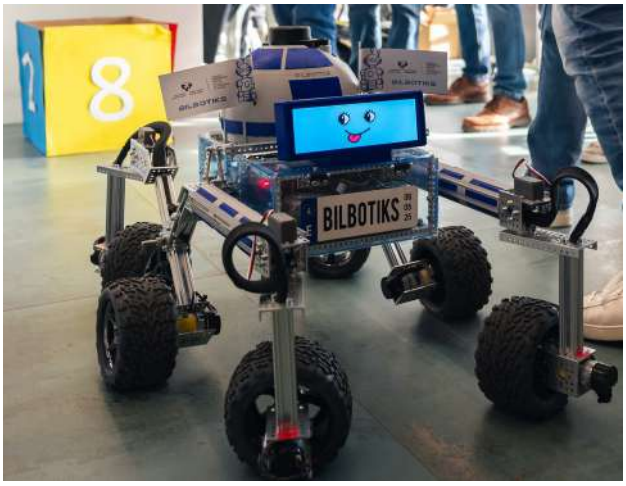
Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

DESARROLLO DE ALGORITMOS DE PERCEPCIÓN Y CONTROL EN ROS2 PARA ROBÓTICA MÓVIL



Estudiante: Arambarri Calvo, Javier

Directora: Cabanes Axpe, Itziar
Codirector: López Novoa, Unai

Curso: 2024-2025

Fecha: Portugalete, 20 de julio de 2025

Agradecimientos

A mi familia, por vuestro apoyo y compañía durante todo el grado.

A Izaro, por tu cariño.

A Ane, Mario, Pablo, Nagore y Álvaro, por ser parte de la familia BilboTiks.

A César Pérez Barrio, por ser una fuente de inspiración y un ejemplo de esfuerzo y dedicación.

A Itziar Cabanes Axpe, por haber liderado el proyecto BilboTiks y haberme brindado la oportunidad de formar parte de él.

A Unai López Novoa, por tu tiempo, interés y colaboración en este trabajo.

Resumen Laburpena Abstract

La robótica móvil e inteligente presenta retos como la percepción del entorno y el control suave, preciso y seguro de los actuadores con el fin de lograr movimientos autónomos. Este Trabajo de Fin de Grado utiliza herramientas de inteligencia artificial, teoría de control y automatización para desarrollar soluciones en *ROS2* sobre la plataforma robótica de código abierto *JPL-OSR*.

Estas soluciones se ponen en valor en el concurso estatal *Sener CEA's Bot Talent 2025* con el objetivo de reconocer números y colores y desplazarse autónomamente por un laberinto cerrado.

Palabras clave: robótica móvil e inteligente, movimientos autónomos, inteligencia artificial, percepción, control, código abierto, *ROS2*.

Robotika mugikor eta adimendunak zenbait erronka ditu, hala nola ingurunea hautematea eta eragingailuak leun, zehatz eta seguru kontrolatzea, mugimendu autonomoak lortzeko. Gradu Amaierako Lan honek adimen artifizialeko tresnak, kontrol-teoria eta automatizazioa erabiltzen ditu *JPL-OSR* kode irekiko plataforma robotikoan *ROS2* soluzioak garatzeko.

Soluzio horiek Sener CEA's Bot Talent 2025 estatuko lehiaketan balioesten dira, zenbakiak eta koloreak antzemateko eta labirinto itxi batean robota modu autonomoan mugitzeko helburuarekin.

Gako-hitzak: robotika mugikor eta adimenduna, mugimendu autonomoak, adimen artifiziala, ingurunea hautematu, eragingailuak kontrolatu, kode irekia, *ROS2*.

Intelligent mobile robotics presents challenges such as environmental sensing and smooth, accurate and safe control of actuators in order to achieve autonomous movements. This Final Degree Project uses artificial intelligence, control theory and automation tools to develop solutions in *ROS2* on the open source robotics platform *JPL-OSR*.

These solutions are highlighted in the Spanish competition *Sener CEA's Bot Talent 2025* with the objective of recognizing numbers and colors and moving autonomously through a closed maze .

Keywords: intelligent mobile robotics, autonomous movements, artificial intelligence, environmental sensing, control of actuators, open source, *ROS2*.

Índice de contenidos

Agradecimientos **3**

Resumen Laburpena Abstract **5**

Índice de contenidos **10**

Índice de ilustraciones **11**

Índice de tablas **15**

Índice de códigos **18**

Lista de acrónimos **21**

Lista de definiciones **23**

1 Introducción **25**

1.1 Contexto 25

1.2 Motivación 28

1.3 Estructura de la memoria 28

2 Objetivos, alcance y beneficios **30**

2.1 Objetivos 30

2.2 Alcance 30

2.3 Beneficios y ODS 31

3 Estado del arte **32**

3.1 Soluciones basadas en IA para percepción del entorno 32
3.2 Algoritmos de control 35
3.3 Arquitecturas en *ROS2* 36

4 Descripción de las especificaciones 37
4.1 Establecimiento de *checkpoints* 37
4.2 Descripción de las pruebas 38

5 Componentes y montaje del rover 40
5.1 Estructura mecánica 40
5.2 Componentes electrónicos y sensores 42
5.3 Dispositivos de control y monitorización 47
5.4 Dispositivos de comunicación 49
5.5 Montaje del *rover* 50

6 Análisis y diseño de la arquitectura 51
6.1 Captura de operaciones 51
6.2 Captura de procedimientos 52
6.3 Especificación de procedimientos 53
6.4 Modelo físico 55
6.5 Diseño de la arquitectura 56
6.6 Diagramas de secuencia 57

7 Desarrollo de la solución 59
7.1 *ROS2* 59
7.2 Paquetes, nodos y mensajes desarrollados en *ROS2* 63
7.3 Movimiento del *rover* 70
7.4 Prueba de percepción 72
7.5 Prueba de control 91
7.6 Interfaces gráficas desarrolladas 99
7.7 Gemelo funcional 101
7.8 Simulación con *Gazebo Harmonic* y *Docker* 102

8 Verificación y evaluación **104**

8.1 Prueba de percepción 104

8.2 Prueba de control 107

9 Planificación **109**

9.1 Definición de tareas 109

9.2 Diagrama *Gantt* 122

9.3 Metodología *Scrum* 124

9.4 Análisis de riesgos 126

9.5 Herramientas 133

9.6 Evaluación económica 134

10 Conclusiones y líneas futuras **137**

10.1 Conclusiones y evaluación de los objetivos 137

10.2 Líneas futuras de trabajo 138

10.3 Repositorio del proyecto 138

Bibliografía **139**

Anexo I: Configuración de la conectividad en el *rover* **146**

I.1. Instalación del sistema operativo 147

I.2. Configurar la red *Wi-Fi* 147

I.3. Configuración de *SSH* 148

I.4. Habilitar puertos serie e *I2C* 149

I.5. Establecer las reglas *udev* 150

I.6. Modificar el fichero *~/.bashrc* 152

I.7. *Firmware* de *Roboclaw* 152

I.8. Librerías *software* utilizadas 152

Anexo II: Código **154**

II.1. Fichero *.yaml* de parámetros del proyecto 155

II.2. Servidor y respuesta del servicio para sacar una foto 156

II.3. Cliente y solicitud del servicio para sacar una foto 157

II.4. Desarrollo del modelo *CNN* 158
II.5. Comprobación del funcionamiento del *LiDAR* 169
II.6. Generar los gráficos de la señal de control en *MATLAB* 170
II.7. Nodos 171
II.8. Interfaces gráficas 196
II.9. Gemelo digital y simulación 206

Índice de ilustraciones

1 *SHAKEY*. 26

2 A la izquierda, el *JPL Open Source Rover Project*. A la derecha, el *Perseverance* en Marte [17]. 27

3 Ejemplo del resultado tras aplicar el algoritmo *Canny* [27]. 32

4 Representación visual de una red neuronal artificial [31]. 35

5 Representación gráfica del algoritmo *ND* [36]. 35

6 Arquitectura por niveles de tareas en *ROS2* [41]. 36

7 A la izquierda, ejemplo con las especificaciones de la caja cuadrada. A la derecha, ejemplo con las especificaciones del laberinto. 39

8 Subsistemas de la estructura mecánica del *rover*: chasis (izquierda), ruedas (centro) y caja de metracrilato donde se colocarán las placas de control (derecha).[16]. 40

9 Perfil de aluminio de montaje universal *goRAIL* [43]. 41

10 Esquema de suspensión *rocker-boogie* [45]. 41

11 Movimiento del *rocker* y del *boogie*. 41

12 Placa base conectada y bus de conexión con la *Raspberry Pi*. 42

13 Placa base [16]. 42

14 A la izquierda, el motor con *encoder* [46]. A la derecha, el servomotor [47]. 44

15 Aspecto visual de la cámara *Orbbec Astra Pro Plus* [48]. 44

16 A la izquierda, el *LiDAR YDLIDAR X4* [49]. A la derecha, la regla de la mano izquierda sobre el dispositivo físico. 45

17 *IMU* [51]. 46
18 A la izquierda, la batería [53]. A la derecha, la batería colocada en el *rover*. . 46
19 A la izquierda, *Roboclaw 2x7A Motor Controller* [54]. A la derecha, el controlador serie colocado en el robot. 47
20 *Raspberry Pi 4B* [55]. 48
21 Arriba, aspecto exterior de la pantalla. Abajo, su circuito integrado. [56]. . . 48
22 A la izquierda, el interruptor. A la derecha, el *display* de voltaje. 49
23 *Router 4G* [58]. 49
24 Arriba, la cúpula que alberga el *LiDAR* y la cámara. Abajo, el marco 3D de la pantalla. 50
25 *Rover* montado. 50

26 Modelo de proceso. 51
27 Modelo de procedimientos. 52
28 Modelo físico. 55
29 Arquitectura propuesta diseñada. 56
30 Diagrama de secuencia de la prueba de percepción. 57
31 Diagrama de secuencia de la prueba de control. 58

32 *ROS2 Humble Hawksbill*. 60
33 Esquema de tópicos y servicios en *ROS2* [62]. 61
34 Esquema de acciones en *ROS2* [62]. 61
35 Grafo computacional *ROS2* desarrollado. 67
36 Esquema oficial de nodos *lifecycle* [65]. 69
37 Nodos involucrados en la prueba de percepción. 72
38 *Imagen con 8 bits por píxel y representación de un fragmento de 8x8 píxeles [67].* 73
39 Servicio para sacar una foto. 74
40 *Representación del espacio de color RGB de 24 bits. Imagen propia.* 75
41 *Representación del espacio de color HSV [72].* 77
42 Resultado de la primera restricción, donde se aprecia la caja como elemento principal de la imagen. 77

43 Resultado de la segunda restricción. No se detecta el color azul con forma circular, no rectangular ni cuadrada. . . . 78
44 *Ejemplo de OCR. Fuente: [74].* . . . 81
45 Esquema del funcionamiento de una *CNN* [75]. . . . 82
46 Ejemplos del *dataset* generado. . . . 83
47 Ejemplo de imágenes eliminadas del *dataset* generado. . . . 83
48 Ejemplo del resultado de aplicar el algoritmo de detección de bordes *Canny* 84
49 Ejemplos de las imágenes sintéticas generadas. . . . 85
50 Precisión y pérdida del modelo durante el entrenamiento. . . . 89
51 A la izquierda, los servomotores en posición para girar sobre sí mismo. A la derecha, la representación general de *roll*, *pitch* y *yaw* [39]. . . . 90
52 Nodos involucrados en la prueba de control. . . . 91
53 Mediciones del *LiDAR* desordenadas. . . . 92
54 Posicionamiento del *LiDAR* en el *rover*. . . . 93
55 Campo de visión inicialmente propuesto. . . . 94
56 Campo de visión implementado. . . . 94
57 Representación gráfica del error máximo. . . . 95
58 Señal de control de salida del controlador proporcional con diferentes valores *Kp*. . . . 96
59 Señal de control de salida del controlador PD con diferentes valores *Kp* y *Kd*. 98
60 Interfaz gráfica para la prueba de percepción. . . . 99
61 Interfaz gráfica para la prueba de control. . . . 99
62 Interfaz gráfica de la cara. . . . 100
63 Valores de la *IMU* generados por el gemelo funcional. . . . 101
64 Capturas de pantalla de la simulación de la prueba de control en *Gazebo Harmonic*. . . . 102
65 Escenario de pruebas para la prueba de percepción. . . . 106
66 Escenarios de pruebas para la prueba de control. . . . 108
67 EDT. . . . 111
68 Diagrama *Gantt*. . . . 123

69 Distribución de los costes del proyecto. 136

70 *Raspberry Pi Imager*. 147

71 Configuración de la red. 147

72 Conectarse automáticamente a la red. 148

73 Herramienta *rapi-config*. 150

Índice de tablas

1 Características del modelo de servomotor utilizado. 44

2 Especificación de los procedimientos de la prueba de percepción. 53
3 Especificación de los procedimientos de la prueba de control. 54
4 Especificación de los procedimientos compartidos por ambas pruebas. . . 54

5 Estados de un nodo con ciclo de vida. 68
6 Transiciones de un nodo con ciclo de vida. 69
7 *Tipo de compresión de imágenes según el formato.* 74
8 Colores en *HSV* de *OpenCV*. 79
9 Rangos de tonalidad en *HSV* de *OpenCV*. 79
10 Rangos de saturación y luminosidad en *HSV* de *OpenCV*. 80
11 Número de imágenes que integran el *dataset* por clases. 86
12 Resultados de las métricas de evaluación del entrenamiento del modelo *CNN* desarrollado. 89
13 Ángulos límite de actuación de los servomotores. 94
14 Descripción de las características del *LiDAR* virtual. 103

15 Test unitarios de los procedimientos de la prueba de percepción. 105
16 Test de integración de la prueba de percepción. 106
17 Test de verificación y validación de la prueba de percepción. 106
18 Test unitarios de los procedimientos de la prueba de control. 107
19 Test de integración de la prueba de control. 108
20 Test de verificación y validación de la prueba de control. 108

21 Tareas, duración, precedencias y fecha estimada. 110
22 Detalles de la tarea 1.1. 112
23 Detalles de la tarea 1.2. 112
24 Detalles de la tarea 1.3. 113
25 Detalles de la tarea 1.4. 113
26 Detalles de la tarea 1.5. 114
27 Detalles de la tarea 1.6. 114
28 Detalles de la tarea 1.7. 115
29 Detalles de la tarea 2.1. 115
30 Detalles de la tarea 3.1. 116
31 Detalles de la tarea 3.2. 116
32 Detalles de la tarea 3.3. 117
33 Detalles de la tarea 3.4. 117
34 Detalles de la tarea 3.5. 117
35 Detalles de la tarea 4.1. 118
36 Detalles de la tarea 4.2. 118
37 Detalles de la tarea 4.3. 119
38 Detalles de la tarea 5.1. 119
39 Detalles de la tarea 5.2. 120
40 Detalles de la tarea 6.1. 120
41 Detalles de la tarea 6.2. 121
42 Detalles de la tarea 6.3. 121
43 Detalles de la tarea 6.4. 122
44 Detalles de la tarea 6.5. 122
45 *Sprints* e historias de usuario. 124
46 Matriz de evaluación de riesgos según el estándar ISO 45001:2018. 126
47 Probabilidad y consecuencias de los riesgos. 126
48 Riesgos asociados al proyecto. 127
49 Evaluación de indisposición provisional del desarrollador por empleo o prácticas. 127

50 Evaluación de no completar en fecha algún *checkpoint* eliminatorio del concurso. 127
51 Evaluación de no cumplir el plazo de entrega de este trabajo. 128
52 Evaluación de averías técnicas generales de *software*. 128
53 Evaluación de cambio en el *firmware* de los componentes. 128
54 Evaluación de actualizaciones de las librerías utilizadas. 128
55 Evaluación de planificación temporal incorrecta. 129
56 Evaluación de no documentar el desarrollo completo durante su realización.129
57 Evaluación de no completar en fecha algún *checkpoint* no eliminatorio del concurso. 129
58 Evaluación de no cumplir los objetivos. 129
59 Evaluación de perder información del proyecto. 130
60 Evaluación de averías técnicas generales de *hardware*. 130
61 Evaluación de indisposición provisional del desarrollador por enfermedad. 130
62 Evaluación de infección de virus informático. 131
63 Evaluación de apagones de la red eléctrica. 131
64 Evaluación de cambio de las especificaciones del proyecto. 131
65 Evaluación de batería descargada del *rover* u ordenador de trabajo. 132
66 Evaluación de perder comunicación con el *rover*. 132
67 Evaluación de rotura del ordenador de trabajo. 132
68 Evaluación de sobrescribir alguna versión de desarrollo al generar una nueva.133
69 Costes de personal. 134
70 Coste del robot. 135
71 Costes de herramientas *software* y *hardware*. 135
72 Evaluación económica total. 136

73 Librerías utilizadas y versiones. 153

Índice de códigos

7.1 Definición del mensaje *geometry_msgs/msg/Vector3*. 62
7.2 Definición del mensaje *sensor_msgs/msg/LaserScan*. 62
7.3 Ejemplo de definición del mensaje de un servicio. 63
7.4 Ejemplo de definición del mensaje de una acción. 63
7.5 Mensaje personalizado para mover los motores. 64
7.6 Mensaje personalizado para mover los servomotores. 64
7.7 Mensaje personalizado para la *IMU*. 65
7.8 Mensaje personalizado para la interfaz gráfica de la cara. 65
7.9 Mensaje personalizado para la interfaz gráfica de la prueba de percepción. 65
7.10 Mensaje personalizado para la interfaz gráfica de la prueba de control. . . 65
7.11 Mensaje personalizado para el servicio de sacar una foto. 66
7.12 Mensaje personalizado para el servicio de adivinar el color y el número. . 66
7.13 Mensaje personalizado para la acción de girar sobre sí mismo según el color y número identificado. 66
7.14 Generar valores distribuidos uniformemente en *Python*. 101
7.15 Definición del *LiDAR* virtual en el fichero *.sdf*. 103
10.1 Comandos *bash* para la instalación del servidor *SSH*. 148
10.2 Comandos *bash* para la instalación del cliente *SSH*. 149
10.3 Comandos *bash* para instalar y acceder a *raspi-config*. 149
10.4 Comando *bash* para reiniciar la *Raspberry Pi*. 149
10.5 Comando *bash* para crear la regla *udev* del puerto serie. 150
10.6 Regla *udev* del puerto serie. 151
10.7 Comando *bash* para crear la regla *udev* del puerto *I2C*. 151

10.8 Regla *udev* del puerto *I2C*. 151
10.9 Comando *bash* para añadir el usuario a los grupos con permisos en los puertos serie e *I2C*. 151
10.10 Activar automáticamente el espacio de trabajo de *ROS2*. 152
10.11 Contenido del fichero *.yaml* de parámetros del proyecto. 155
10.12 Implementación del servidor del servicio para sacar una foto. 156
10.13 Implementación de la función de respuesta del servidor del servicio para sacar una foto. 156
10.14 Solicitud del servicio desde el nodo *pertzepzio_proba_wrapper* para sacar una foto. 157
10.15 Implementación de la primera restricción geométrica. 158
10.16 Implementación de la segunda restricción geométrica. 158
10.17 Implementación y aplicación del modelo de color *HSV* y las máscaras con *OpenCV*. 159
10.18 Capturar imágenes para crear el *dataset*. 160
10.19 Aplicación del algoritmo *Canny* al *dataset* generado. 161
10.20 Generación de imágenes sintéticas. 162
10.21 División del *dataset* en *train*, *val* y *test*. 165
10.22 Codificación de la arquitectura y del entrenamiento del modelo *CNN* generado. 166
10.23 Generar los gráficos de las métricas del modelo *CNN* en *MATLAB*. 168
10.24 Comprobación del funcionamiento del *LiDAR*. 169
10.25 Generar los gráficos de la señal de control en *MATLAB*. 170
10.26 Código del nodo *servoak*. 171
10.27 Código del nodo *motorrak_roboclaw*. 173
10.28 Código del nodo *imu*. 176
10.29 Código del nodo *lidar*. 178
10.30 Código del nodo *kamera*. 181
10.31 Código del nodo *pertzepzio_proba_wrapper*. 184
10.32 Código del nodo *kontrol_proba_wrapper*. 190
10.33 Código del nodo *pertzepzio_proba*. 194

10.34 Código del nodo de la interfaz gráfica de la prueba de percepción. 196
10.35 Código del nodo de la interfaz gráfica de la prueba de control. 199
10.36 Código del nodo de la interfaz gráfica de la cara. 202
10.37 Código del nodo *imu_sim*. 206
10.38 Fichero *.sdf* para la definición del mundo y el robot en *Gazebo Harmonic*. 208
10.39 Código del nodo de *Gazebo Harmonic* para la simulación de la prueba de control. 215

Lista de acrónimos

AGV *Automated Guided Vehicle*

AMR *Autonomous Mobile Robot*

API *Application Programming Interface*

ARM *Advanced RISC Machine*

BOE Boletín Oficial del Estado

CEA Comité Español de Automática

CNN *Convolutional Neural Network*

COTS *Commercial Off The Shelf*

CPU *Central Process Unit*

DC *Direct Current*

DDS *Data Distribution Service*

EDT Estructura de Descomposición del Trabajo

EIB Escuela de Ingeniería de Bilbao

FPS *Frames Per Second*

FSM *Finite State Machine*

IA Inteligencia Artificial

IFR *International Federation of Robotics*

I2C *Inter-Integrated Circuit*

JPL *Jet Propulsion Laboratory*

KNN *K-Nearest Neighbors*

LCD *Liquid Crystal Display*

LiDAR *Light Detection And Ranging*

LTS *Long Term Support*

MIT *Massachusetts Institute of Technology*

MLP *Multilayer Perceptions*

NASA *National Aeronautics and Space Administration*

ND *Nearness Diagram*

OCR *Optical Character Recognition*

ODS Objetivos de Desarrollo Sostenible

OSRF *Open Source Robotics Foundation*

PCB *Printed Circuit Board*

PID Proporcional, Integral, Derivativo

PWM *Pulse Width Modulation*

RAE Real Academia Española

RAM *Random Access Memory*

RC *Radio Control*

ROS *Robot Operating System*

ROS2 *Robot Operating System 2*

SBC *Single Board Computer*

SDK *Software Development Kit*

SLAM *Simultaneous Localization And Mapping*

SRI *Stanford Research Institute*

SSH *Secure Shell*

SVM *Support Vector Machine*

TFG Trabajo de Fin de Grado

TTL *Transistor-Transistor Logic*

Lista de definiciones

Framework: conjunto de herramientas y bibliotecas que ofrece al usuario una organización y estructura básica para comenzar los desarrollos, generalmente de programación.

Robot móvil: máquina equipada con una estructura mecánica, *hardware* y *software* que le permite desplazarse de un lugar a otro.

Robot móvil autónomo: conocido por sus siglas en inglés *AMR* (*Autonomous Mobile Robot*), es un robot móvil que utiliza diferentes dispositivos para percibir un entorno cambiante y desplazarse autónomamente.

Robot de guiado automático: conocido por sus siglas en inglés *AGV* (*Automated Guided Vehicle*), es un robot móvil que se desplaza autónomamente a través de guías fijas instaladas en el entorno de operación.

Rover: robot móvil diseñado principalmente para la exploración espacial, para poder desplazarse y desempeñar diferentes tareas en distintos planetas o elementos astronómicos. A pesar de ello, sus aplicaciones se extienden a cualquier ámbito porque incorporan diversas herramientas, lo que los hace muy completos y versátiles.

Robot industrial: máquina, sistema o manipulador programable y multifuncional con varios grados de libertad, principalmente autómata, concebido para su implantación en los procesos productivos de la industria con el objetivo de automatizarlos y realizar tareas peligrosas. Por ello, estos robots son capaces de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas.

Robot de servicio: máquina que realiza tareas útiles para el ser humano, excluyendo las tareas realizadas por los robots industriales, por ejemplo, las tareas del hogar o el cuidado de personas.

Robot médico: robot de servicio orientado al campo de la salud.

Robot colaborativo: robot industrial o de servicio programado y diseñado para trabajar de forma colaborativa con humanos sin que suponga un riesgo físico para las personas. Suelen incluir sistemas que evitan el choque con las personas con el fin de evitar daños. Los robots colaborativos son conocidos por el nombre de *cobots*.

Robot inteligente: cualquier robot dotado de complejos algoritmos, por ejemplo, algoritmos de inteligencia artificial para procesar y responder a las diferentes situaciones que se plantean en su entorno.

Humanoide: autómata o máquina con forma humana capaz de realizar tareas humanas gobernado por un sistema no determinista, por ejemplo, sistemas de inteligencia artifi-

cial.

Percepción: proceso que recoge información del estado del entorno mediante sensores.

Reconocimiento del entorno: proceso basado en la percepción empleado por un robot para identificarse y ubicarse en el entorno en el que se encuentra. El uso de técnicas de inteligencia y visión artificial está muy extendido para desempeñar esta tarea.

Inteligencia artificial: según la RAE, "*disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico*".

Visión artificial: rama de la inteligencia artificial que permite a las máquinas ver, procesar y comprender las imágenes para reconocer el entorno.

Gemelo digital: réplica digital y virtual lo más exacta posible del sistema físico real que permite ejecutar simulaciones del comportamiento.

Gemelo funcional: componente *software* que permite probar el correcto funcionamiento de un sistema de control o algoritmo, respondiendo mediante activaciones ficticias o simuladas de las señales de control y valores de sensores.

1. Introducción

La robótica móvil es una rama de la robótica que se centra en el diseño y desarrollo de robots capaces de desplazarse de forma autónoma en distintos entornos. Se clasifica dentro de la denominada robótica de servicios, ya que su objetivo principal es asistir o ayudar a las personas y otros seres vivos en una amplia variedad de tareas. Este tipo de robótica se ha posicionado de manera referente en sectores como la logística, la agricultura, la asistencia médica y la exploración submarina y espacial [1].

En el ámbito espacial, los robots móviles, conocidos comúnmente como *rovers*, desempeñan un papel fundamental. Estos sistemas deben ser capaces de operar en entornos desconocidos, donde se enfrentan a desafíos extremos relacionados con la percepción del entorno, la navegación autónoma, la planificación de trayectorias y el control del movimiento [2]. Ejemplos emblemáticos de estos avances son los *rovers* enviados a Marte, como *Curiosity* y *Perseverance* (ver Ilustración 2), que han demostrado la importancia de contar con algoritmos robustos para sortear obstáculos y adaptarse a terrenos irregulares.

Empresas como *Sener*, han apostado por el desarrollo de soluciones avanzadas en este campo, impulsando iniciativas centradas en el diseño de *rovers* con sistemas de percepción sofisticados, gran agilidad y capacidades de control avanzadas para moverse en caminos desconocidos. En este contexto, se enmarca el concurso promovido por la Fundación *Sener*, *Sener-CEA's Bot Talent* [3, 4], cuyo objetivo es fomentar el desarrollo de soluciones tecnológicas para estos desafíos reales.

Este TFG tiene como propósito desarrollar algoritmos que den respuesta a los retos de percepción y control de un *rover* en escenarios no controlados, poniendo a prueba su capacidad de reacción ante situaciones impredecibles. La solución propuesta busca contribuir al avance de la robótica móvil.

1.1. Contexto

La robótica móvil autónoma tiene su origen en *ELSIE* [5] en el año 1948 y en el precursor del *AGV* en 1954 [6]. Asimismo, en la misma década aparece por primera vez el concepto de robot industrial de la mano del inventor estadounidense George Devol, *Unimate*. Fue implantado en una cadena de montaje de la empresa *General Motors* en 1961 con el objetivo de automatizar tareas peligrosas para su ejecución por el ser humano. Devol fundó la primera empresa de fabricación de robots, *Unimation*, con la colaboración de Joseph Engelberger [7].

Todos estos robots eran electromecánicos, pues no existían las computadoras ni los procesadores. Las primeras computadoras aparecen a partir del año 1954 y la primera máquina controlada por una computadora en 1959, lo que permitió desarrollar los primeros *rovers* y brazos robóticos en los años 60 y 70.

Destacan *Lunojod 1*, lanzado por la Unión Soviética a la Luna en 1970 y *SHAKEY* [8], Ilustración 1, del *Stanford Research Institute (SRI)*, gobernado por cámaras, sensores y procesadores combinando *software* y *hardware*. La investigación realizada por el *SRI* obtuvo como resultado el algoritmo de búsqueda A* y los primeros algoritmos de navegación y visión artificial.

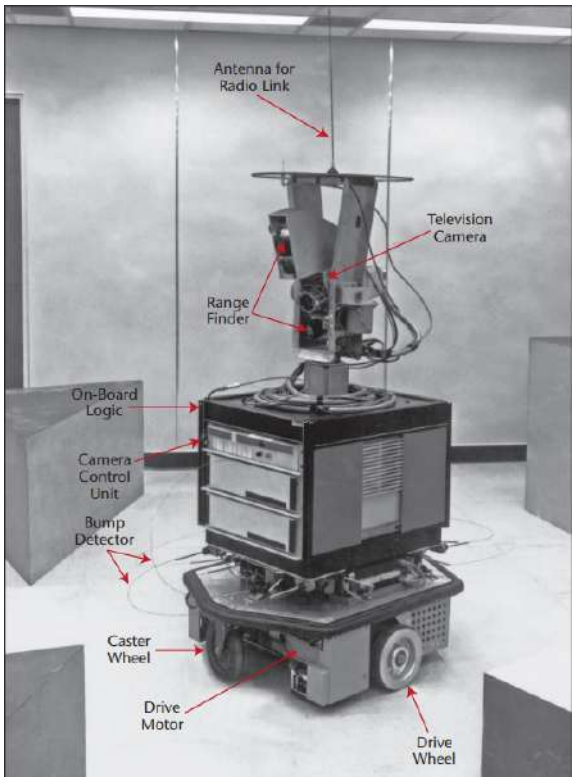


Ilustración 1: *SHAKEY.*

En la década de 1980 se desarrolla *AESOP* [9] (*Automated Endoscopic System for Optimal Positioning*) en el campo de la robótica médica. Este sistema fue el primero en emplear un brazo robótico controlado por voz para sostener y posicionar una cámara endoscópica durante cirugías mínimamente invasivas.

Estos nuevos casos de uso generaron mayor interés en el desarrollo de robots para la realización de tareas cotidianas, por lo que en la década de 1990 aparece la robótica colaborativa [10]. Este tipo de robótica pretende ocupar simultáneamente los espacios de trabajo tanto por personas como por robots sin que supongan un riesgo físico para los trabajadores.

Consecuentemente, el desarrollo de *software* comenzó a ganar mayor relevancia en el campo de la robótica debido a que se necesitaban sistemas de control más sofisticados y flexibles. Plataformas como *ORCA* [11] en 2004 dentro del proyecto *OROCOS* [12] financiado por la Unión Europea, y *ROS* [13] en 2007 comenzaron a facilitar la creación

y personalización de sistemas robóticos, como por ejemplo la robótica de servicios, permitiendo una integración más eficiente entre *hardware* y *software*.

Hoy en día, las investigaciones en el sector de la robótica se centran en proporcionar mayor capacidad de acción autónoma a las máquinas. La *International Federation of Robotics* [14] ya anticipó las tendencias en el sector de la robótica para 2025. Entre ellas resaltan aspectos como la inteligencia artificial, los robots móviles y humanoides con mayores prestaciones, la sostenibilidad y eficiencia energética y nuevos campos de negocio en la robótica, aumentando su potencial y prestaciones en nuevos sectores.

La inteligencia artificial se presenta actualmente como la principal herramienta tecnológica para dotar a los robots y otras máquinas de capacidad para realizar nuevas tareas de forma autónoma, es decir, incrementar su autonomía. En este sentido, el reconocimiento del entorno es uno de los principales retos de la robótica inteligente. El objetivo es que un robot móvil autónomo sea capaz de decidir qué tareas realizar para ubicarse en entornos desconocidos y complejos. Para ello, es necesario incorporar una sensórica y algorítmica que solucionen los problemas de locomoción y navegación autónoma e inteligente [15].

Estos recursos son el aliado perfecto de los *rovers*. Capacitar a estos robots de percepción del entorno en el espacio exterior permite desarrollar investigaciones más intensivas y detalladas sobre otros planetas y cuerpos celestes, por ejemplo, Marte y la cara oculta de la Luna, a donde los humanos no podemos llegar. De hecho, la *NASA* publicó el proyecto *JPL Open Source Rover Project* [16], Ilustración 2, para proporcionar a desarrolladores e investigadores una base replicada de un *rover* real sobre la que trabajar. En el repositorio oficial proveen el listado de materiales necesarios, indicaciones básicas para comenzar el montaje y los pasos para configurar el sistema operativo. Además, los *rover* son máquinas muy versátiles e incorporan diferentes instrumentos y funcionalidades, lo que permiten trabajar e investigar diferentes aspectos de la robótica desde un mismo lugar de trabajo. Por ejemplo, combinan aprendizaje automático y actuadores, pudiendo incluir manipuladores colaborativos para aplicaciones en la robótica de servicios, probar los algoritmos de funcionamiento mediante gemelos funcionales y ser simulados en su totalidad mediante gemelos digitales.

Ilustración 2: A la izquierda, el *JPL Open Source Rover Project*. A la derecha, el *Perseverance* en Marte [17].

En este contexto, el *framework* de programación para robots *ROS*, actualmente *ROS2* y que incluye numerosas librerías *software*, se ha popularizado como punto de partida para el desarrollo de todo tipo de aplicaciones de robótica móvil autónoma e inteligente. Como muestra de ello la librería *Nav2* [18] posibilita la navegación de robots en entornos

complejos proporcionando herramientas y algoritmos de percepción, control, visualización y localización. También incluye simuladores como *Gazebo* [19], *Webots* [20] o *MVSim* [21] y la herramienta de visualización *RViz* [22], ampliamente utilizados hoy en día.

1.2. Motivación

Este trabajo está impulsado por el concurso de robótica *Sener-CEA's Bot Talent*, organizado por el grupo de ingeniería y tecnología Sener, la Fundación Sener y el Comité Español de Automática, CEA, y el interés del autor en el estudio del campo de la robótica móvil.

Los retos del certamen incluyen tareas de trabajo en equipo, *hardware*, *software*, configuración mecánica y electrónica del robot, aprendizaje automático y utilización de *ROS2*, así como la simulación de robots. Por todo ello, resulta un entorno atractivo para desarrollar soluciones encaminadas hacia la robótica móvil autónoma con aplicación real.

Asimismo, este proyecto permite combinar el desarrollo de *software* de inteligencia artificial, una de las mayores tendencias actuales de la ingeniería informática, con herramientas de desarrollo de aplicaciones de robótica. Este factor ha sido clave en la elección y definición de la temática del trabajo, generando un aprendizaje relevante en el campo de la robótica inteligente.

1.3. Estructura de la memoria

La memoria se organiza en diez capítulos y dos anexos, donde se detalla la configuración del *rover*, las librerías empleadas y el código resultante de la solución desarrollada.

La lista de acrónimos contiene el significado de las siglas mencionadas en el trabajo.

La lista de definiciones describe el significado de diferentes conceptos nombrados a lo largo de la memoria.

En el Capítulo 1 se introduce el trabajo junto a su contexto y motivación para su realización.

En el Capítulo 2 se presenta el objetivo, alcance y los beneficios de realizar este trabajo.

En el Capítulo 3 se realiza un breve estudio de las soluciones actuales existentes en el marco del objetivo establecido.

En el Capítulo 4 se describen las especificaciones y requisitos con los que se trabaja y que delimitan el desarrollo de la solución.

En el Capítulo 5 se comentan los componentes físicos empleados para implementar la solución trabajada.

En el Capítulo 6 se realiza el análisis y el diseño de la arquitectura del sistema a desarrollar. Se identifican las operaciones que el robot tiene que realizar, las funciones que las implementan y el orden o secuencia de ejecución.

En el Capítulo 7 se expone el proceso del desarrollo de la solución implementada. Se

abordan diferentes posibilidades y se justifica mediante métricas y gráficos los mejores resultados obtenidos. Durante la explicación, se referencia a las porciones de código correspondientes a cada tarea presentadas en el Anexo II.

En el Capítulo 8 se describen las pruebas realizadas sobre la implementación y los resultados obtenidos para evaluarlos acorde a los requisitos establecidos.

En el Capítulo 9 se presenta la planificación la planificación establecida para el desarrollo de este trabajo. Esta planificación incluye la identificación, organización y detallado de tareas y la evaluación económica.

En el Capítulo 10 se concluye el resultado del proyecto, comparando los resultados con los objetivos. También se plantean diferentes líneas de trabajo para dar continuidad al trabajo y se enlaza el repositorio público.

Finalmente, se incluye la bibliografía consultada y los anexos I y II.

2. Objetivos, alcance y beneficios

2.1. Objetivos

El objetivo general es desarrollar algoritmos de percepción en el entorno y control que permitan a un *rover* identificar números y colores y desplazarse por un entorno cerrado. Este objetivo se enmarca en el concurso *Sener-CEA's Bot Talent*.

Asimismo, para alcanzar el objetivo general se definen los siguientes objetivos específicos:

- realizar el montaje mecánico del *rover*, ensamblando tanto los componentes mecánicos como la fijación del posicionamiento de los sensores.
- realizar el montaje electrónico del *rover*, incluyendo el cableado, la integración de los sensores y controladores con la placa base y el ordenador.
- diseñar una arquitectura de *software* modular que sea escalable y fácilmente adaptable a las diferentes necesidades de cada prueba para reutilizar código.
- implementar los algoritmos diseñados utilizando el *framework ROS2*.
- validar el desarrollo en escenarios reales, evaluando su rendimiento, robustez y capacidad de adaptación a diversas situaciones dinámicas.
- documentar detalladamente cada etapa del desarrollo para facilitar su comprensión y uso por otros investigadores o desarrolladores.
- participar en el concurso con la solución propuesta desarrollada.

2.2. Alcance

El alcance de este proyecto son las pruebas de percepción y control en un escenario desconocido del certamen *Sener-CEA's Bot Talent*. No se abordarán los desarrollos para la prueba de navegación y final.

Estas pruebas se describen en el Capítulo 4.

2.3. Beneficios y ODS

La realización de este proyecto aporta numerosos beneficios desde una perspectiva tanto técnica como social y económica.

Desde un punto de vista técnico, este trabajo introduce combinación de diferentes tecnologías para ofrecer un nuevo enfoque a la hora de crear aplicaciones inteligentes para robots móviles. El reconocimiento del entorno, mediante la identificación de símbolos, colores o formas junto con el control reactivo y preciso de los movimientos contribuye a aumentar la colección de desarrollos ya existentes con el objetivo de servir como inspiración para crear infraestructuras tecnológicas más avanzadas e innovadoras en la industria de la robótica móvil.

Además, se combina con *ROS2*, por lo que el trabajo sigue estándares abiertos que garantizan la accesibilidad a estos desarrollos, alineándose con el ODS 9 de Industria, Innovación e Infraestructura.

Por otro lado, dotar a los robots móviles de mayores capacidades autónomas como las trabajadas en este proyecto abre nuevas posibilidades en los campos de la asistencia a personas con discapacidad visual o cualquier persona que requiera de atención especializada. Asimismo, gran parte de la implementación del código se ha redactado en euskera con el fin de generar más contenido educativo y divulgativo en este idioma, alineándose con el ODS 4 de Educación de Calidad.

En esta misma línea, se ha hecho uso de simulaciones como herramienta clave para garantizar la posibilidad de desarrollar y experimentar con estas tecnologías incluso en contextos donde no se disponga de los recursos materiales necesarios, lo que refuerza su valor educativo, de igualdad y su contribución al ODS 4.

Por último, la implementación de estos algoritmos tiene el potencial de generar ahorros significativos y mejorar la eficiencia operativa en diversas industrias. El uso de visión artificial reduce la necesidad de intervención humana en el procesamiento y análisis de información visual, lo que optimiza el tiempo y los recursos empleados en tareas de monitorización o inspección. Consecuentemente, surgen nuevas demandas, oportunidades y ofertas laborales con diferente especialización. Estas últimas ventajas se enmarcan en el ODS 8 de Trabajo Decente y Crecimiento Económico.

3. Estado del arte

En este capítulo se estudian las diferentes soluciones tecnológicas desarrolladas por otros autores en lo referente a reconocimiento de colores y números mediante visión artificial, algoritmos de control y arquitecturas en *ROS2*.

3.1. Soluciones basadas en IA para percepción del entorno

Para el reconocimiento de colores la mayoría de trabajos se basan en la codificación de las imágenes. Un ejemplo de ello se puede leer en [23], donde se abordan diferentes métodos como *RGB* y *HSL*.

Para el reconocimiento de números diversas publicaciones utilizan técnicas de inteligencia artificial para procesar las imágenes e identificar los patrones necesarios. El trabajo [24] compara diferentes estrategias para identificar dígitos manuscritos. Entre estas técnicas se encuentran *MLP*, *SVM*, *Random Forest* y *Naive Bayes*. Otra herramienta para la clasificación de los números es el algoritmo *KNN* y las redes *CNN* como se describen en [25], siendo esta última junto con el *Deep Learning* la técnica más extendida [26].

Asimismo, en trabajos como [27] y [28] se aplica a las imágenes un procesamiento previo mediante el algoritmo *Canny*. Este algoritmo permite obtener y resaltar los bordes de los elementos que aparecen en la imagen para mejorar la tarea de identificación automática. La importancia de utilizar este procesamiento se observa en los resultados del trabajo investigador [29]. El resaltado generado mediante el algoritmo *Canny* se puede ver en la Ilustración 3.

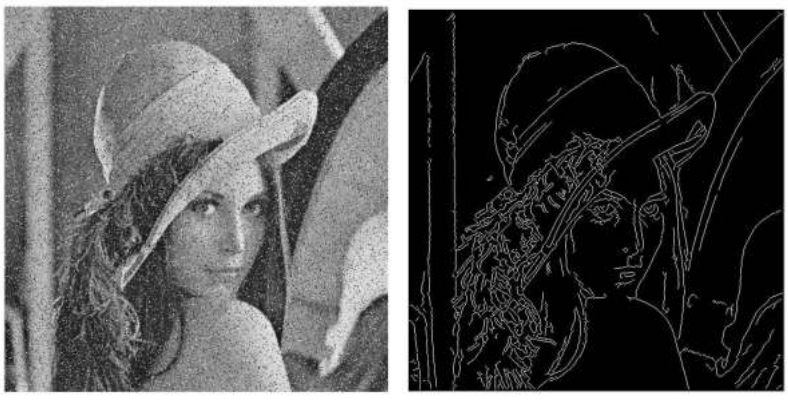


Ilustración 3: Ejemplo del resultado tras aplicar el algoritmo *Canny* [27].

3.1.1. Definición y conceptos básicos de las redes neuronales

Una red neuronal artificial es un modelo computacional y matemático inspirado en el funcionamiento del cerebro humano, diseñado para procesar información y aprender de los datos de manera similar a como lo hacen las neuronas biológicas [30]. Está compuesta por nodos o neuronas artificiales organizadas en capas de entrada, ocultas y de salida, donde cada nodo recibe señales, las procesa mediante funciones matemáticas y transmite el resultado a otros nodos de la siguiente capa, Ilustración 4. Estos modelos forman parte del ámbito de estudio del aprendizaje profundo o *Deep Learning*.

Son una de las herramientas más eficaces para el reconocimiento de formas y patrones, tarea fundamental en la identificación automática de números y caracteres, puesto que se basa en la capacidad de las neuronas para aprender, a partir de grandes volúmenes de ejemplos, las características distintivas que diferencian cada dígito o símbolo.

Durante el proceso de aprendizaje o entrenamiento, la red neuronal analiza múltiples representaciones de datos, en este caso de imágenes de números, y extrae patrones visuales relevantes como la forma y la curvatura de los trazos. La calidad del conjunto de datos empleado es transcendental para ajustar los pesos de manera correcta, es decir, los parámetros numéricos asociados a cada conexión entre neuronas que determinan la importancia o influencia que la neurona tiene sobre otra en el procesamiento de la información. Cuando una neurona recibe varias entradas, cada una de ellas se multiplica por su peso correspondiente antes de sumarse, y la suma ponderada resultante pasa por una función de activación que genera la salida de la neurona.

Una vez entrenada, la red es capaz de clasificar nuevas imágenes de números, incluso si presentan variaciones en tamaño, grosor o estilo, gracias a su habilidad para generalizar a partir de los patrones aprendidos.

Uno de los principales riesgos es el sobreajuste u *overfit*, es decir, aprenderse de memoria los ejemplos utilizados durante el entrenamiento y por tanto, no ser capaz de predecir correctamente imágenes no estudiadas. Para prevenir este fenómeno y conseguir que el modelo generalice adecuadamente la validación se realiza con un conjunto de datos no utilizado durante el entrenamiento y se aplican técnicas de regularización como *L1* y *L2*, *dropout* y *batch normalization*.

La normalización de *batch* consiste en calcular la media y la varianza de las salidas de una determinada capa para normalizar los datos y que sigan una distribución con media cero y varianza uno. Esta normalización genera activaciones uniformes de la capa y estabiliza el entrenamiento.

Por otro lado, la regularización por *dropout* apaga temporalmente algunas neuronas para que el modelo no genere dependencia hacia ninguna de ellas. Para ello, las salidas de las neuronas se establecen a cero con cierta probabilidad, y la tasa de *dropout* indica el porcentaje de la cantidad de neuronas a apagar.

Y finalmente, la regularización *L1* y *L2* añade penalizaciones a la función de pérdida para incentivar al modelo a aprender pesos más pequeños. La primera de ellas añade al error del modelo un término proporcional a la suma de los valores absolutos de los pesos, mientras que la segunda agrega un término proporcional a la suma de los cuadrados de los pesos. Esta regularización evita que algunas conexiones dominen excesivamente el modelo.

Estas redes se caracterizan por los hiperparámetros y los parámetros.

Por un lado, los hiperparámetros son las variables de configuración externas al modelo definidas por el usuario. El número y tipo de capas, las funciones de activación, el algoritmo de optimización y la función de coste, la estrategia de regularización, número de épocas o *epochs*, la tasa de aprendizaje o *learning rate* y el tamaño del *batch* son hiperparámeros. Estos valores no se aprender a partir de los datos, sino que están predefinidos y deben ser ajustados manualmente.

Las capas son de tres tipos principales: entrada, ocultas y salida. La capa de entrada recibe los datos y los propaga a las capas ocultas hasta llegar a la capa de salida que proporciona de manera interpretable el resultado de las operaciones de las capas ocultas. Las capas ocultas pueden ser convolucionales, de activación, de agrupamiento o *pooling* y de normalización.

Las funciones de activación introducen no linealidad a la red, lo que le permite descubrir relaciones complejas no necesariamente lineales. Las funciones utilizadas en este trabajo son *ReLU*, que devuelve el valor de entrada si es positivo y cero si es negativo, y *softmax*, que en problemas de clasificación multiclase convierte el vector de *logits*, valores de salida de una capa previo a la aplicación de la función de activación, en una distribución de probabilidades, donde cada elemento indica la probabilidad de pertenecer a una clase específica y cuya suma total es uno.

Los algoritmos de optimización se utilizan para implementar el ajuste de parámetros o de pesos. El objetivo es minimizar el error, por lo que el algoritmo de optimización utiliza funciones de pérdida como *loss function*, *backpropagation*, descenso del gradiente y *categorical cross-entropy* para ajustar automáticamente los pesos. Este último ha sido el utilizado en este trabajo y consiste en medir cuánto de lejos están las probabilidades predichas de la clase verdadera para penalizar fuertemente si el modelo asigna baja probabilidad a la clase correcta y recompensar si la probabilidad es alta. Los algoritmos de optimización más extendidos son *Adam*, *RMSprop* y *Momentum* [30].

Las *epochs* son el número de veces que el modelo recorre completamente todo el *dataset* de entrenamiento durante el aprendizaje. Una *epoch* corresponde a un ciclo completo en el que todas las instancias del subconjunto de entrenamiento son procesadas una vez por el modelo.

Los *batch* son pequeñas muestras del subconjunto de entrenamiento que se procesan juntas en una sola iteración antes de actualizar los parámetros del modelo. El subconjunto se divide en *batches* o lotes porque procesar todo el *dataset* de una vez es computacionalmente muy costoso. Es decir, si se tienen quinientos ejemplos de entrenamiento y un tamaño de lote de cien, el número de lotes es de cinco. Estos cinco lotes se procesan secuencialmente hasta haber procesado las quinientas instancias. Entre cada lote se ajustan y modifican los parámetros.

El *learning rate* es la tasa de convergencia del modelo, es decir, la rapidez con la que converge al objetivo o respuesta deseada. Cuanto mayor sea menor número de épocas será necesaria, pero puede no llegar a converger.

Por otro lado, los parámetros son los pesos de las conexiones entre las neuronas y son aprendidos automáticamente por la red. Estos pesos son valores numéricos que determinan la fuerza o la importancia de la conexión entre dos neuronas.

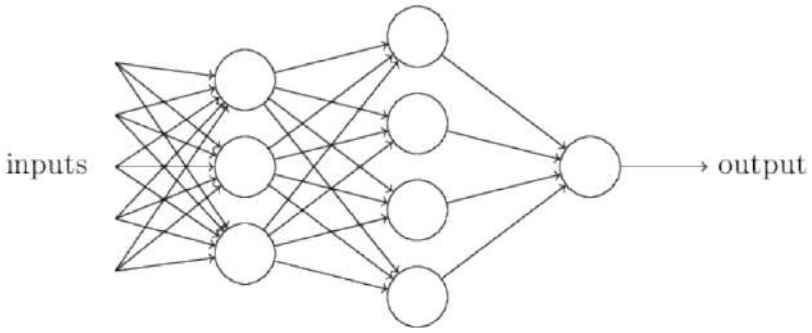


Ilustración 4: Representación visual de una red neuronal artificial [31].

3.2. Algoritmos de control

En la parte de control se identifican dos apartados: uno relativo a la estrategia utilizada y otro para el control de los movimientos.

En cuanto a las estrategias, la más destacada es *SLAM*. Este proceso genera un mapa del entorno y localiza el robot en dicho entorno mientras se desplaza. En el trabajo [32] se propone un modelado utilizando tanto cámaras como escáneres de dos dimensiones, mientras que en el trabajo [33] se plantea una alternativa de bajo costo para esta tarea.

Por otro lado, también existen estrategias que no involucran la generación de un mapa del entorno. Los trabajos [34], [35], [36] y [37] centran su desarrollo en la obtención de la distancia del robot a los objetos de su entorno o la identificación de un espacio libre para poder desplazarse sin colisionarse. Concretamente, en [34] y [35] se implementa en *ROS* el algoritmo de persecución pura reactiva definida en [38], y en [36] y [37] el algoritmo *ND* de la Ilustración 5, donde los obstáculos se representan con círculos negros y el hueco o la región sin obstáculos calculada según las equis objetivo sombreada en color gris.

En cuanto al control de los movimientos, en [39] y [40] se implementa un controlador PID en el lenguaje de programación *Python* para el seguimiento de líneas dibujadas sobre el terreno y corrección de desviaciones en la trayectoria para su aplicación en robótica educativa de Lego.

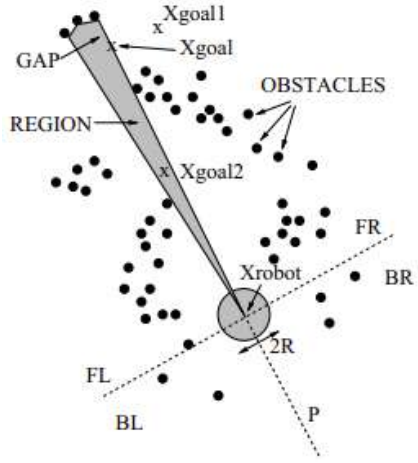


Ilustración 5: Representación gráfica del algoritmo *ND* [36].

3.3. Arquitecturas en *ROS2*

Mediante la arquitectura en *ROS2* se hace referencia a cómo se van a estructurar las tareas a desarrollar dentro el ecosistema de este *framework*.

En el trabajo [41] se proponen tareas ejecutables o nodos clasificados por niveles de funcionalidad. Cada nivel desempeña una labor diferente, como se observa en la Ilustración 6.

El nivel alto se encarga de la percepción, el nivel medio de generar patrones de movimiento y el nivel bajo del control en lazo cerrado de los actuadores.

Por otra parte, los desarrolladores del *JPL Open Source Rover Project* [16] proponen una arquitectura basada en niveles de abstracción. Este planteamiento consiste en separar los nodos o ejecutables de interacción con el *hardware*, mínima abstracción, de los que implementan los algoritmos, máxima abstracción.

Para intercambiar información entre ambos niveles se crea una capa *wrapper*, es decir, un ejecutable que "envuelve" a los componentes de bajo nivel para adaptar su interfaz y funcionamiento permitiendo que los módulos de alto nivel interactúen con ellos de manera sencilla e intuitiva.

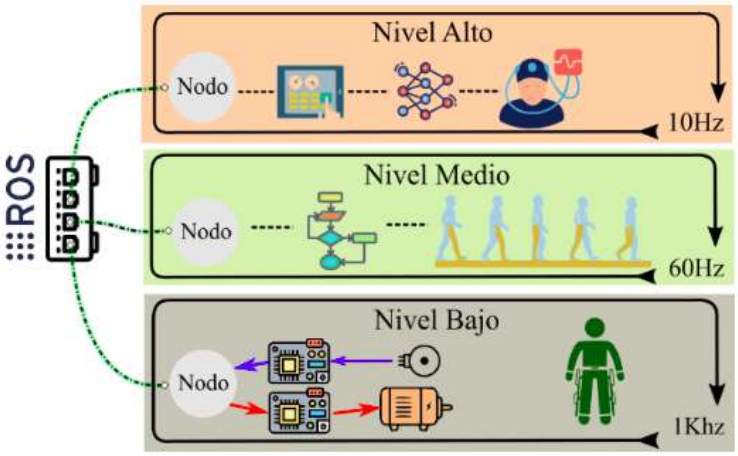


Ilustración 6: Arquitectura por niveles de tareas en *ROS2* [41].

4. Descripción de las especificaciones

A lo largo de este capítulo se detallan las especificaciones establecidas por la organización del concurso *Sener–CEA's Bot Talent 2025* [3, 4]. Junto con la descripción de las pruebas se explican los criterios de evaluación de la organización, las restricciones técnicas impuestas sobre las plataformas y los hitos de seguimiento o *checkpoints* que deberán ser superados para poder participar en la Final de Madrid del 6 de mayo de 2025.

Una de las principales normas es que la realización de cambios en las plataformas está permitida siempre y cuando estas no supongan una modificación sustancial de las prestaciones del *rover* y que aporten una ventaja competitiva. Por ejemplo, añadir más sensores está prohibido. Todos estos cambios tienen que ser notificados a la organización y aprobados previamente por ellos.

4.1. Establecimiento de *checkpoints*

Los *checkpoints* son pruebas intermedias fijadas por la organización para realizar el seguimiento de los equipos. La superación de estas pruebas garantiza que el equipo va adquiriendo los conocimientos y destrezas necesarias para la realización de todas las pruebas y por tanto, da acceso a participar en la final presencial del concurso.

Estas pruebas incluyen tareas básicas de percepción en el entorno, control de los actuadores y navegación y posicionamiento del *rover*.

Por ejemplo, las pruebas intermedias ligadas a este trabajo son el *checkpoint dos*, que exige mostrar cómo el robot es capaz de moverse y desplazarse, el *checkpoint tres*, para mostrar que el equipo es capaz de reconocer colores y números mediante visión artificial, y el *checkpoint seis*, para demostrar a la organización que el equipo es capaz de recoger información del sensor *LiDAR* e interpretarlos para generar una acción.

En resumen, estos exámenes prácticos parciales posibilitan desarrollar modularmente las capacidades necesarias para solucionar las pruebas del concurso.

4.2. Descripción de las pruebas

4.2.1. Prueba de percepción

La primera prueba a la que se debe enfrentar el *rover* en el concurso *Sener-CEA's Bot Talent* es de percepción. Concretamente, la percepción es el proceso mediante el que el robot obtiene información de su entorno utilizando diferentes sensores para obtener un reconocimiento del entorno.

La prueba consiste en identificar el color y el dígito de la caja cuadrada situada frente a la cámara del robot y realizar un giro sobre sí mismo. El sentido de giro y el número de vueltas total a dar vendrá dado por el color y número reconocido, respectivamente.

El color podrá ser azul, rojo o cualquier otro que se cataloga como "distractor", mientras que los números podrán ser del uno al nueve, incluyendo la categoría de "sin número". El color azul indica que el giro debe realizarse en sentido antihorario y el color rojo en sentido horario.

La caja tiene cuarenta centímetros de lado y se coloca a 1,5 metros del robot. En la Ilustración 7 se observa que el dígito tiene un tamaño aproximado de 20 centímetros de lado y la fuente es *Trebuchet MS*.

La puntuación de esta prueba consta de tres partes: 50 puntos para todos los colores bien identificados, 50 puntos para todos los números de cajas bien identificados y haber girado el número de vueltas correspondiente, y 10 puntos de penalización por cada equivocación con cajas distractoras.

En resumen, la puntuación se obtiene de la siguiente fórmula:

$$50 \cdot (\frac{n\ colores\ correctos}{n\ cajas}) + 50 \cdot (\frac{n\ números\ no\ distractores\ correctos}{n\ cajas\ no\ distractoras}) - 10 \cdot n\ distractores\ incorrectos$$

4.2.2. Prueba de control

La segunda prueba del concurso *Sener-CEA's Bot Talent* es de control.

El control es el proceso mediante el que el robot actúa sobre sus dispositivos. En este caso, el control es reactivo porque utiliza sensores para captar información del entorno y en base a esas entradas realiza una acción u otra.

La prueba consiste en recorrer un laberinto cerrado, con anchura fija y giros de 90 grados manteniéndose lo más centrado posible en el carril y sin chocarse con las paredes. Se puede observar un ejemplo en la Ilustración 7.

La anchura del laberinto es de 1,5 metros y se utilizan unas líneas de control, de anchura 1 metro, para evaluar lo centrado que se desplaza el *rover*.

La puntuación de esta prueba consta de dos partes: 50 puntos referentes a lo centrado que se desplaza el *rover*, dentro de las líneas de control (*LdC*), y 50 puntos destinados a evaluar la velocidad a la que el robot resuelve la prueba.

Para evaluar la rapidez se emplea el mejor tiempo de entre todos los participantes y se establece un máximo de 5 minutos, para el que se obtendría una puntuación de 0 puntos

en este apartado.

En resumen, la puntuación se obtiene de la siguiente fórmula:

$$[50 \cdot (1 - \frac{n\ LdC\ mal}{n\ LdC\ total})] + [50 \cdot (1 - \frac{t_{robot} - t_{mejor}}{t_{máx\ definido} - t_{mejor}})]$$

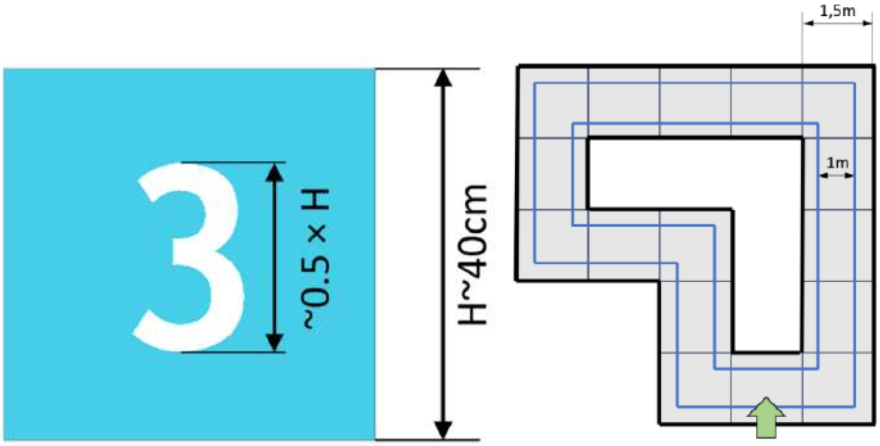


Ilustración 7: A la izquierda, ejemplo con las especificaciones de la caja cuadrada. A la derecha, ejemplo con las especificaciones del laberinto.

5. Componentes y montaje del *rover*

Para solucionar los retos del concurso se proporciona la plataforma *JPL Open Source Rover Project* con los materiales y componentes necesarios para su montaje: una placa *Raspberry Pi 4B* de 8GB de memoria *RAM* con tarjeta micro-*SD* y cables de conexión necesarios, una cámara *Orbbec Astra Pro Plus*, un sensor *LiDAR YDLIDAR X4*, un sensor *IMU Adafruit BNO055* y una batería *Lipo 4s 14.8V* con cargador.

Es una versión simplificada y de código abierto del *rover* de seis ruedas que el *JPL* utiliza para explorar la superficie del planeta Marte, *Perseverance* [42]. Está diseñado con piezas y materiales *COTS*, es decir, con piezas disponibles en proveedores comerciales que no requieren fabricación especializada o personalizada. Este proyecto fue creado para ofrecer una experiencia de enseñanza y aprendizaje en los ámbitos de ingeniería mecánica, *software*, electrónica y robótica.

5.1. Estructura mecánica

La estructura mecánica del *rover* se divide en tres subsistemas: el chasis, las patas con las ruedas y la caja de metacrilato. Estas tres partes principales se muestran en la Ilustración 8.

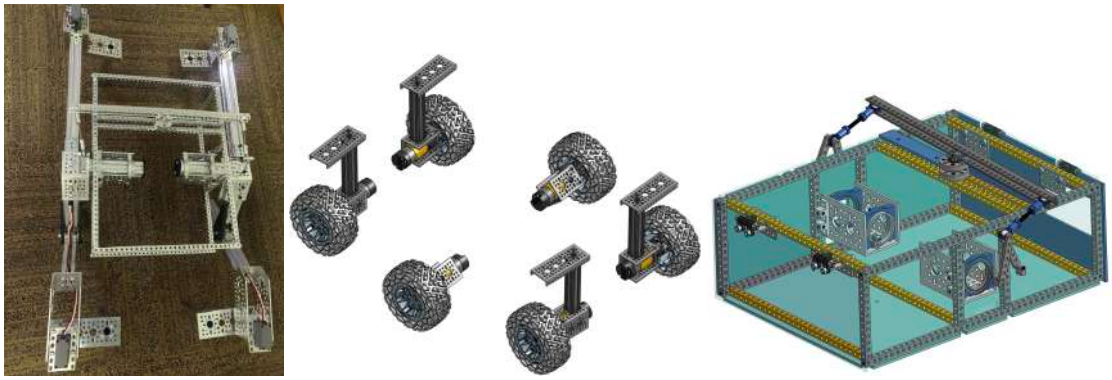


Ilustración 8: Subsistemas de la estructura mecánica del *rover*: chasis (izquierda), ruedas (centro) y caja de metacrilato donde se colocarán las placas de control (derecha).[16].

Tanto el chasis como las patas utilizan los perfiles de aluminio de montaje universal *goRAIL* [43] de la Ilustración 9.

Ilustración 9: Perfil de aluminio de montaje universal *goRAIL* [43].

El chasis incorpora una suspensión de tipo *rocker-bogie*. Esta suspensión le permite superar obstáculos de un 50 % mayor que el diámetro de sus ruedas motrices, motivo por el que es la suspensión más utilizada en el diseño y fabricación de estos robots [44].

Este sistema está compuesto por dos piezas, *rocker* y *boogie*, unidas por una articulación libre. La parte *rocker* se une al cuerpo del robot mediante una articulación diferencial, por tanto, cuando un *rocker* sube, el del otro lado baja.

En la Ilustración 10 se muestra el esquema de esta suspensión y en la Ilustración 11 el funcionamiento de la suspensión en el *rover* con el que se ha trabajado.

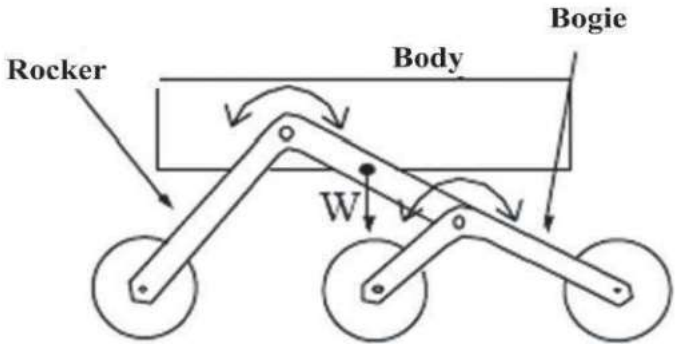


Ilustración 10: Esquema de suspensión *rocker-boogie* [45].

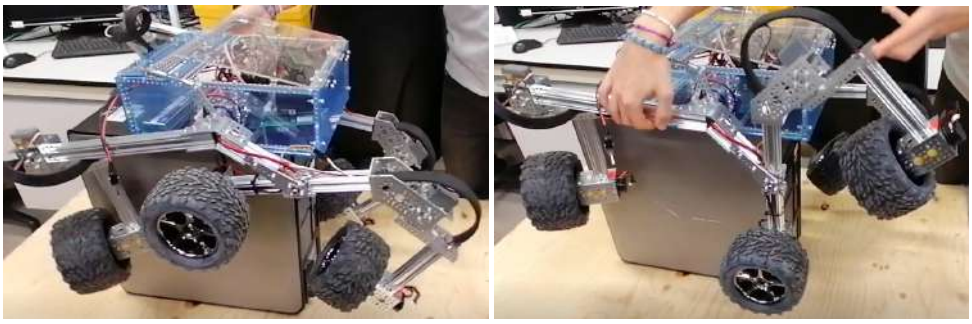


Ilustración 11: Movimiento del *rocker* y del *boogie*.

El *rover* tiene seis ruedas. Cuatro de ellas, las situadas en las esquinas o extremos, cuentan con un servomotor cada una para cambiar la dirección de la rueda, permitiendo al robot moverse en distintas direcciones.

5.2. Componentes electrónicos y sensores

5.2.1. Placa base o *PCB*

La placa base o *PCB* es el circuito electrónico impreso encargado de conectar y conducir la alimentación a todos los dispositivos. En este proyecto la *CPU* no viene integrada en la placa, sino que la *Raspberry Pi* se conecta modularmente mediante una *brain board* y un bus que le proporciona alimentación y comunicación con la placa y dispositivos conectados a la misma.

Algunos componentes como la *IMU* y los controladores de los motores se conectan a la placa, mientras que otros como la cámara y el *LiDAR* a los puertos *USB* de la *Raspberry Pi*.

La placa base puede verse en las ilustraciones 12 y 13 y se incluye en la plataforma *JPL Open Source Rover Project* [16].

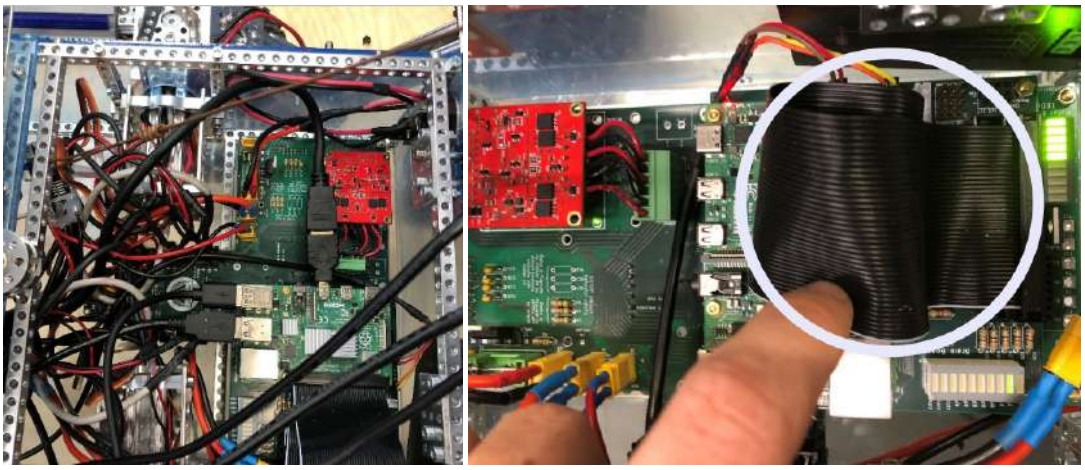


Ilustración 12: Placa base conectada y bus de conexión con la *Raspberry Pi*.

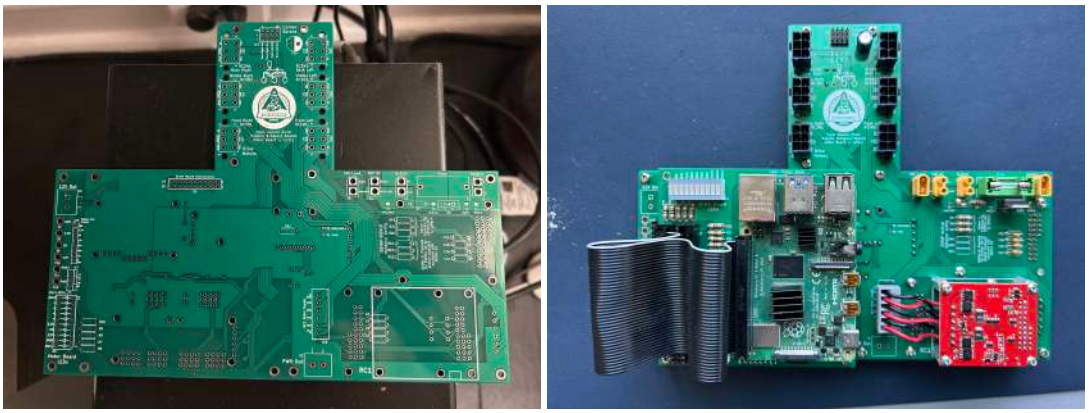


Ilustración 13: Placa base [16].

5.2.2. Motores y *encoders*

Los motores utilizados son los *5203 Series Yellow Jacket Planetary Gear Motor (26.9:1 Ratio, 24mm Length 8mm REX Shaft, 223 RPM, 3.3 - 5V Encoder)* [46]. En la Ilustración 14 se puede ver uno de los motores que da movilidad al *rover*.

Son motores de corriente continua (*DC*) con escobillas diseñados específicamente para aplicaciones de robótica que requieren de alta precisión y durabilidad. Las escobillas son piezas de material conductor, normalmente carbono, que hacen contacto con el conmutador del motor para permitir que la corriente fluya a través de las bobinas del rotor y genere el campo magnético necesario para que el motor gire. Una ventaja de estos motores es que son los más sencillos y los más económicos, aunque suelen necesitar mantenimiento por el desgaste de las escobillas.

Funciona a doce 12V (voltios) con una corriente sin carga de 250mA (miliamperios), tiene una fuerza de rotación, torque o par motor máximo de 38kg/cm (kilogramos por centímetro) y la relación de engranajes interna es de 26.9:1, es decir, el motor debe girar 26,9 veces para que el eje de salida dé una vuelta completa. Se controlan mediante una señal *PWM* (*Pulse Width Modulation*).

Por último, mencionar que incluyen *encoders* integrados. Estos sensores son de tipo magnético, funcionan tanto a 3,3V como a 5V y miden la posición y velocidad de rotación del eje del motor. Para ello, proporcionan pulsos eléctricos que permiten contar cuántas vueltas ha dado el eje. La documentación del fabricante indica que se generan 28 pulsos por revolución del motor interno, es decir, previo a la reducción de engranajes.

Los motores se incluyen en la plataforma *JPL Open Source Rover Project* [16].

5.2.3. Servomotores

Los sermotores empleados son los *2000 Series Dual Mode Servo (25-2, Torque)* [47] de la Ilustración 14. Estos dispositivos son motores eléctricos que, a diferencia de un motor normal, permiten controlar con precisión la posición, velocidad y aceleración de su eje mediante un sistema de retroalimentación. Esto posibilita ajustar el movimiento del motor para que coincida con una señal de control.

Este modelo es de corriente continua y tiene dos modos de operación: el modo posicional, que permite rotar hasta 300 grados sexagesimales con retroalimentación de posición mediante una señal *PWM*, y el modo de rotación continua, que gira de manera continuada controlando la velocidad también mediante una señal *PWM*. El modo posicional es el más habitual.

La relación de engranajes es de 300:1 y puede funcionar con diferentes voltajes dentro del rango 4,8V - 7,4V. En función del voltaje aplicado, su velocidad, torque y corriente varía. Estos datos se recogen en la Tabla 1.

Además, los servomotores ejercen fuerza para mantener la posición, por lo que cuanto mayor sea la carga a la que se someta el servomotor, mayor será la corriente que circule por él, pero nunca superior a la corriente máxima. Consecuentemente, el consumo es dependiente de la carga.

Los servomotores se incluyen en la plataforma *JPL Open Source Rover Project* [16].

	4,8V	6,0V	7,4V
Velocidad sin carga (RPM)	40	50	60
Torque máximo (kg/cm)	17,2	21,6	25,2
Corriente sin carga (mA)	150	160	200
Corriente máxima o de parada (mA)	2000	2500	3000

Tabla 1: Características del modelo de servomotor utilizado.

Ilustración 14: A la izquierda, el motor con *encoder* [46]. A la derecha, el servomotor [47].

5.2.4. Cámara *Orbbec Astra Pro Plus*

Es un dispositivo avanzado de captura en tres dimensiones que integra un sensor de imagen *RGB* y un sensor de profundidad basado en luz estructurada, permitiendo obtener tanto imágenes en color de alta resolución como mapas de profundidad precisos.

Captura simultáneamente la imagen de color y la distancia asociada a cada píxel. La resolución de las imágenes de color es de 1920x1080 píxeles y la del sensor de profundidad de 640x480 píxeles, ambas a 30 *FPS* (*Frames Per Second*). La precisión de la medición de la profundidad es de ± 3 mm a 1 metro y el rango máximo 8 metros.

El campo de visión de la imagen es de 66,1 grados en horizontal y 40,2 grados en vertical, mientras que el del sensor de profundidad es de 58,4 grados en horizontal y 45,5 grados en vertical.

Se conecta directamente a la *Raspberry Pi* mediante un único *USB* para la alimentación y los datos y utiliza una corriente de aproximadamente 200mA sin abrir la captura y 500mA durante la captura de imágenes.

Ilustración 15: Aspecto visual de la cámara *Orbbec Astra Pro Plus* [48].

5.2.5. *LiDAR YDLIDAR X4*

Por sus siglas en inglés, *Light Detection And Ranging*, es un dispositivo que obtiene información del entorno mediante la emisión de pulsos láser. Estos pulsos rebotan en los objetos del entorno y vuelven al sensor para calcular la distancia a cada objeto según el tiempo que tarda el pulso láser en regresar. Calcula las distancias dentro del campo de visión, que varía en función del modelo y fabricante.

En este proyecto se ha utilizado el *YDLIDAR X4* [49] de dos dimensiones de la Ilustración 16 con un campo de visión de 360 grados, siguiendo la regla de la mano izquierda [50] como se muestra también en la Ilustración 16.

Este sensor es capaz de obtener distancias entre 0,12 y 10 metros con una resolución angular de entre 0,43 y 0,86 grados, emite 5.000 pulsos láser por segundo y la frecuencia de escaneo o barrido completo de 360 grados sexagesimales es de entre 6-12Hz (hercios).

Se conecta directamente a la *Raspberry Pi* mediante dos *USB*, uno para la alimentación y otro para los datos. Su consumo habitual es de 350mA, pudiendo tener un pico de 500mA durante el arranque.

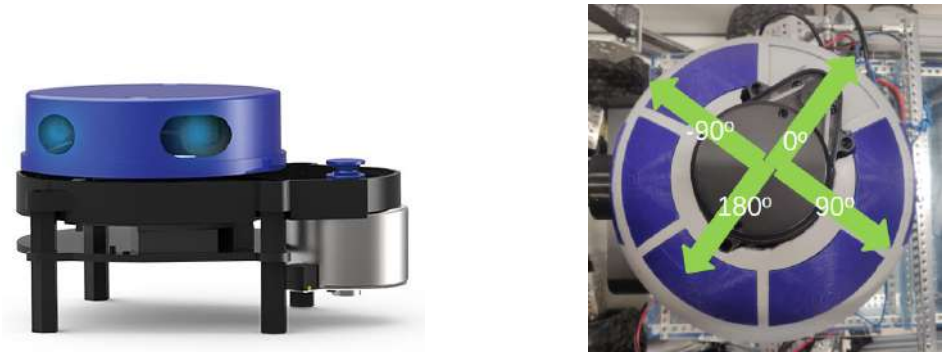


Ilustración 16: A la izquierda, el *LiDAR YDLIDAR X4* [49]. A la derecha, la regla de la mano izquierda sobre el dispositivo físico.

5.2.6. *IMU Adafruit BNO055*

La *IMU Adafruit BNO055* [51] (Unidad de Medición Inercial) es un sensor que combina un acelerómetro, un giroscopio y un magnetómetro en un solo dispositivo, permitiendo medir la orientación, la aceleración y la velocidad angular del robot. Este tipo de sensor es esencial para aplicaciones de navegación y control en vehículos autónomos y otros sistemas que requieren estimaciones precisas de orientación y movimiento en tres dimensiones.

Se compone de tres sensores integrados: acelerómetro, giroscopio y magnetómetro. Estos proporcionan la aceleración, velocidad angular y la detección de campos magnéticos en los tres ejes de rotación. La aceleración se mide en metros por segundo al cuadrado (m/s^2), la velocidad angular en radianes por segundo (rad/s) y el campo magnético en microteslas (μT).

La combinación de estos sensores permite fusionar diferentes datos para proporcionar una medición precisa y directa de la orientación mediante el uso de cuaterniones y

ángulos de Euler.

Su interfaz de comunicación es *I2C* [52], su dirección por defecto 0x28 y funciona tanto con un voltaje de 3,3V como de 5V con un consumo bajo de 12mA.

Se puede ver un ejemplo de este dispositivo en la Ilustración 17.

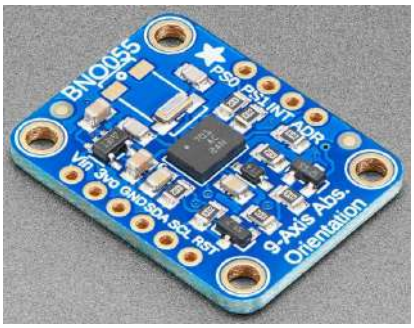


Ilustración 17: *IMU* [51].

5.2.7. Batería *Lipo* 4s 14.8V con cargador

Es una fuente de energía diseñada para alimentar dispositivos que requieren un voltaje estable y alto rendimiento, como drones, vehículos RC y otros dispositivos electrónicos. Se compone de cuatro celdas conectadas en serie proporcionando un voltaje total de 14,8V y una capacidad de 4Ah (amperio por hora).

Este modelo está diseñado para ofrecer una carga equilibrada, es decir, cargar la batería asegurando que todas las celdas se carguen de manera uniforme. Durante la carga, el cargador ajusta individualmente el voltaje de cada celda, evitando que alguna se sobrecargue o se quede con poca carga en comparación con las otras. Esto es crucial para garantizar la longevidad y seguridad de la batería, previniendo posibles daños, sobrecalentamientos o fallos prematuros. La carga equilibrada ayuda a maximizar la eficiencia y la duración de la batería, mejorando el rendimiento general del dispositivo.

En la Ilustración 18 puede verse el aspecto de la batería.

La batería se incluye en la plataforma *JPL Open Source Rover Project* [16].

Ilustración 18: A la izquierda, la batería [53]. A la derecha, la batería colocada en el *rover*.

5.3. Dispositivos de control y monitorización

En esta sección se describen los componentes dedicados al control y monitorización del *rover*.

5.3.1. Controladores de motores *Roboclaw 2x7A*

Los controladores *Roboclaw 2x7A Motor Controller* [54], ver Ilustración 19, son dispositivos inteligentes diseñados para gestionar dos motores de corriente continua con escobillas de manera simultánea. Cada controlador puede suministrar hasta 7,5A de corriente continua por canal, es decir, por motor, y soportar picos de hasta 15A por canal, lo que permite manejar motores de tamaño mediano con demandas variables de potencia.

Estos controladores admiten una amplia gama de tensiones de entrada, desde 6V hasta 34V de corriente continua, lo que les otorga una amplia flexibilidad para diferentes aplicaciones robóticas o de automatización. Ofrecen múltiples opciones de control, incluyendo *USB*, señales *PWM*, entradas analógicas, *TTL* serial y control por radio o RC. En este proyecto se controlan mediante puerto serie, concretamente con el protocolo *packet serial*, en el que los datos se envían en paquetes estructurados y no en *bytes* sueltos.

Los controladores se incluyen en la plataforma *JPL Open Source Rover Project* [16].

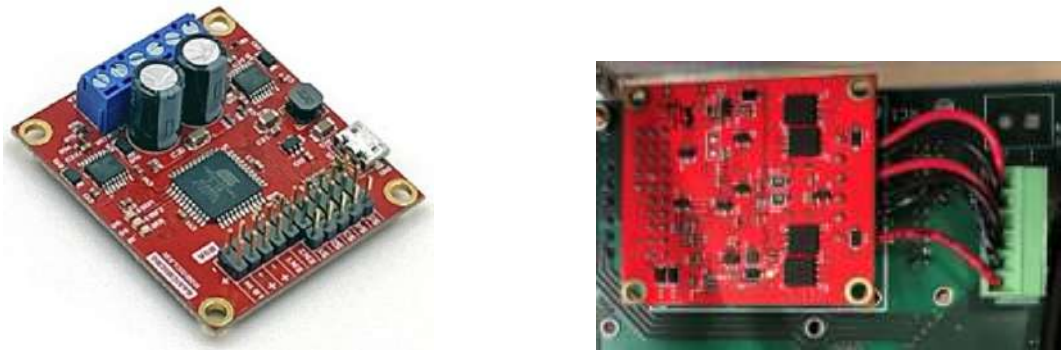


Ilustración 19: A la izquierda, *Roboclaw 2x7A Motor Controller* [54]. A la derecha, el controlador serie colocado en el robot.

5.3.2. *Raspberry Pi 4B* de 8GB de memoria *RAM*

Es una computadora de placa única (*SBC*) pequeña, compacta y de bajo costo, pero con un rendimiento potente y una amplia gama de capacidades de conectividad y expansión. Con 8GB (gigabytes) de memoria *RAM* es ideal para aplicaciones que requieren más memoria y proyectos de computación de alto rendimiento, como por ejemplo, el despliegue y ejecución de redes neuronales y aplicaciones robóticas en tiempo real. La arquitectura del procesador es *ARM*, por lo que será necesario utilizar librerías adaptadas o compatibles, y tiene conectividad *WiFi*, *Bluetooth* y *Ethernet*, además de disponer de dos puertos *USB2.0*, dos puertos *USB3.0* y salida *HDMI* (ver Ilustración 20).

El almacenamiento de los datos, así como el sistema operativo, se realiza mediante una tarjeta de memoria *SD* externa. En este proyecto se ha utilizado una tarjeta de 64GB.

La *Raspberry Pi* se incluye en la plataforma *JPL Open Source Rover Project* [16].

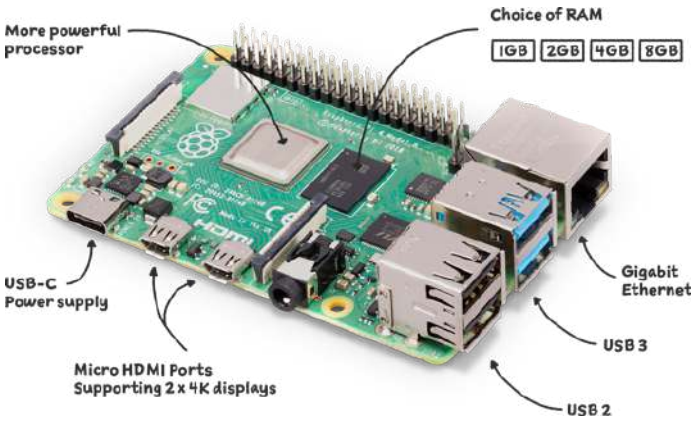


Ilustración 20: *Raspberry Pi 4B* [55].

5.3.3. Pantalla para *Raspberry Pi* de 400x1280 píxeles

Para mejorar el manejo de la información y la comunicación del *rover* se ha añadido una pantalla que permite utilizar el robot sin necesidad de estar conectado mediante otro ordenador. Este dispositivo no viene contemplado en las bases del concurso, por lo que es una mejora permitida que aporta un valor añadido.

Gracias a la pantalla se puede visualizar la cámara y observar qué está viendo el robot, siendo ideal para depurar el desarrollo de la prueba de percepción. Además, posibilita ejecutar los programas de control del *rover* desde la propia *Raspberry Pi* sin necesidad de conectar remotamente un equipo, reduciendo el impacto del riesgo de perder la comunicación inalámbrica, e introducir alguna animación o interfaz gráfica de los programas desarrollados para mejorar la estética y ofrecer información visual en tiempo real de los procesos en ejecución.

Se ha utilizado la *Pantalla Táctil Capacitiva Waveshare de 7,9 pulgadas* y 400x1200 píxeles [56] de la Ilustración 21.

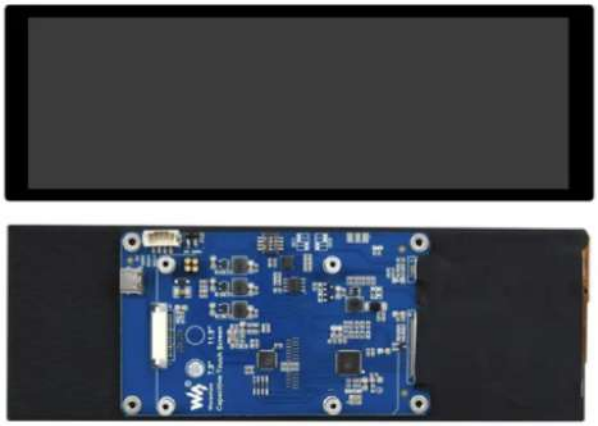


Ilustración 21: Arriba, aspecto exterior de la pantalla. Abajo, su circuito integrado. [56].

5.3.4. Interruptor

El *rover* dispone de un interruptor para suministrar o cortar la energía que recibe de la batería. Tiene tres posiciones: apagado, neutro y encendido. Para encender el robot hay que accionar el interruptor colocándolo en la posición de encendido. Este elemento se puede ver en la Ilustración 22 y se incluye en la plataforma *JPL Open Source Rover Project* [16].

5.3.5. *Display* de voltaje, corriente, potencia y energía

Este dispositivo es una pantalla *LCD* que muestra el voltaje de la electrónica del *rover*, la corriente, la potencia y la energía. Permite monitorizar estos valores para detectar posibles fallos de alimentación, prevenir cortocircuitos y planificar la carga de la batería. Se ha utilizado el modelo *KETOTEK Voltimetro Amperimetro Digital DC 6.5-100V 20A 12V* [57] de la Ilustración 22 incluído en la plataforma *JPL Open Source Rover Project* [16].

Ilustración 22: A la izquierda, el interruptor. A la derecha, el *display* de voltaje.

5.4. Dispositivos de comunicación

5.4.1. *Router 4G*

Para el establecimiento de la conexión y comunicación con el *rover* es muy recomendable disponer de una red local privada mediante un *router* dedicado exclusivamente para esta tarea. La organización no proporciona este dispositivo.

Se ha optado por elegir un *router* con conectividad *4G* para poder insertarle una tarjeta *SIM* y proporcionar acceso a internet desde la propia red local. En este sentido, el modelo utilizado es el *TL-MR110* [58] del fabricante *TP-Link* de la Ilustración 23.

Ilustración 23: *Router 4G* [58].

5.5. Montaje del *rover*

Todos estos componentes y dispositivos tienen que ser cableados y colocados en el robot.

En este proceso destaca la colocación de la *IMU*, del *LiDAR*, de la cámara y de la pantalla. Para ello, los compañeros del equipo BilboTiks diseñaron e imprimieron en 3D una cúpula inspirada en el personaje *R2-D2* de la franquicia *Star Wars* y un marco capaces de albergar estos dispositivos. Estos soportes se observan en las ilustraciones 24 y 25.

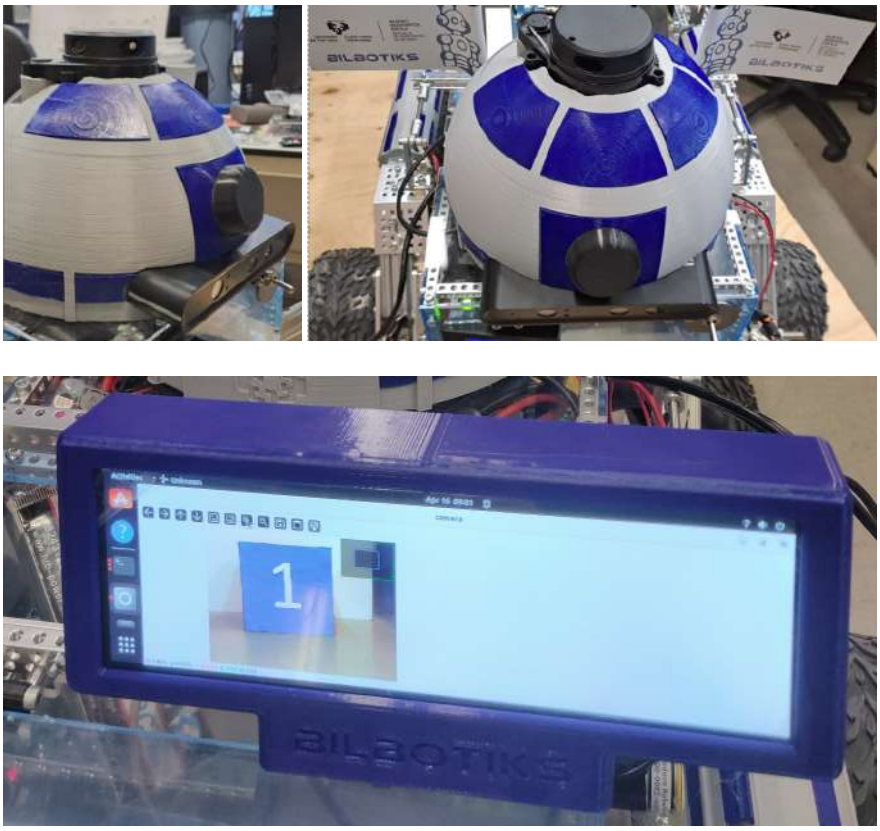


Ilustración 24: Arriba, la cúpula que alberga el *LiDAR* y la cámara. Abajo, el marco 3D de la pantalla.

Ilustración 25: *Rover* montado.

6. Análisis y diseño de la arquitectura

Además del desarrollo de *software*, este proyecto requiere gestionar diversos dispositivos o activos físicos como sensores y actuadores, por lo que se asemeja a un sistema de control de la industria de la automatización. Consecuentemente, además de diagramas de secuencia, se ha realizado un análisis basado en la Metodología para Ingeniería de Automatización o MeiA [59].

Por definición, MeiA es una metodología para el desarrollo de *software* de control de procesos industriales, incluyendo todas las fases desde el análisis y diseño, hasta la implementación y explotación del sistema. Permite conocer las operaciones y procedimientos que se tienen que realizar y estructurarlos dando lugar a los diagramas de secuencia.

En las siguientes secciones se presenta la captura de requisitos mediante los diferentes modelos de MeiA, los diagramas de secuencia y el diseño de la arquitectura.

6.1. Captura de operaciones

Las operaciones son las tareas a realizar para superar las pruebas. Se han identificado mediante el modelo de proceso de MeiA [59], ver Ilustración 26, que recoge estas tareas por diferentes subprocesos, fases o módulos. Cada prueba será un módulo diferente.

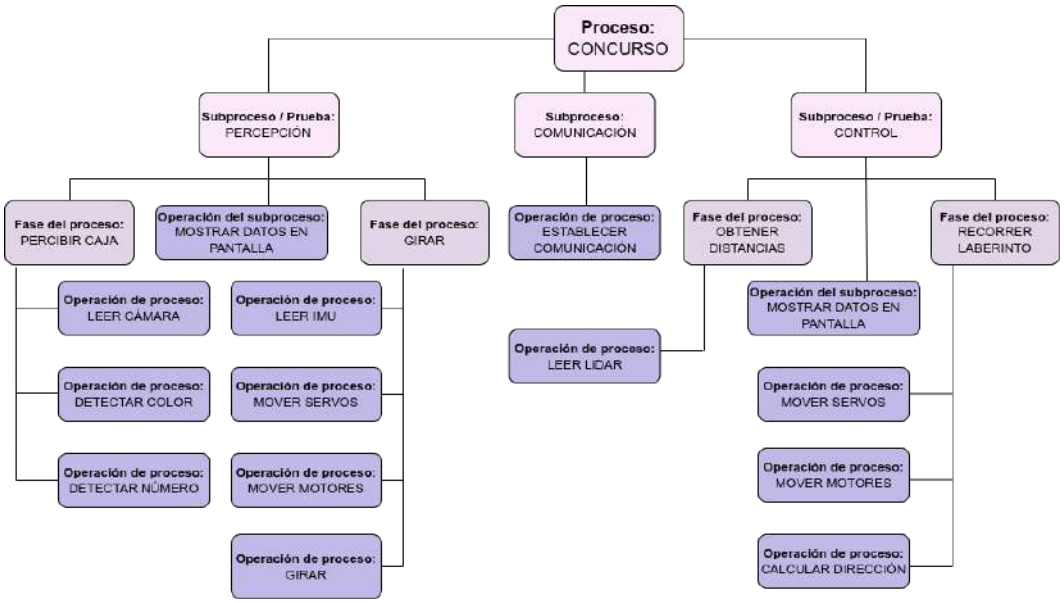


Ilustración 26: Modelo de proceso.

6.2. Captura de procedimientos

Se ha continuado con la Metodología MeiA [59] para determinar los procedimientos a implementar en la solución. Los procedimientos son las funciones a desarrollar que ejecutan las operaciones o tareas identificadas. Se recogen en el modelo de procedimientos de la Ilustración 27.

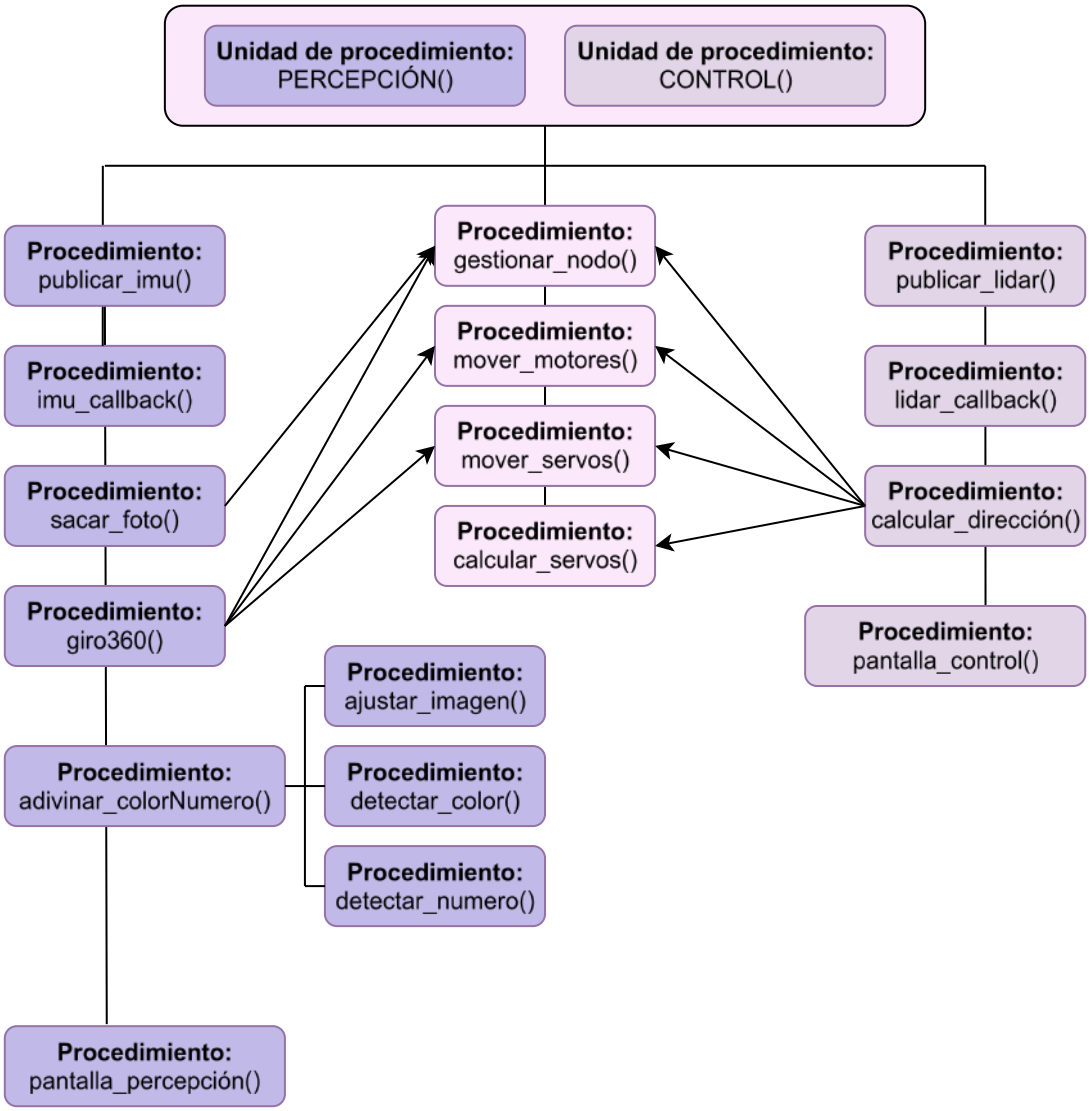


Ilustración 27: Modelo de procedimientos.

6.3. Especificación de procedimientos

A continuación se recogen en tablas los diseños de los procedimientos para superar las pruebas de percepción y de control. También incluyen las precondiciones que tienen que cumplir y las postcondiciones.

6.3.1. Prueba de percepción

Procedim.	Precondición	Postcondición
publicar_imu()	La *IMU* está conectada y preparada para publicar los datos.	Las mediciones del sensor se envían al tópico *imu_datuak*.
imu_callback()	La *IMU* está publicando datos en el tópico *imu_datuak*.	Las mediciones del sensor se almacenan en una variable de la clase.
sacar_foto()	La cámara está conectada y su nodo gestor activado para poder recibir la solicitud del servicio.	Devuelve a través de la respuesta del servicio la imagen obtenida.
giro360()	Recibe como parámetros el color y número identificados.	Gira tantas vueltas en el sentido correcto como número y color detectados.
ajustar_imagen()	Recibe la fotografía realizada.	Devuelve la fotografía recortada y ampliada.
detectar_color()	Recibe la fotografía recortada y ampliada.	Devuelve el color identificado y la región de color de la imagen.
detectar_numero()	Recibe la región de color identificada.	Devuelve el número detectado.
adivinar_colNum()	Todos los procesos o nodos asociados están lanzados y los dispositivos conectados.	Devuelve el color y número identificado.
pantalla_percep()	-	Muestra por pantalla en tiempo real el valor de la *IMU*, la imagen capturada, el número de vueltas totales a dar y el color identificado.

Tabla 2: Especificación de los procedimientos de la prueba de percepción.

6.3.2. Prueba de control

Procedim.	Precondición	Postcondición
publicar_lidar()	El *LiDAR* está conectado y preparado para publicar los datos.	Las mediciones del sensor se envían al tópico *lidar_datuak*.
lidar_callback()	El *LiDAR* está publicando datos en el tópico *lidar_datuak*.	Las mediciones del sensor se almacenan en una variable de la clase.
calcular_direccion()	Se están recibiendo las mediciones del sensor *LiDAR*.	Calcula y aplica la dirección de corrección necesaria para mantener la trayectoria.
pantalla_control()	-	Muestra por pantalla en tiempo real las distancias izquierda, frontal y derecha.

Tabla 3: Especificación de los procedimientos de la prueba de control.

6.3.3. Procedimientos compartidos por ambas pruebas

Procedim.	Precondición	Postcondición
gestionar_nodo()	Recibe el nombre del nodo con ciclo de vida (ver apartado 7.2.3) y la transición a aplicar.	El nodo objetivo con ciclo de vida cambia de estado.
mover_motores()	Recibe un vector de seis números enteros, con magnitud, es decir, positivo o negativo, para indicar la velocidad y dirección a la que accionar cada motor.	Se ha publicado en el tópico *motorrak_mugitu* la dirección y velocidad especificada de cada motor.
mover_servos()	Recibe un vector de cuatro números enteros para indicar el ángulo sexagesimal al que colocar cada servomotor.	Se ha publicado en el tópico *servoak_mugitu* el ángulo especificado de cada servomotor.
calcular_servos()	Recibe un ángulo sexagesimal de giro, entre -90 y 90.	Devuelve la posición angular sexagesimal de cada servomotor tras aplicar una interpolación.

Tabla 4: Especificación de los procedimientos compartidos por ambas pruebas.

6.4. Modelo físico

El modelo físico agrupa los equipos físicos según las operaciones que realizan y por tanto, permite organizar de manera esquemática los dispositivos y elementos físicos con sus señales de control. Todas las pruebas las realizará la unidad del *rover*, mientras que la unidad de desarrollo se encargará de comunicarse y mandar las órdenes al robot.

Los módulos de equipo son conjuntos que aglutinan a diferentes dispositivos físicos conocidos como módulos de control. En el *rover* se diferencian tres módulos de equipo: los componentes electrónicos y sensores, los dispositivos de control y la estructura mecánica. Estos módulos de equipos se dividen, a su vez, en diferentes módulos de control. En la unidad de desarrollo y comunicación se encuentran directamente los módulos de control del *router* y del ordenador de trabajo.

El modelo físico se presenta en la Ilustración 28 junto con las señales de control más relevantes.

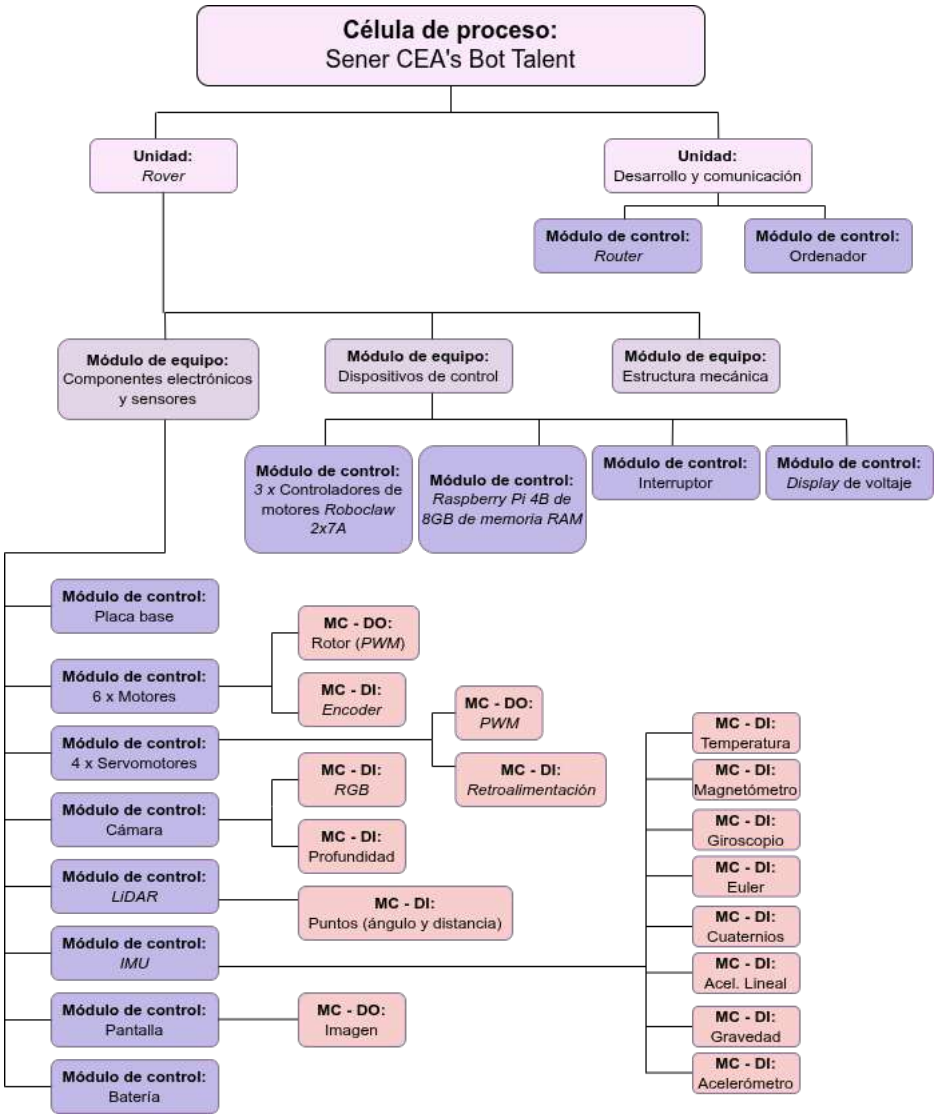


Ilustración 28: Modelo físico.

6.5. Diseño de la arquitectura

Siguiendo el análisis y las arquitecturas planteadas en otros proyectos recogidos en el Capítulo 3, se ha diseñado una arquitectura que divide el control del *hardware* de la implementación de los algoritmos en ejecutables independientes. De esta manera, se establecen tres niveles: alto, medio, bajo. Esto permite mantener organizada la implementación separando la interacción con el *hardware* de la lógica de los algoritmos de más alto nivel.

El nivel alto contendrá toda la lógica necesaria para invocar a las funciones del nivel medio y bajo. Para comenzar una prueba es necesario iniciar varios procesos asociados a los dispositivos físicos y esta capa será la encargada de delegar ese arranque, de forma que el usuario podrá arrancar toda la prueba mediante una única llamada con los *launchers* de *ROS2*. Además, este nivel será el encargado de recibir el estado del robot en cada prueba y gestionarlo.

El nivel medio serán los controladores del nivel bajo, o punto intermedio de comunicación entre el nivel alto y bajo. A las funciones de este nivel se las conoce como *wrapper* o envolventes porque procesan los datos obtenidos del nivel bajo para añadir información o generar una acción que es comunicada al nivel alto.

Por último, el nivel bajo contendrá los procesos asociados al *hardware*, su recogida de datos, limpieza y ordenación, en caso de ser necesario, y envío de los datos.

Estos tres niveles se comunicarán a través de *ROS2* mediante tópicos, servicios y acciones y se recogen de forma esquemática en la Ilustración 29.

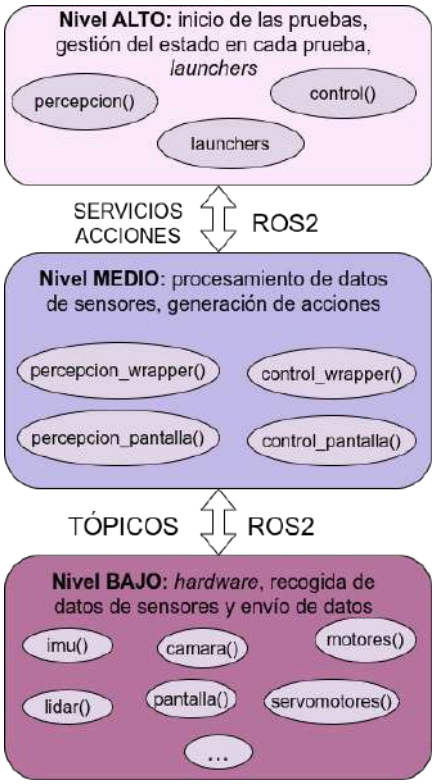


Ilustración 29: Arquitectura propuesta diseñada.

6.6. Diagramas de secuencia

Los diagramas de secuencia recogen visualmente las llamadas de las funciones a realizar y el orden correcto para la ejecución del código desarrollado. Al representar visualmente el flujo de mensajes y eventos, los diagramas de secuencia ayudan a identificar posibles fallos en la lógica del programa o llamadas incorrectas, contribuyendo así a una mayor comprensión y mantenimiento del *software*.

Las cajas de color rosa son los nodos o ejecutables de alto nivel, las lila de nivel medio y las púrpura de nivel bajo.

6.6.1. Prueba de percepción

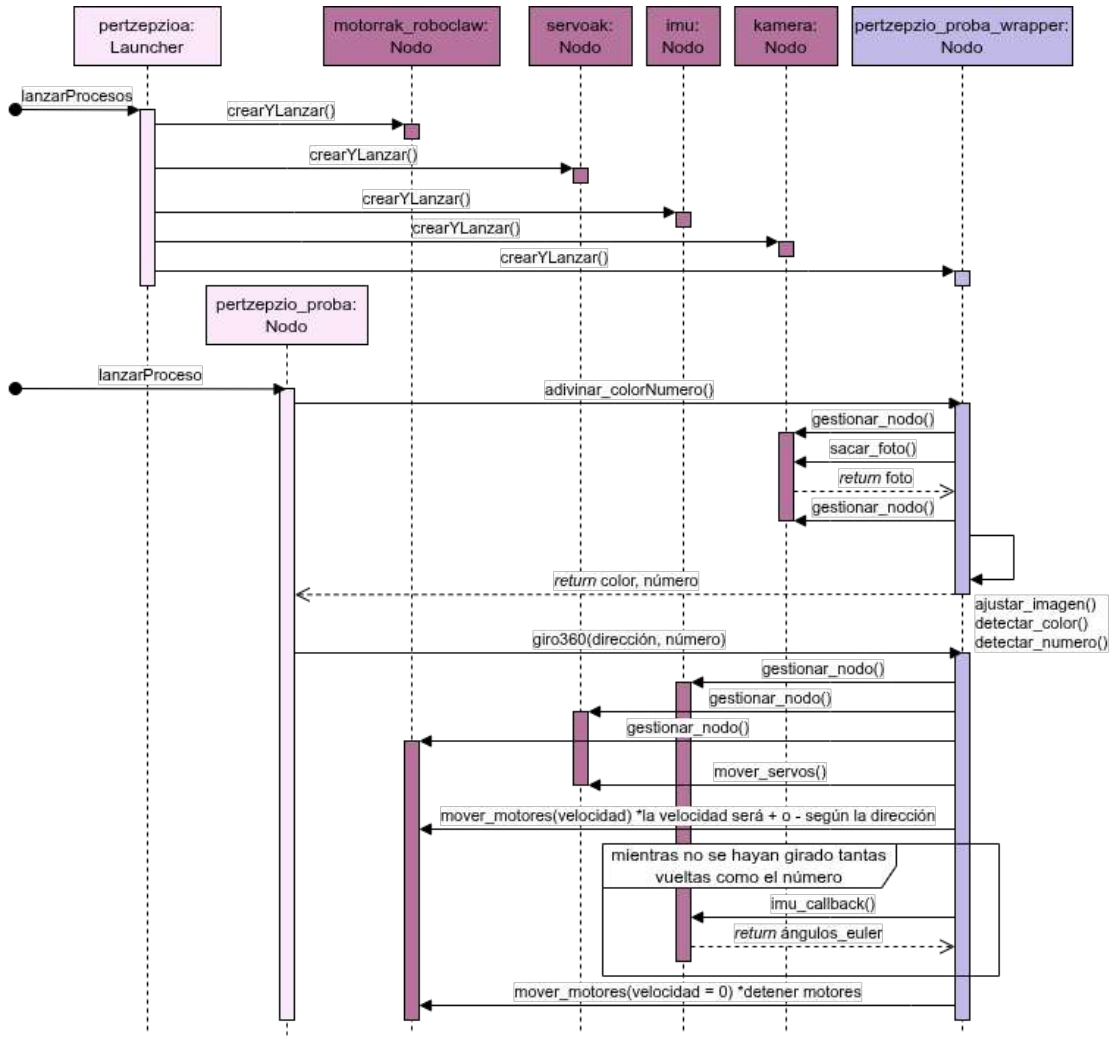


Ilustración 30: Diagrama de secuencia de la prueba de percepción.

6.6.2. Prueba de control

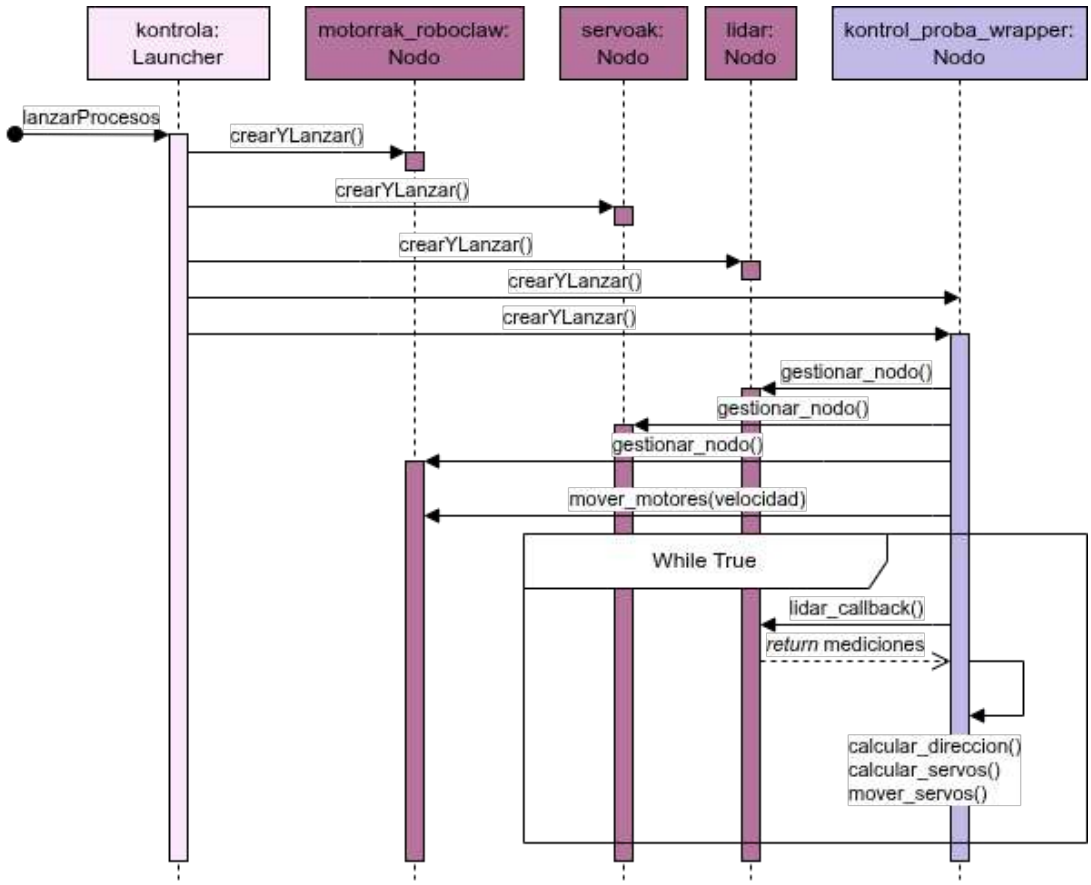


Ilustración 31: Diagrama de secuencia de la prueba de control.

7. Desarrollo de la solución

En este capítulo se aborda el *framework ROS2* junto con las interfaces de comunicación que utiliza y la implementación de la arquitectura diseñada, la programación del movimiento del robot y el desarrollo de los algoritmos, detallando tanto los fundamentos teóricos como la codificación y adaptación a la especificaciones del proyecto. Se ha utilizado el lenguaje de programación *Python* dentro de *ROS2* y grados sexagesimales.

También se incluyen las interfaces gráficas desarrolladas y las simulaciones realizadas con gemelos funcionales y simuladores.

Todo el código se incluye en el Anexo II.

7.1. *ROS2*

7.1.1. Definición y características

ROS2 [60] es un *framework* de código abierto, es decir, un conjunto de herramientas y bibliotecas, diseñado para facilitar el desarrollo de aplicaciones en el ámbito de la robótica, especialmente en la robótica móvil.

Su desarrollo es liderado y respaldado por la *Open Source Robotics Foundation*, lo que garantiza un soporte profesional, una evolución constante y una comunidad activa a nivel mundial.

El uso de *ROS2* aporta numerosas ventajas al desarrollo de este tipo de proyectos ya que utiliza estándares abiertos y ampliamente adoptados en la industria como *Data Distribution Service (DDS)* [61] y cuenta con una gran comunidad de desarrolladores.

Las versiones de *ROS2* son conocidas como distribuciones y están estrechamente relacionadas con una versión específica de *Ubuntu*. Por ello, cada distribución se libera poco tiempo después de la publicación de una nueva versión de *Ubuntu*. Las distribuciones asociadas a versiones *LTS*, son también distribuciones *LTS* con soporte a largo plazo de cinco años.

En el contexto de este proyecto se utiliza la distribución *Humble Hawksbill LTS* [62] asociada a *Ubuntu 22.04 LTS*, publicada en mayo de 2022 y con soporte hasta mayo de 2027. En la Ilustración 32 se muestra el logo de *ROS2 Humble Hawksbill*.

Ilustración 32: *ROS2 Humble Hawksbill.*

Actualmente, la distribución más reciente es *Jazzy Jalisco LTS* [63] asociada a *Ubuntu 24.04 LTS*, publicada en mayo de 2024 y con soporte hasta mayo de 2029. No obstante, se ha decidido utilizar una distribución anterior debido a su madurez, soporte y compatibilidad.

7.1.2. Nodos, tópicos, servicios, acciones y mensajes

En *ROS2* un nodo es un proceso o programa independiente encargado de realizar una tarea específica dentro de un sistema de robótica. En otras palabras, es la unidad básica lógica de ejecución.

Estos nodos comparten información entre ellos mediante tópicos, servicios y acciones, construidos sobre *DDS* [61].

Un tópico es un canal de comunicación que permite el intercambio de mensajes entre nodos de manera asíncrona, siguiendo el modelo publicador-suscriptor. Los nodos pueden publicar mensajes en un tópico o suscribirse a él para recibir los datos enviados por otros nodos. Cada tópico está asociado a un tipo de mensaje específico, lo que garantiza que la información intercambiada sea compatible entre los nodos conectados al mismo tópico. Por ejemplo, los tópicos se utilizan para recibir información en tiempo real de los diferentes sensores del robot.

Un servicio es un mecanismo de comunicación síncrona basado en el modelo cliente-servidor que permite que un nodo cliente envíe una petición a otro nodo servidor. El nodo servidor procesa la solicitud y devuelve una respuesta al nodo cliente. Son útiles para operaciones puntuales que requieren una respuesta inmediata, como pedir el estado de un sensor o sacar una foto. La comunicación mediante servicios involucra un mensaje de petición y otro de respuesta.

En la Ilustración 33 se puede ver un esquema del funcionamiento de los tópicos y servicios.

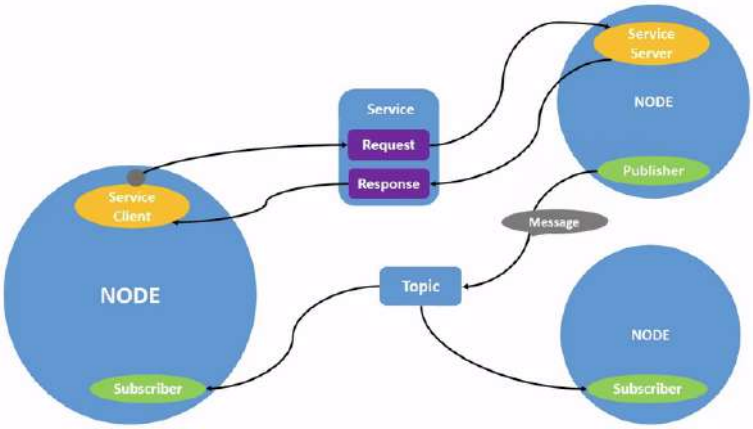


Ilustración 33: Esquema de tópicos y servicios en *ROS2* [62].

Una acción es una interfaz de comunicación asíncrona basada en tópicos y servicios diseñada para desempeñar operaciones largas y prolongadas en el tiempo que necesitan seguimiento. Para ejecutar una acción un nodo cliente solicita a un nodo servidor la ejecución, recibe retroalimentación periódica sobre el progreso a través de un tópico y tiene la opción de cancelar la tarea antes de que finalice. Son ideales para tareas como mover un robot a una posición determinada, donde es necesario monitorizar el avance y poder interrumpir la ejecución si es necesario. La comunicación de una acción involucra mensajes de objetivo, retroalimentación y resultado final.

En la Ilustración 34 se puede ver un esquema del funcionamiento de las acciones.

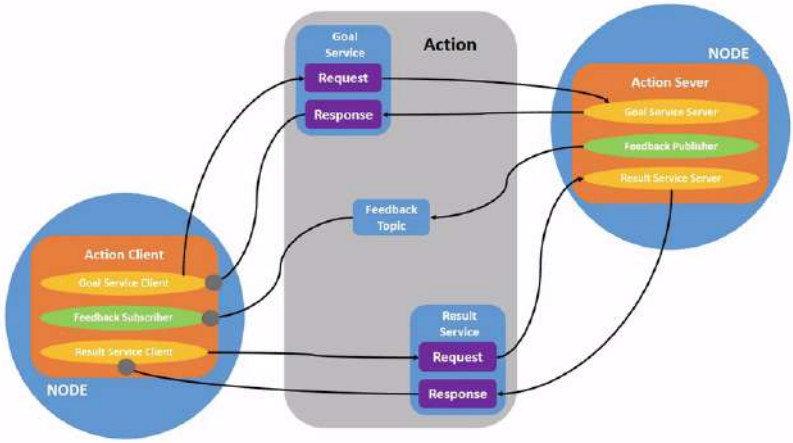


Ilustración 34: Esquema de acciones en *ROS2* [62].

Los mensajes son estructuras de datos que se componen de diferentes campos, cada uno con su nombre y tipo. Existe una serie de mensajes predeterminados, por ejemplo, *geometry_msgs/msg/Vector3* y *sensor_msgs/msg/LaserScan*. El primero tiene los campos *x, y, z* para representar coordenadas de tipo *float64* y el segundo tiene nueve campos capaces de caracterizar los datos de un sensor *LiDAR*. Ambos mensajes se pueden ver en los recuadros de Código 7.1 y 7.2, respectivamente. Los tipos de los campos pueden básicos, como enteros o binarios, o de tipos de mensajes, es decir, un mensaje B puede tener dos campos de tipo "mensaje A".

También se pueden definir mensajes personalizados para adaptarse a las necesidades de la aplicación.

```
# This represents a vector in free space.
# It is only meant to represent a direction. Therefore, it does not
# make sense to apply a translation to it (e.g., when applying a
# generic rigid transformation to a Vector3, tf2 will only apply the
# rotation). If you want your data to be translatable too, use the
# geometry_msgs/Point message instead.

float64 x
float64 y
float64 z
```

Código 7.1: Definición del mensaje *geometry_msgs/msg/Vector3*.

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a
    sonar
# array), please find or create a different message, since
    applications
# will make fairly laser-specific assumptions about this data

Header header            # timestamp in the header is the acquisition
    time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured
    around
                         # the positive Z axis (counterclockwise, if
    Z is up)
                         # with zero angle being forward along the x
    axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if
    your scanner
                         # is moving, this will be used in
    interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min
    or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].
    If your
                         # device does not provide intensities,
    please leave
                         # the array empty.
```

Código 7.2: Definición del mensaje *sensor_msgs/msg/LaserScan*.

En el caso de los servicios, los mensajes incluyen cabecera de solicitud y cabecera de respuesta, separados por una línea de tres guiones, y en las acciones los mensajes incluyen los campos de objetivo o solicitud, resultado y retroalimentación. En el Código 7.3 y 7.4 se presentan como ejemplo la definición del mensaje del servicio *turtlesim/srv/Spawn* y de la acción *test_msgs/action/Fibonacci Action*.

```
#solicitud
float32 x
float32 y
float32 theta
string name
---
#respuesta
string name
```

Código 7.3: Ejemplo de definición del mensaje de un servicio.

```
#objetivo o solicitud
int32 order
---
#resultado
int32[] sequence
---
#retroalimentación
int32[] sequence
```

Código 7.4: Ejemplo de definición del mensaje de una acción.

7.2. Paquetes, nodos y mensajes desarrollados en *ROS2*

Un paquete en *ROS2* es la unidad principal de organización del *software*. Se trata de una carpeta que contiene todos los archivos necesarios para desarrollar y ejecutar una funcionalidad específica. Dentro de un paquete se pueden encontrar los nodos y los archivos de configuración, mensajes personalizados, compilación, lanzamiento y parámetros.

El paquete es el elemento más atómico que se puede construir, compartir y reutilizar. Facilitan la modularidad y la escalabilidad de los proyectos. Cualquier ejecutable de cualquier paquete puede ser llamado desde otro paquete.

Se han desarrollado tres paquetes diferentes:

- ***bilbotiks_controller***: contiene los nodos responsables del control del robot y de las pruebas a solucionar.
- ***bilbotiks_pantaila***: contiene los nodos responsables de la gestión de la interfaz gráfica.
- ***bilbotiks_interfazeak***: contiene la definición de los mensajes personalizados para los tópicos, servicios y acciones.

7.2.1. Mensajes personalizados desarrollados

Se han definido seis mensajes generales, dos mensajes para servicios y un mensaje para una acción.

A continuación, se presenta la definición de cada mensaje junto con una breve descripción.

7.2.1.1. MotorrakMugitu (mover motores)

Este mensaje contiene un vector de enteros con la velocidad de cada uno de los seis motores.

```
# Motor bakoitzaren abiadura
# Abiadura minimoa = -127
# Abiadura maximoa = 127
# Motorrak gelditzeko abiadura = 0

# Robotan jarritako motor zenbakiak arrayaren elementu bakoitzarekiko
    erlazionatzen da:
# 1.go motorra: 0 posizioa
# 2. motorra: 1.go posizioa
# 3. motorra: 2. posizioa
# ...

int8[6] abiadurak
```

Código 7.5: Mensaje personalizado para mover los motores.

7.2.1.2. ServoakMugitu (mover servomotores)

Este mensaje contiene un vector de enteros con el ángulo en el que debe colocarse cada uno de los cuatro servomotores.

```
# Robotan jarritako motor zenbakiak arrayaren elementu bakoitzarekiko
    erlazionatzen da:
# 1.go servo: 0 posizioa
# 2. servo: 1.go posizioa
# 5. servo: 3. posizioa
# ...

int16[4] angeluak
```

Código 7.6: Mensaje personalizado para mover los servomotores.

7.2.1.3. *IMU*

El mensaje predefinido *sensor_msgs/msg/Imu* [64] no contiene campos para todos los valores proporcionados por la *IMU* utilizada en este proyecto, por lo que se ha definido un mensaje que sí contiene todos los campos que proporciona el dispositivo.

```
int8 tenperatura
geometry_msgs/Vector3 azelerometroa
geometry_msgs/Vector3 magnetometroa
geometry_msgs/Vector3 giroskopioa
geometry_msgs/Vector3 euler_angeluak
geometry_msgs/Quaternion kuaternioak
geometry_msgs/Vector3 azelerazio_lineala
geometry_msgs/Vector3 grabitatea
```

Código 7.7: Mensaje personalizado para la *IMU*.

7.2.1.4. Aurpegia (interfaz gráfica de la cara)

Este mensaje permite a la interfaz gráfica conocer qué animación tiene que visualizar, ya que indica el lado al que tienen que mirar los ojos de la cara.

```
# 0 = begiak irekitak
# 1 = begiak ezkerretara
# 2 = begiak eskuinara

int8 begi_norabidea
```

Código 7.8: Mensaje personalizado para la interfaz gráfica de la cara.

7.2.1.5. Pertzepzioa (interfaz gráfica de percepción)

Este mensaje contiene la información necesaria por la interfaz gráfica de la prueba de percepción.

```
float64 imu
int8 emandako_birak
int8 bira_totalak
bool norabidea
```

Código 7.9: Mensaje personalizado para la interfaz gráfica de la prueba de percepción.

7.2.1.6. Kontrola (interfaz gráfica de control)

Este mensaje contiene la información necesaria por la interfaz gráfica de la prueba de control.

```
float32 ezker_distantzia
float32 aurreko_distantzia
float32 eskuin_distantzia
```

Código 7.10: Mensaje personalizado para la interfaz gráfica de la prueba de control.

7.2.1.7. Servicio Argazkia (sacar foto)

Este mensaje contiene una solicitud vacía y una respuesta en formato de imagen.

```
# Argazkia.srv

# Request (vacío)
---

# Response
sensor_msgs/Image image
```

Código 7.11: Mensaje personalizado para el servicio de sacar una foto.

7.2.1.8. Servicio KoloreaZenbakia (adivinar color y número)

Este mensaje contiene una solicitud vacía y una respuesta con dos campos: color y número identificados.

```
# PertzepzioProba.KoloreaZenbakia.srv

# Ez dira daturik behar eskaeran
---
int8 kolorea   # Primer número de salida
int8 zenbakia   # Segundo número de salida
```

Código 7.12: Mensaje personalizado para el servicio de adivinar el color y el número.

7.2.1.9. Acción Bira360 (dar vueltas según el color y número reconocidos)

Este mensaje contiene una solicitud con los campos de color y número, una respuesta con un campo binario para saber si ha finalizado correctamente, el ángulo de inicio, el ángulo final y el número de vueltas que ha dado el robot, y dos campos de retroalimentación que indican el ángulo actual mientras el *rover* gira y el número de vueltas dadas hasta el momento.

```
bool noranzkoa       # 0: ezkerra; 1: eskuina
uint8 zenbakia
---
# Result
bool arrakasta
float64 amaierako_angelua
float64 hasierako_angelua
uint8 bira_kopurua
---
float64 oraingo_angelua
uint8 bira_kopurua
```

Código 7.13: Mensaje personalizado para la acción de girar sobre sí mismo según el color y número identificado.

7.2.2. Grafo computacional desarrollado

El grafo computacional de *ROS2* es la estructura que representa todos los nodos y las conexiones de comunicación entre ellos. Esta representación permite organizar, visualizar y analizar cómo interactúan los nodos para definir el flujo de la información y facilitar la comprensión, depuración y escalabilidad de la aplicación.

En la Ilustración 35 se presenta el grafo desarrollado. Los óvalos púrpura son los nodos del nivel bajo, que además son de tipo *Lifecycle* como se explica en el apartado 7.2.3, los lilas del nivel medio y los rosas del nivel alto. Los rectángulos azules son los tópicos, los verdes los servicios y los rojos las acciones, todos con su tipo de mensaje.

Por ejemplo, el nodo rosa de nivel alto *pertzepzio_proba*, solicita mediante el servicio *koloreaZenbakia_iragarri*, rectángulo verde, el reconocimiento del color y del número de la caja al nodo lila de nivel medio *pertzepzio_proba_wrapper*, y este a su vez, solicita mediante el servicio *argazkia_atera* una foto al nodo púrpura de nivel bajo *kamera*. Para recibir la información de los sensores, los nodos lila de nivel medio se suscriben a los tópicos, recuadros azules.

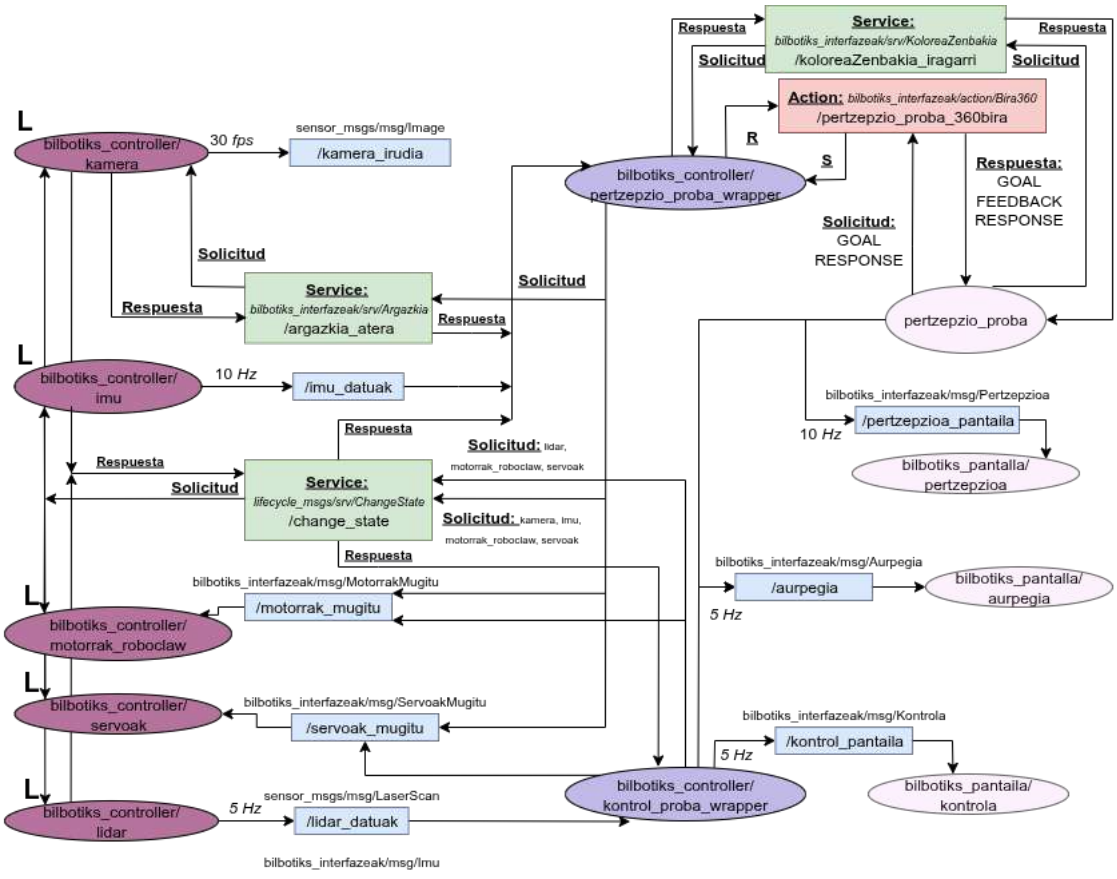


Ilustración 35: Grafo computacional *ROS2* desarrollado.

7.2.3. Nodos de nivel bajo: *Lifecycle*

Los nodos de interacción con el *hardware* se han implementado con un ciclo de vida. Esta característica se conoce como *lifecycle* y fue introducida en la *API* de Python de la distribución *ROS2 Humble Hawksbill*.

Este tipo de nodo implementa una *FSM*, lo que permite controlar el estado del dispositivo físico. En total se definen cuatro estados principales, descritos en la Tabla 5, y siete transiciones, descritas en la Tabla 6. Por ejemplo, cuando un nodo se encuentra en el estado *unconfigured* y se solicita la transición *configure()*, el nodo pasa al estado *inactive*. Todos estos caminos se pueden seguir en la Ilustración 36, donde representa el esquema oficial de un nodo con ciclo de vida.

Esta implementación posibilita encender o apagar los dispositivos *hardware* mediante *software*, y por tanto, disminuir el consumo energético del *rover* y el uso innecesario de recursos de *CPU*, ya que aquellos dispositivos que no se necesiten estarán virtualmente desconectados.

Para cambiar el estado de un dispositivo se utiliza el servicio *change_state* proporcionado por los nodos *Lifecycle*.

Consecuentemente, se han desarrollado cinco nodos con ciclo de vida, uno por cada dispositivo físico: *kamera*, *imu*, *motorrak_roboclaw*, *servoak*, y *lidar*. Todos pertenecen al paquete *bilbotiks_controller*.

Estado	Función
Unconfigured	Estado inicial tras crear el nodo. No tiene recursos asignados ni configuración cargada. Se puede volver a este estado después de un error o limpiar el nodo.
Inactive	El nodo está configurado, pero en general, no está realizando procesamiento de datos ni respondiendo a servicios ni acciones. Permite modificar la configuración y los recursos asignados sin afectar al comportamiento. No obstante, este estado puede incorporar la respuesta a algún servicio como sacar una foto, puesto que el estado de activación respondería a otra función.
Active	Es el estado principal para realizar operaciones. Se procesan datos y se responde a servicios y acciones. En otras palabras, realiza su función principal.
Finalized	Estado de finalización previo a la destrucción del nodo. Permite la depuración del nodo en lugar de destruir el nodo directamente.

Tabla 5: Estados de un nodo con ciclo de vida.

Transiciones	Función
create	Instancia el nodo, el nodo empieza en el estado *unconfigured*.
configure	Transiciona de *unconfigured* a *inactive*. Se asignan los recursos establecidos y carga la configuración necesaria.
cleanup	Transiciona de *inactive* a *unconfigured*. Se liberan los recursos asignados y borra la configuración.
activate	Transiciona de *inactive* a *active*. Se activa el funcionamiento del nodo.
deactivate	Transiciona de *active* a *inactive*. Se liberan los recursos asignados y borra la configuración.
shutdown	Transiciona desde cualquier estado a *finalized*. Se liberan los recursos asignados y borra la configuración antes de destruir el nodo, por lo que no es posible volver a configurar el nodo sin reiniciarlo.
destroy	Elimina el nodo desde el estado *finalized*.

Tabla 6: Transiciones de un nodo con ciclo de vida.

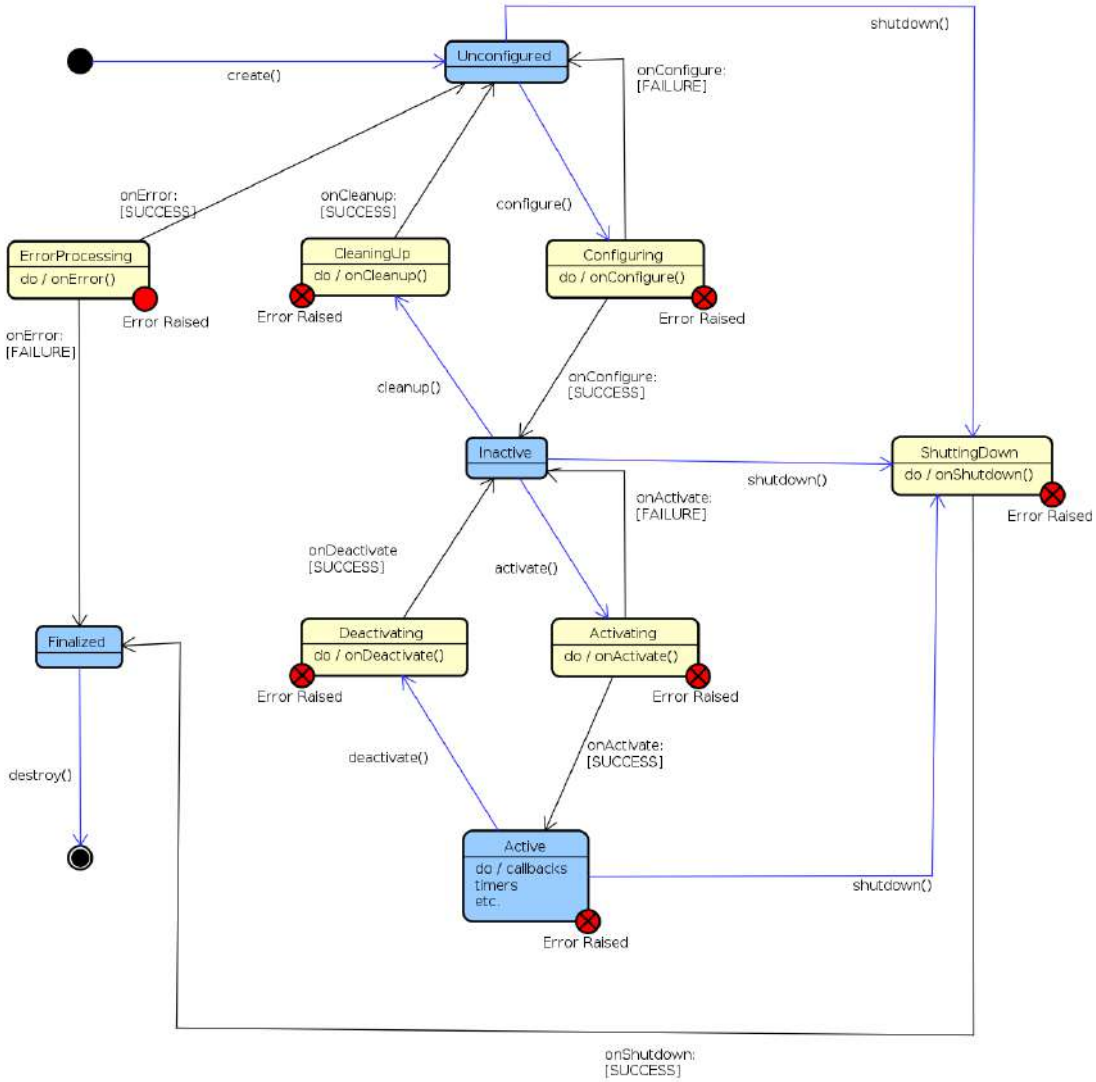


Ilustración 36: Esquema oficial de nodos *lifecycle* [65].

7.2.4. Nodos de nivel medio: *wrapper*

El nivel medio de la arquitectura propuesta permite comunicar las funciones de bajo nivel con las de alto nivel. Las funciones implementadas en este nivel se conocen como *wrapper*, cuyo objetivo es proporcionar una capa de abstracción que simplifique la interacción con el código subyacente.

En este nivel se han implementado los nodos que contienen la lógica, es decir, los algoritmos que permiten solucionar las pruebas. Reciben la información del nivel bajo, la procesan y generan una respuesta para la solicitud del nivel alto.

En total se han desarrollado dos nodos *wrapper*: *pertzepzio_proba_wrapper* y *kontrol_proba_wrapper*. Ambos nodos pertenecen el paquete *bilbotiks_controller*.

7.2.5. Nodos de nivel alto: clientes, interfaces gráficas y *launchers*

El nivel alto corresponde a las funciones de solicitud de inicio de las pruebas y a las interfaces gráficas correspondientes. Solicitan al nivel medio la realización de alguna acción, servicio o prueba y reciben retroalimentación en tiempo real del estado de la prueba para comunicarlo a la interfaz gráfica.

Para iniciar las pruebas hay que arrancar todos los nodos involucrados en ellas, por lo que se utilizan los *launchers* de *ROS2*. Son archivos escritos en *Python* que permiten iniciar y gestionar la ejecución de múltiples nodos de forma coordinada y sencilla en lugar de lanzar cada nodo individualmente. En este archivo se agrupa la configuración necesaria, es decir, qué nodos ejecutar, con qué parámetros y en qué orden.

En total se han desarrollado dos nodos y dos *launchers* en este nivel: nodo *pertzepzio_proba* y los *launchers pertzepzioa* y *kontrola* del paquete *bilbotiks_controller*, y nodo *aurpegia* del paquete *bilbotiks_pantaila*.

7.2.6. Parámetros

En *ROS2* se pueden definir parámetros que se cargan de manera externa al ejecutar los nodos, evitando tener que volver a compilar el paquete si se quiere cambiar algún valor. Estos parámetros se pueden indicar por terminal o desde ficheros *.yaml*, siendo la segunda la opción más cómoda.

En el Código 10.11 se muestra el fichero *.yaml* creado en el proyecto y entre las líneas 30-43 y 50-62 del Código 10.29 se declaran y se leen los parámetros.

7.3. Movimiento del *rover*

Los encargados principales de accionar el movimiento del robot son el nodo que controla los servomotores y el que controla los motores de las ruedas. Ambos tipos de motor se controlar mediante señal *PWM*, por lo que es suficiente con indicar a los servomotores el ángulo en el que deben posicionarse y a los motores la velocidad de movimiento.

En el caso de los servomotores, los valores se reciben en un vector de cuatro posiciones a través de un tópico de *ROS2*, se recorren secuencialmente y se escribe cada ángulo en cada servomotor. Gracias a la sentencia *ServoKit(channels=16).servo[i].angle = angle* de la librería *adafruit_servokit* se escribe el valor del ángulo en el servomotor correspondiente sin esperar la respuesta, es decir, envía la señal y no espera a que termine el movimiento, permitiendo mover todos prácticamente a la vez.

En cuanto a los motores, se introduce la velocidad de giro comprendida entre 0 y 127. La velocidad se publica en el tópico correspondiente con su signo y magnitud, pero después, se procesa la velocidad ya que la magnitud siempre va en positivo y el sentido de giro viene dado por el método *ForwardM1 /2* o *BackwardM1 /2* de la librería *roboclaw*.

El nodo de los servomotores y de los motores puede verse en el Código 10.26 y 10.27.

7.4. Prueba de percepción

Siguiendo el diagrama de secuencia propuesto en 6.6.1, los pasos a seguir son: obtener la imagen de la caja, procesar la imagen para reconocer el color y el número, y por último, girar de manera precisa el número de vueltas y en el sentido acorde a la información de la caja. En la Ilustración 37 se muestran los nodos de *ROS2* desarrollados que desempeñan esta prueba.

El nodo rosa de nivel alto *pertzepzio_proba* solicita al nodo lila de nivel medio *pertzepzio_proba_wrapper* mediante el servicio verde *koloreaZenbakia_iragarri* el color y número identificado en la caja.

Al recibir la solicitud, el nodo de nivel medio solicita al nodo púrpura de nivel bajo sacar una foto mediante el servicio *argazkia_atera*. Tras recibir la foto, la procesa y devuelve el color y el número como resultado del primer servicio al nodo de nivel alto.

Finalmente, el nodo de nivel alto solicita al nodo de nivel medio mediante la acción *pertzepzio_proba_360bira* la realización del giro. Para ello, el nodo de nivel medio se suscribe a los tópicos azules de la *IMU*, motores y servomotores para recibir y publicar la información que gobierna esos dispositivos. Los nodos de nivel bajo se encargan de recibir y publicar dichos valores para controlar los sensores y actuadores.

En este apartado se describe el proceso realizado para el desarrollo de la prueba de percepción.

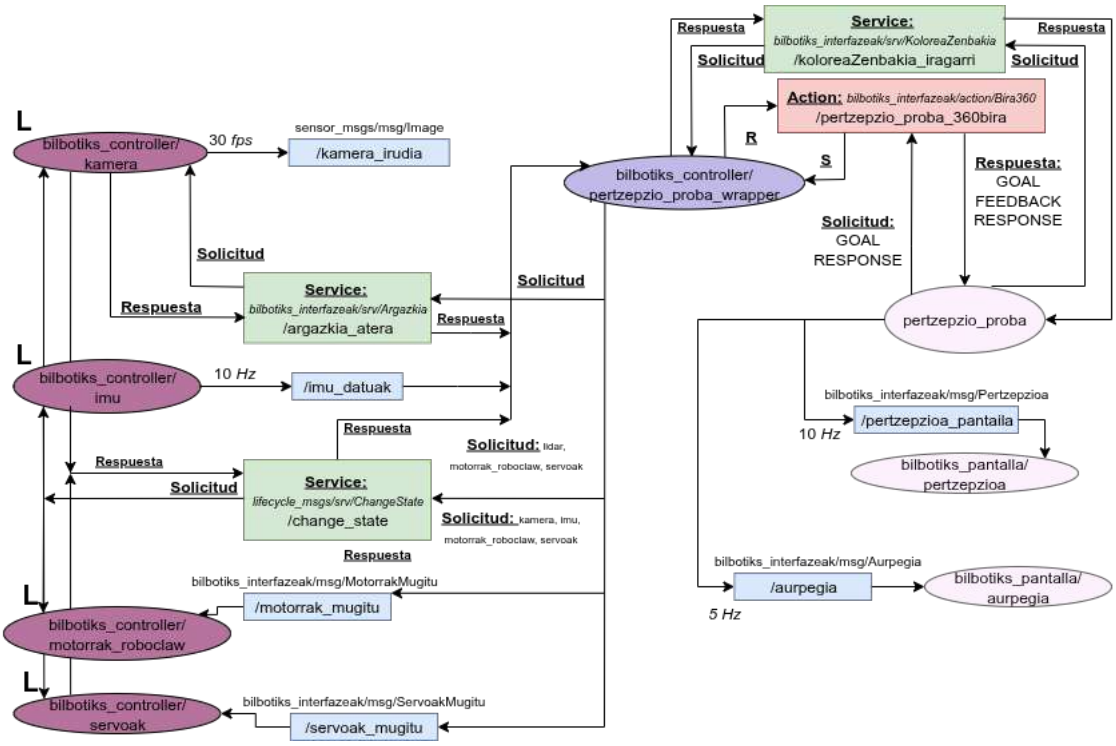


Ilustración 37: Nodos involucrados en la prueba de percepción.

7.4.1. Imágenes digitales

7.4.1.1. Definición y codificación de las imágenes digitales

Una imagen digital es una representación bidimensional de $n \cdot m$ elemetos. A cada uno de estos elementos se le conoce como píxel (*picture element*) [66]. Cada uno de estos píxeles tiene un significado y un tamaño dado por el espacio de color utilizado. Pueden ser de dos, ocho u otra cantidad de bits.

Consecuentemente, el tamaño total de la imagen será el número de píxeles por el número de bits por píxel: $n \cdot m \cdot bits$.

Por ejemplo, una imagen de 1920x1080 píxeles con ocho (8= bits por píxel, tendrá un tamaño de $1920 \cdot 1080 \cdot 8 = 16{,}588{,}800$ bits $/8 = 2073600$ bytes $= 2,0736$ MB (megabytes). El fragmento de la Ilustración 38 tendrá un tamaño de 512 bits.

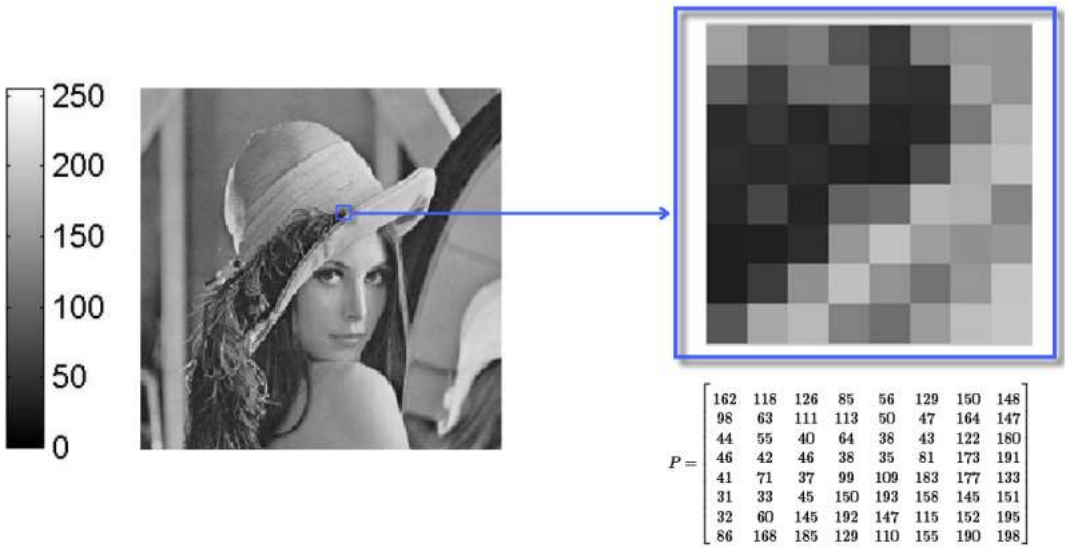


Ilustración 38: *Imagen con 8 bits por píxel y representación de un fragmento de 8x8 píxeles [67].*

7.4.1.2. Compresión de imágenes digitales

Esta técnica elimina información redundante o innecesaria en la codificación de la imagen para reducir el número de bits necesarios para almacenar la imagen. Este proceso puede implicar una pérdida de definición de la imagen en función del algoritmo de compresión utilizado. Asimismo, puede ser reversible.

Los distintos formatos de almacenamiento de imágenes implementan diferentes algoritmos de compresión. En la siguiente tabla se muestran las características de los formatos más extendidos [68]:

Formato	Uso	Compresión
JPEG	Fotografías y escenas reales	Con pérdida
PNG	Gráficos	Sin pérdida
GIF	Diagramas y animaciones simples	Sin pérdida
BMP	Programas de *Microsoft*	Sin pérdida
TIFF	Manipulación de imágenes	Con y sin pérdida

Tabla 7: *Tipo de compresión de imágenes según el formato.*

7.4.1.3. Obtención de la imagen digital en *ROS2* y *OpenCV*

La manera más sencilla de interactuar con la cámara desde *Python* es haciendo uso de la librería *cv2* de *OpenCV*. Después, se utiliza la librería *cv_bridge* para convertir la imagen en un mensaje de *ROS2*, concretamente *sensor_msgs/Image* [69], y poder mandarla a otros nodos. Finalmente, el nodo que recibe la fotografía la vuelve a codificar en formato legible por *cv2* y puede realizar el procesamiento requerido.

En la Ilustración 39 se resumen gráficamente el proceso implementado. Se ha definido un tipo de servicio personalizado, 7.11, en el que la cabecera de la solicitud es vacía y la respuesta un objeto de tipo *sensor_msgs/Image*.

El nodo *kamera* implementa el servidor del servicio *argazkia_atera*, que accede a la cámara, saca una foto, convierte la foto en un mensaje *sensor_msgs/Image* de *ROS2* y la envía en la respuesta del servicio. La implementación del servidor y la función de respuesta están en el Código 10.12 y 10.13.

El nodo *pertzepzio_proba_wrapper* implementa el cliente del servicio y realiza la solicitud, 10.14, que en este caso será vacía. Este envía la petición mediante una llamada asíncrona, pero como no es necesario realizar más procesos en paralelo se espera la respuesta de manera síncrona y se almacena en la variable *argazkia*. Por último, convierte dicha variable, que es un mensaje *ROS2* de tipo *sensor_msgs/Image*, en codificación *BGR8*, la codificación por defecto de *cv2*. Esta codificación se compone de tres canales, *Blue Green Red*, *Azul Verde Rojo*, de ocho bits cada uno.

Ilustración 39: Servicio para sacar una foto.

7.4.2. Identificación de colores y formas

La detección de colores es un problema clásico de la visión artificial. Existen diferentes modelos para la representación de los colores, siendo *HSI* el más cercano a la percepción humana [70]. Estos modelos de color determinan cómo es descrito y codificado cada píxel de la imagen.

Además, para minimizar errores y mejorar el rendimiento del robot se han añadido restricciones de forma geométrica que tienen que cumplir el color detectado para validarlo. El color a identificar es de forma cuadrada, por lo que si se detecta alguno de los colores deseados y no tiene un contorno cuadrado similar a las dimensiones de la caja, no se marcará como detectado. Estas restricciones eliminan el ruido ocasionado por pequeños píxeles.

A continuación se describe la codificación de las imágenes en los espacios de color más empleados y la implementación de esta tarea junto con las restricciones.

7.4.2.1. Modelo *RGB*

El modelo *RGB*, *Red Green Blue*, *Rojo Verde Azul*, utiliza esos tres colores primarios para representar todos los colores. Se basa en un espacio de coordenadas cartesianas donde los colores son puntos del interior del cubo definidos por vectores que se extienden desde el origen [70]. Por tanto, para generar un color se mezcla una cantidad determinada de cada componente o canal primario.

La codificación más habitual de un espacio de color *RGB* es de ocho bits por componente, es decir, veinticuatro bits. De esta manera, cada canal puede tomar valores de 0 a 255.

Este modelo es ideal para procesar imágenes que se expresan en estos tres canales primarios, como se muestra en el trabajo [71] para estimar la clorofila con una cámara de vídeo sobre fondo negro.

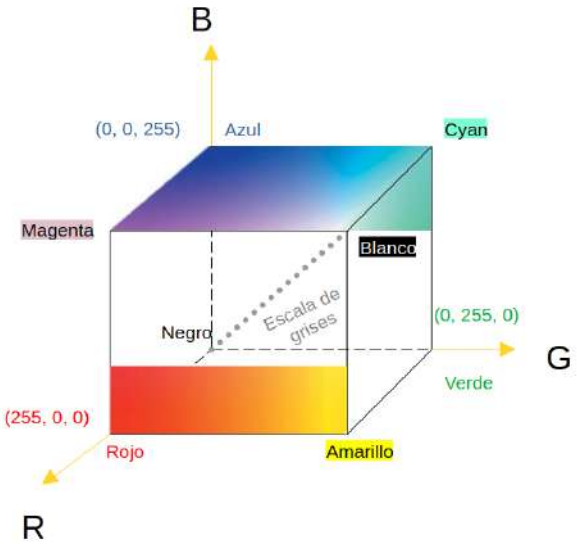


Ilustración 40: *Representación del espacio de color RGB de 24 bits. Imagen propia.*

7.4.2.2. Modelo *CMY*

Emplea los colores secundarios, *Cyan Magenta Yellow*, *Cian Magenta Amarillo*, en lugar de los primarios. Su representación espacial es la misma que la del modelo *RGB*, modificando las coordenadas de los canales utilizados. Cada componente absorbe a otra del espacio *RGB*, por lo que el modelo *CMY* de veinticuatro (24) bits se define de la siguiente manera:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 255 \\ 255 \\ 255 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Una peculiaridad de este modelo es que no puede generar un color negro puro, motivo por el que se le añade una cuarta componente *K* para el negro, dando lugar al espacio de color *CMYK*.

Las impresoras utilizan este modelo porque el papel absorbe la tinta y al tratarse de un sistema sustractivo se generan los colores mediante la absorción de la luz en lugar de la emisión.

7.4.2.3. Modelo *HSV*

El modelo *Hue Saturation Value*, *Tono Saturación Valor*, extrae de las imágenes esas tres características. En primer lugar, es imprescindible definir qué describe cada elemento.

El tono describe la escala de grises de la imagen, esto es, los niveles de brillo que van desde el negro hasta el blanco. En la representación espacial se refiere al ángulo, de 0 grados a 360 grados sobre la circunferencia de la base.

La saturación describe la pureza o la intensidad del color. Una alta saturación se refiere a un color vivo o un objeto o elemento destacado en la imagen. El rango de valores es de cero, sin pigmento o blanco, a uno.

El valor o brillo describe la luminosidad u oscuridad de la imagen, es decir, mide la cantidad de luz que una imagen emite o refleja. El rango de valores es de cero, sin brillo, a uno.

El espacio de color HSV se obtiene a partir de una transformación no lineal del espacio RGB [67]:

$$H = \arccos\left(\frac{\frac{1}{2}\cdot((R-G)+(R-B))}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right)$$

$$S = 1 - 3\cdot\frac{\min(R,G,B)}{R+G+B}$$

$$V = \frac{1}{3}\cdot(R+G+B)$$

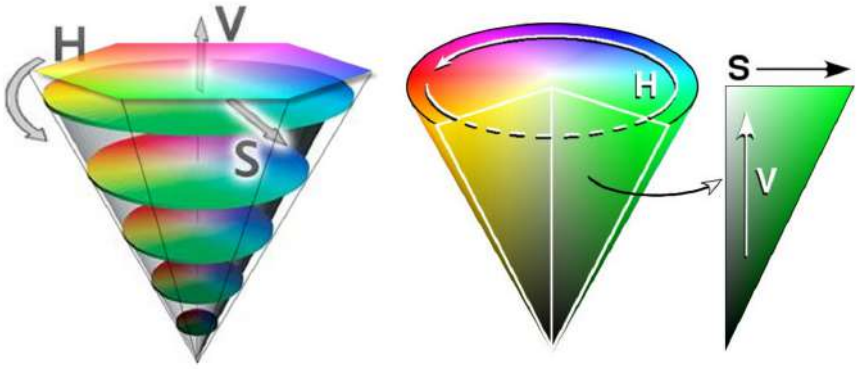


Ilustración 41: *Representación del espacio de color HSV [72].*

La principal ventaja de este modelo es que al separar la información del color de la intensidad luminosa, el tono y la saturación permanecen relativamente estables facilitando la identificación de diferentes variantes de un mismo color. En otras palabras, podría decirse que para la detección de colores el modelo *HSV* es invariante a las condiciones lumínicas.

Consecuentemente, es el modelo de color más adecuado para resolver la prueba de percepción.

7.4.2.4. Restricciones geométricas

Como se menciona en el apartado 4.2.1, se conoce que la caja tiene una forma cuadrada de 40 centímetros de lado y que se colocará centrada a 1,5 metros del *rover*. Por tanto, se han impuesto dos restricciones geométricas:

- Se ha reducido la amplitud de la cámara aplicando un *zoom* centrado del 57 %, un recorte inferior del 30 % y un recorte del 20 % por cada lado. Este procesado permite centrar el enfoque de la cámara en la caja como se observa en la Ilustración 42, disminuyendo la cantidad de objetos que pueden despistar a la tarea. La porción de código correspondiente se puede ver en el Código 10.15.

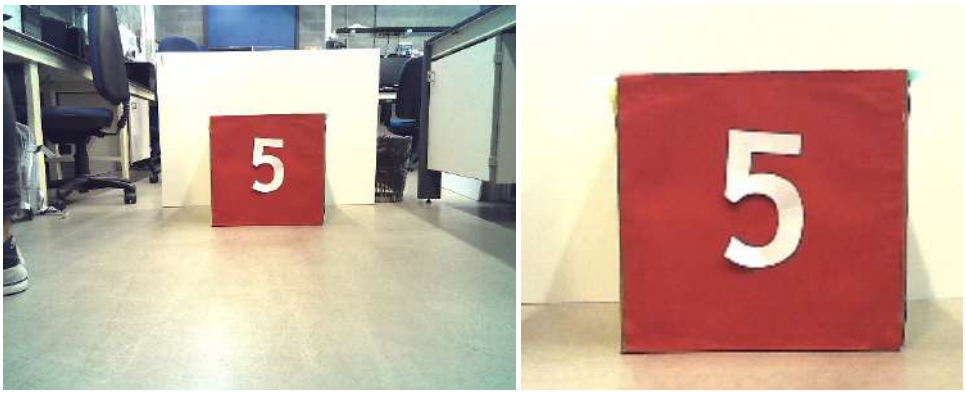


Ilustración 42: Resultado de la primera restricción, donde se aprecia la caja como elemento principal de la imagen.

- Se ha establecido un tamaño máximo de píxeles que puede ocupar la caja, considerando la distancia y su tamaño. Para ello, se han sacado fotografías con la cámara del *rover*, se han identificado los contornos de color mediante el modelo de color *HSV*, apartado 7.4.2.5, y se ha obtenido las dimensiones de cada contorno detectado. El resultado ha sido entre 225 y 275.

  Además, se han realizado numerosas pruebas incluyendo el reflejo del color en el suelo, lo que ha exigido aumentar los valores de píxeles de altura, y la caja posicionada de manera no perpendicular al campo de visión de la cámara, lo que ha requerido disminuir los valores de píxeles de horizontal porque hace efecto de profundidad.

  Por tanto, para que un color sea considerado válido tiene que tener entre 150 y 300 píxeles de ancho y entre 200 y 350 píxeles de alto.

  En la Ilustración 43 se muestra el resultado de esta restricción, concretamente cuando el programa no detecta el color azul en aquellos objetos que no cumplen la restricción. La implementación de la restricción está en el Código 10.16.

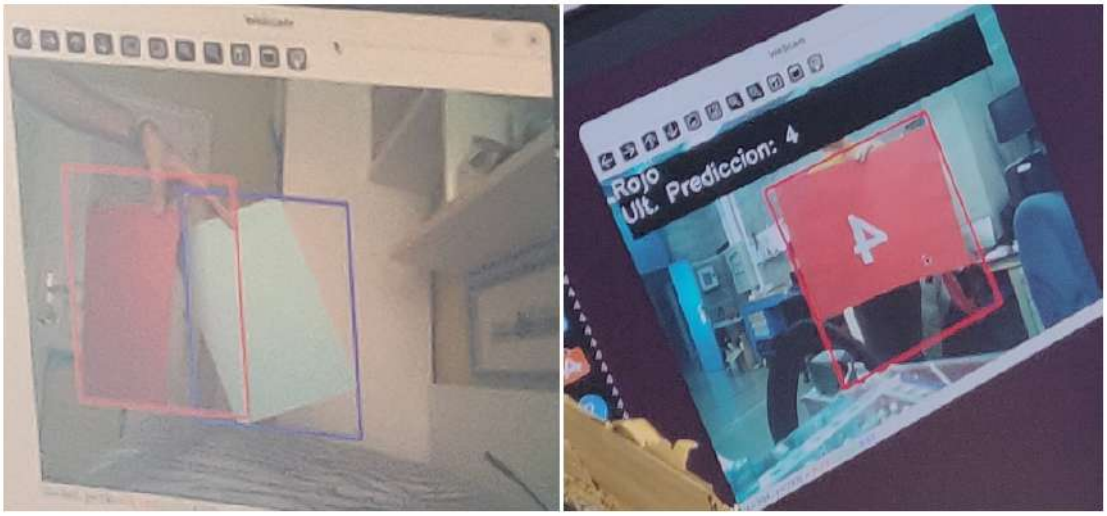


Ilustración 43: Resultado de la segunda restricción. No se detecta el color azul con forma circular, no rectangular ni cuadrada.

7.4.2.5. Modelo *HSV* en *OpenCV*

La librería de *OpenCV* permite obtener fácilmente la codificación de una imagen utilizando el espacio de color *HSV* y representa las componentes con los siguientes rangos de valores [73]:

- *Hue*: [0, 179]
- *Saturation*: [0, 255]
- *Value*: [0, 255]

Por ello, es necesario trasladar los valores del modelo original a los valores utilizados por *OpenCV*.

En primer lugar, hay que obtener los valores de los colores azul y rojo. Para ello, se realizan transformaciones desde el modelo *RGB* como se explica en el apartado 7.4.2.3.

En segundo lugar, se transforman al rango de valores de *OpenCV*.

Los valores resultantes se presentan en la Tabla 8. Hay que tener en cuenta que el valor más importante es el tono o *hue*, ya que nos da la tonalidad del color, mientras que la saturación y el valor permiten generar variantes del tono con diferentes condiciones de intensidad y luminosidad. Por tanto, dos colores puros únicamente se diferenciarán por su tonalidad.

Color	*RGB*	*HSV*	*HSV* en *OpenCV*
Rojo	(255, 0, 0)	(0, 1, 1/3)	(0, 255, 85)
Azul	(0, 0, 255)	(240, 1, 1/3)	(120, 255, 85)

Tabla 8: Colores en *HSV* de *OpenCV*.

En tercer lugar, es necesario determinar para qué rangos de tonalidad se va a considerar cada color. Al obtener una imagen, debido a las condiciones ambientales, es prácticamente imposible obtener la lectura de un color puro, pero esto no significa que los valores obtenidos no correspondan al color que queremos identificar. En este paso se ha utilizado ingeniería inversa, es decir, se ha propuesto un rango no excesivamente amplio y se ha calculado sus valores *RGB* para conocer el color del que se trata hasta determinar el rango deseado. Los rangos resultantes se presentan en la Tabla 9. Para simplificar los cálculos, la saturación y el valor se han considerado como uno 1.

Es importante observar que la tonalidad del rojo está entre el comienzo y final del espacio de color, por lo que su rango va de 160 a 179 y de 0 a 10.

Color	Rango de *H* en *OpenCV*	*RGB*	Color asociado
Azul	80	(0, 255, 170)	Turquesa intenso
	130	(85, 0, 255)	Violeta intenso
Rojo	160	(255, 0, 170)	Magenta intenso
	10	(255, 42, 0)	Naranja intenso

Tabla 9: Rangos de tonalidad en *HSV* de *OpenCV*.

En cuarto lugar, hay que determinar el rango para lo valores de saturación y luminosidad. En este paso también se ha aplicado ingeniería inversa, aunque el proceso puede simplificarse observando la Ilustración 41 y aplicando las siguiente dos reglas:

- Para el color azul, que es menos intenso que el rojo, únicamente tendremos que eliminar el rango de saturación y luminosidad que pueda confundirse con el color blanco y negro. El tono impedirá confundirlo con colores como el verde.
- Para el color rojo, aplicamos la misma regla que para el azul, pero restringiendo más los rangos porque una baja saturación podría confundirse con el amarillo o el naranja, y un bajo valor con el gris.

Los valores resultantes se exponen en la Tabla 10.

Por último, se define una máscara por cada color con los umbrales determinados y se aplican las máscaras a la imagen.

Color	Rango de *SV* en *OpenCV*
Rojo	(100, 255)
Azul	(50, 255)

Tabla 10: Rangos de saturación y luminosidad en *HSV* de *OpenCV*.

Una máscara es una imagen binaria, es decir, en blanco y negro, que destaca las regiones de interés dentro de la imagen original. Los píxeles cuyos valores no están en el rango establecido, son puestos a cero y descartados, y los que sí están en el rango, son puestos a uno y seleccionados. En este proyecto las máscaras se utilizan para seleccionar los píxeles que están dentro de los umbrales de color.

En el caso del color rojo, al estar al comienzo del tono, es necesario definir dos máscaras auxiliares, una para cada rango, y posteriormente, unirlas mediante una operación disyuntiva *OR*, esto es, el píxel podrá estar en la primera máscara auxiliar o en la segunda.

La porción de código correspondiente está en el Código 10.17.

7.4.3. Identificación del número con redes neuronales

Para reconocer el número que aparece en una imagen o fotografía es necesario analizar los patrones y las formas que aparecen en ella. Para ello, es común utilizar herramientas de inteligencia artificial. Las técnicas más populares son *OCR* y el procesamiento de imágenes mediante redes neuronales de tipo convolucional.

En este apartado se describen ambas técnicas y el proceso y desarrollo implementado.

7.4.3.1. *OCR - Optical Character Recognition*

Esta técnica se fundamenta en un modelo de red neuronal entrenada para reconocer caracteres. El entrenamiento incluye miles de muestras de cada carácter posible para que el modelo aprenda cómo es ese carácter y se detiene cuando la precisión de la predicción reconoce adecuadamente cada elemento en casos reales. Además, las imágenes se segmentan en celdas individuales y se procesan por separado. Esta estrategia mejora la actuación del algoritmo separando los caracteres individuales, pero tiene un alto coste computacional provocando largos tiempos de procesamiento.

El modelo de red neuronal puede ser de diferentes tipos, destacando las redes neuronales convolucionales, *CNN* por sus siglas en inglés.

La implementación de técnicas con un coste alto como *OCR* dependerá de la necesidad de precisión y condiciones del entorno. En el caso de la prueba de percepción, la segmentación de la imagen es prescindible debido a la homogeneidad de las imágenes a procesar, por lo que analizar la imagen mediante una red neuronal es suficiente y así poder ahorrar coste computacional.

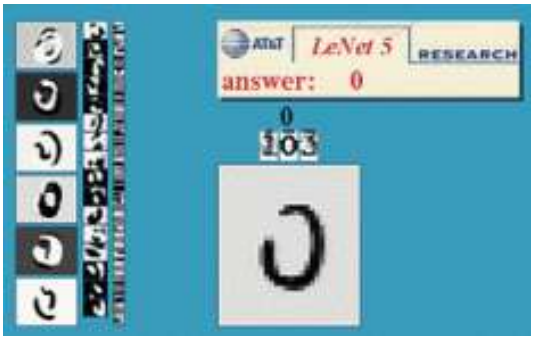


Ilustración 44: *Ejemplo de OCR. Fuente: [74].*

7.4.3.2. *CNN - Convolutional Neural Network*

Una red neuronal convolucional es un tipo de red neuronal diseñada para procesar imágenes extrayendo automáticamente las características que las definen. Como su nombre indica, se fundamenta en la operación matemática de convolución.

Esta operación consiste en aplicar un pequeño filtro o *kernel*, por ejemplo, de 3x3 píxeles, sobre diferentes regiones de la imagen de entrada. En cada posición, el filtro y la región de la imagen se multiplican elemento a elemento y se suman, produciendo un único valor de salida. Este proceso se repite desplazando el filtro por toda la imagen, generando así un mapa de características que resalta patrones locales como bordes, texturas o formas simples [30].

En otras palabras, cada filtro o capa convolucional que se aplica reduce progresivamente la dimensionalidad de la imagen, por lo que cada una de ellas aprende una característica diferente.

Estos mapas de características se van agrupando e introduciendo en las sucesivas capas convolucionales con el objetivo de extraer patrones cada vez más complejos. Estas capas suelen alternarse con capas de *pooling* para reducir la dimensionalidad de los mapas de características, pero conservar la información más relevante.

Posteriormente, se incluyen capas de aplanado o *flatten* para convertir la salida multidimensional de las capas anteriores en un vector unidimensional, capas densas para conectar cada neurona a todas las neuronas de la capa anterior y por último, la capa de salida, que transforma las representaciones internas aprendidas por la red en una predicción interpretable.

Cada capa densa genera más conexiones y por tanto, más pesos. En función de la dificultad de la tarea serán necesarias más o menos capas densas.

La salida de la capa de salida puede ser un valor determinado o, en este caso, la probabilidad de cada clase.

Todo este proceso se puede visualizar de manera esquemática en la Ilustración 45.

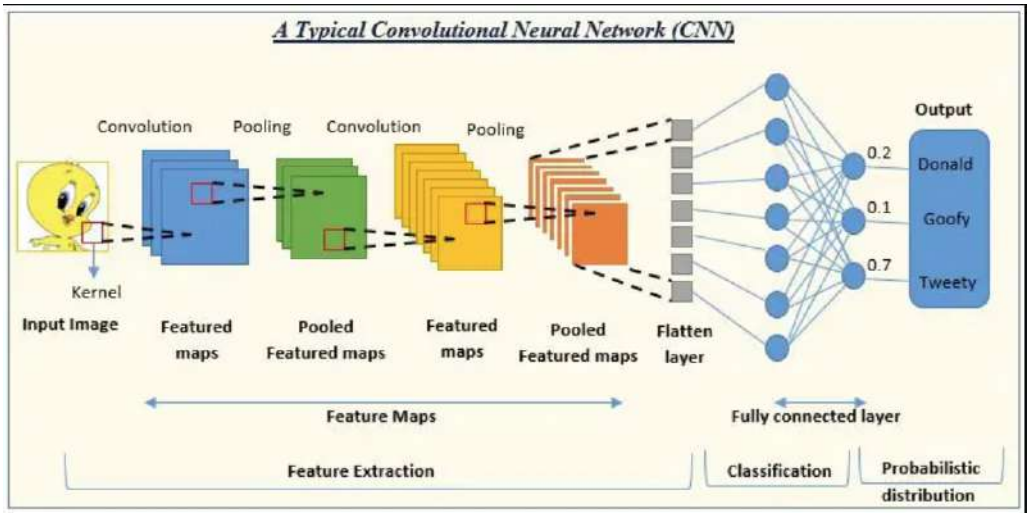


Ilustración 45: Esquema del funcionamiento de una *CNN* [75].

7.4.3.3. Generación del conjunto de datos de entrenamiento

Un conjunto de datos o *dataset* es una colección organizada de datos que contiene la información que un modelo de aprendizaje automático necesita para poder aprender.

La calidad y variedad del conjunto son factores determinantes para el éxito de cualquier modelo. Un *dataset* de calidad debe ser limpio, estar bien etiquetado y ser representativo del problema real que se quiere abordar. Si los datos son insuficientes, están sesgados o contienen errores, el modelo resultante será incapaz de generalizar correctamente y sus predicciones serán poco fiables. Por ello, es fundamental asegurarse de que el conjunto cubra la mayor diversidad posible de casos y escenarios, evitando así que el modelo aprenda patrones erróneos o irrelevantes.

En el caso del procesamiento de imágenes con redes *CNN*, proveer de imágenes con diferentes condiciones ambientales, de posición y orientación ayuda a detectar los patrones que mejor caracterizan a cada una de las clases a predecir.

Además, todas las clases que se quieran predecir tienen que estar representadas de manera equitativa para evitar introducir un sesgo o preferencia hacia ciertas clases. Para lograrlo se aplican técnicas de *undersampling* u *oversampling* que consisten en reducir o aumentar el número de registros de la clase mayoritaria o minoritaria, respectivamente, hasta equilibrar todas las categorías. No obstante, como en este proyecto se ha generado un *dataset* propio se han generado datos equilibrados entre clases y así evitar el uso de estas técnicas.

Este conjunto de datos se divide en tres subconjuntos, cada uno con un propósito diferente:

- **Subconjunto de entrenamiento (*train*)**: es la porción más grande del *dataset* y se utiliza para que el modelo aprenda los patrones y relaciones en los datos. Supone el 70 % del conjunto.
- **Subconjunto de validación (*val*)**: se emplea durante el entrenamiento para ajustar los hiperparámetros y prevenir el sobreajuste, permitiendo evaluar cómo

generaliza el modelo a datos que no ha visto durante el aprendizaje. Supone el 15 % del conjunto.

- **Subconjunto de pruebas o desarrollo (*test/dev*)**: sirve para medir el rendimiento final del modelo una vez completado el entrenamiento y la validación, proporcionando una estimación objetiva de su capacidad de generalización frente a nuevos datos. Las métricas de rendimiento de los modelos se obtienen a partir de este subconjunto de datos. Supone el 15 % del conjunto.

Para generar el conjunto de datos necesario para superar la prueba se ha utilizado la identificación de colores y formas del apartado 7.4.2 y fotografiado las regiones resaltadas por dicho algoritmo. La Ilustración 46 muestra varios ejemplos de las imágenes que forman el *dataset* generado. Para simplificar el proceso, las fotografías únicamente se han obtenido sobre fondo rojo, puesto que al aplicar después el algoritmo de detección de bordes *Canny* del apartado 7.4.3.4, el color del fondo resulta intrascendente.

La porción de código *Python* para la realización de este proceso está en el Código 10.18.

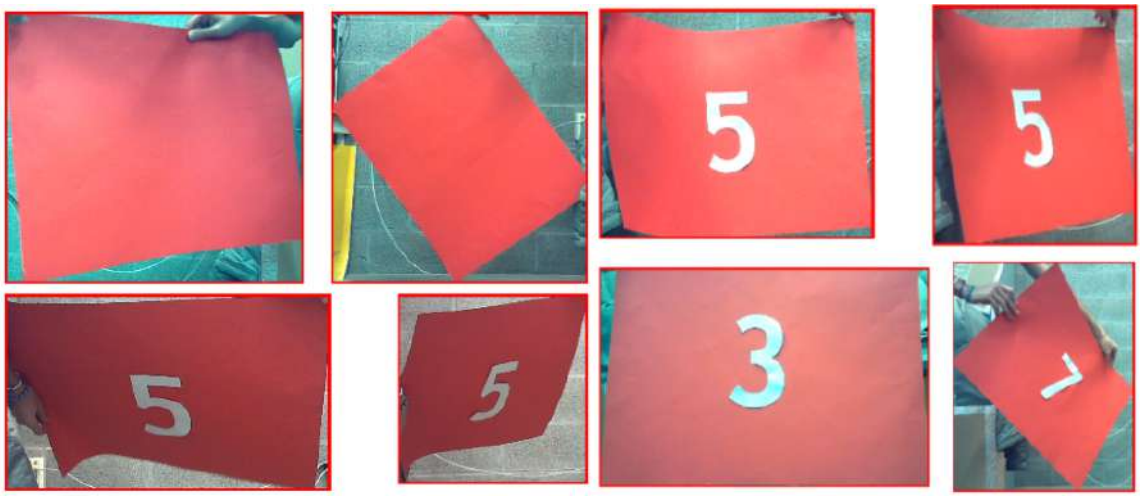


Ilustración 46: Ejemplos del *dataset* generado.

Tras obtener las imágenes se ha realizado un filtrado manual para eliminar aquellas que introduzcan ruido o distracciones al sistema. Un ejemplo de ello es la Ilustración 47.

Ilustración 47: Ejemplo de imágenes eliminadas del *dataset* generado.

Después, se ha aplicado el citado algoritmo de *Canny*, descrito en el apartado 7.4.3.4, con el objetivo de resaltar los bordes del número y facilitar al modelo el aprendizaje de características. En la Ilustración 48 se presenta la imagen original y el resultado de la aplicación de este algoritmo, y la implementación del código en el Código 10.19.

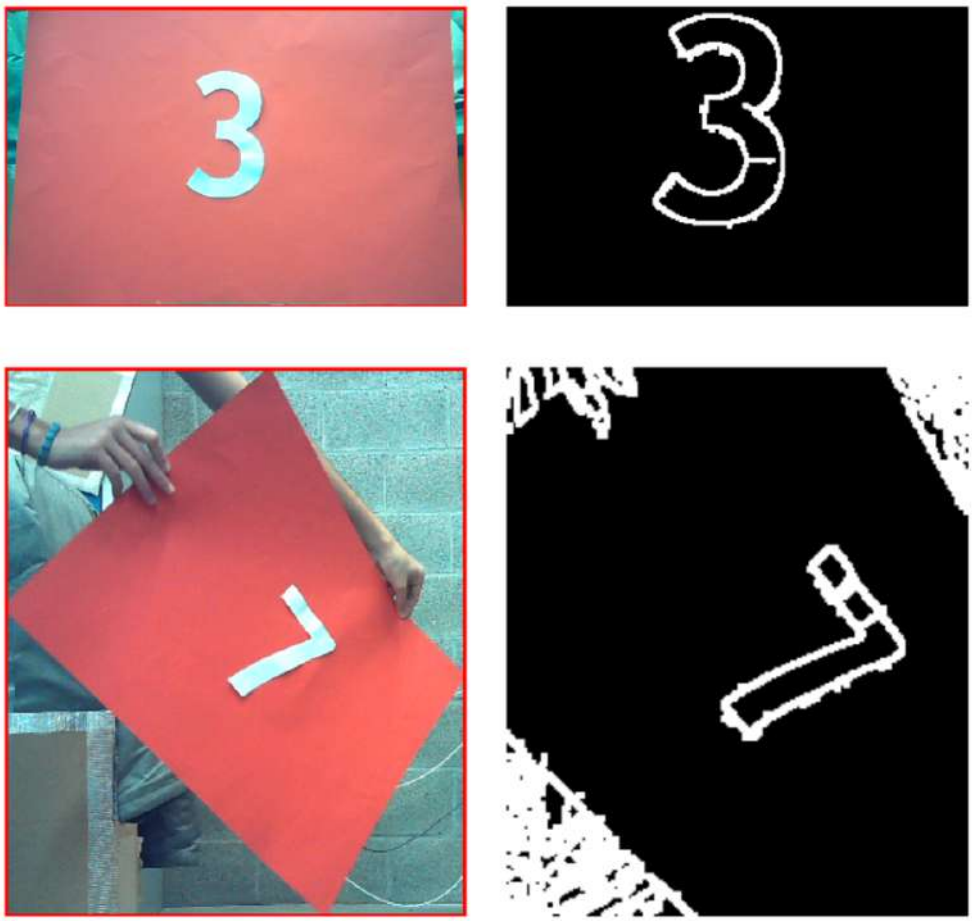


Ilustración 48: Ejemplo del resultado de aplicar el algoritmo de detección de bordes *Canny*

Asimismo, con el fin de reforzar el entrenamiento del modelo se han generado más imágenes de manera sintética, es decir, imágenes generadas por ordenador que simulan a las imágenes reales. Se han creado cien imágenes de este tipo por cada clase, salvo para la categoría de "sin número" o "NaN". En la Ilustración 49 se presentan varios ejemplos de las imágenes sintéticas generadas y la porción del código en el Código 10.20.

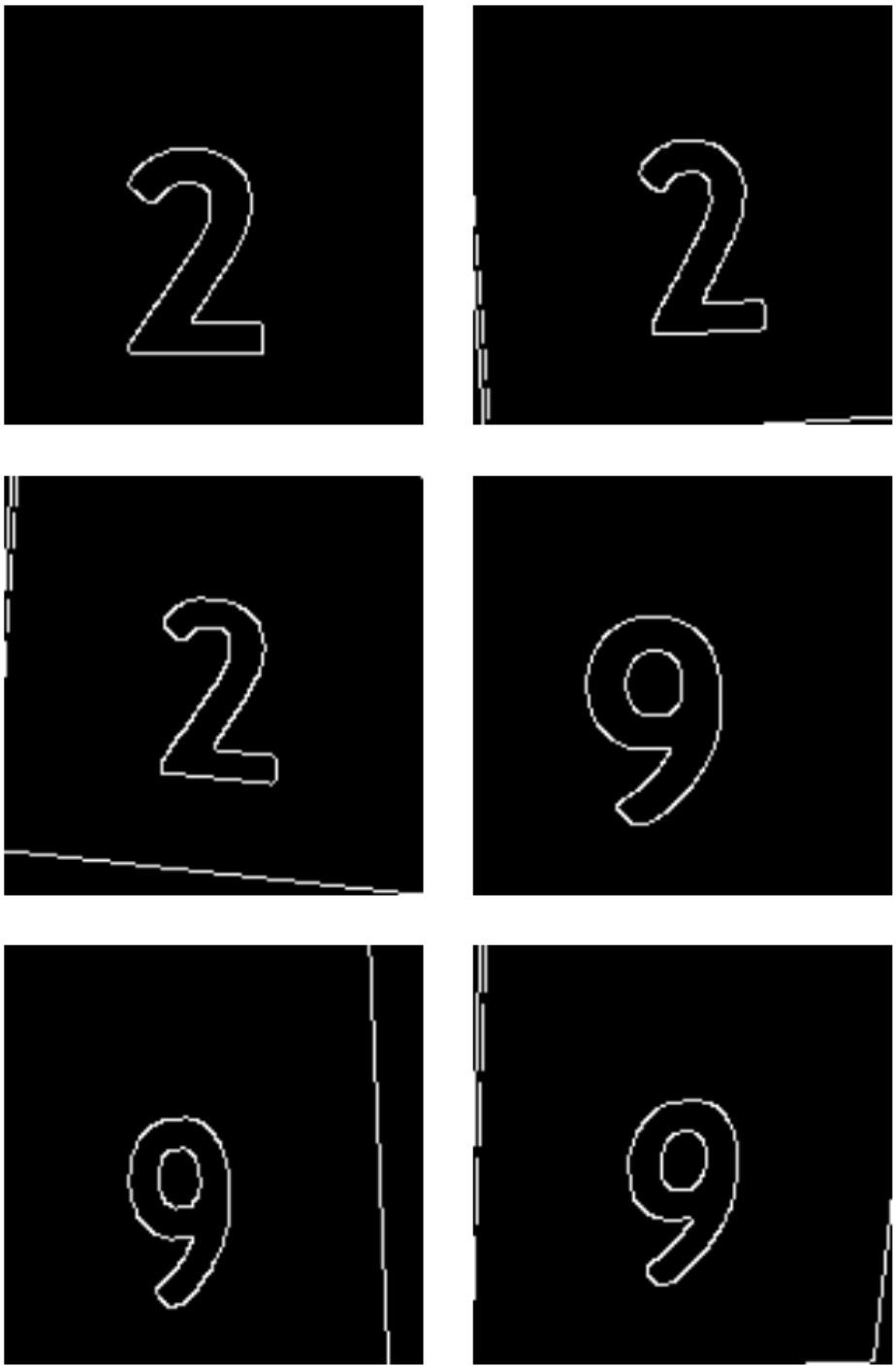


Ilustración 49: Ejemplos de las imágenes sintéticas generadas.

Finalmente, se ha dividido el conjunto de datos en los subconjuntos de entrenamiento, validación y test mediante el trozo de Código 10.21.

En total, 10.736 imágenes diferentes componen el conjunto de datos, con el reparto entre diferentes clases recogido en la Tabla 11.

Clase	Número de imágenes
1	1221
2	1041
3	1200
4	1076
5	1198
6	1168
7	917
8	1066
9	951
NaN	898

Tabla 11: Número de imágenes que integran el *dataset* por clases.

7.4.3.4. Algoritmo de detección de bordes *Canny*

El algoritmo de detección de bordes *Canny* es un enfoque computacional para la detección de bordes en imágenes planteado en el año 1986 por John Canny [76], licenciado en informática, física teórica e ingeniería eléctrica por la Universidad de Adelaida, Australia, doctor por el *MIT* y actual profesor de la Universidad de California Berkeley.

Este algoritmo logra una detección precisa de los bordes y es robusto frente al ruido. Aunque hoy en día se aplica fácilmente con librerías como *OpenCV*, que ya incluyen la implementación, es fundamental conocer sus bases para entender sus resultados y ajustar sus parámetros correctamente.

El objetivo es detectar todos los bordes verdaderos de una imagen, para lo que se necesita localizar dichos bordes con precisión y minimizar las respuestas múltiples a un mismo borde. Una misma transición de intensidad en las diferentes etapas no debe generar varias líneas o marcas paralelas que representen al mismo borde.

La primera etapa es la reducción o supresión del ruido mediante un filtro *gaussiano* para evitar que se generen bordes falsos [28]. Este filtrado sustituye el valor de cada píxel de la imagen por la media de los niveles de intensidad de los píxeles vecinos definida por la máscara del filtro [70], es decir, por un promedio del entorno. Los bordes también se caracterizan por transiciones bruscas de intensidad, por lo que este tipo de filtros tienen el efecto secundario de difuminar los bordes. Sin embargo, los números, cuyos bordes hay que detectar, son de color blanco y al estar colocados sobre un fondo de color rojo o azul, el contraste generado tiene una intensidad suficiente para que el promedio mantenga una transición brusca.

La segunda etapa es la identificación de las zonas donde se producen los mencionados cambios bruscos de intensidad mediante el cálculo de la magnitud y dirección del

gradiente, que indica dónde se ubica el cambio más rápido en la intensidad de la imagen en cada punto. La magnitud permite localizar los bordes, es decir, cambios intensos, y la dirección la orientación del borde.

La tercera etapa consiste en adelgazar los bordes detectados para conseguir bordes de un solo píxel de grosor. El píxel seleccionado será el máximo local en la dirección del gradiente. No obstante, para enfatizar en los bordes y con la intención de que ganen mayor importancia que los posibles bordes de detrás de la cartulina o de la caja, los bordes de los números se han ensanchado y engruesado mediante dilatación.

La cuarta y última etapa es la reducción de la probabilidad de detectar bordes falsos mediante la umbralización con histéresis. Esta técnica permite decidir qué píxeles detectados como posibles bordes deben considerarse como tal, de forma que si la magnitud del gradiente de un píxel es mayor que el umbral alto se considera borde fuerte, si la magnitud del gradiente está entre ambos umbrales se considera borde débil, y si la magnitud del gradiente es menor al umbral bajo se descarta. Los bordes débiles solo se mantienen cuando están unidos a un borde fuerte.

El resultado de aplicar este algoritmo se puede observar en la Ilustración 48 y la implementación en el Código 10.19.

El umbral inferior utilizado es 20 y el superior 110, sobre 255. Por una parte, el valor del umbral inferior permite que se consideren candidatos a aquellos cuyo cambio de intensidad es moderado, y no supone un riesgo puesto que si no están conectados a un borde fuerte se descartarán. Por otra parte, el valor del umbral superior asegura que se detecten los contornos principales del número sin incluir bordes espurios, ya que el contraste entre el número blanco y el fondo de color es alto.

Para la dilatación y ensanchado de los bordes del número se ha empleado un *kernel* o una matriz de 3x3 píxeles que recorre la imagen y convierte en blancos, valor 255, los píxeles vecinos. En otras palabras, este *kernel* contagia de blanco a sus ocho vecinos inmediatos. Un tamaño mayor supone ensanchar aún más los bordes con el riesgo de que estos se unan y el número quede desfigurado.

Tanto los valores del umbral como el tamaño de la matriz y número de iteraciones se han elegido mediante prueba y error.

7.4.3.5. Arquitectura y entrenamiento del modelo *CNN* diseñado

En cuanto al diseño de la arquitectura, tal y como se propone en [30], se selecciona un modelo CNN con siete capas y la siguiente arquitectura:

1. **Capa convolucional y *pooling*.** Recibe la imagen de entrada con un tamaño de 64x64 píxeles y tres canales de *RGB*. Se aplican 32 filtros de 3x3 para extraer las características básicas como bordes, contornos y texturas simples, y función de activación *ReLU*.

   Se reduce la dimensionalidad espacial de los mapas de activación a la mitad con un *kernel* de 2x2 píxeles. En cada bloque, se selecciona el valor máximo con el objetivo de extraer las características más relevantes de cada región y reducir la sensibilidad a pequeñas variaciones en la imagen.

Se utiliza una normalización de *batch* y *dropout* del 30 % para estabilizar el aprendizaje, acelerar la convergencia y prevenir el sobreajuste.

2. **Capa convolucional y *pooling*.** Se aumenta la profundidad de la red de 32 a 64 filtros para extraer patrones más complejos. Se aplica la función de activación *ReLU*.
3. **Capa convolucional y *pooling*.** Se aumenta la profundidad de la red de 64 a 128 filtros para extraer patrones aún más complejos. Se aplica la función de activación *ReLU*.
4. **Capa de aplanado o *flatten***. Se convierten los mapas de características multidimensionales, es decir, los resultados de las capas anteriores, a un vector unidimensional para poder ser introducidos en las capas densas.
5. **Capa densa.** Se compone de 128 neuronas y regularización *L2* para penalizar los *outliers* o valores atípicos en los pesos. Refuerza la capacidad de generalización del modelo. Se aplica la función de activación *ReLU*.
6. **Capa densa.** Se compone de 64 neuronas y refina la representación de características. Se aplica la función de activación *ReLU*.
7. **Capa de salida.** Se compone de 10 neuronas o salidas, una por cada clase. La función de activación *softmax* convierte los valores de las neuronas en probabilidades que suman uno, permitiendo la clasificación en múltiples clases.

Para el entrenamiento se ha establecido un tamaño de lote de 64, un tamaño de imagen de 64x64 píxeles para todas las imágenes y una tasa de aprendizaje de 0,001. Redimensionar todas las imágenes a un tamaño normalizado facilita el entrenamiento.

Además, se utiliza el objeto *ImageDataGenerator* de la librería *keras* (integrada en la librería *tensorflow*) para generar lotes de imágenes en tiempo real y aplicar transformaciones de aumento de datos a estas imágenes mientras se entrena el modelo. Este objeto genera dinámicamente las instancias de datos con los que se va a entrenar el modelo cogiendo de partida las imágenes originales y evita cargar todo el *dataset* en memoria.

Se ha empleado el algoritmo de optimización *Adam* para la implementación de la función de pérdida *categorical crossentropy* y la precisión como métrica de evaluación, que mide la tasa de predicciones acertadas del total realizadas.

También se aplican unos mecanismos de retroalimentación para reducir automáticamente el *learning rate* y detener el entrenamiento cuando la precisión de validación no mejore pasadas cinco o diez épocas, respectivamente, y para guardar el mejor modelo entrenado hasta el momento.

El modelo se ha entrenado a lo largo de cien *epochs* y la codificación de la arquitectura y del entrenamiento se presenta en el Código 10.22.

7.4.3.6. Validación del modelo

En la Ilustración 50 pueden verse los resultados del aprendizaje a lo largo del entrenamiento a través de dos gráficas. En la primera gráfica, de la precisión o *accuracy*, se observa cómo al principio el modelo aprende correctamente de manera rápida y poco a po-

co la convergencia se ralentiza. A partir de la época cuarenta no se produce una mejora sustancial en el modelo.

Si la precisión de validación es significativamente menor que la de entrenamiento, significa que el modelo se ha ajustado excesivamente a los datos de entrenamiento y no generaliza correctamente. El objetivo es que ambas se acerquen a uno.

En el caso de la pérdida o *loss*, aunque la pérdida de entrenamiento disminuya continuamente, si la de validación se estabiliza o aumenta se produce *overfit*. El objetivo es que ambas se acerquen a cero.

En este caso, ambas métricas evolucionan según lo esperado y sus valores en entrenamiento (color azul) y validación (color rojo) son similares, lo que significa que el modelo se comporta según lo esperado y resuelve adecuadamente la tarea para la que ha sido creado.

El gráfico de las métricas se ha generado con *MATLAB* [77], Código 10.23, y en la Tabla 12 se presentan los valores finales de las métricas de evaluación y la tasa de aprendizaje a la que se ha reducido automáticamente.

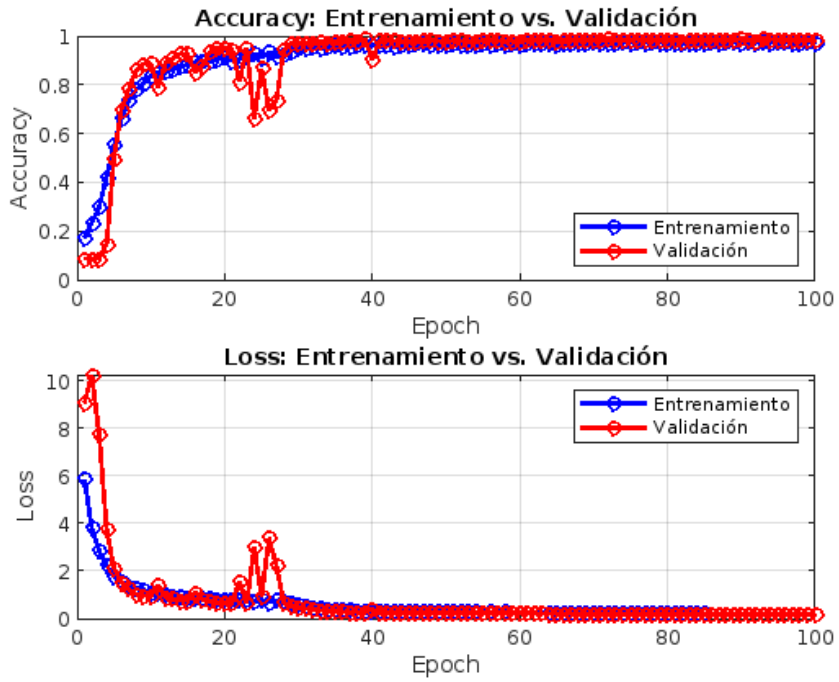


Ilustración 50: Precisión y pérdida del modelo durante el entrenamiento.

Métrica	Valor
Training Accuracy	0,9773
Training Loss	0,1976
Validation accuracy	0,9896
Validation loss	0,1514
Learning rate final	0,0001

Tabla 12: Resultados de las métricas de evaluación del entrenamiento del modelo *CNN* desarrollado.

7.4.3.7. Integración del modelo con la identificación de colores

Para integrar ambos procesos, siguiendo las especificaciones de la prueba, se ha establecido el orden de las operaciones.

En primer lugar, se realizará la detección de color y después, solo si se ha detectado una región de color, sobre esa región se identificará el número que esté visible.

Este orden garantiza que si se coloca un número sobre un color distractor, no será detectado y así obtener la puntuación máxima de la prueba.

La integración del código puede verse entre las líneas 207 y 249 del nodo *pertzepzio_proba_wrapper* en el Código 10.31.

7.4.4. Giros con *IMU*

El primer paso es colocar los servomotores en posición de "araña" para permitir al *rover* girar sobre sí mismo. Esta posición puede verse en la Ilustración 51.

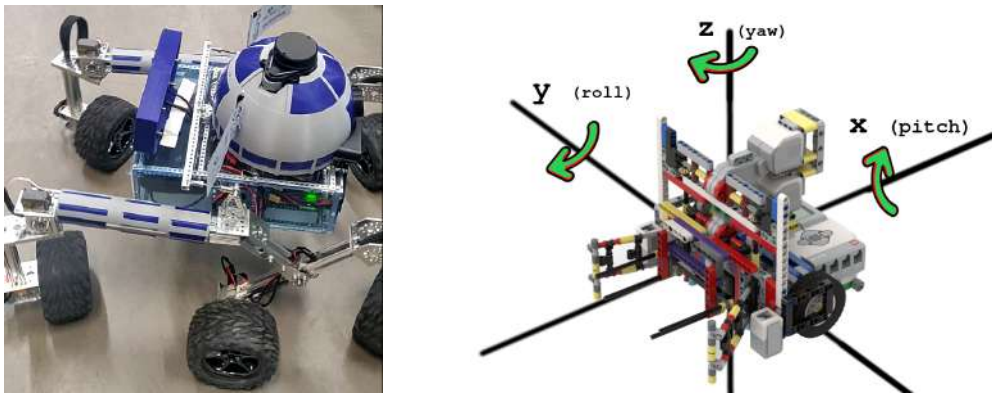


Ilustración 51: A la izquierda, los servomotores en posición para girar sobre sí mismo. A la derecha, la representación general de *roll*, *pitch* y *yaw* [39].

En segundo lugar, se realiza el giro hasta que el valor de la *IMU* sea el deseado. Los motores giran en un sentido u otro en función de la orientación del giro y se utilizan los ángulos de Euler, es decir, *roll*, *pitch* y *yaw*, proporcionados por el sensor.

La representación general de estos tres ángulos se puede ver en la Ilustración 51. El ángulo que permite girar sobre sí mismo es el *yaw*, habitualmente sobre el eje z, pero debido al posicionamiento del sensor en el robot se corresponde con el valor almacenado en el eje x del sensor.

Se ha desarrollado un algoritmo de doble etapa que permite disminuir artificialmente la latencia del sensor cuando el robot está próximo a su objetivo con el fin de aumentar la precisión del giro:

- **Primera etapa**: mientras no se haya superado el 80 % de los grados a girar, es decir $0,8 \cdot 360° \cdot$ número de vueltas a girar:
  - Leer *IMU* y mover el *rover* a velocidad rápida.
  - Si se ha dado una vuelta, contabilizarla.
  - Actualizar grados girados.
- **Segunda etapa**: mientras no se haya superado el 100 % de los grados a girar, es decir, $360° \cdot$ número de vueltas a girar:
  - Leer *IMU* y mover el *rover* a velocidad lenta.
  - Si se ha dado una vuelta, contabilizarla.
  - Actualizar grados girados.

En la línea 74 y entre la 280 y 364 del nodo *pertzepzio_proba_wrapper* en el Código 10.31 se puede ver la implementación de este algoritmo como una acción de *ROS2*. La solicitud de la acción se puede ver en la línea 97 del nodo *pertzepzio_proba* en el Código 10.33.

7.5. Prueba de control

Siguiendo el diagrama de secuencia propuesto en 6.6.2, los pasos a seguir son: obtener la distancia a las paredes del entorno y generar una respuesta en la dirección del movimiento. En la Ilustración 52 se muestran los nodos de *ROS2* desarrollados que desempeñan esta prueba.

El nodo lila de nivel medio *kontrol_proba_wrapper* se suscribe a los tópicos azules del *LiDAR*, motores y servomotores para recibir y publicar la información que gobierna esos dispositivos. Los nodos de nivel bajo se encargan de recibir y publicar dichos valores para controlar los sensores y actuadores.

En este apartado se describe el proceso realizado para el desarrollo de la prueba de control.

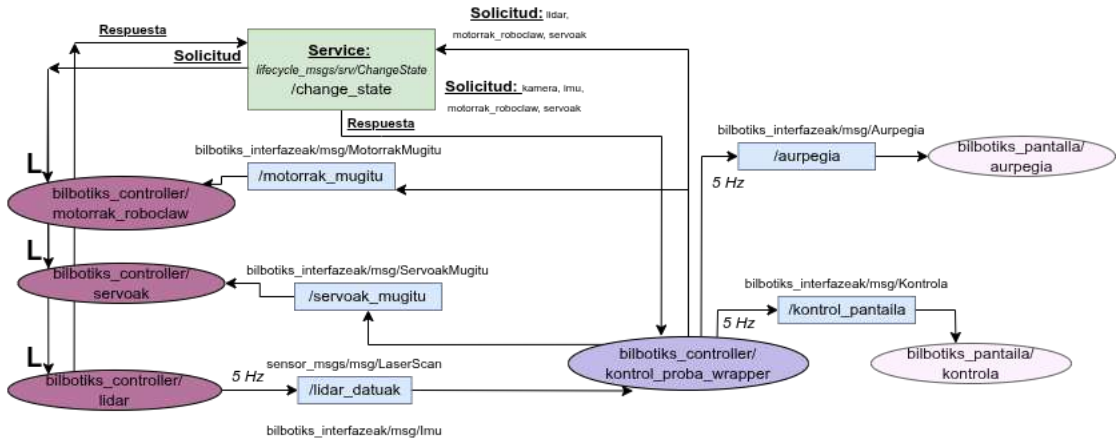


Ilustración 52: Nodos involucrados en la prueba de control.

7.5.1. Estrategia

La estrategia implementada consiste en obtener las distancias laterales a cada pared del pasillo, restarlas y obtener la diferencia entre ellas para conocer si el robot se encuentra a la izquierda o a la derecha del centro del pasillo.

Al restar la distancia derecha a la izquierda, si se obtiene un valor positivo el robot estará desplazado hacia la derecha, mientras que si es negativo lo estará hacia la izquierda.

Para convertir este error en una acción del robot se utiliza un controlador proporcional y derivativo, que obtiene el ángulo al que deben girar los servomotores de las patas delanteras para corregir la trayectoria.

7.5.2. Procesamiento del *LiDAR* y mensaje *LaserScan* en *ROS2*

El primer paso es obtener las mediciones del sensor que indican la distancia a cada pared. Se ha programado haciendo uso del *SDK* y la librería de *Python* del fabricante [78], y para comprobar el correcto funcionamiento del *LiDAR* se ha modificado el código de ejemplo proporcionado por el fabricante para imprimir por pantalla las mediciones, Código 10.24.

```
angle: 14.92187500613913  range:  0.0
angle: 8.062499584029268  range:  1.4859999418258667
angle: 8.375000375351215  range:  1.4880000352859497
angle: 8.671875401399465  range:  1.496999979019165
angle: 8.953124662174402  range:  1.503000020980835
angle: 9.26562459972232   range:  1.5080000162124634
angle: 9.562499625770567  range:  1.5180000066757202
angle: 9.859374651818822  range:  1.5269999504089355
angle: 10.156249677867072  range:  1.534000039100647
angle: 10.453124703915321  range:  1.5420000553131104
angle: 10.749999729963573  range:  1.5499999523162842
angle: 11.062499667511874  range:  1.5549999475479126
angle: 11.343749782060073  range:  1.562999963760376
angle: 11.640624808108324  range:  1.5700000524520874
angle: 19.156250782888534  range:  0.0
angle: 19.45312495436314  range:  0.0
angle: 12.296875359978673  range:  2.3250000476837158
angle: 12.609375297526974  range:  2.3289999961853027
angle: 12.906250323575224  range:  2.3450000286102295
angle: 13.187499584349778  range:  2.361999988555908
angle: 13.500000375571725  range:  2.372999906539917
angle: 13.796875401719975  range:  2.3889999389648438
angle: 14.093749573994579  range:  2.4019999504089355
angle: 14.406250365316527  range:  2.4149999618530273
angle: 14.703124537591131  range:  2.434999942779541
angle: 15.000000417413029  range:  2.444000005722046
angle: 15.296874589687633  range:  2.4590001106262207
angle: 15.609374527235932  range:  2.4760000705718994
angle: 15.90625048785783  range:  2.490000009536743
angle: 16.18750052160683  range:  2.506000041961670
angle: 16.500000459154332  range:  2.5220000743865967
angle: 16.796874631428935  range:  2.5380001068115234
angle: 17.093750511250832  range:  2.552999973297119
angle: 17.406250448879913  range:  2.572999954223633
angle: 17.68750056334733  range:  2.5880000591278076
angle: 17.984374735521937  range:  2.6050000190734863
angle: 18.296874673170237  range:  2.624000072479248
angle: 18.593750552992134  range:  2.6400001049041748
angle: 18.89062472526674  range:  2.6600000858306885
angle: 19.187500685888637  range:  2.6760001182556152
angle: 19.48437477736324  range:  2.694000005722046
```

Ilustración 53: Mediciones del *LiDAR* desordenadas.

En la Ilustración 53 se observa que las mediciones del sensor aparecen desordenadas respecto del ángulo al que corresponden, lo que supone un problema.

Para solventarlo, se ha aplicado una ordenación con la librería *numpy* como se muestra en la línea 144 en el Código 10.29.

Una vez los datos está ordenados, se convierten al formato del tipo de mensaje *LaserScan* de *ROS2* [79]. Los campos principales son el ángulo mínimo, máximo, el tiempo de escaneo y el vector de las distancias. Con esos datos, este mensaje asocia un valor angular a cada elemento del vector de distancias, de ahí la importancia de estar ordenados.

Asimismo, debido a la colocación del sensor en el robot como se observa en la Ilustración 54, el ángulo correspondiente al 0 del *LiDAR* está desplazado 127 grados.

La parte izquierda del sensor se utiliza para obtener el entorno situado a la derecha del *rover*, y la parte derecha para obtener el entorno situado a la izquierda.

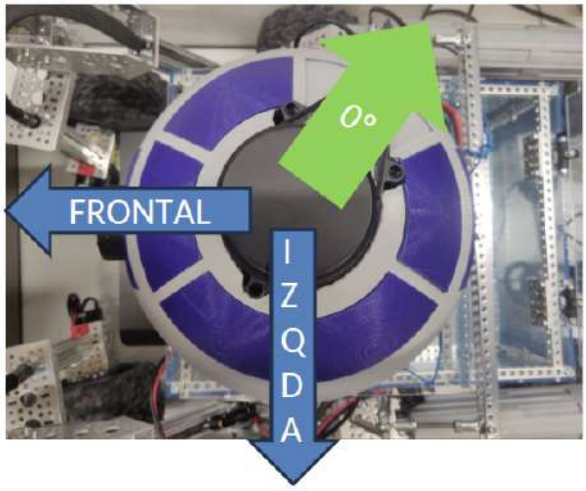


Ilustración 54: Posicionamiento del *LiDAR* en el *rover*.

Finalmente, para recoger las distancias a ambas paredes se ha seleccionado el valor mínimo a cada lado, dentro de un rango de amplitud, Ilustración 55. Esto permite que, aunque el *rover* esté girado, siempre se obtenga la distancia mínima real, puesto que leer un único punto fijo no siempre apuntará a la pared.

La propuesta inicial fue dotar de un campo de visión de 180 grados a cada lado. Sin embargo, en ocasiones, la distancia mínima correspondía a un punto de la pared ya superado por el robot, generando inestabilidad en la corrección de la dirección.

Para corregirlo, se redujo el campo de visión a 90 grados, y finalmente, con el objetivo de anticipar los movimientos correctos para el recorrido del pasillo, se redujo a 22,5 grados como se muestra en la Ilustración 56. Este nuevo campo de visión permite recibir información del entorno que el robot va a enfrentar en un futuro inmediato.

Teniendo en cuenta el desplazamiento del sensor, que sigue la regla de la mano izquierda y que clasifica las distancias en los ángulos del rango [-180, 180], los ángulos correspondientes para formar el campo de visión son, en grados:

- A la izquierda: [157,5, 180]
- A la derecha: [-90, -67,5]

Los resultados obtenidos con cada campo de visión se presentan a lo largo de los apartados 7.5.4 y 7.5.5.

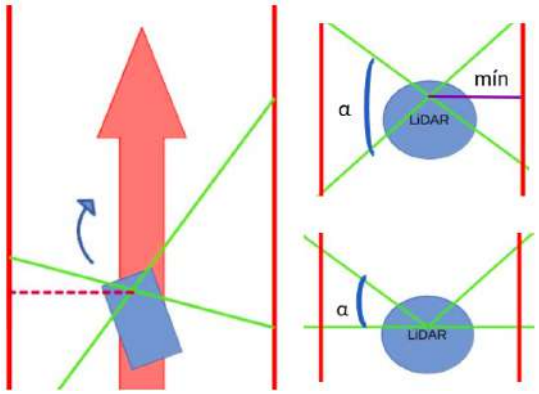


Ilustración 55: Campo de visión inicialmente propuesto.

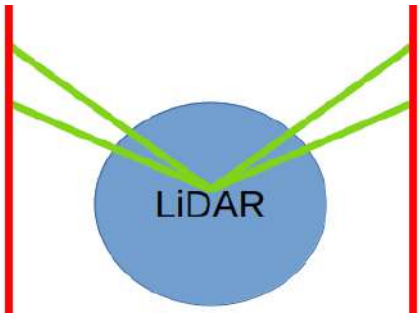


Ilustración 56: Campo de visión implementado.

7.5.3. Movimiento de los servomotores

Se han obtenido las posiciones límite de cada servomotor, Tabla 13, y definido la siguiente interpolación lineal para calcular la posición de cada servomotor delantero dado el ángulo de giro comprendido en el rango [-90, 90], en grados sexagesimales:

$$\text{ángulo servomotor} = \text{máx derecho} - \left(\frac{\text{ángulo de giro}}{90}\right) \cdot (\text{máx derecho} - \text{mín derecho})$$

Servomotor	Izquierda	Derecha	Recto
Trasero derecho - 0	Fijo		15
Delantero derecho - 1	[160, 140)	(140, 120]	140
Delantero izquierdo - 2	(25, 50]	[0, 25)	25
Trasero izquierdo - 3	Fijo		145

Tabla 13: Ángulos límite de actuación de los servomotores.

Por ejemplo, si queremos girar -45, es decir, a la izquierda, el segundo servomotor se colocará en la posición de 150 grados:

$$140 - \left(\frac{-45}{90}\right) \cdot (140 - 120) = 150$$

La implementación de la interpolación está entre las líneas 119 y 135 del nodo *kontrol_proba_wrapper* en el Código 10.32.

7.5.4. Control P: proporcional

Un controlador proporcional es un sistema de control que genera a la salida una señal de control que es proporcional a la señal del error de entrada [80]. En otras palabras, este controlador consiste en convertir el error en una acción de corrección en la magnitud correspondiente.

Se caracteriza por el parámetro *Kp*, que es el factor de corrección o de conversión que multiplica al error para generar la salida. Cuanto mayor sea su valor, más rápida o abrupta será la corrección. No existen reglas concretas para determinar el valor de este factor puesto que las características de cada sistema son diferentes, pero es posible realizar un análisis para conseguir una primera aproximación.

De las especificaciones de la prueba 4.2.2, se sabe que la anchura máxima del laberinto es de 1,5 metros. Por tanto, el error máximo de posición respecto del centro del pasillo es de metro y medio menos la mitad de la anchura del robot como se muestra en la Ilustración 57, pero que por simplificar se considera metro y medio.

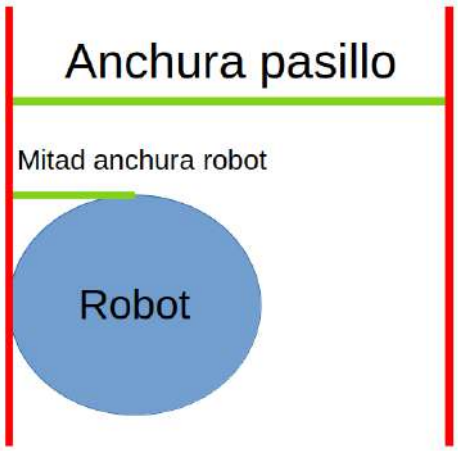


Ilustración 57: Representación gráfica del error máximo.

Además, en el apartado 7.5.3 se ha relacionado la corrección máxima con un giro de 90 grados y que tiene que aplicarse cuando el error sea máximo, por lo que el factor de corrección será, de al menos, 60:

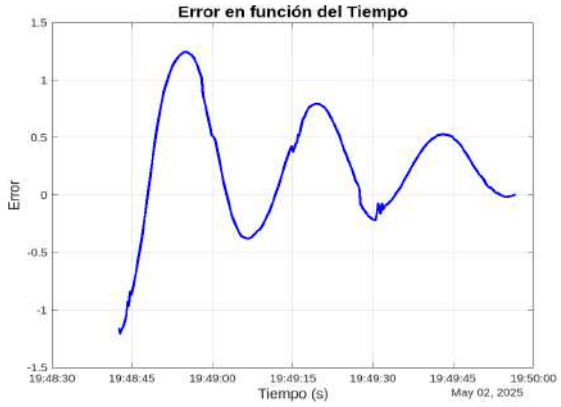
$$Kp_{min} = \frac{90}{1,5} = 60$$

Partiendo de ese valor, se han realizado diferentes pruebas con diferentes valores hasta conseguir el mejor resultado con $Kp = 150$ y $velocidad = 30$.

En la Ilustración 58 se presentan los gráficos de la señal de salida generada durante estas pruebas. En la imagen (d) se obtiene el mejor resultado, donde se observa que a pesar de la oscilación, el error en metros se mantiene cercano a cero, lo que indica que las correcciones en la trayectoria son mínimas. La imagen (e) presenta una oscilación similar,

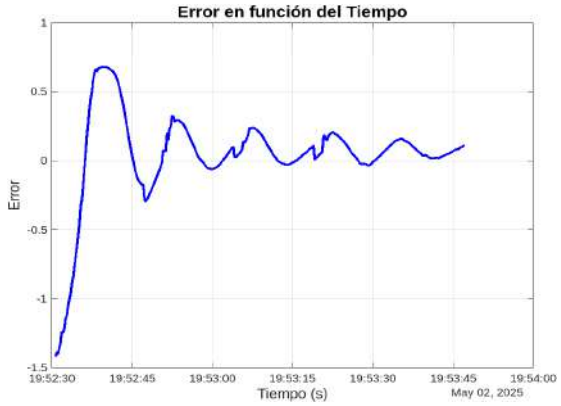
pero con correcciones más abruptas ya que se observan picos.

Estos gráficos se han realizado con *MATLAB* [77] utilizando el Código 10.25 y los valores se obtienen entre las líneas 200 y 215 del nodo *kontol_proba_wrapper* en el Código 10.32.



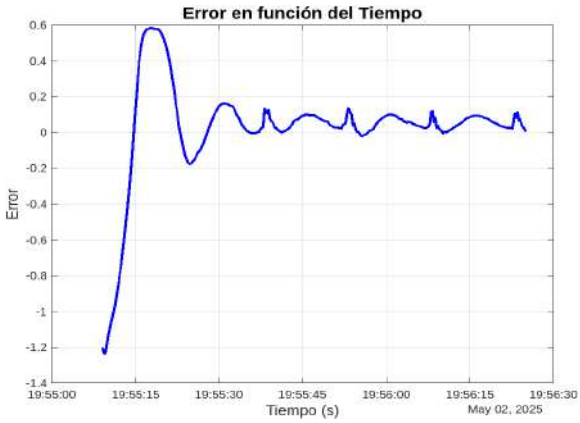
(a) Kp = 60, velocidad = 20

(b) Kp = 100, velocidad = 20



(c) Kp = 150, velocidad = 20

(d) Kp = 150, velocidad = 30



(e) Kp = 200, velocidad = 20

Ilustración 58: Señal de control de salida del controlador proporcional con diferentes valores *Kp*.

7.5.5. Control PD: proporcional y derivativo

Un controlador PD es una mejora del proporcional, ya que se le añade una etapa derivativa. La acción derivativa aumenta la estabilidad relativa del sistema obteniendo así una respuesta con menor sobreimpulso y permite anticipar la respuesta [80].

La etapa derivativa se caracteriza por el factor derivativo *Kd* y la derivada del error en función del tiempo, que corresponde a la diferencia entre el error actual y anterior.

Por tanto, la expresión PD a implementar es:

$$(Kp \cdot error(t)) + (Kd \cdot \frac{\mathrm{d}}{\mathrm{d}t} error(t))$$

es decir:

$$(Kp \cdot \text{error actual}) + [Kd \cdot (\text{error actual} - \text{error anterior})]$$

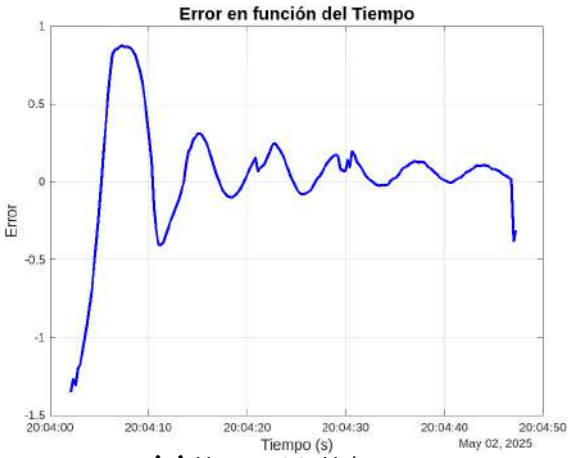
El sistema desarrollado es un sistema de control de lazo cerrado puesto que existe retroalimentación continua entre la entrada, que es la posición del robot respecto del pasillo, y la salida, que es la señal de control.

Por tanto, partiendo del valor $Kp = 150$ del apartado 7.5.4 con un tiempo de ciclo de cuatro segundos, se ha aplicado el método de sintonización por la ganancia crítica en lazo cerrado de *Ziegler-Nichols* [81] y obtenido los valores $Kp = 120$ y $Kd = 45$.

Tras aplicar esos valores y observar la señal de control generada, imagen (a) para un campo de visión de 90 grados e imagen (b) para un campo de visión de 22,5 grados hacia adelante, de la Ilustración 59, se han modificado los parámetros mediante prueba y error aplicando las siguientes reglas:

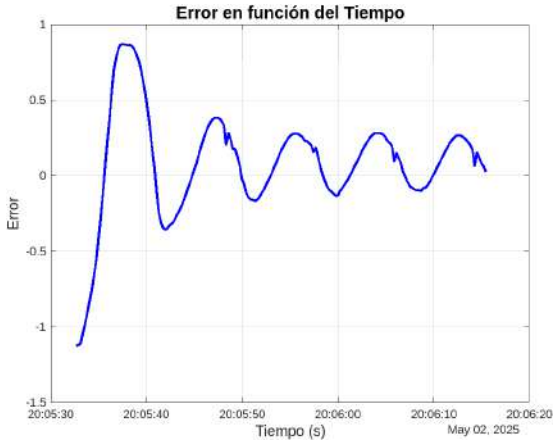
- Para reducir la brusquedad de la acción se reduce la corrección proporcional.
- Para aumentar la estabilidad se aumenta la corrección derivativa, por lo que se necesita menos corrección proporcional ya que se introduce mayor estabilidad y hay que reducir la brusquedad de la acción.
- Un valor de *Kd* demasiado alto aumenta la sensibilidad a los pequeños cambios en el entorno y por tanto, contrarresta la estabilización introducida. Este fenómeno se observa en la imagen (e) de la Ilustración 59.

El mejor resultado obtenido ha sido con $Kp = 100$ y $Kd = 25$, imágenes (c) y (d) de la Ilustración 59. Al reducir el campo de visión, en ambas imágenes se observa claramente cómo se reduce el error en metros, y por tanto, la imagen (d) presenta la mejor solución obtenida, ya que la imagen (e) solo presenta errores en un sentido (error siempre positivo).



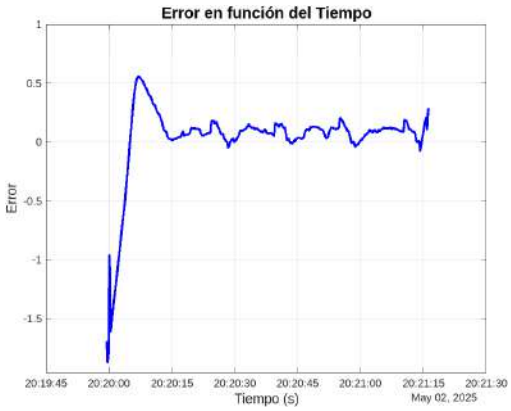
(a) Kp = 120, Kd = 45, velocidad = 30, campo visión = 90º

(b) Kp = 120, Kd = 45, velocidad = 30, campo visión = 22.5º



(c) Kp = 100, Kd = 25, velocidad = 30, campo visión = 90º

(d) Kp = 100, Kd = 25, velocidad = 30, campo visión = 22.5º



(e) Kp = 100, Kd = 45, velocidad = 30, campo visión = 22.5º

Ilustración 59: Señal de control de salida del controlador PD con diferentes valores *Kp* y *Kd*.

7.6. Interfaces gráficas desarrolladas

Con el objetivo de monitorizar visualmente y en tiempo real la realización de ambas pruebas, se han diseñado dos interfaces gráficas.

La pantalla de la prueba de percepción, Ilustración 60, muestra el valor actual del sensor *IMU*, la imagen capturada por la cámara y el número total de vueltas a dar.

La pantalla de la prueba de control, Ilustración 61, muestra la distancia izquierda, frontal y derecha obtenidas por el *LiDAR* mientras el robot recorre el laberinto.

Adicionalmente y con fines estéticos, se ha desarrollado una pantalla que muestra una cara dinámica que mira a los lados según el robot gira. Esta pantalla puede verse en la Ilustración 62.

Todas las pantallas forman parte del paquete *bilbotiks_pantaila*, se controlan mediante tópicos de *ROS2* y sus implementaciones pueden verse en el Código 10.34, 10.35 y 10.36, respectivamente.

Esta implementación mediante tópicos tiene la ventaja de que un error en el sensor no genera un error en las pantallas, ya que si no reciben datos a través del tópico se mostrará un cero. Por ejemplo, si un sensor está dando problemas para arrancar, la pantalla se ejecuta sin dificultades y una vez solventado la pantalla se sincroniza automáticamente sin necesidad de reiniciar el proceso.

Ilustración 60: Interfaz gráfica para la prueba de percepción.

Ilustración 61: Interfaz gráfica para la prueba de control.

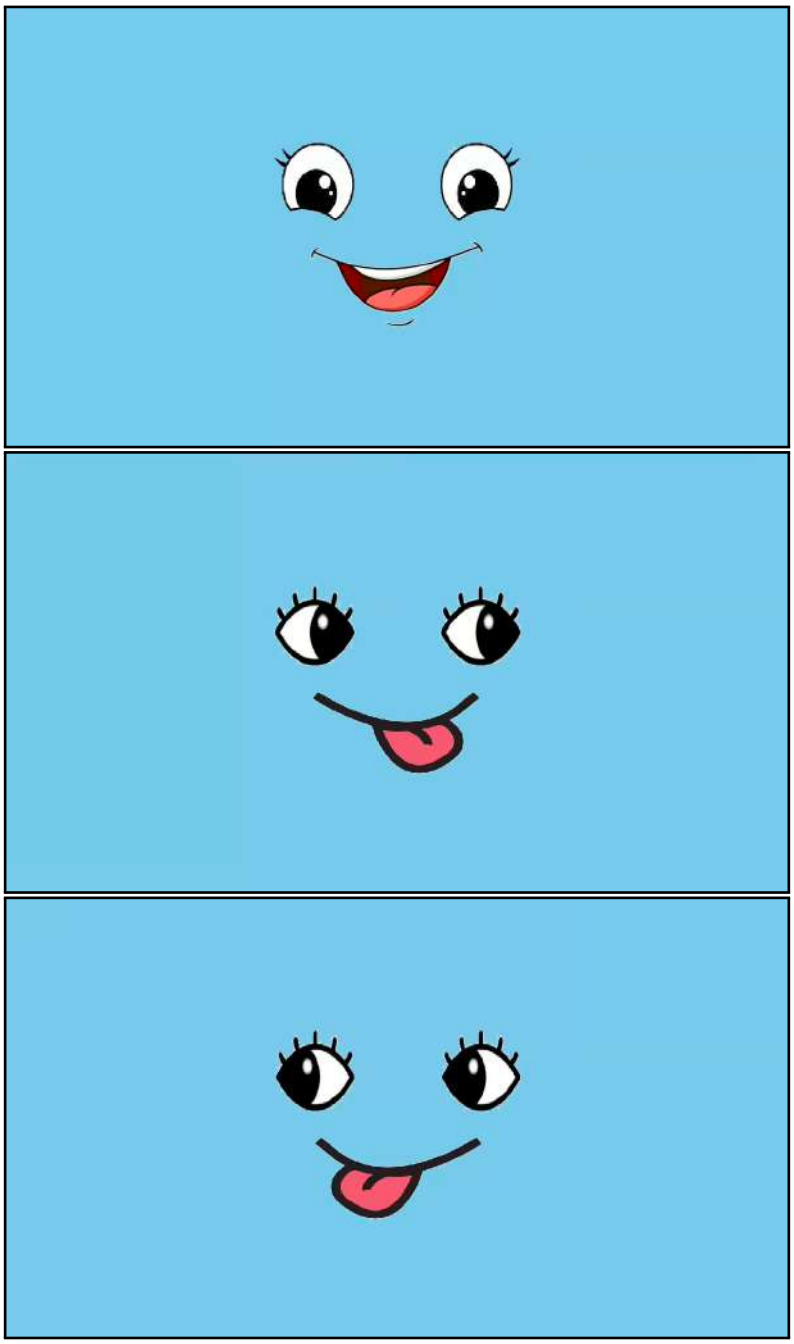


Ilustración 62: Interfaz gráfica de la cara.

7.7. Gemelo funcional

Para el desarrollo de la prueba de percepción se ha desarrollado un nodo simulado de la *IMU* que actúa como gemelo funcional. Este nodo permite ejecutar el código de la prueba sin necesidad de utilizar el robot, con el objetivo de detectar fallos en la codificación y algorítmica.

Esta simulación publica los datos en el tópico correspondiente con la misma frecuencia que el sensor, y estas mediciones se generan aleatoriamente utilizando las librerías *random* y *math*. Todos los valores se generan uniformemente en el rango del valor real. Se muestra un ejemplo en el Código 7.14.

```
random.uniform(0.0, 360.0)
```

Código 7.14: Generar valores distribuidos uniformemente en *Python*.

Para arrancar la simulación con este gemelo funcional es suficiente con sustituir el arranque del nodo de la *IMU* por este nuevo y ajustar el índice de la cámara para que corresponda con la cámara con la que se va realizar la simulación.

Este nodo está en el Código 10.37 y en la Ilustración 63 puede verse un ejemplo de los valores generados.

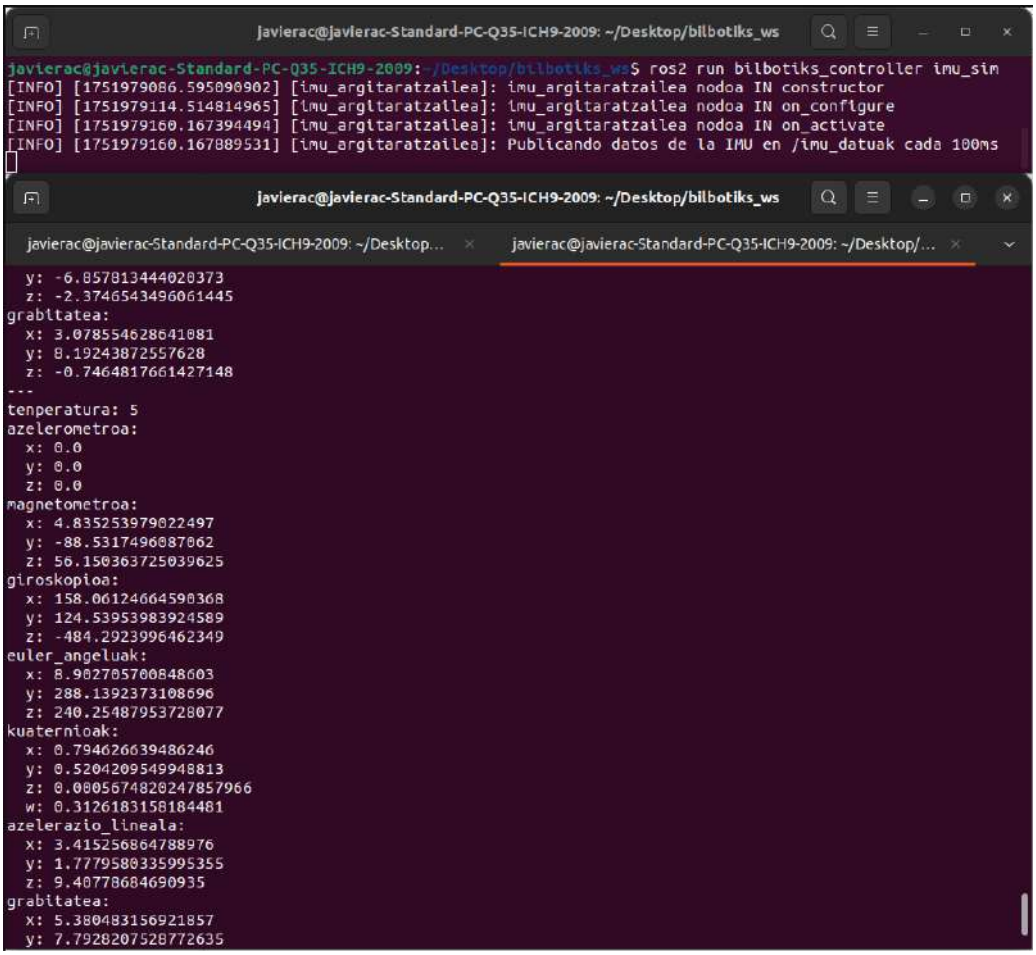


Ilustración 63: Valores de la *IMU* generados por el gemelo funcional.

7.8. Simulación con *Gazebo Harmonic* y *Docker*

Por otra parte y con fines de investigación, se ha generado un entorno virtual simple para simular la estrategia de la prueba de control. Se ha utilizado el simulador *Gazebo Harmonic* [82] de *ROS2 Jazzy Jalisco* [63] sobre la herramienta de virtualización por contenedores *Docker* [83].

Para ello, se ha descargado la imagen *Docker* oficial de *ROS2 Jazzy Jalisco*, proporcionada por *OSRF*, y se ha instalado el simulador para generar la imagen utilizada. Esta imagen está disponible en el repositorio del proyecto, apartado 10.3.

Siguiendo la documentación oficial, se ha generado el modelo de un robot diferencial, dos paredes laterales e implementado el seguidor proporcional con dos estrategias diferentes: obtener un único punto en cada dirección y la descrita en los apartados 7.5.1 y 7.5.2.

El escenario con el robot se define en un fichero *.sdf* con los *plugin* correspondientes, en donde se establece el nombre de los tópicos a través de los que se realiza la comunicación con la simulación. En el Código 7.15 se muestra la definición del *LiDAR* virtual con sus características personalizables y en la Tabla 14 la descripción de las más relevantes.

La codificación del algoritmo se ha realizado en *C++*, y la definición completa del mundo virtual como el nodo desarrollado se encuentran en los códigos 10.38 y 10.39.

En la Ilustración 64 se presentan capturas de pantalla de la simulación en *Gazebo Harmonic*, donde se observa cómo el robot se mantiene centrado entre ambas paredes.

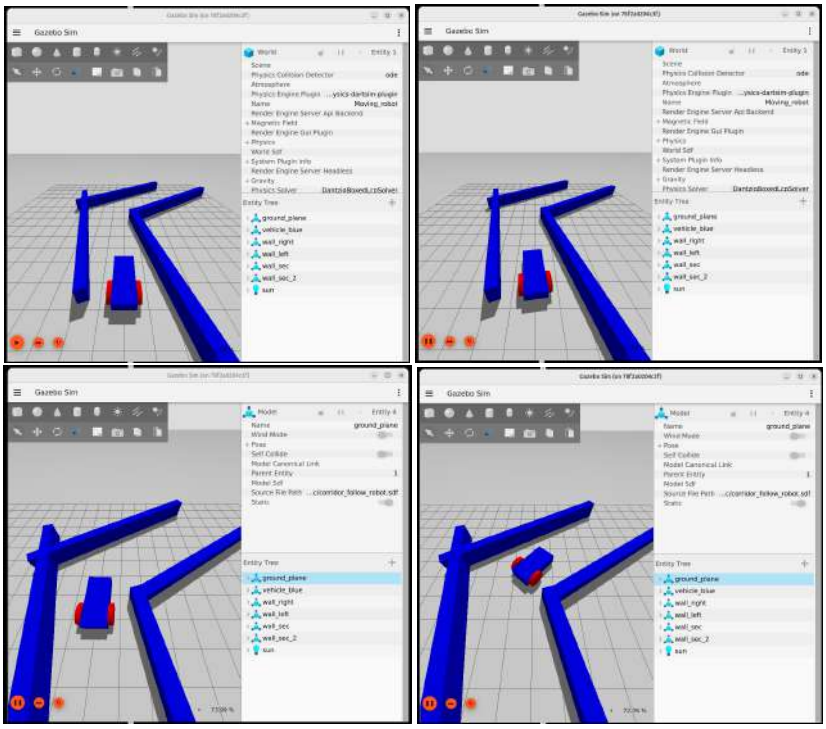


Ilustración 64: Capturas de pantalla de la simulación de la prueba de control en *Gazebo Harmonic*.

```
<sensor name='gpu_lidar' type='gpu_lidar'>"
    <pose relative_to='lidar_frame'>0 0 0 0 0 0</pose>
    <topic>lidar</topic>
    <update_rate>10</update_rate>
    <ray>
        <scan>
            <horizontal>
                <samples>640</samples>
                <resolution>1</resolution>
                <min_angle>-1.5708</min_angle>
                <max_angle>1.5708</max_angle>
            </horizontal>
            ...
        </scan>
        <range>
            <min>0.08</min>
            <max>10.0</max>
            <resolution>0.01</resolution>
        </range>
    </ray>
    <always_on>1</always_on>
    <visualize>true</visualize>
</sensor>
```

Código 7.15: Definición del *LiDAR* virtual en el fichero *.sdf*.

Característica	Descripción
samples	Número de muestras que se obtienen, en este caso, seiscientas cuarenta (640).
min_angle, max_angle	Límites del campo de visión en radianes, siendo cero el frente y siguiendo la regla de la mano derecha. La primera muestra, con índice cero 0, corresponde al ángulo mínimo, -1.5708 radianes o -90 grados, y la última, con índice 639, al máximo, es decir, 1.5708 radianes o 90 grados. Por tanto, el campo de visión es de 180 grados y sabiendo que hay 640 muestras, la diferencia angular entre cada muestra es de $\frac{180}{640} = 0,28125$ grados.
range min, max	Distancia mínima y máxima en metros a la que puede medir el sensor: 0,08 y 10 metros, respectivamente.
range resolution	La resolución espacial es la mínima diferencia de distancia que el sensor puede distinguir entre dos objetos situados a diferentes distancias del sensor. Por ejemplo, con una resolución de 0,01 metros podrá diferenciar dos objetos que estén separados al menos por esa distancia.

Tabla 14: Descripción de las características del *LiDAR* virtual.

8. Verificación y evaluación

En este capítulo se explican y detallan las pruebas realizadas sobre los desarrollos y los resultados obtenidos. Se han realizado tres (3) tipos de test por cada prueba del concurso: pruebas unitarias, pruebas de integración y pruebas de validación.

Las pruebas unitarias permiten validar el funcionamiento individual de cada uno de los procedimientos a realizar, las pruebas de integración verificar la correcta coordinación y sincronización de los procedimientos y las pruebas de verificación y validación para constatar que el desarrollo cumple tanto con los requisitos funcionales y no funcionales como de las partes interesadas. En otras palabras, las pruebas de verificación responden a la pregunta "¿el sistema ha sido construido correctamente?" y las de validación a "¿el sistema correcto ha sido construido?" [59].

En este proyecto las pruebas de integración ratifican el correcto funcionamiento de los nodos de nivel medio y alto, ya que estos invocan a todos los procedimientos verificados mediante las pruebas unitarias, y las pruebas de verificación y validación son las mismas entre sí, puesto que los requisitos de las partes interesadas son los mismos que las especificaciones impuestas por el concurso.

En las siguientes secciones se organizan las tablas que recogen las pruebas realizadas. No se incluyen las pruebas para las interfaces gráficas.

8.1. Prueba de percepción

8.1.1. Pruebas unitarias

Procedim.	Descripción	Resultado	Solución
publicar_imu()	Recibe los datos de la *IMU* utilizando la librería del fabricante y los publica en un tópico. Los datos no tienen que repetirse	Desfavorable. Los mismos valores se publican dos (2) veces porque la frecuencia de la publicación es mayor que la frecuencia a la que trabaja el sensor.	Corregir la frecuencia de la publicación de los datos.

Procedim.	Descripción	Resultado	Solución
imu_callback()	Recibe los datos de la *IMU* a través de un tópico y los almacena en una variable de la clase. Los datos son accesibles desde dicha variable y no tienen que repetirse.	Desfavorable. Los mismos valores se publican dos veces porque la frecuencia de la publicación es mayor que la frecuencia a la que trabaja el sensor.	Corregir la frecuencia de la publicación de los datos.
sacar_foto()	Gestiona el nodo de la cámara y llama al servicio correspondiente para sacar una fotografía. Se debe comprobar que la imagen obtenida corresponde con la realidad.	Favorable.	-
giro360()	El robot tiene que colocar las ruedas en la posición correcta y girar con precisión el número de vueltas indicado. Para ello, gestiona los nodos de los motores y servomotores y los acciona.	Desfavorable. Cada rueda deberá moverse en una dirección correspondiente y una velocidad alta provoca una frenada brusca con falta de precisión.	Corregir la dirección del movimiento de cada motor e implementar el algoritmo de doble etapa del apartado 7.4.4.
ajustar_imagen()	Recortar y ampliar la imagen.	Favorable.	-
detectar_color()	Identificar correctamente el color.	Desfavorable.	Aplicar las restricciones geométricas del apartado 7.4.2.4.
detectar_numero()	Identificar correctamente el número sobre la región de color detectada.	Favorable.	-
adivinar_colNum()	Ajustar la imagen, detectar el color y el número.	Favorable.	-

Tabla 15: Test unitarios de los procedimientos de la prueba de percepción.

8.1.2. Pruebas de integración

Nodo	Tarea	Resultado
pertzepzio_proba	Solicitar y recibir en el formato adecuado el color y número identificado.	Favorable.
	Solicitar el giro del robot en función del color y número detectado.	
pertzepzio_proba_wrapper	Gestionar los nodos de la cámara, motores y servomotores.	Favorable.
	Recibir en tiempo real la *IMU*.	
	Ajustar la imagen obtenida, identificar el color y el número.	
	Girar sobre sí mismo.	

Tabla 16: Test de integración de la prueba de percepción.

8.1.3. Pruebas de verificación y validación

Requisito	Cumple
Identificar el color.	Sí.
Identificar el número.	Sí.
Girar con precisión y rapidez al detectar un color no distractor.	Sí.
No identificar el número en un color distractor.	Sí.
No girar con un color distractor.	Sí

Tabla 17: Test de verificación y validación de la prueba de percepción.

8.1.4. Resultados obtenidos

Se han desarrollado pruebas con todos los números posibles y con los colores rojo, azul, amarillo, verde y negro, incluyendo todas las combinaciones. Se han realizado en dos entornos diferentes, Ilustración 65, y se han superado con la máxima puntuación.

El robot tarda de media 2,5 segundos en realizar la percepción del color y número y necesita 5 segundos por vuelta. El tramo final, a velocidad lenta, requiere 4 segundos. Por tanto, en el caso del mayor número de vueltas, nueve, el robot necesita aproximadamente $2,5 + 5 \cdot 9 + 4 = 51,5$ segundos.

Ilustración 65: Escenario de pruebas para la prueba de percepción.

Estos resultados reflejan que la solución desarrollada es robusta y permite al *rover* realizar las tareas de percepción en un entorno con condiciones cambiantes, como por ejemplo, el material del suelo, ya que aunque las ruedas pudieran patinar, el uso de la *IMU* provoca no se detenga el movimiento hasta que el sensor lo determine.

8.2. Prueba de control

8.2.1. Pruebas unitarias

Procedim.	Descripción	Resultado	Solución
publicar_lidar()	Recibe los datos del *LiDAR* utilizando la librería del fabricante y los publica en un tópico. Los datos tienen que estar correctamente representados, es decir, ordenados por ángulo de captura.	Desfavorable. El sensor proporciona las mediciones desordenadas.	Añadir un preproceso de ordenación.
lidar_callback()	Recibe los datos del *LiDAR* y los almacena en una variable de la clase, desde donde son accesibles. Tienen que estar ordenados por ángulo de captura.	Desfavorable. El sensor proporciona las mediciones desordenadas.	Añadir un preproceso de ordenación.
calcular_servos()	Devuelve el ángulo de cada servomotor según la corrección en la dirección calculada.	Favorable.	-
calcular_dirección()	Gestiona adecuadamente la activación y desactivación de los nodos involucrados y mueve las ruedas y servomotores y calcula la dirección de corrección.	Favorable.	-

Tabla 18: Test unitarios de los procedimientos de la prueba de control.

Nodo	Tarea	Resultado
kontrol_proba_wrapper	Gestionar los nodos del *LiDAR*, motores y servomotores.	Favorable.
	Recibir en tiempo real los valores del *LiDAR*.	
	Generar la corrección en la trayectoria y girar los servomotores en tiempo real.	

Tabla 19: Test de integración de la prueba de control.

8.2.2. Pruebas de integración

8.2.3. Pruebas de verificación y validación

Requisito	Cumple
El robot se mantiene por el centro del pasillo y supera las líneas de control.	Sí.
El robot se mueve de manera fluida y a una velocidad alta.	Sí.
El robot anticipa correctamente los giros.	Sí.

Tabla 20: Test de verificación y validación de la prueba de control.

8.2.4. Resultados obtenidos

Se han desarrollado pruebas en diferentes escenarios, incluyendo trazados con una anchura inferior y comenzando en una posición con el máximo error posible, Ilustración 66. Se han superado con una alta puntuación y un tiempo bajo, ya que se ha conseguido aumentar la velocidad sin perder precisión. Al no conocer el tiempo de otros equipos y robots, no se puede verificar que la puntuación obtenida haya sido máxima.

El robot tarda de media un minuto y cinco segundos (1,05) en dar una vuelta a un circuito sobre una superficie de cincuenta (50) metros cuadrados, aunque este registro varía en función del trazado y de la superficie.

Se ha conseguido un controlador PD con parámetros ajustables para un *rover* con cuatro ruedas direccionables. Estos parámetros incluyen, además de las propias constantes de control, la velocidad y el campo de visión del *LiDAR*. Por tanto, esta solución se puede adaptar a escenarios con diferentes características, bien sea con distinta anchura o trazado.

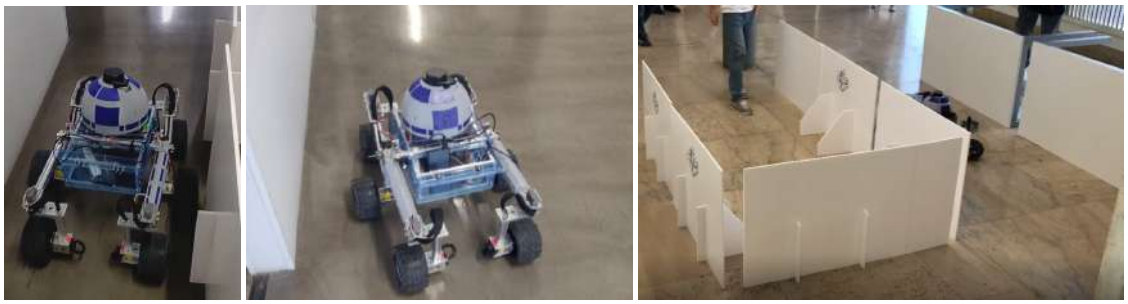


Ilustración 66: Escenarios de pruebas para la prueba de control.

9. Planificación

La realización de este trabajo ha combinado herramientas clásicas de la gestión de proyectos y metodologías ágiles del desarrollo de *software*, aprovechando las ventajas que nos aporta cada una de ellas.

Para la gestión del proyecto, así como la planificación temporal y el control del cumplimiento de los objetivos, se ha trabajado con un diagrama de *Gantt*. Este gráfico ha permitido observar las dependencias de las tareas a realizar entre sí.

9.1. Definición de tareas

Se ha decidido dividir el proyecto es seis fases diferenciadas: planificación, preparación, memoria, análisis, desarrollo y defensa. Cada una de estas etapas se compone de diversas tareas, identificadas en el diagrama EDT de la Ilustración 67. Estas tareas equivalen a 300 horas, lo que es equivalente a los 12 créditos del TFG.

En la Tabla 21 se resumen las tareas con su duración, precedencias, fecha de inicio y fecha final estimada. La descripción detallada de estas tareas se encuentra a partir de la Tabla 22.

Las tareas 2.1, 5.1 y 5.2 se han trabajado con una metodología *Scrum*. Esto se debe a que, por ejemplo, aunque no todo el *rover* esté operativo, se puede avanzar en el desarrollo de la solución para los dispositivos que sí lo estén. Por tanto, la realización de estas tareas no es estrictamente secuencial.

Tarea	Duración	Prec.	F. inicio	F. fin
PLANIFICACIÓN				
1.1. Inscripción concurso	1 hora	-	29/10	31/10
1.2. Calendario	1 hora	1.1	04/11	08/11
1.3. Objetivos y alcance	6 horas	1.1	04/11	27/11
1.3.1. Objetivos principales y secundarios		-	04/11	08/11
1.3.2. Requisitos y especificaciones		-	04/11	08/11
1.3.3. Identificación de tareas		-	04/11	13/11
1.3.4. Descripción de tareas		-	14/11	27/11
1.4. Metodología de planificación	2 horas	1.3.2	11/11	15/11
1.5. Planificación temporal	4 horas	1.4	18/11	29/11

Tarea	Duración	Prec.	F. inicio	F. fin
1.6. Riesgos	2 horas	1.3.1 1.3.2 1.3.3	14/11	20/11
1.7. Control y seguimiento	10 horas	1.6	21/11	20/06
PREPARACIÓN				
2.1. Puesta en marcha del rover	25 horas	-	14/11	13/12
2.1.1. Recepción de componentes		-	14/11	22/11
2.1.2 Montaje del hardware		-	25/11	13/12
2.1.3 Configuración software		-	02/12	13/12
MEMORIA				
3.1. Rúbrica de evaluación y normativa	1 hora	-	26/12	26/12
3.2. Definición de la estructura	2 horas	3.1	27/12	03/01
3.3. Redacción	60 horas	-	06/01	06/06
3.4. Revisión	10 horas	3.2	06/01	06/06
3.5. Aprobación	20 horas	3.2	19/05	20/06
ANÁLISIS				
4.1. Definición tarea a resolver	5 horas	1.3	27/01	06/02
4.1.1. Entorno de aplicación		-	27/01	29/01
4.1.2. Resultado perseguido		-	29/01	31/01
4.2. Estado del arte	5 horas	4.1	03/02	14/02
4.3. Diseño de la solución propuesta	20 horas	4.1	06/02	21/02
4.3.1. Arquitectura		-	06/02	14/02
4.3.2. Herramientas		-	17/02	21/02
DESARROLLO				
5.1. Verificación y validación	10 horas	4.3	24/02	25/04
5.2. Desarrollo	90 horas	4.3	24/02	25/04
5.2.1 Simulación		-	24/02	25/04
5.2.2 Entorno real		-	24/02	25/04
DEFENSA				
6.1. Rúbrica de evaluación y normativa	1 hora	-	09/06	13/06
6.2. Diapositivas	10 horas	3.5 5.2 6.1	23/06	27/06
6.3. Ensayos	5 horas	6.2	30/06	11/07
6.4. Defensa	2 horas	Todas	14/07	18/07
6.5. Concurso	8 horas	5.1,5.2	06/05	06/05

Tabla 21: Tareas, duración, precedencias y fecha estimada.

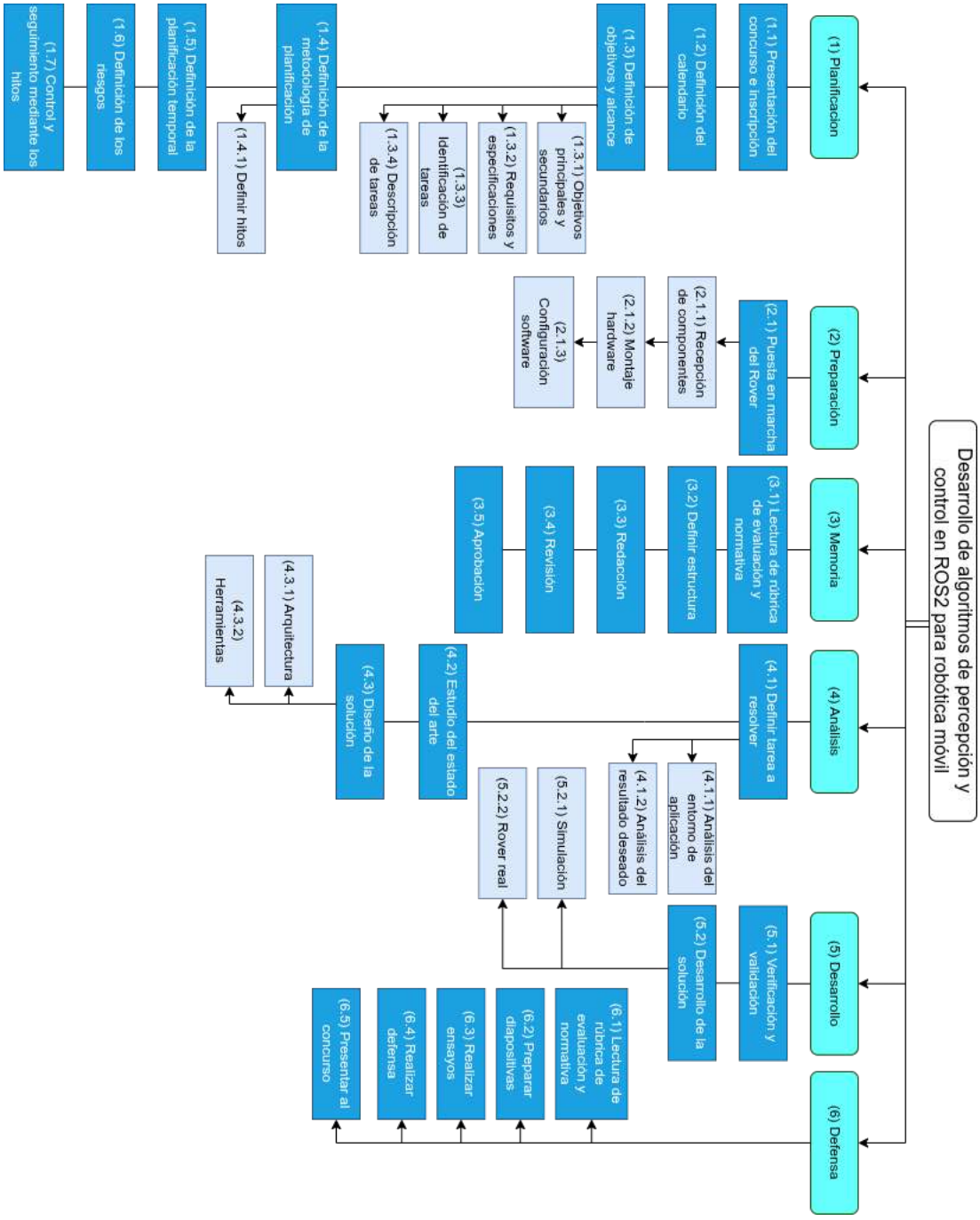


Ilustración 67: EDT.

T1.1. Presentación del concurso e inscripción

Descripción	Primera reunión con el resto de compañeros y compañeras para leer las bases, retos y objetivos del concurso y formalizar la inscripción, así como la elección de los portavoces del equipo.
Objetivos	Inscribirse en el concurso y conocer al equipo.
Entradas	Deseo de participar de los integrantes del equipo.
Salidas/Producto generado	Confirmación de la organización de la inscripción y reparto de roles.
Recursos	Correo electrónico.
Duración	1 hora.
Precedencias	-
Subtareas	-

Tabla 22: Detalles de la tarea 1.1.

T1.2. Definición del calendario

Descripción	Esta tarea consiste en decidir el calendario que va a seguir el equipo para dar respuesta a los retos del certamen.
Objetivos	Leer los plazos del concurso y definir los días y horarios de reuniones del equipo.
Entradas	Fechas de las pruebas intermedias eliminatorias (*checkpoints*) descritas en las bases del concurso.
Salidas/Producto generado	Confirmación de la organización de la inscripción y reparto de roles.
Recursos	Bases del concurso.
Duración	1 hora.
Precedencias	1.1. Presentación del concurso e inscripción.
Subtareas	-

Tabla 23: Detalles de la tarea 1.2.

T1.3. Definición de objetivos y alcance

Descripción	Valorar la dificultad de las pruebas del concurso para conocer cuáles son las más adecuadas para realizar este trabajo.
Objetivos	Determinar qué retos del concurso se van a desarrollar en este trabajo, tareas e hitos.
Entradas	Descripción de las pruebas del concurso.
Salidas/Producto generado	Documento de Objetivos del Proyecto (DOP).
Recursos	Bases del concurso.
Duración	6 horas.
Precedencias	1.1. Presentación del concurso e inscripción.
Subtareas	1.3.1. Objetivos principales y secundarios. 1.3.2. Requisitos y especificaciones. 1.3.3. Identificación de tareas. 1.3.4. Descripción de tareas.

Tabla 24: Detalles de la tarea 1.3.

T1.4. Definición de la metodología de planificación

Descripción	Se analizan las tareas a desempeñar y la planificación más adecuada, bien sea mediante diagramas *Gantt* o metodologías ágiles extendidas en el campo del desarrollo de software.
Objetivos	Decidir la estrategia de planificación.
Entradas	Descripción de tareas y Documento de Objetivos del Proyecto (DOP).
Salidas/Producto generado	Estrategia de planificación.
Recursos	Conocimientos adquiridos en las asignatura de Ingeniería del Software y Gestión de Proyectos del grado.
Duración	2 horas.
Precedencias	1.3.2 Requisitos y especificaciones.
Subtareas	1.4.1. Definir hitos.

Tabla 25: Detalles de la tarea 1.4.

T1.5. Definición de la planificación temporal

Descripción	Se profundiza y detalla cómo se va a organizar el desarrollo del trabajo dentro del marco temporal.
Objetivos	Definir y desarrollar la planificación temporal.
Entradas	Estrategia de planificación elegida.
Salidas/Producto generado	Elementos de la estrategia de planificación escogida: diagrama de *Gantt*, historias de usuario, hitos...
Recursos	Programa informático *The Gantt Project* [84].
Duración	4 horas.
Precedencias	1.4. Definición de la metodología de planificación.
Subtareas	-

Tabla 26: Detalles de la tarea 1.5.

T1.6. Definición de los riesgos

Descripción	Esta tarea consiste en identificar y evaluar los riesgos asociados al proyecto que dificulten conseguir el resultado esperado dentro del plazo establecido. Se establece un plan de prevención y un plan de contingencia.
Objetivos	Minimizar el impacto negativo de imprevistos o dificultades que aparezcan durante el desarrollo y estar preparados para afrontarlos eficazmente.
Entradas	Descripción de tareas y Documento de Objetivos del Proyecto (DOP).
Salidas/Producto generado	Evaluación de riesgos.
Recursos	Conocimientos adquiridos en la asignatura Sistemas de Gestión Integrada del grado.
Duración	2 horas.
Precedencias	1.3.1. Objetivos principales y secundarios. 1.3.2. Requisitos y especificaciones. 1.3.3. Identificación de tareas.
Subtareas	-

Tabla 27: Detalles de la tarea 1.6.

T1.7. Control y seguimiento mediante los hitos

Descripción	Realizar un seguimiento de los hitos y objetivos del proyecto para comprobar que se están logrando según lo establecido y sin anomalías. Este paquete de trabajo permite auditar el proceso del proyecto para garantizar que se cumple con lo establecido.
Objetivos	Garantizar que el resultado final cumple con las expectativas iniciales.
Entradas	Descripción de tareas y Documento de Objetivos del Proyecto (DOP).
Salidas/Producto generado	Estrategia de planificación.
Recursos	-
Duración	10 horas.
Precedencias	1.6. Definición de los riesgos.
Subtareas	-

Tabla 28: Detalles de la tarea 1.7.

T2.1. Puesta en marcha del rover

Descripción	Montar los componentes, alimentarlos, configurarlos y accionarlos por software siguiendo las indicaciones y herramientas del fabricante. Por ejemplo, se configuran el puerto serie de los motores de las ruedas y el puerto de la *IMU*. Esta tarea no incluye el diseño ni la impresión del soporte de la cámara y el sensor *LiDAR*, pues se considera fuera del alcance del proyecto al no ser estrictamente necesario.
Objetivos	Preparar el rover para comenzar el desarrollo de los algoritmos, garantizando el correcto funcionamiento de los dispositivos necesarios y familiarizarse con ellos.
Entradas	Rover y dispositivos por separado.
Salidas/Producto generado	Rover montado y dispositivos conectados, configurados y operativos.
Recursos	Materiales proporcionados por el concurso, Capítulo 5, y documentación oficial y de desarrolladores.
Duración	25 horas.
Precedencias	-
Subtareas	2.1.1. Recepción de componentes 2.1.2. Montaje hardware 2.1.3. Configuración software

Tabla 29: Detalles de la tarea 2.1.

T3.1. Lectura de rúbrica de evaluación y normativa

Descripción	Revisar detalladamente los documentos oficiales proporcionados por la universidad relacionados con los criterios de evaluación del TFG y las normas de redacción y presentación de la memoria.
Objetivos	Comprender los criterios con los que será evaluado el TFG y conocer la normativa con el fin de alinear el desarrollo del trabajo con dichas directrices.
Entradas	Normativa y rúbrica de evaluación del TFG proporcionada por la Escuela de Ingeniería de Bilbao. [85].
Salidas/Producto generado	-
Recursos	-
Duración	1 hora.
Precedencias	-
Subtareas	-

Tabla 30: Detalles de la tarea 3.1.

T3.2. Definir estructura

Descripción	Elaboración de un esquema preliminar que define la estructura de la memoria del TFG, considerando la normativa vigente y las recomendaciones de la tutora y del tutor. Esta estructura servirá como guía para la redacción de la memoria final.
Objetivos	Obtener una plantilla base sobre la que comenzar la redacción de la memoria.
Entradas	Normativa y rúbrica de evaluación del TFG proporcionada por la Escuela de Ingeniería de Bilbao.
Salidas/Producto generado	Plantilla de la memoria del TFG.
Recursos	Editor de textos "LaTeX" *Overleaf* [86] y entorno de desarrollo *Visual Studio Code* [87] con las extensiones "LaTeX" [88] y "LaTeX Workshop" [89].
Duración	2 horas.
Precedencias	3.1. Lectura de rúbrica de evaluación y normativa.
Subtareas	-

Tabla 31: Detalles de la tarea 3.2.

T3.3. Redacción

Descripción	Redactar de manera progresiva esta memoria.
Objetivos	Elaborar una memoria clara, coherente y completa que documente el desarrollo del TFG y que sea apta para obtener una evaluación favorable.
Entradas	Plantilla de la memoria del TFG.
Salidas/Producto generado	Memoria final del TFG.
Recursos	Trabajo desempeñado en la línea de la temática del desarrollo.
Duración	60 horas
Precedencias	-
Subtareas	3.2. Definir estructura.

Tabla 32: Detalles de la tarea 3.3.

T3.4. Revisión

Descripción	Revisión propia y progresiva de la memoria.
Objetivos	Corregir errores de redacción, formato, coherencia estructural, exactitud técnica y adecuación a la normativa para garantizar que la memoria sea de buena calidad.
Entradas	Memoria progresiva del TFG.
Salidas/Producto generado	Memoria final del TFG corregida.
Recursos	-
Duración	10 horas.
Precedencias	3.2. Definir estructura.
Subtareas	-

Tabla 33: Detalles de la tarea 3.4.

T3.5. Aprobación

Descripción	Revisión progresiva y aprobación de la memoria por parte de la tutora y del tutor.
Objetivos	Conseguir la validación final por parte de la directora y codirector para presentar el trabajo.
Entradas	Memoria progresiva del TFG.
Salidas/Producto generado	Memoria final del TFG aprobada.
Recursos	-
Duración	20 horas.
Precedencias	3.2. Definir estructura.
Subtareas	-

Tabla 34: Detalles de la tarea 3.5.

T4.1. Definir tarea a resolver

Descripción	Definición detallada del problema que el trabajo busca resolver. Esta fase incluye el estudio de los requisitos que debe cumplir la solución propuesta.
Objetivos	Establecer de forma clara y precisa qué problema se va a resolver y qué resultado se quiere obtener para diseñar una solución efectiva y eficaz.
Entradas	Bases del concurso.
Salidas/Producto generado	Resultado que se persigue teniendo en cuenta los detalles y aspectos fundamentales del problema o limitaciones identificadas.
Recursos	-
Duración	5 horas.
Precedencias	1.3. Definición de objetivos y alcance.
Subtareas	4.1.1. Análisis del entorno de aplicación. 4.2.2. Análisis del resultado deseado.

Tabla 35: Detalles de la tarea 4.1.

T4.2. Estudio del estado del arte

Descripción	Investigación y análisis de las soluciones, enfoques y tecnologías existentes que abordan el problema. Esta tarea implica la lectura de artículos científicos, libros, conferencias, tesis y otras publicaciones académicas.
Objetivo	Contextualizar el trabajo en el panorama actual del área de estudio, identificando las principales soluciones y tendencias en el campo con el objetivo de fundamentar la solución propuesta y destacar la originalidad aportada.
Entradas	Resultado de la tarea 4.1.
Salidas/Producto generado	Capítulo de la memoria que recoger las principales contribuciones, teorías, enfoques y soluciones existentes.
Recursos	Publicaciones académicas y divulgativas citadas en la bibliografía.
Duración	5 horas.
Precedencias	4.1. Definir tarea a resolver.
Subtareas	-

Tabla 36: Detalles de la tarea 4.2.

T4.3. Diseño de la solución

Descripción	Desarrollo de la propuesta de solución para el problema identificado, basado en el análisis previo y el estado del arte. Esta fase incluye la definición de la arquitectura de la solución, la elección de las tecnologías y herramientas a utilizar, así como la especificación de los detalles técnicos y funcionales.
Objetivo	Crear un diseño detallado de la solución que sea capaz de resolver el problema identificado, ajustándose al análisis previo y captura de requisitos. Este diseño debe ser lo suficientemente claro para ser implementado en una fase posterior.
Entradas	Análisis del problema y la solución propuesta.
Salidas/Producto generado	Diseño detallado de la propuesta de solución: grafo computacional de ROS2, nodos a utilizar y comunicación entre nodos.
Recursos	Publicaciones académicas y divulgativas citadas en la bibliografía.
Duración	20 horas.
Precedencias	4.1. Definir tarea a resolver.
Subtareas	4.3.1. Arquitectura. 4.3.2. Herramientas.

Tabla 37: Detalles de la tarea 4.3.

T5.1. Verificación y validación

Descripción	Evaluación del desarrollo de la solución propuesta mediante pruebas y validaciones que aseguren su correcto funcionamiento y el cumplimiento de los requisitos. Incluye la ejecución de pruebas unitarias, de integración y de usuario, así como la validación de que la solución cumple con los objetivos.
Objetivos	Confirmar que la solución diseñada y desarrollada funciona según lo esperado y resuelve efectivamente y eficazmente las pruebas del concurso.
Entradas	-
Salidas/Producto generado	Aprobación o negativa de superación de las pruebas planteadas.
Recursos	Pruebas diseñadas para testear el desarrollo de la solución.
Duración	10 horas.
Precedencias	4.3. Diseño de la solución.
Subtareas	-

Tabla 38: Detalles de la tarea 5.1.

T5.2. Desarrollo de la solución

Descripción	Implementación de la solución propuesta en la fase de diseño. Incluye la codificación, configuración y desarrollo de los componentes del sistema. Se compone de la fase de simulación y de la fase del entorno real. La etapa de simulación busca simular el *hardware* del robot mediante gemelos digitales o gemelos funcionales para comprobar la corrección de la implementación del grafo computacional de *ROS2*, lo que permite descartar estos errores al trabajar sobre el *rover* y acelerar el desarrollo.
Objetivos	Crear y poner en funcionamiento la solución de acuerdo con el diseño y las especificaciones previas.
Entradas	Diseño de la solución propuesta en el análisis.
Salidas/Producto generado	Desarrollo de la solución.
Recursos	Publicaciones académicas y divulgativas citadas en la bibliografía, así como documentación la documentación oficial de las herramientas utilizadas. Herramientas principales: *ROS2 Humble*, *Visual Studio Code* y *Ubuntu 22.04*.
Duración	90 horas.
Precedencias	4.3. Diseño de la solución.
Subtareas	5.2.1. Simulación. 5.2.2. *Rover* real.

Tabla 39: Detalles de la tarea 5.2.

T6.1. Lectura de rúbrica de evaluación y normativa

Descripción	Revisar detalladamente los documentos oficiales proporcionados por la universidad relacionados con la normativa de la defensa del TFG.
Objetivos	Comprender los criterios con los que será evaluada la defensa y conocer la normativa con el fin de alinearla con dichas directrices.
Entradas	Normativa y rúbrica de evaluación del TFG proporcionada por la Escuela de Ingeniería de Bilbao [85].
Salidas/Producto generado	-
Recursos	-
Duración	1 hora.
Precedencias	-
Subtareas	-

Tabla 40: Detalles de la tarea 6.1.

T6.2. Preparar diapositivas

Descripción	Creación y diseño de las diapositivas para la defensa del TFG, asegurando que las diapositivas sean claras, concisas y visualmente atractivas. Esta tarea incluye la selección de los puntos clave del TFG que deben ser presentados, la organización de la información en un formato adecuado, y la inclusión de elementos visuales como gráficos, tablas o imágenes que ayuden a reforzar la información clave.
Objetivos	Desarrollar una presentación visualmente efectiva que resuma los aspectos más importantes del TFG, facilitando la comprensión del trabajo y destacando los puntos clave durante la defensa.
Entradas	Memoria presentada del TFG.
Salidas/Producto generado	Diapositivas o transparencias para la defensa.
Recursos	*Microsoft PowerPoint* [90].
Duración	10 horas.
Precedencias	3.5. Aprobación. 5.2. Desarrollo de la solución. 6.1. Lectura de rúbrica de evaluación y normativa.
Subtareas	-

Tabla 41: Detalles de la tarea 6.2.

T6.3. Realizar ensayos

Descripción	Practicar la exposición oral, mejorar la fluidez y asegurarse de que el tiempo de la presentación no exceda el límite establecido. Esta tarea incluye la realización de numerosos ensayos, la revisión de las diapositivas y la preparación de posibles preguntas lanzadas por el tribunal.
Objetivos	Mejorar la seguridad, fluidez y calidad de la defensa.
Entradas	Memoria presentada del TFG y diapositivas.
Salidas/Producto generado	Preparación exhaustiva y detallada de la presentación oral.
Recursos	*Microsoft PowerPoint* [90], cronómetro, dispositivos audiovisuales, guión de la defensa y retroalimentación de la tutora y del tutor.
Duración	5 horas.
Precedencias	6.2. Preparar diapositivas.
Subtareas	-

Tabla 42: Detalles de la tarea 6.3.

T6.4. Realizar defensa

Descripción	Presentación formal del TFG ante el tribunal de evaluación. Esta tarea implica exponer de manera clara y estructurada los objetivos, el desarrollo, los resultados y las conclusiones del proyecto, utilizando las diapositivas preparadas como soporte visual, y respondiendo adecuadamente a las preguntas y comentarios del tribunal. La realización de esta tarea suponer el final del proyecto.
Objetivos	Exponer de manera precisa el Trabajo de Fin de Grado, demostrando el dominio del tema, la calidad del trabajo realizado y la capacidad de defender el proyecto. El objetivo es obtener una evaluación positiva que permita la aprobación del TFG.
Entradas	Memoria del TFG presentada y diapositivas.
Salidas/Producto generado	Presentación oral y respuestas a las preguntas del tribunal.
Recursos	Dispositivos audiovisuales y guión de la defensa.
Duración	2 horas.
Precedencias	Todas las tareas.
Subtareas	-

Tabla 43: Detalles de la tarea 6.4.

T6.5. Presentar al concurso

Descripción	Participación en el concurso sin contar el viaje hasta Tres Cantos, lugar de celebración.
Objetivos	Superar las pruebas del concurso gracias a las soluciones desarrolladas en este trabajo.
Entradas	Implementación de la solución.
Salidas/Producto generado	Superación de las pruebas.
Recursos	Todas las herramientas utilizadas durante el desarrollo.
Duración	8 horas.
Precedencias	5.1. Verificación y validación. 5.2. Desarrollo de la solución.
Subtareas	-

Tabla 44: Detalles de la tarea 6.5.

9.2. Diagrama *Gantt*

Este diagrama representa cuándo deben realizarse las tareas descritas, así como las dependencias entre tareas. Permite organizar el trabajo a lo largo de todo el proyecto e incluye las tareas que se definen en el EDT de la Ilustración 67.

La fecha de comienzo de este trabajo fue el 29 de octubre de 2024 y la fecha de fin el 18

de julio de 2025. Los recuadros azules representan una tarea específica, mientras que las formas negras referencian a una fase completa. A la izquierda del diagrama se muestra un índice con el número de la tarea y su fecha de comienzo y de fin, y se presenta en la Ilustración 68.

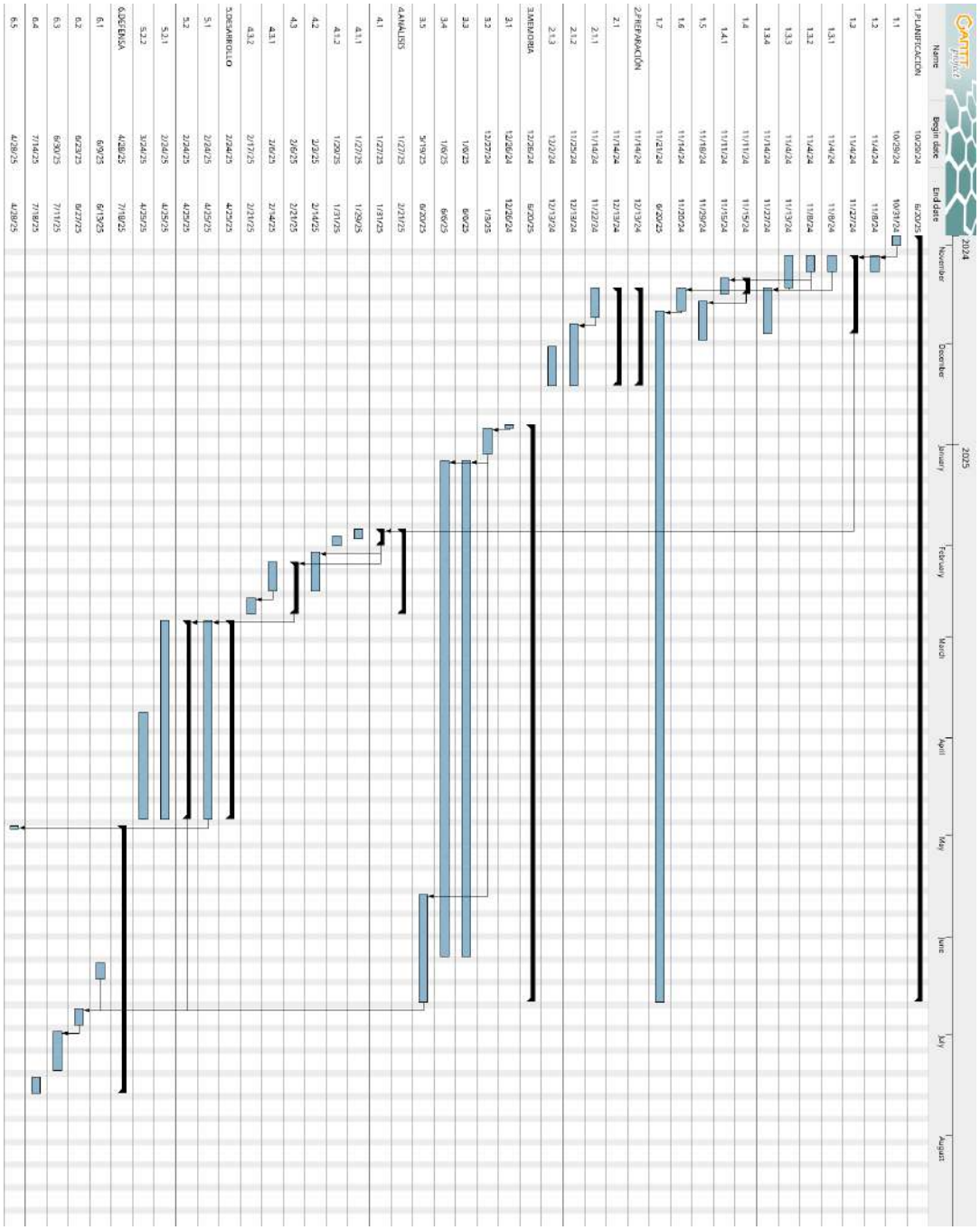


Ilustración 68: Diagrama *Gantt*.

9.3. Metodología *Scrum*

Se han definido tres (3) *sprints* y once (11) historias de usuario (HU). La descripción y resultados perseguidos con cada historia se recogen debajo de la siguiente tabla.

Sprint 1: 2.1. Puesta en marcha del *rover*	HU1: Alimentación de dispositivos.
	HU2: Instalación y configuraci del sistema operativo de la *Raspberry Pi*.
	HU3: *IMU*.
	HU4: *LiDAR*.
	HU5: Cámara.
Sprint 2: 5.1. Verificación y validación	HU6: Verificación de la implementación.
	HU7: Validación de la implementación.
Sprint 3: 5.2. Desarrollo de la solución	HU8: Grafo *ROS2* y nodos para gestión del *hardware*.
	HU9: Prueba de percepción.
	HU10: Prueba de control.
	HU11: Extra.

Tabla 45: *Sprints* e historias de usuario.

HU1: Alimentación de dispositivos

Conectar los sensores y actuadores y comprobar que reciben alimentación suficiente para operar. Los sensores y actuadores objetivo son los controladores serie *Roboclaw* para los motores, motores eléctricos de las ruedas, servomotores, la *IMU*, el *LiDAR* y la cámara.

HU2: Instalación y configuración del sistema operativo de la *Raspberry Pi*

Instalación del sistema operativo *Ubuntu 22.04 LTS* y el entorno de desarrollo *Visual Studio Code* en el ordenador del *rover* y configuración de los puertos serie e *I2C* necesarios y del servicio *SSH*.

La superación de esta historia de usuario proporciona acceso remoto al ordenador del *rover* mediante la terminal de comandos y *Visual Studio Code* y habilita la comunicación de los dispositivos *hardware*.

HU3: *IMU*

Se instalan las librerías y dependencias necesarias para actuar sobre el sensor desde el *software*.

HU4: *LiDAR*

Se instalan las librerías y dependencias necesarias para actuar sobre el sensor desde el *software*.

HU5: Cámara

Se instalan las librerías y dependencias necesarias para actuar sobre el sensor desde el *software*. Tiene que ser accesible tanto la imagen captada como la señal del sensor de profundidad.

HU6: Verificación de la implementación

Se comprueba mediante los test especificados si el resultado del desarrollo cumple con el análisis realizado. En caso de no cumplir, se realizan modificaciones para corregir los errores.

HU7: Validación de la implementación

Se comprueba mediante los test especificados si el resultado del desarrollo supera satisfactoriamente los retos para los que ha sido creado. En caso de no cumplir, se realizan modificaciones para subsanar los errores.

HU8: Grafo *ROS2* y nodos para gestión del hardware

Implementación del grafo computacional diseñado en *ROS2*, especialmente los nodos de los sensores y actuadores que permiten al *rover* realizar acciones.

La superación de esta historia de usuario significa que durante el desarrollo de la solución de los retos únicamente es necesario implementar el proceso abstrayéndose de la complejidad de acceso a los sensores, puesto que los valores de sus lecturas ya estarán almacenados en variables.

HU9: Prueba de percepción

Implementación de la algorítmica necesaria para superar la prueba de percepción.

HU10: Prueba de control

Implementación de la algorítmica necesaria para superar la prueba de control.

HU11: Extra

Implementaciones adicionales que aporten valor añadido a la solución, por ejemplo, la animación de la pantalla.

9.4. Análisis de riesgos

Para cumplir con éxito el objetivo del proyecto, en este apartado se realiza un análisis de riesgos teniendo en cuenta todos los factores posibles. Es fundamental generar una evaluación para la gestión de riesgos que permita minimizar los efectos de cada uno de ellos. Esta evaluación incluye el plan de prevención, el plan de contingencia y el grado.

En la Tabla 46 se definen los niveles de los riesgos, la consecuencia que ocasionarán estos y la probabilidad de que sucedan.

En la Tabla 47 se detallan los valores de porcentaje para los diferentes niveles de probabilidad, así como el nivel de consecuencia y el retraso asociado.

En la Tabla 48, se identifican los diferentes riesgos analizados en este TFG junto con su grado de producirse, evaluados mediante la Tabla 46. La evaluación detallada de cada riesgos se encuentra de la Tabla 49 a la Tabla 68.

El grado de los riesgos se ha determinado utilizando la matriz "daño-riesgo" basada en el punto 6.1.2 del estándar *ISO 45001:2018* [91].

PROBABILIDAD	**CONSECUENCIA**		
	Leve	**Dañina**	**Grave**
Baja	Riesgo trivial	Riesgo tolerable	Riesgo moderado
Media	Riesgo tolerable	Riesgo moderado	Riesgo importante
Alta	Riesgo moderado	Riesgo importante	Riesgo intolerable

Tabla 46: Matriz de evaluación de riesgos según el estándar ISO 45001:2018.

Probabilidad	**Porcentaje**
Baja	0 - 30 %
Media	30 - 65 %
Alta	65 - 100 %

Consecuencia	**Retraso**
Leve	3 días
Dañina	4 - 7 días
Grave	7 días o más

Tabla 47: Probabilidad y consecuencias de los riesgos.

RIESGO	**GRADO**
Indisposición provisional del desarrollador por empleo o prácticas.	
No completar en fecha algún *checkpoint* eliminatorio del concurso.	
No cumplir el plazo de entrega de este trabajo.	
Averías técnicas generales de *software*.	
Cambio en el *firmware* de los componentes.	
Actualizaciones de las librerías utilizadas.	
Planificación temporal incorrecta.	
No documentar el desarrollo durante su realización.	
No completar en fecha algún *checkpoint* no eliminatorio del concurso.	
No cumplir los objetivos.	
Perder información del proyecto.	
Averías técnicas generales de *hardware*.	
Indisposición provisional del desarrollador por enfermedad.	
Infección de virus informático.	
Apagones de la red eléctrica.	

RIESGO	GRADO
Cambio de las especificaciones del proyecto.	
Batería descargada del rover u ordenador de trabajo.	
Perder comunicación con el *rover*.	
Rotura del ordenador de trabajo.	
Sobrescribir alguna versión de desarrollo al generar una nueva.	

Tabla 48: Riesgos asociados al proyecto.

En conclusión, aunque el proyecto no presenta riesgos de elevada gravedad en términos generales, los aspectos técnicos, como las averías y necesidad de cambios en componentes, son los más críticos y deben ser gestionados de manera proactiva para garantizar el éxito del proyecto. La identificación, priorización y tratamiento adecuado de estos riesgos permite reducir su impacto y asegurar la consecución de los objetivos planteados.

Para ello, en las siguientes tablas se recogen los detalles de cada riesgo identificado, así como el plan de contingencia correspondiente que se ha definido en cada caso.

Indisposición provisional del desarrollador por empleo o prácticas

Plan de prevención	Valorar la flexibilidad a la hora de aceptar o rechazar la oferta y avisar a la empresa con el fin de obtener facilidades y no verse afectado el desarrollo del proyecto.
Plan de contingencia	Reorganizar el tiempo dedicado al proyecto para añadir horas adicionales.
Probabilidad	Alta.
Consecuencia	Grave.

Tabla 49: Evaluación de indisposición provisional del desarrollador por empleo o prácticas.

No completar en fecha algún *checkpoint* eliminatorio del concurso

Plan de prevención	Realizar un control y seguimiento del trabajo realizado hasta la fecha y del calendario.
Plan de contingencia	Añadir horas a la planificación para cumplir el plazo y en caso de ser necesario, solicitar una prórroga debidamente justificada.
Probabilidad	Alta.
Consecuencia	Grave.

Tabla 50: Evaluación de no completar en fecha algún *checkpoint* eliminatorio del concurso.

No cumplir el plazo de entrega de este trabajo

Plan de prevención	Realizar un control y seguimiento del trabajo.
Plan de contingencia	Añadir horas a la planificación para cumplir el plazo.
Probabilidad	Media.
Consecuencia	Grave.

Tabla 51: Evaluación de no cumplir el plazo de entrega de este trabajo.

Averías técnicas generales de *software*

Plan de prevención	Mantener el *software* actualizado y utilizar versiones con soporte, tanto oficial como de comunidades de desarrolladores y no obsoletas.
Plan de contingencia	Depurar la incidencia y ponerse en contacto con el soporte técnico oficial y con otros desarrolladores.
Probabilidad	Media.
Consecuencia	Grave.

Tabla 52: Evaluación de averías técnicas generales de *software*.

Cambio en el *firmware* de los componentes

Plan de prevención	Seguir detenidamente las funcionalidades que desaparecerán con la actualización de las siguientes versiones del firmware y diseñar una programación que minimice el uso de dichas funcionalidades.
Plan de contingencia	Cambiar o eliminar las funcionalidades incompatibles o utilizar una versión anterior del *firmware*.
Probabilidad	Media.
Consecuencia	Grave.

Tabla 53: Evaluación de cambio en el *firmware* de los componentes.

Actualizaciones de las librerías utilizadas

Plan de prevención	Comenzar los desarrollos con versiones estables, recientes y suficientemente probadas.
Plan de contingencia	Evitar actualizar una librería si no es estrictamente necesario hasta el fin del proyecto, dejando para una línea futura la adaptación de la solución a las nuevas características de la actualización.
Probabilidad	Alta.
Consecuencia	Dañina.

Tabla 54: Evaluación de actualizaciones de las librerías utilizadas.

Planificación temporal incorrecta

Plan de prevención	Establecer una programación temporal lo más ajustada posible.
Plan de contingencia	Reajustar la planificación temporal para adaptase a la nueva situación.
Probabilidad	Alta.
Consecuencia	Dañina.

Tabla 55: Evaluación de planificación temporal incorrecta.

No documentar el desarrollo durante su realización

Plan de prevención	Realizar la documentación y el desarrollo en paralelo.
Plan de contingencia	Realizar revisiones periódicas, aunque con menor frecuencia que la esperada, para garantizar una solución construida ordenadamente y sin pérdidas de información. Una medida correctora, dependiendo del grado de gravedad de la situación, es revisar la solución parcial y no continuar con ella hasta documentarla mínimamente.
Probabilidad	Alta.
Consecuencia	Dañino.

Tabla 56: Evaluación de no documentar el desarrollo completo durante su realización.

No completar en fecha algún *checkpoint* no eliminatorio del concurso

Plan de prevención	Realizar un control y seguimiento del trabajo realizado hasta la fecha y del calendario.
Plan de contingencia	Añadir horas a la planificación para lograr superar el *checkpoint*.
Probabilidad	Alta.
Consecuencia	Leve.

Tabla 57: Evaluación de no completar en fecha algún *checkpoint* no eliminatorio del concurso.

No cumplir los objetivos

Plan de prevención	Realizar un control y seguimiento del trabajo realizado teniendo en cuenta los objetivos como hoja de ruta.
Plan de contingencia	Replanificar el proyecto con el objetivo de alcanzar los objetivos, o en caso de no verse afectados sustancialmente, reajustar los objetivos.
Probabilidad	Media.
Consecuencia	Dañina.

Tabla 58: Evaluación de no cumplir los objetivos.

Perder información del proyecto

Plan de prevención	Realizar copias de seguridad accesibles, restaurables y periódicas almacenadas en distintas plataformas y equipos, tanto locales como en la nube.
Plan de contingencia	Restaurar las copias de seguridad.
Probabilidad	Baja.
Consecuencia	Grave.

Tabla 59: Evaluación de perder información del proyecto.

Averías técnicas generales de *hardware*

Plan de prevención	Cuidar el material y realizar comprobaciones periódicas del estado y las conexiones físicas.
Plan de contingencia	Evitar el uso del dispositivo defectuoso para limitar el alcance de la avería y sustituir la parte afectada.
Probabilidad	Baja.
Consecuencia	Grave.

Tabla 60: Evaluación de averías técnicas generales de *hardware*.

Indisposición provisional del desarrollador por enfermedad

Plan de prevención	Evitar exponerse a focos de contagio de enfermedades y adoptar hábitos saludables.
Plan de contingencia	Actuar rápidamente para recuperarse a la mayor brevedad posible y reorganizar los plazos de trabajo para minimizar el impacto en el tiempo de ejecución.
Probabilidad	Media.
Consecuencia	Dañina.

Tabla 61: Evaluación de indisposición provisional del desarrollador por enfermedad.

Infección de virus informático

Plan de prevención	Utilizar programas y paquetes *software* oficiales o de fuentes fiables, revisar el código antes de su ejecución para evitar iniciar un programa con funcionalidad maliciosa, bien sea intencionada o accidental, y realizar copias de seguridad accesibles, restaurables y periódicas almacenadas en distintas plataformas y equipos, tanto locales como en la nube.
Plan de contingencia	Si es posible, eliminar la causa de la incidencia y verificar la integridad del resto del sistema. En caso de no ser posible, formatear el equipo y restaurar las copias de seguridad.
Probabilidad	Baja.
Consecuencia	Grave.

Tabla 62: Evaluación de infección de virus informático.

Apagones de la red eléctrica

Plan de prevención	Cuando sea posible, trabajar con los equipos enchufados a la corriente para mantenerlos cargados y realizar copias de seguridad accesibles, restaurables y periódicas almacenadas en distintas plataformas y equipos, tanto locales como en la nube.
Plan de contingencia	Una vez recuperada la corriente, revisar el estado del sistema y restaurar las copias de seguridad si es necesario.
Probabilidad	Baja.
Consecuencia	Grave.

Tabla 63: Evaluación de apagones de la red eléctrica.

Cambio de las especificaciones del proyecto

Plan de prevención	Crear un sistema basado en módulos independientes que permitan reutilizar al máximo las funcionalidades previas.
Plan de contingencia	Revisión y actualización de las nuevas especificaciones.
Probabilidad	Baja.
Consecuencia	Dañina.

Tabla 64: Evaluación de cambio de las especificaciones del proyecto.

Batería descargada del *rover* u ordenador de trabajo

Plan de prevención	Monitorización constante de los niveles de batería y establecer una planificación programada para las cargas, por ejemplo, cada vez que esta descienda del 20 %.
Plan de contingencia	Cargar la batería.
Probabilidad	Baja.
Consecuencia	Leve.

Tabla 65: Evaluación de batería descargada del *rover* u ordenador de trabajo.

Perder comunicación con el *rover*

Plan de prevención	Cuidar el material utilizado para la comunicación (*router* y cable de red), asignar una dirección *IP* estática al *rover* y colocar una pantalla que permita trabajar sobre el rover sin necesidad de una conexión remota.
Plan de contingencia	Reiniciar el servicio de comunicación (*SSH*), reiniciar el ordenador del *rover* o trabajar sobre la pantalla local hasta el restablecimiento de la conexión.
Probabilidad	Baja.
Consecuencia	Leve.

Tabla 66: Evaluación de perder comunicación con el *rover*.

Rotura del ordenador de trabajo

Plan de prevención	Cuidar el equipo y estar atento a las señales que puedan indicar un posible futuro fallo del sistema y realizar copias de seguridad accesibles, restaurables y periódicas almacenadas en distintas plataformas y equipos, tanto locales como en la nube.
Plan de contingencia	Adquirir un nuevo equipo con, al menos, las características mínimas necesarias para desempeñar el desarrollo en progreso y restaurar las copias de seguridad.
Probabilidad	Baja.
Consecuencia	Leve.

Tabla 67: Evaluación de rotura del ordenador de trabajo.

Sobrescribir alguna versión de desarrollo al generar una nueva

Plan de prevención	Llevar un control de versiones con copias de seguridad incrementales tanto locales como en la nube.
Plan de contingencia	Restaurar la última copia de seguridad.
Probabilidad	Baja.
Consecuencia	Leve.

Tabla 68: Evaluación de sobrescribir alguna versión de desarrollo al generar una nueva.

9.5. Herramientas

En esta sección se detallan las herramientas que se utilizan durante el transcurso del proyecto. Es importante destacar que se proponen herramientas libres u *open source*, a excepción de *Visual Studio Code* [87], aunque su uso es gratuito, y *MATLAB*, que se ha usado la versión básica gratuita *online* de 20 horas y existen alternativas gratuitas compatibles como *Octave* [92].

Para las tareas de documentación, planificación, análisis y diseño se utiliza *LibreOffice*, editor de textos "LaTeX" *Overleaf*, *Visual Studio Code* con las extensiones "LaTeX" y "LaTeX Workshop", la aplicación web *draw.io* [93] y el programa de escritorio *The Gantt Project*.

Para desarrollar la implementación y ejecutar la solución en el *rover* se emplea el sistema operativo *Ubuntu 22.04 LTS*, el *framework ROS2 Humble Hawksbill* instalado en la *Raspberry Pi* y *Visual Studio Code* con las extensiones *"Remote-SSH"*, *"Remote-SSH: Editing Configuration Files"*, *"Remote Development"* y *"Remote Explorer"*.

Adicionalmente, con el objetivo de avanzar con el proyecto aún no teniendo el *rover* accesible en todo momento, se realizan simulaciones con gemelos funcionales base en *ROS2* sobre *Docker* y una máquina virtual *KVM* con el gestor *Virtual Machine* [94] que replique el entorno de la *Raspberry Pi*.

El equipo informático que se usará para comunicarse con el rover y trabajar en el proyecto es el ordenador portátil *Gigabyte G6X (2024)* con las siguientes características:

- Procesador: Intel Core i7-13650HX.
- Memoria *RAM*: 32 GB.
- Almacenamiento interno: 1TB.
- Tarjeta gráfica: RTX 4060.

9.6. Evaluación económica

Todo proyecto real tiene un coste asociado. Una vez realizada la planificación temporal se debe estimar el coste que va a tener el proyecto. En este coste se incluye el salario del desarrollador, el precio de los diversos materiales utilizados, así como las amortizaciones que correspondan y una serie de gastos indirectos entre los que se encuentran la electricidad utilizada.

9.6.1. Costes de personal

El salario del programador se obtiene del BOE [95] y del XX Convenio Colectivo Sectorial de Empresas de Ingeniería y Oficinas de Estudios Técnicos. Al tratarse de un TFG, todavía no se tiene el estatus de graduado, por lo que el grupo profesional correspondiente es el segundo y el nivel salarial el tercero. Por tanto, el sueldo anual es de 19.025,86 euros para 1.792 horas de trabajo, es decir, 10,62 euros la hora. A este coste hay que añadirle la contribución a la Seguridad Social, en este caso, un 31,4 % en conceptos de cotizaciones de contingencias comunes (23,6 %), desempleo de tipo general (5,5 %), FOGASA (0,2 %), formación profesional (0,6 %) y accidentes de trabajo y enfermedades profesionales (1,5 %) [96]. Por tanto, como el proyecto tiene un plazo de 9 meses repartidos en 300 horas de trabajo, los gastos asociados al salario son de 4.186,41 euros.

Además, el coste de la supervisión realizada por los tutores también debe ser contabilizada. Siguiendo la planificación descrita, los tutores han realizado las tareas de revisión y aprobación, por lo que se les atribuyen quince horas entre ambos, dividiéndose a partes iguales. Por tanto, teniendo en cuenta los salarios de los tutores [97], su categoría y 1.792 horas de trabajo anuales, el coste es de 341,43 euros.

Recurso	Coste / hora	Seguridad Social	Horas totales	Coste total
Programador	10,62 €	31,4 %	300	4.186,41 €
Directora		Incluido	7	190,27 €
Codirector			7	151,16 €
Coste total de personal				4.527,84 €

Tabla 69: Costes de personal.

9.6.2. Coste del robot

En cuanto a los componentes del *rover*, por un lado está la plataforma de *JPL Open Source Rover Project*, con un coste aproximado de 1.600 dólares estadounidenses [16], es decir, 1.470 euros a fecha del lunes 4 de noviembre de 2024, fecha en la que se confirmó la inscripción al concurso.

Por otro lado, están los dispositivos requeridos por el concurso. La *IMU* tiene un coste de 32,12 euros, el *LiDAR* 78,56 euros, la cámara 137,82 euros, la *Raspberry Pi 4B* 89,95 euros, la pantalla de 86,89 euros y la cúpula de soporte para los sensores y el marco de la pantalla imprimido en material *PLA* de impresión 3D tendrá un coste máximo aproximado de 70 euros. Adicionalmente, se requiere de un *router* para tener una red propia por la que conectarnos al *rover*. El *router* tiene un precio de 69,99 euros.

Tanto con la plataforma como con los dispositivos no se prevé otro uso, por lo que se imputa su coste total.

Asimismo, es preciso incluir una partida dedicada a gastos de envío de los componentes. Teniendo en cuenta que algunos materiales hay que pedirlos a los Estados Unidos de América, cuyos gastos de envío a Europa ascienden aproximadamente a 60 euros, se ha optado por presupuestar 120€ en gastos de envío. También es importante recalcar que durante el desarrollo de este proyecto no existían aranceles en vigor, motivo por el que no se incluyen.

Recurso	Coste de compra
JPL Open Source Rover Project	1.470 €
IMU	32,12 €
LiDAR	78,56 €
Cámara	137,82 €
Raspberry Pi 4B	89,95 €
Pantalla	86,89 €
Cúpula y marco 3D	70 €
Router	69,99 €
Gastos de envío	120 €
Coste total del robot	2.155,33 €

Tabla 70: Coste del robot.

9.6.3. Costes de herramientas *software* y *hardware*

En cuanto a gastos de licencias, es destacable mencionar que no se ha necesitado la adquisición de ninguna licencia como se describe en el apartado 9.5.

El equipo informático dedicado para el proyecto es el ordenador portátil *Gigabyte G6X (2024)* con 1TB de almacenamiento interno y 32GB de memoria *RAM*, que tiene un precio de 1.450 euros. Sabiendo que la vida útil asociada a este tipo de equipos es de cuatro años y que el proyecto se ha extendido durante nueve meses, el coste de este equipo es de 271,88 euros.

Recurso	Coste de compra	Amortización	Uso	Coste en proyecto
Portátil	1.450 €	4 años	9 meses	271,88 €
Coste total	271,88 €			

Tabla 71: Costes de herramientas *software* y *hardware*.

9.6.4. Coste total estimado

Por último, hay que tener en cuenta los gastos indirectos. En este caso, los costes indirectos representan en su mayoría el gasto en electricidad. Dadas las características del proyecto se he decidido presupuestar un 3 % del coste total en gastos indirectos.

El coste total final del proyecto asciende a 7.163,71 euros.

En la Tabla 72 se resume el estudio económico del proyecto y en la Ilustración 69 se recoge gráficamente el peso de cada coste en el proyecto.

Duración del proyecto	300 horas
Plazo del proyecto	9 meses
Tipo	**Coste**
Costes de personal	4.527,84 €
Coste del robot	2.155,33 €
Costes de herramientas *software* y *hardware*	271,88 €
	6.955,05 €
Costes indirectos (3 %)	208,66 €
Total	7.163,71 €

Tabla 72: Evaluación económica total.

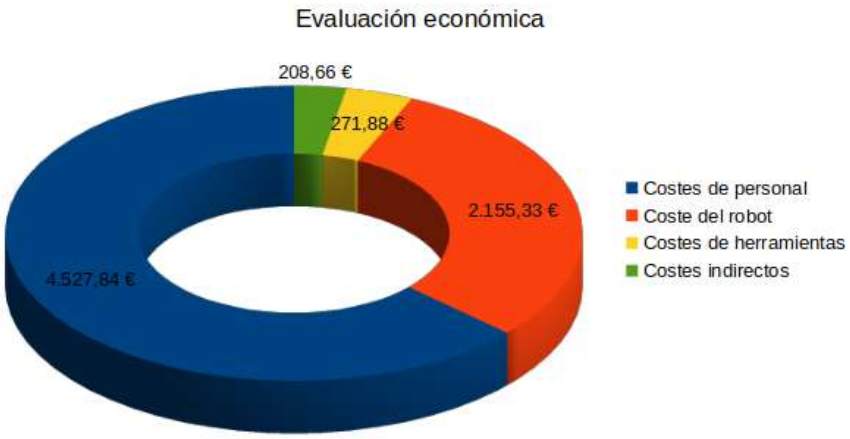


Ilustración 69: Distribución de los costes del proyecto.

10. Conclusiones y líneas futuras

En este último capítulo se recogen las conclusiones obtenidas junto con la evaluación de los objetivos y las líneas futuras de trabajo para dar continuidad al desarrollo realizado.

10.1. Conclusiones y evaluación de los objetivos

El objetivo principal del proyecto es "desarrollar algoritmos de percepción en el entorno y control que permitan a un *rover* identificar números y colores y desplazarse por un entorno cerrado". Tras conocer los resultados de la implementación de los algoritmos trabajados, se concluye que se ha alcanzado el objetivo planteado. Además, dados los buenos resultados obtenidos, se puede asegurar que el proyecto ha sido un éxito.

En cuanto a los objetivos secundarios, destaca la implementación de las soluciones aportadas utilizando *ROS2*, logrando una formación autodidacta en un ámbito emergente y con gran potencial y un resultado actualizado a las tendencias actuales. Asimismo, se ha experimentado que un montaje mecánico robusto y de calidad facilita enormemente el control del los movimientos del robot simplificando la dificultad de la algorítmica a desarrollar.

Durante el desarrollo de la prueba de percepción, generar un conjunto de datos propio ha sido clave para obtener una precisión cercana al 99 %. En un primer momento, se intentó entrenar una red neuronal con *datasets* populares publicados en internet, pero el modelo generaba confusión entre los números 1 y 7, y 6, 8 y 9, e incluían más posibilidades como por ejemplo, números mayores a 9. Además, estos conjuntos no incluían la categoría de "sin número" y tenían un tamaño considerablemente grande, por lo que el modelo de red neuronal era pesado y el rendimiento de la *Raspberry Pi* disminuía, necesitando bastante más tiempo para finalizar la percepción, entre 10 y 15 segundos. Consecuentemente, se planteó generar un conjunto de datos específico para esta aplicación y se ha descubierto que crear un *dataset* correcto para cada utilización ajustado a los requisitos específicos ahorra tiempo de diseño y codificación del modelo, además de mejorar los resultados. Con la solución desarrollada tan sólo se necesitan 2,5 segundos para percibir el entorno.

Además, al hacer los giros no se activaban todos los motores a la vez. En algunos casos, el programa en ejecución se quedaba atascado y los motores no se movían. Con el fin de comprobar la corrección del código y asegurarse de que el algoritmo no estuviera bloqueando el sistema, se creó el gemelo funcional de la *IMU*. Las pruebas fueron positivas, y tras inspeccionar el *rover* se descubrió que una luz roja parpadeaba en la placa, indicando falta de alimentación. Una vez el equipó revisó las soldaduras y con el problema ya corregido, el mismo código funcionaba correctamente. Gracias a este fallo se ha aprendido la importancia del correcto funcionamiento de la electrónica y que

el origen de este tipo de complicaciones no tiene por qué estar en el propio *software*. Asimismo, la realización de pruebas que permitan averiguar la fuente de estos problemas ha resultado crucial.

Por otro lado, en la prueba de control se han ido corrigiendo errores en el planteamiento. En un primer momento, se ideó que la salida del controlador debía ser escalada al rango [-90, 90]. Esto se debía a que con un factor $K_p = 1$, siendo el error máximo 1,5 metros, la corrección máxima generada sería de 1,5, cuando los servomotores funcionan en dicho rango. Tras ir ajustando las constantes de control para mejorar el sistema, se obtuvo que para errores mínimos la corrección era máxima. Al no tener sentido, se dedujo que ese escalado era erróneo e innecesario, ya que es la constante K_p la encargada de aumentar o disminuir la corrección. De aquí se dedujo que el valor mínimo de dicha constante debía ser de 60 (ver apartado 7.5.4).

En último lugar, también se ha producido un aprendizaje en tareas de organización, documentación y revisión. Llevar un registro con toda la información y situaciones generadas ha facilitado enormemente la redacción de esta memoria.

10.2. Líneas futuras de trabajo

La principal línea de trabajo futuro es aumentar la creación y uso de gemelos funcionales y simulaciones. Estas herramientas permiten avanzar en la solución sin utilizar los elementos físicos, generando un entorno seguro, rápido y de bajo coste para realizar pruebas. Concebir un modelo virtual que represente al *rover* es una gran ventaja puesto que permitiría experimentar diferentes estrategias, como por ejemplo, mejorar la prueba de control realizando interpolaciones en las patas traseras o adaptar la implementación de algoritmos como la persecución pura o la obtención del vector neto [98].

En cuanto a la prueba de percepción, se pueden buscar alternativas a las restricciones geométricas mediante el uso del sensor de profundidad o modelos de visión artificial para detectar la caja y su ubicación. De esta manera, se añadiría un paso previo adicional para reconocer la caja y sobre ella se identificaría tanto el número como el color.

10.3. Repositorio del proyecto

La información y el código generado en el proyecto está disponible en el repositorio https://github.com/arambarricalvoj/percepcion-control-ros2-tfg.

Bibliografía

[1] Malgorzata Cognominal, Krystyna Patronymic y Agnieszka Wankowicz. "Evolving Field of Autonomous Mobile Robotics: Technological Advances and Applications". En: *Fusion of Multidisciplinary Research, An International Journal* 2.2 (jul. de 2021), págs. 189-200. DOI: 10.63995/USAS3015.

[2] R. Washington et al. "Autonomous rovers for Mars exploration". En: *1999 IEEE Aerospace Conference. Proceedings (Cat. No.99TH8403)*. Vol. 1. 1999, 237-251 vol.1. DOI: 10.1109/AERO.1999.794236.

[3] Fundación SENER. *Nota de prensa: SENER y CEAs Bot Talent*. Mayo de 2025. URL: https://fundacion.sener/wp-content/uploads/2025/05/2025_05_08_NdP-Sener-CEAs-Bot-Talent.pdf.

[4] Sener Group. *Final del concurso Sener CEAs Bot Talent 2025*. Video, YouTube. Mayo de 2025. URL: https://www.youtube.com/watch?v=DR_lYrUJZxY.

[5] Queensland Institute of Technology. "Future uncertain for Elsie, a sensitive and historic robot". En: *The Queensland Institute of Technology Newspaper* 18 (mayo de 1986). URL: https://digitalcollections.qut.edu.au/3468/6/QIT_May86_Iss18.pdf.

[6] Inser Robotica. *Mobile Industrial Robots: La revolución en el entorno industrial*. URL: https://www.inser-robotica.com/mobile-industrial-robots-revolucion-entorno-industrial/.

[7] Gerard O'Regan. "Unimation". En: *Pillars of Computing: A Compendium of Select, Pivotal Technology Firms*. Cham: Springer International Publishing, 2015. ISBN: 978-3-319-21464-1. DOI: 10.1007/978-3-319-21464-1_34.

[8] Peter E. Hart. *Shakey: From Conception to History*. 2017. URL: https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/General%20Essays/Shakey-aimag-17.pdf.

[9] F. Pugin, P. Bucher y P. Morel. "History of robotic surgery: From AESOP and ZEUS to da Vinci". En: *Journal of Visceral Surgery* 148.5, Supplement (2011). Robotic surgery, e3-e8. ISSN: 1878-7886. DOI: https://doi.org/10.1016/j.jviscsurg.2011.04.007.

[10] J. Edward Colgate et al. "Cobots". US5952796A. Northwestern University. 14 de sep. de 1999. URL: https://patents.google.com/patent/US5952796.

[11] Tobias Kaupp. *Orca-Robotics*. URL: https://orca-robotics.sourceforge.net/.

[12] H. Bruyninckx. "Open robot control software: the OROCOS project". En: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 3. 2001, 2523-2528 vol.3. DOI: 10.1109/ROBOT.2001.933002.

[13] Morgan Quigley et al. "ROS: an open-source Robot Operating System". En: *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*. Kobe, Japan, mayo de 2009.

[14] International Federation of Robotics. *TOP 5 Global Robotics Trends 2025*. URL: https://ifr.org/ifr-press-releases/news/top-5-global-robotics-trends-2025.

[15] Kento Kawaharazuka et al. "Robotic environmental state recognition with pre-trained vision-language models and black-box optimization". En: *Advanced Robotics* 38.18 (2024), págs. 1255-1264. DOI: 10.1080/01691864.2024.2366995.

[16] Jet Propulsion Laboratory y California Institute of Technology. *JPL Open Source Rover Project*. Agosto de 2019. URL: https://open-source-rover.readthedocs.io.

[17] *Mars Perseverance Sol 1503: Right Navigation Camera (Navcam)*. URL: https://mars.nasa.gov/mars2020/multimedia/raw-images/NRF_1503_0800387562_237ECM_N0731522NCAM00266_01_195J.

[18] Steve Macenski et al. "The Marathon 2: A Navigation System". En: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, 2718-2725. DOI: 10.1109/IROS45743.2020.9341207.

[19] N. Koenig y A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". En: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2149-2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[20] O. Michel. "Webots: Professional Mobile Robot Simulation". En: *Journal of Advanced Robotics Systems* 1.1 (2004), págs. 39-42. DOI: https://doi.org/10.5772/5618.

[21] José-Luis Blanco-Claraco et al. "MultiVehicle Simulator (MVSim): Lightweight dynamics simulator for multiagents and mobile robotics research". En: *SoftwareX* 23 (2023), pág. 101443. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2023.101443.

[22] Open Robotics. *RViz*. URL: https://github.com/ros2/rviz.

[23] Jon Aristondo Etxeberria. "Algoritmo de reconocimiento de forma y color para una plataforma robótica". Departamento de Ciencia de la Computación e Inteligencia Artificial. Tesis de Máster. Donostia - San Sebastián, España: Universidad del País Vasco / Euskal Herriko Unibertsitatea, sep. de 2010. URL: https://www.ehu.eus/documents/1545039/1570316/10jaristondo.pdf.

[24] S. M. Shamim et al. "Handwritten Digit Recognition Using Machine Learning Algorithms". En: *Indonesian Journal of Science and Technology* 3.1 (ene. de 2024), págs. 29-39. URL: https://ejournal.kjpupi.id/index.php/ijost/article/view/150.

[25] Wenfei Liu, Jingcheng Wei y Qingmin Meng. "Comparisions on KNN, SVM, BP and the CNN for Handwritten Digit Recognition". En: *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA)*. 2020, págs. 587-590. DOI: 10.1109/AEECA49918.2020.9213482.

[26] Yinglong Li. "Research and Application of Deep Learning in Image Recognition". En: *2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA)*. 2022, págs. 994-999. DOI: 10.1109/ICPECA53709.2022.9718847.

[27] Cai-Xia Deng, Gui-Bin Wang y Xin-Rui Yang. "Image edge detection algorithm based on improved Canny operator". En: *2013 International Conference on Wavelet Analysis and Pattern Recognition*. 2013. DOI: 10.1109/ICWAPR.2013.6599311.

[28] Yulius, Andi Sunyoto y Riyanarto Sarno. "Implementing canny edge detection algorithm for noisy image". En: *Bulletin of Electrical Engineering and Informatics* 8.4 (dic. de 2019), págs. 1260-1267. ISSN: 2089-3191. URL: https://beei.org/index.php/EEI/article/view/1837/2013.

[29] César Osimani. "Análisis y procesamiento de imágenes para la detección del contorno labial en pacientes de odontología". En: nov. de 2014.

[30] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[31] Michael Nielsen. *Neural Networks and Deep Learning*. 2015. URL: http://neuralnetworksanddeeplearning.com/.

[32] Joseph Stewart y Matt Church. *ROS 2 Robot With SLAM*. Oct. de 2024. DOI: 10.13140/RG.2.2.15510.56643.

[33] Daniel D. Yanyachi et al. "Laser RobMap: An open source ROS2 compatible tool for 3D mapping using a Mobile Robot and 2D LiDAR". En: *SoftwareX* 30 (2025), pág. 102142. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2025.102142.

[34] Alba Marqués Reca. "Planificación de caminos reactiva mediante persecución pura en coche a escala utilizando ROS y Gazebo". Trabajo de Fin de Grado. Escuela de Ingenierías Industriales de la Universidad de Málaga, mayo de 2024.

[35] Ahmed Abdulsahib. "Path Following and Motion Control for Articulated Frame Steering Mobile Working Machine Using ROS2". Tesis de Máster. Universidad de Tampere, dic. de 2022. URL: https://trepo.tuni.fi/bitstream/handle/10024/144698/Ahmed_Abdulsahib.pdf?sequence=5.

[36] Javier Minguez y Luis Montano. "Nearness Diagram (ND) Navigation: Collision Avoidance in Troublesome Scenarios". En: *IEEE Transactions on Robotics and Automation* 20.1 (feb. de 2004), págs. 45-59. DOI: 10.1109/TRA.2003.820849.

[37] Francisco J. Real Pérez. "Navegación Reactiva y Seguimiento en Entornos Urbanos mediante Percepción Láser y Mapas Probabilísticos". Proyecto Final de Máster. Escuela Superior de Ingenieros, Universidad de Sevilla, nov. de 2008. URL: https://biblus.us.es/bibing/proyectos/use/abreproy/70073/fichero/memoria.pdf.

[38] J. Morales et al. "Pure-Pursuit Reactive Path Tracking for Nonholonomic Mobile Robots with a 2D Laser Scanner". En: *EURASIP Journal on Advances in Signal Processing* 2009 (dic. de 2009), págs. 1-10. DOI: 10.1155/2009/935237.

[39] Josu Arambarri Calvo y Javier Arambarri Calvo. *EXPLORANDO LA ROBÓTICA: Guía para First Lego League y World Robot Olympiad*. España: Amazon KDP, oct. de 2024. URL: https://www.amazon.es/dp/B0DJDJM1J8.

[40] Josu Arambarri Calvo y Javier Arambarri Calvo. *Algoritmos de programación avanzada para First Lego League y World Robot Olympiad*. España: Amazon KDP, mar. de 2023. URL: https://www.amazon.es/dp/B0BYRK51T4.

[41] Gabriel Delgado-Oleas et al. "Diseño y desarrollo de una arquitectura electrónica bioinspirada para el control de sistemas de asistencia a la locomoción". En: *Revista Iberoamericana de Automática e Informática industrial* 20.3 (abr. de 2023), págs. 293-302. DOI: 10.4995/riai.2023.18748.

[42] NASA Jet Propulsion Laboratory. *Perseverance Rover*. *Rover* construido para la misión *Mars 2020*, lanzado en 2020 y aterrizado en Marte en 2021. 2020. URL: https://mars.nasa.gov/mars2020/.

[43] *goRAIL®*. URL: https://www.gobilda.com/gorail.

[44] Hernán Martín Varela. "Estudio de un rover con suspensión rocker-bogie para misiones en Marte". Tesis Doctoral. UPC, Escuela Superior de Ingenierías Industrial, Aeroespacial y Audiovisual de Terrassa, Departamento de Ingeniería Mecánica, jul. de 2021. URL: http://hdl.handle.net/2117/360872.

[45] Abhisek Verma et al. "Design of Rocker-Bogie Mechanism". En: *International Journal of Innovative Science and Research Technology* 2.5 (2017), págs. 312-319. ISSN: 2456-2165. URL: https://ijisrt.com/wp-content/uploads/2017/05/Design-of-Rocker-Bogie-Mechanism-1.pdf.

[46] *5203 Series Yellow Jacket Planetary Gear Motor (26.9:1 Ratio, 24mm Length 8mm REX Shaft, 223 RPM, 3.3 - 5V Encoder)*. URL: https://www.gobilda.com/5203-series-yellow-jacket-planetary-gear-motor-26-9-1-ratio-24mm-length-8mm-rex-shaft-223-rpm-3-3-5v-encoder/.

[47] *2000 Series Dual Mode Servo (25-2, Torque)*. URL: https://www.gobilda.com/2000-series-dual-mode-servo-25-2-torque/.

[48] *Astra Series - ORBBEC - 3D Vision for a 3D World*. URL: https://www.orbbec.com/products/structured-light-camera/astra-series/.

[49] *YDLIDAR X4*. URL: https://www.ydlidar.com/products/view/5.html.

[50] Ltd. Shenzhen EAI Technology Co. *YDLIDAR X4 DATA SHEET*. 2018.

[51] *Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055*. URL: https://www.adafruit.com/product/2472.

[52] NXP Semiconductors. *UM10204: I2C-bus specification and user manual*. Rev. 6. 2014. URL: https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[53] *14.8V 4S 4000mAh 60C LiPo Battery*. URL: https://hrb-power.com/products/2pcs-hrb-14-8v-4s-4000mah-60c-lipo-battery-for-rc-drone-car-airplane-boat.

[54] *Roboclaw 2x7A Motor Controller*. URL: https://www.basicmicro.com/Roboclaw-2x7A-Motor-Controller_p_55.html.

[55] *Raspberry Pi 4 - 8GB*. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/?variant=raspberry-pi-4-model-b-8gb.

[56] *Pantalla Táctil Capacitiva para RPI, 400x1280, IPS, Interfaz DSI 7,9 pulg Waveshare*. URL: https://eu.robotshop.com/es/products/pantalla-tactil-capacitiva-para-rpi-400x1280-ips-interfaz-dsi-79-pulg-waveshare.

[57] *KETOTEK Voltímetro Amperímetro Digital DC 6.5-100V 20A 12V*. Dispositivo electrónico. Medidor de voltaje, amperaje, potencia y energía eléctrica de panel, pantalla LCD. KETOTEK, 2025.

[58] *TL-MR110 | Router 4G LTE Wi-Fi 4 N300*. URL: https://www.tp-link.com/es/home-networking/5g-4g-router/tl-mr110/.

[59] A. Burgos et al. *MeiA 4.0 (Metodología para ingeniería de Automatización)*. Nov. de 2022. DOI: 10.13140/RG.2.2.22961.66400.

[60] Steven Macenski et al. "Robot Operating System 2: Design, architecture, and uses in the wild". En: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074.

[61] Object Management Group. *Data Distribution Service (DDS) Version 1.4*. 2015. URL: https://www.omg.org/spec/DDS/1.4.

[62] Open Source Robotics Foundation. *ROS 2 Humble Hawksbill Documentation*. Version Humble Hawksbill. Open Source Robotics Foundation. 2022. URL: https://docs.ros.org/en/humble/index.html.

[63] Open Source Robotics Foundation. *ROS 2 Jazzy Jalisco Documentation*. 2024. URL: https://docs.ros.org/en/jazzy/index.html.

[64] Open Robotics. *sensor_msgs/msg/Imu — ROS 2 Humble documentation*. 2022. URL: https://docs.ros.org/en/humble/p/sensor_msgs/msg/Imu.html.

[65] ROS 2 Design. *Node Lifecycle Management*. 2018. URL: https://design.ros2.org/articles/node_lifecycle.html.

[66] Francisco Utray. *Codificación digital de la imagen*. Dykinson, 2015.

[67] Nicolás Aguirre Dobernack. "Implementación de un sistema de detección de señales de tráfico mediante visión artificial basado en FPGA". Trabajo Fin de Grado. Universidad de Sevilla, 2013.

[68] Dra. Nora La Serna, Mg. Luzmila Pro Concepción y Lic. Carlos Yañez Durán. "Compresión de imágenes: Fundamentos, técnicas y formatos". En: *Revista de Ingeniería de Sistemas e Informática de la Universidad Nacional Mayor de San Marcos* 6.1 (2009), págs. 21-29.

[69] Open Robotics. *sensor_msgs/msg/Image Documentation (ROS 2 Humble)*. https://docs.ros.org/en/humble/p/sensor_msgs/msg/Image.html. 2022.

[70] Rafael C. Gonzalez y Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., 2006. ISBN: 013168728X.

[71] Anup Vibhute, Sk Bodhe y Babasaheb More. "Nitrogen Estimation for Grapevine (In Veraison) Using RGB Color Image Processing". En: *Research and Reviews: Journal of Botanical Sciences* 3 (dic. de 2013), págs. 4-7.

[72] A. de la Escalera. *Visión por Computador: Fundamentos y Métodos*. 1.ª ed. Prentice Hall, 2001.

[73] OpenCV. *Espacios de color en OpenCV: Rango de valores HSV*. 2024. URL: https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html.

[74] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 2nd (final draft, Sept. 2021). 2021.

[75] Saily Shah. *Convolutional Neural Network: An Overview*. 2022. URL: https://www.analyticsvidhya.com/back-channel/download-pdf.php?pid=87620.

[76] John Canny. "A Computational Approach To Edge Detection". En: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* PAMI-8 (dic. de 1986), págs. 679-698. DOI: 10.1109/TPAMI.1986.4767851.

[77] The MathWorks Inc. *MATLAB version: 9.13.0 (R2022b)*. Natick, Massachusetts, United States, 2022. URL: https://www.mathworks.com.

[78] EAI Team. *YDLIDAR Python SDK*. https://github.com/YDLIDAR/YDLidar-SDK. 2025.

[79] Open Source Robotics Foundation. *sensor_msgs/msg/LaserScan.msg – ROS2 Humble*. 2022. URL: https://github.com/ros2/common_interfaces/blob/humble/sensor_msgs/msg/LaserScan.msg.

[80] Spartacus Gomáriz Castro et al. *Teoría de control. Diseño electrónico*. Politext 72. Barcelona: Ediciones UPC, S.L., 1998, pág. 392. ISBN: 978-84-8301-266-6.

[81] Picuino. *Controlador PID. Método de Ziegler-Nichols*. 2024. URL: https://www.picuino.com/es/control-ziegler-nichols.html.

[82] Open Source Robotics Foundation. *Gazebo Harmonic*. Documentación oficial de Gazebo Harmonic. 2024. URL: https://gazebosim.org/docs/harmonic.

[83] *Docker: accelerated container application development*. URL: https://www.docker.com/.

[84] *GanttProject: Free Project Management App*. URL: https://www.ganttproject.biz/.

[85] *Trabajo Fin de Grado - Curso 2024/2025*. URL: https://www.ehu.eus/es/web/bilboko-ingeniaritza-eskola/grados/trabajo_fin_de_grado.

[86] *Overleaf, Online LaTeX Editor*. URL: https://www.overleaf.com/.

[87] *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/.

[88] *LaTeX*. URL: https://marketplace.visualstudio.com/items?itemName=mathematic.vscode-latex.

[89] *LaTeX Workshop*. URL: https://marketplace.visualstudio.com/items?itemName=James-Yu.latex-workshop.

[90] *Microsoft PowerPoint*. URL: https://www.microsoft.com/es-es/microsoft-365/powerpoint.

[91] *International Organization for Standarization. ISO 45001:2018(es) Sistemas de gestión de la seguridad y salud en el trabajo — Requisitos con orientación para su uso*. 2018. URL: https://www.iso.org/obp/ui/#iso:std:iso:45001:ed-1:v1:es.

[92] *GNU Octave*. 2025. URL: https://www.octave.org/.

[93] *draw.io*. URL: https://www.drawio.com/.

[94] *Virtual Manager*. URL: https://virt-manager.org/.

[95] Boletín Oficial del Estado. *Resolución de 12 de marzo de 2024, de la Dirección General de Trabajo, por la que se registran y publican las tablas salariales para 2024 del XX Convenio colectivo nacional de empresas de ingeniería, oficinas de estudios técnicos, inspección, supervisión y control técnico y de calidad.* Boletín Oficial del Estado, número BOE-A-2024-5873. 2024. URL: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2024-5873.

[96] Ministerio de Inclusión, Seguridad Social y Migraciones. *Régimen General de la Seguridad Social*. Sitio web del Ministerio de Inclusión, Seguridad Social y Migraciones. 2023. URL: https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537.

[97] Universidad del País Vasco (UPV/EHU). *Retribuciones - gardentasun-ataria*. https://www.ehu.eus/es/web/gardentasun-ataria/ordainketak.

[98] Chris Anderson. *Lessons learned making a Lidar-based maze rover*. DIY Robocars. 2020. URL: https://www.diyrobocars.com/2020/03/19/lessons-learned-making-a-lidar-based-maze-rover/.

[99] *Raspberry Pi OS*. URL: https://www.raspberrypi.com/software/.

[100] *Ubuntu 22.04.5 LTS (Jammy Jellyfish)*. URL: https://releases.ubuntu.com/jammy/.

[101] Chris M. Lonvick y Tatu Ylonen. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253. Ene. de 2006. DOI: 10.17487/RFC4253.

[102] Clearpath Robotics. *Udev Rules*. 2017. URL: https://www.clearpathrobotics.com/assets/guides/kinetic/ros/Udev%20Rules.html.

[103] Basicmicro. *Basicmicro Motion Studio Legacy Version 1.0.0.75*. https://downloads.basicmicro.com/software/Legacy/BasicmicroMotionStudio_1.0.0.75.zip. 2020.

[104] Open Source Robotics Foundation. *rclpy: ROS Client Library for the Python language*. https://github.com/ros2/rclpy. Version 3.3.16, ROS 2 Humble Hawksbill. 2025.

[105] Adafruit Industries. *Adafruit CircuitPython Board*. https://github.com/adafruit/Adafruit_Blinka. 2025.

[106] Adafruit Industries. *Adafruit CircuitPython BNO055*. https://github.com/adafruit/Adafruit_CircuitPython_BNO055. 2025.

[107] Adafruit Industries. *Adafruit CircuitPython ServoKit*. https://github.com/adafruit/Adafruit_CircuitPython_ServoKit. 2025.

[108] Basicmicro. *roboclaw_python_library*. https://github.com/basicmicro/roboclaw_python_library. 2019.

[109] G. Bradski. "The OpenCV Library". En: *Dr. Dobb's Journal of Software Tools* (2000).

[110] Open Source Robotics Foundation. *cv_bridge: ROS package for converting ROS images to OpenCV images*. https://github.com/ros-perception/vision_opencv. 2024.

[111] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. https://www.tensorflow.org/. 2015.

[112] Charles R. Harris et al. *NumPy*. 2020. DOI: 10.1038/s41586-020-2649-2.

[113] Python Software Foundation. *The Python Standard Library*. https://docs.python.org/3/library/. 2025.

[114] Python Packaging Authority. *setuptools: Easily download, build, install, upgrade, and uninstall Python packages*. https://github.com/pypa/setuptools. 2025.

Anexo I: Configuración de la conectividad en el *rover* y librerías utilizadas

En este anexo se detalla la configuración necesaria para activar la conectividad con el robot y las librarías que se han utilizado:

I.1. Instalación del sistema operativo 147

I.2. Configurar la red *Wi-Fi* 147

I.3. Configuración de *SSH* 148

I.4. Habilitar puertos serie e *I2C* 149

I.5. Establecer las reglas *udev* 150

I.6. Modificar el fichero *~/.bashrc* 152

I.7. *Firmware* de *Roboclaw* 152

I.8. Librerías *software* utilizadas 152

I.1. Instalación del sistema operativo

El primer paso es instalar el sistema operativo. Se ha utilizado el programa *Raspberry Pi Imager* [99] para volcar la imagen de *Ubuntu Desktop 22.04.5 LTS* [100] en la tarjeta *micro-SD*. En la interfaz gráfica, Ilustración 70, seleccionar el tipo de *Raspberry Pi*, el sistema operativo y la tarjeta *micro-SD*.

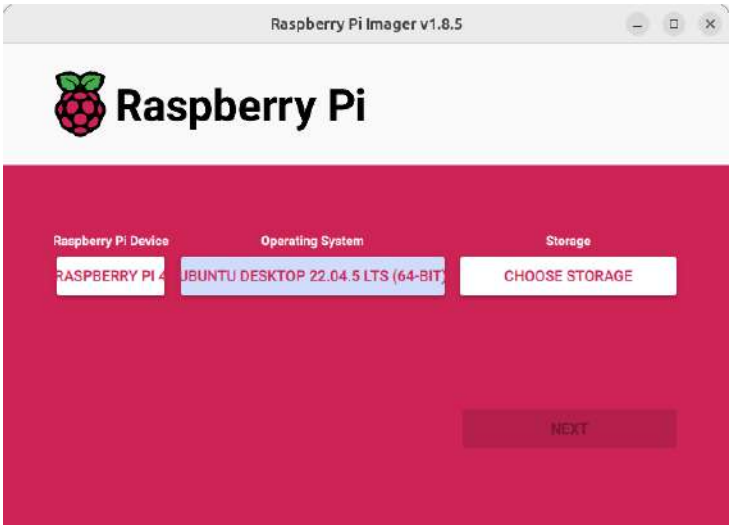


Ilustración 70: *Raspberry Pi Imager.*

Durante el primer arranque del sistema se siguen los pasos guiados de personalización y configuración.

Después, se empieza con la configuración específica. Se debe activar la conexión remota *SSH* y configurar la red *WiFi* a la que se conectará automáticamente, habilitar los puertos serie e *I2C* y establecer reglas *udev*.

En las siguientes secciones se detallan los pasos y comandos de terminal *bash* necesarios para proceder con la configuración.

I.2. Configurar la red *Wi-Fi*

Hay que marcar la opción *Connect automatically* de la red seleccionada como se muestra en las ilustraciones 71 y 72.

Ilustración 71: Configuración de la red.

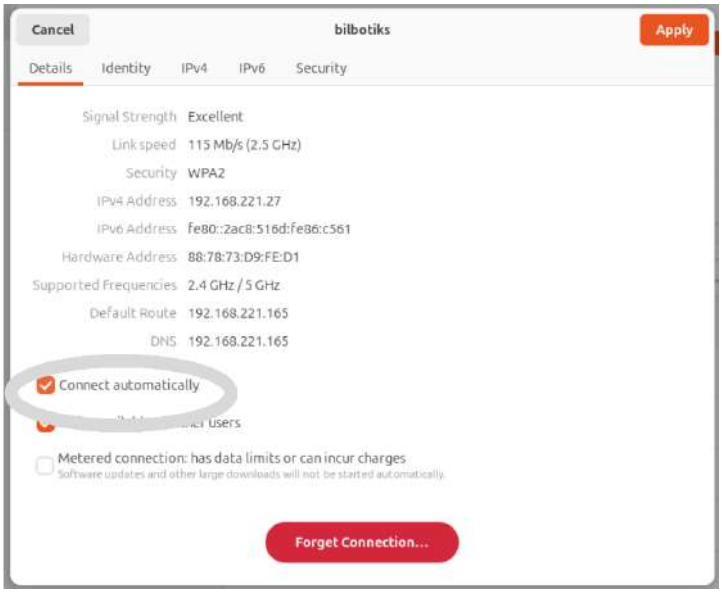


Ilustración 72: Conectarse automáticamente a la red.

I.3. Configuración de *SSH*

SSH [101] es un protocolo inalámbrico de red que permite conectarse y administrar de forma remota y segura dispositivos o servidores a través de una red. El protocolo se basa en una arquitectura cliente-servidor y cifra toda la comunicación entre el cliente y el servidor. Permite ejecutar comandos de manera remota, transferir archivos de forma segura y realizar tareas de administración desde la línea de comandos.

En este proyecto el protocolo *SSH* se utiliza para conectarse y comunicarse con el *rover*.

En primer lugar, hay que configurar el servidor *SSH*, que será el *rover*. Para ello, hay que instalar el servicio, generar la clave criptográfica que acometerá el cifrado, activar y arrancar el servidor y abrir el puerto 22 en el *firewall*:

```
$ sudo apt install openssh-server #Instalar servidor SSH
$ ssh-keygen #Generar la clave criptográfica necesaria

$ sudo systemctl enable ssh #Activar el servidor
$ sudo systemctl start ssh #Arrancar el servidor

$ sudo ufw allow ssh #Habilitar puerto 22 para conexiones remotas
$ sudo ufw enable
$ sudo ufw status
```

Código 10.1: Comandos *bash* para la instalación del servidor *SSH*.

En segundo lugar, hay que configurar el cliente *SSH*, es decir, el ordenador que se va a conectar al servidor. Tanto en versiones posteriores a *Ubuntu 16.04 LTS* como en *Windows 11* el cliente viene instalado por defecto. No obstante, se incluye el comando que instala el cliente en caso de ser necesario:

```
$ sudo apt install openssh-client #Instalar cliente SSH
$ ssh usuario@direccion #Conectarse al rover

#usuario: usuario en el sistema operativo
#direccion: dirección IP del rover en la red
```

Código 10.2: Comandos *bash* para la instalación del cliente *SSH*.

I.4. Habilitar puertos serie e *I2C*

En los sistemas robóticos basados en *Raspberry Pi* es habitual utilizar diferentes interfaces de comunicación para gestionar los distintos componentes electrónicos. El puerto serie se emplea para el control directo de los motores de las ruedas, permitiendo enviar y recibir datos entre la *Raspberry Pi* y el controlador de motores.

Por otro lado, la comunicación *I2C* [52] se utiliza para conectar sensores como la *IMU*, ya que este bus permite la transmisión de datos entre múltiples dispositivos utilizando solo dos líneas o cables, resultando ideal para este tipo de sensores.

La manera más sencilla de habilitar ambos puertos es mediante la herramienta *raspi-config*:

```
$ sudo apt install raspi-config #Instalar raspi-config
$ sudo raspi-config #Ejecutar raspi-config
```

Código 10.3: Comandos *bash* para instalar y acceder a *raspi-config*.

Una vez abierta la configuración desde terminal, Ilustración 73, utilizando el teclado seleccionar la tercera opción de *Interface Options* y seleccionar *Serial* e *I2C*.

Al seleccionar *Serial* salen dos preguntas: la primera para acceder a un intérprete de comandos de inicio de sesión a través del puerto serie, y hay que responder "no", mientras que a la segunda, para habilitar el *hardware* del puerto serie hay que responder "sí".

Al seleccionar *I2C* una única pregunta aparecerá, a la que hay que contestar "sí" para activar la comunicación *I2C*.

Por último, es necesario reiniciar la *Raspberry Pi*:

```
$ sudo shutdown -r now
```

Código 10.4: Comando *bash* para reiniciar la *Raspberry Pi*.

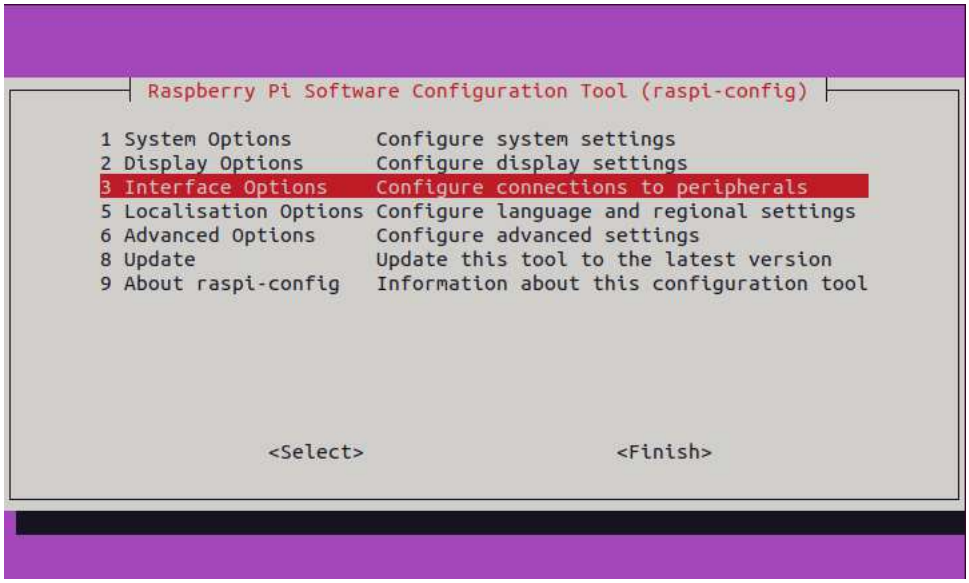


Ilustración 73: Herramienta *rapi-config*.

I.5. Establecer las reglas *udev*

Las reglas *udev* [102] en *Ubuntu* son archivos de configuración que permiten definir cómo debe actuar el sistema operativo cuando se conecta, desconecta o cambia el estado de un periférico. Estas reglas se encuentran normalmente en el directorio */etc/udev/rules.d/* y pueden usarse para asignar nombres fijos a dispositivos, cambiar permisos, crear enlaces simbólicos o ejecutar programas automáticamente al detectar un evento relacionado con el *hardware*.

En este proyecto se utilizan para crear un enlace simbólico a los dispositivos serie e *I2C*.

Para los dispositivos serie:

```
$ sudo nano /etc/udev/rules.d/90-serial.rules #Crear la regla en un
   fichero
```

Código 10.5: Comando *bash* para crear la regla *udev* del puerto serie.

```
#Regla para el puerto serie

KERNEL=="ttyS0", SYMLINK+="serial0" GROUP="tty" MODE="0660"
KERNEL=="ttyAMA0", SYMLINK+="serial1" GROUP="tty" MODE="0660"

#KERNEL referencia al puerto.
#SYMLINK define el nombre simbólico con el que se conocerá al puerto
#GROUP="tty" referencia al grupo que tiene permisos sobre el bus
#MODE="0660" indica los permisos que tiene el grupo. Este modo otorga
   permisos de lectura y escritura al propietario y pertenecientes
   al grupo (rw-). Cualquier otro usuario no tiene acceso.
```

Código 10.6: Regla *udev* del puerto serie.

Para los dispositivos *I2C*:

```
$ sudo nano /etc/udev/rules.d/60-i2c-tools.rules #Crear la regla en
   un fichero
```

Código 10.7: Comando *bash* para crear la regla *udev* del puerto *I2C*.

```
#Regla para el puerto I2C

KERNEL=="i2c-0"     , GROUP="i2c", MODE="0660"
KERNEL=="i2c-[1-9]*", GROUP="i2c", MODE="0666"

#KERNEL=="i2c-0" referencia al primer bus I2C
#GROUP="i2c" referencia al grupo que tiene permisos sobre el bus
#MODE="0660" indica los permisos que tiene el grupo. Este modo otorga
   permisos de lectura y escritura al propietario y pertenecientes
   al grupo (rw-). Cualquier otro usuario no tiene acceso.
```

Código 10.8: Regla *udev* del puerto *I2C*.

Por último, hay que añadir el usuario a los grupos definidos en las reglas y a los correspondientes para poder acceder a los puertos necesarios. Es necesario reiniciar el equipo para que los cambios surtan efecto.

```
$ sudo adduser $USER tty #Permiso para controlar y acceder a
   terminales y consolas del sistema

$ sudo adduser $USER dialout #Permiso para acceder a dispositivos de
   puerto serie

$ sudo adduser $USER i2c #Permiso para acceder al bus i2c
```

Código 10.9: Comando *bash* para añadir el usuario a los grupos con permisos en los puertos serie e *I2C*.

I.6. Modificar el fichero *~/.bashrc*

Por motivos de comodidad es recomendable incluir en el fichero *~/.bashrc* las siguiente dos líneas:

```
source /opt/ros/humble/setup.bash
source ~/bilbotiks_ws/install/setup.bash
```

Código 10.10: Activar automáticamente el espacio de trabajo de *ROS2*.

Estas dos líneas activan nuestro espacio de trabajo y los comandos de *ROS2*.

Durante el proyecto se ha trabajado en la ruta */bilbotiks_ws/install/setup.bash*. En caso de no incluir estas sentencias en dicho fichero, tendrán que ser ejecutadas cada vez que se abra una terminal.

I.7. *Firmware* de *Roboclaw*

El *firmware* es un tipo de *software* especializado que está integrado directamente en el *hardware* de un dispositivo electrónico y se encarga de controlar su funcionamiento básico. Actúa como una capa intermedia entre el *hardware* y el *software* de más alto nivel, en el caso de este proyecto, permite interactuar con los dispositivos mediante *Python* proporcionando las instrucciones esenciales.

El *firmware* suele estar almacenado en memorias no volátiles, ya que sin él no hay posibilidad de programar el dispositivo desde alto nivel.

Se ha trabajado con la versión 4.28 de los controladores *Roboclaw* de las ruedas. Se puede actualizar o bajar la versión de estos dispositivos mediante el programa *Basicmicro Motion Studio 1.75 Legacy Version* [103] del fabricante.

I.8. Librerías *software* utilizadas

En todo proyecto que conlleve desarrollo de *software* es habitual emplear librerías creadas bien por los fabricantes o bien por otros desarrolladores.

En esta sección se listan las librerías necesarias para el desarrollo del proyecto. Al final de esta sección se proporciona una tabla con la versión de cada librería.

Entre las más relevantes se encuentra *rclpy* [104], que forma parte de la instalación completa de *ROS2 Humble Hawksbill* [62] y permite crear nodos *ROS2* en *Python* para la comunicación y control de los distintos módulos del robot.

Para la gestión de *hardware* específico, se han empleado las librerías *board* [105] y *adafruit_bno055* [106] para la *IMU*, *adafruit_servokit* [107] para el control de los servomotores, *roboclaw* [108] para los motores de las ruedas e *ydlidar* junto con su *SDK* [78] correspondiente para el *LiDAR*.

En cuanto a visión artificial, procesamiento de imágenes y aprendizaje automático, se utilizan *cv2* de *OpenCV* [109], *cv_bridge* [110] para convertir imágenes entre *ROS2* y

tensorflow [111].

Para cálculos matemáticos y manipulación de datos, se emplean *numpy* [112] y librerías estándar [113] como *math*, *random*, *csv*, *time* y *datetime*.

Para la comunicación serie y el manejo de datos binarios se utilizan las librerías estándar *serial* y *struct*.

Estas librerías, junto con *setuptools* [114] para la gestión de paquetes *Python*, constituyen el entorno necesario para el desarrollo y despliegue del *software* del robot.

Por último, es fundamental indicar que debido a la arquitectura *ARM* de la *Raspberry Pi* es necesario utilizar versiones específicas de las librerías optimizadas para dicha arquitectura. En la Tabla 73 se recogen las versiones utilizadas tanto en el ordenador para realizar simulaciones, como en la *Raspberry Pi* (Versión en *ARM*).

Librería	**Versión en x64**	**Versión en *ARM***
numpy	2.2.3	1.21.5
opencv_python (cv2)	4.11.0.86	
cv_bridge	1.13.0.post0	
tensorflow	2.19.0	2.19.0
rclpy	*ROS2 Humble Hawksbill*	
board (*Adafruit-Blinka*)	8.51.0	
adafruit_bno055	5.4.18	
adafruit_servokit	1.3.19	
roboclaw	Única	
ydlidar	1.2.4	
math	Python 3.10.12	
struct	Python 3.10.12	
random	Python 3.10.12	
csv	Python 3.10.12	
time	Python 3.10.12	
datetime	Python 3.10.12	
serial	3.5	
setuptools	59.6.0	

Tabla 73: Librerías utilizadas y versiones.

Anexo II: Código

En este anexo se incluye el código *software* del TFG:

II.1. Fichero *.yaml* de parámetros del proyecto 155
II.2. Servidor y respuesta del servicio para sacar una foto 156
II.3. Cliente y solicitud del servicio para sacar una foto 157
II.4. Desarrollo del modelo *CNN* 158
II.5. Comprobación del funcionamiento del *LiDAR* 169
II.6. Generar los gráficos de la señal de control en *MATLAB* 170
II.7. Nodos 171
II.8. Interfaces gráficas 196
II.9. Gemelo digital y simulación 206

II.1. Fichero *.yaml* de parámetros del proyecto

```yaml
motor_kontrolatzailea:
  ros__parameters:
    baud_rate: 115200
    motorren_helbideak: [128, 129, 130]

servo_kontrolatzailea:
  ros__parameters:
    actuation_range: 300
    pulse_width_range: [500, 2500]

pertzepzio_proba_wrapper:
  ros__parameters:
    servoak_360: [90, 65, 105, 65]
    cnn_modelua: "/home/bilbotiks/moverRover/bilbotiks_ws/install/bilbotiks_controller/
    share/bilbotiks_controller/config//modelo_100epochs.keras"
    abiadura_azkar: 30
    abiadura_motel: 13

kontrol_proba_wrapper:
  ros__parameters:
    kp: 100.0
    ki: 0.0
    kd: 25.0
    vel: 30
    izena: "nombreFicheroGuardadoDatos"
    eskuina: [-90.0, -67.5]
    ezkerra: [157.5, 180.0]
    aurrera: [-180.0, -90.0]
```

Código 10.11: Contenido del fichero *.yaml* de parámetros del proyecto.

II.2. Servidor y respuesta del servicio para sacar una foto

```
# Servidor del SERVICE /argazkia_atera en el nodo KAMERA
self.srv_argazkia_atera = self.create_service(Argazkia, 'argazkia_atera', self.
    argazkia_atera)
```

Código 10.12: Implementación del servidor del servicio para sacar una foto.

```
def argazkia_atera(self, request, response):
    self.get_logger().info("'argazkia_atera' zerbitzua deitu da")

    # Abrir el puerto de la cámara
    self.cap_irudia = cv2.VideoCapture(self.port_irudia)
    if not self.cap_irudia.isOpened():
        self.get_logger().error(f"Ezin izan da kamera ireki (VideoCapture {self.
    port_irudia})")

    # Leer el valor del sensor de la cámara y cerrar la captura
    ret_color, frame_color = self.cap_irudia.read()
    self.cap_irudia.release()

    # ¿Lectura correcta?
    if ret_color:
        try:
            # Convertir foto de OpenCV BGR8 a sensor_msgs/Image de ROS2 y añadir marca
    temporal
            Ateratako irudia (OpenCV-ren BGR) ROS mezu bihurtzea
            img_msg = self.bridge.cv2_to_imgmsg(frame_color, encoding="bgr8")
            img_msg.header.stamp = self.get_clock().now().to_msg()

            # Añadir mensaje a la respuesta
            response.image = img_msg
            self.get_logger().info("Argazkia atera da.")
        except CvBridgeError as e:
            self.get_logger().error(f"Errorea argazkia ROS2 kodifikatzean {str(e)}")
    else:
        self.get_logger().error(f"Errorea argazkia ateratzerakoan (VideoCapture {self.
    port_irudia})")

    # Liberar recursos
    self.cap_irudia = None

    # Enviar respuesta
    return response
```

Código 10.13: Implementación de la función de respuesta del servidor del servicio para sacar una foto.

II.3. Cliente y solicitud del servicio para sacar una foto

```
# Solicitud del SERVICE /argazkia_atera en el nodo PERTZEPZIO_PROBA_WRAPPER

# Crear el mensaje de solicitud
request_argazkia = Argazkia.Request()

# Llamar al servicio
future = self.argazkia_bezeroa.call_async(request_argazkia)

# Esperar hasta recibir respuesta
rclpy.spin_until_future_complete(self, future)

# Guardar la respuesta
argazkia = future.result()

# Convertir sensor_msgs/Image a OpenCV
cv_image = self.bridge.imgmsg_to_cv2(argazkia.image, desired_encoding="bgr8")
```

Código 10.14: Solicitud del servicio desde el nodo *pertzepzio_proba_wrapper* para sacar una foto.

II.4. Desarrollo del modelo *CNN*

Implementación de la primera restricción geométrica (7.4.2.4)

```python
# Función para hacer zum centrado
def zoom_center(self, imagen, escala):
    altura, ancho = imagen.shape[:2]
    nuevo_ancho = int(ancho * escala)
    nueva_altura = int(altura * escala)
    inicio_x = (ancho - nuevo_ancho) // 2
    inicio_y = (altura - nueva_altura) // 2
    centro_crop = imagen[inicio_y:inicio_y + nueva_altura, inicio_x:inicio_x + nuevo_ancho]
    zoomed = cv2.resize(centro_crop, (ancho, altura), interpolation=cv2.INTER_LINEAR)
    return zoomed

# Aplicar un zum centradl del 57%
zoom_frame = self.zoom_center(frame, escala=0.57)

# Recorte del 30% inferior
altura_zoom = zoom_frame.shape[0] # Obtener tamaño vertical
recorte_vertical = zoom_frame[:int(altura_zoom * 0.70), :] # Guardar 70% superior

# Recorte del 20% lateral (por ambos lados)
ancho_recorte = recorte_vertical.shape[1] # Obtener tamaño horizontal
izquierda = int(ancho_recorte * 0.20) # Obtener límite lateral izquierdo
derecha = int(ancho_recorte * 0.80) # Obtener límite lateral derecho
final_frame = recorte_vertical[:, izquierda:derecha] # Guardar el 60% central.
```

Código 10.15: Implementación de la primera restricción geométrica.

Implementación de la segunda restricción geométrica (7.4.2.4)

```python
# Por cada contorno azul detectado
for contour in blue_contours:

    #  Obtener coordenadas y píxeles de la imagen
    x, y, w, h = cv2.boundingRect(contour)

    #  Si cumple la restricción --> es caja válida
    if 150 <= w <= 300 and 200 <= h <= 350:
        color_detected = "Azul"
        ...

# Por cada contorno rojo detectado
for contour in red_contours:

    #  Obtener coordenadas y píxeles de la imagen
    x, y, w, h = cv2.boundingRect(contour)

    #  Si cumple la restricción --> es caja válida
    if 150 <= w <= 300 and 200 <= h <= 350:
        color_detected = "Rojo"
        ...
```

Código 10.16: Implementación de la segunda restricción geométrica.

HSV y máscaras con *OpenCV*

```
# Convertir de BGR a HSV
final_hsv = cv2.cvtColor(final_frame, cv2.COLOR_BGR2HSV)

# Definir umbrales
blue_lower_bound = np.array([80, 50, 50])
blue_upper_bound = np.array([130, 255, 255])
red_lower_bound1 = np.array([0, 100, 100])
red_upper_bound1 = np.array([10, 255, 255])
red_lower_bound2 = np.array([160, 100, 100])
red_upper_bound2 = np.array([180, 255, 255])

# Definir máscaras
blue_mask = cv2.inRange(final_hsv, blue_lower_bound, blue_upper_bound)
red_mask1 = cv2.inRange(final_hsv, red_lower_bound1, red_upper_bound1)
red_mask2 = cv2.inRange(final_hsv, red_lower_bound2, red_upper_bound2)
red_mask = cv2.bitwise_or(red_mask1, red_mask2)

# Aplicar máscaras
blue_contours, _ = cv2.findContours(blue_mask.copy(), cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)
red_contours, _ = cv2.findContours(red_mask.copy(), cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)

# Por cada contorno azul detectado
for contour in blue_contours:
    x, y, w, h = cv2.boundingRect(contour)
    if 150 <= w <= 300 and 200 <= h <= 350:
        color_detected = "Azul"

        # Dibujar borde
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 3)
        ...

# Por cada contorno rojo detectado
for contour in red_contours:
    x, y, w, h = cv2.boundingRect(contour)
    if 150 <= w <= 300 and 200 <= h <= 350:
        color_detected = "Rojo"

        # Dibujar borde
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 3)
        ...
```

Código 10.17: Implementación y aplicación del modelo de color *HSV* y las máscaras con *OpenCV*.

Captura de imágenes para crear el *dataset*

```python
import cv2
import numpy as np
import os

cap = cv2.VideoCapture(2)

# Modificar para guardar automáticamente las imágenes en la carpeta y ruta correspondiente
numero = "8"
# Si no existe la ruta, crearla
if not os.path.exists(f'data/{numero}'):
    os.makedirs(f'data/{numero}')
# Función para guardar la imagen, dadas las coordenadas de los píxeles a guardar (no se guarda toda la imagen)
def save_region(image, x, y, w, h, color, count):
    region = image[y:y+h, x:x+w]
    filename = f"data/{numero}/{color}_{count}.png"
    cv2.imwrite(filename, region)

red_count = 0
while True:
    success, frame = cap.read()
    if success:
        # Voltear la imagen capturada
        frame = cv2.flip(frame, 1)

        # Obtener HSV
        frame_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Umbrales del color rojo para detección HSV
        red_lower_bound1 = np.array([0, 100, 100])
        red_upper_bound1 = np.array([10, 255, 255])
        red_lower_bound2 = np.array([160, 100, 100])
        red_upper_bound2 = np.array([180, 255, 255])

        # Máscara del color rojo
        red_mask1 = cv2.inRange(frame_hsv, red_lower_bound1, red_upper_bound1)
        red_mask2 = cv2.inRange(frame_hsv, red_lower_bound2, red_upper_bound2)
        red_mask = cv2.bitwise_or(red_mask1, red_mask2)

        # Aplicar máscara
        red_contours, _ = cv2.findContours(red_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        detected = False
        color_detected = "Color distractorio"
        # Por cada contorno rojo detectado
        for contour in red_contours:
            x, y, w, h = cv2.boundingRect(contour)
            if 50 <= w <= 500 and 100 <= h <= 600:
                # Dibujar rectángulo
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 3)
                detected = True
                color_detected = "Rojo"

                # Guardar región generada
                save_region(frame, x, y, w, h, 'rojo', red_count)
                red_count += 1
        cv2.putText(frame, color_detected, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
        cv2.imshow('Webcam', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Código 10.18: Capturar imágenes para crear el *dataset*.

Aplicación del algoritmo de *Canny* para la detección de bordes en el *dataset*

```python
import cv2
import os

# Directorios
input_dir = "data_cleaned"
output_dir = "data_canny"

# Función para recortar por todos los lados
def recortar_imagen(imagen, porcentaje_recorte=0.20):
    altura, ancho = imagen.shape[:2]
    recorte_vertical = int(altura * porcentaje_recorte)
    recorte_horizontal = int(ancho * porcentaje_recorte)
    return imagen[recorte_vertical:altura - recorte_vertical, recorte_horizontal:ancho - recorte_horizontal]

# Crear el directorio de salida si no existe
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Recorrer el árbol de directorios y archivos
for root, dirs, files in os.walk(input_dir):
    for file in files:
        if file.endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):
            # Ruta completa del archivo de entrada
            input_path = os.path.join(root, file)

            # Leer la imagen
            img = cv2.imread(input_path, 0)  # Leer en escala de grises

            if img is None:
                print(f"Error: No se pudo leer la imagen {input_path}")
                continue

            # Aplicar flip horizontal
            img = cv2.flip(img, 1)

            # Recortar la imagen
            img = recortar_imagen(img)

            # Aplicar detección de bordes con Canny
            canny = cv2.Canny(img, 20, 110)

            # Engrosar los contornos usando dilatación
            kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))  # Tamaño del kernel
            thickened_canny = cv2.dilate(canny, kernel, iterations=1)  # Aumentar iteraciones para engrosar más

            # Crear la estructura de carpetas en el directorio de salida
            relative_path = os.path.relpath(root, input_dir)  # Ruta relativa dentro del directorio de entrada
            output_subdir = os.path.join(output_dir, relative_path)
            if not os.path.exists(output_subdir):
                os.makedirs(output_subdir)

            # Ruta completa del archivo de salida
            output_path = os.path.join(output_subdir, file)

            # Guardar la imagen procesada
            cv2.imwrite(output_path, thickened_canny)
            print(f"Imagen procesada y guardada en: {output_path}")
```

Código 10.19: Aplicación del algoritmo *Canny* al *dataset* generado.

Generación de imágenes sintéticas

```python
import cv2
import numpy as np
import os
import random
from PIL import ImageFont, ImageDraw, Image

# Ruta a la fuente TrebuchetMS Bold (modificar según la ubicación real)
fuente_path = "TrebuchetMS-Bold.ttf"

# Crear carpetas para almacenar los dígitos
def crear_carpetas(base_dir="digitos_trebuchet"):
    if not os.path.exists(base_dir):
        os.makedirs(base_dir)
    for dígito in range(1, 10):
        carpeta = os.path.join(base_dir, str(dígito))
        os.makedirs(carpeta, exist_ok=True)

# Generar ruido en el fondo del cuadrado
def añadir_ruido_al_fondo(imagen, color_cuadrado):
    # Crear una máscara donde el color coincide con el color del cuadrado
    mask = (imagen[:, :, 0] == color_cuadrado[0]) & \
           (imagen[:, :, 1] == color_cuadrado[1]) & \
           (imagen[:, :, 2] == color_cuadrado[2])

    # Generar ruido aleatorio de baja intensidad
    ruido = np.random.randint(0, 50, imagen.shape, dtype=np.uint8)

    # Copiar la imagen original para no modificarla directamente
    imagen_con_ruido = imagen.copy()

    # Añadir el ruido sólo en las zonas fuera del cuadrado
    imagen_con_ruido[~mask] = cv2.add(imagen[~mask], ruido[~mask])
    return imagen_con_ruido

# Dibujar un cuadrado de color con un número blanco usando TrebuchetMS Bold
def dibujar_cuadrado_con_numero(color_cuadrado, dígito, fuente_path):
    # Crear una imagen negra de 250x250 píxeles y 3 canales de 8 bits cada uno
    imagen = np.zeros((250, 250, 3), dtype=np.uint8)

    # Dibujar un cuadrado relleno con el color especificado
    cv2.rectangle(imagen, (25, 25), (210, 210), color_cuadrado, -1)

    # Dibujar el texto con la fuente deseada
    texto = str(dígito)
    font = ImageFont.truetype(fuente_path, 100)
    img_pil = Image.fromarray(imagen)
    draw = ImageDraw.Draw(img_pil)

    # Calcula el tamaño del texto para centrarlo
    bbox = font.getbbox(texto)
    texto_ancho, texto_alto = bbox[2] - bbox[0], bbox[3] - bbox[1]
    texto_x = 25 + (210 - 25 - texto_ancho) // 2
    texto_y = 25 + (210 - 25 - texto_alto) // 2

    # Con cierta probabilidad, aplica un pequeño desplazamiento aleatorio al texto
    if random.random() > 0.7:
        offset_x = random.randint(-10, 10)
        offset_y = random.randint(-10, 10)
        texto_x = max(0, texto_x + offset_x)
        texto_y = max(texto_alto, texto_y + offset_y)

    # Dibujar el texto blanco sobre el cuadrado de color
    draw.text((texto_x, texto_y), texto, font=font, fill=(255, 255, 255, 0))

    # Convertir la imagen de vuelta a formato NumPy
    imagen = np.array(img_pil)
    return imagen

```

```
# Escalar imagen a un tamaño aleatorio para generar variaciones y centrarla en un lienzo de 250x250
def escalar_imagen(imagen, tamaño_min=63, tamaño_max=250):
    nuevo_tamaño = random.randint(tamaño_min, tamaño_max)
    imagen_escalada = cv2.resize(imagen, (nuevo_tamaño, nuevo_tamaño), interpolation=cv2.INTER_LINEAR)

    # Crear un lienzo negro de 250x250
    lienzo = np.zeros((250, 250, 3), dtype=np.uint8)
    y_offset = (250 - nuevo_tamaño) // 2
    x_offset = (250 - nuevo_tamaño) // 2

    # Colocar la imagen escalada en el centro del lienzo
    lienzo[y_offset:y_offset + nuevo_tamaño, x_offset:x_offset + nuevo_tamaño] = imagen_escalada
    return lienzo

# Añadir bordes detectados por Canny fuera del cuadrado de color
def añadir_bordes_canny_fuera(imagen, color_cuadrado):
    mask = (imagen[:, :, 0] == color_cuadrado[0]) & \
           (imagen[:, :, 1] == color_cuadrado[1]) & \
           (imagen[:, :, 2] == color_cuadrado[2])
    imagen_grises = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
    bordes = cv2.Canny(imagen_grises, 50, 150)
    imagen_resultante = imagen.copy()
    imagen_resultante[~mask] = cv2.add(imagen[~mask], cv2.merge([bordes, bordes, bordes])[~mask])
    return imagen_resultante

# Aplicar una transformación de perspectiva aleatoria a la imagen
def aplicar_perspectiva(imagen):
    # Obtener las dimensiones de la imagen
    filas, columnas = imagen.shape[:2]

    # Definir los puntos originales de las esquinas
    puntos_originales = np.float32([
        [0, 0], [columnas, 0], [0, filas], [columnas, filas]
    ])

    # Definir el margen máximo de desplazamiento para las esquinas
    margen = 40

    # Calcula nuevos puntos de destino aleatorios para la transformación
    puntos_transformados = np.float32([
        [random.randint(0, margen), random.randint(0, margen)],
        [columnas - random.randint(0, margen), random.randint(0, margen)],
        [random.randint(0, margen), filas - random.randint(0, margen)],
        [columnas - random.randint(0, margen), filas - random.randint(0, margen)]
    ])

    # Calcula la matriz de transformación de perspectiva
    matriz_perspectiva = cv2.getPerspectiveTransform(puntos_originales, puntos_transformados)

    # Aplica la transformación a la imagen
    imagen_perspectiva = cv2.warpPerspective(imagen, matriz_perspectiva, (columnas, filas))
    return imagen_perspectiva

# Recortar los bordes de la imagen
def recortar_imagen(imagen, porcentaje_recorte=0.20):
    altura, ancho = imagen.shape[:2]
    recorte_vertical = int(altura * porcentaje_recorte)
    recorte_horizontal = int(ancho * porcentaje_recorte)
    return imagen[recorte_vertical:altura - recorte_vertical, recorte_horizontal:ancho - recorte_horizontal]

# Generar las imágenes
def generar_imagenes(base_dir="digitos_trebuchet", num_imagenes_por_fuente=50):
    colores = [(255, 0, 0), (0, 0, 255)]
```

```
    for dígito in range(1, 10):
        for i in range(num_imagenes_por_fuente):
            color_cuadrado = random.choice(colores)
            imagen = dibujar_cuadrado_con_numero(color_cuadrado, dígito, fuente_path)
            imagen_con_bordes = añadir_bordes_canny_fuera(imagen, color_cuadrado)
            imagen_grises = cv2.cvtColor(imagen_con_bordes, cv2.COLOR_BGR2GRAY)
            imagen_canny = cv2.Canny(imagen_grises, 100, 200)
            imagen_recortada = recortar_imagen(imagen_canny)
            guardar_imagen(imagen_recortada, base_dir, dígito, i, perspectiva=False)
            imagen_con_perspectiva = aplicar_perspectiva(imagen_con_bordes)
            imagen_grises_perspectiva = cv2.cvtColor(imagen_con_perspectiva, cv2.
    COLOR_BGR2GRAY)
            imagen_canny_perspectiva = cv2.Canny(imagen_grises_perspectiva, 100, 200)
            imagen_recortada_perspectiva = recortar_imagen(imagen_canny_perspectiva)
            guardar_imagen(imagen_recortada_perspectiva, base_dir, dígito, i,
    perspectiva=True)

# Guardar la imagen en un fichero
def guardar_imagen(imagen, base_dir, dígito, indice, perspectiva):
    sufijo = "_perspectiva" if perspectiva else "_sin_perspectiva"
    carpeta = os.path.join(base_dir, str(dígito))
    os.makedirs(carpeta, exist_ok=True)
    nombre_archivo = f"{dígito}_{indice}{sufijo}.png"
    cv2.imwrite(os.path.join(carpeta, nombre_archivo), imagen)

crear_carpetas()
generar_imagenes()
print("Conjunto de datos generado con éxito.")
```

Código 10.20: Generación de imágenes sintéticas.

División del conjunto de datos en los tres subconjuntos

```python
import os
import shutil
import numpy as np

# Directorios de entrada y salida
input_dir = "drive/My Drive/bilbotiks/canny/DATA/sinSeparar"
train_output_dir = "drive/My Drive/bilbotiks/canny/DATA/train"
validation_output_dir = "drive/My Drive/bilbotiks/canny/DATA/validation"
test_output_dir = "drive/My Drive/bilbotiks/canny/DATA/test"

for dir in [train_output_dir, validation_output_dir, test_output_dir]:
    if not os.path.exists(dir):
        os.makedirs(dir)

# Proporciones para dividir los datos
train_ratio = 0.7
validation_ratio = 0.15
test_ratio = 0.15

# Recorrer el árbol de directorios y archivos
for class_name in os.listdir(input_dir):
    class_path = os.path.join(input_dir, class_name)
    if os.path.isdir(class_path):
        # Crear subcarpetas en los directorios de salida para cada clase
        train_class_dir = os.path.join(train_output_dir, class_name)
        validation_class_dir = os.path.join(validation_output_dir, class_name)
        test_class_dir = os.path.join(test_output_dir, class_name)

        for dir in [train_class_dir, validation_class_dir, test_class_dir]:
            if not os.path.exists(dir):
                os.makedirs(dir)

        # Obtener todos los archivos de la clase actual
        files = [file for file in os.listdir(class_path) if file.endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff'))]

        # Mezclar archivos de forma aleatoria
        np.random.shuffle(files)

        # Calcular el número de archivos para cada conjunto
        total_files = len(files)
        train_count = int(total_files * train_ratio)
        validation_count = int(total_files * validation_ratio)
        test_count = total_files - train_count - validation_count

        # Dividir los archivos en train, validation y test
        train_files = files[:train_count]
        validation_files = files[train_count:train_count + validation_count]
        test_files = files[train_count + validation_count:]

        # Copiar archivos a las carpetas correspondientes
        for file in train_files:
            shutil.copy(os.path.join(class_path, file), os.path.join(train_class_dir, file))
        for file in validation_files:
            shutil.copy(os.path.join(class_path, file), os.path.join(validation_class_dir, file))
        for file in test_files:
            shutil.copy(os.path.join(class_path, file), os.path.join(test_class_dir, file))

print("Archivos divididos en train, validation y test, manteniendo las clases.")
```

Código 10.21: División del *dataset* en *train*, *val* y *test*.

Arquitectura y entrenamiento del modelo *CNN*

```
# Definir las rutas a tus carpetas de datos
train_dir = "drive/My Drive/bilbotiks/canny/DATA/train"
validation_dir = "drive/My Drive/bilbotiks/canny/DATA/validation"
test_dir = "drive/My Drive/bilbotiks/canny/DATA/test"

# Definir el tamaño de las imágenes y la normalización
img_size = (64, 64)
batch_size = 64

# Creación de generadores de datos
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=False,
    fill_mode='nearest'
)

validation_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=False,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=False,
    fill_mode='nearest'
)

# Generadores de datos para entrenamiento, validación y prueba
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical'
)

model = Sequential()

# Capa Convolucional 1
```

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3)) # Ajustado a 0.3

# Capa Convolucional 2
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3)) # Ajustado a 0.3

# Capa Convolucional 3
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3)) # Ajustado a 0.3

# Aplanar las características extraídas
model.add(Flatten())

# Capa Densa con L2 Regularization
model.add(Dense(128, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.3)) # Ajustado a 0.3

# Capa Densa con L2 Regularization
model.add(Dense(64, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.3)) # Ajustado a 0.3

# Capa de salida para 9 clases
model.add(Dense(10, activation='softmax'))

# Compilación del modelo
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Callbacks
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr
    =0.0001)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=
    True)
model_checkpoint = ModelCheckpoint("drive/My Drive/bilbotiks/canny/model_0.3.keras",
    monitor='val_loss', save_best_only=True)

# Entrenamiento del modelo
history = model.fit(
    train_generator,
    epochs=100,
    validation_data=validation_generator,
    callbacks=[reduce_lr, early_stopping, model_checkpoint]
)
```

Código 10.22: Codificación de la arquitectura y del entrenamiento del modelo *CNN* generado.

Generar los gráficos de las métricas del modelo *CNN* en *MATLAB*

```
% Este script carga el archivo metrics.csv y grafica las métricas de entrenamiento.
% Las columnas del CSV deben ser: % epoch, accuracy, loss, val_accuracy, val_loss
%% Cargar datos
metrics = readtable('/MATLAB Drive/cnn_metrics/metricas.csv');

% Extraer las columnas
epochs    = metrics.epoch;
acc       = metrics.accuracy;
loss      = metrics.loss;
val_acc   = metrics.val_accuracy;
val_loss  = metrics.val_loss;

%% Gráfico combinado: Accuracy y Loss en subplots
figure('Name','Métricas de Entrenamiento y Validación','NumberTitle','off');

% Subplot 1: Accuracy
subplot(2,1,1)
plot(epochs, acc, 'bo-', 'LineWidth', 2, 'MarkerSize',6); hold on;
plot(epochs, val_acc, 'ro-', 'LineWidth', 2, 'MarkerSize',6);
xlabel('Epoch');
ylabel('Accuracy');
title('Accuracy: Entrenamiento vs. Validación');
legend('Entrenamiento', 'Validación','Location','southeast');
grid on;

% Subplot 2: Loss
subplot(2,1,2)
plot(epochs, loss, 'bo-', 'LineWidth', 2, 'MarkerSize',6); hold on;
plot(epochs, val_loss, 'ro-', 'LineWidth', 2, 'MarkerSize',6);
xlabel('Epoch');
ylabel('Loss');
title('Loss: Entrenamiento vs. Validación');
legend('Entrenamiento', 'Validación','Location','northeast');
grid on;

% Gráfico para Accuracy de Entrenamiento
figure('Name','Accuracy - Entrenamiento','NumberTitle','off');
plot(epochs, acc, 'b*-', 'LineWidth', 2, 'MarkerSize',6);
title('Accuracy - Entrenamiento');
xlabel('Epoch');
ylabel('Accuracy');
grid on;

% Gráfico para Accuracy de Validación
figure('Name','Accuracy - Validación','NumberTitle','off');
plot(epochs, val_acc, 'r*-', 'LineWidth', 2, 'MarkerSize',6);
title('Accuracy - Validación');
xlabel('Epoch');
ylabel('Accuracy');
grid on;

% Gráfico para Loss de Entrenamiento
figure('Name','Loss - Entrenamiento','NumberTitle','off');
plot(epochs, loss, 'b*-', 'LineWidth', 2, 'MarkerSize',6);
title('Loss - Entrenamiento');
xlabel('Epoch');
ylabel('Loss');
grid on;

% Gráfico para Loss de Validación
figure('Name','Loss - Validación','NumberTitle','off');
plot(epochs, val_loss, 'r*-', 'LineWidth', 2, 'MarkerSize',6);
title('Loss - Validación');
xlabel('Epoch');
ylabel('Loss');
grid on;
```

Código 10.23: Generar los gráficos de las métricas del modelo *CNN* en *MATLAB*.

II.5. Comprobación del funcionamiento del *LiDAR*

```
import os
import ydlidar
import time
import math

if __name__ == "__main__":
    ydlidar.os_init();
    laser = ydlidar.CYdLidar();
    ports = ydlidar.lidarPortList();
    port = "/dev/ydlidar";
    for key, value in ports.items():
        port = value;
    laser.setlidaropt(ydlidar.LidarPropSerialPort, port)
    laser.setlidaropt(ydlidar.LidarPropSerialBaudrate, 128000)
    laser.setlidaropt(ydlidar.LidarPropLidarType, ydlidar.TYPE_TRIANGLE)
    laser.setlidaropt(ydlidar.LidarPropDeviceType, ydlidar.YDLIDAR_TYPE_SERIAL)
    laser.setlidaropt(ydlidar.LidarPropScanFrequency, 10.0)
    laser.setlidaropt(ydlidar.LidarPropSampleRate, 5)
    laser.setlidaropt(ydlidar.LidarPropSingleChannel, True)
    laser.setlidaropt(ydlidar.LidarPropMaxAngle, 180.0)
    laser.setlidaropt(ydlidar.LidarPropMinAngle, -180.0)

    i = 0
    ret = laser.initialize();
    if ret:
        ret = laser.turnOn();
        scan = ydlidar.LaserScan()
        while ret and ydlidar.os_isOk() :
            r = laser.doProcessSimple(scan);
            if r:
                if scan.config.scan_time < 1.0:
                    timeNotZero = 1.0
                else:
                    timeNotZero = scan.config.scan_time
                print("Scan received[",scan.stamp,"]:",scan.points.size(),"ranges is ["
    ,1.0/timeNotZero,"]Hz");

                distance_0 = None
                distance_90 = None

                print(i)
                time.sleep(3)

                sorted_points = sorted(scan.points, key=lambda point: math.degrees(
    point.angle))

                for point in sorted_points:
                        print("angle:", math.degrees(point.angle), " range: ", point.
    range)

                i += 1
            else :
                print("Failed to get Lidar Data")
        laser.turnOff();
    laser.disconnecting();
```

Código 10.24: Comprobación del funcionamiento del *LiDAR*.

II.6. Generar los gráficos de la señal de control en *MATLAB*

```
% Abrir y leer el fichero .csv con los datos
tabla = readtable('/MATLAB Drive/pid/performance_0205_22.5_100.0_0.0_25.0_30.csv');

% Extraer las columnas
error_pid = tabla.Var1; %.Error
tiempo = tabla.Var6; %.Timestamp

% Crear la figura y graficar
figure;
plot(tiempo, error_pid, 'b-', 'LineWidth', 2);
xlabel('Tiempo (s)', 'FontSize', 12);
ylabel('Error', 'FontSize', 12);
title('Error en función del Tiempo', 'FontSize', 14);
grid on;
```

Código 10.25: Generar los gráficos de la señal de control en *MATLAB*.

II.7. Nodos

Servomotores

```python
import rclpy
from rclpy.node import Node
from rclpy.lifecycle import LifecycleNode, LifecycleState, TransitionCallbackReturn
from std_msgs.msg import Int16, Int16MultiArray
from adafruit_servokit import ServoKit
from bilbotiks_interfazeak.msg import ServoakMugitu

class ServoKontrolatzailea(LifecycleNode):
    def __init__(self):
        super().__init__('servo_kontrolatzailea')
        self.kit = None
        self.subscription = None
        self.active = None

        # Parametroak ezarri lehenetsitako balioak
        self.declare_parameter('actuation_range', 300)
        self.declare_parameter('pulse_width_range', (500, 2500))

        self.get_logger().info("servo_kontrolatzailea nodoa IN constructor")

    def on_configure(self, state: LifecycleState):
        self.get_logger().info("servo_kontrolatzailea nodoa IN on_configure")
        self.active = False

        # Parametroak irakurri YAML fitxerotik
        self.actuation_range = self.get_parameter('actuation_range').value
        self.pulse_width_range = tuple(self.get_parameter('pulse_width_range').value)

        # Configuración inicial de servos
        self.kit = ServoKit(channels=16)
        self.servoak_konfiguratu()

        return TransitionCallbackReturn.SUCCESS

    def on_cleanup(self, state: LifecycleState):
        self.get_logger().info("servo_kontrolatzailea nodoa IN on_cleanup")
        self.kit = None

        return TransitionCallbackReturn.SUCCESS

    def on_activate(self, state: LifecycleState):
        self.get_logger().info("servo_kontrolatzailea nodoa IN on_activate")
        self.active = True

        # /servoak_mugitu topikorako harpidetza konfiguratu
        self.subscription = self.create_subscription(
            ServoakMugitu,
            'servoak_mugitu',
            self.servoak_mugitu_callback,
            10
        )

        return super().on_activate(state)

    def on_deactivate(self, state: LifecycleState):
        self.get_logger().info("motorrak_roboclaw nodoa IN on_deactivate")
        self.subscription = None
        self.active = False

        return super().on_deactivate(state)

    def on_shutdown(self, state: LifecycleState):
        self.get_logger().info("motorrak_roboclaw nodoa IN on_shutdown")
        return TransitionCallbackReturn.SUCCESS
```

```
65
66      def servoak_konfiguratu(self):
67              for i in range(4):  # Control de los canales 0 a 3
68                  self.kit.servo[i].actuation_range = self.actuation_range
69                  self.kit.servo[i].set_pulse_width_range(*self.pulse_width_range)
70
71      def servoak_mugitu_callback(self, msg):
72          if self.active:
73              for i, angle in enumerate(msg.angeluak):
74                  if angle != -1:  # Ángulo válido
75                      self.kit.servo[i].angle = angle
76                      self.get_logger().info(f"Servomotorra {i} mugituta {angle:.2f}
    gradutara")
77
78  def main(args=None):
79      rclpy.init(args=args)
80      node = ServoKontrolatzailea()
81
82      try:
83          rclpy.spin(node)
84      except KeyboardInterrupt:
85          node.get_logger().info("Erabiltzaileak nodoa gelditu du.")
86      finally:
87          node.destroy_node()
88          rclpy.shutdown()
89
90  if __name__ == '__main__':
91      main()
```

Código 10.26: Código del nodo *servoak*.

Motores

```python
import rclpy
from rclpy.node import Node
from rclpy.lifecycle import LifecycleNode, LifecycleState, TransitionCallbackReturn
from geometry_msgs.msg import Twist
from bilbotiks_controller.roboclaw_3 import Roboclaw
from bilbotiks_interfazeak.msg import MotorrakMugitu #, MotorrenKodetzaileak

class MotorKontrolatzailea(LifecycleNode):
    def __init__(self):
        super().__init__('motor_kontrolatzailea')
        self.rc = None
        self.baud_rate = None
        self.motor_addresses = None
        self.publisher_ = None
        self.timer_ = None

        # Parametroak ezarri lehenetsitako balioak
        self.declare_parameter('baud_rate', 115200)
        self.declare_parameter('motorren_helbideak', [128, 129, 130])

        self.get_logger().info("motorrak_roboclaw nodoa IN constructor")

    def on_configure(self, state: LifecycleState):
        self.get_logger().info("motorrak_roboclaw nodoa IN on_configure")

        # Parametroak irakurri YAML fitxerotik
        self.baud_rate = self.get_parameter('baud_rate').value
        self.motor_addresses = self.get_parameter('motorren_helbideak').value
        self.get_logger().info(f"{self.motor_addresses}")

        for address in self.motor_addresses:
            self.rc = self.test_connection(address)
            if self.rc is None:
                self.get_logger().info("RC is None")
                #rclpy.shutdown()
        self.get_logger().info("    TOPIKO HARPIDETZAK: /motorrak_mugitu")

        self.motorrak_gelditu()
        return TransitionCallbackReturn.SUCCESS

    def on_cleanup(self, state: LifecycleState):
        self.get_logger().info("motorrak_roboclaw nodoa IN on_cleanup")
        self.baud_rate = None
        self.motor_addresses = None
        self.subscription = None

        return TransitionCallbackReturn.SUCCESS

    def on_activate(self, state: LifecycleState):
        self.get_logger().info("motorrak_roboclaw nodoa IN on_activate")

        # /motorrak_mugitu topikorako harpidetza konfiguratu
        self.subscription = self.create_subscription(
            MotorrakMugitu,
            'motorrak_mugitu',
            self.motorrak_mugitu_callback,
            10 # Buffer isatsaren tamaina
        )

        return super().on_activate(state)

    def on_deactivate(self, state: LifecycleState):
        self.get_logger().info("motorrak_roboclaw nodoa IN on_deactivate")
        self.motorrak_gelditu()
        self.subscription = None

        return super().on_deactivate(state)

```

```
69      def on_shutdown(self, state: LifecycleState):
70          self.get_logger().info("motorrak_roboclaw nodoa IN on_shutdown")
71          self.rc = None
72          return TransitionCallbackReturn.SUCCESS
73
74      def test_connection(self, address):
75          roboclaw0 = Roboclaw("/dev/serial0", self.baud_rate)
76          roboclaw1 = Roboclaw("/dev/serial1", self.baud_rate)
77          connected0 = roboclaw0.Open() == 1
78          connected1 = roboclaw1.Open() == 1
79          if connected0:
80              self.get_logger().info(f"Roboclaw {address} helbidera konektatuta /dev/
    serial0")
81              return roboclaw0
82          elif connected1:
83              self.get_logger().info(f"Roboclaw {address} helbidera konektatuta /dev/
    serial1")
84              return roboclaw1
85          else:
86              self.get_logger().info(f"Ezin da Roboclaw {address} helbidera konektatu, ez
     /dev/serial0 ezta /dev/serial1")
87          return None
88
89      def motorrak_gelditu(self):
90          for address in self.motor_addresses:
91              self.rc.ForwardM1(address, 0)
92              self.rc.ForwardM2(address, 0)
93          self.get_logger().info("Motorrak geldiarazi dira")
94
95      def motorrak_mugitu_callback(self, msg):
96          motorra_1 = int(msg.abiadurak[0])
97          motorra_2 = int(msg.abiadurak[1])
98          motorra_3 = int(msg.abiadurak[2])
99          motorra_4 = int(msg.abiadurak[3])
100         motorra_5 = int(msg.abiadurak[4])
101         motorra_6 = int(msg.abiadurak[5])
102
103         self.get_logger().info("Motorrak mugitzen hurrengo abiadurara:")
104
105         for address in self.motor_addresses:
106             if address == 128: # 2. eta 4. motorrak
107                 if motorra_2 < 0:
108                     self.rc.BackwardM1(address, motorra_2*(-1))
109                 else:
110                     self.rc.ForwardM1(address, motorra_2)
111
112                 self.get_logger().info(f"   Motorra 2 mugitzen {motorra_2} abiadurara")
113
114                 if motorra_4 < 0:
115                     self.rc.BackwardM2(address, motorra_4*(-1))
116                 else:
117                     self.rc.ForwardM2(address, motorra_4)
118
119                 self.get_logger().info(f"   Motorra 4 mugitzen {motorra_4} abiadurara")
120
121             if address == 129: # 5. eta 6. motorrak
122                 if motorra_5 < 0:
123                     self.rc.BackwardM1(address, motorra_5*(-1))
124                 else:
125                     self.rc.ForwardM1(address, motorra_5)
126                 self.get_logger().info(f"   Motorra 5 mugitzen {motorra_5} abiadurara")
127
128                 if motorra_6 < 0:
129                     self.rc.BackwardM2(address, motorra_6*(-1))
130                 else:
131                     self.rc.ForwardM2(address, motorra_6)
132                 self.get_logger().info(f"   Motorra 6 mugitzen {motorra_6} abiadurara")
133
134             if address == 130: # 1.go eta 3. motorrak
135                 if motorra_1 < 0:
```

```
136                     self.rc.BackwardM1(address, motorra_1*(-1))
137                 else:
138                     self.rc.ForwardM1(address, motorra_1)
139                 self.get_logger().info(f"   Motorra 1 mugitzen {motorra_1} abiadurara")
140 
141                 if motorra_3 < 0:
142                     self.rc.BackwardM2(address, motorra_3*(-1))
143                 else:
144                     self.rc.ForwardM2(address, motorra_3)
145                 self.get_logger().info(f"   Motorra 3 mugitzen {motorra_3} abiadurara")
146 
147 def main(args=None):
148     rclpy.init(args=args)
149     motor_controller = MotorKontrolatzailea()
150 
151     try:
152         rclpy.spin(motor_controller)
153     except KeyboardInterrupt:
154         motor_controller.get_logger().info("Erabiltzaileak nodoa gelditu du.")
155         if motor_controller.rc:
156             motor_controller.motorrak_gelditu()
157     finally:
158         motor_controller.destroy_node()
159         rclpy.shutdown()
160 
161 if __name__ == "__main__":
162     main()
```

Código 10.27: Código del nodo *motorrak_roboclaw*.

IMU

```python
import rclpy
from rclpy.node import Node
from rclpy.lifecycle import LifecycleNode, LifecycleState, TransitionCallbackReturn
from bilbotiks_interfazeak.msg import Imu
import board
import adafruit_bno055

class IMUArgitaratzailea(LifecycleNode):
    def __init__(self):
        super().__init__('imu_argitaratzailea')
        self.i2c = None
        self.sensor = None
        self.publisher_ = None
        self.timer_ = None

        self.get_logger().info("imu_argitaratzailea nodoa IN constructor")

    def on_configure(self, state: LifecycleState):
        self.get_logger().info("imu_argitaratzailea nodoa IN on_configure")

        # IMU sentsorea hasieratu
        self.i2c = board.I2C()  # SCL eta SDA erabili
        self.sensor = adafruit_bno055.BNO055_I2C(self.i2c)

        self.publisher_ = self.create_lifecycle_publisher(Imu, 'imu_datuak', 20)
        self.timer_ = self.create_timer(0.1, self.publish_imu_data)  # 100ms-tara (0.1 Hz) argitaratu

        return TransitionCallbackReturn.SUCCESS

    def on_cleanup(self, state: LifecycleState):
        self.get_logger().info("imu_argitaratzailea nodoa IN on_cleanup")
        self.i2c = None
        self.sensor = None
        self.destroy_lifecycle_publisher(self.publisher_)
        self.destroy_timer(self.timer_)

        return TransitionCallbackReturn.SUCCESS

    def on_activate(self, state: LifecycleState):
        self.get_logger().info("imu_argitaratzailea nodoa IN on_activate")

        self.timer_.reset()

        self.get_logger().info('IMU neurriak /imu_datuak argitaratzen 0.100s-tara', once=True)

        return super().on_activate(state)

    def on_deactivate(self, state: LifecycleState):
        self.get_logger().info("imu_argitaratzailea nodoa IN on_deactivate")

        self.timer_.cancel()

        return super().on_deactivate(state)

    def on_shutdown(self, state: LifecycleState):
        self.get_logger().info("imu_argitaratzailea nodoa IN on_shutdown")
        return TransitionCallbackReturn.SUCCESS

    def publish_imu_data(self):
        msg = Imu()

        # Tenperatura
        msg.tenperatura = self.sensor.temperature or 0.0

        # Magnetometroa
        magnetic = self.sensor.magnetic
```

```
        if magnetic:
            msg.magnetometroa.x = magnetic[0] or 0.0
            msg.magnetometroa.y = magnetic[1] or 0.0
            msg.magnetometroa.z = magnetic[2] or 0.0

        # Giroskopioa
        gyro = self.sensor.gyro
        if gyro:
            msg.giroskopioa.x = gyro[0] or 0.0
            msg.giroskopioa.y = gyro[1] or 0.0
            msg.giroskopioa.z = gyro[2] or 0.0

        # Euler angeluak
        euler_angles = self.sensor.euler
        if euler_angles:
            msg.euler_angeluak.x = euler_angles[0] or 0.0  # Roll
            msg.euler_angeluak.y = euler_angles[1] or 0.0  # Pitch
            msg.euler_angeluak.z = euler_angles[2] or 0.0  # Yaw

        # Kuaternioak
        quaternion = self.sensor.quaternion
        if quaternion:
            msg.kuaternioak.x = quaternion[0] or 0.0
            msg.kuaternioak.y = quaternion[1] or 0.0
            msg.kuaternioak.z = quaternion[2] or 0.0
            msg.kuaternioak.w = quaternion[3] or 0.0

        # Azelerazio lineala
        acceleration = self.sensor.acceleration
        if acceleration:
            msg.azelerazio_lineala.x = acceleration[0] or 0.0
            msg.azelerazio_lineala.y = acceleration[1] or 0.0
            msg.azelerazio_lineala.z = acceleration[2] or 0.0

        # Grabitatea
        gravity = self.sensor.gravity
        if gravity:
            msg.grabitatea.x = gravity[0] or 0.0
            msg.grabitatea.y = gravity[1] or 0.0
            msg.grabitatea.z = gravity[2] or 0.0

        # Publicar el mensaje
        self.publisher_.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = IMUArgitaratzailea()

    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info("Erabiltzaileak nodoa gelditu du.")
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Código 10.28: Código del nodo *imu*.

LiDAR

```
import rclpy
from rclpy.node import Node
from rclpy.lifecycle import LifecycleNode, LifecycleState, TransitionCallbackReturn
import ydlidar
from sensor_msgs.msg import LaserScan
import math

class LidarArgitaratzailea(LifecycleNode):
    def __init__(self):
        super().__init__('lidar_argitaratzailea')
        self.lidar = None
        self.port = None
        self.baudrate = None
        self.lidar_type = None
        self.device_type = None
        self.frame_id = None
        self.scan_frequency = None
        self.sample_rate = None
        self.angle_min = None
        self.angle_max = None
        self.range_min = None
        self.range_max = None
        self.single_channel = None

        self.publisher_ = None
        self.timer_ = None

        self.active = None

        # Parametroak
        self.declare_parameter("port", "/dev/ydlidar")
        self.declare_parameter("baudrate", 128000)
        self.declare_parameter("lidar_type", ydlidar.TYPE_TRIANGLE)
        self.declare_parameter("device_type", ydlidar.YDLIDAR_TYPE_SERIAL)
        self.declare_parameter("frame_id", "laser_frame")
        self.declare_parameter("scan_frequency", 10.0) # 6 - 12 Hz
        self.declare_parameter("sample_rate", 5)
        self.declare_parameter("angle_min", -180.0)
        self.declare_parameter("angle_max", 180.0)
        self.declare_parameter("range_min", 0.12)
        self.declare_parameter("range_max", 10.0)
        self.declare_parameter("singleChannel", True)

        self.get_logger().info("lidar_argitaratzailea nodoa IN constructor")

    def on_configure(self, state: LifecycleState):
        self.get_logger().info("lidar_argitaratzailea nodoa IN on_configure")

        self.port = self.get_parameter("port").get_parameter_value().string_value
        self.get_logger().info(f'{self.port}')
        self.baudrate = self.get_parameter("baudrate").get_parameter_value().
    integer_value
        self.lidar_type = self.get_parameter("lidar_type").get_parameter_value().
    integer_value
        self.device_type = self.get_parameter("device_type").get_parameter_value().
    integer_value
        self.frame_id = self.get_parameter("frame_id").get_parameter_value().
    string_value
        self.scan_frequency = self.get_parameter("scan_frequency").get_parameter_value
    ().double_value
        self.sample_rate = self.get_parameter("sample_rate").get_parameter_value().
    integer_value
        self.angle_min = self.get_parameter("angle_min").get_parameter_value().
    double_value
        self.angle_max = self.get_parameter("angle_max").get_parameter_value().
    double_value
        self.range_min = self.get_parameter("range_min").get_parameter_value().
    double_value
```

```
60          self.range_max = self.get_parameter("range_max").get_parameter_value().
    double_value
61          self.single_channel = self.get_parameter("singleChannel").get_parameter_value()
    .bool_value
62
63          # Argitaratzailea
64          self.publisher_ = self.create_lifecycle_publisher(LaserScan, 'lidar_datuak',
    10)
65          self.timer_ = self.create_timer(0.2, self.publish_lidar_data)  # 100ms-tara
    (0.1 Hz) argitaratu
66
67          return TransitionCallbackReturn.SUCCESS
68
69      def on_cleanup(self, state: LifecycleState):
70          self.get_logger().info("lidar_argitaratzailea nodoa IN on_cleanup")
71          self.lidar = None
72          self.port = None
73          self.baudrate = None
74          self.lidar_type = None
75          self.device_type = None
76          self.frame_id = None
77          self.scan_frequency = None
78          self.sample_rate = None
79          self.angle_min = None
80          self.angle_max = None
81          self.range_min = None
82          self.range_max = None
83          self.single_channel = None
84          self.active = None
85          self.destroy_lifecycle_publisher(self.publisher_)
86          self.destroy_timer(self.timer_)
87
88          return TransitionCallbackReturn.SUCCESS
89
90      def on_activate(self, state: LifecycleState):
91          self.get_logger().info("lidar_argitaratzailea nodoa IN on_activate")
92
93          # Lidar hasiarazi
94          self.initialize_lidar()
95
96          self.timer_.reset()
97
98          self.get_logger().info('LIDAR neurriak /lidar_datuak argitaratzen 0.100s-tara',
     once=True)
99
100         return super().on_activate(state)
101
102     def on_deactivate(self, state: LifecycleState):
103         self.get_logger().info("lidar_argitaratzailea nodoa IN on_deactivate")
104         self.active = not (self.lidar.turnOff())
105         self.timer_.cancel()
106
107         return super().on_deactivate(state)
108
109     def on_shutdown(self, state: LifecycleState):
110         self.get_logger().info("lidar_argitaratzailea nodoa IN on_shutdown")
111         return TransitionCallbackReturn.SUCCESS
112
113     def initialize_lidar(self):
114         ydlidar.os_init()
115         self.lidar = ydlidar.CYdLidar()
116         self.lidar.setlidaropt(ydlidar.LidarPropSerialPort, self.port)
117         self.lidar.setlidaropt(ydlidar.LidarPropSerialBaudrate, self.baudrate)
118         self.lidar.setlidaropt(ydlidar.LidarPropLidarType, self.lidar_type)
119         self.lidar.setlidaropt(ydlidar.LidarPropDeviceType, self.device_type)
120         self.lidar.setlidaropt(ydlidar.LidarPropScanFrequency, self.scan_frequency)
121         self.lidar.setlidaropt(ydlidar.LidarPropSampleRate, self.sample_rate)
122         self.lidar.setlidaropt(ydlidar.LidarPropSingleChannel, self.single_channel)
123
124         if not self.lidar.initialize():
```

```
            self.get_logger().error("LiDAR initialization failed")
        else:
            self.active = self.lidar.turnOn()

    def publish_lidar_data(self):
        if self.active:
            scan = ydlidar.LaserScan()
            if self.lidar.doProcessSimple(scan):
                msg = LaserScan()
                msg.header.stamp = self.get_clock().now().to_msg()
                msg.header.frame_id = self.frame_id
                msg.angle_min = math.degrees(scan.config.min_angle)
                msg.angle_max = math.degrees(scan.config.max_angle)
                msg.angle_increment = math.degrees(scan.config.angle_increment)
                msg.scan_time = scan.config.scan_time
                msg.range_min = scan.config.min_range
                msg.range_max = scan.config.max_range

                sorted_points = sorted(scan.points, key=lambda point: math.degrees(
    point.angle))
                msg.ranges = [p.range for p in sorted_points]
                msg.intensities = [p.intensity for p in scan.points]

                self.publisher_.publish(msg)
            else:
                self.get_logger().error("ERROR lidar datuak lortzerakoan")

def main(args=None):
    rclpy.init(args=args)
    node = LidarArgitaratzailea()

    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info("Erabiltzaileak nodoa gelditu du.")

        if node.active:
            node.lidar.turnOff()
            node.lidar = None
            node.port = None
            node.baudrate = None
            node.lidar_type = None
            node.device_type = None
            node.frame_id = None
            node.scan_frequency = None
            node.sample_rate = None
            node.angle_min = None
            node.angle_max = None
            node.range_min = None
            node.range_max = None
            node.single_channel = None
            node.active = None
            node.destroy_lifecycle_publisher(node.publisher_)
            node.destroy_timer(node.timer_)
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Código 10.29: Código del nodo *lidar*.

Cámara

```
import rclpy
from rclpy.lifecycle import LifecycleNode, LifecycleState, TransitionCallbackReturn
from sensor_msgs.msg import Image
import cv2
from cv_bridge import CvBridge, CvBridgeError
from bilbotiks_interfazeak.srv import Argazkia

class KameraArgitaratzailea(LifecycleNode):
    def __init__(self):
        super().__init__('kamera_argitaratzailea')

        self.cap_irudia = None       # Kamera irudia
        self.cap_depth = None         # Kamera sakonera

        self.publisher_irudia = None  # Irudia argitaratzeko objektua
        self.publisher_depth = None   # Sakonera argitaratzeko objektua
        self.timer_ = None
        self.srv_argazkia_atera = None
        self.bridge = CvBridge()      # OpenCV - ROS konbertsioa

        # Parametroak
        self.port_irudia = None
        self.port_sakonera = None
        self.declare_parameter("port_irudia", 0)
        self.declare_parameter("port_sakonera", 2)

        self.get_logger().info("kamera_argitaratzailea nodoa IN constructor")

    def on_configure(self, state: LifecycleState):
        self.get_logger().info("kamera_argitaratzailea nodoa IN on_configure")

        # Parametroak
        self.port_irudia = self.get_parameter("port_irudia").get_parameter_value().
    integer_value
        self.get_logger().info(f'VideoCapture IMAGE: {self.port_irudia}')

        self.srv_argazkia_atera = self.create_service(Argazkia, 'argazkia_atera', self.
    argazkia_atera)

        # Argitaratzaileak sortu
        self.publisher_irudia = self.create_lifecycle_publisher(Image, '/kamera_irudia'
    , 10)

        # Argitaratzaileen tenporizadoreak
        self.timer_ = self.create_timer(1.0/30.0, self.publish_camera_data)
        self.timer_.cancel()

        return TransitionCallbackReturn.SUCCESS

    def on_cleanup(self, state: LifecycleState):
        self.get_logger().info("kamera_argitaratzailea nodoa IN on_cleanup")

        self.destroy_service(self.srv_argazkia_atera)
        self.srv_argazkia_atera = None

        # Kamera askatu
        if self.cap_irudia is not None:
            self.cap_irudia.release()
            self.cap_irudia = None

        # Argitaratzaileak eta tenporizadorea ezabatu
        self.destroy_lifecycle_publisher(self.publisher_irudia)
        self.destroy_timer(self.timer_)

        return TransitionCallbackReturn.SUCCESS

    def on_activate(self, state: LifecycleState):
        self.get_logger().info("kamera_argitaratzailea nodoa IN on_activate")
```

```
66
67          # Argazkia ateratzeko zerbitzua itzali
68          self.destroy_service(self.srv_argazkia_atera)
69          self.srv_argazkia_atera = None
70
71          # Kamera irudia ireki
72          self.cap_irudia = cv2.VideoCapture(self.port_irudia)
73          if not self.cap_irudia.isOpened():
74              self.get_logger().error("No se pudo abrir la cámara de imagen (VideoCapture
     0)")
75              return TransitionCallbackReturn.FAILURE
76
77          # Tenporizadorea berritu
78          self.timer_.reset()
79          return super().on_activate(state)
80
81      def on_deactivate(self, state: LifecycleState):
82          self.get_logger().info("kamera_argitaratzailea nodoa IN on_deactivate")
83
84          # Tenporizadorea gelditu
85          self.timer_.cancel()
86
87          # Argazkia ateratzeko zerbitzua piztu
88          self.srv_argazkia_atera = self.create_service(Argazkia, 'argazkia_atera', self.
    argazkia_atera)
89
90          return super().on_deactivate(state)
91
92      def on_shutdown(self, state: LifecycleState):
93          self.get_logger().info("kamera_argitaratzailea nodoa IN on_shutdown")
94          return TransitionCallbackReturn.SUCCESS
95
96      def publish_camera_data(self):
97          # Leer datos de la cámara de imagen
98          ret_color, frame_color = self.cap_irudia.read()
99          # Leer datos del sensor de profundidad
100         #ret_depth, frame_depth = self.cap_depth.read()
101
102         now = self.get_clock().now().to_msg()
103
104         if ret_color:
105             try:
106                 # Convertir la imagen capturada (BGR de OpenCV) a mensaje ROS
107                 img_msg = self.bridge.cv2_to_imgmsg(frame_color, encoding="bgr8")
108                 img_msg.header.stamp = now
109                 self.publisher_irudia.publish(img_msg)
110             except CvBridgeError as e:
111                 self.get_logger().error(f"Errorea kolore-irudiaren bihurketan: {str(e)}
    ")
112         else:
113             self.get_logger().error("Errorea kameraren irudia atzitzean (VideoCapture
    0)")
114
115     def argazkia_atera(self, request, response):
116         self.get_logger().info("'argazkia_atera' zerbitzua deitu da")
117
118         self.cap_irudia = cv2.VideoCapture(self.port_irudia)
119         if not self.cap_irudia.isOpened():
120             self.get_logger().error(f"Ezin izan da kamera ireki (VideoCapture {self.
    port_irudia})")
121
122         ret_color, frame_color = self.cap_irudia.read()
123         self.cap_irudia.release()
124         if ret_color:
125             try:
126                 # Ateratako irudia (OpenCV-ren BGR) ROS mezu bihurtzea
127                 img_msg = self.bridge.cv2_to_imgmsg(frame_color, encoding="bgr8")
128                 # Nahi izanez gero, timestamp bat eslei dakioke header-ari
129                 img_msg.header.stamp = self.get_clock().now().to_msg()
130                 response.image = img_msg
```

```
131                 self.get_logger().info("Argazkia atera da.")
132             except CvBridgeError as e:
133                 self.get_logger().error(f"Errorea argazkia ROS2 kodifikatzean {str(e)}"
    )
134         else:
135             self.get_logger().error(f"Errorea argazkia ateratzerakoan (VideoCapture {
    self.port_irudia})")
136         self.cap_irudia = None
137 
138         return response
139 
140 def main(args=None):
141     rclpy.init(args=args)
142     node = KameraArgitaratzailea()
143 
144     try:
145         rclpy.spin(node)
146     except KeyboardInterrupt:
147         node.get_logger().info("Erabiltzaileak nodoa gelditu du.")
148     finally:
149         node.destroy_node()
150         rclpy.shutdown()
151 
152 if __name__ == '__main__':
153     main()
```

Código 10.30: Código del nodo *kamera*.

Wrapper de la prueba de percepción

```
##############################################################################
# ROS2 imports...
import rclpy
from rclpy.node import Node
from rclpy.callback_groups import ReentrantCallbackGroup
from rclpy.action import ActionServer
from rclpy.executors import MultiThreadedExecutor
from lifecycle_msgs.srv import ChangeState
from lifecycle_msgs.msg import Transition

from sensor_msgs.msg import Image
from bilbotiks_interfazeak.msg import MotorrakMugitu, ServoakMugitu, Imu
from bilbotiks_interfazeak.action import Bira360
from bilbotiks_interfazeak.srv import KoloreaZenbakia, Argazkia

# Liburutegien imports...
import cv2
from cv_bridge import CvBridge, CvBridgeError
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array
##############################################################################

##############################################################################
# Desactivar el uso de la GPU para TensorFlow
tf.config.set_visible_devices([], 'GPU')
##############################################################################

##############################################################################
class PertzepzioProbaWrapper(Node):
    def __init__(self):
        super().__init__('pertzepzio_proba_wrapper')

        # Behar diren klase objektuak hasiarazi
        self.change_state = None
        self.imu_datuak = None
        self.bridge = CvBridge()

        # Parametroak
        self.declare_parameter('cnn_modelua', '/home/bilbotiks/moverRover/bilbotiks_ws/
    install/bilbotiks_controller/share/bilbotiks_controller/config//modelo_100epochs.
    keras')
        self.declare_parameter('servoak_360', [90, 65, 105, 65])
        self.declare_parameter('abiadura_azkar', 30)
        self.declare_parameter('abiadura_motel', 13)
        self.servo_angeluak_360 = self.get_parameter('servoak_360').value
        self.model = load_model(self.get_parameter('cnn_modelua').get_parameter_value()
    .string_value)
        self.abiadura_azkar = self.get_parameter('abiadura_azkar').get_parameter_value
    ().integer_value
        self.abiadura_motel = self.get_parameter('abiadura_motel').get_parameter_value
    ().integer_value
        self.class_labels = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "NaN"]

        # Paralel callbacks
        self.callback_group = ReentrantCallbackGroup()

        # Argitaratzaileak (PUBLISHERS)
        self.motor_publisher = self.create_publisher(MotorrakMugitu, 'motorrak_mugitu',
     10)
        self.servo_publisher = self.create_publisher(ServoakMugitu, 'servoak_mugitu',
    10)

        # Harpidetzak (SUBSCRIBERS)
        self.imu_subscription = self.create_subscription(
            Imu,
            'imu_datuak',
```

```
            self.imu_callback,
            20,
            callback_group=self.callback_group
        )

        # Zerbitzu zerbitzaria (SERVICE SERVER)
        self.argazkia_zerbitzaria = self.create_service(KoloreaZenbakia, '
    koloreaZenbakia_iragarri', self.koloreaZenbakia_iragarri, callback_group=self.
    callback_group)

        # Zerbitzu bezeroa (SERVICE CLIENT)
        self.argazkia_bezeroa = self.create_client(Argazkia, 'argazkia_atera')

        # Ekintza zerbitzaria (ACTION SERVER)
        self.action_server = ActionServer(
            self,
            Bira360,
            'pertzepzio_proba_360bira',
            self.bira360_callback,
            callback_group=self.callback_group
        )

        self.get_logger().info('pertzepzio_proba_wrapper nodoa hasita.')

    ##########################################################################
    # IRUDIAREN LAGUNTZA FUNTZIOAK
    def zoom_center(self, imagen, escala):
        altura, ancho = imagen.shape[:2]
        nuevo_ancho = int(ancho * escala)
        nueva_altura = int(altura * escala)
        inicio_x = (ancho - nuevo_ancho) // 2
        inicio_y = (altura - nueva_altura) // 2
        centro_crop = imagen[inicio_y:inicio_y + nueva_altura, inicio_x:inicio_x +
    nuevo_ancho]
        zoomed = cv2.resize(centro_crop, (ancho, altura), interpolation=cv2.
    INTER_LINEAR)
        return zoomed

    def load_and_preprocess_image(self, image, target_size=(64, 64)):
        canny = cv2.Canny(image, 20, 110)
        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
        thickened_canny = cv2.dilate(canny, kernel, iterations=1)
        image_proc = cv2.cvtColor(thickened_canny, cv2.COLOR_BGR2RGB)
        image_proc = cv2.resize(image_proc, target_size)
        image_proc = img_to_array(image_proc)
        image_proc = np.expand_dims(image_proc, axis=0)
        image_proc = image_proc / 255.0
        return image_proc

    def predic_digit(self, model, image, class_labels):
        prediction = model.predict(image, verbose=0)
        predicted_class = np.argmax(prediction)
        print("Prediction:", class_labels[predicted_class])
        return class_labels[predicted_class]
    ##########################################################################

    ##########################################################################
    # MUGIMENDUAREN LAGUNTZA FUNTZIOAK
    def servoak_kokatu(self):
        msg = ServoakMugitu()
        msg.angeluak = self.servo_angeluak_360
        self.servo_publisher.publish(msg)
        self.get_logger().info(f'Moviendo servos a: {msg.angeluak}')

    def motorrak_mugitu(self, abiadura):
        msg = MotorrakMugitu()
        msg.abiadurak = abiadura
        self.motor_publisher.publish(msg)

    def motorrak_gelditu(self):
```

```
        msg = MotorrakMugitu()
        msg.abiadurak = [0, 0, 0, 0, 0, 0]
        self.motor_publisher.publish(msg)
    ##########################################################################

    ##########################################################################
    # LIFECYCLE nodoen trantzisioak kudeatzeko
    def nodoa_kudeatu(self, nodoa, transition_id):
        # Lifecycle nodoen egoera kudeatzeko funtzioa
        self.get_logger().info(f'/{nodoa}/change_state')
        self.change_state = self.create_client(ChangeState, f'/{nodoa}/change_state')
        while not self.change_state.wait_for_service(timeout_sec=1.0):
            self.get_logger().info(f'/{nodoa}/change_state zerbitzua prest egoteko zain ...')

        request = ChangeState.Request()
        request.transition.id = transition_id
        future = self.change_state.call_async(request)
        rclpy.spin_until_future_complete(self, future)
        if future.result() is not None:
            self.get_logger().info(f'Trantsizioa ongi burutu da: {nodoa}, {transition_id}')
        else:
            self.get_logger().error('Errorea trantsizioa egitean.')
    ##########################################################################

    ##########################################################################
    # IMU harpidetza (IMU SUBSCRIBER)
    def imu_callback(self, msg: Imu):
        self.imu_datuak = msg
        self.imu_datu_berriak = True
    ##########################################################################

    ##########################################################################
    # ZERBITZUAREN ZERBITZARIA (SERVICE SERVER) Kolorea eta zenbakia iragartzeko prozesua
    def koloreaZenbakia_iragarri(self, request, response):
        self.get_logger().info("koloreaZenbakia_iragarri zerbitzua deitu da.")

        # Argazkia atera
        self.nodoa_kudeatu('kamera_argitaratzailea', Transition.TRANSITION_CONFIGURE) # Kamera aktibatu
        request_argazkia = Argazkia.Request()                                          # Argazkia zerbitzu eskaeraren mezua sortu
        future = self.argazkia_bezeroa.call_async(request_argazkia)                    #
        rclpy.spin_until_future_complete(self, future)
        argazkia = future.result()
        self.nodoa_kudeatu('kamera_argitaratzailea', Transition.TRANSITION_CLEANUP)

        # Argazkia iragarri
        if argazkia.image is None:
            self.get_logger().error("No hay frame disponible para procesar")
            response.kolorea = -1
            response.zenbakia = -1
            return response

        # Convertir sensor_msgs/Image a imagen OpenCV
        cv_image = self.bridge.imgmsg_to_cv2(argazkia.image, desired_encoding="bgr8")
        frame = cv2.flip(cv_image, 1)
        zoom_frame = self.zoom_center(frame, escala=0.57)
        altura_zoom = zoom_frame.shape[0]
        recorte_vertical = zoom_frame[:int(altura_zoom * 0.70), :]
        ancho_recorte = recorte_vertical.shape[1]
        izquierda = int(ancho_recorte * 0.20)
        derecha = int(ancho_recorte * 0.80)
        final_frame = recorte_vertical[:, izquierda:derecha]
        final_frame = cv2.flip(final_frame, 1)

        final_hsv = cv2.cvtColor(final_frame, cv2.COLOR_BGR2HSV)

```

```
193         blue_lower_bound = np.array([80, 50, 50])
194         blue_upper_bound = np.array([130, 255, 255])
195         red_lower_bound1 = np.array([0, 100, 100])
196         red_upper_bound1 = np.array([10, 255, 255])
197         red_lower_bound2 = np.array([160, 100, 100])
198         red_upper_bound2 = np.array([180, 255, 255])
199 
200         blue_mask = cv2.inRange(final_hsv, blue_lower_bound, blue_upper_bound)
201         red_mask1 = cv2.inRange(final_hsv, red_lower_bound1, red_upper_bound1)
202         red_mask2 = cv2.inRange(final_hsv, red_lower_bound2, red_upper_bound2)
203         red_mask = cv2.bitwise_or(red_mask1, red_mask2)
204 
205         blue_contours, _ = cv2.findContours(blue_mask.copy(), cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)
206         red_contours, _ = cv2.findContours(red_mask.copy(), cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)
207 
208         detected = False
209         color_detected = "Ninguno"
210         predicted_number = "NaN"
211 
212         self.get_logger().error("Antes de contornos")
213 
214         for contour in blue_contours:
215             x, y, w, h = cv2.boundingRect(contour)
216             if 150 <= w <= 300 and 200 <= h <= 350:
217                 color_detected = "Azul"
218                 cropped_image = final_frame[y:y + h, x:x + w]
219                 preprocessed_img = self.load_and_preprocess_image(cropped_image)
220                 predicted_number = self.predic_digit(self.model, preprocessed_img, self
    .class_labels)
221                 detected = True
222                 break
223 
224         if not detected:
225             for contour in red_contours:
226                 x, y, w, h = cv2.boundingRect(contour)
227                 if 150 <= w <= 300 and 200 <= h <= 350:
228                     color_detected = "Rojo"
229                     cropped_image = final_frame[y:y + h, x:x + w]
230                     preprocessed_img = self.load_and_preprocess_image(cropped_image)
231                     predicted_number = self.predic_digit(self.model, preprocessed_img,
    self.class_labels)
232                     detected = True
233                     break
234 
235         # Convertir sensor_msgs/Image a imagen OpenCV
236         try:
237             # Guarda la imagen en un fichero (por ejemplo, "capturada.jpg")
238             filename = "capturadaWrapper.jpg"
239             if cv2.imwrite(filename, cv_image):
240                 self.get_logger().info(f"Imagen guardada en {filename}")
241         except CvBridgeError as e:
242             self.get_logger().error("Error al convertir la imagen: {}".format(e))
243 
244         self.get_logger().info(f"Detección -> Color: {color_detected}, Número: {
    predicted_number}")
245         # Mostrar la imagen usando OpenCV
246 
247         color_val = {"Azul": 1, "Rojo": 2}.get(color_detected, -1)
248 
249         try:
250             number_val = int(predicted_number) if predicted_number != "NaN" else -1
251         except Exception as e:
252             self.get_logger().error(f"Error al parsear el dígito: {e}")
253             number_val = -1
254 
255         self.get_logger().info(f"Detección -> Color: {color_detected} ({color_val}),
    Número: {predicted_number} ({number_val})")
256 
```

```
        response.kolorea = color_val
        response.zenbakia = number_val
        return response
    ##########################################################################

    ##########################################################################
    # EKINTZAREN ZERBITZARIA (ACTION SERVER) Bira egiteko prozesua
    def bira360_callback(self, goal_handle):
        self.nodoa_kudeatu('imu_argitaratzailea', Transition.TRANSITION_CONFIGURE)
        self.nodoa_kudeatu('imu_argitaratzailea', Transition.TRANSITION_ACTIVATE)
        self.nodoa_kudeatu('motorrak_roboclaw', Transition.TRANSITION_CONFIGURE)
        self.nodoa_kudeatu('motorrak_roboclaw', Transition.TRANSITION_ACTIVATE)
        self.nodoa_kudeatu('servoak', Transition.TRANSITION_CONFIGURE)
        self.nodoa_kudeatu('servoak', Transition.TRANSITION_ACTIVATE)

        # Ekintzaren jomuga jaso (ACTION GOAL)
        noranzkoa = goal_handle.request.noranzkoa  # 0: ezkerra, 1: eskuina
        bira_kopurua = goal_handle.request.zenbakia

        self.get_logger().info('Kolorea-Zenbakia ekintzaren jomuga:')
        self.get_logger().info(f'   noranzkoa={noranzkoa}')
        self.get_logger().info(f'   bira kopurua={bira_kopurua}')

        emandako_birak = 0
        hasierako_angelua = self.imu_datuak.euler_angeluak.x
        self.get_logger().info(f'Hasierako angelua: {hasierako_angelua}')
        oraingo_angelua = hasierako_angelua
        anterior = hasierako_angelua
        self.servoak_kokatu()

        if noranzkoa == 0:
            self.motorrak_mugitu([self.abiadura_azkar, self.abiadura_azkar, self.
    abiadura_azkar*(-1), self.abiadura_azkar*(-1), self.abiadura_azkar, self.
    abiadura_azkar])
        else:
            self.motorrak_mugitu([self.abiadura_azkar*(-1), self.abiadura_azkar*(-1),
    self.abiadura_azkar, self.abiadura_azkar, self.abiadura_azkar*(-1), self.
    abiadura_azkar*(-1)])

        angulo_total = 0
        desfase = 10

        feedback_msg = Bira360.Feedback()
        while angulo_total < (360 * (bira_kopurua-0.2))-desfase:
                rclpy.spin_once(self, timeout_sec=0.1) #?

                if self.imu_datu_berriak:
                    oraingo_angelua = self.imu_datuak.euler_angeluak.x if hasattr(self.
    imu_datuak, 'euler_angeluak') else oraingo_angelua
                    self.new_imu_data = False

                    # Publicar feedback al cliente
                    feedback_msg.oraingo_angelua = oraingo_angelua
                    feedback_msg.bira_kopurua = emandako_birak
                    self.get_logger().info(f'Oraingo angelua: {feedback_msg.
    oraingo_angelua}')
                    goal_handle.publish_feedback(feedback_msg)

                    # Calcula la diferencia entre actual y anterior considerando el
    ciclo
                    if oraingo_angelua != 0.0 and oraingo_angelua > 0:
                        if noranzkoa:
                            diferencia = (oraingo_angelua - anterior + 360) % 360
                        else:
                            diferencia = (anterior - oraingo_angelua + 360) % 360
                        anterior = oraingo_angelua  # Actualiza la lectura anterior

                        angulo_total += diferencia

                    self.get_logger().info(f'TOTAL: {angulo_total}')
```

```
320
321         if noranzkoa:
322             self.motorrak_mugitu([self.abiadura_motel*(-1), self.abiadura_motel*(-1),
    self.abiadura_motel, self.abiadura_motel, self.abiadura_motel*(-1), self.
    abiadura_motel*(-1)])
323         else:
324             self.motorrak_mugitu([self.abiadura_motel, self.abiadura_motel, self.
    abiadura_motel*(-1), self.abiadura_motel*(-1), self.abiadura_motel, self.
    abiadura_motel])
325
326         while angulo_total < (360 * bira_kopurua) - desfase:
327
328             rclpy.spin_once(self, timeout_sec=0.1) #?
329
330             if self.imu_datu_berriak:
331                 oraingo_angelua = self.imu_datuak.euler_angeluak.x if hasattr(self.
    imu_datuak, 'euler_angeluak') else oraingo_angelua
332                 self.new_imu_data = False
333
334                 # Publicar feedback al cliente
335                 feedback_msg.oraingo_angelua = oraingo_angelua
336                 feedback_msg.bira_kopurua = emandako_birak
337                 self.get_logger().info(f'Oraingo angelua: {feedback_msg.oraingo_angelua
    }')
338                 goal_handle.publish_feedback(feedback_msg)
339
340                 # Calcula la diferencia entre actual y anterior considerando el ciclo
341                 if oraingo_angelua != 0.0:
342                     if noranzkoa:
343                         diferencia = (oraingo_angelua - anterior + 360) % 360
344                     else:
345                         diferencia = (anterior - oraingo_angelua + 360) % 360
346                     anterior = oraingo_angelua  # Actualiza la lectura anterior
347
348                     angulo_total += diferencia
349
350                 self.get_logger().info(f'TOTAL: {angulo_total}')
351
352         self.motorrak_gelditu()
353
354         success = True
355         goal_handle.succeed()
356
357         result = Bira360.Result()
358         result.arrakasta = success
359         result.amaierako_angelua = oraingo_angelua
360         result.hasierako_angelua = hasierako_angelua
361         result.bira_kopurua = emandako_birak
362
363         return result
364     ##########################################################################
365
366 ##########################################################################
367
368 def main(args=None):
369     rclpy.init(args=args)
370     node = PertzepzioProbaWrapper()
371     rclpy.spin(node)
372     node.destroy_node()
373     rclpy.shutdown()
374
375 if __name__ == '__main__':
376     main()
```

Código 10.31: Código del nodo *pertzepzio_proba_wrapper*.

Wrapper de la prueba de control

```python
#################################################################################
# ROS2 imports...
import rclpy
from rclpy.node import Node
from rclpy.lifecycle import LifecycleNode, LifecycleState, TransitionCallbackReturn
from rclpy.action import ActionServer
from rclpy.callback_groups import ReentrantCallbackGroup
from rclpy.executors import MultiThreadedExecutor
from lifecycle_msgs.msg import Transition
from lifecycle_msgs.srv import ChangeState
from sensor_msgs.msg import LaserScan

from bilbotiks_interfazeak.msg import MotorrakMugitu, ServoakMugitu

# Liburutegien imports...
import numpy as np

# SAVE DATA TFG
import csv
import time
from datetime import datetime
# SAVE DATA TFG
#################################################################################


#################################################################################
class KontrolProbaWrapper(Node):
    def __init__(self):
        super().__init__('kontrol_proba_wrapper')

        # Behar diren klase objektuak hasiarazi
        self.last_error = 0
        self.integral = 0
        self.lidar_datuak = None
        self.mugitzen = True

        # Parametroak ezarri lehenetsitako balioak
        self.declare_parameter('kp', 100.0)
        self.declare_parameter('ki', 0.0)
        self.declare_parameter('kd', 25.0)
        self.declare_parameter('vel', 20)
        self.declare_parameter('izena', 'izena')
        self.declare_parameter('eskuina', [-90.0, -45.0])
        self.declare_parameter('ezkerra', [135.0, 180.0])
        self.declare_parameter('aurrera', [-180.0, -90.0])

        # Parametroak irakurri YAML fitxerotik
        self.kp = self.get_parameter('kp').value
        self.ki = self.get_parameter('ki').value
        self.kd = self.get_parameter('kd').value
        self.vel = self.get_parameter('vel').value
        self.nombre = self.get_parameter('izena').value
        self.eskuina_angeluak_lidar = self.get_parameter('eskuina').value
        self.ezkerra_angeluak_lidar = self.get_parameter('ezkerra').value
        self.aurrera_angeluak_lidar = self.get_parameter('aurrera').value

        # Argitaratzaileak (PUBLISHERS)
        self.motor_publisher = self.create_publisher(MotorrakMugitu, 'motorrak_mugitu',
     10)
        self.servo_publisher = self.create_publisher(ServoakMugitu, 'servoak_mugitu',
    10)

        # Harpidetzak (SUBSCRIBERS)
        self.subscription = self.create_subscription(
            LaserScan,
            'lidar_datuak',
            self.lidar_callback,
            10
```

```
        )

        self.nodoa_kudeatu('motorrak_roboclaw', Transition.TRANSITION_CONFIGURE) # Motorrak konfiguratu
        self.nodoa_kudeatu('motorrak_roboclaw', Transition.TRANSITION_ACTIVATE) # Motorrak aktibatu
        self.nodoa_kudeatu('servoak', Transition.TRANSITION_CONFIGURE) # Servoak konfiguratu
        self.nodoa_kudeatu('servoak', Transition.TRANSITION_ACTIVATE) # Servoak aktibatu
        self.nodoa_kudeatu('lidar_argitaratzailea', Transition.TRANSITION_CONFIGURE) # Kamera aktibatu
        self.nodoa_kudeatu('lidar_argitaratzailea', Transition.TRANSITION_ACTIVATE) # Lidar aktibatu

        self.servoak_kokatu(self.servo_posizioak_kalkulatu(0))
        self.motorrak_mugitu()
        self.get_logger().info('/kontrola_proba_wrapper nodoa hasita')


    ##########################################################################
    # LIFECYCLE nodoen trantzisioak kudeatzeko
    def nodoa_kudeatu(self, nodoa, transition_id):
        # Lifecycle nodoen egoera kudeatzeko funtzioa
        self.get_logger().info(f'/{nodoa}/change_state')
        self.change_state = self.create_client(ChangeState, f'/{nodoa}/change_state')
        while not self.change_state.wait_for_service(timeout_sec=1.0):
            self.get_logger().info(f'/{nodoa}/change_state zerbitzua prest egoteko zain ...')

        request = ChangeState.Request()
        request.transition.id = transition_id
        future = self.change_state.call_async(request)
        rclpy.spin_until_future_complete(self, future)
        if future.result() is not None:
            self.get_logger().info(f'Trantsizioa ongi burutu da: {nodoa}, {transition_id}')
        else:
            self.get_logger().error('Errorea trantsizioa egitean.')
    ##########################################################################

    ##########################################################################
    # MUGIMENDUAREN LAGUNTZA FUNTZIOAK
    def servoak_kokatu(self, angeluak):
        # Servomotorrak kokatu
        msg = ServoakMugitu()
        msg.angeluak = angeluak  # 360º-ko biraketa egiteko behar diren servomotorren angeluak
        self.servo_publisher.publish(msg)
        self.get_logger().info(f'Servoak posizio egokira mugitzeko /servoak_mugitu topikoan argitaratuta: {msg.angeluak}')

    def motorrak_mugitu(self, abiadura=[-20, 20, -20, 20, -20, 20]):
        # Motorrak mugitu
        msg = MotorrakMugitu()
        abiadura =[self.vel*(-1), self.vel, self.vel*(-1), self.vel, self.vel*(-1), self.vel]
        msg.abiadurak = abiadura
        self.motor_publisher.publish(msg)
        self.get_logger().info(f'motorrak mugitzeko /motorrak_mugitu topikoan argitaratutako abiadurak: {msg.abiadurak}')

        return True

    def servo_posizioak_kalkulatu(self, desired_turn_angle):
        if desired_turn_angle > 90:
            desired_turn_angle = 90
        elif desired_turn_angle < -90:
            desired_turn_angle = -90

```

```
125         self.get_logger().info(f'DESIRED: {desired_turn_angle}')
126 
127         servo_angles = [0] * 4
128         servo_angles[0] = 15 #int(max(0, 15 + (desired_turn_angle / 90) * (60 - 15)))
129         servo_angles[1] = int(max(0, 140 - (desired_turn_angle / 90) * (140 - 120)))
130         servo_angles[2] = int(max(0, 25 - (desired_turn_angle / 90) * (25 - 0)))
131         servo_angles[3] = 145 #int(max(0, 145 + (desired_turn_angle / 90) * (180 - 145)
    ))
132 
133         self.get_logger().info(f'{servo_angles}')
134 
135         return servo_angles
136     ##############################################################################
137 
138 
139     ##############################################################################
140     # LIDAR harpidetza (LIDAR SUBSCRIBER)
141     def lidar_callback(self, msg):
142         if self.mugitzen:
143             self.lidar_datuak = msg
144             self.angeluak = np.linspace(msg.angle_min, msg.angle_max, len(msg.ranges))
145             self.ranges = np.nan_to_num(np.array(msg.ranges))
146             self.ranges[self.ranges < 0.0011] = 1e6
147             self.ranges[self.ranges > 8.0] = 8.0
148 
149             correcion = self.norabidea_kalkulatu()
150             servo_angles = self.servo_posizioak_kalkulatu(correcion)
151             self.servoak_kokatu(servo_angles)
152     ##############################################################################
153 
154 
155     ##############################################################################
156     # PID
157     def norabidea_kalkulatu(self):
158 
159         #Eskuina
160         angle_start = self.eskuina_angeluak_lidar[0] #-90
161         angle_end = self.eskuina_angeluak_lidar[1] #-67.5 #45
162         start_index = int((angle_start - self.lidar_datuak.angle_min) / self.
    lidar_datuak.angle_increment)
163         end_index = int((angle_end - self.lidar_datuak.angle_min) / self.lidar_datuak.
    angle_increment)
164         sublista = self.ranges[start_index:end_index+1]
165         min_right = np.argmin(self.ranges[start_index:end_index+1])
166         distanciaDerecha = sublista[min_right]
167 
168         #Ezkerra
169         angle_start = self.ezkerra_angeluak_lidar[0] #157.5 #135
170         angle_end = self.ezkerra_angeluak_lidar[1] #180
171         start_index = int((angle_start - self.lidar_datuak.angle_min) / self.
    lidar_datuak.angle_increment)
172         end_index = int((angle_end - self.lidar_datuak.angle_min) / self.lidar_datuak.
    angle_increment)
173         sublista = self.ranges[start_index:end_index+1]
174         min_left = np.argmin(self.ranges[start_index:end_index+1])
175         distanciaIzquierda = sublista[min_left]
176 
177         #Aurrera
178         angle_start = self.aurrera_angeluak_lidar[0] #-180
179         angle_end = self.aurrera_angeluak_lidar[1] #-90
180         start_index = int((angle_start - self.lidar_datuak.angle_min) / self.
    lidar_datuak.angle_increment)
181         end_index = int((angle_end - self.lidar_datuak.angle_min) / self.lidar_datuak.
    angle_increment)
182         sublista = self.ranges[start_index:end_index+1]
183         min_front = np.argmin(self.ranges[start_index:end_index+1])
184         distanciaFrente = sublista[min_front]
185 
186         self.get_logger().info(f'Distancia derecha: {distanciaDerecha}')
187         self.get_logger().info(f'Distancia frente: {distanciaFrente}')
```

```
188         self.get_logger().info(f'Distancia izquierda: {distanciaIzquierda}')
189 
190         # Calcular la corrección PID
191         error = (distanciaIzquierda - distanciaDerecha)
192         d = error - self.last_error
193         i = self.integral + error
194         correcion = (error * (self.kp) + i * (self.ki) +d * (self.kd)) * (-1)
195 
196         self.get_logger().info(f'ERROR: {error}')
197         self.get_logger().info(f'CORRECCIÓN: {correcion}\n')
198 
199         ##########################################################################
200         #SAVE FOR TFG
201         # Ruta del archivo donde se guardarán los datos
202         archivo = f"performance_{self.nombre}_{self.kp}_{self.ki}_{self.kd}_{self.vel}.
    csv"
203 
204         # Abrir el archivo en modo escritura (o crear si no existe)
205         with open(archivo, mode='a', newline='') as file:
206             escritor = csv.writer(file)
207 
208             # Escribir encabezados solo si el archivo está vacío
209             if file.tell() == 0:
210                 escritor.writerow(["Error", "Correccion", "CorrecionEscalado","
    DistanciaFrente", "DistanciaDerecha", "DistanciaIzquierda", "Timestamp"])
211 
212             timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")[:-3]
213             valores = [error, correcion, distanciaFrente, distanciaDerecha,
    distanciaIzquierda, timestamp]
214             escritor.writerow(valores)
215         #SAVE FOR TFG
216         ##########################################################################
217 
218         return correcion
219     ##########################################################################
220 
221 
222 def main(args=None):
223     rclpy.init(args=args)
224     node = KontrolProbaWrapper()
225     executor = MultiThreadedExecutor()
226     executor.add_node(node)
227     executor.spin()
228     node.destroy_node()
229     rclpy.shutdown()
230 
231 if __name__ == '__main__':
232     main()
```

Código 10.32: Código del nodo *kontrol_proba_wrapper*.

Prueba de percepción

```python
#################################################################################
# ROS2 imports...
import rclpy
from rclpy.node import Node
from rclpy.action import ActionClient
from lifecycle_msgs.srv import ChangeState

from bilbotiks_interfazeak.srv import KoloreaZenbakia
from bilbotiks_interfazeak.action import Bira360
#################################################################################


#################################################################################
class PertzepzioProba(Node):
    def __init__(self):
        super().__init__('pertzepzio_proba')

        # Cliente de servicio
        self.cli_zerbitzua = self.create_client(KoloreaZenbakia, '
    koloreaZenbakia_iragarri')

        # Esperar hasta que el servicio esté disponible
        while not self.cli_zerbitzua.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('KoloreaZenbakia zerbitzua itxaroten...')

        # Cliente de acción
        self.cli_ekintza = ActionClient(self, Bira360, 'pertzepzio_proba_360bira')

    def zerbitzua_deitu(self):
        # Solicitar servicio
        request = KoloreaZenbakia.Request()

        future = self.cli_zerbitzua.call_async(request)
        rclpy.spin_until_future_complete(self, future)

        if future.result() is not None:
            self.get_logger().info(f'Zerbitzuaren erantzuna: {future.result().zenbakia
    }, {future.result().kolorea}')
        else:
            self.get_logger().error('Errorea zerbitzua deitutakoan')

        return future.result().kolorea, future.result().zenbakia

    def ekintza_deitu(self, norabidea, bira_kopurua):
        # Llamar a la acción
        self.get_logger().info(f'{bira_kopurua} bira {"eskuinara" if norabidea == 1
    else "ezkerretara"} egiteko ekintza deitu')

        goal_msg = Bira360.Goal()
        goal_msg.noranzkoa = norabidea  # 1 para derecha, 0 para izquierda
        goal_msg.zenbakia = bira_kopurua  # Número de bira_kopurua

        # Llamar a la acción y esperar la respuesta
        self.cli_ekintza.wait_for_server()

        # Enviar la solicitud de acción
        future = self.cli_ekintza.send_goal_async(goal_msg)
        rclpy.spin_until_future_complete(self, future)

        if future.result():
            self.get_logger().info(f'Ekintza arrakastaz bidalita: {future.result()}')
        else:
            self.get_logger().error('Errorea ekintza bidalitakoan')

        # Asignar el resultado del future a goal_handle
        goal_handle = future.result()
        if not goal_handle.accepted:
```

```
            self.get_logger().error('El objetivo no fue aceptado por el servidor de acciones.')
            return

        self.get_logger().info('Objetivo aceptado. Esperando resultado...')

        # Esperar al resultado de la acción
        result_future = goal_handle.get_result_async()
        rclpy.spin_until_future_complete(self, result_future)

        result = result_future.result()
        if result is not None:
            self.get_logger().info(f'Resultado de la acción: {result.result}')
        else:
            self.get_logger().error('No se recibió el resultado de la acción.')
##############################################################################


def main(args=None):
    rclpy.init(args=args)
    bezeroa = PertzepzioProba()

    # Kolorea eta zenbakia iragarri
    kolorea, zenbakia = bezeroa.zerbitzua_deitu()

    # Bira egin (kolorea 1: eskuina; kolorea 2: ezkerra)
    if kolorea == 2:
        norabidea = True
    elif kolorea == 1:
        norabidea = False
    else:
        bezeroa.destroy_node()
        rclpy.shutdown()
    bezeroa.ekintza_deitu(norabidea=norabidea, bira_kopurua=zenbakia)

    # Nodoa amaitu
    bezeroa.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Código 10.33: Código del nodo *pertzepzio_proba*.

II.8. Interfaces gráficas

Interfaz gráfica de la prueba de percepción

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
import cv2
import pygame
import numpy as np

# Asegúrate de que el paquete bilbotiks_interfazeak esté en tu path de ROS2
from bilbotiks_interfazeak.msg import Pertzepzioa

class CombinedDisplayNode(Node):
    def __init__(self):
        super().__init__('combined_display_node')
        self.get_logger().info('Nodo combined_display_node iniciado.')

        # Variables para recoger los 4 datos (aunque solo se muestran 2)
        self.imu = 0
        self.emandako_birak = 0  # Se recoge pero no se visualiza
        self.bira_totalak = 0
        self.norabidea = 0  # Se usará para definir el color de "Vueltas totales"

        # Suscripción para recibir mensajes del tópico de sensores
        self.create_subscription(
            Pertzepzioa,
            '/pertzepzioa_pantaila',
            self.sensor_callback,
            10
        )

        # Inicialización de la cámara
        self.cap = cv2.VideoCapture(0)
        if not self.cap.isOpened():
            self.get_logger().error("Error: no se pudo acceder a la cámara")
            rclpy.shutdown()
            return

        # Configuración de Pygame:
        pygame.init()
        # Ventana de 1280x400 dividida en 3 columnas: izquierda (datos), central (cámara) y derecha (datos)
        self.width, self.height = 1280, 400
        self.col_width = self.width // 3
        self.screen = pygame.display.set_mode((self.width, self.height), pygame.FULLSCREEN)
        pygame.display.set_caption("Combined Display: Webcam y Datos Sensor")
        self.clock = pygame.time.Clock()
        self.fullscreen = True  # Bandera para alternar pantalla completa

        # Nuevos tamaños de fuentes:
        # - Fuente para los labels descriptivos: 54 puntos (50% mayor que 36)
        # - Fuente para los datos: 120 puntos (100% mayor que 60)
        self.font_small = pygame.font.SysFont("Arial", 54)
        self.font_big = pygame.font.SysFont("Arial", 120)

    def sensor_callback(self, msg: Pertzepzioa):
        # Actualizamos todas las variables con los datos recibidos
        self.imu = msg.imu
        self.emandako_birak = msg.emandako_birak  # Recogido pero no se muestra
        self.bira_totalak = msg.bira_totalak
        self.norabidea = msg.norabidea

    def run(self):
        running = True
        while rclpy.ok() and running:
```

```
            # Procesamos eventos de Pygame para permitir cierre y alternar fullscreen
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                elif event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_ESCAPE:
                        if self.fullscreen:
                            self.screen = pygame.display.set_mode((self.width, self.height))
                            self.fullscreen = False
                        else:
                            self.screen = pygame.display.set_mode((self.width, self.height), pygame.FULLSCREEN)
                            self.fullscreen = True

            # Leer y procesar el frame de la cámara para la columna central
            ret, frame = self.cap.read()
            if not ret:
                self.get_logger().error("Error: no se pudo leer la imagen de la cámara")
                running = False
                break

            webcam_frame = cv2.resize(frame, (self.col_width, self.height))
            webcam_frame = cv2.cvtColor(webcam_frame, cv2.COLOR_BGR2RGB)
            frame_surface = pygame.surfarray.make_surface(np.rot90(webcam_frame))

            # Componer la ventana:
            self.screen.fill((0, 0, 0))
            # Columna izquierda: fondo blanco
            left_rect = pygame.Rect(0, 0, self.col_width, self.height)
            pygame.draw.rect(self.screen, (255, 255, 255), left_rect)
            # Columna central: fondo negro (para la cámara)
            center_rect = pygame.Rect(self.col_width, 0, self.col_width, self.height)
            pygame.draw.rect(self.screen, (0, 0, 0), center_rect)
            # Columna derecha: fondo blanco
            right_rect = pygame.Rect(2 * self.col_width, 0, self.col_width, self.height)
            pygame.draw.rect(self.screen, (255, 255, 255), right_rect)

            # Mostrar la imagen de la cámara en la columna central
            self.screen.blit(frame_surface, (self.col_width, 0))

            # Reposicionar los textos en las columnas laterales:
            # En cada columna el grupo (descriptivo + dato) se centra verticalmente en 200.
            # Se posiciona el label descriptivo más arriba para evitar solapamiento.
            # Por ejemplo, en la columna izquierda:
            left_center_x = self.col_width // 2
            imu_label_text = self.font_small.render("IMU", True, (0, 0, 0))
            # Ubicamos el label descriptivo en y = 100 (más arriba)
            imu_label_rect = imu_label_text.get_rect(center=(left_center_x, 100))
            imu_value_text = self.font_big.render(str(self.imu), True, (0, 0, 0))
            # Ubicamos el dato en y = 300 (más abajo)
            imu_value_rect = imu_value_text.get_rect(center=(left_center_x, 300))
            self.screen.blit(imu_label_text, imu_label_rect)
            self.screen.blit(imu_value_text, imu_value_rect)

            # En la columna derecha (para "Vueltas totales"):
            right_center_x = 2 * self.col_width + (self.col_width // 2)
            vueltas_label_text = self.font_small.render("Vueltas totales", True, (0, 0, 0))
            vueltas_label_rect = vueltas_label_text.get_rect(center=(right_center_x, 100))
            vueltas_color = (255, 0, 0) if self.norabidea == 0 else (0, 0, 255)
            vueltas_value_text = self.font_big.render(str(self.bira_totalak), True, vueltas_color)
            vueltas_value_rect = vueltas_value_text.get_rect(center=(right_center_x, 300))
            self.screen.blit(vueltas_label_text, vueltas_label_rect)
```

```
124             self.screen.blit(vueltas_value_text, vueltas_value_rect)
125
126             pygame.display.flip()
127             self.clock.tick(30)  # Limitar a 30 FPS
128
129             # Permitir procesar callbacks ROS sin bloquear
130             rclpy.spin_once(self, timeout_sec=0.001)
131
132         self.cap.release()
133         pygame.quit()
134
135 def main(args=None):
136     rclpy.init(args=args)
137     node = CombinedDisplayNode()
138     try:
139         node.run()
140     except KeyboardInterrupt:
141         node.get_logger().info("Interrupción por el usuario")
142     finally:
143         node.destroy_node()
144         rclpy.shutdown()
145
146 if __name__ == '__main__':
147     main()
```

Código 10.34: Código del nodo de la interfaz gráfica de la prueba de percepción.

Interfaz gráfica de la prueba de control

```python
#!/usr/bin/env python3
import rclpy
from rclpy.lifecycle import LifecycleNode, TransitionCallbackReturn, LifecycleState
import tkinter as tk
import sys
from bilbotiks_interfazeak.msg import Kontrola

class KontrolPantaila(LifecycleNode):
    def __init__(self):
        super().__init__('kontrol_pantaila')
        self.get_logger().info("/kontrol_pantaila nodoa IN constructor")

        # Jasoko diren datuak gordetzeko aldagaiak
        self.ezker_distantzia = 0
        self.aurreko_distantzia = 0
        self.eskuin_distantzia = 0

        # tkinter leihoak eta etiketak
        self.window = None
        self.label_dato1 = None
        self.label_dato2 = None
        self.label_dato3 = None

        # Etiquetas descriptivas
        self.label_ezkerra = None
        self.label_aurre = None
        self.label_eskuina = None

        # Timer para actualizar la ventana Tkinter
        self.timer = None

        # Suscripción al tópico (se crea en on_configure)
        self.subscription = None

        self.is_activate = False

    def on_configure(self, state: LifecycleState):
        self.get_logger().info("/kontrol_pantaila nodoa IN on_configure")
        try:
            # Crear la ventana principal y ajustar la geometría para disponer de dos filas
            self.window = tk.Tk()
            self.window.geometry("600x150")
            self.window.title("Kontrol probaren pantaila")
            self.window.configure(bg="white")

            # Configurar las columnas para que se distribuyan de forma homogénea
            self.window.columnconfigure(0, weight=1)
            self.window.columnconfigure(1, weight=1)
            self.window.columnconfigure(2, weight=1)

            # Fuente para el texto en la interfaz
            label_font = ("Arial", 20)

            # Fila 0: mostrar los datos
            self.label_dato1 = tk.Label(self.window, text="0", font=label_font, bg="white", fg="black")
            self.label_dato1.grid(row=0, column=0, padx=10, pady=5)

            self.label_dato2 = tk.Label(self.window, text="0", font=label_font, bg="white", fg="black")
            self.label_dato2.grid(row=0, column=1, padx=10, pady=5)

            # En la columna derecha se mostrará el dato con color variable
            self.label_dato3 = tk.Label(self.window, text="0", font=label_font, bg="white", fg="black")
            self.label_dato3.grid(row=0, column=2, padx=10, pady=5)

```

```
            # Fila 1: etiquetas descriptivas para identificar cada dato
            descr_font = ("Arial", 14)
            self.label_ezkerra = tk.Label(self.window, text="Izquierda", font=
    descr_font, bg="white", fg="black")
            self.label_ezkerra.grid(row=1, column=0, padx=10, pady=2)

            self.label_aurre = tk.Label(self.window, text="Frente", font=descr_font, bg
    ="white", fg="black")
            self.label_aurre.grid(row=1, column=1, padx=10, pady=2)

            self.label_eskuina = tk.Label(self.window, text="Derecha", font=descr_font,
     bg="white", fg="black")
            self.label_eskuina.grid(row=1, column=2, padx=10, pady=2)

            # Configurar el cierre de la ventana para liberar recursos
            self.window.protocol("WM_DELETE_WINDOW", self._on_window_close)

            # Crear la suscripción al tópico /pertzepzioa_pantaila
            self.subscription = self.create_subscription(
                Kontrola,
                '/kontrol_pantaila',
                self.subscription_callback,
                10
            )
        except Exception as e:
            self.get_logger().error("Error al configurar tkinter o la suscripción: " +
    str(e))
            return TransitionCallbackReturn.FAILURE

        self.get_logger().info("Interfaz y suscripción configuradas correctamente")
        return TransitionCallbackReturn.SUCCESS

    def subscription_callback(self, msg):
        if self.is_activate:
            # Actualizar los datos a partir del mensaje recibido.
            # Se asume que el mensaje Pertzepzioa tiene los atributos: imu,
    emandako_birak, bira_totalak y norabidea.
            self.ezker_distantzia = round(msg.ezker_distantzia, 2)
            self.aurreko_distantzia = round(msg.aurreko_distantzia, 2)
            self.eskuin_distantzia = round(msg.eskuin_distantzia, 2)


            # Actualizar las etiquetas con los datos
            self.label_dato1.config(text=f"{self.ezker_distantzia}")
            self.label_dato2.config(text=f"{self.aurreko_distantzia}")
            self.label_dato3.config(text=f"{self.eskuin_distantzia}")

            # La línea de log se puede descomentar para visualizar la info recibida
            # self.get_logger().info(f"Mensaje recibido: {self.ezker_distantzia}, {self
    .aurreko_distantzia}, {self.eskuin_distantzia} (norabidea: {self.norabidea})")

    def _on_window_close(self):
        self.get_logger().info("Leihoa ixteko eskatu da.")
        if self.window:
            self.window.destroy()
        rclpy.shutdown()
        sys.exit(0)

    def on_activate(self, state: LifecycleState):
        self.get_logger().info("/kontrol_pantaila nodoa IN on_activate")
        self.is_activate = True
        # Crear un timer para refrescar la ventana (por ejemplo, 30 fps)
        if self.timer is None:
            self.timer = self.create_timer(1 / 30.0, self.timer_callback)
        return TransitionCallbackReturn.SUCCESS

    def timer_callback(self):
        # Actualizar la ventana de tkinter para reflejar los cambios realizados en la
    suscripción
        if self.window:
```

```
128             try:
129                 self.window.update_idletasks()
130                 self.window.update()
131             except tk.TclError:
132                 pass
133 
134     def on_deactivate(self, state: LifecycleState):
135         self.get_logger().info("/kontrol_pantaila nodoa IN on_deactivate")
136         self.is_activate = False
137         if self.timer is not None:
138             self.timer.cancel()
139             self.timer = None
140         return TransitionCallbackReturn.SUCCESS
141 
142     def on_cleanup(self, state: LifecycleState):
143         self.get_logger().info("/kontrol_pantaila nodoa IN on_cleanup")
144         if self.window:
145             self.window.destroy()
146             self.window = None
147         return TransitionCallbackReturn.SUCCESS
148 
149     def on_shutdown(self, state: LifecycleState):
150         self.get_logger().info("/kontrol_pantaila nodoa IN on_shutdown")
151         return TransitionCallbackReturn.SUCCESS
152 
153 def main(args=None):
154     rclpy.init(args=args)
155     interface_node = KontrolPantaila()
156     try:
157         rclpy.spin(interface_node)
158     except KeyboardInterrupt:
159         interface_node.get_logger().info("Erabiltzaileak nodoa gelditu du.")
160     finally:
161         interface_node.destroy_node()
162         rclpy.shutdown()
163 
164 if __name__ == '__main__':
165     main()
```

Código 10.35: Código del nodo de la interfaz gráfica de la prueba de control.

Interfaz gráfica de la cara

```python
import rclpy
from rclpy.lifecycle import LifecycleNode, LifecycleState, TransitionCallbackReturn
from bilbotiks_interfazeak.msg import Aurpegia  # Usamos el mensaje tal como está
import pygame
import sys
import random

class AurpegiaPantaila(LifecycleNode):
    def __init__(self):
        super().__init__('aurpegia')
        self.get_logger().info("Constructor de AurpegiaPantaila")

        # Variables que se inicializarán en on_configure
        self.screen = None
        self.WIDTH = None
        self.HEIGHT = None
        self.clock = None
        self.blink_timer = 0
        self.eyes_closed = False
        self.min_open_time = 500   # ms
        self.max_open_time = 2000  # ms
        self.closed_time = 150     # ms
        self.next_blink_interval = None

        # Dirección que se selecciona a partir de los mensajes, por defecto "center"
        self.desired_direction = "center"  # Puede ser "center", "left" o "right"

        # Referencias a los sprites (se cargarán en on_configure)
        self.left_eye_open = None
        self.left_eye_closed = None
        self.right_eye_open = None
        self.right_eye_closed = None

        self.left_eye_left = None
        self.right_eye_left = None
        self.left_eye_right = None
        self.right_eye_right = None

        self.smile = None
        self.smile_left = None
        self.smile_right = None

        # Posiciones en pantalla de los sprites
        self.left_eye_pos = (200, 150)
        self.right_eye_pos = (500, 150)
        self.smile_pos = (300, 300)

        # Banderas y apuntadores para la suscripción y el timer
        self.deactivated = False             # Controla si se pausa la animación/
    blinking
        self.subscription_active = False     # Controla si se deben procesar mensajes (
    on_activate)
        self.subscription = None
        self.timer = None

    def on_configure(self, state: LifecycleState):
        self.get_logger().info("AurpegiaPantaila en on_configure - Configurando ventana
     y sprites")
        # Inicializar Pygame y configurar la ventana
        pygame.init()
        self.WIDTH, self.HEIGHT = 800, 600
        self.screen = pygame.display.set_mode((self.WIDTH, self.HEIGHT))
        pygame.display.set_caption("AurpegiaPantaila: Cara del Robot")

        # En este ejemplo se usa una ruta absoluta para cargar los recursos instalados.
        # Puedes ajustar la forma en que se obtiene la ruta, por ejemplo usando
    pkg_resources o importlib.resources.
```

```
        path = '/home/javierac/mounted/moverRover/bilbotiks_lifecyle_ws/install/bilbotiks_pantaila/lib/python3.10/site-packages/bilbotiks_pantaila/aurpegia_irudiak'

        # Cargar los sprites de los ojos y la boca
        try:
            self.left_eye_open = pygame.image.load(f"{path}/izquierda_abierto.png").convert_alpha()
            self.left_eye_closed = pygame.image.load(f"{path}/izquierda_cerrado.png").convert_alpha()
            self.right_eye_open = pygame.image.load(f"{path}/derecha_abierto.png").convert_alpha()
            self.right_eye_closed = pygame.image.load(f"{path}/derecha_cerrado.png").convert_alpha()

            # Sprites para mirar a los lados
            self.left_eye_left = pygame.image.load(f"{path}/mirar_izquierda_trans.png").convert_alpha()
            self.right_eye_left = pygame.image.load(f"{path}/mirar_izquierda_trans.png").convert_alpha()
            self.left_eye_right = pygame.image.load(f"{path}/mirar_derecha_trans.png").convert_alpha()
            self.right_eye_right = pygame.image.load(f"{path}/mirar_derecha_trans.png").convert_alpha()

            # Cargar los sprites de la boca
            self.smile = pygame.image.load(f"{path}/smile.png").convert_alpha()
            self.smile_left = pygame.image.load(f"{path}/smile_izquierda.png").convert_alpha()
            self.smile_right = pygame.image.load(f"{path}/smile_derecha.png").convert_alpha()
        except Exception as e:
            self.get_logger().error(f"Error al cargar sprites: {e}")
            return TransitionCallbackReturn.FAILURE

        # Escalar los sprites
        eye_size = (100, 100)
        self.left_eye_open = pygame.transform.scale(self.left_eye_open, eye_size)
        self.left_eye_closed = pygame.transform.scale(self.left_eye_closed, eye_size)
        self.right_eye_open = pygame.transform.scale(self.right_eye_open, eye_size)
        self.right_eye_closed = pygame.transform.scale(self.right_eye_closed, eye_size)
        self.left_eye_left = pygame.transform.scale(self.left_eye_left, eye_size)
        self.right_eye_left = pygame.transform.scale(self.right_eye_left, eye_size)
        self.left_eye_right = pygame.transform.scale(self.left_eye_right, eye_size)
        self.right_eye_right = pygame.transform.scale(self.right_eye_right, eye_size)

        smile_size = (200, 100)
        self.smile = pygame.transform.scale(self.smile, smile_size)
        self.smile_left = pygame.transform.scale(self.smile_left, smile_size)
        self.smile_right = pygame.transform.scale(self.smile_right, smile_size)

        # Inicializar el reloj y el temporizador del parpadeo
        self.clock = pygame.time.Clock()
        self.blink_timer = 0
        self.next_blink_interval = random.randint(self.min_open_time, self.max_open_time)

        self.get_logger().info("AurpegiaPantaila configurado correctamente")
        return TransitionCallbackReturn.SUCCESS

    def on_activate(self, state: LifecycleState):
        self.get_logger().info("AurpegiaPantaila en on_activate")
        # Activar tanto la animación como la suscripción
        self.deactivated = False
        self.subscription_active = True

        # Crear la suscripción al tópico /aurpegia (con mensaje de tipo Aurpegia)
        self.subscription = self.create_subscription(
            Aurpegia,
            '/aurpegia',
```

```
            self.aurpegia_callback,
            10
        )
        # Iniciar un timer para actualizar la animación (alrededor de 60 fps)
        if self.timer is None:
            self.timer = self.create_timer(1/60.0, self.timer_callback)
        return TransitionCallbackReturn.SUCCESS

    def aurpegia_callback(self, msg):
        # Solo procesamos si la suscripción está activa
        if not self.subscription_active:
            return

        """
        Callback para actualizar la dirección según el valor recibido en msg.begi_norabidea.
        0 → "center", 1 → "left", 2 → "right".
        """
        valor = msg.begi_norabidea
        if valor == 0:
            self.desired_direction = "center"
        elif valor == 1:
            self.desired_direction = "left"
        elif valor == 2:
            self.desired_direction = "right"
        else:
            self.get_logger().warn("Valor desconocido recibido, fijando a 'center'")
            self.desired_direction = "center"
        self.get_logger().info(f"Dirección actualizada a: {self.desired_direction}")

    def timer_callback(self):
        dt = self.clock.tick(60)
        # Procesar eventos de Pygame para mantener la ventana responsive
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

        # Si el nodo está activado (no desactivado), actualizamos el parpadeo;
        # si está desactivado, mantenemos la imagen fija.
        if not self.deactivated:
            self.blink_timer += dt
            if not self.eyes_closed:
                if self.blink_timer > self.next_blink_interval:
                    self.eyes_closed = True
                    self.blink_timer = 0
                    self.next_blink_interval = self.closed_time
            else:
                if self.blink_timer > self.next_blink_interval:
                    self.eyes_closed = False
                    self.blink_timer = 0
                    self.next_blink_interval = random.randint(self.min_open_time, self.max_open_time)
        else:
            # En modo desactivado, forzamos un estado estático (por ejemplo, ojos abiertos)
            self.eyes_closed = False

        # Dibujar fondo y sprites según la dirección y el estado (blinking o estático)
        self.screen.fill((135, 206, 235))  # Color azul cielo
        if not self.eyes_closed:
            if self.desired_direction == "center":
                self.screen.blit(self.left_eye_open, self.left_eye_pos)
                self.screen.blit(self.right_eye_open, self.right_eye_pos)
            elif self.desired_direction == "left":
                self.screen.blit(self.left_eye_left, self.left_eye_pos)
                self.screen.blit(self.right_eye_left, self.right_eye_pos)
            elif self.desired_direction == "right":
                self.screen.blit(self.left_eye_right, self.left_eye_pos)
                self.screen.blit(self.right_eye_right, self.right_eye_pos)
```

```
188         else:
189             self.screen.blit(self.left_eye_closed, self.left_eye_pos)
190             self.screen.blit(self.right_eye_closed, self.right_eye_pos)
191 
192         # Seleccionar y dibujar la expresión de la boca según la dirección
193         if self.desired_direction == "center":
194             current_smile = self.smile
195         elif self.desired_direction == "left":
196             current_smile = self.smile_left
197         elif self.desired_direction == "right":
198             current_smile = self.smile_right
199 
200         self.screen.blit(current_smile, self.smile_pos)
201         pygame.display.flip()
202 
203     def on_deactivate(self, state: LifecycleState):
204         self.get_logger().info("AurpegiaPantaila en on_deactivate")
205         # En on_deactivate, la animación se pausa y la suscripción se inactiva
206         self.deactivated = True
207         self.subscription_active = False
208         self.subscription = None
209         return TransitionCallbackReturn.SUCCESS
210 
211     def on_cleanup(self, state: LifecycleState):
212         self.get_logger().info("AurpegiaPantaila en on_cleanup")
213         self.subscription = None
214         # Si se destruye la ventana, se llama a pygame.quit()
215         self.screen = None
216         pygame.quit()
217         return TransitionCallbackReturn.SUCCESS
218 
219     def on_shutdown(self, state: LifecycleState):
220         self.get_logger().info("AurpegiaPantaila en on_shutdown. Cerrando Pygame")
221         return TransitionCallbackReturn.SUCCESS
222 
223 def main(args=None):
224     rclpy.init(args=args)
225     aurpegia_pantaila = AurpegiaPantaila()
226     try:
227         rclpy.spin(aurpegia_pantaila)
228     except KeyboardInterrupt:
229         aurpegia_pantaila.get_logger().info("Nodo interrumpido por el usuario")
230     finally:
231         aurpegia_pantaila.destroy_node()
232         rclpy.shutdown()
233 
234 if __name__ == '__main__':
235     main()
```

Código 10.36: Código del nodo de la interfaz gráfica de la cara.

II.9. Gemelo digital y simulación

Gemelo digital de la *IMU*

```python
import rclpy
from rclpy.lifecycle import LifecycleNode, LifecycleState, TransitionCallbackReturn
from bilbotiks_interfazeak.msg import Imu
import random
import math

class IMUArgitaratzailea(LifecycleNode):
    def __init__(self):
        super().__init__('imu_argitaratzailea')
        self.publisher_ = None
        self.timer_ = None

        self.get_logger().info("imu_argitaratzailea nodoa IN constructor")

    def on_configure(self, state: LifecycleState):
        self.get_logger().info("imu_argitaratzailea nodoa IN on_configure")

        # Modo simulación: No se inicializa hardware (I2C ni sensor real)
        self.publisher_ = self.create_lifecycle_publisher(Imu, 'imu_datuak', 20)
        self.timer_ = self.create_timer(0.1, self.publish_imu_data)  # Cada 100ms

        return TransitionCallbackReturn.SUCCESS

    def on_cleanup(self, state: LifecycleState):
        self.get_logger().info("imu_argitaratzailea nodoa IN on_cleanup")
        self.destroy_lifecycle_publisher(self.publisher_)
        self.destroy_timer(self.timer_)
        return TransitionCallbackReturn.SUCCESS

    def on_activate(self, state: LifecycleState):
        self.get_logger().info("imu_argitaratzailea nodoa IN on_activate")
        self.timer_.reset()
        self.get_logger().info('Publicando datos de la IMU en /imu_datuak cada 100ms',
    once=True)
        return super().on_activate(state)

    def on_deactivate(self, state: LifecycleState):
        self.get_logger().info("imu_argitaratzailea nodoa IN on_deactivate")
        self.timer_.cancel()
        return super().on_deactivate(state)

    def on_shutdown(self, state: LifecycleState):
        self.get_logger().info("imu_argitaratzailea nodoa IN on_shutdown")
        return TransitionCallbackReturn.SUCCESS

    def publish_imu_data(self):
        msg = Imu()

        # Simulación de la temperatura (valor entero)
        msg.tenperatura = random.randint(-10, 40)  # Ejemplo: rango entre -10 y +40 °C

        # Simulación del magnetómetro (valores en float)
        msg.magnetometroa.x = random.uniform(-100.0, 100.0)
        msg.magnetometroa.y = random.uniform(-100.0, 100.0)
        msg.magnetometroa.z = random.uniform(-100.0, 100.0)

        # Simulación del giroscopio (valores en float)
        msg.giroskopioa.x = random.uniform(-500.0, 500.0)
        msg.giroskopioa.y = random.uniform(-500.0, 500.0)
        msg.giroskopioa.z = random.uniform(-500.0, 500.0)

        # Simulación de los ángulos Euler (valores float entre 0 y 360)
        msg.euler_angeluak.x = random.uniform(0.0, 360.0)
        msg.euler_angeluak.y = random.uniform(0.0, 360.0)
```

```
64          msg.euler_angeluak.z = random.uniform(0.0, 360.0)
65
66          # Simulación del cuaternión: genero 4 números aleatorios y los normalizo para
        obtener un cuaternión unitario
67          q = [random.uniform(-1.0, 1.0) for _ in range(4)]
68          norm = math.sqrt(sum(x * x for x in q))
69          q = [x / norm for x in q]
70          msg.kuaternioak.x = q[0]
71          msg.kuaternioak.y = q[1]
72          msg.kuaternioak.z = q[2]
73          msg.kuaternioak.w = q[3]
74
75          # Simulación de la aceleración lineal (valores en float)
76          msg.azelerazio_lineala.x = random.uniform(-10.0, 10.0)
77          msg.azelerazio_lineala.y = random.uniform(-10.0, 10.0)
78          msg.azelerazio_lineala.z = random.uniform(-10.0, 10.0)
79
80          # Simulación de la gravedad (valores en float)
81          msg.grabitatea.x = random.uniform(-9.81, 9.81)
82          msg.grabitatea.y = random.uniform(-9.81, 9.81)
83          msg.grabitatea.z = random.uniform(-9.81, 9.81)
84
85          # Publicar el mensaje
86          self.publisher_.publish(msg)
87          self.get_logger().debug("Datos simulados publicados.")
88
89  def main(args=None):
90      rclpy.init(args=args)
91      node = IMUArgitaratzailea()
92
93      try:
94          rclpy.spin(node)
95      except KeyboardInterrupt:
96          node.get_logger().info("El usuario ha interrumpido el nodo.")
97      finally:
98          node.destroy_node()
99          rclpy.shutdown()
100
101 if __name__ == '__main__':
102     main()
```

Código 10.37: Código del nodo *imu_sim*.

Descripción del mundo y robot virtual en *Gazebo Harmonic*

```
<?xml version="1.0" ?>
<sdf version="1.8">
    <world name="laberinto">
        <physics name="1ms" type="ignored">
            <max_step_size>0.001</max_step_size>
            <real_time_factor>1.0</real_time_factor>
        </physics>

        <!-- Plugins para cargar la escena -->
        <plugin
            filename="gz-sim-physics-system"
            name="gz::sim::systems::Physics">
        </plugin>
        <plugin
            filename="gz-sim-user-commands-system"
            name="gz::sim::systems::UserCommands">
        </plugin>
        <plugin
            filename="gz-sim-scene-broadcaster-system"
            name="gz::sim::systems::SceneBroadcaster">
        </plugin>
        <!-- -->

        <!-- Plugins de sensores -->
        <!-- IMU -->
        <plugin filename="gz-sim-imu-system"
            name="gz::sim::systems::Imu">
        </plugin>

        <!-- LiDAR -->
        <plugin
            filename="gz-sim-sensors-system"
            name="gz::sim::systems::Sensors">
            <render_engine>ogre2</render_engine>
        </plugin>
        <!-- -->

        <!-- Iluminación de la escena y del mundo -->
        <light type="directional" name="sun">
            <cast_shadows>true</cast_shadows>
            <pose>0 0 10 0 0 0</pose>
            <diffuse>0.8 0.8 0.8 1</diffuse>
            <specular>0.2 0.2 0.2 1</specular>
            <attenuation>
                <range>1000</range>
                <constant>0.9</constant>
                <linear>0.01</linear>
                <quadratic>0.001</quadratic>
            </attenuation>
            <direction>-0.5 0.1 -0.9</direction>
        </light>
        <!-- -->

        <!-- Suelo, necesario para no caer al vacío -->
        <model name="suelo">
            <static>true</static>
            <link name="link">
                <collision name="collision">
                <geometry>
                    <plane>
                    <normal>0 0 1</normal>
                    </plane>
                </geometry>
                </collision>
                <visual name="visual">
                <geometry>
                    <plane>
                    <normal>0 0 1</normal>
```

```
                    <size>100 100</size>
                    </plane>
                </geometry>
                <material>
                    <ambient>0.8 0.8 0.8 1</ambient>
                    <diffuse>0.8 0.8 0.8 1</diffuse>
                    <specular>0.8 0.8 0.8 1</specular>
                </material>
                </visual>
            </link>
        </model>
        <!-- -->

        <!-- Modelo del robot -->
        <model name='robot' canonical_link='chassis'>
            <pose relative_to='world'>-3.55 0 0 0 0 0</pose>

            <!-- Cuadro para colocar el LiDAR -->
            <frame name="lidar_frame" attached_to='chassis'>
                <!-- respecto del chassis -->
                <pose>0.8 0 0.5 0 0 0</pose>
            </frame>

            <link name='chassis'>
                <pose relative_to='__model__'>0.5 0 0.4 0 0 0</pose>

                <!--Propiedades de inercia de la masa del link, matiz de inercia -->
                <!-- valores por defecto del tutorial -->
                <inertial>
                    <mass>1.14395</mass>
                    <inertia>
                        <ixx>0.126164</ixx>
                        <ixy>0</ixy>
                        <ixz>0</ixz>
                        <iyy>0.416519</iyy>
                        <iyz>0</iyz>
                        <izz>0.481014</izz>
                    </inertia>
                </inertial>

                <!--Características visuales del robot -->
                <visual name='visual'>
                    <geometry>
                        <box>
                            <size>2.0 1.0 0.5</size> <!--en metros -->
                        </box>
                    </geometry>
                    <!-- Añadir color o material -->
                    <material>
                        <ambient>0.0 0.0 1.0 1</ambient>
                        <diffuse>0.0 0.0 1.0 1</diffuse>
                        <specular>0.0 0.0 1.0 1</specular>
                    </material>
                </visual>

                <!--Características de colisión del robot -->
                <collision name='collision'>
                    <geometry>
                        <box>
                            <size>2.0 1.0 0.5</size>
                        </box>
                    </geometry>
                </collision>

                <!--Sensor IMU -->
                <sensor name="imu_sensor" type="imu">
                    <always_on>1</always_on>
                    <update_rate>1</update_rate>
                    <visualize>true</visualize>
                    <topic>imu</topic>
```

```
                </sensor>

                <!--Sensor LiDAR -->
                <sensor name='gpu_lidar' type='gpu_lidar'>
                    <pose relative_to='lidar_frame'>0 0 0 0 0 0</pose>
                    <topic>lidar</topic>
                    <update_rate>10</update_rate>
                    <ray>
                        <scan>
                            <horizontal>
                                <samples>640</samples>
                                <resolution>1</resolution>
                                <min_angle>-1.5708</min_angle>
                                <max_angle>1.5708</max_angle>
                            </horizontal>
                            <vertical>
                                <samples>1</samples>
                                <resolution>0.01</resolution>
                                <min_angle>0</min_angle>
                                <max_angle>0</max_angle>
                            </vertical>
                        </scan>
                        <range>
                            <min>0.08</min>
                            <max>10.0</max>
                            <resolution>0.01</resolution>
                        </range>
                    </ray>
                    <always_on>1</always_on>
                    <visualize>true</visualize>
                </sensor>
            </link>

            <!-- Rueda izquierda -->
            <link name='rueda_izq'>
                <!--ángulos en radianes-->
                <pose relative_to="chassis">-0.5 0.6 0 -1.5707 0 0</pose>

                <inertial>
                    <mass>2</mass>
                    <inertia>
                        <ixx>0.145833</ixx>
                        <ixy>0</ixy>
                        <ixz>0</ixz>
                        <iyy>0.145833</iyy>
                        <iyz>0</iyz>
                        <izz>0.125</izz>
                    </inertia>
                </inertial>

                <visual name='visual'>
                    <geometry>
                        <cylinder>
                            <radius>0.4</radius>
                            <length>0.2</length>
                        </cylinder>
                    </geometry>
                    <material>
                        <ambient>1.0 0.0 0.0 1</ambient>
                        <diffuse>1.0 0.0 0.0 1</diffuse>
                        <specular>1.0 0.0 0.0 1</specular>
                    </material>
                </visual>

                <collision name='collision'>
                    <geometry>
                        <cylinder>
                            <radius>0.4</radius>
                            <length>0.2</length>
                        </cylinder>
```

```
                    </geometry>
                </collision>
            </link>

            <!-- Rueda derecha -->
            <link name='rueda_der'>

                <pose relative_to="chassis">-0.5 -0.6 0 -1.5707 0 0</pose>

                <inertial>
                    <mass>1</mass>
                    <inertia>
                        <ixx>0.145833</ixx>
                        <ixy>0</ixy>
                        <ixz>0</ixz>
                        <iyy>0.145833</iyy>
                        <iyz>0</iyz>
                        <izz>0.125</izz>
                    </inertia>
                </inertial>

                <visual name='visual'>
                    <geometry>
                        <cylinder>
                            <radius>0.4</radius>
                            <length>0.2</length>
                        </cylinder>
                    </geometry>
                    <material>
                        <ambient>1.0 0.0 0.0 1</ambient>
                        <diffuse>1.0 0.0 0.0 1</diffuse>
                        <specular>1.0 0.0 0.0 1</specular>
                    </material>
                </visual>

                <collision name='collision'>
                    <geometry>
                        <cylinder>
                            <radius>0.4</radius>
                            <length>0.2</length>
                        </cylinder>
                    </geometry>
                </collision>
            </link>

            <!-- Cuadro para la rueda giratoria -->
            <frame name="caster_frame" attached_to='chassis'>
                <pose>0.8 0 -0.2 0 0 0</pose>
            </frame>

            <!-- Rueda giratoria -->
            <link name='caster'>

                <pose relative_to='caster_frame'/>

                <inertial>
                    <mass>1</mass>
                    <inertia>
                        <ixx>0.1</ixx>
                        <ixy>0</ixy>
                        <ixz>0</ixz>
                        <iyy>0.1</iyy>
                        <iyz>0</iyz>
                        <izz>0.1</izz>
                    </inertia>
                </inertial>

                <visual name='visual'>
                    <geometry>
                        <sphere>
```

```
                            <radius>0.2</radius>
                        </sphere>
                    </geometry>
                    <material>
                        <ambient>0.0 1 0.0 1</ambient>
                        <diffuse>0.0 1 0.0 1</diffuse>
                        <specular>0.0 1 0.0 1</specular>
                    </material>
                </visual>

                <collision name='collision'>
                    <geometry>
                        <sphere>
                            <radius>0.2</radius>
                        </sphere>
                    </geometry>
                </collision>
            </link>


            <!--Articulación para rotar la rueda izquierda -->
            <joint name='rueda_izq_joint' type='revolute'>
                <pose relative_to='rueda_izq'/>
                <parent>chassis</parent>
                <child>rueda_izq</child>
                <axis>
                    <!-- Dada la posición y orientación, giro sobre el eje y-->
                    <xyz expressed_in='__model__'>0 1 0</xyz>
                    <limit>
                        <lower>-1.79769e+308</lower> <!--negativo infinito-->
                        <upper>1.79769e+308</upper> <!--positivo infinito-->
                    </limit>
                </axis>
            </joint>

            <!--Articulación para rotar la rueda derecha -->
            <joint name='rueda_der_joint' type='revolute'>
                <pose relative_to='rueda_der'/>
                <parent>chassis</parent>
                <child>rueda_der</child>
                <axis>
                    <!-- Dada la posición y orientación, giro sobre el eje y-->
                    <xyz expressed_in='__model__'>0 1 0</xyz>
                    <limit>
                        <lower>-1.79769e+308</lower> <!--negativo infinito-->
                        <upper>1.79769e+308</upper> <!--positivo infinito-->
                    </limit>
                </axis>
            </joint>

            <!--Articulación para la rueda giratoria -->
            <joint name='caster_wheel' type='ball'>
                <parent>chassis</parent>
                <child>caster</child>
            </joint>

            <!--Plugin para un control diferencial-->
            <plugin
                filename="gz-sim-diff-drive-system"
                name="gz::sim::systems::DiffDrive">
                <left_joint>rueda_izq_joint</left_joint>
                <right_joint>rueda_der_joint</right_joint>
                <wheel_separation>1.2</wheel_separation>
                <wheel_radius>0.4</wheel_radius>
                <odom_publish_frequency>1</odom_publish_frequency>
                <topic>cmd_vel</topic>
            </plugin>
        </model>
        <!-- -->

```

```
        <!--Pared izquierda-->
        <model name='pared_izq_1'>
            <static>true</static>
            <pose>6.03 0.12 0 0 0 0.54</pose>
            <link name='box'>
                <visual name='visual'>
                    <geometry>
                        <box>
                            <size>0.5 10.0 2.0</size>
                        </box>
                    </geometry>
                    <material>
                        <ambient>0.0 0.0 1.0 1</ambient>
                        <diffuse>0.0 0.0 1.0 1</diffuse>
                        <specular>0.0 0.0 1.0 1</specular>
                    </material>
                </visual>
                <collision name='collision'>
                    <geometry>
                        <box>
                            <size>0.5 10.0 2.0</size>
                        </box>
                    </geometry>
                </collision>
            </link>
        </model>
        <model name='pared_izq_2'>
            <static>true</static>
            <pose> 1.08 3.02 0 0 0 -1.3</pose>
            <link name='box'>
                <visual name='visual'>
                    <geometry>
                        <box>
                            <size>0.5 10.0 2.0</size>
                        </box>
                    </geometry>
                    <material>
                        <ambient>0.0 0.0 1.0 1</ambient>
                        <diffuse>0.0 0.0 1.0 1</diffuse>
                        <specular>0.0 0.0 1.0 1</specular>
                    </material>
                </visual>
                <collision name='collision'>
                    <geometry>
                        <box>
                            <size>0.5 10.0 2.0</size>
                        </box>
                    </geometry>
                </collision>
            </link>
        </model>

        <!--Pared derecha-->
        <model name='pared_der_1'>
            <static>true</static>
            <pose>-2.77 -1.98 0 0 0 -1.3</pose>
            <link name='box'>
                <visual name='visual'>
                    <geometry>
                        <box>
                            <size>0.5 10.0 2.0</size>
                        </box>
                    </geometry>
                    <material>
                        <ambient>0.0 0.0 1.0 1</ambient>
                        <diffuse>0.0 0.0 1.0 1</diffuse>
                        <specular>0.0 0.0 1.0 1</specular>
                    </material>
                </visual>
                <collision name='collision'>
```

```
                    <geometry>
                        <box>
                            <size>0.5 10.0 2.0</size>
                        </box>
                    </geometry>
                </collision>
            </link>
        </model>
        <model name='pared_der_2'>
            <static>true</static>
            <pose>4.2 -4.85 0 0 0 0.54</pose>
            <link name='box'>
                <visual name='visual'>
                    <geometry>
                        <box>
                            <size>0.5 10.0 2.0</size>
                        </box>
                    </geometry>
                    <material>
                        <ambient>0.0 0.0 1.0 1</ambient>
                        <diffuse>0.0 0.0 1.0 1</diffuse>
                        <specular>0.0 0.0 1.0 1</specular>
                    </material>
                </visual>
                <collision name='collision'>
                    <geometry>
                        <box>
                            <size>0.5 10.0 2.0</size>
                        </box>
                    </geometry>
                </collision>
            </link>
        </model>
    </world>
</sdf>
```

Código 10.38: Fichero *.sdf* para la definición del mundo y el robot en *Gazebo Harmonic*.

Nodo de *Gazebo Harmonic* para la simulación de la prueba de control

```cpp
#include <gz/msgs/twist.pb.h>
#include <gz/msgs/laserscan.pb.h>
#include <gz/transport/Node.hh>

std::string topic_pub = "/cmd_vel";   // Publicar en este tema
gz::transport::Node node;
auto pub = node.Advertise<gz::msgs::Twist>(topic_pub);

void cb(const gz::msgs::LaserScan &_msg)
{
  gz::msgs::Twist data;

  // Parámetros
  float target_distance_side = 1.0; // Distancia deseada a la pared lateral
  int index_left = 639;            // Índice para la pared izquierda (+90°)
  int index_right = 0;            // Índice para la pared derecha (-90°)
  int index_front = 320;           // Índice para el frente (0°)
  float threshold_front = 1.5;     // Umbral para evitar colisiones frontales

  ////////////////////////////////////
  // Leer distancias del Lidar
  /* ESTRATEGIA 1 - MEDICIONES FIJAS*/
  //float distance_left = _msg.ranges(index_left);
  //float distance_right = _msg.ranges(index_right);
  //float distance_front = _msg.ranges(index_front);

  /* ESTRATEGIA 2 - DISTANCIA MÍNIMA EN UN RANGO*/
  // Lado izquierdo: índices 320 a 639
  float distance_left = std::numeric_limits<float>::infinity();
  for (int i = 320; i <= 639; ++i) {
      if (_msg.ranges(i) < distance_left) {
          distance_left = _msg.ranges(i);
      }
  }
  // Lado derecho: índices 0 a 320
  float distance_right = std::numeric_limits<float>::infinity();
  for (int i = 0; i <= 320; ++i) {
      if (_msg.ranges(i) < distance_right) {
          distance_right = _msg.ranges(i);
      }
  }
  ////////////////////////////////////

  // Calcular error lateral
  float error_side = distance_left - distance_right;

  // Control proporcional para corrección lateral
  float k_p_side = 1.0; // Ganancia proporcional para la pared lateral
  float angular_correction = k_p_side * error_side;

  data.mutable_linear()->set_x(0.5);        // Velocidad constante hacia adelante
  data.mutable_angular()->set_z(angular_correction); // Corrección basada en la pared
     izquierda

  // Publicar comandos de movimiento
  pub.Publish(data);
}

////////////////////////////////////////////////
int main(int argc, char **argv)
{
  std::string topic_sub = "/lidar";   // Suscribirse a este tema
  // Suscribirse a un tema registrando un callback.
  if (!node.Subscribe(topic_sub, cb))
  {
    std::cerr << "Error suscribiéndose al tema [" << topic_sub << "]" << std::endl;
    return -1;
  }
```

```
68
69    // Esperar por la señal de apagado.
70    gz::transport::waitForShutdown();
71
72    return 0;
73  }
```

Código 10.39: Código del nodo de *Gazebo Harmonic* para la simulación de la prueba de control.