Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Inteligencia artificial

Integración de LLMs en Sistemas de Navegación Autónoma de Robocar

Trabajo fin de estudio presentado por:	Rubén Higuera Castillo
Tipo de trabajo:	Tipo 2: Desarrollo Software
Director/a:	Óscar Sánchez Rueda
Fecha:	Abril 2026

Resumen

Pendiente para entrega 2-3

Abstract

Pendiente para entrega 2-3

Figura 1: Punto de partida de Robocar

Índice de contenidos

Índice de figuras

1. Introducción

Este capítulo introductorio presenta los fundamentos que motivaron la realización de este trabajo, el problema concreto que se aborda y la propuesta de solución planteada. Se estructura en tres apartados: la motivación del trabajo, el planteamiento del problema y la solución propuesta, y la descripción de la organización del documento.

1.1. MOTIVACIÓN

La robótica autónoma ha experimentado un avance sostenido durante las últimas décadas, impulsada por la mejora en la capacidad computacional, el abaratamiento de sensores y el desarrollo de frameworks de software robótico como ROS2 (Robot Operating System 2). Sin embargo, uno de los grandes retos que persiste en esta área es la interfaz entre el ser humano y el robot: para que un sistema robótico ejecute una tarea, el usuario debe conocer los interfaces técnicos del sistema, los comandos específicos o los tópicos ROS2 correspondientes. Este requisito limita enormemente el espectro de usuarios capaces de interactuar con sistemas robóticos y, por extensión, su aplicabilidad práctica.

En paralelo, los Grandes Modelos de Lenguaje (LLMs, del inglés Large Language Models) han emergido como una tecnología transformadora que ha demostrado capacidades extraordinarias de comprensión y generación de lenguaje natural, razonamiento y planificación de tareas. Modelos como GPT-4, Claude o Llama 3 son capaces de interpretar instrucciones ambiguas, resolver referencias contextuales y descomponer objetivos complejos en secuencias de acciones atómicas. La integración de estos modelos con sistemas robóticos, lo que la literatura denomina Embodied AI, representa uno de los paradigmas de investigación más activos del momento. Empresas como Boston Dynamics, Figure AI, Physical Intelligence (Pi) o Tesla están apostando fuertemente por esta dirección, que promete democratizar la interacción humano-robot mediante el lenguaje natural como interfaz universal.

El presente Trabajo Fin de Máster se enmarca en esta confluencia entre robótica autónoma y procesamiento de lenguaje natural. El trabajo tiene su origen en dos Trabajos Fin de Grado desarrollados entre 2023 y 2024 en la Escuela Técnica Superior de Ingeniería de Sistemas Informáticos (ETSISI) de la Universidad Politécnica de Madrid. El primero, “Diseño y Construcción de Robocar” (Rubén Higuera Castillo y Kento Reinoso Hayashi, 2024), sentó las

bases hardware del sistema: diseño mecánico y electrónico de un robot coche autónomo de bajo coste, selección e integración de sensores (HC-SR04, VL53L0X, cámara USB, IMU MPU6050, encoders ópticos), diseño y fabricación de PCBs a medida, y un primer stack de software basado en ROS2 con panel de control. El segundo trabajo, "Diseño e Implementación de Seguimiento de Carril de Robocar" (Rubén Higuera Castillo, 2024), evolucionó el sistema hacia la autonomía: se implementó un algoritmo de seguimiento de carril basado en técnicas clásicas de visión por computadora (detección de bordes con el operador de Canny y la transformada de Hough), junto con un filtro de Kalman para la estimación robusta de la posición del carril y un controlador PID para la dirección. El resultado fue un robot capaz de navegar de forma autónoma por un circuito de carriles marcados sobre el suelo.

Esta navegación reactiva, si bien funcional, presenta limitaciones fundamentales desde la perspectiva de la inteligencia artificial aplicada. El robot es capaz de seguir el carril marcado sobre el suelo de forma autónoma, pero su autonomía se circunscribe al dominio de la percepción visual inmediata: no dispone de representación alguna del entorno, no puede razonar sobre el espacio en el que opera ni planificar trayectorias hacia destinos semánticos concretos, y no puede recibir instrucciones en lenguaje natural. Para interactuar con el sistema, el usuario debe necesariamente conocer los topics de ROS2 o los comandos de teleoperación. Es precisamente este salto (de la navegación reactiva basada en percepción visual a la navegación goal-directed con comprensión semántica del entorno) el que aborda este Trabajo Fin de Máster, en continuidad directa con los trabajos previos sobre la misma plataforma robótica.

La relevancia científico-técnica de este trabajo se fundamenta en tres pilares. En primer lugar, la integración de LLMs con sistemas robóticos es un área de investigación activa que está transformando la interacción humano-robot, con publicaciones destacadas en el ámbito de Vision-Language Navigation (VLN) y robots que ejecutan instrucciones en lenguaje natural (Ahn et al., 2022; Brohan et al., 2023). En segundo lugar, la propuesta presenta una arquitectura novedosa y replicable basada en el protocolo estándar Model Context Protocol (MCP) como capa de integración entre el LLM y ROS2, lo que facilita su extensibilidad y reutilización en otros sistemas robóticos. En tercer lugar, la demostración práctica de un robot de servicio doméstico controlable mediante lenguaje natural tiene aplicaciones directas en asistencia a personas mayores, logística doméstica y vigilancia autónoma.

1.2. Planteamiento del trabajo

La limitación identificada en el estado actual del sistema puede formularse como un problema de tres dimensiones complementarias: en primer lugar, la ausencia de representación del entorno, dado que el robot no dispone de un mapa del espacio en el que opera y no puede localizarse en él; en segundo lugar, la ausencia de planificación, ya que el robot no es capaz de calcular rutas hacia destinos específicos; y en tercer lugar, la ausencia de interfaz natural, puesto que el único modo de interacción disponible requiere conocimiento técnico explícito del stack ROS2.

Este trabajo propone una solución integral a las tres dimensiones mediante el diseño e implementación de una arquitectura en capas. En la capa de percepción, se implementará un sistema de Localización y Mapeo Simultáneo (SLAM) mediante LiDAR 2D de tecnología DTOF, que permitirá al robot construir un mapa del entorno y localizarse en él con precisión. Sobre este mapa, una base de datos de lugares etiquetados proporcionará la semántica necesaria para asociar coordenadas físicas con nombres comprensibles por humanos (“cocina”, “salón”, “pasillo”). En la capa de navegación, el stack Nav2 de ROS2 recibirá los destinos semánticos resueltos a coordenadas y calculará rutas óptimas evitando obstáculos. Finalmente, la capa de interfaz integrará un LLM conectado al sistema robótico mediante el estándar Model Context Protocol (MCP), de forma que el usuario pueda emitir comandos en lenguaje natural que el modelo traducirá en llamadas a las herramientas del robot.

El resultado esperado es un sistema capaz de ejecutar comandos como “ve a la cocina” o “patrulla el pasillo y vuelve al salón” de forma autónoma, sin que el usuario necesite conocimiento alguno sobre ROS2, coordenadas o algoritmos de planificación. Este trabajo se sitúa en la línea de investigación denominada Embodied AI, y contribuye con una arquitectura de referencia replicable basada en estándares abiertos (ROS2, MCP) sobre hardware de bajo coste.

1.3. Estructura del trabajo

El presente documento se organiza en cinco capítulos. El Capítulo 2 presenta el contexto del problema y el estado del arte, analizando desde una perspectiva de arquitectura por capas las tecnologías que sustentan el sistema: mapeo y localización mediante SLAM, planificación y

navegación con Nav2, razonamiento mediante LLMs, e integración a través del protocolo MCP. El Capítulo 3 define los objetivos concretos del trabajo y la metodología de desarrollo adoptada, articulada en cinco fases incrementales. El Capítulo 4 desarrolla la contribución específica del TFM, detallando los requisitos del sistema, el diseño de la arquitectura, la implementación de cada componente y su integración. El Capítulo 5, finalmente, recoge las conclusiones del trabajo, evalúa el cumplimiento de los objetivos planteados y propone líneas de trabajo futuro.

2. Contexto y estado del arte

Este capítulo presenta el marco teórico y el estado del arte sobre el que se fundamenta el trabajo. La estructura sigue la arquitectura en capas del sistema propuesto: se analiza primero el contexto del problema, y a continuación se estudian en profundidad las tres capas tecnológicas del sistema (percepción mediante SLAM, planificación y navegación con Nav2, e interfaz de lenguaje natural con LLMs y el Model Context Protocol), para cerrar con una síntesis de trabajos relacionados.

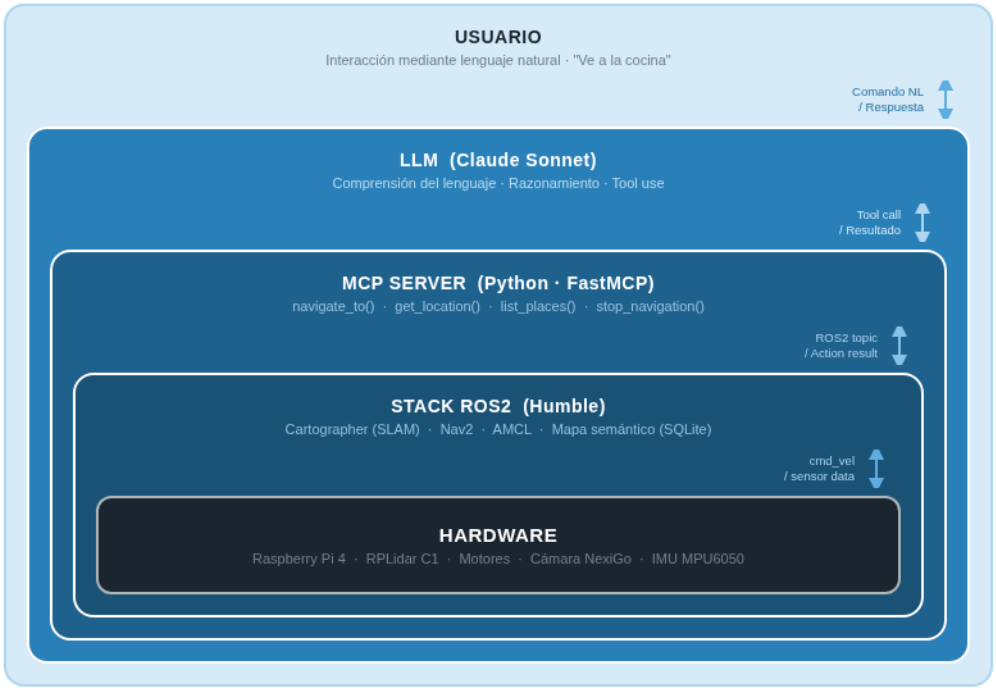

Figura 2. Arquitectura por capas del sistema propuesto

Fuente: elaboración propia.

2.1.Contexto del problema

La robótica autónoma de servicio, entendida como la capacidad de un robot móvil para desplazarse en entornos no estructurados con el objetivo de realizar tareas útiles para el ser humano, ha experimentado un avance sostenido a lo largo de las últimas décadas. Sin embargo, la interacción entre el ser humano y el robot sigue siendo, en la mayor parte de los sistemas actuales, un cuello de botella significativo: el usuario debe conocer los comandos específicos, los tópicos ROS2 correspondientes o la secuencia de acciones de bajo nivel que

dan lugar al comportamiento deseado. Este requisito limita tanto el espectro de usuarios como la aplicabilidad práctica de los sistemas robóticos (Mower et al., 2023).

El trabajo precedente de este TFM resolvió dos piezas fundamentales del rompecabezas. El primer Trabajo Fin de Grado (Higuera Castillo y Reinoso Hayashi, 2024) abordó el diseño y construcción íntegra del robot: chasis, electrónica, PCBs a medida, integración de sensores y un primer stack de software basado en ROS2 Humble. El segundo (Higuera Castillo, 2024) añadió la capacidad de navegación reactiva: seguimiento automático de carril mediante detección de bordes con el operador de Canny y la transformada de Hough, filtro de Kalman para estimación robusta del carril y controlador PID para la dirección. El resultado fue un robot capaz de seguir un carril trazado sobre el suelo sin intervención humana.

No obstante, la navegación reactiva que resulta de esa segunda etapa presenta una limitación estructural: el robot responde únicamente a estímulos sensoriales inmediatos, sin representación alguna del entorno ni capacidad de razonar sobre destinos. El robot puede seguir un carril, pero no puede planificar una ruta hacia un destino semántico (“ve a la cocina”), ni mantener una representación del mundo que le permita saber dónde está en un mapa, ni interpretar órdenes expresadas en lenguaje natural. Superar estas tres limitaciones es precisamente el problema que aborda el presente TFM.

El problema puede formularse de forma precisa: dado un robot móvil con capacidad de navegación reactiva básica, ¿cómo dotarlo de (1) localización y mapeo simultáneos del entorno doméstico, (2) navegación dirigida a objetivos semánticos, y (3) interfaz de lenguaje natural que permita a cualquier usuario interactuar con él mediante comandos conversacionales? La solución propuesta integra tres grandes áreas de investigación (SLAM, navegación autónoma con Nav2 y procesamiento de lenguaje natural mediante LLMs) en una arquitectura coherente y modular.

Figura 3: SLAM y Nav2

2.2.Capa de percepción: SLAM

La capa de percepción tiene como misión dotar al robot de conciencia espacial: saber dónde está (localización) y qué forma tiene el entorno (mapeo). El paradigma que resuelve simultáneamente ambos problemas es el SLAM (Simultaneous Localization and Mapping), que se ha convertido en uno de los campos de mayor actividad en robótica desde sus formulaciones teóricas iniciales (Durrant-Whyte y Bailey, 2006; Bailey y Durrant-Whyte, 2006).

2.2.1. Fundamentos del SLAM

El problema SLAM puede enunciarse de forma rigurosa en términos probabilísticos. Sea $x_1:_t$ la secuencia de estados del robot (posición y orientación) desde el instante inicial hasta el instante t, m el mapa del entorno (conjunto de landmarks o celdas de ocupación), y $z_1:_t$ la

secuencia de observaciones sensoriales. El objetivo del SLAM es estimar la distribución de probabilidad conjunta $P(x_{1:t}, m \mid z_{1:t}, u_{1:t})$, donde $u_{1:t}$ son los comandos de movimiento ejecutados (Thrun et al., 2005).

La dificultad del SLAM reside en la naturaleza cíclica del problema: para localizar el robot se necesita el mapa, y para construir el mapa se necesita conocer la posición del robot. Las soluciones modernas abordan esta interdependencia mediante estimación bayesiana incremental, representando la incertidumbre a través de filtros de partículas (FastSLAM), filtros de Kalman extendido (EKF-SLAM) o, en las formulaciones más recientes, métodos de optimización sobre grafos de poses (Grisetti et al., 2010).

Un concepto central en SLAM es el de cierre de bucle (loop closure): la capacidad del sistema para reconocer que el robot ha vuelto a visitar un lugar ya mapeado y corregir la deriva acumulada en la trayectoria estimada. Sin cierre de bucle, los errores de odometría se acumulan indefinidamente, degradando la calidad del mapa. Las soluciones modernas implementan el cierre de bucle mediante métodos de reconocimiento de lugar basados en descriptores visuales o en la comparación de subgrafos de escaneos Lídar (Cummins y Newman, 2011).

2.2.2. Tipos de SLAM y comparativa

La clasificación más habitual del SLAM atiende al tipo de sensor principal empleado. El SLAM visual (Visual SLAM o VSLAM) usa cámaras como sensor primario, con sistemas de referencia como ORB-SLAM3 (Campos et al., 2021) (considerado el estado del arte en SLAM monocular, estéreo y RGB-D) o RTAB-Map (Létourneau, 2013), más pragmático en su integración con ROS2. El VSLAM ofrece hardware de bajo coste, pero es sensible a las condiciones de iluminación y a la textura visual del entorno: entornos con paredes lisas de un solo color o con iluminación variable pueden provocar la pérdida de seguimiento de características (Mur-Artal et al., 2017).

El SLAM con LiDAR usa sensores de medición de distancia por tiempo de vuelo o triangulación láser, generando nubes de puntos del entorno. Sus ventajas son robustez frente a variaciones de iluminación, independencia de la textura visual y mediciones métricas precisas. Cartographer (Hess et al., 2016), desarrollado por Google, es el sistema de referencia para

SLAM 2D con LiDAR en entornos interiores, con una integración nativa en ROS2 y resultados contrastados en múltiples plataformas robóticas. Para LiDAR 3D existen alternativas como LOAM (Zhang y Singh, 2014) o FAST-LIO2 (Xu et al., 2022), aunque su complejidad computacional los hace menos adecuados para plataformas embebidas de bajo coste.

El SLAM visual-inercial (Visual-Inertial SLAM) combina cámara e IMU para aumentar la robustez ante movimiento rápido y oclusiones parciales, con sistemas como VINS-Mono (Qin et al., 2018) u ORB-SLAM3-VI. Las cámaras RGB-D (profundidad + color) como Intel RealSense permiten un SLAM más rico semánticamente, pero a un coste económico mayor. Finalmente, los sistemas híbridos multi-sensor (LVI-SAM, R3LIVE) combinan LiDAR, cámara e IMU para la máxima robustez, si bien su complejidad de calibración y carga computacional los hace inadecuados para el contexto de este TFM.

Para el presente trabajo, la decisión sobre el sensor de percepción principal se abordó con un análisis de compromiso entre robustez, coste y viabilidad sobre hardware embebido. El SLAM visual puro presenta un riesgo elevado en entornos domésticos típicos (paredes lisas, iluminación variable), con tasas de éxito estimadas del 60–70% según el contexto. El SLAM con LiDAR 2D, por el contrario, opera con independencia de las condiciones visuales, proporciona mediciones métricas precisas y se integra directamente con el stack de navegación Nav2, con tasas de éxito del 90–95% en entornos interiores estándar.

2.2.3. Cartographer como SLAM principal

Cartographer (Hess et al., 2016) es un sistema de SLAM en tiempo real desarrollado por Google que soporta configuraciones 2D y 3D. Su arquitectura se articula en torno a dos módulos complementarios: el tracking local, que mantiene una estimación de la posición del robot dentro de un submapa mediante un filtro de Kalman y la alineación de escaneos láser (scan matching); y el cierre de bucle global, que optimiza el grafo de poses completo cuando detecta que el robot regresa a una zona previamente visitada, mediante un método de optimización de grafos láxos (branch-and-bound sobre una cuadrícula jerárquica).

Desde el punto de vista de integración con ROS2, Cartographer publica el mapa de ocupación (nav_msgs/OccupancyGrid), la posición del robot (tf2 map→odom→base_link) y la trayectoria estimada, que son exactamente los mensajes consumidos por Nav2. La configuración de

parámetros de Cartographer permite adaptarse a diferentes plataformas: frecuencia del LiDAR, resolucíon del mapa, umbrales del scan matching y frecuencia del cierre de bucle. Para plataformas embebidas como la Raspberry Pi, se recomienda reducir la resolución del submapa y la frecuencia de optimización global para mantener el rendimiento en tiempo real.

2.2.4. Hardware de percepción: RPLidar C1

El sensor LiDAR seleccionado para este trabajo es el RPLidar C1 de Slamtec. Se trata de un LiDAR 2D de 360° basado en tecnología DTOF (Direct Time-of-Flight), que mide el tiempo de vuelo del pulso láser de forma directa, en contraposición a los métodos de triangulación láser empleados en modelos anteriores de la misma familia. Las características técnicas principales del RPLidar C1 son: rango máximo de 12 m, frecuencia de muestreo de 5.000 puntos por segundo, frecuencia de rotación de 7–15 Hz, precisión de ±1 cm en condiciones normales y peso inferior a 100 g.

La ventaja principal de la tecnología DTOF sobre la triangulación láser en el contexto de interiores domésticos es su mayor robustez frente a superficies reflectantes, negro profundo o materiales absorbentes, que pueden introducir errores significativos en los sensores de triangulación (Slamtec, 2024). La interfaz del sensor es USB-UART y dispone de controlador nativo para ROS2 (rplidar_ros), lo que simplifica la integración en el stack existente. El RPLidar C1 publica un topic /scan de tipo sensor_msgs/LaserScan que es directamente consumido por Cartographer y por el costmap de Nav2, sin necesidad de preprocesamiento adicional.

2.3. Capa de navegación: Nav2

Una vez que el sistema dispone de un mapa y sabe localizarse en él, la capa de navegación tiene como misión planificar y ejecutar rutas entre la posición actual del robot y un objetivo dado. Nav2 (Navigation2) es el stack oficial de navegación de ROS2 y el sucesor del clásico ROS Navigation Stack, completamente reescrito para soportar el ciclo de vida de nodos de ROS2, comunicación entre nodos mediante DDS (Data Distribution Service) y la nueva API de acciones (Nav2, 2024).

2.3.1. Arquitectura del stack Nav2

La arquitectura de Nav2 sigue el patrón de composición de comportamientos basado en árboles de comportamiento (Behavior Trees, BT). El componente central es el BT Navigator,

que orquesta la ejecución de tareas de navegación a través de un XML que describe el árbol de comportamiento activo. Este diseño facilita la personalización y extensión sin modificar el código fuente: añadir nuevos comportamientos de recuperación, cambiar el planificador o modificar la condición de éxito son operaciones que se realizan editando el BT XML (Macenski et al., 2020).

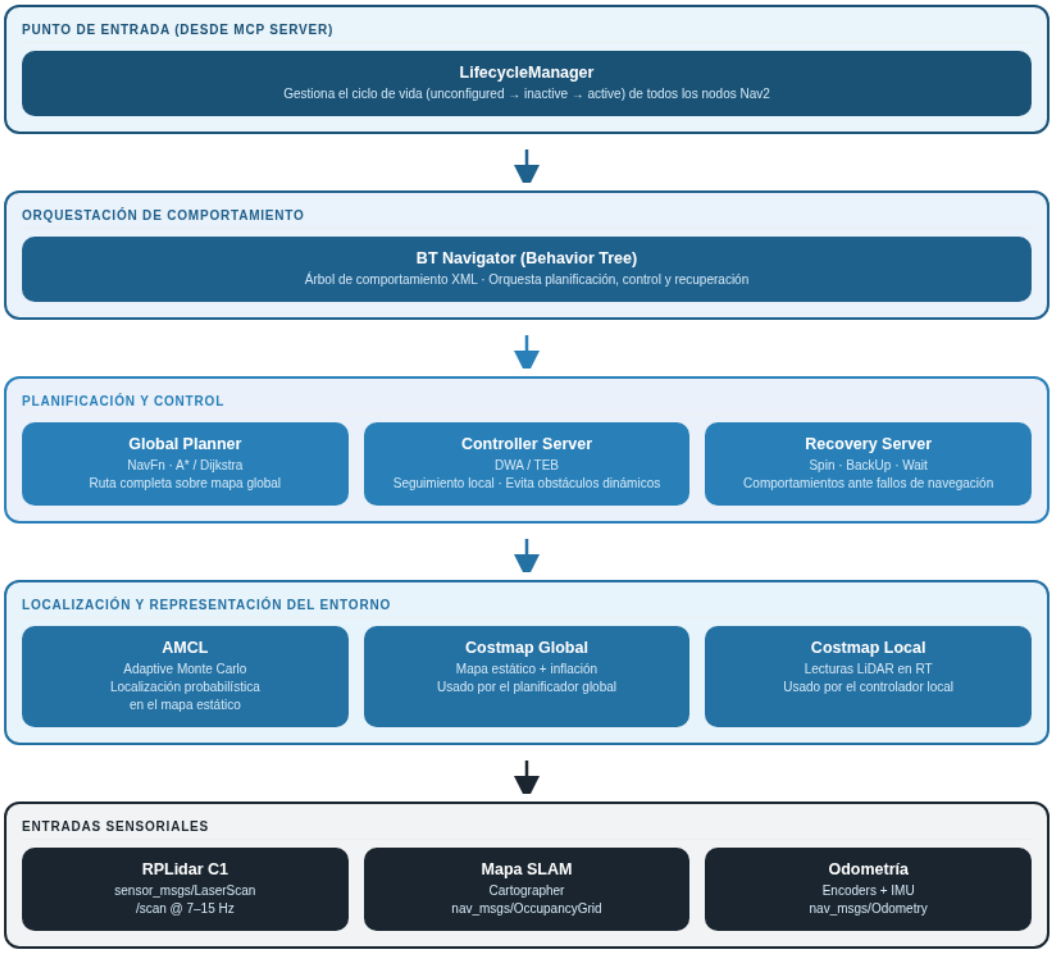

Figura 4: Arquitectura interna del stack Nav2 (ROS2 Humble)

Fuente: elaboración propia.

Los nodos que componen el stack son gestionados mediante el LifecycleManager, que implementa el modelo de ciclo de vida de ROS2 (unconfigured → inactive → active → finalized). Esto garantiza que los nodos se inicialicen en el orden correcto, que las dependencias estén satisfechas antes de activar cada componente, y que el sistema pueda detenerse de forma ordenada. Entre los nodos gestionados se encuentran el planificador global, el controlador local, el servidor de mapa de coste y el servidor de recuperación.

2.3.2. Planificación global y control local

La planificación en Nav2 se divide en dos niveles. El planificador global (Global Planner) calcula una ruta completa desde la posición actual hasta el destino, usando el mapa de ocupación global. El planificador por defecto es NavFn, que implementa el algoritmo de Dijkstra o A* sobre el mapa de coste global. NavFn garantiza optimalidad bajo condiciones estáticas, aunque alternativas como Theta* (Nash et al., 2010) o Smac Planner (Macenski et al., 2021) ofrecen trayectorias más suaves o soportan restricciones cinéticas más exigentes.

El controlador local (Local Planner o Controller Server) se encarga de seguir la ruta global adaptándose a obstáculos dinámicos no presentes en el mapa global. El controlador por defecto en Nav2 es DWA (Dynamic Window Approach), que muestrea un espacio de velocidades admisibles (ventana dinámica), simula trayectorias y selecciona la que optimiza una función de coste que pondera progreso hacia el objetivo, distancia a obstáculos y suavidad del movimiento (Fox et al., 1997). Una alternativa muy usada en plataformas ackermann (vehículos tipo coche) es TEB (Timed Elastic Band), que optimiza la trayectoria localmente como una banda elástica parameterizada en el tiempo, respetando explícitamente las restricciones cinéticas del robot (Rösmann et al., 2013).

2.3.3. Costmaps y localización

Los mapas de coste (costmaps) son representaciones 2D del entorno que asignan un coste a cada celda en función de la proximidad a obstáculos. Nav2 mantiene dos costmaps: el global, construido sobre el mapa estático generado por SLAM y usado por el planificador global; y el local, actualizado en tiempo real con las lecturas del LiDAR y usado por el controlador local. Cada costmap puede configurar capas (layers): la capa estática, la capa de inflación (que añade un margen de seguridad alrededor de los obstáculos proporcional al radio del robot) y la capa de obstáculos (que integra las lecturas sensoriales en tiempo real).

Para la localización en tiempo real, Nav2 usa AMCL (Adaptive Monte Carlo Localization), un filtro de partículas que estima la distribución de probabilidad sobre la posición del robot dado el mapa ya construido. AMCL ajusta dinámicamente el número de partículas según la incertidumbre estimada: cuando la localización es precisa, reduce el número de partículas para ahorrar cómputo; cuando detecta alta incertidumbre (por ejemplo, tras una colisión o un

período sin movimiento), lo incrementa (Thrun et al., 2005). El resultado es una estimación robusta de la posición del robot en el mapa durante la navegación.

2.4. Capa de interfaz natural: LLMs y MCP

La capa de interfaz natural es la contribución más novedosa de este TFM. Su función es servir de puente entre el lenguaje humano y las capacidades robóticas del sistema, permitiendo que cualquier usuario pueda controlar el robot mediante comandos conversacionales sin necesidad de conocer la arquitectura subyacente.

2.4.1. Grandes modelos de lenguaje y uso de herramientas

Los Grandes Modelos de Lenguaje (LLMs, Large Language Models) son modelos de redes neuronales entrenados sobre corpus masivos de texto mediante objetivos de modelado del lenguaje. La arquitectura Transformer (Vaswani et al., 2017), con sus mecanismos de atención multi-cabeza, es la base de todos los LLMs modernos de referencia: GPT-4 (OpenAI, 2023), Claude (Anthropic, 2024), Gemini (Google, 2024) y Llama 3 (Meta AI, 2024). Estos modelos han demostrado capacidades emergentes de razonamiento, planificación de tareas y comprensión contextual que van muy más allá de la generación de texto (Wei et al., 2022).

Una de las capacidades más relevantes para este trabajo es el uso de herramientas (tool use o function calling), introducido por OpenAI en 2023 y adoptado rápidamente por el resto de proveedores. Mediante esta capacidad, el LLM no solo genera texto, sino que puede emitir llamadas estructuradas a funciones externas definidas por el desarrollador, interpretando la respuesta de dichas funciones para continuar su razonamiento. Esto convierte al LLM en un agente capaz de interactuar con sistemas externos de forma estructurada y controlada (Schick et al., 2023). El LLM actúa como un planificador de alto nivel que descompone el objetivo del usuario en una secuencia de llamadas a herramientas, cada una de las cuales corresponde a una acción concreta del sistema robótico.

2.4.2. Embodied AI: LLMs en robótica

La integración de LLMs con sistemas robóticos se enmarca en el paradigma de la Embodied AI, que estudia la inteligencia artificial en agentes con cuerpo físico capaces de percibir y actuar sobre el mundo real. Los trabajos pioneros en esta línea incluyen SayCan (Ahn et al., 2022), que combina las capacidades de razonamiento de un LLM con la viabilidad física

estimada por una función de valor aprendida, permitiendo que el robot seleccione acciones que sean tanto relevantes para el objetivo como ejecutables en el contexto actual. RT-2 (Zitkovich et al., 2023) va más allá y entrena un modelo de visión-lenguaje-acción end-to-end que genera comandos de control directamente desde la imagen y el texto.

En el ámbito de la navegación en lenguaje natural (Vision-and-Language Navigation, VLN), trabajos como LM-Nav (Shah et al., 2023) demuestran que los LLMs pueden descomponer instrucciones de navegación en términos semánticos y anclarlos a representaciones del entorno. La aproximación de este TFM es más pragmática y eficiente en cómputo: en lugar de un modelo end-to-end, se usa un LLM como planificador de alto nivel que invoca herramientas del sistema de navegación mediante un protocolo estándar, desacoplando la lógica de lenguaje de la lógica de movimiento y favoreciendo la modularidad y la extensibilidad del sistema.

2.4.3. Model Context Protocol (MCP)

El Model Context Protocol (MCP) es un estándar abierto publicado por Anthropic en noviembre de 2024 que define un protocolo de comunicación entre aplicaciones LLM (hosts) y servidores de herramientas externas (Anthropic, 2024). La arquitectura MCP se articula en tres roles: el host, que es la aplicación que gestiona el LLM y la interfaz con el usuario; el cliente MCP, embebido en el host, que se comunica con los servidores; y el servidor MCP, que expone un conjunto de herramientas (tools), recursos (resources) y prompts predefinidos al LLM.

Desde el punto de vista del protocolo, MCP define tres primitivas fundamentales. Las herramientas (tools) son funciones que el LLM puede invocar para realizar acciones en el sistema externo; corresponden al mecanismo de function calling de los LLMs, pero con un formato de descripción estándar. Los recursos (resources) son fuentes de datos que el servidor expone al LLM, como el estado del robot o el listado de lugares conocidos. Los prompts son plantillas predefinidas que el servidor puede ofrecer al LLM para guiar su comportamiento. La comunicación entre cliente y servidor puede realizarse mediante stdio (proceso local) o HTTP con SSE (Server-Sent Events) para servicios remotos.

La relevancia de MCP para este trabajo es doble. Primero, proporciona un protocolo estándar que desacopla el LLM del sistema robótico: el servidor MCP puede implementarse,

modificarse y extenderse sin cambios en la lógica del LLM o en el stack ROS2. Segundo, al ser un estándar abierto con soporte nativo en los principales LLMs (Claude, GPT-4, Gemini), garantiza que la arquitectura desarrollada en este TFM sea transferible a otros modelos y plataformas, contribuyendo a su replicabilidad científica.

2.4.4. Arquitectura LLM—MCP—ROS2

La integración de las tres capas en una arquitectura coherente requiere definir cómo fluye la información desde el usuario hasta el actuador físico. El LLM recibe el comando del usuario en lenguaje natural y razona sobre qué herramientas invocar y en qué orden. El servidor MCP actúa como pasarela: traduce cada llamada de herramienta del LLM en una operación concreta sobre el sistema ROS2 (publicación de un tópico, llamada a un servicio o acción ROS2) y devuelve el resultado al LLM. El stack Nav2, al recibir el objetivo de navegación, planifica y ejecuta la ruta de forma autónoma, notificando su complección al servidor MCP, que a su vez informa al LLM.

Este diseño por capas tiene propiedades arquitecturales importantes. La separación de responsabilidades garantiza que cada capa puede evolucionar independientemente: se puede sustituir el LLM (por ejemplo, de Claude a Llama 3 local) sin modificar el servidor MCP, o añadir nuevas herramientas robóticas al servidor MCP sin cambiar la lógica de navegación. La sincronía entre el LLM y el robot se gestiona mediante el patrón request-response del protocolo de acciones de ROS2 (rclpy action client/server), que bloquea la ejecución hasta que el robot alcanza el destino o el navegador reporta un fallo, momento en que el MCP retorna el resultado al LLM para que continúe su razonamiento.

Figura 2. Diagrama de secuencia: procesamiento del comando "Ve a la cocina"

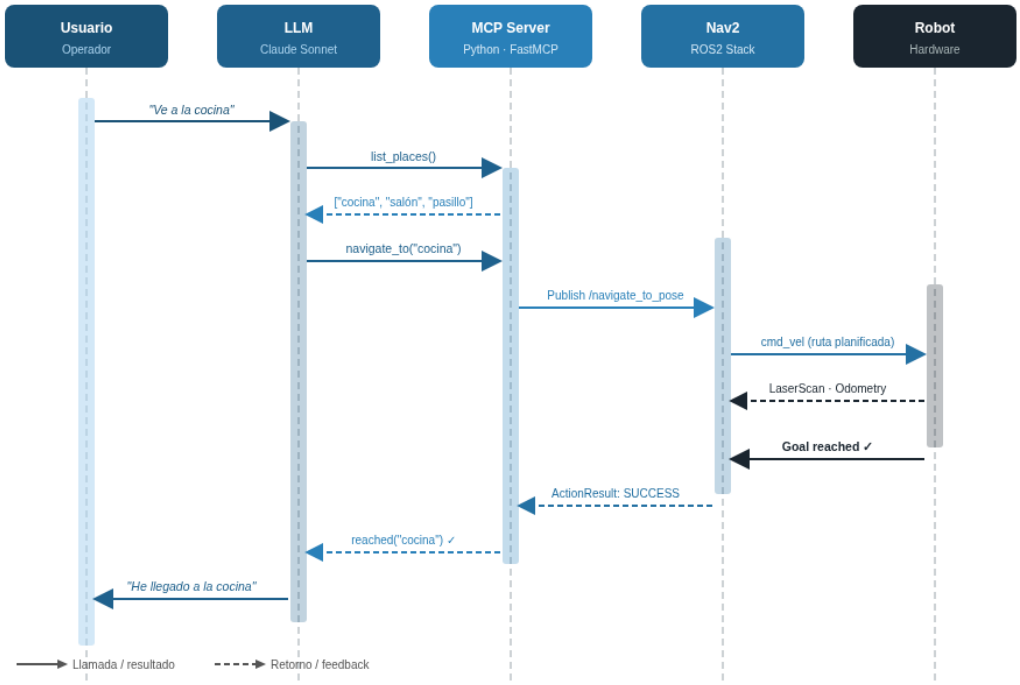

Figura 5: Procesamiento del comando "Ve a la cocina"

Fuente: elaboración propia.

2.5. Trabajos relacionados y síntesis

La propuesta de este TFM se sitúa en la confluencia de tres líneas de investigación activas. En SLAM para interiores domésticos, trabajos como el de Hess et al. (2016) con Cartographer o el de Mur-Artal et al. (2017) con ORB-SLAM3 han establecido los sistemas de referencia. Para plataformas de bajo coste, trabajos como el de Macenski et al. (2020) sobre el robot TurtleBot3 demuestran la viabilidad del stack Nav2 completo en hardware equivalente al Raspberry Pi 4. En la línea de interfaces LLM para robótica, la literatura reciente es extensa: Inner Monologue (Huang et al., 2022), LLM-Planner (Song et al., 2023) y Code as Policies (Liang et al., 2023) proponen distintas estrategias para conectar LLMs con capacidades robóticas.

La diferenciación de este trabajo respecto a los anteriores reside en tres aspectos. Primero, la elección de MCP como protocolo de comunicación estandarizado entre el LLM y el robot, frente a las soluciones ad hoc basadas en prompts o llamadas directas a la API que predominan en la literatura. Segundo, la construcción sobre hardware real de bajo coste (Raspberry Pi 4,

RPLidar C1) y software totalmente de código abierto (ROS2, Cartographer, Nav2, FastMCP), lo que hace la arquitectura accesible y replicable. Tercero, la continuidad con un sistema hardware preexistente (los dos TFGs precedentes) que proporciona una plataforma robusta y documentada sobre la que construir, en lugar de partir de cero.

En síntesis, el estado del arte revisado en este capítulo justifica las decisiones de diseño adoptadas: LiDAR 2D con Cartographer para un SLAM robusto en entornos domésticos, Nav2 como stack de navegación estándar y probado en ROS2, y la arquitectura LLM—MCP—ROS2 como interfaz de lenguaje natural que maximiza la modularidad y la transferibilidad del sistema. El capítulo siguiente concreta estos principios en objetivos específicos y una metodología de desarrollo estructurada en cinco fases.

3. Objetivos concretos y metodología de trabajo

Este capítulo concreta los objetivos del trabajo y la metodología mediante la que se abordarán, constituyendo el puente entre el estado del arte analizado en el capítulo anterior y el desarrollo que se llevará a cabo. Se presentan el objetivo general, los cuatro objetivos específicos derivados de él, la metodología de desarrollo adoptada y las métricas que permitirán valorar el éxito del trabajo.

3.1. Objetivo general

El objetivo general de este TFM es diseñar e implementar una arquitectura software que integre técnicas de SLAM, planificación de rutas y Grandes Modelos de Lenguaje para dotar al robocar de la capacidad de interpretar comandos en lenguaje natural y ejecutar navegación autónoma goal-directed en entornos interiores controlados.

Este objetivo supera la limitación fundamental del trabajo previo (la navegación estrictamente reactiva basada en seguimiento de carril) y añade tres capacidades nuevas: conciencia espacial del entorno (mapa y localización), navegación dirigida a destinos semánticos (“ve a la cocina”) e interfaz de lenguaje natural como único punto de entrada para el usuario. La aportación central es la arquitectura LLM—MCP—ROS2, que establece un protocolo estándar y modular de comunicación entre el razonador de alto nivel (LLM) y el sistema robótico (ROS2).

3.2. Objetivos específicos

El objetivo general se descompone en cuatro objetivos específicos, formulados de acuerdo con el criterio SMART (específicos, medibles, alcanzables, relevantes y acotados en el tiempo):

OE1. Implementar un sistema de SLAM 2D con LiDAR (Cartographer + RPLidar C1) sobre el robocar existente, capaz de generar un mapa de ocupación del entorno y mantener la localización del robot con un error medio inferior a 10 cm, y construir sobre él un mapa semántico con al menos tres lugares etiquetados almacenados en una base de datos SQLite.

OE2. Integrar el stack de navegación Nav2 (ROS2 Humble) sobre el mapa generado, logrando que el robot complete con éxito más del 90 % de los intentos de navegación

entre pares de waypoints conocidos en un entorno de prueba controlado, sin obstáculos dinámicos.

OE3. Diseñar e implementar un MCP Server en Python con las herramientas navigate_to(), get_current_location(), list_known_places() y stop_navigation(), que sirva de pasarela entre el LLM y el sistema ROS2, gestionando el ciclo de vida de las acciones de navegación y retornando resultados estructurados al modelo.

OE4. Validar el pipeline end-to-end mediante una batería de pruebas funcionales que cubra comandos simples (navegación directa), comandos compuestos (secuencias de destinos) y casos de error (lugar desconocido, navegación fallida), alcanzando una tasa de interpretación correcta de comandos válidos superior al 85 %.

3.3. Metodología de desarrollo

Se adopta una metodología de desarrollo en espiral incremental, estructurada en cinco fases secuenciales con validación explícita al final de cada una. Este enfoque permite detectar problemas de integración de forma temprana y ajustar el alcance si alguna fase presenta dificultades imprevistas, antes de comprometer las fases siguientes. Cada fase tiene un criterio de éxito definido que actúa como puerta de entrada a la siguiente.

Fase 1: SLAM y mapa semántico (3–4 semanas)

Integración del RPLidar C1 en el stack ROS2 mediante el paquete rplidar_ros. Configuración de Cartographer para SLAM 2D en interiores. Teleoperación del robot para construir el mapa del entorno de prueba. Creación de la base de datos SQLite con etiquetas semánticas de lugares clave. Criterio de éxito: mapa coherente generado y robot localizándose sobre él con error inferior a 10 cm.

Fase 2: Navegación goal-directed con Nav2 (3–4 semanas)

Configuración del stack Nav2 (BT Navigator, planificador global NavFn, controlador DWA, costmaps y AMCL) sobre el mapa generado en la fase anterior. Implementación del servicio de resolución semántica: nombre de lugar → coordenadas. Validación de navegación point-to-point entre waypoints del mapa en entorno controlado. Criterio de éxito: tasa de éxito de navegación superior al 90 %.

Fase 3: MCP Server y bridge ROS2 (2–3 semanas)

Diseño e implementación del MCP Server en Python con FastMCP. Implementación de las cuatro herramientas: navigate_to(), get_current_location(), list_known_places() y stop_navigation(). Integración con rclpy para publicar objetivos de navegación como acciones ROS2 y recibir el resultado. Criterio de éxito: herramientas invocables manualmente desde cliente MCP con comportamiento correcto verificado.

Fase 4: Integración con LLM (2–3 semanas)

Conexión del MCP Server con la Claude API. Diseño del system prompt que define el rol del LLM como asistente de navegación con acceso a las herramientas del robot. Implementación del cliente host que gestiona el bucle de conversación y las llamadas a herramientas. Criterio de éxito: pipeline completo funcionando (desde el comando de texto del usuario hasta el movimiento del robot) sin intervención manual.

Fase 5: Pruebas, evaluación y documentación (2–3 semanas)

Diseño y ejecución de la batería de pruebas end-to-end: comandos simples, secuencias, lugares desconocidos y comandos ambiguos. Recogida y análisis de métricas. Identificación de casos de fallo y refinamiento. Redacción de la memoria final y preparación de la defensa.

3.4. Métricas de éxito

La evaluación del trabajo se realizará sobre cuatro métricas cuantitativas, directamente vinculadas a los objetivos específicos:

Error de localización (OE1): error cuadrático medio entre la posición estimada por AMCL y la posición real del robot, medido en un conjunto de puntos de referencia marcados en el mapa. Umbral de éxito: error medio inferior a 10 cm.

Tasa de éxito de navegación (OE2): porcentaje de intentos de navegación entre pares de waypoints que concluyen con el robot dentro de un radio de 20 cm del destino, sobre un total de al menos 20 ensayos. Umbral de éxito: superior al 90 %.

Tasa de interpretación correcta (OE4): porcentaje de comandos válidos en lenguaje natural que el LLM traduce a la secuencia correcta de llamadas a herramientas, sobre un conjunto de prueba de al menos 20 comandos categorizados. Umbral de éxito: superior al 85 %.

Latencia de respuesta (OE3–OE4): tiempo transcurrido desde que el usuario envía el comando hasta que el robot inicia el movimiento, medido sobre 10 ensayos. Umbral de éxito: mediana inferior a 5 segundos en condiciones de conectividad normal con la API.

4. Desarrollo específico de la contribución

Pendiente para entrega 2-3

5. Conclusiones y trabajo futuro

Pendiente para entrega 3

5.1. Conclusiones

5.2. Líneas de trabajo futuro

Referencias bibliográficas

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Ruano, R. J., Jeffrey, K., … Zeng, A. (2022). Do as I can, not as I say: Grounding language in robotic affordances. En *Proceedings of the 6th Conference on Robot Learning (CoRL 2022)* (pp. 287–318). PMLR.

Anthropic. (2024). *Model Context Protocol specification*. https://modelcontextprotocol.io/

Bailey, T., & Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, 13(3), 108–117. https://doi.org/10.1109/MRA.2006.1638022

Campos, C., Elvira, R., Rodríguez, J. J. G., Montiel, J. M. M., & Tardós, J. D. (2021). ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM. *IEEE Transactions on Robotics*, 37(6), 1874–1890. https://doi.org/10.1109/TRO.2021.3075644

Cummins, M., & Newman, P. (2011). Appearance-only SLAM at large scale with FAB-MAP 2.0. *The International Journal of Robotics Research*, 30(9), 1100–1123. https://doi.org/10.1177/0278364910385483

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: Part I. *IEEE Robotics & Automation Magazine*, 13(2), 99–110. https://doi.org/10.1109/MRA.2006.1638022

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1), 23–33. https://doi.org/10.1109/100.580977

Goodrich, M. A., & Schultz, A. C. (2007). Human-robot interaction: A survey. *Foundations and Trends in Human-Computer Interaction*, 1(3), 203–275. https://doi.org/10.1561/1100000005

Grisetti, G., Kümmerle, R., Stachniss, C., & Burgard, W. (2010). A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, 2(4), 31–43. https://doi.org/10.1109/MITS.2010.939925

Hess, W., Kohler, D., Rapp, H., & Andor, D. (2016). Real-time loop closure in 2D LiDAR SLAM. En *2016 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1271–1278). IEEE. https://doi.org/10.1109/ICRA.2016.7487258

Higuera Castillo, R. (2023). *Diseño y desarrollo de un robot coche autónomo de bajo coste con ROS2* [Trabajo de Fin de Grado, Universidad Internacional de La Rioja].

Higuera Castillo, R., & Reinoso Hayashi, K. (2024). *Diseño e implementación de seguimiento de carril de Robocar* [Trabajo de Fin de Grado, Universidad Internacional de La Rioja].

Huang, W., Abbeel, P., Pathak, D., & Mordatch, I. (2022). Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. En *Proceedings of the 39th International Conference on Machine Learning (ICML 2022)* (pp. 9118–9147). PMLR.

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., Sermanet, P., Brown, N., Jackson, T., Luu, L., Levine, S., Hausman, K., & Ichter, B. (2022). Inner monologue: Embodied reasoning through planning with language models. En *Proceedings of the 6th Conference on Robot Learning (CoRL 2022)*. PMLR.

Labbé, M., & Michaud, F. (2019). RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2), 416–446. https://doi.org/10.1002/rob.21831

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., & Zeng, A. (2023). Code as policies: Language model programs for embodied control. En *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. https://doi.org/10.1109/ICRA48891.2023.10160591

Macenski, S., Martín, F., White, R., & Clavero, J. G. (2020). The Marathon 2: A navigation system. En *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 2718–2725). IEEE. https://doi.org/10.1109/IROS45743.2020.9341207

Macenski, S., & Jambrecic, I. (2021). SMAC Planner: A flexible, production-ready path planner for ROS 2 autonomous mobile robots. En *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. https://doi.org/10.1109/IROS51168.2021.9636566

Mur-Artal, R., & Tardós, J. D. (2017). ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5), 1255–1262. https://doi.org/10.1109/TRO.2017.2705103

Nash, A., Daniel, K., Koenig, S., & Felner, A. (2007). Theta*: Any-angle path planning on grids. En *Proceedings of the 22nd AAAI Conference on Artificial Intelligence* (pp. 1177–1183). AAAI Press.

OpenAI. (2023). *GPT-4 technical report*. https://arxiv.org/abs/2303.08774

Qin, T., Li, P., & Shen, S. (2018). VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4), 1004–1020. https://doi.org/10.1109/TRO.2018.2853729

Rösmann, C., Feiten, W., Wösch, T., Hoffmann, F., & Bertram, T. (2013). Efficient trajectory optimization using a sparse model. En *2013 European Conference on Mobile Robots (ECMR)* (pp. 138–143). IEEE. https://doi.org/10.1109/ECMR.2013.6698833

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2024). Toolformer: Language models can teach themselves to use tools. En *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*. Curran Associates.

Shah, D., Osiński, B., Murphy, B., & Levine, S. (2023). LM-Nav: Robotic navigation with large pre-trained models of language, vision, and action. En *Proceedings of the 6th Conference on Robot Learning (CoRL 2022)*. PMLR.

Slamtec. (2023). *RPLIDAR C1 product brief*. https://www.slamtec.com/en/Lidar/C1

Song, C. H., Wu, J., Washington, C., Sadler, B. M., Chao, W.-L., & Su, Y. (2023). LLM-Planner: Few-shot grounded planning for embodied agents with large language models. En *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV 2023)* (pp. 2998–3009). IEEE.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. MIT Press.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. En *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. Curran Associates.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., & Fedus, W. (2022). Emergent abilities of large language models. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=yzkSU5zdwD

Xu, W., Cai, Y., He, D., Lin, J., & Zhang, F. (2022). FAST-LIO2: Fast direct LiDAR-inertial odometry. *IEEE Transactions on Robotics*, 38(4), 2053–2073. https://doi.org/10.1109/TRO.2022.3141876

Zhang, J., & Singh, S. (2014). LOAM: Lidar odometry and mapping in real-time. En *Robotics: Science and Systems X*. https://doi.org/10.15607/RSS.2014.X.007

Zitkovich, B., Yu, T., Xu, S., Xu, P., Xiao, T., Xia, F., Wu, J., Wohlhart, P., Welker, S., Wahid, A., Vuong, Q., Vanhoucke, V., Tran, H., Soricut, R., Sermanet, P., Peng, X. B., Singh, K., Salazar, R., Ryoo, M. S., … Zeng, A. (2023). RT-2: Vision-language-action models transfer web knowledge to robotic control. En *Proceedings of the 7th Conference on Robot Learning (CoRL 2023)*. PMLR.

Anexo A. Código fuente y datos analizados

https://github.com/rubenhigorg/robocar