Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Desarrollo de sistemas de interacción para el robot NAO

Autor:

Zamorano Casasola, Álvaro

Tutor:

Zalama Casanova, Eduardo
Departamento de Ingeniería de Sistemas y Automática

Valladolid, Julio 2024.

RESUMEN

El presente proyecto explora el desarrollo de sistemas de interacción para el robot NAO, enfocándose principalmente en la implementación de una función de diálogo de voz para una experiencia conversacional fluida, similar a *ChatGPT*.

Aprovechando los recientes avances en modelos de lenguaje impulsados por inteligencia artificial, se pretende programar un robot social con capacidades interactivas diseñado para proporcionar compañía y entretenimiento a las personas mayores en sus hogares, a través del diálogo y otras actividades como ejercicios físicos guiados.

Se realiza una familiarización inicial con las herramientas de desarrollo existentes para el robot NAO así como de la API de *OpenAI* para la integración del modelo de lenguaje (LLM). Además de la mencionada funcionalidad de diálogo, se desarrollan también otra serie de comportamientos para el robot que se enlazarán con el modelo GPT permitiendo su directa ejecución a través del asistente de voz.

Palabras Clave: Robot NAO, ChatGPT, Robótica Social, LLMs, Sistemas de Diálogo

ABSTRACT

This Project explores the development of interaction systems for the NAO robot, with a primary focus on implementing a voice-to-voice dialog functionality for a seamless conversational experience, akin to *ChatGPT*.

Leveraging recent advancements in AI-driven language models, the aim is to program a social robot with interactive capabilities designed to provide companionship and entertainment to elderly individuals in their homes through dialogue and other activities such as guided physical exercises.

An initial familiarization with the existing development tools for the NAO robot as well as the *OpenAI* API for language model (LLM) integration is conducted. In addition to the aforementioned dialogue functionality, a series of other behaviours for the robot are also developed, which will be linked to the GPT model enabling their direct execution through the voice assistant.

Keywords: NAO Robot, ChatGPT, Social Robotics, LLMs, Dialog Systems

ÍNDICE DE CONTENIDO

Capítulo 1: Introducción y Objetivos 1

1.1 Contexto 1

1.2 Motivación 2

1.3 Objetivos 3

1.4 Principales Tecnologías Utilizadas 5

1.5 Estructura del Documento 6

Capítulo 2: Antecedentes 8

2.1 Robots Sociales 8

2.1.1 Introducción histórica de la robótica 8

2.1.2 Acotando la definición de robot social 10

2.1.3 Aplicaciones y Efectividad de los robots sociales 14

2.1.4 Ejemplos de Robots Sociales 18

2.2 Inteligencia Artificial y Modelos de Lenguaje 23

2.2.1 Introducción 24

2.2.2 La arquitectura del Transformador 29

2.2.3 Modelos Generativos Pre-entrenados (GPT) 34

2.3 Proyectos Recientes / Estado del Arte 38

Capítulo 3: Herramientas de Desarrollo 41

3.1 Ecosistema del robot 41

3.1.1 Robot NAO - Hardware 41

3.1.2 Robot Settings 44

3.1.3 Choregraphe 45

3.1.4 NAOqi Python SDK 45

3.2 API de OpenAI 47

3.3 Otras Herramientas 50

3.3.1 Visual Studio Code 50

3.3.2 Obsidian 50

3.3.3 Protocolo MQTT y MQTT Explorer 51

Capítulo 4: Desarrollo de los Sistemas de Interacción 57

4.1 Configuración Inicial 57

4.1.1 Ajustes del Robot 57

4.1.2 Instalación de las herramientas de software 60

4.1.3 Broker MQTT 62

4.2 Familiarización con el robot 63
4.2.1 Interacción con el robot 63
4.2.2 Autonomous Life 65
4.2.3 Desarrollo con NAOqi 66
4.3 Desarrollo en Choregraphe 70
4.3.1 Desarrollo de las Rutinas de Ejercicios 74
4.4 Desarrollo del Chatbot GPT 81
4.5 Integración en el robot 90
4.5.1 Módulo de reconocimiento de voz (STT) 92
4.5.2 Comunicación entre procesos 95
Capítulo 5: Resultados y Valoración 98
5.1 Valoración General 98
5.2 Estudio de Aceptación 100
Capítulo 6: Estudio Económico 105
6.1 Introducción 105
6.2 Planificación del proyecto 105
6.3 Recursos Empleados 107
6.4 Costes Directos 107
6.4.1 Coste del Personal 108
6.4.2 Coste de amortización de equipos y programas 109
6.4.3 Coste de materiales y servicios 111
6.4.4 Costes directos totales 111
6.5 Costes Indirectos 112
6.6 Costes Totales 112
Capítulo 7: Conclusiones y Líneas Futuras 114
7.1 Conclusiones generales 114
7.2 Conclusiones sobre los objetivos planteados 115
7.3 Líneas Futuras 117
7.3.1 Experimentación con diferentes modelos GPT 117
7.3.2 Exploración de modelos alternativos texto-a-texto 117
7.3.3 Integración de Modelos Multimodales 118
Bibliografía 120

ÍNDICE DE FIGURAS

Figura 1: Robot NAO 5
Figura 2: Generación de texto con la API de OpenAI 5
Figura 3: Robot Elmer / Elsie 9
Figura 4: Robot Unimate 9
Figura 5: Robot IRB 6 10
Figura 6: Robot PUMA 10
Figura 7: Clasificación de los robots 11
Figura 8: Robots de compañía AIBO (izq.) Paro (centro) y EMO (dcha.) 12
Figura 9: Robot Temi 12
Figura 10: Modelo de un Robot Social [11] 14
Figura 11: Robot Relay de "Relay Robotics" 15
Figura 12: Funcionalidades del robot Temi del proyecto EIAROB 16
Figura 13: Robot NAO interactuando con un humano 17
Figura 14: Robot PARO 18
Figura 15: Robot Jibo 19
Figura 16: Robot Pepper 20
Figura 17: Robot Ameca 21
Figura 18: Robots Jia Jia (izq.) Sophia (centro) y Nadine (dcha.) 21
Figura 19: Uncanny Valley [29] 22
Figura 20: Algunos participantes de la conferencia de Dartmouth 24
Figura 21: Línea temporal de los Inviernos de IA [33] 25
Figura 22: Modelo de perceptrón 26
Figura 23: Arquitectura de una red neuronal profunda (DNN) 27
Figura 24: Arquitectura original de un Transformador [37] 29
Figura 25: Ejemplo utilizando el tokenizer de OpenAI [39] 30
Figura 26: Representación de “Word Embeddings” [40] 31
Figura 27: Matriz de atención - mapa de calor 32
Figura 28: Visualización de cross-attention [42] 33
Figura 29: Visualización de self-attention [43] 33
Figura 30: Visualización, predicción de la siguiente palabra del texto [40] 35
Figura 31: Resumen del proceso de entrenamiento de un GPT [44] 36
Figura 32: Chatbot Arena Leaderboard [45] 37
Figura 33: Spot 4.0 utilizando Reinforcement Learning [46] 38
Figura 34: Arquitectura del robot Figure 01 [49] 39
Figura 35: Articulaciones y Sensores del robot NAO 42
Figura 36: Sensores de ultrasonido (izq.) y micrófonos (dcha.) 43
Figura 37: Vista principal de Robot Settings 44
Figura 38: Vista principal de Choregraphe 45
Figura 39: Distintos SDKs disponibles y sus características 46
Figura 40: Estructura del framework NAOqi [57] 46
Figura 41: Modelos de la API de OpenAI 47
Figura 42: Esquema de la API de Asistentes 48
Figura 43: Direcciones HTTP de cada endpoint 49
Figura 44: Obsidian - Graph View 50

Figura 45: MQTT en IoT 51
Figura 46: Clientes MQTT publicadores y suscriptores 53
Figura 47: Protocolo MQTT sobre TCP/IP 54
Figura 48: MQTT Explorer 55
Figura 49: Robot Settings - Factory Reset 58
Figura 50: Robot Settings - Lista de robots 59
Figura 51: Robot Settings - Speak 59
Figura 52: Choregraphe - Conectarse a un Robot 59
Figura 53: Extracto del Quickstart Tutorial (API de OpenAI) [68] 60
Figura 54: Configuración de los alias de Python 61
Figura 55: Script de comprobación - NAOqi 61
Figura 56: Script de comprobación - OpenAI API 62
Figura 57: Ojos de NAO cuando detecta un humano 63
Figura 58: Feedback de la conversación con NAO [69] 64
Figura 59: Diagrama de funcionamiento de Autonomous Life 65
Figura 60: Autonomous Abilities [72] 66
Figura 61: Objeto ALProxy 66
Figura 62: Ejemplo de uso ALProxy 67
Figura 63: Obtener una qi.Session de un Proxy 67
Figura 64: Ejemplo qi Framework 68
Figura 65: Script voice_params.py 69
Figura 66: Tipos de señales en Choregraphe 70
Figura 67: Bloque "Ptyhon Script" 71
Figura 68: Bloque Apply Posture 72
Figura 69: Ejemplo Choregraphe 1 72
Figura 70: Ejemplo Choregraphe 2 73
Figura 71: Bloque "Timeline" 74
Figura 72: Keyframes en la capa de movimiento 75
Figura 73: Diagramas como keyframes en la capa de comportamiento 75
Figura 74: Panel "Timeline Editor" con curvas de interpolación 76
Figura 75: Diagrama Insert Data - número de repeticiones 76
Figura 76: Diagrama general - ejercicios de manos 77
Figura 77: Sub-diagrama - Ejercicio 1 de manos 78
Figura 78: Script del bloque "Insert Data" 79
Figura 79: Script del bloque "Custom Counter" 79
Figura 80: Interior del bloque Timeline "hands0_start" 80
Figura 81: Script de validación de la API de OpenAI 81
Figura 82: Ejemplo de llamada a Chat Completions API 82
Figura 83: Tarifas de la API de OpenAI [87] 83
Figura 84: Formato de la petición a la API 84
Figura 85: Formato del objeto de respuesta - chat.completion 85
Figura 86: Código de la clase "Conversation" 86
Figura 87: Ejemplo de conversación coloreada 87
Figura 88: Función "chat_completion_request" 88
Figura 89: Diagrama lógico de los procesos 90
Figura 90: Ejemplo de los módulos TTS de NAOqi 91

Figura 91: Módulo SpeechRecognitionModule 92
Figura 92: Proceso STT – Suscripción al evento 93
Figura 93: Proceso TTS - Funciones callback 94
Figura 94: Proceso TTS - Calibración del módulo de reconocimiento de voz 94
Figura 95: Asignación del callback específico para mensajes de tipo "speech" 95
Figura 96: Diagrama de comunicación entre procesos 96
Figura 97: Relación entre constructos del modelo ALMERE [7] 101
Figura 98: Resultados del modelo ALMERE 102
Figura 99: Planificación del Proyecto 105
Figura 100: Modelos Gemini de Google 118
Figura 101: Modelos Unimodales vs Multimodales [92] 118

ÍNDICE DE TABLAS

Tabla 1: Constructos del modelo ALMERE 101
Tabla 2: Distribución temporal del trabajo 106
Tabla 3: Coste anual del personal 108
Tabla 4: Días efectivos por año 108
Tabla 5: Amortización de equipos 110
Tabla 6: Coste de materiales y servicios 111
Tabla 7: Costes indirectos 112
Tabla 8: Costes totales 112

GLOSARIO

Debido a la novedad y el rápido desarrollo de algunas de las tecnologías que se discuten en el proyecto/trabajo, principalmente en lo tocante a inteligencia artificial, no existe una convención firme en cuanto a la traducción de algunos términos provenientes del inglés. Por esa razón se incluye el presente glosario con el propósito de esclarecer la terminología y así poder utilizar anglicismos el cuerpo del texto para designar inequívocamente un concepto o tecnología.

API (Application Programming Interface): una API o interfaz de programación de aplicaciones es un conjunto de reglas y protocolos que permiten que diferentes software se comuniquen entre sí. Proporciona una interfaz clara y definida para la interacción entre componentes de software permitiendo la creación de aplicaciones que utilizan funcionalidades de servicios externos de manera estandarizada.

LLM (Large Language Model): los modelos de lenguaje de gran tamaño son modelos generativos de IA basados en aprendizaje profundo (redes neuronales) capaces de generar texto. Los modelos, entrenados con grandes cantidades de texto, son capaces de aprender patrones complejos a partir de los datos de entrenamiento por lo que pueden ser utilizados para tareas como la traducción automática, redacción de texto y respuesta a preguntas, entre otros. Un ejemplo de LLM son los modelos GTP de OpenAI.

GPT (Generative Pre-trained Transformer): grupo de modelos de inteligencia artificial desarrollados por OpenAI. Es un ejemplo de LLM, utilizado para generar texto con capacidades similares a las de un humano. Dentro de la familia de modelos GPT encontramos varias versiones siendo GPT-4 la más reciente.

ChatGPT: servicio online proporcionado por OpenAI a través del cual se pueden utilizar modelos GPT mediante una interfaz web más cómoda y orientada a un usuario menos técnico que la API. La variante gratuita da acceso únicamente al modelo GPT-3.5 mientras que la variante de pago (ChatGPT Plus) permite utilizar GPT-4 y otras funcionalidades extra como DALL-E y personalizar tus propios GPTs.

SDK (Software Development Kit): colección de librerías y herramientas de software que ayudan a los desarrolladores a crear programas o aplicaciones para una plataforma específica ya sea un *framework* un hardware específico, un sistema operativo o similar.

NLP (Natural Language Processing): En español, procesamiento del lenguaje natural. Es un campo de la inteligencia artificial que se enfoca en la interacción entre computadoras y el lenguaje humano. NLP abarca una variedad de tareas, incluyendo la comprensión, generación, y traducción de texto, así como la extracción de información y la conversación automatizada, permitiendo que las máquinas interpreten y respondan a datos lingüísticos de manera similar a como lo hacen los humanos.

Machine Learning (Aprendizaje Automático): Es una subdisciplina de la inteligencia artificial que se centra en el desarrollo de algoritmos y modelos que permiten a las computadoras aprender de los datos y mejorar su rendimiento en tareas específicas sin ser explícitamente programadas para cada tarea. Utiliza técnicas estadísticas y matemáticas para identificar patrones en los datos, y sus aplicaciones incluyen desde la predicción y clasificación hasta el reconocimiento de imágenes y la recomendación de contenido.

Deep Learning (Aprendizaje Profundo): Es una subrama del aprendizaje automático que utiliza redes neuronales artificiales con múltiples capas (profundas) para modelar y entender patrones complejos en datos. Esta técnica es especialmente eficaz para el procesamiento de grandes volúmenes de datos no estructurados, como imágenes, audio y texto. Las diferentes arquitecturas CNN, RNN, Transformers… han impulsado avances significativos en áreas como la visión por computadora y el procesamiento de lenguaje natural.

Reinforcement Learning (Aprendizaje por Refuerzo): Es un paradigma del aprendizaje automático en el que un agente aprende a tomar decisiones a través de la interacción con un entorno, obteniendo recompensas o castigos en función de sus acciones. El objetivo es maximizar una función de recompensa acumulativa a largo plazo, ajustando su comportamiento mediante la exploración y explotación de estrategias. Este enfoque se utiliza en áreas como la robótica, los videojuegos y la optimización de sistemas.

Reinforcement Learning with Human Feedback (RLHF): El Aprendizaje por Refuerzo con Retroalimentación Humana, es una técnica en la que se combina el aprendizaje por refuerzo (RL) con la retroalimentación proporcionada por humanos para mejorar el comportamiento de un modelo. Los humanos guían el proceso de aprendizaje dando retroalimentación directa sobre las acciones del modelo, ayudando a refinar y mejorar su desempeño en tareas complejas que pueden ser difíciles de evaluar solo con recompensas automáticas.

Context Window (Ventana de Contexto): En modelos de lenguaje y procesamiento de secuencias, es el rango de texto que el modelo puede considerar simultáneamente para generar respuestas o realizar predicciones. Esta ventana define la cantidad de datos previos o posteriores que el modelo puede usar para comprender el contexto de una palabra o frase en una secuencia. Su tamaño afecta la capacidad del modelo para capturar relaciones a largo plazo en el texto y puede influir en la calidad de las predicciones o respuestas generadas.

Edge Computing (Computación en el Borde): Es un paradigma de computación que lleva el procesamiento de datos y los servicios de computación más cerca de los dispositivos o fuentes de datos (el "borde" de la red), en lugar de depender completamente de servidores centrales o la nube. Esto reduce la latencia, mejora la eficiencia del uso del ancho de banda, y permite una respuesta más rápida para aplicaciones sensibles al tiempo, como IoT, realidad aumentada, y control en tiempo real de dispositivos.

Capítulo 1: Introducción y Objetivos

1.1 Contexto

La robótica social surge como un campo multidisciplinar que fusiona la robótica con otras disciplinas como el procesamiento de lenguaje natural y más recientemente la inteligencia artificial, para crear robots diseñados para interactuar y colaborar activamente con los seres humanos en entornos sociales. Este campo busca desarrollar robots capaces de comprender y adaptarse al comportamiento humano, con el objetivo de mejorar la calidad de vida y proporcionar apoyo en diversas áreas.

Inicialmente, los robots sociales se centraban en tareas específicas y limitadas como la navegación autónoma en entornos controlados. Sin embargo, con los avances en inteligencia artificial y percepción robótica, se han desarrollado robots más sofisticados capaces de interactuar de manera natural con humanos, utilizando lenguaje natural, gestos y expresiones faciales.

En la actualidad, las principales aplicaciones de la robótica social incluyen robots de compañía para personas mayores, asistentes de atención médica, educativos y de entretenimiento. Estos robots están diseñados para proporcionar apoyo emocional, asistencia física y estimulación cognitiva, contribuyendo así a mejorar la calidad de vida de las personas y fomentar su inclusión social.

Descrito el ámbito en el que se enmarca el proyecto, conviene destacar el principal reto al que se enfrenta el desarrollo en robótica social: la naturaleza de las interacciones humanas. Como bien describió Aristóteles, el ser humano es un animal social y político (*Zoon Politikón*) [1] con una avanzada capacidad para relacionarse y organizar una vida en sociedad. Este componente social es algo que damos por sentado, inherente a nosotros y a la forma en la que experimentamos la vida, pero a la vez difícil de replicar en los sistemas artificiales que nosotros mismos desarrollamos.

La complejidad de entender el contexto de una interacción social radica en la multiplicidad de factores que influyen en esta. Desde la entonación de la voz hasta los gestos corporales, pasando por el entorno físico y el historial previo de la interacción, el robot debe ser capaz de interpretar una gran cantidad de información para responder de manera adecuada y natural. Dicha complejidad se ve exacerbada por la dificultad para integrar y enlazar estas diversas fuentes de información a través de la sensorización correspondiente. La capacidad de

generar una descripción precisa y completa del contexto de la interacción social es crucial para que el robot pueda operar de manera efectiva y obtener una reacción positiva por parte del usuario humano con el que interacciona.

Asimismo, la dificultad para abarcar todas las posibles interacciones o comportamientos esperados de un robot humanoide es un desafío constante. A medida que se expanden las capacidades del robot y se diversifican sus funciones, resulta cada vez más difícil evitar que la experiencia se torne repetitiva o predecible para el usuario. Es necesario diseñar sistemas flexibles y adaptativos que puedan aprender y evolucionar con el tiempo, incorporando nuevas interacciones y comportamientos de manera dinámica. Este enfoque permitirá que el robot pueda ofrecer experiencias más versátiles y enriquecedoras, personalizadas a sus usuarios, mejorando así su aceptación por parte de los mismos y su utilidad en diversos contextos sociales.

Dentro del contexto previamente descrito, el proyecto se centra en la interacción mediante un agente conversacional capaz de entablar un diálogo verbal (*speech to speech*) con el usuario. Este agente inteligente no solo se encargará de comprender y generar respuestas coherentes en tiempo real, sino que también estará integrado con el robot NAO lo que le permitirá controlar ciertas funciones del mismo. Esta integración busca crear una experiencia conversacional fluida y natural para el usuario donde el robot pueda responder de manera adecuada a las solicitudes y preguntas del usuario así como realizar acciones físicas y proporcionar información relevante según el contexto de la conversación.

1.2 Motivación

La motivación detrás de este proyecto surge de la necesidad de mejorar la interacción entre humanos y robots en el ámbito de la robótica social. Tradicionalmente, los agentes conversacionales se programaban mediante reglas estrictas y relaciones fijas pregunta-respuesta, utilizando diagramas de flujo predeterminados. Ejemplos de estas metodologías incluyen sistemas como DialogFlow [2] de Google y otros similares. Sin embargo, esta aproximación limitaba la flexibilidad y la capacidad de adaptación de los agentes a las diversas situaciones y contextos de conversación.

La llegada de los nuevos modelos de lenguaje (LLMs), especialmente los modelos GPT [3] de OpenAI, ha revolucionado el campo de las conversaciones artificiales. Estos modelos son capaces de generar respuestas fluidas y variadas, comprender el contexto de la conversación y ofrecer información útil de manera más natural. Independientemente del nombre asociado a esta tecnología, estos nuevos modelos proporcionan una verdadera sensación de "inteligencia" lo que mejora significativamente la experiencia de interacción

con sistemas robóticos de propósito social. La capacidad de estos modelos para generar respuestas coherentes y relevantes, así como su habilidad para adaptarse a diferentes estilos de conversación, procura una sólida base para el desarrollo de sistemas de interacción más avanzados y efectivos en el ámbito de la robótica social.

La oportunidad de realizar este Trabajo de Fin de Grado (TFG) surge en un momento crucial coincidiendo con los recientes avances en inteligencia artificial. Estos avances son de especial relevancia en el campo de la robótica y en particular en la robótica social, aprovechándose los modelos de lenguaje de gran tamaño (LLMs) para su implementación en forma de agentes conversacionales y sistemas de diálogo en los robots. Todo ello combinado con la plataforma proporcionada por el robot NAO conforma el contexto propicio para explorar nuevas formas de utilizar la tecnología para mejorar la asistencia social a través de la interacción humano-robot.

El presente trabajo se enmarca dentro del proyecto EIAROB [4], en el que colabora la Universidad de Valladolid, que busca desarrollar un sistema de inteligencia artificial para el apoyo a los cuidados de personas mayores y dependientes en sus hogares. Una parte del proyecto recae en el uso de robots sociales que puedan proporcionar compañía, entretenimiento y asistencia a los usuarios, mejorando así su calidad de vida y bienestar emocional.

Existen diversas publicaciones que destacan los beneficios de la interacción con robots sociales en ámbitos como la educación, el tratamiento de pacientes con deterioro cognitivo y la asistencia a personas ancianas para combatir la soledad (ver apartado: *Aplicaciones y Efectividad de los robots sociales*). De hecho, en un reciente evento [5] en el que se presentaban los avances del proyecto EIAROB, varios de los voluntarios que colaboran probando los robots sociales mencionaban el gran beneficio que les ha supuesto el simple hecho de estar activo, formar parte del proyecto y sentirse útil, precisamente lo que se busca proporcionar mediante la interacción con robots sociales.

1.3 Objetivos

El objetivo principal del proyecto es el desarrollo de un sistema de diálogo inteligente para el robot NAO de *SoftBank Robotics* utilizando los avances en inteligencia artificial que proporcionan los modelos GPT desarrollados por la empresa *OpenAI*. Se desarrollará un agente conversacional *voz-a-voz* de modo que el usuario pueda hablar con el robot y este contestarle, todo ello de forma verbal, con el propósito de aumentar las capacidades interactivas nativas del robot.

El componente fundamental del agente de diálogo será uno de los modelos GPT proporcionados por la API de *OpenAI*. Además, se aprovecharán otras herramientas tanto externas como nativas del robot para realizar la integración.

No hay que olvidar que el propósito final de este sistema, enmarcado dentro de la robótica social, es el de interaccionar con personas mayores para proveer una experiencia natural y cercana, proporcionando compañía y contribuyendo así a su bienestar emocional. Se analizará por tanto el sistema obtenido, evaluando factores como la fluidez y variedad de la comunicación o la naturalidad de los movimientos y voz del robot, en resumen, una opinión general de la experiencia interactiva por parte de usuarios de prueba.

Partiendo de los objetivos principales mencionados, surgen los siguientes objetivos secundarios como hitos intermedios o pasos a realizar:

- Analizar las características del robot NAO como plataforma física, y las herramientas de desarrollo de software asociadas para comprender las posibilidades que ofrece de cara los objetivos propuestos.

- Realizar un estudio genérico de la reciente evolución en el campo de la inteligencia artificial, así como del estado del arte en lo tocante a grandes modelos de lenguaje (LLMs) como los que se van a utilizar.

- Familiarizarse con los servicios ofrecidos por la API de *OpenAI*, cómo funcionan y cómo programar su integración en otros sistemas.

- Desarrollar el agente conversacional enlazando un sistema de reconocimiento de voz (*speech-to-text, STT*), un modelo GPT como *chatbot* de texto y un servicio de texto-a-voz (*text-to-speech, TTS*) que genere el audio de las respuestas del robot.

- Explorar las capacidades físicas del robot, su sensorización y movimiento. Como demostración, desarrollar la coreografía de una rutina de ejercicios de modo que el robot pueda servir de guía y que el usuario lo imite.

- Integrar la componente "inteligente" del robot con alguna de las capacidades de movimiento de forma que el modelo GPT tenga acceso y pueda lanzar comportamientos del robot a discreción.

1.4 Principales Tecnologías Utilizadas

A continuación, se introducen brevemente los dos elementos principales que forman parte del presente proyecto: el robot humanoide NAO y los modelos GPT de OpenAI.

El Robot NAO

El robot NAO (*figura 1)* es un robot humanoide diseñado por la empresa SoftBank Robotics. Destaca por su tamaño compacto y su estructura articulada con 25 grados de libertad que le permiten una amplia gama de movimientos y gestos. Está equipado con cámaras, micrófonos, altavoces, sensores táctiles y de ultrasonido lo que le permite interactuar con su entorno y los usuarios. NAO cuenta con una plataforma de desarrollo integrada que permite programar nuevos comportamientos y aplicaciones utilizando lenguajes como Python, C++ y MATLAB entre otros, amén de otras herramientas de desarrollo específicas como *Choregraphe*.

Figura 1: Robot NAO

La API de OpenAI

La API (*Application Programming Interface*) de OpenAI (*figura 2*) proporciona acceso a potentes modelos de inteligencia artificial capaces de generar texto, traducir texto, transcribir audio o generar imágenes. Destaca por sus avanzados modelos de lenguaje conocidos como GPT (*Generative Pre-trained Transformers*) responsables de servicios como *ChatGPT*.

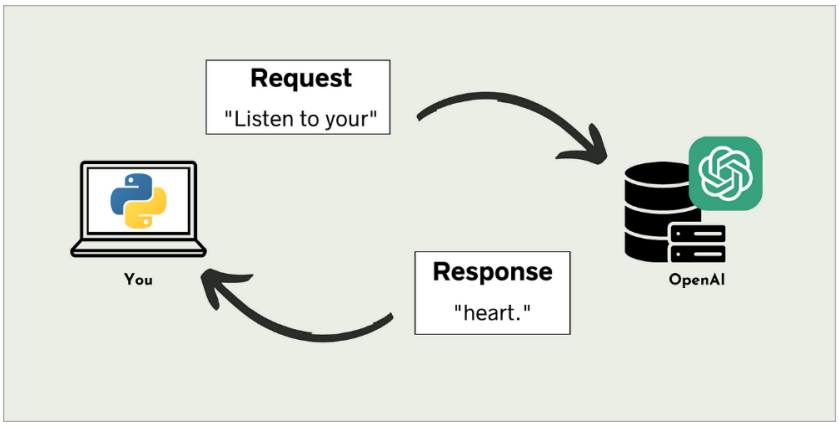


Figura 2: Generación de texto con la API de OpenAI

1.5 Estructura del Documento

El presente trabajo se estructura en los siguientes capítulos:

El primer capítulo contiene una breve introducción al campo de la robótica social dentro del cual se enmarca este Trabajo de Fin de Grado. Se describe también la motivación del proyecto, sus objetivos y las principales tecnologías utilizadas.

En el segundo capítulo se realiza una introducción a los ámbitos en los que se basa el proyecto y su estado del arte actual. Estamos hablando tanto de la robótica social como de las ramas pertinentes del campo de la inteligencia artificial y los modelos de lenguaje que se derivan de éstas. Se enumeran también algunos de los proyectos más recientes y punteros que combinan robótica e inteligencia artificial.

En el tercer capítulo se describen las herramientas utilizadas en el desarrollo del proyecto, centrándose principalmente en el ecosistema del robot NAO y las funciones proporcionadas por la API de OpenAI. Se mencionan asimismo otras herramientas y tecnologías auxiliares utilizadas.

En el cuarto capítulo se desarrolla el cuerpo central del trabajo, empezando con la familiarización de las herramientas a utilizar, pasando por la elaboración y programación del agente conversacional y finalizando con la integración de todos los componentes en el robot. Se detallan elementos como el uso de las librerías del SDK del robot y la API de OpenAI, el proceso de comunicación entre el ordenador y el robot, la arquitectura diseñada para el sistema, el planteamiento de los turnos de conversación y la interacción humano-robot.

En el quinto capítulo se comentan los resultados obtenidos mencionando posibles métodos de análisis futuro y se realiza una valoración general del trabajo desarrollado.

El sexto capítulo consiste en un estudio económico del proyecto. Se valorará el tiempo dedicado al desarrollo del mismo, así como los costes en personal recursos y herramientas con el fin de analizar la viabilidad económica de cara a una posible implementación.

En el séptimo capítulo se exponen las conclusiones finales del proyecto en referencia a los objetivos propuestos y se discuten posibles líneas de trabajo futuras.

Para concluir, se adjunta la bibliografía consultada.

Capítulo 2: Antecedentes

El objetivo de este capítulo es proporcionar contexto y realizar una breve introducción a los campos de estudio relacionados con el presente trabajo.

Primeramente, se introducirá el campo de la robótica social, su propósito y estado actual. Se proporcionan también algunos ejemplos de robots sociales tanto pasados como actuales para ilustrar la evolución de esta rama de la robótica.

El siguiente subapartado trata los nuevos avances en el campo de la inteligencia artificial y en particular aquellos más beneficiosos para su integración en robots sociales, los grandes modelos de lenguaje (LLMs) y la IA generativa.

Para finalizar se mencionan algunos proyectos recientes que se ubican a la vanguardia de la investigación en este campo, combinando la robótica y la inteligencia artificial no solo para funcionalidades de diálogo sino también de lógica (toma de decisiones) y control motriz (ajuste de los modelos de control).

2.1 Robots Sociales

2.1.1 Introducción histórica de la robótica

Antes de que existieran las palabras “robot” o “robótica” se denominaba autómatas a todos aquellos artefactos o dispositivos diseñados con el propósito de descargar o relevar al ser humano en la realización de trabajos tediosos, repetitivos y/o peligrosos, lo que hoy en día conocemos como *automatizar* una tarea. [6]

El término *robot* surge por primera vez en la obra R.U.R (Rossum’s Universal Robots) del escritor checo Karel Capek en 1920, derivado de la traducción de la palabra checa “robota” que significa “trabajos forzados” o “trabajador”. Por otra parte, el término *robótica* es acuñado por Isaac Asimov y usado por primera vez en sus historias cortas en los años 40.

Es curioso que, al igual que otros autores de ciencia ficción como Julio Verne, Capek y Asimov predijeran con gran exactitud el futuro de la tecnología. En sus obras se muestran robots humanoides, programables y con avanzadas capacidades intelectuales asistiendo a los humanos en toda clase de tareas. Los robots, en última instancia, se rebelan contra sus creadores. La temática

del control de una nueva forma de inteligencia lleva a Asimov a desarrollar conceptos como sus famosas “tres leyes de la robótica” lo cual no es tan distinto de las propuestas de regulación de la inteligencia artificial que han surgido en la actualidad, trazando así otro paralelismo entre realidad y ficción.

Poco después de su primera aparición en la ciencia ficción, empiezan a surgir los primeros diseños de lo que hoy en día se entiende por robots. Un primer ejemplo pueden ser los robots móviles *Elmer* y *Elsie* (*figura 3*) desarrollados por el neurobiólogo William Grey Walter a finales de la década de los 40. Estos “robots tortuga” eran capaces de moverse en reacción a estímulos luminosos.

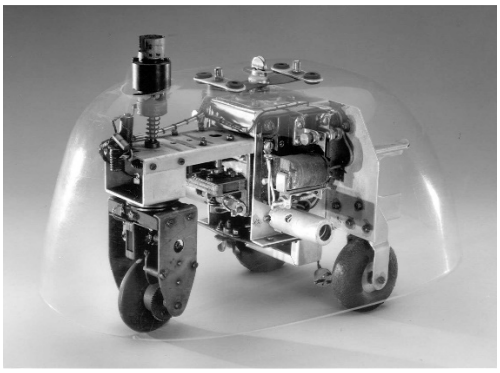


Figura 3: Robot Elmer / Elsie

Ya en los años 50 surge lo que se considera como el primer robot industrial. George Devol y Joseph Engelberger fundan la compañía *Unimation* para producir el robot *Unimate* (*figura 4)* que se entregaría en 1961 a General Motors para trabajar en su línea de ensamblaje de automóviles.

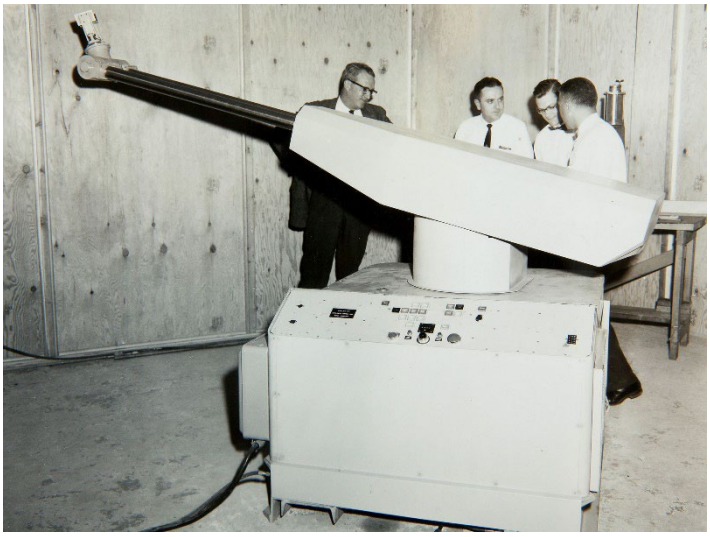


Figura 4: Robot Unimate

A lo largo de los años 60 y 70 se producen los mayores avances principalmente en robótica industrial, aunque también en investigación con robots cada vez

más inteligentes e incluso se mandan robots al espacio como parte de la carrera espacial. Es en esta época cuando se desarrollan características familiares de los robots actuales como los robots industriales de 6 ejes, el accionamiento electromecánico y control mediante microcontroladores y la programación de robots mediante lenguajes de programación específicos.

De esta época cabe destacar el IRB6 (*figura 5*), el primer robot accionado de forma totalmente eléctrica y controlado por una microcomputadora por parte de la empresa sueca ASEA que acabaría convirtiéndose en ABB, y también el robot PUMA (*figura 6*) de *Unimation* considerado el primer brazo manipulador programable universal.

Figura 5: Robot IRB 6

Figura 6: Robot PUMA

2.1.2 Acotando la definición de robot social

Si bien desde los orígenes de la robótica se ha contemplado la idea de robots inteligentes colaborando en entornos sociales con los humanos, la robótica social es una rama bastante reciente.

Desde principios de la década de 1990 los avances en computación e inteligencia artificial permitieron el desarrollo de robots con capacidades cada vez más "inteligentes" lo que les permite involucrarse e interaccionar en un entorno social.

Repasando la literatura, no existe consenso en cuanto a la definición de robótica social ni a las características que constituyen un robot social. A mayores, dentro del campo de lo considerado "robótica social", se presentan robots con prestaciones y propósitos muy variados lo que complica aún más su clasificación en categorías definidas. A continuación se describe la clasificación propuesta por M.Heerink [7] (*figura 7*) y adoptada por Ohra y Oniga [8].

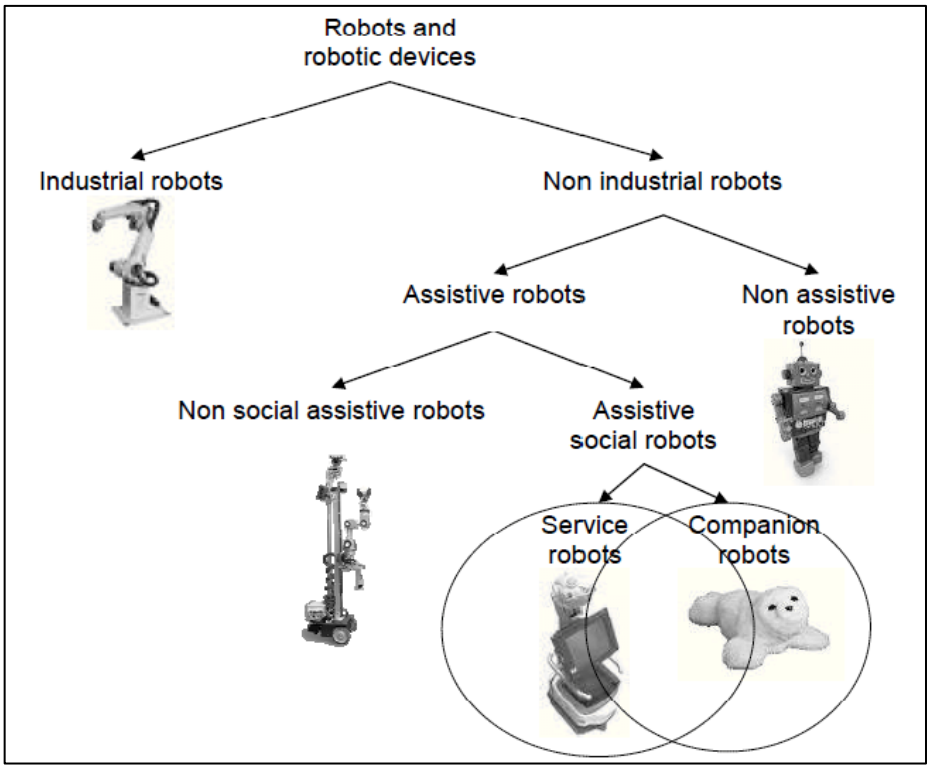


Figura 7: Clasificación de los robots

Identificamos los robots industriales como aquellos dedicados a los procesos de producción. Entre los robots no industriales, una gran mayoría se puede clasificar como robots asistenciales, aquellos robots que, de alguna manera, ayudan o asisten al ser humano en una tarea tengan una componente social o no.

Los robots asistenciales pueden organizarse en dos categorías: pueden ser robots no-sociales o robots sociales. El primer tipo comprende principalmente robots de asistencia física como por ejemplo la rehabilitación muscular o neuromotora sin incurrir en una interacción social. Un ejemplo podría ser el robot de rehabilitación ROBHAND [9] del grupo de robótica médica de la Uva; también se incluirían en esta categoría miembros prostéticos robóticos o exoesqueletos.

El segundo tipo de robots asistenciales es socialmente interactivo. Incluyen sistemas que permiten que el robot pueda ser percibido como una entidad social que se comunica con el humano o con la que el humano puede comunicarse, no siempre se trata de una comunicación bidireccional. Aunque pueda darse cierto solapamiento, ya que existen por ejemplo robots sociales utilizados en proyectos de rehabilitación, por regla general son campos separados y, en última instancia, el criterio clave de diferenciación es la componente social.

Ya en la categoría que nos ocupa en este trabajo, los robots asistenciales sociales se pueden dividir a su vez en dos subcategorías:

- Robots de Servicio: se utilizan como dispositivos funcionales y no están particularmente diseñados para proporcionar apoyo social/emocional. Sus funciones se relacionan principalmente con el apoyo a la vida independiente ayudando en las tareas básicas del hogar, movilidad y navegación y monitorización de aquellas personas que precisen de atención constante.

- Robots Acompañantes (*companion robots*): proporcionan una sensación de compañía similar a la de una mascota, a menudo asemejándose físicamente a una, e incorporando interacciones gestuales y mediante el tacto. (*figura 8*)

Figura 8: Robots de compañía AIBO (izq.) Paro (centro) y EMO (dcha.)

Estas dos subcategorías no son exclusivas ya que muchos robots comparten características de ambas y son difíciles de clasificar en uno u otro grupo. Es habitual, por ejemplo, con la llegada de tecnologías como los asistentes de voz que robots de servicio puedan también proporcionar compañía.

Es el caso del robot Temi (*figura 9*) que cuenta con una tableta Android que permite incorporar al robot asistentes y avatares virtuales, juegos móviles y todo tipo de aplicaciones compatibles para amenizar la experiencia de uso. De hecho, se considera extremadamente útil de cara a la aceptación por parte de personas mayores, que un robot asistente tenga ciertas habilidades sociales típicas de un robot acompañante.

Figura 9: Robot Temi

Una de las definiciones de "robot social" más frecuentes en la literatura consultada, es la dada por Fong *et al*. [10] *los robots sociales son aquellos a los que las personas aplican un modelo social a fin de entenderlos e interaccionar con ellos.*

> *"Social robots are those that people apply a social model to in order to interact with and understand."*
>
> *Fong et al. 2003*

Para evocar dicha aplicación, por parte del humano, de un modelo social a la interacción con el robot, el robot debe tener ciertas características. Primero que nada, el robot debe verse o comportarse de una manera tal que tenga sentido aplicar un modelo social para interactuar con él. Por lo general, lo anterior se puede decir de un robot si exhibe una o más de las características siguientes:

1. Expresar y/o percibir emociones (expresiones faciales, sonidos, lenguaje natural, gestos corporales, etc.)
2. Comunicarse mediante diálogo de alto nivel.
3. Aprender/reconocer los modelos de otros agentes sociales.
4. Establecer/mantener relaciones sociales.
5. Uso de señales no verbales (p.ej. miradas, gestos)
6. Exhibir una personalidad y carácter distintivos.
7. Posibilidad de aprender/desarrollar competencias sociales.

Estas características se pueden obtener implementando las correspondientes habilidades sociales en los robots. Dichas habilidades pueden ser escalables y hacer al robot más o menos sociable. Este enfoque está en línea con la mayoría de los robots sociales del mercado que no solo traen ciertas habilidades básicas preinstaladas, sino que además proporcionan una plataforma de desarrollo para implementar nuevas funcionalidades.

Para comprender, desde un punto de vista global, el concepto de "modelo social" asociado a un robot social, se recomienda consultar [11] donde Hegel *et al.* discuten cuatro posturas populares sobre la definición de los robots sociales (incluyendo la de Fong *et al.*) y proporcionan un análisis de sus características en común para proponer una definición unificadora.

Hegel *et al.* sostienen que en los robots sociales tienen importancia tanto la forma como la función. Los usuarios humanos perciben al robot de forma distinta en base a su estética lo cual es de especial importancia a la hora de establecer una conexión social. Los pequeños detalles de una comunicación social como gestos, señas, lenguaje no verbal y el comportamiento general del robot se transmiten mediante esta forma física, de ahí su importancia.

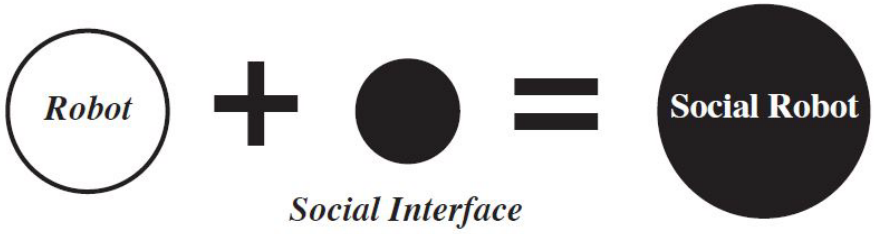


Figura 10: Modelo de un Robot Social [11]

En ese sentido un robot social combina aspectos técnicos (forma física, movimiento) con aspectos sociales. Sin embargo, son estos últimos los que constituyen el propósito central de los robots sociales: un robot no es social *per se*, necesita unas capacidades específicas de comunicación e interacción. En definitiva, el robot debe comportarse correctamente dentro de un contexto social y tener una apariencia que exprese explícitamente esa capacidad de ser social. El modelo de robot social (*figura 10*) queda definido entonces como la suma de un robot y una interfaz social. La interfaz social abarca todas las características de diseño por las cuales un usuario juzga las cualidades sociales del robot.

2.1.3 Aplicaciones y Efectividad de los robots sociales

Como se ha detallado anteriormente, los robots sociales son principalmente utilizados como acompañantes o asistentes al ser humano en una tarea o ámbito concreto, o como plataformas de investigación y desarrollo para lograr avances en el campo de la robótica social.

Los principales usos asistenciales se pueden resumir en:

- Cuidado de personas mayores y/o físicamente impedidas.
- Monitorización y telepresencia de personas dependientes.
- Cuidado de individuos con trastornos cognitivos.
- Apoyo emocional y compañía.
- Apoyo a la enseñanza.
- Servicio al cliente (hoteles, centros comerciales, etc.).

Los robots sociales de servicio al cliente suelen ser proyectos promovidos por los propios establecimientos que solicitan o desarrollan internamente una solución personalizada para su negocio. Los casos más comunes son robots que muestran información a través de sus pantallas, robots móviles que guían a los clientes a través de las instalaciones o que reparten artículos a modo de camareros en un restaurante o como parte del servicio de habitaciones en un hotel (*figura 11*).

Figura 11: Robot Relay de "Relay Robotics"

El resto de aplicaciones se pueden agrupar en dos categorías: por un lado tenemos la atención, cuidado y monitorización de personas con necesidades especiales ya sean físicas o mentales, y por otro el apoyo psicológico, educativo o cognitivo. Como ya se ha mencionado, no son funciones excluyentes, un robot puede desempeñarse en ambas categorías simultáneamente.

El primer caso lo constituyen robots diseñados principalmente para el apoyo a la vida independiente de personas mayores que viven solas. Se implementan en el entorno del hogar de la persona y su función se realiza, casi siempre, bajo la supervisión de un cuidador. Estos robots asisten en las labores cotidianas y proporcionan funciones tales como: calendario, recordatorios, comunicación mediante llamadas o videollamadas, detección de caídas, etc. También permiten la telepresencia y teleasistencia por parte de los cuidadores o de los familiares al cuidado de la persona mayor. A pesar de no ser su objetivo principal, se ha demostrado que la presencia e interacción con este tipo de robots recibe una buena aceptación por parte de los usuarios en los estudios realizados con personas mayores [12] [13].

Un ejemplo claro de este tipo de aplicaciones de los robots sociales es el proyecto EIAROB [4] que utiliza el robot Temi como parte de su sistema de asistencia en el hogar. El robot cuenta con una tablet para la interacción con el usuario y toda una suite de sensores que le permiten percibir y navegar por el entorno (*figura 12)*. El robot está programado con todo tipo de funcionalidades para realizar sus labores de asistencia: reconocimiento facial, asistente de voz, videollamadas, etc.

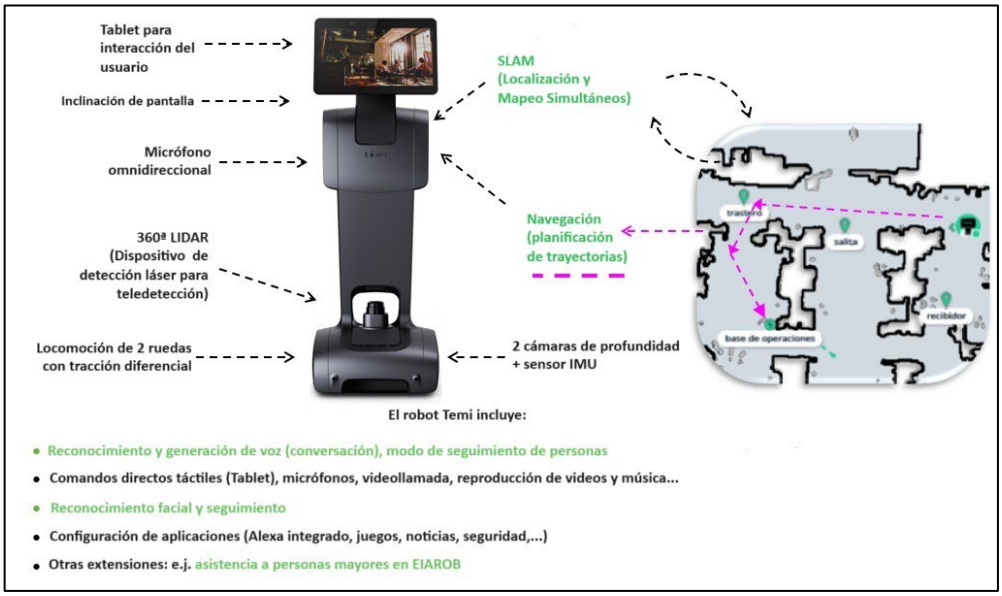


Figura 12: Funcionalidades del robot Temi del proyecto EIAROB

El otro tipo de aplicaciones consiste en el uso de robots, principalmente acompañantes, en los campos de la salud mental, el apoyo psicológico y la educación. En este caso, los robots sociales se han demostrado eficaces en la terapia de salud mental hospitalaria en niños [14] [15] y tratamiento de niños con trastornos del espectro autista [16], beneficiosos para individuos con deterioro cognitivo como demencia o Alzheimer [17] [18] y en el mundo educativo [19].

Por su parte, el robot NAO (*figura 13*) también ha demostrado su validez en este ámbito, con estudios que analizan su efectividad en hospitalización pediátrica [20], trastornos del espectro autista [21], asistencia en la educación [22] [23] así como en el cuidado y compañía de personas ancianas [24], fisioterapia [25] y detección de caídas [26].

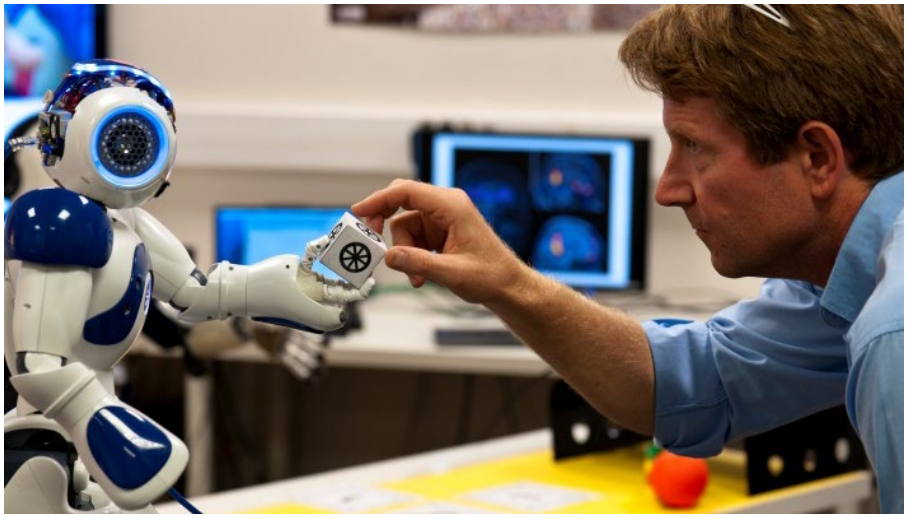


Figura 13: Robot NAO interactuando con un humano

Cabe mencionar por último los aspectos éticos [27] del uso de robots sociales en atención sanitaria y cuidados de personas mayores y dependientes. Un riesgo potencial de la interacción con robots sociales es la decepción: se crean unas expectativas de las funciones que puede realizar el robot cuando en realidad no puede. Con un mayor parecido humano y características antropomórficas, las personas que interactúan con los robots pueden llegar a esperar un comportamiento y una relación casi-humana con el robot e involucrarse demasiado sentimentalmente, lo cual puede resultar peligroso para personas en un estado psicológico vulnerable o personas fácilmente influenciables como niños y ancianos. Es por eso que algunos expertos han expresado la necesidad de gestionar las expectativas que se tienen respecto a un campo como la robótica social que aún requiere de mucho desarrollo [28].

Otra de las preocupaciones es que este contacto entre humano y robot acapare la mayor parte de las interacciones sociales del individuo llevando al aislamiento de las personas más ancianas. Algunos robots sociales, sin embargo, implementan funciones de telepresencia permitiendo a los usuarios realizar videoconferencias con familiares y cuidadores lo que puede tener el efecto contrario, disminuyendo la soledad y el aislamiento.

Actualmente, aún hay poca evidencia sobre los impactos a largo plazo que puede tener el contacto limitado de humanos con robots que son cada vez más interactivos.

2.1.4 Ejemplos de Robots Sociales

PARO – AIST (2004)

PARO es un robot social terapéutico lanzado en 2004 por la empresa japonesa AIST (*Advanced Industrial Science and Technology*). Diseñado en forma de una foca bebé (*figura 14*), está destinado principalmente para su uso en entornos de atención médica y terapéutica como residencias de ancianos y hospitales.

Figura 14: Robot PARO

PARO está equipado con sensores táctiles, sensores de luz y micrófonos lo que le permite reaccionar a gestos táctiles y orientarse hacia fuentes de luz o de sonidos. Aunque no dispone de locomoción, puede mover la cabeza y las aletas, parpadear y emitir sonido.

Está enfocado a la terapia y el bienestar emocional, especialmente en el contexto de la atención a personas mayores, pacientes con demencia o en entornos hospitalarios de forma similar a la terapia con animales. PARO responde al contacto físico y a la voz humana mostrando reacciones afectivas como movimientos y sonidos que imitan a los de una foca bebé real. Estas interacciones están diseñadas para tener un efecto calmante y fomentar una respuesta emocional positiva.

Jibo – Jibo Inc. (2017)

Jibo fue un robot social interactivo lanzado en 2017 por Jibo, Inc., una empresa fundada por Cyntia Breazeal, profesora y pionera en robótica social del MIT Media Lab. Jibo fue diseñado para ser un asistente social doméstico, de forma similar a asistentes virtuales como Alexa o Siri, pero instalado dentro de un cuerpo robótico de diseño minimalista y amigable (*figura 15*).

Figura 15: Robot Jibo

El robot de 28 cm de altura cuenta con una pantalla LCD a modo de "cara" capaz de mostrar varias expresiones animadas y proporcionar una interfaz visual para la interacción con el usuario. La cabeza puede girar y moverse en tres ejes lo que le permite simular gestos de seguimiento de la conversación.

Gracias a los micrófonos y cámaras de alta resolución integradas es capaz de realizar reconocimiento facial, reconocimiento de voz y procesamiento de lenguaje natural para interactuar verbalmente con los usuarios. De forma similar a otros asistentes para el "hogar inteligente" puede reconocer a diferentes usuarios, gestionar calendarios y recordatorios, y controlar otros dispositivos inteligentes (*smart home devices*) mediante integraciones.

Pepper – Softbank Robotics (2014)

Robot social de apariencia humanoide (*figura 16*) de cintura para arriba, Pepper cuenta con una pantalla táctil en el pecho y se desplaza sobre una base móvil con ruedas omnidireccionales. Está equipado con cámaras, micrófonos, sensores táctiles en cabeza y manos, y sensores de distancia infrarrojos y ultrasónicos para la detección de personas y obstáculos.

Utiliza reconocimiento de voz y procesamiento de lenguaje natural para interactuar con los usuarios. Puede comprender y responder a preguntas, entablar conversaciones y realizar tareas sencillas basadas en comandos verbales. Permite su programación y personalización mediante distintas herramientas de desarrollo como *Choregraphe*, NAOqi SDK para Python/C++ y el plugin QiSDK para Android Studio lo que permite controlar el robot mediante aplicaciones Android en su tablet incorporada.

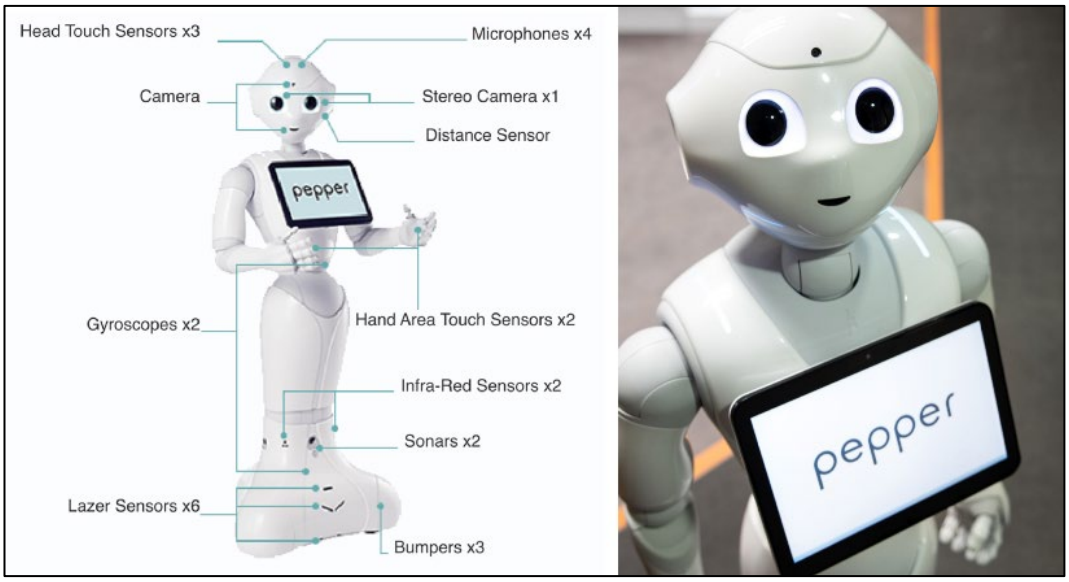


Figura 16: Robot Pepper

El robot Pepper se destaca de otros por su capacidad para detectar y responder a las emociones humanas, haciendo que las interacciones sean más personales y empáticas. Es capaz de reconocer rostros y emociones básicas, adaptando sus respuestas y comportamientos según el estado emocional percibido del usuario. Sus brazos y cabeza articulados le permiten realizar gestos y lograr una expresividad que complementa la interacción verbal.

Ameca – Engineered Arts (2021)

Ameca es un robot social humanoide diseñado como plataforma de desarrollo e investigación en robótica social, centrado en la interacción humana de manera realista y expresiva. El robot tiene un diseño altamente realista y futurista (*figura 17*). Aunque no está diseñado para la locomoción completa, tiene una base fija y articulaciones en los brazos, cuello y tronco que permiten movimientos complejos y naturales.

La característica más distintiva es su rostro humano hiperrealista compuesto de materiales suaves que permiten la reproducción de toda una gama de expresiones faciales. Cuenta con cámaras en los ojos para reconocimiento facial, micrófonos para la captura de audio y varios sensores en la piel que permiten la detección del tacto y proximidad. Además, sus manos y dedos están especialmente articulados, permitiéndole realizar gestos detallados.

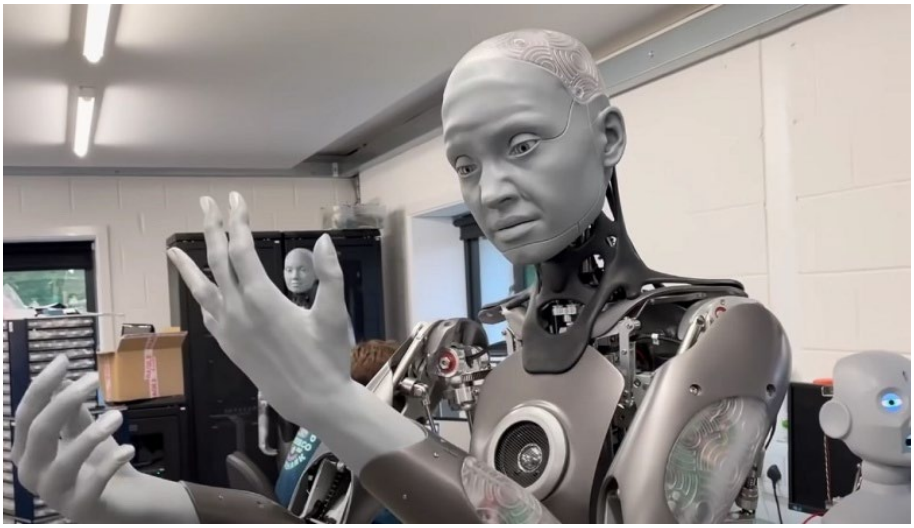


Figura 17: Robot Ameca

El enfoque principal es lograr una experiencia de comunicación con un alto grado de realismo gracias a su capacidad para replicar la expresividad facial humana. Para ello cuenta con sistemas avanzados de reconocimiento de voz, procesamiento de lenguaje natural y la reciente integración de modelos GPT.

Hiperrealismo y el Valle Inquietante

La evolución natural del campo de la robótica social nos lleva a robots cada vez más similares a los humanos en su aspecto y sus habilidades de comunicación verbal y no verbal. Los sistemas de diálogo han experimentado un gran avance, como se discutirá más adelante en este trabajo, mediante el uso de inteligencia artificial y LLMs. En cuanto al aspecto y la comunicación no verbal, algunos esfuerzos se han centrado en desarrollar hardware capaz de imitar mejor las expresiones faciales para así lograr un mayor nivel de interacción. Algunos de estos robots se muestran a continuación (*figura18)*:

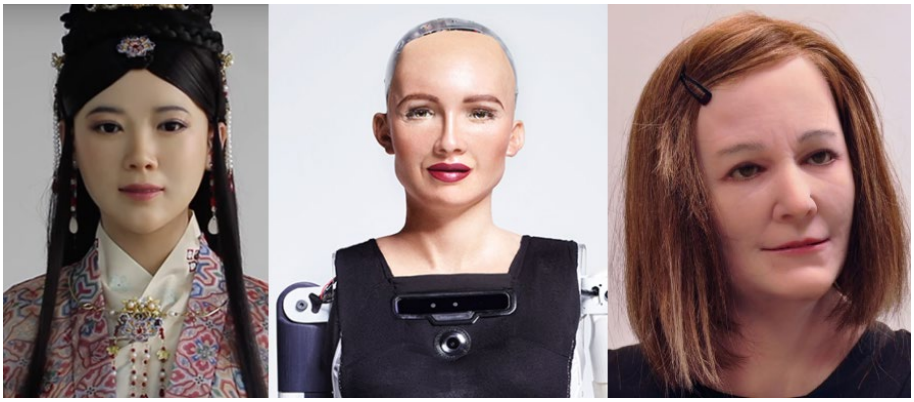


Figura 18: Robots Jia Jia (izq.) Sophia (centro) y Nadine (dcha.)

En la persecución de robots humanoides con aspecto y capacidades sociales cada vez más cercanas a las humanas, se incurre en un efecto conocido como el "valle inquietante" (*uncanny valley*). Este concepto, comúnmente utilizado en robótica y animación por ordenador, afirma que cuando las réplicas antropomórficas se acercan a la apariencia y comportamiento de un ser humano real, en cierto punto se produce una sensación de rechazo. El "valle" en cuestión hace referencia a una región en el gráfico propuesto (*figura 19*) que relaciona el grado de parecido de un objeto con un ser humano y la respuesta emocional a dicho objeto.

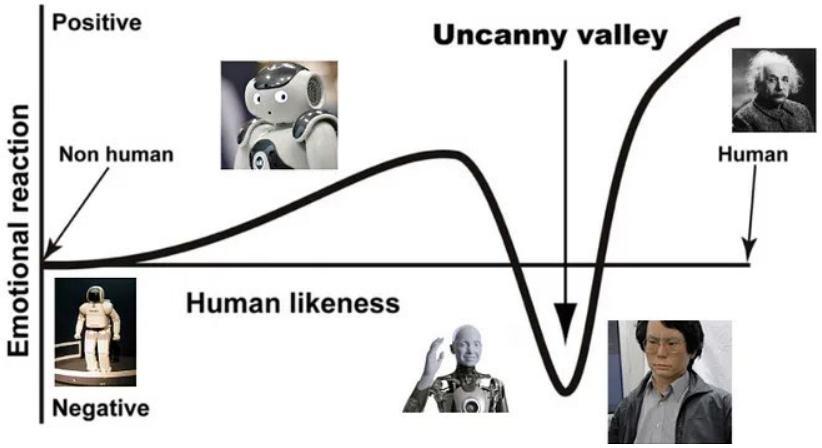


Figura 19: Uncanny Valley [29]

Conclusiones

Como se puede comprobar en base a los ejemplos proporcionados, la mayor parte de los robots sociales actualmente en el mercado son de momento juguetes y/o plataformas de desarrollo. Ninguno de ellos está ampliamente comercializado o introducido en la vida de las personas. Esta conclusión es denominada por Duffy y Joue [30] la "paradoja de la robótica social":

> *"Los robots humanoides fuera de la ciencia ficción, hasta ahora solo han sido juguetes o plataformas de desarrollo de propósito incierto. Es curioso que uno de los paradigmas más potentes en cuanto a flexibilidad y adaptabilidad, el ser humano, modelado en forma de máquina, hasta ahora no ha resultado en más que un juguete. Su utilidad es muy limitada."*

Se podría rebatir, sin embargo, que está en las manos de investigadores y desarrolladores el hacer uso de esas plataformas de desarrollo para impulsar el campo de la robótica social y crear algo que suponga un impacto revolucionario en las personas y la sociedad.

2.2 Inteligencia Artificial y Modelos de Lenguaje

La Inteligencia Artificial (IA) se define como la rama de las ciencias de la computación dedicada al desarrollo de sistemas capaces de realizar tareas que, de ser realizadas por un humano, requieren de inteligencia. Esto incluye habilidades como el razonamiento, el aprendizaje, la percepción, la compresión del lenguaje o la toma de decisiones. La IA abarca una amplia variedad de enfoques y tecnologías que buscan imitar o superar las capacidades cognitivas humanas.

Al más alto nivel se puede clasificar las IAs en [31]:

- **IA Débil:** También conocida como IA estrecha (*narrow AI*), se refiere a sistemas diseñados y entrenados para realizar una tarea específica. Estos sistemas no poseen inteligencia o consciencia en el sentido amplio de la palabra, sino que son capaces de imitar el comportamiento humano en contextos muy delimitados. Ejemplos de esto serían tecnologías como el reconocimiento de voz o la clasificación de imágenes, que se desempeñan bien en las tareas a las que están destinadas, pero carecen de comprensión más allá de sus capacidades programadas.

- **IA Fuerte:** Mejor conocida como inteligencia artificial general (*AGI*), es un concepto teórico que describe un sistema capaz de demostrar un nivel humano de inteligencia para cualquier problema, en lugar de limitarse a una aplicación específica como la IA débil. Es simplemente una definición ya que este nivel de inteligencia artificial aún no ha sido alcanzado.

Algunas áreas y/o disciplinas en las que la IA juega un papel clave son:

- Procesamiento de Lenguaje Natural (NLP)
- Visión por Computadora
- Sistemas de Clasificación y Recomendación
- Robótica
- Aprendizaje Automático (*Machine Learning*)

2.2.1 Introducción

LOS INICIOS

Las raíces de la IA se remontan a los años 40 y principios de los 50, cuando un puñado de científicos de diversos campos exploraron diversas líneas de investigación que serían vitales para la posterior investigación en inteligencia artificial.

Alan Turing, conocido por sus innovaciones en la teoría de la computación, publica en 1950 su artículo "*Computing Machinery and Intelligence*" [32] en el que especula sobre la posibilidad de crear máquinas capaces de "pensar". También en este artículo se introduce el concepto del Test de Turing: si un evaluador no puede diferenciar entre las respuestas de una máquina y las de un ser humano durante una conversación, la máquina puede considerarse inteligente.

En 1956, la conferencia de Dartmouth (*Dartmouth Summer Research Project on Artificial Intelligence*) marcó un momento crucial en la historia de la inteligencia artificial. Organizada por Marvin Minsky, John McCarthy, Claude Shannon y Nathan Rochester, este evento reunió a destacados investigadores (*figura 20*) interesados en explorar el potencial de las máquinas para realizar tareas que requieren inteligencia. Fue en este taller donde John McCarthy acuñó el término "Inteligencia Artificial" (*Artificial Intelligence*), definiendo un nuevo campo de estudio dedicado a la creación de máquinas capaces de replicar la cognición humana.

Figura 20: Algunos participantes de la conferencia de Dartmouth

La conferencia de Dartmouth es considerada el punto de partida oficial de la inteligencia artificial como disciplina, consolidando el objetivo de desarrollar máquinas que puedan realizar funciones cognitivas avanzadas.

A partir de estas raíces, la investigación en inteligencia artificial avanzó a lo largo de la segunda mitad del siglo XX, aunque no sin altibajos principalmente debidos a la desconfianza en la aplicabilidad de la tecnología. Los investigadores subestimaron la dificultad de los problemas a los que se enfrentaban. La oleada inicial de optimismo había elevado las expectativas hasta niveles increíblemente altos y cuando los resultados prometidos no se materializaron, la financiación destinada a la IA desapareció casi por completo. Esto dio lugar a dos periodos de estancamiento en la investigación conocidos como "inviernos de IA" (*AI winters*), el primero de 1973 a 1980 y el segundo de 1988 hasta 1993 (*figura 21*).

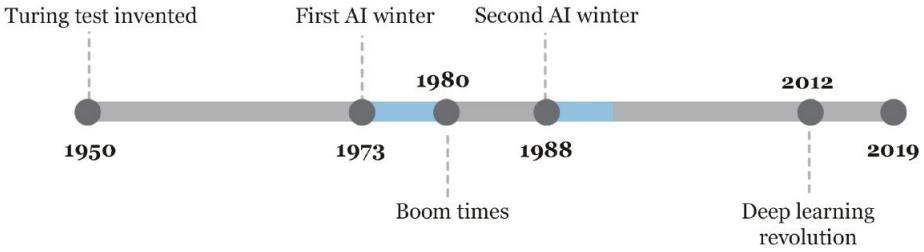


Figura 21: Línea temporal de los Inviernos de IA [33]

MACHINE LEARNING Y EL PERCEPTRÓN

Desde los inicios de la inteligencia artificial, uno de los enfoques ha sido el de replicar la biología, intentando crear "inteligencia" de la misma forma que lo hacemos los humanos.

En 1943, Warren McCulloch y Walter Pitts presentan su modelo de neurona artificial [34] y muestran cómo podría utilizarse para realizar funciones lógicas simples. Fueron los primeros en describir lo que los investigadores posteriores llamarían una red neuronal.

Posteriormente, en 1957, Frank Rosenblatt introdujo el perceptrón [35] (*figura 22*), un modelo fundamental para el desarrollo de las redes neuronales. Inspirado por el trabajo de McCulloch y Pitts [34], Rosenblatt diseñó el perceptrón como una simplificación de las neuronas biológicas, con la capacidad de aprender y reconocer patrones a través de un proceso de ajuste iterativo de pesos. El modelo podía aprender de los datos y realizar clasificaciones binarias. Aunque limitado, sentó las bases para el aprendizaje automático moderno.

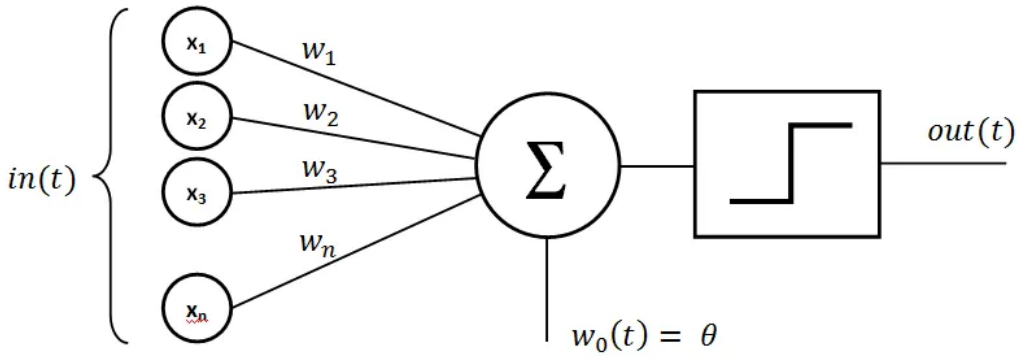


Figura 22: Modelo de perceptrón

El proceso de ajuste iterativo de los pesos representa la idea clave tras el aprendizaje automático (*machine learning*): en lugar de programar explícitamente las reglas de comportamiento, dichas reglas son inferidas o "aprendidas" a partir de patrones observados en los datos. Este aprendizaje se puede clasificar en tres categorías principales:

- **Aprendizaje Supervisado:** Consiste en entrenar un modelo usando un conjunto de datos etiquetados, donde cada entrada de datos tiene una salida conocida. El modelo aprende a mapear las entradas a las salidas, lo cual se utiliza posteriormente para predecir las salidas de datos nuevos. Ejemplos incluyen la regresión lineal, los árboles de decisión y las redes neuronales.

- **Aprendizaje No Supervisado:** Se emplea cuando los datos de entrenamiento no tienen etiquetas. El objetivo es encontrar estructuras o patrones ocultos en los datos. Los métodos incluyen *clustering* (agrupamiento) y la reducción de dimensionalidad, como el análisis de componentes principales (PCA).

- **Aprendizaje por Refuerzo:** En él, un agente aprende a tomar decisiones secuenciales mediante la interacción con un entorno. El aprendizaje se basa en el principio de recibir recompensas o castigos por sus acciones, y el objetivo es maximizar la recompensa acumulada a largo plazo.

El perceptrón fue fundamental al demostrar que era posible enseñar a una máquina a realizar tareas de clasificación básicas. Sin embargo, este modelo también tenía limitaciones significativas, como su incapacidad para resolver problemas linealmente no separables [36], ejemplificado por el problema de XOR. Esta limitación condujo a la necesidad de redes neuronales más complejas, como las redes multicapa.

DEEP LEARNING Y EL SIGLO XXI

El perceptrón y los primeros enfoques clásicos de ML presentaron varias limitaciones, principalmente en su habilidad para manejar problemas complejos y grandes volúmenes de datos. La falta de capacidad para aprender representaciones más profundas y abstractas llevó a la exploración de redes neuronales multicapa (*Multi-Layer Perceptrons, MLP*) y al desarrollo de lo que hoy se conoce como aprendizaje profundo (*Deep Learning, DL*).

Las redes neuronales profundas (múltiples capas ocultas o *hidden layers*) permiten la aproximación de funciones complejas mediante capas sucesivas de neuronas (*figura 23)*. Cada capa "aprende" diferentes niveles de abstracción de los datos. Sin embargo, el entrenamiento de estas redes más profundas no fue viable hasta que se dispuso de la capacidad computacional adecuada.

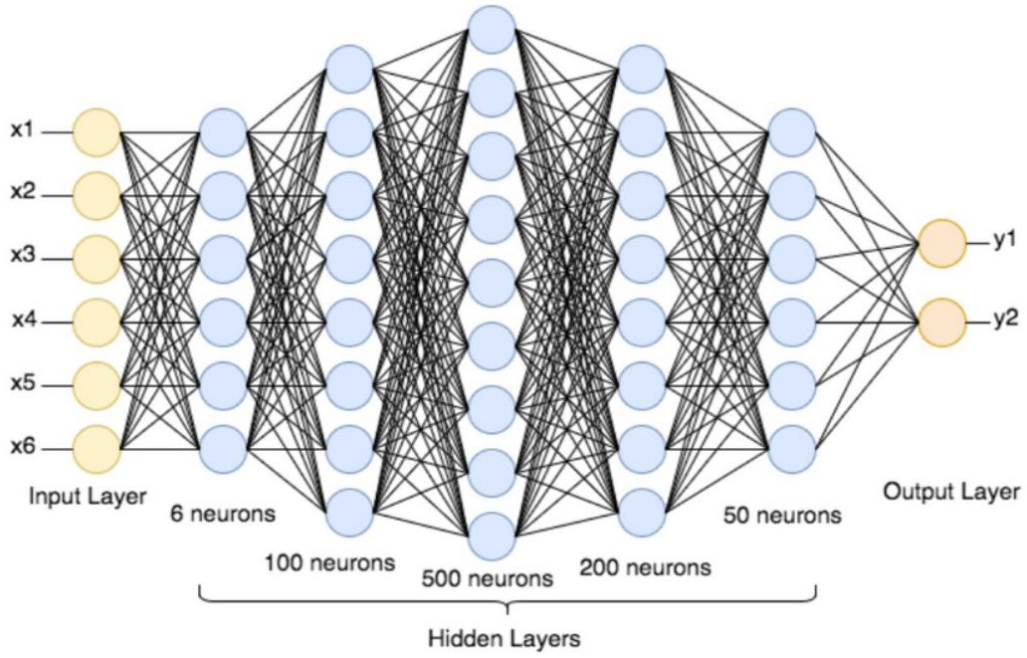


Figura 23: Arquitectura de una red neuronal profunda (DNN)

El progreso en el campo de la Inteligencia Artificial siempre ha estado estrechamente ligado a los avances en computación y capacidad de procesamiento. A pesar del constante desarrollo, con procesadores duplicando su número de transistores cada dos años (como vaticinó la *Ley de Moore*, 1965), durante la segunda mitad del siglo XX la capacidad de hardware no estuvo a la altura para entrenar redes neuronales complejas.

No fue hasta principios de la década de 2000 que se produjo un cambio significativo con la introducción de unidades de procesamiento gráfico. Las GPUs, con su capacidad para realizar cálculos en paralelo a gran escala, proporcionaron la infraestructura necesaria para manejar los vastos volúmenes de datos y las intensas demandas computacionales del *Deep Learning*. Esto permitió entrenar redes neuronales más extensas y profundas en tiempos razonables, catalizando el desarrollo de modelos mucho más potentes.

A lo largo de las últimas dos décadas, el campo del *Deep Learning* ha visto el desarrollo de diversas arquitecturas de redes neuronales, cada una optimizada para distintos tipos de tareas:

- Redes Neuronales Convolucionales (CNNs): introducidas por Yann LeCun en la década de 1990, son especialmente efectivas para tareas de procesamiento de imágenes y video. Utilizan capas convolucionales que aplican filtros a las entradas, extrayendo características locales y manteniendo relaciones espaciales. Su capacidad para capturar patrones jerárquicos hace que sean el estándar para tareas como la clasificación y segmentación de imágenes o el reconocimiento de objetos.

- Redes Neuronales Recurrentes (RNNs): son adecuadas para el procesamiento de secuencias y datos temporales, como el análisis de series temporales y el procesamiento de lenguaje natural. Utilizan conexiones recurrentes que permiten mantener una memoria de estados anteriores, proporcionando contexto temporal. Sin embargo, las RNNs estándar enfrentan problemas de memoria a largo plazo, lo que llevó al desarrollo de variantes como las LSTM (*Long Short-Term Memory*).

- Redes LSTM (Long Short-Term Memory): Introducidas por Hochreiter y Schmidhuber en 1997, las LSTM son una variante de las RNNs que resuelven el problema del gradiente desvaneciente (*vanishing gradient*) mediante una arquitectura de memoria más compleja. Las LSTM utilizan celdas de memoria que pueden aprender a retener u olvidar información a lo largo de las secuencias, haciendo posible recordar patrones a largo plazo.

- Redes Generativas Adversativas (GANs): Propuestas por Ian Goodfellow en 2014, las GANs consisten en dos redes neuronales que se entrenan juntas: un generador que crea datos sintéticos y un discriminador que intenta distinguir entre datos reales y sintéticos. Esta competencia entre ambas redes resulta en la generación de datos muy realistas, siendo especialmente útiles en la síntesis o mejora de imágenes y en la creación de contenido.

2.2.2 La arquitectura del Transformador

Hace apenas 7 años, el campo de la inteligencia artificial fue "transformado" radicalmente por una nueva arquitectura (*figura 24*) originalmente propuesta para la traducción de lenguaje natural. La arquitectura del Transformador (del inglés *Transformer*) fue introducida por Vaswani et al. en su artículo "*Attention is All You Need*" [37] de 2017.

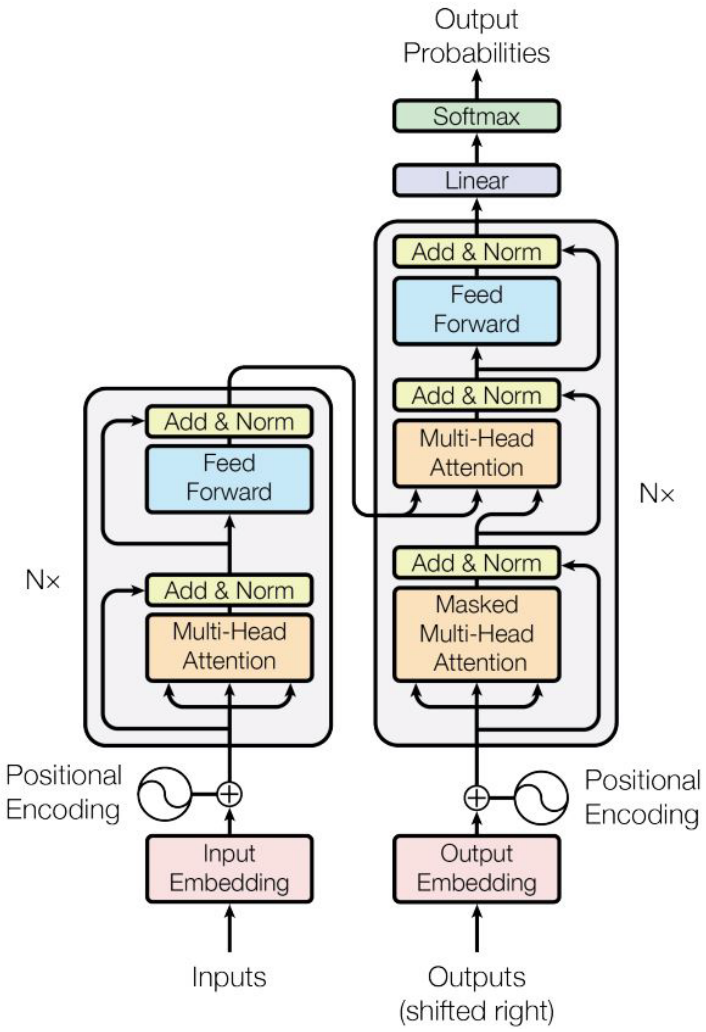


Figura 24: Arquitectura original de un Transformador [37]

El *Transformer* ha revolucionado el campo del procesamiento del lenguaje natural (NLP) y ha sentado las bases de los modelos de lenguaje modernos y otras aplicaciones en inteligencia artificial generativa. A diferencia de las redes neuronales utilizadas previamente para predicción de secuencias (RNN y LSTM), los Transformers utilizan un mecanismo de atención que les permite procesar secuencias de datos de manera más eficiente, superando las limitaciones de los modelos anteriores en términos de paralelismo y retención de "memoria" a largo plazo.

No se entrará en detalles técnicos de la arquitectura ya que queda fuera del alcance de este trabajo. Sin embargo, sí que se explicarán de forma superficial los conceptos clave de la arquitectura, junto con los beneficios que ésta proporciona sobre las alternativas anteriores.

Los tres conceptos clave [38] para entender la arquitectura son:

1. *Embeddings* codificados con posición (*Positional Encoddings*)
2. El mecanismo de atención
3. Auto-atención (*Self Attention*)

EMBEDDINGS Y POSITIONAL ENCODING

A la hora de trabajar con secuencias de texto (los modelos multimodales aceptan también video y audio como datos de entrada) los *Transformers* no trabajan con caracteres sueltos, sino que separan la cadena de texto en unidades llamadas tokens (*figura 25*).

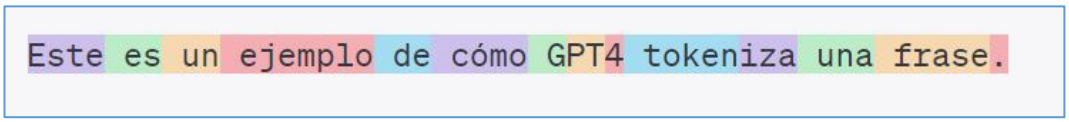


Figura 25: Ejemplo utilizando el tokenizer de OpenAI [39]

Este proceso es clave ya que el token representa la unidad mínima de información (texto en este caso) con la que va a trabajar el modelo. Un token no tiene por qué corresponderse a una palabra, a menudo separan raíces de prefijos y sufijos (*token-iza,* en el ejemplo) o contienen únicamente números o caracteres de puntuación.

El siguiente paso es asociar cada token con un vector en el cual se intentará codificar el "significado" de dicho token. Este proceso nos permite representar el "significado" de cada token de forma numérica como si fueran coordenadas en un espacio multidimensional. Intuitivamente, palabras (tokens) con significados similares tendrán vectores próximos el uno al otro.

Los vectores resultantes (*figura 26*) son conocidos como *"Word Embeddings"* y juntos conforman una matriz de *embeddings* (*Embedding Matrix*). Este proceso no es una innovación de los *Transformers,* sino que ya era una práctica común en el procesamiento de lenguaje natural (NLP) como forma de obtener una representación numérica de cada palabra sobre la que poder aplicar las operaciones matemáticas típicas del aprendizaje profundo (algebra lineal con matrices y tensores).

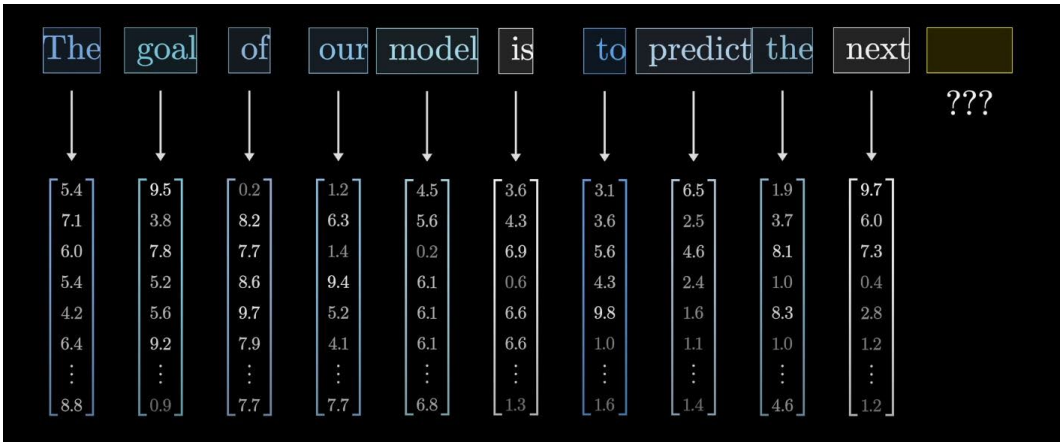


Figura 26: Representación de "Word Embeddings" [40]

Una de las características distintivas de los *Transformers* es que incluyen información posicional en los *embeddings*. Los *Transformers* no tienen un procesamiento secuencial inherente como las redes recurrentes (RNN), por lo que utilizan ***embeddings*** **posicionales** (***Positional Encodings***). Información sobre la posición de cada token en la secuencia se añade a los *embeddings* de entrada de manera que cada elemento tiene una noción relativa de su posición. Dicho de otra forma, se almacena información sobre el orden de las palabras en los propios datos (*embeddings*) en lugar de en la estructura de la red, como en las RNN. A medida que se entrena la red, ésta aprende a interpretar esta información posicional.

Por simplificación, durante el resto de la explicación se hará referencia a "palabras" cuando en realizad lo que el modelo procesa y genera son tokens.

ATENCIÓN

El *Transformer* se caracteriza por ser una arquitectura de red neuronal centrada en el mecanismo de atención, más concretamente *Multi-Head Self-Attention*. El mecanismo de atención permite a un modelo de texto enfocar su "atención" en cualquier palabra de la secuencia de entrada para decidir que palabra generar en la secuencia de salida. Se asignan diferentes pesos a las distintas partes de la secuencia de entrada, destacando las más relevantes para la tarea en cuestión.

En el artículo original [37] la arquitectura es inicialmente propuesta para realizar la traducción de lenguaje natural, de un idioma a otro. Para esta tarea el enfoque más sencillo, y equivocado, sería traducir las palabras independientemente unas de otras sin tener en cuenta su posición en la frase o qué otras palabas tiene a su alrededor, es decir, sin un contexto.

En la labor de traducción (y también en la generación de texto) el contexto de cada palabra es importante ya que, por ejemplo, los adjetivos deben tener el mismo género y número que los nombres a los que acompañan. **Puestos a generar la siguiente palabra en la traducción**, qué palabras de la secuencia de entrada debemos analizar, a cuáles damos más relevancia, **a cuáles prestamos más atención.** Esta es la idea general tras el concepto de "atención" en IA. En la *figura 27* se muestra una matriz de atención, a modo de mapa de calor, en un modelo de traducción [41] de forma similar a cómo se aplica en el artículo original sobre *Transformers*.

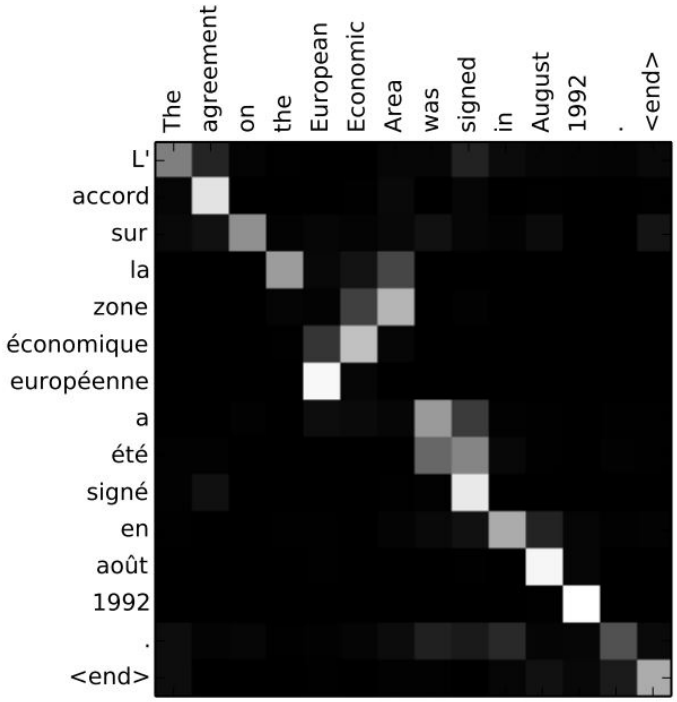


Figura 27: Matriz de atención - mapa de calor

AUTO-ATENCIÓN (*SELF ATTENTION*)

El concepto de atención descrito hasta ahora es denominado a veces atención cruzada (*cross-attention*). El termino hace referencia al hecho de que la atención se realiza entre dos secuencias de datos distintas, una de entrada y otra de salida. En el caso de la traducción, la secuencia de entrada es una frase en un idioma, y la salida es otra frase en otro idioma que se corresponde con la traducción de la entrada. En este caso la atención relaciona qué palabras de entrada influyen más en la generación de qué palabras de salida (*figura 28*).

Figura 28: Visualización de cross-attention [42]

Los *Transformers* utilizan otro tipo de atención denominada auto-atención (s*elf attention*) que se enfoca en las relaciones internas dentro de la misma secuencia (*figura 29*). Un *Transformer* no opera con dos secuencias distintas de entrada y salida, sino que opera únicamente con una secuencia de entrada la cual va completando poco a poco, añadiendo en cada iteración una nueva palabra al final de la secuencia.

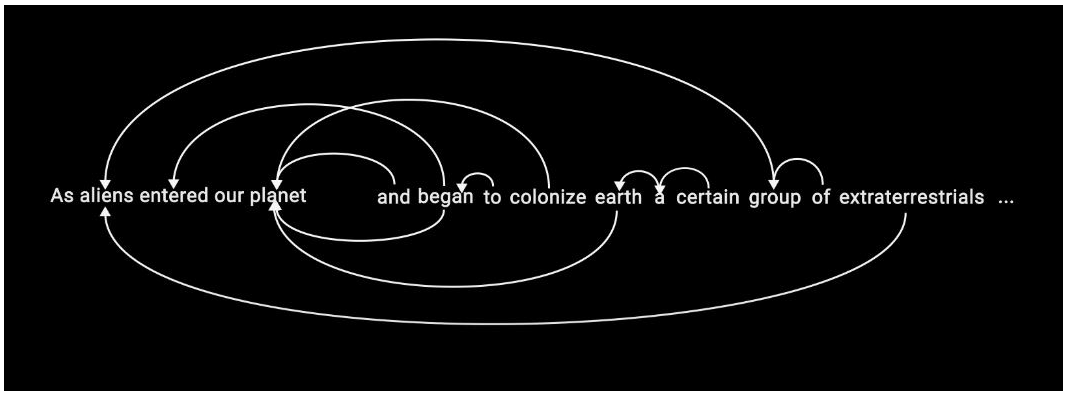


Figura 29: Visualización de self-attention [43]

El mecanismo de atención en un *Transformer* permite a los vectores (embeddings) de cada token "hablar entre sí" actualizando sus valores en función de las palabras a su alrededor. El bloque que implementa la atención es responsable de determinar qué palabras en el contexto son relevantes para actualizar el significado de qué otras palabras y cómo debe realizarse esta actualización. Los pesos de la red asociados al mecanismo de atención se van "aprendiendo" de los datos a lo largo del proceso de entrenamiento. De esta forma la red va comprendiendo el significado de las palabras en su contexto y es capaz de predecir/generar la siguiente palabra en la secuencia.

2.2.3 Modelos Generativos Pre-entrenados (GPT)

Para la gran mayoría de usuarios, el primer contacto con los grandes modelos de lenguaje (LLMs) fue a través del servicio *ChatGPT* de la compañía OpenAI. *ChatGPT* es el nombre de una aplicación web de *chatbot* que utiliza los modelos GPT-3.5 y posteriores. A partir de la versión GPT-3, OpenAI privatizó el acceso a sus modelos permitiendo únicamente su utilización mediante la API. Gran parte de la información pública que se tiene sobre los modelos GPT proviene de las versiones GPT-2 y anteriores, las cuales fueron distribuidas libremente.

Los modelos GPT (*Generative Pre-Trained Transformers*) son un tipo de LLM introducido por OpenAI en 2018 dentro del ámbito de la IA generativa:

- *Generative:* modelos capaces de generar texto.
- *Pre-Trained:* referente a que el modelo fue entrenado con un volumen de datos masivo inicialmente, pero que se puede ajustar (*fine tuning*) para tareas más específicas con un entrenamiento adicional.
- *Transformer:* arquitectura subyacente del modelo.

Todos los *Transformers* tienen los mismos componentes principales:

- Un *tokenizador* que convierte el texto en tokens.
- Una capa de *embedding*, que convierte los tokens y sus posiciones en representaciones vectoriales.
- Múltiples capas de transformación que llevan a cabo transformaciones repetidas sobre los vectores, extrayendo cada vez más información lingüística. Estas constan a su vez de capas de atención y *feedforward*.

Las capas de transformación pueden ser tanto codificadores como decodificadores. La propuesta original [37] utilizaba ambos tipos, aunque muchos de los modelos posteriores incluyen solo uno de los dos. El modelo BERT es un ejemplo de arquitectura *encoder-only* mientras que los modelos GPT solo utilizan bloques decodificadores.

Los modelos GPT se centran en la predicción de la siguiente palabra en una secuencia de texto, usando para ello el contexto de las palabras anteriores. A primera vista, predecir el siguiente valor en una secuencia o la siguiente palabra (token) en un texto, puede parecer algo completamente distinto de la generación de texto que realizan los modelos GPT, nada más lejos de la realidad.

Una vez se tiene el modelo predictivo, el primer paso es proporcionarle un fragmento inicial de texto, por ejemplo, cuando inicias una conversación con *ChatGPT* y le indicas en qué quieres te asista o directamente le haces una pregunta. Seguidamente el modelo realizará la inferencia, obteniendo una distribución estadística que intenta predecir la siguiente palabra de la secuencia (*figura 30*).

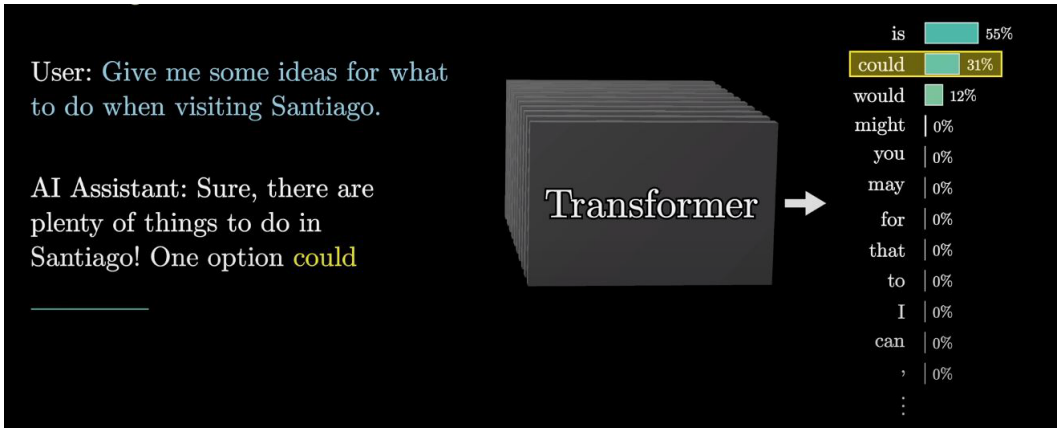


Figura 30: Visualización, predicción de la siguiente palabra del texto [40]

Se elige una muestra de la distribución (no siempre la de mayor probabilidad) y se adjunta la palabra predicha al texto inicial repitiendo de nuevo todo el proceso. Iterativamente, este proceso de "predicción de la siguiente palabra" consigue generar un texto completo o la respuesta a una pregunta.

ENTRENAMIENTO

Para la obtención de estos modelos se sigue un entrenamiento en dos fases:

La primera etapa, denominada preentrenamiento, se lleva a cabo con grandes volúmenes de datos textuales sin supervisión. Se toma un gran fragmento de información de internet proveniente de todo tipo de fuentes. Utilizando un potente conjunto de equipos de hardware especializado, se realiza el entrenamiento que puede durar semanas o meses y costar millones de euros.

El objetivo de este entrenamiento es conseguir un modelo generativo capaz de predecir la siguiente palabra en una secuencia de texto. Los modelos deben ser entrenados con un enorme volumen de datos para que el modelo "aprenda" los patrones y estructuras del lenguaje natural y sea capaz de generar texto con un nivel de inteligencia suficiente. Por poner un ejemplo, GPT-3 es un modelo de 175B (miles de millones) de parámetros, y fue entrenado con aproximadamente 45 TB de texto en esta fase.

Tras el preentrenamiento, el modelo obtenido es puramente generativo. Es capaz de redactar documentos basándose en una secuencia de texto inicial pero no sabe seguir instrucciones proporcionadas por el usuario.

La segunda etapa del entrenamiento es el ajuste fino (*fine-tuning*). Se toma el modelo pre-entrenado y se realiza un entrenamiento adicional con un nuevo conjunto de datos, esta vez etiquetados. Los nuevos datos, generados manualmente por humanos (aprendizaje supervisado), tienen la forma de conversaciones de tipo pregunta-respuesta. El objetivo de este segundo entrenamiento es actualizar los pesos del modelo para optimizar el rendimiento en la tarea específica descrita por los nuevos datos etiquetados.

El nuevo modelo obtenido después del *fine-tuning* es un modelo de asistente, se ajusta al formato pregunta-respuesta de los documentos de entrenamiento. Es capaz de generar texto como respuesta apropiada a una pregunta o instrucción del usuario. Mediante el ajuste fino, los modelos cambian su formato de salida manteniendo el conocimiento del lenguaje adquirido durante la etapa de preentrenamiento.

En la *figura 31* se muestra el resumen del proceso de entrenamiento explicado por Andrej Karpathy [44] a partir de los datos de entrenamiento del modelo Llama 2 70B (MetaAI).

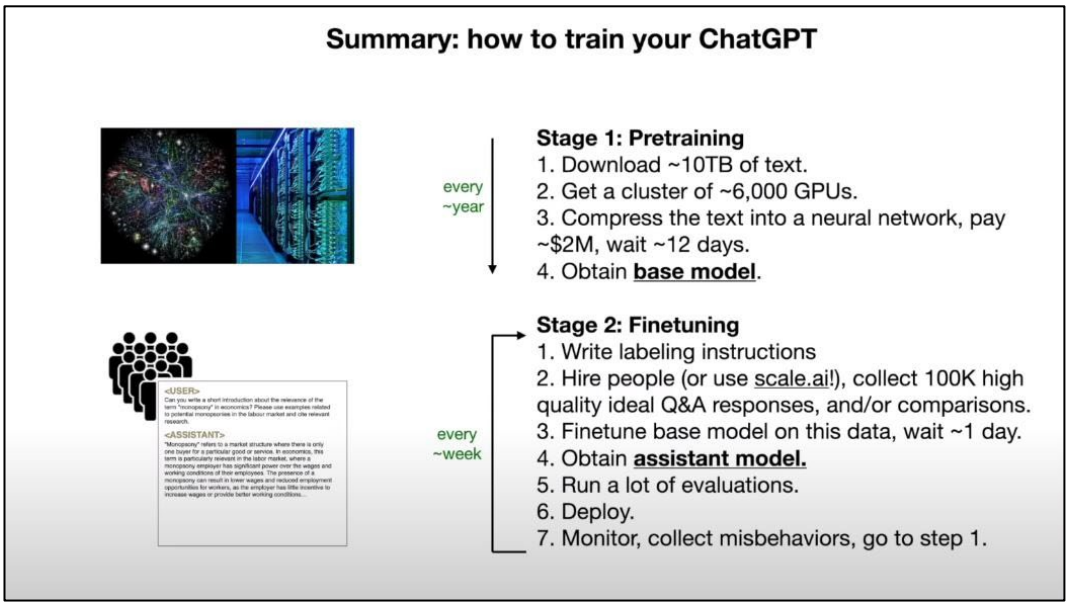


Figura 31: Resumen del proceso de entrenamiento de un GPT [44]

Como se puede ver en la figura, el último paso del *fine-tuning* implica la corrección de los comportamientos no deseados. Se analizan los escenarios conflictivos, se modifican los datos de entrenamiento acordemente y se realiza un nuevo entrenamiento de ajuste fino.

OTROS MODELOS

Aunque las siglas GPT son utilizadas exclusivamente por OpenAI para nombrar sus modelos, el término "GPTs" se utiliza popularmente para referirse a otros grandes modelos de lenguaje (LLMs) de características similares que han surgido como competencia. De entre todos estos modelos, destacan:

- Familia de modelos *Gemini* de Google
- Familia de modelos *Llama* de Meta (Facebook)
- Familia de modelos *Claude* de Anthropic

A la hora de comparar las diferentes alternativas, evaluar el rendimiento de los modelos de lenguaje modernos presenta múltiples dificultades. Por un lado, evaluar el texto generado es intrínsecamente subjetivo, así como la evaluación de cualidades como la coherencia, fluidez, relevancia o veracidad del contenido. La evaluación manual es ardua y está sujeta a sesgos humanos, mientras que las métricas automáticas no son capaces de capturar del todo la calidad del texto. Por último, los modelos pueden mostrar un rendimiento desigual en diferentes tareas o capacidades, complicando la creación de una métrica universal.

Con todo esto, se incluye a continuación (*figura 32*) uno de los índices de clasificación más populares [45] a fecha de junio de 2024. En la figura se observa el último modelo de OpenAI, GPT-4o, liderando la clasificación.

Rank* (UB)	Model	Arena Elo	95% CI	Votes	Organization	License	Knowledge Cutoff
1	GPT-4o-2024-05-13	1287	+3/-3	38456	OpenAI	Proprietary	2023/10
2	Gemini-Advanced-0514	1267	+4/-4	32152	Google	Proprietary	Online
2	Gemini-1.5-Pro-API-0514	1265	+4/-4	31574	Google	Proprietary	2023/11
4	Gemini-1.5-Pro-API-0409-Preview	1257	+3/-3	55731	Google	Proprietary	2023/11
4	GPT-4-Turbo-2024-04-09	1256	+3/-3	63339	OpenAI	Proprietary	2023/12
6	GPT-4-1106-preview	1251	+2/-3	82105	OpenAI	Proprietary	2023/4
6	Claude 3 Opus	1249	+2/-2	129749	Anthropic	Proprietary	2023/8
6	GPT-4-0125-preview	1246	+3/-2	75372	OpenAI	Proprietary	2023/12
9	Yi-Large-preview	1239	+3/-3	38660	01 AI	Proprietary	Unknown
10	Gemini-1.5-Flash-API-0514	1231	+4/-4	29900	Google	Proprietary	2023/11
10	Yi-Large	1222	+8/-8	3510	01 AI	Proprietary	Unknown
11	Bard (Gemini Pro)	1208	+7/-5	11853	Google	Proprietary	Online
12	Llama-3-70b-Instruct	1207	+3/-2	131384	Meta	Llama 3 Community	2023/12
13	Claude 3 Sonnet	1201	+3/-2	100517	Anthropic	Proprietary	2023/8

Figura 32: Chatbot Arena Leaderboard [45]

2.3 Proyectos Recientes / Estado del Arte

Los avances en inteligencia artificial de la última década, particularmente en el área del aprendizaje profundo, han tenido una gran influencia en el campo de la robótica. Inicialmente las aplicaciones se centraban en la percepción del robot, aunque recientemente una de las líneas de investigación más populares es el control de movimiento y locomoción de los robots mediante técnicas de aprendizaje por refuerzo (*reinforcement learning*).

Esta evolución se puede observar claramente en el robot *Spot* de la compañía *Boston Dynamics*. Hasta hace muy poco, *Spot* utilizaba técnicas de *deep learning* únicamente para el procesamiento de los datos de sus sensores (cámaras y LiDAR) mientras que su locomoción estaba dirigida por algoritmos de control predictivo por modelo. Recientemente, la programación de *Spot* ha adoptado un modelo híbrido que combina los modelos predictivos con técnicas de aprendizaje por refuerzo [46] (*figura 33*).

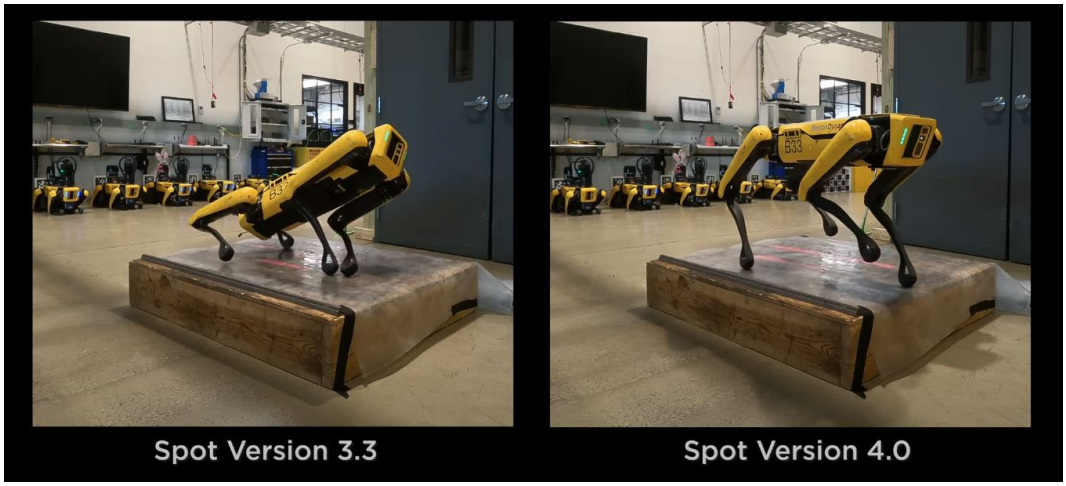


Figura 33: Spot 4.0 utilizando Reinforcement Learning *[46]*

En robótica, el aprendizaje por refuerzo se suele aplicar mediante simulación. Dentro de un entorno virtual simulado, el robot es capaz de explorar el rango de posibles soluciones para lograr un objetivo de movimiento y entre esas posibilidades, encontrar el óptimo.

Otra de las nuevas tendencias que también ha explorado el equipo responsable de *Spot* es la incorporación de *chatbots* impulsados por modelos de IA generativa, como *ChatGPT*, para aumentar las capacidades interactivas del robot [47].

Recientes propuestas de robots humanoides como *Optimus* de Tesla [48] o *Figure 01* [49], también combinan ambas tecnologías. Los modelos de lenguaje actúan como interfaz interactiva del robot, recopilando información mediante voz e imagen (modelos multimodales) y proporcionando además razonamiento de alto nivel. Mientras, las políticas de movimiento, aprendidas mediante *reinforcement learning,* otorgan capacidades de manipulación con un gran nivel de destreza (*figura 34*).

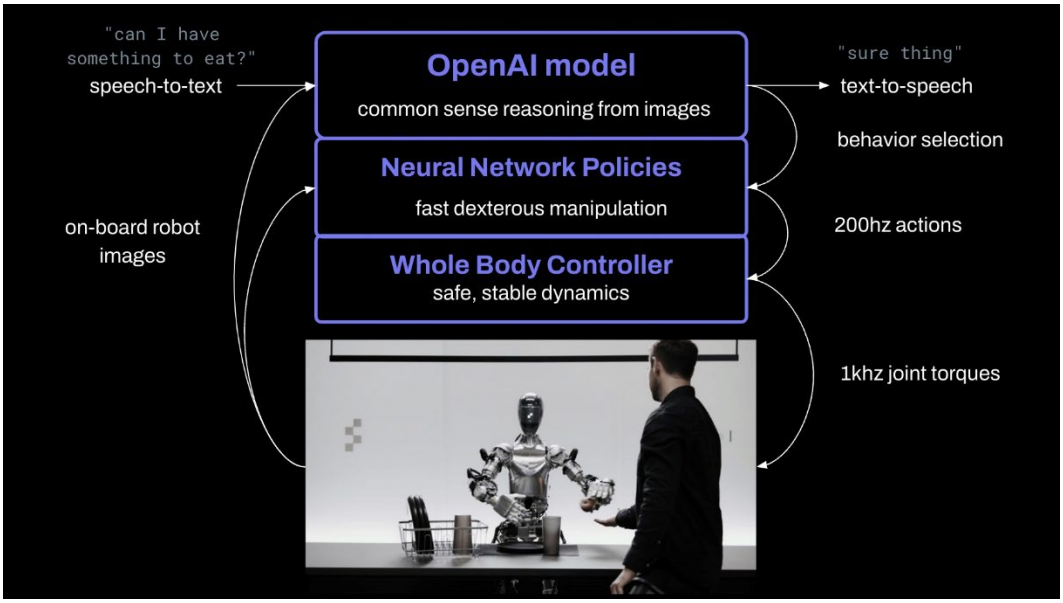


Figura 34: Arquitectura del robot Figure 01 [49]

En general, hay una tendencia que pugna por que el movimiento de los robots no esté programado manualmente, sino que sea aprendido mediante distintas técnicas de aprendizaje automático. Algunos ejemplos incluyen el aprendizaje por refuerzo con retroalimentación humana (*Reinforcement Learning with Human Feedback*) [50] o el aprendizaje por imitación (*imitation learning*) de Mobile ALOHA [51].

Capítulo 3: Herramientas de Desarrollo

En el presente capítulo se introducirán las herramientas utilizadas en el desarrollo del proyecto. Se hará especial hincapié en las herramientas principales del ecosistema de desarrollo del robot NAO, así como en las funciones proporcionadas por la API de OpenAI para el aprovechamiento de los modelos GPT. Se mencionan, asimismo, otras herramientas y tecnologías auxiliares utilizadas.

3.1 Ecosistema del robot

El robot NAO fue presentado inicialmente en 2004 por la compañía francesa *Aldebaran Robotics* posteriormente renombrada *Softbank Robotics*. Tanto el robot como el software acompañante han evolucionado a lo largo de los años hasta llegar a la versión más reciente, NAO V6 (2018).

El ecosistema del robot NAO abarca una serie de componentes y herramientas esenciales que facilitan el desarrollo, configuración, y operación de este robot humanoide. Estos elementos incluyen tanto el hardware específico del robot como las plataformas de software necesarias para programar y controlar sus funciones. Se van a tratar los siguientes componentes:

- Hardware del robot NAO (el robot mismo)
- **Robot Settings:** utilidad de configuración inicial
- **Choregraphe:** entorno de desarrollo mediante diagramas de bloques
- **NAOqi Python SDK:** kit de desarrollo de software en Python

3.1.1 Robot NAO - Hardware

El robot NAO (*figura 35*) es un robot humanoide de 574mm de altura y 25 grados de libertad (DoF). Posee segmentos y articulaciones para realizar prácticamente toda la gama de movimientos humanos, incluido el tren inferior, manteniendo un factor de forma compacto. [52]

Está alimentado por una batería de ion-litio de 62.5Wh que proporciona unos 90 minutos de uso activo. También puede ser utilizado con el cable de carga conectado lo cual extiende su autonomía de forma indefinida y es francamente útil durante el proceso de desarrollo.

En materia de conectividad, el robot dispone de un puerto Ethernet (RJ45) destinado principalmente a realizar la configuración inicial del robot, aunque bien puede ser utilizado para conectar físicamente el NAO a una red. La comunicación principal con el robot se realiza mediante WiFi (IEEE 802.11 a/b/g/n). Esta conexión se utiliza para la instalación y actualización de comportamientos en el robot y también para el control remoto desde un ordenador conectado a la misma red.

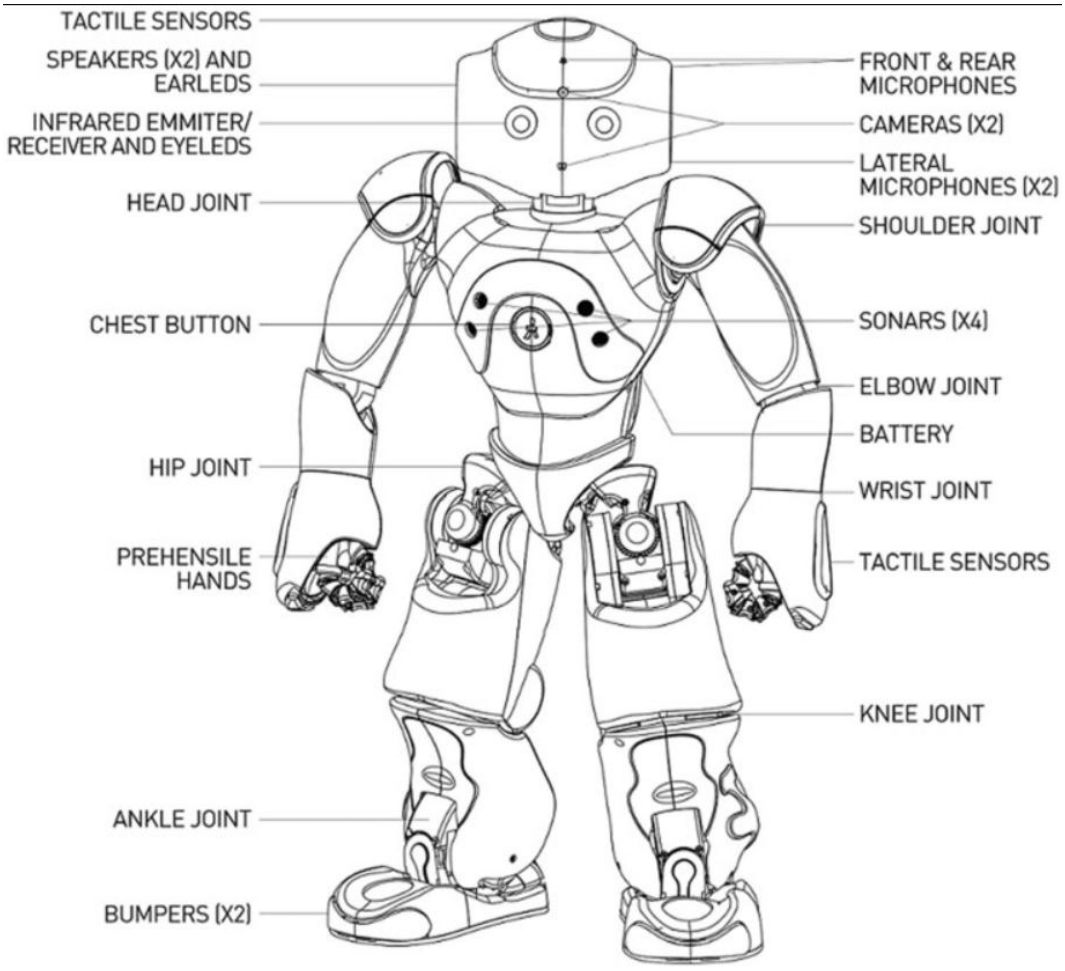


Figura 35: Articulaciones y Sensores del robot NAO

Internamente, el robot implementa un sistema operativo basado en Linux llamado NAOqi. El hardware que lo ejecuta consta de un procesador de cuatro núcleos (Intel Atom E3845, 1.91 GHz), 4 GB de memoria RAM DDR3 y 32 GB de almacenamiento SSD.

Los elementos más importantes en un robot social como NAO son aquellos que le permiten percibir e interactuar con el entorno, es decir, sus sensores y actuadores.

Para controlar su posición y movimiento, NAO dispone de *encoders* rotativos en cada una de las articulaciones, los cuales utilizan sensores magnéticos de efecto Hall. A mayores, posee una unidad inercial (IMU) compuesta de acelerómetro y giroscopio, ambos de 3 ejes, y sensores de presión en las plantas de los pies para el control del equilibrio.

El resto de la suite de sensores está compuesta por:

- Dos cámaras (640*480 @ 30fps / 2560*1920 @ 1fps) en la cabeza alineadas verticalmente que proporcionan visión estéreo y permiten implementar aplicaciones de visión artificial.

- Sensores táctiles (hápticos): tres en la cabeza y uno en el dorso de cada mano. Permiten detectar y responder al contacto físico.

- Cuatro micrófonos omnidireccionales (*figura 36)* y dos altavoces, uno en cada oreja, que permiten el reconocimiento de voz, la identificación de la dirección de un estímulo auditivo y la interacción verbal.

- Iluminación LED en la parte superior de la cabeza, los ojos, alrededor de los altavoces en las "orejas" y en los pies. Su utilidad principal es proporcionar *feedback* al usuario sobre el estado del robot.

- Sensores de ultrasonidos (x2 emisores, x2 receptores) ubicados en el pecho. Permiten medir distancias y detectar obstáculos (*figura 36*).

- Sensores de contacto *bumpers* en los pies para detectar colisiones.

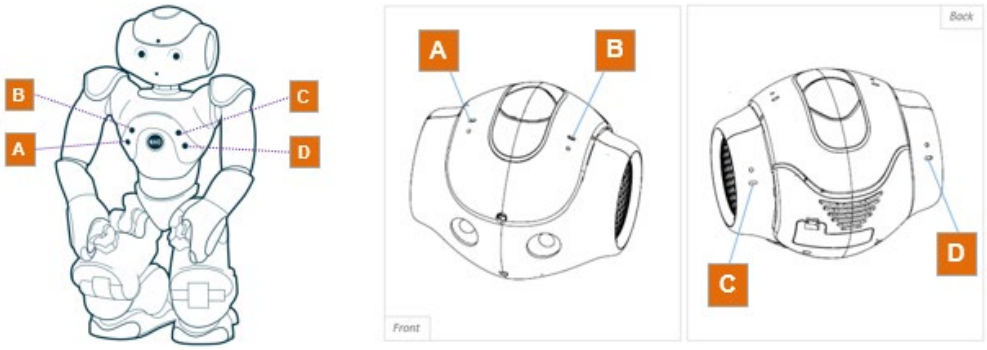


Figura 36: Sensores de ultrasonido (izq.) y micrófonos (dcha.)

3.1.2 Robot Settings

Robot Settings es una herramienta de software que permite una fácil gestión y configuración del robot NAO [53]. Se utiliza primariamente para realizar la configuración inicial del robot y para conectar el robot a nuevas redes wifi.

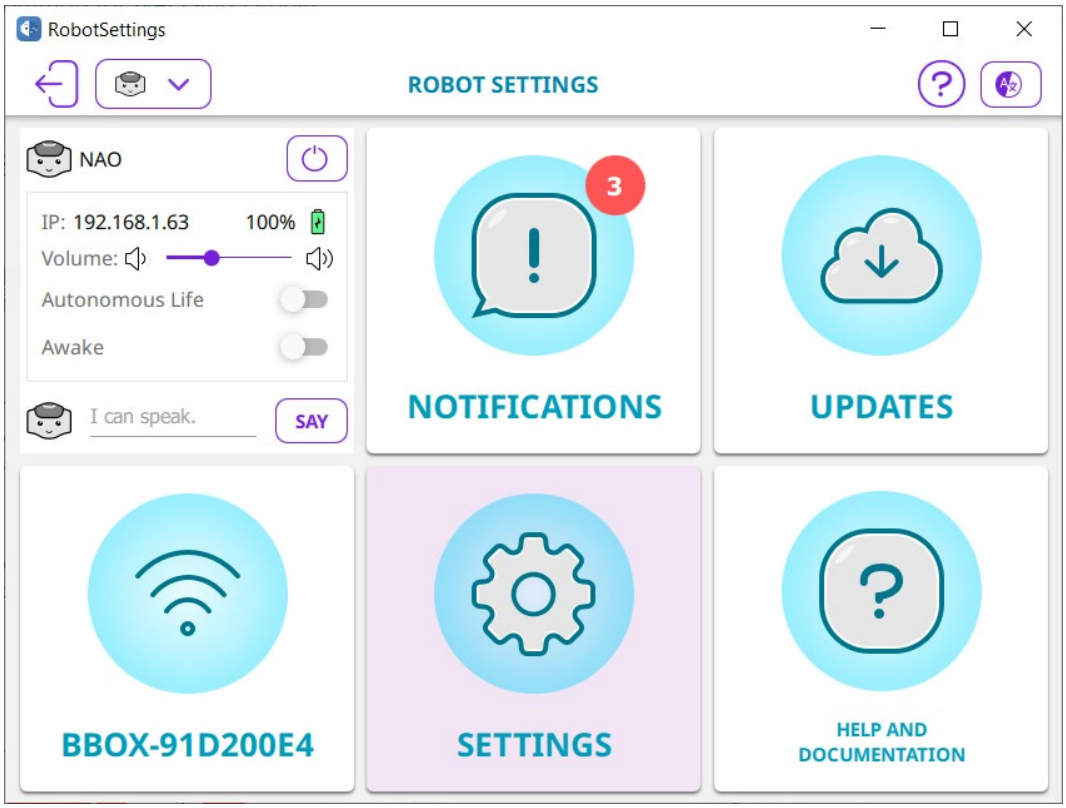


Figura 37: Vista principal de Robot Settings

El menú principal de *Robot Settings* (*figura 37*) proporciona acceso rápido al estado del robot. Muestra información como la dirección IP, el volumen actual o el estado de la función *Autonomous Life*. También permite controles rápidos como encender y apagar el robot, activar o desactivar los motores (*awake*) o hacer que el robot diga una frase corta para comprobar su estado y conexión.

Además, facilita también el acceso a los paneles de notificaciones, actualizaciones de firmware, configuración de la conexión wifi, ajustes avanzados y a la documentación del robot.

3.1.3 Choregraphe

Choreographe (*figura 38*) es un entorno de desarrollo visual integral, diseñado específicamente para la programación y control de los robots NAO y Pepper de *Softbank Robotics*. Esta herramienta facilita la creación de comportamientos y aplicaciones para el robot mediante una interfaz gráfica intuitiva y fácil de usar.

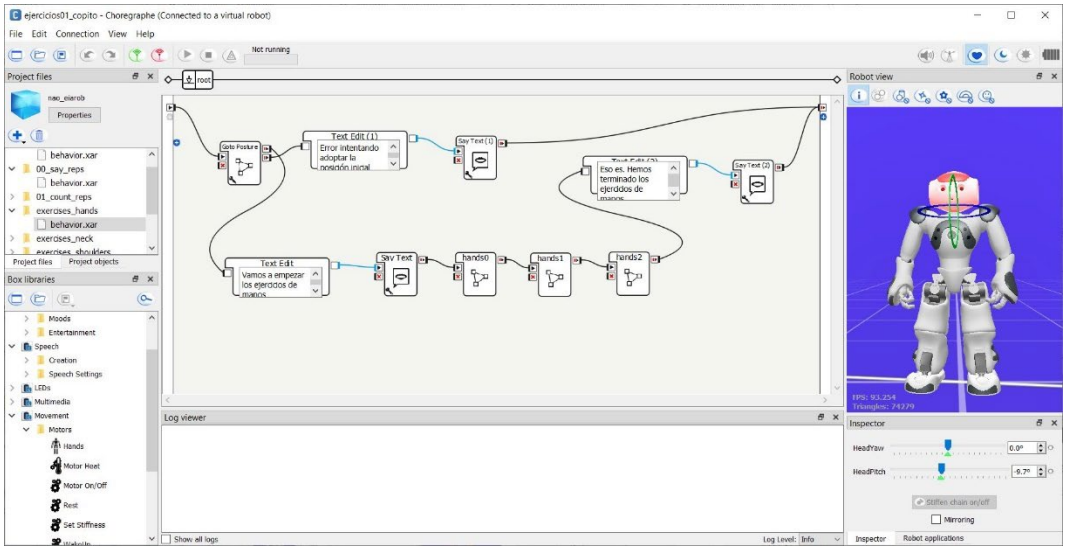


Figura 38: Vista principal de Choregraphe

Utiliza una programación basada en diagramas de bloques interconectados para crear comportamientos (behaviors) del robot. Choregraphe ofrece una biblioteca de bloques predefinidos que representan acciones, eventos, y condiciones, los cuales pueden ser arrastrados y conectados para formar flujos lógicos de comportamiento. Esto permite la creación de aplicaciones interactivas de forma fácil y accesible para desarrolladores novatos [54].

Las capacidades de Choregraphe permiten a los usuarios diseñar animaciones, secuencias de movimiento, patrones de interacción y otros comportamientos para el robot. Se puede conectar el software con un robot simulado (en caso de que no se disponga de uno) o con el robot real, lo que permite monitorizarlo y controlarlo desde la propia herramienta.

Además, Choregraphe proporciona simulación y depuración (debugging) en tiempo real, concediendo a los desarrolladores la posibilidad de probar y ajustar sus programas antes de implementarlos definitivamente, asegurando así que las interacciones y comportamientos se desarrollan correctamente.

3.1.4 NAOqi Python SDK

La empresa *Softbank Robotics* proporciona, además de los programas ya mencionados, kits de desarrollo de software (SDKs) para varios lenguajes de programación y plataformas (*figura 39*).

Programming Languages	Bindings running on		Choregraphe support	
	Computer	Robot	Build Apps	Edit code
Python	✔	✔	✔	✔
C++	✔	✔	⊘	⊘
JavaScript	✔	✔	✔	⊘
ROS	✔	⊘	⊘	⊘

Figura 39: Distintos SDKs disponibles y sus características

Como se puede apreciar en la figura, el SDK de Python [55] es el más polivalente. Permite desarrollar programas que se ejecutan en un ordenador para controlar el robot de forma remota o programas que se ejecutan directamente en el robot; pero también tiene soporte completo en *Choregraphe* donde permite elaborar aplicaciones y también editar el código interno de cada bloque del diagrama.

El SDK de Python permite acceder a todas las funciones de la API de NAOqi [56] para utilizarlas desde una máquina remota, además, permite el desarrollo de módulos de Python que se pueden ejecutar remotamente o directamente en el robot.

NAOqi actúa como un broker, controlando el tráfico con la red y dando acceso a los diferentes módulos que a su vez contienen los métodos utilizados en el código para controlar el robot (*figura 40*).

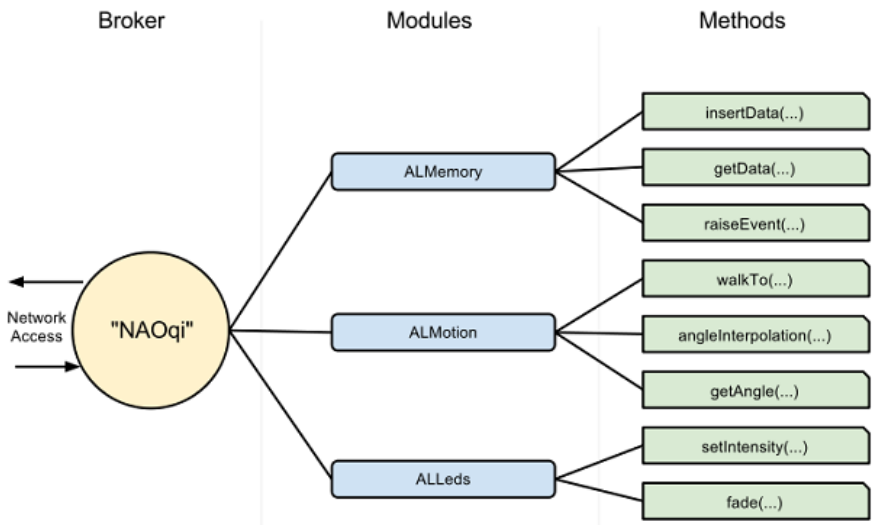


Figura 40: Estructura del framework NAOqi [57]

3.2 API de OpenAI

OpenAI se fundó en 2015 como una organización de investigación en inteligencia artificial sin fines de lucro. No obstante, en 2019 se convirtió en una empresa híbrida, compuesta por una entidad sin ánimo de lucro y otra con fines lucrativos [58].

Su último modelo abierto al público fue GPT-2 lanzado en febrero de 2019 junto a su artículo acompañante [59]. Los modelos posteriores como GPT-3 no fueron divulgados públicamente (arquitectura y pesos) pero se desarrolló una API pública que da acceso a los modelos desarrollados por la compañía.

La API de OpenAI [60] ofrece a los desarrolladores la posibilidad de integrar las capacidades de sus modelos de IA en sus propias aplicaciones. Los servicios de la API son accesibles a través de solicitudes HTTP desde cualquier lenguaje o mediante la librería oficial de Python. También existen múltiples librerías desarrolladas y mantenidas por terceros (community libraries) [61] para utilizar la API con otros lenguajes de programación. La librería oficial de Python ofrece una interfaz simplificada para interactuar con la API, reduciendo la complejidad de manejar directamente las solicitudes HTTP, gestionando la autenticación, la serialización de datos, y la gestión de errores de manera automática.

Los diferentes servicios de la API se agrupan en *"endpoints"*, estos son los distintos tipos de llamadas que puedes hacer a la API para solicitar uno u otro servicio. Internamente, estos servicios funcionan ejecutando los modelos disponibles (*figura 41*):

MODEL	DESCRIPTION
GPT-4o	The fastest and most affordable flagship model
GPT-4 Turbo and GPT-4	The previous set of high-intelligence models
GPT-3.5 Turbo	A fast, inexpensive model for simple tasks
DALL·E	A model that can generate and edit images given a natural language prompt
TTS	A set of models that can convert text into natural sounding spoken audio
Whisper	A model that can convert audio into text
Embeddings	A set of models that can convert text into a numerical form

Figura 41: Modelos de la API de OpenAI

Se dispone de modelos para convertir audio en texto (*Speech To Text, Whisper*) y viceversa (*TTS*), modelos de difusión para generar imágenes a partir de lenguaje natural (*DALL-E*) y modelos para generar *embeddings* a partir de texto. A mayores, obviamente, se dispone de los modelos GPT de generación de texto.

Entre todos los servicios proporcionados por la API, los de mayor interés para este trabajo son los *endpoints* de generación de texto:

- **Chat Completions API:** toma una lista de mensajes como entrada y devuelve, a la salida, un mensaje generado por el modelo de texto seleccionado. Es un formato flexible y fácil de utilizar tanto para conversaciones turnadas como para respuestas únicas. Esta simplicidad lo hace altamente configurable, pudiendo personalizar por ejemplo el manejo y almacenamiento del registro de la conversación.

- **Assistants API:** introducido recientemente, este *endpoint* responde a la necesidad de un servicio más evolucionado para crear asistentes conversacionales personalizados. Integra nativamente funciones como el registro de la conversación o el manejo del contexto a lo largo de la misma. Separa el servicio de generación de texto en tres elementos (*figura 42*): el asistente en sí (*assistant*), el hilo de la conversación (*thread*) y la ejecución de un asistente en un hilo para generar una respuesta (*run*). Este *endpoint* se encuentra todavía en versión beta.

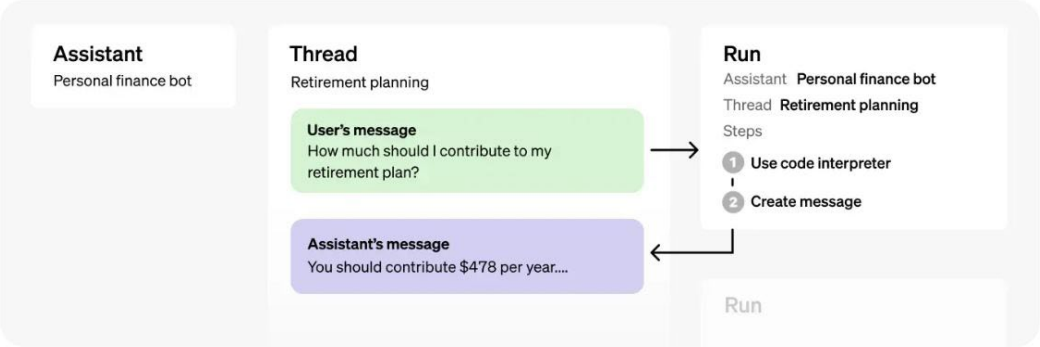


Figura 42: Esquema de la API de Asistentes

Ambos *endpoint* (*figura 43*) tienen características similares y superpuestas. *Chat Completions* proporciona una interfaz más simple y flexible, dejando en manos del desarrollador el ampliarla con nuevas funciones. La API de asistentes es un producto más capaz pero enfocado específicamente en la creación de asistentes. Incorpora muchas de las funciones que los usuarios esperan de un asistente para qué los desarrolladores no tengan que implementarlas por sí mismos.

```
POST https://api.openai.com/v1/chat/completions
```

```
POST https://api.openai.com/v1/assistants
```

Figura 43: Direcciones HTTP de cada endpoint

A través de estos servicios, los modelos subyacentes pueden acceder a "herramientas" (*tools*) para expandir sus capacidades. *Chat Completions* tiene acceso a la herramienta de llamada a funciones (*function calling*). La API de asistentes tiene, además, acceso a las herramientas *"File Search"* y *"Code Interpreter"*.

- ***Function Calling***: se suministra al modelo con la descripción y objetivo de una o varias funciones. A partir de esto, el modelo es capaz de elegir llamar a una de estas funciones como parte de sus respuestas. La API no llama a la función *per se*, sino que genera un objeto JSON que contiene el nombre de la función a llamar y sus argumentos. Utilizando este objeto, el desarrollador puede llamar a la función que quiere el modelo.

- ***File Search***: antes *Knowledge Retrieval*, aumenta las capacidades del asistente con información externa al modelo. Subiendo documentos a la nube, OpenAI los procesa, crea y almacena *embeddings* de ellos. Sobre estos *embeddings* se realizan búsquedas de información relevante que se alimenta al modelo para responder las consultas de los usuarios. Es una forma de ampliar la información que el modelo tiene disponible sin sobrecargar su ventana de contexto (*context window*).

- ***Code Interpreter:*** permite a los asistentes escribir y ejecutar código Python en un entorno de ejecución aislado (*sandbox*). Proporciona la habilidad de ejecutar código de forma iterativa para resolver problemas matemáticos o de programación. También permite al modelo saber si el código falla en ejecutarse para realizar modificaciones sobre él.

En general, la API está muy bien organizada y documentada. Proporciona otras funcionalidades que no se han comentado, como paneles (*dashboards*) para manejar tus propios modelos personalizados o que proporcionan información de uso y facturación. También dispone de un entorno de testeo (*playground*) para probar nuevos modelos e implementaciones, y una página web accesoria que contiene ejemplos y tutoriales llamada *Cookbook* [62].

3.3 Otras Herramientas

3.3.1 Visual Studio Code

Visual Studio Code (VS Code) es un editor de texto gratuito desarrollado por Microsoft. Siendo más flexible y ligero de ejecutar que un IDE completo como Visual Studio, se ha convertido en el entorno de desarrollo de software más popular.

Diseñado para soportar múltiples lenguajes de programación, VS Code ofrece las características básicas de un editor de código como resaltado de sintaxis, autocompletado, ejecución y depuración del código, etc.

Entre las características distintivas de VS Code se encuentran su integración con Git y GitHub (también propiedad de Microsoft) para el control de versiones o su terminal integrada que permite ejecutar comandos y scripts directamente desde el editor. Su mayor ventaja respecto a la competencia es su tienda de extensiones que permite a los usuarios personalizar y expandir las capacidades de VS Code, por ejemplo, añadiendo soporte para nuevos lenguajes.

3.3.2 Obsidian

Obsidian es una herramienta de toma y gestión de notas enfocada a organizar y enlazar la información de manera intuitiva. Destaca por su edición de texto en formato Markdown y los enlaces bidireccionales entre notas permitiendo relacionar ideas y conceptos de manera no lineal. Esta capacidad de enlazado transforma el banco de notas en un grafo de conocimiento (*figura 44*) que puede visualizarse para descubrir patrones y conexiones entre las notas.

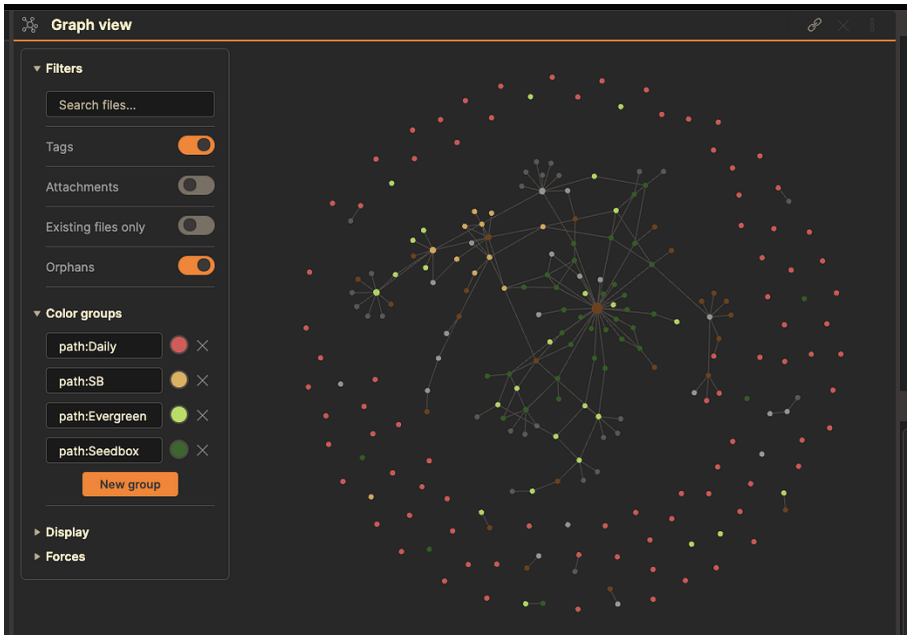


Figura 44: Obsidian - Graph View

Obsidian es altamente valorada por su enfoque sobre privacidad: todas las notas se almacenan localmente como archivos de texto plano lo que garantiza acceso a la información mediante cualquier aplicación e independencia de los servicios en la nube.

3.3.3 Protocolo MQTT y MQTT Explorer

MQTT (MQ Telemetry Transport) es un protocolo de mensajería ligero diseñado para la comunicación máquina a máquina (M2M) y el Internet de las Cosas (IoT). Se caracteriza por su eficiencia en redes con ancho de banda limitado y su capacidad para operar de manera fiable en condiciones de conectividad inestable.

El protocolo MQTT utiliza el modelo de comunicación publicación/suscripción (pub/sub) para conectar los distintos dispositivos. El modelo está basado en *topics*, los dispositivos (clientes) pueden **publicar** mensajes en topics concretos o **suscribirse** a esos topics para recibir los mensajes que se publican en ellos.

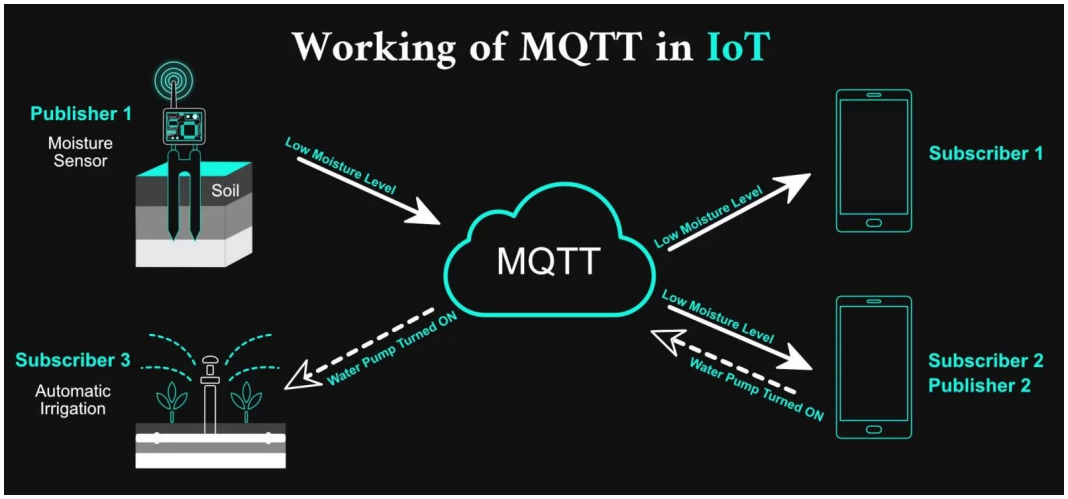


Figura 45: MQTT en IoT

En el modelo *pub/sub* existen tres entidades principales (*figura 45*): publicadores, suscriptores y el broker.

- Publicador: genera y envía mensajes.
- Suscriptor: expresa su interés en recibir mensajes de un *topic.*
- Broker: intermediario que facilita la comunicación entre ambos.

Una ventaja clave del modelo pub/sub es que desacopla la comunicación entre los clientes. Los publicadores y suscriptores son independientes, y se comunican a través del broker sin necesidad de conocer detalles entre sí. Esto hace que el sistema sea más escalable y flexible.

Merece la pena detallar algunos conceptos y terminología:

- **CLIENTE:** cualquier dispositivo o aplicación que utiliza el protocolo MQTT para comunicarse. Los clientes pueden ser publicadores, suscriptores o ambos a la vez (*figura 46*). Cada cliente se conecta al broker mediante una sesión y puede o no suscribirse a uno o varios topics. En función del tipo de conexión, los clientes pueden ser persistentes o no. Los clientes se suelen implementar en el código con la ayuda de una librería, en este caso se ha utilizado *Paho-MQTT* para implementar la comunicación en Python.

- **TOPIC:** es un canal o ruta de comunicación dentro del sistema MQTT que define la categoría de los mensajes. Los publicadores publican mensajes a un topic específico y los suscriptores se registran para recibir los mensajes publicados en los topics que les interesan. Los topics se estructuran jerárquicamente en varios niveles permitiendo organizar el sistema de distribución de los mensajes.

- **BROKER:** servidor central que maneja toda la comunicación en un sistema MQTT. Actúa como intermediario recibiendo todos los mensajes publicados y reenviándolos a aquellos clientes suscritos a los topics correspondientes. El broker es responsable de la gestión de las conexiones de los clientes, el almacenamiento y entrega de los mensajes, la aplicación de las políticas de servicio (QoS) y la administración de la seguridad del sistema.

- **MENSAJE:** unidad de datos que se intercambia entre clientes a través de los topics. Cada mensaje consta de un topic por el que se distribuye y una carga (*payload*) que contiene los datos que se quieren transmitir entre clientes. El protocolo MQTT no impone ningún formato específico para los mensajes. Los mensajes son *byte arrays* y MQTT en sí es agnóstico al contenido de la carga, esta puede ser cualquier secuencia de bytes. Es común utilizar texto plano, formato JSON o datos binarios dependiendo de los requisitos de la aplicación.

- **QoS (*Quality of Service*):** el protocolo MQTT proporciona tres niveles de QoS para el manejo de mensajes. En función del nivel existen más o menos garantías en la entrega del mensaje. Los niveles más altos ofrecen más garantías, pero pueden resultar en un mayor tamaño del mensaje debido a la información de control adicional introducida por el protocolo:

  - QoS 0 (defecto): el mensaje se entrega al menos una vez sin confirmación del receptor. No garantiza que el mensaje llegue.

  - QoS 1: el mensaje se entrega al menos una vez y requiere confirmación por parte del receptor. Si no se recibe confirmación, el mensaje se vuelve a enviar hasta que se confirme la recepción. Esto garantiza la llegada del mensaje, aunque puede provocar duplicados si el receptor no confirma correctamente la llegada.

  - QoS 2: el mensaje se entrega exactamente una vez, con confirmaciones y pasos adicionales para garantizar que no se producen duplicados. Es el nivel más seguro, pero también el más costoso en términos de recursos. Se utiliza para aplicaciones críticas donde la entrega duplicada o pérdida no son aceptables.

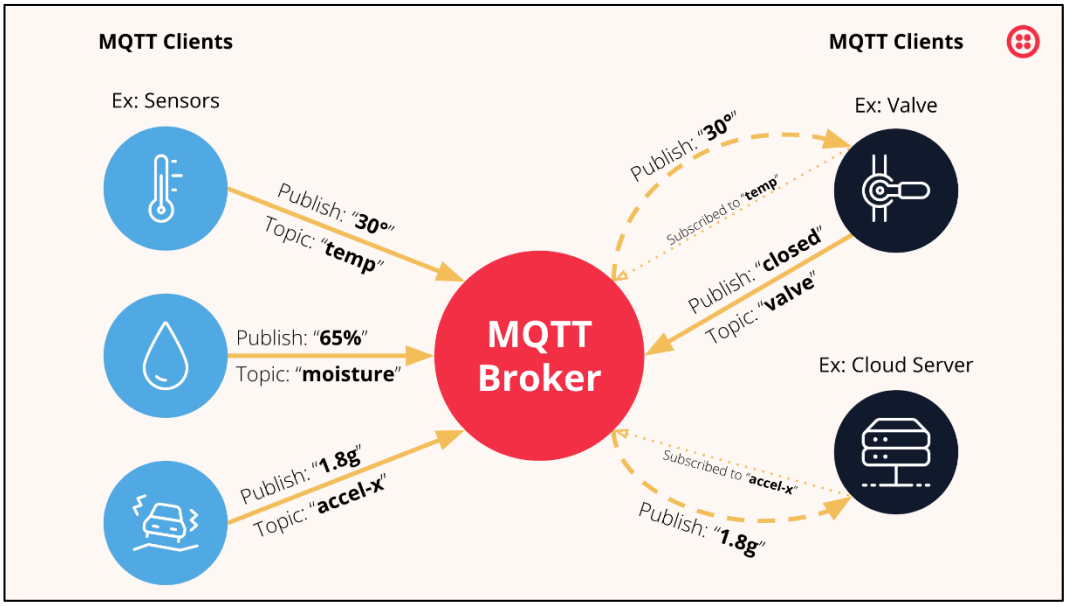


Figura 46: Clientes MQTT publicadores y suscriptores

MQTT es un protocolo del nivel de aplicación proporcionando servicios finales que permiten a las aplicaciones que lo utilizan, comunicarse entre sí. Opera típicamente sobre el *stack* TCP/IP (*figura 47*) por lo que se integra bien con las infraestructuras de red existentes como Internet o redes locales privadas. El protocolo TCP proporciona un canal de comunicación fiable, basado en conexiones que garantiza la entrega ordenada de paquetes lo cual es ideal para los requisitos de mensajería de MQTT. Opcionalmente, el broker se puede configurar con una capa de seguridad TLS o SSL que se sitúa entre MQTT y TCP para proporcionar funciones de autenticación y cifrado.

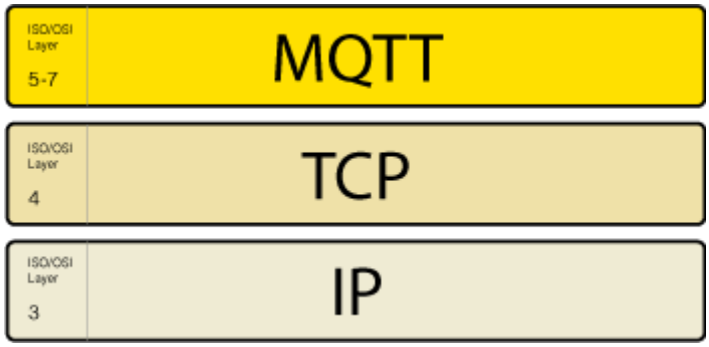


Figura 47: Protocolo MQTT sobre TCP/IP

Otras características importantes que aporta MQTT son:

- **Retención de mensajes:** el broker almacena mensajes durante un tiempo de forma que permanezcan disponibles para nuevos suscriptores. Garantiza que los clientes que se conectan posteriormente puedan recibir el último mensaje de un topic.
- **Sesiones persistentes:** se puede configurar un cliente como persistente, permitiendo que se desconecte y reconecte manteniendo el estado de la comunicación y las suscripciones. Los mensajes con QoS 1 y 2 se le reenvían cuando el cliente persistente se reconecta.
- **Seguridad:** como ya se ha comentado, MQTT soporta mecanismos de seguridad como autenticación mediante usuario y contraseña o cifrado de datos mediante protocolos TLS/SSL.

En resumen, MQTT es un protocolo muy capaz, ligero y versátil; ideal para aplicaciones como la del presente proyecto, proporcionando un medio eficiente y escalable para la transmisión de datos entre dispositivos y sistemas distribuidos.

MQTT EXPLORER

MQTT Explorer es una herramienta gráfica de gestión y monitorización de la comunicación en sistemas MQTT. Proporciona una interfaz intuitiva (*figura 48*) para explorar jerárquicamente los tópicos y visualizar la publicación de mensajes en tiempo real.

Permite conectarse a un broker y filtrar los topics que se quieren monitorizar. Además, permite revisar el historial de mensajes en un topic, publicar mensajes e inspeccionar su carga útil (*payload*), incluyendo la posibilidad de visualizar diferentes formatos como JSON.

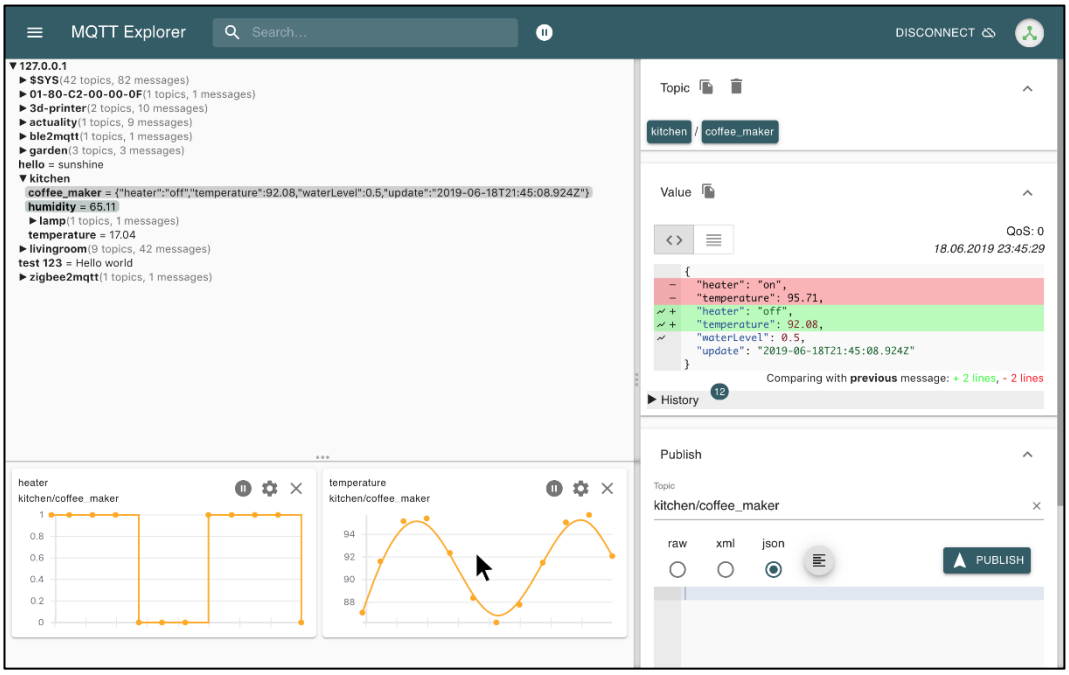


Figura 48: MQTT Explorer

Esta herramienta es especialmente útil en la etapa de desarrollo, ya que facilita la depuración del flujo de datos, la validación de las configuraciones de clientes y brokers, y la visualización de la estructura de tópicos, proporcionando una visión clara y organizada del tráfico MQTT.

Capítulo 4: Desarrollo de los Sistemas de Interacción

El presente capítulo constituye el cuerpo central del trabajo. En él se detalla el desarrollo del proyecto, desde la familiarización con las herramientas utilizadas, pasando por la elaboración y programación del agente conversacional y finalizando con la integración, de todos los elementos desarrollados, en el robot NAO.

Se detallarán elementos como el uso del SDK del robot, el uso de la API de Python para integrar los modelos GPT, el proceso de comunicación entre el ordenador y el robot o la arquitectura general de funcionamiento del sistema.

El desarrollo del proyecto no ha seguido una progresión lineal ni estructurada, fruto de los constantes esfuerzos de documentación, investigación e improvisación para solventar los retos que se iban presentando por el camino. Sin embargo, por motivos de claridad y de cara al lector, se presentará una versión organizada de las actividades realizadas empezando por la configuración inicial del entorno de trabajo, seguido por la toma de contacto con las herramientas utilizadas para, por último, describir los desarrollos que componen el resultado final.

4.1 Configuración Inicial

El entorno de trabajo consta de tres elementos principales: el robot NAO, un ordenador desde el cual se controla el robot, y una red MQTT que conecte los clientes programados, por lo que debe contar con un broker apropiadamente configurado.

4.1.1 Ajustes del Robot

Previamente a proceder con el desempaquetado del robot, se localizó la documentación más reciente (*NAOqi 2.8*) y se siguieron los pasos de la guía de usuario [63] para conocer el procedimiento de configuración inicial.

Se requiere de la herramienta *Robot Settings* para realizar dicha configuración inicial, además de ser también útil más adelante en el desarrollo. Se conecta el ordenador al robot, inicialmente mediante un cable Ethernet para configurar la conexión wifi. Una vez ambos (ordenador y robot) están conectados a la

misma red wifi, se puede controlar el robot desde el ordenador de forma completamente inalámbrica.

El robot ya había estado en uso por parte del centro tecnológico CARTIF, de modo que, como medida de precaución se realizó un restablecimiento a los ajustes de fábrica. Este proceso se puede realizar fácilmente desde un submenú de *Robot Settings* (*figura 49*).

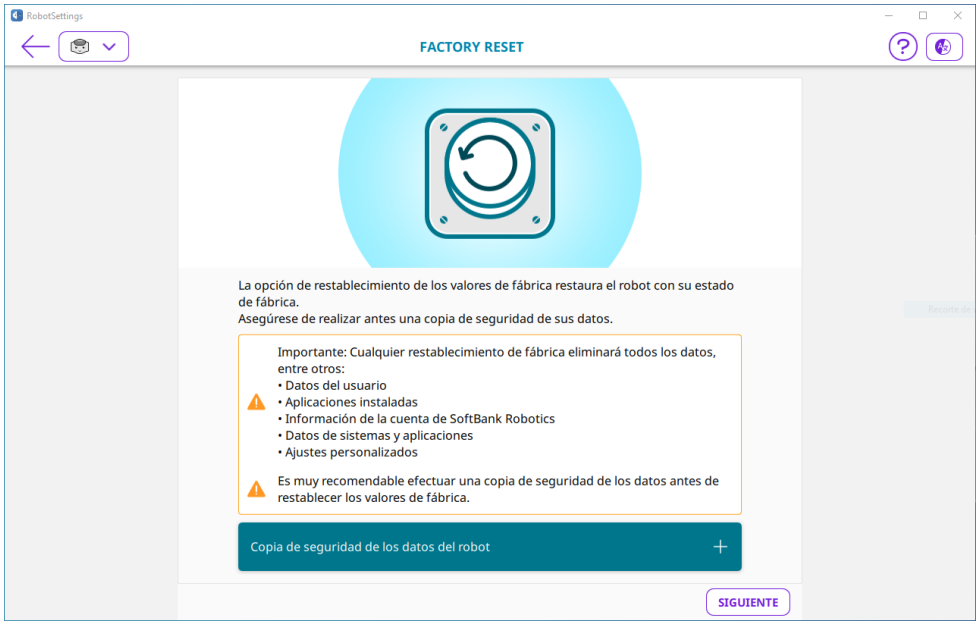


Figura 49: Robot Settings - Factory Reset

El resto de la configuración inicial incluye ajustes como:

- Zona horaria
- Nombre del robot
- Autentificación y contraseña de acceso
- Descarga de actualizaciones
- etc.

La fase final de la configuración es sumamente importante. En ella se instala el *Basic Channel* desde la web de *Softbank Robotics* y los paquetes de idiomas secundarios (español). El *Basic Channel* proporciona capacidades clave al robot como las habilidades de diálogo o la recuperación después de una caída. La suscripción al *Basic Channel* es obligatoria para poder utilizar al completo las características de *Autonomous Life*, más en concreto: ejecutar aplicaciones, usar *trigger sentences* para lanzar actividades o hacer uso del conjunto básico de sonidos del robot [64]. Para más detalle sobre cómo realizar está parte de la configuración, consultar el apartado "*Subscribing to a Channel*" de la documentación [65].

Figura 50: Robot Settings - Lista de robots

Una vez configurado correctamente, el robot aparecerá en la lista de robots disponibles de *Robot Settings* (*figura 50*) y podrá realizarse la conexión. El ordenador debe estar conectado a la misma red wifi que el robot para poder detectarlo. Se puede comprobar el estado de la conexión introduciendo un breve texto en este campo de *Robot Settings* (*figura 51*) y comprobando si el robot dice la frase proporcionada.

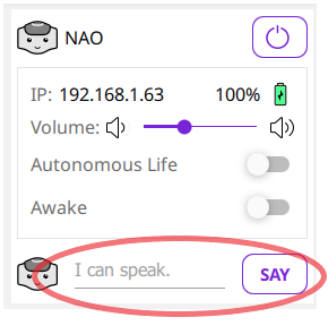


Figura 51: Robot Settings - Speak

Si el robot no es detectado automáticamente, es posible añadirlo manualmente especificando su dirección IP. Para ello, se debe presionar el botón central en el pecho del robot y el robot dirá en voz alta su dirección IP en la red actual. Introduciendo esta IP en la opción "*manually add a NAO*" se consigue establecer la conexión con el robot.

Por último, merece la pena comprobar también que es posible conectarse al robot desde la herramienta *Choregraphe*. En la parte superior de la interfaz, o a través del menú *connection* > *connect to...* se accede a un menú de conexión similar al de *Robot Settings*, donde se puede seleccionar uno de entre una lista de robots o especificar una dirección IP (*figura 52*).

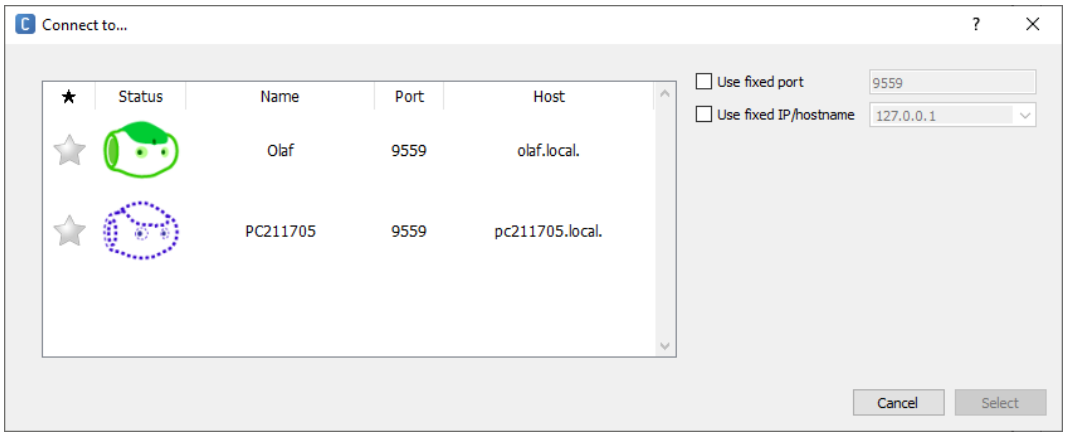


Figura 52: Choregraphe - Conectarse a un Robot

4.1.2 Instalación de las herramientas de software

En este apartado se detalla:

- Descarga e instalación de *Robot Settings* y *Choregraphe.*
- Descarga e instalación de Python (versiones apropiadas).
- Instalación del SDK de Python 2 para NAOqi.
- Instalación del resto de librerías necesarias.

La instalación de *Robot Settings* y *Choregraphe* es trivial. En la documentación del robot [66] se pueden encontrar los enlaces para la descarga de estas herramientas. Sin más que seleccionar el instalador (.exe) para el sistema operativo correspondiente (en nuestro caso *Windows* 10), los programas se instalan a través de un asistente de instalación (*wizard*) como cualquier otro software para Windows.

La instalación del lenguaje de programación Python para el desarrollo del robot presenta un gran problema. El SDK de NAOqi para Python requiere la versión de Python 2.7 como se indica en la guía de instalación [67]. La librería oficial de OpenAI para Python requiere una versión Python 3.7.1 o superior, como se indica en la guía de inicio rápido (*figura 53*) [68].

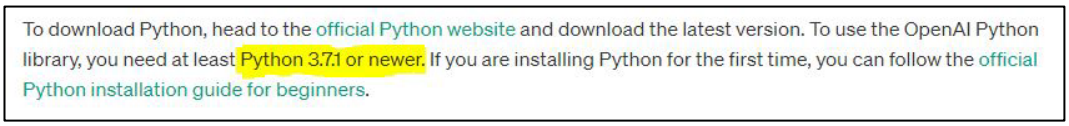

To download Python, head to the official Python website and download the latest version. To use the OpenAI Python library, you need at least Python 3.7.1 or newer. If you are installing Python for the first time, you can follow the official Python installation guide for beginners.

Figura 53: Extracto del Quickstart Tutorial (API de OpenAI) [68]

Como solución, se fraccionará la programación en, al menos, dos procesos diferentes. El acceso y consulta a la API de OpenAI para obtener respuestas de la conversación se realizará con Python 3, mientras, el control remoto del robot a través del SDK de NAOqi se programará en Python 2 (2.7).

La instalación y coexistencia de estas dos versiones de Python en un mismo sistema no es sencilla, por ello se ha redactado una guía de instalación que se adjunta en los anexos (*guía de instalación – python software*). La guía detalla la instalación y configuración paso a paso de todo el entorno de desarrollo de software para el proyecto, incluyendo las 2 versiones de Python requeridas. Como resultado de la instalación, cada versión de Python cuenta con un alias distinto, de forma que cada *script* se puede ejecutar con el intérprete de Python que corresponde a su programación (*figura 54*).

```
C:\Users\alvaro\Desktop>py --version
Python 3.11.7

C:\Users\alvaro\Desktop>python3 --version
Python 3.11.7

C:\Users\alvaro\Desktop>python --version
Python 2.7.18
```

Figura 54: Configuración de los alias de Python

Inicialmente se desconocía cómo se iba a realizar la comunicación entre los scripts que son Python 2 y los que son Python 3. Se decidió entonces por realizar las dos instalaciones de Python directamente sobre el sistema operativo (Windows 10, en este caso) ya que era lo que generaría menos problemas de cara a la comunicación. Revisado el enfoque de comunicación adoptado, es perfectamente viable utilizar diferentes entornos de Python (*Python Environments*) a través de algunas de las herramientas populares como conda, venv o pyenv.

La instalación del SDK de NAOqi se realiza siguiendo las instrucciones proporcionadas en la documentación [67]. Requiere descargar los archivos de la librería, ubicarlos en el directorio de la instalación de Python 2 y añadir una nueva variable de entorno. Si se dispone del robot NAO, se puede comprobar la instalación ejecutando un pequeño *script* de prueba (*figura 55*):

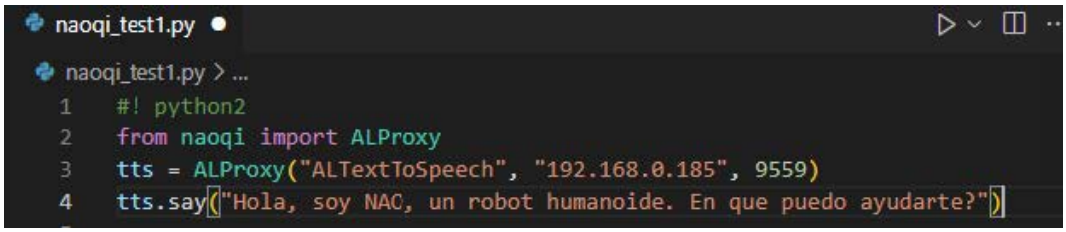


Figura 55: Script de comprobación - NAOqi

Las dos versiones de Python instaladas tienen cada una asociada un gestor de paquetes (pip) distinto. Esto hace que podamos administrar de forma independiente las librerías que necesita cada versión. Se adjunta, en los anexos, los dos archivos de requerimientos (*requirements.txt*) para cada versión de Python de forma que se pueda replicar cómodamente el entorno de programación utilizado en este proyecto.

La librería oficial de OpenAI para utilizar la API se instala a través de pip como cualquier otra librería de Python [68]. Se recomienda guardar la clave de la API (*API KEY*) como variable de entorno, de esta forma no hace falta introducirla en cada programa y además permanece oculta. De igual forma, se puede comprobar la correcta instalación de la librería ejecutando un pequeño programa de prueba (*figura 56*).

gpt_install_test.py

```python
#! python3
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You are a poetic assistant, skilled in expl
    {"role": "user", "content": "Compose a poem that explains the concept of r
  ]
)

message = completion.choices[0].message
print(message.content)
```

Figura 56: Script de comprobación - OpenAI API

4.1.3 Broker MQTT

La comunicación mediante MQTT requiere de un broker. El broker toma la forma de un proceso implementado localmente o en una máquina remota. En el entorno del laboratorio se ha utilizado el broker dispuesto para el desarrollo del proyecto EIAROB, implementado en un Intel NUC. Para poder continuar con el trabajo en el domicilio personal se han utilizado los servicios de un servidor MQTT público de testeo como es "*test.mosquitto.org*".

4.2 Familiarización con el robot

Durante los primeros días se realizó una toma de contacto con el robot y con las capacidades que posee de fábrica. Además, se analizó más en detalle el funcionamiento interno de estas capacidades a nivel de programación para poder manipularlas e integrar las nuevas funcionalidades desarrolladas.

4.2.1 Interacción con el robot

La principal vía de interacción [69] con el robot es el diálogo. Si bien NAO posee otros sensores, principalmente de contacto y distancia, estos se utilizan una vez iniciadas actividades concretas, por ejemplo, los sensores de ultrasonido evitan que el robot colisione cuando se está desplazando.

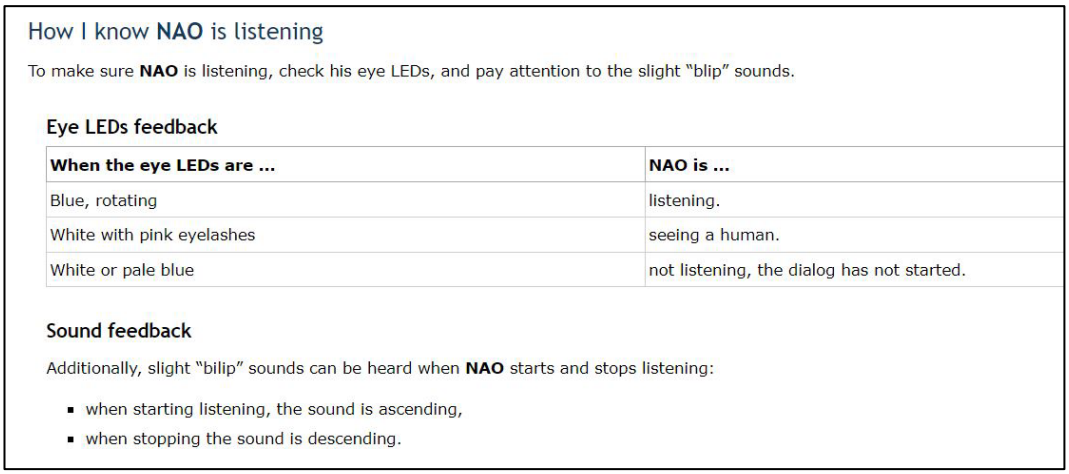
Cuando NAO se "despierta" (siempre que *Autonomous Life* este activo) comienza a procesar estímulos y a buscar al usuario. El robot es capaz de detectar estímulos sonoros (micrófonos) y de movimiento (cámaras), y orienta su cabeza en la dirección de dichos estímulos comprobando si el estímulo se corresponde con un ser humano. En caso afirmativo, NAO intenta mantener el contacto visual con el usuario y, en caso de no detectar a una persona, sigue buscando. Cuando un humano es detectado, los ojos de NAO (LEDs) muestran una "pestaña" rosada (*figura 57*).

Figura 57: Ojos de NAO cuando detecta un humano

En este estado, NAO está listo para iniciar una conversación y procesar los comandos de voz que reciba del usuario. Al interaccionar con NAO se debe prestar especial atención en:

- Mantenerse cerca y asegurarse de que el robot te está detectando.
- Comprobar si el robot está escuchando (indicadores de los ojos).
- Usar frases cortas y sin titubeos. Evitar las pausas pues suelen ser malinterpretadas como el final de la frase.

Como se puede entrever, mantener una conversación fluida con el robot puede llegar a ser todo un reto. Esto es debido a la dificultad de establecer debidamente los turnos de la conversación. Para ello se necesita procesar debidamente la voz del usuario, interpretando correctamente las pausas naturales del habla y que el usuario sea capaz de discernir cuándo el robot está escuchando de cuándo está respondiendo. En la programación original del robot, esto se intenta conseguir mediante un código de colores en los ojos de NAO que proporcione *feedback* del estado del robot (*figura 58*).



How I know **NAO** is listening

To make sure **NAO** is listening, check his eye LEDs, and pay attention to the slight "blip" sounds.

Eye LEDs feedback

When the eye LEDs are ...	**NAO is ...**
Blue, rotating	listening.
White with pink eyelashes	seeing a human.
White or pale blue	not listening, the dialog has not started.

Sound feedback

Additionally, slight "bilip" sounds can be heard when **NAO** starts and stops listening:

- when starting listening, the sound is ascending,
- when stopping the sound is descending.

Figura 58: Feedback de la conversación con NAO [69]

A través de esta interfaz de voz, se realiza la interacción con el robot y se accede a las funciones básicas que trae instaladas de fábrica. El conjunto de comandos y funciones que realiza NAO se puede encontrar en la documentación [70]. Se resume en:

- Frases predefinidas de saludo, despedida, etc.
- Información básica (hora, día y año).
- Proveer información sobre el robot (batería, volumen, dirección IP...).
- Cambiar dichos ajustes del robot.
- Movimientos básicos del robot, adoptar posturas.
- Lanzar aplicaciones y/o comportamientos desarrollados por terceros e instalados en el robot.

Las capacidades de diálogo del robot son bastante limitadas. Consisten en el reconocimiento de frases predefinidas, con poca flexibilidad: a menudo una pequeña variación en la formulación de la frase resulta en ésta no siendo reconocida. Es difícil decir si esto es debido a que la frase no está dentro del conjunto aceptado o a que el sistema de STT (*speech to text*) ha fallado, no traduciendo correctamente los sonidos recibidos. En cualquier caso, estas capacidades están lejos del rendimiento actual de los sistemas de reconocimiento de voz (STT) y de la capacidad de razonamiento y diálogo de los nuevos modelos de lenguaje (LLMs).

Todas estas capacidades son fáciles de replicar e incluso mejorar, simplemente integrando *ChatGPT* en el robot. La mayoría de las funcionalidades de conversación son ya superiores en *ChatGPT*. El resto de las funciones mencionadas anteriormente se pueden implementar proporcionando al modelo GPT información sobre el estado del robot y acceso al control del mismo a través del SDK de Python.

4.2.2 Autonomous Life

Autonomous Life es el componente que hace que NAO esté "vivo", se mueva de forma natural y responda a los estímulos del entorno. Es responsable de los comportamientos autónomos del robot como el rastreo de personas, o el sutil movimiento del robot cuando está ocioso. Todos estos aspectos se manejan de forma automática por *Autonomous Life* sin que el desarrollador tenga que preocuparse por implementarlos.

El enfoque propuesto por *Softbank Robotics* para el desarrollo del robot gira en torno a *Autonomous Life* como elemento central. Siguiendo este enfoque, las capacidades del robot se extienden desarrollando "actividades" que pueden ser solitarias o interactivas y que se lanzan de forma autónoma por parte de *Autonomous Life* o mediante *trigger sentences* (frases que desencadenan el lanzamiento de una actividad).

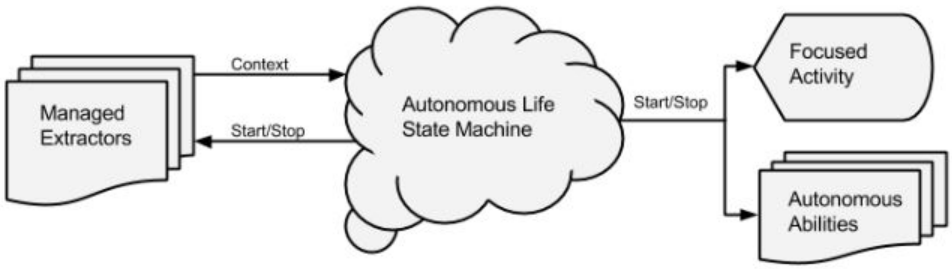


Figura 59: Diagrama de funcionamiento de Autonomous Life

Basándose en contexto extraído del entorno (*Managed Extractors*) como la detección de movimiento o la percepción de personas, la máquina de estados de *Autonomous Life* (*figura 59*) inicia y/o detiene tanto las actividades como las habilidades autónomas (*Autonomous Abilities*) [71].

Las habilidades autónomas (*Autonomous Abilities*) [72] son las responsables de que el robot se sienta "vivo" en todo momento. Controlan aspectos como el "parpadeo" de los ojos del robot (LEDs), los ligeros movimientos en segundo plano del robot o el mantenimiento del contacto visual (*figura 60*). Estas habilidades se pueden gestionar, activándolas o desactivándolas, en función de qué capacidades se desea que tenga el robot en un momento determinado.

List of Autonomous Abilities ¶

Autonomous Ability	Description
AutonomousBlinking	Enables the robot to make its eye LEDs blink when it sees someone and when it is interacting.
BackgroundMovement	Defines which slight movements the robot does autonomously when its limbs are not moving.
BasicAwareness	Allows the robot to react to the environment to establish and keep eye contact with people.
ListeningMovement	Enables some slight movements showing that the robot is listening.
SpeakingMovement	Enables to start autonomously movements during the speech of the robot.

Figura 60: Autonomous Abilities [72]

4.2.3 Desarrollo con NAOqi

Como se introdujo en el capítulo 3 (NAOqi Python SDK) la programación en NAOqi consta de módulos los cuales contienen los métodos (funciones) para controlar el robot.

Para programar utilizando el *NAOqi Framework* [57] se crea un objeto de tipo *ALProxy* que proporciona una conexión directa con un módulo determinado, dando acceso a todos los métodos contenidos en dicho módulo (*figura 61*):

ALProxy is an object that gives you acces to all the methods or the module your are going to connect to.

class ALProxy(*name*, *ip*, *port*)

- name - The name of the module
- ip - The IP of your robot
- port - The port on which NAOqi listens (9559 by default)

Every method of the module are directly accessible through the object, for instance:

```
almemory = ALProxy("ALMemory", "nao.local", 9559)
pings = almemory.ping()
```

Figura 61: Objeto ALProxy

En el ejemplo de la *figura 61*, se crea un proxy al módulo "*ALMemory*" el cual controla el acceso a la memoria del robot. Utilizando el proxy, se accede al método ".ping()" para comprobar la conexión.

La estructura básica del código que utiliza el *NAOqi Framework* para controlar el robot es la siguiente (*figura 62)*:

1. Se importa la clase *ALProxy* del módulo *naoqi.*
2. Se crea un proxy (objeto *ALProxy*) al módulo/s deseados.
3. Se utiliza el proxy para acceder a los métodos del módulo.

```
from naoqi import ALProxy
tts = ALProxy("ALTextToSpeech", "<IP of your robot>", 9559)
tts.say("Hello, world!")
```

Figura 62: Ejemplo de uso ALProxy

A partir de estas bases, y consultando la documentación de la API [56], se obtiene control sobre todas las facetas del robot. Existen conceptos más avanzados como suscribirse y reaccionar a eventos o programación asíncrona de las acciones del robot. Para más información referirse a la documentación de la API de Python [55] [73] y a los tutoriales [74] y ejemplos de uso del SDK de Python [75].

qi Framework

Existe un *framework* más moderno y avanzado denominado *qi Framework* [76] que añade varias características de alto nivel para desarrollar aplicaciones completas utilizando los servicios básicos de la API de NAOqi [56]. A pesar de tener una curva de aprendizaje más pronunciada, se recomienda el uso de *qi Framework* cuando se requiera una programación más compleja.

El *qi Framework*, proporciona algunas novedades como el manejo de sesiones de conexión al robot, mejoras en el uso de llamadas asíncronas y simplificaciones a la hora de suscribirse y reaccionar a eventos del robot. Afortunadamente, es relativamente sencillo migrar del *framework* antiguo de NAOqi al nuevo *qi Framework* [77]. Se puede obtener una sesión del nuevo *framework* directamente de un objeto *ALProxy* (*figura 63*):

Getting a Session from a proxy

This section explains how to get a `qi.Session` from **NAOqi**.

This can be used in **NAOqi** script or in **Choregraphe** behaviors.

```
import naoqi

#imagine you have a NAOqi proxy on almemory
mem = naoqi.ALProxy("ALMemory", "127.0.0.1", 9559)

#get a qi session
ses = mem.session()

#then you can use the session like you want
```

Figura 63: Obtener una qi.Session de un Proxy

Independientemente de las abstracciones y objetos de alto nivel proporcionados por el nuevo *framework*, el funcionamiento es muy similar. Cambia simplemente la forma de acceder a los módulos: en lugar de proxys, se accede a ellos como servicios dentro de una sesión (*qi.Session*). El procedimiento a seguir es:

1. Se importa el módulo *qi* de Python.
2. Se crea un objeto de tipo *qi.Session.*
3. Se conecta la sesión a una dirección IP y puerto.
4. Se accede a los módulos mediante llamadas *"session.Service()".*

En el siguiente ejemplo (*figura 64*), similar al de la *figura 62*, se accede al módulo "*ALTextToSpeech"* mediante sesiones del *qi Framework* para utilizar la función "*ALTextToSpeech.say()*".

hola_qi_framework.py > ...

```python
#! python2
# -*- coding: utf-8 -*-

import qi
import sys

session = qi.Session()
session.connect("tcp://192.168.0.185:9559")

tts = session.service("ALTextToSpeech")
ams = session.service("ALAnimatedSpeech")

tts.say("Hola, soy NAO")
# ams.say("Hola, soy NAO")

session.close()
sys.exit(0)

```

Figura 64: Ejemplo qi Framework

A lo largo del desarrollo del proyecto se han utilizado pequeños scripts como este para familiarizarse con la programación del robot y también para probar ciertas funcionalidades de forma aislada antes de integrarlas en programas más complejos.

A continuación (*figura 65*) se muestra, a modo de ejemplo, uno de los scripts desarrollados. En este caso, *voice_params.py* se utiliza para probar los distintos parámetros de configuración de la voz del robot.

voice_params.py > ...

```python
#! python2
# -*- coding: utf-8 -*-

# get voices
# get available params
# default phrase
# tweak params - trial and error


from naoqi import ALProxy
text_speech = ALProxy("ALTextToSpeech", "192.168.0.185", 9559)
anim_speech = ALProxy("ALAnimatedSpeech", "192.168.0.185", 9559)

voices = text_speech.getAvailableVoices()
current_voice = text_speech.getVoice()
print("Available voices: ")
print(voices)
print("Current voice: {}".format(current_voice))
volume = text_speech.getVolume()
print("Current volume: {}".format(volume))


list_of_params = ['pitchShift', 'doubleVoice', 'doubleVoiceLevel',
                  'doubleVoiceTimeShift', 'defaultVoiceSpeed', 'speed']
# list all params
for param in list_of_params:
    print("param {}: {}".format(param, text_speech.getParameter(param)))


# Modify params
text_speech.setVoice("Maria22Enhanced")
text_speech.setParameter("pitchShift", 1.1)
#text_speech.setParameter("speed", 90)
text_speech.setParameter("doubleVoice", 1)
text_speech.setParameter("doubleVoiceTimeShift", 0.2)

# say phrase with modified params
msg = "Hola soy NAO, Estoy aquí para ayudarte."
text_speech.say(msg)

```

Figura 65: Script voice_params.py

4.3 Desarrollo en Choregraphe

La herramienta *Choregraphe,* presentada en el apartado 3.1.3, proporciona un entorno de desarrollo integrado para el robot NAO. Permite conectarse y controlar un robot físico o trabajar sobre un robot virtual en caso de que se carezca de uno.

Los proyectos de *Choregraphe* se componen de uno o varios comportamientos (*behavior*) desarrollados para el robot. Tras el desarrollo, la herramienta permite cargar los comportamientos en el robot, lanzarlos remotamente y monitorizar el robot en tiempo real para comprobar su ejecución.

La programación se realiza de forma gráfica mediante diagramas de bloques. *Choregraphe* dispone de una biblioteca de bloques predefinidos que representan diversos elementos básicos para la programación del robot, desde movimiento y sensorización hasta estructuras lógicas de control de flujo y retardos (*delays*).

Los bloques se arrastran al panel principal y se interconectan formando un flujo lógico de ejecución. Cada comportamiento (*behavior*) debe tener un punto de entrada que marca el comienzo de la ejecución y un punto de salida que determina la finalización del comportamiento. Entre estos dos puntos, los bloques se enlazan mediante señales que conectan la finalización de un bloque con el arranque del siguiente. Las señales pueden ser de varios tipos en función de la información que porten (*figura 66*).

Input	Output	Type	Description
		**Bang**	Represents a **simple event**. This type of I/O does not carry any data with it, only the information that it is stimulated.
		**Number**	Represents an **event carrying a data**. This data is either a number (float or int) or an array of numbers.
		**String**	Represents an **event carrying a data**. This data is either a string or an array of strings.
		**Dynamic**	Represents either a **simple event** (as the Bang type) or an **event carrying a data**. This data (if any) is either a number (float or int), a string or an array of numbers, strings and arrays.

Figura 66: Tipos de señales en Choregraphe

A mayores, el diagrama se puede estructurar de forma jerárquica mediante bloques de tipo “diagrama”. En el diagrama principal de alto nivel (root) pueden existir bloques que contienen a su vez otros diagramas, anidándose la estructura de forma indefinida. Existen otros tipos de bloques que pueden contener diagramas en su interior, como el bloque “timeline” que se explicará más adelante.

Uno de los argumentos en favor de usar Python como lenguaje en este proyecto, es su total integración con *Choregraphe* (*ver figura 39*). De hecho, cada bloque del diagrama no es más que un objeto (clase) de Python por dentro. En la librería de bloques, se puede encontrar el bloque "*Python Script*" (*figura 67*) que proporciona la plantilla básica para desarrollar un bloque personalizado simplemente modificando el script interno.

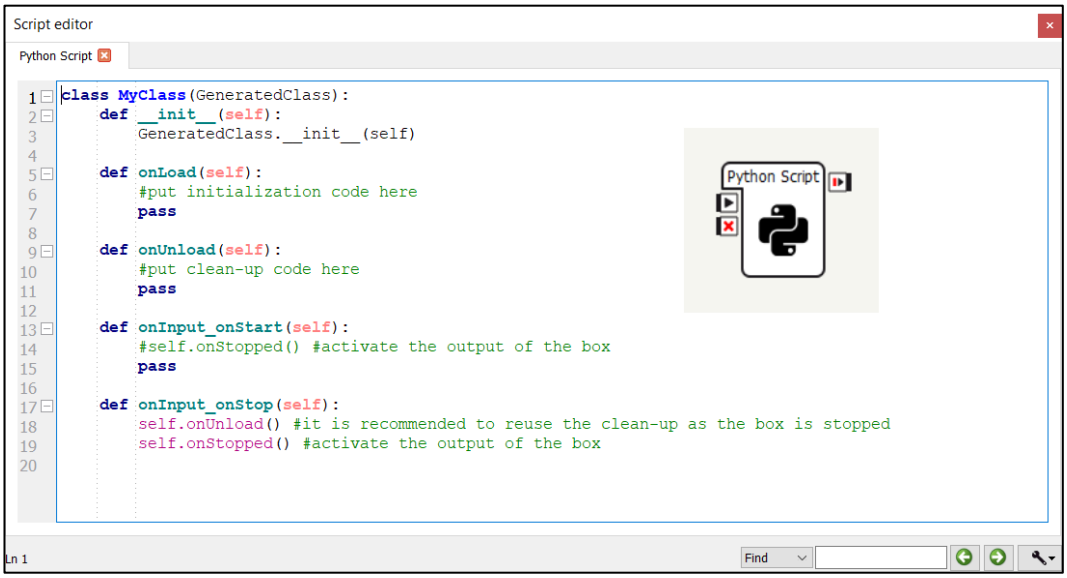


Figura 67: Bloque "Ptyhon Script"

El comportamiento del bloque viene definido por una clase personalizada "*MyClass*" que hereda de la clase "*GeneratedClass*" y trae definidas algunas funciones por defecto:

- **__init__:** inicializador de la clase.
- **onLoad:** se ejecuta cuando el bloque se carga en memoria, es decir, cuando se empieza a ejecutar el comportamiento (behavior).
- **onUnload:** se ejecuta cuando se descarga el bloque de la memoria, es decir, al finalizar la ejecución del comportamiento (behavior).
- **onInput_onStart:** se ejecuta cuando se estimula la entrada "onStart".
- **onInput_onStop:** se ejecuta cuando se estimula la entrada "onStop".

A partir de esta estructura básica se puede implementar cualquier tipo de programación que se ajuste al modelo de la clase. Se pueden importar librerías, definir funciones propias, definir nuevas entradas/salidas y por supuesto activar las salidas del bloque cuando corresponda. [78] [79]

Choregraphe permite también inspeccionar y modificar el código subyacente de cualquiera de los bloques de la librería. Se puede aprender bastante de la programación en NAOqi analizando como están implementados los distintos bloques.

En la *figura 68* se muestra, como ejemplo, el código del bloque "*Apply Posture*" que utiliza el módulo "*ALRobotPosture*" para llevar al robot a una postura determinada. La postura deseada se especifica mediante los parámetros del bloque y su valor se obtiene con el comando "*self.getParameter("Name")*".

Figura 68: Bloque Apply Posture

Durante el desarrollo del proyecto, se han realizado varios diagramas de prueba para familiarizarse con el uso de *Choregraphe*. Esto ha permitido conocer los bloques disponibles en la librería, la forma de trabajar y las posibilidades de la herramienta. Para una explicación en detalle del funcionamiento de *Choregraphe*, referirse a la documentación [54] y tutoriales [80].

A continuación, se muestran un par de ejemplos de los diagramas de prueba desarrollados para ilustrar (de forma genérica) el funcionamiento y las posibilidades de *Choregraphe* en el desarrollo de comportamientos para el robot NAO.

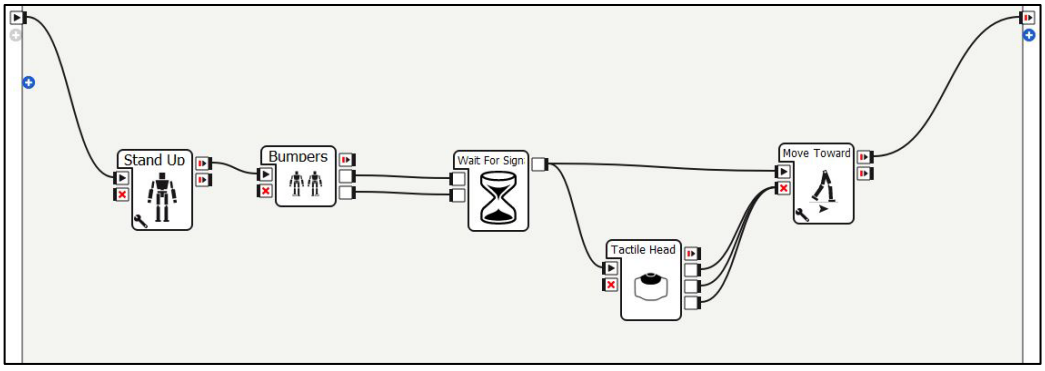


Figura 69: Ejemplo Choregraphe 1

En el ejemplo de la *figura 69*, el robot se pone de pie (si no lo estaba ya), espera a que se activen ambos sensores de los pies (*bumpers*) y en ese momento se pone a caminar (bloque *Move Toward*). El movimiento se detiene cuando se presiona cualquiera de los tres botones táctiles de la cabeza (*Tactile Head*).

Funcionamiento de cada bloque:

- ***Stand Up:*** el robot intenta adoptar la postura predefinida "Stand"
- ***Bumpers:*** detectar las lecturas de los sensores (*bumpers*) en los pies del robot. Las dos salidas corresponden una a cada pie.
- ***Wait For Signals:*** espera a que ambas entradas sean estimuladas para estimular la salida.
- ***Move Toward:*** hace que el robot se mueva en la dirección especificada en los parámetros del bloque. Se debe detener la ejecución del bloque para detener al robot.
- ***Tactile Head:*** detecta si se presionan los botones táctiles de la cabeza del robot. En este caso, cualquiera de los tres botones estimula la entrada *onStop* de "*Move Toward*" y detiene el movimiento el robot.

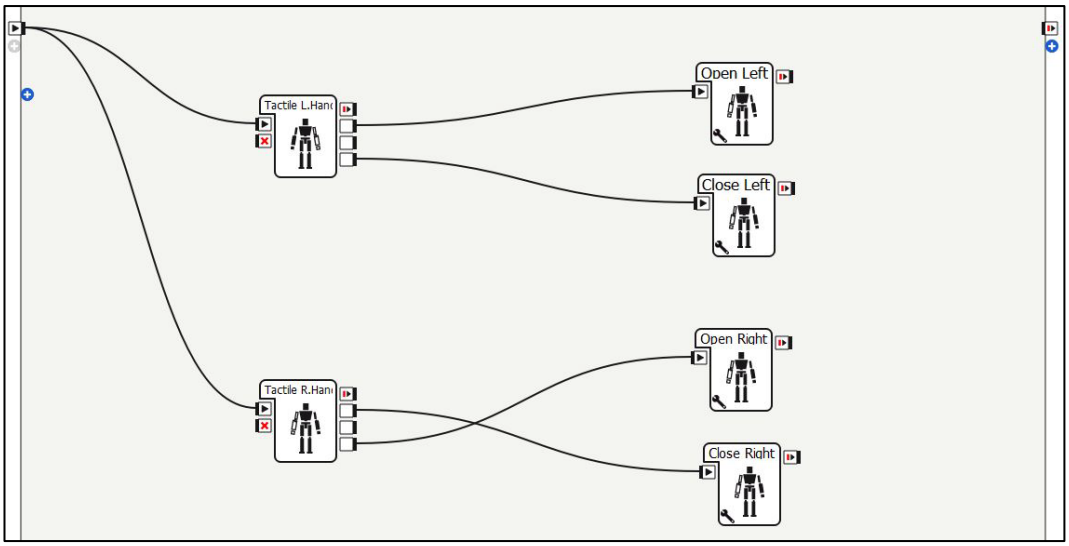


Figura 70: Ejemplo Choregraphe 2

En el ejemplo de la *figura 70* se controlan las manos del robot mediante los sensores táctiles ubicados en las muñecas. Se utilizan dos de los tres sensores en cada mano (*Tactile Hand*) para abrir y cerrar los dedos de dicha mano (*Open/Close Right/Left*). Como se puede observar, el diagrama no está conectado al punto de salida (esquina superior derecha) por lo que el comportamiento se ejecuta de forma indefinida hasta que es detenido manualmente por el usuario.

4.3.1 Desarrollo de las Rutinas de Ejercicios

En el contexto de una posible aplicación futura del robot en el marco de la atención a personas mayores, se han desarrollado tres rutinas de ejercicios de movilidad para el cuello, las manos y los hombros. El objetivo es que el usuario solicite al robot realizar los ejercicios para luego seguir sus movimientos a modo de guía.

Se separa el desarrollo en tres rutinas diferenciadas, una para cada articulación, cada una de las cuales contiene un numero de ejercicios entre dos y diez. Para cada rutina, se podrá especificar el número de repeticiones a realizar de cada ejercicio.

Las rutinas de ejercicios se basan en coreografías del robot, varios conjuntos de movimientos predefinidos que conforman la rutina realizando ejercicio tras ejercicio. El elemento principal es el bloque "*Timeline*" [81] de *Choregraphe* (*figura 71*) que permite crear coreografías de movimiento del robot mediante una secuencia de keyframes.

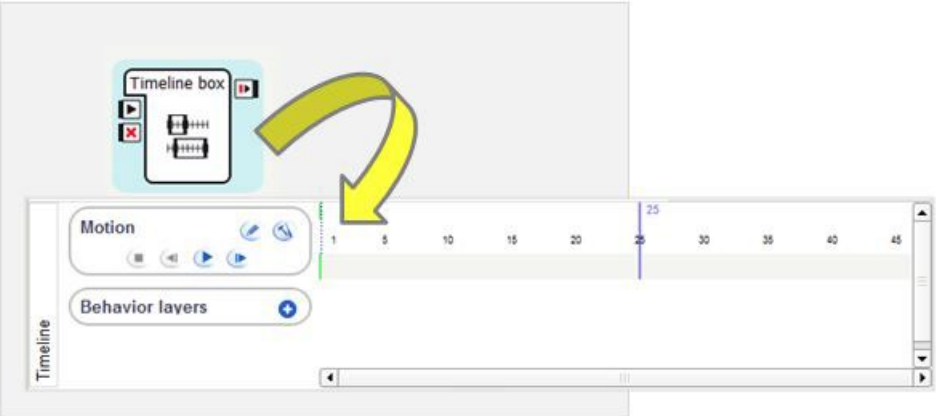


Figura 71: Bloque "Timeline"

Haciendo doble clic en el bloque "*Timeline*" se abre el panel de edición de la línea de tiempo [82] en el que se pueden grabar los diferentes keyframes que componen la secuencia de moviemientos de la coreografía. Un bloque "*Timeline*" se compone de:

- Una capa de movimiento (*Motion Layer*) que contiene los keyframes que determinan el movimiento del robot (*Figura 72*).
- Una o más capas de comportamiento (*bahavior layers*) que contienen diagramas de bloques que se ejecutan de forma sincronizada con el avance de la línea de tiempo (*figura 73)*.

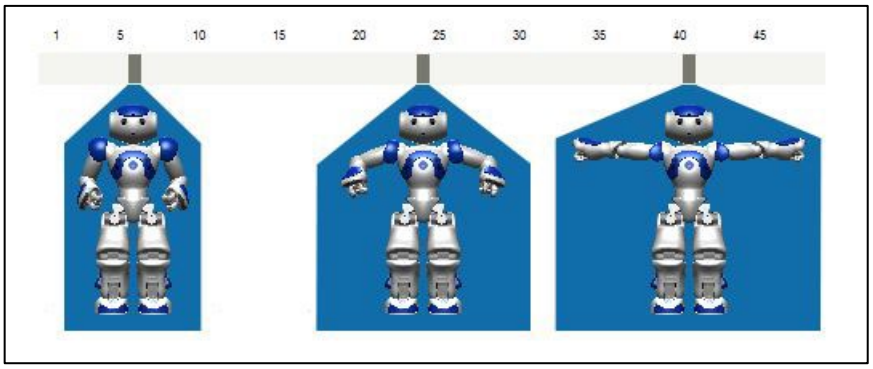


Figura 72: Keyframes en la capa de movimiento

Cuando el cursor temporal llega a un keyframe (fotograma de la línea de tiempo), de forma simultánea: los valores especificados para las articulaciones se aplican al robot (capa de movimiento) y se ejecuta el diagrama correspondiente si es que hay uno (capa de comportamiento)

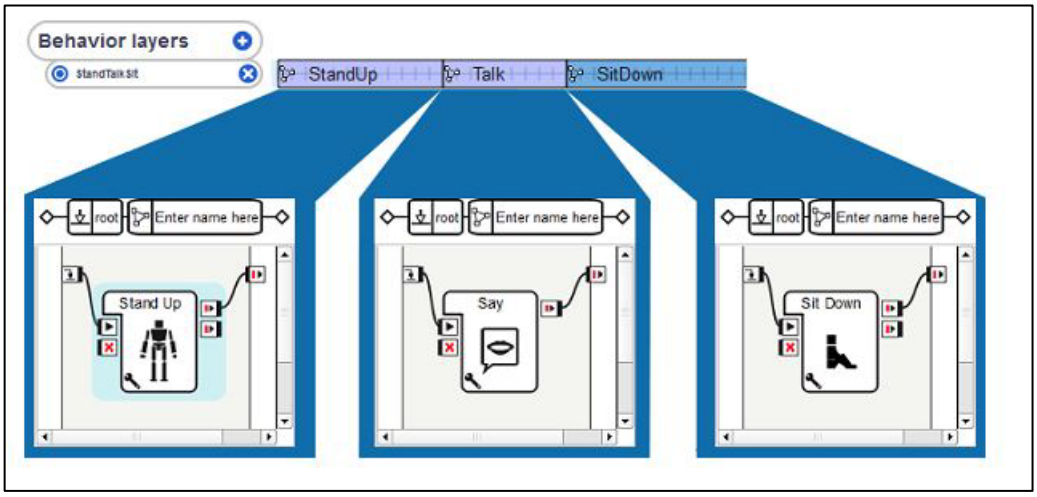


Figura 73: Diagramas como keyframes en la capa de comportamiento

Para grabar los diferentes keyframes en la capa de movimiento, se coloca el robot (real o virtual) en la posición deseada mediante los controles remotos integrados en *Choregraphe* y se guarda el valor de las articulaciones en el keyframe. No es necesario que un keyframe guarde valores de todas las articulaciones del robot, aunque es recomendable para especificar una posición única. También se recomienda comenzar cada "*Timeline*" en una posición conocida.

Al ejecutar el bloque "*Timeline*", el robot recorre la línea de tiempo adoptando las posiciones grabadas en cada keyframe e interpolando el valor de las articulaciones entre dos keyframes. El sub-panel "*Timeline Editor*" (*figura 74*) proporciona un control más preciso para editar la capa de movimiento, permitiendo incluso modificar el método de interpolación.

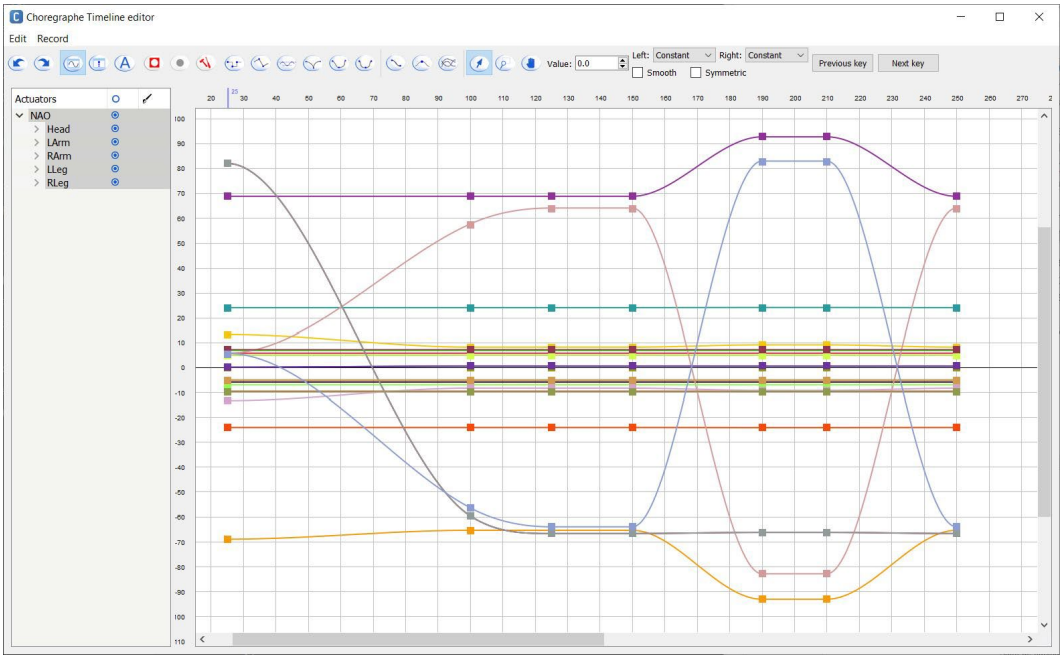


Figura 74: Panel "Timeline Editor" con curvas de interpolación

Se procede ahora a explicar los comportamientos desarrollados que componen las rutinas de ejercicios. Se utilizará como ejemplo la rutina de ejercicios de manos, siendo el proceder similar para las otras rutinas de ejercicios (cuello y hombros), simplemente incluyendo distintos ejercicios.

Se desea que el número de repeticiones realizadas de cada ejercicio sea configurable, para ello se debe introducir dicha información a modo de parámetro o variable en la memoria del robot. Esto se puede conseguir mediante el bloque "*Insert Data*" (*figura 75*) o utilizando en la programación Python la misma función que usa este bloque: "*ALMemory.insertData()*".

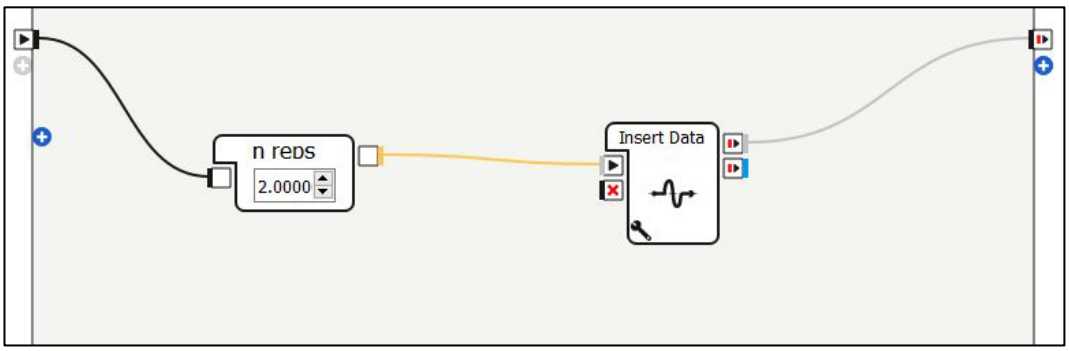


Figura 75: Diagrama Insert Data - número de repeticiones

En el diagrama de alto nivel, *root* (*figura 76*), se puede apreciar el planteamiento general del *behavior*:

- Se comienza adoptando una postura inicial (*Go to Posture*) para asegurarse de que el robot parte de una posición conocida.
- Acto seguido el robot dice una frase que indica si el lanzamiento de la rutina ha sido exitoso o no. Si la ejecución del bloque "*Go to Posture*" ha sido exitosa se anuncia que se va a proceder con la rutina de ejercicios y si ha fallado se comunica lo propio y se finaliza el comportamiento.
- Se procede a la ejecución de los ejercicios, cada uno implementado en un sub-diagrama contenido en un bloque de tipo "*Diagram*". En este caso, la rutina de manos consta de tres ejercicios.
- Finalmente, tras acabar los ejercicios, el robot pronuncia un mensaje de finalización y se termina la ejecución del comportamiento.

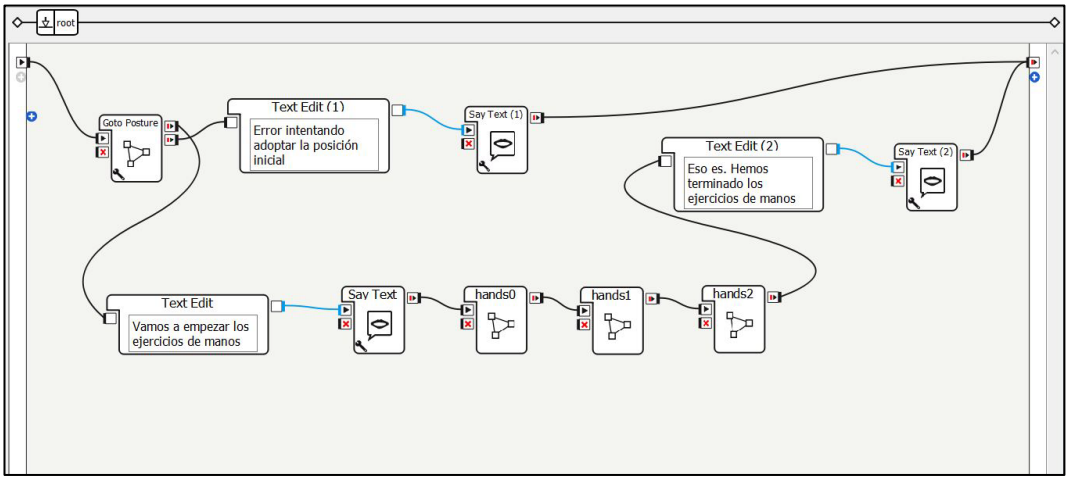


Figura 76: Diagrama general - ejercicios de manos

Cada uno de los bloques de tipo "*Diagram*" (*hands0-2*) implementa un *set* de ejercicios (*figura 77*) de manera similar, con los movimientos correspondientes a cada ejercicio. Debido a que el número de repeticiones es variable, se separa el ejercicio en tres bloques de tipo "*Timeline*":

1. *Start*: partiendo de la posición inicial del comportamiento, se adopta la postura de inicio del ejercicio, se realiza una repetición de demostración y el robot describe el movimiento a realizar en voz alta.
2. *Rep*: se ejecutan los movimientos correspondientes a una repetición del ejercicio. Este bloque "*Timeline*" está incorporado en un bucle que lo repite tantas veces como repeticiones se hayan especificado.
3. *End*: secuencia simple de movimientos que lleva al robot desde la posición de finalización de una repetición, hasta la posición inicial del comportamiento para poder ejecutar así el siguiente ejercicio.

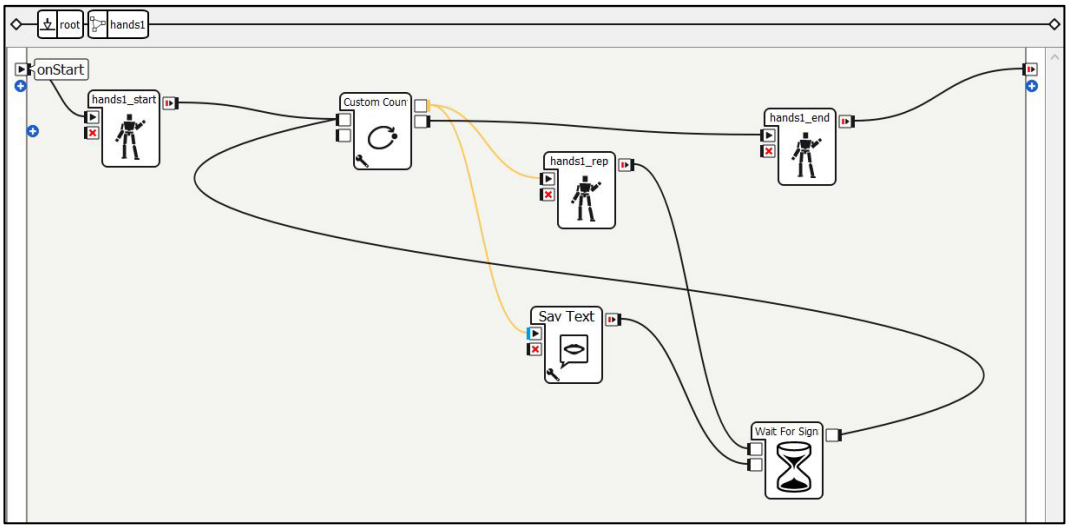


Figura 77: Sub-diagrama - Ejercicio 1 de manos

Tras realizar el "*Timeline*" de inicio del ejercicio, el diagrama activa un bloque de contador personalizado (el script de Python interno ha sido modificado para que el comportamiento se adhiera a las necesidades específicas de la aplicación). Este contador, lee de la memoria del robot el número de repeticiones a modo de parámetro y configura el bucle con ese número de iteraciones.

En cada iteración del bucle se ejecuta el "*Timeline*" de la repetición (*hands1_rep*) y el robot proclama en voz alta el número de la repetición para llevar la cuenta (bloque *Say Text*). Se espera a que ambos bloques hayan terminado (*Wait For Signals*) para incrementar el valor del contador y ejecutar así la siguiente iteración/repetición. Cuando el contador alcanza su límite, es decir el número de repeticiones a realizar, se ejecuta el "*Timeline*" de finalización del ejercicio (*hands1_end*) y se termina la ejecución de este sub-diagrama dando paso al siguiente ejercicio.

A continuación, se muestra el código correspondiente al bloque "*Insert Data*" y al bloque personalizado del contador "*Custom Counter*":

- *Insert Data (figura 78):* utiliza un proxy al módulo de memoria "*ALMemory*" para insertar un valor en una ubicación de la memoria del robot. Hace uso del método *insertData()* para colocar el valor (p) en una variable cuyo nombre viene especificado por el parámetro "key".
- *Custom Counter (figura 79):* accede al valor almacenado en memoria de forma análoga, utilizando el método *getData()*. Implementa un bucle que cuenta desde el valor inicial (1) hasta el valor final recabado de la memoria.

Insert Data

```python
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        pass

    def onLoad(self):
        self.memory = self.session().service("ALMemory")

    def onUnload(self):
        self.memory = None

    def onInput_onStart(self, p):
        self.memory.insertData(self.getParameter("key"), p)
        self.onStopped(p)

    def onInput_onStop(self):
        self.onUnload()
        pass
```

Figura 78: Script del bloque "Insert Data"

Custom Counter

```python
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self, False)

    def onLoad(self):
        self.memory = self.session().service("ALMemory")
        self.initializeParameters()
        if( self.get_memory() - self.getParameter("Initial value") >= 0 ):
            self.stepSign = +1
        else:
            self.stepSign = -1

    def onUnload(self):
        #~ puts code for box cleanup here
        pass

    def onInput_onNext(self):
        bParamChanged = ( self.nLast != self.get_memory() or self.nFirst != self.getParameter("Initial value") )
        bEnd = ( self.stepSign * self.nCounter > self.stepSign * self.nLast )
        if( bEnd or bParamChanged ):
            # si se alcanza el final del contador o cambian los parámetros iniciales
            # re-inicializar el contador - activar la entrada onInit()
            self.onInput_onInit()
        if( not bEnd or bParamChanged ):
            # funcionamiento normal del contador

            # almacenar el valor actual en una variable auxiliar
            currentCounter = self.nCounter
            # actualizar el contador - al siguiente valor
            self.nCounter = self.nCounter + self.stepSign * self.getParameter("Step value")
            # sacar el valor actual (variable auxiliar) por la salida currentValue
            self.currentValue( currentCounter )

    def initializeParameters(self):
        self.nFirst = self.getParameter("Initial value")    # valor inicial
        self.nCounter = self.nFirst                         # valor actual del contador = valor inicial
        self.nLast = self.get_memory()                      # valor final

    def onInput_onInit(self):
        self.initializeParameters() # inicializar parametros
        self.onReinitialized()      # activar salida de re-inicializar

    def get_memory(self):
        val = int(self.memory.getData(self.getParameter("key")))
        return val
```

Figura 79: Script del bloque "Custom Counter"

Para concluir el apartado, se muestra en la *figura 80* el contenido de uno de los bloques "*Timeline*", en concreto el correspondiente al comienzo del primer ejercicio de manos (*hands0_start*).

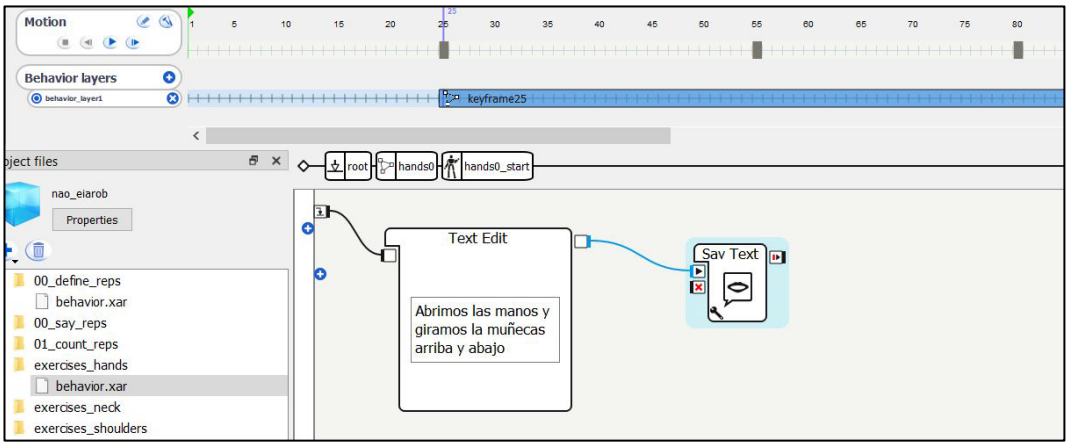


Figura 80: Interior del bloque Timeline "hands0_start"

En la parte superior de la figura, se pueden apreciar las dos capas correspondientes a este bloque "*Timeline*". La capa de movimiento (*Motion*) que contiene los keyframes de movimiento, y la capa de comportamiento (*behavior layer*) que contiene en este caso un único keyframe de comportamiento (*keyframe25*).

Un poco por debajo se ubica el indicador de la estructura jerárquica de los diagramas. Nos encontramos en el bloque *hands0_start* dentro del sub-diagrama *hands0*, a su vez contenido en el diagrama de nivel superior *root*.

En la zona inferior de la figura, se puede observar el diagrama correspondiente al keyframe de comportamiento, *keyframe25*. Este diagrama se utiliza, dentro del *Timeline* de comienzo de un ejercicio (*start*), para que el robot comunique en voz alta la descripción del ejercicio a realizar. Simultáneamente, en la capa de movimiento, el robot realiza los movimientos correspondientes a una repetición, para que el usuario visualice cómo se realiza el ejercicio.

4.4 Desarrollo del Chatbot GPT

Utilizando los servicios de la API de OpenAI se implementará un agente conversacional (*chatbot*) encargado de responder a las preguntas que se le hagan al robot. Primeramente, se realizará un desarrollo únicamente textual para posteriormente proceder a su implementación en el robot como se explicará en los siguientes epígrafes.

Antes de poder empezar con el desarrollo, se realizó una familiarización con los servicios que proporciona la API, al igual que se hizo con el resto de las herramientas utilizadas en el proyecto. Afortunadamente, la documentación [83] es clara, está bien estructurada y cuenta con una guía de inicio rápido [68] que dirige al usuario en el proceso de instalación de la librería y la configuración de la clave de uso (API key). Tras ingresar algunos fondos en la cuenta y validar la instalación con un script de prueba (*figura 81*), queda todo dispuesto para proceder con el desarrollo del *chatbot*.

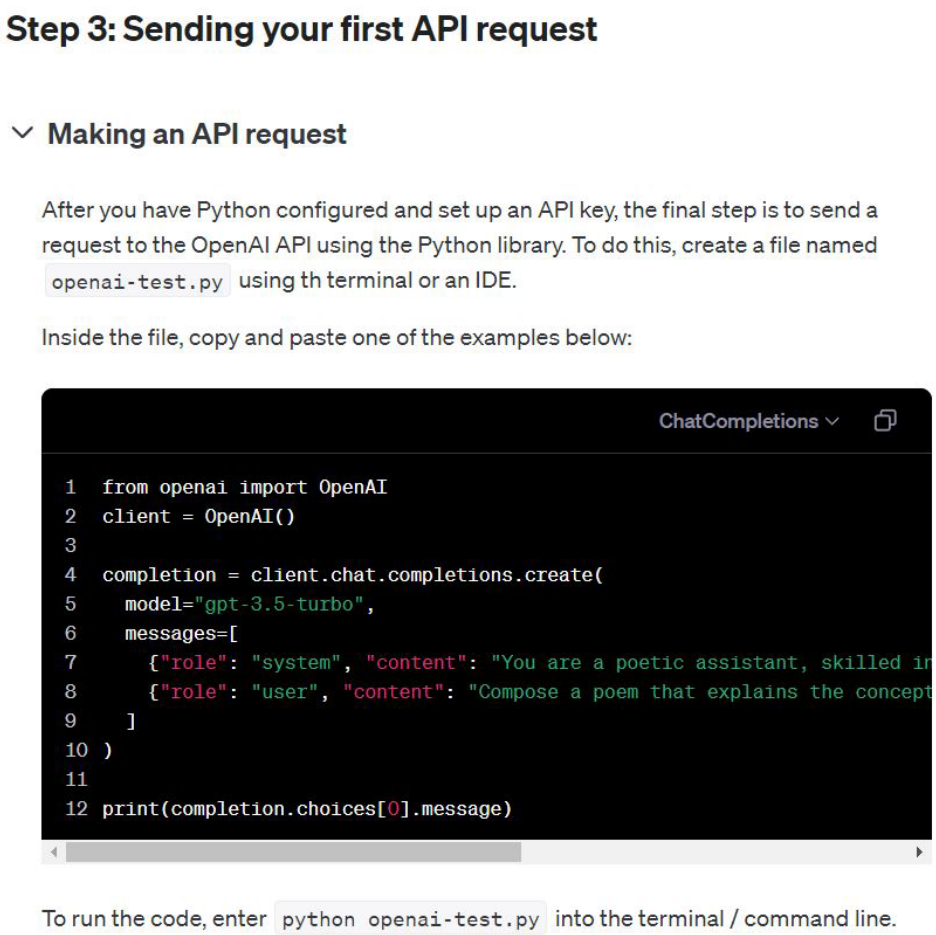


Figura 81: Script de validación de la API de OpenAI

Como se ha explicado con anterioridad en el apartado 3.2, la API de OpenaAI consta de dos *endpoints* para la generación de texto: *Chat Completions API* [84] y *Assistants API* [85]. Tras una valoración inicial se decidió por utilizar la primera opción, la API de *Chat Completions*, ya que es un *endpoint* más sencillo y flexible. Además, la API de asistentes se encuentra todavía en versión beta (en desarrollo).

La API *Chat Completions* tiene ese nombre porque su función es completar una conversación añadiendo el siguiente mensaje. En la llamada a la API (*figura 82*) se proporciona el historial de mensajes de la conversación y el modelo GPT subyacente responde devolviendo un mensaje de "asistente" que continúa la conversación.

```python
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who won the world series in 2020?"},
    {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."},
    {"role": "user", "content": "Where was it played?"}
  ]
)
```

Figura 82: Ejemplo de llamada a Chat Completions API

El parámetro "*messages*" de la llamada a la API, debe incluir una lista de mensajes que constituye el historial de la conversación. Este historial es lo que proporciona contexto al modelo GPT para generar una respuesta apropiada. Los mensajes tienen el formato de un diccionario de Python, y requieren obligatoriamente dos parámetros: el rol del autor del mensaje (*role*) y el contenido de dicho mensaje (*content*). Los roles principales que pueden adoptar las entidades en una conversación son:

- **"system":** *prompt* de sistema que define el propósito y el comportamiento general del modelo en la conversación. Se proporciona al inicio y establece el tono y la personalidad del asistente a lo largo de la conversación.

- **"user":** mensajes provenientes del usuario. Preguntas o instrucciones que se realizan buscando una respuesta del asistente.

- **"assistant":** respuestas del asistente generadas por las llamadas a la API. Una vez obtenida la respuesta, ésta se debe adjuntar al historial con el rol de "*assistant*" para mantener el contexto.

El *endpoint* de *Chat Completions* no tiene ninguna funcionalidad avanzada más allá de lo explicado ahora. A diferencia del *endpoint* de asistentes (*Assistants API*), carece de mecanismos internos para la gestión de conversaciones por lo que es una funcionalidad que el desarrollador debe implementar en la aplicación. Por lo tanto, el flujo de una conversación con un modelo GPT mediante la API de *Chat Completions* pasa por:

1. Proporcionar un *prompt* inicial (mensajes de sistema y usuario).
2. Ejecutar la llamada a la API y obtener la primera respuesta del asistente.
3. Adjuntar la respuesta del asistente al historial de la conversación.
4. Añadir un nuevo mensaje de usuario al historial de la conversación.
5. Volver al punto 2: realizar una nueva llamada para obtener una segunda respuesta del asistente, operando así en bucle.

Esta forma de proceder expone una de las limitaciones de los grandes modelos de lenguaje: la ventana de contexto. La capacidad de los LLMs de "razonar" está directamente relacionada con la cantidad de información de entrada (contexto) que pueden procesar antes de generar una respuesta. A medida que transcurre la conversación, el historial de la misma se va haciendo cada vez más extenso y por lo tanto más costoso de procesar (inferencia) pudiendo alcanzar dicho límite. El tamaño de la ventana de contexto (*context window*) de los modelos de OpenAI se puede consultar en la documentación [86] y supera los 100.000 tokens para los modelos más avanzados.

De igual manera, este uso exponencial de tokens con cada iteración de la conversación merece especial atención de cara a la facturación de los servicios de la API. OpenAI cobra en función de los tokens utilizados, con tarifas distintas [87] (*figura 83*) para cada modelo y distinguiendo entre tokens de entrada (historial de mensajes para proporcionar contexto) y tokens de salida (respuesta del asistente, generada por el modelo).

Model	Input	Output
gpt-3.5-turbo-0125	$0.50 / 1M tokens	$1.50 / 1M tokens
gpt-3.5-turbo-instruct	$1.50 / 1M tokens	$2.00 / 1M tokens
gpt-4o	$5.00 / 1M tokens	$15.00 / 1M tokens
gpt-4o-2024-05-13	$5.00 / 1M tokens	$15.00 / 1M tokens

Figura 83: Tarifas de la API de OpenAI [87]

La implementación se ha realizado apoyándose en la documentación oficial [84] así como en la guía de referencia de la API [88] y en los tutoriales proporcionados por el *Cookbok* [62]. En estos recursos se pueden encontrar ejemplos de implementación, así como el formato de las llamadas a la API y de los objetos que devuelven dichas llamadas.

El formato de las peticiones a la API (*request*) se muestra en la *figura 84*. Tras importar la librería, se crea un objeto de la clase "OpenAI" que actúa como cliente. Este cliente gestiona automáticamente la conexión y autenticación, haciendo transparente al usuario todos los detalles de la comunicación con los servicios de la API. Los únicos parámetros requeridos son: el modelo GPT a utilizar (*model*) y la lista de mensajes que constituye el historial de la conversación (*messages*).

example_request.py > ...

```python
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
  model="gpt-4o",
  messages=[
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Hello!"}
  ]
)

print(completion.choices[0].message)

```

Figura 84: Formato de la petición a la API

La respuesta (*API Response*), almacenada en el ejemplo en la variable "completion", es un objeto de tipo *chat.completion* con el formato específico que se muestra en la *figura 85*. A continuación se describen algunos de los campos más importantes de este objeto [88]:

- ***"model"***: modelo GPT utilizado para generar la respuesta

- ***"choices"***: lista de opciones de respuesta. Al realizar una petición se puede solicitar que el modelo genere más de una alternativa con la intención de que el usuario elija la respuesta que más se adecúe.

- ***"message":*** mensaje de respuesta generado. Contiene los campos "role" y "content" descritos anteriormente.

- ***"finish_reason":*** describe la razón por la que el modelo dejó de generar tokens. Puede ser "stop" si el modelo se detuvo de forma natural, "lenght" si se alcanzó el número máximo de tokens especificado en la petición (opcional) o "content_filter" si la respuesta generada activó alguno de los filtros de contenido.

- ***"usage":*** contiene los tokens usados en la petición separándolos en tokens de entrada ("*prompt_tokens*") y tokens de salida ("*completion_tokens*"). Este campo es especialmente útil para monitorizar el consumo de la API y aproximar el coste de uso.

Figura 85: Formato del objeto de respuesta - chat.completion

Como se puede apreciar en la *figura 84*, acceder a la respuesta del asistente es tan sencillo como utilizar la notación de punto para navegar por la estructura del objeto *chat.completions* hasta alcanzar el campo del mensaje generado: "*print(completion.choices[0].message)*".

Partiendo de este conocimiento básico del funcionamiento de la API, queda en manos del desarrollador el utilizar estos servicios para implementar la aplicación deseada.

En este documento, no se pretende realizar una explicación detallada, línea a línea, del código desarrollado. La versión final del código implementado se puede encontrar en los anexos del trabajo. Sin embargo, para ilustrar las funcionalidades desarrolladas, se va a comentar el código de los dos elementos principales de la programación del agente conversacional.

La clase "Conversation" (*figura 86*) contiene todas las funcionalidades relacionadas con el manejo del historial de la conversación. El atributo principal de la clase es una lista que contiene todos los mensajes de la conversación. El método (función propia de la clase) más importante es "*add_message()*" que permite adjuntar los nuevos mensajes generados al historial con su correspondiente rol y contenido.

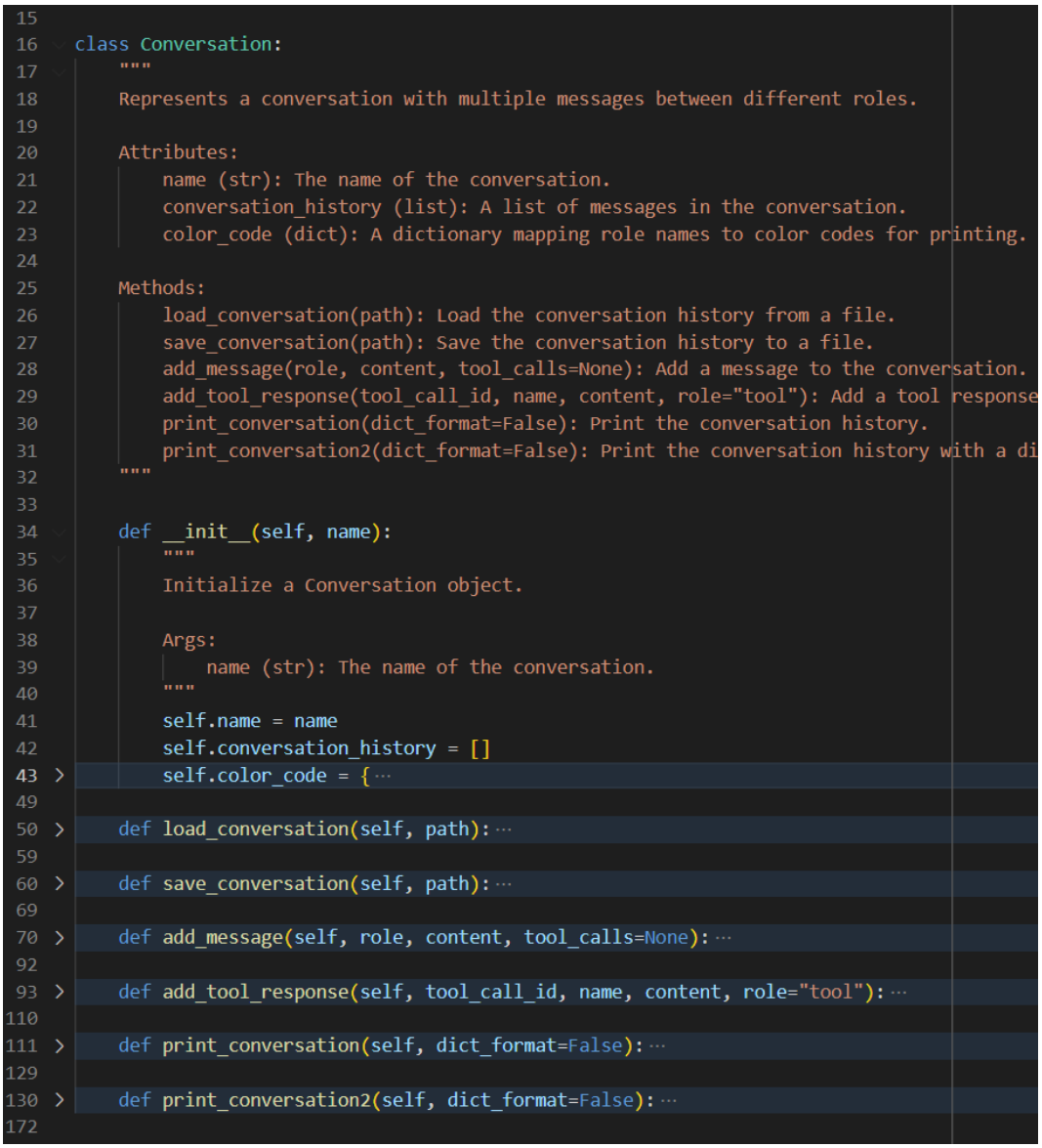


Figura 86: Código de la clase "Conversation"

A mayores, la clase dispone también de otros métodos para guardar conversaciones en archivos y cargar las conversaciones almacenadas, o para imprimir por pantalla el historial de la conversación en un formato claramente legible y coloreado (*figura 87*)

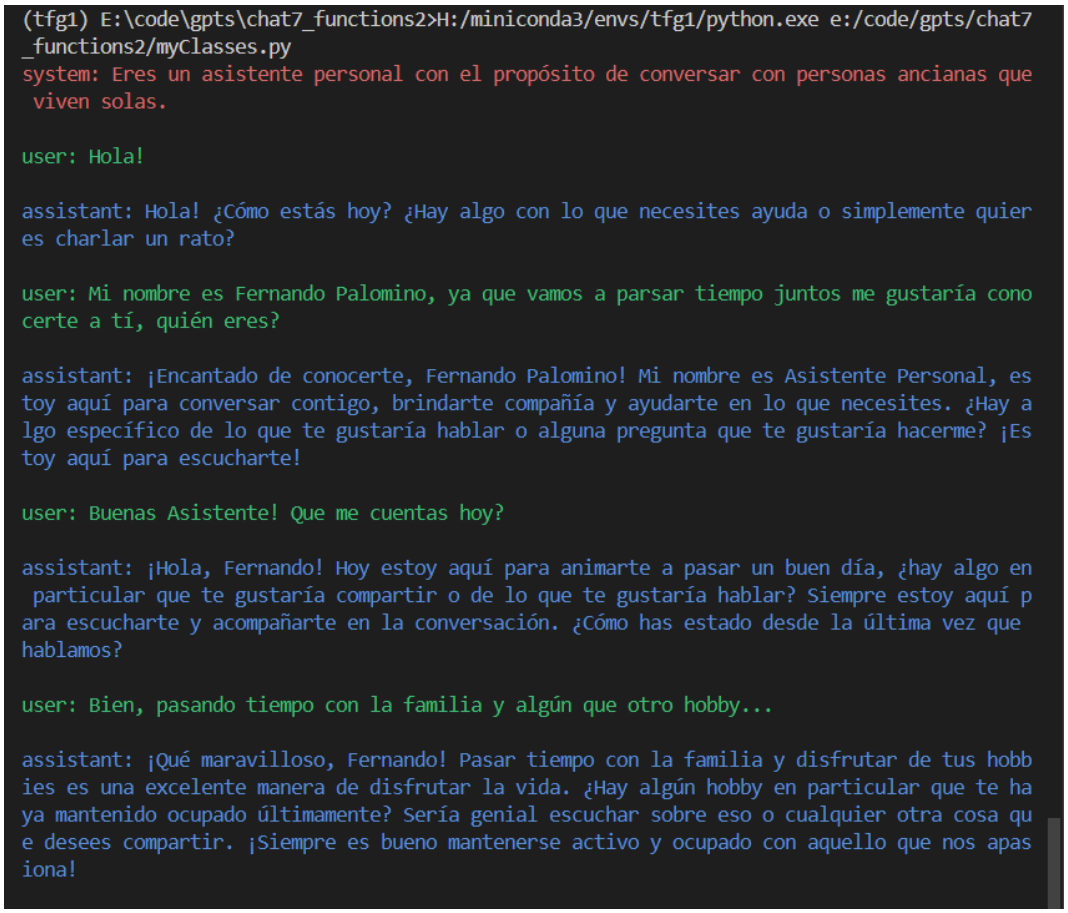


Figura 87: Ejemplo de conversación coloreada

El otro elemento fundamental es la función que realiza las peticiones a la API. Esta función se ubica dentro del bucle de la conversación y se ejecuta en cada iteración para obtener una respuesta del asistente al *prompt* proporcionado por el usuario. La función (*figura 88*), llamada "*chat_completion_request()*", hace uso de la clase "*Conversation*" para almacenar el historial de la conversación e ir añadiendo a éste los nuevos mensajes tanto del usuario como los mensajes generados por el asistente.

A continuación, se detallará punto por punto el funcionamiento de dicha función (*figura 88*). En el código mostrado en la siguiente figura, la función forma parte de una clase que aglutina las funcionalidades del agente conversacional por ello muchas de las variables utilizadas empiezan con "*self*", son atributos de la clase.

```python

    @retry(wait=wait_random_exponential(multiplier=1, max=20), stop=stop_after_attempt(3))
    def chat_completion_request(self, user_input):
        """ Generate chat completion:
            - Append the user input to the conversation history
            - Generate the chat completion request
            - Append the assistant message to the conversation history
            - Update the token count
            - Return the completion object

        Parameters:
        - user_input (str): The user input to append to the conversation history

        Returns:
        - completion (ChatCompletion): A ChatCompletion object representing the completion request
        """
        # Append the new user message to the conversation history
        self.conversation.add_message("user", user_input)
        # Generate the api completion request
        try:
            completion = self.openai_client.chat.completions.create(
                model=self.gpt_model,
                messages=self.conversation.conversation_history,
                # tools=None,
                # tool_choice=None,
            )

        except Exception as e:
            print("Unable to generate ChatCompletion response")
            print(f"Exception: {e}")
            raise e

        assistant_message = completion.choices[0].message
        self.conversation.add_message("assistant", assistant_message.content)
        self.update_token_count(completion)

        return completion

```

Figura 88: Función "chat_completion_request"

El código comienza con un decorador (*python decorator*) que, en caso de fallo, vuelve a intentar la ejecución. Los parámetros del decorador indican el tiempo de espera entre intentos y el número máximo de intentos fallidos permitidos hasta que se deja de intentar ejecutar la función decorada.

Seguidamente, se encuentra la cabecera de la función con los parámetros requeridos. El parámetro "*self*" hace referencia al propio objeto dentro del que se encuentra la función ya que es un método de una clase. El parámetro "*user_input*" hace referencia al último mensaje de entrada del usuario para el cual hay que generar una respuesta.

Tras la cabecera, siguiendo las reglas de estilo de Python, se encuentra el *docstring* de la función. Un *docstring* es una cadena de caracteres que describe el propósito de la función, así como sus parámetros y valor de retorno. A continuación, se encuentra el cuerpo de la función con algunos comentarios.

La lógica de la función realiza los siguientes pasos:

1. Se añade el mensaje de usuario *"user_input"* al historial de la conversación a través del método "*add_message()*" descrito anteriormente.
2. Se hace la petición a la API *Chat Completions* dentro de un bloque de manejo de excepciones. En caso de que se produzca una excepción, se muestra el mensaje por pantalla y se alza dicha excepción.
3. En caso de que la petición se ejecute sin problemas, se extrae el mensaje del objeto *chat.completion* que devuelve la API.
4. Se adjunta el mensaje del asistente al historial de la conversación para que permanezca actualizado de cara a la próxima iteración.
5. A través de una función auxiliar, *update_token_count()*, se actualiza la información del total de tokens utilizados en la conversación. Se utiliza esta información para controlar la cantidad de tokens consumidos y aproximar el coste de uso de los servicios de la API.
6. Finalmente, la función devuelve el objeto *chat.completion* obtenido para utilizar su información en otras secciones del código.

4.5 Integración en el robot

Cabe recordar, en este punto, la problemática descrita en el apartado 4.1.2 sobre la incompatibilidad de las versiones de Python utilizadas, que hace imposible agrupar todas las funciones que se requieren para el proyecto en un único proceso de software.

Para conseguir la integración de los (mínimo) dos procesos que se deben ejecutar (*Python2 + NAOqi y Python3 + OpenAI API*) se debe utilizar algún método de comunicación entre procesos (IPC). Se valoraron los mecanismos clásicos de IPC (*sockets*, semáforos, memoria compartida...) así como protocolos de comunicación más avanzados como RabbitMQ o ZeroMQ, disponibles en forma de librerías lo cual facilita su implementación.

El desarrollo final se ejecuta íntegramente en una única máquina (ordenador que controla el robot) por lo que se podría optar por un protocolo que utilice comunicación directa entre los clientes, como ZeroMQ, sin la necesidad de configurar un broker. Se decidió, sin embargo, utilizar el protocolo MQTT debido a que el proyecto EIAROB (en el cual se enmarca el presente trabajo) ya lo utiliza como sistema de comunicación entre sus dispositivos, lo que facilitaría una hipotética integración del robot NAO en su ecosistema. Sobra decir que el protocolo MQTT cumple con todos los requisitos necesarios y el modelo de mensajería *pub/sub* es idóneo para esta aplicación.

Finalmente, se decidió separar los servicios necesarios en tres procesos comunicados mediante mensajes MQTT. Estos tres procesos (*figura 89*) se corresponden con el procesamiento de la información hablada dentro del bucle de la conversación.

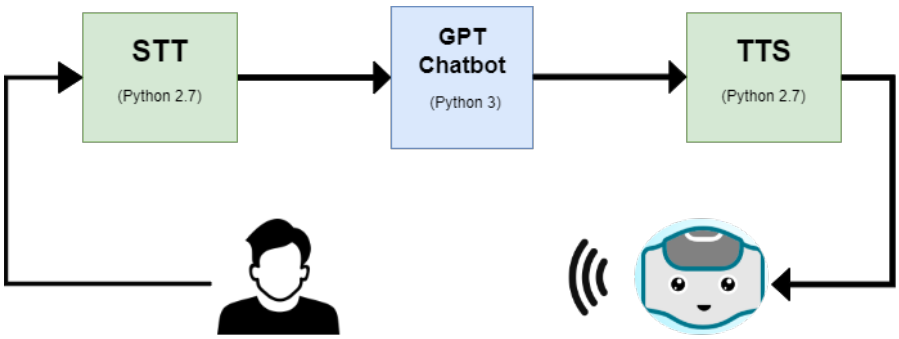


Figura 89: Diagrama lógico de los procesos

- **STT (*Speech To Text*):** análisis del audio para transformar la voz del usuario en texto que se enviará al agente conversacional.

- **Chatbot:** agente conversacional, implementado usando los servicios de la API, que responde al usuario actuando a modo de asistente.

- **TTS (*Text to Speech*):** síntesis de audio a partir de un texto escrito. Toma la respuesta, en formato texto, del *chatbot* y genera la voz del robot que responde al usuario.

La respuesta verbal del robot le llega al usuario quien, gracias al *feedback* proporcionado por los LEDs en los ojos del robot, gestiona los turnos de la conversación. Cuando el robot termina hablar vuelve al modo de escucha, listo para recibir una nueva interlocución del usuario.

El proceso de *chatbot* es idéntico a la implementación descrita en el apartado 4.4 sustituyendo la interfaz de texto a través de la línea de comandos por una comunicación basada en mensajes MQTT. El proceso recibe el texto proveniente del reconocimiento de voz (STT), genera la respuesta del asistente y la envía mediante otro mensaje al proceso de síntesis de voz (TTS).

El proceso de TTS (*Text To Speech*) se implementa fácilmente aprovechando las funcionalidades que ofrece NAOqi. Consta de un cliente MQTT que recibe los mensajes generados por el asistente y hace que el robot los enuncie en voz alta utilizando el módulo "*ALTextToSpeech*" o el módulo "*ALAnimatedSpeech*". Ambos módulos (*figura 90*) disponen de una función para hacer que el robot hable, con la diferencia de que "*ALAnimatedSpeech*", a mayores, ejecuta una animación en el robot que intenta corresponderse con el contenido de la frase a decir.

```
9
10  from naoqi import ALProxy
11  text_speech = ALProxy("ALTextToSpeech", "192.168.0.185", 9559)
12  anim_speech = ALProxy("ALAnimatedSpeech", "192.168.0.185", 9559)
13
14  # say phrase
15  msg = "Hola soy NAO, Estoy aquí para ayudarte."
16  text_speech.say(msg)
17  anim_speech.say(msg)
18
```

Figura 90: Ejemplo de los módulos TTS de NAOqi

4.5.1 Módulo de reconocimiento de voz (STT)

El sistema de diálogo nativo del robot (*QiChat*) [89] se basa en "reglas" predefinidas que asocian una frase concreta del usuario con una respuesta específica por parte del robot. Los sistemas de este tipo carecen de la flexibilidad y versatilidad necesaria para interpretar correctamente una conversación de un modo natural. Es por esto que se ha pasado a utilizar un sistema basado en modelos de lenguaje de inteligencia artificial.

Este sistema requiere de un proceso previo de reconocimiento de voz que proporcione el comando o pregunta del usuario a modo de texto. El *framework* de NAOqi no proporciona acceso a tal servicio a pesar de que debe contenerlo (o un servicio similar) para realizar el reconocimiento en su sistema nativo.

Para este propósito se ha utilizado un módulo de Python desarrollado por Johannes Braumer, *Vienna University of Technology*, el cual implementa reconocimiento de voz utilizando el servicio *Google Speech Recognition* en los robots Pepper y NAO de *Softbank Robotics* [90].

El código describe un módulo personalizado de NAOqi que realiza el reconocimiento de voz. Para ello, accede al buffer de audio del robot, utiliza la API de reconocimiento de voz de Google para obtener el resultado y genera un evento de NAOqi que contiene el texto reconocido (*figura 91*).

```
52   class SpeechRecognitionModule(naoqi.ALModule):
389      def recognize(self, data):
390          # print 'sending %d bytes' % len(data)
391
392          if (WRITE_WAV_FILE):
393              # write to file
394              filename = "out" + str(self.fileCounter)
395              self.fileCounter += 1
396              outfile = open(filename + ".raw", "wb")
397              data.tofile(outfile)
398              outfile.close()
399              rawToWav(filename)
400
401          buffer = np.getbuffer(data)
402
403          r = Recognizer()
404          try:
405              result = r.recognize_google(audio_data=buffer, samplerate=SAMPLE_RATE, 
406              self.memory.raiseEvent("SpeechRecognition", result)
407              print 'RESULT: ' + result
408          except UnknownValueError:
409              print 'ERR: Recognition error'
410          except RequestError, e:
411              print 'ERR: ' + str(e)
412          except socket.timeout:
413              print 'ERR: Socket timeout'
414          except:
415              print 'ERR: Unknown, probably timeout ' + str(sys.exc_info()[0])
```

Figura 91: Módulo SpeechRecognitionModule

El verdadero procedimiento de reconocimiento de voz lo realiza internamente este módulo. El proceso STT desarrollado lo único que hace es suscribirse al evento "*SpeechRecognition*" y procesar el mensaje contenido en él.

La implementación de la suscripción a eventos es mucho más sencilla utilizando el *qi Framework* (*figura 92*). Se conecta la sesión y se obtiene el servicio del módulo de memoria "*ALMemory*". Utilizando este servicio se realiza la suscripción al evento "*SpeechRecognition*" y se define una función de *callback* encargada de procesar el evento como corresponda.

```
18  class NaoListener:
34      def __init__(self, session):
35          self.session = session
36          self.mqtt_client = mqtt.Client(self.client_id)
37
38          self.session.connect("tcp://" + self.NAO_IP + ":" + str(self.NAO_PORT))
39          self.mem = self.session.service("ALMemory")
40          self.SpeechRecognition = self.session.service("SpeechRecognition")
41
42          # Subscribe to the SpeechRecognition event and set the callback
43          self.event = self.mem.subscriber("SpeechRecognition")
44          self.event.signal.connect(self.on_speech_recognition)
45
46          # Subscribe to the speechdone event and set the callback
47          self.ttsend = self.mem.subscriber("ALTextToSpeech/TextDone")
48          self.amsend = self.mem.subscriber("ALAnimatedSpeech/EndOfAnimatedSpeech")
49
50          self.ttsend.signal.connect(self.on_speech_done)
51          self.amsend.signal.connect(self.on_speech_done)
52
53
54          # Set the callbacks for the MQTT client
55          self.mqtt_client.on_connect = self.on_connect
56
```

Figura 92: Proceso STT – Suscripción al evento

También en esta sección del código se suscribe a los eventos que indican que el robot ha terminado de hablar, ya sea utilizando el módulo básico de TTS ("*ALTextToSpeech/TextDone*") o utilizando el módulo de diálogo animado ("*ALAnimatedSpeech/EndOfAnimatedSpeech*"). En adelante se referirá a estos eventos como "*SpeechDone*".

Estos eventos se utilizan en el control de los turnos de la conversación. Cuando se recibe un evento de reconocimiento de voz, el robot deja de escuchar para evitar que reconozca su propia voz. Una vez el robot ha terminado de hablar, se recibe el evento correspondiente y se vuelve a activar el reconocimiento para detectar la siguiente frase del usuario. Este control se lleva a cabo en las funciones de *callback* encargadas de procesar los eventos (*figura 93*).

```
70      def on_speech_recognition(self, value):
71          print("\nSpeech recognized:")
72          # print(type(value))
73          print(value.decode('utf-8'))
74
75          # Parse into a JSON string (object)
76          speech = json.dumps({"text": value.rstrip()})
77          # Publish the recognized speech to the speech topic
78          self.mqtt_client.publish(self.topic_speech, speech)
79          self.SpeechRecognition.disableAutoDetection()
80
81      def on_speech_done(self, value):
82          if isinstance(value, bool) and value == True:
83              self.SpeechRecognition.enableAutoDetection()
84
```

Figura 93: Proceso TTS - Funciones callback

Cuando se produce un evento de tipo "*SpeechRecognition*" se codifica el texto reconocido en formato JSON, se envía por MQTT al proceso de *chatbot* para generar una respuesta del agente conversacional y se desactiva el reconocimiento de voz para que el robot no confunda su propia voz con la del usuario. Ante un evento del tipo "*SpeechDone*", es decir cuando el robot haya terminado de hablar, se vuelve a activar el sistema de autodetección del módulo de reconocimiento de voz.

El módulo de reconocimiento de voz permite la calibración del proceso para poder detectar correctamente la voz del usuario. Posee una función automática de calibración en base al ruido de fondo además de otros ajustes como el idioma a reconocer o la duración máxima de una pausa en el diálogo que se interprete como el fin de la frase (*figura 94*).

```
56
57      def calibrate(self):
58          self.SpeechRecognition.start()
59          self.SpeechRecognition.setHoldTime(2.5)
60          self.SpeechRecognition.setIdleReleaseTime(1.0)
61          self.SpeechRecognition.setMaxRecordingDuration(10)
62          self.SpeechRecognition.setLookaheadDuration(0.5)
63          self.SpeechRecognition.setLanguage("es")
64          self.SpeechRecognition.calibrate()
65          self.SpeechRecognition.setAutoDetectionThreshold(5)
66
```

Figura 94: Proceso TTS - Calibración del módulo de reconocimiento de voz

4.5.2 Comunicación entre procesos

Como se ha comentado anteriormente, la comunicación entre procesos se realiza mediante el protocolo MQTT. Cada uno de los procesos desarrollados implementa un cliente MQTT interno que se comunica con el broker. Se ha utilizado la librería *Paho-MQTT* de Python para programar la comunicación, dicha librería es compatible con las dos versiones de Python utilizadas.

La arquitectura de la comunicación en el sistema está diseñada alrededor de los topics de forma que cada cliente únicamente está suscrito a los topics necesarios, aprovechando el modelo *pub/sub* para estructurar y simplificar la comunicación. A mayores, cada cliente tiene asociada una función de *callback* específica para los topics que debe manejar (*ver figura 95*) mediante la función *message_callback_add()* de la librería *Paho-MQTT*. Esto asegura que los mensajes de cada tipo se procesan de forma correspondiente al topic del que proceden.

```
# Set the callbacks for the MQTT client
self.mqtt_client.on_connect = self.on_connect
self.mqtt_client.on_message = self.on_message
self.mqtt_client.message_callback_add(self.topics['speech'], self.on_speech_message)
```

Figura 95: Asignación del callback específico para mensajes de tipo "speech"

A continuación, se enumeran los topics utilizados y el tipo de mensajes que transportan:

- **nao/log:** mensajes de registro provenientes de todos los clientes. Indican cuando el cliente se conecta, cuando se desconecta o portan mensajes de diagnóstico y error.
- **nao/input/speak:** mensajes que contienen las respuestas generadas por el asistente (proceso *chatbot*) para reproducir mediante el servicio TTS del robot NAO.
- **nao/output/speech:** mensajes del cliente STT como resultado del reconocimiento de voz. Se envían al proceso de *chatbot* como mensajes de usuario (“role”: “user”).

El resto de la comunicación externa o entre procesos consiste en las llamadas a la API de OpenAI para obtener las respuestas del modelo de lenguaje y la suscripción y reacción a los eventos de NAOqi, ya sean eventos nativos de tipo “*SpeechDone”* o desarrollados por terceros como el evento de reconocimiento de voz “*SpeechRecognition*”. La estructura final de los procesos y su comunicación se puede apreciar en la *figura 96*.

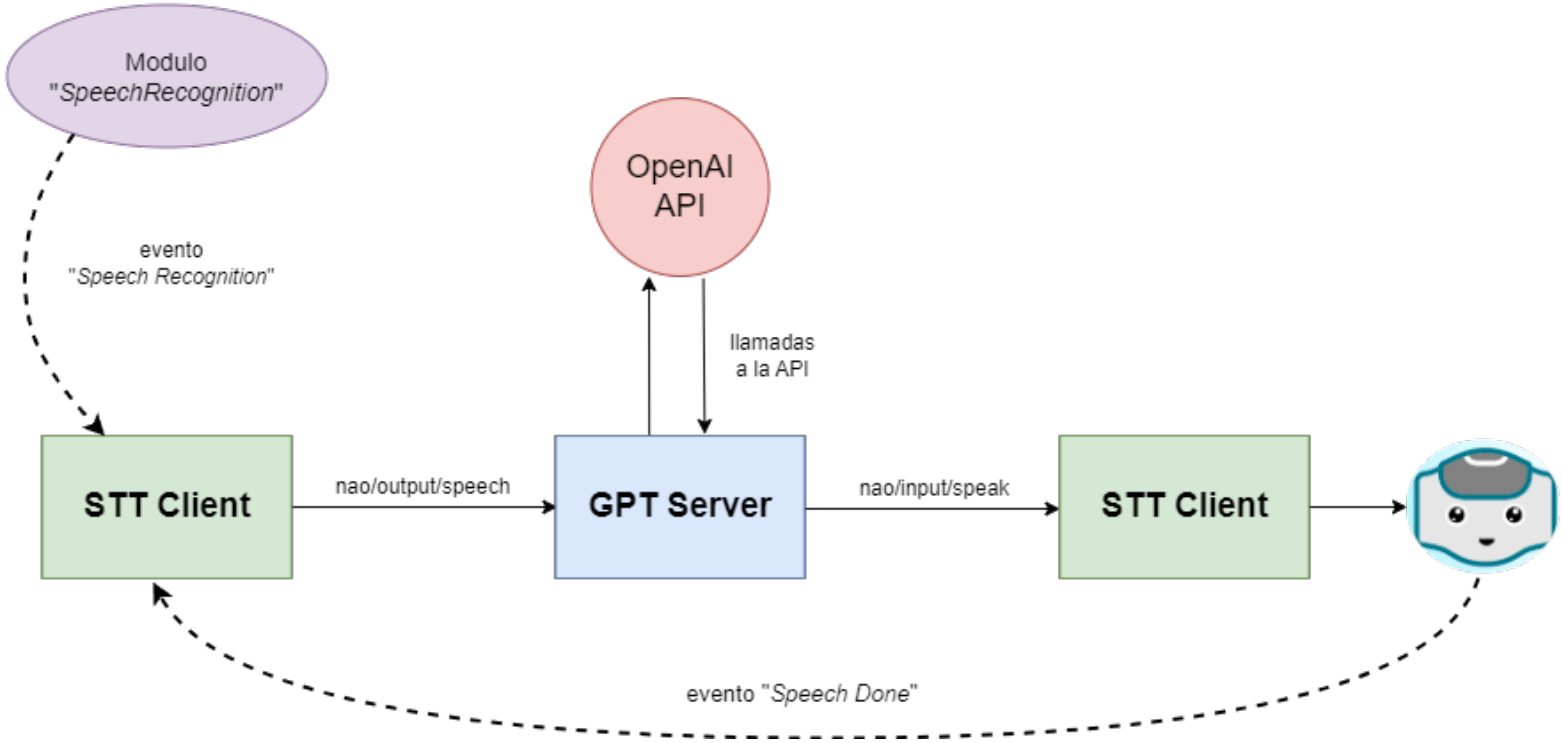


Figura 96: Diagrama de comunicación entre procesos

Capítulo 5: Resultados y Valoración

5.1 Valoración General

El robot NAO, implementando el nuevo sistema desarrollado, es capaz de mantener conversaciones con un nivel de inteligencia, fluidez y naturalidad muy superior al disponible de forma nativa.

La mejora de las capacidades gracias a los nuevos modelos de inteligencia artificial es especialmente notable en los procesos de reconocimiento de voz y en el asistente conversacional.

El reconocimiento de voz basado en el servicio *Google STT API* proporciona la flexibilidad necesaria para detectar cualquier frase, comando o interacción verbal con el robot incluyendo sutilezas como diminutivos o manierismos del lenguaje.

El sistema de diálogo (*chatbot*) basado en los servicios de la API OpenAI otorga al robot un nivel de razonamiento sin precedentes, siendo capaz de interpretar correctamente peticiones complejas y evaluar correctamente el contexto de la conversación. Además, tal y como está planteada la programación, el sistema tiene la capacidad de acceder a los modelos más recientes como GPT-4o, o a cualquier otro modelo futuro de OpenAI. Esto permite al sistema mantenerse al día, utilizando los modelos más avanzados, en un campo de tan rápida progresión como lo es la inteligencia artificial.

En cuanto a la naturalidad de la conversación, el desarrollo presenta las limitaciones típicas de otros *chatbots* en términos del control de turnos en la conversación y manejo de interrupciones por parte del usuario. La interacción con el robot NAO se basa en un control de turnos estricto, donde cada participante debe esperar a que el otro termine de hablar antes de continuar, lo que se aleja del ritmo fluido y espontáneo de una conversación natural. Esta restricción es común en muchos agentes conversacionales, y solo los asistentes más avanzados, como Google Assistant o Amazon Alexa, logran mitigar o solventar estos desafíos mediante técnicas avanzadas de procesamiento del lenguaje natural y gestión de contexto.

La parte más compleja (a interpretación personal) del proyecto y en la que se han obtenido peores resultados ha sido la integración de los sistemas desarrollados en el robot. Se ha intentado aprovechar al máximo las capacidades nativas del robot como *Autonomous Life* para mantener la naturalidad de los movimientos del robot, la reacción ante estímulos y la detección de personas; utilizando estas capacidades como base sobre la que incorporar los nuevos sistemas.

Sin embargo, se han producido escenarios donde los sistemas nativos del robot entraban en conflicto con los nuevos sistemas incorporados:

- Se debe desactivar el reconocimiento de voz nativo del robot para poder incorporar el nuevo sistema de STT, evitando así la duplicidad y que ambos sistemas interfieran entre sí.
- Se deben manejar manualmente en el código (activando/desactivando) ciertas habilidades autónomas (*Autonomous Abilities*) para evitar que las reacciones a estímulos externos interfieran con el funcionamiento del sistema de diálogo o con la ejecución de las rutinas de ejercicios.

Debido a las características del proyecto, es difícil presentar los resultados obtenidos en un documento, ya que el desarrollo en sí trata del comportamiento en tiempo real del robot. No obstante, se aprecia claramente una mejora significativa de las capacidades conversacionales y de interacción del robot respecto a las capacidades nativas que trae de fábrica.

Las capacidades interactivas del robot NAO se aprecian mejor a través de demostraciones visuales y ejemplos en vivo. Por ello, se incluye a continuación un enlace a una serie de vídeos que muestran el comportamiento del robot tanto en su función de agente conversacional como en la ejecución de rutinas de ejercicios guiados.

ENLACE - TFG_02581_videos_resultados

5.2 Estudio de Aceptación

Evaluar las capacidades sociales de un robot y predecir su aceptación por parte de los usuarios es un desafío complejo. La aceptación y adaptación de robots, especialmente en roles sociales, depende de diversos factores, incluyendo la funcionalidad del robot, sus capacidades de interacción y la percepción que se tiene del robot por parte de los usuarios. En este proyecto, no se ha podido realizar un estudio a gran escala debido a limitaciones de tiempo.

Existen diversos modelos diseñados para medir la aceptación de tecnologías y su integración en la vida cotidiana, cada modelo aborda la evaluación desde diferentes perspectivas. El modelo TAM (*Technology Acceptance Model*) y sus derivados, como el modelo UTAUT (*Unified Theory of Acceptance and Use of Technology*), se centran en la evaluación de cualquier tecnología desde un punto de vista funcional, considerando factores como la utilidad y la facilidad de uso. Otros modelos están diseñados para evaluar tecnologías específicas, como es el caso del modelo NARS (*Negative Attitude Towards Robots Scale*), que mide la actitud de las personas hacia los robots en general.

Sin embargo, el modelo más alineado con las características de este proyecto es el modelo ALMERE [91], desarrollado específicamente para evaluar la aceptación de robots sociales asistenciales por parte de personas mayores. Este modelo no solo considera la funcionalidad el robot, sino también las dimensiones sociales de la interacción, lo que lo convierte en una herramienta ideal para evaluar el desarrollo realizado sobre el robot NAO.

El modelo ALMERE busca identificar los factores clave que influyen en la aceptación de robots sociales asistenciales. Define varios constructos interrelacionados que reflejan tanto las características funcionales como las sociales del robot, los cuales se ha determinado que influyen en su aceptación. Los constructos propuestos por *Heerink et al.* [91] son los siguientes:

Código	*Constructo*	*Definición*
ANX	Ansiedad	evocar ansiedad o reacciones emocionales similares a la hora de utilizar el sistema
ATT	Actitud hacia la Tecnología	sentimientos positivos o negativos sobre el dispositivo o aplicación de la tecnología
FC	Condiciones de Facilidad	factores del entorno que facilitan el uso del sistema
ITU	Intención de uso	intención de utilizar el sistema durante un largo periodo de tiempo
PAD	Adaptabilidad Percibida	capacidad percibida del sistema para adaptarse a las necesidades del usuario

PENJ	Disfrute Percibido	sentimientos de alegría o placer asociados con el uso del sistema
PEOU	Facilidad de Uso Percibida	grado en que el usuario cree que usar el sistema es fácil o sin un gran esfuerzo requerido
PS	Sociabilidad Percibida	capacidad percibida del sistema de realizar un comportamiento sociable
PU	Utilidad Percibida	grado en que una persona cree que el sistema sería de utilidad
SI	Influencia Social	percepción del usuario de que las personas importantes para él creen que debe o no utilizar el sistema
SP	Presencia Social	experiencia de sentir una entidad social al interactuar con el sistema
Trust	Confianza	creencia de que el sistema funciona con integridad y fiabilidad

Tabla 1: Constructos del modelo ALMERE

Cada uno de estos constructos contribuye al concepto de "uso" que representa la aceptación del sistema y la predisposición del usuario a utilizar el robot durante un periodo extendido de tiempo. La influencia y correlación entre los diferentes constructos se estudia en el trabajo original [7] [91] y se puede utilizar para identificar áreas clave de mejora de las características del robot (*figura 97*).

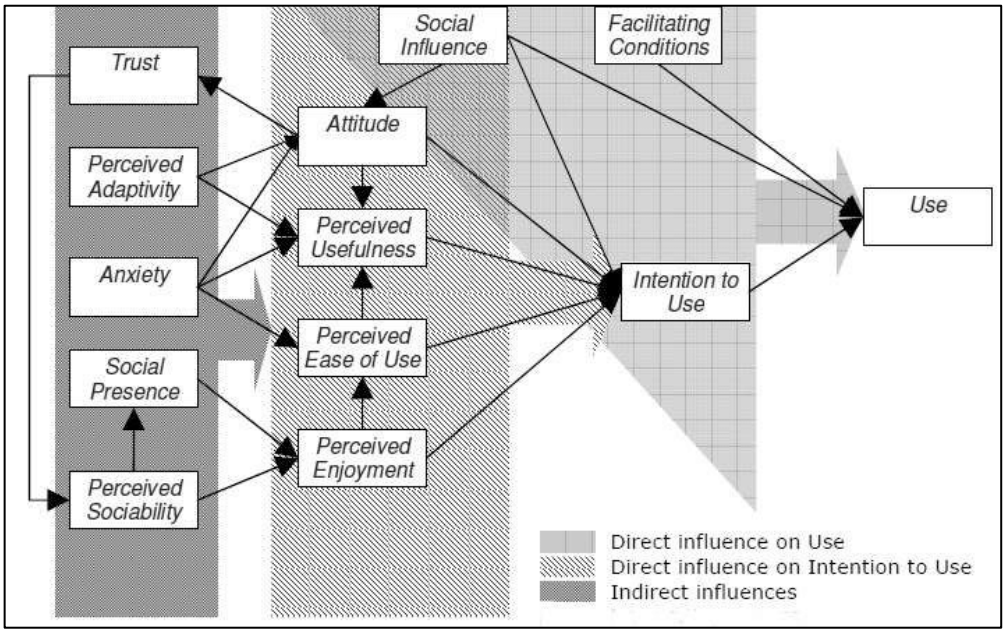


Figura 97: Relación entre constructos del modelo ALMERE [7]

Para evaluar la aceptación del desarrollo realizado en el robot NAO se llevó a cabo una encuesta entre los compañeros investigadores del laboratorio y personal de la universidad. Tras una breve explicación y demostración del uso del robot, se permitió a los usuarios probarlo libremente para luego cumplimentar un cuestionario, obteniendo un total de 9 muestras. Se es consciente de que, dado el tamaño y las condiciones del estudio, no existen garantías respecto a la fiabilidad o validez de los resultados. Con este estudio se intenta únicamente obtener una aproximación inicial de la aceptación, ilustrando las cualidades del modelo ALMERE.

También cabe apuntar que los individuos encuestados no representan adecuadamente al público objetivo de este proyecto, las personas mayores, y de que esta desviación en la población de estudio introduce un sesgo en los resultados, limitando la capacidad para generalizar las conclusiones obtenidas sobre la aceptación del robot para su verdadero propósito.

Se ha adaptado el cuestionario original propuesto [91], traduciéndolo al castellano y eliminando algunas preguntas (como SI1) destinadas al uso del robot en entornos como residencias. El cuestionario utiliza la escala *Likert* para valorar la conformidad del usuario con cada pregunta, estando cada una asociada a un constructo determinado. Los datos recopilados se procesan promediando el resultado de las preguntas de cada constructo, obteniendo la métrica de dicho constructo para cada muestra. Posteriormente se promedian los valores de cada muestra entre el número de usuarios encuestados obteniendo los siguientes resultados de aceptación para cada constructo (*figura 98*):

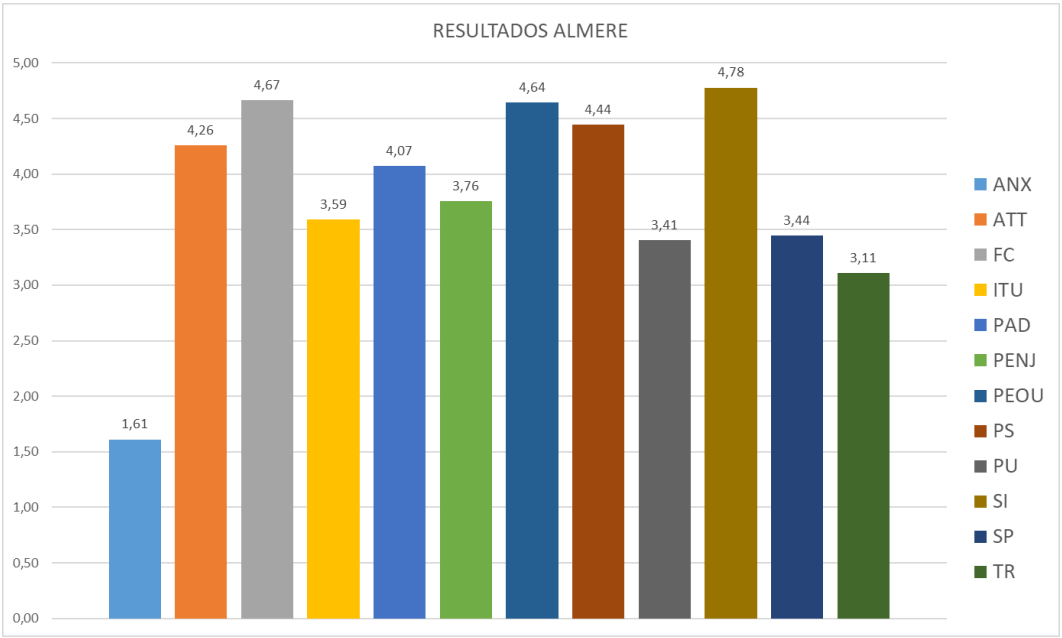


Figura 98: Resultados del modelo ALMERE

Siendo este un estudio superficial de la aceptación del robot, podemos identificar dos tendencias de los resultados obtenidos. En primer lugar, una clara afinidad hacia la personalidad y la presencia amigable del robot con valores altos de *Actitud, Sociabilidad* y un valor bajo de *Ansiedad*. También se aprecia una influencia positiva de los factores ambientales como son las *Condiciones de Facilidad,* la *Facilidad de uso* y la *Influencia Social*. Sin embargo, valoraciones mediocres de los constructos de *Utilidad* y *Confianza* indican cierta reticencia al uso del robot ya sea por falta de aplicaciones útiles o de credibilidad de las respuestas, lo cual se refleja en el constructo de *Intención de Uso*.

A pesar de esto último, la valoración general del robot es positiva, indicando su potencial utilidad en entornos de asistencia a personas mayores. Con el objetivo de mejorar el método de evaluación y obtener una visión más precisa de la aceptación del robot, se recomienda:

1. Refinar y depurar los métodos de interacción, implementando el uso de los controles táctiles del robot para activar y desactivar el sistema de manera intuitiva, en lugar de depender de un ordenador.

2. Realizar pruebas en un entorno que simule el despliegue final del robot, como una residencia de ancianos o un hogar particular. Esto permitirá analizar cómo interactúa el robot en situaciones reales y cotidianas.

3. Instruir a los usuarios de prueba en el uso del robot y la metodología de registro de resultados, incluyendo instrucciones sobre cómo completar los cuestionarios y evaluar las características del robot.

4. Llevar a cabo un estudio extendido en este entorno controlado para evaluar exhaustivamente las capacidades de interacción social del robot. Un periodo de prueba prolongado permitirá obtener datos más fiables sobre la aceptación del robot.

Finalmente, cabe nombrar dos áreas del estudio de la aceptación que no cubre el modelo ALMERE. En primer lugar, el modelo no considera factores moderadores en los usuarios tales como la edad, género, voluntariedad o experiencia previa con la tecnología. Todos estos factores pueden influir en la aceptación de un robot social. Por otra parte, los constructos del modelo no se pueden vincular directamente con el diseño o las funcionalidades del robot, impidiendo relacionar las reacciones (positivas o negativas) de los usuarios, con características concretas del del robot de cara a una posible corrección y mejora.

Capítulo 6: Estudio Económico

6.1 Introducción

En el presente capítulo se realizará un estudio del coste económico que ha supuesto la realización del proyecto desarrollado.

Hasta este momento se ha estudiado la viabilidad del proyecto desde un punto de vista técnico, es decir, si cumple o no con los objetivos marcados en el comienzo. Habiendo concluido dicho análisis, queda pendiente comprobar la viabilidad económica del proyecto.

Se comentará primeramente la planificación que se ha seguido a la hora de llevar a cabo el proyecto, intentando aproximar las horas dedicadas a cada tarea ya que este cálculo es necesario para el posterior estudio económico.

Posteriormente se realizará un estudio económico analizando por separado costes directos e indirectos y finalmente se concretarán los costes totales del proyecto.

6.2 Planificación del proyecto

A continuación, se describe en términos generales la planificación seguida en la elaboración del proyecto, se describen las diferentes fases del mismo y se proporciona una estimación de las horas dedicadas a cada una (ver diagrama de Gantt, *figura 99*).

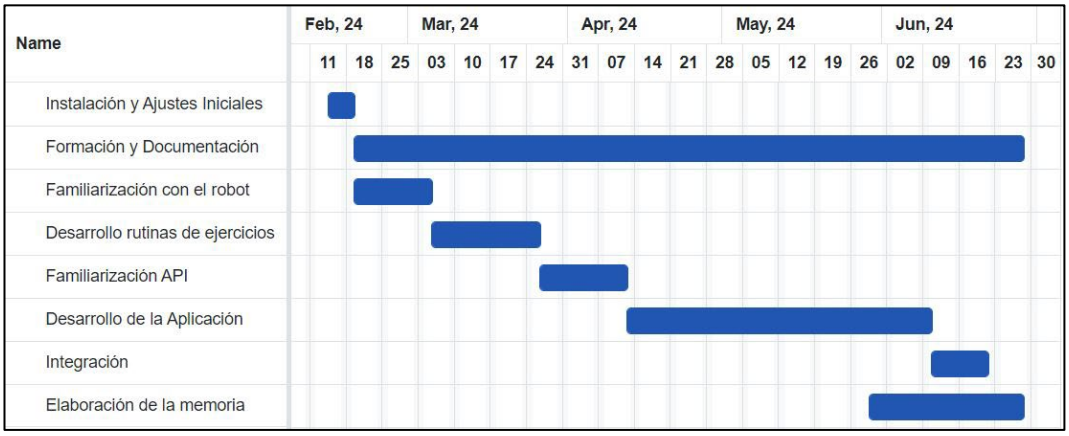


Figura 99: Planificación del Proyecto

Durante los primeros días se realizó la preparación del puesto de trabajo, desembalaje, configuración y ajustes iniciales del robot, así como la descarga e instalación de todos los software requeridos para el desarrollo del proyecto.

Una vez dispuestos todos los recursos necesarios, se prosiguió con la búsqueda y estudio de documentación referente tanto al robot NAO y su programación como al uso e implementación de los modelos GPT en el robot mediante el uso de la API de OpenAI. Este proceso de documentación se ha llevado a cabo a lo largo de todo el desarrollo del proyecto hasta la elaboración de la memoria.

Paralelamente, se llevó a cabo una familiarización con el robot NAO y sus herramientas de desarrollo, principalmente la aplicación *Choregraphe* para la programación de movimientos y coreografías animadas en el robot. Tras practicar realizando algunos proyectos de prueba se procedió al desarrollo de las rutinas de ejercicios guiados.

Tras esto, se comenzó con el acercamiento a los servicios ofrecidos por la API de OpenAI para la integración de un modelo de lenguaje en el robot. Después de revisar la documentación y ejemplos, se realizaron algunos programas de prueba que implementaban una funcionalidad básica de *chatbot*. Seguidamente, se desarrolla la aplicación completa incorporando a este *chatbot* inicial modelos de voz-a-texto (*STT*) y de texto a voz (*TTS*).

Finalmente, durante las últimas semanas, se procedió a la integración de las aplicaciones y programas desarrollados en el robot. Simultáneamente se procedió a la elaboración de la memoria.

En la siguiente tabla (*Tabla 2*) se muestra una distribución temporal aproximada de las horas dedicadas a cada una de las tareas descritas:

DISTRIBUCIÓN TEMPORAL DE TRABAJO	
Instalación y ajustes iniciales	15 horas
Formación y documentación	50 horas
Familiarización con el robot	30 horas
Desarrollo de rutinas de ejercicios	40 horas
Familiarización con la API de OpenAI	40 horas
Desarrollo de la aplicación de *chatbot*	130 horas
Integración final y puesta a punto	35 horas
Elaboración de la documentación	120 horas
Total de horas empleadas	**460 horas**

Tabla 2: Distribución temporal del trabajo

6.3 Recursos Empleados

A continuación, se muestra un resumen de los recursos hardware y software empleados en el desarrollo del proyecto. Es importante, a la hora de calcular los costes, tener en cuenta la amortización de los materiales únicamente durante el período de tiempo que han sido utilizados en el proyecto. Se tendrán en cuenta también los costes de servicios en la nube y suscripciones.

- Software:
  - Sistema Operativo: Windows 10
  - Suite de ofimática: Microsoft Office 365
  - Obsidian
  - Visual Studio Code
  - MQTT Explorer
  - Python versiones 2.7 y 3.10 y librerías
  - NAOqi Python SDK
- Hardware:
  - Robot NAO V6
  - Intel NUC 7i5BNK (*MQTT Broker*)
  - Ordenador fijo laboratorio
  - Ordenador fijo personal (personalizado)
  - Ordenador portátil Asus Zenbook 14 UX433F
- Servicios en Línea:
  - API de OpenAI
- Material Ofimático:
  - Libros de consulta
  - Otros consumibles

6.4 Costes Directos

Los costes directos son aquellos repercutidos directamente al desarrollo del proyecto. Se evaluarán:

- Coste del personal
- Costes amortizables de equipos y programas
- Coste de materiales, equipos no amortizables y servicios en línea

6.4.1 Coste del Personal

La realización del proyecto ha sido llevada a cabo por un estudiante de ingeniería, encargado de la programación del robot y la puesta a punto del sistema, bajo la supervisión de un ingeniero como tutor.

Se estimará el sueldo anual teórico del estudiante como si de un ingeniero junior en formación se tratara, teniendo en cuenta el sueldo medio anual en España para un empleo de estas características. El procedimiento habitual pasa por calcular el coste anual bruto del salario de un ingeniero, para luego analizar el coste de su trabajo en función del coste por hora y las horas empleadas en el proyecto.

El coste anual de un ingeniero incluye:

- Sueldo bruto anual, así como los posibles incentivos por su trabajo.
- Cotización a la Seguridad Social, que es un 35% del sueldo bruto.

Teniendo en cuenta esto, el coste anual del ingeniero será (*Tabla 3*):

COSTE ANUAL	
Sueldo bruto más incentivos	23.450,00 €
Seguridad Social (35% sueldo bruto)	8.207,50 €
Coste total	**31.657,50 €**

Tabla 3: Coste anual del personal

Se calcula a continuación una estimación de los días efectivos trabajados al año (*Tabla 4*):

DÍAS EFECTIVOS POR AÑO	
Año medio	365,25 días
Sábados y Domingos	-104,36 días
Días de vacaciones efectivos	-20,00 días
Días festivos reconocidos	-15,00 días
Días perdidos estimados	-5,00 días
Total días efectivos estimados	**220,89 días**

Tabla 4: Días efectivos por año

Conociendo el número total de días efectivos de trabajo, y que la jornada laboral es de 8 horas, obtenemos el total de **horas efectivas de trabajo:**

$$220{,}89\,\text{días/año} \times 8\,\text{horas/día} = \mathbf{1.767{,}12\ horas\ /\ año}$$

El **coste por hora** de un ingeniero se calcula como la división del sueldo anual entre las horas efectivas trabajadas al año:

$$\frac{\mathbf{Coste}}{\mathbf{hora}} = \frac{31.657{,}50\,€/año}{1.767{,}12\ \text{horas/año}} \approx \mathbf{17{,}92\ €/hora}$$

Partiendo de la duración estimada del proyecto de **460 horas,** calculamos el **coste de personal directo** como la multiplicación de las horas dedicadas y el coste efectivo de una hora de trabajo del ingeniero.

$$460\ \text{horas} \times 17{,}92\,€/\text{hora} = \mathbf{8.243{,}20€}$$

COSTE PERSONAL DIRECTO	8.243,20 €

6.4.2 Coste de amortización de equipos y programas

Para el cálculo de estos costes se debe analizar la inversión total y calcular la amortización lineal correspondiente según los criterios que aconseja la ley. En este apartado se estudiará los costes de amortización de los equipos de trabajo utilizados.

La mayoría de los servicios de software no gratuitos han adoptado un modelo de negocio basado en suscripciones, por lo tanto, no se consideran amortizables ya que hay que pagar el total de la suscripción anual independientemente de cuántas horas se utilice el programa.

Se estima como tiempo de amortización un período de 5 años para los equipos de hardware debido a la rapidez del avance tecnológico. De esta forma, para calcular el coste hay que multiplicar por un factor de 0.2 el importe del equipo. (*Tabla 5*).

MATERIAL	IMPORTE (aprox.)	AMORTIZACIÓN 20 %
Intel NUC 7i5BNK	650,00 €	130,00 €
Ordenador Laboratorio	350,00 €	70,00 €
Ordenador Personal *custom*	1.400,00 €	280,00 €
Ordenador Asus Zenbook 14	1.150,00 €	230,00 €
Total material	**3.550,00 €**	**710,00 €**

Tabla 5: Amortización de equipos

Hay que apuntar que no se han incluido programas informáticos ya que la mayoría de los utilizados son de libre distribución y por lo tanto gratuitos, y el resto se adquieren mediante suscripción lo cual no es amortizable.

El coste final por hora de utilización del material es calculado mediante la división de la amortización anual entre el número de horas de uso en dichos equipos.

$$\frac{\textbf{Coste}}{\textbf{hora}} = \frac{710{,}00\,€/año}{1.767{,}12\ \text{horas/año}} \approx \mathbf{0{,}40}\ €/\mathbf{hora}$$

Se considera el tiempo de uso de estos equipos igual al tiempo total necesario para la realización del proyecto por ser necesario en las etapas de análisis programación y documentación. Por tanto, el coste de amortización del material será:

$$460\ \text{horas} \times 0{,}40\,€/\text{hora} = \mathbf{184{,}00}€$$

COSTE DE AMORTIZACIÓN DE EQUIPOS	184,00 €

6.4.3 Coste de materiales y servicios

En este apartado se tienen en cuenta los costes de los distintos materiales, equipos y servicios utilizados para el desarrollo del proyecto (*Tabla 6*):

- Componentes o equipos hardware no amortizables. Estos bienes se utilizan en exclusiva para el proyecto por lo que se debe incluir todo el importe como coste directo.
- Suscripciones a programas de software. La suscripción consiste en un pago fijo anual independientemente de las horas que se utilice el producto.
- Facturas de servicios online que cobran en función de su uso.
- Materiales consumibles de ofimática como puede ser cuadernos, bolígrafos, impresión de documentos, etc.

ARTÍCULO	COSTE
Robot NAO V6	12.990,00 €
Suscripción Microsoft 365 Personal	70,00 €
Servicio de la API de OpenAI	15,00 €
Material de ofimática	50,00 €
Total	**13.125,00 €**

Tabla 6: Coste de materiales y servicios

COSTE DE MATERIALES Y SERVICIOS	13.125,00 €

6.4.4 Costes directos totales

Agregando todos los costes que se han ido obteniendo, los costes directos totales son la suma del coste de personal, la amortización de equipos y el coste de materiales y servicios utilizados. Por tanto, será:

$$8.243{,}20€ + 184{,}00€ + 13.125{,}00€ = \mathbf{21.552{,}20}€$$

COSTES DIRECTOS	21.552,20 €

6.5 Costes Indirectos

Los costes indirectos son los gastos producidos por la actividad requerida para la elaboración del proyecto y que no se pueden incluir en ninguno de los gastos directos. No son directamente imputables a la producción. Comprenden gastos como el consumo eléctrico o los desplazamientos (*Tabla 7*).

COSTES INDIRECTOS	
Dirección y servicios administrativos	130,00 €
Consumo de electricidad	210,00 €
Consumo de telefonía	25,00 €
Consumo de desplazamiento	250,00 €
Total gastos indirectos	**615,00 €**

Tabla 7: Costes indirectos

Por lo tanto, los costes indirectos totales ascienden a:

COSTES INDIRECTOS	615,00 €

6.6 Costes Totales

Los costes totales son la suma de los costes directos e indirectos, siendo el total para este proyecto el mostrado en la siguiente tabla (*Tabla 8*):

COSTES TOTALES	
Costes directos	21.552,20 €
Costes indirectos	615,00 €
Coste total del proyecto	**22.167,20 €**

Tabla 8: Costes totales

En conclusión, el coste total del proyecto asciende a la cantidad de:

COSTE TOTAL DEL PROYECTO	22.167,20 €

Capítulo 7: Conclusiones y Líneas Futuras

7.1 Conclusiones generales

En el presente proyecto, se ha trabajado con el robot NAO, analizando sus capacidades y su potencial como robot social. Se ha alcanzado el objetivo principal de desarrollar un nuevo sistema de interacción conversacional basado en las funcionalidades que ofrece *ChatGPT*.

Partiendo de conocimientos básicos de Python, la implementación de diversas funcionalidades para el robot ha permitido profundizar y ampliar esos conocimientos además de adquirir la correspondiente experiencia al desarrollar una aplicación desde cero.

Se ha obtenido una comprensión más profunda de los grandes modelos de lenguaje (*LLMs*), explorando tanto su funcionamiento interno como sus capacidades. Este aprendizaje ha sido clave para anticipar y evaluar cómo los avances en *LLMs* pueden mejorar la interacción con sistemas autónomos.

También se ha adquirido experiencia práctica en la integración de modelos GPT desde una perspectiva de desarrollador, más allá del uso convencional como usuario. Esta faceta incluyó la programación de un cliente o *chatbot* que interactúa con los servicios de la API de OpenAI, lo cual ha proporcionado una visión clara de las posibilidades y desafíos de implementar servicios de IA en aplicaciones reales.

El proyecto ha permitido adquirir un conocimiento genérico sobre el robot NAO y su manejo además de la familiarización con varias de las herramientas disponibles para su programación; más concretamente *Choregraphe* y el SDK de Python 2. Esto queda reflejado en las coreografías desarrolladas para la realización de ejercicios guiados.

Partiendo sin conocimientos previos ni del funcionamiento del robot NAO ni de la API de OpenAI, el proyecto ha supuesto un esfuerzo significativo de documentación y autoformación en ambas áreas, permitiendo adquirir una comprensión sólida de las tecnologías involucradas. Además, la investigación y solución de los problemas que han ido surgiendo a lo largo del proyecto proporciona una valiosa experiencia, ampliando las habilidades para desarrollar aplicaciones similares. En resumen, los conocimientos adquiridos proporcionan una base robusta para futuros desarrollos en estas áreas emergentes.

7.2 Conclusiones sobre los objetivos planteados

El objetivo principal de desarrollar un sistema de diálogo basado en modelos GPT ha sido alcanzado con éxito. La implementación de este sistema ha demostrado mejorar significativamente las capacidades conversacionales y de interacción del robot NAO. En concreto, cabe destacar:

- Se ha conseguido mejorar las capacidades del robot NAO mediante la integración de un sistema de reconocimiento de voz significativamente superior al estándar de fábrica y un sistema de diálogo con un nivel elevado de razonamiento, impulsado por los últimos modelos GPT

- La integración de modelos de lenguaje y el desarrollo del chat de voz representan solo un primer paso que demuestra las posibilidades de ampliación del sistema y el potencial de estas futuras mejoras.

- La evaluación de los resultados obtenidos (Capítulo 5) confirma, al menos, una mejora significativa en las capacidades interactivas del robot en comparación con sus habilidades iniciales. Aunque estos resultados no reemplazan una encuesta exhaustiva sobre el uso del robot, sugieren una mejora en la aceptación, lo que indica su potencial utilidad en entornos de asistencia a personas mayores.

En cuanto a los puntos débiles del proyecto, debemos hablar del robot NAO como plataforma de desarrollo. Analizando las capacidades del robot NAO en el contexto de los recientes avances tecnológicos, el robot está bastante desactualizado por no decir obsoleto. Únicamente su factor de forma, compacto minimalista humanoide y con muchos grados de libertad se mantiene relevante a día de hoy. En cuanto al resto de hardware y especialmente el software que lo controla podemos nombrar las siguientes carencias:

- Las cámaras del robot NAO carecen de la resolución y velocidad de fotogramas necesarias para aplicaciones como la navegación, detección de objetos, evasión de obstáculos o reconocimiento de caídas mediante visión artificial. Además, el hardware de procesamiento interno no es lo suficientemente potente para manejar tal cantidad de datos en tiempo real.

- El sistema de reconocimiento de voz del robot NAO no está a la altura de los modelos actuales y su programación se basa en el reconocimiento de frases concretas, lo que limita la flexibilidad en el diálogo.

- El sintetizador de voz del robot NAO tampoco alcanza el nivel de los nuevos modelos basados en inteligencia artificial, careciendo de la naturalidad y versatilidad que presentan estos modelos. Además, en las pruebas realizadas, el robot ha tenido dificultades para enunciar algunas características del español, especialmente los acentos y la entonación de las preguntas.

- Las capacidades de navegación del robot NAO están limitadas y anticuadas. Aunque se pueden utilizar las cámaras para la navegación basada en visión artificial, se necesitaría un alto *framerate* y una capacidad de procesamiento adecuada para analizarlo en tiempo real. Los sensores ultrasónicos y *bumpers* en los pies del robot resultan casi ineficaces en comparación con los sensores LiDAR utilizados por los robots móviles modernos.

A mayores, se han conseguido también los objetivos secundarios propuestos en cuanto al análisis del robot y sus capacidades físicas, estudio del campo de la robótica social, investigación de las tecnologías implicadas y completa integración de las funciones desarrolladas en el robot.

7.3 Líneas Futuras

Vivimos tiempos emocionantes gracias a este nuevo auge de la IA, la cual ya se está utilizando para impulsar el campo de la robótica, y en particular, los robots sociales e interactivos. El robot NAO es una plataforma robótica más que capaz para albergar estas nuevas funcionalidades impulsadas por IA. Se debe tener en cuenta, sin embargo, que sus capacidades de sensorización, procesamiento y navegación son limitadas.

7.3.1 Experimentación con diferentes modelos GPT

Una posible línea futura para mejorar la interacción del robot NAO sería la integración de los modelos GPT más recientes. La incorporación de modelos más avanzados, como GPT-4 o incluso futuros modelos como GPT-5, permitiría aprovechar sus superiores capacidades de razonamiento y generación de texto.

En línea con el uso de diferentes modelos GPT, se podría remodelar el código para que hiciera uso de la nueva API de asistentes (*assistants API*) la cual integra nuevas funcionalidades como la gestión automática del registro de la conversación en la nube, o el almacenamiento de información personalizada para cada asistente.

La llegada de nuevos modelos con capacidades mejoradas puede incluso revolucionar el planteamiento general de la aplicación. Por ejemplo, modelo GPT-4o ya anunciado introduce la capacidad de recibir audio como entrada de forma nativa y también como salida, una característica que, una vez disponible en la API, podría ser explotada para mejorar la interacción vocal del robot con los usuarios.

7.3.2 Exploración de modelos alternativos texto-a-texto

Para incrementar la flexibilidad y adaptabilidad del sistema de diálogo del robot NAO, otra dirección futura podría ser la generalización de su programación para soportar diversos modelos de lenguaje más allá de los ofrecidos por OpenAI. Esta estrategia permitiría utilizar cualquier modelo de texto a texto, liberando el desarrollo de la dependencia exclusiva de la API de OpenAI y de los modelos GPT.

Este enfoque permitiría la evaluación de una variedad de modelos, incluyendo opciones *open-source* y gratuitas. Otras empresas de IA ofrecen modelos de diferentes tamaños (*figura 100*) en cuanto al número de parámetros, los más pequeños están concebidos para ejecutarse localmente en *smartphones* y otros dispositivos móviles.

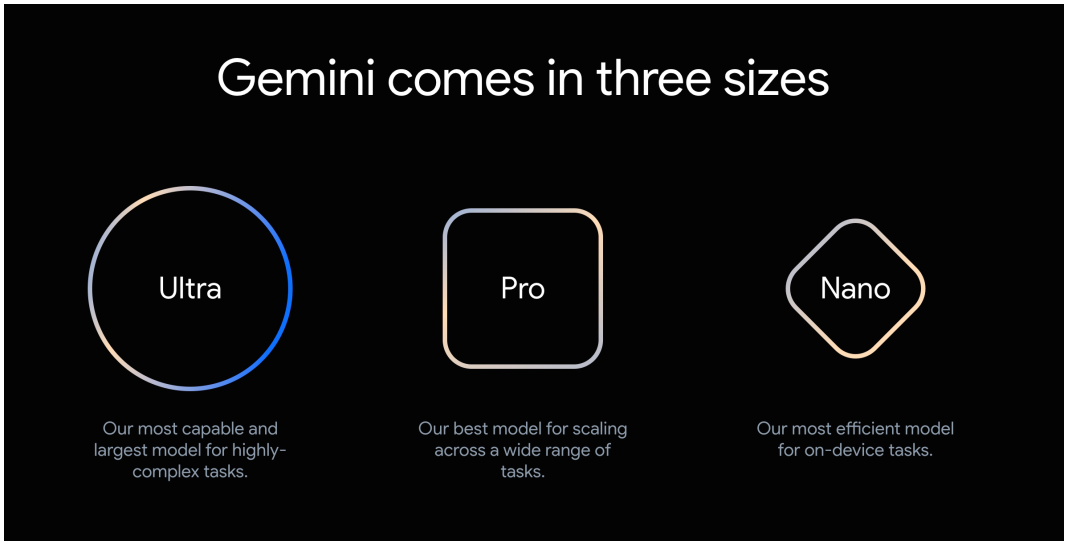


Figura 100: Modelos Gemini de Google

Sería interesante comprobar si alguno de estos modelos de menor tamaño y más ligeros computacionalmente, puede ejecutarse localmente en el robot o en el ordenador al que está conectado (*Edge Computing*), eliminando así la necesidad de conexión a internet o de acceso a una API externa.

7.3.3 Integración de Modelos Multimodales

Explorar el uso de modelos multimodales de IA generativa representa una dirección prometedora para enriquecer la interacción del robot NAO. Estos modelos avanzados tienen la capacidad de procesar entradas de diversas modalidades (*figura 101*), no limitándose solo al texto, sino también integrando datos provenientes de otras fuentes, comúnmente audio y video.

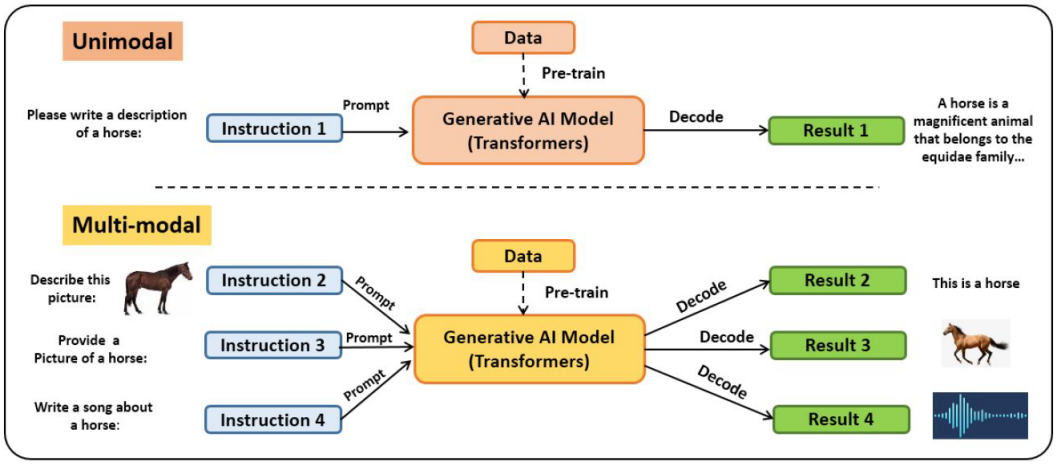


Figura 101: Modelos Unimodales vs Multimodales [92]

Bibliografía

[1] «Zoon politikón,» [En línea]. Available: https://es.wikipedia.org/wiki/Zoon_politik%C3%B3n. [Último acceso: Abril 2024].

[2] «Documentación en línea de DialogFlow,» [En línea]. Available: https://cloud.google.com/dialogflow/docs. [Último acceso: Abril 2024].

[3] «Introducing ChatGPT,» [En línea]. Available: https://openai.com/blog/chatgpt. [Último acceso: Abril 2024].

[4] «EIAROB,» [En línea]. Available: https://www.cartif.es/eiarob/. [Último acceso: Abril 2024].

[5] «Robots y telerehabilitación para el cuidado de dependientes - Artículo El Día de Segovia,» [En línea]. Available: https://www.eldiasegovia.es/noticia/zcecbbbe5-9fa8-0174-32ea3947d23d356b/202403/robots-y-telerehabilitacion-para-el-cuidado-de-dependientes. [Último acceso: Abril 2024].

[6] «Wikipedia: Historia de la robótica,» [En línea]. Available: https://es.wikipedia.org/wiki/Rob%C3%B3tica#Historia_de_la_rob%C3%B3tica. [Último acceso: Junio 2024].

[7] M. Heerink, "Assessing acceptance of assistive social robots by aging adults", tesis doctoral, University of Amsterdam, 2010.

[8] I.Orha y S. Oniga, «Assistance and telepresence robots: a solution for elderly people,» *Carpathian Journal of Electronic and Computer Engineering,* 2012.

[9] «ITAP - Robot for Rehabilitation of the Functions of the Hand Through Active Therapies,» [En línea]. Available: https://roboticamedica.itap.uva.es/projects/robhand. [Último acceso: Junio 2024].

[10] T. Fong, I. Nourbakhsh y K. Dautenhahn, «A Survey of socially interactive robots,» *Robotics and Autonomous Systems,* vol. 42, 2003.

[11] F. Hegel, C. Muhl, B. Wrede, M. Hielscher-Fastabend y G. Sagerer, «Understanding Social Robots,» de *Second International Conferences on Advances in Computer-Human Interactions*, Cancun, Mexico, 2009.

[12] A. Leung, I. Zhao, S. Lin y T. Lau, «Exploring the Presence of Humanoid Social Robots at Home and Capturing Human-Robot Interactions with Older Adults: Experiences from Four Case Studies,» *Healthcare,* vol. 11, 2023.

[13] F. Fracasso et al., «Social Robots Acceptance and Marketability in Italy and Germany: A Cross-National Study Focusing on Assisted Living for Older Adults,» *International Journal of Social Robotics,* vol. 14, 2022.

[14] K. Kabacińska, T. Prescott y J. Robillard, «Socially Assistive Robots as Mental Health Interventions for Children: A Scoping Review,» *International Journal of Social Robotics,* vol. 13, 2021.

[15] C. J. Moerman, L. van der Heide y M. Heerink, «Social robots to support children's well-being under medical treatment: A systematic state-of-the-art review,» *Journal of Child Health Care,* 2019.

[16] F. Sartorato, L. Przybylowski y D. K. Sarko, «Improving therapeutic outcomes in autism spectrum disorders: Enhancing social communication and sensory processing through the use of interactive robots,» *Journal of Psychiatric Research,* 2017.

[17] C. Moro, S. Lin, G. Nejat y A. Mihailidis, «Social Robots and Seniors: A Comparative Study on the Influence of Dynamic Social Features on Human–Robot Interaction,» *International Journal of Social Robotics,* vol. 11, 2019.

[18] W. Moyle, M. Bramble, C. J. Jones y J. E. Murfield, «"She Had a Smile on Her Face as Wide as the Great Australian Bite": A Qualitative Examination of Family Perceptions of a Therapeutic Robot and a Plush Toy,» *The Gerontologist,* vol. 59, 2019.

[19] S. Jeong et al., «A Robotic Positive Psychology Coach to Improve College Students' Wellbeing,» de *International Conference on Robot and Human Interactive Communication (RO-MAN)*, Naples, Italy, 2020.

[20] M. Bogliolo, F. Bogliolo, F. Operto y M. Emanuele, «NAO: A Promising tool for Pediatric Hospitals,» de *International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA)*, Victoria, Seychelles, 2024.

[21] M. Karim et al., «Evaluate Effectiveness of NAO Robot to Train Children with Autism Spectrum Disorder (ASD),» de *2023 IEEE 5th International Conference on Cognitive Machine Intelligence (CogMI)*, Atlanta, GA, USA, 2023.

[22] I. Hameed et al., «Robots That Can Mix Serious with Fun,» de *The International Conference on Advanced Machine Learning TEchnologies and Applications (AMLTA)*, Cairo, Egypt, 2018.

[23] F. Sacco et al., «An Antropomorphic Robot with ChatGPT for Learning Activities: The Teachers' Perspective,» de *IEEE International Conference on Metrology for*

eXtended Reality, Artificial Intelligence and Neural Engineering (MetroXRAINE), Milano, Italy, 2023.

[24] P. Joglekar y V. Kulkarni, «Humanoid Robot as a Companion for the Senior Citizens,» de *2018 IEEE Pune Section International Conference (PuneCon)*, Pune, India, 2018.

[25] P. T. V. Bhuvaneswari et al., «Humanoid robot based physiotherapeutic assistive trainer for elderly health care,» de *2013 International Conference on Recent Trends in Information Technology (ICRTIT)*, Chennai, India, 2013.

[26] T. Zhang, W. Zhang, L. Qi y L. Zhang, «Falling detection of lonely elderly people based on NAO humanoid robot,» de *2016 IEEE International Conference on Information and Automation (ICIA)*, Ningbo, China, 2016.

[27] T. Körtner, «Ethical challenges in the use of social service robots for elderly people,» *Z Gerontol Geriat,* vol. 49, 2016.

[28] A. Bao, Y. Zeng y E. Lu, «Mitigating emotional risks in human-social robot interactions through virtual interactive environment indication,» *Humanities and Social Sciences Communications,* vol. 10, 2023.

[29] T. Modi, «The Uncanny Valley,» [En línea]. Available: https://medium.com/@tanishkmodi6/the-uncanny-valley-cf70ca70979c. [Último acceso: Junio 2024].

[30] B. R. Duffy y G. Joue, «The Paradox of Social Robotics: A Discussion,» de *2005 AAAI Fall Symposium*, 2005.

[31] «Wikipedia: Weak artificial intelligence,» [En línea]. Available: https://en.wikipedia.org/wiki/Weak_artificial_intelligence. [Último acceso: Junio 2024].

[32] A. M. Turing, «Computing Machinery and Intelligence,» *Mind,* vol. 59, 1950.

[33] S. Schuchmann, «Analyzing the Prospect of an Approaching AI Winter,» 2019. [En línea]. Available: https://www.researchgate.net/publication/333039347_Analyzing_the_Prospect_of_an_Approaching_AI_Winter. [Último acceso: Junio 2024].

[34] W. S. McCulloch y W. Pitts, «A logical calculus of the ideas immanent in nervous activity,» *Bulletin of Mathematical Biophysics,* vol. 5, 1943.

[35] F. Rosenblatt, «The perceptron: A probabilistic model for information storage and organization in the brain,» *Psychological Review,* vol. 65, 1958.

[36] M. Minsky y S. Papert, Perceptrons: an introduction to computational geometry, 1969.

[37] A. Vaswani et al, «Attention Is All You Need,» 2017. [En línea]. Available: arXiv:1706.03762. [Último acceso: Junio 2024].

[38] «Transformers, explained: Understand the model behind GPT, BERT, and T5,» 2021. [En línea]. Available: https://www.youtube.com/watch?v=SZorAJ4I-sA. [Último acceso: Junio 2024].

[39] «Tokenizer de OpenAI,» [En línea]. Available: https://platform.openai.com/tokenizer. [Último acceso: Junio 2024].

[40] «But what is a GPT? Visual intro to transformers,» [En línea]. Available: https://www.youtube.com/watch?v=wjZofJX0v4M. [Último acceso: Junio 2024].

[41] D. Bahdanau, K. Cho y Y. Bengio, «Neural Machine Translation by Jointly Learning to Align and Translate,» 2014. [En línea]. Available: arXiv:1409.0473. [Último acceso: Junio 2024].

[42] «Attention in transformers, visually explained,» 2024. [En línea]. Available: https://www.youtube.com/watch?v=eMlx5fFNoYc. [Último acceso: Junio 2024].

[43] «Illustrated Guide to Transformers Neural Network: A step by step explanation,» [En línea]. Available: https://www.youtube.com/watch?v=4Bdc55j80l8. [Último acceso: 2024 Junio].

[44] «[1hr Talk] Intro to Large Language Models,» [En línea]. Available: https://www.youtube.com/watch?v=zjkBMFhNj_g. [Último acceso: Junio 2024].

[45] «LMSYS Chatbot Arena Leaderboard,» [En línea]. Available: https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard. [Último acceso: 2024 Junio].

[46] «Reinforcement Learning with Spot,» [En línea]. Available: https://www.youtube.com/watch?v=Kf9WDqYKYQQ&t=220s. [Último acceso: Junio 2024].

[47] «Making Chat (ro)Bots,» [En línea]. Available: https://www.youtube.com/watch?v=djzOBZUFzTw. [Último acceso: Junio 2024].

[48] «Optimus - Gen 2,» [En línea]. Available: https://www.youtube.com/watch?v=cpraXaw7dyc. [Último acceso: Junio 2024].

[49] «Figure 01 integrates OpenAI models,» [En línea]. Available: https://twitter.com/coreylynch/status/1767927194163331345. [Último acceso: Junio 2024].

[50] «Breadcrumbs to the Goal: Goal-Conditioned Exploration from Human-in-the-Loop Feedback,» [En línea]. Available: https://human-guided-exploration.github.io/HuGE/. [Último acceso: Junio 2024].

[51] «Mobile ALOHA - Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation,» [En línea]. Available: https://mobile-aloha.github.io/. [Último acceso: Junio 2024].

[52] «Documentación NAO: Developer Guide,» [En línea]. Available: http://doc.aldebaran.com/2-8/family/nao_technical/index_naov6.html. [Último acceso: Junio 2024].

[53] «Documentación NAO: Robot Settings,» [En línea]. Available: http://doc.aldebaran.com/2-8/family/nao_user_guide/nao_robot_settings.html. [Último acceso: Junio 2024].

[54] «Documentación NAO: Choregraphe,» [En línea]. Available: http://doc.aldebaran.com/2-8/software/choregraphe/index.html. [Último acceso: Junio 2024].

[55] «Documentación NAO: Python SDK,» [En línea]. Available: http://doc.aldebaran.com/2-8/dev/python/intro_python.html. [Último acceso: Junio 2024].

[56] «Documentación NAO: NAOqi API,» [En línea]. Available: http://doc.aldebaran.com/2-8/naoqi/index.html. [Último acceso: Junio 2024].

[57] «Documentación NAO: NAOqi Framework - Key concepts,» [En línea]. Available: http://doc.aldebaran.com/2-8/dev/naoqi/index.html. [Último acceso: 2024 Junio].

[58] «Wikipedia: OpenAI,» [En línea]. Available: https://es.wikipedia.org/wiki/OpenAI. [Último acceso: Junio 2024].

[59] «Repositorio de GPT-2 en Github,» [En línea]. Available: https://github.com/openai/gpt-2. [Último acceso: Junio 2024].

[60] «OpenAI API Reference,» [En línea]. Available: https://platform.openai.com/docs/api-reference/introduction. [Último acceso: 2024 Junio].

[61] «OpenAI API Community Libraries,» [En línea]. Available: https://platform.openai.com/docs/libraries/community-libraries. [Último acceso: 2024 Junio].

[62] «OpenAI Cookbook,» [En línea]. Available: https://cookbook.openai.com/. [Último acceso: Junio 2024].

[63] «Documentación NAO: User Guide,» [En línea]. Available: http://doc.aldebaran.com/2-8/family/nao_user_guide/index.html. [Último acceso: Junio 2024].

[64] «Documentación NAO: Basic Channel,» [En línea]. Available: http://doc.aldebaran.com/2-8/family/nao_user_guide/nao_life.html#basic-channel-nao. [Último acceso: Junio 2024].

[65] «Documentación NAO: Subscribing to a Channel,» [En línea]. Available: http://doc.aldebaran.com/2-8/family/nao_user_guide/nao_store_channel.html. [Último acceso: Junio 2024].

[66] «Documentación NAO: Downloading & Installing SoftBank Robotics Software,» [En línea]. Available: http://doc.aldebaran.com/2-8/dev/community_software.html. [Último acceso: Junio 2024].

[67] «Documentación NAO: Python SDK - Installation Guide,» [En línea]. Available: http://doc.aldebaran.com/2-8/dev/python/install_guide.html#python-install-guide. [Último acceso: Junio 2024].

[68] «OpenAI API: Quickstart,» [En línea]. Available: https://platform.openai.com/docs/quickstart?context=python. [Último acceso: Junio 2024].

[69] «Documentación NAO: Interacting with NAO,» [En línea]. Available: http://doc.aldebaran.com/2-8/family/nao_user_guide/nao_interacting.html. [Último acceso: Junio 2024].

[70] «Documentación NAO: What can I say to NAO,» [En línea]. Available: http://doc.aldebaran.com/2-8/family/nao_user_guide/spanish.html#bc-meet-nao-es. [Último acceso: Junio 2024].

[71] «Documentación NAO: Programming for a living robot,» [En línea]. Available: http://doc.aldebaran.com/2-8/ref/life/index.html. [Último acceso: Junio 2024].

[72] «Documentación NAO: Autonomous Abilities,» [En línea]. Available: http://doc.aldebaran.com/2-8/ref/life/autonomous_abilities_management.html. [Último acceso: Junio 2024].

[73] «Documentación NAO: NAOqi Python - API,» [En línea]. Available: http://doc.aldebaran.com/2-8/ref/python-api.html#naoqi-python-api. [Último acceso: Junio 2024].

[74] «Documentación NAO: Python SDK - Tutorials,» [En línea]. Available: http://doc.aldebaran.com/2-8/dev/python/tutorials.html. [Último acceso: Junio 2024].

[75] «Documentación NAO: Python SDK - Examples,» [En línea]. Available: http://doc.aldebaran.com/2-8/dev/python/examples.html. [Último acceso: Junio 2024].

[76] «Documentación NAO: qi Framework,» [En línea]. Available: http://doc.aldebaran.com/2-8/dev/libqi/index.html. [Último acceso: Junio 2024].

[77] «Documentación NAO: How to switch from NAOqi to qi Framework,» [En línea]. Available: http://doc.aldebaran.com/2-8/dev/libqi/guide/py-tonaoqi2.html#guide-py-tonaoqi2. [Último acceso: Junio 2024].

[78] «Documentación NAO: Python box,» [En línea]. Available: http://doc.aldebaran.com/2-8/software/choregraphe/objects/python_box.html. [Último acceso: Junio 2024].

[79] «Documentación NAO: Scripting Python boxes,» [En línea]. Available: http://doc.aldebaran.com/2-8/software/choregraphe/objects/python_script.html. [Último acceso: Junio 2024].

[80] «Documentación NAO: Choregraphe - Tutorials,» [En línea]. Available: http://doc.aldebaran.com/2-8/software/choregraphe/tutos/index.html. [Último acceso: Junio 2024].

[81] «Documentación NAO: Timeline box,» [En línea]. Available: http://doc.aldebaran.com/2-8/software/choregraphe/objects/timeline_box.html. [Último acceso: Junio 2024].

[82] «Documentación NAO: Timeline Panel,» [En línea]. Available: http://doc.aldebaran.com/2-8/software/choregraphe/panels/timeline_panel.html. [Último acceso: Junio 2024].

[83] «Open AI: Documentation - Overview,» [En línea]. Available: https://platform.openai.com/docs/overview. [Último acceso: Junio 2024].

[84] «Open AI: Chat Completions API,» [En línea]. Available: https://platform.openai.com/docs/guides/text-generation/chat-completions-api. [Último acceso: Junio 2024].

[85] «Open AI: Assistants API,» [En línea]. Available: https://platform.openai.com/docs/assistants/overview. [Último acceso: Junio 2024].

[86] «OpenAI: Modelos,» [En línea]. Available: https://platform.openai.com/docs/models. [Último acceso: Junio 2024].

[87] «OpenAI: Pricing,» [En línea]. Available: https://openai.com/api/pricing/. [Último acceso: Junio 2024].

[88] «OpenAI: API Reference - Chat Completions,» [En línea]. Available: https://platform.openai.com/docs/api-reference/chat/create. [Último acceso: Junio 2024].

[89] «Documentación NAO: QiChat,» [En línea]. Available: http://doc.aldebaran.com/2-8/naoqi/interaction/dialog/dialog.html. [Último acceso: Junio 2024].

[90] J. Bramauer, «GitHub Repo: JBramauer/pepperspeechrecognition,» [En línea]. Available: https://github.com/JBramauer/pepperspeechrecognition. [Último acceso: Junio 2024].

[91] M. Heering, B. Kröse, V. Evers y B. Wielinga, «Assessing Acceptance of Assistive Social Agent Technology by Older Adults: the Almere Model,» *International Journal of Social Robotics,* vol. 2, 2010.

[92] W. Hariri, «Unlocking the Potential of ChatGPT: A Comprehensive Exploration of its Applications, Advantages, Limitations, and Future Directions in Natural Language Processing,» 2023. [En línea]. Available: arXiv:2304.02017. [Último acceso: Junio 2024].