



GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE

Escuela de Ingeniería de Fuenlabrada

Curso académico 2024-2025

Trabajo Fin de Grado

Robot Nao en ROS 2

Del flasheo a la empatía: migración, interacción humano-robot
y navegación autónoma

Tutor: Rodrigo Pérez Rodríguez

Autor: Antonio Roldán Sandúa



Este trabajo se distribuye bajo los términos de la licencia internacional CC BY-NC-SA 4.0 (Creative Commons Atribución-NoComercial-CompartirIgual 4.0).

Usted es libre de:

- **Compartir:** copiar y redistribuir el material en cualquier medio o formato.
- **Adaptar:** remezclar, transformar y crear a partir del material.

El licenciante no puede revocar estas libertades mientras se cumplan los términos de la licencia:

- **Atribución:** Debe otorgar el crédito adecuado, incluir un enlace a la licencia e indicar si se han realizado cambios. Puede hacerlo de cualquier manera razonable, pero no de forma que sugiera que usted o su uso cuentan con el respaldo del licenciante.
- **No Comercial:** No puede utilizar el material con fines comerciales.
- **Compartir Igual:** Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia que el original.

Agradecimientos

Quiero agradecer a mi familia por haberme apoyado durante todos estos meses de duro trabajo, y ahora que estoy terminando la carrera, por haber hecho posible que estudie lo que siempre he querido. Han sido unos años difíciles, viviendo lejos de ellos, pero he sentido siempre su apoyo a pesar de la distancia.

También quiero dar las gracias a mis compañeros y amigos de la carrera, por haber hecho que estos años hayan sido más llevaderos, y por haber demostrado siempre un gran compañerismo; sin ellos no habría sido posible terminar mis estudios.

Muchas gracias a mi tutor del TFG, por darme la oportunidad y confiar en mí para la realización de este trabajo.

Ahora que he terminado, ya no volveré a quedarme encerrado en el laboratorio por pasarme de la hora de cierre, aunque creo que, en parte, echaré de menos esas largas tardes. Aprovecho para agradecer a la conserje su amabilidad y pedirle perdón por interrumpirle la merienda para que me abriera la puerta tantos días.

Han sido muchos días, muchas horas y muchos cafés, pero ha merecido la pena, aunque solo sea por todo lo que he podido aprender durante la realización de este trabajo.

Dedicado a mi madre y a mi padre, por la fuerza para salir adelante que han demostrado este último año.

Madrid, 11 de julio de 2025

Antonio Roldán Sandúa

Resumen

La robótica ha avanzado enormemente en los últimos años, y los robots humanoides, como el Nao, han jugado un papel crucial en la investigación en áreas como *Human-Robot Interaction* (HRI), educación y robótica autónoma. El uso de plataformas de desarrollo propietarias como en el caso del robot Nao, que originalmente viene con su propio sistema operativo, *NAOqi*, ha sido común en diversas investigaciones. Sin embargo, la evolución de herramientas y frameworks como *Robot Operating System 2* (ROS 2) ofrece nuevas posibilidades para integrar el robot Nao con un sistema operativo abierto y flexible, lo que permite acceder a un ecosistema más amplio de bibliotecas y herramientas.

En este contexto, el objetivo principal del presente trabajo es migrar el robot Nao V6 [1] de su sistema operativo NAOqi a un sistema basado en Linux, específicamente *Ubuntu 22.04*, y hacer uso del middleware ROS 2, concretamente la distribución de desarrollo *Rolling*. Esta migración no solo permite aprovechar las herramientas avanzadas de ROS 2, sino que también mejora la flexibilidad y la capacidad de integración del robot Nao con otros robots y sistemas, convirtiéndolo en una plataforma más poderosa para la investigación. Además, uno de los objetivos de este trabajo es facilitar esta migración para los investigadores, ya que la información disponible actualmente está dispersa y, en muchos casos, es difícil de acceder, lo que complica considerablemente el proceso.

En respuesta a la problemática expuesta, se han implementado varias soluciones clave para facilitar dicha migración. Como contribución principal, se ha creado una serie de *README* detallados (actualmente en español, próximamente en inglés) que documentan de manera clara y estructurada todo el proceso, desde la creación de la imagen de *Ubuntu* y su flasheo en el Nao, hasta la instalación y uso de los diferentes paquetes necesarios para el control y la interacción del robot. Estos documentos permiten replicar fácilmente el proceso de instalación y configuración, eliminando la dispersión de información y facilitando el acceso a los recursos necesarios [2].

Además, se han mejorado una serie de paquetes existentes, con el objetivo de incrementar la seguridad y añadir nuevas funcionalidades. También se ha desarrollado un nuevo paquete que integra múltiples funcionalidades, combinando y coordinando HRI y locomoción en un solo módulo, convirtiendo al Nao en un robot mucho más completo y versátil.

Finalmente, se ha integrado el robot Nao en el sistema Nav2 de ROS 2, mediante el cálculo de odometría con algoritmos de fusión de sensores y la adición de un sensor *Light Detection and Ranging* (LiDAR), lo que proporciona capacidades de navegación autónoma y mejora aún más la funcionalidad del robot en entornos dinámicos controlados.

Los experimentos se han realizado con un robot real Nao V6 en el Laboratorio de Robótica y Sistemas Ubícuos de la Escuela de Ingeniería de Fuenlabrada de la Universidad Rey Juan Carlos. Los resultados que se han obtenido han sido favorables, logrando un funcionamiento generalmente correcto del sistema. Además, la documentación creada ha sido probada y se ha confirmado la facilidad para seguirla, permitiendo configurar el robot con una instalación base sólida, sobre la que sería posible para los investigadores desarrollar funcionalidades más avanzadas en el futuro.

En un desarrollo más avanzado a partir de esta base, se ha dotado al robot de la capacidad de realizar gestos, caminar, levantarse tras caídas y navegar autónomamente, y se ha conseguido una interacción fluida y natural con el robot, permitiendo asignarle una personalidad específica, por ejemplo, orientada a la asistencia a personas mayores, y mantener conversaciones coherentes con él.

Finalmente, se ha comprobado experimentalmente que el robot es capaz de navegar de forma autónoma usando el sistema Nav2 y Simultaneous Localization and Mapping (SLAM), generando su propio mapa y moviéndose de un punto a otro sin intervención externa. Esto confirma que la integración funciona en la práctica y que el Nao puede desenvolverse por sí mismo en un entorno real, combinando todas sus capacidades (gestos, locomoción, recuperación ante caídas y navegación) de manera coordinada.

Acrónimos

HRI *Human-Robot Interaction*

ROS *Robot Operating System*

ROS 1 *Robot Operating System 1*

ROS 2 *Robot Operating System 2*

LiDAR *Light Detection and Ranging*

Wi-Fi *Wireless Fidelity*

CMOS *Complementary Metal-Oxide-Semiconductor*

FSR *Force Sensitive Resistors*

IMU *Inertial Measurement Unit*

LED *Light Emitting Diode*

RGB *Red Green Blue*

RAM *Random Access Memory*

RTX *Ray Tracing Texel eXtreme*

GPU *Graphics Processing Unit*

CPU *Central Processing Unit*

GB *Gigabyte*

USB *Universal Serial Bus*

Hz *Hertz*

GHz *Gigahertz*

SSH *Secure Shell*

PC *Personal Computer*

LoLA *Low Level Abstraction*

2-D *Two-Dimensional*

3-D *Three-Dimensional*

PCL *Point Cloud Library*

TTS *Text To Speech*

STT *Speech To Text*

V4L2 *Video for Linux 2*

TF *Transform Frame*

VS Code *Visual Studio Code*

DDS *Data Distribution Service*

API *Application Programming Interface*

SPL *Standard Platform League*

STEM *Science, Technology, Engineering and Mathematics*

LLM *Large Language Model*

URDF *Unified Robot Description Format*

ENU *East-North-Up*

OpenCV *Open Source Computer Vision Library*

cv2 *OpenCV Python interface*

SLAM *Simultaneous Localization and Mapping*

GPT *Generative Pre-trained Transformer*

ASR *Automatic Speech Recognition*

VAD *Voice Activity Detection*

CUDA *Compute Unified Device Architecture*

FTDI *Future Technology Devices International*

QoS *Quality of Service*

TFG *Trabajo de Fin de Grado*

Índice general

1. Introducción	1
1.1. Contexto tecnológico	1
1.2. El robot Nao	3
1.3. Objetivos	5
2. Integración de Nao en ROS 2	7
2.1. Introducción	7
2.2. Hardware empleado	7
2.3. Software y herramientas	9
2.3.1. Instalación del sistema operativo	11
2.4. Base funcional del robot Nao en ROS 2	12
2.4.1. Nodo nao_lola: interfaz con LoLA	12
2.4.2. Publicación de transformaciones (TF)	13
2.4.3. Odometría: estimación de orientación y posición del Nao	15
2.4.4. Conversión y calibración de la cámara del Nao	19
2.4.5. Simulación con Webots y publicación de cámara	21
2.5. Conclusión	23
3. Marcha bípeda en el Nao con el paquete walk	24
3.1. Introducción	24
3.2. Estructura del paquete walk	24
3.3. Modificaciones personalizadas	25
3.3.1. Sincronización de brazos	25
3.3.2. Ajuste de parámetros para mejorar la estabilidad	26
3.3.3. Recuperación tras caída	27
3.4. Preparación postural y lógica de arranque	29
3.5. Activación y control	30
3.6. Cálculo de la posición para la odometría	30
3.7. Pruebas realizadas	33

3.8. Conclusión	34
4. Interacción Humano–Robot en el Nao	36
4.1. Uso de <code>nao_pos</code> para gestos predefinidos	36
4.1.1. Formato de los archivos <code>.pos</code>	37
4.1.2. Funcionamiento del nodo <code>nao_pos_server</code>	37
4.2. Reconocimiento de voz (STT)	38
4.3. Modelo de diálogo con GPT (LLM)	39
4.4. Síntesis de voz (TTS)	39
4.5. Detección y seguimiento de rostros	40
4.6. Coordinación de nodos para un HRI completo	41
4.7. Pruebas realizadas y conclusiones	45
5. Navegación autónoma en el robot Nao	46
5.1. Introducción	46
5.2. Limitaciones del hardware: conexión del sensor LiDAR	47
5.3. Implementación de SLAM en el Nao	48
5.3.1. Montaje y publicación de los sensores	48
5.3.2. Control de la marcha para mapeo	48
5.3.3. Lanzamiento de los nodos y configuración	48
5.3.4. Creación del mapa del entorno	49
5.3.5. Visualización y validación en RViz	50
5.4. Navegación autónoma	51
5.5. Resumen y próximos pasos	52
6. Conclusiones	53
6.1. Conclusiones	53
6.2. Competencias adquiridas	55
6.3. Trabajos futuros	57
Bibliografía	59

Índice de figuras

2.1.	Robot Nao V6	8
2.2.	RPlidar S2	8
2.3.	Portátil MSI GP66 Leopard UE	9
2.4.	Árbol TF del Nao	15
2.5.	Integración del simulador Webots y visualización en RViz	22
3.1.	Prueba de odometría satisfactoria con el Nao caminando	34
4.1.	Conversación real con el Nao	38
4.2.	Flujo general de interacción HRI en el Nao	40
4.3.	Diagrama de estados del nodo ModeSwitcher	44
5.1.	SLAM con el robot Nao teleoperado en un PC	50

Listado de códigos

2.1.	Launch file de robot_state_publisher en ROS 2	14
2.2.	Nodo Python para conversión de imagen de la cámara	20
2.3.	Parámetros de calibración intrínseca de la cámara	20
3.1.	Fragmento de cálculo de odometría en Python	32
4.1.	Ejemplo de archivo de postura .pos para el Nao	37
4.2.	Ejecución de gesto con nao_pos desde terminal	37
4.3.	Llamada de servicio a GPT desde ROS 2	39
5.1.	Lanzamiento de mode_switcher en el Nao	48
5.2.	Lanzamiento de nodos para navegación y visualización	49
5.3.	Lanzamiento de nodos SLAM y Nav2 en distrobox	49

Listado de ecuaciones

2.1.	Actualización de la orientación con cuaterniones	16
2.2.	Función de error para la gravedad	17
2.3.	Actualización de cuaterniones en el filtro	18
3.1.	Incrementos de posición global en odometría bípeda	31

Índice de tablas

1.1. Principales generaciones del robot Nao.	3
--	---

Capítulo 1

Introducción

“Surely, if we take on thinking partners—or, at the least, thinking servants—in the form of machines, we will be more comfortable with them, and will relate to them more easily, if they are shaped like humans. It will be easier to be friends with human-shaped robots than with specialized machines of unrecognizable shape. And I sometimes think that, in the desperate straits of humanity today, we would be grateful to have nonhuman friends, even if they are only the friends we build ourselves.”

Isaac Asimov, *Robot Visions* [3]

1.1. Contexto tecnológico

En las últimas décadas, el desarrollo de software para sistemas robóticos ha experimentado una transformación significativa gracias a la adopción de plataformas de código abierto. Entre dichas plataformas destaca *Robot Operating System* (ROS) [4], introducida en 2007, que se ha consolidado como el entorno de referencia para la creación de aplicaciones robóticas de código abierto a nivel mundial. ROS proporciona un conjunto unificado de herramientas y bibliotecas que facilitó por primera vez que investigadores e ingenieros colaboraran sobre una base común, evitando la hermeticidad y dispersión de las soluciones propietarias de cada fabricante.

La evolución natural de este ecosistema dio lugar a ROS 2, la segunda generación del sistema operativo robótico, concebida específicamente para responder a los nuevos retos y exigencias del sector moderno. ROS 2 fue desarrollada para superar ciertas limitaciones de *Robot Operating System 1* (ROS 1), incorporando mejoras fundamentales en diversos aspectos como la comunicación entre componentes, la compatibilidad con múltiples sistemas operativos y el rendimiento en tiempo real.

Gracias a estas capacidades, ROS 2 se ha convertido en una herramienta clave en el desarrollo de robots actuales, que demandan plataformas más escalables, fiables y preparadas para aplicaciones industriales y de investigación de última generación.

Un aspecto crucial de ROS 2 es su naturaleza abierta (*open source*). A diferencia del software propietario, que impone costes de licencia y mantiene el código fuente cerrado, ROS 2 se distribuye bajo una licencia de código abierto permisiva (*Apache 2.0*). Esto significa que los usuarios tienen acceso completo al código fuente del sistema y la libertad de adaptarlo a sus necesidades específicas. La posibilidad de inspeccionar y modificar el código proporciona una flexibilidad enorme para personalizar el comportamiento de los robots, algo difícilmente alcanzable en entornos cerrados cuyas funciones de adaptación suelen ser muy limitadas a lo previsto por el fabricante. Asimismo, al basarse en estándares abiertos ampliamente utilizados (por ejemplo, el middleware *Data Distribution Service* (DDS) para comunicaciones) y contar con el respaldo de una extensa comunidad internacional, ROS 2 facilita la interoperabilidad entre distintos componentes y promueve la innovación colaborativa en robótica.

La adopción de ROS 2 cobra especial relevancia al considerar plataformas robóticas cuyo software original es propietario. Un caso emblemático es el robot humanoide Nao, creado por Aldebaran Robotics (hoy parte de SoftBank Robotics), el cual funciona con un sistema cerrado denominado *NAOqi*. Este entorno obliga a los desarrolladores a interactuar con el robot exclusivamente a través de la *Application Programming Interface* (API) y las herramientas oficiales del fabricante. En consecuencia, las posibilidades de extender o modificar las capacidades del Nao quedan limitadas a lo que dicha API soporte, dificultando su integración con componentes externos no contemplados originalmente. Por ejemplo, se ha documentado que la capacidad de personalizar funciones de diálogo o gestionar estados de error en el Nao es muy limitada bajo su software predeterminado, lo que ha restringido ciertos experimentos de interacción humano-robot en entornos de investigación [5]. En general, en arquitecturas cerradas como *NAOqi* cualquier función nueva depende de actualizaciones oficiales, ya que el usuario carece de medios para alterar el código base del sistema por sí mismo.

Frente a estas limitaciones, surge la motivación de migrar plataformas como Nao hacia entornos abiertos basados en ROS 2. Al adoptar ROS 2 de forma nativa en el robot, se independiza la plataforma de las restricciones impuestas por el software propietario del fabricante, permitiendo acceder directamente al hardware con las

capacidades ampliadas que ofrece ROS 2. Esto abre la puerta a integrar el robot en un ecosistema estándar donde puede comunicarse con otros dispositivos, incorporar funcionalidades avanzadas (desde algoritmos de visión hasta módulos de inteligencia artificial) y beneficiarse de las mejoras continuas aportadas por la comunidad, sin depender exclusivamente del proveedor.

En definitiva, la transición de sistemas cerrados a ROS 2 representa un paso natural para potenciar el desarrollo en robótica y constituye la motivación central del presente Trabajo de Fin de Grado.

1.2. El robot Nao

Desde su concepción en *Project Nao* (2004) [6] y la fundación de Aldebaran Robotics en 2005, el Nao se ha convertido en uno de los robots humanoides más difundidos en ámbitos educativos y de investigación. Su primera aparición pública fue como plataforma oficial de la *Standard Platform League* (SPL) de la RoboCup [7] en 2007, cuando sustituyó al célebre perro robótico *Aibo* [8]; un año después se entregaron las primeras unidades a los equipos participantes [9]. Desde la Academics Edition (V3, 2008), el robot Nao ha evolucionado a través de diversas generaciones: V3.2 (2009), V3.3 (2010), Nao Next Gen (V4, 2011), Nao Evolution (V5, 2014) y Nao V6 (Power 6, 2018)[10], y hoy existen más de 13 000 unidades desplegadas en más de 70 países, con presencia en más de 600 universidades, laboratorios y centros educativos.

Modelo	Año de lanzamiento
Nao Academics Edition (V3/V3+)	2008/2009
Nao V3.2	2009
Nao V3.3	2010
Nao Next Gen (V4)	2011
Nao Evolution (V5)	2014
Nao V6 (Power 6)	2018

Tabla 1.1: Principales generaciones del robot Nao.

Principales ámbitos de uso

- **Investigación y docencia** *Science, Technology, Engineering and Mathematics (STEM)*. El Nao se utiliza como plataforma estándar en muchas asignaturas de robótica, programación e inteligencia artificial. Su cuerpo compacto, la facilidad para programarlo en lenguajes Python/C++ y

la abundancia de recursos didácticos han permitido crear cursos universitarios y actividades de enseñanza secundaria orientadas a estimular el interés por la robótica en personas jóvenes.

- **Competición RoboCup [7].** En la SPL, todos los equipos compiten con robots Nao iguales, lo que permite centrar la innovación en algoritmos de visión artificial, localización y cooperación multi-robot [11]. Esta competición ha ayudado a realizar avances en percepción, planificación de tareas y marcha bípeda en robots, que después ha sido posible trasladar a entornos reales.
- **Interacción humano-robot (HRI).** Gracias a su tamaño no intimidante, expresividad (*Light Emitting Diode (LED)*s, gestos, voz) y diferentes sensores, el Nao se ha convertido en un referente para estudiar la interacción social con robots. Más de una década de trabajos académicos analiza cómo su apariencia y “personalidad” influye en la aceptación por parte de niños y adultos [12].
- **Terapia y salud.** Diversos estudios clínicos han demostrado que el Nao mejora habilidades sociales e imitación en niños con trastorno del espectro autista y reduce la ansiedad en sesiones de rehabilitación [13]. Investigaciones recientes exploran su uso como asistente cognitivo con personas mayores [14].
- **Atención al público.** Entidades bancarias japonesas han empleado al Nao como recepcionista bilingüe para atender a los clientes [15]. Asimismo, hoteles y museos lo han utilizado como guía interactivo, aprovechando su capacidad para reconocer rostros y conversar brevemente [16].

Evolución tecnológica

Cada generación del robot Nao ha traído mejoras tanto en la parte mecánica como en la potencia de procesamiento. El modelo actual del Nao cuenta con un diseño más avanzado, con mejores recursos para moverse y percibir su entorno, adaptándose a las especificaciones necesarias modernas. Más adelante, en la Sección 2.2, se describe el hardware en detalle. Sin embargo, la potencia que tiene para tareas modernas como la visión artificial no es muy alta, así que la comunidad ha buscado formas de ampliar lo que puede hacer usando conexiones *Wireless Fidelity (Wi-Fi)* y ROS 2, procesando parte del trabajo en un *Personal Computer (PC)* externo.

Además, se han creado simulaciones realistas, como es el caso de Webots [17]. Gracias a ellas, es posible probar y mejorar el sistema sin depender siempre del robot

real, lo que hace que el desarrollo sea más rápido y flexible. Así, se pueden experimentar nuevas ideas y funcionalidades de forma segura antes de llevarlas al robot físico.

Situación actual y retos

SoftBank Robotics anunció en 2024 haber superado las 20 000 unidades de Nao y 17 000 de *Pepper*, pudiendo considerarse al Nao como estándar educativo en su campo [18]. Sin embargo, su software propietario original (*NAOqi*) permanece cerrado, lo que limita la integración con herramientas modernas. Esto ha impulsado iniciativas comunitarias —incluida la presente migración a ROS 2— para dotar al robot de un sistema abierto, actualizar sus algoritmos y garantizar su relevancia en los próximos años. Con ello, el Nao continúa sirviendo como herramienta de investigación académica en aplicaciones reales, desde la terapia asistida por robots hasta la educación STEM global.

1.3. Objetivos

El propósito general de este Trabajo de Fin de Grado es transformar el robot Nao en una plataforma abierta plenamente funcional basada en ROS 2, ampliando sus capacidades en educación e investigación y reduciendo la dependencia de software propietario. Para ello, se plantean los objetivos específicos que se enumeran a continuación, descritos de forma genérica y orientados a resultados medibles:

1. **Migración del sistema operativo.** Sustituir el sistema propietario *NAOqi* por una distribución *GNU/Linux* actual (*Ubuntu 22.04*), asegurando la compatibilidad total con el hardware y el firmware de los sensores y actuadores.
2. **Plataforma base sobre ROS 2.** Disponer de un conjunto mínimo de nodos que permitan sensar y actuar de forma estandarizada: publicación de `tf`, estimación de odometría adecuada a un robot bípedo, acceso a cámaras, *Inertial Measurement Unit* (IMU), micrófonos y LEDs, y envío de comandos a articulaciones.
3. **Capacidades avanzadas de locomoción y gestos.** Dotar al robot de la capacidad de caminar de manera configurable, así como la habilidad de realizar gestos fluidos que permitan una expresión corporal más natural durante la interacción.

4. **Interacción humano-robot más completa.** Integrar visión artificial para seguimiento de rostros, reconocimiento de voz, generación de lenguaje natural y síntesis de voz, combinados para dar la capacidad al robot de tener conversaciones coherentes con humanos en diferentes contextos.
5. **Navegación autónoma.** Dotar al Nao de la capacidad de planificar y recorrer trayectos en entornos conocidos o parcialmente mapeados, utilizando fuentes de percepción adicionales (por ejemplo, un LiDAR ligero) y garantizando la seguridad de la marcha bípeda durante el desplazamiento.
6. **Simulación fiel al robot real.** Emplear un simulador que replique conrealismo la cinemática, sensores y control de bajo nivel del Nao, de modo que los mismos nodos de ROS 2 funcionen sin modificaciones tanto en un entorno virtual como en el hardware físico.
7. **Documentación reproducible.** Entre los objetivos de este trabajo destaca la creación de un repositorio público y reproducible que sirva como guía de referencia para la migración del robot Nao a ROS 2, accesible para la comunidad investigadora internacional [2]. Más concretamente: elaborar guías claras y concisas que permitan a otros investigadores replicar todo el proceso de migración, instalación de paquetes y puesta en marcha del sistema, reduciendo la actual dispersión de la información.

Capítulo 2

Integración de Nao en ROS 2

Given enough eyeballs, all bugs are shallow.

Linus Torvalds, *The Cathedral and the Bazaar* [19]

2.1. Introducción

El objetivo principal de este capítulo es documentar el proceso de integración del robot Nao con el middleware robótico ROS 2, estableciendo una base sobre la que puedan desarrollarse después funcionalidades más complejas [20, 2]. Esta integración implica reemplazar el sistema operativo propietario del robot por una distribución *Linux* moderna, junto con un conjunto mínimo de paquetes base que expongan su hardware a ROS 2. Además, se añade la posibilidad de trabajar con un simulador, sin requerir el acceso constante al robot real. A lo largo de esta sección se detallan los pasos realizados y se ofrece una base replicable para otros investigadores.

2.2. Hardware empleado

A continuación, se detallan los elementos físicos utilizados para la implementación y prueba del sistema desarrollado:

- **Robot:** Se ha utilizado un robot *Nao V6*, que está equipado con los siguientes sensores: dos cámaras *Complementary Metal-Oxide-Semiconductor* (CMOS) (superior e inferior), cuatro micrófonos omnidireccionales, tres sensores táctiles ubicados en la cabeza (frontal, central y trasero), sensores capacitivos en las manos, una IMU que incluye acelerómetro, giroscopio y magnetómetro, ocho *Force Sensitive Resistors* (FSR) en las plantas de los pies (cuatro en cada pie), codificadores (encoders) en cada una de sus 25 articulaciones, un sensor interno de temperatura y dos sensores ultrasónicos en el pecho.

En cuanto a actuadores, el Nao dispone de: 25 articulaciones controladas mediante servomotores, LEDs *Red Green Blue* (RGB) ubicados en ojos, orejas, pecho y pies, y altavoces integrados para la reproducción de audio.

El hardware del ordenador integrado en el robot *Nao V6* consiste en una *Central Processing Unit* (CPU) Intel Atom E3845 de 1,91 *Gigahertz* (GHz), 4 *Gigabyte* (GB) de memoria *Random Access Memory* (RAM), 32 GB de almacenamiento interno tipo flash, conectividad Ethernet y Wi-Fi, y un *Universal Serial Bus* (USB) [21].



Figura 2.1: Robot Nao V6

- **Sensores adicionales:** Para tareas de localización, se ha añadido un sensor LiDAR RPlidar S2 en la parte superior de la cabeza del Nao, conectado por cable al USB de un PC externo.



Figura 2.2: RPlidar S2

- **Plataforma de cómputo:** Aparte del ordenador interno del Nao, en el que se ejecutan la mayoría de los procesos, se ha utilizado un ordenador portátil MSI GP66 Leopard UE, con CPU Intel i7-10870H, 16GB de RAM y una NVIDIA GeForce *Ray Tracing Texel eXtreme* (RTX) 3060 Laptop *Graphics Processing Unit* (GPU), conectado a la misma red Wi-Fi que el Nao, con el objetivo de ejecutar tareas con exigencia de cómputo como visión artificial y navegación.



Figura 2.3: Portátil MSI GP66 Leopard UE

- **Entorno de pruebas:** Las pruebas reales se han realizado en un entorno controlado dentro del Laboratorio de Robótica y Sistemas Ubícuos de la escuela de Ingeniería de Fuenlabrada, en el campus de la Universidad Rey Juan Carlos de Fuenlabrada, delimitado y acondicionado para experimentos con robots autónomos.

2.3. Software y herramientas

En cuanto al software, se ha configurado un entorno completo para el desarrollo, simulación, control y análisis del sistema propuesto:

- **Sistema operativo:** *Ubuntu 22.04 LTS*, versión estable y ampliamente soportada por las principales herramientas de desarrollo en robótica, instalado tanto en el ordenador portátil como en el Nao, en cuyo caso se accede por *Secure Shell* (SSH) al no disponer de interfaz gráfica [22].
- **Middleware robótico:** *ROS 2 Rolling*, una distribución de desarrollo continuo de ROS 2. Esta distribución incorpora las últimas novedades y mejoras del

ecosistema, por lo que los paquetes disponibles suelen estar más actualizados y avanzados en comparación con las versiones estables [23, 20]. Este sistema ofrece soporte para arquitecturas distribuidas, comunicaciones robustas en tiempo real y mayor seguridad.

- **Simulador:** *Webots*, empleado para probar algoritmos de percepción y actuación en un entorno virtual antes de ejecutarlos en el mundo real [17]. Esta configuración permite lanzar el nodo `nao_lola_client` directamente en el PC, simulando la comunicación con el middleware *Low Level Abstraction* (LoLA) del robot Nao, lo que facilita el desarrollo y la validación de comportamientos sin necesidad de disponer del hardware físico [24].
- **Lenguajes de programación:** El desarrollo del software se ha realizado principalmente en *C++* y *Python*, los dos lenguajes oficialmente soportados en ROS 2 y ampliamente utilizados por la comunidad robótica internacional.
- **Librerías y paquetes utilizados:**

- `nao_lola`, interfaz ROS 2 que expone la funcionalidad de LoLA para los sensores y actuadores del Nao a 83 *Hertz* (Hz) [25, 26].
- `nao_led`, control avanzado de los LED RGB de ojos, orejas, pecho y pies [27].
- `nao_pos`, ejecución de gestos y posturas almacenadas en archivos `.pos` [28].
- `walk`, controlador de marcha bípeda para el Nao en ROS 2 [29, 30].
- `nao_ik` y `nao_phase_provider`, cinemática inversa y detección de fases de paso para caminar con estabilidad [31, 32].
- `usb_cam`, publicación de las imágenes de las cámaras del Nao vía *Video for Linux 2* (V4L2) en un topic de ROS 2 [33].
- `OpenCV 4.7.0`, para visión artificial (procesado de imágenes RGB de las cámaras del Nao).
- `audio_common`, captura y reproducción de audio de los micrófonos y altavoces integrados [34].
- `nao_hri`, para interacción humano-robot HRI (conversaciones, gestos, LEDs, seguimiento de rostros) [35].
- `YOLOv8 object_tracker`, detección y seguimiento de rostros/objetos en tiempo real [36].

- `chat_service`, integración con OpenAI API para diálogo natural utilizando modelos *Large Language Model* (LLM).
- `gstt_service` y `gtts_service`, puente con OpenAI *Speech To Text* (STT) y Google Cloud *Text To Speech* (TTS).
- `sync`, script para copiar al Nao un workspace ROS 2 precompilado desde el PC.
- `robot_state_publisher + nao_description`, publicación de la cinemática y los frames (*Transform Frame* (TF)) del Nao [2, 37, 38, 39].
- `imu_filter_madgwick`, filtrado y fusión de los datos de la IMU para estimar la orientación espacial del Nao [40].
- `nao_ros2`, paquete propio que coordina cálculo de odometría, locomoción y HRI con seguridad, además de añadir algunas funcionalidades nuevas [2].
- *Point Cloud Library* (PCL), para el tratamiento de nubes de puntos generadas con cámaras de profundidad externas.
- `pointcloud_to_laserscan`, conversión de nubes *Three-Dimensional* (3-D) a escaneos *Two-Dimensional* (2-D) para navegación (prueba inicial para simular un láser con una cámara de profundidad, descartada por la decisión final de utilizar un láser real).
- `sllidar_ros2`, integración de sensores LiDAR en ROS 2 para medir distancias y detectar obstáculos en el entorno del robot mediante datos 2-D.
- Nav2, para la navegación autónoma [41].
- SLAM Toolbox, para la creación y actualización de mapas 2-D en tiempo real [42].
- Webots LoLA Controller, controlador de simulación en Webots [17] que emula LoLA para utilizar `nao_lola_client` en el PC [24].
- **Entorno de desarrollo:** *Visual Studio Code* (VS Code), utilizado para la edición, depuración y gestión de los proyectos de software.
- **Control de versiones:** Git y GitHub para gestionar el código fuente y realizar seguimiento del progreso.

2.3.1. Instalación del sistema operativo

La instalación de *Ubuntu* en el robot se realiza generando una imagen personalizada con el repositorio **NaoImage** [22]. Esta herramienta, mantenida por el equipo NaoDevils,

permite crear imágenes compatibles con el hardware del Nao V6 y que incluyen soporte para dispositivos esenciales como las cámaras y la red. El proceso general es:

1. Descargar o clonar el repositorio `NaoImage` [22].
2. Configurar los parámetros de compilación para generar una imagen basada en *Ubuntu 22.04* a partir de la imagen oficial de *NAOqi*
3. Asegurarse de flashear primero una versión de *NAOqi* específica, para evitar incompatibilidades con el firmware de los motores.
4. Utilizar el flasher oficial para flashear la nueva imagen *Linux* en el robot.

Una vez completado el proceso, el robot inicia directamente en *Ubuntu* y ha de ser preparado para instalar y ejecutar nodos de ROS 2.

Para una explicación más detallada sobre la configuración, instalación de dependencias y puesta en marcha de los nodos en ROS 2, se recomienda consultar la documentación incluida en el paquete `nao_ros2` [2], donde se describen paso a paso todos los procedimientos necesarios para dejar el robot operativo en este entorno.

2.4. Base funcional del robot Nao en ROS 2

En esta sección se explica cómo se ha dejado el Nao listo para ser utilizado en ROS 2 y cómo se ha calculado y configurado cada parte para que funcione correctamente.

2.4.1. Nodo `nao_lola`: interfaz con LoLA

Para integrar el hardware del Nao con ROS 2, se ha utilizado el paquete `nao_lola`, desarrollado por la organización *ROS Sports* [26]. Este nodo actúa como adaptador entre el middleware interno del robot y ROS 2, publicando los datos de sensores y suscribiéndose a los comandos para los actuadores en topics estandarizados del middleware robótico.

Funcionalidad principal

- El nodo `nao_lola_client` lee los datos de los sensores del robot, proporcionados por LoLA, el middleware de bajo nivel de Aldebaran que se ejecuta en el robot Nao.

- Publica los estados de las articulaciones y datos de sensores —como la IMU, FSR, sonar, botones y más— en topics de ROS 2, a una frecuencia típica de 83 Hz.
- También se suscribe a topics destinados a actuadores, permitiendo enviar comandos a los motores de las articulaciones o cambiar el estado de los LEDs del robot.

En resumen, este nodo es esencial para comunicarse con el middleware LoLA y permite que el sistema ROS 2 tenga acceso directo a los sensores y actuadores del robot Nao.

2.4.2. Publicación de transformaciones (TF)

Las TF (marcos de referencia en ROS) permiten conocer en todo momento la posición y orientación relativa de cada parte del robot y de sus sensores en el espacio, lo que es fundamental para coordinar la percepción y actuación de un robot.

La estructura del árbol de transformaciones del robot Nao se genera automáticamente a partir de su modelo *Unified Robot Description Format* (URDF) y el estado de sus articulaciones [2]. El árbol TF se construye desde el frame raíz `base_link`, al que se unen los frames de torso, cabeza, cámaras, brazos, piernas y sensores como la IMU y los FSR de los pies.

Los nodos implicados en este proceso son:

- `nao_lola_client`: lanzado en el robot, publica el estado actual de las articulaciones del Nao en el topic `joint_states`, proporcionando las posiciones reales necesarias para actualizar el árbol TF [25, 26].
- `robot_state_publisher`: publica las transformaciones fijas y dinámicas de los links del robot, a partir del topic `joint_states`, usando el URDF [43].

Gracias a ello, ROS 2 conoce en todo momento la posición y orientación tridimensional de cada sensor y actuador del robot.

A continuación se muestra el launch file que carga el URDF y lanza el nodo `robot_state_publisher`:

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    urdf_file = os.path.join(
        get_package_share_directory('nao_description'),
        'urdf', 'nao.urdf')
    with open(urdf_file, 'r') as infp:
        robot_desc = infp.read()

    return LaunchDescription([
        Node(
            package='robot_state_publisher',
            executable='robot_state_publisher',
            name='robot_state_publisher',
            output='screen',
            parameters=[{'robot_description': robot_desc}]
        )
    ])

```

Código 2.1: Launch file de `robot_state_publisher` en ROS 2

Así, se carga el archivo URDF desde el paquete `nao_description` y se pasa como parámetro al nodo `robot_state_publisher`. Este nodo se encarga de suscribirse al topic `joint_states`, calcular las actualizaciones del árbol TF y publicarlas en los topics `/tf` y `/tf_static` [44].

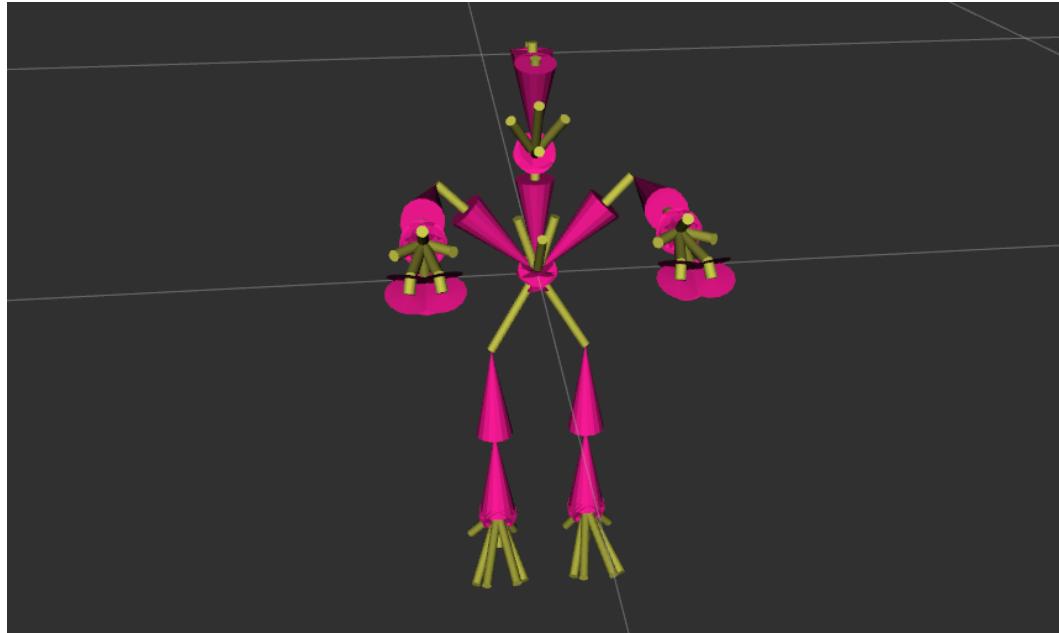


Figura 2.4: Visualización del árbol de transformaciones (TF) del robot Nao en *RViz* (solo flechas para claridad)

2.4.3. Odometría: estimación de orientación y posición del Nao

Uno de los problemas más recurrentes en robótica, especialmente en plataformas como el robot Nao, es conocer con precisión su posición y orientación respecto al entorno. La estimación de la odometría es fundamental para dotar al robot de capacidades de localización y navegación autónoma.

La fusión sensorial tiene como principal objetivo reducir y corregir los errores individuales y acumulativos que pueden presentar los sensores, permitiendo así calcular la posición y orientación de forma más precisa y robusta. En robots móviles con ruedas, la odometría se estima habitualmente a partir de los encoders de las ruedas, proporcionando generalmente buenos resultados. Sin embargo, este método puede verse afectado por factores como la variación del diámetro de las ruedas (debido a ruedas deshinchadas, mala calibración o desgaste), o el deslizamiento, lo que genera errores que se acumulan con el tiempo. Por este motivo, se suelen emplear algoritmos de fusión adicionales que integran otros sensores, como LiDAR o balizas visuales, que permiten realizar correcciones absolutas de la posición, ya que proporcionan una referencia externa y precisa sobre la ubicación real del robot, eliminando la deriva acumulada en la estimación.

En el caso de los robots bípedos como el Nao, la ausencia de ruedas añade una complicación extra en el cálculo de la odometría, ya que no se dispone de encoders de rueda. En este trabajo se propone una técnica de estimación de la odometría adaptada a este tipo de plataformas. Para ello, se emplea la IMU integrada en el Nao, que proporciona datos de aceleraciones lineales y velocidades angulares, para calcular la orientación mediante la aplicación de un filtro de Madgwick. Además, se utilizan los desplazamientos lineales calculados por el paquete `walk`, que aplica técnicas de cinemática inversa para estimar los pasos y desplazamientos del robot durante la marcha. Después, toda esta información sensorial se fusiona mediante un filtro de Kalman, logrando así una odometría precisa y robusta.

Esta estimación de la odometría, en combinación con la información proporcionada por un sensor LiDAR y el algoritmo *Adaptive Monte-Carlo Localizer* (AMCL) integrado en `Nav2`, permite determinar con precisión la posición del robot en el entorno, eliminando la deriva en la estimación.

¿Qué es un filtro de Madgwick?

El filtro de Madgwick es un algoritmo de fusión sensorial desarrollado para estimar la orientación tridimensional de un objeto utilizando los datos de una unidad inercial (IMU), que generalmente incluye acelerómetros y giroscopios, y opcionalmente magnetómetros. La principal ventaja de este filtro se encuentra en su capacidad para calcular estimaciones precisas de la orientación con una baja carga computacional, que lo hace especialmente adecuado para aplicaciones en robótica y sistemas embebidos.

Desde un punto de vista técnico, el filtro de Madgwick se basa en la representación de la orientación mediante cuaterniones, $\mathbf{q} = [q_0, q_1, q_2, q_3]$, evitando así los problemas de singularidad de otras representaciones como los ángulos de Euler. La actualización de la orientación se realiza integrando las velocidades angulares medidas por el giroscopio, según la siguiente ecuación diferencial:

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \boldsymbol{\omega} \quad (2.1)$$

Ecuación 2.1: Actualización de la orientación del robot usando cuaterniones y velocidades angulares.

donde:

- \mathbf{q} es el cuaternion que representa la orientación actual del robot,
- ω es el cuaternion puro formado a partir de la velocidad angular medida por el giroscopio, es decir, $\omega = [0, \omega_x, \omega_y, \omega_z]$ donde $(\omega_x, \omega_y, \omega_z)$ son las componentes de la velocidad angular,
- \otimes denota el producto de cuaterniones,
- $\dot{\mathbf{q}}$ es la derivada temporal del cuaternion de orientación.

El filtro de Madgwick combina esta integración con una corrección periódica basada en el error entre la dirección estimada y la medida real del vector de gravedad (obtenido del acelerómetro) y, si se utiliza, del campo magnético (magnetómetro). El objetivo es minimizar la función de error:

$$\mathbf{f}(\mathbf{q}, \mathbf{a}) = \mathbf{q}^* \otimes \mathbf{a} \otimes \mathbf{q} - \mathbf{g} \quad (2.2)$$

Ecuación 2.2: Función de error del filtro de Madgwick para la alineación con el vector gravedad.

donde:

- \mathbf{q} es el cuaternion que representa la orientación actual del robot,
- \mathbf{q}^* es el conjugado del cuaternion \mathbf{q} ,
- \mathbf{a} es el vector de aceleración medido por el acelerómetro, representado como un cuaternion puro (con parte escalar cero), es decir, $\mathbf{a} = [0, a_x, a_y, a_z]$,
- \mathbf{g} es el vector de gravedad esperado en el marco de referencia global, normalmente $\mathbf{g} = [0, 0, 0, g]$ con $g \approx 9,81 \text{ m/s}^2$,
- \otimes denota el producto de cuaterniones.

La corrección de la orientación se realiza mediante un método de gradiente descendente, ajustando la estimación del cuaternion según la dirección de máximo descenso del error, controlado por un parámetro β que determina la magnitud de la corrección:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + (\dot{\mathbf{q}} - \beta \nabla_{\mathbf{q}} f) \Delta t \quad (2.3)$$

Ecuación 2.3: Ecuación de actualización de cuaterniones con corrección por gradiente descendente.

donde:

- \mathbf{q}_k es el cuaternion de orientación en el instante k ,
- $\dot{\mathbf{q}}$ es la derivada del cuaternion (2.1),
- β es el parámetro de convergencia del filtro,
- $\nabla_{\mathbf{q}} f$ es el gradiente de la función de error f (2.2) respecto al cuaternion,
- Δt es el intervalo de tiempo de integración.

Este mecanismo permite que el filtro corrija automáticamente la deriva (*drift*) acumulada por el giroscopio, utilizando la dirección de la gravedad como referencia absoluta, incluso cuando no se dispone de magnetómetro o este resulta poco fiable debido a interferencias.

Implementación en ROS 2 para el robot Nao

En el contexto de este trabajo, el filtro de Madgwick se ha implementado mediante el paquete `imu_filter_madgwick` [40] en ROS 2 para estimar la orientación del robot Nao. Se ha configurado el filtro para utilizar únicamente los datos de acelerómetro y giroscopio.

La principal ventaja de esta implementación, como se ha explicado previamente, es la utilización del vector de gravedad, obtenido a partir de las mediciones del acelerómetro, como referencia absoluta para corregir la orientación estimada. Gracias a ello, el filtro puede compensar de manera continua la deriva acumulada por el giroscopio, asegurando que la orientación calculada sea fiel a la realidad en todo momento. Finalmente, el filtro también es capaz de identificar y eliminar el componente de la gravedad de las mediciones del acelerómetro (utilizando la propia orientación calculada), permitiendo así obtener las aceleraciones lineales puras debidas exclusivamente a los movimientos reales del robot, y evitando la contaminación de esos

datos por la aceleración de la gravedad.

La orientación calculada se publica en el marco de referencia estándar *East-North-Up* (ENU), lo que facilita la integración con otros sistemas de percepción y navegación dentro del entorno ROS 2. Durante las pruebas, el comportamiento del filtro resultó bastante realista y reactivo al visualizar la orientación en **RViz** [45]: los cambios se reflejaban de forma rápida y precisa, y la estimación era estable incluso ante pequeños movimientos. Además, se podía observar cómo, en ocasiones, sin mover el robot, la orientación se corregía ligeramente aprovechando el efecto de la gravedad, eliminando la deriva acumulada por el giroscopio. Esta implementación permite obtener una estimación de orientación robusta y en tiempo real, fundamental para el control de movimientos del robot y la navegación autónoma.

La estimación de la posición en la odometría se explica en el siguiente capítulo, dedicado al paquete **walk**, donde se describe cómo se utiliza la orientación obtenida con el filtro de Madgwick para calcular la localización completa del robot.

2.4.4. Conversión y calibración de la cámara del Nao

Una de las primeras dificultades que se encontró al trabajar con la cámara del Nao fue que la imagen se publica en un formato poco común para los estándares de ROS 2. Concretamente, el formato que utiliza el robot es **YUV422**, que no es directamente compatible con muchos nodos de procesamiento de imagen ni con la mayoría de visualizadores. Además, la imagen original está referenciada a un frame que no está correctamente registrado y está invertida 180 grados.

Para trabajar con la cámara en ROS 2 se utiliza el paquete **usb_cam**, que publica la imagen cruda en un topic, pero además ha sido necesario crear un nodo extra para realizar la conversión del formato y corregir la imagen.

Básicamente, lo que hace el nodo es:

1. Suscribirse al topic original (`/image_raw`) publicado por **usb_cam**.
2. Convertir la imagen de **YUV422** a **RGB** usando *OpenCV Python interface* (`cv2`) [46], asegurándose de que las dimensiones sean las correctas.
3. Invertir la imagen (originalmente está boca abajo, probablemente por cómo el driver interpreta el sensor).

4. Publicar la imagen final en un nuevo topic (`/image_rgb`), asociándola al frame correcto de la cámara (`CameraTop_frame`) y añadiendo la información de calibración con los parámetros adecuados.

He aquí un fragmento del nodo en lenguaje *Python*:

```
def image_callback(self, msg):
    # Convert from YUV422 to RGB
    yuv_image = self.bridge.imgmsg_to_cv2(msg)
    img = yuv_image.reshape(480, 640, 2)
    rgb_image = cv2.cvtColor(img, cv2.COLOR_YUV2BGR_YUYV)
    rgb_image = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2RGB)

    # Flip the image vertically (the original was upside down)
    rgb_image = cv2.rotate(rgb_image, cv2.ROTATE_180)

    # Publish the result
    rgb_msg = self.bridge.cv2_to_imgmsg(rgb_image, encoding='rgb8')
    rgb_msg.header.frame_id = 'CameraTop_frame'
    self.image_pub.publish(rgb_msg)
```

Código 2.2: Fragmento del nodo Python para convertir la imagen de la cámara del Nao de YUV422 a RGB y corregir la orientación.

Otra parte importante es la calibración de la cámara. Para poder usar visión artificial y obtener resultados realistas con ROS 2, es necesario tener los parámetros de la cámara bien definidos (sobre todo los parámetros intrínsecos, como la matriz de calibración y la distorsión). Para calibrar la cámara se utilizó el método clásico del tablero de ajedrez, haciendo varias capturas desde diferentes ángulos y utilizando la herramienta de calibración de *Open Source Computer Vision Library* (OpenCV) [47]. Los parámetros que se obtienen se incluyen en el mensaje `CameraInfo` y se publican junto con la imagen para que cualquier nodo suscriptor pueda utilizarlos directamente.

```
self.camera_info.width = 640
self.camera_info.height = 480
self.camera_info.k = [544.303, 0.0, 319.5,
                     0.0, 544.303, 239.5,
                     0.0, 0.0, 1.0]
self.camera_info.d = [0.0607252, -0.253295, 0.0, 0.0, 0.0]
```

Código 2.3: Ejemplo de parámetros de calibración intrínseca de la cámara del robot Nao para ROS 2.

En este contexto, la calibración intrínseca de la cámara (matriz de calibración y parámetros de distorsión) se complementa con la información extrínseca, es decir, la

posición y orientación de la cámara respecto al robot, la cual ya ha sido definida y publicada en el árbol de transformaciones (TF) descrito previamente en la Sección 2.4.2. De este modo, la cámara queda correctamente integrada en el sistema y puede utilizarse directamente para tareas de percepción y procesamiento de imágenes en ROS 2.

2.4.5. Simulación con Webots y publicación de cámara

Como parte fundamental del entorno de desarrollo, se incluye un simulador que permite ejecutar y probar los nodos de ROS 2 sin necesidad de disponer del robot físico. Para ello se utiliza el simulador *Webots* [17], junto con un controlador que emula la interfaz LoLA del Nao. Dicho controlador se incluye en el paquete `WebotsLoLaController`, desarrollado por el equipo Bembelbots [24].

Este controlador simula tanto la cinemática del robot como sus sensores (IMU, tacto, cámaras, etc.) y gestiona los actuadores en tiempo real, reproduciendo de manera bastante realista el comportamiento del proceso LoLA del hardware físico en el simulador.

Mejoras introducidas en el controlador

El controlador original no publicaba las imágenes de las cámaras en topics de ROS 2; por ello, se creó un fork con una rama que extiende esta funcionalidad, permitiendo que desde *Webots*:

- Las imágenes de las cámaras (superior e inferior) se conviertan a un formato compatible con ROS 2.
- Se publiquen en un topic estándar (`/image_rgb`) junto con el correspondiente mensaje `CameraInfo` asociado al frame de la cámara del árbol de TF.

Esto permite que los nodos de visión artificial funcionen indistintamente en el simulador o en el robot real.

Ventajas de usar un simulador durante el desarrollo

Gracias a esta integración, donde el nodo `nao_lola_client` se lanza en el PC (en lugar de en el robot real) y se conecta directamente al controlador del simulador, se consiguen varias ventajas clave:

- Se pueden validar y depurar algoritmos desde un entorno seguro, evitando dañar el hardware real.
- El ciclo de desarrollo es más rápido, ya que no se depende del acceso físico al robot.

La simulación en *RViz* y *Webots* resulta realista, con un tiempo de respuesta y precisión muy similares al robot real.

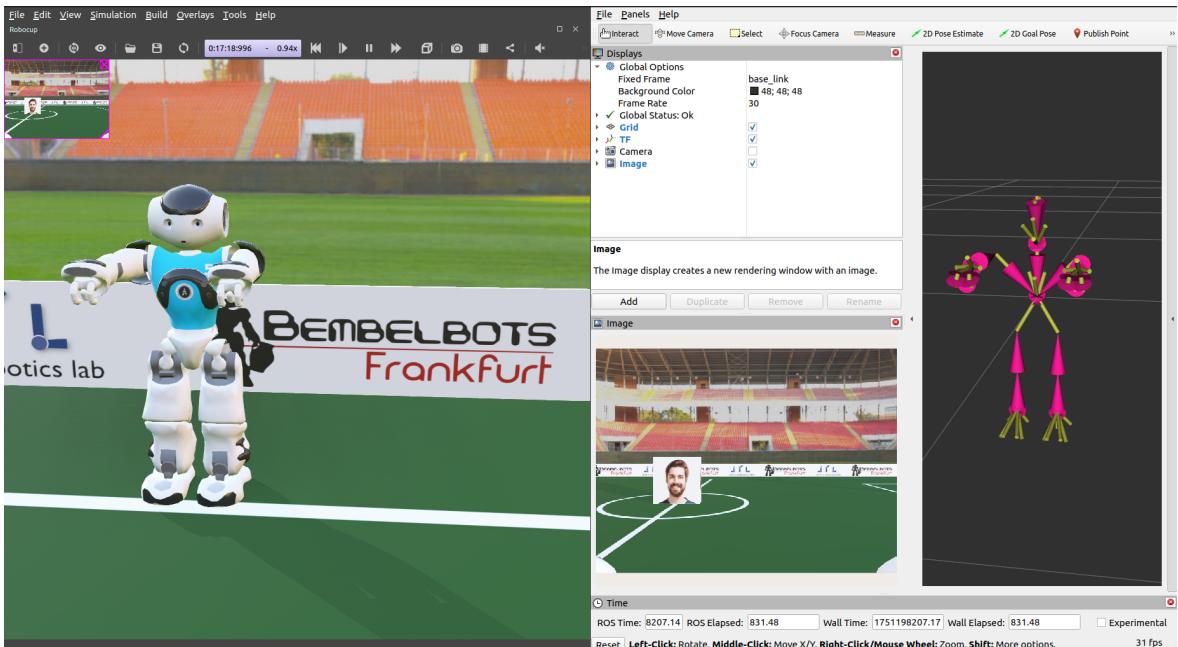


Figura 2.5: Integración del simulador Webots y visualización en RViz

En esta imagen se muestra la integración entre el simulador *Webots* (izquierda) y la visualización en *RViz* (derecha). El simulador genera los datos de sensores y actuadores, como la imagen de la cámara, que se publica en ROS 2.

El árbol de transformaciones (TF) no es generado directamente por el simulador, sino que se publica siguiendo el procedimiento explicado en la Sección 2.4.2: el nodo `nao_lola_client`, en este caso ejecutándose en el PC, recibe los datos del simulador a través del controlador y publica, entre otras cosas, las posiciones de las articulaciones en el topic `joint_states`, a partir de los cuales se construye el árbol TF y se visualiza en *RViz* igual que si se tratara del robot real.

Para los experimentos de detección de rostros con *YOLOv8* [36], se añadió manualmente la imagen de una cara en el entorno simulado. Esto permitió validar en tiempo real la correcta publicación del topic de la cámara y la capacidad de los

algoritmos de visión artificial para detectar objetos en la simulación, igual que en el robot real.

2.5. Conclusión

La integración del robot Nao en ROS 2 presentada en este capítulo sienta las bases para un desarrollo más flexible y moderno sobre una plataforma abierta. Se ha conseguido dejar el robot completamente operativo bajo una instalación de *Ubuntu* y un conjunto de paquetes base, permitiendo exponer todos sus sensores y actuadores a través del middleware de referencia en robótica actual.

El trabajo realizado incluye desde la instalación y configuración del sistema operativo, el acceso a los sensores y actuadores del robot, la publicación de los marcos de referencia (TF) y la calibración de la cámara, hasta una estimación de la orientación y odometría utilizando técnicas efectivas como el filtro de Madgwick. Además, la posibilidad de usar un simulador como *Webots* y la adaptación del controlador **WebotsLoLaController** han sido claves para acelerar el desarrollo y validar los nodos sin depender siempre del hardware físico.

El resultado es una base funcional sobre la que construir comportamientos y aplicaciones más complejas, tanto en el robot real como en simulación. Este sistema ha sido pensado para ser modificado y ampliado, facilitando así el trabajo colaborativo y la experimentación de posibles algoritmos de percepción, control y navegación en el futuro.

Además, al estar basado en código abierto, cualquier persona interesada puede acceder, adaptar y mejorar este trabajo, contribuyendo así a la comunidad y a una robótica más accesible para todos.

Capítulo 3

Marcha bípeda en el Nao con el paquete walk

The journey of a thousand miles begins with a single step.

Lao Tzu [48]

3.1. Introducción

La marcha bípeda en robots humanoides es una de las tareas más complejas y desafiantes dentro de la robótica. Para dotar al robot Nao de esta capacidad en el entorno ROS 2, se ha utilizado el paquete `walk`, un motor de locomoción originalmente desarrollado por Kenji Brameld y colaboradores dentro de la organización *ROS Sports* [30]. Este paquete ha sido adaptado específicamente para ROS 2 y para las particularidades del robot Nao V6, permitiendo al robot caminar de forma controlada, robusta y en tiempo real.

A lo largo de este capítulo se describe en detalle la arquitectura del paquete, su funcionamiento interno, los parámetros principales que afectan al comportamiento del robot durante la marcha, así como las modificaciones personalizadas realizadas en este proyecto para mejorar la estabilidad y la seguridad.

3.2. Estructura del paquete walk

El paquete `walk` se basa en una arquitectura modular que permite generar comandos de posición para las articulaciones del robot Nao, con el objetivo de que el robot camine. Estos comandos se publican periódicamente en el topic `/effectors/joint_positions`, para ser interpretados después por el nodo `nao_lola` (LoLA) en el robot.

Los componentes principales del paquete son:

- **Generador de pasos:** calcula trayectorias para las plantas de los pies en función de la velocidad deseada, frecuencia de paso y amplitud.
- **Cinemática inversa:** determina los ángulos articulares necesarios para alcanzar las posiciones deseadas de los pies.
- **Lectura de presión en los pies:** utiliza los sensores de fuerza (FSR) ubicados en las plantas de los pies para monitorizar la distribución de peso y detectar eventos como el contacto con el suelo o el equilibrio durante la marcha.
- **Control de brazos (ampliación):** coordina el movimiento de los brazos con la marcha para mejorar la estabilidad del robot.
- **Gestión de estados (ampliación):** maneja las transiciones entre estar de pie, caminar, detenerse o recuperarse de una caída.

El nodo principal se lanza como `walk`, y recibe comandos de activación mediante el topic `/walk_control` de tipo `std_msgs/Bool`, permitiendo iniciar o detener la marcha de manera segura.

3.3. Modificaciones personalizadas

Durante el desarrollo del TFG se ha creado un fork personalizado del paquete `walk`, disponible en GitHub [29]. Este fork incluye diversas mejoras orientadas a mejorar la estabilidad, seguridad y realismo de la marcha en el Nao V6.

3.3.1. Sincronización de brazos

Se ha añadido un módulo que mueve los brazos del robot en coordinación con los pasos, imitando la oscilación natural del cuerpo humano durante la marcha. Este movimiento ayuda a compensar las inercias y mejora la estabilidad general. Los brazos se mueven en fase contraria a las piernas, generando un efecto de mayor equilibrio y naturalidad al caminar.

El movimiento se define como una función lineal parametrizada, y puede activarse o desactivarse fácilmente. Se han añadido parámetros para ajustar la amplitud, desfase y posición base de los brazos (`arm_base_position`, `arm_swing_amplitude`, etc.).

3.3.2. Ajuste de parámetros para mejorar la estabilidad

Durante las pruebas se identificaron diversas situaciones en las que el robot tenía tendencia a perder el equilibrio, especialmente al iniciar la marcha o al caminar durante períodos más largos de tiempo. Para resolver estos problemas se ha realizado un ajuste de diferentes parámetros de configuración, definidos en el archivo `params.cpp` [29]. Algunos de los más relevantes son:

- **max_forward**: Velocidad máxima de avance del robot en línea recta (m/s). Se ha fijado en 0.1 para evitar desplazamientos demasiado rápidos que comprometan la estabilidad.
- **max_left**: Velocidad máxima de desplazamiento lateral (m/s). Establecida en 0.05, permite movimientos controlados hacia los lados.
- **max_turn**: Velocidad angular máxima de giro (rad/s). Se ha limitado a 0.5 para cambios de orientación más seguros.
- **speed_multiplier**: Factor de escala global para todas las velocidades. Con un valor de 0.8, reduce proporcionalmente las velocidades para suavizar la marcha.
- **foot_lift_amp**: Altura máxima a la que se levanta el pie durante la fase de oscilación (m). Disminuida a 0.002 para reducir el riesgo de pérdida de equilibrio y minimizar el impacto contra el suelo.
- **period**: Tiempo total de un ciclo completo de paso (s). Ajustado a 0.3, reduce la frecuencia de pasos, otorgando al robot más tiempo para estabilizarse en cada fase.
- **dt**: Intervalo de tiempo entre ejecuciones del algoritmo de generación de comandos (s). Establecido en 0.01, ofrece un control reactivo y preciso del movimiento.
- **sole_x**: Coordenada X de la planta del pie con respecto a la cadera en posición estática (m). Configurada en -0.022, asegura una postura de equilibrio adecuada para el Nao.
- **sole_y**: Coordenada Y lateral de la planta del pie respecto a la cadera (m). Se ha definido como 0.05 para una distancia apropiada entre ambos pies.
- **sole_z**: Coordenada Z (altura) de la planta del pie respecto a la cadera (m). Ajustada a -0.315 para mejorar el equilibrio (ligera flexión de las rodillas).

- **max_forward_change:** Incremento máximo permitido de la velocidad de avance en un solo paso (m/s). Establecido en 0.04, ayuda a evitar aceleraciones bruscas.
- **max_left_change:** Incremento máximo de velocidad lateral en un ciclo (m/s). También configurado en 0.04 para garantizar desplazamientos suaves.
- **max_turn_change:** Incremento máximo permitido en la velocidad de giro por ciclo (rad/s). Con un valor de 0.6, se evitan giros bruscos que puedan desestabilizar al robot.
- **footh_forward_multiplier:** Multiplicador que determina cuánto se eleva el pie según la magnitud del paso en dirección frontal. Fijado en 0.1, mejora la trayectoria que hace el pie sin excederse en altura.
- **footh_left_multiplier:** Igual que el parámetro anterior pero aplicado al eje lateral. Con un valor de 0.15, regula la esa elevación durante pasos laterales.

La combinación de todos estos ajustes da como resultado una marcha más suave, estable y segura, especialmente adecuada para entornos como el laboratorio donde se realizaron las pruebas.

3.3.3. Recuperación tras caída

La robustez de la marcha bípeda en robots humanoides implica necesariamente mecanismos efectivos para detectar y gestionar situaciones de caída. Para ello, se ha desarrollado e integrado en el nodo un algoritmo basado en una máquina de estados que monitorea continuamente el estado del robot mediante el uso combinado de los datos de la IMU y los sensores de presión en los pies (FSR).

Detección de caída

La detección de caídas se realiza analizando simultáneamente dos fuentes de información sensorial:

- **Aceleración vertical (IMU):** Se analizan continuamente los valores de aceleración vertical medidos por la IMU. Cuando esta aceleración excede un umbral específico (**FALL_Z_THRESHOLD**), se sospecha que el robot ha caído y comienza un temporizador. Si esta condición se mantiene por encima de un período de tiempo definido (**FALL_TIME_THRESHOLD**), se confirma la caída.

- **Sensores de presión en los pies (FSR):** Se monitoriza continuamente el estado de los sensores de presión (FSR) situados en las plantas de los pies. Si la presión detectada en todos los sensores cae por debajo de un umbral durante un cierto tiempo, se activa automáticamente una parada de emergencia (`fsr_emergency_stop_`). Este mecanismo proporciona una importante medida de seguridad al robot en caso de la pérdida de contacto inesperada de los pies con el suelo, como ocurre en una caída. Además, corrige un problema conocido del paquete original `walk`, en el que dicha pérdida de contacto provocaba movimientos bruscos e inestabilidad en las articulaciones, pudiendo producir daños en los actuadores. Gracias a esto, el Nao puede ser levantado mientras camina y colocado en otra posición de manera segura, mejorando notablemente la fiabilidad de uso de este paquete.

Máquina de estados y recuperación

Una vez confirmada la caída por alguno de estos mecanismos, el robot pasa a un estado de seguridad (`security_fall_`), en el que se realizan automáticamente varias acciones secuenciales para asegurar una recuperación efectiva:

1. **Desactivación inmediata de la marcha:** Al detectar la caída, el sistema deshabilita automáticamente el movimiento del robot mediante la desactivación de los comandos de control de las articulaciones.
2. **Reinicio del estado interno del sistema de locomoción:** Para garantizar una recuperación limpia y sin interferencias previas, el estado interno del módulo de locomoción es reiniciado, restaurando todos los valores internos (fase, trayectorias de los pies, estado del paso actual, etc.) a condiciones iniciales seguras.
3. **Comando de recuperación automática:** Una vez que se confirma la parada de seguridad, se determina la orientación de la caída analizando la aceleración longitudinal del robot (`accel_x`). Dependiendo de esta orientación, se selecciona automáticamente la maniobra más adecuada para levantarse (`getupFront` para caídas hacia adelante, `getupBack` para caídas hacia atrás).
4. **Ejecución del gesto de recuperación:** La acción seleccionada se envía al módulo `nao_pos`, que se encarga de ejecutar movimientos predefinidos, para levantar el robot del suelo de forma segura y efectiva.

5. **Monitorización y confirmación de recuperación:** El sistema espera la confirmación de que la acción de `nao_pos` se ha realizado correctamente para asegurar que la recuperación se haya completado de manera satisfactoria. Al finalizar el gesto de recuperación, si previamente la locomoción estaba activada, el sistema restablece automáticamente la capacidad de caminar, volviendo a poner todos los actuadores a su estado de rigidez (stiffness) y realizando la preparación postural previa a la marcha, descrita en la Sección 3.4.

Manejo avanzado de falsos positivos

Para evitar falsas alarmas que podrían resultar de movimientos bruscos puntuales o datos ruidosos, el sistema:

- Emplea una ventana temporal de evaluación (`FALL_FREQUENCY_TIME_WINDOW`) para filtrar sacudidas rápidas, vibraciones o movimientos bruscos puntuales que podrían activar falsamente el detector de caídas. Solo si se detectan múltiples indicios consecutivos de caída dentro de ese intervalo (más de `FALL_FREQUENCY_THRESHOLD`) se inicia el procedimiento completo de recuperación, lo que también aporta robustez frente a errores sensoriales ocasionales.
- Implementa una lógica de restablecimiento que, en caso de normalización rápida de la aceleración o restauración de contacto con el suelo, aborta la secuencia de recuperación, permitiendo al robot retomar su funcionamiento normal sin necesidad de una interrupción.

Este mecanismo, que combina análisis sensoriales con una gestión de estados internos, asegura que el robot pueda operar con seguridad en entornos dinámicos, reduciendo significativamente riesgos y maximizando su autonomía.

3.4. Preparación postural y lógica de arranque

Para asegurar que la marcha del robot sea estable y se pueda activar o desactivar deliberadamente, es fundamental que el Nao adopte siempre una postura inicial correcta antes de comenzar a caminar o tras recuperarse de una caída. En este trabajo, este aspecto se ha abordado mediante la ejecución de dos gestos secuenciales, gestionados a través del paquete `nao_pos`:

- **Gesto stand para piernas y tronco:** Primero, se activa el gesto `stand`, el cual alinea las piernas y el tronco en una posición vertical, ligeramente agachada

y simétrica. Esta postura garantiza que las articulaciones de cadera, rodillas y tobillos se encuentren en una posición neutra y estable, favoreciendo una transición suave y controlada hacia la fase de locomoción.

- **Colocación de brazos para la marcha:** Después, se ejecuta un gesto específico que posiciona ambos brazos en una postura recogida y alineada con el tronco. Este ajuste reduce el riesgo de colisiones accidentales con el entorno, evitando el peligro de iniciar la marcha con los brazos mal colocados.

Ambos gestos se lanzan automáticamente tanto en la activación del nodo `walk` como tras cualquier recuperación por caída. Gracias a ello, el robot siempre parte de una configuración conocida y segura, minimizando los riesgos asociados a posturas forzadas, articulaciones desalineadas o brazos mal posicionados.

Además, este procedimiento se combina con un mecanismo para evitar que comandos de velocidad sean aplicados cuando, una vez iniciada la marcha, el robot aún no está perfectamente estabilizado, reduciendo la probabilidad de tambaleos o caídas inesperadas al iniciar la locomoción.

3.5. Activación y control

El control del nodo `walk` se realiza principalmente a través del topic `/walk_control`, que acepta mensajes tipo `Bool`. Al recibir `true`, el robot comienza a caminar según los parámetros actuales. Al recibir `false`, se detiene.

Para realizar pruebas o cambiar parámetros dinámicamente, se pueden usar comandos `ros2 param set` como:

```
ros2 param set /walk foot_lift_amp 0.01
ros2 param set /walk period 1.2
```

También se podrían integrar estos comandos en un nodo autónomo para un mayor control y flexibilidad de la marcha.

3.6. Cálculo de la posición para la odometría

Uno de los retos clave al trabajar con robots bípedos como el Nao es estimar con precisión la posición y la trayectoria recorrida durante la marcha. En este proyecto, la

solución implementada se basa en combinar la información de orientación obtenida a través del filtro de Madgwick (descrita en el capítulo anterior, Sección 2.4.3) con los desplazamientos lineales que calcula el propio paquete `walk` a partir de la cinemática inversa de las piernas.

El cálculo de la odometría se realiza mediante un nodo específico llamado `nao_walk_odometry`, desarrollado en *Python*. Este nodo integra la información de dos fuentes principales:

- La **orientación** del robot, que se recibe en tiempo real desde la IMU (filtro de Madgwick) en forma de cuaternion, y se extrae la rotación en el eje Z, para usarla como referencia principal de la orientación en el cálculo de la posición.
- La **velocidad lineal** (avances y desplazamientos laterales), que el propio paquete `walk` publica en el topic `/walk/current_twist` utilizando mensajes de tipo `Twist`. Estas velocidades reflejan los desplazamientos calculados internamente por el generador de pasos y la cinemática inversa de las piernas.

El nodo recibe continuamente ambos datos, y utiliza la orientación para obtener los desplazamientos lineales en el marco global. El cálculo de la posición se realiza mediante integración directa en el plano (X, Y):

- Se almacena la posición actual (x, y) y el ángulo de orientación θ .
- En cada ciclo de actualización (20 Hz), se calculan los incrementos de posición según las velocidades lineales recibidas, obteniendo el avance y el desplazamiento lateral en el eje global:

$$\begin{aligned}\Delta x &= (v_x \cos \theta - v_y \sin \theta) \Delta t \\ \Delta y &= (v_x \sin \theta + v_y \cos \theta) \Delta t\end{aligned}\tag{3.1}$$

Ecuación 3.1: Cálculo de los incrementos de posición global ($\Delta x, \Delta y$) a partir de la orientación y las velocidades lineales en el robot Nao.

Donde:

- Δx : Desplazamiento en x
- Δy : Desplazamiento en y
- v_x : Componente de velocidad en x
- v_y : Componente de velocidad en y

- θ : Ángulo de rotación (radianes)
 - Δt : Intervalo de tiempo
- Se suma cada incremento a la posición previa, manteniendo actualizada la estimación global de la posición.

Fragmento de la ecuación aplicada en *Python*:

```
delta_x = (self.vx * math.cos(self.theta) - self.vy * math.sin(self.theta))
          * dt
delta_y = (self.vx * math.sin(self.theta) + self.vy * math.cos(self.theta))
          * dt
self.x += delta_x
self.y += delta_y
```

Código 3.1: Fragmento de código en Python que calcula los incrementos de posición del robot Nao para la odometría, usando la orientación y velocidades lineales.

La orientación θ proviene de la IMU, calculada mediante el filtro de Madgwick, lo que evita la deriva que tendría una simple integración del giroscopio. Además, con este método se consigue que la dirección de avance y cualquier giro del robot, incluso giros inesperados provocados externamente o por deslizamientos de los pies, se reflejen fielmente en la estimación de la odometría.

El nodo `nao_walk_odometry` se encarga de publicar el mensaje de odometría (`nav_msgs/Odometry`) con la posición y orientación estimadas, así como la velocidad actual.

Mejora de la precisión con filtro de Kalman

Para mejorar aún más la fiabilidad de la localización, la posición calculada y la orientación se integran en un filtro de Kalman mediante el paquete `robot_localization`. Este filtro permite fusionar diferentes fuentes de información para corregir errores y compensar la deriva, y se ha optado por su utilización para que en el futuro se puedan añadir más sensores y generar una estimación de posición más robusta y precisa.

El nodo `robot_localization` es el encargado de publicar el mensaje de odometría filtrada en el topic `/odometry/filtered` y de publicar la transformación (TF) de `odom` a `base_link`.

Gracias a esta arquitectura, el robot Nao puede usar una odometría bastante fiable para tareas como SLAM o navegación.

3.7. Pruebas realizadas

Las pruebas del sistema de locomoción se realizaron en el Laboratorio de Robótica y Sistemas Ubícuos de la Escuela de Ingeniería de Fuenlabrada de la URJC. El robot fue capaz de ejecutar secuencias de marcha estables, detenerse de forma controlada y reanudar la locomoción tras breves pausas.

En concreto, se evaluaron los siguientes aspectos, con resultados favorables:

- Inicio y parada suave del nodo `walk`.
- Coordinación entre los movimientos de brazos y piernas.
- Estabilidad del robot con los parámetros modificados.
- Capacidad de recuperación tras una caída.
- Coordinación de la locomoción con otras acciones, como gestos o diálogo.
- Cálculo y seguimiento de la odometría durante la marcha.
- Prueba funcional de seguridad: el robot fue levantado del suelo durante la marcha para comprobar que el sistema detenía automáticamente el movimiento, permitiendo recolocarlo en otra posición y reanudar la marcha con normalidad al volver a apoyarlo.

Los resultados obtenidos fueron positivos, logrando una marcha fluida y sin caídas espontáneas en condiciones normales. En situaciones más extremas (como superficies deslizantes o la presencia de obstáculos que provocaban caídas), el robot fue capaz de levantarse, recuperarse de forma segura y reanudar la marcha de manera autónoma. Además, el sistema de odometría permitió monitorizar en todo momento la posición del robot, lo que resulta especialmente útil para tareas de navegación y mapeo.

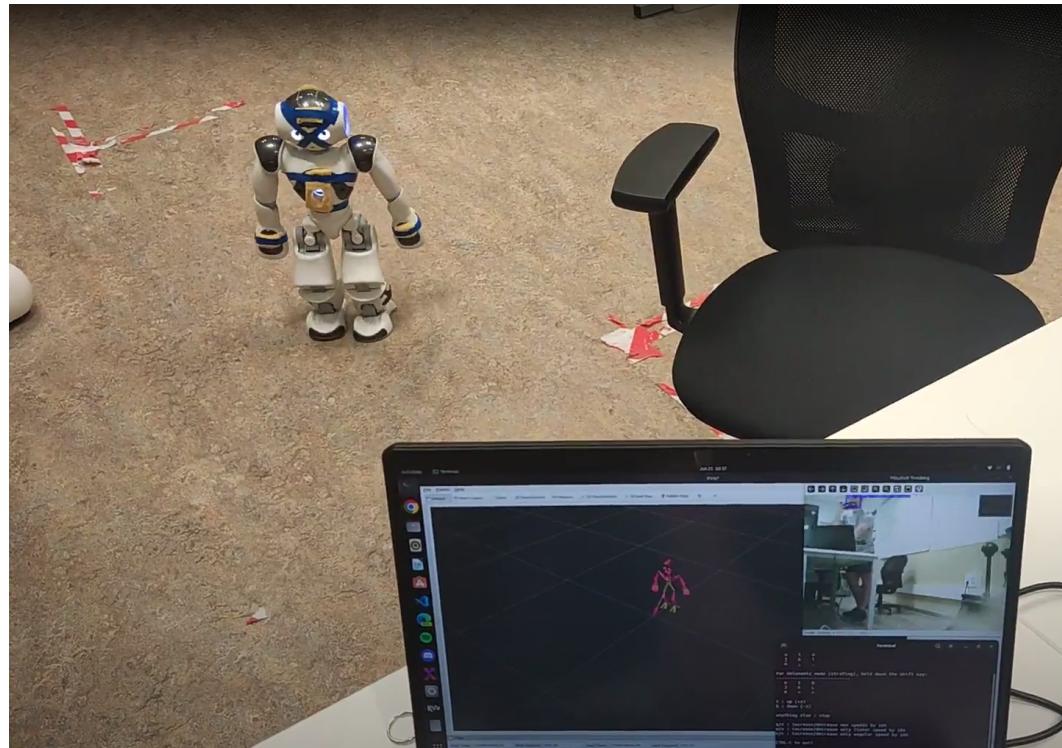


Figura 3.1: Prueba de odometría satisfactoria con el Nao caminando

3.8. Conclusión

La utilización del paquete `walk`, con ajustes personalizados y mecanismos de seguridad añadidos, ha permitido dotar al robot Nao de una marcha bípeda realista, estable y funcional en ROS 2. La seguridad del sistema se ha visto reforzada gracias a la integración de mecanismos específicos para la detección y recuperación ante caídas, así como a la implementación de medidas como la parada automática al detectar que el robot es levantado del suelo. Esta funcionalidad no solo protege al robot y a su entorno, sino que también facilita la manipulación manual, permitiendo recolocarlo y reanudar la marcha de forma sencilla y segura.

Un aspecto especialmente relevante es la incorporación del cálculo de odometría durante la marcha. Estimar la posición y trayectoria en robots bípedos es una tarea compleja debido a la inestabilidad y a la acumulación de errores durante el movimiento, pero resulta imprescindible para la navegación autónoma y otras funcionalidades avanzadas. La odometría implementada proporciona al sistema una herramienta fundamental para el desarrollo futuro, permitiendo que el robot se oriente, planifique rutas y ejecute tareas en entornos desconocidos con mayor precisión.

En conjunto, esta base robusta y segura permite integrar a futuro sistemas de navegación autónoma, detección y evitación de obstáculos, o modos de interacción más complejos, ampliando así las capacidades del robot como plataforma autónoma y social.

Capítulo 4

Interacción Humano–Robot en el Nao

La interacción humano-robot (HRI) representa uno de los retos y oportunidades más relevantes de la robótica actual, especialmente en plataformas humanoides como el Nao. En el contexto de este trabajo, la transición del Nao V6 a un entorno completamente abierto —basado en *Ubuntu 22.04* y ROS 2— supone un gran cambio para la investigación y el desarrollo de algoritmos HRI avanzados, superando las limitaciones impuestas por el software propietario original.

Bajo esta nueva arquitectura, el robot Nao se convierte en un agente social y autónomo, capaz de integrar canales de percepción y comunicación: visión artificial, síntesis y reconocimiento de voz (STT, TTS), gestos fluidos y naturales, y la habilidad de mostrar expresiones y estados de ánimo mediante sus LEDs. Esta arquitectura modular, distribuida y estandarizada a ROS 2, permite controlar de forma coordinada la locomoción, el diálogo y la expresión no verbal del robot, abriendo la puerta a una interacción más natural y personalizable con los seres humanos.

Se han implementado mejoras en un fork del paquete original `hni` de Antonio Bono [49], ahora llamado `nao_hri` [35], que incluye reconocimiento de voz mejorado, uso de *Generative Pre-trained Transformer* (GPT) en español con personalidad específica e integración con `nao_pos`.

4.1. Uso de `nao_pos` para gestos predefinidos

El paquete `nao_pos`, creado por el ya mencionado desarrollador Kenji Brameld y mejorado en un fork personalizado por Antonio Bono [50], permite al robot Nao ejecutar gestos y posturas almacenadas en archivos `.pos`, describiendo de forma simple y eficiente cómo se deben mover sus articulaciones. Se ha adaptado el paquete original de Antbono para incorporar nuevos gestos [28].

4.1.1. Formato de los archivos .pos

Los archivos .pos son tablas en las que cada fila define una postura del robot. Cada columna corresponde a una articulación, y la última columna, DUR, indica la duración en milisegundos que el robot debe tardar en alcanzar esa postura. Por ejemplo:

HY	HP	LSP	LSR	...	RH	DUR
0	0	90	10	...	0	1000
0	30	80	20	...	0	1500

Código 4.1: Ejemplo de formato de archivo .pos: cada fila representa una postura y su duración para el robot Nao.

Aquí, cada valor numérico representa el ángulo de la articulación correspondiente. Al ejecutar este archivo, el robot mueve sus articulaciones desde la postura anterior hasta la nueva en el tiempo especificado por DUR.

4.1.2. Funcionamiento del nodo nao_pos_server

El servidor nao_pos_server proporciona un servicio ROS 2 que permite ejecutar estos gestos utilizando una llamada sencilla:

```
ros2 action send_goal /nao_pos_action
nao_pos_interfaces/action/PosPlay "action_name: 'stand'"
```

Código 4.2: Ejemplo de llamada a la acción para ejecutar un gesto predefinido en el Nao usando nao_pos.

Este servicio:

1. Carga el archivo `stand.pos` desde el paquete.
2. Publica las posiciones articulares de manera secuencial, respetando los tiempos indicados y con movimientos fluidos.
3. Devuelve una confirmación cuando el gesto ha finalizado.

Esto facilita la integración de movimientos del Nao en situaciones más complejas, por ejemplo, al responder en una conversación.



Figura 4.1: Nao responde a la pregunta “Hola Nao, ¿que tal estás?” con “Hola, estoy muy bien, gracias, aquí, listo para ayudarte. Y tú, ¿cómo estás?”, mientras saluda con la mano (gesto de `nao_pos`).

4.2. Reconocimiento de voz (STT)

Para la conversión de voz a texto se ha utilizado el modelo Whisper de OpenAI, integrándolo en un servicio ROS 2 personalizado. Whisper es un sistema *Automatic Speech Recognition* (ASR) multilingüe de código abierto entrenado con cientos de miles de horas de audio.

Estudios recientes muestran que Whisper supera ampliamente en precisión a otros servicios populares (por ejemplo, Google Speech-to-Text) [51]. Debido a esto, y dado que estamos trabajando en español, se ha optado por utilizar Whisper en lugar del STT de Google Cloud. El flujo del STT en el sistema es el siguiente: el micrófono del Nao captura la voz de la persona y envía el audio a través de la red al servidor que ejecuta el modelo Whisper, que devuelve el texto transcritto. En este sistema, el reconocimiento de voz se activa al presionar una tecla en el PC. En ese momento, el nodo correspondiente comienza a escuchar utilizando detección automática de voz *Voice Activity Detection* (VAD) y sigue grabando hasta que detecta que la persona ha terminado de hablar, según un umbral configurable. A continuación, el audio grabado se envía al modelo Whisper para la transcripción y se recibe el texto transcritto, con bastante acierto.

4.3. Modelo de diálogo con GPT (LLM)

El texto obtenido por STT se envía a un servicio de chat basado en OpenAI GPT. Para ello se define una interfaz de servicio ROS 2 tipo `Chat` (definida en el paquete `hni_interfaces`) que invoca la API de OpenAI con el modelo LLM GPT-4o. El sistema de chat mantiene un contexto de mensajes (prompt), incluyendo un mensaje `system` inicial que define la personalidad y rol del Nao. En este caso, el prompt inicial le indica al GPT que es la sexta versión del robot Nao en el laboratorio de Fuenlabrada, cuya misión es ayudar en rehabilitación con un carácter amable y motivador. De este modo, cuando se le pregunta algo (por ejemplo, “Hola Nao, ¿cómo estás?”), el GPT genera una respuesta adecuada al personaje. Internamente, el código del servidor de chat (en `chat_service.py`) hace lo siguiente: recibe la petición con el texto de entrada, llama a la API de OpenAI y devuelve la respuesta generada.

Desde el terminal, la interfaz de servicio podría llamarse con:

```
ros2 service call /chatGPT_service  
hni_interfaces/srv/Chat "{text: 'Hola Nao, ¿cómo estás?'}"
```

Código 4.3: Llamada al servicio de chatGPT para obtener una respuesta en español en el sistema de HRI del Nao.

La respuesta será un string en español (por ejemplo, “Hola, me encuentro muy bien, ¡gracias!, ¿cómo puedo ayudarte?”). Según las reglas del prompt, el GPT también puede decidir mencionar gestos (palabras clave como saludar, decir pequeño, grande, etc.) si la pregunta lo sugiere. El nodo cliente interpreta estas indicaciones buscando palabras específicas en la respuesta y lanza luego la acción correspondiente al servidor de `nao_pos`.

4.4. Síntesis de voz (TTS)

Para que el Nao reproduzca la respuesta generada, se convierte el texto a audio. En la versión original de antbono se usaba Google Cloud TTS en inglés, mediante el servicio `gtts_service`, y se ha mantenido este enfoque por su alta calidad también en español, y sobre todo su velocidad de respuesta, aunque es posible sustituirlo (por ejemplo, por la API TTS de OpenAI, que tiene voces más realistas y naturales, pero por ahora extremadamente lentas y caras). El servicio ROS 2 `gtts_service` recibe una cadena de texto y utiliza la librería `google.cloud.texttospeech` para convertirla a voz. Después,

el audio resultante (formato OGG/Opus) se envía al nodo de reproducción de sonido del Nao (`sound_play`) para su reproducción.

De este modo, cerramos el ciclo de comunicación: el humano habla, STT convierte a texto, GPT genera una respuesta, y finalmente TTS convierte a voz la respuesta.

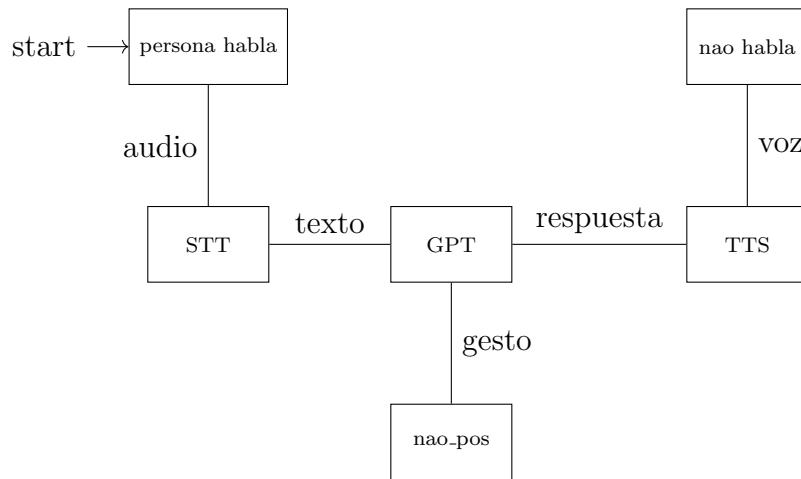


Figura 4.2: Diagrama simplificado del flujo de interacción conversacional con el Nao

4.5. Detección y seguimiento de rostros

Además de la comunicación verbal, el Nao puede utilizar su cámara para detectar personas y mantener la atención en sus caras. Para ello, antbono integró un modelo YOLOv8 especializado en detección de rostros. YOLOv8 es un modelo de visión artificial de última generación que ofrece una gran precisión y velocidad para detección de objetos en tiempo real [36].

En esta arquitectura, la detección de rostros no se ejecuta en el propio Nao, sino en un PC externo más potente, con GPU y Nvidia *Compute Unified Device Architecture* (CUDA) [52]. Se transmite la imagen de la cámara del Nao a través de un topic de ROS 2 con el nodo `usb_cam`, y el nodo `yolo_face_track_server`, en el PC (conectado al mismo Wi-Fi que el Nao), procesa el video con YOLOv8 para localizar rostros. Cuando se detecta un rostro, se calcula su posición en la imagen y se envía un comando al actuador de la cabeza del robot mediante un gesto de `nao_pos`. Con ello, el Nao gira, sube y baja la cabeza para seguir la cara de la persona. Esta funcionalidad utiliza un formato cliente/servidor de acción de ROS 2: el nodo cliente envía metas (goals) al servidor `nao_pos_action`, que escucha posibles acciones de gestos para el

movimiento de la cabeza.

El resultado de todo esto es que el Nao mantiene contacto visual con la persona que le está hablando, dando una sensación de conexión, que no sería posible si el robot mirase a otro sitio mientras se mantiene una conversación con él.

La integración completa se realiza mediante un conjunto de nodos y acciones de ROS 2 coordinados por archivos de lanzamiento (launchers). Por ejemplo, los `launcher`, `experiment_nao_launch.py` y `experiment_pc_launch.py` inician simultáneamente: el cliente de voz a texto (STT), el servidor de chat (GPT), el servidor de texto a voz (TTS), el servidor de seguimiento de rostros y los servidores `nao_pos` para cabeza y cuerpo.

4.6. Coordinación de nodos para un HRI completo

El nodo `ModeSwitcher` actúa como coordinador principal entre el nodo de marcha bípeda `walk` y el sistema de interacción humano-robot (HRI) en el robot Nao. El nodo `ModeSwitcher` se encarga de sincronizar la locomoción con las posturas y gestos, garantizando que el robot camine sólo cuando esté en la postura adecuada y deteniendo la marcha cuando sea necesario.

En la implementación del nodo, se configuran diversos suscriptores y publicadores para coordinar los diferentes nodos de control. Se crea una suscripción al topic `/target` (mensaje tipo `geometry_msgs/Twist`), asociada a un callback, para recibir información de la velocidad comandada al robot y actuar en consecuencia, así como al topic `/nao_pos_action/status` (mensaje tipo `std_msgs/String`) para recibir estados de acciones de gestos de `nao_pos` que se están ejecutando.

Por otro lado, publica en los siguientes topics: `/walk_status` y `/walk_control` (mensajes tipo `std_msgs/Bool`), para indicar al módulo `walk` cuándo comenzar o detenerse. Además, publica mensajes para controlar el estado de la rigidez de las articulaciones, en `/effectors/joint_stiffnesses` y los equivalentes para controlar la posición de éstas, en `/effectors/joint_positions`, así como mensajes tipo `SolePoses` en `/motion/sole_poses` para fijar la posición inicial (ligeramente agachado) antes de caminar.

Finalmente, envía los comandos de gestos predefinidos de `nao_pos` en los topics `action_req_legs` y `action_req_arms` (por ejemplo, “stand” o “armsDown”) y espera confirmaciones vía `/nao_pos_action/status`.

1. Al recibir un mensaje `Twist` en `/target`, primero se comprueba que haya pasado suficiente tiempo desde la última petición (se limita la frecuencia de procesamiento a 0.5 Hz [hay que tener en cuenta que esta limitación de frecuencia es solo para la detección de cambios de estado, y el nodo `walk`, cuando está activado, responde con su frecuencia normal]). A continuación, el nodo determina si el mensaje implica movimiento comprobando que, al menos, una componente lineal o angular no sea cero. Un mensaje tipo `Twist` de ROS 2 describe una velocidad en el espacio libre, descompuesta en partes lineales y angulares; por tanto, cualquier componente distinta de cero indica que se desea mover el robot.
2. Si el mensaje indica movimiento y el robot actualmente *no* está caminando (`is_walking == False`), el nodo publica `/walk_status = True` para señalar al módulo de locomoción que debe prepararse para iniciar la marcha. A continuación llama a la función `ensure_stand_then_walk()`, que verifica el estado de postura actual antes de comenzar a caminar. En caso contrario (si ya se estaba caminando), simplemente se ignora, ya que el robot ya se encuentra en marcha y el paquete `walk` se encarga de responder a esos comandos de movimiento.
3. Si el mensaje indica que no hay movimiento (todos los componentes del mensaje tipo `Twist` son cero) y el robot *estaba* caminando (`is_walking == True`), el nodo detiene la marcha. Publica `/walk_status = False` y `/walk_control = False` para informar al nodo de locomoción que debe detenerse, y marca `is_walking = False`. Luego llama a la función `ensure_stand()` para garantizar que el robot vuelva a adoptar la postura de pie, que le da estabilidad. En este modo el robot es capaz de interaccionar con personas utilizando el diálogo y el seguimiento de rostros.

Para iniciar la transición entre estar parado y caminar, el nodo emplea las siguientes funciones auxiliares:

- `ensure_stand_then_walk()`: si la bandera `swing_completed` es falsa (es decir, no se ha completado el gesto “swing” [movimiento de piernas que da naturalidad al

nao mientras habla]), no hace nada y espera. Cuando `swing_completed` pasa a `true`, verifica si el robot ya está de pie. Si no lo está (`is_standing == False`), llama a `ensure_stand()` para que el robot adopte la postura de pie. Si ya está de pie, llama a `start_walking()`.

- `ensure_stand()`: si `is_standing` es `false`, publica inmediatamente el gesto `stand` en `action_req_legs`.
- `start_walking()`: realiza la secuencia de acciones para iniciar la marcha:
 - Primero llama a `set_stiffness()` para fijar todas las articulaciones a su máxima rigidez, con el objetivo de lograr un mayor control de los movimientos durante la marcha.
 - Luego llama a `set_walk_sole_position()` para definir la posición inicial de las piernas.
 - Después, envía a `action_req_arms` la instrucción del gesto “`armsDown`” para poner los brazos hacia abajo.
 - Finalmente espera brevemente (2,5 segundos) y publica `/walk_control = True` para activar la marcha.
- `set_stiffness()` construye un mensaje `JointStiffnesses` con rigidez 1.0 en todos los índices de articulación (25 articulaciones totales) y lo publica en `/effectors/joint_stiffnesses`. Este mensaje permite fijar la rigidez de cada articulación del Nao, preparando al robot para realizar movimientos de locomoción.
- `set_walk_sole_position()` construye un mensaje `SolePoses` indicando la posición inicial de las piernas: 5 cm de separación lateral y 31.5 cm de distancia hacia abajo para cada pie (valores `position.y` y `position.z`), creando una postura estable para el arranque. Al publicar este mensaje `SolePoses`, un nodo de cinemática inversa convierte esta orden en posiciones articulares concretas (luego publicadas en `/effectors/joint_positions`)

Finalmente, al cabo de la breve pausa para permitir que el robot adopte la postura inicial y habiendo publicado `/walk_control = True`, se activa activa la marcha (el nodo `walk` monitorea este topic para activarse o desactivarse). Después de esto, el nodo actualiza sus booleanos internos (`is_walking = True`, `is_standing = False`, `swing_completed = False`).

El callback `action_status_callback` procesa las confirmaciones de los comandos de postura enviados. Cada vez que llega un mensaje con el campo `.data` indicando un estado de éxito (`succeeded`), el nodo interpreta qué acción se completó: si el mensaje contiene `only_legs_fast`, considera que el movimiento de piernas (`swing`) ha finalizado y marca `swing_completed = True`; si el mensaje contiene `stand`, entiende que la postura de ponerse de pie se completó y marca `is_standing = True`. De este modo, el nodo sabe exactamente cuándo las acciones previas han terminado, antes de avanzar al siguiente estado. Un callback de este estilo se utiliza también en el nodo `walk` para saber cuándo es posible lanzar un comando para levantarse tras una caída.

En resumen, este nodo coordina la interacción de dos sistemas críticos: el de locomoción y el de HRI. Se comunica con el módulo de marcha bípeda indicándole cuándo iniciarse/detenerse y de manera paralela, se integra con el sistema de gestos `nao_pos`, enviando solicitudes y monitorizando las confirmaciones.

Gracias a esta coordinación, el comportamiento completo resulta controlado y seguro: el robot sólo arranca la marcha cuando ha adoptado correctamente la postura de pie. Esta arquitectura de nodos permite un HRI más completo, donde cada componente mantiene su utilidad y trabaja de manera coordinada con el resto de módulos.

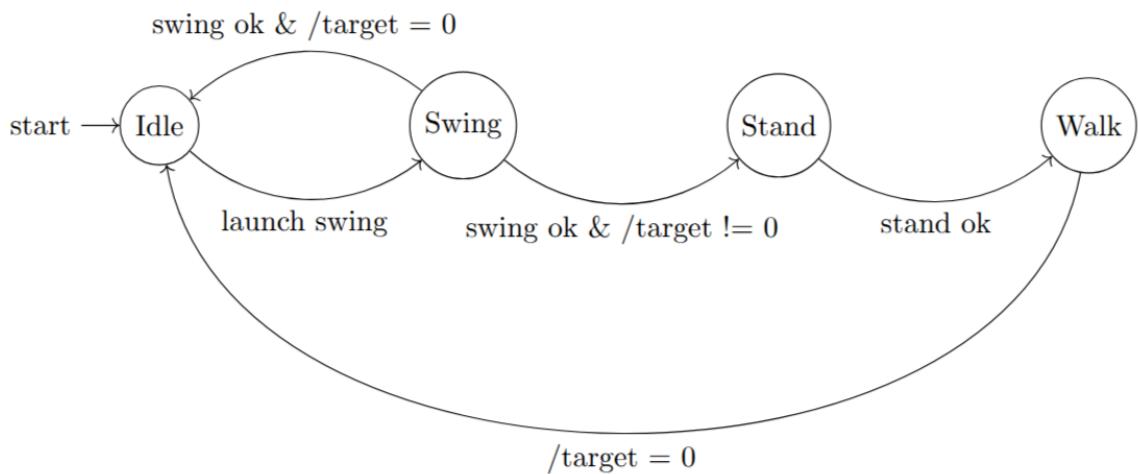


Figura 4.3: Estados y transiciones principales del nodo coordinador ModeSwitcher para locomoción y gestos en el Nao.

4.7. Pruebas realizadas y conclusiones

El sistema de interacción humano-robot desarrollado para el robot Nao ha sido validado a través de diferentes experimentos realizados en el Laboratorio de Robótica y Sistemas Ubícuos de la Escuela de Ingeniería de Fuenlabrada. Durante las pruebas, el robot demostró ser capaz de mantener conversaciones fluidas en español, reconocer la voz de los usuarios utilizando Whisper incluso en entornos con ruido moderado, y generar respuestas personalizadas gracias a la integración con GPT en español y un prompt adaptado al contexto del laboratorio.

Además, la detección y seguimiento de rostros mediante YOLOv8 permitió al Nao mantener la atención visual con las personas que tenía delante, orientando la cabeza según fuera necesario. El sistema de gestos, gestionado con `nao_pos`, permitió acompañar las respuestas verbales con movimientos, mejorando la expresividad del robot y haciendo la interacción más creíble y cercana.

Uno de los aspectos más relevantes fue la capacidad para coordinar todos estos sistemas de manera robusta y segura. Gracias al nodo `ModeSwitcher`, las transiciones entre diferentes modos de funcionamiento (como pasar de reposo a marcha, o viceversa) se realizaron sin problemas, garantizando en todo momento la seguridad y coherencia del comportamiento del robot.

En conclusión, la arquitectura propuesta, basada en ROS 2 y componentes abiertos, ha permitido transformar al Nao en una plataforma robusta y ampliable para la interacción humano-robot. Este sistema no solo supera las limitaciones del entorno propietario original, sino que sienta una base sólida para el desarrollo de nuevas capacidades en investigación, educación o asistencia.

Capítulo 5

Navegación autónoma en el robot Nao

5.1. Introducción

En este capítulo se detalla el proceso de integración y validación de la navegación autónoma en el robot Nao, alcanzando el objetivo de que el robot no solo pueda construir un mapa de su entorno mediante SLAM utilizando ROS 2, sino también navegar de manera autónoma hacia distintos puntos del espacio. A lo largo del desarrollo, se han realizado pruebas reales que han permitido comprobar el funcionamiento conjunto de todos los módulos y sentar las bases para futuras mejoras.

El trabajo combina diferentes componentes desarrollados y adaptados previamente: el cálculo de odometría específico para robots bípedos, la correcta publicación de las transformaciones TF, el control de la marcha a través de los nodos `mode_switcher` y `walk`, así como la integración del sensor LiDAR para dotar al robot de percepción del entorno. Todo este proceso, junto con las instrucciones y recursos necesarios para reproducirlo, se encuentra documentado en el repositorio de referencia desarrollado como parte principal de este proyecto [2].

En definitiva, este capítulo describe tanto los retos técnicos superados como los resultados obtenidos, mostrando que es posible dotar al Nao de navegación autónoma completa sobre ROS 2, y facilitando que cualquier persona interesada pueda replicar o extender el sistema a través del repositorio abierto del proyecto.

5.2. Limitaciones del hardware: conexión del sensor LiDAR

Uno de los principales retos durante el desarrollo ha sido la integración del sensor LiDAR (RPlidar S2) en el sistema, ya que la idea inicial era conectarlo directamente al puerto USB del robot Nao para que todo el procesamiento pudiera hacerse dentro del propio robot, sin depender de un ordenador externo conectado por cable.

No obstante, la integración del LiDAR en el Nao ha supuesto uno de los principales retos técnicos de este proyecto. Se han realizado numerosos intentos para lograr que el robot reconociera el sensor: desde probar diferentes versiones del kernel, instalar el driver `cp210x` ya compilado, hasta compilar manualmente el kernel con el soporte adecuado e intentar cargarlo en el propio Nao. También se ha experimentado utilizando un adaptador serie-`ftdi` en lugar del adaptador original, para utilizar un driver que sí que incluía el kernel del Nao, y se han probado varios modelos de LiDAR disponibles en el laboratorio. En todos los casos, los resultados no han sido satisfactorios: aunque con el adaptador *Future Technology Devices International* (FTDI) se llegó a recibir un primer frame de escaneo, rápidamente aparecía un error y el LiDAR dejaba de funcionar. Es posible que un adaptador de mayor calidad o potencia pueda solucionar la limitación, pero tras invertir un número considerable de horas de pruebas no se consiguió establecer una conexión estable y funcional.

Por ahora, la única opción viable ha sido conectar el LiDAR directamente al PC, donde sí está disponible el driver correspondiente en el kernel de Linux y el sistema funciona sin problema. Esta solución debe considerarse provisional. Se espera que, en futuras actualizaciones de la imagen de `NaoImage` (como ya se ha solicitado formalmente mediante una *issue* en el repositorio de `NaoDevils`), se incluya el driver `cp210x` en el kernel, lo que permitiría utilizar el LiDAR directamente desde el Nao y operar así de forma completamente autónoma. Mientras tanto, el flujo de trabajo para SLAM se ha implementado con el LiDAR conectado al PC —usando, si es necesario, un cable largo— y con el Nao comunicándose por Wi-Fi con el ordenador para el envío y recepción de información a través de topics.

5.3. Implementación de SLAM en el Nao

5.3.1. Montaje y publicación de los sensores

Para que el sistema de SLAM funcione correctamente, es imprescindible que ROS 2 conozca con precisión la posición del LiDAR en el marco del robot. En este caso, el LiDAR está montado en la parte superior de la cabeza del Nao y se conecta al PC por USB. Para que ROS 2 lo interprete bien, se publica una transformación estática (`static_transform_publisher`) que fija la posición del frame del LiDAR respecto al frame `Head` del robot. Esto permite que el mapa generado esté perfectamente alineado con el cuerpo del robot, incluso cuando éste gira la cabeza, y que todos los datos sean coherentes cuando se visualizan en *RViz* o se usan para navegación.

5.3.2. Control de la marcha para mapeo

Para poder mapear de forma segura y controlada, se utiliza el nodo `mode_switcher`, que gestiona el movimiento del robot y permite enviarle comandos de movimiento al nodo `walk` de manera segura, desde un PC o utilizando un joystick (implementado en el repositorio `nao_ros2` [2]). De esta forma, se puede teleoperar al Nao y asegurarse de completar la exploración del entorno. Todo el cálculo de odometría se realiza mediante los nodos explicados en capítulos anteriores, se publican los mensajes `nav-msgs/Odometry` y las TFs necesarias para que SLAM tenga acceso en tiempo real tanto a la posición estimada del robot como a los datos del LiDAR.

5.3.3. Lanzamiento de los nodos y configuración

El sistema completo se distribuye entre el Nao y el PC, coordinados por ROS 2 y la red Wi-Fi local. Los launcher y comandos utilizados en las pruebas reales han sido los siguientes:

- **En el Nao:**

```
ros2 launch nao_ros2 mode_switcher_nao_launch.py
```

Código 5.1: Lanzamiento del nodo `mode_switcher` directamente en el robot Nao para control seguro de la marcha.

Esto lanza el nodo `mode_switcher` en el propio Nao, que le permite caminar de manera controlada y teleoperada desde el PC.

```
ros2 launch nao_ros2 mode_switcher_pc_launch.py
ros2 launch sllidar_ros2 sllidar_s2_launch.py
ros2 run tf2_ros static_transform_publisher \
0.0 0.0 0.0 3.14 0.0 0.0 Head laser
ros2 launch nao_ros2 nao_odometry_launch.py
ros2 run rviz2 rviz2
```

Código 5.2: Secuencia de comandos para lanzar nodos necesarios en el PC: teleoperación, lectura del LiDAR, publicación de la transformación estática, cálculo de odometría y visualización en RViz.

- **En el PC:**

Aquí se lanzan los nodos necesarios para recibir los datos del robot, teleoperarlo, leer los datos del LiDAR, publicar la transformación del LiDAR respecto a la cabeza, calcular la odometría y mostrar todo en *RViz*.

- **En la máquina virtual (distrobox) del PC:**

Para ejecutar los nodos principales de navegación y SLAM, ha sido necesario utilizar una *distrobox* configurada con *Ubuntu 22.04* y *ROS 2 Humble*, en lugar de la distribución *Rolling* que se emplea en el resto del sistema. Esto se debe a que el paquete *Nav2* no está disponible para *Rolling*, por lo que la única manera de lanzar correctamente los nodos de navegación y planificación ha sido recurrir a *Humble*, que sí mantiene soporte completo y estable para *Nav2*.

```
ros2 launch nav2_bringup bringup.launch.py slam:=True \
map:=/home/andoni/empty_map.yaml params_file:=nav2_params.yaml

ros2 launch slam_toolbox online_async_launch.py \
params_file:=nav2_params.yaml /odom:=/odometry/filtered
```

Código 5.3: Comandos utilizados para lanzar los nodos principales de navegación y mapeo en una máquina virtual con Ubuntu 22.04 y ROS 2 Humble.

En este entorno, se ejecutan tanto el paquete *slam_toolbox* como los nodos de *nav2_bringup*, utilizando los parámetros personalizados definidos en *nav2_params.yaml*.

5.3.4. Creación del mapa del entorno

La exploración y mapeo del entorno se realiza de manera teleoperada: una vez lanzados todos los nodos y verificado en *RViz* que el sistema está funcionando

correctamente (se visualiza el robot, el mapa y las lecturas del LiDAR en tiempo real), se mueve al Nao usando el control de marcha. El robot avanza, gira y recorre las diferentes partes del laboratorio mientras el sistema de SLAM va construyendo y actualizando el mapa.

Durante este proceso, el paquete `slam_toolbox` es capaz de fusionar la odometría estimada (proporcionada por el sistema propio de odometría del robot bípedo) y los datos del LiDAR para crear un mapa bastante preciso del entorno. El mapa se puede guardar para su uso posterior en navegación autónoma, y la calidad final depende mucho de la precisión de la odometría, la estabilidad del robot y la cobertura del entorno durante el mapeo.

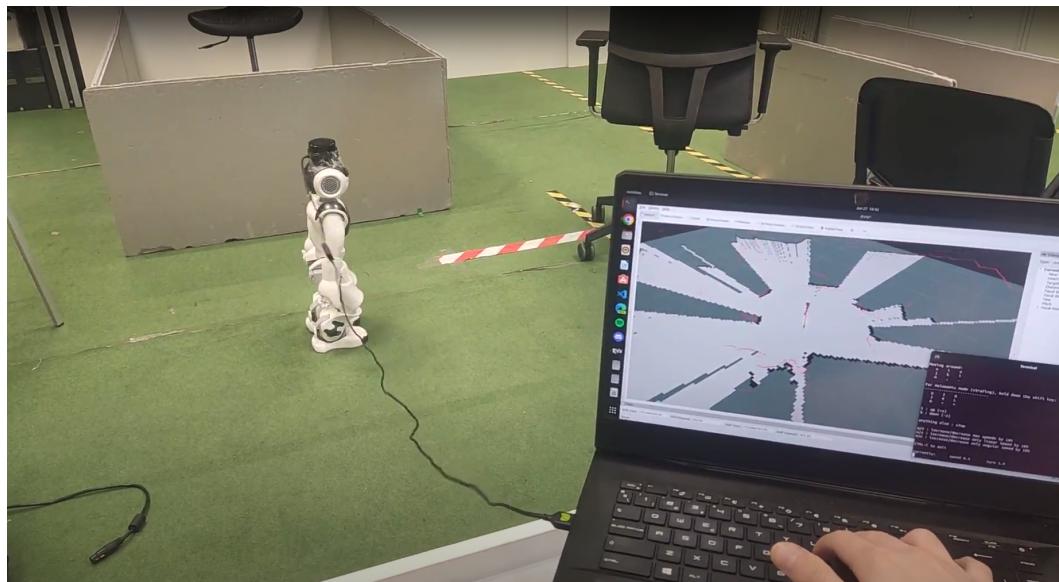


Figura 5.1: SLAM con el robot Nao teleoperado en un PC

5.3.5. Visualización y validación en *RViz*

Para la visualización se utiliza *RViz* como herramienta principal. En *RViz* se puede ver en tiempo real:

- El árbol de TFs del Nao y la posición estimada según la odometría.
- Los puntos y escaneos del LiDAR, coincidentes con el mapa generado.
- El mapa generado por `slam_toolbox`, que se va actualizando a medida que el Nao explora el entorno.
- Las trayectorias recorridas y cualquier posible error de alineación entre el LiDAR y el frame de odometría del robot.

El resultado ha sido satisfactorio: se ha conseguido generar un mapa del laboratorio, con paredes, sillas y zonas abiertas bien diferenciadas. Se ha demostrado la fiabilidad del sistema de odometría utilizado en el Nao y ha sido posible corregir pequeños errores de estimación y derivas con el LiDAR.

5.4. Navegación autónoma

La navegación autónoma ha sido uno de los retos más interesantes y satisfactorios del proyecto. Después de varias sesiones de prueba y algún que otro contratiempo, se ha conseguido que el Nao recorra trayectorias hacia puntos en un mapa de manera completamente autónoma, utilizando ROS 2 y el stack de navegación Nav2.

Una vez que se consiguió que Nav2 reconociera correctamente el mapa generado por SLAM (no sin algún problema inicial, ya que parecía vacío o no se visualizaba bien), se pudo definir la posición inicial y empezar a lanzar goals al robot desde la interfaz de RViz. La respuesta del sistema fue sorprendentemente buena: el `mode_switcher` activaba la marcha en cuanto se enviaba un objetivo, y el robot caminaba hasta llegar al punto deseado (eso sí, con un poco de inestabilidad provocada por el cable del LiDAR colgando). La parada también se realizaba de manera suave y controlada, algo que preocupaba especialmente en un robot bípedo como el Nao, pero que se tuvo en cuenta durante el desarrollo del `mode_switcher`.

Eso sí, hubo varios desafíos prácticos. El cable del LiDAR colgando provocaba ciertos movimientos inestables, haciendo que el robot se desequilibrara más de lo esperado. También hubo algún problema con las limitaciones del entorno: las paredes de corcho del laboratorio eran algo más bajas que la altura del sensor, lo que generaba pequeñas imprecisiones en la detección de obstáculos.

A pesar de estos detalles, los giros del robot eran precisos y, en general, la navegación se comportó mucho mejor de lo que imaginaba al principio. El sistema demostró robustez, y sobre todo, una buena integración entre la percepción, la localización y el control de la marcha.

Se considera que el objetivo de navegación autónoma está cumplido, aunque por supuesto hay margen de mejora. Principalmente, la estabilidad debería aumentar cuando se pueda conectar el LiDAR al propio Nao, y convendría investigar más sobre el

problema puntual de visualización del mapa. En cualquier caso, la base está sentada y el sistema funciona: el Nao ya es capaz de navegar de manera autónoma por un entorno conocido.

El proceso de lanzamiento de nodos, así como los parámetros recomendados y las posibles configuraciones alternativas, están detallados en el repositorio del proyecto [2].

5.5. Resumen y próximos pasos

En resumen, aunque por el momento no ha sido posible integrar el LiDAR directamente en el robot debido a la incompatibilidad del driver `cp210x` con el kernel actual, se ha logrado implementar y validar con éxito todo el sistema de SLAM y navegación utilizando el LiDAR conectado externamente al PC. Gracias a esto, el Nao ha sido capaz de crear mapas reales y precisos del entorno, así como de navegar de forma autónoma utilizando el stack de Nav2.

Con esta base técnica ya probada, el siguiente paso será trasladar tanto la lógica de SLAM como el sistema de navegación directamente al propio Nao en cuanto se disponga de una versión del sistema operativo que incluya el driver necesario. Así, se podrá alcanzar la movilidad completamente autónoma, sin depender de equipos externos, aprovechando al máximo las capacidades del robot.

Capítulo 6

Conclusiones

En este capítulo se resumen los retos abordados, las soluciones implementadas y los experimentos realizados para validar el sistema, destacando la aportación del repositorio de referencia [2]. Finalmente, se plantean posibles líneas de trabajo futuro para seguir mejorando y ampliando el sistema.

6.1. Conclusiones

En primer lugar, la migración al sistema operativo GNU/Linux se ha realizado con éxito. El robot Nao V6 arranca ahora sobre Ubuntu 22.04, garantizando la compatibilidad completa con el firmware y todos los componentes (sensores, motores y LEDs) originales. Sobre esta base se ha implementado una plataforma mínima de ROS 2: se ha configurado el nodo `nao_lola_client` para conectar el bus LoLA interno con ROS 2, se ha generado y publicado el árbol de transformaciones (TF) del modelo URDF del Nao, se ha calibrado la cámara e integrado la IMU, los micrófonos y otros sensores en nodos estándar de ROS 2. Además, se ha implementado un filtro de Madgwick para fusionar la información del acelerómetro y giroscopio, permitiendo estimar la orientación y la odometría del robot bípedo. Gracias a esto, todos los sensores y actuadores del Nao quedan accesibles a través de ROS 2, cumpliendo así el objetivo de disponer de una plataforma base abierta y funcional sobre la que desarrollar nuevas capacidades.

A continuación, se abordaron las capacidades avanzadas de locomoción y gestos. Se adoptó el paquete `walk` de ROS 2 para el Nao V6, incluyendo un fork personalizado con mejoras. Se añadió sincronización de los brazos en fase opuesta a las piernas durante la marcha, aumentando la estabilidad y naturalidad del movimiento. También se diseñaron gestos predefinidos con el paquete `nao_pos`: por ejemplo, los gestos `hello` y `big` para mejorar la interacción con el Nao al conversar con él. Se han utilizado gestos como `stand` y `armsDown`, que se integran en la lógica de arranque y recuperación

de la marcha; de este modo, el robot siempre empieza a caminar desde una postura equilibrada. En resumen, el Nao puede caminar de forma controlada y configurable, y acompañar sus movimientos con gestos coordinados, cumpliendo así los objetivos de locomoción fluida y expresión corporal natural.

En cuanto a la interacción humano-robot, se integraron distintas tecnologías que enriquecen la comunicación. Se incluyó reconocimiento de voz en español (utilizando Whisper) y síntesis de voz (TTS), junto con un modelo de diálogo basado en GPT adaptado al contexto local. De esta forma, el robot es capaz de mantener conversaciones coherentes y personalizadas en español. A la vez, se utilizó percepción visual mediante detección y seguimiento de rostros con un detector YOLOv8, lo que permite al Nao dirigir su mirada hacia la persona que tiene delante.

La arquitectura también involucra un nodo coordinador (**ModeSwitcher**) que gestiona la interacción entre diálogo, gestos y locomoción, asegurando transiciones seguras (por ejemplo, deteniendo la marcha para hablar y retomándola al finalizar). Estas capacidades han sido validadas experimentalmente: el robot demostró mantener conversaciones fluidas incluso en entornos con ruido moderado, acompañando la voz con gestos y expresiones LED adecuados, confirmando así el objetivo de interacción humano-robot más completa.

Respecto a la navegación autónoma, se logró que el robot pudiera caminar de forma autónoma gracias a la integración de un LiDAR ligero (RPlidar S2) y la integración de la estimación de odometría para robots bípedos. Aunque, debido a limitaciones de hardware del Nao (puerto USB y controladores), el LiDAR se conectó al ordenador externo, el sistema de SLAM en ROS 2 funcionó correctamente, permitiendo generar mapas reales y precisos del entorno y navegar de manera autónoma. Estas pruebas demuestran que la integración es viable y funcional, y dejan la base preparada para que, en el futuro, instalando el controlador apropiado en el sistema operativo, tanto el cálculo de SLAM como la planificación puedan ejecutarse directamente en el robot, completando así un sistema de navegación completamente autónomo y sin depender de equipos externos.

En el ámbito de la simulación, se empleó Webots junto con el controlador **WebotsLoLaController** que emula el middleware LoLA. La simulación resultó fiel al robot real: tanto la cinemática como los datos de los sensores (cámaras simuladas, IMU,

FSR, etc.) presentan un comportamiento muy similar al hardware. Gracias a esto, los mismos nodos de ROS 2 pueden ejecutarse sin modificaciones en entornos virtual y real, facilitando la experimentación y validación continuas.

Finalmente, se trabajó en la documentación reproducible del proyecto. Se redactaron guías y scripts detallados que describen paso a paso el proceso de flasheo del Nao, la instalación de ROS 2, la configuración de los paquetes base y los procedimientos de prueba de las funcionalidades implementadas. Estas instrucciones han sido validadas con éxito por investigadores externos, confirmando su claridad y utilidad práctica. Como resultado, cualquier investigador interesado puede replicar correctamente el proceso de migración e instalación, eliminando la dispersión de la información [2].

En conjunto, cada objetivo específico planteado ha sido abordado: el Nao ha sido transformado en una plataforma abierta y ampliable que cumple los requisitos iniciales en locomoción, interacción y percepción. Esto abre el camino para futuros desarrollos en investigación, educación o asistencia mediante robótica social.

6.2. Competencias adquiridas

El Trabajo de Fin de Grado ha permitido desarrollar y consolidar un conjunto amplio de competencias del **Grado en Ingeniería de Robótica Software**. A continuación se enumeran las que se han puesto en práctica de forma más directa, indicando brevemente cómo se han evidenciado en el proyecto:

Competencias generales

- **CG01.** Conocimiento de materias básicas y tecnologías, utilizando Linux, ROS 2, Webots y herramientas de control de software, lo que demuestra versatilidad y capacidad de aprendizaje.
- **CG02.** Resolución de problemas con iniciativa, creatividad y toma de decisiones: integración hardware-software heterogénea (Nao, LiDAR, PC externo), ajustes de locomoción y fusión sensorial.
- **CG03.** Redacción, representación e interpretación de documentación técnica: elaboración de las guías de migración, README y manuales de uso reproducibles.
- **CG04.** Análisis y valoración del impacto social de la robótica: discusión de la interacción humano-robot y de la aceptación social del Nao como robot

asistencial.

- **CG05.** Planificación y gestión de proyectos: organización de hitos (flasheo, capa base ROS 2, HRI, navegación) y estimación de recursos de hardware y tiempo.
- **CG07.** Trabajo en equipos multidisciplinares y comunicación oral y escrita: colaboración con investigadores del laboratorio y redacción integral del *Trabajo de Fin de Grado* (TFG).

Competencias específicas

- **CE06.** Programación de aplicaciones robustas, eficientes y seguras: nodos ROS 2 en C++/Python con control de excepciones y gestión de concurrencia.
- **CE10.** Diseño y programación de sistemas en red: arquitectura distribuida Nao–PC usando DDS y *Quality of Service* (QoS) en ROS 2.
- **CE11.** Métodos de interconexión de elementos inteligentes en red: configuración de topics, servicios y acciones para sincronizar sensores, locomoción y HRI.
- **CE12.** Selección de sensorización y actuación: integración del LiDAR RPLidar S2 y ajuste de gestos/LEDs según requisitos del entorno.
- **CE15.** Desarrollo de aplicaciones robóticas sobre middlewares: implementación completa sobre ROS 2 Rolling con launchers modulares.
- **CE16.** Análisis y control de sistemas dinámicos: parametrización del paquete `walk` y ajuste de la cinemática inversa del Nao.
- **CE17.** Explotación de un sistema operativo: personalización de un kernel de *Linux* en *Ubuntu 22.04* para un sistema con recursos limitados.
- **CE18.** Programación de aplicaciones concurrentes: nodos ROS 2 multithread, gestión de callbacks y sincronización de sensores.
- **CE19.** Sistemas de localización, mapeado y navegación: implementación de SLAM Toolbox y Nav2 con fusión LiDAR–odometría en un robot bípedo.
- **CE21.** Aplicación de técnicas de IA: integración de GPT para diálogo y YOLOv8 para percepción visual orientada a interacción.
- **CE25.** Procesamiento de información 3-D: conversión de nubes de puntos a `LaserScan` y filtrado para navegación (prueba inicial con cámara de profundidad).

- **CE26.** Creación y uso de modelos en simulación: Webots y WebotsLoLaController para pruebas realistas.
- **CE28.** Diseño y construcción de robots móviles: adaptación del Nao como plataforma móvil autónoma con capacidad de navegación.
- **CE29.** Diseño de sistemas orientados a la interacción con personas: módulo HRI que combina voz, gestos y expresiones LED.
- **CE32.** Síntesis e integración de competencias en un proyecto profesional: el propio TFG, defendido ante tribunal universitario.

6.3. Trabajos futuros

A partir del estado actual del sistema, las posibilidades de mejora y ampliación son muy variadas. Como paso inmediato, una de las prioridades sería integrar completamente el LiDAR en el propio Nao. Esto incluye tanto la instalación del controlador `cp210x` en el kernel, lo que permitiría ejecutar todo el SLAM y la navegación de manera nativa en el robot, como la mejora del montaje físico del sensor para eliminar los problemas de estabilidad que he experimentado durante las pruebas.

Por otro lado, hay margen para seguir optimizando los algoritmos ya implementados. Un avance lógico sería perfeccionar aún más la odometría para robots bípedos, o incluso experimentar con alternativas basadas en visión usando las cámaras del propio Nao, lo que podría aumentar tanto la precisión como la robustez del sistema en entornos más complejos.

En cuanto a la marcha bípeda, una posible mejora interesante sería desarrollar una locomoción todavía más robusta y adaptable, capaz de enfrentarse a irregularidades del terreno, empujones o cambios bruscos en el entorno. Para ello, se podría explorar el uso de redes neuronales y técnicas de aprendizaje por refuerzo, permitiendo que el robot aprenda a recuperar el equilibrio y adaptarse en tiempo real a situaciones imprevistas. Esto permitiría llevar la autonomía del Nao a un nuevo nivel, haciéndolo más seguro y eficiente en aplicaciones reales y entornos dinámicos menos controlados.

En cuanto al HRI, mirando un poco más allá, el siguiente paso natural sería validar y probar todo el sistema en escenarios más realistas, como hogares, oficinas o incluso entornos públicos. Esto ayudaría a identificar retos adicionales como la latencia en las

comunicaciones, la calidad de la interacción o la necesidad de cumplir con normativas de seguridad y autonomía, especialmente relevantes en aplicaciones de robótica de servicio.

Otra línea muy prometedora es la integración del Nao con otros robots o dispositivos inteligentes, ya sea a través de sistemas domóticos o conectándolo a plataformas en la nube, lo que permitiría escenarios de colaboración e intercambio de información en tiempo real.

Finalmente, existe un enorme potencial para aplicar técnicas más avanzadas de inteligencia artificial, como modelos de diálogo más naturales o el aprendizaje por refuerzo para personalizar el comportamiento del robot en función de las preferencias o necesidades del usuario. Todo esto, además, abre la puerta a futuros proyectos en ámbitos educativos, sanitarios o de asistencia social.

En resumen, el trabajo realizado hasta ahora deja una base sólida para evolucionar el prototipo hacia una solución práctica y versátil. Queda margen de sobra para optimizar el rendimiento, validar el sistema en condiciones reales y adaptarlo a los requisitos de regulaciones y seguridad, necesarios para su utilización fuera del laboratorio.

Bibliografía

- [1] RobotLab, “Robot nao v6 edición estándar.” <https://www.robotlab.com/latam/store/robot-nao-para-la-educacion>, 2025. Consultado en 2025.
- [2] Antonio Roldán, “nao_ros2: Documentación completa para instalar Ubuntu y ROS 2 en el Nao.” https://github.com/geriabot/nao_ros2, 2025.
- [3] I. Asimov, *Robot Visions*. Penguin, 1990. Cita en la introducción del TFG.
- [4] Wikipedia contributors, “Robot Operating System — Wikipedia, The Free Encyclopedia.” https://en.wikipedia.org/wiki/Robot_Operating_System, 2025. Consultado el 22 de junio de 2025.
- [5] S. I. Behrens *et al.*, “Gendered robot voices and their influence on trust,” in *Companion of the 2018 ACM/IEEE Int. Conf. on Human-Robot Interaction*, pp. 63–64, 2018. Estudio sobre la influencia de la voz del robot en la confianza (HRI 2018).
- [6] Aldebaran Robotics, “Unveiling of NAO Evolution: A Stronger Robot and a More Comprehensive Operating System.” <https://web.archive.org/web/20150201191553/https://www.aldebaran.com/en/press/press-releases/unveiling-of-nao-evolution-a-stronger-robot-and-a-more-descriptioncomprehensive-operating>, 2014. Accedido el 28 de junio de 2025.
- [7] RoboCup Federation, “RoboCup: International Robot Competition and Research.” <https://www.robocup.org/>, 2025. Accedido el 28 de junio de 2025.
- [8] Engadget, “Nao robot replaces aibo in robocup standard platform league.” <https://www.engadget.com/2007-08-16-nao-robot-replaces-aibo-in-robocup-standard-platform-league.html>, August 2007. Consultado en 2025.

- [9] Wired, “Humanoid robot nao ready for prime time.” <https://www.wired.com/2008/11/humanoid-robot/>, November 2008. Consultado en 2025.
- [10] Wikipedia contributors, “Nao (robot) — Wikipedia, The Free Encyclopedia.” [https://en.wikipedia.org/wiki/Nao_\(robot\)#Specifications](https://en.wikipedia.org/wiki/Nao_(robot)#Specifications), 2024. Consultado el 22 de junio de 2025.
- [11] RoboCup SPL, “Robocup standard platform league.” <https://spl.robocup.org/>, 2024. Consultado en 2025.
- [12] S. Gee, “How is nao doing now!.” <https://www.i-programmer.info/news/169-robotics/15196-how-is-noa-doing-now.html>, 2022. Consultado en 2025.
- [13] C. Duque *et al.*, “Improving imitation skills in children with asd using robots,” *Frontiers in Robotics and AI*, 2025. Aceptado para publicación.
- [14] A. Krishna *et al.*, “Enhancing human-robot interaction in healthcare,” *Frontiers in Robotics and AI*, 2025. Aceptado para publicación.
- [15] Expansión.com, “Mitsubishi ufj incorpora robots a su plantilla para atender a los clientes.” <https://www.expansion.com/2015/01/14/empresas/banca/1421233991.html>, January 2015. Consultado en 2025.
- [16] PROVEN Robotics, “Nao robot uses: 7 surprising applications.” <https://provenrobotics.ai/nao-robot-uses/>, 2024. Consultado en 2025.
- [17] Cyberbotics Ltd., “Webots: Open Source Robot Simulator.” <https://cyberbotics.com/>, 2024.
- [18] The Robot Report, “Aldebaran put in receivership.” <https://www.therobotreport.com/aldebaran-pepper-nao-robots-receivership/>, 2025. Consultado en 2025.
- [19] Eric S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly Media, 1999.
- [20] Antonio Bono and Kenji Brameld and Luca D'Alfonso and Giovanni Fedele, “Open Access NAO (OAN): a ROS2-based software framework for HRI applications with the NAO robot,” in *IEEE RO-MAN*, 2024.
- [21] Aldebaran Robotics, “Nao v6 datasheet,” tech. rep., Aldebaran Robotics, 2018. Consultado en 2025.

- [22] NaoDevils, “NaoImage: Ubuntu for NAO V6.” <https://github.com/NaoDevils/NaoImage>, 2023.
- [23] Open Robotics, “Robot Operating System 2 (ROS 2 Rolling) Wiki.” <https://docs.ros.org/en/rolling/index.html>, 2025.
- [24] Bembelbots, “WebotsLoLaController: Webots controller for Nao V6 LoLA interface.” <https://github.com/Bembelbots/WebotsLoLaController>, 2023. Consultado en junio de 2025.
- [25] Antonio Roldán, “nao_lola (fork personalizado).” https://github.com/andoniroldan/nao_lola.git, 2025.
- [26] ROS Sports, “nao_lola.” https://github.com/ros-sports/nao_lola, 2025.
- [27] Antonio Bono, “nao_led.” https://github.com/antbono/nao_led.git, 2024.
- [28] Antonio Roldán, “nao_pos (fork personalizado).” https://github.com/andoniroldan/nao_pos.git, 2025.
- [29] Antonio Roldán, “walk (fork personalizado).” <https://github.com/andoniroldan/walk.git>, 2025.
- [30] Kenji Brameld, “walk.” <https://github.com/ijnek/walk.git>, 2024.
- [31] Kenji Brameld, “nao_ik.” https://github.com/ijnek/nao_ik.git, 2024.
- [32] Kenji Brameld, “nao_phase_provider.” https://github.com/ijnek/nao_phase_provider.git, 2024.
- [33] ROS device drivers, “usb_cam.” https://github.com/ros-drivers/usb_cam, 2025.
- [34] Roland Kersten, “audio_common para ROS 2.” https://github.com/rolker/audio_common.git, 2024.
- [35] Antonio Roldán, “nao_hri (fork personalizado).” https://github.com/geriabot/nao_hri.git, 2024.
- [36] Ultralytics, “YOLOv8: State-of-the-Art Real-Time Object Detection.” <https://yolov8.com/>, 2025. Accedido el 28 de junio de 2025.

- [37] ROS Developers, “robot_state_publisher: Ros package.” https://github.com/ros/robot_state_publisher, 2025. Repositorio oficial en GitHub. Consultado en junio de 2025.
- [38] Kenji Brameld, “nao.” <https://github.com/ijnek/nao.git>, 2025.
- [39] ros-naoqi, “nao_meshes2.” https://github.com/ros-naoqi/nao_meshes2.git, 2024.
- [40] Sebastian Madgwick, “Madgwick IMU Filter.” https://wiki.ros.org imu_filter_madgwick, 2022.
- [41] Nav2 Team, “Navigation 2.” <https://github.com/ros-planning/navigation2>, 2024.
- [42] Steve Macenski, “slam_toolbox.” https://github.com/SteveMacenski/slam_toolbox, 2024.
- [43] ROS 2 Authors, “robot_state_publisher: publishes link transforms from joint states.” https://github.com/ros/robot_state_publisher. Consultado en junio de 2025.
- [44] ROS 2 Documentation, “Using URDF with robot_state_publisher.” <https://docs.ros.org/en/foxy/Tutorials/Intermediate/URDF/Using-URDF-with-Robot-State-Publisher.html>. Consultado en junio de 2025.
- [45] Open Source Robotics Foundation, “RViz: 3D Visualization Tool for ROS.” <https://wiki.ros.org/rviz>, 2025. Consultado en junio de 2025.
- [46] OpenCV Team, “OpenCV: Open Source Computer Vision Library.” <https://opencv.org/>, 2025. Consultado en junio de 2025.
- [47] OpenCV Team, “Camera calibration With OpenCV - Square Chessboard Tutorial.” https://docs.opencv.org/4.x/dc/d43/tutorial_camera_calibration_square_chess.html, 2025. Consultado en junio de 2025.
- [48] Lao Tzu, “The journey of a thousand miles begins with a single step.” Proverbio tradicional chino. Traducción común al inglés., c. 6th century BCE. Consultado en junio de 2025.
- [49] Antonio Bono, “hni.” <https://github.com/antbono/hni>, 2023.

- [50] Antonio Bono, “nao_pos.” https://github.com/antbono/nao_pos.git, 2025.
- [51] OpenAI, “Whisper: Robust Speech Recognition via Large-Scale Weak Supervision.” <https://openai.com/research/whisper>, 2022. Consultado en junio de 2025.
- [52] Wikipedia contributors, “CUDA.” <https://es.wikipedia.org/wiki/CUDA>, 2025. Consultado: 30 de junio de 2025.
- [53] Andoni Roldan, “Fork de WebotsLoLaController con publicación de cámara en ROS2.” <https://github.com/andoniroldan/WebotsLoLaController>, 2025. Consultado en junio de 2025.
- [54] ijnek, “nao_description: URDF model of Nao for ROS 2.” https://github.com/ijnek/nao_description. Consultado en junio de 2025.