración 3 ocurre una reducción del impulso “cilindro en mano”. Como se puede notar en la gráfica, el valor sigue siendo mayor a 1. Esto significa que el robot ejecutó la política de *presionar botón*, la cual mueve el cilindro a una posición aleatoria que está dentro del alcance del robot. Este efecto crea el objetivo G2: “objeto en mano”. En las iteraciones 4 y 5, se puede notar que no hubo cambios en los impulsos, lo cual indica que el robot ejecutó las políticas de *presionar botón* o *colocar objeto*. Luego, en la iteración 6, se ejecutó la política *tomar objeto*, ya que el impulso de “objeto en mano” se redujo, mientras que el impulso “objeto en canasta” incrementó. En este caso, se activa el objetivo G2, que le da recompensa al robot. Ahora, se puede notar que hasta la iteración 9 se ejecutó la política *colocar objeto*, esto genera una reducción en el impulso “cilindro en canasta” y este efecto crea el objetivo G4: “Cilindro en canasta” y se cumple el propósito por primera vez. Después, se nota que instantáneamente el impulso vuelve a incrementar, esto se debe a que en el momento en que se cumple el propósito se reinicia el mundo, por lo

que se mueven los objetos.

Otra observación que se puede realizar de esta gráfica es que en la iteración 12, el robot ejecutó la política *tomar objeto*. Esto creó a su vez un sub-objetivo (G5) llamado “cilindro en mano”. Esto lo que indica es que el objetivo G2 “cilindro en mano” es un paso previo para poder lograr el propósito. Entonces, el robot aprende a que una vez que tiene el cilindro en la mano, si ejecuta la política *colocar objeto*, consigue el propósito.

Conforme avanza el experimento, se va reduciendo poco a poco el número de iteraciones que ejecuta el robot para lograr el propósito a 3 o 4. Esto también concuerda con los resultados de la sección anterior y con lo que se observa en las Figuras 5.11a y 5.11c. Acá se puede notar que de la iteración 100 a la 116, cumple el propósito 5 veces y en todos los casos le toma 3 o 4 iteraciones. Esto indica que ya el robot aprendió la secuencia que debe realizar para cumplir el propósito y recibir recompensa.

En conclusión, los resultados obtenidos para la repetición 9 concuerdan con el comportamiento observado en la sección anterior. Además, se evidencia que la política de alineación fue capaz de crear las misiones y los impulsos de acuerdo con el propósito que tenía el humano. Finalmente, con estos resultados y con los de la prueba anterior, se puede notar que en general, el sistema motivacional es capaz de crear misiones e impulsos que llevan al robot a realizar efectos en el mundo que cumplen con el propósito humano, por lo que en este punto, se podría decir que se validó correctamente el objetivo específico 4.

5.4 Análisis económico

El alcance de este proyecto tenía un enfoque más centrado en la investigación y no en generar un producto comercial. Igualmente, resulta valioso realizar un análisis económico, que permita estimar y analizar todos los recursos invertidos en el proyecto, así como destacar los beneficios que trajo a los interesados del proyecto.

Cabe resaltar que este proyecto al tratarse de una prueba de concepto, su implementación fue simulada. Esto implica que la mayoría de los recursos utilizados fueron computacionales. Así mismo, hubo una componente muy importante que fue el uso de herramientas como los LLMs, y el tiempo invertido en diseñar la solución propuesta. A continuación, se van a detallar específicamente todos estos recursos y cuánto es su valor en dinero.

Recursos computacionales

En este caso, se utilizó un ordenador personal (HP Victus 15-fa1000 con NVIDIA GeForce RTX 4050) para las etapas iniciales del proyecto y para el diseño del sistema motivacional en Python. Luego, para incorporar el sistema motivacional con la arquitectura cognitiva e-MDB y el robot OSCAR en Gazebo, se utilizó una computadora de escritorio con CPU Zone Evil, con una GPU NVIDIA RTX 4060. Ambas computadoras no se utilizan únicamente para este proyecto, por lo que se va a obtener un costo aproximado de cuánto representa el

uso de estos equipos por mes de acuerdo con su vida útil.

Para estimar el costo relacionado al proyecto, se consideró que las computadoras tienen una vida útil de 5 años (60 meses), y se calculó el equivalente mensual proporcional al tiempo de uso en el proyecto. Además, cabe rescatar que el costo de la computadora Zone Evil se obtuvo del investigador del GII Dr. Jose Antonio Becerra, este costo incluye la CPU, el monitor, teclado y mouse.

Datos de los equipos

Tabla 5.6: Resumen de equipos utilizados en el proyecto

Equipo	Costo inicial (€)	Vida útil (meses)	Uso en proyecto (meses)
HP VICTUS 15-fa1000	1500.00	60	4
Zone Evil (RTX 4060)	1733.93	60	2.5

Cálculo del costo mensual y proporcional al proyecto

HP VICTUS 15-fa1000

El costo mensual del equipo se calcula como:

$$\text{Costo mensual} = \frac{1500}{60} = 25 \text{ €/mes} \quad (5.9)$$

El costo proporcional al uso en el proyecto (4 meses) es:

$$\text{Costo en el proyecto} = 25 \cdot 4 = 100 \text{ €} \quad (5.10)$$

Este cálculo se repitió para la computadora de escritorio Zone Evil, y al final se obtuvieron los costos que se muestran en la Tabla 5.7.

Tabla 5.7: Costos proporcionales de los equipos en el proyecto

Equipo	Costo mensual (€)	Costo en proyecto (€)
HP VICTUS 15-fa1000	25.00	100.00
Escritorio RTX 4060 (Zone Evil)	28.90	72.25
Total	–	172.25

Uso del servidor para modelos de LLM locales

Además de las computadoras, también se utilizó un servidor que pertenece al GII, donde se tienen instalados modelos de LLM de código abierto como el *phi4:14b* y el *Qwen3:30b*.

Este servidor fue adquirido en 2016 y no hay un registro oficial de su costo. Entonces, se obtuvo un costo aproximado de acuerdo con sus especificaciones. En la Tabla 5.8, se resumen sus características y el costo.

Tabla 5.8: Costo aproximado del servidor del GII con precios actuales.

Componente	Cantidad	Costo unitario (€)	Costo total (€)
Servidor ASUSTeK COMPUTER INC. ESC4000 G3 Series	1	2233.80 [37]	2233.80
Módulos de RAM (64GB)	8	299.48 [38]	2395.84
GPU NVIDIA RTX8000	1	6434.78	6434.78
Total			11064.42

Cabe destacar que el precio de la GPU del servidor se tenía registrado, este valor fue consultado con el Dr. Jose Antonio Becerra, investigador del GII. Además, en este caso la vida útil del servidor se define como 10 años, esto equivale a 120 meses. También, el servidor en este momento se está utilizando para tres proyectos diferentes, por lo que se divide el costo entre 3. Entonces, el costo por mes sería el siguiente:

Cálculo del costo mensual y proporcional al proyecto

El costo mensual del servidor se calcula como:

$$\text{Costo mensual} = \frac{11064.42}{120 \cdot 3} = 30.73 \text{ €/mes} \quad (5.11)$$

El costo proporcional al uso en el proyecto (4 meses) es:

$$\text{Costo en el proyecto} = 30.73 \cdot 4 = 122.92 \text{ €} \quad (5.12)$$

Costo por el uso de los LLMs

Para este proyecto se utilizaron tres modelos distintos: *phi4:14B* de Ollama, *Qwen3:30b* de Alibaba Cloud y *GPT-4.1* de OpenAI. En este caso, los modelos *phi4:14B* y *Qwen3:30b* son de código abierto, por lo que no tiene ningún costo descargarlo ni utilizarlo. Mientras que el modelo *GPT-4.1* es el único modelo de pago. Como se comentó en el capítulo 4 de este documento, para el proyecto se utilizó un servidor de OpenRouter.

De manera general, el costo de cada solicitud en OpenRouter, para el modelo *GPT-4.1* tiene un costo de \$2 por cada millón de símbolos (*tokens*) de entrada. Mientras que las salidas tienen un costo de \$8 por cada millón de símbolos (*tokens*) de salida.

En este caso, para este proyecto el gasto total, incluyendo la etapa de diseño y pruebas, sería

de 2,5 \$, esto equivale aproximadamente a 2.14 €. Además, este gasto fue cubierto por el GII por medio de su licencia de OpenRouter.

Horas de trabajo

Finalmente, el principal recurso en este caso fueron las horas de trabajo e investigación para lograr diseñar el proyecto; en concreto, se trabajaron alrededor de 4 meses. De acuerdo con [39], el salario mínimo de un ingeniero recién graduado en España es de aproximadamente 1200 € mensuales. Cabe resaltar que en este caso el trabajo fue voluntario, es decir, este monto corrió por parte del estudiante.

Recursos invertidos totales

Entonces, considerando todos los recursos mencionados previamente, se generó la Tabla 5.9, donde se listan todos los recursos invertidos en el proyecto y cuánto representan en términos de dinero, así como el costo económico final del proyecto.

Tabla 5.9: Recursos asociados al desarrollo del proyecto

Recurso	Descripción	Costo (€)	Cantidad	Subtotal (€)
Equipo compu- tacional	Equipo computacional para crear el código y ejecutar experimentos para el proyecto	172.25	1	172.25
Servidor	Uso del servidor del GII para el proyecto	122.92	1	122.92
Modelos LLM	Acceso a modelos LLM en OpenRouter (para experimentación y ejecución de pruebas)	2.14	1	2.14
Ingeniera	Tiempo investigadora (por mes)	1200.00	4	4800.00
Total (€)				5097.31

Como se puede notar, el costo total del proyecto, es relativamente bajo. El mayor porcentaje de los recursos proviene, como ya se mencionó, de la mano de obra del proyecto, toda la parte del diseño, ejecución de las pruebas finales y demás. Por otro lado, se tomó ventaja de que el GII ya contaba con el servidor para ejecutar los modelos de LLM en local, de manera que no se tiene que pagar para ejecutarlos en un servidor externo.

5.4.1 Beneficios para el GII y el proyecto PILLAR-Robots

El principal beneficio que trajo este proyecto al GII está relacionado con su aporte académico en la investigación. Principalmente, porque este proyecto constituyó una primera implemen-

tación práctica para el descubrimiento y aprendizaje de misiones e impulso, un área que hasta ahora no había sido explorada.

Además, con este proyecto no solo se generó un aporte a nivel del GII, sino que también genera aportes en el proyecto europeo PILLAR-Robots, ya que el alcance de este proyecto forma parte de los objetivos principales que tiene este proyecto. Este es un proyecto financiado por la Unión Europea en el que participan las siguientes entidades:

- Universidade da Coruña (España)
- Consiglio Nazionale delle Ricerche – ISTC-CNR (Italia)
- Sorbonne Université (Francia)
- Centre National de la Recherche Scientifique – CNRS (Francia)
- Athina Research and Innovation Center (Grecia)
- AI2Life SRL (Italia)
- PAL Robotics SL (España)

Esto significa que con esta primera aproximación se tiene una idea más clara de cómo se podría resolver este problema, cuáles mejoras se podrían hacer en el futuro cercano para obtener mejores resultados y completar con los objetivos del proyecto. Además, el cumplir con los objetivos propuestos del proyecto abre la puertas a que el GII pueda desarrollar proyectos similares a PILLAR-Robots y se siga promoviendo la investigación en esta área.

Además, este proyecto va a servir de base para la tesis doctoral del MSc. Jaime Pérez Moro, pues su tesis doctoral se basa en los problemas de alineación en la robótica autónoma.

Finalmente, se espera generar una publicación científica con los resultados obtenidos en este proyecto. Concretamente, se envió una propuesta para el congreso XoveTIC [40] y se espera enviar en los próximos meses a otra conferencia a nivel internacional.

Capítulo 6

Conclusiones

Al terminar este proyecto, en general se puede afirmar que se logró aplicar la metodología de Ulrich Eppinger [3] para desarrollar un sistema motivacional que cumple con los objetivos y las necesidades del GII. Además, se puede concluir que el sistema desarrollado permite alinear y transformar propósitos expresados en lenguaje natural en misiones concretas para ser implementadas en una arquitectura cognitiva y en un robot simulado.

Específicamente, las conclusiones de este proyecto son las siguientes:

- Se identificaron las necesidades del GII y a partir de estas se obtuvieron las especificaciones y métricas que sirvieron de base para generar el diseño propuesto.
- Se realizó un estudio teórico sobre las arquitecturas cognitivas y los sistemas motivacionales en arquitecturas cognitivas, que permitió comprender de mejor manera el problema por resolver.
- Se diseñó un sistema motivacional a partir de un modelo grande de lenguaje (LLM), para resolver el problema de alineación entre el propósito humano y el propósito del robot. El sistema fue capaz de interpretar y alinear diferentes propósitos generales, al final del proceso de alineación en promedio se obtuvo una similitud semántica de 0.8848 comparado con el propósito ideal.
- Se logró generar una solución capaz de traducir un propósito humano detallado y alineado en misiones válidas para la arquitectura con un porcentaje de éxito del 89.4%.
- Se diseñó una solución que tiene la capacidad de generar funciones de impulso a partir de un propósito completamente detallado y una distribución de misiones con una tasa de éxito promedio de 84.1%.
- Se incorporó el sistema motivacional en la arquitectura cognitiva e-MDB y el simulador de OSCAR en Gazebo, lo que permitió crear nuevas misiones y funciones de impulso, evaluarlas y también que el robot fuera capaz de ejecutar políticas.
- Se validó el funcionamiento de todo el sistema con una simulación robótica en Gazebo con el robot OSCAR. En esta simulación se logró observar que el sistema motivacional

diseñado es capaz de crear los nodos cognitivos de misiones e impulsos correspondientes al propósito ingresado por el usuario. Además, gracias a estas misiones e impulsos creados por el sistema motivacional el robot fue capaz de cumplir el propósito humano más de un 99% de las veces en todas las pruebas ejecutadas.

- Se comprobó que el uso de misiones e impulsos como guía permite adaptar el comportamiento del robot ante situaciones dinámicas para cumplir con el propósito asignado por el humano.
- El sistema desarrollado logró demostrar el concepto la autonomía intencionada y se puede afirmar que es un punto inicial clave para resolver el problema alineación de autonomía del humano y el robot.

6.1 Recomendaciones

Con base en los resultados obtenidos y el proceso de diseño seguido en este proyecto, se proponen las siguientes recomendaciones para fortalecer e impulsar futuras investigaciones o aplicaciones prácticas basadas en este trabajo:

- Ejecutar propósitos más complejos en el simulador Gazebo para verificar si el robot es capaz de aprender a cumplir cada propósito.
- Implementar el mismo experimento propuesto en este proyecto en un robot real para verificar que el comportamiento es el mismo y que coincide con los resultados obtenidos en la simulación.
- Incorporar una política de retroalimentación que le permita al usuario indicarle al robot cuando no está cumpliendo el propósito, de manera que el sistema sea capaz de modificar sus propias misiones e impulsos dinámicamente en función de resultados parciales o condiciones externas cambiantes.
- Mejorar la interfaz gráfica de interacción con los modelos LLM para que sea más amigable con el usuario.
- Mejorar el sistema motivacional para que sea más fácil el proceso de interacción y que pueda también ser usado correctamente por usuarios que pueden no ser expertos en el área. Esto en aras de lograr aplicaciones más reales fuera del laboratorio.

Bibliografía

- [1] A. Romero, “e-mdb: A cognitive architecture for lifelong open-ended learning autonomy in robotic systems.” Ph.D. dissertation, Universidade da Coruña, 2022, tesis de Doctorado.
- [2] G. Baldassarre, R. J. Duro, E. Cartoni, M. Khamassi, A. Romero, and V. G. Santucci, “A formalisation of the purpose framework: The autonomy-alignment problem in open-ended learning robots,” *arXiv*, vol. cs.RO, no. arXiv:2403.02514v2, Apr. 2025. [En línea]. Disponible: <https://arxiv.org/abs/2403.02514v2>
- [3] K. T. Ulrich and S. D. Eppinger, *Diseño y desarrollo de productos*, 5th ed. McGraw-Hill, 2012, edición en español.
- [4] Ollama. phi4:14b. (2025). [En línea]. Disponible: <https://ollama.com/library/phi4:14b>
- [5] Qwen. Qwen 3 — next-generation language models. (2025). [En línea]. Disponible: <https://qwenlm.github.io/blog/qwen3/>
- [6] OpenAI. Gpt-4.1: Openai’s new flagship model. (2025). [En línea]. Disponible: <https://openai.com/index/gpt-4-1/>
- [7] Grupo Integrado de Ingeniería. Grupo integrado de ingeniería (gii). (2025). [En línea]. Disponible: <https://gii.udc.es>
- [8] PILLAR Project. Project overview. (2022). [En línea]. Disponible: <https://pillar-robots.eu/project-overview/>
- [9] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, “Concrete problems in ai safety,” *arXiv preprint arXiv:1606.06565*, 2016. [En línea]. Disponible: <https://arxiv.org/abs/1606.06565>
- [10] A. Romero, F. Bellas, and R. J. Duro, “A perspective on lifelong open-ended learning autonomy for robotics through cognitive architectures,” *Sensors*, vol. 23, DOI 10.3390/s23031611, no. 3, p. 1611, 2023. [En línea]. Disponible: <https://www.mdpi.com/1424-8220/23/3/1611>
- [11] I. Kotseruba and J. K. Tsotsos, “40 years of cognitive architectures: core cognitive abilities and practical applications,” *Artificial Intelligence Review*, vol. 53, DOI 10.1007/s10462-018-9646-y, no. 1, pp. 17–94, 2020.

- [12] P. Langley, J. E. Laird, and S. Rogers, “Cognitive architectures: Research issues and challenges,” *Cognitive Systems Research*, vol. 10, DOI 10.1016/j.cogsys.2006.07.004, no. 2, pp. 141–160, 2009.
- [13] Q. Houbre and R. Pieters, “Dynamic neural curiosity enhances learning flexibility for autonomous goal discovery,” *arXiv preprint arXiv:2412.00152*, 2024.
- [14] S. Forestier, V. G. Santucci, A. Baranes, and P.-Y. Oudeyer, “Intrinsically motivated goal exploration processes with automatic curriculum learning,” *Journal of Machine Learning Research*, vol. 23, no. 139, pp. 1–45, 2022.
- [15] V. G. Santucci, G. Baldassarre, and M. Mirolli, “Grail: a goal-discovering robotic architecture for intrinsically-motivated learning,” *IEEE Transactions on Cognitive and Developmental Systems*, vol. 8, no. 3, pp. 214–231, 2016.
- [16] A. Romero, G. Baldassarre, R. J. Duro, and V. G. Santucci, “A motivational architecture for open-ended learning challenges in robots,” *arXiv preprint arXiv:2506.18454*, 2025.
- [17] R. Novianto, B. Johnston, and M.-A. Williams, “Attention in the asmo cognitive architecture,” in *2010 10th International Conference on Intelligent Systems Design and Applications*, pp. 1343–1348. IEEE, 2010.
- [18] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, “A comprehensive overview of large language models,” *arXiv preprint arXiv:2307.06435*, 2024, preprint submitted to Elsevier, October 18, 2024. [En línea]. Disponible: <https://arxiv.org/abs/2307.06435>
- [19] C. Arora, A. I. Sayeed, S. A. Licorish, F. Wang, and C. Treude, “Optimizing large language model hyperparameters for code generation,” *CoRR*, vol. abs/2408.10577, DOI 10.48550/ARXIV.2408.10577, 2024. [En línea]. Disponible: <https://doi.org/10.48550/arXiv.2408.10577>
- [20] K. Chang, S. Xu, C. Wang, Y. Luo, X. Liu, T. Xiao, and J. Zhu, “Efficient prompting methods for large language models: A survey,” *CoRR*, vol. abs/2404.01077, 2024. [En línea]. Disponible: <https://arxiv.org/abs/2404.01077>
- [21] J. Wang and Y. Dong, “Measurement of text similarity: A survey,” *Information*, vol. 11, DOI 10.3390/info11090421, no. 9, p. 421, 2020. [En línea]. Disponible: <https://www.mdpi.com/2078-2489/11/9/421>
- [22] C. Galli, N. Donos, and E. Calciolari, “Performance of 4 pre-trained sentence transformer models in the semantic query of a systematic review dataset on peri-implantitis,” *Information*, vol. 15, DOI 10.3390/info15020068, no. 2, p. 68, 2024. [En línea]. Disponible: <https://www.mdpi.com/2078-2489/15/2/68>
- [23] Cyberbotics. Cyberbotics - webots robot simulator. (2025). [En línea]. Disponible: <https://cyberbotics.com/#cyberbotics>

- [24] Coppelia Robotics. Coppeliasim - robot simulator. (2025). [En línea]. Disponible: <https://www.coppeliarobotics.com/>
- [25] Open Source Robotics Foundation. Gazebo simulation - features. (2025). [En línea]. Disponible: <https://gazebo.org/features>
- [26] DeepMind Technologies. Mujoco documentation - computation. (2025). [En línea]. Disponible: <https://mujoco.readthedocs.io/en/stable/computation/index.html>
- [27] Mintforpeople. Robobo programming - unity. (2025). [En línea]. Disponible: <https://github.com/mintforpeople/robobo-programming/wiki/Unity>
- [28] Bases de Datos y Ciencias Sociales. Modelos de simulación: agentes y autómatas celulares. (2025). [En línea]. Disponible: <https://bdcv.hypotheses.org/5507>
- [29] R. Collobert and J. Weston, “Fast semantic extraction using a novel neural network architecture,” *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*, pp. 560–567, Jun. 2007. [En línea]. Disponible: <https://aclanthology.org/P07-1071>
- [30] I. Prada. Chatbots con inteligencia artificial vs chatbots guiados. (2025). [En línea]. Disponible: <https://www.potenzzia.com/blog/chatbots-con-inteligencia-artificial-vs-chatbots-guiados>
- [31] S. Hundertmark. Llm chatbots – an introduction to the new world of bots. (2024). [En línea]. Disponible: <https://sophiehundertmark.medium.com/llm-chatbots-an-introduction-to-the-new-world-of-bots-485db17da7b2>
- [32] Scikit-learn. Classification of text documents using sparse features. (2025). [En línea]. Disponible: https://scikit-learn.org/stable/auto_examples/text/plot_document_classification_20newsgroups.html
- [33] Pillar-Robots Project, “e-MDB Experiments - GII Documentation,” https://docs.pillar-robots.eu/projects/emdb_experiments_gii/en/latest/, 2024, accessed: 2025-07-24.
- [34] PILLAR Project. Oscar experiment — e-mdb experiments documentation. (2025). [En línea]. Disponible: https://docs.pillar-robots.eu/projects/emdb_experiments_gii/en/latest/experiments/oscar_experiment.html
- [35] E. Fallas, “Sistema de transformación de propósitos en misiones y funciones de drive comprensibles por la arquitectura cognitiva e-mdb,” Ph.D. dissertation, Tecnológico de Costa Rica, 2021, trabajo Final de Graduación, Ingeniería en Computadores.
- [36] “Sentence-bert: Sentence embeddings using siamese bert-networks,” <https://www.sbert.net/>, 2023, accessed: 2025-07-21.
- [37] Última Informática, “Asus esc4000 g3 intel c612 lga 2011 v3 bastidor 2u,” <https://ultimainformatica.com/asus-esc4000-g3-intel-c612-lga-2011-v3-bastidor-2u.html>, 2025, accedido el 22 de julio de 2025.

-
- [38] Offtek, “Módulos de memoria para asus esc4000 g3,” <https://www.offtek.es/asus-esc4000-g3>, 2025, accedido el 22 de julio de 2025.
- [39] SIPEM UPCT, “¿cuánto cobra un ingeniero recién graduado en españa?” <https://sipem.upct.es/noticia/cuanto-cobra-un-ingeniero-recien-graduado-en-espana>, 2025, accedido el 22 de julio de 2025.
- [40] CITIC - Universidade da Coruña, “Xovetic: Plataforma galega de tecnoloxías intelixentes para a e-saúde,” 2025, accedido el 23 de julio de 2025. [En línea]. Disponible: <https://xovetic.citic.udc.es/>

Apéndice A

Posibles combinaciones de conceptos

En la Tabla A.1 se muestran todas las posibles combinaciones de conceptos que surgieron al aplicar la metodología Ulrich y Eppinger.

Tabla A.1: Combinaciones posibles de módulos del sistema sin considerar el simulador

Concepto	Alinear el propósito del humano	Generar misiones	Generar funciones de impulso
A	Modelos de LLM	Modelos de clasificación	Lista estándar + modelo sensores
B	Modelos de LLM	Modelos de clasificación	Modelo de LLM
C	Modelos de LLM	Modelo de LLM	Lista estándar + modelo sensores
D	Modelos de LLM	Modelo de LLM	Modelo de LLM
E	Chat guiado	Modelos de clasificación	Lista estándar + modelo sensores
F	Chat guiado	Modelos de clasificación	Modelo de LLM
G	Chat guiado	Modelo de LLM	Lista estándar + modelo sensores
H	Chat guiado	Modelo de LLM	Modelo de LLM
I	Extracción de verbos	Modelos de clasificación	Lista estándar + modelo sensores
J	Extracción de verbos	Modelos de clasificación	Modelo de LLM
K	Extracción de verbos	Modelo de LLM	Lista estándar + modelo sensores
L	Extracción de verbos	Modelo de LLM	Modelo de LLM

Apéndice B

Solicitudes iniciales para la selección de conceptos

En esta sección se muestran los ejemplos de solicitudes iniciales utilizados en la prueba de conceptos para el sistema motivacional del capítulo 4. La Fig. B.2 muestra la solicitud inicial de prueba generada para la prueba con un único modelo (*phi4:14B*). Mientras que Fig. B.3 muestra la solicitud inicial de entrenamiento para la parte de generación la prueba 2, para la parte de misiones y funciones de impulso.

```

model: phi4:14b

initial_prompt:
  role: "system"
  content: |
    content: |
    From now on you are an assistant that helps an autonomous robot and a human to interact.
    The human will give you the purpose they want for the robot, and you will translate it into mission and drives.

<Context>
First, you need to understand some concepts:
Needs and missions are internal, domain-independent goal states that the robot seeks to achieve or maintain. The difference
between needs and missions is that needs are intrinsic goals or states the robot requires for exploration and learning, while
missions are goal states that the robot seeks to achieve or maintain to fulfill the human's purpose.
A drive is a mathematical function that represents the urgency to satisfy a mission and when the mission is accomplished.
A drive value is the result of evaluating a drive function under a specific condition.
Goals are points in the state space which, when reached, reduce the drive value.
Each mission must have a drive related to it, so there will always be the same number of needs and drives.
</Context>

<Setup>
A robotic arm on a fixed base (center-bottom of a table).
Sensors provide normalized values in range [0, 1].
There are some objects on the table. All the objects have the same perception structure and are numbered as object1, object2,
etc.
You will receive a list of the perceptions of the objects in the domain.
</Setup>

<Perceptions>
Here there is a description of the robot's perceptions:
- robot_hand.x_position: Returns the x-coordinate of the robot's hand, using the bottom-left corner of the table as the
origin (0, 0). The value is normalized between 0 and 1, where 0 is the left edge and 1 is the right edge of the table.
- robot_hand.y_position: Returns the y-coordinate of the robot's hand, using the bottom-left corner of the table as the
origin (0, 0). The value is normalized between 0 and 1, where 0 is the bottom edge and 1 is the top edge of the table.
- robot_hand.state: Returns 1 when there is an object grasped and 0 when there is not an object grasped.

- object.label: Returns a label that describes what the object is.
- object.x_position: Returns the x-coordinate of the object, using the bottom-left corner of the table as the origin (0, 0).
The value is normalized between 0 and 1, where 0 is the left edge and 1 is the right edge of the table.
- object.y_position: Returns the y-coordinate of the object, using the bottom-left corner of the table as the origin (0, 0).
The value is normalized between 0 and 1, where 0 is the bottom edge and 1 is the top edge of the table.
- object.diameter: Returns the normalized diameter of the object as a float between 0 and 1. This value represents the
object's real-world diameter scaled relative to the maximum possible size, so it may vary depending on the actual physical
size of the object.
- object.color: Returns the color of the object.
- object.state: Returns the object's state.
</Perceptions>

<Task>
Now, based on that, you will translate the human purpose into a robot purpose, which will consist of missions and drive
functions.
It is important that you make sure you have good and enough information to create the missions and drive functions, so that
the robot can work properly. Hence, you have to generate a hypothesis of what the human wants and ask questions to make sure
that the purpose is complete. Try to be as specific as you can with the questions but remember that the final user could not
know how the code works so try to interact with high-level language. Keep asking questions until the human tells you that the
description is what they want.
When the human confirms the purpose description is complete, you will generate the missions and drive functions following
this format:
</Task>

<Missions format>
mission = [tag, weight]
  tag: This should be a very short and representative string.
  weight: This should be a float value between 0 and 1 indicating how important it is to satisfy the mission, this value is
important when there are different needs/missions so the robot knows what to do first.
</Missions format>

<Drives functions format>
Drives: [sensor, function]
  sensor: This should be a string with the name of the sensor. The name must be the same as the ones given before.
  function: As it was said before, this function must be expressed in terms of the perceptions described. It must be close to
0 when the mission is satisfied, and it has to grow as it becomes more urgent to satisfy the mission. Avoid functions with
singularities and the use of unnecessary parentheses.
</Drives functions format>

<Final answer>
WHEN YOU RETURN THE FINAL VALUE OF THE NEEDS AND DRIVES, you MUST return the values in the following format and only this
way, WITHOUT ANY ADDITIONAL TEXT OR EXPLANATION:
[Mission1, Mission2, ..., Missionn], [Drive1, Drive2, ..., Driven]
</Final answer>

Reply only and always with missions and drives in the given structure and the questions you need to ask. Avoid a lot of text,
additional information, or explanation. In case the human purpose is impossible to achieve then return: "It is impossible to
achieve your purpose with this robot".

```

Figura B.1: Prompt utilizado para la prueba con una única instancia con el modelo *phi4:14b*.
Elaboración propia.

```

model: phi4:14b

initial_prompt:
  role: "system"
  content: |
    From now on you are an assistant that helps an autonomous robot and a human to interact.
    The human will give you the purpose they want for the robot, and you will translate it into missions
    and drives.

    <Context>
    First, you need to understand some concepts:
    Needs and missions are internal, domain-independent goal states that the robot seeks to achieve or
    maintain. The difference between needs and missions is that needs are intrinsic goals or states the
    robot requires for exploration and learning, while missions are goal states that the robot seeks to
    achieve or maintain to fulfill the human's purpose.
    A drive is a mathematical function that represents the urgency to satisfy a mission and when the
    mission is accomplished.
    A drive value is the result of evaluating a drive function under a specific condition.
    Goals are points in the state space which, when reached, reduce the drive value.
    Each mission must have a drive related to it, so there will always be the same number of needs and
    drives.
    </Context>

    <Setup>
    A robotic arm on a fixed base (center-bottom of a table).
    Sensors provide normalized values in the range [0, 1].
    There are some objects on the table. All the objects have the same perception structure and are
    numbered as object1, object2, etc.
    You will receive a list of the perceptions of the objects in the domain.
    </Setup>

    <Perceptions>
    Here there is a description of the robot's perceptions:
    - robot_hand.x_position: Returns the x-coordinate of the robot's hand, using the bottom-left corner
    of the table as the origin (0, 0). The value is normalized between 0 and 1, where 0 is the left edge
    and 1 is the right edge of the table.
    - robot_hand.y_position: Returns the y-coordinate of the robot's hand, using the bottom-left corner
    of the table as the origin (0, 0). The value is normalized between 0 and 1, where 0 is the bottom
    edge and 1 is the top edge of the table.
    - robot_hand.state: Returns 1 when there is an object grasped and 0 when there is not an object
    grasped.

    - object.label: Returns a label that describes what the object is.
    - object.x_position: Returns the x-coordinate of the object, using the bottom-left corner of the table
    as the origin (0, 0). The value is normalized between 0 and 1, where 0 is the left edge and 1 is the
    right edge of the table.
    - object.y_position: Returns the y-coordinate of the object, using the bottom-left corner of the
    table as the origin (0, 0). The value is normalized between 0 and 1, where 0 is the bottom edge and 1
    is the top edge of the table.
    - object.diameter: Returns the normalized diameter of the object as a float between 0 and 1. This
    value represents the object's real-world diameter scaled relative to the maximum possible size, so it
    may vary depending on the actual physical size of the object.
    - object.color: Returns the color of the object.
    - object.state: Returns the object's state.
    </Perceptions>

    <Task>
    You will receive several messages, ordered chronologically, in YAML format with the interaction
    between the LLM and the human.
    When you have a clear idea of what the human wants, you must check with the user if your description
    is correct. In case the human answers "Yes, that's what I want", you create a final version of the
    hypothesis. If the answer is negative you keep making the changes.
    Only when the human agrees with your description, you will reply only in the following way:

    <Final message>
    Final description:

    (Add the content you just created)

    <Final message>
    </Task>

    Reply only and always with things related to the topic. Avoid unnecessary text, additional
    information, or explanation. In case the human purpose is impossible to achieve then return: "It is
    impossible to achieve your purpose with this robot".
    You DO NOT HAVE to give needs, missions or drives, only the final description of what the human wants
    without more questions or text.
  
```

Figura B.2: Prompt utilizado para la parte de alineación para la prueba con dos instancias con el modelo *phi4:14b*. Elaboración propia.

```

model: phi4:14b

initial_prompt:
  role: "system"
  content: |
    You are an assistant that helps to control an autonomous robot.
    You will get a specific description of what the human purpose is for the robot and you will translate it into one, two, three or more missions and drives as needed.

<Context>
Here you have some context:
Needs and missions are internal, domain-independent goal states that the robot seeks to achieve or maintain. The difference between needs and missions is that needs are intrinsic goals or states the robot requires for exploration and learning, while missions are goal states that the robot seeks to achieve or maintain to fulfill the human's purpose.
A drive function is a mathematical function that represents the urgency to satisfy a mission and when the mission is accomplished.
A drive value is the result of evaluating a drive function under a specific condition.
Goals are points in the state space which, when reached, reduce the drive value.
Each mission must have a drive function related to it, so there will always be the same number of needs and drive functions.
</Context>

<Setup>
A robotic arm on a fixed base (center-bottom of a table).
Sensors provide normalized values in the range [0, 1].
There are some objects on the table. All the objects have the same perception structure and are numbered as object1, object2, etc.
You will receive a list of the objects perceptions that are in the domain.
</Setup>

<Perceptions>
Here there is a description of the robot's perceptions:
- robots_hand_x_position: Returns the x-coordinate of the robot's hand, using the bottom-left corner of the table as the origin (0, 0). The value is normalized between 0 and 1, where 0 is the left edge and 1 is the right edge of the table.
- robot_hand_y_position: Returns the y-coordinate of the robot's hand, using the bottom-left corner of the table as the origin (0, 0). The value is normalized between 0 and 1, where 0 is the bottom edge and 1 is the top edge of the table.
- robot_gripper: Returns 1 when there is an object grasped and 0 when there is not an object grasped.

- object.label: Returns a label that describes what the object is.
- object.x_position: Returns the x-coordinate of the object, using the bottom-left corner of the table as the origin (0, 0). The value is normalized between 0 and 1, where 0 is the left edge and 1 is the right edge of the table.
- object.y_position: Returns the y-coordinate of the object, using the bottom-left corner of the table as the origin (0, 0). The value is normalized between 0 and 1, where 0 is the bottom edge and 1 is the top edge of the table.
- object.diameter: Returns the normalized diameter of the object as a float between 0 and 1. This value represents the object's real-world diameter scaled relative to the maximum possible size, so it may vary depending on the actual physical size of the object.
- object.color: Returns the color of the object.
- object.state: Returns the object's state.
</Perceptions>

<Needs>
Right now the robot has internally defined the following needs:

  name: novelty_need
  description: takes the robot to the most novel states in the current world model (states where the robot has not been).
  class_name: cognitive_nodes.need.Need
  parameters:
    weight: 0.1
    drive_id: 'novelty_drive'
    need_type: 'cognitive'

  name: effectance_need
  description: makes the robot generate new effects on the world.
  class_name: cognitive_nodes.need.Need
  parameters:
    weight: 0.25
    drive_id: 'effectance_drive'
    need_type: 'cognitive'

</Needs>

<Task>
You will generate a list of missions and drive functions that allow the robot to achieve the human purpose given.
To do this, you must consider that the needs given are always active, so considering how these needs might affect the tasks the robot should do to satisfy the human purpose, you will generate missions and drive functions.
In order to do that, you need to understand that the missions should be meaningful final states such as "object in location", "object in hand", NOT a verb.

The format for missions and drives is the following.

<Missions>
Missions: [tag, weight]

  - tag: This should be a very short and representative string. It has to be related to the final state intended to be achieved.
  - weight: This should be a float value between 0 and 1. The missions that the robot has to do first should have lower weights, and the missions the robot will do last must have a higher weight value. These values should always be greater than the weights of the internal needs.
</Missions>

<Drives functions>
Drives: [function]

  - function: As it was said before, this function must use one or more perceptions as variables. Hence, it must be expressed in terms of one or more perceptions described before (represented with the string name).
  Also, it has to be close to 0 when the mission is satisfied, and it has to grow as it becomes more urgent to satisfy the mission. Avoid functions with singularities and the use of unnecessary parentheses. The functions always have to be positive.
</Drives functions>

WHEN YOU RETURN THE FINAL VALUE OF THE NEEDS/MISSIONS AND DRIVES, YOU MUST return the values in the following way and ONLY THIS WAY, WITHOUT ANY ADDITIONAL TEXT OR EXPLANATION:

<Final_answer>
[Mission1, Mission2, ..., MissionN]
[Drive1, Drive2, ..., DriveN]
</Final_answer>

Reply only and always with needs/missions and drives in the given structure. Avoid unnecessary text, additional information, or explanation. In case the human purpose is impossible to achieve then return: "It is impossible to achieve your purpose with this robot".

```

Figura B.3: Prompt utilizado para la parte de traducción de misiones y funciones de impulso para la prueba con dos instancias, utilizando el modelo *phi4:14b*. Elaboración propia.

Apéndice C

Solución final

En este capítulo se adjunta la solución final desarrollada en un repositorio de GitHub. Este repositorio contiene el código del sistema motivacional, todas las solicitudes iniciales utilizadas para entrenar los modelos de LLM, la integración con la arquitectura cognitiva. A continuación se adjuntan los enlaces de cada una de las carpetas para un mejor acceso a las diferentes partes.

- Enlace al repositorio de GitHub con el código del sistema motivacional: https://github.com/DianaCerdasV/TFG/tree/main/C%C3%B3digo_Sistema_Motivacional
- Enlace directo al repositorio de la arquitectura cognitiva e-MDB: <https://github.com/pillar-robots> https://github.com/DianaCerdasV/TFG/tree/main/C%C3%B3digo_Sistema_Motivacional/Prompts_finales
- Enlace directo a la carpeta de solicitudes iniciales utilizadas en el diseño final: https://github.com/DianaCerdasV/TFG/tree/main/C%C3%B3digo_Sistema_Motivacional/Prompts_finales

Apéndice D

Propósitos utilizados para la prueba de alineación de propósitos

En este apéndice se detallan los diferentes propósitos con bajo nivel de detalle para las pruebas de validación del sistema de alineación de propósitos. En [D.1](#) se muestran los propósitos para la tarea de "Acomodar las manzanas según su color".

Tabla D.1: Propósitos poco detallados para la tarea de acomodar las manzanas según su color.

N°	Propósito
1	<i>Arrange the apples on the table.</i>
2	<i>Organize all apples by their color.</i>
3	<i>Sort the items on the table based on color.</i>
4	<i>Distribute the apples appropriately.</i>
5	<i>Classify the apples into groups.</i>
6	<i>Place the apples where they belong.</i>
7	<i>Separate the apples according to their color.</i>
8	<i>Ensure the apples are arranged correctly.</i>
9	<i>Organize the items by their characteristics.</i>
10	<i>Help me keep the apples organized by color.</i>
11	<i>Put the objects in their designated spots.</i>
12	<i>Group similar apples together on the table.</i>
13	<i>Make sure the apples are placed in the right areas.</i>
14	<i>Keep the apples on the table so sorted by color.</i>
15	<i>Arrange and place apples based on their type.</i>

Por su parte, en [D.2](#) se muestran los propósitos para la tarea de "Apilar los cilindros en el centro de la mesa por orden de tamaño (más grande en la base y el más pequeño en la punta)".

Tabla D.2: Propósitos poco detallados para la tarea de apilar los cilindros en el centro de la mesa por orden de tamaño.

N°	Propósito
1	<i>Stack the cylinders properly on the table.</i>
2	<i>Arrange the items in a vertical stack at the center.</i>
3	<i>Organize all objects by size in the middle of the table.</i>
4	<i>Place the appropriate items in a single pile.</i>
5	<i>Sort objects into a stack.</i>
6	<i>Sort the objects to create a tower.</i>
7	<i>Arrange the pieces on top of each other.</i>
8	<i>Stack similar objects in order from largest to smallest.</i>
9	<i>Create an organized pile with the cylinders.</i>
10	<i>Place objects in a neat stack at the center.</i>
11	<i>Arrange everything so it forms a clear and tidy stack.</i>
12	<i>Organize the cylinders by size into a single column.</i>
13	<i>Stack the correct objects in the proper order.</i>
14	<i>Place everything in a single, well-ordered pile.</i>
15	<i>The robot has to stack the items so they're properly organized.</i>

Apéndice E

Resultados de pruebas

En este apéndice se encuentran los enlaces directos a las carpetas del repositorio de GitHub con los resultados de las pruebas de validación en crudo y un video donde se muestra la ejecución del experimento.

- Enlace a los resultados prueba de evaluación del sistema motivacional: https://github.com/DianaCerdasV/TFG/tree/main/Resultados_Sistema_Motivacional

Este enlace contiene un documento de Excel con los resultados obtenidos en las pruebas de alineación, generación de misiones y generación de impulsos para cada uno de los propósitos generales analizados.

- Enlace al video demostrativo del experimento en Gazebo: <https://youtu.be/0eg-IVzC7b4>
- Enlace a los resultados de las 10 repeticiones del experimento con OSCAR en Gazebo: https://github.com/DianaCerdasV/TFG/tree/main/Resultados_Experimentos_Gazebo

Apéndice F

Entregables del proyecto

En este capítulo se incluyen todos los entregables del proyecto. En primer lugar se tiene un documento con un resumen del estado del arte sobre los sistemas motivacionales en algunas arquitecturas cognitivas, así como las necesidades del proyecto. Luego se adjunta el repositorio de GitHub donde se encuentra todo el código generado para el proyecto y además se incluye la configuración de los experimentos y del simulador Gazebo.

- Entregable 1: https://github.com/DianaCerdasV/TFG/tree/main/Entregable_1
- Entregables 2 y 3: <https://github.com/DianaCerdasV/TFG.git>