



GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE

Escuela de Ingeniería de Fuenlabrada

Curso académico 2024-2025

Trabajo Fin de Grado

Robot de bajo coste para el mantenimiento de carreteras

Tutor: Julio Vega Pérez

Autor: Julia López Augusto



Este trabajo se distribuye bajo los términos de la licencia internacional CC BY-NC-SA International License (Creative Commons AttributionNonCommercial-ShareAlike 4.0). Usted es libre de (*a*) *compartir*: copiar y redistribuir el material en cualquier medio o formato; y (*b*) *adaptar*: remezclar, transformar y crear a partir del material. El licenciador no puede revocar estas libertades mientras cumpla con los términos de la licencia:

- *Atribución.* Usted debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciante.
- *No comercial.* Usted no puede hacer uso del material con propósitos comerciales.
- *Compartir igual.* Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la la misma licencia del original.

Agradecimientos

Primero de todo me gustaría agradecer a mi tutor Julio toda la confianza y el apoyo depositado para poder hacer posible este proyecto. Sin ti nada de esto hubiera sido posible.

También agradecer a todos los profesores que he tenido a lo largo de la carrera lo importante que habéis sido para mí tanto en mi crecimiento académico, profesional como personal. Gracias a vosotros me habéis hecho ingeniera.

Otras personas que han sido, son y serán fundamentales en mi vida son mis padres; a los cuáles estaré eternamente agradecida por no soltarme nunca de la mano y darme todas las herramientas que han hecho posible ser quien soy hoy en día. Os quiero muchísimo.

Una persona que apareció en mi vida hace ya unos 7 años fue Fran, quien trajo luz en medio de tanta oscuridad y ha sido el mejor acompañante, amigo y amante de cada aventura que se presenta en la vida. Te amo con todo mi corazón y gracias de verdad por todo, ya lo sabes.

Los abuelos deberían ser eternos pero desgraciadamente en mi caso marcharon mucho antes de lo debido. Sé que les encantaría ver a su nieta graduada como ingeniera y me encantaría poder darles un abrazo muy fuerte y poder celebrarlo juntos pero bueno, sé que allá donde estéis estaréis muy orgullosos de mí tanto como yo os echo de menos a vosotros. Gracias por los años que pude conocerlos y disfrutarlos, quedarán para siempre en mi retina.

Los amigos son la familia que uno puede elegir y hoy en día puedo estar muy orgullosa y agradecida de la gente que tengo a mi lado. Gracias a Jimena, Elisa, Sofía, Maryu, Santi, Gonzalo y Álvaro por ser tan maravillosos y por estar ahí estos 4 años de alegrías, lloros, mucho trabajo y dolores de cabeza; sois increíbles. Gracias también a Ángela, Roberto y Lío por aparecer antes de esta aventura y por seguir estando ahí contra viento y marea. Os quiero mucho a todos.

*A mi misma;
por nunca tirar de la toalla*

Chozas de Canales, 26 de Diciembre de 2024

Julia López Augusto

Resumen

Escribe aquí el resumen del trabajo. Un primer párrafo para dar contexto sobre la temática que rodea al trabajo.

Un segundo párrafo concretando el contexto del problema abordado.

En el tercer párrafo, comenta cómo has resuelto la problemática descrita en el anterior párrafo.

Por último, en este cuarto párrafo, describe cómo han ido los experimentos.

Acrónimos

SRI *Stanford Research Institute*

KUKA *Keller und Knappich Augsburg*

ABB *Asea Brown Boveri*

BSA *Backtracking Spiral Algorithm*

FLL *First Lego League*

AGV *Automatic Guided Vehicles*

AMR *Autonomous Mobile Robots*

MITMA *Ministerio de Transportes, Movilidad y Agenda Urbana*

CEDEX *Centro de Estudios y Experimentación de Obras Públicas*

AEC *Asociación Española de la Carretera*

ACEX *Asociación de Empresas de Conservación y Explotación de Infraestructuras*

SIG *Sistema de Información Geográfica*

IA *Inteligencia Artificial*

GPS *Global Positioning System*

DANA *Depresión Aislada en Niveles Altos*

CAD *Computer-Aided Design*

PDCA *Plan Do Check Act*

CSI *Camera Serial Interface*

UART *Universal Asynchronous Receiver-Transmitter*

PWM *Pulse Width Modulation*

LTS *Long Time Support*

CLI *Command Line Interface*

YOLO *You Only Look Once*

RCE *Red de Carreteras del Estado*

DIY *Do It by Yourself*

URDF *Unified Robot Description Format*

mAP *Mean Average Precision*

EMA *Media Móvil Exponencial*

HRI *Interacción Humano-Robot*

VFF *Virtual Force Field*

Índice general

1. Introducción	1
1.1. La robótica	1
1.1.1. Robots industriales	4
1.1.2. Robots de servicio	5
1.2. Robots de campo	8
1.3. Robots de bajo coste	9
1.4. Conservación de carreteras en España	12
2. Estado del arte	16
3. Objetivos	25
3.1. Descripción del problema	25
3.2. Requisitos	26
3.3. Competencias	27
3.3.1. Competencias empleadas	27
3.3.2. Competencias adquiridas	28
3.4. Metodología	29
3.5. Plan de trabajo	29
4. Plataforma de desarrollo	32
4.1. Hardware	32
4.1.1. Raspberry Pi 4	32
4.1.2. Raspberry PiCamera	33
4.1.3. GPS NEO 6M	33

4.1.4. Sevomotor estándar Parallax	34
4.1.5. Ruedas	35
4.1.6. Google Coral USB	35
4.1.7. Power bank	36
4.1.8. Rueda loca	37
4.1.9. Ordenador principal	37
4.2. Software	38
4.2.1. Ubuntu	39
4.2.2. FreeCAD	40
4.2.3. Python	40
4.2.4. OpenCV	42
4.2.5. ROS 2	43
4.2.6. Gazebo	44
4.2.7. Herramientas de monitorización	45
4.2.8. Google Colaboratory	46
4.2.9. YOLOv8	46
4.2.10. TensorFlow Lite	47
4.2.11. Interfaz Web	47
5. Diseño y construcción del robot	49
5.1. Geometría del robot	49
5.2. Disposición de los componentes hardware	51
5.3. Bocetos	52
5.4. Diseño CAD	53
5.4.1. Chasis	54
5.4.2. Soporte de la cámara	58
5.4.3. Carcasa	59
5.4.4. Sujeción trasera	60
5.5. Impresión y montaje	62

6. Soporte software del robot	70
6.1. Simulación	70
6.1.1. URDF/Xacro	70
6.1.2. ROS 2 Control	72
6.1.3. Robot State Publisher	74
6.1.4. Launcher	75
6.1.5. Ejecución en simulador	75
6.2. Ejecución del entorno real	78
6.2.1. Cámara	79
6.2.2. Google Coral USB	81
6.2.3. Módulo GPS	82
6.2.4. Servomotores	83
6.3. Software aplicado al robot físico	84
6.3.1. Red neuronal para la detección de baches	85
6.3.2. Cámara pinhole	90
6.3.3. Algoritmo de la lazada	93
6.3.4. Interfaz Web	94
6.3.5. Autónomo	95
7. Experimentos	100
7.1. ¿Aplicación Completa?	100
7.2. Snippets	100
7.3. Verbatim	101
7.4. Ecuaciones	101
7.5. Tablas o cuadros	102
7.6. Segunda sección	103
7.6.1. Números	103
7.6.2. Listas	103
7.7. Corrector ortográfico	104

8. Conclusiones	105
8.1. Conclusiones	105
8.2. Corrector ortográfico	106
9. Anexo	107
9.1. Construcción del robot	107
9.2. Ejecución del robot en simulación	107
9.3. Configuración del robot real	108
9.3.1. Cámara	108
9.3.2. Google Coral	109
9.3.3. Módulo GPS	109
9.3.4. Módulo GPS	113
Bibliografía	114

Índice de figuras

1.1.	Ingenios de la antigüedad con fines religiosos	2
1.2.	Ingenios de la antigüedad con fines no religiosos	2
1.3.	Electro y Sparko	3
1.4.	Algunos robots del siglo XX	4
1.5.	Robots industriales	5
1.6.	Robots de limpieza Roomba, de iRobot ⁶	6
1.7.	Robótica enfocada al entretenimiento	7
1.8.	Robots de salud	7
1.9.	Robots de logística	8
1.10.	Robots de campo	9
1.11.	Mano Antropomórfica Híbrida Rígida y Suave	10
1.12.	Clasificación de deterioros	13
1.13.	Agrietamiento	14
1.14.	Degradoación del material de la capa de rodadura	14
1.15.	Deformación de la capa de rodadura sin degradación de material	15
1.16.	Otro tipo de daños	15
2.1.	Robot usado para la reconstrucción 3D de baches	17
2.2.	Proyecto HERON	18
2.3.	Visualización de la señal en un mapa en eyesNroad	19
2.4.	Plataforma robótica diseñada para el sellado de grietas	21
3.1.	Método PDCA	30

4.1.	Raspberry Pi 4 ³⁸	33
4.2.	Raspberry PiCamera V2 ³⁹	34
4.3.	Módulo GPS NEO 6M ⁴⁰	34
4.4.	Servomotor Parallax ⁴¹	35
4.5.	Ruedas utilizadas	36
4.6.	Google Coral USB ⁴⁴	36
4.7.	Xiaomi power bank ⁴⁵	37
4.8.	Rueda loca ⁴⁶	37
4.9.	ASUS VivoBook 14 ⁴⁷	38
4.10.	Logo de Ubuntu	39
4.11.	Logo de Freecad	40
4.12.	Logo de Python	41
4.13.	Sentencias NMEA capturadas del GPS	42
4.14.	Logo de OpenCV	43
4.15.	Distribuciones de ROS 2 usadas	44
4.16.	Logo de Gazebo	45
4.17.	Logo de Rviz	45
4.18.	Logo de Google Colab	46
4.19.	Logo de YOLOv8	47
4.20.	Logo de TensorFlow Lite	47
4.21.	Logo de Open Street Maps	48
5.1.	PiBot	50
5.2.	PiBot con cámara modificada	50
5.3.	Esquema de los grados de libertad de PiBot	51
5.4.	Esquema de conexiones del PiBotJ	52
5.5.	Bocetos creados a mano	53
5.6.	Maqueta en proceso	54
5.7.	Maqueta final	55

5.8. Planos de los componentes	56
5.9. Comprobación mediante calibre	57
5.10. Distintas vistas del chasis	57
5.11. Distintas vistas del chasis atornillado	58
5.12. Inclinación de la cámara	58
5.13. Distintas vistas del soporte de la cámara	59
5.14. Distintas vistas del soporte de la cámara atornillado	59
5.15. Distintas vistas de la carcasa	60
5.16. Distintas vistas de la carcasa atornillada	61
5.17. Distintas vistas de la pieza para la sujeción trasera	61
5.18. Pieza para la sujeción trasera atornillada	62
5.19. Impresora FDM Creality Ender3 V2 ⁷⁹	62
5.20. Piezas impresas	63
5.21. Hama Beads ⁸⁰	64
5.22. Soldando pines al módulo GPS	64
5.23. Robot completo montado en FreeCAD	66
5.24. PiBotJ con ruedas de ActivityBot	67
5.25. Ensamblaje ruedas azules genéricas	68
5.26. PiBotJ con ruedas azules	69
6.1. Sistemas de Coordenadas de PiBotJ	73
6.2. Interfaces que tiene definidas PiBotJ	74
6.3. Diagrama de robot_state_publisher	75
6.4. Esquema de launch_sim.launch.py	76
6.5. Topics disponibles al lanzar el robot	76
6.6. Herramienta usada para mover las ruedas	77
6.7. Herramienta usada para visualizar la cámara	77
6.8. Herramienta usada para visualizar los valores del GPS	78
6.9. Herramienta usada para rotar la cámara	78

6.10. Cámara de PiBotJ en funcionamiento	79
6.11. Media de la matriz de intrínsecos	81
6.12. Parámetros extrínsecos	81
6.13. Valores publicados de latitud y longitud captados del GPS	82
6.14. Demostración de una correcta localización	83
6.15. Diagrama para la posición central del servo	84
6.16. Distintas direcciones de PiBotJ	84
6.17. Gráficas de resultados del entrenamiento	85
6.18. Gráficas de resultados del entrenamiento: métricas	86
6.19. Demostración del modelo funcionando	87
6.20. Contorno del baches detectados	91
6.21. Funcionamiento de una cámara esteneopeica ¹²⁴	91
6.22. Relación de fórmulas del modelo pinhole	92
6.23. Sistemas de coordenadas	92
6.24. La hipótesis suelo asume que todos los objetos están en el suelo	93
6.25. Demostración del algoritmo de la lazada	94
6.26. Interfaz web	95
6.27. Conversión de medidas de la carretera	96
6.28. Carretera creada en cartulina	96
6.29. Filtro resultante aplicado para la detección de líneas	97
6.30. Distintos casos de la detección de líneas	98
6.31. Diagrama de fuerzas del algoritmo VFF	99
7.1. Robot aspirador Roomba de iRobot.	103
9.1. Dispositivos de video y media disponibles en PiBotJ	109
9.2. Configuración exitosa del Google Coral	110

Listado de códigos

6.1.	Macro que define <i>fixed joint</i>	71
6.2.	Macro que define una <i>mesh link</i>	72
6.3.	Macro que permite a Gazebo simular una cámara	73
6.4.	Cómo convertir los datos del modelo a un formato adecuado	89
6.5.	Filtro para obtener las líneas blancas	97
7.1.	Función para buscar elementos 3D en la imagen	101
7.2.	Cómo usar un Slider	101

Listado de ecuaciones

6.1.	Fórmula para calcular la distancia focal teórica	80
6.2.	Fórmula para calcular el centro de la imagen	80
6.3.	Fórmula de la media móvil exponencial	90
6.4.	Fórmula del algoritmo de la lazada	94
6.5.	Ecuación de VFF	98
7.1.	Ejemplo de ecuación con fracciones	102
7.2.	Ejemplo de ecuación con array y letras y símbolos especiales	102

Índice de cuadros

2.1. Ventajas y desventajas de los métodos de detección de baches	23
4.1. Especificaciones técnicas del ordenador usado	38
4.2. Diferencias entre Ubuntu 20.04 y Ubuntu 22.04	40
5.1. Características usadas para la impresión	63
5.2. Tornillería necesaria	65
5.3. Coste proyecto	67
7.1. Parámetros intrínsecos de la cámara	102

Capítulo 1

Introducción

La motivación nos impulsa a comenzar y el hábito nos permite continuar

Jim Ryun

La robótica ha sufrido una transformación enorme a lo largo de su historia, aunque siempre teniendo en mente el mismo objetivo: cumplir con el deseo humano. Debido a esa transformación y ese deseo se ha podido consolidar este campo en la actualidad, que abarca cada sector que se pueda imaginar. Otra vertiente que ha destacado en la robótica estos últimos años ha sido la creación de robots de bajo coste para que puedan llegar a un mayor número de personas y se puedan beneficiar de esta ciencia.

En el presente capítulo se va a abordar el contexto de la robótica, explicando brevemente su historia para poder entender realmente qué es la robótica y lo que es un robot. También se van a encuadrar los tipos de robots que existen y sus múltiples aplicaciones. Todo esto nos ayudará a poder entender mejor dónde se encuadra el presente trabajo, proporcionando los conocimientos necesarios, tanto teóricos como prácticos, que se describirán a lo largo del documento.

1.1. La robótica

La robótica es el campo de la ingeniería que se enfoca en el diseño, la construcción y la programación de robots. Y un robot se podría definir como un sistema informático formado por sensores y actuadores imprecisos, ya que operan en el mundo real que es imperfecto también. Los robots realizan tareas repetitivas, aburridas y peligrosas, y tienen que ser sensibles al entorno. Sin embargo, no existe una definición unívoca al respecto y depende del campo y de la época de la que queramos hablar. Para poder entenderlo, se va a hacer un breve resumen sobre la historia de la robótica.

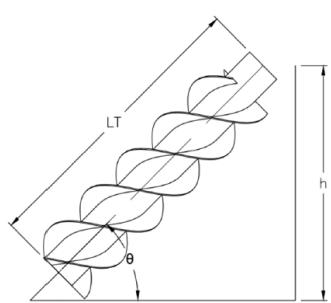
Desde la antigüedad se han desarrollado ingenios o autómatas, de los cuales muchos de ellos tenían fines religiosos, como los mostrados en la Figura 1.1.

Estatuas de Memnon ¹Guerreros de Terracota ²

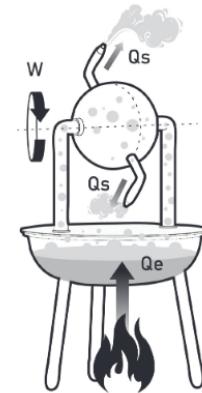
Figura 1.1: Ingenios de la antigüedad con fines religiosos

Otras invenciones que destacaban por otras aplicaciones fueron el tornillo de Arquímedes de Siracusa, la eolípila de Herón de Alejandría, el ornitóptero de Leonardo Da Vinci y el hombre de palo de Juanelo Turriano, entre otras.

En [Cuenca Sánchez et al., 2023] se trata el principio de generación hidroeléctrica, tomando en cuenta el modelo tornillo de Arquímedes. También en el artículo [Giri, 2020] se traza una línea histórica de las máquinas térmicas, teniendo en cuenta a la eolípila. Dichas invenciones aparecen reflejadas en la Figura 1.2.



Tornillo de Arquímedes



Eolípila

Figura 1.2: Ingenios de la antigüedad con fines no religiosos

Durante el siglo XX la ciencia dejó de ser una actividad desarrollada en aislamiento y se desarrolló en laboratorios con más gente. El movimiento del positivismo lógico fomentó las investigaciones, ya que este movimiento trataba de dar importancia a la

¹https://es.wikipedia.org/wiki/Colosos_de_Memn%C3%B3n

²https://es.wikipedia.org/wiki/Guerreros_de_terracota

ciencia y dejar de lado la filosofía; también, el contexto de las guerras mundiales y de las bombas nucleares influyeron significativamente en este aspecto. Ya se empieza a acuñar la palabra robot y surge Electro y Sparko de Westinghouse Electric Corporation (Figura 1.3), tratado en muchas ocasiones como uno de los primeros robots, como se describe en [Bidaud, 2017]. Un descubrimiento muy destacado fue, y sigue siendo hoy en día, la Leyes de la Robótica de Isaac Asimov, así lo transmite [Barceló, 2004].



Figura 1.3: Electro y Sparko

[Nilsson et al., 1984] nos cuenta que en 1969 se construye por el *Stanford Research Institute* (SRI) Internacional un prototipo experimental llamado Shakey, que era una unidad independiente de un metro y medio, equipado con dos motores, cámara de televisión y una radio conectada a un ordenador, capaz de navegar en entornos cerrados y estructurados de una forma autónoma. Sus objetivos eran aprender del medio y ser capaz de planificar trayectorias de movimiento, y las tareas que le asignaron fueron mover y detectar bloques. Sin embargo, cada movimiento podría tardar más de una hora en computarse y, aún así, podrían producirse fallos. A Shakey se le conoce como el primer robot móvil (Figura 1.4 izquierda).

Otro robot muy conocido y descrito en [Earnest, 2012] es la carreta de Stanford,

que era capaz de ver y moverse en cualquier ambiente. Con la cámara que disponía, era capaz de calcular y trazar distancias. Sin embargo, tardaba cinco horas en recorrer treinta metros (Figura 1.4 derecha).



Robot Shakey



Carreta de Stanford

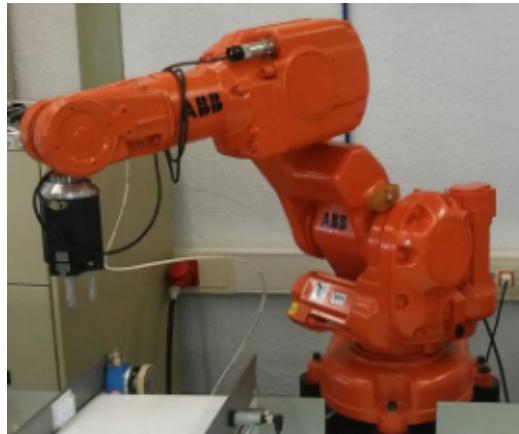
Figura 1.4: Algunos robots del siglo XX

Ya a partir de la década de los 80 se decidió que el acceso a los robots fuese para todo el mundo, lo que provocó asombro, inquietud y miedo, ya que el desconocimiento suele generar rechazo. El mundo de la literatura y cine tampoco ayudaba en ese aspecto, ya que se presentaba al robot como algo perjudicial para la humanidad. Afortunadamente, esta situación va menguando con el tiempo, y se está consiguiendo ver a los robots como un asistente del ser humano que lo que quiere es mejorar su calidad de vida. Las dos grandes áreas de investigación centradas en tal propósito son la robótica industrial y la robótica de servicio, que veremos detalladamente a continuación.

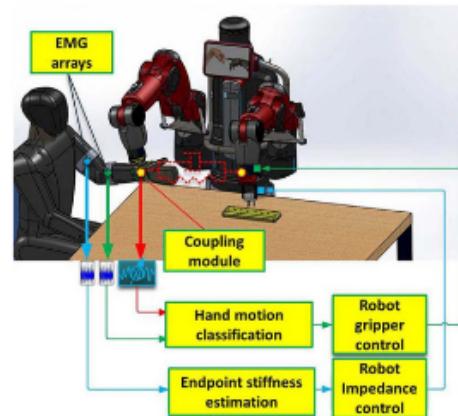
1.1.1. Robots industriales

Los robots industriales son, de forma resumida, brazos robotizados y manipuladores que tienen más de tres grados de libertad, trabajan en entornos controlados y usan efectores como: pinzas, ventosas, etc. Usan control por posición y planificación de trayectorias para poder controlar dichos brazos y poder realizar operaciones como *pick and place*, ensamble de piezas, entre otros. Una variante de los robots industriales son los cobots: capaces de interactuar y colaborar con humanos como se describe

en [El Zaatari et al., 2019] (Figura 1.5 derecha). Dentro del mercado de los robots industriales se puede ver que tiene proveedores asentados como KUKA³ o ABB⁴.



Brazo robot ABB IRB 140 ⁵



Cobot

Figura 1.5: Robots industriales

1.1.2. Robots de servicio

Los robots de servicios son todos aquellos que no son industriales; por lo tanto, tienen menos de 3 grados de libertad, no trabajan en un entorno controlado, son más difíciles de programar, toman aplicaciones heterogéneas y todavía se encuentran en un mercado inmaduro. Existen muchos tipos de robots de servicio, de los cuales a continuación se enumeran algunos campos con algunos de sus respectivos ejemplos.

Robots de limpieza

En [Plaza, 2023] se define a los robots de limpieza como aquellos robots que se encargan de eliminar la suciedad. Dependiendo de sus características, pueden ser capaces de aspirar y fregar el suelo, o de limpiar los cristales de las ventanas. Estos últimos son comunes en edificios que tienen grandes ventanales y un difícil acceso a ellos. Las aspiradoras se encuentran en un mercado mundial asentado cuyos inicios eran aspiradoras que usaban sensores de contacto, encoders y una navegación pseudoaleatoria (Figura 1.6 izquierda), y han evolucionado hasta el punto de usar mapas para poder navegar y poder localizarse. También usan algoritmos sofisticados, como navegación de cobertura por barridos sistemáticos BSA, y se componen de

³<https://www.kuka.com/es-es>

⁴<https://new.abb.com/es>

⁵<https://www.youtube.com/watch?v=BBrLA0r89KY>

sensores más sofisticados, como láseres y cámaras, que son capaces de detectar obstáculos, como calcetines, y ser capaz de esquivarlos (Figura 1.6 derecha).



Modelo económico



Gama alta

Figura 1.6: Robots de limpieza Roomba, de iRobot⁶

Robots de entretenimiento

Son aquellos robots que tienen aplicaciones heterogéneas en un mercado educativo asentado, pero también existen muchos prototipos sin un uso comercial claro. En educación se ha ido introduciendo la robótica de manera muy atractiva y didáctica a los estudiantes hasta el punto de conseguir tener una asignatura destinada a la robótica y poder participar en competiciones como: *First Lego League* (FLL)⁷, Robocup Junior⁸ y Robocampeones⁹, entre otros (Figura 1.7 izquierda). En dicha asignatura se adquieren conocimientos generales de programación, impresión 3D, lógica, introducción a la electrónica y los microcontroladores.

Los prototipos nombrados anteriormente suelen ser demostradores tecnológicos que se crean para exhibiciones, se encuentran a la vanguardia de la tecnología y sirven para atraer un mayor número de clientes como puede ser: Spot de Boston Dynamics, Pepper de Softbank o Sophia de Hanson Robotics. En la Figura 1.7 (derecha) se pueden ver un ejemplo de estos demostradores.

Robots de salud

Los robots de salud sirven para mejorar la calidad de la atención médica y apoyar a los profesionales de la salud en diversas tareas. De esas tareas se pueden enumerar las siguientes: telepresencia, asistentes personales, cirugía, desinfección, esterilización,

⁶<https://www.irobot.es/>

⁷<https://firstlegoleague.soy/>

⁸<https://junior.robocup.org/>

⁹<https://sites.google.com/view/robocampeonesfuenlabrada/>

¹⁰<https://www.uclm.es/noticias/febrero2019/toledo/finalfirstlegoleague>

¹¹<https://bostondynamics.com/products/spot/>

FLL Toledo ¹⁰Spot de Boston Dynamics ¹¹

Figura 1.7: Robótica enfocada al entretenimiento

transporte interno y rehabilitación. De este gran abanico de tareas se puede destacar algunas aplicaciones que se puede ver en la Figura 1.8.

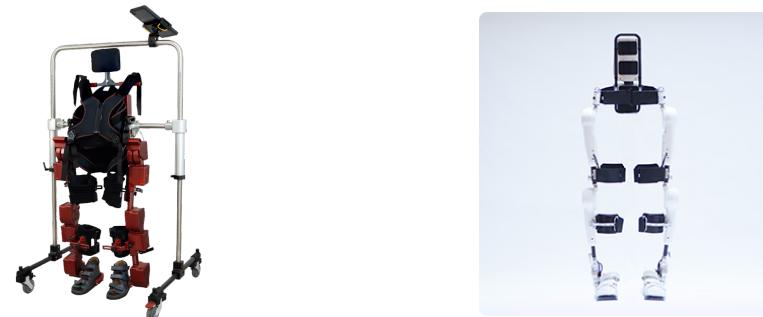
Robot Da Vinci ¹²Robot Mako ¹³Marsi Bionics ¹⁴CyberDyne ¹⁵

Figura 1.8: Robots de salud

¹²<https://www.abexsl.es/es/sistema-robotico-da-vinci/que-es>

¹³<https://www.stryker.com/content/dam/stryker/joint-replacement/systems/mako-system-overview/resources>

¹⁴<https://www.marsibionics.com/>

¹⁵<https://www.cyberdyne.com/>

Robots de logística

Los robots de logística son sistemas automatizados diseñados para mejorar la eficiencia, precisión y velocidad en la gestión de la cadena de suministro y operaciones logísticas en cadenas de montaje y almacenes. Generalmente, los sistemas automatizados son flotas de robots a las que se les aplica distintas arquitecturas *software* como puede ser AGV o AMR para poder realizar la tarea asignada. También existen prototipos de robots de reparto que son capaces de hacer entrega de última milla. La Figura 1.9 muestra ejemplos de robots de reparto.



Skypod de Exotec ¹⁶



Amazon Prime Air ¹⁷

Figura 1.9: Robots de logística

1.2. Robots de campo

En [Thorpe and Durrant-Whyte, 2003] se define a los robots de campo como la automatización de muchas plataformas terrestres, marítimas y aéreas en aplicaciones como la minería, la manipulación de carga, la agricultura, la exploración y explotación submarina, las carreteras, la exploración planetaria, la vigilancia costera y el rescate, entre otros. La robótica de campo se caracteriza por la aplicación de los principios robóticos más avanzados en cuanto a sensado, control y razonamiento en entornos no estructurados y difíciles. El atractivo de la robótica de campo es que es una ciencia desafiante, involucra los últimos principios de ingeniería y diseño de sistemas, y ofrece la verdadera posibilidad de que los principios robóticos hagan una contribución económica y social sustancial en muchas áreas de aplicación diferentes. En general, los robots de campo son plataformas móviles que trabajan al aire libre, a menudo produciendo interacciones fuertes con sus entornos, sin supervisión humana.

En los últimos años se ha podido notar un gran progreso en el desarrollo y la implementación de sistemas robóticos de campo. En la Figura 1.10 se pueden ver

¹⁶<https://exotecbydexter.com/skypod/>

¹⁷<https://www.aboutamazon.es/noticias/innovacion/prime-air>

ejemplos al respecto. Por contra, debido a las condiciones que se tienen que someter estos robots, su coste es elevado e imposibilita que su uso se pueda extender a aquellos lugares que necesiten de su servicio y que tienen bajos recursos. Es por esto que existe la creciente necesidad de que, para ciertas aplicaciones que no tienen condiciones tan extremas, se creen robots asequibles que puedan lidiar con determinadas tareas. A continuación veremos este campo de investigación.



TX Robotic Strawberry Harvester¹⁸



FrontRunner
Autonomous Haulage System (AHS)¹⁹



Drone Tello²⁰



Perseverance²¹

Figura 1.10: Robots de campo

1.3. Robots de bajo coste

Los robots de bajo coste son aquellos robots diseñados y fabricados con el objetivo de ser económicos, accesibles y fáciles de producir. Estos robots suelen emplear componentes menos costosos y métodos de fabricación simplificados para reducir el precio final. Algunas características clave de los robots de bajo coste incluyen:

- *Componentes asequibles.* Utilizan materiales de propósito general y componentes electrónicos más baratos, coordinados por alguna placa de bajo coste como Arduino o Raspberry Pi.

¹⁸<https://advanced.farm/technology/strawberry-harvester/>

¹⁹<https://www.komatsu.com/en/technology/smart-mining/loading-and-haulage-autonomous-haulage-system/>

²⁰<https://www.ryzerobotics.com/es/tello>

²¹<https://es.wikipedia.org/wiki/Perseverance>

- *Simplicidad en el diseño.* Tienen diseños más sencillos que han sido creados usando técnicas de diseño e impresión 3D, facilitando su actualización y mantenimiento.
- *Accesibilidad.* Están diseñados para ser utilizados por cualquier tipo de persona, sin tener una formación avanzada de la materia.
- *Educación y prototipos.* Ampliamente usados en educación para facilitar el aprendizaje y también en la creación de prototipos rápidos y asequibles.
- *Versatilidad.* Se adaptan a numerosas aplicaciones, desde las más sencillas hasta proyectos más complejos.

En resumen, los robots de bajo coste permiten la democratización de la tecnología robótica, facilitando su acceso a un público más amplio y fomentando la innovación y el aprendizaje en diferentes campos. En la Figura 1.11 se puede apreciar una aplicación real reciente de la robótica de bajo coste de la universidad *Carnegie Mellon* descrita en el artículo [Shaw and Pathak, 2024].



Figura 1.11: Mano Antropomórfica Híbrida Rígida y Suave

Con esta base tecnológica, uno de los desafíos donde los robots de bajo coste están mostrando un gran potencial es en tareas repetitivas, como las tareas de mantenimiento, por ejemplo de infraestructuras como las carreteras. A medida que se buscan soluciones más eficientes y económicas para mantener y mejorar las infraestructuras, los robots de bajo coste pueden ser una pieza clave en este proceso.

El mantenimiento de infraestructuras públicas, y en especial el de las carreteras, es una tarea de vital importancia para garantizar la seguridad y el bienestar de las personas. Sin embargo, también representa un reto logístico y económico significativo. Las carreteras, expuestas constantemente a condiciones climáticas adversas, tráfico pesado y el desgaste natural, requieren un mantenimiento constante para prevenir accidentes y asegurar la movilidad eficiente de personas y mercancías.

Tradicionalmente, este mantenimiento ha dependido de métodos manuales y laboriosos. Equipos de trabajadores inspeccionan las carreteras, identifican los daños y proceden a repararlos, lo que implica un alto coste en tiempo, recursos humanos y materiales. Además, la intervención en las carreteras conlleva cortes de tráfico que generan congestión y molestias tanto para conductores como para peatones.

Aquí es donde la tecnología de los robots de bajo coste puede ofrecer una solución disruptiva. El desarrollo de estos robots ha llegado a un punto en el que pueden ser integrados en el proceso de mantenimiento de carreteras de manera eficaz. Al ser equipados con actuadores, sensores avanzados y algoritmos de *Inteligencia Artificial* (IA), estos robots tienen la capacidad de detectar y arreglar automáticamente irregularidades en el pavimento, evaluando su tamaño y gravedad. Esta automatización permitiría realizar inspecciones más frecuentes y precisas, reduciendo el margen de error humano, facilitando intervenciones más rápidas y localizadas, y mejorando las condiciones de seguridad de los trabajadores.

La posibilidad de desplegar múltiples robots de bajo coste a lo largo de una red de carreteras también representa una ventaja significativa. Estos robots pueden realizar inspecciones de forma continua, recorriendo largas distancias y detectando problemas antes de que se conviertan en riesgos graves para la seguridad vial. Además, los robots pueden operar en entornos donde el acceso humano es complicado o peligroso, como en carreteras rurales o áreas montañosas.

En este contexto, la versatilidad y el bajo coste de los robots también resultan beneficiosos cuando se trata de reparaciones. Actualmente, las reparaciones suelen ser costosas y temporales, lo que significa que el mismo área puede requerir múltiples intervenciones a lo largo del tiempo. Sin embargo, con robots capaces de aplicar reparaciones rápidas y precisas en el lugar y momento adecuados, se podría reducir la frecuencia de intervenciones costosas y prolongar la vida útil del pavimento.

Es por todo lo expuesto anteriormente que es necesario conocer la situación de las carreteras en España, conocer qué tipos de deterioros existen, así cómo definir en cuál de todos se va a centrar este proyecto. De este modo, se podrá implementar soluciones

innovadoras, como el uso de robots de bajo coste, que no solo optimicen los recursos, sino que también mejoren la seguridad vial y la sostenibilidad del sistema de carreteras a largo plazo.

1.4. Conservación de carreteras en España

Según informa el *Ministerio de Transportes, Movilidad y Agenda Urbana* (MITMA)²² la red de carreteras de España tiene, a 31 de diciembre de 2023, 165.375 kilómetros, de los cuales 26.473 km forman la *Red de Carreteras del Estado* (RCE), que gestiona el MITMA y recoge el 52,5 % del tráfico total y el 64,57 % del tráfico pesado. Además, hay 71.145 km que están gestionados por las Comunidades Autónomas y soportan el 42 % del tráfico, y 67.770 km por las Diputaciones (que suponen el 5,5 % del tráfico restante).

España es uno de los países que tiene mayor número de kilómetros de carreteras, y es por ello que tienen que existir distintas entidades que se encarguen de supervisar su mantenimiento. Además de las expuestas previamente, hay varias organizaciones que juegan un papel destacado en la promoción, estudio y mejora continua de las infraestructuras viarias. El *Centro de Estudios y Experimentación de Obras Públicas* (CEDEX)²³, por ejemplo, es un organismo de referencia en la investigación y experimentación en el ámbito de las obras públicas y la movilidad. Fundado en 1957, el CEDEX trabaja para mejorar y conservar las infraestructuras, impulsar una movilidad segura y sostenible, y proteger el medioambiente.

Otra entidad relevante es la *Asociación Española de la Carretera* (AEC)²⁴, una entidad sin ánimo de lucro fundada en 1949, que trabaja en la defensa y promoción de las carreteras. La AEC se enfoca en aspectos como la seguridad vial, la sostenibilidad y la calidad de las infraestructuras, adaptando sus actividades a las necesidades y desafíos contemporáneos, como la digitalización y la descarbonización del transporte.

En el ámbito de la conservación propiamente dicha, la *Asociación de Empresas de Conservación y Explotación de Infraestructuras* (ACEX)²⁵, creada en 1995, agrupa a empresas dedicadas a la conservación de carreteras y se centra en promover la eficiencia y sostenibilidad en el mantenimiento de estas infraestructuras, así como en mejorar

²²<https://www.transportes.gob.es/carreteras/catalogo-y-evolucion-de-la-red-de-carreteras>

²³<https://www.cedex.es/presentacion>

²⁴<https://www.aecarretera.com/quienes-somos>

²⁵<https://www.acex.eu/la-asociacion/>

la seguridad vial y laboral. También es reseñable destacar sus premios anuales²⁶ que permiten avanzar a pasos agigantados en materia de conservación y seguridad vial.

Sin embargo, parece ser que la labor de todas estas plataformas no es suficiente. Noelia Soage²⁷, redactora en ABC Motor, cuenta que se estuvieron inspeccionando 13.000 kilómetros del total de la red de carreteras de España de las cuales, se presentan graves deterioros en más del 50%; los cuales pueden afectar a la estructura o a la superficie de la plataforma, comprometiendo la comodidad, eficiencia y seguridad de la circulación.

Para poder dar mayor contexto a los deterioros, es necesario hacer una clasificación de ellos sabiendo que se pueden presentar en pavimentos flexibles, semiflexibles y semirrígidos urbanos, y los podemos clasificar en cuatro grandes categorías: agrietamiento (Figura 1.13), degradación del material de la capa de rodadura (Figura 1.14), degradación de la capa de rodadura sin degradación de material (Figura 1.15) y otro tipo de daños (Figura 1.16). Todo está esquematizado en la Figura 1.12.



Figura 1.12: Clasificación de deterioros

²⁶<https://www.acex.eu/premios-acex/>

²⁷<https://www.abc.es/motor/reportajes/emisiones-accidentes-aumento-mal-estado-carreteras-senales-20230309230204-nt.html?ref=https>



Grietas longitudinales

Grietas transversales

Grietas piel de cocodrilo

Figura 1.13: Agrietamiento



Bache



Meteorización



Blandón



Hueco

Figura 1.14: Degradación del material de la capa de rodadura

Tras haber mostrado los tipos de deterioros que existen, nos centramos en el mantenimiento de uno de ellos, ya que el problema es muy grande e intentar tratar todos los casos se torna inabordable para este proyecto. Así, de todos los tipos mostrados, se van a tratar los baches.

Una vez conocido el deterioro en que se va a centrar este proyecto, los organismos existentes en España y todas las oportunidades que un robot *low-cost* ofrece para el mantenimiento de carreteras, se puede decir que este proyecto se centra en el desarrollo de un robot de campo de tamaño compacto y bajo coste, diseñado para ser fácil de usar y controlar. El robot ha sido fabricado completamente mediante impresión 3D y está equipado con herramientas ampliamente utilizadas en el campo de la robótica.



Figura 1.15: Deformación de la capa de rodadura sin degradación de material



Figura 1.16: Otro tipo de daños

Esta combinación permite que cualquier persona, incluso sin un conocimiento profundo en robótica, pueda replicar el robot y ponerlo en funcionamiento. En el próximo capítulo, se presentarán diversos prototipos y futuras aplicaciones que guardan una cierta relación con el tipo de robot desarrollado.

Capítulo 2

Estado del arte

Todo progreso depende de la irracionalidad del hombre razonable.

George Bernard Shaw

En el presente capítulo se van a describir algunos de los prototipos y posibles aplicaciones más destacables sobre la detección y mantenimiento del pavimento de las carreteras usando inteligencia artificial y técnicas robóticas. En este estado del arte se revisan más de diez soluciones tecnológicas innovadoras desarrolladas recientemente.

Sistema de reconstrucción 3D de baches

Este sistema, explicado en [Bruno et al., 2023], es creado como parte del proyecto Infrarob²⁸ y desarrollado en el marco del proyecto europeo Horizon 2020, utiliza una Raspberry Pi 4B acoplada a un robot autónomo para capturar imágenes usando una cámara y coordenadas GPS, usando el módulo de baches en carreteras. Las imágenes, tomadas desde diferentes ángulos, son procesadas mediante técnicas fotogramétricas para generar modelos 3D de los baches, permitiendo calcular con precisión el volumen que debe ser rellenado. El sistema también integra un *Sistema de Información Geográfica* (SIG) para mejorar la gestión del mantenimiento de pavimentos. (Figura 2.1)

Las ventajas de este proyecto son las siguientes:

- *Bajo coste.* Utiliza componentes de bajo coste como: Raspberry Pi, Raspberry PiCamera y el módulo GPS NEO-6M, que también es *low-cost*.

²⁸<https://infrarobproject.com/>



Figura 2.1: Robot usado para la reconstrucción 3D de baches

- *Precisión en la detección.* Capacidad para detectar baches de hasta 75 cm de diámetro y crear modelos 3D precisos.
- *Integración con SIG.* Facilita la planificación de reparaciones.

Las desventajas de este proyecto son las siguientes:

- *Limitaciones climáticas.* No puede operar en condiciones adversas ni durante la noche.
- *Uso de software de pago.* Utiliza ContextCapture, que es de pago, para el procesamiento fotogramétrico.
- *Movimiento circular para detección del volumen.* Una vez detectado el bache, es necesario moverse alrededor de él para que el algoritmo de fotogrametría sea capaz de encontrar su volumen, lo que ralentiza el cálculo.

El proyecto Herón

En [Katsamenis et al., 2022] se habla del Proyecto HERON²⁹ que también parte del programa Horizon 2020. Este proyecto propone una solución integral para el mantenimiento de infraestructuras viales mediante el uso de vehículos modulares

²⁹<https://www.heron-h2020.eu/>

robóticos autónomos y drones. El sistema está diseñado para optimizar la eficiencia y seguridad de las operaciones de mantenimiento, utilizando sensores y escáneres para crear mapas 3D, inteligencia artificial para coordinar flujos de trabajo, y módulos de análisis de imágenes para detectar defectos. Todavía está en etapa de desarrollo. (Figura 2.2)



Figura 2.2: Proyecto HERON

Las ventajas de este proyecto son las siguientes:

- *Automatización completa.* Minimiza la intervención humana y reduce riesgos laborales.
- *Eficiencia.* El objetivo es mejorar la capacidad y eficiencia de las redes viales.
- *Intercambio de datos en tiempo real.* Otro de los objetivos es mejorar la toma de decisiones
- *Diseño modular.* Para poder maximizar sus capacidades, facilitar el transporte y reducir los costes de mantenimiento y accidentes.

Las desventajas de este proyecto son las siguientes:

- *Coste inicial elevado.* La implementación de tecnologías avanzadas es costosa.
- *Dependencia tecnológica.* Requiere una infraestructura tecnológica robusta para su funcionamiento.

EyesNroad

EyesNroad es una plataforma que emplea IA y *Machine Learning* para identificar baches, señalización y otros deterioros en carreteras a partir de videos capturados por cámaras georreferenciadas instaladas en vehículos. La plataforma genera inventarios detallados y reportes del estado de las carreteras, facilitando el control y mantenimiento de las mismas. Esta aplicación ha sido desarrollada para participar en los XX premios de conservación de carreteras de ACEX (año 2024)³⁰, que ha resultado ganadora en la en la categoría general. Se puede ver su aspecto en la Figura 2.3.



Figura 2.3: Visualización de la señal en un mapa en eyesNroad

Las ventajas de este proyecto son las siguientes:

- *Facilidad de implementación.* Puede utilizarse con cámaras comunes y es compatible con dispositivos móviles.
- *Amplia detección.* No solo identifica baches, sino también señalización y otros elementos viales.
- *Actualización continua.* La base de datos se amplía constantemente para cubrir más regiones.

Las desventajas de este proyecto son las siguientes:

³⁰<https://www.acex.eu/candidatura-3-4/>

- *Limitación geográfica.* Actualmente, su funcionamiento completo está limitado a Portugal.
- *Es de pago.* Para cualquier persona que desee usar esta herramienta tiene que asumir los costes.
- *No calcula volumen.* La plataforma no ofrece estimaciones del volumen de los baches, lo que podría ser útil para reparaciones.

El proyecto OMICRÓN

En este proyecto se ha creado un sistema robótico que busca automatizar y robotizar el proceso de sellado de grietas en pavimentos, con el objetivo de eliminar la exposición de los operarios a condiciones peligrosas. El sistema integra un brazo robótico y herramientas de inteligencia artificial para detectar grietas y realizar el sellado de manera autónoma, mejorando así la seguridad y eficiencia en las operaciones de mantenimiento. Esta aplicación ha sido desarrollada para participar en los XX premios de conservación de carreteras de ACEX (año 2024)³¹, que ha resultado ganadora en la en la categoría asociados. Puedes ver su aspecto en la Figura 2.4.

Las ventajas de este proyecto son las siguientes:

- *Seguridad laboral.* Elimina la necesidad de que los operarios sufran accidentes de tráfico y tengan contacto con materiales peligrosos.
- *Reducción de tiempo.* Debido a la automatización del proceso, se puede reducir el tiempo de intervención.
- *Innovación en materiales.* Al invertir en crear un mástico nuevo, se puede operar a temperaturas más bajas.

Las desventajas de este proyecto son las siguientes:

- *Es un prototipo.* Todavía está en fase temprana de implementación lo que le impide a los clientes tener una disponibilidad inmediata.
- *Alto coste.* Necesita componentes caros para poder llevar a cabo sus objetivos, como un camión o un brazo robótico, entre otros.

³¹<https://www.acex.eu/candidatura-12-5/>



Figura 2.4: Plataforma robótica diseñada para el sellado de grietas

- *Complejidad técnica.* Ya que se trata de una automatización completa, necesita un alto nivel de precisión y coordinación, lo que supone un gran desafío para su implementación.

Sistemas de reconstrucción 3D usando SFM y aprendizaje profundo

Este sistema, descrito en [Wang et al., 2023], propone una solución innovadora para la reconstrucción y segmentación 3D de baches en pavimentos. El enfoque aborda las limitaciones de los métodos tradicionales basados en imágenes 2D al utilizar técnicas de fotogrametría para generar perfiles 3D detallados de los baches. El sistema se compone de un método de reconstrucción conocido como Estructura Desde el Movimiento de Pavimentos (PP-SFM) y una red de segmentación 3D, Trans-3D-Seg, que incorpora módulos de transformadores para mejorar la precisión de la segmentación de nubes de puntos 3D. El sistema muestra una alta precisión, con una puntuación F1 del 92,58 % y una precisión general del 93,44 %. Además, se ha demostrado su robustez en diferentes condiciones, como distintas alturas de adquisición y en luz oscura. Otro sistema muy parecido que cumple con todo lo anterior expuesto y que además incorpora un láser

para conseguir una segmentación más robusta, está definido en [Ahmed et al., 2022].

Las ventajas de estos proyectos son las siguientes:

- *Alta precisión.* Los sistemas proporcionan una segmentación precisa de baches, evidenciada por las altas puntuaciones en la precisión y la puntuación F1.
- *Robustez.* Los métodos son robustos en diversas condiciones, incluyendo iluminación variable y diferentes alturas de adquisición, lo cual es crucial para aplicaciones prácticas en el terreno.
- *Facilidad para la integración.* Ya que los sistemas están preparados para operar en cualquier cámara, facilitan la integración.

Las desventajas de estos proyectos son las siguientes:

- *Alto coste computacional.* Debido a la complejidad de las aplicaciones, necesitan un procesamiento intensivo.
- *Hardware potente.* La implementación del sistema necesita *hardware* potente y recursos computacionales considerables para el procesamiento de imágenes y la ejecución de modelos de aprendizaje profundo.
- *Movimiento circular para detección del volumen.* Una vez detectado el bache, es necesario moverse alrededor de él para que el algoritmo de fotogrametría sea capaz de encontrar su volumen, lo que ralentiza el cálculo.

En relación a la detección de los baches se pueden encontrar métodos que se pueden enmarcar en tres distintas categorías, descritas en [Kim et al., 2022], y que se dividen en: baches detectados por visión, baches detectados por vibración y baches detectados por reconstrucción 3D. Cada categoría tiene una serie de ventajas y desventajas, que es importante conocer a la hora de elegir el tipo de método a usar, y que aparecen descritas en el Cuadro 2.1. A continuación se enumeran alguno de los métodos de cada categoría.

Métodos	Ventajas	Desventajas
Visión	Más rentable que reconstrucción 3D Adeuada para ciertos baches	Limitaciones en profundidad Efecta condiciones metereológicas
Vibración	Más rentable de todos Requiere poco almacenamiento Soporta tiempo real	Limitaciones en formas Depende de la arquitectura
Reconstrucción 3D	Forma del bache más precisa	Más caro de todos

Cuadro 2.1: Ventajas y desventajas de los métodos de detección de baches

En visión se puede encontrar el método de [Park et al., 2021], que usa modelos de YOLOv4, YOLOv4-tiny y YOLOv5, un dataset total de 665 imágenes (70 % entrenamiento, 10 % validación, 20 % pruebas) y aplicado sobre Tesla K80 GPU (12 GB) en Google Colab. Otro método es el de [Wanli Ye and Xiao, 2021], que utiliza un modelo de prepooling CNN con un dataset de 96.000 imágenes (72.000 entrenamiento, 24.000 pruebas), una precisión optimizada hasta de 0,9825. Todo ello probado sobre un Intel Core i7, 32 GB RAM, GeForce GTX 1080 (8 GB).

En vibración se puede encontrar el método de [Du et al., 2020], que usa un filtro *Butterworth*, un modelo gaussiano mejorado y un algoritmo de *k-nearest neighbor*. Se usan unas 118 muestras de bultos, 174 muestras de planos y 103 de bache. Tiene una precisión de 0,96 para baches y 0,94 para bultos. Se usa un Cavalier con un smartphone Redmi Note 8 Pro con frecuencia de muestreo de 400 Hz. Otro método es el de [Allouch et al., 2017] que usa un filtro paso bajo, con transformada de Fourier. Para clasificar se usa un árbol de decisión C4.5 (0,9860), SVM (0,9525) y Naïve Bayes (0,9690). Todo ello se ha probado sobre el smartphone Samsung Galaxy Alpha con frecuencia de muestreo de 50 Hz.

Finalmente, para la reconstrucción 3D se puede encontrar el método de [Dhiman and Klette, 2020], que usa visión estéreo de un solo marco, fusión de múltiples marcos, aprendizaje por transferencia con Mask R-CNN y aprendizaje por transferencia con YOLOv2, todo ello usando una GeForce GTX 1080 y un Tesla K80. Otro método es el de [Ul Haq et al., 2019], que usa un filtro de paso-alto, una ecualización de histograma, una combinación de *waypoints* con triangulación estéreo. Este método ha sido probado en dos webcams A4Tech PKS-732 montadas en un trípode o en la parte trasera de un coche usando un ordenador con sistema operativo Windows.

Los prototipos, métodos y desarrollos tecnológicos presentados demuestran el potencial de la robótica y la IA en el mantenimiento y detección de deterioros en

el pavimento. Desde la creación de modelos 3D de baches hasta la automatización del sellado de grietas, estos avances no solo mejoran la seguridad y eficiencia de las operaciones, sino que también optimizan la gestión y planificación del mantenimiento vial. No obstante, cada solución presenta desafíos específicos, como la dependencia de condiciones climáticas, la limitación geográfica y los costes asociados, que deben ser considerados al evaluar su implementación en diferentes contextos.

En el siguiente capítulo se va a proceder a definir una serie de requisitos, metodología y un plan de trabajo que se ha seguido para dar solución a los objetivos que se plantean en este proyecto.

Capítulo 3

Objetivos

Establecer metas es el primer paso para convertir lo invisible en visible.

Tony Robbins

Tras haber establecido el marco contextual del presente proyecto, se procede a presentar la descripción del problema, los requisitos, las competencias tanto adquiridas como empleadas, la metodología y el plan de trabajo seguido.

3.1. Descripción del problema

La idea de este trabajo fin de grado nace tras los acontecimientos vividos por la *Depresión Aislada en Niveles Altos* (DANA) que afectó a la zona de Toledo y Madrid el pasado septiembre de 2023. Estos hechos dejaron inundaciones, pueblos anegados, carreteras cortadas, muchos vecinos perdieron sus casas, y desgraciadamente se cobró la vida de tres personas.

Tras lo sucedido, las carreteras fueron las infraestructuras que más tardaron en arreglarse. La principal causa de que eso ocurriese es la falta de fondos, lo que conlleva un escaso mantenimiento. Además, los operarios que conforman la mano de obra llevan tiempo exigiendo mejoras de seguridad en su entorno laboral, como defiende la Plataforma de Trabajadores de Conservación de Carreteras³².

La solución propuesta en este trabajo busca ayudar a mejorar esta situación, proporcionando un robot de bajo coste y accesible para cualquier persona y que sirva para poder mejorar el mantenimiento de las carreteras y reducir el riesgo de exposición de los operarios. Por lo tanto, este proyecto pretende, como objetivo principal, crear

³²https://conservacion.es/index.php?option=com_content&view=article&id=1&Itemid=101

un robot que, usando materiales de bajo coste, sea capaz de navegar por las carreteras, detectar los baches que vaya encontrando y sea capaz de estimar el área del bache para hacer una estimación media del volumen que ocupa dicho bache y poder ser tapado. De igual manera, todo quedará registrado en una interfaz web en la que cada bache quedará marcado sobre un mapa con su correspondiente descripción para que los operarios puedan operar cuando estimen oportuno.

Con el fin de alcanzar este objetivo principal, se han establecido los siguientes subobjetivos:

1. Investigar los robots o soluciones actuales que cumplan con las características y objetivos establecidos.
2. Seleccionar los componentes *hardware* de bajo coste necesarios para construir el esqueleto del robot.
3. Analizar las diferentes opciones de diseño que más encajen con la forma del robot.
4. Diseñar las piezas en CAD usando herramientas de *software* libre.
5. Usar material típico de impresión 3D para imprimir las partes del robot, como puede ser ABS o PLA.
6. Desarrollar un modelo del robot para que pueda ser usado en simulación usando herramientas robóticas.
7. Desarrollar software necesario usando herramientas robóticas para poder controlar el robot físico.
8. Realizar algunos experimentos en entorno reales o adaptados.

3.2. Requisitos

Tras nombrar los objetivos y subobjetivos a cumplir en este proyecto, se enumeran los requisitos que se han de satisfacer:

1. El coste total de la fabricación del robot no debe superar los 250€.
2. Todas las piezas diseñadas deben poderse imprimir en cualquiera impresora convencional.

3. Se usará Ubuntu con soporte a largo plazo como sistema operativo, tanto para el ordenador como para el robot.
4. A fin de facilitar la implementación de este proyecto para cualquier tipo de usuario, no será necesario disponer de ninguna tarjeta gráfica de uso dedicado para entrenar los modelos.
5. Los modelos entrenados se deben ajustar a las limitaciones hardware del robot.
6. Se busca que sea un proyecto a largo plazo, por eso se debe realizar la integración con la plataforma ROS 2.

3.3. Competencias

A continuación se detallan las competencias que se han empleado y adquirido para la realización del presente trabajo fin de grado.

3.3.1. Competencias empleadas

Las competencias empleadas para la realización de este proyecto, y que han sido tomadas de las distintas asignaturas del grado, son las siguientes:

1. Evolución y futuro de la robótica: *CE1*. Capacidad para analizar la evolución de la Ingeniería Robótica y ser capaz de identificar sus aplicaciones, oportunidades de emprendimiento y su impacto en el futuro. Esta competencia ha sido empleada para poder desarrollar los Capítulos 1 y 2 de este proyecto.
2. Laboratorio de sistemas: *CE9*. Capacidad de conocer y manejar los sistemas y las herramientas de las que dispone para su gestión y programación. Esta competencia ha sido empleada para poder configurar y ejecutar código del robot en entornos no gráficos.
3. Sensores y actuadores: *CE12*. Capacidad de diseñar robots y sistemas inteligentes atendiendo a los elementos de sensorización y actuación más adecuados dependiendo de la aplicación, los requerimientos del sistema y las condiciones del entorno. Esta competencia ha sido empleada para poder encontrar los componentes *hardware* necesarios para poder llevar a cabo el esqueleto del robot.
4. Arquitectura *software* para robots: *CE15*. Capacidad de diseñar y programar aplicaciones robóticas y sistemas inteligentes en red usando *middlewares*,

mecanismos de comunicación y estándares propios del ámbito de la Ingeniería Robótica. Esta competencia ha sido empleada en el momento de decidir el tipo de arquitectura *software* necesaria a implementar en el robot.

5. Visión artificial: *CE25*. Capacidad de conocer y aplicar métodos de extracción de información a partir de la información percibida por cámaras y sensores 3D al desarrollo de aplicaciones en robots y sistemas inteligentes. Esta competencia ha sido empleada para poder extraer información de la cámara y ser capaz de tratarla.
6. Mecatrónica: *CE32*. Capacidad de diseñar y construir robots móviles. Esta competencia ha sido empleada en el diseño e impresión en 3D de mi robot.
7. Aprendizaje automático: *CE27*. Capacidad de construir sistemas capaces de resolver problemas a partir de información no estructurada proporcionada por ejemplos o por la experiencia. Esta competencia ha sido empleada para poder ser capaz de crear modelos de aprendizaje automático y aplicarlos al robot.

3.3.2. Competencias adquiridas

Las competencias adquiridas con el desarrollo de este trabajo fin de grado, y que aparecen descritas en la guía docente de la propia asignatura, son las siguientes:

1. *CB2*. Que los estudiantes sepan aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro de su área de estudio. Esta competencia se adquiere gracias a la aplicación de las competencias empleadas justificadas anteriormente y que se pueden ver plasmadas en todo el proyecto.
2. *CB4*. Que los estudiantes puedan transmitir información, ideas, problemas y soluciones a un público tanto especializado como no especializado. Esta competencia se adquiere al describir de forma precisa y comprensible todo el proceso complejo implicado en este proyecto dentro del presente documento.
3. *CB5*. Que los estudiantes hayan desarrollado aquellas habilidades de aprendizaje necesarias para emprender estudios posteriores con un alto grado de autonomía. Esta competencia se logra al adquirir el conocimiento suficiente para desarrollar este trabajo de forma totalmente autónoma, contrastando información con distintas fuentes, haciendo pruebas con distintos tipos de *software*, entre otras.

4. *CE28.* Desarrollo de las capacidades adecuadas para realizar un ejercicio original individual (o excepcionalmente colectivo), presentarlo y defenderlo ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas del campo de la Robótica de naturaleza profesional en el que se sintetizan e integren las competencias adquiridas en las enseñanzas. Esta última competencia se cumple con el desarrollo de este proyecto: que abarca desde la elección del tema, el conocer el estado del arte, la implementación tanto *hardware* como *software*, el desarrollo de la presente memoria hasta su defensa ante un tribunal.

3.4. Metodología

Para llevar a cabo este proyecto se ha optado por seguir una metodología que se iniciaba con una profunda investigación sobre el estado del arte para comprobar la viabilidad del desarrollo del presente proyecto. Posteriormente, tras haber elegido el *hardware* necesario para el robot, se decidió usar una metodología experimental que ayudó a decidir el diseño final que tendría el robot. Una vez cumplido esto, se imprimieron y ensamblaron las distintas piezas.

Para dar soporte *software* al robot real se probó y configuró cada sensor y actuador del robot en distintos sistemas operativos y versiones hasta encontrar la combinación que mejor cumpliese todos los objetivos. Una vez conseguido eso, se decidió seguir un ciclo de desarrollo conocido como *Plan Do Check Act* (PDCA) y así poder ir haciendo pequeños avances consistentes hasta llegar a una versión completamente funcional. Para el desarrollo del robot simulado también se decidió usar la metodología PDCA. Esta metodología sigue los siguientes pasos cíclicos descritos en la imagen 3.1.

3.5. Plan de trabajo

El desarrollo del presente trabajo fin de grado se ha dividido en las siguientes etapas:

1. *Investigación del estado del arte.* En esta fase inicial, se realizaron búsquedas en plataformas online como Google Scholar³³, Web of Science del FECYT³⁴ que está basa en Web of Science de Clarivate³⁵, y otras relacionadas con el mantenimiento

³³<https://scholar.google.es/>

³⁴<https://www.webofscience.com/wos/alldb/basic-search>

³⁵<https://clarivate.com/products/scientific-and-academic-research/research-discovery-and-workflow-solutions/webofscience-platform/>



Figura 3.1: Método PDCA

de carreteras con el fin de encontrar soluciones al problema descrito.

2. *Planteamiento hardware del robot.* Una vez conocida la viabilidad del proyecto, se decidió estudiar qué componentes *low-cost* eran necesarios para dar forma al esqueleto del robot.
3. *Diseño de prototipos del robot.* Tras conocer cuál será el esqueleto del robot, usando cartón y pegamento se hicieron una serie de prototipos hasta encontrar la solución final. Posteriormente se usó la herramienta FreeCAD para modelar las distintas piezas.
4. *Impresión 3D y ensamblaje de las piezas.* Una vez el diseño estaba hecho, se decidió imprimirlo usando filamento de tipo PLA azul. Finalmente, se produjo el ensamblaje de todas las piezas.
5. *Desarrollo del robot en simulación.* Tras tener el robot completamente montado, se decidió desarrollar el modelo del robot pero esta vez para que se pudiera trabajar con él en simulación; en este caso, usando Gazebo.

6. *Configuración hardware del robot.* Tras tener al robot listo en simulación, se decidió configurar cada componente hardware del robot en distintos sistemas operativos hasta encontrar el que mejor encajase con la arquitectura del mismo.
7. *Desarrollo software del robot en físico.* Una vez el robot está completamente configurado y listo para operar en un entorno real, se desarrollaron una serie de nodos en ROS2. Estos ayudan al correcto funcionamiento de cada componente hardware siguiendo el propósito buscado.
8. *Experimentos en un entorno real.* En esta etapa final, se realizaron distintos experimentos en el entorno real para demostrar que se cumplía el objetivo principal.

Asimismo, a lo largo de todo el proceso se ha ido elaborando la presente memoria. La dinámica seguida con el tutor ha sido de reuniones semanales o cada dos semanas, dependiendo de la disponibilidad por las dos partes. En dichas reuniones se comentaban todos los avances realizados y se proponían aspectos a mejorar, sugerencias, y se definían nuevos objetivos a conseguir.

Todo el contenido del proyecto está alojado en un repositorio público de GitHub³⁶. Además, todo el trabajo diario está documentado en el apartado Wiki³⁷ de dicho repositorio; dividido en *diario* y en *evolución del proyecto*. El apartado de *diario* trata de contar de forma coloquial qué se ha ido realizando cada día o cada pocos días. Por otro lado, en *evolución del proyecto* se puede encontrar la explicación detallada de ciertos códigos, así como de conceptos teóricos, entre otros detalles.

Tras conocer todos los objetivos, subobjetivos, requisitos, competencias, metodología y plan de trabajo llevado a cabo para la realización de este proyecto, se procede a tratar las plataformas de desarrollo usadas.

³⁶<https://github.com/RoboticsURJC/tfg-jlopez>

³⁷<https://github.com/RoboticsURJC/tfg-jlopez/wiki>

Capítulo 4

Plataforma de desarrollo

*Las herramientas adecuadas en las manos adecuadas
pueden cambiar el mundo*

Steve Jobs

Tras haber establecido los objetivos que se pretenden alcanzar en este proyecto, en este capítulo se van a tratar las distintas plataformas de desarrollo, tanto *hardware* como *software*, que han contribuido a la consecución de dichos objetivos.

4.1. Hardware

En este apartado se van a describir el conjunto de componentes *hardware* que se han adquirido para llevar a cabo este proyecto. Siguiendo la filosofía *low-cost* y *Do It by Yourself* (DIY), se ha primado el conseguir el menor coste de cada componente.

4.1.1. Raspberry Pi 4

La Raspberry Pi es una computadora de bajo coste y, con su tamaño compacto, es ideal para proyectos de electrónica, programación y educación. Esta placa, en su cuarta versión, dispone de un procesador ARM Cortex-A72 de cuatro núcleos a 1,50 GHz fabricado en 28 nm, y con tres configuraciones de memoria. Su alimentación la recibe por el puerto USB-C, tiene dos conectores micro HDMI, conexión Wi-Fi, Bluetooth 5.0, dos USB 2.0 y dos USB 3.0. Permite la compatibilidad con la mayoría de accesorios gracias al conector GPIO de cuarenta pines y el conector *Camera Serial Interface* (CSI). Gracias a esos cuarenta pines, el usuario puede interactuar con una amplia variedad de dispositivos externos, como sensores, LEDs y motores, lo que la hace excelente para proyectos de automatización y robótica.

Sin embargo, la Raspberry Pi usa procesadores ARM, que, aunque eficientes energéticamente, no están optimizados para realizar tareas intensivas de IA, como entrenar modelos de redes neuronales o ejecutar inferencias en tiempo real a gran escala. En la Figura 4.1 se puede ver una imagen de la placa con referencia a las distintas partes de la misma. Esta placa tiene un coste de unos 65€.



Figura 4.1: Raspberry Pi 4³⁸

4.1.2. Raspberry PiCamera

Esta cámara (Figura 4.2) tiene un tamaño de 23,86 x 25 x 9 mm, utiliza el sensor de imagen IMX219PQ de Sony, que ofrece imágenes de vídeo de alta velocidad y alta sensibilidad. Dispone también de funciones de control automático, como el control de exposición, el balance de blancos y la detección de luminancia. Para conectarse, usa un cable plano diseñado específicamente para el puerto CSI de la placa. Su coste aproximado es de 18€.

4.1.3. GPS NEO 6M

Para poder identificar dónde se encuentra cada bache es necesario hacer uso de un posicionamiento global mediante satélites. En este caso se decidió usar el módulo NEO 6M (Figura 4.3), cuyo precio aproximado es de 9€. Este módulo utiliza el chipset u-blox NEO-6M, que proporciona un seguimiento preciso de la posición utilizando satélites

³⁸<https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>

³⁹<https://www.raspberrypi.com/products/camera-module-v2/>



Figura 4.2: Raspberry PiCamera V2³⁹

GPS, permitiendo determinar la latitud, longitud, altitud, y velocidad. Además, incluye una antena cerámica externa de alta ganancia (puede ser interna en algunas versiones) que mejora la recepción de la señal GPS, incluso en áreas con una señal débil.

Tiene una precisión de aproximadamente 2,5 metros en condiciones abiertas. Soporta hasta veintidós satélites simultáneamente, lo que le permite ofrecer posicionamiento confiable y estable. Asimismo, utiliza comunicación por *Universal Asynchronous Receiver-Transmitter* (UART), comúnmente a 9.600 bps, lo que facilita su integración con muchas placas del mercado.



Figura 4.3: Módulo GPS NEO 6M⁴⁰

4.1.4. Sevomotor estándar Parallax

Este tipo de servo (Figura 4.4) tiene un rango de rotación de 0 a 180 grados y se puede controlar digitalmente mediante la técnica de *Pulse Width Modulation* (PWM) con un pulso alto de 0,75–2,25 ms en intervalos de 20 ms. Este tipo de servo es muy común en aplicaciones de animatrónica y robótica. Su precio ronda los 17€.

⁴⁰<https://www.u-blox.com/en/product/neo-6-series>

⁴¹<https://www.parallax.com/product/parallax-standard-servo/>



Figura 4.4: Servomotor Parallax⁴¹

4.1.5. Ruedas

Para implementar las ruedas del prototipo robótico de este proyecto se decidió tomar las ruedas del kit robótico *ActivityBot*, que usa unas ruedas compatibles con los motores descritos anteriormente, además de ser muy ligeras y estables. Se trata de una rueda de plástico con neumático tipo junta tórica (Figura 4.5 izquierda). El perfil estrecho convierte a esta rueda en ideal para aplicaciones que requieren una dirección precisa, y el diámetro de la rueda es de 66 mm. También se pueden adquirir individualmente, con un coste de 4,54€ la unidad.

Para enriquecer el proyecto, se han empleado otro tipo de ruedas cuyo diámetro es igual a las ruedas del *ActivityBot* pero el grosor del neumático es superior y tiene un mayor agarre sobre la superficie. En este caso son ruedas con el neumático azul (Figura 4.5 derecha) pero existen de muchos tipos y con precio similar a las ruedas *ActivityBot*.

4.1.6. Google Coral USB

El Google Coral USB Accelerator (Figura 4.6) es un dispositivo que permite acelerar tareas de IA en dispositivos que no cuentan con *hardware* especializado para ello. Está diseñado para ejecutarse con modelos de aprendizaje automático utilizando la unidad de procesamiento de tensores de Google, lo que mejora significativamente la velocidad de inferencia en aplicaciones de IA. Está optimizado para ejecutar modelos preentrenados en TensorFlow Lite, la versión ligera de TensorFlow diseñado para

⁴²https://es.rs-online.com/web/p/accesorios-para-kits-de-desarrollo/8430897?srsltid=AfmB0orv3a6_tQNdqAoKx_21Mn1m2MAum68oApyvr5mq8ExPTuh_CVNy

⁴³<https://es.aliexpress.com/item/1005005145020093.html>

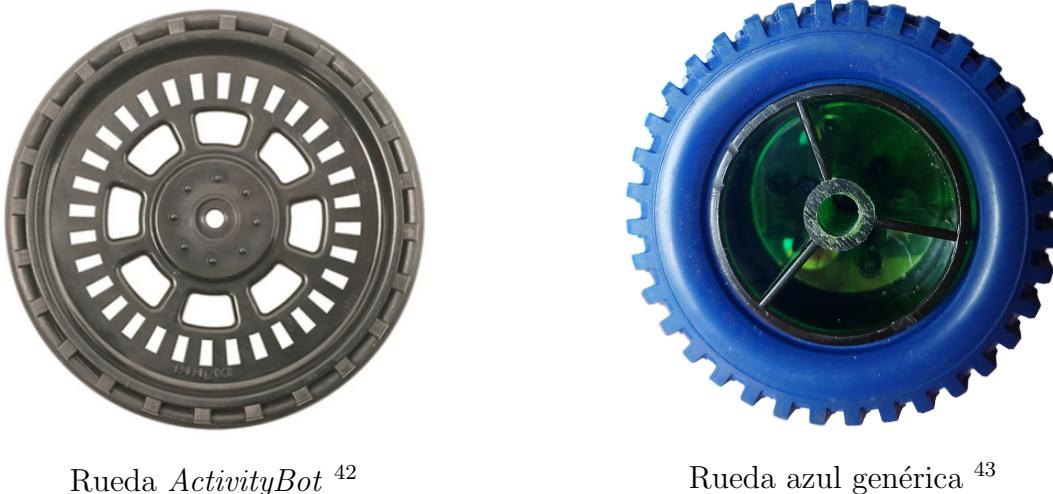
Rueda *ActivityBot*⁴²Rueda azul genérica⁴³

Figura 4.5: Ruedas utilizadas

dispositivos con recursos limitados. Al ser una placa lanzada hace varios años al mercado, sólo es compatible con las versiones de Python 3.6 hasta la 3.9.

Por sus cualidades descritas, se ha decidido este componente para mejorar la detección de baches en tiempo real y solventar las limitaciones de la Raspberry Pi. Su precio es de unos 65€.

Figura 4.6: Google Coral USB⁴⁴

4.1.7. Power bank

Para poder alimentar a todos los componentes se ha incluido en el prototipo robótico una *power bank* de gran capacidad para que permita la autonomía del robot durante un tiempo más que razonable. Concretamente, se ha elegido la Xiaomi Redmi Power Bank de 20000 mAh, cuyas dimensiones son: 73,6 x 27,3 x 154 mm y después de muchos ciclos de carga y descarga, mantienen un buen rendimiento sin perder demasiada capacidad. Su precio ronda los 20€.

⁴⁴<https://coral.ai/products/accelerator/>

⁴⁵<https://ams.buy.mi.com/es/item/3202200053?&skupanel=1>



Figura 4.7: Xiaomi power bank⁴⁵

4.1.8. Rueda loca

Para poder conseguir un movimiento correcto y poder encontrar el punto de apoyo para el robot, es necesario incorporar al robot una rueda loca. Tras investigar qué rueda encajaba mejor, se ha decidido incluir una rueda loca como la que aparece en la Figura 4.8, que tiene las siguientes dimensiones: 5,3 x 2,9 x 2 cm. El precio de esta es de 1,13€.



Figura 4.8: Rueda loca⁴⁶

4.1.9. Ordenador principal

Para poder desarrollar programas, hacer pruebas en simulación, permitir conectarse por SSH a la Raspberry Pi y poder comandar acciones al robot, ha sido necesario tener un ordenador principal que permitiera realizar todas las tareas. El ordenador que aparece en la Figura 4.9 es el que se ha empleado en este proyecto, y cumple las características descritas en el Cuadro 4.1.

⁴⁶https://www.amazon.es/dp/B0BZZCJJT8?ref=cm_sw_r_mwn_dp_N64JV6SGENWN4YPSD849&ref_=cm_sw_r_mwn_dp_N64JV6SGENWN4YPSD849&social_share=cm_sw_r_mwn_dp_N64JV6SGENWN4YPSD849&language=es-ES

⁴⁷<https://www.asus.com/es/laptops/for-home/vivobook/vivobook-14-k413/>

Figura 4.9: ASUS VivoBook 14⁴⁷

Características	Descripción
Pantalla	14 pulgadas Full HD (1.920x1.080) tecnología LED antirreflejo
Procesador (CPU)	Intel Core i7 de 10 ^a generación
Memoria RAM	8 GB
Almacenamiento	512 GB
Tarjeta gráfica (GPU)	Intel UHD Graphics de Comet Lake-U GT2
Sistema Operativo	Windows 10 y Ubuntu 22.04
Puertos	1x USB 3.2 Gen 1 Tipo-A 1x USB 3.2 Gen 1 Tipo-C 2x USB 2.0 1x HDMI lector de tarjetas microSD entrada combo de audio
Conectividad	Wi-Fi 5 (802.11ac), Bluetooth 4.1 / 5.0
Batería	37 Whr
Peso	1,4 kg
Dimensiones	32,5 x 21,6 x 1,99 cm

Cuadro 4.1: Especificaciones técnicas del ordenador usado

4.2. Software

En este apartado se van a describir los programas y librerías que han sido necesarios usar cumplir con los objetivos descritos en el Capítulo 3.

4.2.1. Ubuntu

Ubuntu⁴⁸ (Figura 4.10) es una de las muchas distribuciones del sistema operativo GNU/Linux; esta, concretamente, está basada en Debian GNU/Linux, y es mantenida por Canonical Ltd. Puede utilizarse en ordenadores y servidores. Está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario; es por ello que se ha convertido en uno de los sistemas operativos más populares, cuenta con una amplia comunidad de soporte que le brinda continuamente actualizaciones, que incluyen parches de seguridad y mejoras.



Figura 4.10: Logo de Ubuntu

Entre todas las versiones existentes de Ubuntu, para la realización del proyecto se ha usado Ubuntu 22.04 LTS y Ubuntu 20.04 LTS. El ordenador usado (descrito en la Sección 4.1.9) tiene instalado Ubuntu 22.04 *Long Time Support* (LTS) pero el robot inicialmente usaba Ubuntu 22.04 y finalmente está usando Ubuntu 20.04 LTS Server, una versión de Ubuntu sin interfaz gráfica. La decisión de usar Ubuntu 20.04 es debido a que el dispositivo Google Coral USB (descrito en la Sección 4.1.6) es compatible con versiones de Python entre 3.6 y 3.9, y en Ubuntu 22.04 la versión de Python que se instala por defecto es la 3.10, y el intento de cambiar dicha versión genera numerosos problemas de dependencias, sobre todo con la versión de ROS 2 usada. Asimismo, en numerosos foros aconsejaban migrar a Ubuntu 20.04⁴⁹ que usa Python 3.8. En el Cuadro 4.2 se muestran algunas de las diferencias entre Ubuntu 22.04 y Ubuntu 20.04.

⁴⁸<https://ubuntu.com/>

⁴⁹<https://robotics.stackexchange.com/questions/104413/>

Características	Ubuntu 20.04	Ubuntu 22.04
Fecha de lanzamiento	Abril 2020	Abril 2022
Soporte	Hasta Abril 2025	Hasta Abril 2027
Kernel	Linux 5.4	Linux 5.15
Entorno de escritorio	GNOME 3.36	GNOME 42
Python	Python 3.8	Python 3.10
Soporte gráfico NVIDIA	Soporte estándar	Soporte mejorado con Wayland

Cuadro 4.2: Diferencias entre Ubuntu 20.04 y Ubuntu 22.04

4.2.2. FreeCAD

Esta herramienta de código abierto⁵⁰ (Figura 4.11) es un *software* de diseño asistido por ordenador utilizado principalmente para la creación de modelos 3D en diferentes áreas, como la ingeniería, arquitectura, y diseño de productos. Está diseñado para ser altamente modular, formado por *workbenches*, lo que permite a los usuarios adaptar y extender su funcionalidad según sus necesidades.

FreeCAD se basa en el concepto de modelado paramétrico, lo que permite modificar el diseño fácilmente. Los usuarios pueden retroceder en la historia de un modelo y cambiar parámetros que actualizan automáticamente el diseño. También FreeCAD es multiplataforma, cuenta con una consola de Python integrada y soporta una amplia gama de formatos de archivo como STL y SVG, entre otros. Esta ha sido la herramienta elegida para hacer el diseño 3D de la pieza y la generación del formato STL para su posterior impresión.



Figura 4.11: Logo de FreeCAD

4.2.3. Python

Python⁵¹ (Figura 4.12) es un lenguaje de programación de alto nivel, interpretado y de propósito general, ampliamente reconocido por su simplicidad y legibilidad. Fue

⁵⁰<https://www.freecad.org/>

⁵¹<https://es.python.org/>

creado por Guido van Rossum y lanzado por primera vez en 1991, y ha crecido rápidamente en popularidad debido a su versatilidad y facilidad de uso, tanto para principiantes como para programadores experimentados, lo que le convierte en un lenguaje multiplataforma y multiparadigma.

Python es un lenguaje interpretado, lo que significa que el código se ejecuta línea a línea sin necesidad de ser compilado. Esto facilita el desarrollo rápido y la depuración. Tiene una sintaxis sencilla, legible y es ampliamente utilizado en áreas como: desarrollo web (Django y Flask), ciencia de datos y aprendizaje automático (NumPy, Pandas, TensorFlow, y PyTorch), automatización y scripting, entre otros.

Debido a esa simplicidad y versatilidad, en Raspberry Pi se considera Python como uno de los lenguajes de programación oficiales recomendados, y es por ello que se ha decidido usar este lenguaje para este proyecto.



Figura 4.12: Logo de Python

Software matemático

NumPy (Numerical Python) es una biblioteca fundamental para el cálculo numérico en Python. Está diseñada para facilitar el manejo eficiente de vectores, grandes matrices y arrays multidimensionales, junto con una amplia colección de funciones matemáticas para realizar operaciones sobre estos arrays. Se ha usado en este proyecto para poder calcular el área del bache.

Software para localización

PyNMEA2 es una biblioteca de Python que se utiliza para analizar y generar mensajes en formato NMEA (National Marine Electronics Association) (Figura 4.13), un estándar utilizado en dispositivos GPS y otros sistemas de navegación marina. Es especialmente útil cuando trabajas con módulos GPS en proyectos de Raspberry Pi u

otros dispositivos embebidos, ya que te permite interpretar la información que estos dispositivos envían, como la localización, velocidad y tiempo.

```
julao@raspberrypi:~/Desktop/tfg-jlopez $ cat /dev/ttyAMA0
7,,,,,,5.04,2.05,4.61*0C
$GPGSV,2,1,07,08,55,313,24,10,,,33,16,55,172,28,18,02,056,*42
$GPGSV,2,2,07,23,35,053,31,26,19,162,21,27,73,041,28*49
$GPGLL,4006.17899,N,00402.83247,W,091646.00,A,A*77
$GPRMC,091647.00,A,4006.17923,N,00402.83276,W,1.467,,270624,,,A*6C
$GPVTG,,T,,M,1.467,N,2.717,K,A*24
$GPGGA,091647.00,4006.17923,N,00402.83276,W,1,04,2.05,542.2,M,50.2,M,,*44
$GPGSA,A,3,16,08,23,27,,,,,,5.04,2.05,4.61*0C
$GPGSV,2,1,07,08,55,313,23,10,,,32,16,55,172,28,18,02,056,*44
$GPGSV,2,2,07,23,35,053,31,26,19,162,21,27,73,041,27*46
$GPGLL,4006.17923,N,00402.83276,W,091647.00,A,A*74
```

Figura 4.13: Sentencias NMEA capturadas del GPS

PyNMEA2 te permite interpretar las sentencias NMEA, que son los datos en bruto que los módulos GPS envían, usualmente en forma de texto. Algunas sentencias comunes son:

\$GPGGA: proporciona datos como la latitud, longitud, y altitud.

\$GPRMC: contiene información esencial de ubicación, velocidad y tiempo.

\$GPGLL: latitud y longitud.

En este proyecto, se van a emplear las sentencias que contengan latitud y longitud para estimar la ubicación de cada bache.

4.2.4. OpenCV

OpenCV⁵² (Figura 4.14) es una biblioteca *software* de código abierto diseñada para la visión artificial y el procesamiento de imágenes. Fue desarrollada inicialmente por Intel en 1999 y ahora es mantenida por una gran comunidad de desarrolladores. OpenCV es ampliamente utilizada en aplicaciones que requieren análisis de imágenes, detección de objetos, reconocimiento de rostros, visión computacional, y mucho más. En este proyecto se ha usado concretamente para la detección del bache y el cálculo del área.

⁵²<https://opencv.org/>



Figura 4.14: Logo de OpenCV

4.2.5. ROS 2

ROS, por sus siglas en inglés, *Robot Operating System* se trata de un conjunto de bibliotecas y herramientas que ayudan a los desarrolladores a construir sistemas robóticos. ROS proporciona servicios de nivel bajo como abstracción de *hardware*, control de dispositivos, paso de mensajes entre procesos y gestión de paquetes, así como herramientas para simulación, pruebas y desarrollo de algoritmos.

ROS 2, la segunda generación de ROS, se ha rediseñado para superar las limitaciones de su predecesor. Está basado en un nuevo middleware llamado DDS (Data Distribution Service), que permite una mejor comunicación entre nodos, mayor seguridad, y más opciones de transporte y calidad de servicio (QoS). También mejora la escalabilidad, el soporte para sistemas distribuidos y la capacidad para aplicaciones en tiempo real. Además, ROS 2 es multiplataforma e interoperable; soportando múltiples lenguajes de programación, como C++ y Python.

Según se ha explicado en la Sección 4.2.1, debido a que se ha necesitado para este proyecto usar Ubuntu 20.04 LTS y Ubuntu 22.04 LTS; se ha tenido que usar las dos distribuciones más estable de ROS para cada versión de Ubuntu: ROS 2 Foxy⁵³ y ROS 2 Humble⁵⁴, respectivamente (Figura 4.15).

ROS 2 Control

ROS 2 Control⁵⁵ es un framework dentro de ROS 2 diseñado para facilitar el control de robots en tiempo real. Proporciona una infraestructura modular y escalable para manejar controladores que gestionan los actuadores (motores, servomotores, etc.) de un robot, así como la lectura de sensores. Se utiliza comúnmente en robots móviles, brazos robóticos, drones y otras plataformas robóticas.

Los componentes principales de ROS 2 Control son:

⁵³<https://docs.ros.org/en/foxy/Installation.html>

⁵⁴<https://docs.ros.org/en/humble/index.html>

⁵⁵https://control.ros.org/rolling/doc/getting_started/getting_started.html



Logo ROS 2 Foxy



Logo ROS 2 Humble

Figura 4.15: Distribuciones de ROS 2 usadas

- *Hardware Components/Interfaces.* Realizan comunicación con *hardware* físico y pueden ser: sistemas, sensores y actuadores. Cada *hardware* tendrá una serie de *command interfaces* o de *state interfaces* que permitirán monitorizar o interactuar con cada componente.
- *Resource Manager.* Abstacta y gestiona los *hardware interfaces*, permitiendo la abstracción de los *state interfaces* y los *command interfaces*.
- *Controller Manager.* Gestiona controladores e interfaces de *hardware* en el framework ROS 2 Control.
- *Controllers.* Utilizan la teoría de control para interactuar con el *hardware*. Pueden ser creados desde cero o usar los que viene por defecto de la librería de ROS 2 Control, que suplen las necesidades en la mayoría de las ocasiones.

En este proyecto se ha decidido usar ROS 2 Control para crear un modelo del robot en simulación y poder trabajar con él usando la filosofía de ROS 2 Control.

4.2.6. Gazebo

Gazebo⁵⁶ (Figura 4.16) es un simulador de robots que permite modelar entornos físicos tridimensionales y probar robots en ellos sin necesidad de tener el robot físico y evitar así posibles caídas o golpes. Es ampliamente utilizado en la investigación y desarrollo de robótica, especialmente en combinación con ROS y ROS 2. Además, ofrece

⁵⁶<https://gazebosim.org/home>

una simulación realista en un entorno 3D, y por ello se decidió usar este simulador para el proyecto.



Figura 4.16: Logo de Gazebo

4.2.7. Herramientas de monitorización

Para el desarrollo del proyecto, se emplean diversas herramientas de monitorización que permiten visualizar y controlar los datos y procesos dentro del entorno de ROS 2, y para ello se ha decidido usar las herramientas que se describen a continuación.

Rviz

RViz⁵⁷ (Figura 4.17) es una herramienta de visualización en 3D utilizada en ROS y ROS 2 para representar información del sistema robótico. Permite a los usuarios ver, en tiempo real, datos como la posición de un robot, sensores, cámaras, mapas y otros elementos. Se ha utilizado para monitorear el comportamiento del robot en simulación.



Figura 4.17: Logo de Rviz

RQT Image View

RQT Image View⁵⁸ es una herramienta gráfica dentro del ecosistema de ROS y ROS 2 que permite visualizar imágenes en tiempo real de un flujo de datos de imágenes

⁵⁷<http://wiki.ros.org/rviz>

⁵⁸http://wiki.ros.org/rqt_image_view

publicado por una cámara en un sistema robótico. Se ha usado esta herramienta para monitorizar la cámara del robot físico.

ROS2cli

ROS 2 *Command Line Interface* (CLI)⁵⁹ es una herramienta que permite interactuar con el sistema ROS 2 mediante comandos desde la terminal. Ofrece una forma rápida y sencilla de acceder a las funciones y características de ROS 2 sin necesidad de escribir o ejecutar código complejo. Ha sido la herramienta principal en este proyecto para ejecutar los distintos nodos, comprobar si estaban bien lanzados, si se producía buena comunicación entre ellos, entre otras aplicaciones.

4.2.8. Google Colaboratory

Google Colaboratory⁶⁰ (Figura 4.18) es un servicio en la nube proporcionado por Google que permite escribir y ejecutar código en Python a través de un entorno de Jupyter Notebook. No requiere configuración para su uso y proporciona acceso gratuito a recursos de computación, incluidos GPUs y TPUs. Es especialmente usado para el aprendizaje automático, la ciencia de datos y la educación. Gracias a esta herramienta, se puede cumplir el requisito nº.4 de los indicados en la Sección 3.2.



Figura 4.18: Logo de Google Colab

4.2.9. YOLOv8

YOLOv8⁶¹ (Figura 4.19) desarrollado por la empresa Ultralytics, es una versión avanzada del popular modelo de detección de objetos *You Only Look Once* (YOLO), diseñado para identificar y localizar objetos en imágenes y videos en tiempo real.

⁵⁹<https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools.html>

⁶⁰<https://colab.research.google.com/>

⁶¹<https://docs.ultralytics.com/es>

Esta versión es conocida por su eficiencia y precisión en la detección de objetos. Es compatible con TensorFlow y PyTorch. También ofrece optimización para diferentes plataformas, incluidos como EdgeTPU, para dispositivos con recursos limitados como puede ser Raspberry Pi. Por todo lo anterior, esta herramienta ha sido elegida para el entrenamiento del modelo de detección de baches.



Figura 4.19: Logo de YOLOv8

4.2.10. TensorFlow Lite

TensorFlow es una plataforma de código abierto desarrollada por Google para el aprendizaje automático y el desarrollo de redes neuronales. Su diseño permite a los desarrolladores e investigadores crear, entrenar y desplegar modelos de aprendizaje automático de manera eficiente en una variedad de dispositivos, desde ordenadores de alto rendimiento hasta dispositivos con recursos limitados como puede ser Raspberry Pi. Concretamente, para poder ejecutar modelos en dispositivos como las Raspberry Pi, se usa la versión TensorFlow Lite⁶² (Figura 4.20), ya que está preparada para optimizar los modelos. Es por ello que cualquier modelo que se quiera usar se tiene que convertir a este formato.



Figura 4.20: Logo de TensorFlow Lite

4.2.11. Interfaz Web

Para poder hacer más amigable la interacción humano-robot, se ha decidido plasmar los datos obtenidos a través de una interfaz web, y para ello se ha decidido usar las

⁶²<https://ai.google.dev/edge/litert>

herramientas que se describen a continuación.

ROS2bridge Server

ROS2bridge Server⁶³ es parte de ros2bridge_suite, y ofrece una capa de transporte WebSocket para la comunicación bidireccional entre páginas web y ROS 2. Convierte mensajes JSON en llamadas a ROS 2 y viceversa, permitiendo que las páginas web interactúen con ROS 2.

OpenStreetMaps

OpenStreetMaps⁶⁴ (Figura 4.21) es un proyecto internacional colaborativo desde 2004 que proporciona datos geográficos gratuitos y abiertos, permitiendo la creación de mapas detallados y editables de cualquier parte del mundo por usuarios voluntarios que den crédito a OpenStreetMaps. Esta herramienta es utilizada en una amplia gama de aplicaciones, incluyendo navegación, análisis de datos geográficos, y proyectos de código abierto relacionados con mapas y geolocalización.



Figura 4.21: Logo de Open Street Maps

Tras conocer todas las plataformas *software* y *hardware* empleadas para la realización del presente trabajo fin de grado, es el momento de describir el desarrollo completo llevado a cabo para la construcción *hardware* del robot y su correspondiente soporte *software*, lo que se explicará detalladamente en los siguientes capítulos.

⁶³http://wiki.ros.org/rosbridge_server

⁶⁴<https://www.openstreetmap.org>

Capítulo 5

Diseño y construcción del robot

La perfección se logra no cuando no hay nada más que añadir, sino cuando no hay nada más que quitar

Antoine de Saint-Exupéry

Tras haber expuesto todas las plataformas de desarrollo utilizadas en este proyecto, en este capítulo se describirá el proceso paso a paso, desde la concepción inicial hasta la construcción y ensamblaje, para que el robot sea completamente operativo.

5.1. Geometría del robot

En este apartado se detalla el proceso llevado a cabo para definir la idea y la forma elegida para el robot. La aplicación de este proyecto se encuentra dentro de los robots de campo, y es por ello que estos tipos de robots son mayoritariamente plataformas que trabajan en entornos no estructurados, como se comentó en la Sección 1.2. Por ello, es necesario que la estructura del robot se asemeje a esos tipos de robots y los más comunes son los robots con ruedas.

Sin embargo, los robots de campo son de gran coste y de grandes dimensiones, lo que hacía inviable que entidades con recursos limitados pudieran adquirirlos. Es por ello que se decidió apostar por los robots de bajo coste. Una primera idea hasta conseguir la solución final al proyecto la tomamos del artículo, [Vega and Cañas, 2018], donde los autores nos presentan PiBot (Figura 5.1), una plataforma robótica educativa de 20 x 10 x 8 cm basada en Raspberry Pi 3 y PiCamera, diseñada para facilitar la enseñanza de robótica a estudiantes de secundaria. Ofrece una infraestructura de *software* abierta en Python y comandos de alto nivel para facilitar el aprendizaje. Además, incluye un modelo 3D imprimible y una versión simulada en Gazebo, disponibles públicamente para que estudiantes y escuelas puedan aprender y practicar robótica sin necesidad del

robot físico.



Figura 5.1: PiBot

Para poder continuar con la investigación de ese proyecto, se adquirió una unidad del robot PiBot. Además, se creó una estructura de metal para que la cámara cambiase su disposición, mirase hacia el suelo y estuviese orientada de forma natural para evitar futuros cálculos innecesarios. El diseño en este punto quedó como muestra la Figura 5.2.

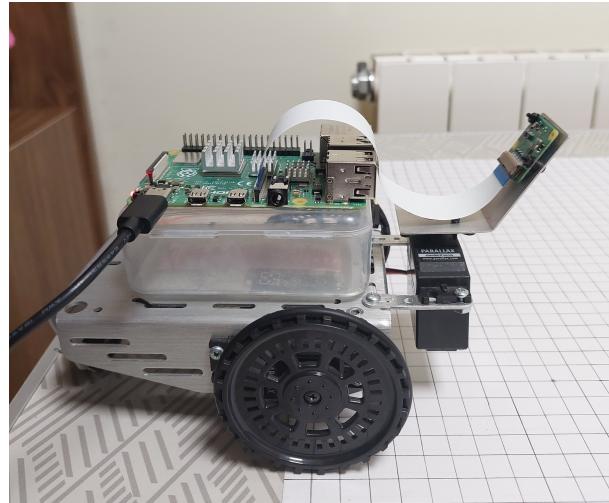


Figura 5.2: PiBot con cámara modificada

De PiBot interesa que tiene dos grados de libertad para el movimiento del robot, ya que cuenta con dos ruedas con motores independientes y una rueda loca, lo que le permite desplazarse a lo largo del eje X e Y y, por lo tanto, puede ir hacia delante y atrás y girar sobre sí mismo. Otro grado de libertad que resulta útil en PiBot para su

aplicación en este proyecto es el giro sobre el eje Z del motor sobre el que está montada la cámara, lo que le permite aumentar el campo de visión (Figura 5.3).

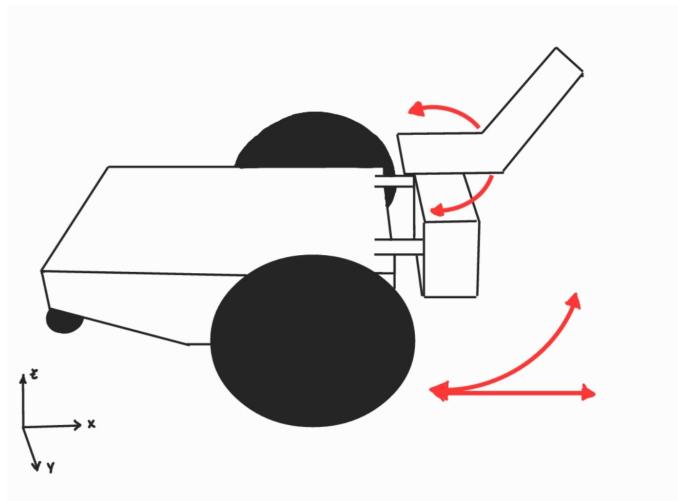


Figura 5.3: Esquema de los grados de libertad de PiBot

Una vez puesto en marcha el PiBot, se realizaron una serie de modificaciones *hardware* para poder cumplir con el objetivo principal descrito en la Sección 3.1; fue necesario añadir una serie de componentes *hardware* a PiBot para poder formar el esqueleto completo del nuevo prototipo robótico, que fue bautizado como PiBotJ. A continuación se describen detalladamente los distintos componentes *hardware*.

5.2. Disposición de los componentes hardware

Una vez conocido las características que eran útiles de PiBot para PiBotJ, se definió la disposición de los componentes *hardware*, descritos en la Sección 4.1, para poder confeccionar el esqueleto completo. La Figura 5.4 muestra todas la conexiones que fueron necesarias usar en la Raspberry Pi para dar soporte a todos los componentes del PiBotJ.

La alimentación a la placa se realiza a través del puerto USB-C. En uno de los puertos USB 3.0 se conectó el Google Coral y, en el puerto CSI, la Raspberry PiCamera. Asimismo, sobre los pines se conectaron los servomotores estándar de Parallax y el módulo GPS. Para el servomotor derecho fue necesario usar el pin 4 para el cable de alimentación, el pin 12 (GPIO 18) para el cable de la señal y el pin 6 para el cable de tierra. Para el servomotor izquierdo fue necesario usar el pin 2 para el cable de alimentación, el pin 7 (GPIO 4) para el cable de la señal y el pin 9 para el cable de tierra. Finalmente, para el módulo GPS había que conectarlo al puerto serie y fue

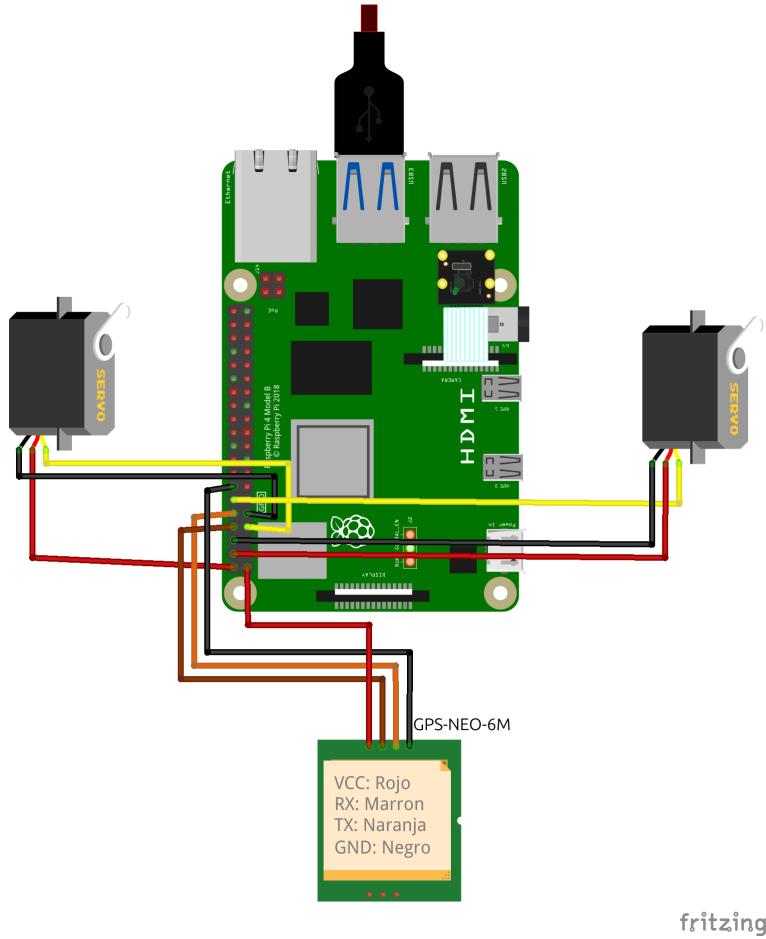


Figura 5.4: Esquema de conexiones del PiBotJ

necesario usar el pin 1 para el cable de la alimentación, el pin 10 (GPIO 15) para el cable de transmisión de señal (TX), el pin 8 (GPIO 14) para el cable de recepción de señal (RX) y el pin 14 para el cable de tierra.

5.3. Bocetos

La realización de bocetos es una etapa fundamental en el proceso de diseño, que se realiza antes de iniciar el modelado en 3D, ya que su propósito es obtener una visión clara de la estructura y disposición de los componentes que conforman el PiBotJ. Una vez que se definió el esqueleto completo que este necesitaba, se creó una serie de bocetos (Figura 5.5) que permitieron afinar los detalles antes de realizar el diseño en 3D.

Antes de realizar el diseño CAD de las piezas se creó una maqueta a tamaño real (Figuras 5.6 y 5.7) para poder tener una idea de cómo sería la aplicación final y así intentar no malgastar material de impresión. Tras tener una idea clara de cómo iba a ser PiBotJ, se comenzó con el diseño de cada una de las piezas.

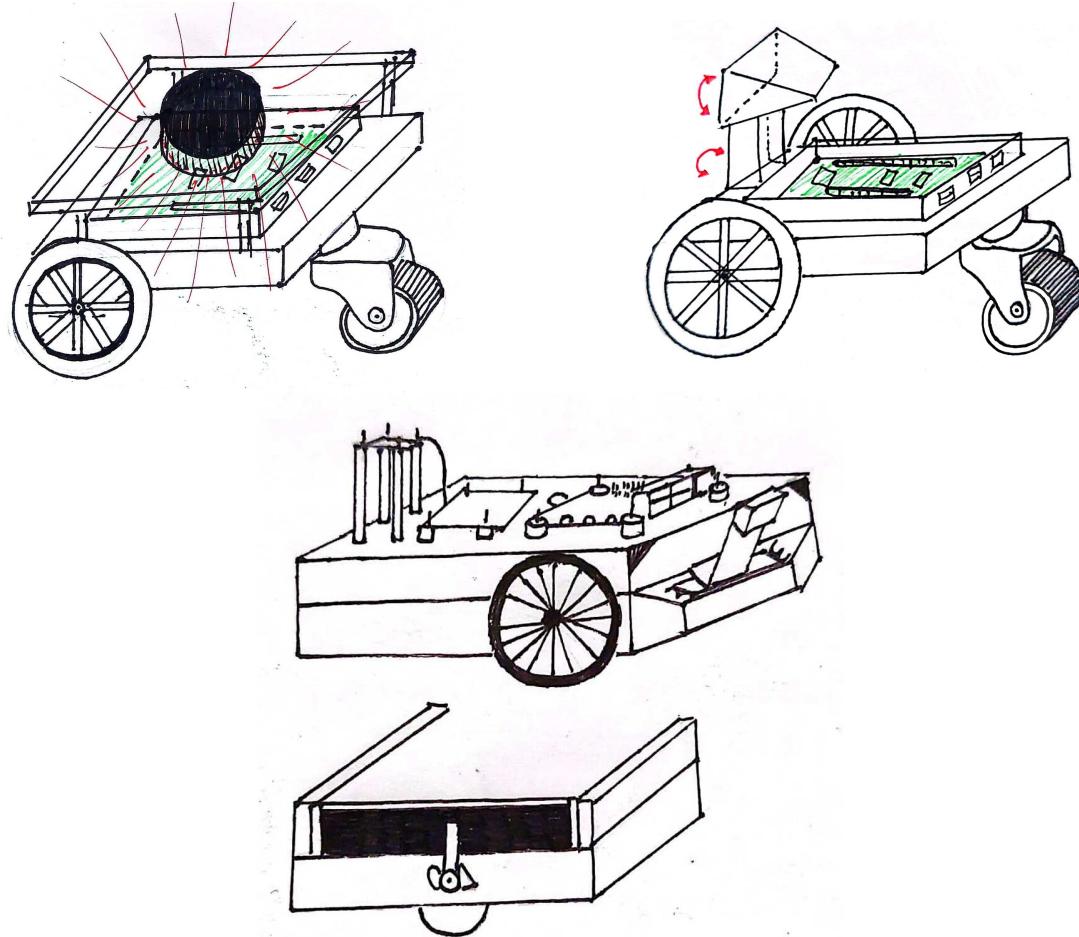


Figura 5.5: Bocetos creados a mano

5.4. Diseño CAD

Para hacer el diseño CAD y la maqueta explicada en el apartado anterior, se crearon unos planos a mano (Figura 5.8) de las diferentes piezas involucradas en el diseño. Las medidas fueron precisas gracias al uso del calibre (Figura 5.9).

Para el diseño de PiBotJ se empleó la herramienta de modelado FreeCAD⁶⁵, con el objetivo de utilizar *software* libre, permitiendo que cualquier persona pueda acceder y modificar las piezas. El diseño se dividió en cuatro partes, cada una de las cuáles tenía una finalidad específica, descritas a continuación.

Para llevar a cabo el diseño de todas las piezas se han seguido los tutoriales de dos cursos de FreeCAD: el del profesor Juan González (también conocido como *Obijuan*)⁶⁶, y el de dcahue-inginería⁶⁷, ambos fundamentales en el desarrollo del proyecto. Además,

⁶⁵<https://www.freecad.org/>

⁶⁶https://www.youtube.com/watch?v=2_DbFzFV9D4

⁶⁷<https://www.youtube.com/watch?v=4zp2DrWv8Wk>

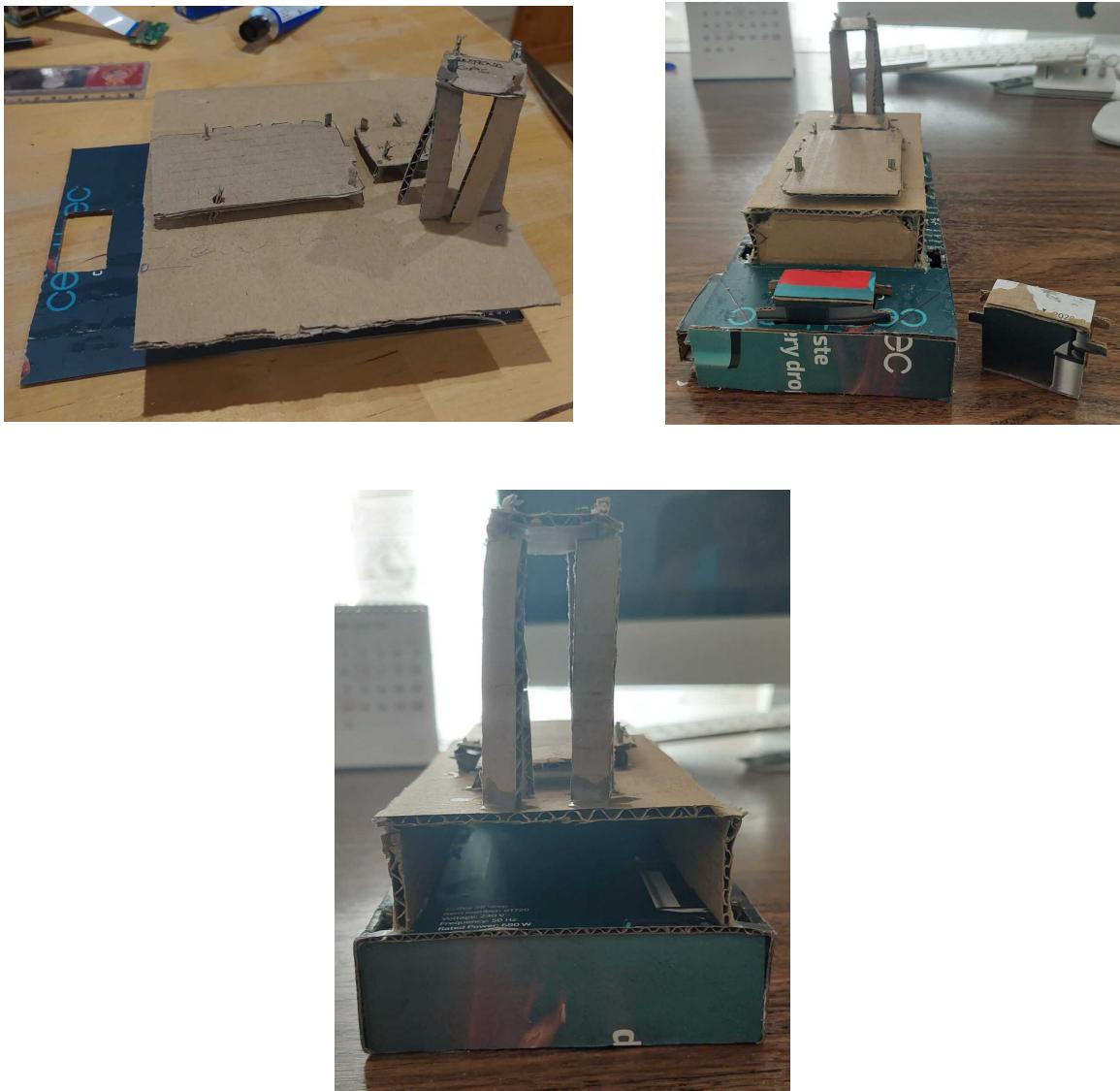


Figura 5.6: Maqueta en proceso

se han utilizado otros tutoriales específicos, como los dedicados a la creación de *shape binders*⁶⁸ y la realización de planos inclinados⁶⁹, esenciales para diseñar la inclinación de la cámara.

5.4.1. Chasis

La estructura principal, que da soporte al robot, fue diseñada para alojar los motores de las ruedas y de la cámara. En la parte trasera se incorporó un prisma rectangular, destinado a la colocación de la rueda loca, mientras que en el lado izquierdo se le dotó

⁶⁸<https://www.youtube.com/watch?v=MCY5IrWrHrU>

⁶⁹<https://www.youtube.com/watch?v=T4hKW1mLrCw>



Figura 5.7: Maqueta final

con un orificio circular para sujetar la *power bank*. En la parte superior se añadieron seis aberturas rectangulares, para permitir el paso ordenado de los cables, manteniendo una estética limpia y organizada desde el exterior.

Además, a este chasis se le dotó de cuatro orificios circulares, que permitieron fijar la carcasa mediante tornillos. Se puede encontrar tanto su versión compatible con FreeCAD⁷⁰, como con el formato de diseño 3D por excelencia, STL⁷¹. La Figura 5.10 muestra el chasis desde distintas perspectivas, tal como será preparada para la

⁷⁰<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/design/base.FCStd>

⁷¹<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/design/base.stl>



Figura 5.8: Planos de los componentes

impresión en una impresora 3D convencional. Por su parte, la Figura 5.11 presenta nuevamente el chasis, pero esta vez equipada con los componentes de *hardware* necesarios.



Figura 5.9: Comprobación mediante calibre

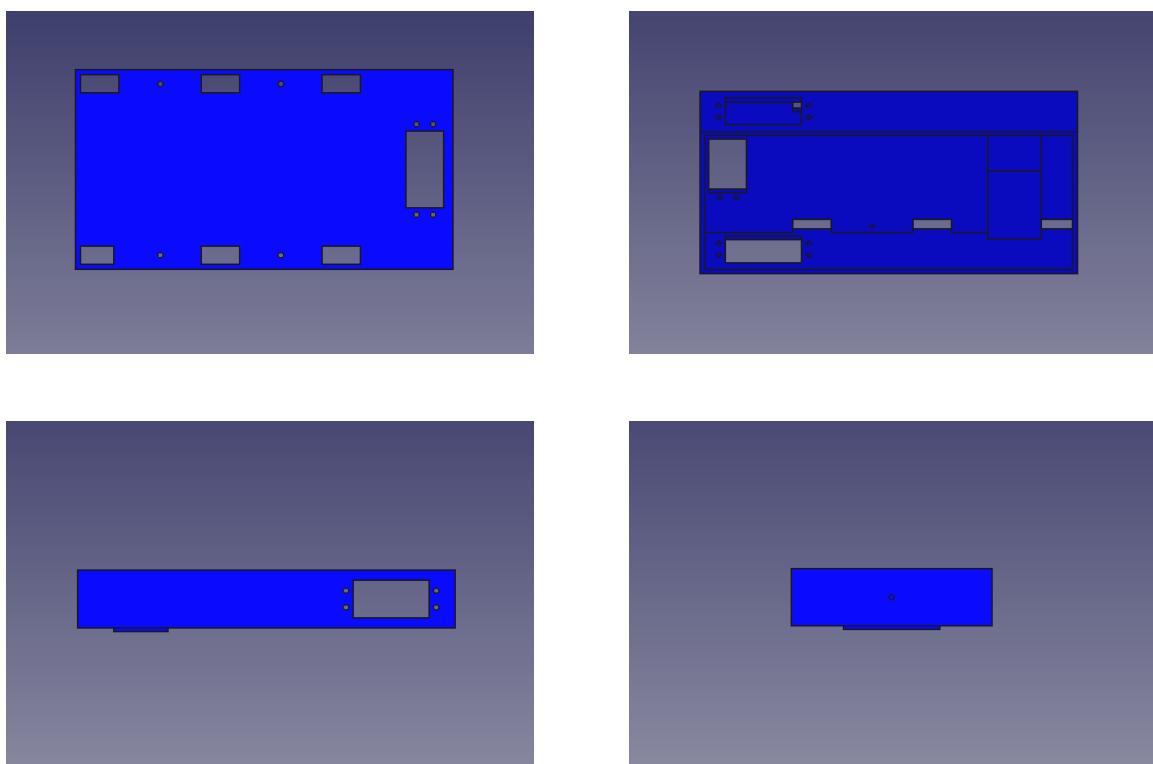


Figura 5.10: Distintas vistas del chasis

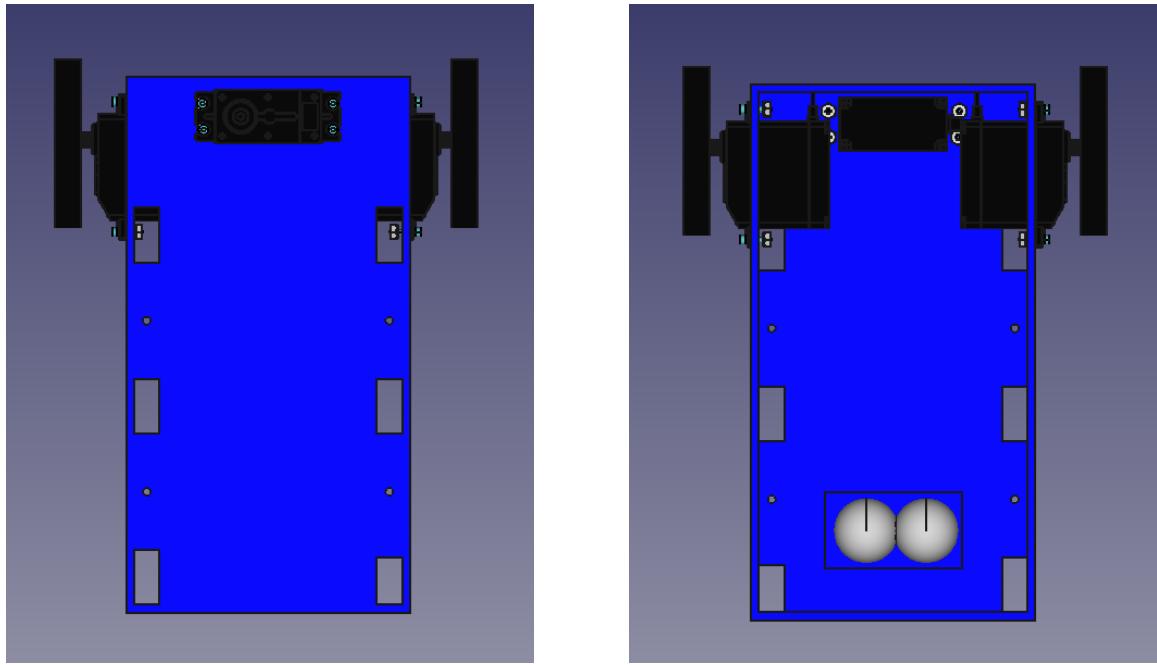


Figura 5.11: Distintas vistas del chasis atornillado

5.4.2. Soporte de la cámara

Para posicionar la cámara de manera que mire hacia el suelo y pueda captar los baches, fue necesario fijarla con una rotación sobre el eje y. En este caso, dicha rotación fue de 50 grados, o 130 grados si se toma como referencia la base de la pieza que va atornillada al motor (Figura 5.12). Esta base fue dotada con dos orificios diagonales que permiten su fijación al motor.

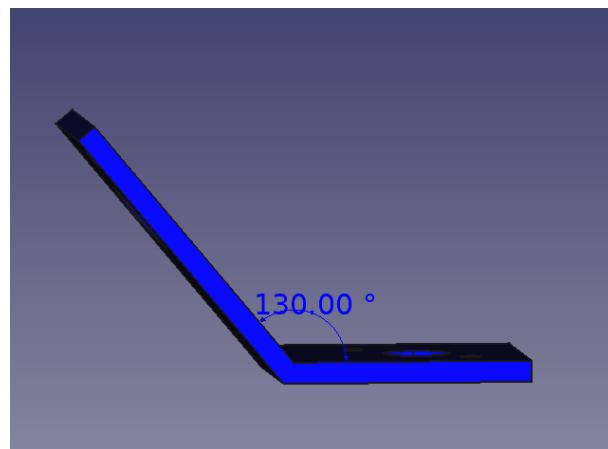


Figura 5.12: Inclinación de la cámara

A la parte inclinada de la pieza se le incluyó un orificio diseñado para alojar el

sensor CMOS, garantizando una correcta alineación y visión. Además, esta sección fue dotada con dos orificios adicionales para asegurar el sensor CMOS y mantenerlo firmemente en su lugar. Se puede encontrar tanto su versión compatible con FreeCAD⁷² como con el formato STL⁷³. La Figura 5.13 muestra el soporte de la cámara desde distintas perspectivas, tal como será preparada para la impresión en una impresora 3D convencional. Por su parte, la Figura 5.14 presenta nuevamente el soporte de la cámara, pero esta vez atornillado sobre los componentes *hardware* necesarios.

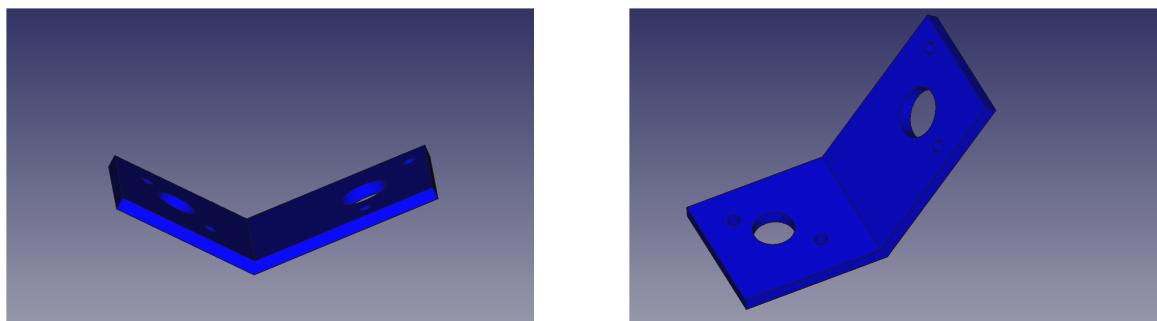


Figura 5.13: Distintas vistas del soporte de la cámara

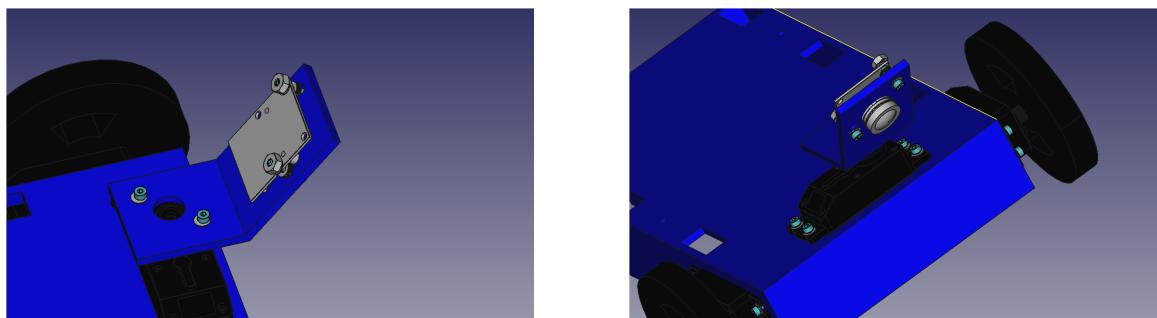


Figura 5.14: Distintas vistas del soporte de la cámara atornillado

5.4.3. Carcasa

Esta carcasa fue diseñada para alojar la placa Raspberry Pi, el módulo GPS y la *power bank* en su interior. La cara superior fue dotada de doce orificios circulares, destinados a atornillar tanto la Raspberry Pi como el módulo GPS, y seis aberturas que se conectaban con la cara inferior, alineándose con las aberturas correspondientes del chasis, para garantizar un paso de cables ordenado.

⁷²<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/design/camara.FCStd>

⁷³<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/design/camara.stl>

La cara inferior, además de las seis aberturas, fue dotada con cuatro orificios adicionales para permitir el atornillado del chasis. En el lateral izquierdo, la pieza se dejó abierta para facilitar la inserción de la *power bank*, mientras que en el lado derecho se añadieron dos cuadrantes que permitieron retirar la *power bank* cuando sea necesario. Esta pieza está disponible tanto en formato compatible con FreeCAD⁷⁴ como en STL⁷⁵.

La Figura 5.15 muestra la carcasa desde varias perspectivas, lista para la impresión en una impresora 3D convencional. La Figura 5.16 presenta nuevamente la carcasa, pero equipada con los componentes *hardware* necesarios.



Figura 5.15: Distintas vistas de la carcasa

5.4.4. Sujeción trasera

Para asegurar que la *power bank* se mantenga en su lugar dentro del robot, se diseñó una pieza que se atornilla en el lado izquierdo del chasis. Esta pieza fue definida como

⁷⁴<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/design/parte-superior.FCStd>

⁷⁵<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/design/parte-superior.stl>

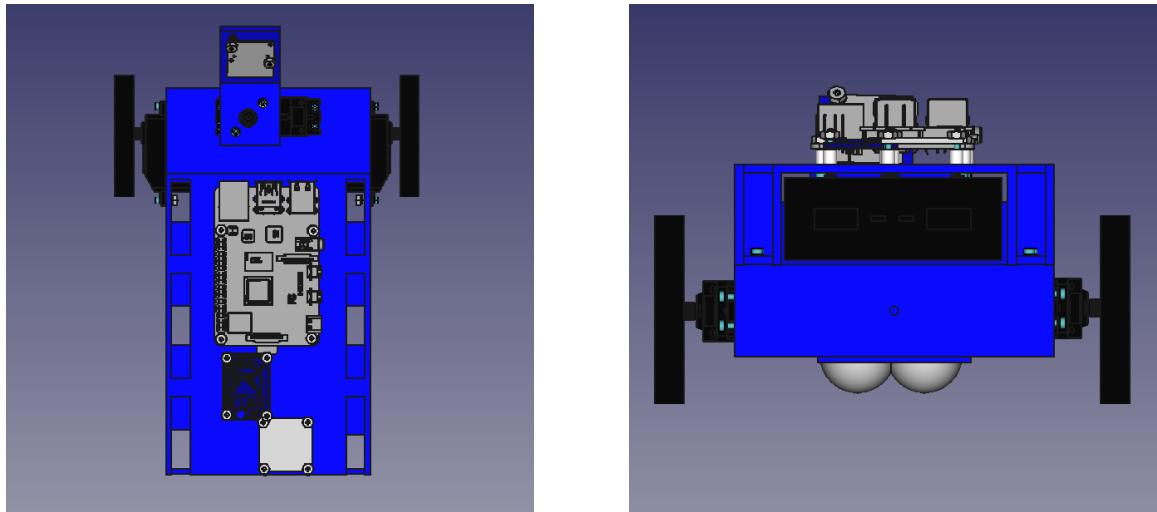


Figura 5.16: Distintas vistas de la carcasa atornillada

un prisma rectangular con más de 30 mm de altura, aproximadamente 5 mm de largo y 3 mm de ancho (Figura 5.17). La pieza debe colocarse en posición vertical para evitar que la *power bank* se deslice, y en posición horizontal cuando se desea extraer esta. Existe una versión en FreeCAD⁷⁶ y otra en formato STL⁷⁷. La Figura 5.18 muestra cómo quedaría montada la pieza sobre el robot.



Figura 5.17: Distintas vistas de la pieza para la sujeción trasera

Una vez detalladas cada una de las piezas CAD diseñadas, se continúa describiendo el proceso de impresión y montaje que se ha seguido para conseguir finalmente el prototipo robótico PiBotJ.

⁷⁶<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/design/sujeccion-trasera.FCStd>

⁷⁷<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/design/sujeccion-trasera.stl>



Figura 5.18: Pieza para la sujeción trasera atornillada

5.5. Impresión y montaje

En esta sección se presentan todos los detalles que deben considerarse para replicar este proyecto mediante impresión 3D.

En nuestro caso, para la impresión de PiBotJ, se usó la impresora FDM Creality Ender3 V2 (Figura 5.19), un rollo de PLA convencional azul y Ultimaker Cura⁷⁸ como *software* de impresión. Para la impresión de todas las piezas se emplearon las mismas características que muestra el Cuadro 5.1. Para este proyecto fue necesario imprimir una pieza de cada una de las explicadas en el apartado de Diseño CAD (Sección 5.4), como muestra la Figura 5.20. La duración de impresión fue en torno a 50 horas.

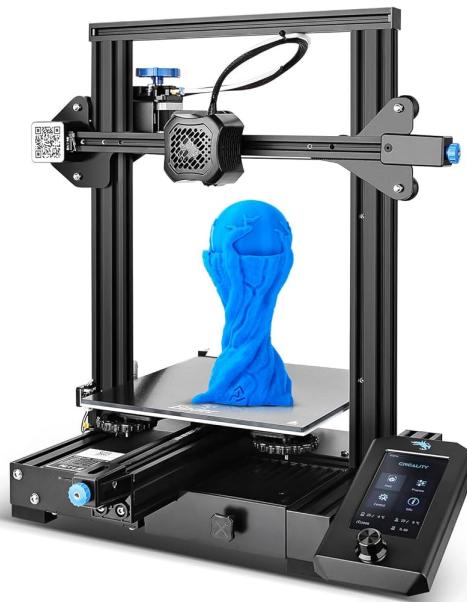


Figura 5.19: Impresora FDM Creality Ender3 V2⁷⁹

⁷⁸<https://ultimaker.com/es/software/ultimaker-cura/>

Características	Parámetros
Calidad	Altura de capa: 0,2 mm Ancho de línea: 0,4 mm
Paredes	Grosor de pared: 0,8 mm Cantidad de líneas de pared: 2 Alineación de costura en Z: <i>Esquina más afilada</i> Preferencia de costura en esquina: <i>Ocultación inteligente</i>
Relleno	Densidad de relleno: 15 % Patrón de relleno: <i>Gyroid</i>
Velocidad	Velocidad de impresión: 50 mm/s Velocidad de la primera capa: 20 mm/s

Cuadro 5.1: Características usadas para la impresión

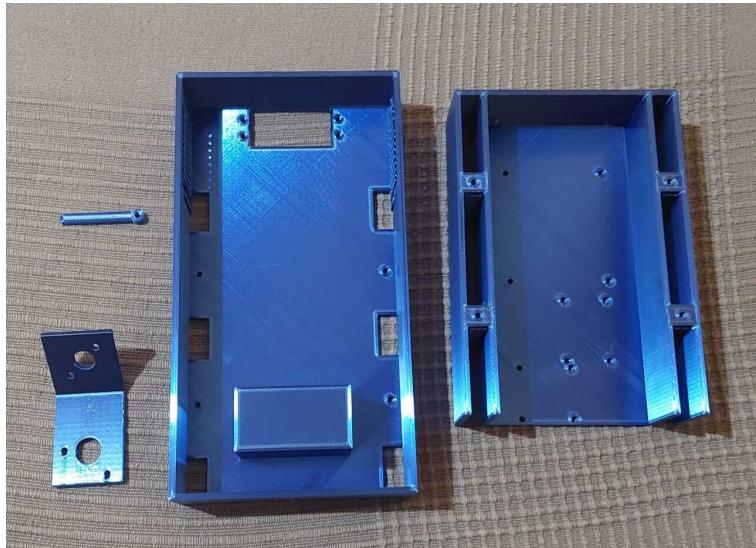


Figura 5.20: Piezas impresas

Una vez impresas todas las piezas y retirados sus soportes generados, fue el momento del montaje, cuya duración fue de dos horas aproximadamente, pero puede ser más dependiendo de las habilidades del usuario.

Uno de los elementos que se tuvo que tener en cuenta para el montaje fueron los Hama Beads (Figura 5.21). Los Hama Beads son cilindros de plástico pequeños con un círculo en el centro usados comúnmente para manualidades y para la creación de objetos de decoración. En este caso, se usaron para evitar que tanto la Raspberry Pi como el módulo GPS toquen directamente la superficie impresa. Además, por su círculo interior pasaban perfectamente los tornillos usados.

⁷⁹<https://www.creality.com/es/products/ender-3-v2-neo-3d-printer>

⁸⁰<https://www.hamabeads.es/>

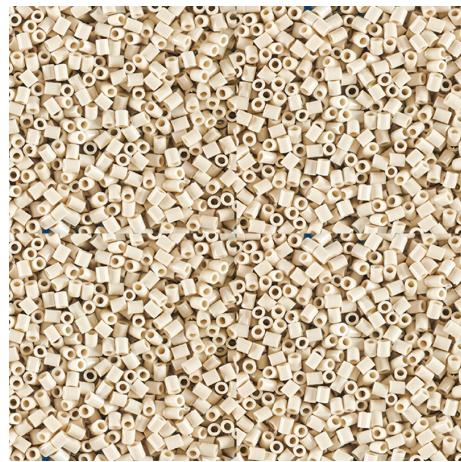


Figura 5.21: Hama Beads⁸⁰

Otro aspecto a tener en cuenta es que, para la placa del módulo GPS, fue necesario soldarle unos pines (Figura 5.22) para posteriormente conectarle los cables sin pérdidas de señal. También fue necesario agrandar con una taladradora los cuatro agujeros que tiene la antena del módulo para que puedan entrar bien los tornillos.



Figura 5.22: Soldando pines al módulo GPS

Una vez solventados los contratiempo anteriores, se puede pasar a la fase de fijación de piezas mediante tornillos. El Cuadro 5.2 muestra toda la tornillería necesaria. Los

tornillos, tuercas y arandelas se pueden obtener en cualquier ferretería. Se recomienda usar un fijador para evitar que se aflojen los tornillos.

Además, las Figuras 5.11, 5.14, 5.16 y 5.18 han sido tomadas gracias a un diseño creado en FreeCAD⁸¹, de PiBotJ completo, que muestra el lugar donde se sitúa la tornillería del robot y ayudará al usuario facilitar el ensamblaje. La Figura 5.23 muestra el robot completo montado. Para conseguir realizar el diseño de PiBotJ completo en FreeCAD, se tomaron los ficheros .stl obtenidos de la Sección 5.4. Además, se tuvo que diseñar aparte: la antena del módulo GPS, la batería, la rueda loca, un Hama Bead y la rueda; todos los ficheros se pueden encontrar en la carpeta `misc`⁸². La placa del módulo GPS necesitó ser editada partiendo de una versión de Thinkercad⁸³ y, por otro lado, el resto de componentes se tomaron de la librería de FreeCAD⁸⁴.

Componente	Tornillos	Tuercas	Arandelas	Hama Beads blancas
Motores	12 M2 10mm	12	24	
Picamera (base)	2 M2 10mm	2	4	
Picamera (cámara)	2 M2 12mm	2	12	
Raspberry Pi	4 M2 12mm	4		4
Placa GPS	4 M2 16mm	4	8	4
Antena GPS	4 M2 16mm	4		4
Sujección entre placas	4 M2 16mm	4	8	
Sujección trasera	1 M2 16mm	1	2	

Cuadro 5.2: Tornillería necesaria

Para sujetar el cable de alimentación de la Raspberry Pi se utilizó una brida. Para optimizar el espacio ocupado por los cables se emplearon gomas pequeñas. Además, fue necesario utilizar diez cables macho-hembra para alimentar los dos motores y el módulo GPS, mientras que el motor de la cámara se quedó sin conectar, ya que la cámara se va a mantener fija, dejando su movilidad como una posible línea futura de trabajo.

Respecto a las ruedas, se han probado dos tipos: las del kit *ActivityBot* y las de goma azul genéricas. Según la aplicación, se puede optar por una u otra. A continuación, se explicará el montaje de cada tipo.

Para utilizar las ruedas del kit *ActivityBot* simplemente fue necesario atornillarlas al robot con el tornillo que viene incluido, ya que estas ruedas fueron diseñadas

⁸¹<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/design/robotcompleto.FCStd>

⁸²<https://github.com/RoboticsURJC/tfg-jlopez/tree/main/design/misc>

⁸³<https://www.tinkercad.com/things/6etb23HSw0w-gy-neo6mv2-neo-6m>

⁸⁴<https://github.com/FreeCAD/FreeCAD-library/tree/master>



Figura 5.23: Robot completo montado en FreeCAD

específicamente para motores Parallax. El aspecto del robot con este tipo de ruedas es el que muestra la Figura 5.24.

Por otro lado, para poder usar la ruedas genéricas, fue necesario realizar algunas modificaciones. En primer lugar, se usó una sierra para cortar las partes sobrantes (Figura 5.25, izquierda). También fue necesario perforar la rueda con un taladro para que el tornillo que conecta al motor encajara fácilmente. Y, tras esto, fue necesario colocar dos topes de motor sobre la superficie lisa de la rueda (Figura 5.25, derecha) y hacer los agujeros correspondientes para atornillarlos. En este caso, se utilizaron siete tornillos M2 de 8 mm. El aspecto del robot con este tipo de ruedas es el que muestra la Figura 5.26.

En el Capítulo de Experimentos se verá qué rueda funciona mejor según el tipo de superficie. Para resumir lo explicado, el Cuadro 5.3 enumera todos los componentes necesarios para construir a PiBotJ, junto con sus respectivos precios.

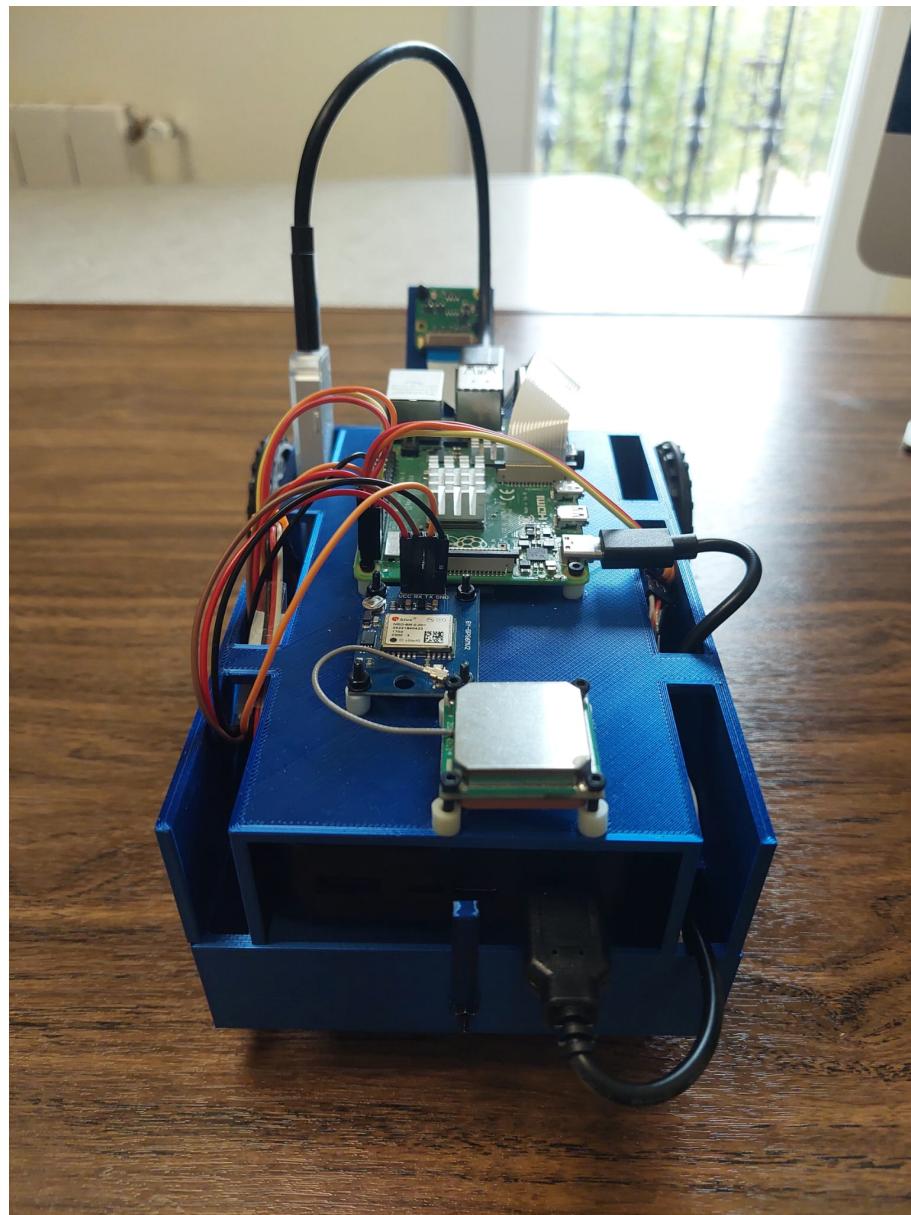


Figura 5.24: PiBotJ con ruedas de ActivityBot

Componente	Precio
Motores Parallax	34€
Picamera	18€
Raspberry Pi	65€
Módulo GPS	9€
Ruedas ActivityBot/Azules	9€
Google Coral USB	65€
Powerbank	20€
Rueda Loca	1,13€
Tornillos, tuercas, arandelas y Hama Beads	3€
Cables, gomas y brida	3€
Rollo de PLA gastado	10€

Cuadro 5.3: Coste proyecto

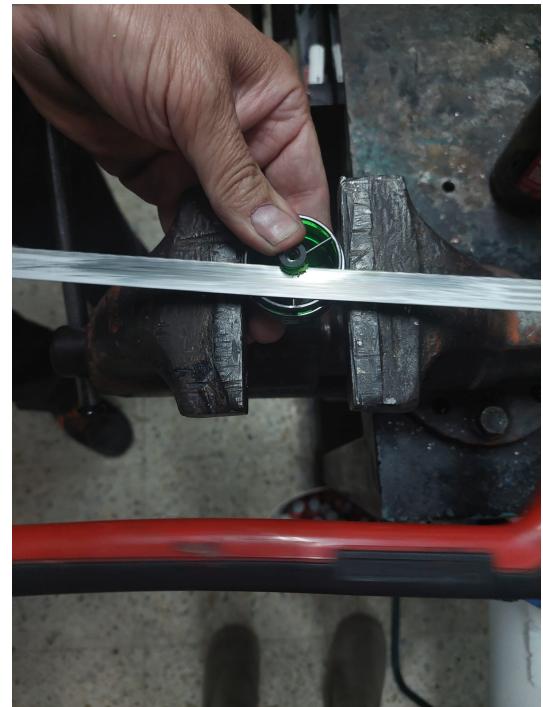
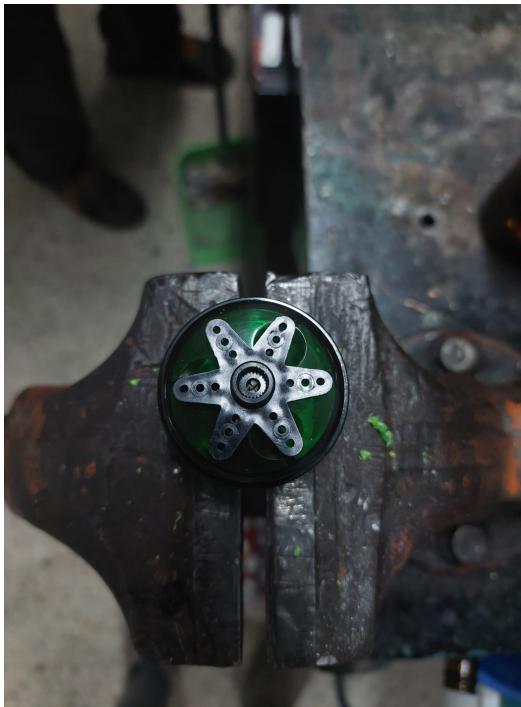


Figura 5.25: Ensamblaje ruedas azules genéricas

El coste total del proyecto es de 237,13€, por lo que está por debajo del límite establecido de 250€, cumpliendo de este modo con el objetivo establecido en la Sección 3.1.

Llegados a este punto, se ha completado el diseño y construcción del prototipo robótico PiBotJ. A continuación, se detallará el soporte *software* que se ha dado a este robot para alcanzar el objetivo del proyecto.

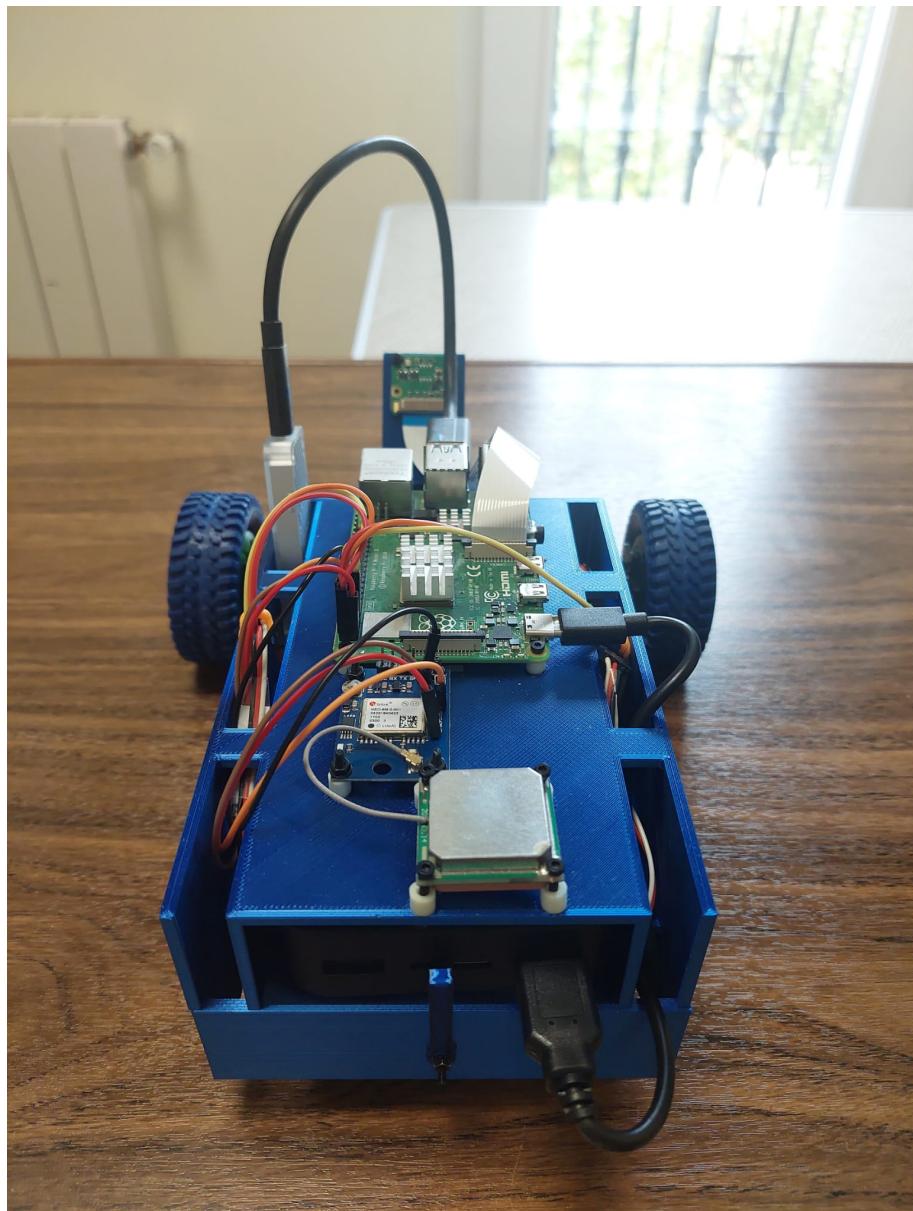


Figura 5.26: PiBotJ con ruedas azules

Capítulo 6

Soporte software del robot

El software es una gran combinación entre arte e ingeniería

Bill Gates

Una vez contado todo el proceso llevado a cabo para el diseño y construcción del prototipo robótico descrito, en este capítulo se explica la implementación *software* desarrollado para dar soporte a dicho prototipo.

6.1. Simulación

En esta sección se describe el proceso seguido para conseguir poner a PiBotJ en funcionamiento a través de simulación; en concreto, a través del simulador Gazebo. Esta parte ha sido desarrollada en el ordenador principal, explicado en la Sección 4.1.9, apoyándose en el sistema ROS 2 Humble⁸⁵.

6.1.1. URDF/Xacro

Primero de todo fue necesario definir las estructuras y propiedades del robot. Para ello, se decidió usar el formato *Unified Robot Description Format* (URDF)⁸⁶ y *Xacro*⁸⁷, muy comunes en aplicaciones robóticas. URDF usa un formato de ficheros XML y describe al robot como un conjunto de *links* (enlaces), que están conectadas por una serie de *joints* (articulaciones); mientras que *Xacro* usa también un formato de ficheros XML que permite crear URDF de manera más modular, reutilizable y eficiente

⁸⁵<https://docs.ros.org/en/humble/Installation.html>

⁸⁶<http://wiki.ros.org/urdf>

⁸⁷<http://wiki.ros.org/xacro>

mediante el uso de macros y propiedades⁸⁸.

En este proyecto se decidió crear una serie de ficheros *Xacro*, cada uno dedicado a las distintas partes del robot (*camera.xacro*⁸⁹, *gps.xacro*⁹⁰ y *robot_core.xacro*⁹¹). Cada fichero *Xacro* necesita siempre tener la mismas etiquetas de *<joint>* y *<link>*, empleadas según convenga. Es importante usar herramientas como validadores de XML⁹² para evitar problemas.

Existen cuatro tipos de *joint*: *prismatic*, *continuous*, *revolute* y *fixed*. En el Código 6.1 se puede ver la definición de una *fixed joint*; si se quiere usar otro tipo de *joint*, hay que añadir algunos campos⁹³.

```
<joint name="gps_joint" type="fixed">
  <parent link="chassis"/>
  <child link="gps_frame"/>
  <origin xyz="-0.1 0.0 0.04" rpy="0 0 0"/>
</joint>
```

Código 6.1: Macro que define una *fixed joint*

En relación a los *link*, obligatoriamente tienen que tener las macros de *<visual>*, *<collision>* e *<inertial>*, si no pueden surgir errores⁹⁴. Existen cuatro tipos de geometría: *box*, *cylinder*, *sphere* y *mesh*. El Código 6.2 muestra un ejemplo de una *mesh link*⁹⁵.

Una vez definidas las distintas partes del robot, se describen las interacciones del robot con el simulador; para ello se usa la macro *<gazebo>*. En el presente proyecto, se ha simulado el sensor cámara y el módulo GPS. Existen muchos tipos de interacciones; en el Código 6.3 se muestra; a modo de ejemplo, cómo simular el sensor cámara. Por su parte, para simular el módulo GPS se ha utilizado un mensaje del tipo NavSatFix⁹⁶. Se pueden encontrar más ejemplos en la web oficial⁹⁷.

⁸⁸<https://articulatedrobotics.xyz/tutorials/ready-for-ros/urdf/>

⁸⁹https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ROS2/src/pibotj_r2c/description/camera.xacro

⁹⁰https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ROS2/src/pibotj_r2c/description/gps.xacro

⁹¹https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ROS2/src/pibotj_r2c/description/robot_core.xacro

⁹²<https://www.xmlvalidation.com/index.php?id=1&L=0#xml-9-6--1732781305>

⁹³<https://articulatedrobotics.xyz/tutorials/ready-for-ros/urdf/#joint-tags>

⁹⁴<https://answers.gazebosim.org/question/25166/problem-changing-joint-from-fixed-to-revolute/>

⁹⁵<https://articulatedrobotics.xyz/tutorials/ready-for-ros/urdf/#link-tags>

⁹⁶http://docs.ros.org/en/api/sensor_msgs/html/msg/NavSatFix.html

⁹⁷<http://wiki.ros.org/urdf/XML/Gazebo>

```

<link name="camera_link">
  <visual>
    <origin xyz="0.02 0.01 0.0 " rpy="0 0 ${-pi/2}"/>
    <geometry>
      <mesh filename="package://pibotj_r2c/meshes/camara.stl"
            scale="0.001 0.001 0.001"/>
    </geometry>
    <material name="Blue">
      <color rgba="#{0/255} #{0/255} #{255/255} 1.0"/>
    </material>
  </visual>
  <collision>
    <origin xyz="0.0 0.0 0.0 " rpy="0 0 ${-pi/2}"/>
    <geometry>
      <mesh filename="package://pibotj_r2c/meshes/camara.stl"
            scale="0.001 0.001 0.001"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0.0 0.0 0.0" rpy="0 0 ${-pi/2}"/>
    <mass value="0.03"/> <!--30g-->
    <inertia ixx="0.01" ixy="0.0" ixz="0.0" iyy="0.005" iyz="0.0"
              izz="0.005"/>
  </inertial>
</link>

```

Código 6.2: Macro que define una *mesh link*

Asimismo, se han definido los sistemas de coordenadas que aparecen en la Figura 6.1. Una vez se definió el robot y los sensores necesarios para que Gazebo interactuara con el modelo, se hizo uso de ROS 2 Control.

6.1.2. ROS 2 Control

Como se explicó en la Sección 4.2.5, ROS 2 Control es un *framework* que permite gestionar y controlar los sensores y actuadores de manera eficiente, y es por ello que se decidió aplicar a este proyecto a través del fichero `ros2_control.xacro`⁹⁸.

Para este proyecto se definió un sistema llamado *GazeboSystem*, encargado del *hardware interface*; de esta forma, es más sencillo añadir el número de sensores y actuadores que se necesiten. En muchos casos, se decidió controlar a *left_wheel_joint*, *right_wheel_joint* y a *camera_joint*, que son aquellos *joints* que tenían asignados un motor en la vida real.

⁹⁸https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ROS2/src/pibotj_r2c/description/ROS2_control.xacro

```

<gazebo reference="camera_link_optical">
  <material>Gazebo/Blue</material>
    <sensor name="camera" type="camera">
      <pose> 0 0 0 0 0 0 </pose>
      <visualize>true</visualize>
      <update_rate>10</update_rate>
      <camera>
        <horizontal_fov>1.089</horizontal_fov>
        <image>
          <format>R8G8B8</format>
          <width>640</width>
          <height>480</height>
        </image>
        <clip>
          <near>0.05</near>
          <far>8.0</far>
        </clip>
      </camera>
      <plugin name="camera_controller">
        <filename>libgazebo_ros_camera.so</filename>
        <frame_name>camera_link_optical</frame_name>
      </plugin>
    </sensor>
  </gazebo>

```

Código 6.3: Macro que permite a Gazebo simular una cámara



Figura 6.1: Sistemas de Coordenadas de PiBotJ

Al ejecutar `ros2 control list_hardware_interfaces` se puede ver las interfaces asociadas de lectura/escritura y de monitorización que tiene cada *joint* a controlar. En la Figura 6.2 se pueden ver las interfaces que tiene PiBotJ y, por ende, cómo se va a controlar cada interfaz.

```
juloau@juloau-VivoBook:~/Desktop/TFG/tfg-jlopez/code/ros2$ ros2 control list_hardware_interfaces
command interfaces
    camera_joint/position [available] [claimed]
    left_wheel_joint/velocity [available] [claimed]
    right_wheel_joint/velocity [available] [claimed]
state interfaces
    camera_joint/position
    left_wheel_joint/position
    left_wheel_joint/velocity
    right_wheel_joint/position
    right_wheel_joint/velocity
```

Figura 6.2: Interfaces que tiene definidas PiBotJ

Tras definir los *joints* a controlar, era necesario definir en un fichero YAML el tipo de controladores que utilizaba el *controller manager*. Dentro de cada controlador había que asignar los *joints* definidos previamente al controlador que necesite cada una. En este caso, se asignaron las ruedas a un control diferencial y, la cámara, se decidió controlar por posición, usando el *topic \pos_cont*⁹⁹.

Una vez el robot estaba completamente definido en los diferentes ficheros *Xacro*, era necesario unirlos todos en otro fichero llamado `robot.urdf.xacro`¹⁰⁰ para poder facilitar la publicación del estado del robot, y eso se consigue usando `robot_state_publisher`.

6.1.3. Robot State Publisher

En la Web oficial de ROS¹⁰¹ se explica que `robot_state_publisher` es esencial para publicar las transformaciones (tf) entre los diferentes *links* del robot y para proporcionar información sobre el estado de sus *joints* a cualquier componente en el sistema. Aunque pueda parecer complicado, la Figura 6.3, obtenida de la web de Articulated Robotics¹⁰², resume muy bien los pasos que sigue `robot_state_publisher`.

Una vez todos los elementos a utilizar estaban listos, era el momento de aglutinarlos todos y crear un launcher que los inicializara y los pusiera en ejecución.

⁹⁹<https://control.ros.org/humble/index.html>

¹⁰⁰https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ros2/src/pibotj_r2c/description/robot.urdf.xacro

¹⁰¹http://wiki.ros.org/robot_state_publisher

¹⁰²<https://articulatedrobotics.xyz/tutorials/mobile-robot/concept-design/concept-urdf#quick-recap>

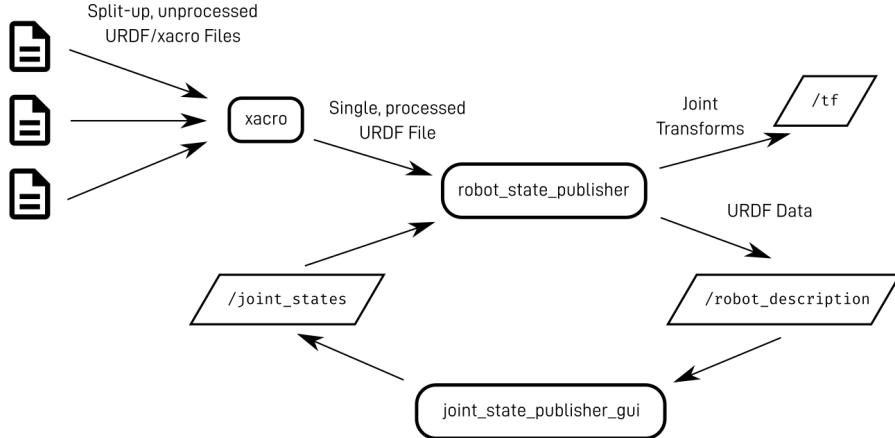


Figura 6.3: Diagrama de robot_state_publisher

6.1.4. Launcher

El *launcher* que se decidió crear para este proyecto parte del creado por Johnnewans¹⁰³, creador de Articulated Robotics, y fue modificado según las necesidades, siendo finalmente el resultante `launch_sim.launch.py`¹⁰⁴. Para facilitar su entendimiento, la Figura 6.4 explica los componentes que forman parte del *launcher*. Con esto, ya podemos ejecutar al completo el PiBotJ.

6.1.5. Ejecución en simulador

Para mostrar todos los *topics* que forman parte del robot, lanza el siguiente comando: `ros2 topic list`, y su salida se puede ver en la Figura 6.5.

De todos esos *topics*, para conseguir el objetivo propuesto en la Sección 3.1, es necesario utilizar únicamente: `/camera/image_raw`, para ver la imagen de la cámara; `/cmd_vel`, para mover las ruedas; `/gps/data`, para ver los valores de posición del GPS; y `/pos_cont/commands`, para mover el motor de la cámara.

Para mover las ruedas hay muchas formas de hacerlo, pero en este caso se usó `rqt_robot_steering`, como muestra la Figura 6.6. Si se quiere visualizar la cámara, se puede usar `rviz2` o `ros2 run rqt_image_view rqt_image_view`, que fue el comando usado para probar su funcionamiento (Figura 6.7). Por otro lado, los datos vertidos por el módulo GPS se pueden visualizar usando `ros2 topic echo /gps/data`, como muestra la Figura 6.8 y finalmente, para mover el motor de la cámara por posición, es

¹⁰³https://github.com/joshnewans/articubot_one/blob/main/launch/launch_sim.launch.py

¹⁰⁴https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ros2/src/pibotj_r2c/launch/launch_sim.launch.py

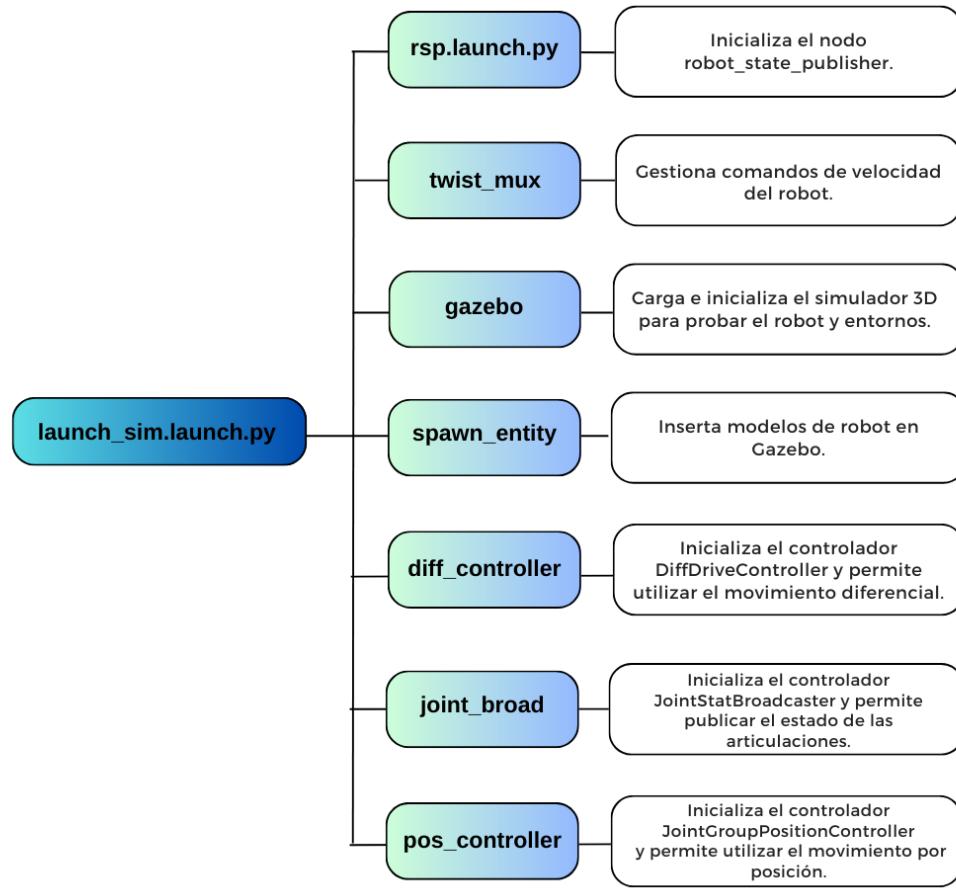


Figura 6.4: Esquema de launch_sim.launch.py

```
julioau@julioau-VivoBook:~/Desktop/TFG/tfg-jlopez/code/ros2$ ros2 topic list
/camera/camera_info
/camera/image_raw
/clock
/cmd_vel
/diagnostics
/diff_cont/cmd_vel_unstamped
/diff_cont/odom
/diff_cont/transition_event
/dynamic_joint_states
/gps/data
/gps/gps_controller/vel
/joint_broad/transition_event
/joint_states
/parameter_events
/performance_metrics
/pos_cont/commands
/pos_cont/transition_event
/robot_description
/rosout
/tf
/tf_static
```

Figura 6.5: Topics disponibles al lanzar el robot

necesario usar el comando siguiente:

```
ros2 topic pub /pos_cont/commands std_msgs/msg/Float64MultiArray \
```

```
"data: [-0.5]"
```

Siendo los valores que van dentro de *data* desde 3 (giro hacia la izquierda) hasta -3 (giro hacia la derecha).

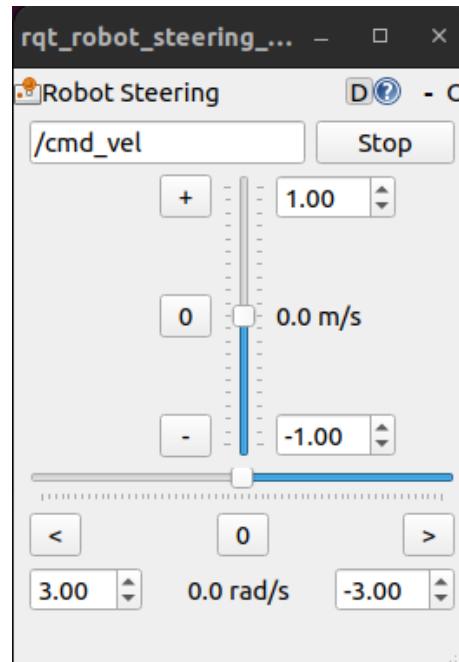


Figura 6.6: Herramienta usada para mover las ruedas

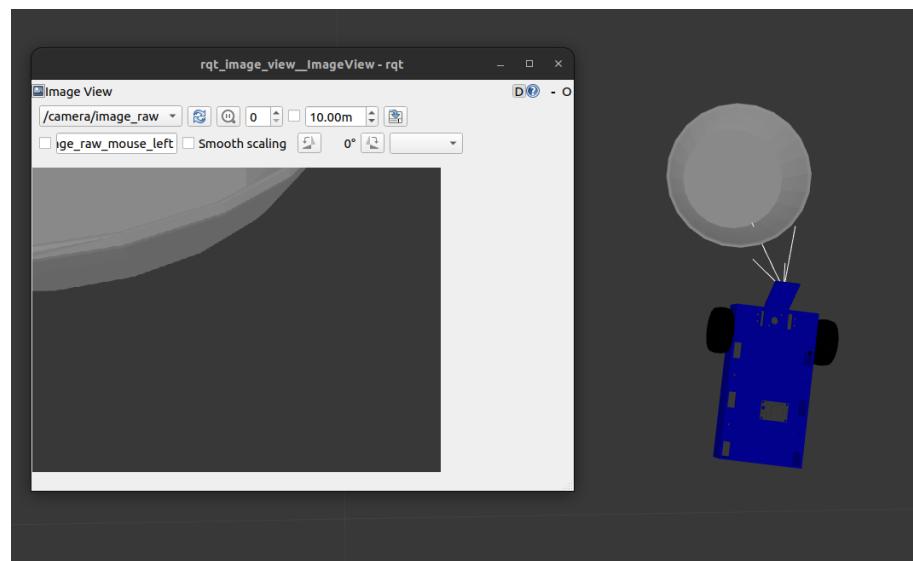


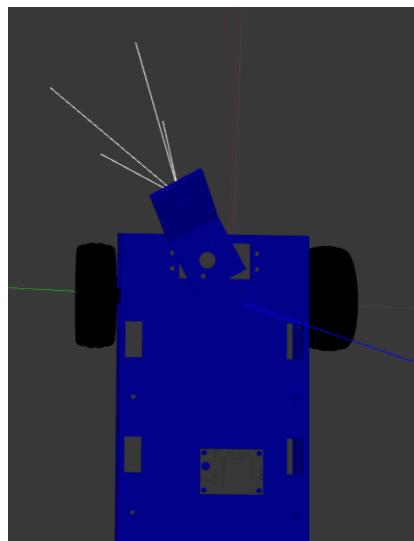
Figura 6.7: Herramienta usada para visualizar la cámara

El resultado final, con el robot de PiBotJ funcionando en simulación, se puede apreciar en la secuencia mostrada en la Figura 6.9, pudiéndose ver la ejecución completa

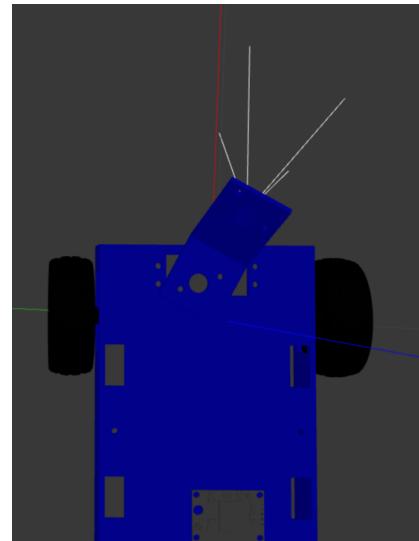
```
juloau@juloau-VivoBook:~/Desktop/TFG/tfg-jlopez/code/ros2$ ros2 topic echo /gps/data
header:
  stamp:
    sec: 2757
    nanosec: 444000000
  frame_id: gps_frame
status:
  status: 0
  service: 1
latitude: -5.2929458126972635e-06
longitude: 1.0408556761408664e-05
altitude: 0.04928559251129627
position_covariance:
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
position_covariance_type: 2
---
```

Figura 6.8: Herramienta usada para visualizar los valores del GPS

en el vídeo¹⁰⁵.



Rotación hacia la izquierda



Rotación hacia la derecha

Figura 6.9: Herramienta usada para rotar la cámara

6.2. Ejecución del entorno real

Una vez definido el robot a través de sus topics, ya se puede lanzar su ejecución (usando la RPi 4 como núcleo computacional) para realizar su tarea.

El sistema operativo instalado en la placa controladora fue Ubuntu 20.04 Server LTS y la distribución Foxy de ROS2, como se explicó en la Sección 4.2.1 y 4.2.5

¹⁰⁵<https://www.youtube.com/watch?v=A0yi7Y1Lpq0>

respectivamente. Se hizo uso de SSH a efectos de usar el PC principal como cliente. Para ello, se instaló el paquete de openssh-client.

Es importante recordar que, al usar ROS 2, todos los nodos que estén ejecutándose dentro de una red Wifi, serán visibles para cualquier dispositivo que esté conectado dentro de esa red Wifi. Gracias a eso, se facilitó la depuración y monitorización de cada sensor y actuador de PiBotJ.

En los siguientes apartados se detalla cómo se ha configurado cada dispositivo que conforma el PiBotJ.

6.2.1. Cámara

Partiendo del `cameraPublisher.py` de este tutorial¹⁰⁶, se pudo crear un nodo llamado `camera_node`¹⁰⁷ que permitía leer y publicar la imagen del dispositivo de la cámara. También se puede monitorizar usando las mismas herramientas que en la Sección 6.1.5 para controlar la cámara en simulación (Figura 6.10).

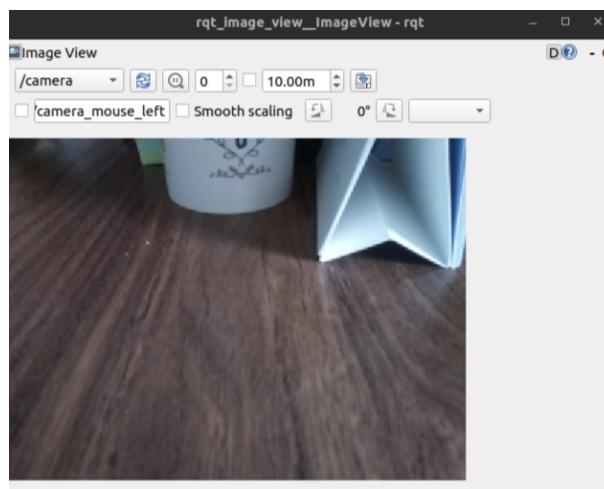


Figura 6.10: Cámara de PiBotJ en funcionamiento

Para conformar un modelo teórico de la cámara se necesitan conocer sus parámetros intrínsecos.

Parámetros intrínsecos

Los parámetros intrínsecos definen la geometría interna, la óptica y extrínsecos de la cámara. Estos determinan cómo la cámara proyecta los puntos del mundo 3D al

¹⁰⁶https://www.youtube.com/watch?v=6e94ZnYn0_U

¹⁰⁷https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ros2/src/pibotj_rr/pibotj_rr/camera.py

plano de la imagen en 2D, siendo constantes mientras no varíen las características y posiciones relativas entre la óptica y el sensor imagen. Estos parámetros los provee el fabricante de la cámara, aunque se pueden obtener mediante un proceso de calibración.

Para los cálculos teóricos se decidió consultar en numerosas fuentes de información y gracias a la Web *Elinux*¹⁰⁸, se pudieron obtener, sus parámetros técnicos. Asimismo, siguiendo el libro [Hartley and Zisserman, 2004] se consiguieron las fórmulas correspondientes a la distancia focal (Ecuación 6.1) y al centro de la imagen (Ecuación 6.2).

$$f_x = F \cdot \frac{\text{Res. horizontal}}{\text{tam. sensor horizontal}} \quad f_x = 3,04 \cdot \frac{640}{3,674} = 529,56 \quad (6.1)$$

$$f_y = F \cdot \frac{\text{Res. vertical}}{\text{tam. sensor vertical}} \quad f_y = 3,04 \cdot \frac{480}{2,760} = 528,69 \quad (6.2)$$

Ecuación 6.1: Fórmula para calcular la distancia focal teórica

$$c_x = \frac{\text{Res. horizontal}}{2} \quad c_x = \frac{640}{2} = 320 \quad (6.3)$$

$$c_y = \frac{\text{Res. vertical}}{2} \quad c_y = \frac{480}{2} = 240 \quad (6.4)$$

Ecuación 6.2: Fórmula para calcular el centro de la imagen

Para los cálculos prácticos ha sido necesario seguir los pasos de calibración creando los ficheros `calibration.py`¹⁰⁹ y `process.py`¹¹⁰ siguiendo el tutorial¹¹¹. El fichero `calibration.py` captura las imágenes con un patrón de ajedrez, y el fichero `process.py` genera la matriz de parámetros intrínsecos y de distorsión. El proceso se repitió diez veces, siendo la matriz resultante la obtenida como media de las diez matrices. Sus valores son: $f_x = 497,66$, $f_y = 502,16$, $c_x = 325,3$ y $c_y = 240,18$ (Figura 6.11)

Podemos ver que los valores de fábrica difieren de los obtenidos empíricamente. Esto es debido a que cada cámara tiene sus pequeñas diferencias, propias del proceso de calibración.

¹⁰⁸https://elinux.org/Rpi_Camera_Module#Technical_Parameters_.28v.2_board.29

¹⁰⁹<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/camera/calibration/calibration.py>

¹¹⁰<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/camera/calibration/process.py>

¹¹¹<https://www.youtube.com/watch?v=XFBKwme5HYk>

```
calibration_data.txt calibration_data.txt
pi@raspberrypi:~/Desktop/tfg-jlopez/code/camera/calibration/valid_tests/cal_matrix $ python3 meanintrinsic
matrix.py
[[497.6612503890001, 0.0, 325.295473259], [0.0, 502.156098021, 240.18401386500005], [0.0, 0.0, 1.0]]
pi@raspberrypi:~/Desktop/tfg-jlopez/code/camera/calibration/valid_tests/cal_matrix $ cd
```

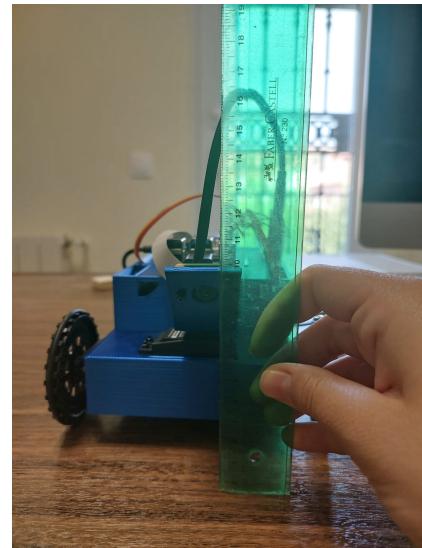
Figura 6.11: Media de la matriz de intrínsecos

Parámetros extrínsecos

Los parámetros extrínsecos relacionan los sistemas de referencia del mundo real y la cámara, describiendo la posición y orientación de la cámara en el sistema de coordenadas del mundo real. Dichos parámetros incluyen la rotación y la traslación de la cámara. En este caso, la cámara tiene una rotación sobre el eje Y de 50º (Figura 6.12 izquierda) y está trasladada 8,8 cm en el eje Z (Figura 6.12 derecha).



Rotación de la cámara



Traslación de la cámara

Figura 6.12: Parámetros extrínsecos

6.2.2. Google Coral USB

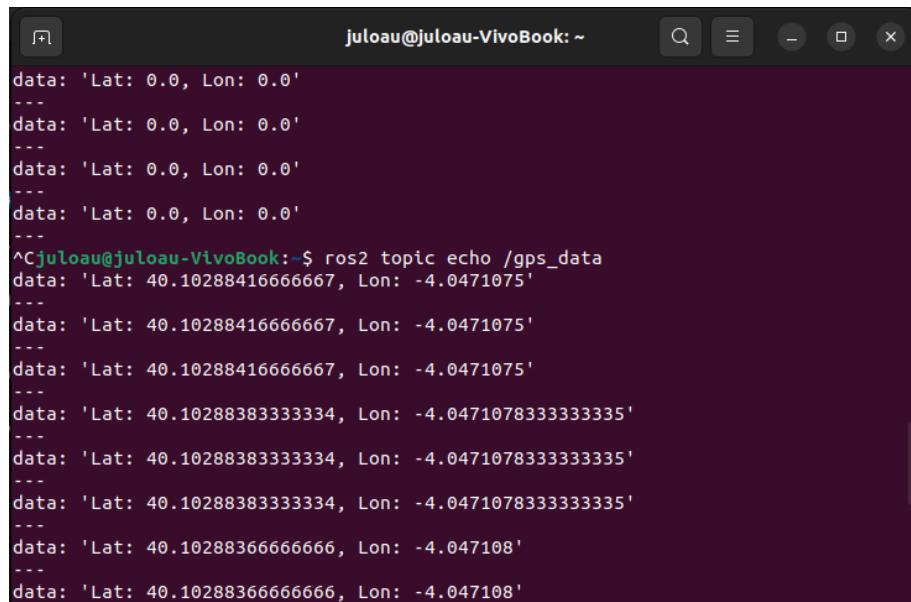
Como se comentó en la Sección 4.1.6, existen severas limitaciones al tratar de acelerar tareas en dispositivos que no cuentan con hardware específico para ello. Por ello, se decidió incluir el dispositivo USB Google Coral. Para su configuración se han seguido los pasos indicados en su web oficial¹¹².

¹¹²<https://coral.ai/docs/accelerator/get-started/>

6.2.3. Módulo GPS

El módulo GPS utiliza una comunicación UART, o comunicación serial, mediante puerto de transmisión TX y puerto de recepción RX. En nuestro caso, se utilizan los pines tratados en la Sección 5.2.

EL serial se encuentra configurado correctamente en `/dev/ttyAMA0` gracias a los pasos de instalación descritos en el Anexo ..., como muestra la Figura 4.13. Para tratar la información obtenida, se decidió usar la librería llamada PyNMEA2, detallada en la Sección 4.2.3, y que se desarrolló con el código del nodo `gps_node`¹¹³ para parsear la información y únicamente publicar la latitud y longitud, como muestra la Figura 6.13. De los valores obtenidos, se decidió usar un conversor de coordenadas¹¹⁴ para demostrar que los valores recibidos eran correctos, como muestra la Figura 6.14.



```
jul0au@jul0au-VivoBook: ~
data: 'Lat: 0.0, Lon: 0.0'
...
data: 'Lat: 40.10288416666667, Lon: -4.0471075'
...
data: 'Lat: 40.10288416666667, Lon: -4.0471075'
...
data: 'Lat: 40.10288416666667, Lon: -4.0471075'
...
data: 'Lat: 40.10288383333334, Lon: -4.0471078333333335'
...
data: 'Lat: 40.10288383333334, Lon: -4.0471078333333335'
...
data: 'Lat: 40.10288383333334, Lon: -4.0471078333333335'
...
data: 'Lat: 40.10288366666666, Lon: -4.047108'
...
data: 'Lat: 40.10288366666666, Lon: -4.047108'
```

Figura 6.13: Valores publicados de latitud y longitud captados del GPS

Debido a la distribución de Ubuntu usada, hay problemas al encender la Raspberry Pi si tiene algo a través del puerto serie¹¹⁵, pero las sugerencias de otros usuarios no conseguían hacer funcionar bien el módulo; por lo tanto, la única solución encontrada fue desconectar el módulo GPS hasta que se inicia sesión a través de ssh y luego conectar el módulo.

El módulo GPS realmente capta información válida cuando se enciende el LED

¹¹³https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ros2/src/pibotj_rr/pibotj_rr/gps_node.py

¹¹⁴<https://www.coordenadas-gps.com/convertidor-de-coordenadas-gps>

¹¹⁵https://wiki.ubuntu.com/EoanErmione/ReleaseNotes#Raspberry_Pi

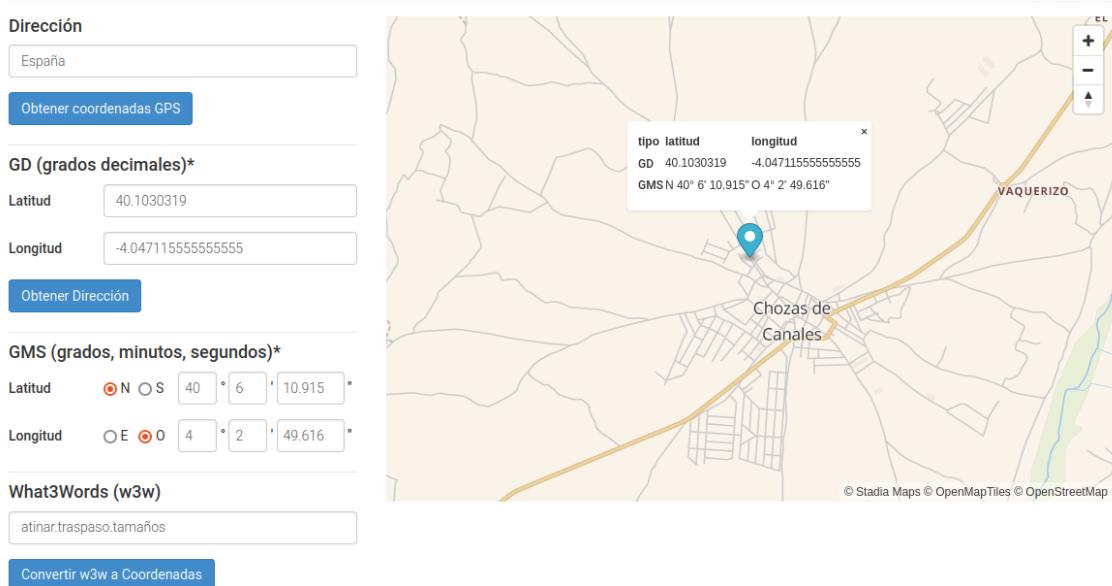


Figura 6.14: Demostración de una correcta localización

azul que tiene integrado. Una forma fiable para que el módulo reciba todo el rato señal correcta es dejarle en el exterior hasta que se encienda el LED y luego poder operar con el módulo en interior o exterior.

6.2.4. Servomotores

Los últimos componentes que faltaban por configurar son los motores, que van conectados a los pines GPIO

Tras seguir los pasos de configuración del Anexo.., importando la librería `RPi.GPIO` se pueden operar directamente. Estos servomotores se mueven por posición entre 0° y 180° , y se controlan mediante modulación del ancho del pulso (PWM), donde la posición del eje del servo depende de la duración del pulso. Para mantener su posición, el servo necesita recibir un pulso cada 20 ms ¹¹⁶ (Figura 6.15). Para cambiar ese PWM¹¹⁷ es necesario usar el procedimiento `ChangeDutyCycle()`, que modifica el porcentaje de tiempo que la señal está activa durante un período de la señal. En este caso, el ciclo de trabajo determina la posición del eje del servo siendo 0° de 2,5% de *duty cycle* y 180° de 12,5% *duty cycle*. Para facilitar la lógica de movimiento se creó una fórmula de conversión de grados a porcentaje de trabajo. En la Figura 6.16 se muestran los distintos valores necesarios para hacer los movimientos que necesitaba PiBotJ: hacia adelante, hacia atrás, girar hacia la izquierda mientras avanza hacia adelante, y girar

¹¹⁶<https://docs.rs-online.com/0e85/0900766b8123f8d7.pdf>

¹¹⁷<https://solectroshop.com/es/blog/que-es-pwm-y-como-usarlo--n38>

hacia la derecha mientras avanza hacia adelante.

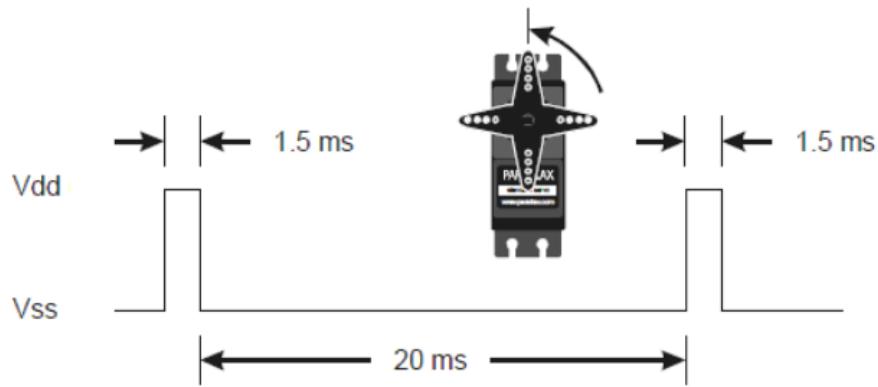


Figura 6.15: Diagrama para la posición central del servo

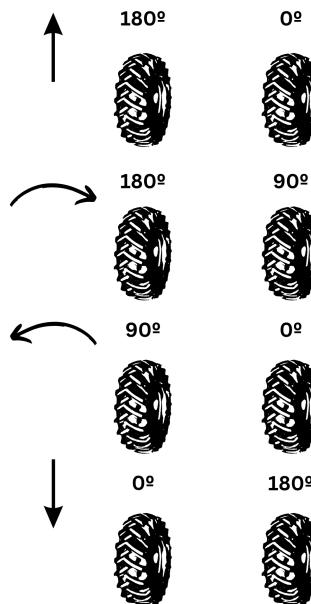


Figura 6.16: Distintas direcciones de PiBotJ

Una vez configurados todos los componentes del robot, ya se puede pasar a desarrollar el *software* que conforma el comportamiento final de este.

6.3. Software aplicado al robot físico

PiBotJ ya estaba listo para poder recibir la inteligencia que necesitaba para conseguir el objetivo principal descrito en la Sección 3.1. A diferencia de la Sección

6.1.2 que empleaba ROS 2 control, en este caso al estar todo configurado a bajo nivel, se decidió usar una arquitectura de nodos con *topics* publicadores y suscriptores.

6.3.1. Red neuronal para la detección de baches

Para conseguir la detección, se procedió al entrenamiento de un modelo de aprendizaje supervisado que detectaba baches. Se decidió usar YOLOv8, el cual hace uso de una única red neuronal convolucional para detectar objetos en imágenes en tiempo real y con facilidad de exportación a diferentes dispositivos.

Para poder entrenar dicho modelo, fue necesario usar un *dataset* tomado de [moinfaisal, 2023] que estaba formado por 202 imágenes ya segmentadas. El entrenamiento duró alrededor de una hora y con una duración de 120 *epochs*. Una vez pasada esa hora, se pudo obtener el modelo entrenado en formato .pt y todos los resultados del entrenamiento aparecen mostrados en la Figura 6.17 y 6.18.

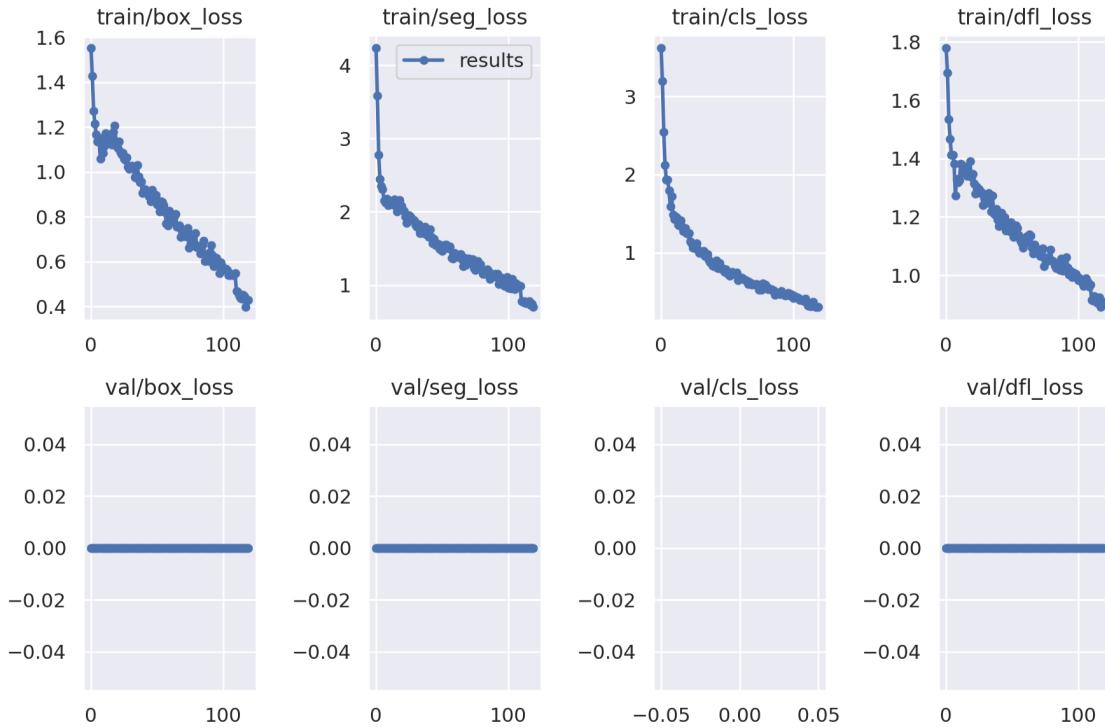


Figura 6.17: Gráficas de resultados del entrenamiento

En la Figura 6.17 se puede ver que aparecen cuatro gráficas en la parte superior midiendo el error obtenido durante el entrenamiento del modelo y se puede apreciar que disminuye a medida que pasa el tiempo y por lo tanto, se puede decir que existe un correcto entrenamiento. Por otro lado, las cuatro imágenes que aparecen debajo

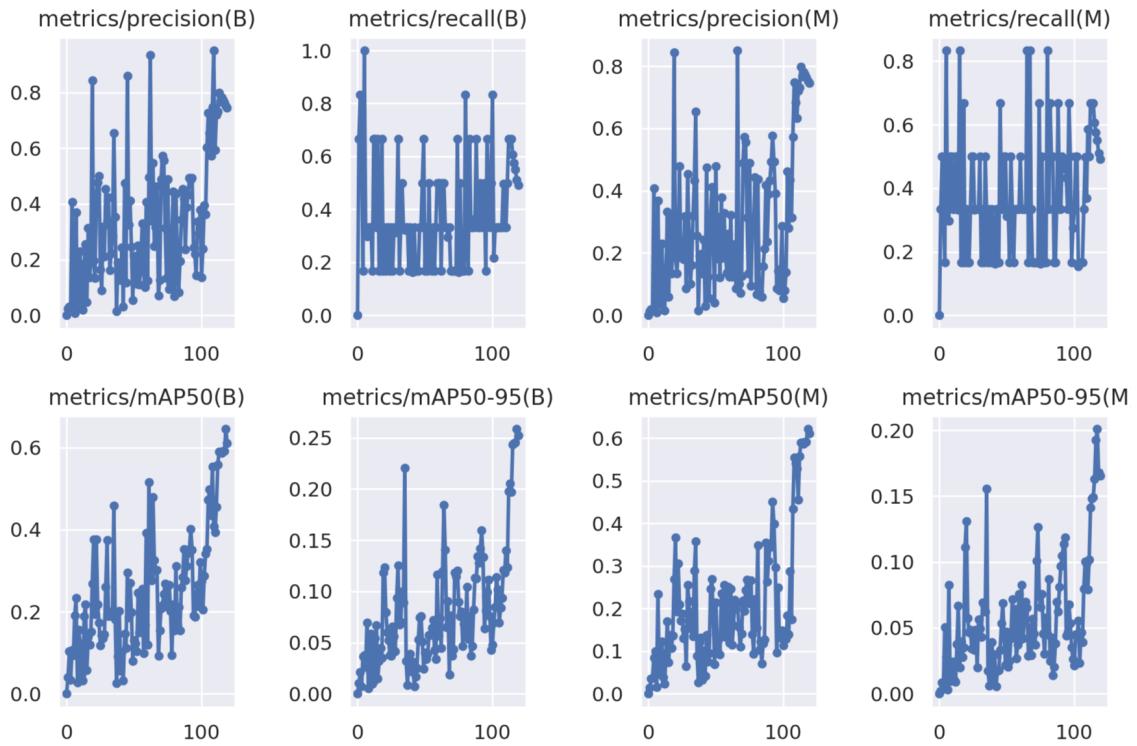


Figura 6.18: Gráficas de resultados del entrenamiento: métricas

se tratan de las pérdidas calculadas en el conjunto de validación. Si las pérdidas en validación aumentan mientras las pérdidas de entrenamiento disminuyen, puede ser una señal de *overfitting* (sobreajuste), en este caso al solo usar una imagen de validación el valor permanece constante sobre el 0.

En la Figura 6.18, en concreto en la parte superior, se puede ver las métricas de precisión y exhaustividad (*recall*) que evalúan la capacidad del modelo para identificar correctamente las instancias de cada clase ya que valores son crecientes. En la parte inferior, la métrica *Mean Average Precision* (mAP) es creciente e indica que el modelo está mejorando en la detección y clasificación ya que, evalúa el equilibrio entre precisión y exhaustividad.

Para comprobar que el modelo funciona correctamente, se ha aplicado sobre una serie de imágenes mostradas en la Figura 6.19 y sobre un video¹¹⁸. Todo está documentado en un Jupyter notebook¹¹⁹ basado en el tutorial de Muhammad Moin¹²⁰.

Una vez demostrado que el modelo es capaz de detectar bien los baches, es el momento de convertir ese modelo para que fuera capaz de trabajar dentro de la RPi

¹¹⁸<https://www.youtube.com/watch?v=Gnwciv4pWf0>

¹¹⁹<https://colab.research.google.com/drive/1QRFWn5tQNxs6ldnomCZNv3DUDi8FudW2?usp=sharing>

¹²⁰<https://www.youtube.com/watch?v=bA10bHUIErM>

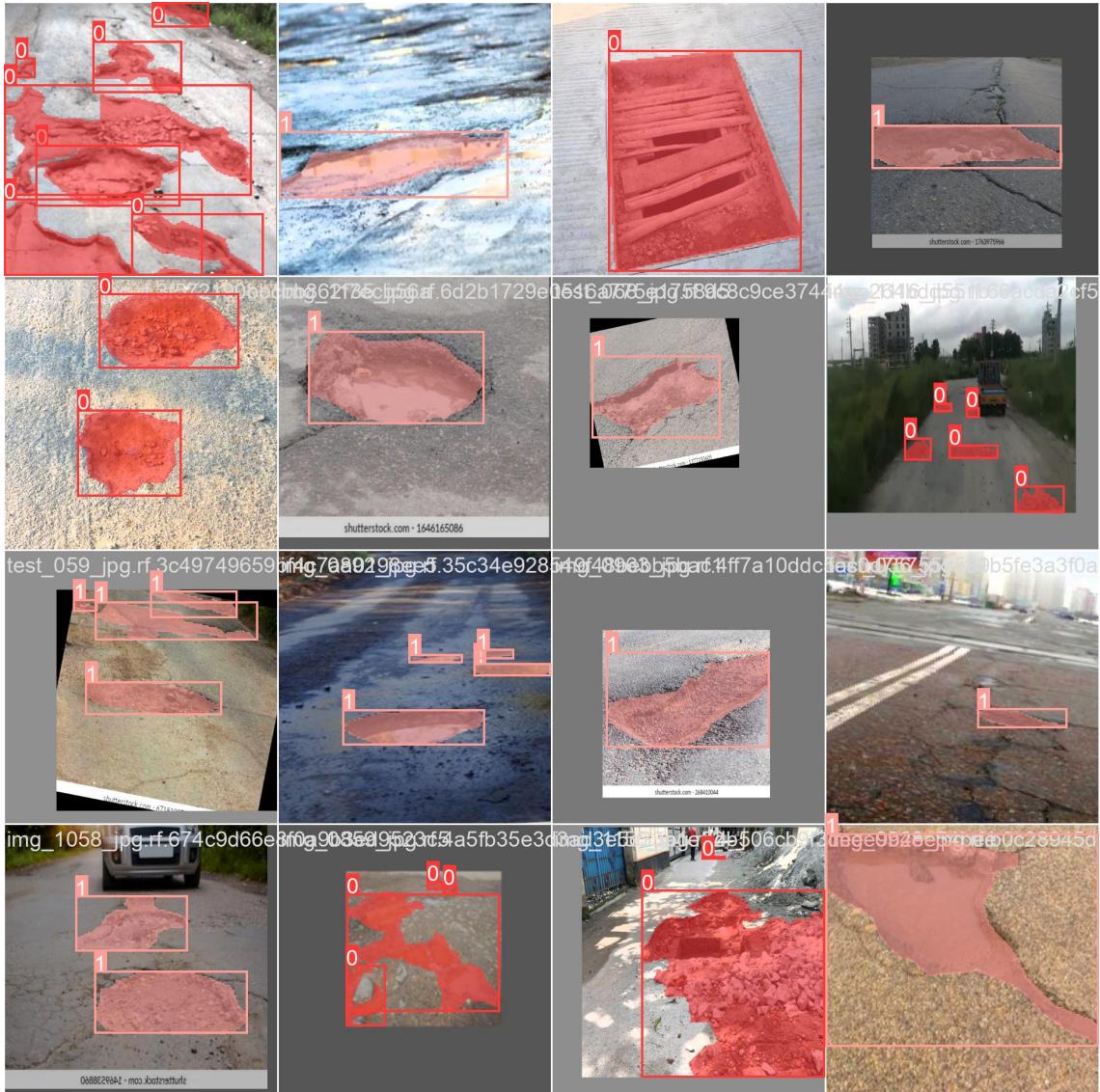


Figura 6.19: Demostración del modelo funcionando

4 y Google Coral. Gracias a este video¹²¹, se pudo conseguir realizar este Jupyter notebook¹²² que convierte el modelo entrenado a un formato edgetpu de 192x192 imágenes, para que el dispositivo Google Coral pueda operar con él, llamado *bestv2_full_integer_quant_edgetpu.tflite*¹²³.

¹²¹<https://www.youtube.com/watch?v=w4yH0RvDBw0>

¹²²<https://colab.research.google.com/drive/1XU20NX5Mq2zNW1Zu3hmNIylqbto5hGfm?usp=sharing>

¹²³https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ros2/src/pibotj_rr/custom_model_lite/bestv2_full_integer_quant_edgetpu.tflite

Valores obtenidos del modelo

Del presente modelo se pueden obtener los detalles de entrada y de salida del modelo, una vez se ha cargado. De entrada podemos encontrar los siguientes valores:

```
[{'name': 'serving_default_images:0', 'index': 0, 'shape':
array([ 1, 192, 192,    3], dtype=int32), 'shape_signature':
array([ 1, 192, 192,    3], dtype=int32), 'dtype': <class 'numpy.int8'>,
'quantization': (0.01865844801068306, -14), 'quantization_parameters':
{'scales': array([0.01865845], dtype=float32),
'zero_points': array([-14], dtype=int32), 'quantized_dimension': 0},
'sparsity_parameters': {}}]
```

Este tensor define la entrada para un modelo cuantizado que toma imágenes RGB de tamaño 192x192, en formato entero de 8 bits (int8). La cuantización utiliza una escala de 0.0187 y un punto cero de -14, mapeando los valores al rango del tensor int8, lo que ayuda a mejorar la eficiencia del modelo. Por otro lado, de salida podemos encontrar los siguientes valores:

```
[{'name': 'PartitionedCall:0', 'index': 2, 'shape':
array([ 1,   38, 756], dtype=int32), 'shape_signature':
array([ 1,   38, 756], dtype=int32), 'dtype': <class 'numpy.int8'>,
'quantization': (0.017339961603283882, 12), 'quantization_parameters':
{'scales': array([0.01733996], dtype=float32),
'zero_points': array([12], dtype=int32), 'quantized_dimension': 0},
'sparsity_parameters': {}},
{'name': 'PartitionedCall:1', 'index': 1, 'shape':
array([ 1,   48, 48,   32], dtype=int32), 'shape_signature':
array([ 1,   48, 48,   32], dtype=int32), 'dtype': <class 'numpy.int8'>,
'quantization': (0.020256992429494858, -114), 'quantization_parameters':
{'scales': array([0.02025699], dtype=float32),
'zero_points': array([-114], dtype=int32), 'quantized_dimension': 0},
'sparsity_parameters': {}}]
```

Estos son dos tensores de salida en un modelo cuantizado. El primer tensor (PartitionedCall:0) tiene forma [1, 38, 756], y el segundo tensor (PartitionedCall:1)

tiene forma [1, 48, 48, 32]. Ambos utilizan el formato int8 y aplican parámetros de cuantización específicos que ajustan el rango de valores, optimizando el modelo para dispositivos de bajo consumo. De ambos tensores el que nos interesa es el segundo y es debido a su forma. Posee 32 canales y para este proyecto únicamente nos interesan los canales 0 y 1 como muestra la Figura 6.19.

Aplicación del modelo

Una vez se conocen las características que posee el modelo, es necesario extraer la información y convertirla en el formato adecuado para poder operar con ella (Código 6.4)

```

# Redimensionar la imagen a las dimensiones requeridas por el modelo
input_shape = self.input_details[0]['shape']
height, width = input_shape[1], input_shape[2]
resized_frame = cv2.resize(frame, (width, height))

# Convertir la imagen a formato adecuado
input_data = np.expand_dims(resized_frame, axis=0).astype(np.float32)
# Normalizacion
input_data = (input_data - 127.5) / 127.5

# Convertir la imagen a int8
scale, zero_point = self.input_details[0]['quantization']
input_data = (input_data / scale + zero_point).astype(np.int8)

# Establecer el tensor de entrada y realizar la inferencia
self.interpreter.set_tensor(self.input_details[0]['index'], input_data)
self.interpreter.invoke()

# Saca informacion del tensor de salida 2 que tiene forma [1, 48, 48,
# 32]
output_data1 =
    self.interpreter.get_tensor(self.output_details[1]['index'])
# elimina la dimensipn del batch (el primer valor del array)
prediction1 = np.squeeze(output_data1)

# Descuantificar la mascara de baches mirando el canal 0
pothole_mask_class0 = (prediction1[:, :, 0].astype(np.float32) -
    zero_point) * scale
# Descuantificar la mascara de baches mirando el canal 1
pothole_mask_class1 = (prediction1[:, :, 1].astype(np.float32) -
    zero_point) * scale

```

Código 6.4: Cómo convertir los datos del modelo a un formato adecuado

Los valores de pothole_mask_class0 y pothole_mask_class1 ya se pueden

tratar y establecer una umbralización de 0.6 para considerar dichos valores si han detectado bache o no. Sin embargo, los valores de `pothole_mask_class0` y `pothole_mask_class1` eran muy inestables y se consideró la técnica de *Media Móvil Exponencial* (EMA) (Ecuación 6.5) para que no fluctuasen tanto los valores y darle más peso a las medidas más recientes. Esta técnica es común en análisis técnico, como se describe en [Hansun, 2013], pero se decidió incluir en ese proyecto ya que satisfacía las necesidades buscadas y por su liviana carga computacional.

$$S_t = \alpha \cdot Y_t + (1 - \alpha) \cdot S_{t-1}$$

Ecuación 6.3: Fórmula de la media móvil exponencial

Siendo α el valor de grado de disminución de ponderación, Y_t el valor actual en tiempo t, S_t es el valor de EMA en tiempo t y S_{t-1} es el valor de EMA en tiempo t-1.

Obtención del contorno y coordenadas

Después de saber si en una imagen había bache o no, se consideraría bache si la detección ocurre durante 3 segundos sin interrupciones. Una vez pasado ese tiempo, habrá que publicar los píxeles asociados a ese contorno.

Para conseguir los píxeles, se toma la máscara binarizada usando el umbral definido previamente de 0.6. De esa forma, es posible encontrar los píxeles del contorno más grande de la máscara pero, esos píxeles están en base a las dimensiones de la máscara que es de 48x48 píxeles; así que, es necesario aplicar un factor de escala para que los píxeles estén en base a la imagen que es de 192x192 píxeles. Se puede ver un resultado final del contorno detectado en la Figura 6.20.

6.3.2. Cámara pinhole

Extraer información de la cámara no es fácil y por ello se decidió simplificar la cámara definiéndola como una cámara *pinhole*, o también conocida como una cámara estenopeica. El modelo *pinhole* describe una cámara (sin lente) y con una pequeña apertura. Los rayos de luz atraviesan la apertura y proyectan una versión invertida de lo que se capta en el fondo de la cámara (o el plano de la imagen) (Figura 6.21).

Este modelo explica cómo se realiza la proyección de un punto en el espacio tridimensional hacia el plano bidimensional de una imagen. En otras palabras, el modelo

¹²⁴https://es.wikipedia.org/wiki/C%C3%A1mara_estenopeica

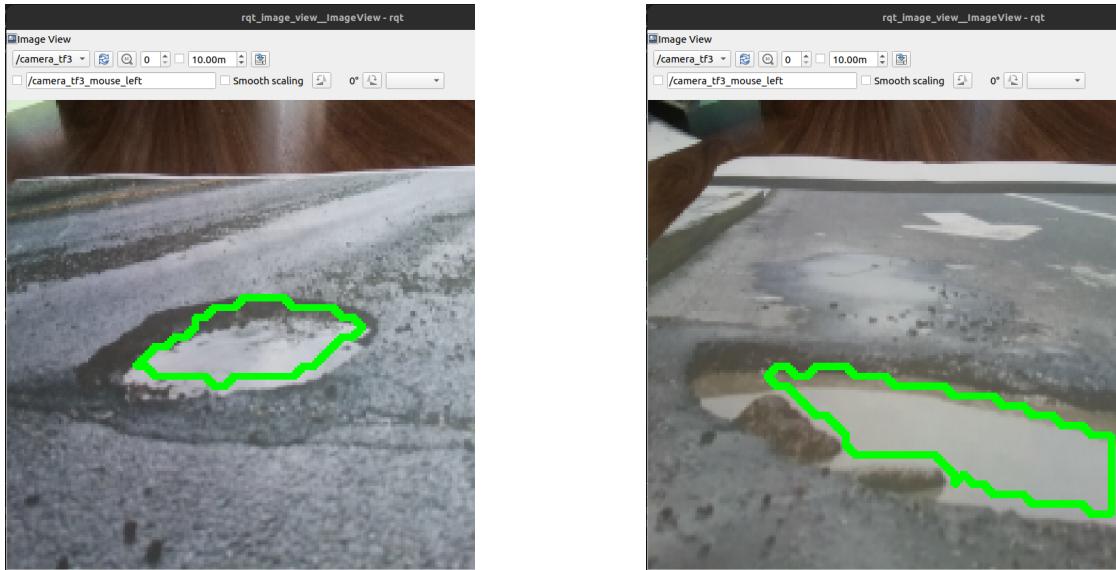
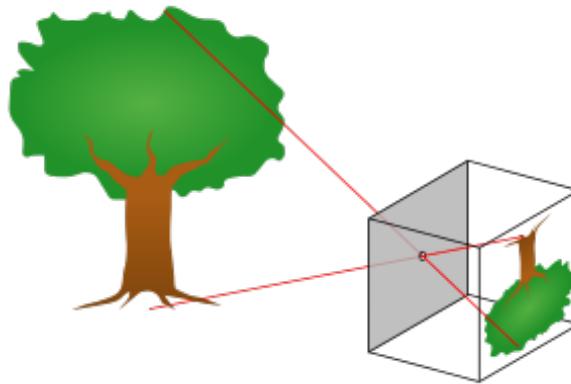


Figura 6.20: Contorno del baches detectados

Figura 6.21: Funcionamiento de una cámara esteneopeica ¹²⁴

de cámara define una fórmula matemática que, a partir de las coordenadas de un punto en el espacio 3D, con referencia a la cámara, permite calcular las coordenadas en las que ese punto aparecería en una fotografía. Un resumen de fórmulas matemáticas aparecen en la Figura 6.22.

En la Figura 6.22 se pueden ver los pámetros intrínsecos y extrínsecos que son tomados de la calibración de la cámara, descritos en la Sección 6.2.1. Dicha calibración se realizó sobre imágenes de 640x480 y los píxeles del contorno del bache está en base de imágenes de 192x192, por lo que era necesario aplicar otro factor de conversión.

Una vez se conoce cómo funciona el modelo pinhole, es importante destacar que para poder convertir los píxeles del mundo 2D al mundo real, hay que tener en cuenta que hay tres sistemas de coordenadas diferentes como muestra la Figura 6.23 que ha sido obtenida del artículo [Ortiz et al., 2017] y que el modelo pinhole únicamente es

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & [s] & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Diagrama que muestra la relación entre las coordenadas del punto proyectivo en pixeles (u, v) y las coordenadas 3D en el mundo real (X, Y, Z). La ecuación se divide en tres secciones:

- Parámetros intrínsecos:** Matriz de 3x3 que incluye la distancia focal (f_x, f_y), el factor de escala para homogeneizar ($[s]$) y los centros de imagen (c_x, c_y).
- Parámetros extrínsecos:** Matriz de 3x4 que incluye las matrices de rotación (r_{ij}) y traslación (t_x, t_y, t_z).
- Matrices de rotación y traslación:** Se muestra como una subsección de los parámetros extrínsecos.

Figura 6.22: Relación de fórmulas del modelo pinhole

capaz de convertir del sistema de coordenadas del mundo real al sistema de coordenadas de la cámara, algo a tener en cuenta.

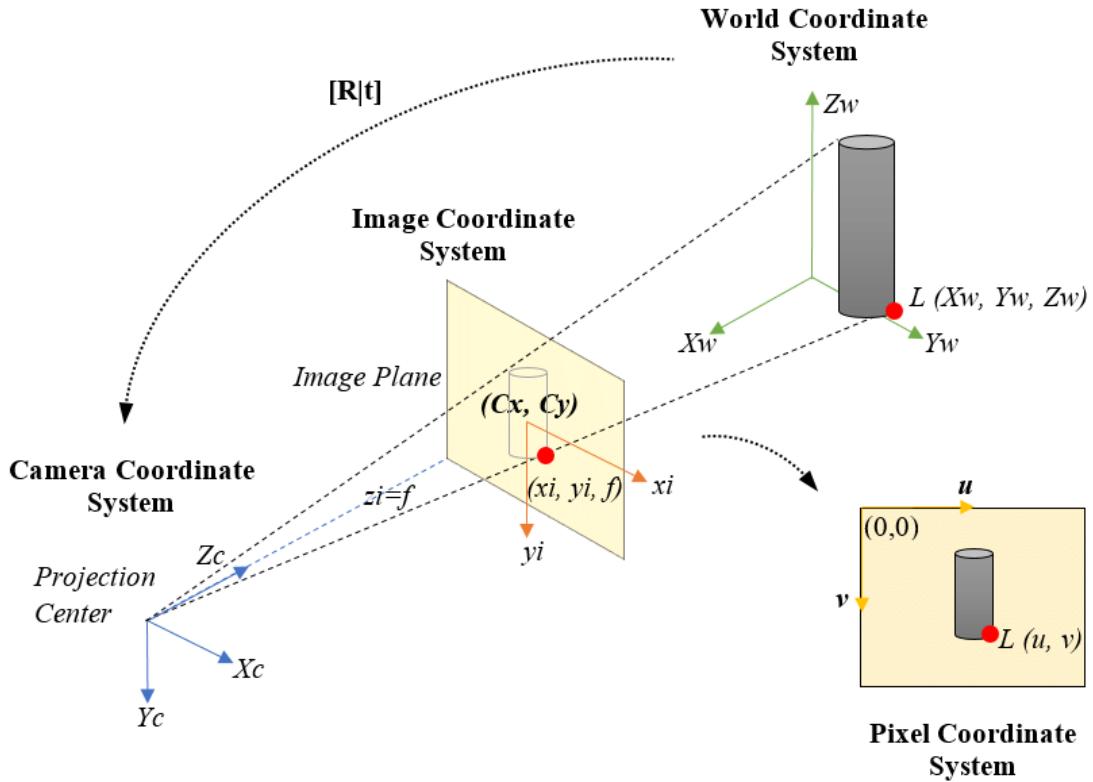


Figura 6.23: Sistemas de coordenadas

Otro dato importante a tener en cuenta es que solo disponemos de una cámara, y

por tanto, el sistema no puede encontrar el punto 3D asociado en una imagen debido a que cada punto de la línea OX proyecta al mismo píxel en el plano imagen; se pierde el valor de la profundidad (eje Z). Además, al hacer la conversión entre los distintos sistemas de coordenadas del modelo *pinhole*, el valor de profundidad (de Z) también se pierde. Es por todo lo dicho anteriormente que hay que emplear la hipótesis suelo.

La hipótesis suelo (Figura 6.24 tomada del artículo [Vega and Cañas, 2019]) supone que todos los objetos del mundo en 3D están pegados al suelo, por lo tanto, asumiendo que $Z = 0$, nos permitirá estimar la conversión de 2D a 3D con una sola cámara.

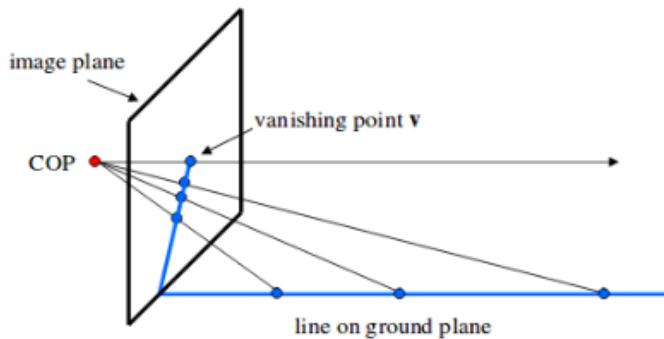


Figura 6.24: La hipótesis suelo asume que todos los objetos están en el suelo

Para conseguir la solución a este problema la tomamos del artículo [Vega and Cañas, 2019], donde los autores nos presentan un sistema de visión de código abierto que simplifica el uso de cámaras en la robótica educativa. En él se detallan todos los pasos que se han explicado previamente y el código asociado con la obtención de los píxeles en coordenadas del mundo en 3D, se encuentra definido en `camera_pinholev2_node.py`¹²⁵.

6.3.3. Algoritmo de la lazada

Para calcular el área de las coordenadas del mundo real, se ha optado por seguir el algoritmo de la lazada¹²⁶ o también conocido como el área de Gauss. Este algoritmo matemático es usado para calcular el área de un polígono simple cuyos vértices están descritos como pares de coordenadas en el plano. Tiene múltiples aplicaciones en agrimensura y topografía.

El nombre de este algoritmo es debido al constante cruce de productos de las coordenadas, que simula atarse los cordones, como muestra la 6.5.

¹²⁵https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ros2/src/pibotj_rr/pibotj_rr/camera_pinholev2_node.py

¹²⁶https://es.wikipedia.org/wiki/F%C3%B3rmula_de_%C3%A1rea_de_Gauss

$$A = \frac{1}{2} \left| \sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n \right|$$

Ecuación 6.4: Fórmula del algoritmo de la lazada

Donde A es el área del polígono, n es el número de lados del polígono y (x_i, y_i) , $i = 1, 2, \dots, n$ son los vértices del polígono de forma alternativa.

Un ejemplo que ilustra este proceso y demuestra que se cumple el algoritmo de la lazada con respecto a la fórmula usada para calcular el área de un rectángulo, se puede apreciar en la Figura 6.25.

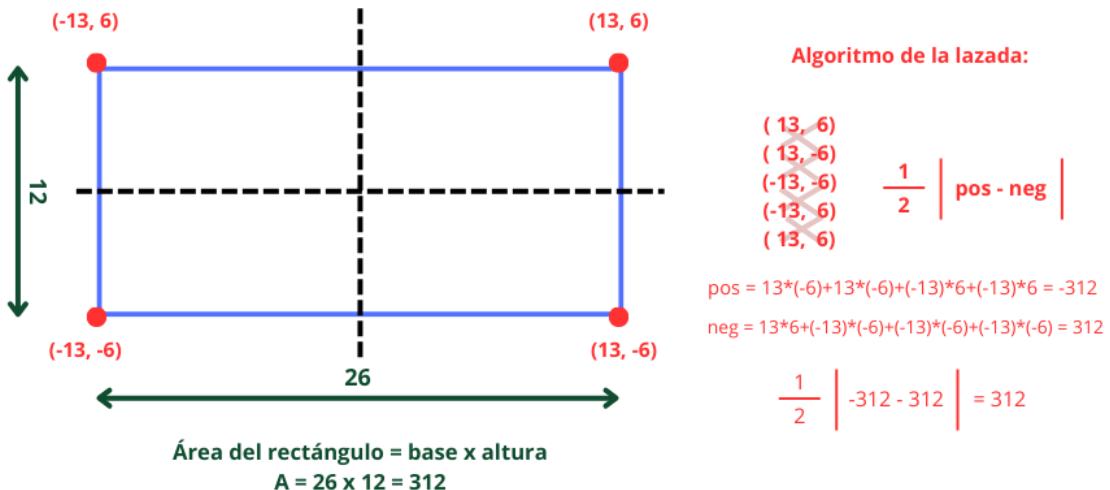


Figura 6.25: Demostración del algoritmo de la lazada

Este algoritmo aparece aplicado una vez que se ha convertido las coordenadas al sistema del mundo real en `camera_pinholev2_node.py`¹²⁷.

6.3.4. Interfaz Web

La forma que se decidió usar la *Interacción Humano-Robot* (HRI) para poder cumplir con el objetivo principal descrito en la Sección 3.1, fue a través de una interfaz web. Dicha interfaz web, ha sido creada gracias a las herramientas descritas en la Sección 4.2.11. `Rosbridge_server` permite a las páginas web (que actúan como clientes) comunicarse con el servidor (que en este caso se inicializa en PiBotJ). Para acceder como cliente, en una ventana de un navegador hay que escribir

¹²⁷https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ROS2/src/pibotj_rr/pibotj_rr/camera_pinholev2_node.py

`http://<ip de PiBotJ>:8000/index.html`, sustituyendo `<ip de PiBotJ>` por la IP obtenida al conectarse PiBotJ a la red.

Una vez habilitada esa comunicación, ya se podía apreciar la página web¹²⁸ (Figura 6.26). Se decidió incluir un mapa, usando OpenStreetMaps, en el que apareciese un marcador que tiene asociado el valor de área/volumen por cada bache detectado. Por otro lado, se decidió incluir una botonera para poder ser capaz de controlar el movimiento de las ruedas de PiBotJ cuando se quiera usar el modo teleoperado. Dicha botonera se conforma de cuatro botones que publican los distintos movimientos al robot, y cuando se deja de pulsar, el robot se para.

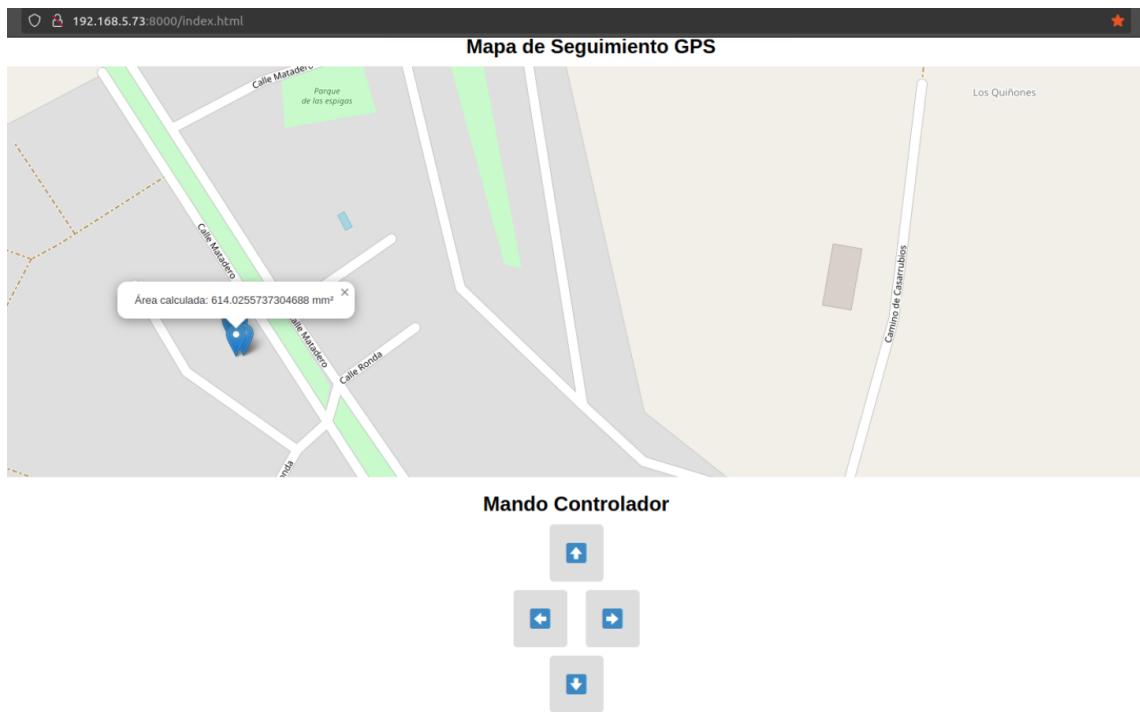


Figura 6.26: Interfaz web

6.3.5. Autónomo

PiBotJ ya se ha visto que se puede controlar de forma teleoperada, usando la botonera definida en la Sección 6.3.4 pero, para tener otra alternativa con respecto a la navegación de PiBotJ, se ha decidido implementar una lógica para que su movimiento sea autónomo.

Lo ideal para este caso es que PiBotJ fuera capaz de navegar autónomamente por carreteras pero debido a la imposibilidad de probarlo en ese entorno, se decidió hacer

¹²⁸<https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ros2/index.html>

una recreación de una carretera real¹²⁹ a escala 769:10000 sobre una cartulina negra. En la Figura 6.27 (izquierda) se muestran las medidas reales y en la Figura 6.27 (derecha) se muestra las medidas definidas. El resultado final aparece en la Figura 6.28.

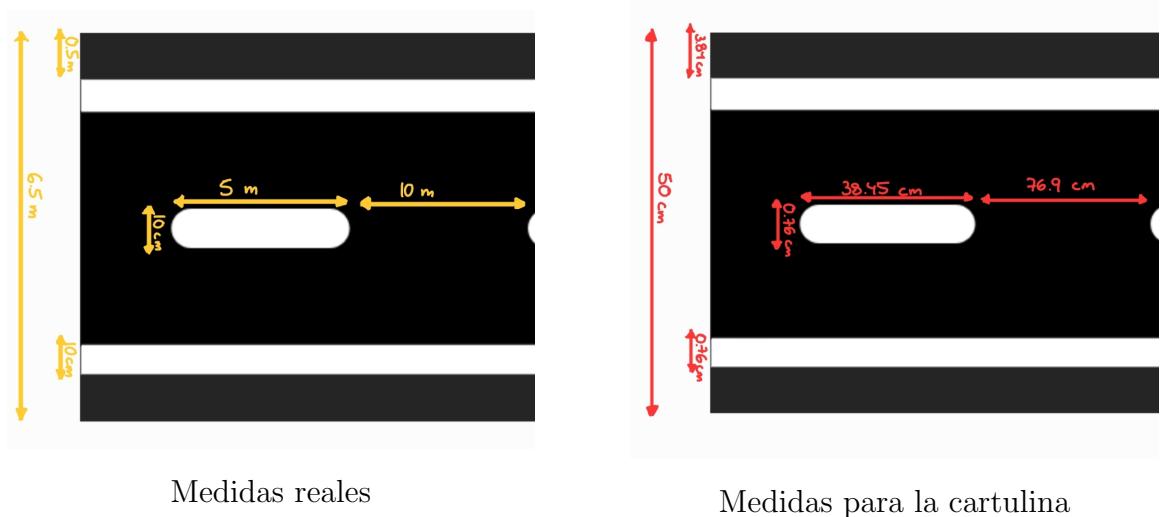


Figura 6.27: Conversión de medidas de la carretera

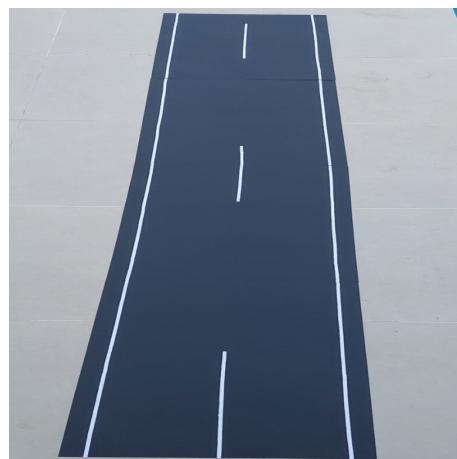


Figura 6.28: Carretera creada en cartulina

Una vez definido el entorno donde se iba a trabajar, era el momento de definir la lógica a usar y se decidió unir un algoritmo de detección de líneas junto a la técnica de navegación local conocida como *Virtual Force Field* (VFF).

Detección de líneas

Es importante tener en cuenta que este algoritmo se va a ejecutar únicamente cuando no haya bache en el entorno controlado. Para conseguirlo, es necesario procesar

¹²⁹<https://www.totalmentereflejante.com/medidas-rayas-de-carretera/>

la imagen en escala de grises para detectar los contornos de las líneas, usando una umbralización adaptativa para destacar los bordes y una operación de apertura morfológica para eliminar pequeños puntos blancos que representan ruido en la imagen binaria (Código 6.5). El resultado final queda reflejado en la Figura 6.29 y el código está plasmado en `camera_dl_node.py`¹³⁰.

```
# Convierte la imagen a escala de grises
gray = cv2.cvtColor(resized_frame, cv2.COLOR_BGR2GRAY)
# Aplica un umbral para destacar los bordes
th1 = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 23, -50)

# Elimina ruido
kernel = np.ones((3, 3), np.uint8)
opened_th1 = cv2.morphologyEx(th1, cv2.MORPH_OPEN, kernel)
```

Código 6.5: Filtro para obtener las líneas blancas

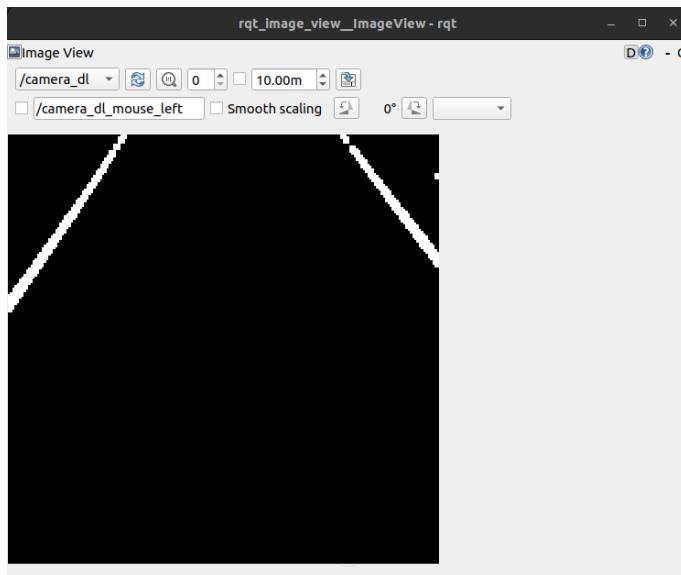


Figura 6.29: Filtro resultante aplicado para la detección de líneas

En la Figura 6.30 se puede ver los distintos tipos de situaciones a las que se enfrenta PiBotJ cuando detecta líneas y dependiendo de la situación, PiBotJ reaccionaría de una forma u otra. Cuando detecta las dos líneas, PiBotJ sigue recto mientras que si detecta únicamente una línea, deberá avanzar hacia adelante y si dicha línea no se encuentra en el centro de la imagen, PiBotJ giraría hasta que se situase en el centro.

¹³⁰https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ros2/src/pibotj_rr/pibotj_rr/camera_dl_node.py

Por otro lado, si no detecta líneas avanzaría recto durante 3 segundos y empezaría a girar.

Distintos casos de detección de líneas de PiBot

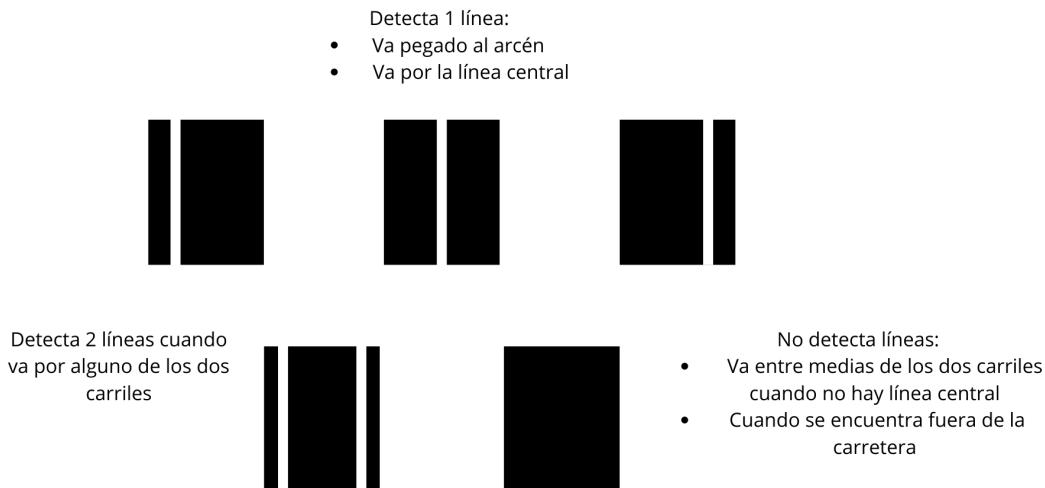


Figura 6.30: Distintos casos de la detección de líneas

Virtual Force Field

Es un método de navegación local para que un robot autónomo evite obstáculos en su entorno. Este método se compone de una fuerza repulsiva para evitar el obstáculo y de una fuerza atractiva para alcanzar el destino objetivo¹³¹. Estas fuerzas se suman vectorialmente para obtener una fuerza resultante, que indica la dirección y velocidad de movimiento del robot. Esta fuerza resultante se puede calcular siguiendo la Ecuacion 6.5.

$$F_{resultante} = \alpha \cdot F_{atractiva} + \beta \cdot F_{repulsiva}$$

Ecuación 6.5: Ecuación de VFF

El robot se mueve en la dirección de la fuerza resultante, ajustando su trayectoria en tiempo real según las posiciones del obstáculo y el objetivo. En este caso, se decidió fijar como objetivo, un punto siempre hacia adelante y como repulsión, el punto más cercano detectado del contorno al robot, en coordenadas del mundo real, y aplicar

¹³¹<https://robmovjuloau2022.blogspot.com/2022/11/task-3-local-navigation-with-vff.html>

la fuerza repulsiva respecto a él. La Figura 6.31 muestra la dirección de las distintas fuerzas. Todo está documentado en `camera_vff_node.py`¹³².

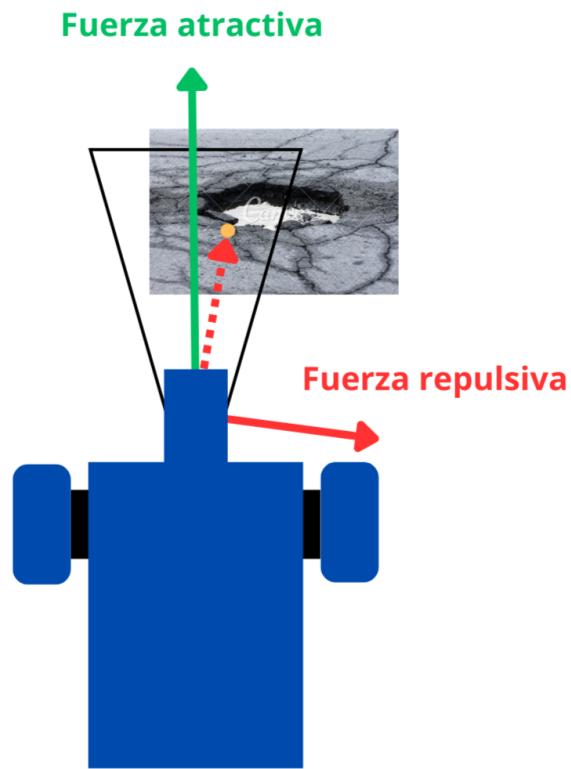


Figura 6.31: Diagrama de fuerzas del algoritmo VFF

¹³²https://github.com/RoboticsURJC/tfg-jlopez/blob/main/code/ROS2/src/pibotj_rr/pibotj_rr/camera_vff_node.py

Capítulo 7

Experimentos

Toda la vida es un experimento. Cuantos más experimentos hagas, mejor

Ralph Waldo Emerson

Escribe aquí un párrafo explicando brevemente

7.1. ¿Aplicación Completa?

Teleoperado y entorno controlado

explicar experimentos impresora del instituto

explicar filtro de imagen v1 con los valores de canny puestos a ojo

explicar tipos de tflite usadas

<https://github.com/RoboticsURJC/tfg-jlopez/wiki/Experimentos> (diseño
carretera y la primera prueba hecha en la calle con las ruedas de ActivityBot)

7.2. Snippets

Puede resultar interesante, para clarificar la descripción, mostrar fragmentos de código (o *snippets*) ilustrativos. En el Código 7.1 vemos un ejemplo escrito en C++.

En el Código 7.2 vemos un ejemplo escrito en Python.

```

void Memory::hypothesizeParallelograms () {
    for(it1 = this->controller->segmentMemory.begin(); it1++) {
        squareFound = false; it2 = it1; it2++;
        while ((it2 != this->controller->segmentMemory.end()) &&
            (!squareFound)) {
            if (geometry::haveACommonVertex((*it1), (*it2), &square)) {
                dist1 = geometry::distanceBetweenPoints3D ((*it1).start,
                    (*it1).end);
                dist2 = geometry::distanceBetweenPoints3D ((*it2).start,
                    (*it2).end);
            }
            // [...]
}

```

Código 7.1: Función para buscar elementos 3D en la imagen

```

def mostrarValores():
    print (w1.get(), w2.get())

    master = Tk()
    w1 = Scale(master, from_=0, to=42)
    w1.pack()
    w2 = Scale(master, from_=0, to=200, orient=HORIZONTAL)
    w2.pack()
    Button(master, text='Show', command=mostrarValores).pack()

mainloop()

```

Código 7.2: Cómo usar un Slider

7.3. Verbatim

Para mencionar identificadores usados en el código —como nombres de funciones o variables— en el texto, usa el entorno literal o verbatim `hypothesizeParallelograms()`. También se puede usar este entorno para varias líneas, como se ve a continuación:

```

void Memory::hypothesizeParallelograms () {
// add your code here
}

```

7.4. Ecuaciones

Si necesitas insertar alguna ecuación, puedes hacerlo. Al igual que las figuras, no te olvides de referenciarlas. A continuación se exponen algunas ecuaciones de ejemplo:

Ecuación 7.1 y Ecuación 7.1.

$$H = 1 - \frac{\sum_{i=0}^N \left(\frac{d_{js} + d_{je}}{2} \right)}{M}$$

Ecuación 7.1: Ejemplo de ecuación con fracciones

$$v(\text{entrada}) = \begin{cases} 0 & \text{if } \epsilon_t < 0,1 \\ K_p \cdot (T_t - T) & \text{if } 0,1 \leq \epsilon_t < M_t \\ K_p \cdot M_t & \text{if } M_t < \epsilon_t \end{cases} \quad (7.1)$$

Ecuación 7.2: Ejemplo de ecuación con array y letras y símbolos especiales

7.5. Tablas o cuadros

Si necesitas insertar una tabla, hazlo dignamente usando las propias tablas de L^AT_EX, no usando pantallazos e insertándolas como figuras... En el Cuadro 7.1 vemos un ejemplo.

Parámetros	Valores
Tipo de sensor	Sony IMX219PQ[7] CMOS 8-Mpx
Tamaño del sensor	3.674 x 2.760 mm (1/4"format)
Número de pixels	3280 x 2464 (active pixels)
Tamaño de pixel	1.12 x 1.12 um
Lente	f=3.04 mm, f/2.0
Ángulo de visión	62.2 x 48.8 degrees
Lente SLR equivalente	29 mm

Cuadro 7.1: Parámetros intrínsecos de la cámara

En los textos puedes poner palabras en *cursiva*, para aquellas expresiones en sentido *figurado*, palabras como *robot*, que está fuera del diccionario castellano, o bien para resaltar palabras de una colección: (a) es la primera letra del abecedario, (b) es la segunda, etc.

Al poner las dos líneas del anterior párrafo, este aparecerá separado del anterior. Si no las pongo, los párrafos aparecerán pegados. Sigue el criterio que consideres más oportuno.

7.6. Segunda sección

No olvides incluir imágenes y referenciarlas, como la Figura 7.1.



Figura 7.1: Robot aspirador Roomba de iRobot.

Ni tampoco olvides de poner las URLs como notas al pie. Por ejemplo, si hablo de la Robocup¹.

7.6.1. Números

En lugar de tener secciones interminables, como la Sección 1.1, divídelas en subsecciones.

Para hablar de números, mételos en el entorno *math* de L^AT_EX, por ejemplo, $1,5Kg$. También puedes usar el símbolo del Euro como aquí: 1.500€ .

7.6.2. Listas

Cuando describas una colección, usa `itemize` para ítems o `enumerate` para enumerados. Por ejemplo:

- *Entorno de simulación.* Hemos usado dos entornos de simulación: uno en 3D y otro en 2D.
- *Entornos reales.* Dentro del campus, hemos realizado experimentos en Biblioteca y en el edificio de Gestión.

1. Primer elemento de la colección.

¹<http://www.robocup.org>

2. Segundo elemento de la colección.

Referencias bibliográficas Cita, sobre todo en este capítulo, referencias bibliográficas que respalden tu argumento. Para citarlas basta con poner la instrucción `\cite` con el identificador de la cita. Por ejemplo: libros como [?], artículos como [?], URLs como [?], tesis como [?], congresos como [?], u otros trabajos fin de grado como [?].

Las referencias, con todo su contenido, están recogidas en el fichero `bibliografia.bib`. El contenido de estas referencias está en formato BibTeX. Este formato se puede obtener en muchas ocasiones directamente, desde plataformas como Google Scholar u otros repositorios de recursos científicos.

Existen numerosos estilos para reflejar una referencia bibliográfica. El estilo establecido por defecto en este documento es APA, que es uno de los estilos más comunes, pero lo puedes modificar en el archivo `memoria.tex`; concretamente, cambiando el campo `apalike` a otro en la instrucción `\bibliographystyle{apalike}`.

Y, para terminar este capítulo, resume brevemente qué vas a contar en los siguientes.

7.7. Corrector ortográfico

Una vez tengas todo, no olvides pasar el corrector ortográfico de L^AT_EXa todos tus ficheros `.tex`. En Windows, el propio editor TeXworks incluye el corrector. En Linux, usa `aspell` ejecutando el siguiente comando en tu terminal:

```
aspell --lang=es --mode=tex check capitulo1.tex
```

Capítulo 8

Conclusiones

La mente lo es todo. En lo que piensas, te conviertes

Buda

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo, que básicamente será una recapitulación de los problemas que has abordado, las soluciones que has prouesto, así como los experimentos llevados a cabo para validarlos. Y con esto, cierras la memoria.

8.1. Conclusiones

Enumera los objetivos y cómo los has cumplido.

Enumera también los requisitos implícitos en la consecución de esos objetivos, y cómo se han satisfecho.

No olvides dedicar un par de párrafos para hacer un balance global de qué has conseguido, y por qué es un avance respecto a lo que tenías inicialmente. Haz mención expresa de alguna limitación o peculiaridad de tu sistema y por qué es así. Y también, qué has aprendido desarrollando este trabajo.

Por último, añade otro par de párrafos de líneas futuras; esto es, cómo se puede continuar tu trabajo para abarcar una solución más amplia, o qué otras ramas de la investigación podrían seguirse partiendo de este trabajo, o cómo se podría mejorar para conseguir una aplicación real de este desarrollo (si es que no se ha llegado a conseguir).

8.2. Corrector ortográfico

Una vez tengas todo, no olvides pasar el corrector ortográfico de L^AT_EXa todos tus ficheros *.tex*. En Windows, el propio editor TeXworks incluye el corrector. En Linux, usa `aspell` ejecutando el siguiente comando en tu terminal:

```
aspell --lang=es --mode=tex check capitulo1.tex
```

Capítulo 9

Anexo

Las ideas no duran mucho. Hay que hacer algo con ellas

Santiago Ramón y Cajal

En este anexo se van a tratar los pasos necesarios para conseguir replicar este proyecto.

9.1. Construcción del robot

9.2. Ejecución del robot en simulación

Es necesario instalar los siguientes programas:

```
sudo apt update && sudo apt upgrade
sudo apt install ros-humble-ros2-control ros-humble-ros2-controllers
sudo apt install ros-humble-rviz2
sudo apt install ros-humble-gazebo-ros-pkgs
sudo apt install ros-humble-xacro ros-humble-robot-state-publisher
sudo apt install ros-humble-joint-state-publisher
```

Una vez instalado los programas, es el momento de ponerlo en ejecución y para ello, únicamente hay que escribir los siguientes comandos:

```
colcon build --packages-select pibotj_r2c    # compila los paquetes
source ./install/setup.bash                  # configura variables
ros2 launch pibotj_r2c launch_sim.launch.py
```

Si la primera vez que se lance el robot ocurre algún error, es normal y hay que reiniciar el proceso.

9.3. Configuración del robot real

En este apartado es importante instalar Ubuntu 20.04 y ROS 2 Foxy en vez de simulación que fue Ubuntu 22.04 y ROS2 Humble (como se puede ver en el apartado anterior).

Se va a trabajar conectándose por ssh al robot, escribiendo `ssh usuario@ip_dispositivo` y posteriormente, introduciendo la contraseña creada en el proceso de instalación. Esta era la forma de comunicación con el robot ya que esta distribución, no tenía interfaz gráfica. De las variantes de ssh se usaba scp -r para copiar directorios entre el robot y el ordenador local y permitir subir los cambios a Github, y para desarrollar código se usaba un *plugin* de VSCode que permite usar VSCode conectándose al robot usando ssh¹.

En los siguientes apartados se detalla cómo se ha configurado cada dispositivo que conforma el PiBotJ.

9.3.1. Cámara

La cámara está conectada al puerto CSI y, para hacerla funcionar, se necesitan los siguientes programas:

```
sudo apt-get install gstreamer1.0-tools \
gstreamer1.0-plugins-base gstreamer1.0-plugins-good \
gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly
```

Una vez instalados, fue necesario editar el fichero `/boot/firmware/config.txt` añadiendo `start_x=1` al final y se tuvo que reiniciar la Raspberry Pi. Para comprobar que la configuración fue correcta, era necesario usar el comando `v4l2-ctl --list-devices` que comprueba la lista de dispositivos de video y media que están disponibles en el sistema. En la Figura 9.1 se puede ver el resultado de dicho comando y muestra que la cámara se encontraba conectada en el dispositivo `/dev/video0`.

Sabiendo que la cámara estaba disponible, era necesario comprobar que realmente la imagen se pudiera ver correctamente.

¹<https://code.visualstudio.com/docs/remote/ssh-tutorial>

```
julioau@raspberrypi:~$ v4l2-ctl --list-devices
bcm2835-codec-decode (platform:bcm2835-codec):
    /dev/video10
    /dev/video11
    /dev/video12
    /dev/media1

bcm2835-isp (platform:bcm2835-isp):
    /dev/video13
    /dev/video14
    /dev/video15
    /dev/video16
    /dev/media0

mmal service 16.1 (platform:bcm2835-v4l2):
    /dev/video0
```

Figura 9.1: Dispositivos de video y media disponibles en PiBotJ

9.3.2. Google Coral

Se pueden encontrar diferentes pasos de instalación dependiendo de las necesidades del usuario, pero se pueden resumir para cumplir con las características de la arquitectura de este dispositivo de la siguiente manera:

```
echo "deb https://packages.cloud.google.com/apt \
coral-edgetpu-stable main" | \
sudo tee /etc/apt/sources.list.d/coral-edgetpu.list

curl https://packages.cloud.google.com/apt/doc/apt-key.gpg \
| sudo apt-key add -

sudo apt-get update
sudo apt-get install libedgetpu1-std

# Conecta el USB en uno de los puertos 3.0

sudo apt-get install python3-pycoral
```

Finalmente, el proceso se debería haber completado cuando la salida del último comando se debería ver como muestra la Figura 9.2.

9.3.3. Módulo GPS

La estructura seguida en este tutorial² ha sido de gran ayuda para configurar el módulo GPS que, para esta arquitectura, supone seguir los siguientes pasos:

Crea el fichero `/etc/udev/rules.d/99-ttyAMA0.rules` que tiene que contener:

²<https://sparklers-the-makers.github.io/blog/robotics/use-neo-6m-module-with-raspberry-pi/>

```
julioau@raspberrypi: ~ $ sudo apt-get install python3-pycoral
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  python3-tflite-runtime
The following NEW packages will be installed:
  python3-pycoral python3-tflite-runtime
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 3,819 kB of archives.
After this operation, 16.5 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 https://packages.cloud.google.com/apt coral-edgetpu-stable/main arm64 python3-tflite-runtime arm64 2.5.0.post1 [3,014 kB]
Get:2 https://packages.cloud.google.com/apt coral-edgetpu-stable/main arm64 python3-pycoral arm64 2.0.0 [804 kB]
Fetched 3,819 kB in 2s (1,741 kB/s)
Selecting previously unselected package python3-tflite-runtime.
(Reading database ... 207067 files and directories currently installed.)
Preparing to unpack .../python3-tflite-runtime_2.5.0.post1_arm64.deb ...
Unpacking python3-tflite-runtime (2.5.0.post1) ...
Selecting previously unselected package python3-pycoral.
Preparing to unpack .../python3-pycoral_2.0.0_arm64.deb ...
Unpacking python3-pycoral (2.0.0) ...
Setting up python3-tflite-runtime (2.5.0.post1) ...
Setting up python3-pycoral (2.0.0) ...
julioau@raspberrypi: ~
```

Figura 9.2: Configuración exitosa del Google Coral

`KERNEL=="ttyAMA0", MODE=="0666", GROUP="dialout"` y para que se actualicen en el sistema es necesario recargar las reglas, escribiendo por terminal lo siguiente:

```
sudo udevadm control --reload-rules
sudo udevadm trigger
```

Como esta distribución no tiene interfaz web, no tiene instalado por defecto `raspi-config`. Así, para su instalación se ha seguido el siguiente tutorial³ y, una vez instalado, hay que ejecutarlo usando `sudo raspi-config`. Dentro de `raspi-config` hay que desplazarse hasta *Interfacing options* y *serial*, hay que desabilitar *serial login shell*, habilitar *serial interface* y reiniciar con `sudo reboot`.

Finalmente los siguientes ficheros tienen que contener la siguiente información:

```
cat /boot/firmware/config.txt

# Please DO NOT modify this file; if you need to modify the boot config,
# the "usercfg.txt" file is the place to include user changes. Please
# refer to the README file for a description of the various configuration
# files on the boot partition.

# The unusual ordering below is deliberate; older firmwares (in particular
# the version initially shipped with bionic) don't understand the
# conditional
# [sections] below and simply ignore them. The Pi4 doesn't boot at all
# with firmwares this old so it's safe to place at the top. Of the Pi2 and
# Pi3, the Pi3 uboot happens to work happily on the Pi2, so it needs to go
```

³https://github.com/EmilGus/install_raspi-config/tree/master

```
# at the bottom to support old firmwares.

[pi4]
kernel=uboot_rpi_4.bin

[pi2]
kernel=uboot_rpi_2.bin

[pi3]
kernel=uboot_rpi_3.bin

[pi0]
kernel=uboot_rpi_3.bin

[all]
device_tree_address=0x03000000

[pi4]
max_framebuffers=2
arm_boost=1

[all]
# Enable the audio output, I2C and SPI interfaces on the GPIO header. As
# these
# parameters related to the base device-tree they must appear *before* any
# other dtoverlay= specification
dtparam=audio=on
dtparam=i2c_arm=on
dtparam=spi=on

# Comment out the following line if the edges of the desktop appear
# outside
# the edges of your display
disable_overscan=1

# If you have issues with audio, you may try uncommenting the following
# line
# which forces the HDMI output into HDMI mode instead of DVI (which
# doesn't
# support audio output)
#hdmi_drive=2

# Config settings specific to arm64
arm_64bit=1
dtoverlay=dwc2

[cm4]
# Enable the USB2 outputs on the IO board (assuming your CM4 is plugged
# into
# such a board)
dtoverlay=dwc2,dr_mode=host
```

[all]

```
# The following settings are "defaults" expected to be overridden by the
# included configuration. The only reason they are included is, again, to
# support old firmwares which don't understand the "include" command.
```

```
enable_uart=1
cmdline=cmdline.txt
```

```
include syscfg.txt
include usercfg.txt
```

```
start_x=1
```

```
cat /boot/firmware/cmdline.txt
```

```
elevator=deadline net.ifnames=0 dwc_otg.lpm_enable=0 root=LABEL=writable \
rootfstype=ext4 rootwait fixrtc quiet splash cfg80211.ieee80211_regdom=GB
```

```
cat /boot/firmware/usercfg.txt
```

```
# Place "config.txt" changes (dtparam, dtoverlay, disable_overscan, etc.)
# in
# this file. Please refer to the README file for a description of the
# various
# configuration files on the boot partition.
dtoverlay=pi3-disable-bt
```

```
cat /boot/firmware/syscfg.txt
```

```
# This file is intended to be modified by the pibootctl utility. User
# configuration changes should be placed in "usercfg.txt". Please refer
# to the
# README file for a description of the various configuration files on the
# boot
# partition.
```

```
enable_uart=1
dtparam=audio=on
dtparam=i2c_arm=on
dtparam=spi=on
cmdline=cmdline.txt
```

9.3.4. Módulo GPS

Hay que instalar los siguientes programas⁴:

```
sudo apt-get update  
sudo apt-get install rpi.gpio-common python3-pigpio python3-gpiozero \  
python3-rpi.gpio
```

Para asegurarnos que funcionaban perfectamente habia que añadir *dialout* a *groups*.

⁴<https://cms.gutow.uwosh.edu/Gutow/useful-chemistry-links/software-tools-and-coding/computer-and-coding-how-tos/allowing-access-to-gpio-i2c-and-spi-on-pi-under-ubuntu-20.04>

Bibliografía

- [Ahmed et al., 2022] Ahmed, A., Ashfaque, M., Ulhaq, M. U., Mathavan, S., Kamal, K., and Rahman, M. (2022). Pothole 3d reconstruction with a novel imaging system and structure from motion techniques. *IEEE Transactions on Intelligent Transportation Systems*, 23(5):4685–4694.
- [Allouch et al., 2017] Allouch, A., Koubâa, A., Abbes, T., and Ammar, A. (2017). Roadsense: Smartphone application to estimate road conditions using accelerometer and gyroscope. *IEEE Sensors Journal*, 17(13):4231–4238.
- [Barceló, 2004] Barceló, M. (2004). De nuevo los robots. *Paradojas*, (64):1–2.
- [Bidaud, 2017] Bidaud, P. (2017). Les robots et les hommes. page 40.
- [Bruno et al., 2023] Bruno, S., Loprencipe, G., Di Mascio, P., Cantisani, G., Fiore, N., Polidori, C., D’Andrea, A., and Moretti, L. (2023). A robotized raspberry-based system for pothole 3d reconstruction and mapping. *Sensors*, 23(13).
- [Cuenca Sánchez et al., 2023] Cuenca Sánchez, A., Farinango Galeano, W., and Murillo Zambrano, J. (2023). Diseño de un sistema de generación microhidráulica basado en un tornillo de arquímedes. *Ingenius. Revista de Ciencia y Tecnología*, (29):98–107.
- [Dhiman and Klette, 2020] Dhiman, A. and Klette, R. (2020). Pothole detection using computer vision and learning. *IEEE Transactions on Intelligent Transportation Systems*, 21(8):3536–3550.
- [Du et al., 2020] Du, R., Qiu, G., Gao, K., Hu, L., and Liu, L. (2020). Abnormal road surface recognition based on smartphone acceleration sensor. *Sensors*, 20(2).
- [Earnest, 2012] Earnest, L. (2012). The stanford cart. Consultado el 19 de julio de 2024.

- [El Zaatari et al., 2019] El Zaatari, S., Marei, M., Li, W., and Usman, Z. (2019). Cobot programming for collaborative industrial tasks: An overview. *Robotics and Autonomous Systems*, 116:162–180.
- [Giri, 2020] Giri, L. A. (2020). Máquinas térmicas desde la antigüedad al siglo xvii: análisis histórico desde la filosofía de la técnica.
- [Hansun, 2013] Hansun, S. (2013). A new approach of moving average method in time series analysis. In *2013 Conference on New Media Studies (CoNMedia)*, pages 1–4.
- [Hartley and Zisserman, 2004] Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition.
- [Katsamenis et al., 2022] Katsamenis, I., Bimpas, M., Protopapadakis, E., Zafeiropoulos, C., Kalogerias, D., Doulamis, A., Doulamis, N., Martín-Portugués Montoliu, C., Handanos, Y., Schmidt, F., Ott, L., Cantero, M., and Lopez, R. (2022). Robotic maintenance of road infrastructures: The heron project. In *Proceedings of the 15th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '22, page 628–635, New York, NY, USA. Association for Computing Machinery.
- [Kim et al., 2022] Kim, Y.-M., Kim, Y.-G., Son, S.-Y., Lim, S.-Y., Choi, B.-Y., and Choi, D.-H. (2022). Review of recent automated pothole-detection methods. *Applied Sciences*, 12(11).
- [Llopis Castelló and Pérez Zuriaga, 2020] Llopis Castelló, D. and Pérez Zuriaga, A. M. (2020). Deterioros en pavimentos urbanos.
- [moinfaisal, 2023] moinfaisal (2023). Pothole detection project new dataset. <https://universe.roboflow.com/moinfaisal/pothole-detection-project-new>. visited on 2024-10-26.
- [Nilsson et al., 1984] Nilsson, N. J. et al. (1984). *Shakey the robot*, volume 323. Sri International Menlo Park, California.
- [Ortiz et al., 2017] Ortiz, L., Gonçalves, L., and Cabrera, E. (2017). A generic approach for error estimation of depth data from (stereo and rgb-d) 3d sensors.
- [Park et al., 2021] Park, S.-S., Tran, V.-T., and Lee, D.-E. (2021). Application of various yolo models for computer vision-based real-time pothole detection. *Applied Sciences*, 11(23).

- [Plaza, 2023] Plaza, J. M. C. (2022-2023). Robótica de servicio: Robots de limpieza. Consultado el 19 de julio de 2024.
- [Shaw and Pathak, 2024] Shaw, K. and Pathak, D. (2024). LEAP hand v2: Dexterous, low-cost anthropomorphic hybrid rigid soft hand for robot learning. In *2nd Workshop on Dexterous Manipulation: Design, Perception and Control (RSS)*.
- [Thorpe and Durrant-Whyte, 2003] Thorpe, C. and Durrant-Whyte, H. (2003). Field robots. In *Robotics Research: The Tenth International Symposium*, pages 329–340. Springer.
- [Ul Haq et al., 2019] Ul Haq, M. U., Ashfaque, M., Mathavan, S., Kamal, K., and Ahmed, A. (2019). Stereo-based 3d reconstruction of potholes by a hybrid, dense matching scheme. *IEEE Sensors Journal*, 19(10):3807–3817.
- [Vega and Cañas, 2018] Vega, J. and Cañas, J. (2018). PiBot: An open low-cost robotic platform with camera for STEM education. *Electronics*, 7:430–446.
- [Vega and Cañas, 2019] Vega, J. and Cañas, J. (2019). Open vision system for low-cost Robotics education. *Electronics*, 8:1295–1315.
- [Wang et al., 2023] Wang, N., Dong, J., Fang, H., Li, B., Zhai, K., Ma, D., Shen, Y., and Hu, H. (2023). 3d reconstruction and segmentation system for pavement potholes based on improved structure-from-motion (sfm) and deep learning. *Construction and Building Materials*, 398:132499.
- [Wanli Ye and Xiao, 2021] Wanli Ye, Wei Jiang, Z. T. D. Y. and Xiao, J. (2021). Convolutional neural network for pothole detection in asphalt pavement. *Road Materials and Pavement Design*, 22(1):42–58.