UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Escuela de Ingenierías Industriales

Ingeniería de Sistemas y Automática

Trabajo Fin de Grado

Diseño e implementación del sistema de control de un vehículo robótico de orugas articuladas

Design and implementation of the control system for an articulated tracked robotic vehicle

Grado en Ingeniería Electrónica Industrial

Autor: Mario García Jiménez

Tutor: Antonio José Muñoz Ramírez

Málaga, Mayo de 2025

UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Diseño e implementación del sistema de control de un vehículo robótico de orugas articuladas

- **Autor:** Mario García Jiménez.
- **Tutor:** Antonio José Muñoz Ramírez.
- **Departamento:** Ingeniería de Sistemas y Automática.
- **Titulación:** Grado en Ingeniería Electrónica Industrial.

- **Palabras clave** Robot, Motor CC, Matlab, Simulink, CAN, Control de vehículo, Modelado de Sistemas.

Resumen

El proyecto se enmarca dentro del equipo RoboRescue UMA, cuyo fin es la robótica de rescate para asistir a profesionales en situaciones de emergencia, en donde hay vidas en juego y se necesita la ayuda de una máquina para solventar de la mejor manera el riesgo.

El presente Trabajo de Fin de Grado trata de abarcar toda la construcción, diseño, modelado e implementación del sistema físico de un robot todoterreno. El movimiento del robot se realiza en base a un sistema de cuatro orugas basculantes controlado por un ordenador, junto con dos placas con microcontroladores, como son el Arduino MKR WiFi 1010 y el ESP32 de Espressif Systems.

El robot contiene en cada una de las cuatro orugas, un servomotor y un motor paso a paso. Los servomotores son sin escobillas y de la marca MyActuator, modelo RMD-X8-S2-V3; poseen un bus CAN para el control del movimiento de tracción del vehículo, implementado en el Arduino MKR WiFi 1010.

Los cuatro motores del tipo paso a paso de la marca Walfront, controlados mediante el ESP32, hacen que las orugas basculen sobre uno de los extremos, produciendo el movimiento de elevación. En el control de posición de las orugas, se usan sensores angulares absolutos de efecto Hall AS5600 (encoders), que permiten obtener el ángulo de la oruga.

El robot se opera con un mando inalámbrico que interacciona con el ordenador de abordo, encargado de traducir las órdenes del usuario a las placas controladoras. Destacando como una de las órdenes más complejas, el nivelado automático de la plataforma gracias a un sensor inercial IMU MPU9250.

UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Design and implementation of the control system

for an articulated tracked robotic vehicle

- **Author:** Mario García Jiménez.
- **Tutor:** Antonio José Muñoz Ramírez.
- **Department:** Systems and Automatic Engineering.

- **Degree:** Industrial Electronics Engineering Degree.
- **Keywords:** Robot, DC Motor, Matlab, Simulink, CAN, Vehicle Control, System Modeling, Arduino.

Abstract

The project is part of the RoboRescue UMA team, whose mission is to provide rescue robotics to assist professionals in emergency situations where lives are at stake and the help of a machine is needed to best address the risk.

This Final Degree Project aims to cover the entire construction, design, modeling, and implementation of the physical system of an all-terrain robot. The robot's movement is based on a system of four tilting tracks controlled by a computer, along with two microcontroller boards, the Arduino MKR WiFi 1010 and the ESP32 from Espressif Systems.

The robot contains a servomotor and a stepper motor on each of its four tracks. The servomotors are brushless and from the MyActuator brand, model RMD-X8-S2-V3; They have a CAN bus for controlling the vehicle's traction motion, implemented in the Arduino MKR WiFi 1010.

The four Walfront stepper motors, controlled by the ESP32, cause the tracks to tilt on one end, producing the lifting motion. Track position control uses AS5600 Hall effect absolute angle sensors (encoders), which allow the track angle to be obtained.

The robot is operated with a wireless remote control that interacts with the on-board computer, which translates user commands to the control boards. One of the most complex commands is the automatic leveling of the platform thanks to an MPU9250 IMU inertial sensor.

La mejor forma de predecir el futuro es inventarlo.

Alan Kay

Agradecimientos

Agradecer a mi tutor Antonio, por su tiempo y ayuda durante el desarrollo de este proyecto.

A mis padres y mi hermana por su apoyo en lo personal y motivación.

A Nazareth por su confianza y su apoyo constante.

Y al equipo de RoboRescue UMA.

Glosario de Términos

Software Conjunto de instrucciones que se dan a un dispositivo electrónico para que ejecute unas acciones previstas.

Hardware Partes físicas (tangibles) de un dispositivo electrónico.

Firmware Tipo de software que se programa en la lectura de memoria de un dispositivo, encargado de cumplir las funciones básicas del mismo.

Robot Sistema electromecánico (o simulación del mismo) reprogramable que realiza un trabajo específico.

Microcontrolador Circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria.

Sistema de control Serie de instrucciones lógicas utilizadas para obtener unas señales de salida, generalmente instrucciones a seguir, de un sistema a partir de unas entradas dadas.

Feedback Proceso de control por el que una parte de la salida vuelve a entrar como una entrada al sistema en la siguiente iteración.

bit Unidad mínima de memoria en computación, que puede valer 0 o 1.

byte Conjunto de 8 bits, expresable mediante 8 números binarios o 2 hexadecimales.

Acrónimos

CC o DC	Corriente Continua
CAN	Controller Area Network
I2C	Inter Integrated Circuit
SPI	Serial Peripheral Interface
MCU	Microcontroller Unit
IMU	Inertial Measurement Unit
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
ASCII	American Standard Code for Information Interchange
PC	Personal Computer
PWM	Pulse Width Modulation
WiFi	Wireless Fidelity
UDP	User Datagram Protocol
XCP	Universal Measurement and Calibration Protocol

Índice general

1. Introducción **13**

1.1. Introducción 13

1.2. Objetivos 16

1.3. ¿Por qué es interesante este trabajo? 16

1.4. Contenido de la Memoria 17

2. Componentes de Hardware **19**

2.1. Introducción 19

2.2. Motores 20

2.2.1. Motores RMD X8 S2 V3 20

2.2.2. Motores Paso a Paso 23

2.3. Microcontroladores 26

2.3.1. Microcontrolador Arduino MKR WiFi 1010 26

2.3.2. Microcontrolador ESP32 DEVKIT V1 28

2.4. Periféricos 30

2.4.1. Multiplexor I2C TCA9548A 8CH 30

2.4.2. Sensor de posición rotatorio magnético AS5600 31

2.4.3. IMU MPU9250 36

2.5. Batería 37

2.5.1. Introducción 37

2.5.2. Materiales usados 38

2.6. Esquemático de la batería 40

2.6.1. Construcción de la batería 42

3. Esquema eléctrico del sistema de control 49

3.1. Introducción 49

3.2. Asignación de pines y conexiones 49

4. Componentes Software 55

4.1. Introducción 55

4.2. MATLAB 55

4.3. Arduino IDE 58

5. Sistema de control del vehículo robótico 61

5.1. Introducción 61

5.2. Sistema de comunicaciones 63

5.3. Modos de funcionamiento del robot 64

5.4. Funciones del PC 65

5.5. Funciones de los microcontroladores 65

5.5.1. Arduino MKR WiFi 1010 65

5.5.2. ESP32 66

6. Programación del control PC 67

6.1. Introducción 67

6.2. Vista general del modelo 68

6.3. Adquisión de datos 69

6.4. Procesamiento de las entradas 70

6.5. Transmisión de datos 78

7. Programación del control Arduino MKR WiFi 1010 79

7.1. Introducción 79

7.2. Vista general del modelo 79

7.3. Recepción de datos por WiFi 80

7.4. Procesamiento del vector de datos 81

7.4.1. Datos recibidos del PC 82

7.4.2. Datos recibidos del ESP32 83
7.5. Control de motores 86
7.5.1. Procesamiento del vector data 87
7.5.2. Subsistema del modo de funcionamiento 1 88
7.5.3. Subsistema del modo de funcionamiento 2. Control de posición absoluta 92
7.5.4. Subsistema del modo de funcionamiento 3 y 4. Control incremental . 105
7.5.5. Subsistema del modo de funcionamiento 5. Nivelado automático . . . 110
7.6. Envío y recepción de datos de los motores de tracción 114
7.6.1. Envío de mensajes CAN 114
7.6.2. Lectura de mensajes CAN 116
7.6.3. Procesamiento de errores 118

8. Programación del control ESP32 121
8.1. Introducción 121
8.2. Librerías usadas 122
8.3. Variables globales, locales y funciones usadas 123

9. Conclusiones 135
9.1. Introducción 135
9.2. Conclusiones 135
9.3. Impacto personal del TFG 137
9.4. Ampliaciones futuras 137

Bibliografía 139

A. Modelo del control PC versión simplificada 141

Índice de figuras

1.1. Equipo RoboRescue UMA con robot Donatello y Horu 14
1.2. FUHGA3 14
1.3. MARKHOR 15

2.1. Motor de tracción modelo RMD X8 S2 V3 20
2.2. Ejemplo de mensaje CAN 21
2.3. Soldadura de cables CAN bus en modo estrella 22
2.4. Prueba donde se aprecia un primer plano de la Shield CAN conectada a los cables CAN del motor de tracción de una oruga 22
2.5. Motor paso a paso Walfront 23
2.6. Driver DM542 24
2.7. Motor paso a paso y reductora 1:80 24
2.8. Tabla sobre la configuración de los microsteps 25
2.9. Arduino MKR WiFi 1010 26
2.10. Shield CAN para Arduino MKR WiFi 1010 27
2.11. Arduino MKR WiFi 1010 placa prototipado 27
2.12. ESP32 28
2.13. Placa de prototipado ESP32 29
2.14. Placas de prototipado de Arduino MKR WiFi 1010 y ESP32 ancladas al chasis del robot 29
2.15. Multiplexor modelo TCA9548A 8CH 30
2.16. Sensor de posición AS5600 31
2.17. Variación del RAW ANGLE en función de la posicion del imán 32

2.18. Diseño 3D del soporte para el imán . . . 33
2.19. Proceso de soldadura de los encoders . . . 33
2.20. Diseño 3D del soporte del sensor AS5600 . . . 34
2.21. Codificadores en soporte de impresión 3D . . . 34
2.22. Diseño 3D de la oruga completa y soporte para encoder . . . 35
2.23. Pruebas del DM542 para determinar la configuración óptima de step/rev usando como controlador el ESP32 . . . 35
2.24. IMU MPU9250 . . . 36
2.25. Batería del robot Horu basada en celdas LiFePO4 32700 . . . 37
2.26. Imagen donde se ve el soldador por puntos usado y un paquete de 12 celdas 38
2.27. Celda LiFePO4 32700 3.2V 6000mAh . . . 38
2.28. Tira de Níquel usada para soldar las celdas entre sí . . . 39
2.29. Soldador por puntos . . . 40
2.30. Esquemático de la batería . . . 40
2.31. Gráfica de descarga de la celda 32700 3.2V . . . 41
2.32. Gráfica de carga de la celda 32700 3.2V . . . 42
2.33. Unión de terna de celdas . . . 42
2.34. Conjunto de celdas unidas . . . 43
2.35. Conexión del BMS con las celdas . . . 43
2.36. Interfaz de la aplicación Smart BMS . . . 45
2.37. BMS y pantalla del BMS que indican el estado de la batería . . . 45
2.38. Proceso de sellado de la batería mediante termoretráctil . . . 46
2.39. Cargador de la batería . . . 47
2.40. Cargador de la batería en proceso de carga . . . 47
2.41. Batería posicionada en la parte baja del chasis . . . 48

3.1. Esquema eléctrico del sistema de control de Horu . . . 49
3.2. Conexión de driver DM542 de la oruga F.L. con ESP32 . . . 51
3.3. Parte delantera del multiplexor . . . 51
3.4. Parte trasera del multiplexor . . . 51

4.1. MATLAB . . . 55
4.2. Ejemplo del entorno Simulink . . . 57

4.3. Modo Run en Simulink 57
4.4. Modo Build Deploy Start y Monitor & Tune en Simulink 57
4.5. Arduino 58
4.6. Entorno del Arduino IDE 59
4.7. Selección de la placa microcontroladora ESP32 59

5.1. Controles del mando de Xbox 62
5.2. Diagrama del sistema de comunicaciones de Horu 63

6.1. Modelo del Control PC 68
6.2. Subsistema Xinput Controller 69
6.3. S-Function Xbox 69
6.4. Subsistemas encargados del procesamiento de las entradas 70
6.5. Subsistema encargado de la adquisición de consignas de movimiento de los motores de tracción 71
6.6. Interior del subsistema de movimiento del modo de tracción 71
6.7. Subsistemas encargados del procesamiento de la variable Y y del modo de operación 72
6.8. Interior del subsistema del procesamiento de los triggers 73
6.9. Chart principal del control PC 73
6.10. Subsistema del estado de conexión 77
6.11. Interior del subsistema de estado de conexión 77
6.12. Interior del subsistema *UDP Send* 78
6.13. Subsistema activo con enable mediante un detector de cambio 78

7.1. Modelo del control Arduino MKR WiFi 1010 79
7.2. Recepción de datos WiFi 80
7.3. Subsistema activo por cambio de entrada de datos recibidos 81
7.4. Interior del subsistema de procesamiento de data_raw vector 82
7.5. Datos recibidos desde el PC 82
7.6. Datos recibidos del ESP32 por Arduino MKR WiFi 1010 83
7.7. Sistema de referencia de la oruga frontal derecha 84
7.8. Subsistema de ajuste de ángulos 85
7.9. Control de modos de movimiento en Arduino MKR WiFi 1010 86

7.10. Subsistema data_process . . . 87
7.11. Subsistema de control de tracción diferencial . . . 88
7.12. Subsistema de conversión de grados/segundo a mensaje CAN . . . 90
7.13. Bloque de escritura del mensaje *CAN* para la oruga trasera izquierda . . . 90
7.14. Sistema de referencia del robot . . . 92
7.15. Subsistema del modo de funcionamiento 2 . . . 93
7.16. Subsistema de conversión de vectores de posición . . . 93
7.17. Interior del subsistema de conversión de vectores de posición . . . 94
7.18. Subsistema de envío de datos al ESP32 . . . 95
7.19. Interior del subsistema de envío de datos hacia el ESP32 . . . 95
7.20. Subsistemas para el control de la compensación de los motores de tracción . 96
7.21. Sistema de referencia de la oruga frontal derecha . . . 97
7.22. Esquema para el estudio de la compensación . . . 98
7.23. Chart encargado de la selección de la actuación para la compensación de la oruga F.R. . . . 100
7.24. Máquina de estados para el control de la compensación de la oruga F.R. . . . 101
7.25. Case0 . . . 102
7.26. Case1 . . . 102
7.27. Case2 . . . 102
7.28. Case3 . . . 103
7.29. Case4 . . . 103
7.30. Subsistema activo por cambio de entrada de la oruga R.L. . . . 104
7.31. Interior del subsistema CAN TRANSMIT RL . . . 104
7.32. Horu situado a 225º . . . 104
7.33. Horu situado a 135º . . . 104
7.34. Sistema de referencia de sentidos de giro horarios, respecto de la oruga F.R. 105
7.35. Subsistema del modo de control incremental de posición . . . 105
7.36. Subsistemas encargados del envío de datos al ESP32 . . . 106
7.37. Subsistemas encargados de la compensación en el modo de control incremental de posición . . . 107
7.38. Caso de la oruga F.R. en el control incremental de posición . . . 108
7.39. Horu en el modo de posición incremental . . . 109
7.40. Sistema de referencia de ángulos Roll y Pitch respecto de la IMU . . . 110

7.41. Subsistemas encargados del nivelado automático . . . 110
7.42. Subsistema encargado de la adquisición de ángulos . . . 111
7.43. Algoritmo de adquisición de ángulos de inclinación roll y pitch (interior subsistema Gyro & Accel.) . . . 112
7.44. Subsistemas encargados del filtrado y cálculo de los ángulos roll y pitch . . . 112
7.45. Subsistema encargado del envío de mensajes CAN hacia los motores de tracción . . . 114
7.46. Interior del subsistema *Send CAN msg* . . . 114
7.47. Subsistema de envío de mensaje CAN a la oruga F.L. . . . 115
7.48. Recepción de datos CAN provenientes de los motores de tracción . . . 116
7.49. Bloque CAN Receive . . . 116
7.50. Subsistema encargado del procesamiento del mensaje CAN recibido . . . 117
7.51. Subsistema encargado de mostrar códigos de errores . . . 118
7.52. Subsistema para el control Watchdog en Simulink . . . 119
7.53. Bloques watchdog PC y ESP32 . . . 119

8.1. Diagrama del conexionado del ESP32 . . . 121
8.2. Tabla de parámetros del driver DM542 de los motores paso a paso . . . 129

A.1. Subsistemas del control PC con tres estados . . . 143
A.2. Diagrama de estados del control PC simplificado . . . 144

Índice de tablas

3.1. Tabla de conexionado de la IMU, multiplexor, drivers DM542 y CAN bus motores de tracción 52
3.2. Tabla de conexionado de los sensores AS5600 53

6.1. Datagrama de datos recibidos del PC por Arduino MKR WiFi 1010 67

7.1. Datagrama de datos recibidos del PC por Arduino MKR WiFi 1010 83
7.2. Datagrama de datos recibidos del ESP32 por Arduino MKR WiFi 1010 84

Capítulo 1

Introducción

1.1 Introducción

El propósito de este Trabajo de Fin de Grado es demostrar los conocimientos adquiridos en el Grado de Ingeniería Electrónica Industrial, así como su aplicación en la resolución de problemas en situaciones reales, en un campo actual y con gran potencial como es la Robótica de Rescate.

El sector de la Robótica de Rescate avanza hacia una mayor autonomía y versatilidad en diversos tipos de terrenos, lo cual es crucial dada la creciente frecuencia de catástrofes naturales y emergencias domésticas. Este campo proporciona una visión objetiva en entornos peligrosos e inaccesibles para los seres humanos, impulsando soluciones innovadoras. Por tanto, este Trabajo de Fin de Grado se centra en desarrollar una solución efectiva para estos escenarios críticos, utilizando tecnología robótica avanzada.

El robot que ha servido como plataforma de desarrollo para este Trabajo de Fin de Grado pertenece al equipo RoboRescue UMA (ver figura 1.1), un grupo de estudiantes enfocado en diseñar y construir robots destinados a competir en la RoboCup Rescue Robot League y en cualquier ámbito de robótica.

La RoboCup Rescue Robot League es una competición internacional integrada en el evento global RoboCup. Esta categoría tiene como objetivo fomentar el diseño, desarrollo y evaluación de robots, tanto autónomos como teleoperados, capaces de operar en escenarios de rescate tras desastres naturales o situaciones de emergencia.

En los últimos años, se observa una tendencia creciente hacia el uso de robots con orugas articuladas. Un ejemplo destacado es el FUHGA 3 (ver figura 1.2), desarrollado por el equipo SHINOBI de la Universidad de Kyoto, Japón. Este robot incorpora orugas en casi toda su superficie inferior, lo que le permite superar obstáculos de gran altura. Además, está equipado con un brazo robótico de seis grados de libertad. Otro aspecto destacable es su bajo peso en comparación con otros modelos de prestaciones similares, como el MARKHOR, que se describe más adelante.

Figura 1.1: Equipo RoboRescue UMA con robot Donatello y Horu

Figura 1.2: FUHGA3

Por su parte, el MARKHOR (ver figura 1.3) es un robot desarrollado por el equipo Capra Club de la Escuela Técnica Superior de Montreal, diseñado específicamente para misiones de búsqueda y rescate dentro de la RoboCup Rescue Robot League. Entre sus características más relevantes destacan sus orugas denominadas “aletas” por su particular forma y disposición. Estas orugas suelen estar elevadas y sólo una pequeña porción entra en contacto con el suelo, lo que reduce la fricción, mejora la capacidad de giro y permite desplazarse por espacios estrechos con mayor agilidad. Además, el MARKHOR está equipado con sensores de distancia por infrarrojos y una cámara WiFi que permite su operación a distancia mediante control remoto.

En la actualidad, los robots aplican métodos de control implementados en microcon-

Figura 1.3: MARKHOR

troladores de fabricantes como Arduino, Texas Instruments, Microchip o Espressif Systems; dichos dispositivos, a su vez, suelen estar supervisados y coordinados por un ordenador compacto que centraliza el procesamiento y la comunicación con los distintos módulos del sistema.

Los robots actuales suelen equiparse con una amplia variedad de sensores para garantizar un control preciso y una percepción completa de su entorno: en primer lugar, incorporan sensores de posición (encoders) ópticos o magnéticos y, en ocasiones, potenciómetros, que proporcionan retroalimentación de posición y velocidad en ejes y ruedas; asimismo, incluyen unidades de medición inercial (IMU), que integran acelerómetros, giróscopos y magnetómetros, para estimar la orientación y la aceleración, fundamentales en la estabilización y navegación autónoma. Para la detección de obstáculos y el mapeado del entorno se recurre a sensores de ultrasonidos e infrarrojos, así como LiDAR, que ofrece nubes de puntos de alta resolución a distancias mayores.

En robótica de rescate y móvil, las baterías han evolucionado hacia químicas de litio de alta seguridad (LiFePO4) y formulaciones de estado sólido, que ofrecen mayor ciclo de vida y estabilidad térmica.

1.2 Objetivos

Este Trabajo de Fin de Grado trata del diseño e implementación del sistema de control de un vehículo robótico de orugas articuladas. El robot se mueve gracias a cuatro orugas articuladas, cada una de ellas con dos motores (uno de tracción y uno de elevación), por lo que permitirá una gran amplitud de movimientos que se pueden programar. Para programar los motores se usarán dos placas con microcontrolador (Arduino MKR WiFi 1010 y ESP32), interconectadas con un PC y entre sí mediante conexiones WiFi.

Los objetivos de este Trabajo de Fin de Grado se pueden dividir en dos secciones:

- Objetivos hardware:
  - Construcción de una batería a medida para el robot con celdas LiFePo4 32700.
  - Diseño de esquema eléctrico y de control de los componentes.
  - Ensamblaje y conexionado de los componentes hardware del robot (motores, batería, sensores, microcontroladores, PC).
- Objetivos software:
  - Adquisición de datos de sensores angulares absolutos de efecto hall AS5600 y de Unidad de Movimento Inercial MPU9250.
  - Programación de microcontroladores en entorno de Arduino y Matlab/Simulink.
  - Control de motores en posición absoluta e incremental y modo de nivelado horizontal automático del robot.
  - Creación de protocolo de comunicación entre dispositivos (PC, ESP32, Arduino MKR WiFi 1010) con identificación de fallos.

1.3 ¿Por qué es interesante este trabajo?

El presente Trabajo de Fin de Grado se centra en el diseño e implementación de un modelo inicial de programación del sistema de control de Horu, un robot de rescate. Horu está destinado a operar en entornos hostiles que requieren alta fiabilidad y un control preciso para realizar tareas críticas, como la búsqueda y rescate de víctimas, el transporte de materiales y la inspección de estructuras colapsadas. Para operar eficazmente en estas situaciones, Horu cuenta con cuatro orugas articuladas que le permiten desplazarse por terrenos difíciles. Además, incorpora un brazo manipulador ubicado en la parte superior del robot, diseñado para asistir en las tareas mencionadas.

La construcción de una batería LiFePO4 con celdas 32700 constituye uno de los objetivos hardware iniciales del proyecto. La elección de este tipo de batería, fabricada con celdas de litio-ferrofosfato, se debe a que ofrece mayor seguridad (frente a perforaciones), menor calentamiento y una vida útil más prolongada en comparación con otras baterías de litio. Además, al ser una batería diseñada a medida para integrarla en la parte baja del chasis

del robot, se reduce el riesgo de daños por posibles impactos; especialmente considerando que el robot estará diseñado para operar en terrenos irregulares y con obstáculos salientes.

Uno de los aspectos más relevantes de este proyecto es la arquitectura de control distribuido, basada en un Arduino MKR WiFi 1010 y un ESP32 de Espressif Systems, coordinados por un ordenador de a bordo Intel NUC i7. Para resolver la complejidad de comunicar en tiempo real los ocho motores del robot (cuatro de tracción y cuatro de elevación) entre los microcontroladores y el PC, se desarrollará un protocolo de comunicación WiFi propio que garantice baja latencia, sincronización precisa y fiabilidad en la transmisión de comandos.

Un desafío importante a resolver en este trabajo está relacionado con el movimiento de elevación de las orugas. El motor de elevación carece de par para poder elevar el vehículo; además, la goma de la oruga debe desplazarse en conjunto con el motor de elevación para evitar frenar o dañar los componentes mecánicos de la oruga. La solución consistirá en una combinación del movimiento simultáneo de los motores de elevación y tracción.

Finalmente, se resuelve la problemática de nivelación del vehículo, muy importante para labores de manipulación y teleoperación (mantener un punto de vista horizontal). Esto se consigue con la utilización de un sensor inercial, llegando a realizar una nivelación de forma autónoma.

1.4 Contenido de la Memoria

Este documento se estructura en nueve capítulos y un anexo, siendo este el primer capítulo, correspondiente a la introducción. A continuación, se presenta un resumen del contenido de los ocho capítulos restantes:

- El capítulo 2 se centra en los componentes físicos empleados en el desarrollo de este Trabajo de Fin de Grado, describiendo también el proceso seguido para la construcción de la batería principal del robot. Los componentes están organizados en categorías funcionales, como microcontroladores, motores, periféricos, entre otros.

- El diseño del esquema eléctrico del sistema de control diseñado para Horu se describe en el capítulo 3. A partir de este diseño, se explican las conexiones entre los componentes y se justifica su configuración según los requisitos de control, alimentación y comunicación del robot.

- El capítulo 4 aborda los componentes software del sistema, explicando los entornos utilizados durante el desarrollo del proyecto. El principal software empleado ha sido Matlab, utilizado para el modelado y simulación de sistemas como los motores de tracción, y Simulink, que facilita la programación basada en modelos. También se ha utilizado Arduino IDE como entorno para la programación de los motores de elevación.

- El esquema general de control del robot se describe en el capítulo 5. Aquí se profundiza en el flujo de comunicaciones entre los módulos del sistema, los modos de operación implementados y las funcionalidades asignadas tanto a los microcontroladores como al

ordenador principal. Además, se especifican las responsabilidades de cada dispositivo y los protocolos y flujos de datos utilizados.

- En el capítulo 6 se describe el modelo de programación del control en el PC. Se detalla cómo se programaron los distintos bloques funcionales en el ordenador principal, el cual actúa como interfaz de usuario (UI). Este equipo recibe las órdenes de movimiento del usuario a través de un mando Xbox y, mediante bloques de Simulink, se gestiona la lógica de control. Según las entradas del usuario, se determina el estado activo y se envían los datos correspondientes al microcontrolador a través de WiFi.

- El modelo de programación del control Arduino MKR WiFi 1010 se explica en el capítulo 7. Se detalla el código implementado en el microcontrolador Arduino MKR WiFi 1010, responsable del control de los motores de tracción según las consignas recibidas desde el ordenador principal. Este microcontrolador interpreta las órdenes vía WiFi y, dependiendo del modo de operación activo, coordina el comportamiento de los ocho motores mediante comunicación con el ESP32. Además, ejecuta los modos de control definidos en el sistema: control por posición absoluta, control incremental y nivelado automático.

- Durante el desarrollo del capitulo 8 se describe el código de programación para el control del ESP32. Este microcontrolador se encarga de controlar los motores de elevación en función de los datos recibidos del Arduino MKR WiFi 1010. Se detallan las funciones de comunicación entre ambos microcontroladores utilizando el protocolo UDP, así como las rutinas encargadas de la lectura de los sensores de posición absoluta y el ajuste de los motores de elevación según los datos obtenidos y el modo de operación activo.

- El capítulo 9 presenta las conclusiones de cada uno de los capítulos, contrastándolas con los objetivos preliminares propuestos al inicio del trabajo. También se incluye un apartado sobre las mejoras futuras que se pueden implementar como continuación de este Trabajo de Fin de Grado.

- En el Apéndice A se propone una versión simplificada del modelo de control del PC propuesto en el capítulo 6.

Capítulo 2

Componentes de Hardware

2.1 Introducción

En este capítulo de la memoria se expondrán y justificarán los componentes que conforman el robot Horu. Algunos de ellos pueden adquirirse fácilmente en el mercado, mientras que otros, como la batería, han sido fabricados manualmente, como se verá descrito más adelante. Las orugas, en concreto, han sido realizadas gracias a Trabajos de Fin de Grado de antiguos compañeros de RoboRescue UMA.

Los perfiles de aluminio, tornillería y demás componentes mecánicos, así como los elementos eléctricos como convertidores de tensión, caja de fusibles, relés, interruptores y otros componentes que conforman el robot, han sido montados y ensamblados junto al equipo de RoboRescue UMA durante el desarrollo de este proyecto.

Para garantizar un ensamblaje adecuado y una correcta integración de los componentes en el chasis del robot, se han diseñado y fabricado piezas mediante impresión 3D. Esto ha permitido realizar pruebas durante el desarrollo del proyecto, modificando las piezas conforme a las necesidades o inconvenientes que iban surgiendo, como los soportes de los encoders magnéticos.

2.2 Motores

El movimiento del robot se consigue mediante la actuación de ocho motores: cuatro servomotores de corriente continua sin escobillas RMD X8 S2 V3 de MyActuator, responsables de proporcionar la tracción al robot, y cuatro motores paso a paso controlados por el driver DM542 de Walfront. Estos últimos permiten la articulación de las orugas, facilitando la superación de obstáculos.

2.2.1 Motores RMD X8 S2 V3

Figura 2.1: Motor de tracción modelo RMD X8 S2 V3

Los RMD X8 S2 V3 de la marca china MyActuator son unos motores sin escobillas, de núcleo hueco (ver figura 2.1). El hueco lo ocupa una primera etapa reductora de velocidad de relación 1:36 lo que significa que la velocidad de giro de la rueda será 1/36 de la velocidad suministrada por el motor, a cambio de un mayor par. Una característica importante de estos es que incorporan tanto la etapa de potencia como la de electrónica de control. La parte de electrónica de control la realiza el chip MC-X-300-O 6825. Tienen un consumo medio de 80 W cada uno y operan con un voltaje máximo de 48 V. Son capaces de alcanzar hasta 1440 rpm y generar un par motor de hasta 60 Nm. Cada motor pesa 900 gramos y presenta una inercia de giro de 96 kg·cm², lo que influye en su aceleración y deceleración. Además, poseen un radio de 4.9 cm, un dato relevante para la programación de la cinemática del robot.

En cuanto a la conectividad, pueden comunicarse con otros dispositivos mediante UART, RS485 y CAN, aunque solo estos dos últimos protocolos permiten comandar el motor

directamente mediante el envío de mensajes.

Estos motores cuentan con un software propio para su calibración y actualización de firmware, el cual se ha utilizado a lo largo del desarrollo del proyecto, principalmente para la verificación de errores. Sin embargo, este software tiene limitaciones, ya que no permite programar acciones complejas y requiere que el motor esté conectado al PC mediante UART en lugar de utilizar el bus CAN.

A pesar de estas restricciones, el software permite ajustar manualmente ciertos parámetros del motor, como las constantes proporcional e inercial de giro. También es posible modificar configuraciones de comunicación, como el baudrate de las conexiones CAN y RS485, asignar una ID al motor para su identificación en dichas conexiones o incluso restaurarlo a su estado de fábrica, restableciendo todos sus parámetros a los valores predeterminados.

En este proyecto, los motores se controlan a través del bus CAN, un sistema de comunicación half-duplex que, aunque no permite la transmisión y recepción simultánea de mensajes, posibilita el control de múltiples motores mediante un único cable, utilizando un esquema de conexión Master-Slave.

Existen numerosos comandos que pueden enviarse al motor, los cuales se detallarán más adelante. Sin embargo, el formato general de cada mensaje, ya sea un comando dirigido al motor o una respuesta de feedback, sigue la siguiente estructura (ver figura 2.2):

- 12 bits de ID.
- 8 campos de datos de 8 bits cada uno (1 byte por campo).

ID	Data [0]	Data [1]	Data [2]	Data [3]	Data [4]	Data [5]	Data [6]	Data [7]
0x141	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Figura 2.2: Ejemplo de mensaje CAN

Además, la ID del motor es un número del 1 al 32 (a elección del usuario) sumado a 0x140 para enviar el mensaje (0x140 + 1~32 será la ID). Para comandar varios motores con un solo mensaje, se usará la ID 0x280, y en recepción se recibirán mensajes con la ID 0x240.

La programación de estos motores se ha realizado en Simulink, utilizando el microcontrolador Arduino MKR WiFi 1010 (ver figura 2.4) junto con la placa MKR Shield CAN, que facilita la comunicación mediante el protocolo CAN entre los motores y el sistema de control, como se detallará más adelante.

Este método de programación ha sido elegido por la facilidad de integración que proporciona la Shield CAN entre el Arduino MKR WiFi 1010 y los motores.

El conexionado de los motores al microcontrolador Arduino MKR WiFi 1010 y al módulo CAN se ha realizado soldando todos los cables de CAN-HIGH de los cuatro motores a un pin común, y, por otro lado, los cuatro cables de CAN-LOW a otro pin común (ver figura 2.3). Estos nodos, creados a partir de las uniones de los cables, se han conectado directamente a la Shield CAN.

Cabe destacar que, debido al método de conexionado empleado, ha sido necesario hacer uso de la resistencia de 120Ω que incorpora la Shield, habilitándola mediante el switch correspondiente.

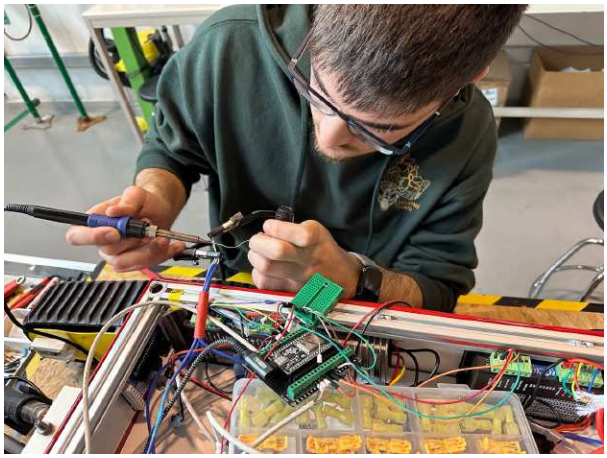


Figura 2.3: Soldadura de cables CAN bus en modo estrella

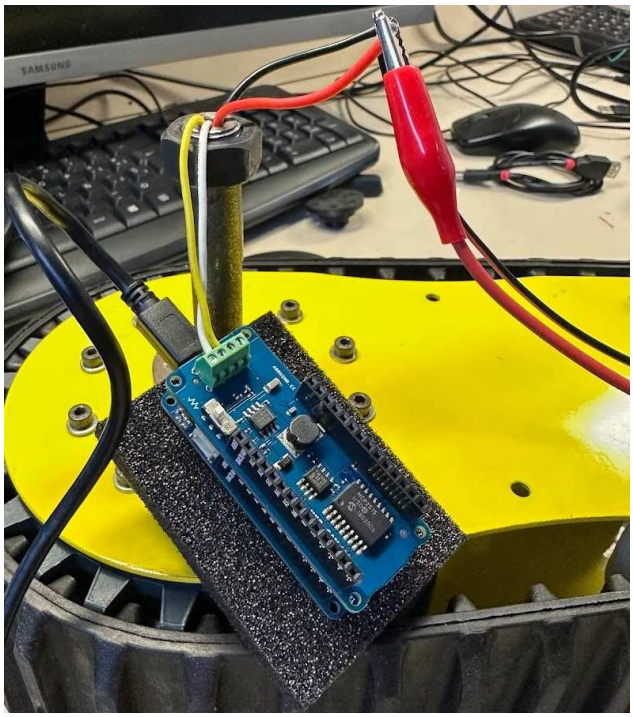


Figura 2.4: Prueba donde se aprecia un primer plano de la Shield CAN conectada a los cables CAN del motor de tracción de una oruga

Un problema que se presentó al programar los motores de tracción fue que la nomenclatura utilizada por la controladora de los motores difiere de la de la Shield CAN. Es decir, durante la primera prueba para conectar uno de los motores de tracción a la Shield, se conectaron CANH de la controladora del motor con CANH de la Shield, y CANL de la controladora con CANL de la Shield, pero no se estableció comunicación. Como solución, se cambiaron de posición los cables del bus CAN y funcionó correctamente. Tras analizar el comportamiento, se concluyó que la serigrafía de la controladora tiene invertidas las dos salidas del bus.

2.2.2 Motores Paso a Paso

Figura 2.5: Motor paso a paso Walfront

Los motores paso a paso utilizados en este proyecto son el modelo *57HS112-3004A08-D21* de la marca Walfront (ver figura 2.5). Lo más relevante es el controlador de cada uno de ellos: el *Microstep Driver DM542* (ver figura 2.6). Este dispositivo actúa como un amplificador de corriente y un controlador de movimiento, permitiendo que el motor funcione con mayor precisión y suavidad.

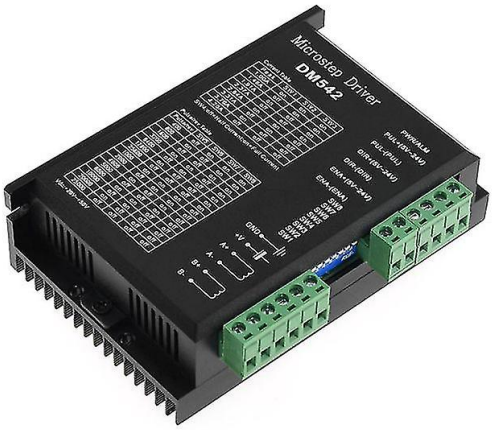


Figura 2.6: Driver DM542

Cada motor incorpora una reductora con una relación de 80:1 (ver figura 2.7), lo que significa que la velocidad de giro de salida es 1:80 de la velocidad original del motor. Como consecuencia, se incrementa el par motor hasta 3 Nm, lo que mejora la capacidad de movimiento en función de la carga aplicada.

Figura 2.7: Motor paso a paso y reductora 1:80

El DM542 permite el control de los llamados *microsteps*, es decir, la realización de movimientos con precisión en fracciones de grado. Los *microsteps* son configurables mediante la combinación de unos interruptores ubicados en el propio driver (ver figura 2.8). Este tipo de control mejora significativamente la resolución del movimiento y reduce las vibraciones del motor, una característica esencial en aplicaciones donde la precisión es crítica, como en impresoras 3D.

Microstep Resolution Selection

Microstep resolution is set by SW5, 6, 7, 8 of the DIP switch as shown in the following table:

Microstep	Steps/rev.(for 1.8°motor)	SW5	SW6	SW7	SW8
2	400	OFF	ON	ON	ON
4	800	ON	OFF	ON	ON
8	1600	OFF	OFF	ON	ON
16	3200	ON	ON	OFF	ON
32	6400	OFF	ON	OFF	ON
64	12800	ON	OFF	OFF	ON
128	25600	OFF	OFF	OFF	ON
5	1000	ON	ON	ON	OFF
10	2000	OFF	ON	ON	OFF
20	4000	ON	OFF	ON	OFF
25	5000	OFF	OFF	ON	OFF
40	8000	ON	ON	OFF	OFF
50	10000	OFF	ON	OFF	OFF
100	20000	ON	OFF	OFF	OFF
125	25000	OFF	OFF	OFF	OFF

Figura 2.8: Tabla sobre la configuración de los microsteps

Este driver en particular permite dividir cada revolución en un rango de entre 400 y 25000 pasos. A mayor cantidad de divisiones, mayor será la precisión del movimiento, aunque a costa de una menor velocidad de avance del motor. En este caso se utilizará la configuración de 2 *microstep*, con la que se consiguen 400 steps/rev.

El driver DM542 (ver figura 2.6) cuenta, además de las conexiones al motor y la alimentación de 38V, con seis entradas de control: PUL+, PUL-, DIR+, DIR-, ENA+ y ENA-. Estas señales se interpretan de manera diferencial, es decir, como la diferencia de tensión entre cada par de valores, con el fin de reducir el ruido común en ambas entradas.

- La señal PUL- recibirá una señal PWM idéntica para todos los motores, que determinará la velocidad de movimiento en función de su frecuencia. Esta señal PWM será creada por el microcontrolador ESP32.
- La señal DIR- recibirá un valor digital de 0V o 5V que definirá la dirección de giro del motor.
- La señal ENA+ se utilizará para habilitar o deshabilitar el driver del motor, lo que es crucial para evitar un consumo excesivo de energía y el sobrecalentamiento de los motores.

En cuanto a la programación, el control de estos motores es significativamente más simple que el de los motores descritos anteriormente. Con tan solo tres señales digitales se puede realizar el control: una señal PWM para regular la velocidad, una señal digital para cambiar la dirección y otra para encender o apagar el motor.

2.3 Microcontroladores

2.3.1 Microcontrolador Arduino MKR WiFi 1010

El microcontrolador Arduino MKR WiFi 1010, diseñado por la empresa italiana *Arduino* (ver figura 2.9), es una placa electrónica programable que combina la versatilidad de programación de un microcontrolador Arduino con capacidades avanzadas de conectividad WiFi, así como la posibilidad de establecer conexiones TCP/IP y UDP/IP con dispositivos remotos. Además, para el desarrollo de este proyecto se le ha conectado una Shield CAN que permite la comunicación mediante el bus CAN entre el microcontrolador y los motores de tracción.

Figura 2.9: Arduino MKR WiFi 1010

El MKR Shield CAN (ver figura 2.10) está basado en el controlador MCP2515 y el transceptor TJA1049. También incorpora el convertidor buck TPS54232 de *Texas Instruments*, un convertidor de potencia CC-CC que reduce la tensión de entrada.

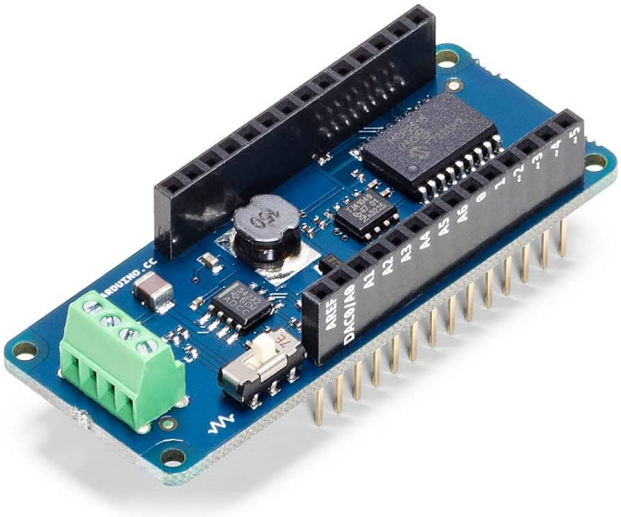


Figura 2.10: Shield CAN para Arduino MKR WiFi 1010

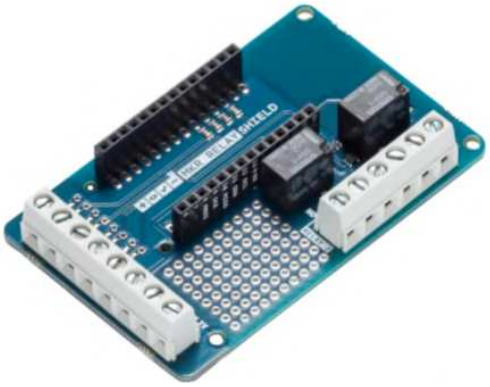


Figura 2.11: Arduino MKR WiFi 1010 placa prototipado

Esta placa dispone de una entrada micro USB tipo B para su conexión a un PC, la cual también proporciona alimentación de 5V. Sin embargo, no es estrictamente necesario que esté conectada a un ordenador para su funcionamiento, ya que puede alimentarse de forma independiente a través del pin VIN, con un rango de tensión de 5-7V. Además, cuenta con un conector específico para baterías Li-Po de 3.7V.

El microcontrolador ofrece una amplia variedad de pines, entre los cuales se encuentran 7 entradas analógicas y 14 pines de entrada/salida digital. De estos últimos, 2 están destinados a comunicaciones seriales y 7 pueden ser utilizados para control PWM. También dispone de pines para comunicación mediante I2C, los cuales serán utilizados, como se verá más adelante.

Este microcontrolador será el encargado del control de movimiento de los motores de tracción, ya que, como se mencionó anteriormente, la comunicación con estos se establece mediante el protocolo CAN. Por otro lado, el Arduino MKR WiFi 1010 también se comunicará con el ESP32 y el ordenador principal mediante WiFi, para la transmisión y recepción de los datos necesarios para el control.

Para fijar el microcontrolador al robot, se ha utilizado una placa de prototipado (ver figura 2.11), que cuenta con dos relés que se emplearán para el control de las luces del robot.

2.3.2 Microcontrolador ESP32 DEVKIT V1

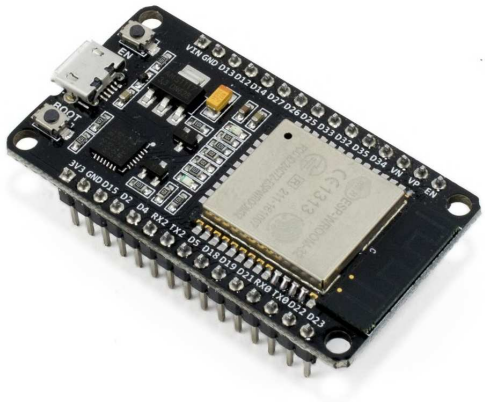


Figura 2.12: ESP32

La placa microcontroladora ESP32 DEVKIT V1, basada en el chip ESP32 de *Espressif Systems* (ver figura 2.12), es una placa de desarrollo programable que ofrece conectividad WiFi y Bluetooth, además de una alta capacidad de procesamiento. Gracias a su arquitectura de doble núcleo Xtensa LX6, permite ejecutar múltiples tareas simultáneamente, lo que la hace ideal para aplicaciones en Internet de las Cosas (IoT), automatización y sistemas embebidos avanzados.

Esta placa cuenta con interfaces UART, SPI, I2C y PWM, lo que facilita la comunicación con una amplia variedad de periféricos. Además, dispone de ADC y DAC para la conversión analógica-digital y digital-analógica, respectivamente.

La placa ESP32 DEVKIT V1 se alimenta a través de un puerto USB tipo C que proporciona 5V, aunque también puede recibir alimentación externa mediante el pin VIN, con un rango de tensión de 5-12V. Su bajo consumo energético la hace adecuada para aplicaciones que requieren funcionamiento continuo.

En cuanto a la distribución de pines, la placa cuenta con entre 30 y 38 pines, dependiendo de la versión, incluyendo hasta 18 pines de entrada/salida digital, capacidades de comunicación serie y soporte para múltiples protocolos de red.

El anclaje de la placa al robot se realizará mediante la siguiente placa de prototipado (ver figuras 2.13 y 2.14):

Figura 2.13: Placa de prototipado ESP32

Este microcontrolador será responsable de adquirir los datos de los sensores de posición absoluta, instalados en los motores paso a paso, estableciendo una comunicación mediante el protocolo I2C con el multiplexor. Además, se encargará de controlar los drivers de potencia DM542 de los motores paso a paso, así como de enviar y recibir datos por WiFi al Arduino MKR WiFi 1010 y al PC.

A diferencia del microcontrolador anterior, este será programado en lenguaje de programación C/C++ utilizando el entorno de desarrollo Arduino IDE.

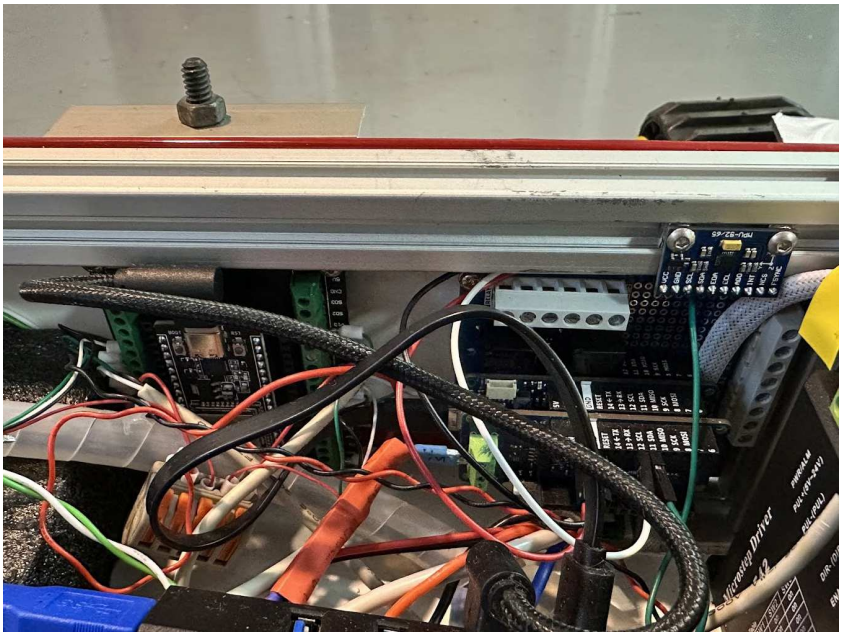


Figura 2.14: Placas de prototipado de Arduino MKR WiFi 1010 y ESP32 ancladas al chasis del robot

2.4 Periféricos

2.4.1 Multiplexor I2C TCA9548A 8CH

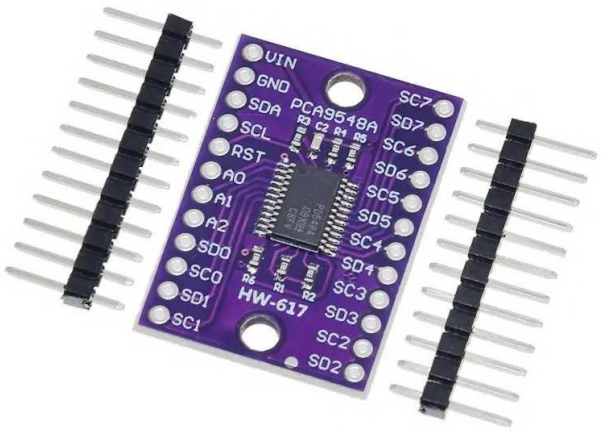


Figura 2.15: Multiplexor modelo TCA9548A 8CH

El multiplexor I2C TCA9548A (ver figura 2.15) es un circuito integrado diseñado para permitir la conexión de múltiples dispositivos I2C a un único bus maestro, evitando conflictos de direcciones y facilitando la expansión de sensores y módulos en un sistema embebido. Funciona permitiendo la selección de hasta ocho buses I2C secundarios a través de comandos enviados desde el bus I2C principal. Cada canal puede activarse o desactivarse de forma independiente mediante el envío de un byte de control, lo que permite al maestro comunicarse con diferentes dispositivos sin interferencias. En el caso del robot, el maestro será el ESP32.

El dispositivo cuenta con una dirección I2C base configurable mediante tres pines de dirección (A0, A1, A2), lo que permite la conexión de hasta 8 multiplexores en el mismo sistema, proporcionando un total de 64 canales I2C disponibles, por si fuese necesario más entradas, pero no es el caso.

Entre sus características principales se incluyen un voltaje de operación de 1.65V a 5.5V, compatibilidad con buses I2C de alta velocidad (hasta 400 kHz), ocho canales I2C con selección independiente, baja corriente de operación y modo de apagado para ahorro de energía, protección contra voltajes inversos y limitación de corriente.

Las entradas de este serán las salidas I2C de los cuatro codificadores magnéticos AS5600, y la salida irá conectada mediante I2C al ESP32, como se verá en el esquema eléctrico de un próximo capítulo.

2.4.2 Sensor de posición rotatorio magnético AS5600

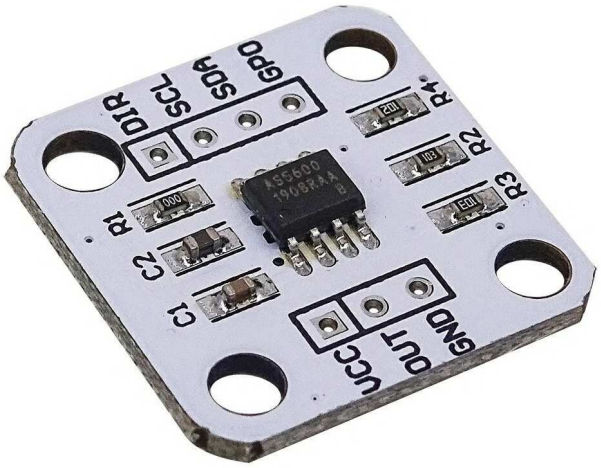


Figura 2.16: Sensor de posición AS5600

El codificador magnético AS5600 (encoder) es un sensor de posición angular sin contacto basado en tecnología magnética (ver figura 2.16) . Está diseñado para proporcionar mediciones precisas en aplicaciones de control de posición y detección de ángulos, lo que lo hace ideal para motores, actuadores y sistemas de posicionamiento. Su funcionamiento se basa en la detección de un campo magnético generado por un imán externo, convirtiéndolo en una señal de salida digital o analógica.

El AS5600 opera mediante una interfaz I2C o una salida analógica PWM, siendo en este caso utilizada la interfaz I2C para la comunicación con el microcontrolador ESP32. Su resolución de 12 bits permite una alta precisión en la detección de ángulos, facilitando una respuesta suave y exacta en aplicaciones de control. Además, el sensor permite la configuración del punto cero (de posición), lo que permite realizar calibraciones personalizadas para adaptarse a diferentes orugas.

El principio de funcionamiento del AS5600 se basa en la medición de la posición angular de un imán que rota sobre su superficie. El sensor mide el campo magnético generado por el imán y lo convierte en una señal digital proporcional al ángulo de rotación. Para garantizar su correcto funcionamiento, el imán debe colocarse alineado con el sensor. Para ello, se ha realizado el montaje sobre una pieza plástica fabricada mediante impresión 3D.

Este sensor proporciona una salida digital de 12 bits $(2^{12} = 4096)$, lo que significa que el valor de salida varía en un rango de 0 a 4095 para representar un giro completo de 360°. En otras palabras, cada unidad en el valor de salida corresponde a un pequeño incremento angular, lo que permite obtener mediciones extremadamente detalladas del ángulo de rotación. Este valor de salida es denominado $\mathrm{raw\ angle}$ y varía en función de la posición del

imán con respecto al sensor.

Para ilustrar su comportamiento, se presentan algunos valores de referencia de la salida en relación con el ángulo de rotación del imán (ver figura 2.17):

- **0°** (raw angle = 0)
- **90°** (raw angle = 1024)
- **180°** (raw angle = 2048)
- **270°** (raw angle = 3072)

El AS5600 utiliza un imán bipolo (con polos Norte-Sur) para detectar la posición angular sin necesidad de contacto mecánico, lo que lo hace más resistente al desgaste y adecuado para aplicaciones en entornos exigentes, como es el caso del robot. A medida que el imán gira, el sensor mide la variación del campo magnético y genera una salida angular precisa y acondicionada; por ello, no será necesaria una etapa de filtrado de la señal.

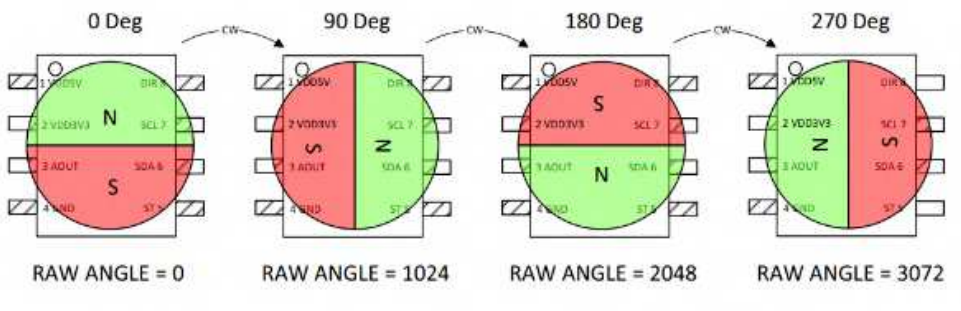
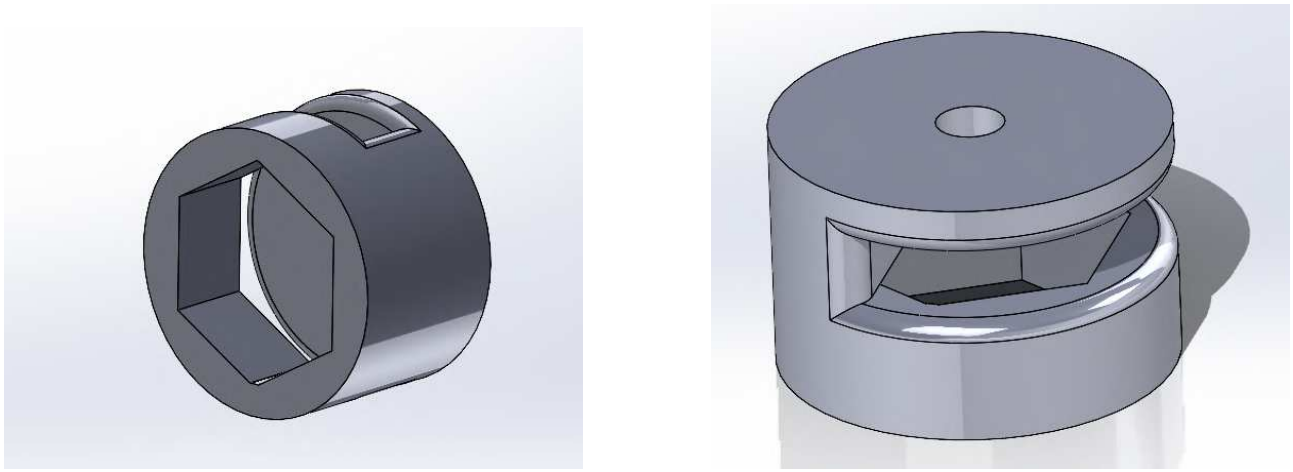


Figura 2.17: Variación del RAW ANGLE en función de la posicion del imán

La pieza impresa en 3D que sostiene el imán se muestra en las figuras 2.18a y 2.18b; dicho soporte se montará solidario a la tuerca de la oruga que conecta con el motor paso a paso.



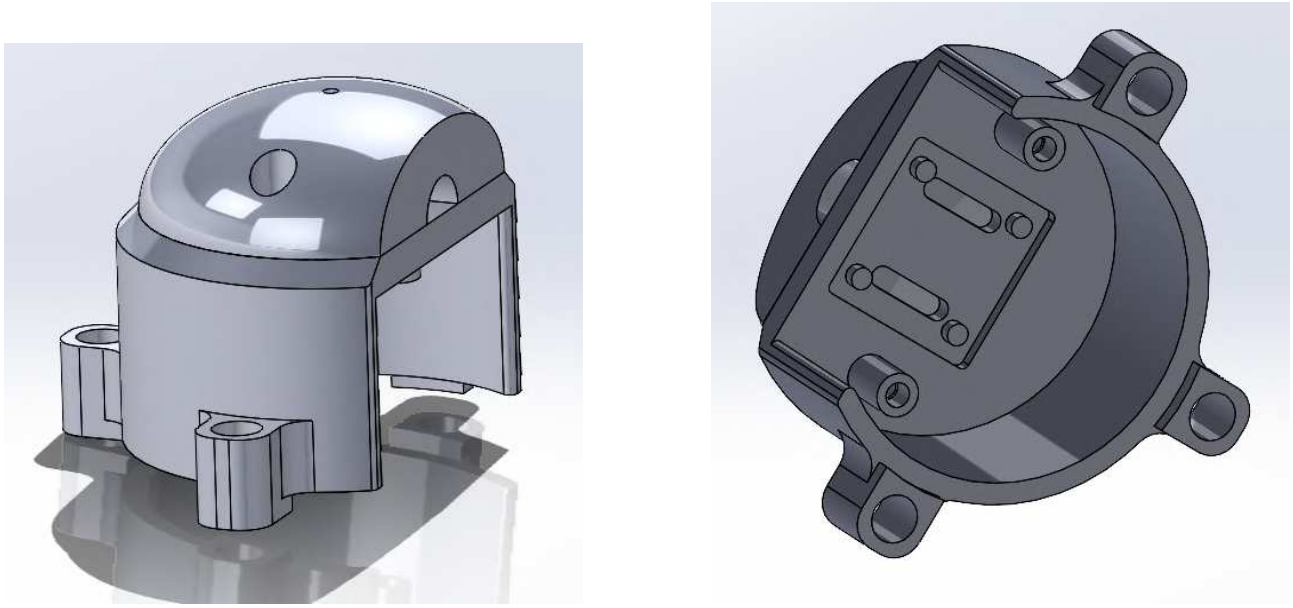
(a) Vista 1 **(b)** Vista 2

Figura 2.18: Diseño 3D del soporte para el imán

Los pines de alimentación (Vin) de los cuatro encoders reciben 3.3V provenientes del ESP32. La comunicación de los cuatro encoders con el microcontrolador ESP32 se realiza a través del multiplexor, utilizando el protocolo de comunicación I2C; todo esto se detallará en el siguiente capítulo. El conexionado se ha realizado mediante soldadura por estaño (ver figura 2.19).

Figura 2.19: Proceso de soldadura de los encoders

Los codificadores se han montado sobre piezas impresas en 3D (ver figuras 2.20a y 2.20b), colocadas frente a los motores paso a paso. Se puede ver cómo quedan finalmente ensambladas con los sensores en la figura 2.21.



(a) Vista 1 **(b)** Vista 2

Figura 2.20: Diseño 3D del soporte del sensor AS5600

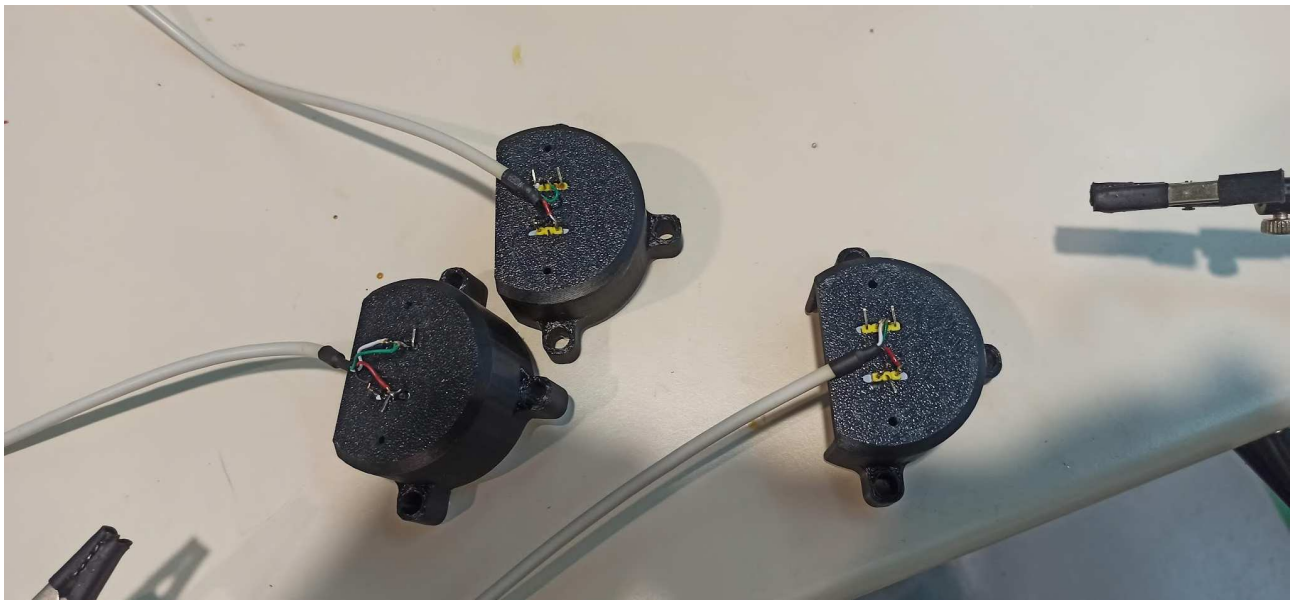


Figura 2.21: Codificadores en soporte de impresión 3D

Finalmente, en la figura 2.22 se muestra el modelo 3D del ensamblaje completo de los soportes del encoder y la oruga, y en la figura 2.23 se incluye una captura de las pruebas iniciales realizadas para determinar la correlación entre la velocidad de giro del motor paso a paso y la precisión de lectura del encoder.

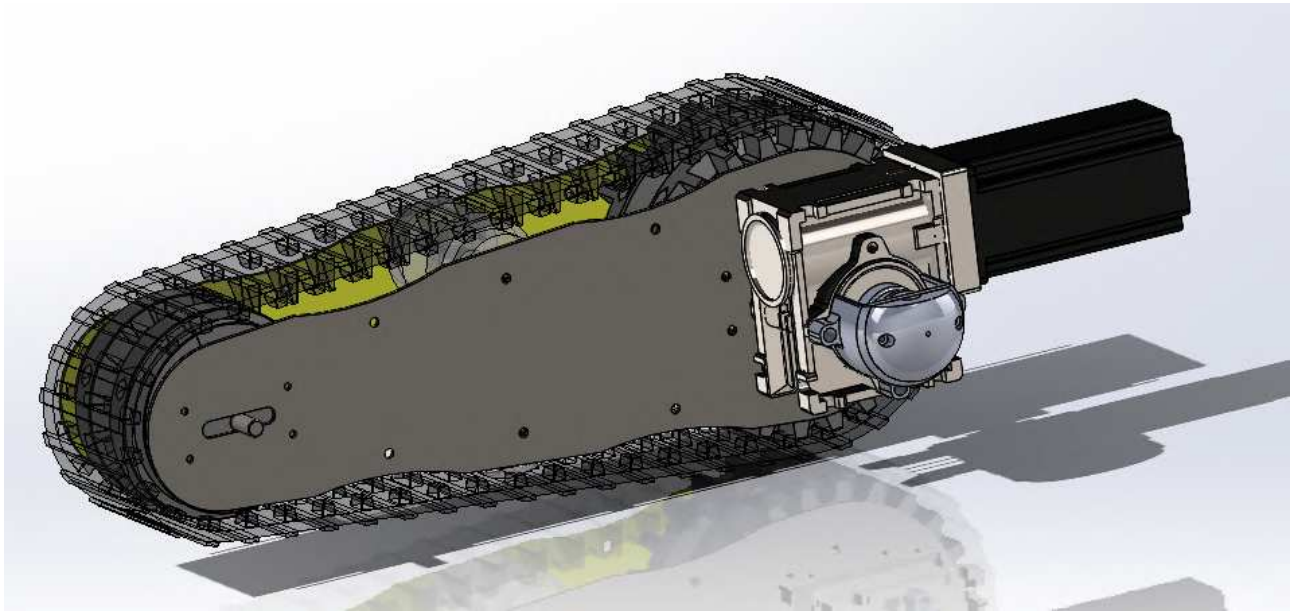


Figura 2.22: Diseño 3D de la oruga completa y soporte para encoder

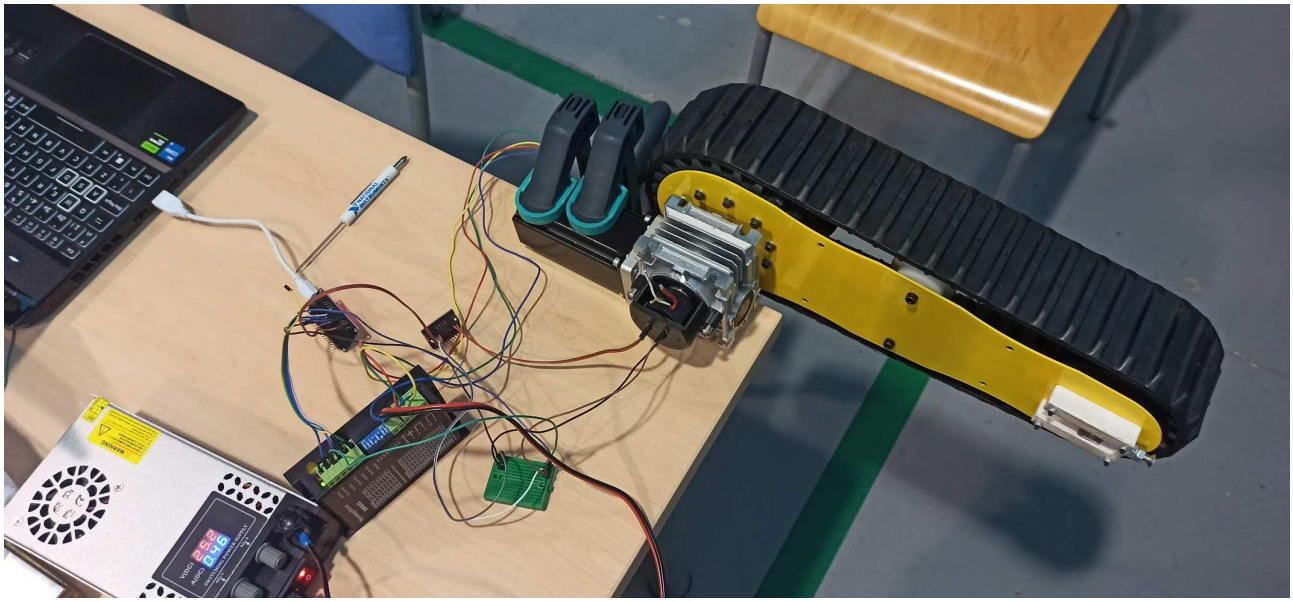


Figura 2.23: Pruebas del DM542 para determinar la configuración óptima de step/rev usando como controlador el ESP32

2.4.3 IMU MPU9250

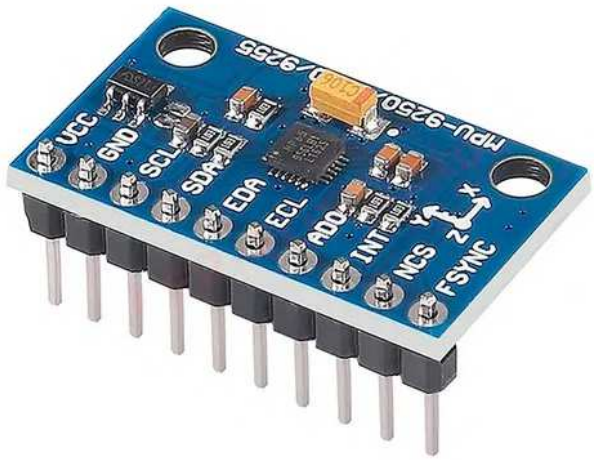


Figura 2.24: IMU MPU9250

La IMU MPU9250 (ver figura 2.24) es una unidad de medición inercial la cual incorpora tres sensores principales: acelerómetro, giroscopio y magnetómetro. El acelerómetro MEMS detecta fuerzas específicas, incluyendo la gravedad, lo que permite calcular ángulos de inclinación (pitch y roll). El giroscopio mide la velocidad angular, útil para detectar giros rápidos, pero presenta deriva a largo plazo. El magnetómetro AK8963 mide el campo magnético terrestre y se usa principalmente para calcular el yaw (rumbo), complementando así al giroscopio.

El módulo cuenta con un procesador digital de movimiento (DMP) interno, capaz de ejecutar algoritmos básicos de fusión de sensores para calcular la orientación, reduciendo la carga computacional del microcontrolador externo. El MPU9250 opera a tensiones de 2.4 V a 3.6 V, tiene un bajo consumo de energía (aproximadamente 3.5 mA) y permite configuraciones de muestreo de hasta 1 kHz.

La calibración es un paso crítico para obtener datos precisos. Para el acelerómetro y giroscopio, se debe realizar una calibración estática para corregir offset y escala. El magnetómetro requiere una calibración de campo duro y blando (hard iron y soft iron) para compensar distorsiones magnéticas. Los datos finales de orientación se obtienen aplicando algoritmos de fusión, como el filtro de Madgwick o Mahony, que combinan las ventajas de los tres sensores: la estabilidad a largo plazo del magnetómetro, la respuesta rápida del giroscopio y la referencia gravitacional del acelerómetro.

En aplicaciones de robótica, se emplea la MPU9250 para estabilización de plataformas, control de balance, navegación inercial, mapeo simultáneo (SLAM) y sistemas autónomos donde es esencial conocer la actitud del robot. En el caso de este robot, se usará junto a los demás componentes para realizar un nivelado de manera automática del robot.

Dado que el robot incorporará un brazo robótico destinado a manipular objetos, abrir puertas, entre otras funciones, será fundamental que el proceso de nivelado se realice correctamente, con el fin de fijar el espacio de trabajo tanto del robot como del manipulador.

2.5 Batería

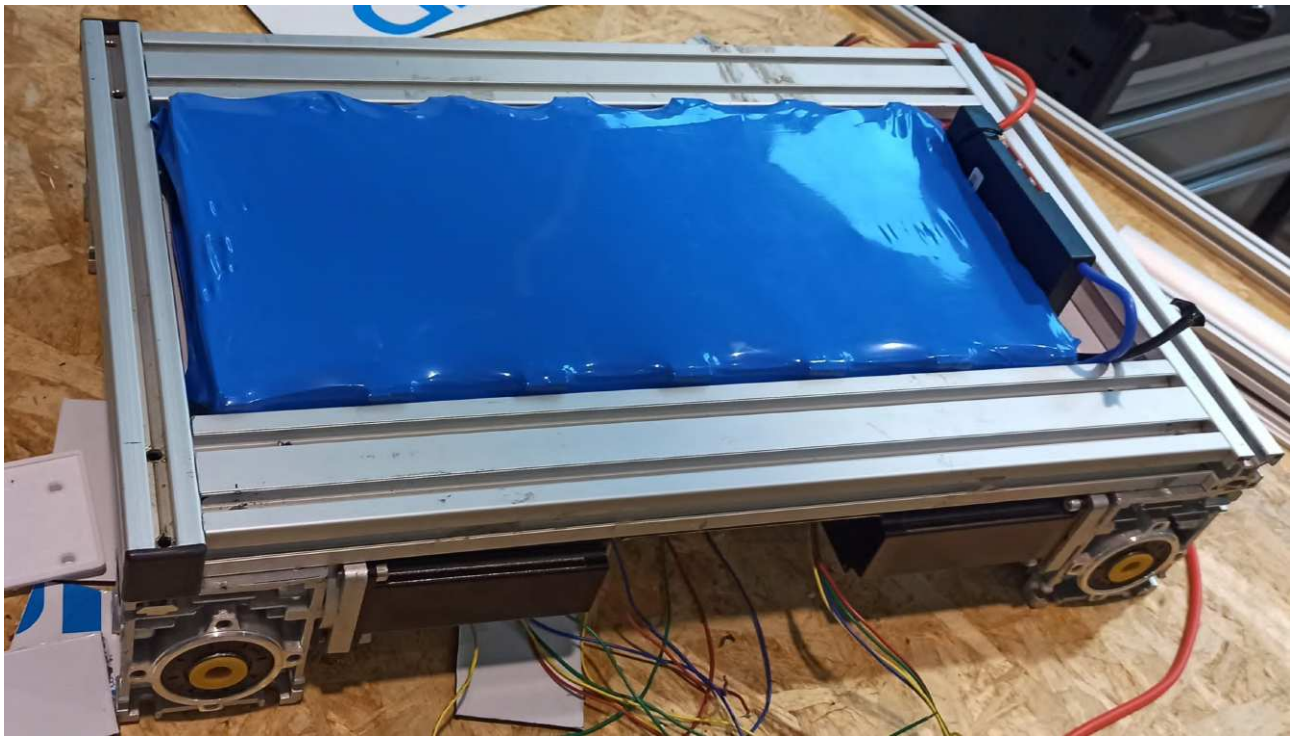


Figura 2.25: Batería del robot Horu basada en celdas LiFePO4 32700

2.5.1 Introducción

En esta sección del capítulo se describirá el proceso seguido para la construcción de una batería LiFePO4 tipo 12s3p (12 paquetes en serie, de 3 celdas en paralelo cada uno). Esta batería proporcionará la energía necesaria para alimentar todos los sistemas del robot (ver figura 2.25). El método seleccionado para la unión entre celdas ha sido la soldadura por puntos, una técnica comúnmente empleada en la construcción de baterías debido a su fiabilidad y baja resistencia eléctrica en las uniones. Este proceso consiste en hacer pasar una corriente eléctrica de alta intensidad a través de las tiras de níquel y los terminales de las celdas. Dicha corriente genera calor suficiente como para fundir localmente el metal en el punto de contacto, permitiendo la fusión entre ambos materiales bajo presión controlada.

Para lograr una correcta soldadura, los electrodos del soldador deben ejercer presión sobre las dos piezas metálicas a unir (en este caso, el níquel y el terminal de la celda). En la figura inferior (ver figura 2.26), se puede observar que los cables que conectan los electrodos al soldador son de cobre, al igual que los electrodos, pero estos últimos están recubiertos con una capa de óxido de aluminio. Este recubrimiento evita que el material del electrodo se adhiera a las tiras de níquel durante el proceso de soldadura.

Figura 2.26: Imagen donde se ve el soldador por puntos usado y un paquete de 12 celdas

2.5.2 Materiales usados

Para la construcción de la batería del robot se han utilizado los siguientes materiales:

Celdas LiFePo4

Figura 2.27: Celda LiFePO4 32700 3.2V 6000mAh

Las celdas LiFePO4 (litio hierro fosfato) son reconocidas por ser más seguras en comparación con otras químicas de baterías de litio (como las de Li-ion o LiPo). Tienen menor riesgo de sobrecalentamiento, explosión o incendio. En la batería se usarán 36 celdas exactamente.

En general, este tipo de celdas (ver figura 2.27) ofrecen una gran cantidad de ciclos de carga y descarga (2000), gracias a su construcción interna. El material activo en los electrodos es químicamente más estable en comparación con otros tipos de baterías de litio, como las Li-ion, que emplean óxido de litio-cobalto. Esta mayor estabilidad permite que las celdas de LiFePO4 tengan una menor tasa de degradación con el tiempo, especialmente durante los ciclos repetidos de carga y descarga.

Tira de níquel puro

Para la conexión de las celdas en grupos de tres en paralelo y, posteriormente, la unión de los doce grupos en serie, se han utilizado tiras de níquel puro (ver figura 2.28). Estas tiras permiten la interconexión de los terminales de cada celda. El espesor del níquel utilizado es de 0,25 mm, lo que asegura una adecuada conductividad y resistencia durante los ciclos de carga y descarga.

Figura 2.28: Tira de Níquel usada para soldar las celdas entre sí

Soldador por puntos

Para realizar la conexión entre las celdas por medio de la tira de níquel, se ha utilizado un soldador por puntos (ver figura 2.29).

Figura 2.29: Soldador por puntos

2.6 Esquemático de la batería

El esquemático de la batería ha sido diseñado utilizando la plataforma Fritzing (ver figura 2.30). El diseño de la batería se ha optimizado para proporcionar una capacidad adecuada en el rango de 43.8V a 38.4V, que es el voltaje requerido para alimentar los ocho motores utilizados en el sistema.

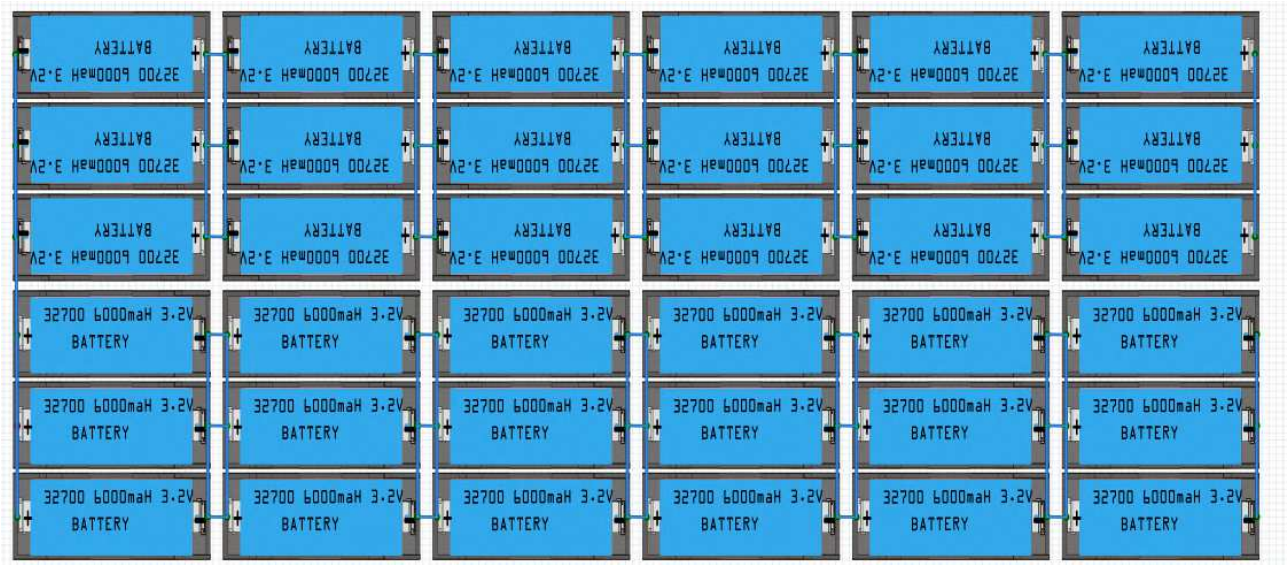


Figura 2.30: Esquemático de la batería

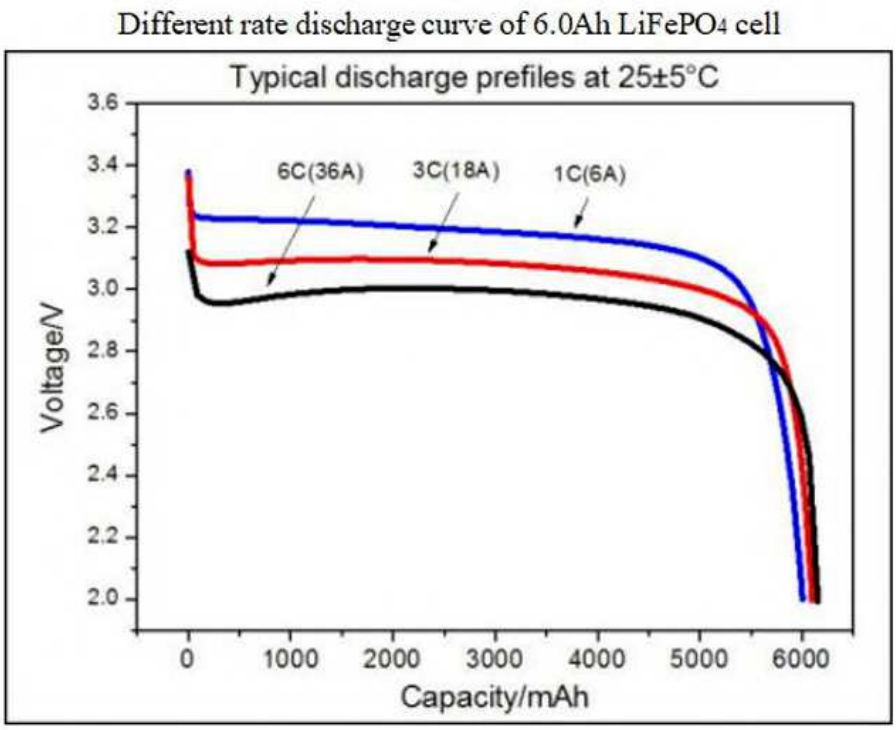


Figura 2.31: Gráfica de descarga de la celda 32700 3.2V

Cálculos justificativos

Al conectar 3 celdas en paralelo (3P), la capacidad total del conjunto se incrementa, ya que las capacidades individuales de cada celda se suman. Esto proporciona una capacidad total de:

$$\textit{Capacidad} = 6000\,\textit{mAh} \times 3 = 18000\,\textit{mAh} \tag{2.1}$$

La configuración 12S implica que se conectan 12 celdas en serie, lo que eleva el voltaje total de la batería. Dado que cada conjunto de 3 celdas en paralelo proporciona un voltaje de 3.2V (ver figura 2.31), al conectar 12 de estos grupos en serie, el voltaje total se calcula de la siguiente manera:

$$\textit{Voltaje\_min} = 3,2\,V \times 12 = 38,4\,V \tag{2.2}$$

Este valor corresponde al voltaje mínimo de descarga de la batería. Para calcular el voltaje máximo de la misma, se debe tener en cuenta el voltaje de corte en carga (*Charge–Cut–Off Voltage*) de cada celda, tal como especifica el fabricante (ver figura 2.32). De este modo, el voltaje máximo de la batería se expresa como:

$$\textit{Voltaje\_max} = 3{,}65\,V \times 12 = 43{,}8\,V \tag{2.3}$$

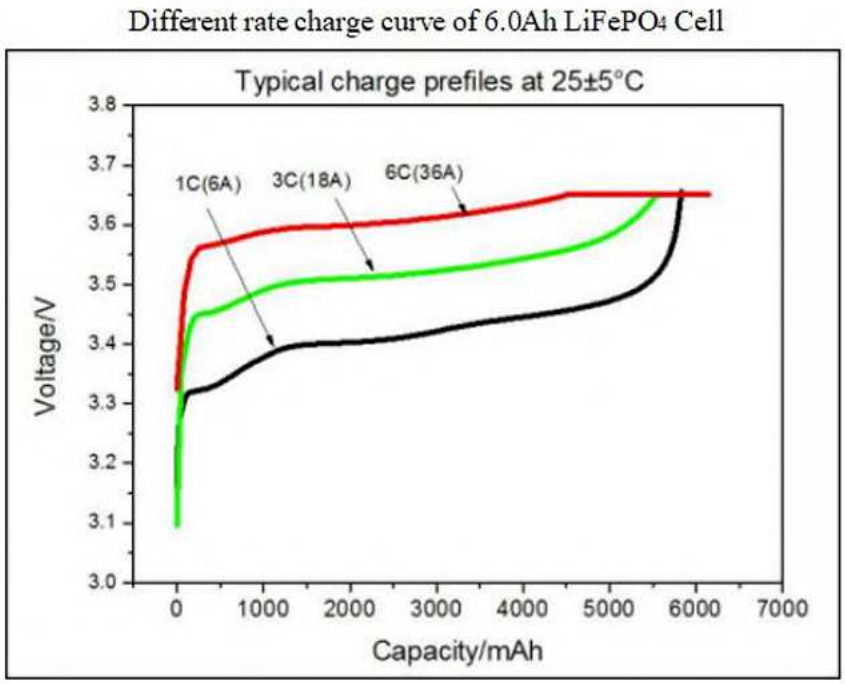


Figura 2.32: Gráfica de carga de la celda 32700 3.2V

2.6.1 Construcción de la batería

El proceso de fabricación y ensamblaje de la batería se ha realizado en el taller de Ingeniería de Sistemas y Automática.

1. Soldadura de los terminales de las celdas en grupos de 3

Se han soldado las celdas en grupos de tres (ver figura 2.33), respetando la polaridad adecuada descrita en el esquema eléctrico anterior.

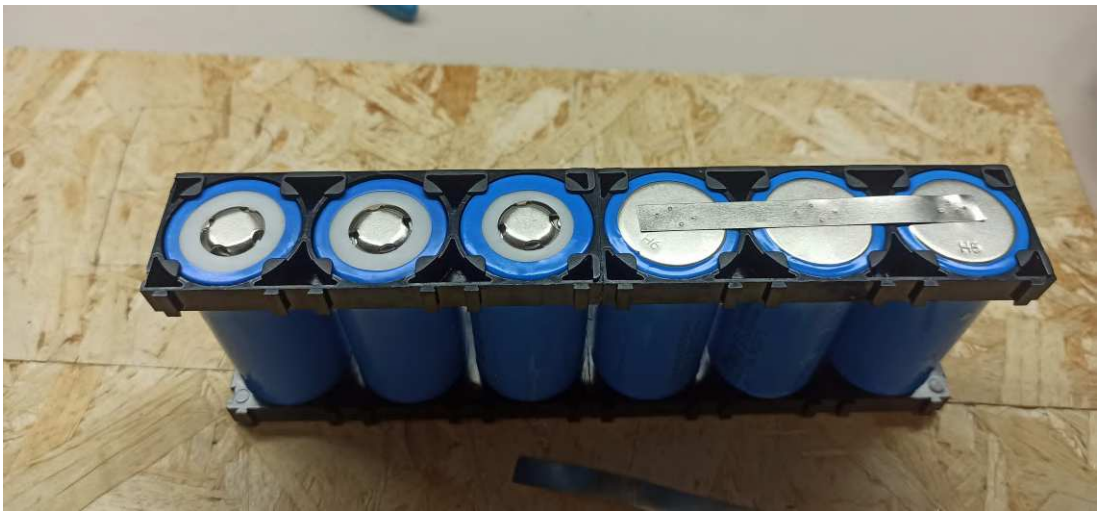


Figura 2.33: Unión de terna de celdas

2. Unión de grupos de celdas

Una vez se han soldado las tres celdas en paralelo que constituirán un paquete, se procedió a soldar en serie todo el conjunto (ver figura 2.34). Para la soldadura entre dos grupos, se usó una tira de níquel a modo de puente entre terminales de la celda. Y una vez se tenían unidas los dos grupos se soldaba con el siguiente y así sucesivamente.

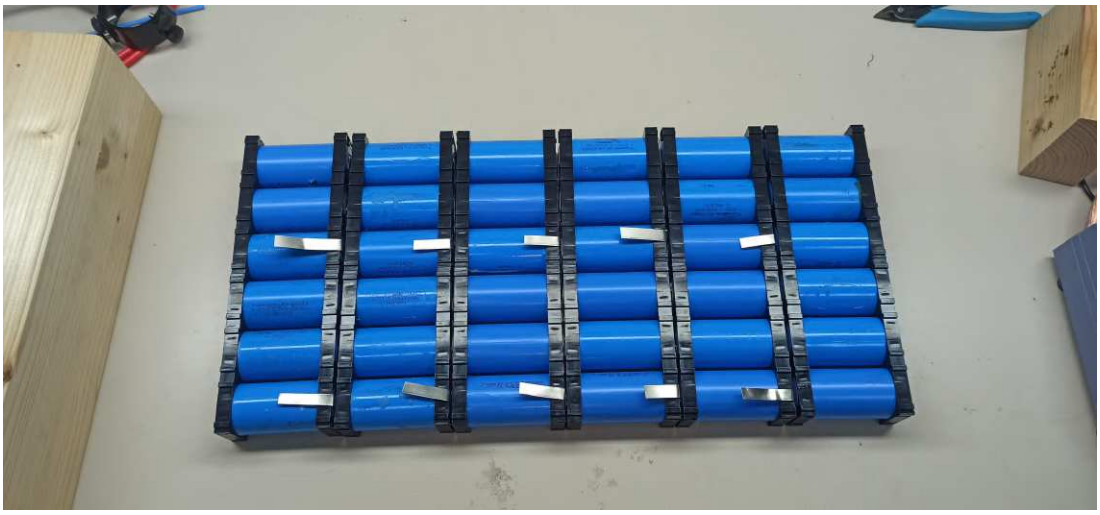


Figura 2.34: Conjunto de celdas unidas

Asimismo, se ha montado una tira de níquel que sobresale de la unión de dos grupos de celdas, donde se soldarán los cables del BMS (Battery Management System).

3. Conexión de la batería con el BMS

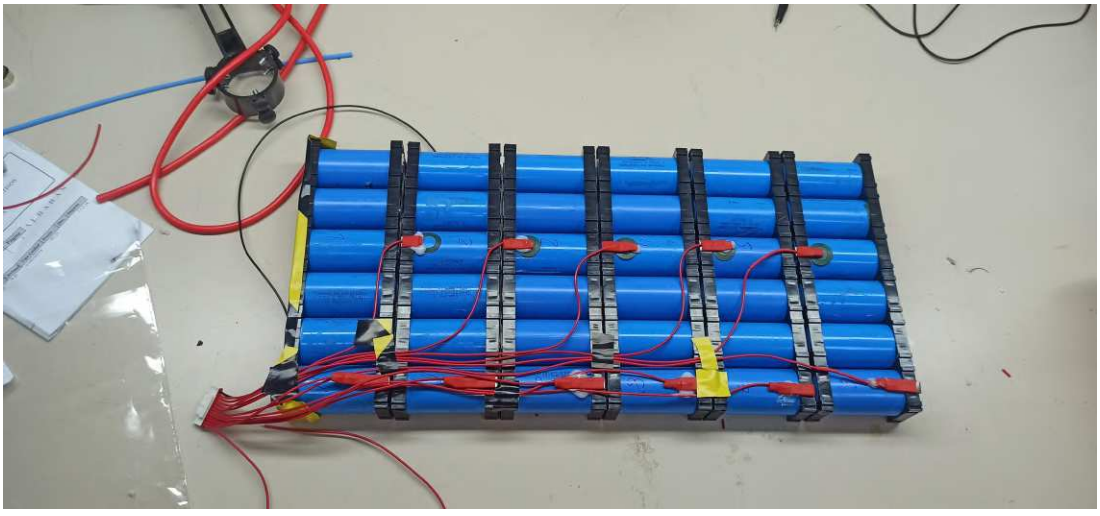


Figura 2.35: Conexión del BMS con las celdas

A continuación, se encuentra el BMS (Battery Management System), que es un componente electrónico integrado en las baterías de litio para evitar accidentes derivados del comportamiento del litio durante sus fases de carga y descarga. Su principal función es ges-

tionar y monitorear cada celda de la batería para asegurar que opere dentro de parámetros seguros y óptimos.

El conexionado del BMS con los conjuntos de 3 celdas se ha realizado utilizando los cables proporcionados por DALY (ver figura 2.35). Es importante destacar que el BMS integra un sensor de temperatura para asegurar que las celdas no se sobrecalienten durante la carga o descarga, ya que el calor excesivo podría causar incendios o explosiones, lo cual es una preocupación clave para el uso del robot. Este ha sido instalado en la parte media de la batería.

Como se explicó anteriormente, las celdas deben mantener una tensión estable de aproximadamente 3.2 V - 3.4 V. Para gestionar esto, el BMS realiza el balanceo de celdas, distribuyendo la energía de manera uniforme entre ellas para que todas alcancen el mismo nivel de carga y descarga. Este proceso mejora la eficiencia y prolonga la vida útil de la batería.

El BMS también protege las celdas cortando la corriente de carga del grupo al que pertenezca cuando una o varias celdas alcanzan su límite de voltaje máximo (sobrecarga) o mínimo (sobredescarga). Esto previene que las celdas se dañen o entren en condiciones inseguras.

Algunos BMS pueden enviar información a otros sistemas de control mediante buses de comunicación como I2C, CAN o UART, lo que permite una supervisión externa del estado de la batería.

Además, este cuenta con una aplicación llamada *Smart BMS* que permite programar el porcentaje máximo de carga de la batería, consultar el porcentaje de carga de cada conjunto, la corriente que está consumiendo en el momento, o los ciclos de carga y descarga completados.

La conexión con el BMS se realiza mediante Bluetooth, permitiendo visualizar los resultados desde un móvil.

Los datos que se muestran en la aplicación incluyen:

- Valor de tensión máximo de las celdas.
- Valor de tensión mínimo de las celdas.
- Valor promedio de voltaje entre las celdas.
- Diferencia de voltaje entre la celda con la tensión mínima y la celda con la máxima.
- Número de ciclos completados de la batería (considerando carga y descarga).
- Potencia que está entregando la batería.
- Medición de temperatura del sensor ubicado en el centro de la batería.

Dentro de la aplicación se puede encontrar lo siguiente (ver figura 2.36):

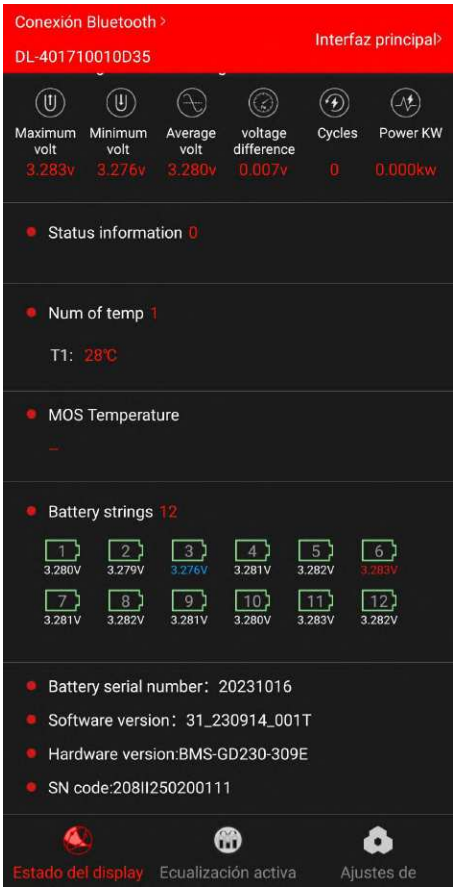


Figura 2.36: Interfaz de la aplicación Smart BMS

Por otro lado, el kit BMS también incorpora una pantalla que será instalada en la parte trasera de Horu, en la cual se muestra el nivel de tensión actual de la batería, la corriente que está suministrando o recibiendo, la temperatura de las celdas, entre otros datos (ver figura 2.37).

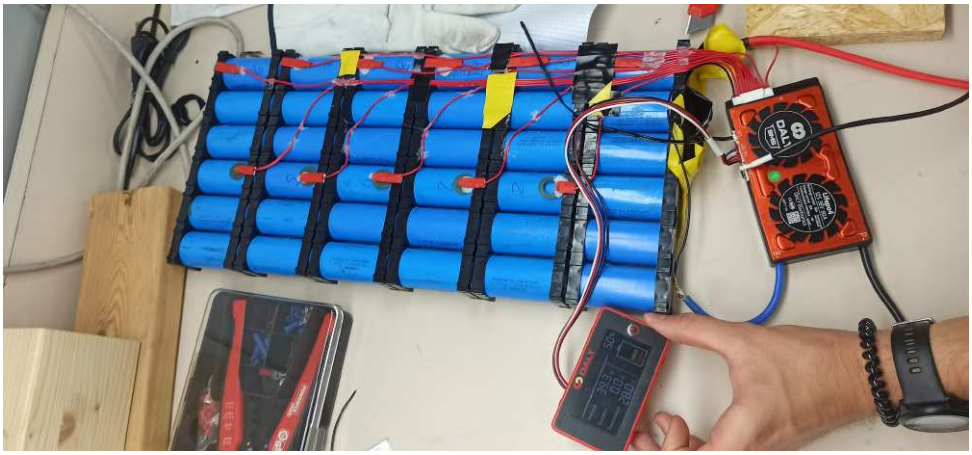


Figura 2.37: BMS y pantalla del BMS que indican el estado de la batería

4. Recubrimiento de la batería con termoretráctil

Para garantizar un correcto aislamiento de la batería del robot, se ha utilizado plástico termoretráctil (ver figura 2.38), lo que proporciona una gran uniformidad al conjunto de la batería y ofrece un grado adicional de aislamiento frente a la humedad y el polvo.

La aplicación de este material sobre la batería se ha realizado mediante el uso de una pistola de aire caliente, que permite que el plástico termoretráctil se reduzca y ajuste perfectamente a la forma de la batería.

Como se puede observar en la figura 2.38, se han extraído dos cables previamente soldados a los terminales positivo y negativo de la batería. Estos cables serán conectados al BMS y a la caja de fusibles, que distribuirá la alimentación necesaria para el funcionamiento del robot.

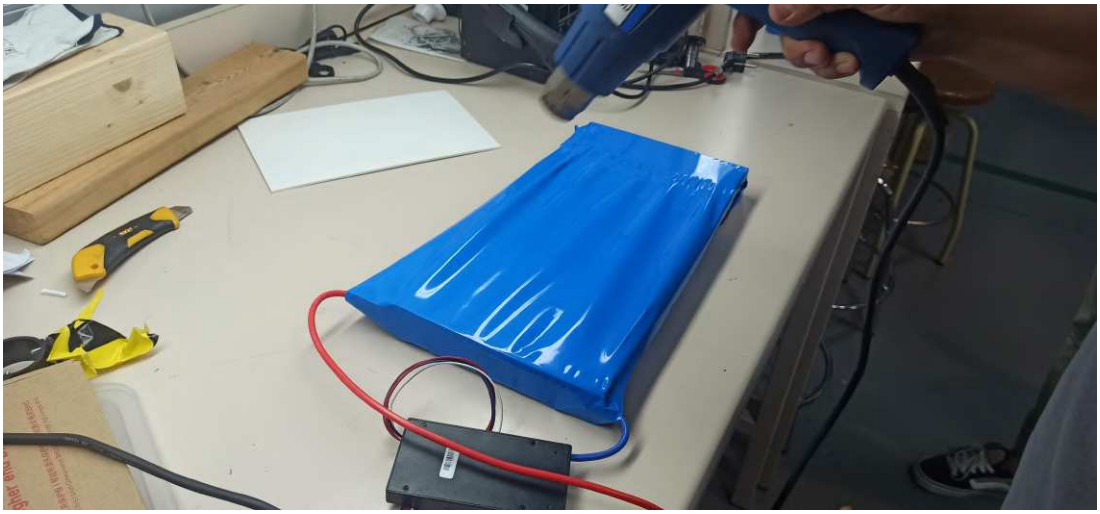


Figura 2.38: Proceso de sellado de la batería mediante termoretráctil

Cargador de la batería:

Para realizar la carga de la batería de Horu se utiliza el siguiente cargador (ver figura 2.39):

Figura 2.39: Cargador de la batería

Este cargador inteligente para baterías de iones de litio es capaz de aportar unos 42V y unos 10A de carga para la batería (ver figura 2.40):

Figura 2.40: Cargador de la batería en proceso de carga

Batería terminada:

Finalmente, en la siguiente figura (ver figura 2.41), se puede observar cómo queda la batería ensamblada en la parte baja del chasis del robot.

La batería será cubierta por el chasis de aluminio del robot, el cual le proporcionará una sujeción adecuada y protección frente a todo tipo de golpes.

Para realizar la carga, se ha instalado en un lateral del robot un conector tipo Anderson Powerpole (PP).

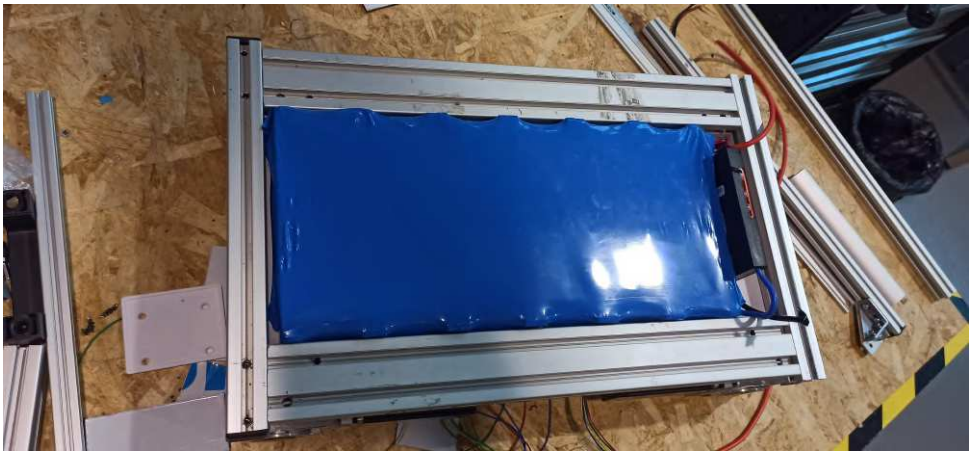


Figura 2.41: Batería posicionada en la parte baja del chasis

Capítulo 3

Esquema eléctrico del sistema de control

3.1 Introducción

En este capítulo se presenta el esquema eléctrico del sistema de control de Horu (ver figura 3.1). Se describen las conexiones de alimentación de los motores y periféricos, así como las conexiones utilizadas para su control mediante las placas de prototipado de los microcontroladores ESP32 y Arduino MKR WiFi 1010. El esquema ha sido elaborado utilizando la herramienta Fritzing, que dispone de una amplia biblioteca de componentes y permite, además, la inserción de diseños personalizados.

3.2 Asignación de pines y conexiones

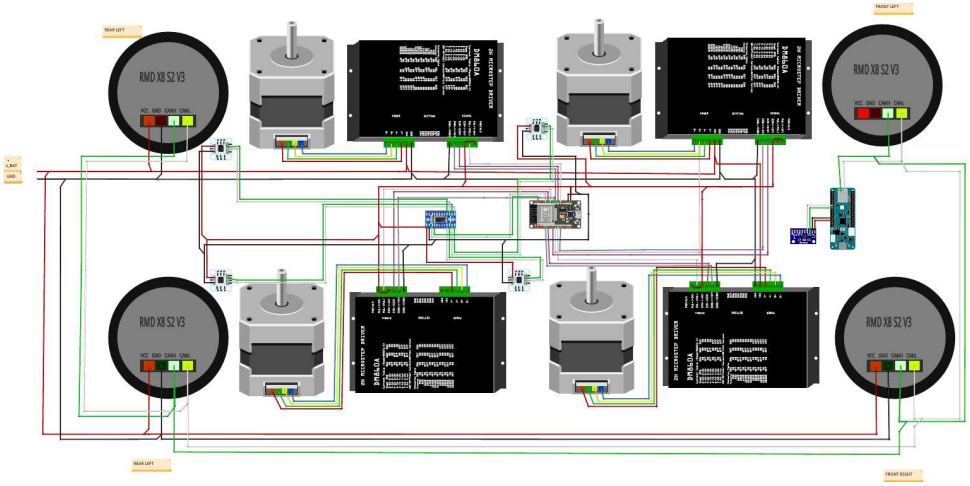


Figura 3.1: Esquema eléctrico del sistema de control de Horu

La alimentación de los ocho motores proviene directamente de la batería, no es necesario ningún convertidor de tensión intermedio. Cada motor está protegido individualmente mediante un fusible, ubicado en una caja de fusibles general.

Los cuatro drivers de potencia (DM542) utilizados para el control de los motores paso a paso disponen de señales de *Enable* independientes, gestionadas por salidas digitales del microcontrolador ESP32. Aunque sería posible unificar todas las señales *Enable* en un único pin, se ha optado por controlarlas de forma individual para permitir una programación más selectiva. Cada señal *Enable* se activa mediante un nivel de tensión alto (HIGH, 3.3V) proporcionado por el ESP32. Por otro lado, la señal de control PWM (Pulse) es común para los cuatro drivers, por lo que los pines PUL- de todos los drivers están conectados a una misma salida digital del ESP32. Por otro lado, los pines de $\mathrm{PUL}+$ están conectados a la salida del regulador de 3.3V que tiene el ESP32.

Los motores de tracción se controlan mediante el bus CAN. Para ello, se ha acoplado una Shield CAN al microcontrolador Arduino MKR WiFi 1010, la cual es alimentada directamente desde el propio microcontrolador. Las conexiones CAN se han realizado respetando el código de colores: cable blanco para $\mathrm{CAN\_H}$ y cable verde para $\mathrm{CAN\_L}$.

Ambos microcontroladores se alimentan de un Hub USB que proporciona unos 5V. El ESP32 suministra a su vez los 3.3V necesarios para alimentar los cuatro encoders magnéticos y el multiplexor por la salida de 3V3.

La comunicación con los encoders y el multiplexor se realiza mediante el protocolo I2C, conectando las líneas del bus I2C (SDA y SCL) a los pines 21 y 22 del ESP32, respectivamente. Se ha utilizado un código de color para facilitar su identificación: blanco para SDA y verde para SCL.

Por último, la IMU está conectada también mediante el bus I2C al microcontrolador Arduino MKR WiFi 1010, utilizando los pines SDA y SCL 11 y 12, respectivamente. Su alimentación, de 5V, es proporcionada directamente por el Arduino MKR WiFi 1010, por el pin llamado VCC.

Todas las masas de los diferentes componentes eléctricos/electrónicos del robot están unidas en la caja de fusibles.

El conexionado de los pines del ESP32 a los drivers de potencia de los motores paso a paso se ha realizado mediante crimpado de terminales (ver figura 3.2).

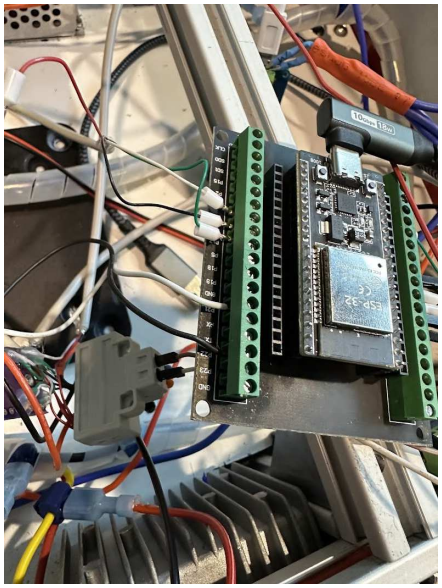


Figura 3.2: Conexión de driver DM542 de la oruga F.L. con ESP32

En lo que respecta a las conexiones entre el multiplexor y los cuatro sensores de posición, se han soldado los terminales I2C de cada sensor a una de las entradas del multiplexor (ver figuras 3.3 y 3.4). Posteriormente, la salida I2C del multiplexor se conecta a la entrada I2C del microcontrolador ESP32.

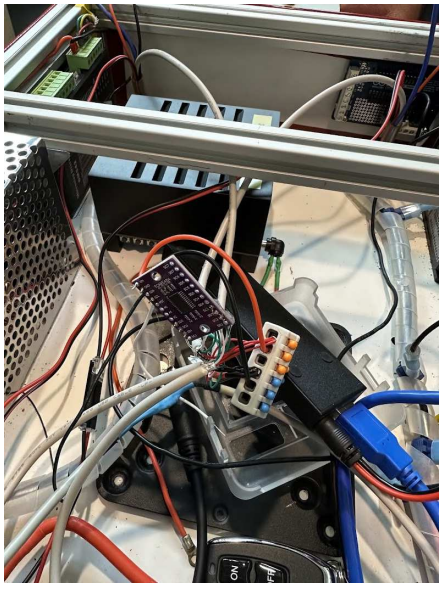


Figura 3.3: Parte delantera del multiplexor

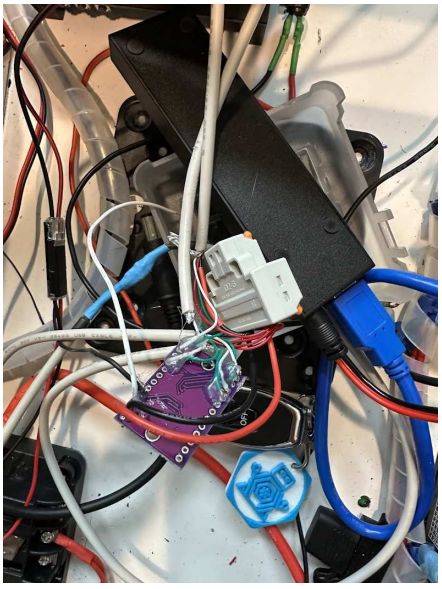


Figura 3.4: Parte trasera del multiplexor

A modo de resumen, tenemos la tabla adjunta al final del capítulo (ver tabla 3.1 y tabla 3.2):

Dispositivo / Sección	Señal	Conexión
RMD X8 S2 V3	CANH	CANH (compartido, SHIELD)
	CANL	CANL (compartido, SHIELD)
IMU MPU9250 (MKR)	SDA	Pin 11
	SCL	Pin 12
	VCC	5v
Front Right DRIVER DM542 (ESP32)	PUL+	3.3V (ESP32)
	DIR+	3.3V
	ENA-	GND
	PUL-	GPIO27
	DIR-	GPIO32
	ENA+	GPIO33
Front Left DRIVER DM542 (ESP32)	PUL+	3.3V
	DIR+	3.3V
	ENA-	GND
	PUL-	GPIO26
	DIR-	GPIO25
	ENA+	GPIO14
Rear Right DRIVER DM542 (ESP32)	PUL+	3.3V
	DIR+	3.3V
	ENA-	GND
	PUL-	GPIO18
	DIR-	GPIO19
	ENA+	GPIO5
Rear Left DRIVER DM542 (ESP32)	PUL+	3.3V
	DIR+	3.3V
	ENA-	GND
	PUL-	GPIO4
	DIR-	GPIO16
	ENA+	GPIO17
Multiplexor (ESP32)	VCC	3.3V
	SDA	GPIO21
	SCL	GPIO22

Tabla 3.1: Tabla de conexionado de la IMU, multiplexor, drivers DM542 y CAN bus motores de tracción

Dispositivo / Sección	Señal	Conexión
Sensor AS5600 Front Right	SDA	Puerto 0 (Multiplexor)
	SCL	Puerto 0 (Multiplexor)
	VCC	3.3V (ESP32)
Sensor AS5600 Front Left	SDA	Puerto 1 (Multiplexor)
	SCL	Puerto 1 (Multiplexor)
	VCC	3.3V (ESP32)
Sensor AS5600 Rear Right	SDA	Puerto 2 (Multiplexor)
	SCL	Puerto 2 (Multiplexor)
	VCC	3.3V (ESP32)
Sensor AS5600 Rear Left	SDA	Puerto 3 (Multiplexor)
	SCL	Puerto 3 (Multiplexor)
	VCC	3.3V (ESP32)

Tabla 3.2: Tabla de conexionado de los sensores AS5600

Capítulo 4

Componentes Software

4.1 Introducción

En este capítulo se describen los distintos programas empleados durante el desarrollo del proyecto, tanto para la programación de un modelo de control para el Arduino MKR WiFi 1010 y el ordenador, como para la implementación de un código en C/C++ para el ESP32. Las dos herramientas principales utilizadas han sido MATLAB y Arduino IDE.

4.2 MATLAB

Figura 4.1: MATLAB

MATLAB es un entorno de programación diseñado específicamente para el cálculo numérico, la visualización de datos y el desarrollo de algoritmos (ver figura 4.1). Su nombre, derivado de ”Matrix Laboratory”, refleja su enfoque en la eficiente manipulación de matrices, fundamental en numerosas aplicaciones de ingeniería. MATLAB permite la creación de scripts y funciones que ejecutan cálculos especificados en el código, ofreciendo múltiples formas de expresar los resultados.

No limitándose a funciones de cálculo, MATLAB es capaz de realizar tareas avanzadas como el reconocimiento de imágenes y el diseño de redes neuronales. En este proyecto, se ha empleado la versión R2024b de MATLAB.

El entorno Simulink (ver figura 4.2), una aplicación integrada en MATLAB, se destaca como la función principal para este proyecto. Simulink no solo hereda las funcionalidades de MATLAB, permitiendo la integración de funciones MATLAB directamente en sus bloques, sino que también facilita la programación mediante un entorno gráfico de bloques. Esto posibilita la creación de programas que, en código convencional, serían considerablemente complejos, como la configuración de redes WiFi y la transmisión continua de datos, con solo unos pocos bloques.

Así como en MATLAB trabajamos con scripts (.m), en Simulink trabajamos con modelos (.slx). Un conjunto organizado de modelos y/o scripts se denomina proyecto, donde se aloja todo el código y la lógica del presente trabajo.

Simulink también dispone de una variedad de add-ons útiles, algunos preinstalados y otros disponibles mediante un gestor de add-ons. Para este proyecto, se ha utilizado el Paquete de Soporte de Arduino para Simulink, facilitando la programación directa de placas Arduino desde Simulink. Además de la programación, este paquete ofrece configuración de velocidad de transmisión de datos, manejo de conexiones WiFi (en placas compatibles) y diversas opciones para la ejecución de programas.

En Simulink, se ha realizado la programación de los diferentes sistemas de control, tanto del ordenador principal del robot como del microcontrolador encargado de los servomotores (Arduino MKR WiFi 1010). Además, ha permitido integrar las comunicaciones entre ambos microcontroladores de manera visual, ya que con herramientas como *Display* y *Scope* se puede visualizar en todo momento los datos que se están leyendo y transmitiendo a los ocho motores. Esto es posible gracias al modo ”Monitor & Tune”de Simulink, que, a diferencia de otros modos de programación, permite el monitoreo en tiempo real de la ejecución del programa del Arduino MKR WiFi 1010. El PC se ejecutará en modo ”Run” (ver figura 4.3), que es el único modo que permite interactuar con el mando de Xbox en tiempo real. En este modo, al igual que en ”Monitor & Tune”(ver figura 4.4), es posible ver en tiempo real el valor de las variables y el estado en el que se encuentra el robot, entre otros aspectos.

Cuando se desee cargar en el Arduino MKR WiFi 1010 el código definitivo o hacer alguna prueba y no sea necesaria la visualización del modelo, el modo a ejecutar será ”Build, Deploy & Start”(ver figura 4.4), que permitirá cargar el código en la ROM de la placa.

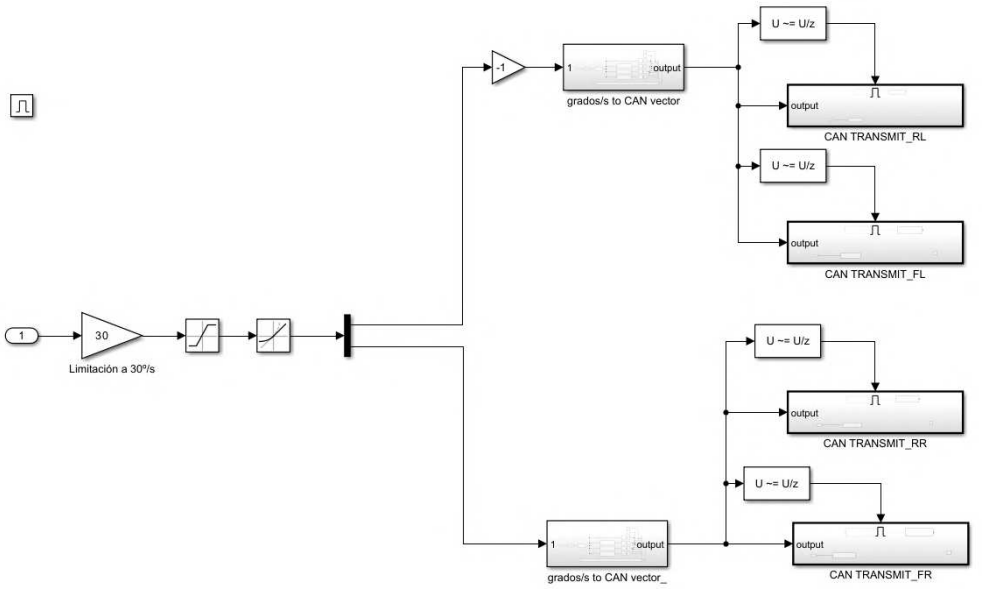


Figura 4.2: Ejemplo del entorno Simulink

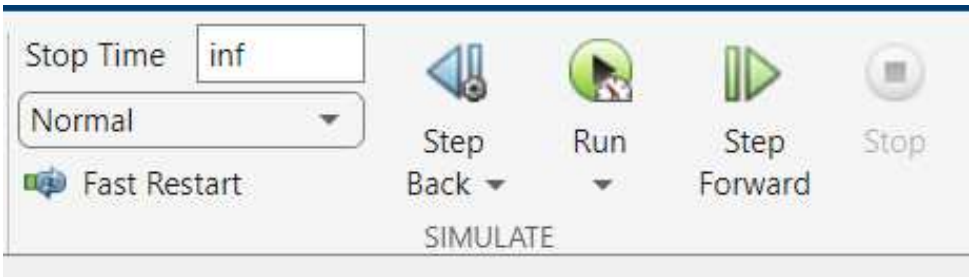


Figura 4.3: Modo Run en Simulink

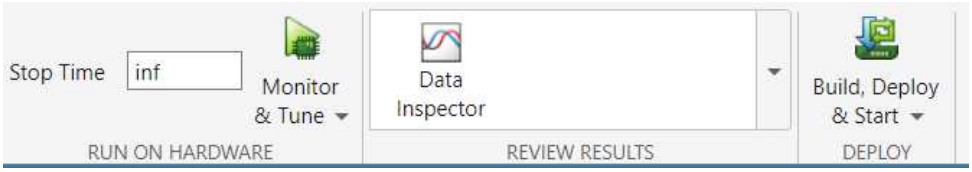


Figura 4.4: Modo Build Deploy Start y Monitor & Tune en Simulink

4.3 Arduino IDE

Figura 4.5: Arduino

Arduino IDE (Integrated Development Environment) es un entorno de desarrollo utilizado para la escritura, compilación y carga de código en placas basadas en la plataforma Arduino (ver figura 4.5). Esta herramienta ha sido diseñada con un enfoque educativo y accesible, convirtiéndose en la vía de entrada para muchas personas al mundo de la electrónica y la programación de microcontroladores. Las placas Arduino, generalmente de bajo costo, junto con un entorno de desarrollo gratuito y de código abierto, hacen de esta plataforma una opción ideal para proyectos académicos y prototipado rápido.

El entorno proporciona un editor de texto optimizado para la escritura de funciones y estructuras de control, empleando un lenguaje de programación basado en C++ (ver figura 4.6), simplificado para facilitar su comprensión. Además, el Arduino IDE incluye un compilador integrado que analiza el código en busca de errores sintácticos o semánticos, ofreciendo sugerencias para su corrección.

Una vez verificado y compilado, el código se transfiere directamente a la placa mediante una conexión USB, donde es almacenado y ejecutado. Esta simplicidad en el flujo de trabajo —escribir, compilar y cargar— permite un desarrollo iterativo ágil, esencial en etapas de prueba y validación de sistemas embebidos.

El microcontrolador programado en este entorno ha sido el ESP32. Arduino IDE ofrece una amplia variedad de librerías que han facilitado el desarrollo, como por ejemplo ‘Wire.h‘, que permite establecer comunicación I2C con el multiplexor, y ‘WiFiUdp.h‘, utilizada para el envío y recepción de datagramas mediante el protocolo UDP a través de WiFi.

La versión de Arduino IDE usada en este proyecto es la 2.3.3, y la placa seleccionada para programar es la ”DOIT ESP32 DEVKIT V1” (ver figura 4.7).

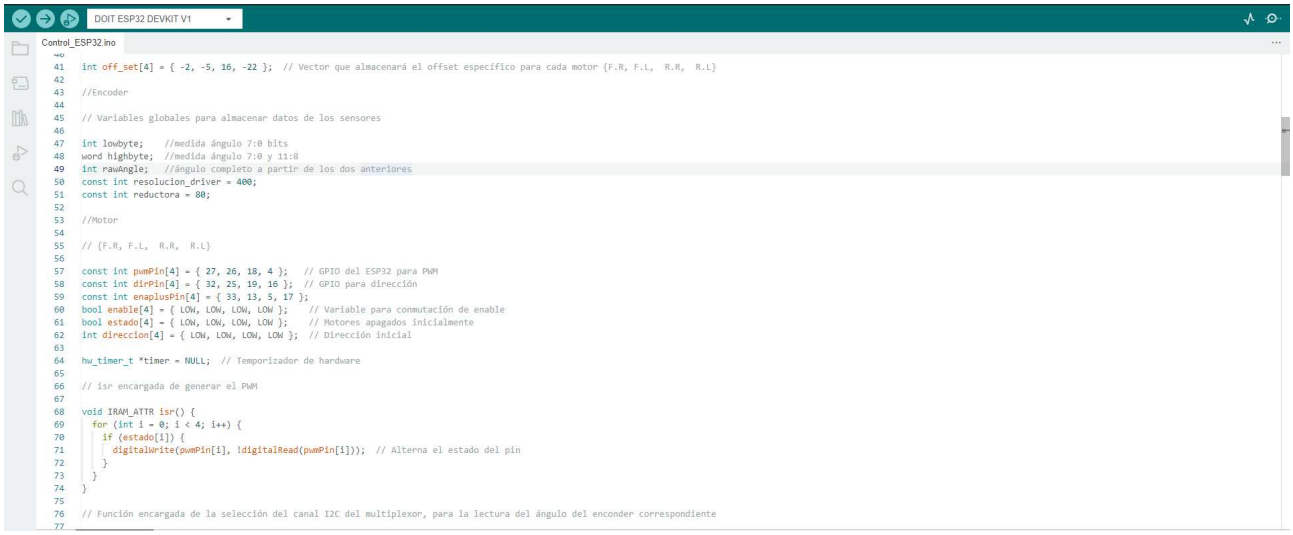


Figura 4.6: Entorno del Arduino IDE

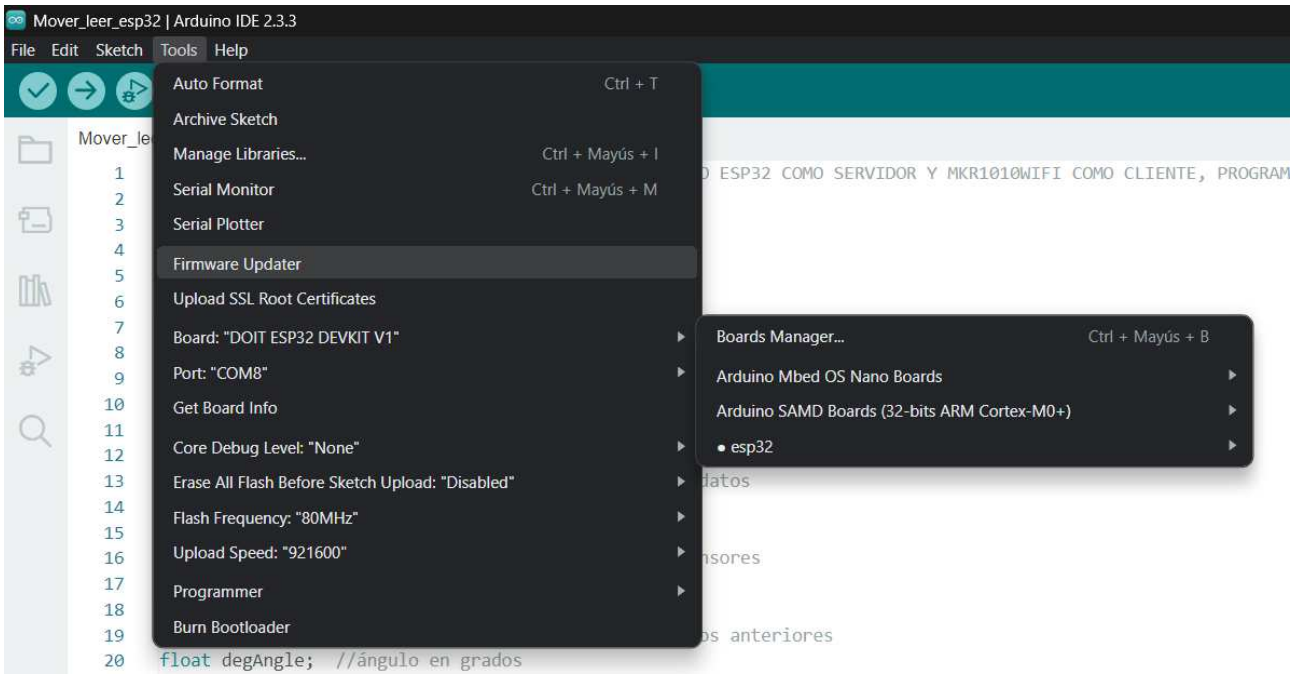


Figura 4.7: Selección de la placa microcontroladora ESP32

Capítulo 5

Sistema de control del vehículo robótico

5.1 Introducción

En el presente capítulo se describe la arquitectura de comunicaciones y el funcionamiento del sistema de control de movimiento implementado en el robot de orugas articuladas desarrollado en el marco de este Trabajo de Fin de Grado. El objetivo principal del capítulo es ofrecer una visión general de cómo se coordinan los distintos elementos del sistema.

El robot está compuesto por cuatro orugas articuladas, accionadas mediante un total de ocho motores: cuatro servomotores encargados del desplazamiento, y cuatro motores paso a paso dedicados al movimiento de balanceo de la articulación de las orugas, como se ha descrito anteriormente. La programación y gestión de estos motores se distribuye entre dos microcontroladores: un Arduino MKR WiFi 1010 y un ESP32, cada uno responsable del control de un subconjunto de actuadores. El Arduino MKR WiFi 1010 ha sido programado íntegramente en Simulink y el ESP32 con Arduino IDE.

La coordinación entre ambos procesos de control se realiza mediante el protocolo de comunicación WiFi UDP, a través del cual se transmiten paquetes de datos que contienen las instrucciones de movimiento y datos sobre el estado en el que se encuentran los diferentes procesos.

La red WiFi de 2.4 GHz es creada por el propio robot mediante un router integrado.

La provisión de las consignas de movimiento del sistema se realiza mediante un ordenador (PC), en el cual se ejecuta un modelo desarrollado en Simulink. Dicho modelo tiene una serie de bloques que permiten conectar el PC con un mando de videojuegos tipo Xbox, utilizado como dispositivo de interfaz de usuario. A través de este modelo de programación, las órdenes de movimiento generadas por el usuario son interpretadas, procesadas y transmitidas a los procesos de control que se ejecutan en los microcontroladores.

Los controles del mando de Xbox de los cuales se adquirirán las consignas son los de la siguiente figura (ver figura 5.1):

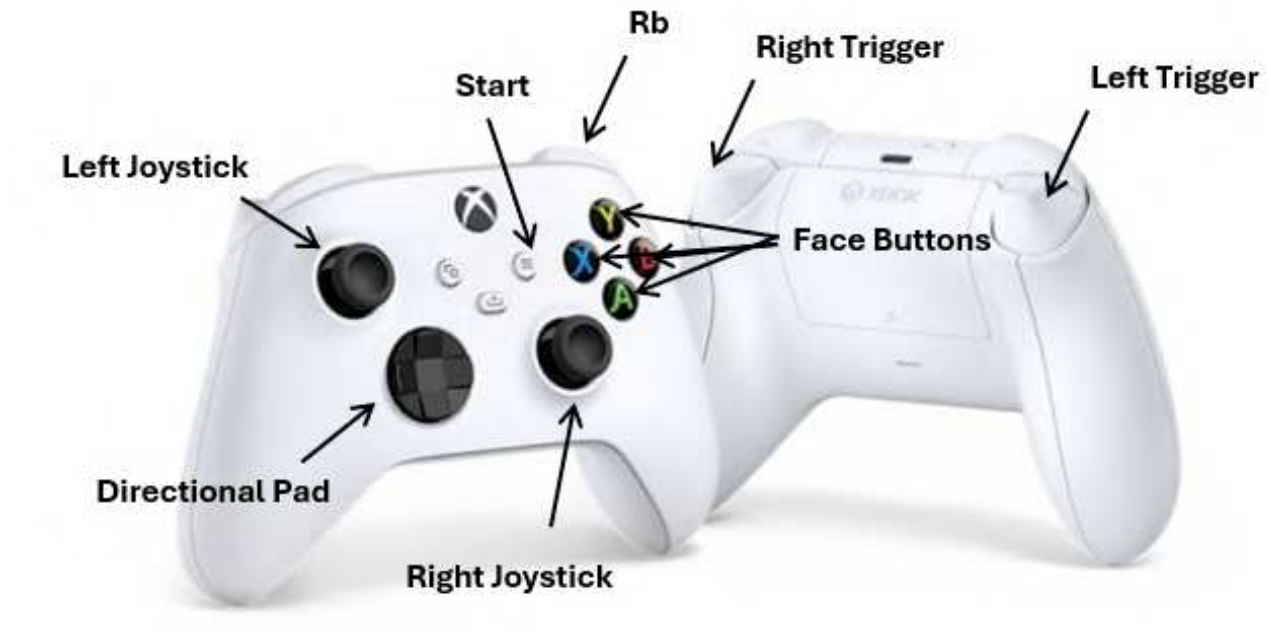


Figura 5.1: Controles del mando de Xbox

- **Rb:** El botón *Rb* será utilizado para cambiar el modo de funcionamiento del robot mediante sucesivas pulsaciones de este.
- **Left/Right** trigger**:** Estos dos gatillos analógicos permitirán controlar el sentido de giro de las orugas en el modo de funcionamiento de control incremental por pares (modo 4). El trigger derecho hará que el par de orugas seleccionado gire en sentido horario, mientras que el trigger izquierdo lo hará en sentido antihorario.
- **Left** joystick**:** Este joystick analógico controlará los motores de tracción en el modo de funcionamiento 1 para el desplazamiento lineal. Al posicionar el joystick hacia delante, el robot avanzará; si se posiciona hacia atrás, el robot retrocederá.
- **Right** joystick**:** Este joystick analógico servirá para controlar los motores de tracción en el modo 1 para el desplazamiento angular. Si el joystick se posiciona hacia la derecha, el robot girará hacia la derecha mediante un control diferencial de los motores; si se posiciona hacia la izquierda, el giro será hacia la izquierda.
- **Direction Pad:** El *Direction Pad* o cruceta está compuesto por cuatro botones digitales que permiten seleccionar el par de orugas a controlar: cruceta arriba para el par delantero, cruceta abajo para el par trasero, cruceta derecha para el tren derecho y cruceta izquierda para el tren izquierdo. Con la ayuda de los triggers se comandará el sentido de giro del par seleccionado cuando el modo de funcionamiento sea el 4.

  En el modo de funcionamiento 3, dependiendo del botón pulsado de la cruceta, se combinará el movimiento de las cuatro orugas para inclinar la plataforma robótica conjuntamente: cruceta arriba para inclinar hacia delante, cruceta abajo hacia detrás, cruceta izquierda hacia la izquierda y cruceta derecha hacia la derecha.

- **Face buttons:** Estos cuatro botones digitales (A, B, X, Y) permitirán seleccionar el ángulo de posicionamiento de las orugas mediante el control de posición absoluta del robot (modo 2). Las posiciones asignadas son respectivamente 225º, 180º, 90º y 135º.
- **Start:** Este botón activa el modo de nivelado automático mientras se mantenga pulsado, como medida de seguridad; al soltarlo, el proceso se detiene de inmediato.

5.2 Sistema de comunicaciones

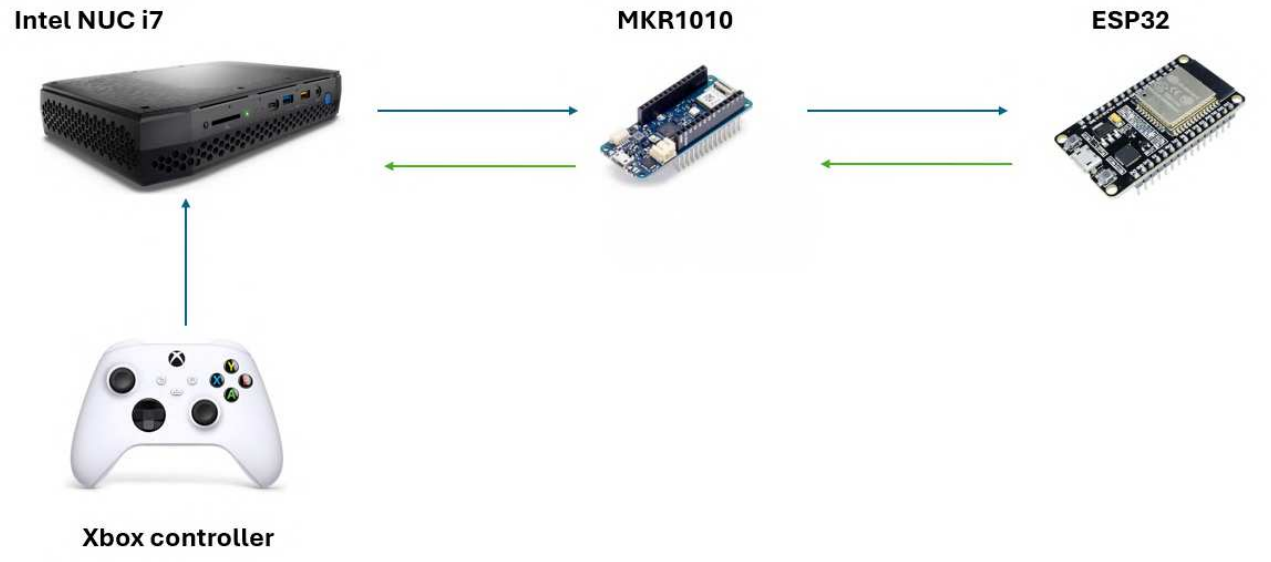


Figura 5.2: Diagrama del sistema de comunicaciones de Horu

El sistema de comunicaciones interno de Horu (ver figura 5.2) constituye un pilar fundamental para la coordinación entre los distintos módulos de control. Dada la arquitectura empleada, basada en dos microcontroladores —un Arduino MKR WiFi 1010 y un ESP32—, ha sido necesario implementar un mecanismo de intercambio de información eficiente y fiable que garantice la correcta ejecución de las órdenes de movimiento, así como la sincronización de los motores.

Para ello, se ha optado por el uso del protocolo UDP (User Datagram Protocol), un protocolo perteneciente al nivel de transporte del modelo OSI, situado entre la capa de red y la capa de aplicación. Se trata de un protocolo no orientado a conexión, lo que permite el envío de datagramas de forma rápida y directa sobre redes IP, sin necesidad de establecer previamente una conexión entre el emisor y el receptor. Cada datagrama UDP incluye en su cabecera toda la información necesaria para llegar a su destino, como las direcciones IP y los puertos de origen y destino.

Debido a su bajo retardo de transmisión y simplicidad, UDP resulta especialmente adecuado para aplicaciones donde la velocidad de respuesta es prioritaria frente a la integridad total de los datos, como ocurre en el sistema de control robótico implantado. No obstante,

su uso conlleva ciertas limitaciones, ya que no garantiza la entrega de mensajes, carece de control de flujo y no incluye mecanismos de confirmación de recepción, lo que implica la posibilidad de pérdida o asincronía de los paquetes.

Para paliar estas deficiencias, en el presente proyecto se ha implementado un sistema de notificación y gestión de errores a nivel de aplicación, mediante el cual cada microcontrolador puede enviar códigos de error predefinidos. Esto permite que cualquier dispositivo conectado a ese puerto pueda detectar condiciones de fallo. Por ejemplo, si el ESP32 detecta un fallo en la lectura de alguno de los encoders, se envía un mensaje de error al Arduino MKR WiFi 1010 y al PC para su correspondiente tratamiento.

A través del protocolo UDP se transmiten paquetes de comunicación que contienen instrucciones de control, generadas por un programa central ejecutado en un PC. También se establece una conexión entre los dos microcontroladores; concretamente, el Arduino MKR WiFi 1010 es responsable de enviar al ESP32 tanto el modo de funcionamiento como los datos específicos que debe ejecutar, actuando como nodo coordinador dentro del sistema. Y el ESP32 enviará al Arduino MKR WiFi 1010 el valor de lectura de los encoders.

A cada dispositivo se le ha asignado una dirección IP específica, ya que el robot lleva integrado un router, como se ha comentado anteriormente. Por lo tanto, se dispone de direcciones estáticas configurables:

- Ordenador Intel Nuc i7: 192.168.10.124
- Microcontrolador ESP32: 192.168.10.102
- Microcontrolador Arduino MKR WiFi 1010: 192.168.10.101

5.3 Modos de funcionamiento del robot

Con el objetivo de dotar al sistema de una mayor flexibilidad operativa y adaptabilidad a distintas condiciones de trabajo, el vehículo robótico ha sido diseñado para operar bajo diferentes modos de funcionamiento, cada uno de los cuales configura un comportamiento específico del sistema de control y de los actuadores (motores). Estos modos son seleccionados desde el mando, enviados desde el PC al microcontrolador principal (Arduino MKR WiFi 1010) y este al ESP32.

Los diferentes modos de funcionamiento del robot, que serán seleccionados mediante el botón Rb del mando Xbox, son los siguientes:

- Modo de tracción (1): En este modo, el control mediante los dos joysticks permite enviar órdenes de movimiento a los cuatro motores de tracción. Se ha implementado un sistema de control diferencial, en el cual el joystick izquierdo controla la velocidad de desplazamiento hacia adelante o atrás del robot, mientras que el joystick derecho gestiona la diferencia de velocidad y el sentido para permitir el giro del robot. En este modo, el microcontrolador ESP32 no interviene, ya que el control de los motores de tracción recae exclusivamente sobre el Arduino MKR WiFi 1010.

- Modo de control de posición (2): En el segundo modo, mediante los botones X, Y, A y B, se podrá situar al robot en unas posiciones angulares predefinidas (90º,135º,180º,225º). Para ello el mensaje a transmitir esta vez irá tanto al Arduino MKR WiFi 1010 como al ESP32, ya que es necesario la actuación de los motores de elevación conjuntamente con los de tracción. En el siguiente capítulo se explica en detalle el mensaje que se transmite para llevar a cabo este modo de funcionamiento.

- Modo de control incremental (3,4): En el modo 3, la cruceta del mando permite nivelar el robot de manera manual. Según el botón pulsado, el robot se inclinará en la dirección indicada por la cruceta, moviendo las cuatro orugas a la vez. A diferencia del modo anterior, el modo 4, que forma parte del modo incremental, permite un control independiente de los ejes delantero y trasero, y del tren derecho y tren izquierdo (el tren izquierdo son las orugas delantera izquierda y trasera izquierda y el tren derecho son las orugas delantera derecha y trasera derecha). Esto se logra utilizando los $\mathrm{trigger}$ para aumentar o disminuir la posición del par de orugas seleccionado mediante la cruceta. En el siguiente capítulo se explica en detalle el mensaje que se transmite para llevar a cabo este modo de funcionamiento.

- Modo de nivelado automático (5): En este modo, aprovechando el control incremental de los dos modos anteriores (3 y 4) y utilizando la IMU, el robot será capaz de nivelarse automáticamente. Este proceso de nivelación se activará mientras el usuario mantenga presionado el botón de Start. Esta funcionalidad es especialmente crítica en terrenos irregulares, ya que garantiza que el robot se mantenga estable, para poder actuar con el futuro brazo robótico.

5.4 Funciones del PC

El ordenador del robot es el encargado de adquirir los datos necesarios para el control del robot mediante un mando de Xbox. Ejecutará un modelo de Simulink que permitirá gestionar los modos de funcionamiento, los valores de los $\mathrm{trigger}$, las consignas de posición y los valores de los $\mathrm{joysticks}$, entre otros. Este ordenador será el punto de integración directa entre el usuario y el robot. Además, se encargará de procesar los valores de entrada provenientes del mando y enviarlos al Arduino MKR WiFi 1010 mediante WiFi y protocolo UDP para su posterior ejecución.

5.5 Funciones de los microcontroladores

5.5.1 Arduino MKR WiFi 1010

El Arduino MKR WiFi 1010 actúa como controlador principal en el vehículo robótico, siendo responsable de la propagación de las acciones a ejecutar hacia el ESP32. Además, gracias a la Shield CAN, se encarga del control de los cuatro motores de tracción.

Este microcontrolador recibe tramas UDP desde el PC y las procesa para determinar el modo de funcionamiento del robot (como se explicó anteriormente), así como los datos necesarios para ejecutar dicho modo. Además, también recibe tramas UDP del ESP32 con los datos de posición angular de las orugas.

Por otro lado, adquirirá la información de la IMU MPU9250, para el control del nivelado automático.

5.5.2 ESP32

El ESP32 es el microcontrolador secundario del robot, encargado de leer las cuatro posiciones angulares de los motores de elevación gracias al multiplexor. Además, se encarga de controlar los drivers de los motores paso a paso, procesando los mensajes recibidos del Arduino MKR WiFi 1010.

Adicionalmente, los datos de los codificadores de posición se transmiten al Arduino MKR WiFi 1010 mediante tramas UDP de nuevo, permitiendo que este último realice las correcciones necesarias en el movimiento de los motores de tracción, contribuyendo así a la precisión y estabilidad del desplazamiento de las orugas del robot.

Capítulo 6

Programación del control PC

6.1 Introducción

En el presente capítulo se expone el modelo completo que se ejecuta en ordenador (PC). Este modelo se encarga de procesar las señales de entrada provenientes del mando Xbox, conectado al ordenador a través de Bluetooth. A continuación, los datos son transmitidos al microcontrolador Arduino MKR WiFi 1010 mediante comunicación inalámbrica WiFi utilizando el protocolo UDP, con el objetivo de controlar los actuadores del sistema, específicamente los motores.

La trama creada para enviar desde el PC hacia el Arduino MKR WiFi 1010 es la siguiente:

DATAGRAMA	**DATO 1**	**DATO 2**	**DATO 3**	**DATO 4**	**DATO 5**
Variable	id	action_left_train	action_right_train	o0	o1

DATAGRAMA	**DATO 6**	**DATO 7**	**DATO 8**	**DATO 9**	**-**
Variable	o2	o3	mode	code_error	-

Tabla 6.1: Datagrama de datos recibidos del PC por Arduino MKR WiFi 1010

Como se verá en el desarrollo del capítulo, se envían 9 datos del tipo int16 (entero con signo de 16 bits). El significado de cada byte se detalla a medida que se va describiendo el modelo.

6.2 Vista general del modelo

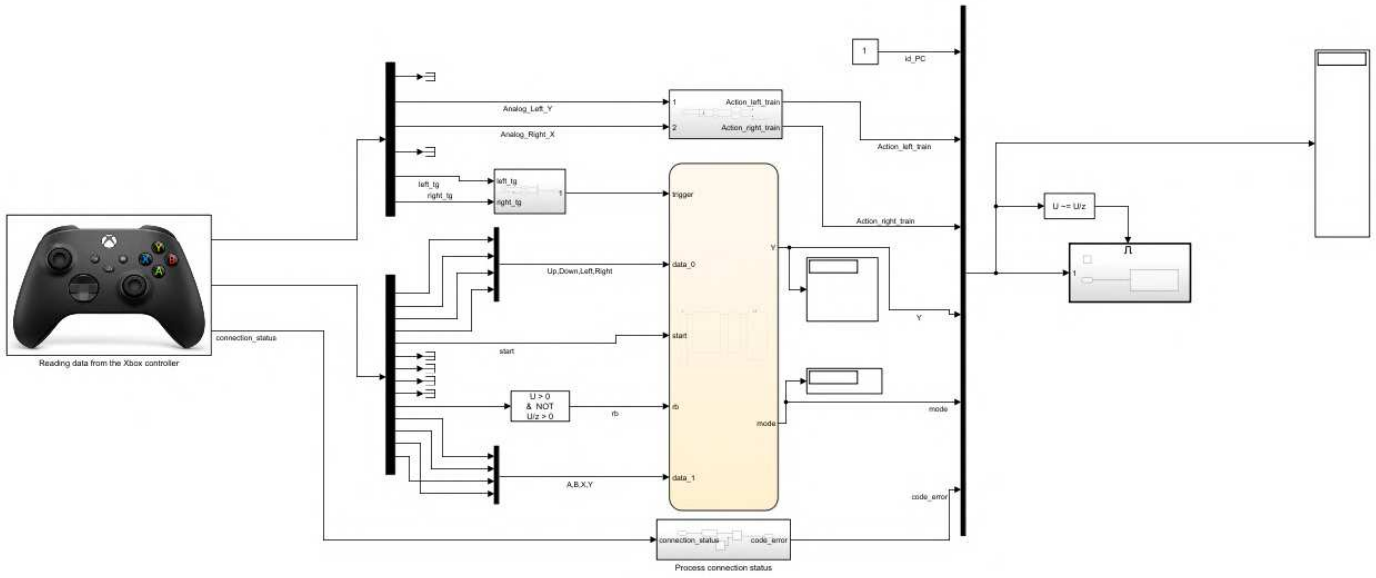


Figura 6.1: Modelo del Control PC

Este modelo (ver figura 6.1) se fundamenta en una serie de subsistemas que, en primer lugar, adquieren los datos del mando de Xbox (inputs). Posteriormente, otros subsistemas se encargan de realizar un paquete de 9 datos $\mathrm{int16}$, dependiendo de las acciones que el usuario que maneja el robot esté realizando, mientras que un último subsistema se encarga de enviar este paquete mediante WiFi UDP hacia el Arduino MKR WiFi 1010. Por consiguiente, el funcionamiento del sistema puede dividirse en tres partes fundamentales: adquisición, procesamiento y transmisión.

6.3 Adquisión de datos

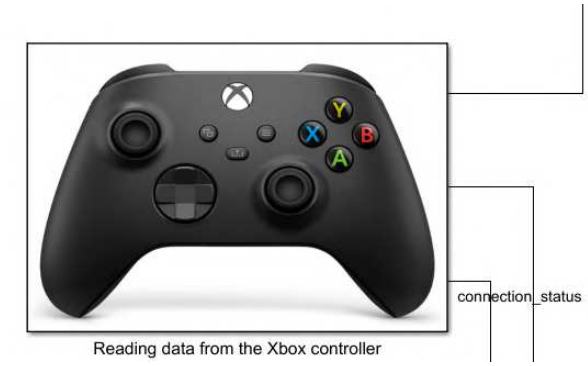


Figura 6.2: Subsistema Xinput Controller

Simulink incorpora un complemento (*Add-On*) que permite la conexión de un mando Xbox al PC a través de Bluetooth (ver figura 6.2). Este subsistema incluye una S-Function (ver figura 6.3) que retorna vectores de tamaño 6, 14 y 3, los cuales representan distintos valores de entrada en función del botón, joystick o trigger que se esté accionando.

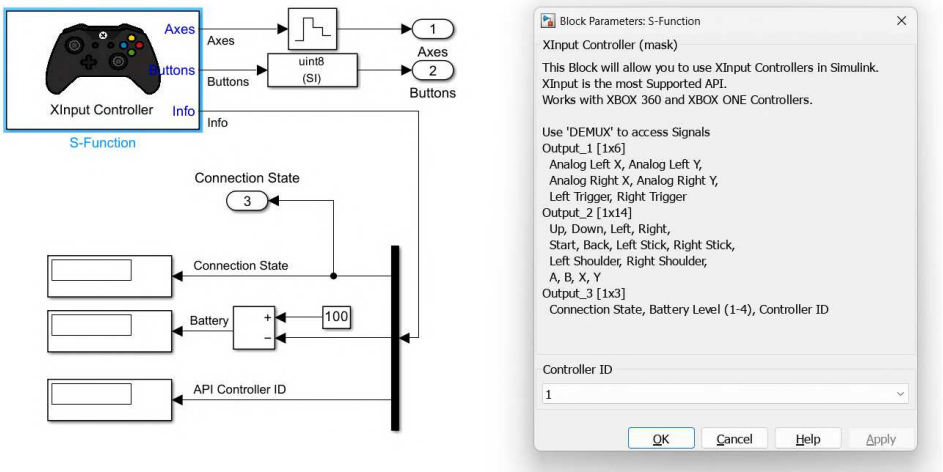


Figura 6.3: S-Function Xbox

Además, devuelve el estado de conexión del mando, el cual será utilizado para propagarlo en caso de conexión o desconexión hacia los microcontroladores, así como el porcentaje de batería disponible.

6.4 Procesamiento de las entradas

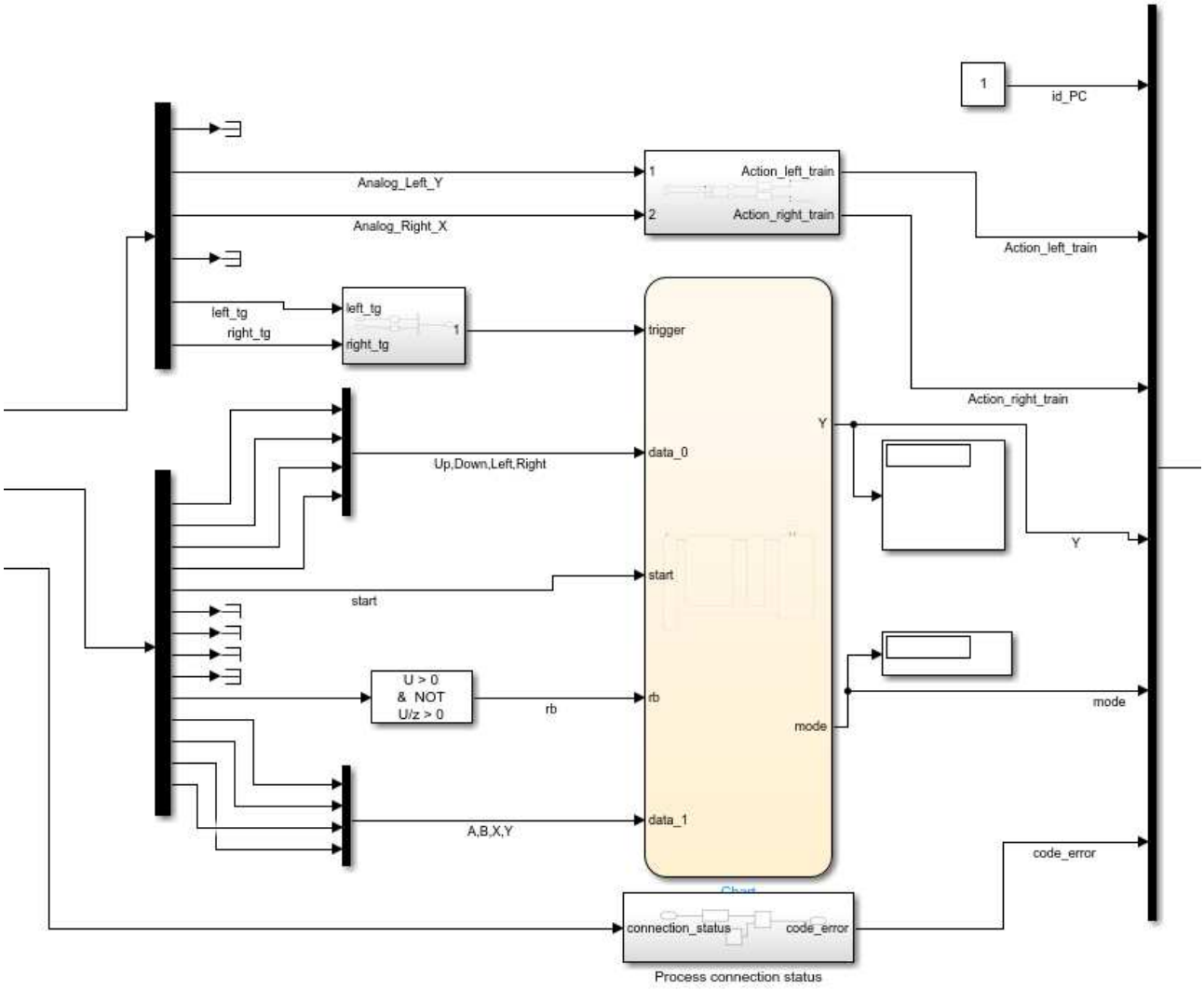


Figura 6.4: Subsistemas encargados del procesamiento de las entradas

El siguiente paso en el proceso de control del robot consiste en interpretar la intención del usuario, en función de las entradas recibidas desde el mando y del modo de operación seleccionado (ver figura 6.4). A partir de esta información, el sistema generará la trama de datos correspondiente para su transmisión.

Los dos primeros bytes del mensaje contienen el identificador del emisor (id_PC), que en este caso corresponde al PC, representado por el valor 1. Esta identificación resulta imprescindible, dado que el microcontrolador Arduino MKR WiFi 1010 recibirá datos simultáneamente tanto del ESP32 como del PC a través del mismo puerto UDP. Por lo tanto, es necesario distinguir el origen de cada mensaje para su correcto procesamiento.

A continuación, los siguientes cuatro bytes del mensaje están destinados a representar

las velocidades del tren izquierdo (orugas *Front Left* y *Rear Left*) y del tren derecho (orugas *Front Right* y *Rear Right*) del robot.

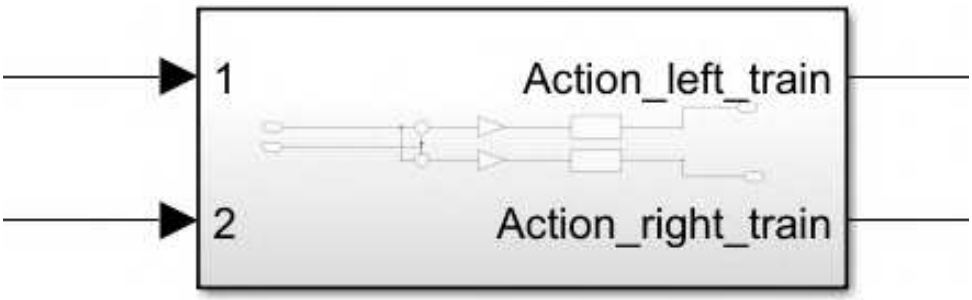


Figura 6.5: Subsistema encargado de la adquisición de consignas de movimiento de los motores de tracción

El subsistema representado en la figura anterior (ver figura 6.5) tiene como objetivo determinar la velocidad y dirección de movimiento de los motores de tracción cuando el sistema opera en el *modo 1*. El control de dichos motores se realiza a través de los dos joysticks del mando, permitiendo un control combinado de la velocidad lineal y angular del robot.

El joystick izquierdo, al desplazarse hacia adelante, genera un movimiento de avance del robot; mientras que al desplazarse hacia atrás, produce un movimiento de retroceso. Por su parte, el joystick derecho controla el giro: al moverse hacia la izquierda, provoca un desplazamiento del robot hacia ese lado, y al moverse hacia la derecha, el desplazamiento será hacia la derecha.

El subsistema encargado de gestionar este comportamiento se compone de los siguientes elementos (ver figura 6.6):

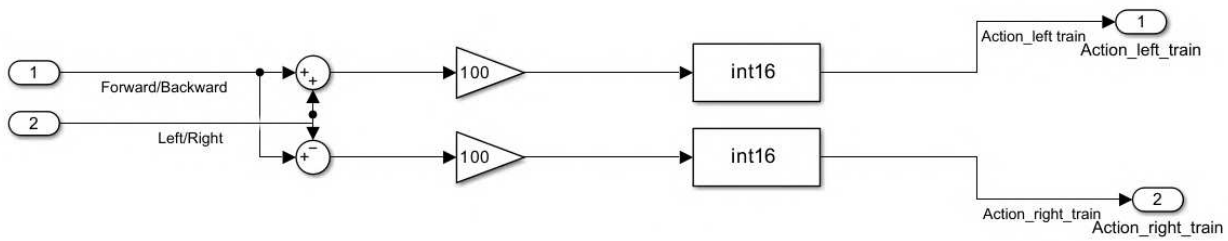


Figura 6.6: Interior del subsistema de movimiento del modo de tracción

Dado que los datos de entrada son valores analógicos de tipo int8 (entero de 8 bits), comprendidos en el rango de $[-1, 1]$, se aplican operaciones de suma y resta sobre dichos

valores con el fin de generar las señales de control para la variable de actuación (velocidad) de los cuatro motores de tracción, los cuales se agrupan en dos subconjuntos: tren derecho y tren izquierdo. Para incrementar la precisión en el control, los datos generados se multiplican por 100 y se convierten a tipo $\mathrm{int16}$ antes de su transmisión.

Los siguientes 10 bytes de la trama a transmitir corresponden a las consignas de actuación de los motores (variable de salida Y, con 8 bytes) y al modo de funcionamiento (variable mode con 2 bytes).

Para la implementación del control de posición, tanto absoluto como incremental, se emplea el diagrama de subsistemas mostrado a continuación (*Subsystem Chart*) (ver figura 6.7), el cual permite la gestión del sistema mediante un enfoque basado en estados.

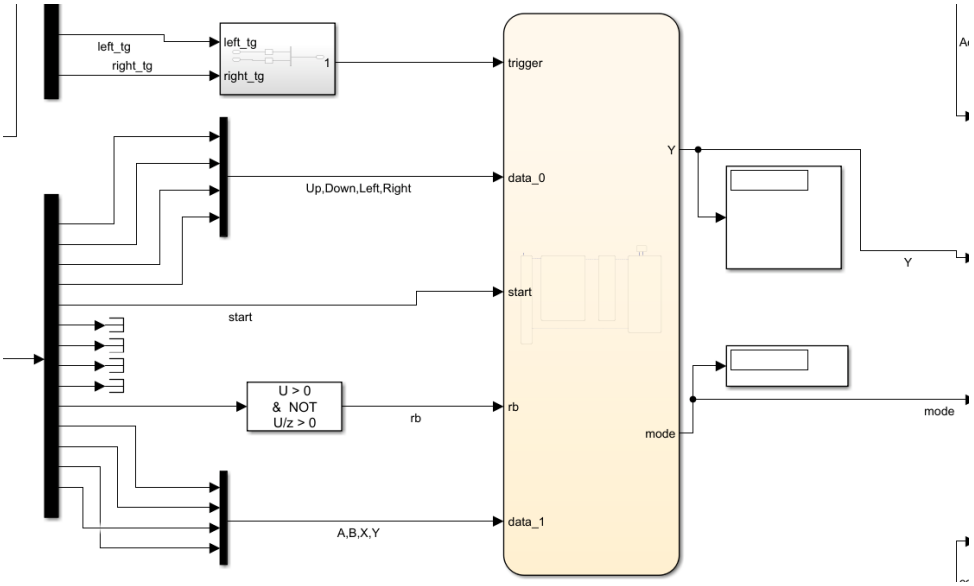


Figura 6.7: Subsistemas encargados del procesamiento de la variable Y y del modo de operación

Las entradas al *Subsystem Chart* son las siguientes:

- $\mathrm{trigger}$: Vector de dos posiciones de tipo $\mathrm{uint8}$ (entero sin signo de 8 bits), que indica el estado (pulsado/no pulsado) de los *triggers* izquierdo y derecho del mando (ver figura 6.8).
- $\mathrm{data\_0}$: Vector de cuatro posiciones de tipo $\mathrm{int8}$, donde cada componente toma el valor 1 o 0 en función de si se está pulsando uno de los botones de la cruceta direccional (*Up*, *Down*, *Left*, *Right*).
- start: Variable de tipo $\mathrm{int8}$ que, mientras permanezca activa (botón pulsado), permite la transición al modo de nivelado automático.
- rb: Variable de tipo $\mathrm{int8}$ asociada al botón *RB*, cuya activación permite avanzar de forma secuencial entre los distintos modos de operación. Para detectar únicamente el flanco ascendente (cuando el valor cambia de bajo a alto), se emplea un bloque $\mathrm{Detect\ Rise\ Change}$ en el flujo del sistema.

- $\mathrm{data\_1}$: Vector de cuatro elementos de tipo $\mathrm{int8}$, donde cada componente toma el valor 1 o 0, dependiendo de si se pulsa el botón A, B, X o Y.

La salida del sistema está constituida por un vector de cuatro posiciones, denotado como Y, compuesto por datos de tipo $\mathrm{int16}$. Dependiendo del estado activo en el sistema de control, los elementos de este vector representarán bien valores de posición objetivo o bien el sentido de giro de los motores.

Adicionalmente, se transmite el modo de operación actual. De este modo, los ocho siguientes bytes del mensaje corresponden a la variable de salida Y, mientras que los dos bytes posteriores contienen la información relativa al modo de funcionamiento.

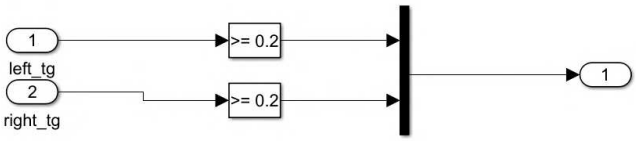


Figura 6.8: Interior del subsistema del procesamiento de los triggers

Ahora se explicará la máquina de estados finitos diseñada para controlar el vector de datos Y, y el modo de funcionamiento (mode) (ver figura 6.9):

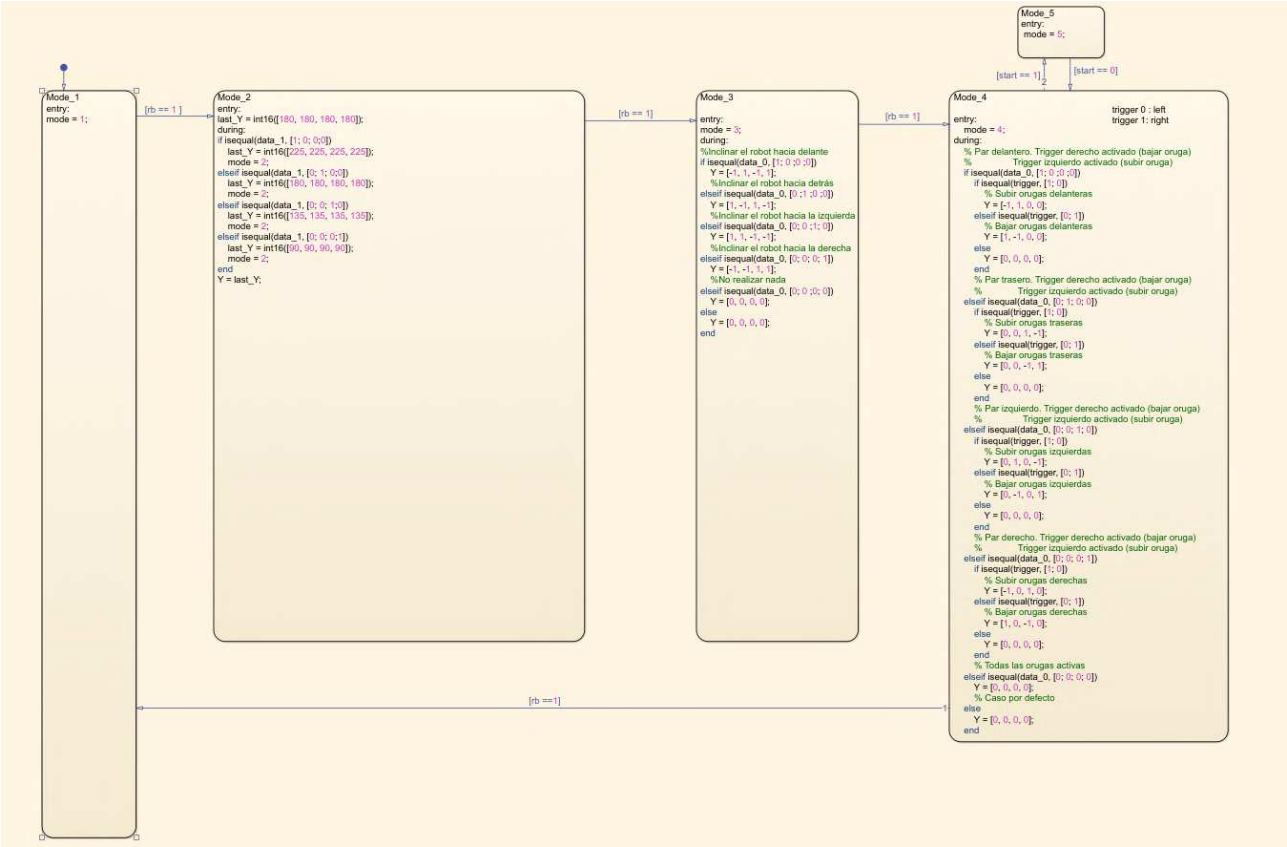


Figura 6.9: Chart principal del control PC

Estado 1:

Código 6.1: Estado 1: Modo de tracción

```
1
2 Mode_1
3
4 entry:
5     mode = 1;
```

El primer estado corresponde al *modo 1*, en el cual se activa únicamente el movimiento de los motores de tracción (ver código 6.1). La lógica implementada en este estado es sencilla: al entrar en dicho estado, se asigna el valor 1 a la variable de salida mode, lo que indica al sistema que debe operar exclusivamente en modo de tracción.

Estado 2:

Código 6.2: Estado 2: Modo de control de posición absoluta

```
1  Mode_2
2  entry:
3  last_Y = int16([180, 180, 180, 180]);
4
5  during:
6  if isequal(data_1, [1; 0; 0; 0])
7      last_Y = int16([225, 225, 225, 225]);
8      mode = 2;
9  elseif isequal(data_1, [0; 1; 0; 0])
10     last_Y = int16([180, 180, 180, 180]);
11     mode = 2;
12 elseif isequal(data_1, [0; 0; 1; 0])
13     last_Y = int16([135, 135, 135, 135]);
14     mode = 2;
15 elseif isequal(data_1, [0; 0; 0; 1])
16     last_Y = int16([90, 90, 90, 90]);
17     mode = 2;
18 end
19
20 Y = last_Y;
```

El segundo estado corresponde al *modo 2*, destinado al control de posición absoluta (ver código 6.2). En este estado, y en función del valor de la variable de entrada data_1 (anteriormente descrita, asociada a los botones A, B, X y Y), se genera un vector que contiene la posición angular objetivo correspondiente al botón activado.

Estado 3:

Código 6.3: Estado 3: Modo de control incremental de cuatro orugas

```
1 Mode_3
2
3 entry:
```

```
4  mode = 3;
5
6  during:
7  % Inclinar el robot hacia delante
8  if isequal(data_0, [1; 0 ;0 ;0])
9      Y = [-1, 1, -1, 1];
10 % Inclinar el robot hacia detrás
11 elseif isequal(data_0, [0 ;1 ;0 ;0])
12     Y = [1, -1, 1, -1];
13 % Inclinar el robot hacia la izquierda
14 elseif isequal(data_0, [0; 0 ;1; 0])
15     Y = [1, 1, -1, -1];
16 % Inclinar el robot hacia la derecha
17 elseif isequal(data_0, [0; 0; 0; 1])
18     Y = [-1, -1, 1, 1];
19 % No realizar nada
20 else
21     Y = [0, 0, 0, 0];
22 end
```

En este modo (ver código 6.3), el robot recibe como entrada el vector data_0, que contiene los comandos de control provenientes de la cruceta direccional. Según el patrón de pulsación detectado, se genera un vector de control Y que define los sentidos de giro de las orugas para lograr la inclinación deseada del robot: hacia adelante, hacia atrás, hacia la izquierda, hacia la derecha o ninguna acción. Cada componente del vector Y controla el sentido de giro de una oruga completa, procesamiento que se lleva a cabo tanto en el Arduino MKR WiFi 1010 como en el ESP32, como se detallará en los siguientes capítulos. Este estado permite un control directo sobre el balance del robot, que será útil por ejemplo al subir/bajar escaleras.

Estado 4:

Código 6.4: Estado 4: Modo de control incremental de par de orugas

```
1
2  Mode_4
3
4  entry:
5      mode = 4;
6  during:
7      % Par delantero. Trigger derecho activado (bajar oruga)
8      %                Trigger izquierdo activado (subir oruga)
9      if isequal(data_0, [1; 0 ;0 ;0])
10         if isequal(trigger, [1; 0])
11             % Subir orugas delanteras
12             Y = [-1, 1, 0, 0];
13         elseif isequal(trigger, [0; 1])
14             % Bajar orugas delanteras
15             Y = [1, -1, 0, 0];
16         else
17             Y = [0, 0, 0, 0];
18         end
19     % Par trasero. Trigger derecho activado (bajar oruga)
```

```
20      %               Trigger izquierdo activado (subir oruga)
21      elseif isequal(data_0, [0; 1; 0; 0])
22          if isequal(trigger, [1; 0])
23              % Subir orugas traseras
24              Y = [0, 0, 1, -1];
25          elseif isequal(trigger, [0; 1])
26              % Bajar orugas traseras
27              Y = [0, 0, -1, 1];
28          else
29              Y = [0, 0, 0, 0];
30          end
31      % Par izquierdo. Trigger derecho activado (bajar oruga)
32      %                 Trigger izquierdo activado (subir oruga)
33      elseif isequal(data_0, [0; 0; 1; 0])
34          if isequal(trigger, [1; 0])
35              % Subir orugas izquierdas
36              Y = [0, 1, 0, -1];
37          elseif isequal(trigger, [0; 1])
38              % Bajar orugas izquierdas
39              Y = [0, -1, 0, 1];
40          else
41              Y = [0, 0, 0, 0];
42          end
43      % Par derecho. Trigger derecho activado (bajar oruga)
44      %              Trigger izquierdo activado (subir oruga)
45      elseif isequal(data_0, [0; 0; 0; 1])
46          if isequal(trigger, [1; 0])
47              % Subir orugas derechas
48              Y = [-1, 0, 1, 0];
49          elseif isequal(trigger, [0; 1])
50              % Bajar orugas derechas
51              Y = [1, 0, -1, 0];
52          else
53              Y = [0, 0, 0, 0];
54          end
55      % Todas las orugas activas
56      elseif isequal(data_0, [0; 0; 0; 0])
57          Y = [0, 0, 0, 0];
58      % Caso por defecto
59      else
60          Y = [0, 0, 0, 0];
61      end
```

En el *modo 4* (ver código 6.4), las salidas son las mismas que en el estado anterior (*modo 3*), con la diferencia de que en este caso se actúa sobre pares de orugas, en lugar de sobre las cuatro orugas de manera conjunta. Esta modificación permite un ajuste más personalizado cuando el robot se encuentra en situaciones complejas, donde solo sea necesario accionar un par de orugas para realizar el equilibrado de forma manual.

El funcionamiento se realiza en conjunto con el valor de la cruceta seleccionado ($\mathrm{data\_0}$) y el $\mathrm{trigger}$ pulsado. El $\mathrm{trigger}$ servirá para determinar el sentido de giro del par de orugas correspondiente. Por ejemplo, si se desea elevar las orugas delanteras, se podrá lograr pulsando el $\mathrm{trigger}$ izquierdo (sentido antihorario) mientras se mantiene pulsada la cruceta hacia

adelante (UP). En cambio, para bajar las orugas, se pulsará el trigger derecho (DOWN). Si se dejan de pulsar los triggers, el robot se detendrá en la posición alcanzada.

Estado 5:

Código 6.5: Estado 5: Modo de nivelación horizontal automática

```
1
2 Mode_5
3
4 entry:
5     mode = 5;
```

Mientras el robot se encuentre en el *modo 4*, si se pulsa el botón Start, el sistema transitará al *modo 5* y empezará el nivelado horizontal automático (ver código 6.5), es decir, la salida mode se actualizará a 5. Al soltar el botón Start, el sistema regresará al *modo 4* y el nivelado se detendrá. En este estado, no es necesario realizar el control sobre la variable de salida Y, ya que dicha variable se determinará en el Arduino MKR WiFi 1010, según los valores proporcionados por la IMU y los cuatro encoders .

Finalmente, en el procesamiento de datos, los últimos 2 bytes se destinan a propagar el estado de conexión del mando a los microcontroladores (ver figura 6.10). Estos bytes permiten indicar si el mando está correctamente conectado y operativo.

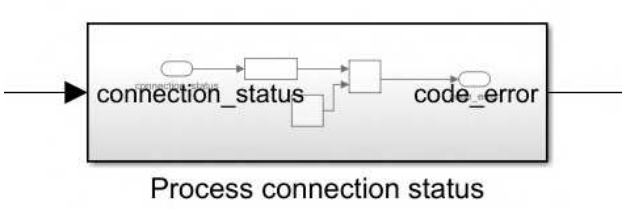


Figura 6.10: Subsistema del estado de conexión

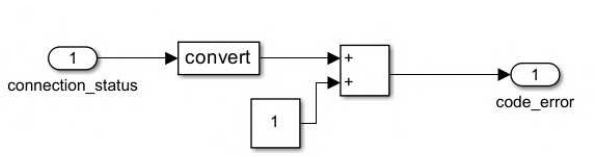


Figura 6.11: Interior del subsistema de estado de conexión

La entrada corresponde al estado de conexión del mando, siendo ’1’ cuando está conectado y ’0’ cuando no lo está. Para evitar posibles problemas derivados de recibir un ’0’ en el mensaje UDP —lo cual podría interpretarse erróneamente como una pérdida de conexión o un error en la transmisión—, se ha decidido sumar 1 al estado original (ver figura 6.11). De este modo, un valor de ’2’ indica que el mando está conectado, mientras que un valor de ’1’ indica que está desconectado.

6.5 Transmisión de datos

Lo último que resta es el envío de datos hacia el Arduino MKR WiFi 1010. Este envío se realizará únicamente cuando se detecte un cambio en el vector compuesto por 9 datos de tipo int16, utilizando para ello el bloque *Detect Change* (ver figura 6.13). El envío de los datos se efectuará, como se mencionó previamente, mediante WiFi y utilizando el protocolo UDP. Para ello, se ha empleado el bloque *UDP SEND*, configurado de la siguiente manera (ver figura 6.12):

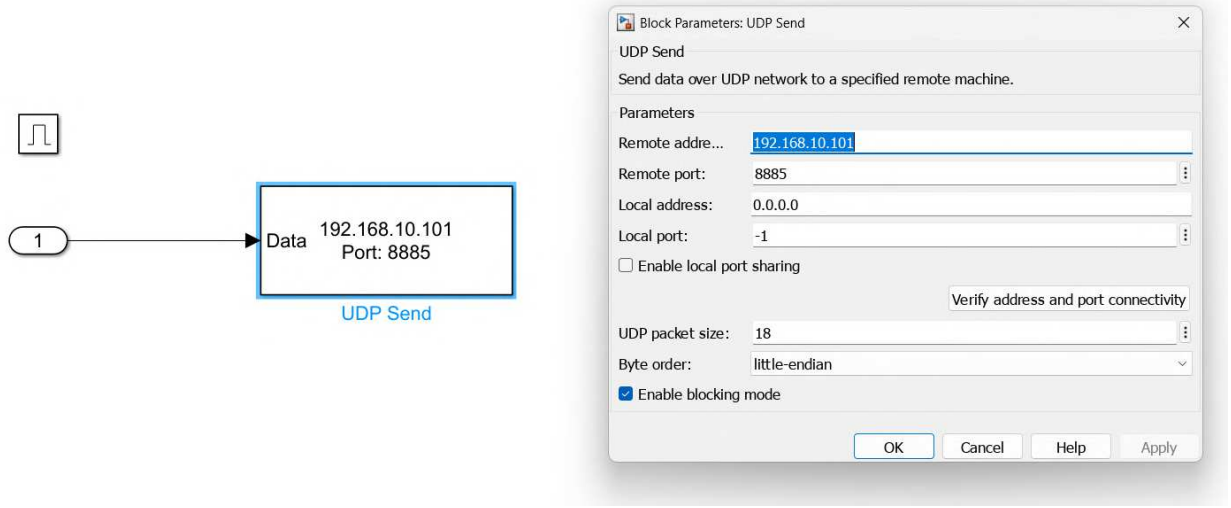


Figura 6.12: Interior del subsistema *UDP Send*

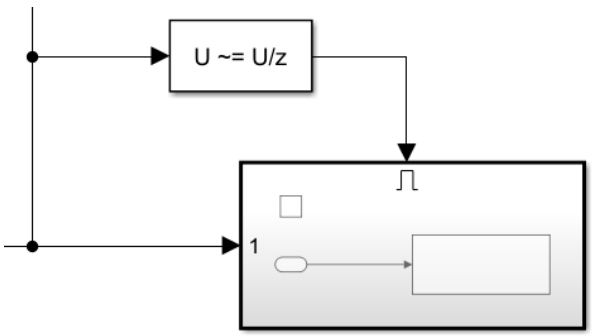


Figura 6.13: Subsistema activo con enable mediante un detector de cambio

Se configura el puerto de envío del Arduino MKR WiFi 1010 (8885) y la IP de este (192.168.10.101), así como el tamaño del paquete, que es de 2 bytes por cada dato y 9 datos en total, lo que da un tamaño de 18 bytes.

En el anexo al final de la memoria se presenta un modelo del control PC, desarrollado con solo tres estados. En dicho anexo se explican las ventajas de este modelo en comparación con el de programación descrito en este capítulo.

Capítulo 7

Programación del control Arduino MKR WiFi 1010

7.1 Introducción

En el presente capítulo se describe el funcionamiento y las responsabilidades del Arduino MKR WiFi 1010 dentro del sistema robótico. Este microcontrolador actúa como unidad principal de control, siendo el encargado de coordinar las acciones generales del vehículo en cuanto a los motores de tracción, así como gestionar la comunicación con el sistema central (PC) y propagar las órdenes necesarias hacia el microcontrolador ESP32 para la ejecución de movimientos específicos en los motores de elevación.

7.2 Vista general del modelo

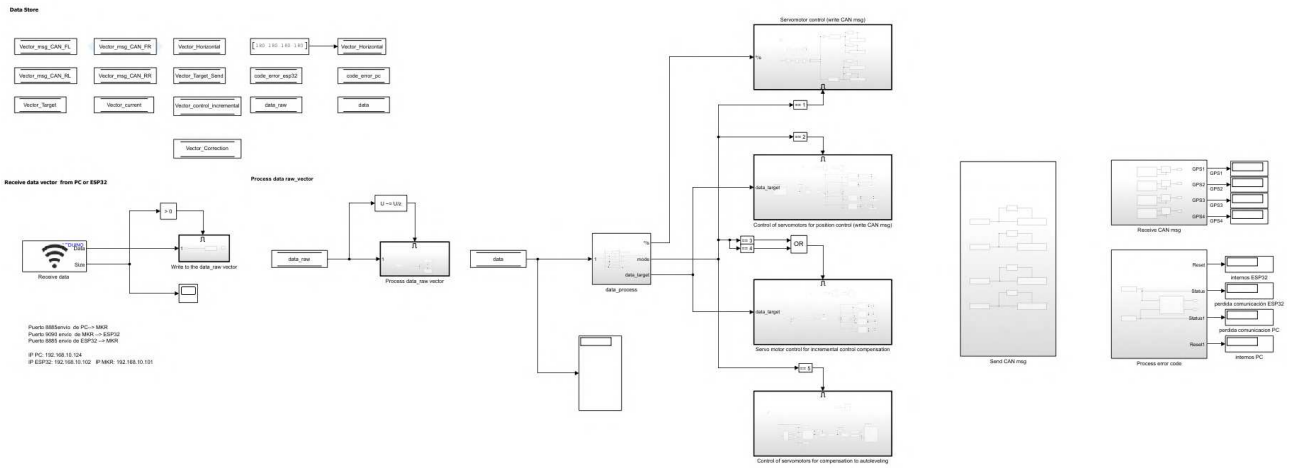


Figura 7.1: Modelo del control Arduino MKR WiFi 1010

En el modelo de programación mostrado en la figura anterior 7.1 (ver figura 7.1), el cual se ejecuta en el Arduino MKR WiFi 1010, se lleva a cabo una serie de procesos. En primer lugar, se adquieren y procesan los datos enviados por el PC. Posteriormente, se realiza la conversión de estos datos y el procesamiento de los diferentes modos de funcionamiento del robot. Finalmente, se envían los mensajes a los motores de tracción mediante el protocolo CAN.

Adicionalmente, se ha implementado un subsistema para la recepción de los datos provenientes de los motores de tracción. Estos datos son procesados para interpretar la información que el motor está procesando, como la temperatura actual, la corriente que están consumiendo o la velocidad en grados por segundo. Finalmente, se incluye un subsistema encargado de la visualización de errores, lo cual permite al usuario identificar posibles fallos en el sistema.

La parte superior izquierda del modelo corresponde a los *data_store* definidos para crear vectores de datos locales en el modelo. Estos *data_store* permiten almacenar y gestionar los datos dentro del sistema, facilitando la programación y la organización del flujo de información en el modelo.

7.3 Recepción de datos por WiFi

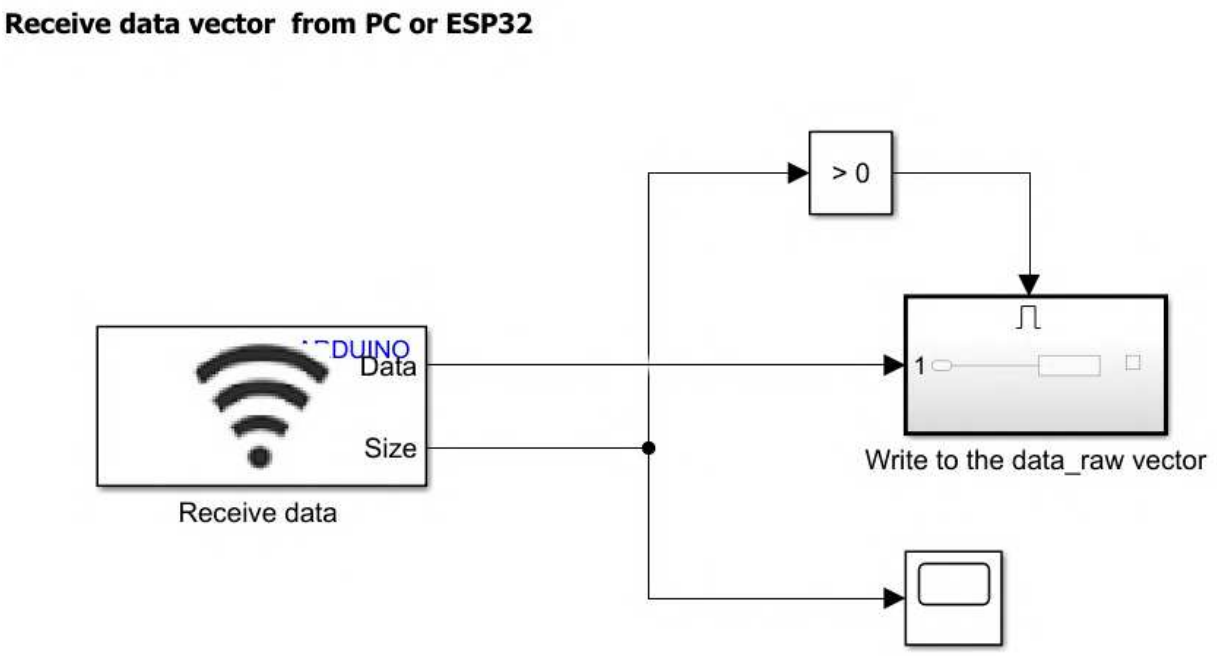


Figura 7.2: Recepción de datos WiFi

El primer bloque y uno de los más relevantes en el modelo implementado para el Arduino MKR WiFi 1010 es el bloque Arduino WiFi UDP Receive (ver figura 7.2). Este compo-

nente permite recibir datos a través del protocolo UDP y se configura especificando el puerto de escucha, el tipo de dato esperado, así como el tamaño del vector de datos que se desea recibir. Adicionalmente, se define el tiempo de muestreo (*Sample Time*), que establece la frecuencia con la que el microcontrolador ejecutará la acción de lectura de datos entrantes, garantizando así una comunicación periódica y sincronizada con el sistema emisor. Se ha configurado a unos 0.01 segundos. El sistema emisor puede ser tanto el PC como el ESP32, por lo que es necesario ver de quién es el dato para poder entenderlo correctamente.

Cada vez que se recibe un paquete de datos a través del bloque *Arduino WiFi UDP Receive*, se activa el subsistema con habilitación (*Enable*) denominado *Write to the data_raw vector*. Este subsistema se encarga de almacenar temporalmente los datos recibidos en un vector interno llamado *data_raw*, el cual actúa como contenedor principal de la información entrante. Este enfoque se ha usado en muchos de los bloques programados, permite desacoplar la adquisición de datos del procesamiento posterior, facilitando la organización del modelo y mejorando la trazabilidad de las señales a lo largo del sistema.

7.4 Procesamiento del vector de datos

Del procesamiento de los datos (*data_raw*) se encarga el siguiente grupo de bloques (ver figura 7.3):

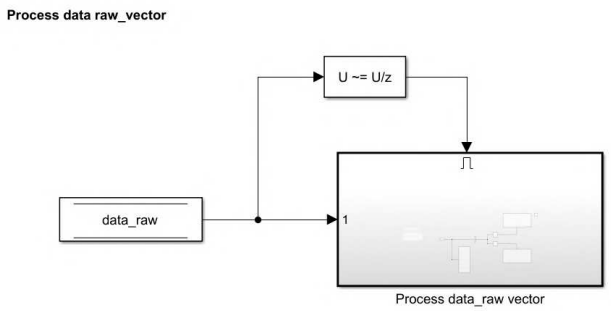


Figura 7.3: Subsistema activo por cambio de entrada de datos recibidos

El bloque encargado de habilitar el subsistema Process data_raw vector es un detector de cambio, denominado Detect Change, el cual emite una señal lógica alta cada vez que detecta una variación en su entrada. Por tanto, para evitar la saturación del sistema, resulta imprescindible procesar únicamente aquellos datos que difieran de los valores previamente almacenados.

Dentro del subsistema *Process data_raw vector* se encuentran los siguientes subsistemas (ver figura 7.4):

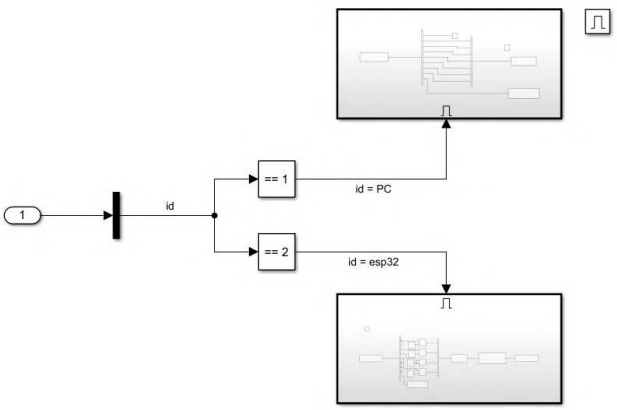


Figura 7.4: Interior del subsistema de procesamiento de data_raw vector

Se extrae el primer dato del *data_raw* con un bloque *Demux* (demultiplexor) de una salida. Este primer elemento del datagrama recibido corresponde al identificador id, tal como se muestra en la tabla 7.1. Este valor permite determinar el origen del mensaje, lo cual resulta esencial para diferenciar entre los dos emisores posibles.

En el puerto del Arduino MKR WiFi 1010, como se describió en capítulos anteriores, pueden recibirse tanto las instrucciones procedentes del PC, que contienen órdenes de movimiento, como los mensajes enviados por el ESP32, que incluyen los ángulos actuales de las orugas.

7.4.1 Datos recibidos del PC

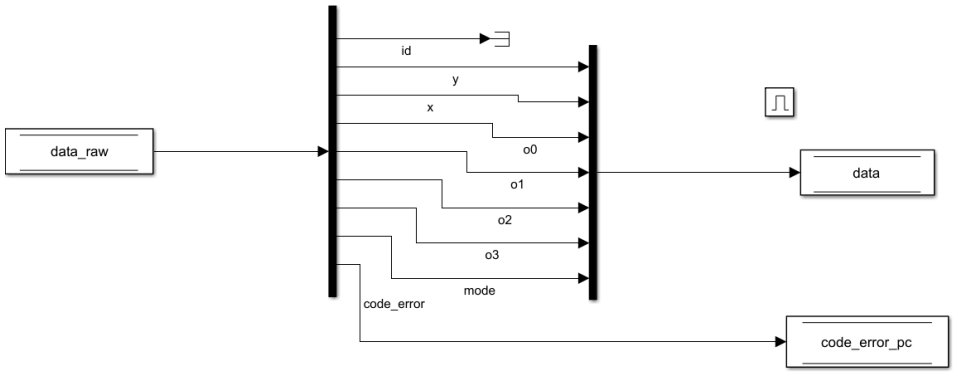


Figura 7.5: Datos recibidos desde el PC

El datagrama del PC tiene el siguiente formato:

DATAGRAMA	DATO 1	DATO 2	DATO 3	DATO 4	DATO 5
Variable	id	action_left_train	action_right_train	o0	o1

DATAGRAMA	DATO 6	DATO 7	DATO 8	DATO 9	-
Variable	o2	o3	mode	code_error	-

Tabla 7.1: Datagrama de datos recibidos del PC por Arduino MKR WiFi 1010

Si el *id* extraído es el 1, se procesa entonces el subsistema de la figura 7.5. Dentro de este subsistema se extraen los datos del vector sin procesar (*data_raw*) mediante un bloque *Demux*, que permite separar cada elemento individual. Posteriormente, estos datos se reorganizan y se agrupan nuevamente mediante un bloque *Mux* (multiplexor) para formar el vector interno *data,* que contiene la información estructurada para el control del sistema. La información que contiene *data* es la siguiente:

Las variables $\mathrm{action\_left\_train}$ y $\mathrm{action\_right\_train}$ (dato 2-3) corresponden a las velocidades de ambos trenes del robot, el lado izquierdo y el derecho, respectivamente. Estas son calculadas a partir de los valores de los $\mathrm{joysticks}$ del mando conectado al PC. A continuación, se encuentran cuatro datos (dato 4-7) identificados como $\mathrm{o0}$, $\mathrm{o1}$, $\mathrm{o2}$ y $\mathrm{o3}$ los cuales pueden representar, dependiendo del modo de funcionamiento, bien posiciones angulares concretas que se enviarán al ESP32 para que este controle los motores paso a paso hasta alcanzar dichos ángulos, o bien el sentido de giro ($\mathrm{-1}$, $\mathrm{0}$, $\mathrm{1}$) que deben adoptar estos motores en un control incremental manual.

El dato 8, situado en la penúltima posición del datagrama recibido, especifica el modo de operación del robot. Por último, el dato 9 se reserva para la codificación de errores: su contenido permite identificar las posibles anomalías detectadas por cualquiera de los nodos del sistema, en este caso, las derivadas del PC.

7.4.2 Datos recibidos del ESP32

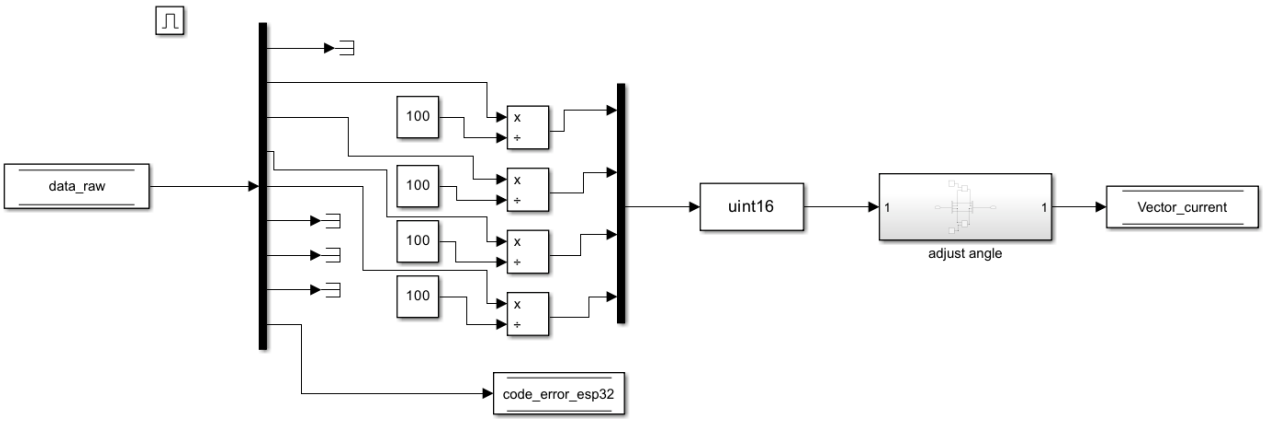


Figura 7.6: Datos recibidos del ESP32 por Arduino MKR WiFi 1010

Por otro lado, si el *id* recibido es el 2, se activa el subsistema de la figura 7.6.

Los datos que el ESP32 envía al Arduino MKR WiFi 1010 corresponden a los ángulos actuales ($\mathrm{angle\_current}$) de las orugas. Esto se puede ver en la tabla inferior (ver tabla 7.2) . Los valores son transmitidos desde el ESP32 en formato $\mathrm{int32}$, por lo que se requiere una conversión previa: en primer lugar, se dividen entre 100 para reducir su escala y adaptarlos a un rango manejable, convirtiéndolos a tipo $\mathrm{int16}$. Posteriormente, se realiza una conversión a tipo $\mathrm{uint16}$, dado que, debido al sistema de referencia adoptado y a la disposición de los imanes en los codificadores magnéticos, no es necesario representar ángulos negativos.

DATAGRAMA	**DATO 1**	**DATO 2**	**DATO 3**	**DATO 4**	**DATO 5**
Variable	id	angle_current_FR	angle_current_FL	angle_current_RR	angle_current_RL

DATAGRAMA	**DATO 6**	**DATO 7**	**DATO 8**	**DATO 9**	**-**
Variable	null	null	null	code_error	-

Tabla 7.2: Datagrama de datos recibidos del ESP32 por Arduino MKR WiFi 1010

Para garantizar una programación eficiente y sencilla en la gestión de las orugas del robot, es necesario realizar un tratamiento adecuado de los datos recibidos. Dado que los cuatro motores de elevación no se encuentran en la misma orientación respecto al chasis del robot, se ha realizado una transformación en el sistema de referencia de dos de las cuatro orugas (F.L. y R.R.), tomando como base la oruga frontal derecha (F.R.).

En este sentido, se ha diseñado una estrategia que utiliza los ángulos medidos en la oruga F.R. como referencia, aplicando posteriormente un ajuste a las demás orugas en función de cómo se encuentra su sistema de referencia respecto al de la F.R. El sistema de referencia establecido para esta oruga, que sirve de modelo para las demás, se muestra en la figura inferior (ver figura 7.7).

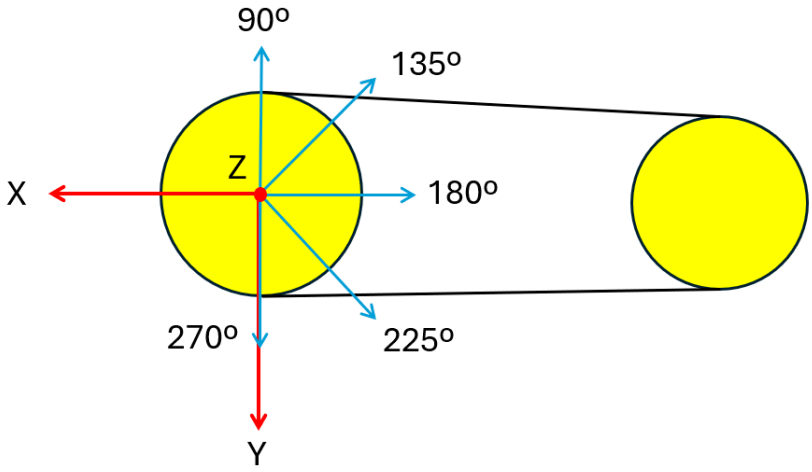


Figura 7.7: Sistema de referencia de la oruga frontal derecha

Este enfoque simplifica la implementación de los algoritmos de movimiento, al permitir el tratamiento homogéneo de todos los ángulos, lo cual facilita el control de las orugas y evita las complicaciones derivadas de las discrepancias en las mediciones angulares entre ellas.

Para realizar esta transformación, se encuentra el subsistema *adjust_angle* (ver figura 7.8), que de entrada tiene un vector de cuatro posiciones tipo $\mathrm{uint16}$ (entero sin signo de 16 bits), correspondientes a los ángulos actuales de las orugas y a la salida devuelve el vector interno nombrado como $\mathrm{Vector\_current}$ que tendrá los ángulos correctamente tratados, para su uso en siguientes subsistemas.

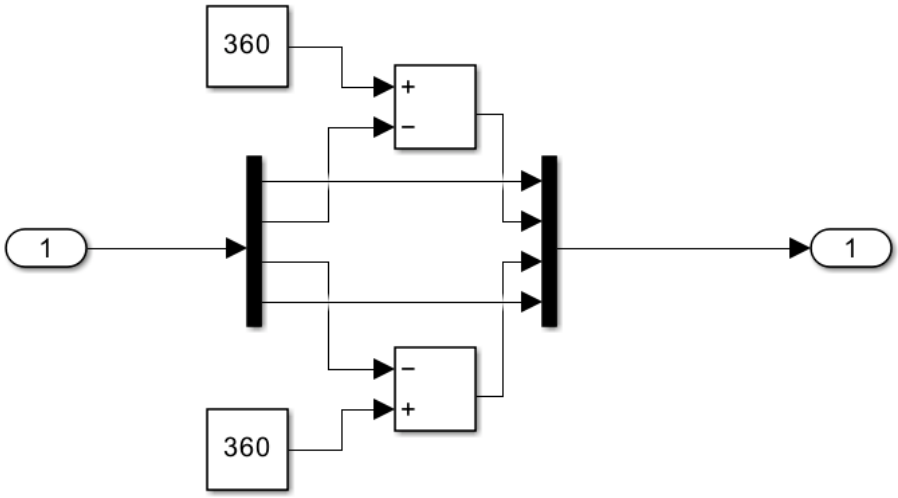


Figura 7.8: Subsistema de ajuste de ángulos

Para simplificar la programación y asegurar una coherencia en las posiciones angulares de las orugas, se ha establecido que las orugas frontal derecha (F.R.) y trasera izquierda (R.L.) comparten el mismo sistema de referencia, ya que los motores están dispuestos en el chasis del robot de manera idéntica en ambas orugas.

Por otro lado, las orugas de la diagonal opuesta, es decir, la frontal izquierda (F.L.) y la trasera derecha (R.R.), requieren un ajuste adicional en sus ángulos medidos. Este ajuste consiste en sustraer el ángulo medido de 360 grados, lo que permite obtener las mismas posiciones angulares para las cuatro orugas de manera coherente y sencilla. De esta forma, se asegura que todas las orugas se comporten de manera uniforme, a pesar de las diferencias en la disposición de los motores en las distintas diagonales.

7.5 Control de motores

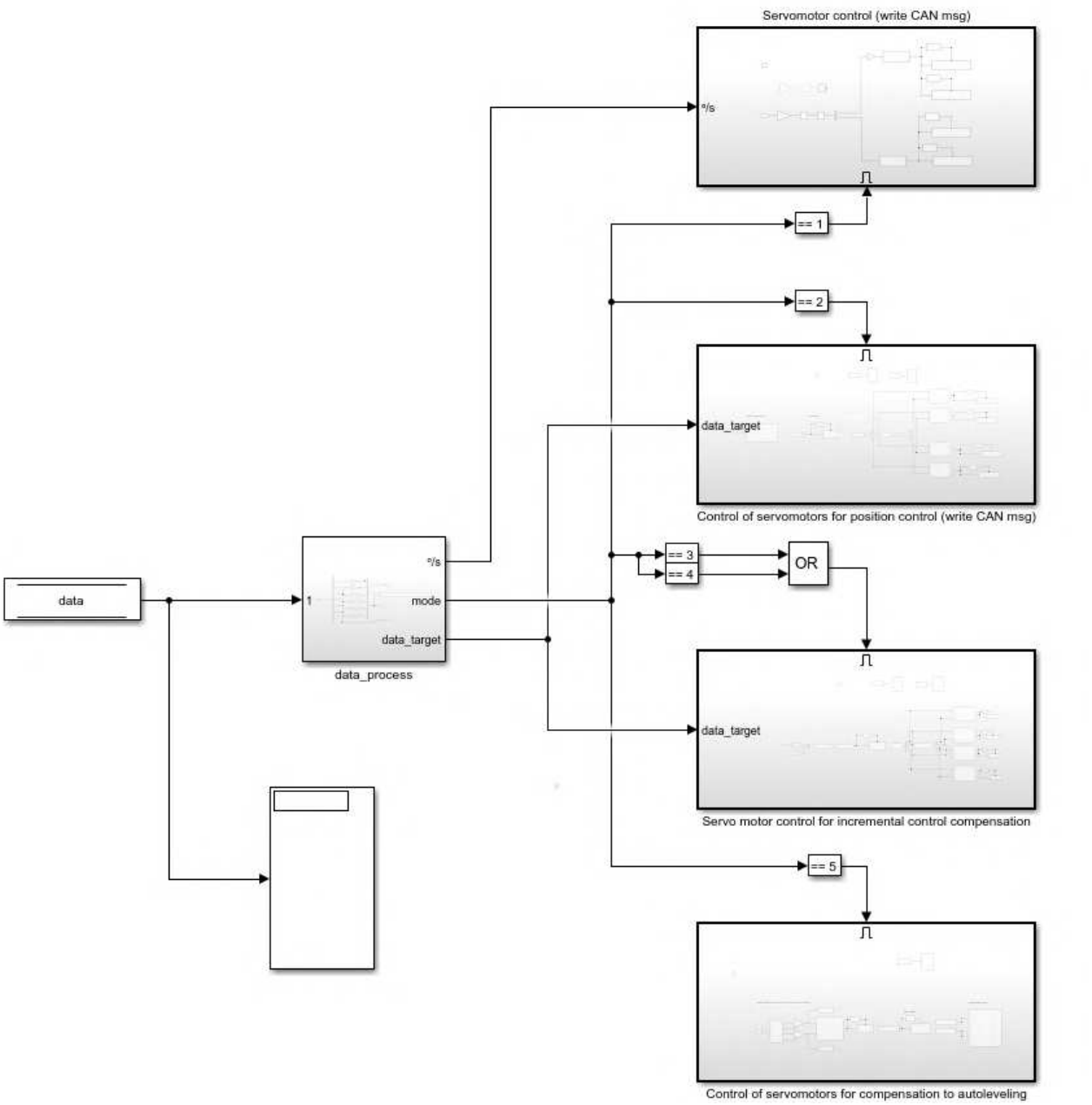


Figura 7.9: Control de modos de movimiento en Arduino MKR WiFi 1010

El siguiente paso en el control del robot, en lo que respecta al microcontrolador Arduino MKR WiFi 1010, es el procesamiento del modo de funcionamiento y, por consiguiente, la activación del subsistema asociado a ese modo de funcionamiento (ver figura 7.9). La idea es que, dependiendo del modo de funcionamiento, se ejecute un subsistema u otro. La entrada a este control es el vector interno denominado data, proveniente del subsistema

anterior $\mathrm{data\_raw\ vector}$. Este vector data contiene, según el modo de funcionamiento, unas consignas u otras (como se ha explicado anteriormente); pueden ser ángulos objetivo o sentidos de giro si estamos en el control incremental para el nivelado manual.

7.5.1 Procesamiento del vector data

El objetivo del subsistema *data_process* (ver figura 7.10) es simplificar el funcionamiento del control, extrayendo para cada modo de funcionamiento los datos necesarios.

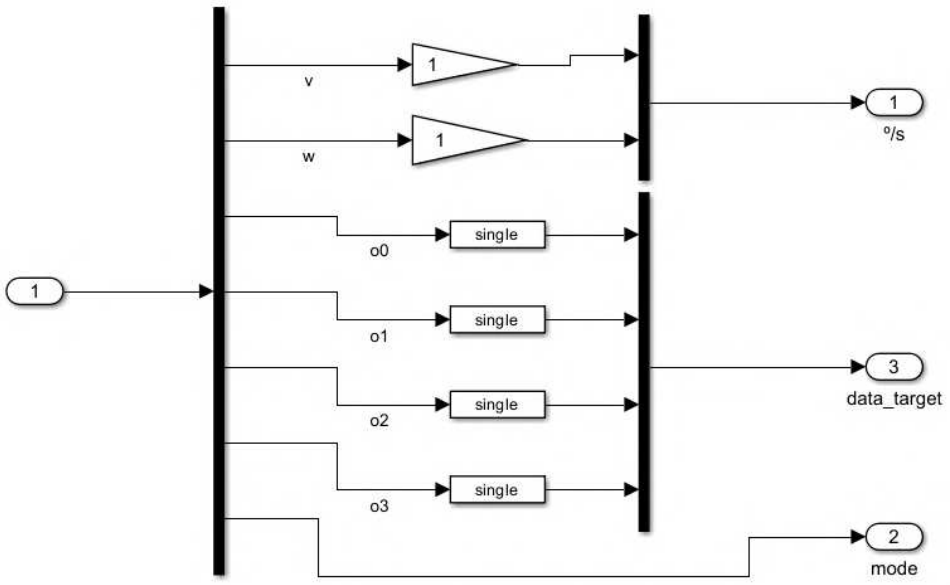


Figura 7.10: Subsistema data_process

Dentro del subsistema *data_process*, se encuentra un *Demux* 1-7 y dos *Mux*: uno de 2-1 y otro de 4-1. Con el primer multiplexor se obtienen las consignas de velocidad lineal deseada de manera diferencial para cada una de las orugas. Estas consignas son convertidas de $\mathrm{int16}$ a single con una ganancia de 1 (bloque *Gain)* que se encuentra en ambas entradas del multiplexor.

Por otro lado, también se obtienen los datos descritos anteriormente ($\mathrm{o0}$, $\mathrm{o1}$, $\mathrm{o2}$, $\mathrm{o3}$) en formato single, que serán la entrada del segundo multiplexor. Con estos datos se gestionarán los modos de control de posición e incremental descritos previamente.

Finalmente, el último dato del vector data es el modo de funcionamiento, que servirá como variable para seleccionar el subsistema asociado. Dependiendo del modo, los motores de tracción se comportarán de forma diferente, como se verá a continuación.

7.5.2 Subsistema del modo de funcionamiento 1

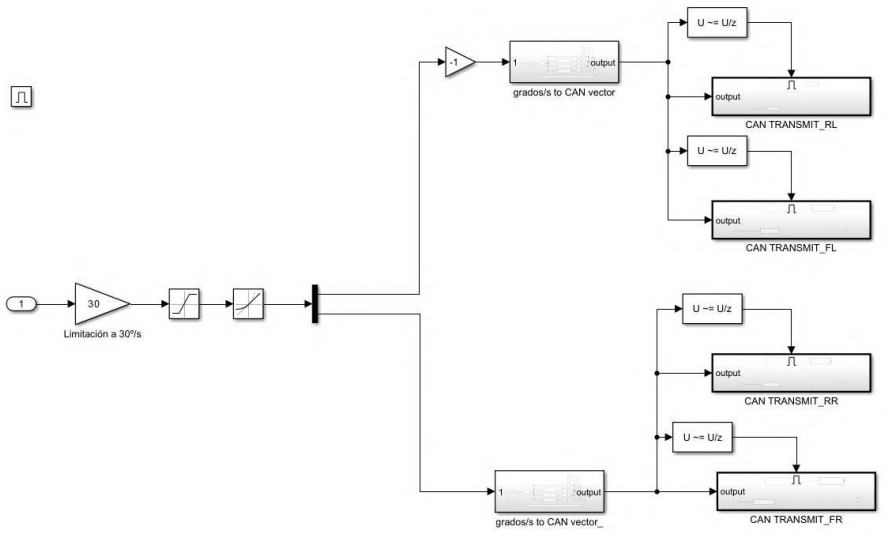


Figura 7.11: Subsistema de control de tracción diferencial

El primer modo de funcionamiento (modo de tracción) habilita la actuación sobre los motores de tracción. Estos se controlan mediante consignas de velocidad lineal de manera diferencial para cada uno de los trenes de orugas (tren izquierdo y tren derecho). Mediante un control diferencial de estas velocidades, los motores de tracción propulsan el robot para que se desplace tanto en traslación lineal como en rotación, posibilitando así el giro controlado del vehículo.

Desde el punto de vista mecánico, se requiere que la posición angular de las orugas se sitúe aproximadamente en $135°$ (según el sistema de referencia de la oruga F.R.), con el fin de evitar esfuerzos innecesarios sobre la estructura, si lo que se desea es girar. Si la oruga se encontrase en una posición de $180°$, el área de contacto con el suelo sería máxima. Esto dificultaría el giro sobre sí misma y podría provocar deformaciones o daños estructurales en la oruga debido a la resistencia mecánica generada por la goma y el peso del robot.

El subsistema encargado de ello se puede ver en la figura anterior (ver figura 7.11). Al inicio, contiene un bloque *Gain* (ganancia) con un valor de 30. Esta ganancia se aplica a la entrada proveniente de los $\mathrm{joysticks}$ del mando de Xbox, cuyas señales se encuentran en el rango de -1 a 1, con el objetivo de escalar dichas señales a valores útiles para el control en grados por segundo.

A continuación, se incorpora un bloque *Saturation* de la librería de Simulink, encargado de limitar las velocidades máximas y mínimas a un rango de $\pm 40°/\mathrm{s}$, estableciendo así un margen seguro de operación para las orugas.

Seguidamente, se emplea un bloque *Rate Limiter*, también de Simulink, cuya función es asegurar una transición progresiva tanto en la aceleración como en la desaceleración,

evitando así variaciones bruscas que podrían comprometer la estabilidad del sistema o provocar un comportamiento indeseado.

Tras el control de la señal de velocidad, se dispone un bloque *Demux* que separa la señal para el control independiente de los lados izquierdo y derecho del robot. La primera salida del demultiplexor corresponde al lado izquierdo.

En este lado se incorpora una ganancia de -1, necesaria para invertir el sentido de giro. Esta inversión es debida a la disposición mecánica de los motores, ya que un valor positivo de entrada (por ejemplo, $5°/s$) produciría un giro antihorario, generando un desplazamiento hacia atrás. Por lo tanto, se invierte la señal para garantizar un comportamiento coherente con el desplazamiento deseado.

Por el contrario, en el lado derecho no es necesaria ninguna corrección, ya que los motores están dispuestos de forma que una entrada positiva genera un giro horario, impulsando al robot hacia adelante de forma directa.

Una vez obtenidas las velocidades angulares deseadas para cada motor de tracción (expresadas en grados por segundo), es necesario traducir dichos valores a un formato comprensible por los controladores de los motores RMD, que operan mediante el protocolo de comunicación CAN.

Para ello, se realiza una conversión de las velocidades angulares en los datos que conforman el mensaje CAN. En particular, los motores solo requieren que se modifiquen los últimos 4 bytes del mensaje, los cuales controlan la velocidad. Dichos bytes corresponden a los bytes 5 a 8 del mensaje CAN estándar de 8 bytes, donde el byte 5 representa el byte menos significativo (LSB) y el byte 8 el más significativo (MSB), siguiendo un orden inverso (*little-endian*).

Antes de dividir el valor de velocidad angular deseada en bytes, este se multiplica por un factor de 100, conforme a las especificaciones del fabricante detalladas en el manual de usuario de los motores RMD. Esta multiplicación permite que el valor esté en las unidades requeridas por el protocolo de control del motor.

Una vez convertido a un número entero de 32 bits, se realiza la extracción de los 4 bytes menos significativos para componer el mensaje CAN a enviar, respetando el orden especificado.

El subsistema encargado de la conversión de las velocidades angulares a mensajes CAN se muestra en la siguiente figura (ver figura 7.12) :

A modo de ejemplo:

- Para una velocidad de 200 grados/segundo en sentido horario:

$$200 \times 100 = 20000 = 0x00004E20$$

El mensaje CAN completo sería:

$$\text{0xA2 0x00 0x00 0x00 0x20 0x4E 0x00 0x00}$$

- Para una velocidad de -200 grados/segundo (sentido antihorario):

$$-200 \times 100 = -20000 \Rightarrow 0xFFFFB1E0 \text{ (en complemento a dos)}$$

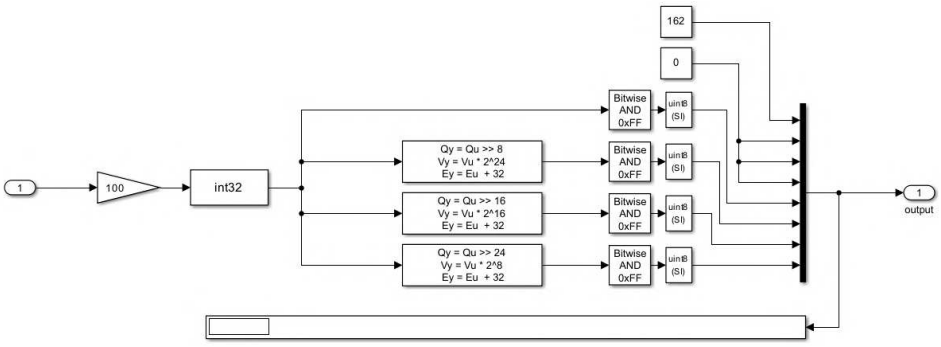


Figura 7.12: Subsistema de conversión de grados/segundo a mensaje CAN

El mensaje CAN resultante será:

$$0\text{xA2}\ 0\text{x00}\ 0\text{x00}\ 0\text{x00}\ 0\text{xE0}\ 0\text{xB1}\ 0\text{xFF}\ 0\text{xFF}$$

Cada vez que se envía un mensaje con instrucción 0xA2, el motor responde automáticamente con un paquete de datos que incluye información de estado: temperatura interna, corriente instantánea, velocidad angular y posición. Estas respuestas se reciben mediante cuatro bloques *CAN Receive,* uno por cada motor, los cuales están configurados para extraer y gestionar los valores de velocidad angular. Estos datos pueden utilizarse posteriormente para visualización, registro o control en lazo cerrado.

Tras la generación del mensaje CAN en el subsistema de conversión, se ha implementado un subsistema específico para cada oruga del robot. Este incluye un bloque *Detect Change* que actúa como señal de habilitación (*enable*), permitiendo que la transmisión del mensaje solo se realice cuando se detecta un cambio en el valor de velocidad. Esta estrategia reduce el tráfico en el bus CAN, ya que evita el envío de mensajes redundantes.

En el subsistema correspondiente a la oruga trasera izquierda (*CAN_TRANSMIT_R.L*), se utiliza un bloque *Data Write Store* como el que se muestra en la figura 7.13.

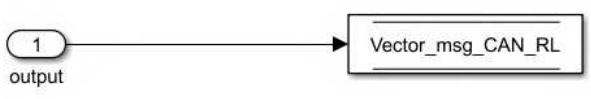


Figura 7.13: Bloque de escritura del mensaje *CAN* para la oruga trasera izquierda

En total, se han definido cuatro bloques Data Write Store, uno por cada oruga:

- Vector_msg_CAN_FR – Oruga delantera derecha (Front Right)
- Vector_msg_CAN_FL – Oruga delantera izquierda (Front Left)
- Vector_msg_CAN_RR – Oruga trasera derecha (Rear Right)
- Vector_msg_CAN_RL – Oruga trasera izquierda (Rear Left)

Estos bloques almacenan temporalmente los mensajes *CAN* ya formateados, listos para su transmisión. Aunque sería posible realizar la transmisión directamente desde estos subsistemas mediante el bloque *CAN Transmit,* se ha optado por esta estructura modular para centralizar el envío de mensajes en un único punto del modelo. Esta decisión mejora la claridad, facilita la depuración y permite una mayor flexibilidad en futuras ampliaciones del sistema.

7.5.3 Subsistema del modo de funcionamiento 2. Control de posición absoluta

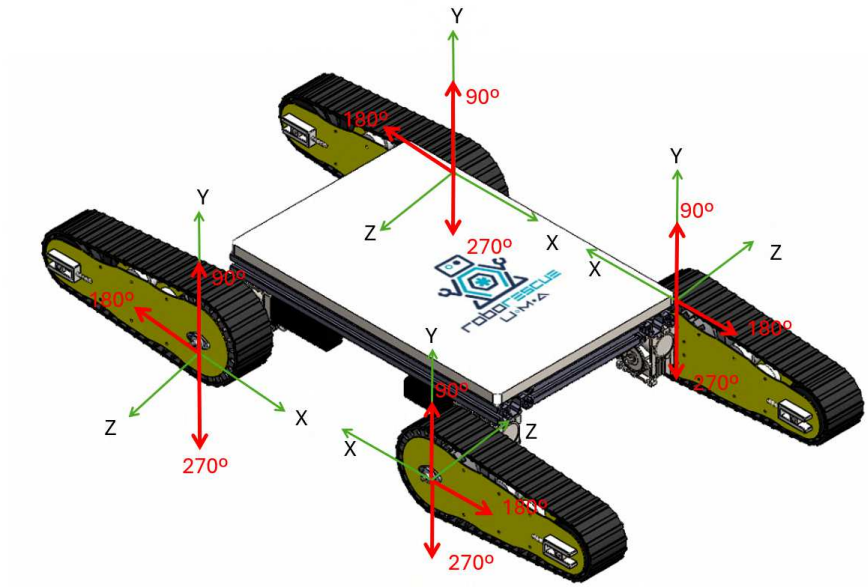


Figura 7.14: Sistema de referencia del robot

En el segundo modo de funcionamiento, el sistema utiliza los ocho motores disponibles, a diferencia del modo anterior, en el que solo se activaban los cuatro motores de tracción. Este modo incorpora un control de posición para las orugas, que permite elevar el robot a posiciones concretas (ver ejemplos en la figura 7.32 y 7.33). El sistema de referencia del robot se muestra en la figura 7.14.

El subsistema encargado de este control se denomina *Control_of_servomotors_for_compensation*. Este subsistema sigue la misma estrategia que el correspondiente al modo anterior y se activa mediante una señal de habilitación (*enable*). Por tanto, solo se ejecuta cuando el modo de funcionamiento seleccionado desde el PC es el modo 2, tal y como se muestra en la figura 7.9. Para ello, se utiliza un bloque *Compare to constant* configurado con el valor ”2”, que habilita el subsistema únicamente cuando la variable mode es igual a ”2”.

La entrada principal del subsistema es un vector de ángulos objetivo ($\mathrm{data\_target}$), que indica la posición angular deseada para cada una de las orugas de forma independiente. Como salida, el subsistema genera:

- Los mensajes CAN necesarios para controlar la velocidad de los motores de tracción.
- El mensaje UDP destinado al ESP32, que contiene las consignas de posición para los motores paso a paso.

Ahora se explicará en detalle los bloques que componen el subsistema del control de posición absoluta (ver figura 7.15):

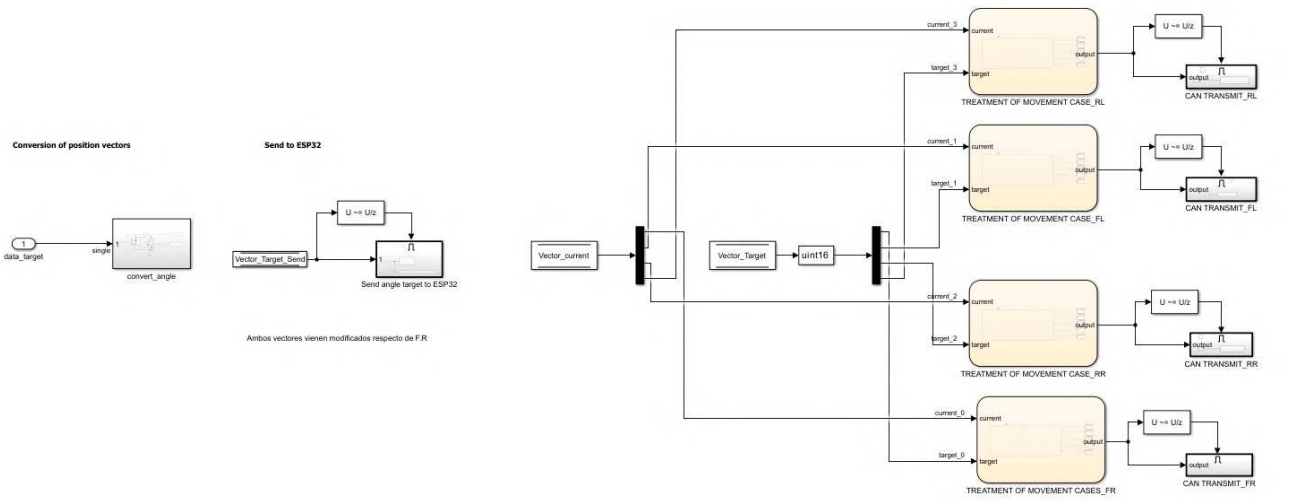


Figura 7.15: Subsistema del modo de funcionamiento 2

El primer subsistema empezando por la izquierda se puede ver en la siguiente figura (ver figura 7.16):

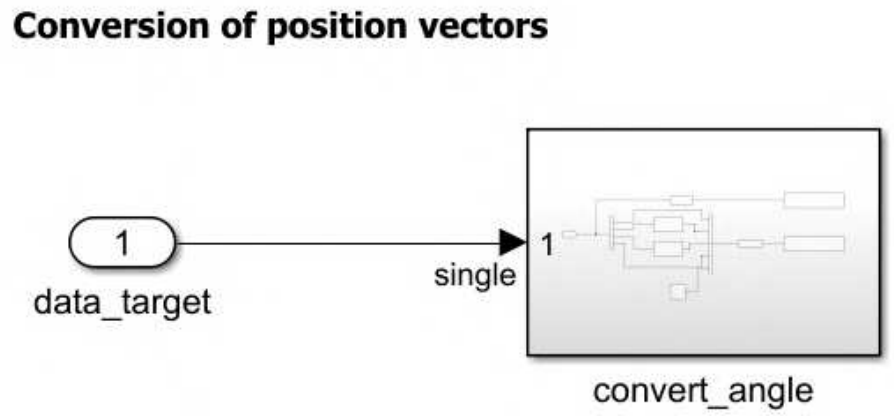


Figura 7.16: Subsistema de conversión de vectores de posición

Este subsistema, tomando como entrada el vector $\mathrm{data\_target}$ mencionado previamente, el cual en el caso del modo 2, contiene los ángulos de posición absoluta de las orugas. Tiene como función principal la generación de datos que se enviarán hacia el microcontrolador ESP32, encargado de controlar los motores paso a paso y la generación de un vector auxiliar interno con el que se realizará el control.

No obstante, estos datos no pueden transmitirse directamente en su forma original. Para simplificar la programación y el diseño del modelo, se ha estandarizado el tratamiento de los datos con referencia a la oruga delantera derecha ($\mathrm{F.R.}$) (ver figura 7.14). Esto implica que las orugas situadas en la diagonal opuesta (delantera izquierda F.L. y trasera derecha R.R.) requieren una transformación previa para adecuar su referencia de orientación.

La conversión necesaria consiste en restarle a 360º los ángulos objetivo de dichas orugas (ver figura 7.17). Esto se debe a que, por la disposición simétrica del sistema de orugas, las orugas F.L. y R.R. en la posición vertical donde F.R. marca 90º ellas marcan 270º.

$$\theta_{ajustada} = \begin{cases} 360 - \theta_{target}, & \text{si la oruga pertenece a la diagonal opuesta a F.R.} \\ \theta_{target}, & \text{en caso contrario} \end{cases} \tag{7.1}$$

De este modo, se garantiza una interpretación coherente de los ángulos por parte del ESP32. Ya que este trabaja con los ángulos reales, sin transformación.

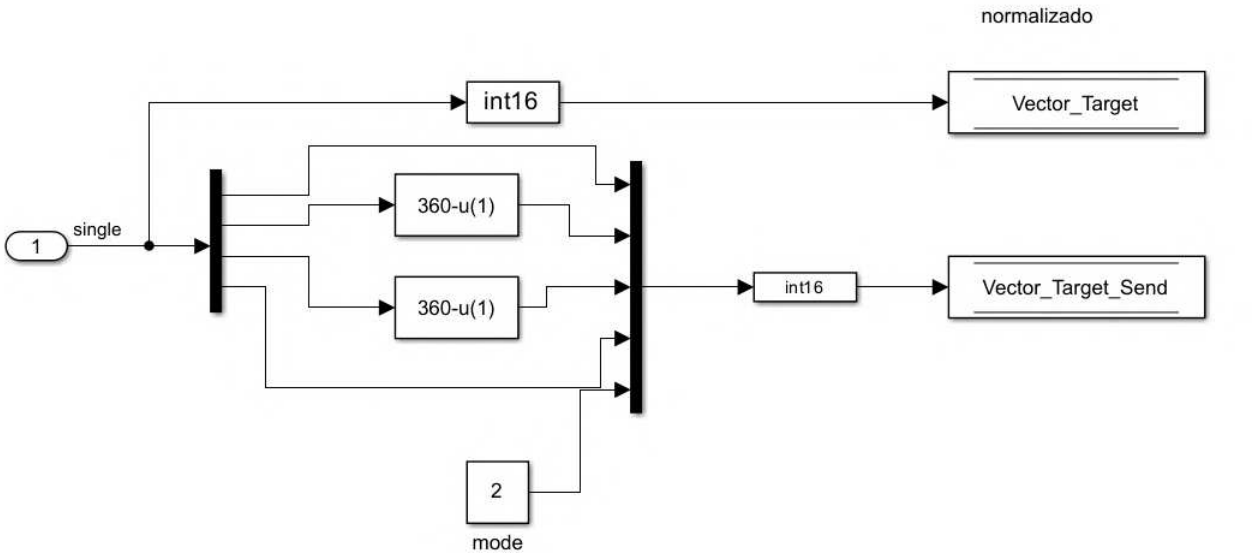


Figura 7.17: Interior del subsistema de conversión de vectores de posición

Nótese que se define un vector objetivo $\mathrm{Vector\_Target}$, sin modificar, el cual se emplea internamente para la compensación; y un vector objetivo $\mathrm{Vector\_Target}$ *_Send*, modificado, que se envía al ESP32. Asimismo, este mensaje incorpora también el modo de funcionamiento.

El siguiente subsistema (ver figura 7.18) es el encargado del envío de datos (Vector_Target_Send) al ESP32, para que puedan actuar los motores de elevación conjuntamente con los de tracción.

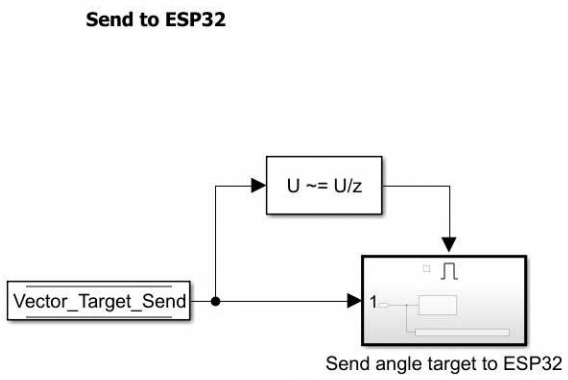


Figura 7.18: Subsistema de envío de datos al ESP32

Que en el interior contiene lo siguiente:

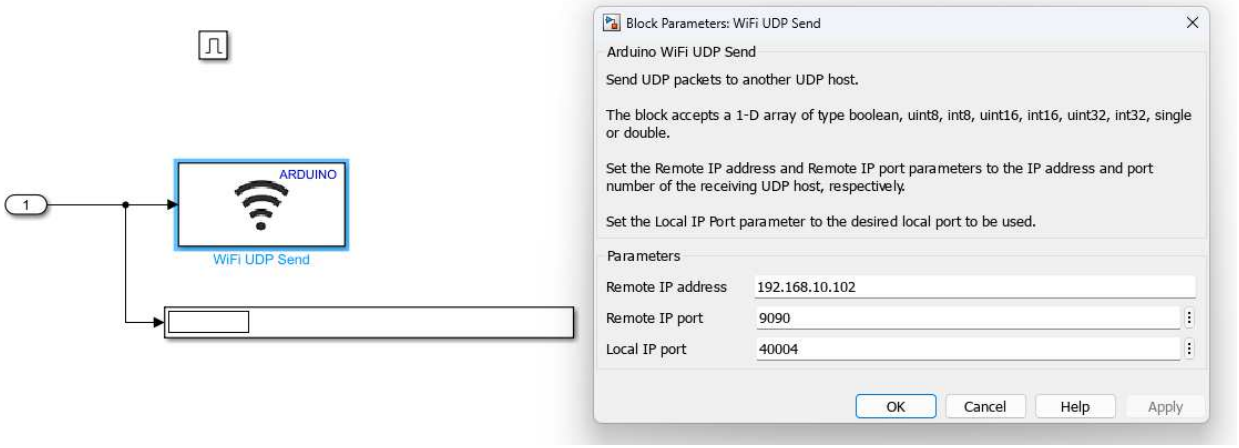


Figura 7.19: Interior del subsistema de envío de datos hacia el ESP32

Para que el envío de consignas al ESP32 sea efectivo, es necesario configurar correctamente la dirección IP de destino y el puerto de escucha en el modelo de Simulink. Esta configuración se realiza dentro del bloque de transmisión *WiFi UDP Send* (ver figura 7.19).

En concreto, la dirección IP debe corresponder con la asignada al ESP32 dentro de la red local creada con el router del robot, mientras que el puerto remoto debe coincidir con el configurado en el código del microcontrolador ESP32 para WiFi UDP. La dirección IP estática asignada al ESP32 es la 192.168.10.102 y el puerto que se ha configurado para que el Arduino MKR WiFi 1010 envíe y el ESP32 reciba (escuche) es el 9090.

A continuación, en el flujo de control, se presenta el conjunto de bloques (ver figura 7.20) responsables de gestionar la compensación en los motores de tracción.

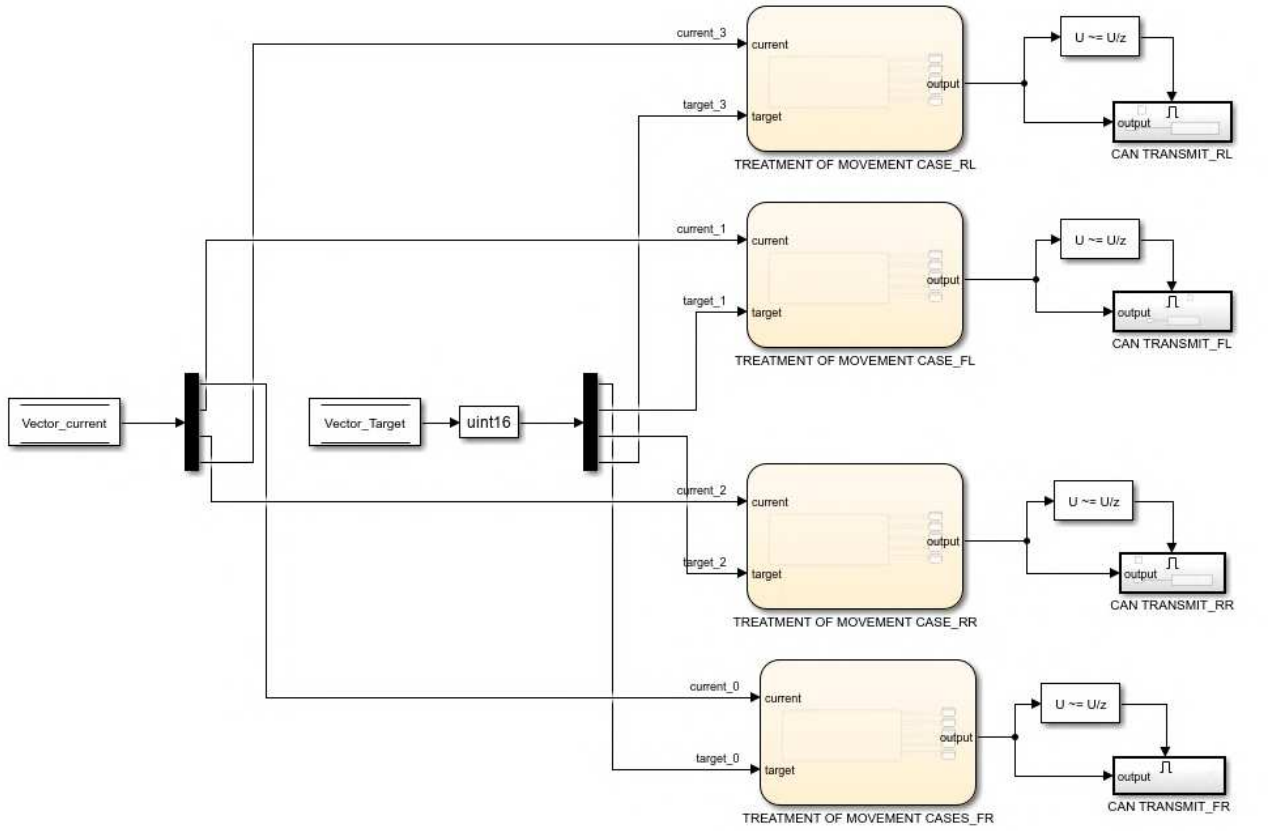


Figura 7.20: Subsistemas para el control de la compensación de los motores de tracción

La necesidad de compensación por parte del motor de tracción surge para evitar que, durante el giro controlado de la oruga mediante el motor paso a paso, se desplace o se retuerza de forma no deseada la goma. Dado que el movimiento inducido por el paso a paso genera una rotación sobre un extremo de la oruga, se requiere que el motor de tracción actúe simultáneamente para mantener el centro de la oruga aproximadamente en la misma posición.

Esta compensación no es constante, sino que depende directamente del ángulo actual de la oruga, el cual se obtiene a través del encoder correspondiente. En función de dicho ángulo, la estrategia de compensación varía.

El sistema de referencia de la oruga frontal derecha es el siguiente (ver figura 7.21), partiendo de este, se ha realizado la programación de las otras tres:

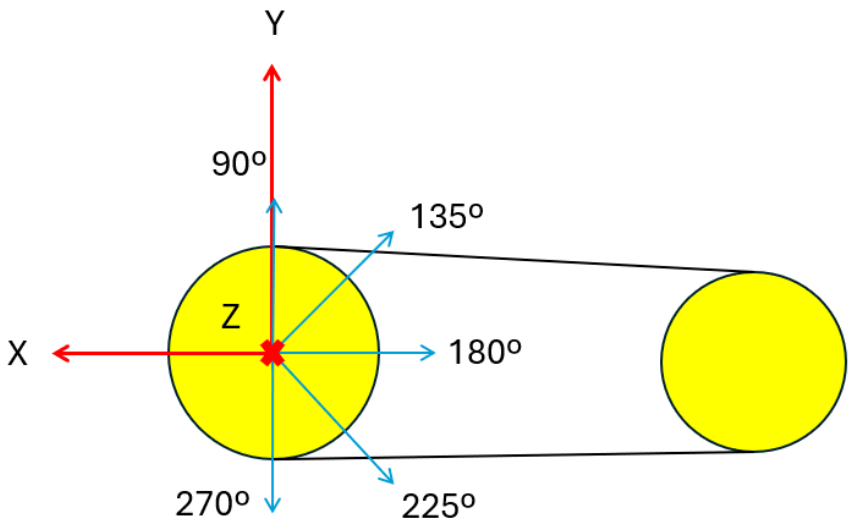


Figura 7.21: Sistema de referencia de la oruga frontal derecha

- **Si el ángulo de la oruga se encuentra entre 270º y 180º:** La polea pasiva de la oruga estará apoyando con el suelo. En este caso, el motor de tracción debe aplicar una compensación proporcional, calculada en función del ángulo, para anular el desplazamiento que generaría el paso a paso. La compensación debe ser precisa, por ello se ajusta continuamente la velocidad del motor de tracción conforme se modifica el ángulo en el que se encuentra la oruga. A medida que se acerca hacia la vertical de los 270º, la velocidad irá disminuyendo.
- **Si el ángulo está entre 180º y 90º:** En esta situación, la polea activa de la oruga, es la que estará apoyando con el suelo. La lógica es más simple; basta con que el motor de tracción gire en el sentido opuesto al del motor paso a paso, con la velocidad constante del paso a paso.

Para esbozar de manera visual el recorrido de la oruga y calcular la velocidad que deben alcanzar los motores de tracción durante el desplazamiento de los motores paso a paso, se ha representado el sistema mediante el esquema mostrado en la figura 7.22, el cual detalla el análisis geométrico del movimiento de la oruga delantera derecha tomando como referencia el desplazamiento del centro de la polea pasiva conforme ésta rota impulsada por el motor paso a paso.

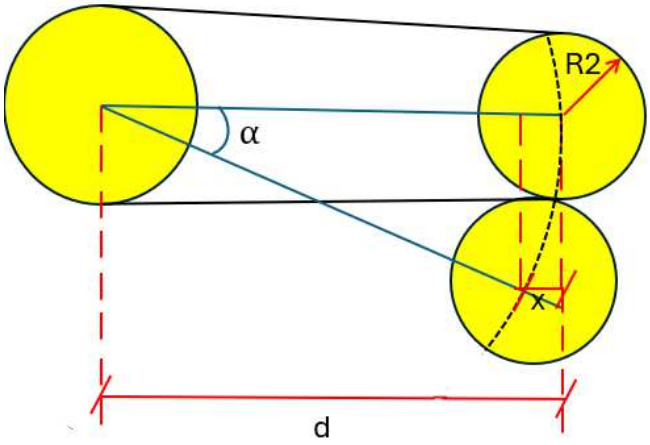


Figura 7.22: Esquema para el estudio de la compensación

Sean:

- R_2: Radio de la rueda menor que forma parte del sistema de la oruga.
- d: Distancia entre ejes de la oruga (longitud entre los centros de las ruedas mayor y menor).
- x: Desplazamiento horizontal del centro de la rueda menor, que se desea contrarrestar con la acción del motor de tracción.
- α: Ángulo que se desplaza el centro de la rueda menor respecto de la horizontal mientras se realiza la compensación.

La finalidad de este análisis es obtener una expresión que relacione el ángulo α con el desplazamiento horizontal x. Esta relación permitirá calcular la velocidad lineal (y posteriormente angular) que debe tener el motor de tracción para compensar el movimiento inducido por el paso a paso, asegurando que la oruga permanezca estacionaria o mantenga su trayectoria deseada sin generar tensiones ni deslizamientos indebidos.

Una vez obtenida la expresión de $x(\alpha)$, se podrá derivar con respecto al tiempo para obtener la velocidad de compensación necesaria, que se transformará posteriormente en la consigna de velocidad para el motor de tracción.

Por ello, para iniciar el análisis, se parte de la expresión del desplazamiento horizontal x del centro de la rueda menor en función del ángulo α:

$$x = d \cdot (1 - \cos(\alpha)) \tag{7.2}$$

Donde:

- x es el desplazamiento horizontal del eje de la rueda menor.
- d es la distancia entre los ejes de las ruedas que componen la oruga.

- α es el ángulo que describe la posición angular del eje de la rueda menor respecto a la horizontal.

Si se deriva la ecuación (7.2) con respecto al tiempo, se obtiene la velocidad lineal que debe desarrollar el motor de tracción para compensar el desplazamiento inducido por el motor paso a paso:

$$\frac{dx}{dt} = v_{servo} = d \cdot \sin(\alpha) \cdot \frac{d\alpha}{dt} \tag{7.3}$$

Donde:

- v_{servo} es la velocidad lineal que debe aplicar el motor de tracción para mantener la oruga estática.
- $\frac{d\alpha}{dt}$ es la velocidad angular del motor paso a paso, que se considera constante y se denota como $\omega_{stepper}$.

Por tanto, la expresión final queda:

$$v_{servo} = d \cdot \sin(\alpha) \cdot \omega_{stepper} \tag{7.4}$$

Esta velocidad de compensación se convierte posteriormente en una consigna de velocidad angular para el motor de tracción, teniendo en cuenta el radio de la rueda motriz (polea activa de la oruga).

Finalmente, para obtener la consigna de velocidad angular del motor de tracción, se realiza la conversión de la velocidad lineal v_{servo} utilizando el radio R_2 de la rueda motriz (la menor que forma parte de la oruga, polea pasiva):

$$\omega_{servo} = \frac{v_{servo}}{R_2} \tag{7.5}$$

Sustituyendo la expresión de v_{servo} obtenida anteriormente en la ecuación (7.4):

$$\omega_{servo} = \frac{d \cdot \sin(\alpha) \cdot \omega_{stepper}}{R_2} \tag{7.6}$$

Esta expresión permite calcular la velocidad angular que debe aplicar el motor de tracción para mantener la oruga inmóvil mientras el motor paso a paso modifica la orientación de la rueda menor.

Analizando las unidades obtenidas en la expresión anterior, se observa que la velocidad angular ω_{servo} se encuentra en radianes por segundo (rad/s). Sin embargo, el controlador de los motores de tracción está diseñado para recibir consignas de velocidad en grados por segundo (°/s). Por ello, es necesario realizar una conversión de unidades, multiplicando por el factor $\frac{180}{\pi}$:

$$\omega_{servo} = d \cdot \sin(\alpha) \cdot \omega_{stepper} \cdot \frac{1}{R_2} \cdot \frac{180}{\pi} [\frac{^\circ}{s}] \tag{7.7}$$

De este modo, se obtiene la consigna de velocidad angular del motor de tracción en grados por segundo, lista para ser enviada como consigna al motor de tracción correspondiente.

Analizando la expresión obtenida, se observa que tanto la velocidad angular del motor paso a paso ($\omega_{stepper}$), como la distancia entre ejes de la oruga (d), y el radio de la rueda menor (R_2), son constantes conocidas del sistema. Por tanto, se concluye que la única variable que influye dinámicamente en la velocidad de compensación que deben seguir los motores de tracción es el ángulo α, es decir, la posición angular instantánea del centro de la rueda menor respecto a la horizontal.

Esta dependencia hace que la velocidad de compensación que se transmite a los motores de tracción varíe en función del ángulo actual de la oruga, permitiendo así un movimiento coordinado entre el paso a paso y el motor de tracción, evitando torsiones y asegurando que la oruga permanezca en su lugar mientras se realiza el posicionamiento angular.

Una vez realizado este análisis, se puede explicar correctamente el funcionamiento de los siguientes subsistemas del modo de funcionamiento 2.

Como se muestra en la siguiente figura (ver figura 7.23) , el control de la compensación se realiza en función del ángulo actual de la oruga y el ángulo en el que se quiere posicionar. Por ello, las entradas son los vectores internos $\mathrm{Vector\_Current}$ y $\mathrm{Vector\_Target}$, provenientes del ESP32 y del PC, respectivamente, que se separan mediante demux para cada *Chart*.

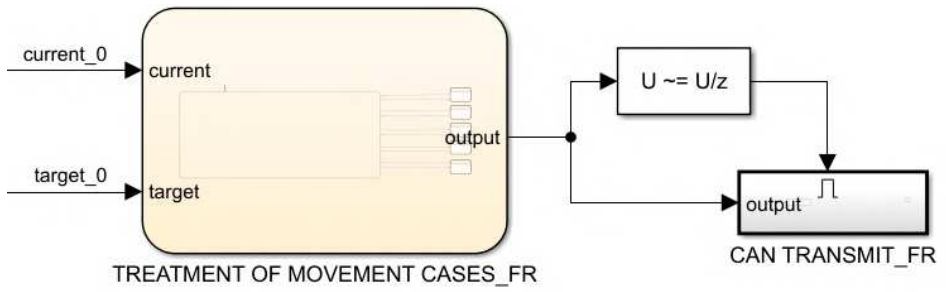


Figura 7.23: Chart encargado de la selección de la actuación para la compensación de la oruga F.R.

La programación se ha realizado mediante una máquina de estados (ver figura 7.24), en la que las entradas son las anteriormente comentadas y la salida será el vector de velocidad a enviar al motor de la F.R. en este caso.

Figura 7.24: Máquina de estados para el control de la compensación de la oruga F.R.

El estado principal (*State0*) será, en el que dependiendo de dónde se encuentre la oruga y hacia dónde se quiera posicionar, controle el estado siguiente a ejecutar (*Case0, Case1, Case2, Case3, Case4*):

Código 7.1: Código del estado State0 en el diagrama de estados

```
State0
entry:
selector = -1;
during:
if (abs((int32(target) - int32(current)))< 2)
    selector = 0; % Caso 0: Parada del servo motor cuando ambos ángulos son aproximadamente iguales

elseif (target < current) && (current < 180)
    selector = 1;  % Caso 1: Subida del paso a paso hacia 90 º, el servo motor  actuará con una ...
        velocidad constante de 5º/s en s.h

elseif (target > current) && (current < 180)
    selector = 2;  % Caso 2: Bajada del paso a paso hacia 180º, el servo motor actuará con una ...
        velocidad constante de 5º/s en s.ah

elseif (target > current) && (current > 180)
    selector = 3;  % Caso 3: Bajada del paso a paso hacia 270º, el servo motor actuará con una ...
        velocidad variable dependiente del ángulo actual en s.ah

elseif (target < current) && (current > 180)
    selector = 4;  % Caso 4: Subida del paso a paso hacia 180º, el servo motor actuará con una ...
        velocidad variable dependiente del ángulo actual en s.h

else
    selector = 5;
end
```

La ejecución del código anterior (ver código 7.1) dará paso a un *Simulink State*, un bloque de Simulink que permite programar en su interior diferentes subsistemas, dependiendo de una o más variables. En este caso, la variable selector.

Si selector = 0: Se manda el mensaje CAN encargado de la parada del motor según el manual de programación de los motores (ver figura 7.25). Esto significa que no necesita actuar el motor porque la diferencia entre el ángulo objetivo y el actual es mínima.

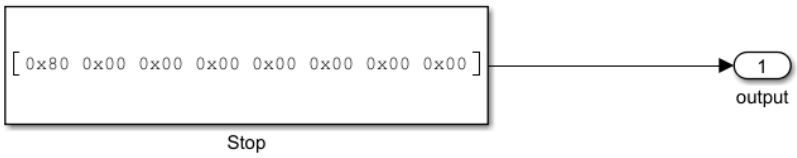


Figura 7.25: Case0

Si selector = 1: El motor paso a paso girará en sentido antihorario, y por ello el servomotor debe ir en sentido horario a una velocidad constante de unos 5º/s (ver figura 7.26).

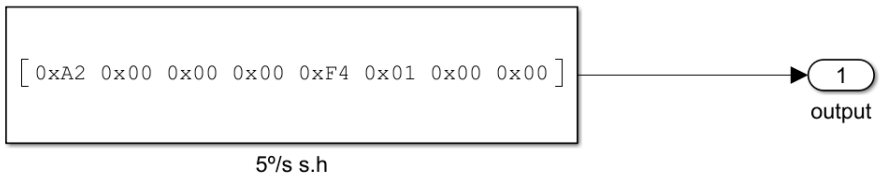


Figura 7.26: Case1

Si selector = 2: El motor paso a paso girará en sentido horario, y por ello el servomotor debe ir en sentido antihorario a una velocidad constante de unos 5º/s (ver figura 7.27). De ahí que la velocidad a enviar es de -5º/s y el mensaje es:

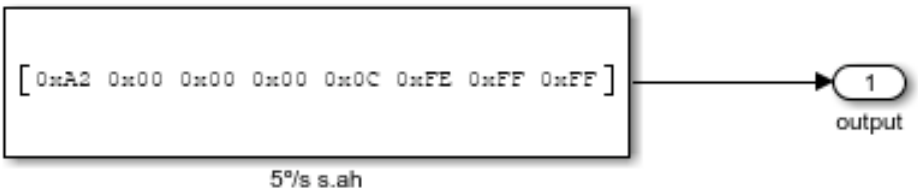


Figura 7.27: Case2

En estos 3 casos no ha sido necesario el uso de los vectores de entrada, ya que en ninguno de los estados era necesario ver en cada momento la posición de la oruga.

Si selector = 3: En este caso, es necesaria la relación matemática obtenida previamente para realizar la compensación.

El State Simulink del Case3 (ver figura 7.28), contiene la ecuación:

$$\omega_{servo} = \frac{d \cdot \sin(\alpha) \cdot \omega_{stepper}}{R_2} \tag{7.8}$$

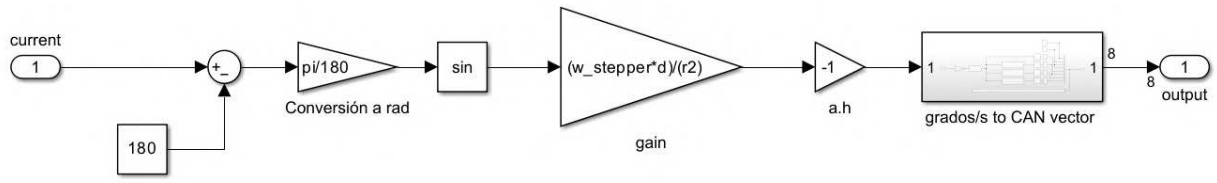


Figura 7.28: Case3

Y seguida a una ganancia de -1, donde se impone el sentido antihorario y tras esto el subsistema comentado al principio del capítulo de conversión de grados/segundo a un vector CAN.

Si $\mathrm{selector}$ = 4: En este caso es necesaria también la relación matemática, ya que la subida hasta los 180º depende del ángulo medido (ver figura 7.29).

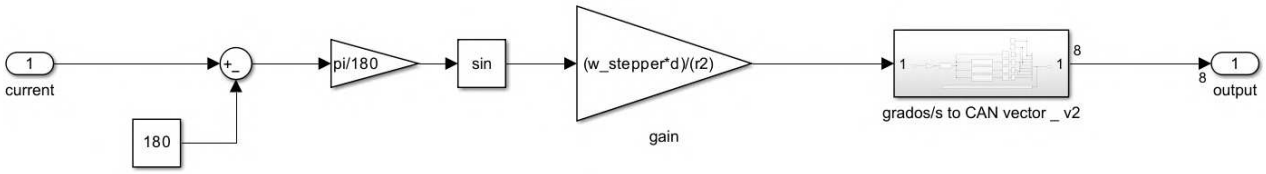


Figura 7.29: Case4

En este caso, el sentido del servomotor es horario y no es necesaria una ganancia de -1 para imponer el sentido. En cuanto a la programación de las orugas restantes:

- La oruga R.L. es igual en programación a la F.R.
- Las orugas F.L. y R.R. son similares en programación a la F.R solo que los estados Simulink 1 y 2 se intercambian, al igual que el 3 y 4.

Gracias al método de programación realizado con respecto a la oruga frontal derecha (F.R.), no ha sido necesario desarrollar una lógica de control compleja e individualizada para cada oruga. La oruga trasera izquierda (R.L.) comparte directamente la misma lógica que la F.R., mientras que las orugas situadas en la diagonal opuesta requieren únicamente una inversión del sentido de giro en cada uno de los estados definidos en Simulink.

En la salida de cada bloque *Chart* (la variable output) se encuentra la asignación del mensaje CAN correspondiente (ver figuras 7.30 y 7.31), el cual se almacena en el vector interno asociado a cada oruga. Esta estrategia de control se ha planteado con el objetivo de optimizar la comunicación con el microcontrolador, de modo que los mensajes CAN solo se transmitan cuando exista una variación con respecto al mensaje previamente enviado, evitando así una sobrecarga innecesaria en el bus de comunicación.

Una vez tenemos los mensajes creados, simplemente queda enviarlos. Para ello se tiene la última parte del modelo programado en el Arduino MKR WiFi 1010 que se describirá al final del capítulo.

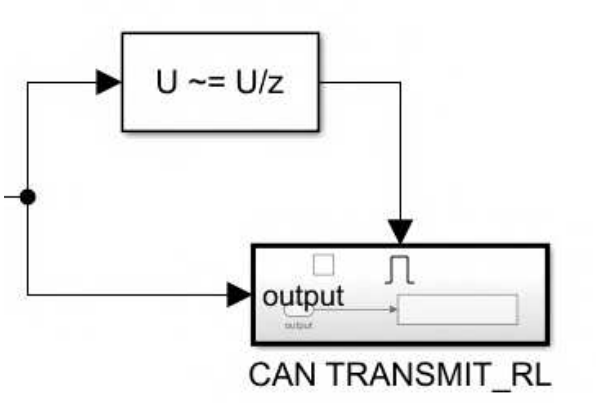


Figura 7.30: Subsistema activo por cambio de entrada de la oruga R.L.

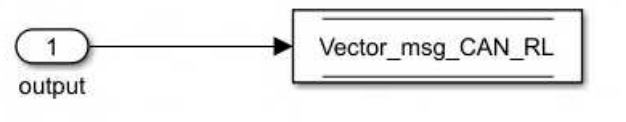


Figura 7.31: Interior del subsistema CAN TRANSMIT RL

Se presentan unas imágenes de Horu, controlado mediante el modo de funcionamiento 2, en dos posiciones diferentes (ver figuras 7.32 y 7.33):

Figura 7.32: Horu situado a 225º

Figura 7.33: Horu situado a 135º

7.5.4 Subsistema del modo de funcionamiento 3 y 4. Control incremental

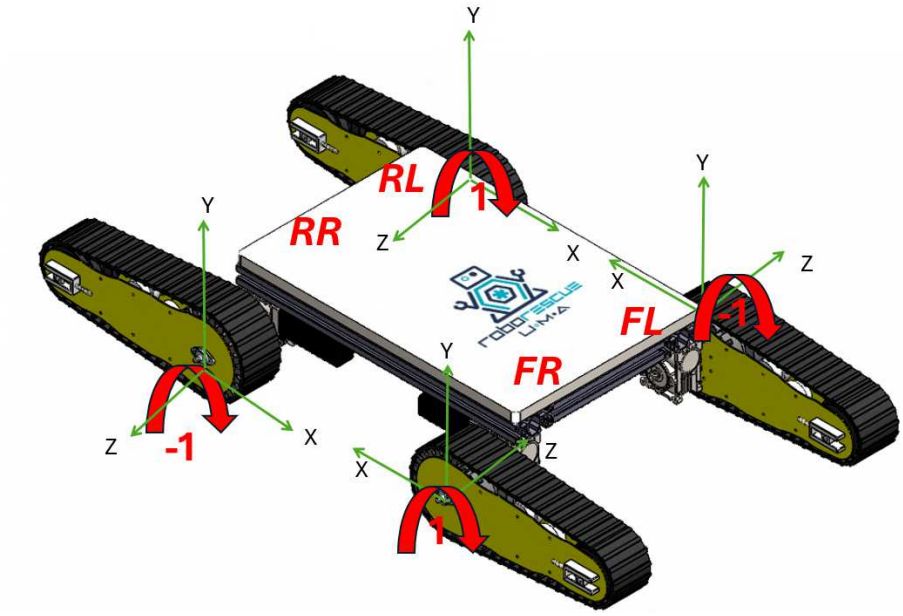


Figura 7.34: Sistema de referencia de sentidos de giro horarios, respecto de la oruga F.R.

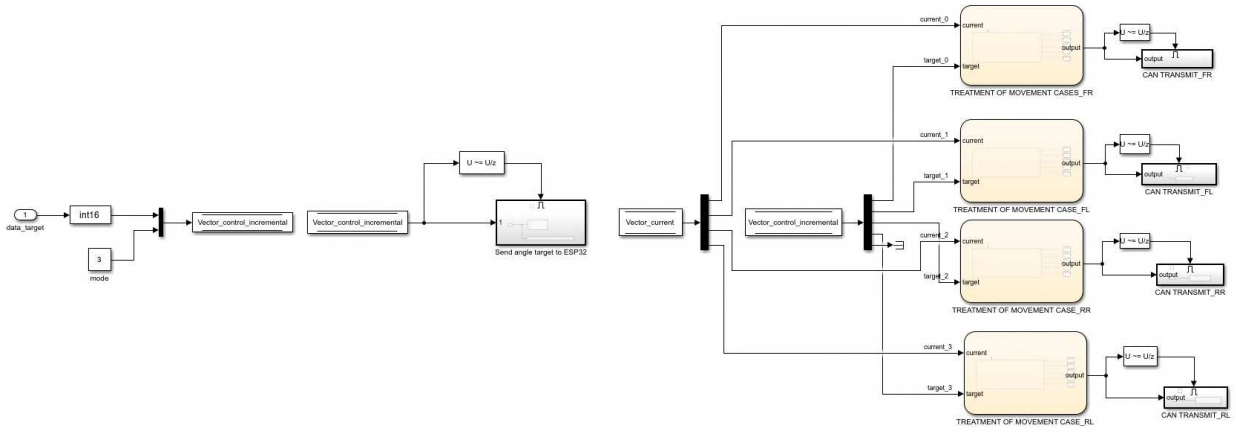


Figura 7.35: Subsistema del modo de control incremental de posición

El subsistema encargado de realizar el control incremental de la posición se muestra en la figura 7.35. Este modo de funcionamiento, como se ha explicado anteriormente, permite nivelar el robot de manera manual y situar las orugas en una posición favorable para superar obstáculos, entre otras acciones. Los ejemplos ilustrativos de este modo pueden consultarse al final de la sección, en la figura 7.39. En este modo se actuará sobre los ocho motores, ya que nuevamente es necesaria la compensación.

El subsistema se habilitará cuando el modo recibido sea ”3” o ”4”, dado que las consignas enviadas desde el PC son idénticas en ambos casos. Por tanto, la entrada principal del subsistema es nuevamente el vector $\mathrm{data\_target}$, el cual contiene las cuatro consignas correspondientes al sentido de giro de cada oruga de forma independiente (ver figura 7.34).

Los primeros bloques dentro del subsistema corresponden al envío de consignas al ESP32 (ver figura 7.36). Al igual que en el modo anterior (control de posición absoluta), el ESP32 será el encargado de gestionar los motores de elevación y de enviar, en todo momento, los ángulos actuales de las orugas hacia el Arduino MKR WiFi 1010.

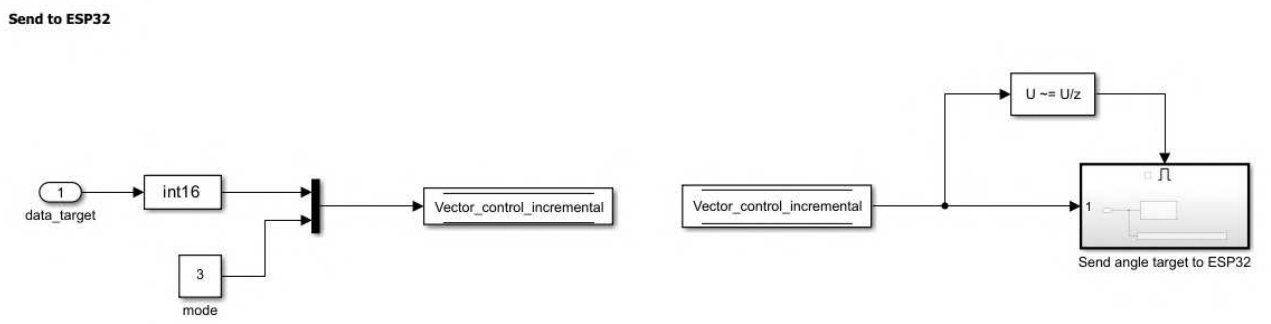


Figura 7.36: Subsistemas encargados del envío de datos al ESP32

El $\mathrm{data\_target}$ recibido para los modos 3 y 4, como se ha comentado, corresponderá a los sentidos de giro de las orugas (de los motores paso a paso), los cuales se pasarán localmente al $\mathrm{vector\_control\_incremental}$. Al igual que en el modo anterior, es necesario enviar los datos al ESP32, donde se indicarán los sentidos de giro y el modo de funcionamiento.

Tras el subsistema de envío al ESP32, se encuentra el siguiente subsistema encargado del control de los motores de tracción (ver figura 7.37).

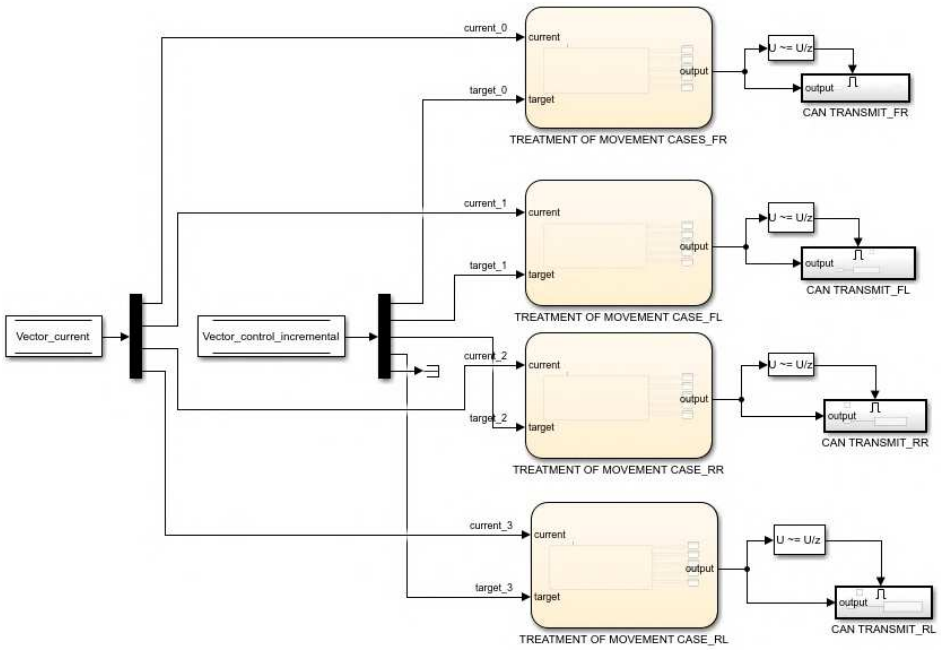


Figura 7.37: Subsistemas encargados de la compensación en el modo de control incremental de posición

La estructura del sistema mantiene una apariencia similar al modo de operación anterior. Se parte de unos datos de entrada que corresponden a los ángulos actuales del vehículo, junto con un vector de referencia que define los valores objetivo que en este caso corresponderán a los sentidos de giro de los motores paso a paso. A partir de esta información, se emplean bloques *Stateflow Chart*, que permiten gestionar, en función del sentido de giro, la compensación adecuada para cada motor de tracción. Finalmente, se realiza el envío de los datos mediante mensajes CAN, que se encargan de transmitir las órdenes correspondientes a los motores para ejecutar las acciones determinadas por el sistema de control.

Como se muestra en la figura 7.34, el signo asignado al sentido de giro horario varía en función de la oruga que se desee mover. Cabe destacar que toda la lógica de programación se ha desarrollado tomando como referencia la oruga F.R., con el objetivo de simplificar el código. Debido a esta elección, el sentido de giro horario definido para la oruga F.R. coincide con el de la oruga R.L., ya que ambas se encuentran en la misma diagonal. Por el contrario, este sentido será opuesto para las orugas situadas en la diagonal contraria, es decir, F.L. y R.R. .

En el interior del chart *TREATMENT OF MOVEMENT CASE FR*. tenemos la siguiente máquina de estados (ver figura 7.38):

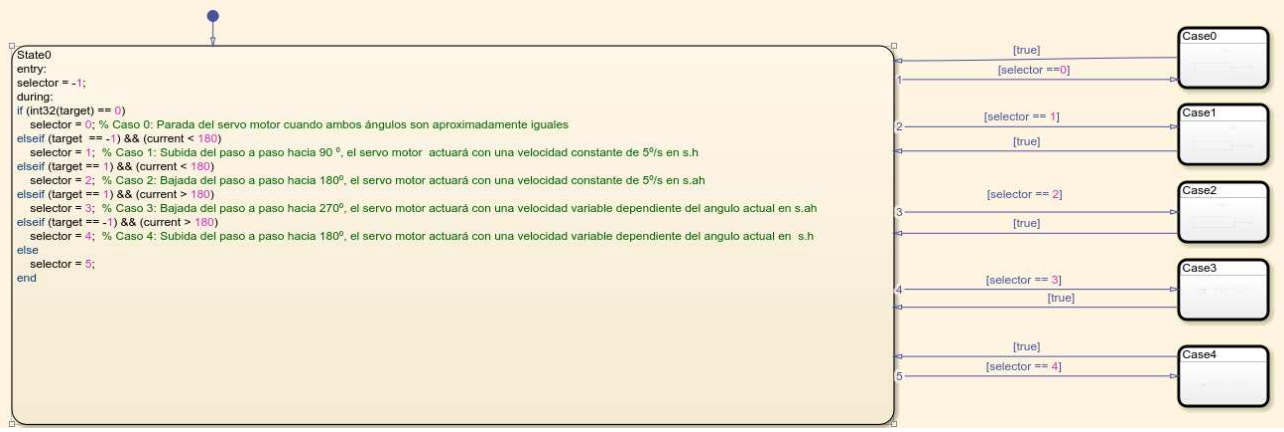


Figura 7.38: Caso de la oruga F.R. en el control incremental de posición

En el estado principal, el código es el siguiente (ver código 7.2):

Código 7.2: Stateflow Chart F.R y R.L

```
State0
entry:
selector = -1;
during:
if (int32(target) == 0)
    selector = 0; % Caso 0: Parada del servo motor
elseif (target  == -1) && (current < 180)
    selector = 1;  % Caso 1: Subida del paso a paso hacia 90 º, el servo motor  actuará con una ...
        velocidad constante de 5º/s en s.h
elseif (target == 1) && (current < 180)
    selector = 2;  % Caso 2: Bajada del paso a paso hacia 180º, el servo motor actuará con una ...
        velocidad constante de 5º/s en s.ah
elseif (target == 1) && (current > 180)
    selector = 3;  % Caso 3: Bajada del paso a paso hacia 270º, el servo motor actuará con una ...
        velocidad variable dependiente del angulo actual en s.ah
elseif (target == -1) && (current > 180)
    selector = 4;  % Caso 4: Subida del paso a paso hacia 180º, el servo motor actuará con una ...
        velocidad variable dependiente del angulo actual en  s.h
else
    selector = 5;
end
```

De nuevo, el código tiene la finalidad de seleccionar el modo de compensación de los motores de tracción, pero en este caso dependiendo de si queremos ir en sentido horario o antihorario con el motor paso a paso. Cada *Simulink Case* es igual que en el subsistema del modo anterior (control de posición), por lo que no se hará de nuevo la explicación de cada caso.

Controlar el sentido de giro de la oruga, ya sea en dirección horaria o antihoraria, permite elevar o descender la esquina correspondiente del robot, facilitando así el ajuste de su inclinación o nivelación en función de las necesidades del entorno. En el caso de que no se necesite que se realice nada sobre ella, la consigna será de 0.

Como se ha explicado en el control del PC, desde el mando de Xbox podremos realizar de manera manual un nivelado usando esta lógica de control.

El anterior código es válido para la oruga R.L. y para las otras dos restantes sería el

siguiente (ver código 7.3):

Código 7.3: Stateflow Chart F.L. y R.R

```
State0
entry:
selector = -1;
during:
if (int32(target) == 0)
    selector = 0; % Caso 0: Parada del servo motor
elseif (target  == 1) && (current < 180)
    selector = 1;  % Caso 1: Subida del paso a paso hacia 90 º, el servo motor  actuará con una ...
        velocidad constante de 5º/s en s.h
elseif (target == -1) && (current < 180)
    selector = 2;  % Caso 2: Bajada del paso a paso hacia 180º, el servo motor actuará con una ...
        velocidad constante de 5º/s en s.ah
elseif (target == -1) && (current > 180)
    selector = 3;  % Caso 3: Bajada del paso a paso hacia 270º, el servo motor actuará con una ...
        velocidad variable dependiente del angulo actual en s.ah
elseif (target  == 1) && (current > 180)
    selector = 4;  % Caso 4: Subida del paso a paso hacia 180º, el servo motor actuará con una ...
        velocidad variable dependiente del angulo actual en  s.h
else
    selector = 5;
end
```

Donde lo único que cambia con respecto al código de F.R. son los sentidos de giro, es decir, que en F.R. se establece como horario con 1, mientras que en el código anterior se establece como horario con -1.

En el modelo del PC, como se ha comentado en el capítulo anterior, los modos 3 y 4 envían los mismos datos al Arduino MKR WiFi 1010 (los sentidos de giro). Por lo tanto, como se muestra en la figura 7.9, este subsistema se activará tanto si el modo es 3 como si es 4, gracias a la puerta lógica OR.

Figura 7.39: Horu en el modo de posición incremental

7.5.5 Subsistema del modo de funcionamiento 5. Nivelado automático

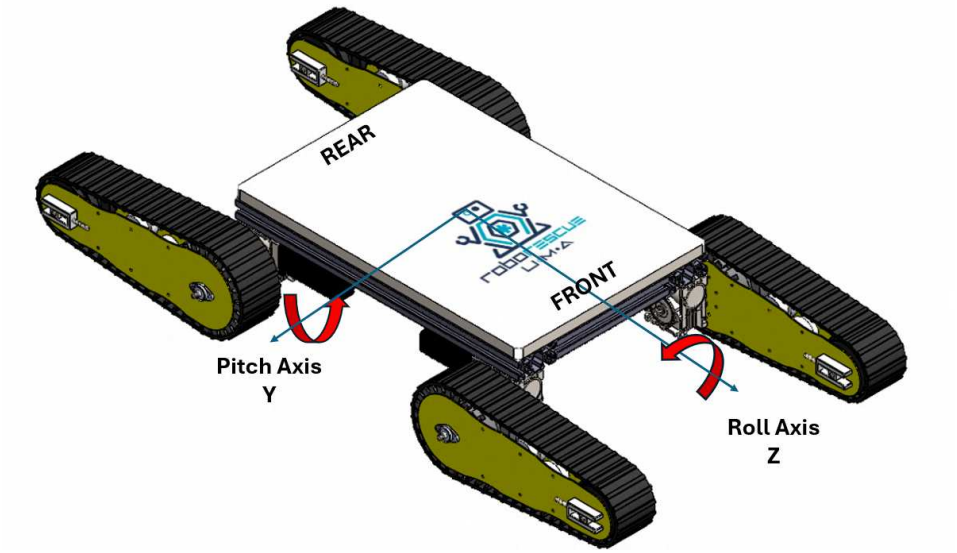


Figura 7.40: Sistema de referencia de ángulos Roll y Pitch respecto de la IMU

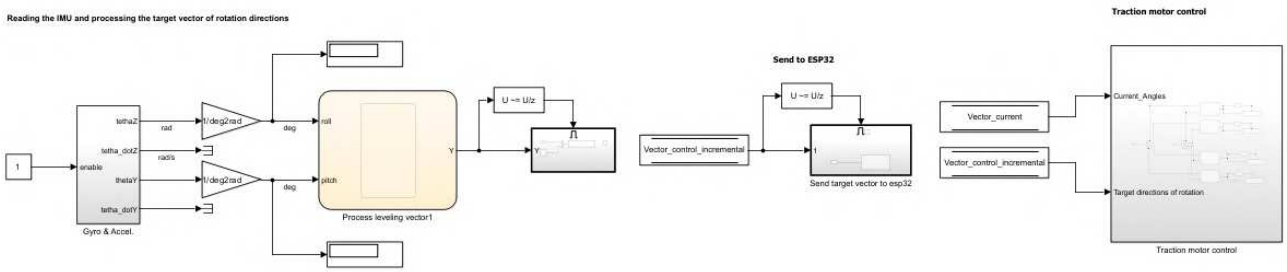


Figura 7.41: Subsistemas encargados del nivelado automático

El siguiente modo de funcionamiento (modo 5) tiene como objetivo realizar un nivelado de manera automática del robot cuando el modo recibido es el 5 (ver figura 7.41); usando como datos para el nivelado automático, los ángulos de inclinación Y, Z, que se extraen de una IMU MPU9250. La estrategia de control de los motores de tracción se realizará de la misma manera que en los modos anteriores (3 y 4), mediante un control incremental de la posición, por lo que el subsistema de control de compensación de estos será el mismo que antes.

Para realizar el nivelado, es necesario definir unos ejes de coordenadas del vehículo, según la ubicación de la IMU que se encuentra anclada al chasis del vehículo en la parte frontal de este, justo al lado de los dos microcontroladores (ver figura 2.14). El sistema de referencia de la IMU elegido según su ubicación en el robot es el que se muestra en la figura anterior (ver figura 7.40). Roll y pitch son ángulos que definen la orientación del robot respecto al plano horizontal.

- Roll (ϕ): rotación sobre el eje Z (longitudinal), define inclinación lateral. Será positivo cuando se incline en sentido horario mirando desde la parte trasera del vehículo.
- Pitch (θ): rotación sobre el eje Y (transversal), define inclinación hacia adelante o atrás. Será positivo cuando se incline hacia arriba mirando desde la parte trasera del vehículo.

Por lo tanto, lo primero que debe realizar el modelo de control de nivelado automático será adquirir los datos del acelerómetro y giroscopio del MPU9250 conectado al Arduino MKR WiFi 1010. De ello se encarga la primera parte del subsistema:

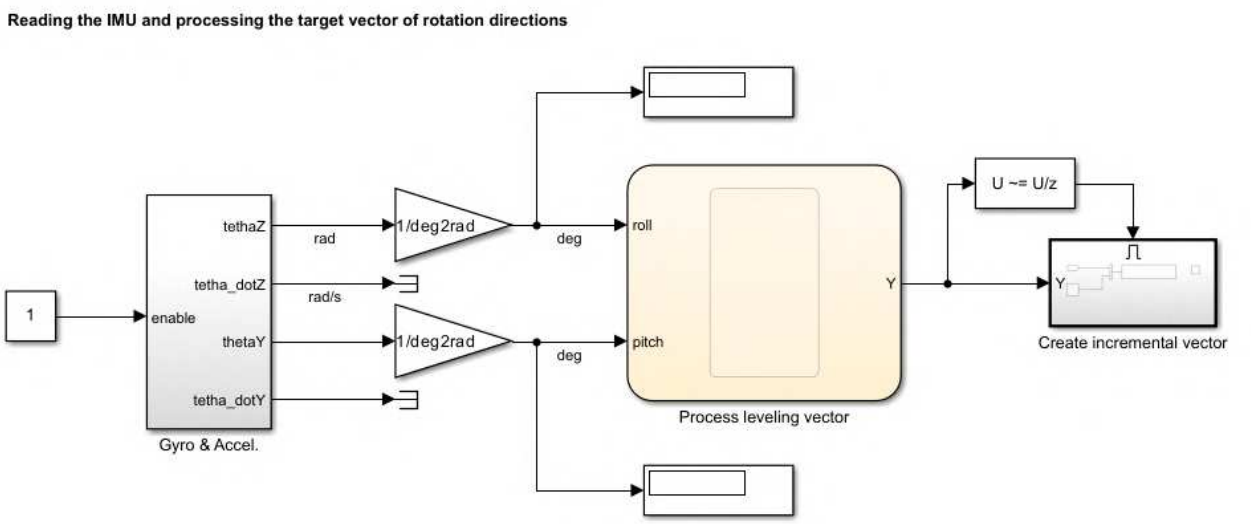


Figura 7.42: Subsistema encargado de la adquisición de ángulos

El primer subsistema de la figura 7.42 se encarga de recibir los ángulos del MPU9250. Simulink integra el bloque *MPU9250 IMU Sensor* (ver figura 7.44), que permite leer los valores del acelerómetro, giroscopio, magnetómetro y temperatura que está midiendo la IMU. Por lo tanto, esto ha facilitado enormemente la programación, ya que se pueden leer los datos sin necesidad de procesarlos.

Un problema encontrado durante el desarrollo del nivelado ha sido que los datos medidos no eran lo suficientemente estables para poder realizar un control en base a ellos. Además, los ángulos de inclinación en los ejes no se devuelven directamente. Por lo que ha sido necesario desarrollar un algoritmo que permita calcular los ángulos, realizar una calibración del giroscopio al inicio y un filtrado de los datos.

Para ello, se ha utilizado un algoritmo que se implementó en un Trabajo de Fin de Grado anterior, *CONTROL AUTO-BALANCEADO TIPO PÉNDULO INVERTIDO PARA EL ROBOT MÓVIL PIERO* de D. Jesús Muñoz Martínez (ver figura 7.43). Este método permitía obtener el ángulo pitch de un robot de un eje, pero para el caso del robot Horu, es necesario realizarlo tanto en pitch como en roll, por lo que el modelo se ha visto modificado de la siguiente manera (ver figura 7.43):

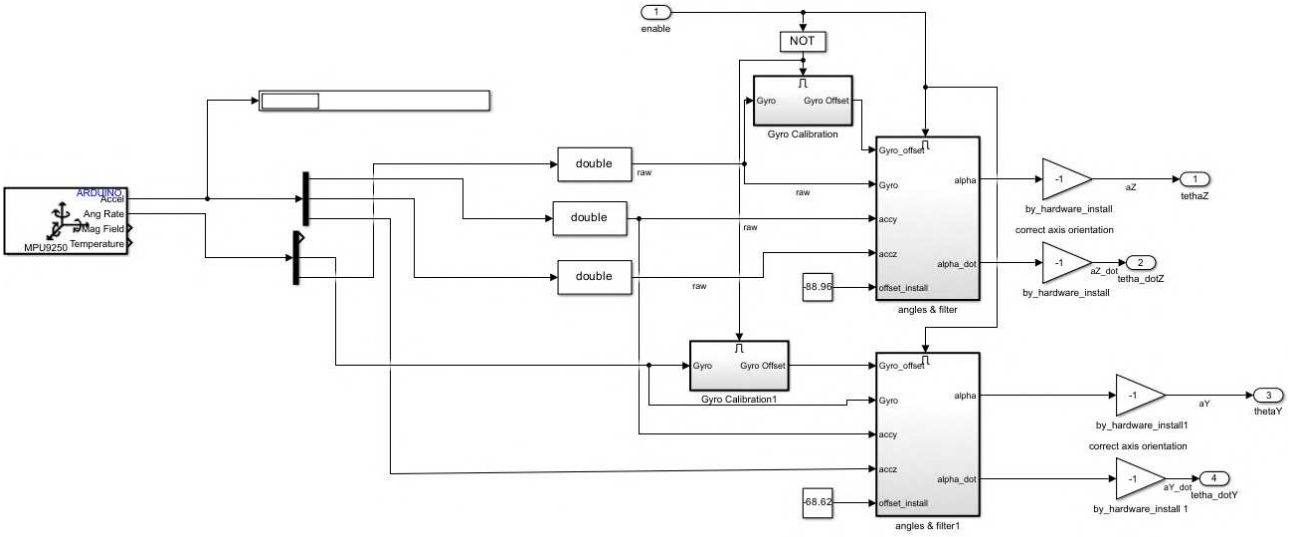


Figura 7.43: Algoritmo de adquisición de ángulos de inclinación roll y pitch (interior subsistema Gyro & Accel.)

Este subsistema (ver figura 7.43) permite ajustar el offset de los ángulos al iniciar el nivelado. Posteriormente, a partir de las ecuaciones correspondientes, calcula los valores de los ángulos. Además, el subsistema es capaz de realizar un filtrado de los datos obtenidos (ver figura 7.44).

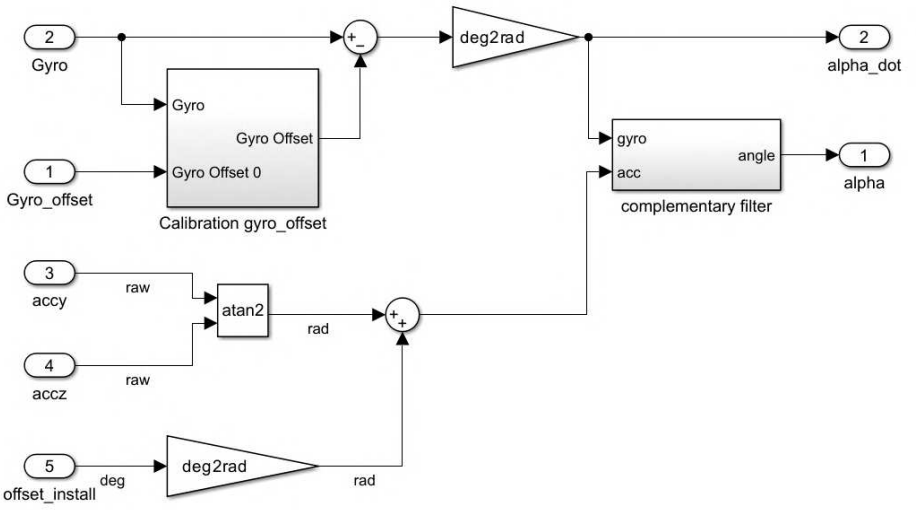


Figura 7.44: Subsistemas encargados del filtrado y cálculo de los ángulos roll y pitch

Una vez se tienen los ángulos de inclinación del vehículo en radianes con las ganancias que se ven en la figura 7.42, estos son convertidos a grados para realizar, mediante un *Chart*, el control sobre los motores, tanto de tracción como de elevación. En el *Chart Process leveling vector* se encuentra el siguiente código (ver código 7.4):

Código 7.4: Stateflow Chart modo de nivelado horizontal automático

```
Mode_5
entry:
    Y = int16([0, 0, 0, 0]);
during:
    limite = 2;
    Y_pitch = int16([0, 0, 0, 0]);
    Y_roll  = int16([0, 0, 0, 0]);
    % Corrección por pitch
    if (pitch > limite)
        Y_pitch = int16([1, -1, 1, -1]);
    elseif (pitch < -limite)
        Y_pitch = int16([-1, 1, -1, 1]);
    end
    % Corrección por roll
    if (roll > limite)
        Y_roll = int16([1, 1, -1, -1]);
    elseif (roll < -limite)
        Y_roll = int16([-1, -1, 1, 1]);
    end
    % Suma final
    Y = Y_pitch + Y_roll;
    % Saturación entre -1 y 1
    for i = 1:4
        if Y(i) > 1
            Y(i) = 1;
        elseif Y(i) < -1
            Y(i) = -1;
        end
    end
```

El código realiza un control dependiendo de un límite establecido, que es de 2 grados. Si el vehículo está inclinado hacia atrás, es decir, si el pitch es positivo (ángulo mayor de 0º), la corrección será inclinar el robot hacia delante. Por el contrario, si el pitch es negativo (ángulo menor de 0º), la corrección será inclinar el robot hacia atrás.

Por otro lado, en cuanto a la inclinación transversal, si se tiene que el roll es positivo (ángulo mayor de 0º), el vehículo estará inclinado hacia la derecha, por lo que la corrección es hacer que el robot se incline hacia la izquierda. Por otro lado, si el roll es negativo (ángulo menor de 0º), la corrección será inclinar el robot hacia la derecha compensando hasta que se llegue a 0º. La manera de actuación sobre los motores es que, mientras los de un mismo eje o tren, según el caso, bajen, los otros suben. Por ejemplo, si está inclinado hacia atrás, los motores del eje delantero (F.R. y F.L.) se elevarán (s.ah) y los motores del eje trasero descenderán (R.R. y R.L.), esto hará que el robot suba la parte trasera y baje o mantenga la parte delantera. Del mismo modo se actuará si se inclina hacia un lado u otro, de una manera combinada.

La estrategia de control que se ha implementado consiste en formar un vector, con las direcciones de giro horarias o antihorarias de la oruga completa, respecto de F.R. como en los modos de operación 3 y 4 descritos anteriormente. Se ha realizado así para controlar de manera eficiente el nivelado, en ambos ejes de manera simultánea.

Tras esto se encuentran los bloques descritos en modos anteriores, el envío del vector incremental al ESP32 y el procesamiento del control de los motores de tracción para realizar la compensación dependiente del vector incremental. Por lo que, como son los mismos bloques y la misma estrategia de control, no se describirán.

7.6 Envío y recepción de datos de los motores de tracción

7.6.1 Envío de mensajes CAN

Una vez configurados los cuatro mensajes CAN con las consignas de velocidad o de pare de los motores de tracción, únicamente resta su transmisión a los controladores de los motores correspondientes (ver figura 7.45 y 7.46) . A continuación, se expone el método empleado:

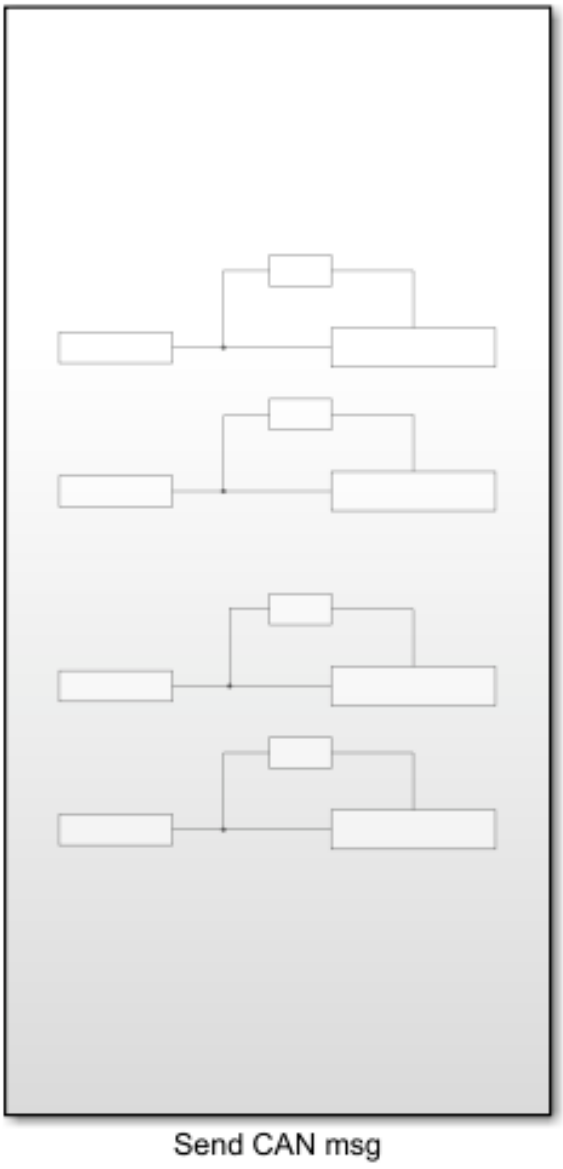


Figura 7.45: Subsistema encargado del envío de mensajes CAN hacia los motores de tracción

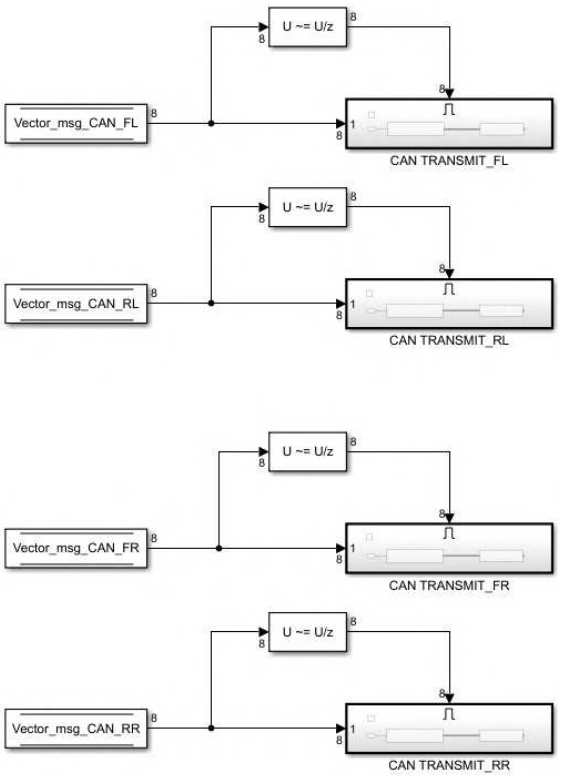


Figura 7.46: Interior del subsistema *Send CAN msg*

Dentro del subsistema *Send CAN msg* se encuentra el envío del mensaje CAN de cada oruga de manera independiente. Al tener un subsistema habilitado con *enable*, y bloque *Detect Change*, el envío se realizará solamente cuando el dato contenido en cada Vector_msg_XX cambie, como se ha ido realizando durante todo el proyecto.

Dentro del *subsistem CAN_TRANSMIT_FL* por ejemplo se encuentra lo siguiente (ver figura 7.47):

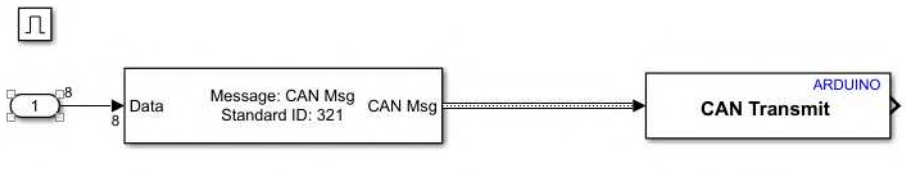


Figura 7.47: Subsistema de envío de mensaje CAN a la oruga F.L.

El primer bloque, el CAN Pack (bloque izquierdo de la figura 7.47) es un bloque que, utilizando los datos de entrada y especificando la ID de destino y la longitud en bytes de los datos a enviar, genera el mensaje CAN a enviar.

Para los comandos utilizados en este proyecto, solo es necesario especificar la ID del motor y, por supuesto, los datos que se desean enviar, los cuales cada motor interpretará como la acción a realizar. Cada motor posee una ID específica para la recepción de mensajes y otra distinta para el envío de información de retroalimentación hacia el Arduino MKR WiFi 1010. Estas IDs están expresadas en formato hexadecimal: 0x141, 0x142, 0x143 y 0x144 (correspondientes a 321, 322, 323 y 324 en decimal) para la recepción, y 0x241, 0x242, 0x243 y 0x244 (577, 578, 579 y 580 en decimal) para la emisión.

- El motor de la oruga F.L. tiene el ID: 321
- El motor de la oruga F.R. tiene el ID: 322
- El motor de la oruga R.R. tiene el ID: 323
- El motor de la oruga R.L. tiene el ID: 324

Por último, el bloque *CAN Transmit* envía el mensaje al destino especificado en el *CAN Pack*, haciendo uso del chip MCP2515 de la Shield CAN. El resultado de ese bloque es un mensaje de estado, un número de 8 bits que indica, si se ha dado un fallo, qué ha fallado concretamente. Si no se ha dado ningún fallo y el motor ha recibido bien el mensaje, el resultado mostrado será 00000000.

7.6.2 Lectura de mensajes CAN

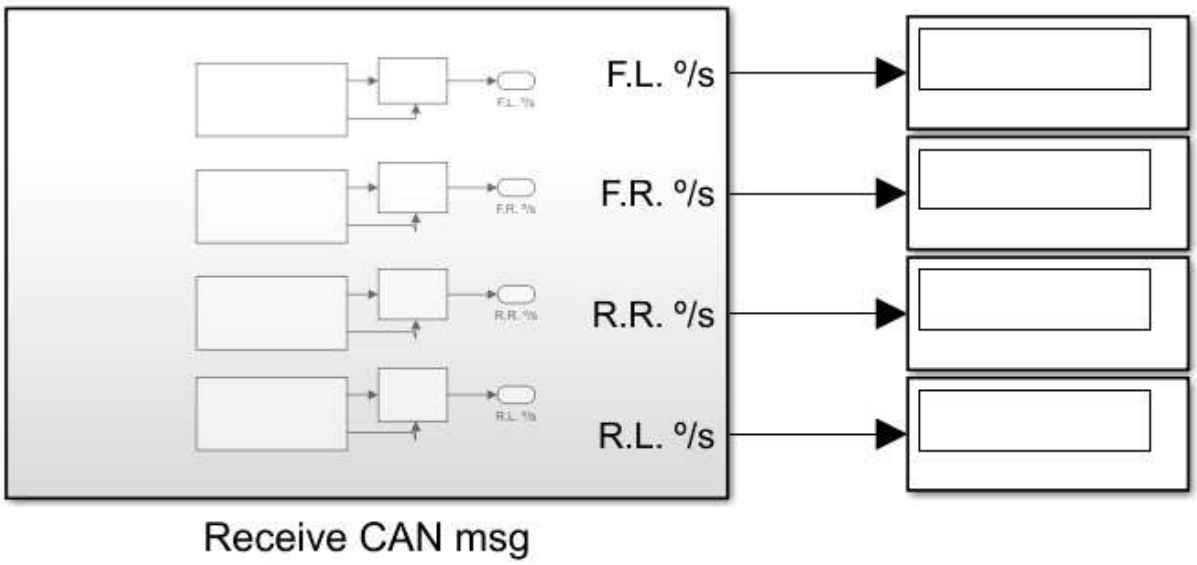


Figura 7.48: Recepción de datos CAN provenientes de los motores de tracción

En cuanto a la recepción de datos de los motores de tracción (ver figura 7.48), se ha realizado el proceso contrario al subsistema anterior. En este caso, se leerán los datos de los cuatro motores mediante el bloque *CAN Receive* (ver figura 7.49), configurando la ID de la cual se desea leer el mensaje, la longitud de este y el tipo de dato. Este proceso permite recibir información relevante sobre el estado de los motores, como temperatura, corriente y velocidad, para ser procesada y utilizada en el control del robot.

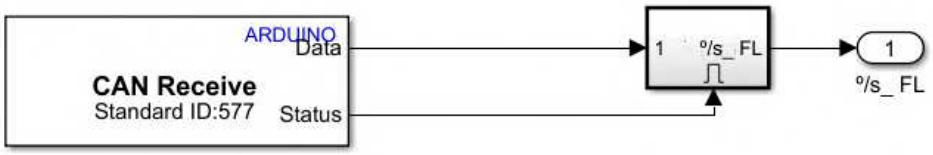


Figura 7.49: Bloque CAN Receive

La cuestión a resolver ahora es decodificar el mensaje CAN e interpretar correctamente cada una de las partes de este. Para ello, se ha creado el siguiente subsistema (ver figura 7.50), que se ha desarrollado en base a lo que el fabricante de los motores de tracción proporciona en la documentación. Este subsistema se encarga de desglosar el mensaje recibido y extraer la información relevante, como la temperatura, corriente y velocidad de los motores, para que pueda ser procesada y utilizada en el control del robot.

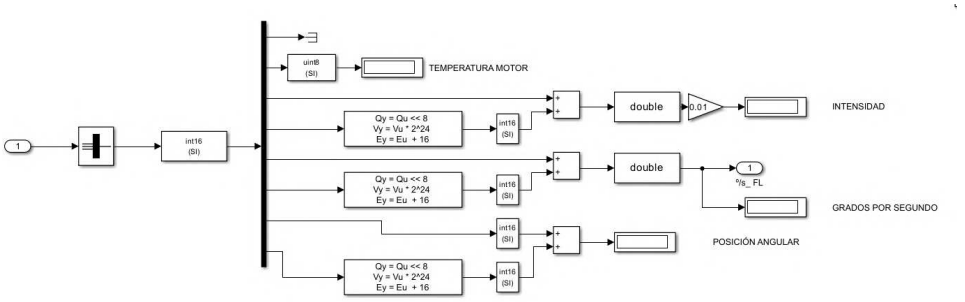


Figura 7.50: Subsistema encargado del procesamiento del mensaje CAN recibido

La primera parte es un convertidor de BUS a vector, para poder tratar cada parte de la trama por separado. Luego, sigue un convertidor de $\mathrm{uint8}$ a $\mathrm{int16}$. Con un demultiplexor de 1-8, extraeremos los 8 bytes correspondientes del mensaje para poder procesarlos adecuadamente y acceder a cada uno de los datos que conforman la información de los motores de tracción.

- 1 byte: ID del dispositivo.
- 2 byte: Temperatura del motor en uint8.
- 3 y 4 byte: Corriente instantánea del motor.
- 5 y 6 byte: Velocidad angular en grados/segundo del motor.
- 7 y 8 byte: Posición angular en grados del motor.

Una vez sabemos qué compone cada parte de la trama CAN, mediante una serie de operaciones de bits y sumas, podremos obtener los valores de las variables anteriores con una precisión de $\pm 0{,}01$.

Las IDs de cada motor son:

- El motor de la oruga F.L. tiene el ID: 577
- El motor de la oruga F.R. tiene el ID: 578
- El motor de la oruga R.R. tiene el ID: 579
- El motor de la oruga R.L. tiene el ID: 580

7.6.3 Procesamiento de errores

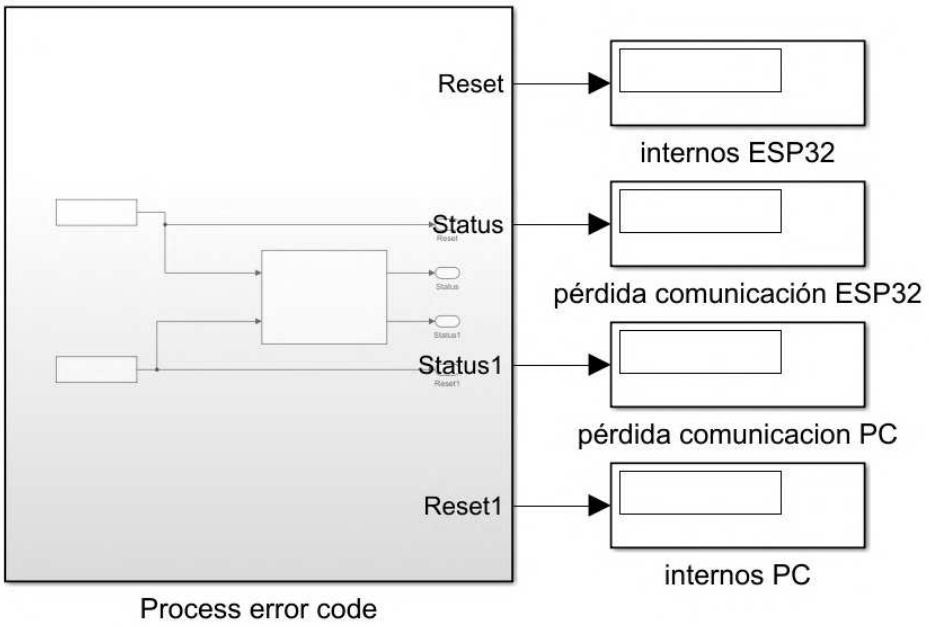


Figura 7.51: Subsistema encargado de mostrar códigos de errores

El subsistema mostrado en la figura 7.51 es responsable de mostrar los errores internos tanto del PC como del ESP32 en el Arduino MKR WiFi 1010. Estos errores se extraen de los bytes correspondientes del datagrama recibido de ambos dispositivos. A continuación, se presenta una lista con los tipos de errores codificados en los 2 bytes propuestos para este propósito:

- Si el código de error proviene del PC y su valor es '1', indica que se ha perdido la conexión con el mando de Xbox. En cambio, si el código es '2', significa que el mando sigue conectado (variable *Reset1*).
- En lo que respecta a los errores del ESP32, se ha utilizado una codificación de 4 bits. Si el problema está relacionado con la lectura de datos desde los encoders, el código de error es el siguiente (variable *Reset2*):
  - Si el fallo corresponde al encoder F.R., el código será 0001 (1 en decimal).
  - Si el fallo corresponde al encoder F.L., el código será 0010 (2 en decimal).
  - Si el fallo corresponde al encoder R.R., el código será 0100 (4 en decimal).
  - Si el fallo corresponde al encoder R.L., el código será 1000 (8 en decimal).
  - Para combinaciones de fallos en múltiples encoders, se utilizarán combinaciones de los valores anteriores (por ejemplo, 0011 para fallos en F.R. y F.L.).

El subsistema *Process error code* tiene en el interior lo siguiente (ver figura 7.52):

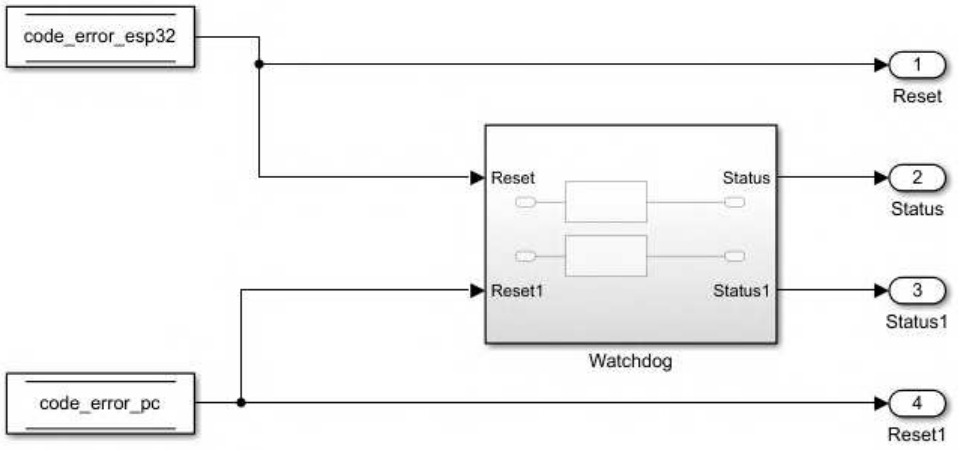


Figura 7.52: Subsistema para el control Watchdog en Simulink

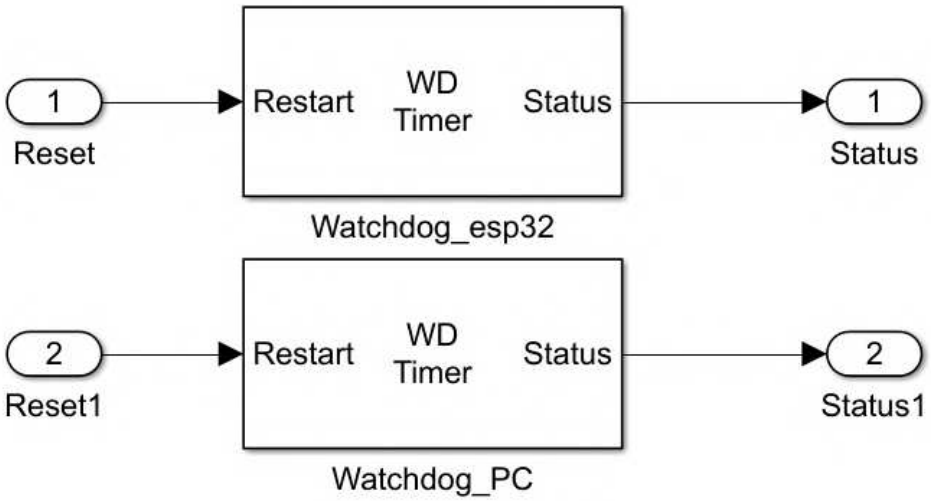


Figura 7.53: Bloques watchdog PC y ESP32

Para detectar la pérdida de conexión con el Arduino MKR WiFi 1010 o con el ESP32, se ha implementado en el interior del subsistema *Watchdog* el siguiente código utilizando un *software watchdog timer* de la librería *Motor Control Blockset* de Matlab/Simulink (ver figura 7.53). Este mecanismo permite generar una señal de aviso cuando no se han recibido datos, en este caso, de los vectores $\mathrm{code\_error\_esp32}$ y $\mathrm{code\_error\_pc}$. La señal de aviso se activará a nivel alto y será visible en los displays, lo que permitirá al usuario percatarse de la desconexión y tomar las acciones necesarias.

El comportamiento deseado es que, en caso de no recibir datos de alguno de los dos dispositivos, el programa se detenga inmediatamente y se proceda a detener los ocho motores en una posición segura. Esta funcionalidad se implementará en futuras ampliaciones de este proyecto.

Capítulo 8

Programación del control ESP32

8.1 Introducción

En el presente capítulo se describe el código implementado en el microcontrolador ESP32. Este microcontrolador actúa siguiendo las consignas enviadas por el Arduino MKR WiFi 1010 para la ejecución de movimientos en los motores de elevación.

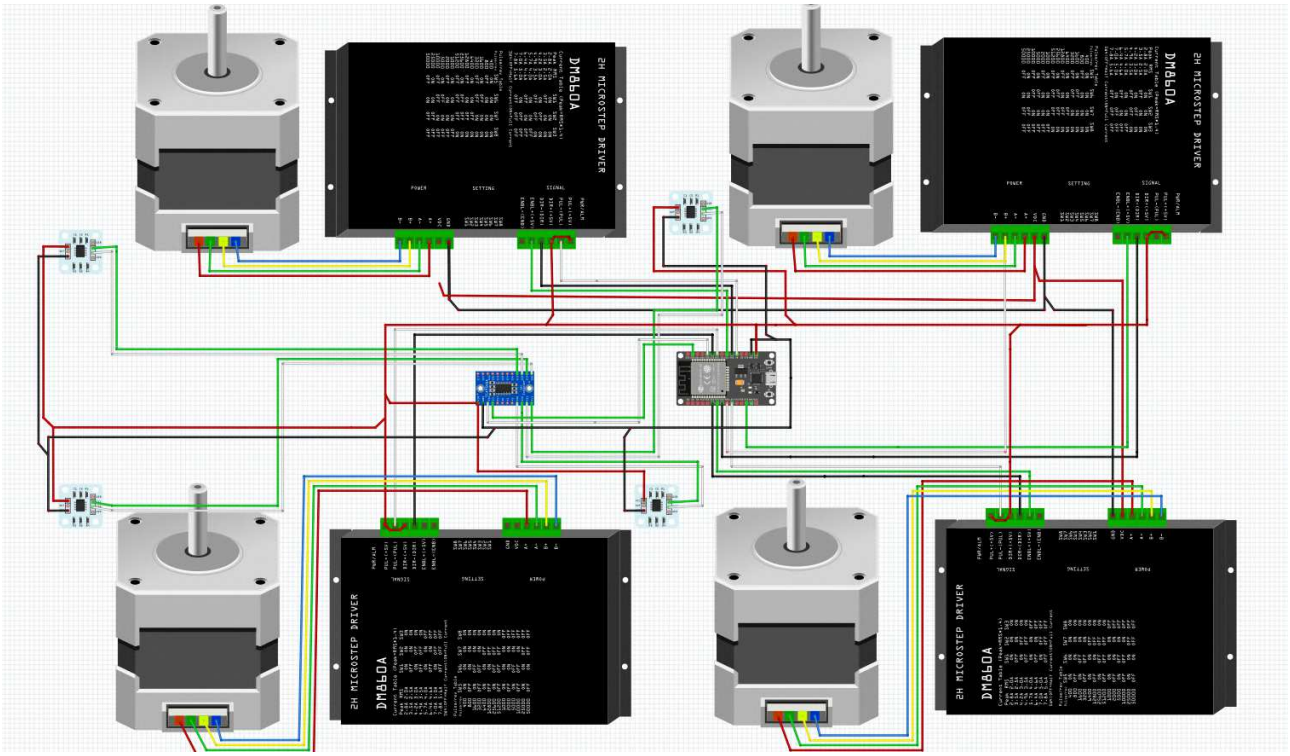


Figura 8.1: Diagrama del conexionado del ESP32

El microcontrolador ESP32 es el encargado de realizar la lectura de los ángulos proporcionados por los sensores angulares absolutos de efecto Hall (encoders modelo AS5600) cuyo diagrama de conexión aparece en la figura 8.1. El entorno de programación utilizado es Arduino IDE, como se verá en el desarrollo del capítulo.

Este capítulo no tiene como objetivo explicar el código línea por línea, ya que hacerlo extendería considerablemente el contenido. En su lugar, se centrará en comentar y argumentar los aspectos más relevantes de manera general.

El código está comentado siguiendo el enfoque estándar de la programación estructurada. Cada sección del código está precedida por un comentario (en inglés) que describe su propósito, como la configuración del WiFi o la comunicación mediante I2C.

8.2 Librerías usadas

Las tres librerías usadas en el desarrollo del código Arduino son las siguientes:

- ”WiFi.h”: La librería WiFi.h permite gestionar la conexión del ESP32 a redes WiFi. Ofrece funciones para conectarse a una red, desconectarse, obtener la dirección IP, comprobar el estado de la conexión y manejar eventos relacionados con la red inalámbrica.
- ”WiFiUdp.h”: La librería WiFiUdp.h permite enviar y recibir paquetes de datos utilizando el protocolo UDP a través de la conexión WiFi del ESP32. Proporciona funciones para iniciar la comunicación, enviar mensajes a direcciones específicas y recibir datos de otros dispositivos en la red. Es una herramienta útil para aplicaciones donde se requiere una comunicación rápida y ligera sin necesidad de establecer una conexión permanente, como en el caso de mensajes de control o telemetría. En el caso de este código, será el método de comunicación con el otro microcontrolador, con el PC principal o cualquier otro dispositivo que se establezca como cliente del ESP32 y esté escuchando la información en el puerto correspondiente.
- ”Wire.h”: La librería Wire.h permite establecer una comunicación mediante el protocolo I2C entre el ESP32 y otros dispositivos. Facilita el envío y la recepción de datos, la configuración como maestro o esclavo en el bus y la gestión de múltiples dispositivos conectados a las mismas líneas de datos y reloj. Es fundamental en aplicaciones donde se utilizan sensores, memorias u otros periféricos que requieren comunicación serie a dos hilos. En este caso, el maestro será el microcontrolador ESP32 y el esclavo será el multiplexor TCA9548A de ocho canales, al cual se conectan como entradas los cuatro sensores AS5600.

8.3 Variables globales, locales y funciones usadas

Código 8.1: Configuración inicial y variables globales del ESP32

```
1
2  #include <WiFi.h>
3  #include <WiFiUdp.h>
4  #include "Wire.h"
5  // ------------------------------------------------------------------------------
6  // Multiplexer Configuration
7  // ------------------------------------------------------------------------------
8  #define TCAADDR 0x70 // I2C address of TCA9548A
9
10 // ------------------------------------------------------------------------------
11 // WiFi Configuration
12 // ------------------------------------------------------------------------------
13 WiFiUDP udp; // UDP object
14 const char *ssid = "Horu";        // WiFi SSID (Robot network)
15 const char *password = "Horu27T-I"; //WiFi Horu password
16
17 IPAddress clientIP(192, 168, 10, 101); //  Arduino MKR WiFi 1010 WiFi IP address
18 const int localPort = 9090;             // ESP32 local UDP port for receiving
19 const int localPort_S = 8885;           // UDP port for sending ( Arduino MKR WiFi 1010)
20
21 // ------------------------------------------------------------------------------
22 // Global Variables
23 // ------------------------------------------------------------------------------
24
25 // Angle management
26 int16_t receivedValues[4] = {180, 180, 180, 180};  // Stores the 4 received target angles
27 float angle_sensor[4] = {0, 0, 0, 0};              // Measured angle for each encoder
28 float angle_target[4] = {180, 180, 180, 180};       // Target angle for each motor
29 int off_set[4] = {-2, -5, 16, -15};                 // Offset for each motor {F.R, F.L, ...
       R.R, R.L}
30
31 int mode; // Control mode
32
33 // Encoder reading
34 int lowbyte;         // Angle lower byte (8 bits)
35 word highbyte;       // Angle higher byte (8 bits)
36 int rawAngle;        // Complete angle value
37 const int resolucion_driver = 400; // Driver resolution
38 const int reducer = 80;            // Gear reduction
39
40 // Motor control {F.R, F.L, R.R, R.L}
41 const int pwmPin[4] = {27, 26, 18, 4};     // PWM pins
42 const int dirPin[4] = {32, 25, 19, 16};    // Direction pins
43 const int enaplusPin[4] = {33, 13, 5, 17}; // Enable pins
44
45 bool enable[4] = {LOW, LOW, LOW, LOW};     // Enable state for each motor
46 bool state[4] = {LOW, LOW, LOW, LOW};     // Motor active state (initially off)
47 int direction[4] = {LOW, LOW, LOW, LOW};  // Motor direction
48
49 hw_timer_t *timer = NULL; // Hardware timer
```

En esta primera parte del código (ver código 8.1), se configuran las librerías usadas, las variables globales y constantes requeridas por las comunicaciones usadas (I2C, WiFi), algunas variables que permiten guardar los ángulos tanto recibidos, actuales y objetivos.

Asimismo, se declaran las variables específicas de los encoders, que permiten la lectura y escritura de los ángulos de las orugas. Además, se definen vectores de tipo

bool, cuyos elementos representan el estado de diversas señales de control, activas o inactivas. Cada posición del vector almacena un valor true o false, lo que facilita la gestión de transiciones basadas en flancos, tales como los estados de los motores o las señales de habilitación (enable). Finalmente, se configuran vectores que contienen la numeración de los pines del ESP32 asignados para la conexión con los drivers de los motores paso a paso.

Cabe destacar la última línea del código anterior $hw\_timer\_t * timer = NULL;$ en la cual se declara un puntero de tipo hw_timer_t, que es un tipo de dato específico para trabajar con temporizadores de hardware en el ESP32. Este puntero se utilizará para gestionar un temporizador de hardware, que se usará en la interrupción de creación de la PWM encargada de los motores paso a paso.

Código 8.2: Rutina de interrupción y selección de canal del multiplexor

```
1  // ---------------------------------------------------------------------------
2  // Interrupt Service Routine (ISR) for PWM generation
3  // ---------------------------------------------------------------------------
4  void IRAM_ATTR PWM_generator_isr() {
5    for (int i = 0; i < 4; i++) {
6      if (state[i]) {
7        digitalWrite(pwmPin[i], !digitalRead(pwmPin[i])); // Toggle PWM output
8      }
9    }
10 }
11
12 // ---------------------------------------------------------------------------
13 // Multiplexer Channel Selection
14 // ---------------------------------------------------------------------------
15 void tcaselect(uint8_t i) {
16   if (i > 7) return;
17   Wire.beginTransmission(TCAADDR);
18   Wire.write(1 << i);
19   Wire.endTransmission();
20 }
```

El código anterior (ver código 8.2) contiene las siguientes funciones:

La función PWM_generator_isr() está definida como una Rutina de Servicio de Interrupción (ISR) y es responsable de la generación de señales PWM para controlar los dispositivos conectados a los pines indicados en pwmPin. Esta ISR se ejecuta automáticamente al producirse una interrupción, permitiendo la conmutación del estado lógico de los pines asociados a la señal PWM. En la implementación, se recorre el arreglo state[] y, cuando un estado es verdadero (indicando que se debe generar PWM), el valor lógico del pin correspondiente se invierte utilizando las funciones digitalWrite y digitalRead.

Por otro lado, la función tcaselect(uint8_t i) está diseñada para seleccionar uno de los canales del multiplexor TCA9548A mediante comunicación I2C. El parámetro i indica el canal a activar. La función verifica que el valor de i esté dentro del rango válido (0 a 7) y, en caso afirmativo, envía una señal al multiplexor para habilitar el canal correspondiente utilizando las funciones Wire.beginTransmission, Wire.write y Wire.endTransmission.

Código 8.3: Función de lectura de ángulo del sensor AS5600

```
// ---------------------------------------------------------------------------
// ReadAngle Function: Reads the angle from a specific sensor based on the sensor ID
// ---------------------------------------------------------------------------
bool ReadAngle(uint8_t sensorID) {
  // Select the correct I2C channel for the given sensor ID using the multiplexor
  tcaselect(sensorID);

  // Begin I2C communication with the sensor at address 0x36
  Wire.beginTransmission(0x36);

  // Write the register address (0x0D) to start reading the low byte of the angle
  Wire.write(0x0D);

  // End the transmission and check if there was an error
  if (Wire.endTransmission() != 0) {
    // If there's an error, print a message and return false
    Serial.print("Error: I2C sensor not detected on channel ");
    Serial.println(sensorID);
    return false;
  }

  // Request 1 byte of data from the sensor (low byte of the angle)
  Wire.requestFrom(0x36, 1);

  // If no data is available, return false
  if (Wire.available() == 0) return false;

  // Read the low byte of the angle data
  lowbyte = Wire.read();

  // Begin I2C communication again to read the high byte of the angle
  Wire.beginTransmission(0x36);
  Wire.write(0x0C);  // Write the register address (0x0C) to get the high byte
  if (Wire.endTransmission() != 0) return false;

  // Request 1 byte of data from the sensor (high byte of the angle)
  Wire.requestFrom(0x36, 1);

  // If no data is available, return false
  if (Wire.available() == 0) return false;

  // Read the high byte of the angle data
  highbyte = Wire.read();

  // Shift the high byte left by 8 bits to combine it with the low byte
  highbyte = highbyte << 8;

  // Combine the high byte and low byte to form the raw angle value
  rawAngle = highbyte | lowbyte;

  // Calculate the actual angle using the raw value and apply an offset
  angle_sensor[sensorID] = rawAngle * 0.087890625 + off_set[sensorID];

  // Return true if the angle was successfully read and processed
  return true;
}
```

La función ReadAngle que se encarga de leer el valor del ángulo de un sensor específico , basado en su sensorID se encuentra en el código 8.3. Utiliza el protocolo I2C para la comunicación con el sensor. La función primero selecciona el canal I2C correspondiente al sensor, luego lee los bytes bajo y alto del ángulo. Estos bytes se combinan para formar el valor bruto del ángulo, el cual se procesa aplicando un factor de escala ($0{,}087890625$). Este factor de escala proviene de la división de los 360 grados totales entre los 2^{12} bits de resolución que presenta el encoder magnético, tras esto se aplica

un desplazamiento (off_set[sensorID]). Si la comunicación con el sensor es exitosa, la función devuelve true, de lo contrario, devuelve false.

Esto último servirá para implementar un mecanismo de detección y propagación de fallos hacia el Arduino MKR WiFi 1010 y el PC.

Código 8.4: Control incremental de posición

```
// ------------------------------------------------------------------------------
// Incremental Control for Motor Direction and State
// ------------------------------------------------------------------------------
void incrementalControl(int16_t control[4], int i) {
  const int clockwise[4] = {LOW, HIGH, HIGH, LOW};
  const int counterclockwise[4] = {HIGH, LOW, LOW, HIGH};

  if (control[i] == 1) {
    direction[i] = clockwise[i];
    state[i] = true;
    enable[i] = false;
    digitalWrite(dirPin[i], direction[i]);
  } else if (control[i] == -1) {
    direction[i] = counterclockwise[i];
    state[i] = true;
    enable[i] = false;
    digitalWrite(dirPin[i], direction[i]);
  } else {
    state[i] = false;
    enable[i] = true;
  }

  digitalWrite(enaplusPin[i], enable[i]);
}
```

La función incrementalControl (código 8.4) permite gestionar de forma incremental la dirección de giro y el estado de un motor. Dependiendo del valor del vector control[i], se establece el sentido de giro del motor (horario o antihorario) o se desactiva su funcionamiento. Si el control es igual a 1, se configura para girar en sentido horario; si es -1, en sentido antihorario; y si es 0, se desactiva el motor. Además, se actualizan las salidas digitales correspondientes para reflejar estos estados.

Código 8.5: Control de posición absoluta

```
// ------------------------------------------------------------------------------
// Position Control for Motor based on Target vs Current Angle
// ------------------------------------------------------------------------------
void positionControl(float current, float target, int i) {
  const int clockwise[4] = {LOW, HIGH, HIGH, LOW};
  const int counterclockwise[4] = {HIGH, LOW, LOW, HIGH};

  direction[i] = (current < target) ? clockwise[i] : counterclockwise[i];
  digitalWrite(dirPin[i], direction[i]);

  state[i] = true;
  enable[i] = false;
  digitalWrite(enaplusPin[i], enable[i]);
}
```

La función positionControl (ver código 8.5) realiza el control de la posición del motor comparando el ángulo actual con el ángulo objetivo. Según si el ángulo actual es menor o mayor que el objetivo, se establece la dirección de giro del motor (horaria o

antihoraria). Además, activa el estado de funcionamiento del motor deshabilitando su señal de habilitación correspondiente.

Código 8.6: Función para gestionar la conexión WiFi

```
// ---------------------------------------------------------------------------
// Check and Reconnect to WiFi if Disconnected
// ---------------------------------------------------------------------------
void checkWiFiConnection() {
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("WiFi connection lost. Attempting to reconnect...");
    WiFi.disconnect();
    WiFi.begin(ssid, password);

    int attempts = 0;
    while (WiFi.status() != WL_CONNECTED && attempts < 10) {
      delay(1000);
      Serial.print(".");
      attempts++;
    }

    if (WiFi.status() == WL_CONNECTED) {
      Serial.println("\nWiFi reconnected successfully.");
    } else {
      Serial.println("\nFailed to reconnect to WiFi.");
    }
  }
}
```

La función checkWiFiConnection (ver código 8.6) comprueba si el ESP32 mantiene la conexión a la red WiFi. En caso de detectar una desconexión, realiza hasta diez intentos sucesivos de reconexión. Si la conexión se restablece con éxito, emite un mensaje de confirmación por el puerto serie; en caso contrario, informa del fallo de reconexión.

En el Arduino MKR WiFi, la detección de esta situación se implementará mediante un mecanismo de tipo *Watchdog*, como se describió en el capítulo anterior: si no se reciben mensajes del ESP32 durante un intervalo predefinido, se ejecutarán las acciones de recuperación correspondientes.

Código 8.7: Función para gestionar la conexión I2C

```
// ---------------------------------------------------------------------------
// Scan and Display I2C Devices on the Bus
// ---------------------------------------------------------------------------
void scanI2CDevices() {
  Serial.println("Scanning for I2C devices...");
  for (uint8_t address = 1; address < 127; address++) {
    Wire.beginTransmission(address);
    if (Wire.endTransmission() == 0) {
      Serial.print("Device found at address 0x");
      Serial.println(address, HEX);
    }
  }
  Serial.println("I2C scan completed.");
}
```

La función scanI2CDevices (código 8.7) efectúa un escaneo del bus I2C en busca de dispositivos conectados. Para ello, recorre todas las direcciones válidas (de 1 a 126)

e informa, mediante salida por el puerto serie, de aquellas que responden. En la configuración actual únicamente se detectará la dirección del multiplexor; no obstante, este procedimiento permite verificar de forma directa la comunicación maestro-esclavo ante la incorporación de nuevos dispositivos.

Código 8.8: Función SetUp

```
// ------------------------------------------------------------------------------
// Setup Function
// ------------------------------------------------------------------------------
void setup() {
  // Start serial communication
  Serial.begin(9600);
  Wire.begin();
  scanI2CDevices();

  // Motor pins setup
  for (int i = 0; i < 4; i++) {
    pinMode(dirPin[i], OUTPUT);
    pinMode(pwmPin[i], OUTPUT);
    pinMode(enaplusPin[i], OUTPUT);

    digitalWrite(dirPin[i], LOW);
    digitalWrite(enaplusPin[i], LOW);
  }

  // WiFi Connection
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nWiFi connection established");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());

  // UDP Initialization
  udp.begin(localPort);
  Serial.print("UDP port started: ");
  Serial.println(localPort);

  // Timer setup for PWM generation
  timer = timerBegin(1000000);          // Timer 0, prescaler 80 -> 1 MHz
  timerAttachInterrupt(timer, &PWM_generator_isr);  // Timer ISR
  timerAlarm(timer, 1200, true,0);      // Trigger interrupt every 1200 µs
  timerStart(timer);                    // Start timer
  delay(1000);
}
```

Ahora se presenta la función setup del código (ver código 8.8), donde se establecen los puertos de comunicación utilizando algunas funciones declaradas previamente y se configuran los pines de salida necesarios para el control del sistema, así como la configuración e inicialización del timer.

La señal PWM (Pulse Width Modulation) se genera manualmente mediante el uso de un temporizador de hardware de frecuencia 80 MHz. Se configura un timer para lanzar interrupciones periódicas cada 1200 microsegundos, momento en el cual se ejecuta una rutina de interrupción (PWM_generator_ISR). Dentro de dicha ISR, se verifica el estado de cada motor y, en caso de estar habilitado, se alterna el valor lógico de la salida digital asociada, generando así una señal cuadrada de frecuencia

fija como se ha explicado anteriormente. De este modo, se consigue una modulación por ancho de pulso sin hacer uso de funciones estándar como $\mathrm{analogWrite}()$, la cual no es compatible con el ESP32. La frecuencia y el ciclo de trabajo de la señal PWM dependen del intervalo de interrupción configurado y del patrón de activación de los pines.

La frecuencia de la señal PWM generada puede calcularse mediante la expresión:

$$f_{\text{PWM}} = \frac{1}{T_{\text{interrupción}}} \tag{8.1}$$

donde $T_{\text{interrupción}}$ representa el periodo de la interrupción configurada. En este caso, como el temporizador lanza una interrupción cada 1200 μs, la frecuencia resultante es:

$$f_{\text{PWM}} = \frac{1}{1200 \times 10^{-6}} \approx 833\,\text{Hz} \tag{8.2}$$

De este modo, cada salida PWM alterna su estado a una frecuencia de aproximadamente 833 Hz.

Que está dentro de los límites establecidos por el fabricante del driver (ver figura 8.2):

Parameters	DM542			
	Min	Typical	Max	Unit
Output Current	1.0	-	4.2 (3.0 RMS)	A
Input Voltage	+20	+36	+50	VDC
Logic Signal Current	7	10	16	mA
Pulse input frequency	0	-	200	kHz
Pulse Width	2.5	-	-	uS
Pulse Voltage	5	-	24	VDC
Isolation resistance	500			MΩ

Figura 8.2: Tabla de parámetros del driver DM542 de los motores paso a paso

Esto consigue que el motor gire a una velocidad constante de aproximadamente 5 grados/s.

A continuación se presenta el desarrollo del bucle principal del microcontrolador (loop), que implementa de forma cíclica la lógica de control del sistema. En esta sección se incluyen:

- Gestión de comunicaciones UDP.
- Lectura de sensores (AS5600).
- Ejecución de funciones de control.
- Envío de información relevante hacia el Arduino WiFi 1010.

De este modo se garantiza el correcto funcionamiento en tiempo real de la aplicación embebida, con un periodo de muestreo fijo de 10 ms.

Código 8.9: Función loop

```
1  void loop() {
2
3    // Mark the start of the cycle with the millis() function
4    unsigned long startTime = millis();
5
6    uint8_t error_code = 0;  // Error code for the angle sensors
7    bool sendMsg = false;    // Flag to determine if a message should be sent via UDP
8
9    // Check the WiFi connection
10   checkWiFiConnection();
11
12   // Read the UDP packet of 10 bytes (5 int16_t values)
13   int16_t tempValues[5];  // Array to store UDP packet values (declared outside the if to ...
         be used later)
14
15   int packetSize = udp.parsePacket();  // Check if 'theres an available UDP packet
16   if (packetSize == 10) {  // If the packet size is 10 bytes
17     char incomingPacket[10];  // Array to store the incoming packet
18     int len = udp.read(incomingPacket, 10);  // Read the 10-byte packet
19     if (len == 10) {  // If the full 10 bytes are received successfully
20       memcpy(tempValues, incomingPacket, 10);  // Copy the packet values into tempValues array
21
22       // Extract the mode (last value in the packet)
23       mode = tempValues[4];  // The mode is in the last value
24       Serial.print("Operating mode: ");
25       Serial.println(mode);
26     }
27   }
28 }
```

En esta sección del (loop) (ver código 8.9) se realiza la inicialización de las siguientes variables locales:

- unsigned long startTime: almacena el tiempo actual, en milisegundos, devuelto por millis(), garantizando la periodicidad del ciclo de control.
- bool sendMsg: bandera lógica (flanco lógico) que determina si se debe enviar un paquete UDP al Arduino MKR WiFi 1010.
- int16_t error_code: variable que almacena el código de error, representado en bits, correspondiente a fallos de lectura de los encoders.
- int16_t temp_values[5]: array que guarda los cinco valores (5 × 2 bytes = 10 bytes) recibidos vía WiFi desde el Arduino MKR WiFi 1010.

A continuación, se verifica la conexión WiFi y, a continuación, se recibe y procesa un paquete UDP de 10 bytes, interpretado como cinco valores de tipo int16_t. Del contenido del paquete se extrae el modo de funcionamiento, el cual condicionará el comportamiento posterior del sistema.

Los cuatro primeros elementos de este vector de datos corresponden a los valores objetivo, ya sea para el control de posición mediante la variable angle_target o para el control incremental mediante los sentidos de giro de los motores paso a paso.

siguiendo con la ejecución del código, se encuentra lo siguiente:

Código 8.10: Control de modos en loop

```
// Position control mode (mode 2)
if (mode == 2) {
  // Assign the first 4 values of the packet to the target angles
  for (int i = 0; i < 4; i++) {
    angle_target[i] = float(tempValues[i]);  // Assign the received values as target angles
  }

  // Compare the target angles with the current sensor angles
  for (int i = 0; i < 4; i++) {
    if (abs(angle_target[i] - angle_sensor[i]) > 1) {  // If the difference is greater ... than 1 degree
      positionControl(angle_sensor[i], angle_target[i], i);  // Control the 'actuators ... position
      sendMsg = true;  // Set the flag to send a message
    } else {
      state[i] = false;  // If the angle is close enough, disable the state
      enable[i] = true;   // Enable the actuator
      digitalWrite(enaplusPin[i], enable[i]);  // Send the signal to activate the actuator
    }
  }
}

// Incremental control mode (mode 3)
else if (mode == 3 || mode==4 || mode == 5) {
  // Assign the UDP packet values to the received commands (-1, 0, 1 for incremental ... control, 1 = clockwise, 0 stop, -1 counterclockwise)
  for (int i = 0; i < 4; i++) {
    receivedValues[i] = tempValues[i];  // Store the received commands for incremental ... control
    incrementalControl(receivedValues, i);  // Perform the incremental control based on ... the commands
  }
}
```

En este fragmento de código se gestiona el control de los actuadores (motores de elevación) en función del modo de operación recibido desde el Arduino MKR WiFi 1010 (ver código 8.10).

- **Modo 2: Control de posición.** Inicialmente se almacenan los cuatro valores correspondientes a los ángulos objetivo en un array de tipo float de cuatro posiciones. Posteriormente, mediante un bucle, se comparan los ángulos objetivo con los ángulos actuales medidos por los encoders. Si la diferencia entre ambos supera un umbral de 1 grado, se llama a la función positionControl, la cual ajusta la posición de la oruga activando el motor en la dirección correspondiente hasta que se alcance la posición deseada.

  Una vez que la diferencia entre el ángulo actual y el objetivo es inferior a 1 grado, se procede a desactivar el driver del motor paso a paso mediante la desactivación de su señal enable, con el fin de reducir el consumo energético del sistema.

- **Modo 3, 4 ó 5: Control incremental.** En este modo, los cuatro primeros valores recibidos a través del paquete UDP se copian en el vector receivedValues. A continuación, se llama a la función incrementalControl para ejecutar el modo de control incremental de los motores, según el valor de consigna recibida (giro horario, antihorario o parada), como se ha descrito anteriormente.

Código 8.11: Función para leer errores y enviar mensajes UDO

```
1   // Reset the error code
2   error_code = 0;
3
4   // Loop through each sensor and read the angle values
5   for (int i = 0; i < 4; i++) {
6     bool encoder_state = ReadAngle(i);  // Read the angle of each sensor
7     if (!encoder_state) {
8       error_code |= (1 << i);  // Mark the bit for the sensor that failed
9       sendMsg = true;          // If 'theres an error, set the flag to send the message
10    }
11  }
12
13  // If the flag to send a message is set (due to error or angle change), send the UDP message
14  if (sendMsg) {
15    // Prepare the UDP message with the data
16    int16_t buffer[9];
17    buffer[0] = 2;  // Message type 2 = comes from ESP32
18
19    // Fill the buffer with the sensor angles, multiplying by 100 for more precision
20    for (int i = 1; i <= 4; i++) {
21      buffer[i] = int16_t(angle_sensor[i - 1] * 100);  // Multiply the angle by 100 for ...
          more precision
22    }
23
24    buffer[5] = 0;
25    buffer[6] = 0;
26    buffer[7] = 0;
27    buffer[8] = error_code;  // Include the error code in the message
28
29    // Send the UDP message
30    udp.beginPacket(clientIP, localPort_S);  // Start the UDP packet
31    udp.write((uint8_t *)&buffer, sizeof(buffer));  // Write the buffer data to the packet
32    udp.endPacket();  // End the UDP packet and send it
33  }
34
35  // Wait until the 10ms cycle is completed
36  unsigned long elapsedTime = millis() - startTime;
37  if (elapsedTime < 10) {
38    delay(10 - elapsedTime);  // Delay to make the cycle last 10ms
39  }
```

Finalmente, en el bucle loop se implementa la siguiente secuencia de operaciones (ver código 8.11):

1. Verificación de sensores: Para cada ciclo se comprueba la respuesta de los sensores. Si alguno no responde correctamente, se actualiza el *error code* correspondiente y se activa la bandera sendMsg para señalizar la incidencia.
2. Cálculo de compensación angular: Cuando los motores están en movimiento, también se activa la bandera sendMsg, ya que es necesario enviar el ángulo actual al Arduino MKR WiFi 1010 para calcular la velocidad de compensación de los motores de tracción.
3. Preparación y envío de paquete UDP: Si la bandera sendMsg está activa, se construye un buffer de 18 bytes con la siguiente estructura:
   - Bytes 0–1: identificador de ESP32 (valor “2”).
   - Bytes 2–9: los cuatro ángulos de los sensores, cada uno codificado en 2 bytes (multiplos de 100 para mejorar la resolución).
   - Bytes 10–11: *error code*.

- Bytes 12–17: seis bytes reservados para posibles ampliaciones (sensores adicionales).

A continuación, el buffer se envía al receptor (Arduino MKR WiFi 1010).

4. Garantía del periodo de muestreo: Para mantener una frecuencia de 100 Hz (periodo mínimo de 10 ms), al final de cada ciclo se calcula el tiempo transcurrido desde el inicio (mediante $\mathrm{millis}()$) y, si no se han completado los 10 ms, se invoca $\mathrm{delay}()$ con el tiempo restante. De este modo, cada iteración del bucle dura exactamente 10 ms.

Capítulo 9

Conclusiones

9.1 Introducción

En este capítulo se expondrán las conclusiones derivadas del desarrollo del presente proyecto, evaluando el grado de cumplimiento de los objetivos planteados inicialmente, así como una valoración global del trabajo realizado en su conjunto. También se incluirá un apartado dedicado a posibles líneas de mejora o futuras extensiones del proyecto. Finalmente, se aportará una reflexión personal sobre la experiencia adquirida y el impacto del proyecto a nivel formativo.

9.2 Conclusiones

A continuación se presentan los objetivos alcanzados en el marco del proyecto, especificando el procedimiento seguido para su consecución y la conclusión derivada de cada uno de ellos:

- Construcción de una batería hecha a medida encastrada en el chasis del vehículo para su blindaje, cuyas especificaciones son 12s3p de LiFePO4 18Ah. Este constituyó el objetivo inicial necesario para habilitar el avance del resto del proyecto. A lo largo de varias semanas, se trabajó conjuntamente con miembros del equipo RoboRescue UMA en el ensamblaje de celdas, la soldadura por punto de estas, terminales de conexión y la configuración del BMS. Este hito permitió consolidar conocimientos sobre tipologías de celdas, técnicas avanzadas de ensamblaje de baterías y criterios de evaluación para verificar el cumplimiento de especificaciones en sistemas personalizados, garantizando que la batería satisficiera los requisitos establecidos.
- Diseño del esquema eléctrico y de control para el conexionado de los componentes. Dado el gran número de componentes electrónicos y su cableado (al menos cuatro

cables por más de veinte dispositivos) fue determinante para facilitar su implantación en el robot. Esta fase resultó ser la de menor complejidad, en gran parte gracias a los conocimientos previamente adquiridos en el área de Tecnología Electrónica, técnicas de soldadura y aislamiento de señales.

- Adquisición de datos de sensores angulares absolutos de efecto Hall AS5600 y de unidad de movimiento inercial MPU9250.

  La adquisición de datos procedentes de los sensores angulares absolutos de efecto Hall AS5600 resultó especialmente sencilla en el entorno Arduino. Estos valores fueron fundamentales para la implementación de los controles de posición incrementales, ya que permitieron medir con precisión el ángulo de cada elemento móvil. Durante mi formación en las asignaturas de Informática Industrial e Instrumentación Electrónica, ya había estudiado en detalle el funcionamiento de encoders absolutos, lo que facilitó tanto la interpretación de las lecturas como la configuración óptima de las frecuencias de muestreo. De este modo, se garantizó la fiabilidad y la estabilidad del sistema de control posicional.

- Control de tracción con Arduino MKR WiFi 1010 y Simulink. La selección del microcontrolador principal se debió a su compatibilidad con una Shield CAN, que proporciona una comunicación CAN con los motores de tracción de forma robusta y eficiente, pero sobretodo un conexionado integrado con este Arduino. El control de los motores de tracción implementado se ha mostrado válido para la tracción del vehículo y fundamental para el apoyo en el movimiento de elevación.

- En el control del movimiento de elevación con ESP32, el primer paso consistió en seleccionar un microcontrolador secundario que cumpliera con los requisitos requeridos para el control de los drivers DM542 de los motores de elevación. El elemento decisivo fue la generación de PWM a frecuencias específicas; descartando microcontroladores como el Arduino Nano BLE y el propio Arduino MKR WiFi 1010, y la programación en Simulink. El control de posición realizado ha resultado satisfactorio pese a la limitación efectiva de potencia de los DM542, siendo incapaz de aportar 2/3 de la potencia de sus especificaciones.

- El segundo paso consistió en implementar el control posicional de elevación de las orugas. Gracias a la lectura de los encoders, cada oruga alcanzó la posición deseada de forma individual; sin embargo, esta estrategia resultó insuficiente para soportar el peso completo del vehículo. A continuación, se incorporó la fuerza de los motores de tracción al sistema de elevación. La coordinación simultánea de ambos motores implicó un notable esfuerzo de programación: se abordaron múltiples casos de funcionamiento, se añadieron restricciones de seguridad y se calcularon con precisión las velocidades y los tiempos de ejecución para garantizar la sincronización de los movimientos.

- Se ha logrado un nivelado automático del robot. La adquisición de datos procedentes de la IMU resultó especialmente sencilla gracias a la integración con Simulink. En un primer momento, el modo de nivelación automática presentó inestabilidades derivadas de lecturas con mucho ruido; no obstante, al adaptar y optimizar el algoritmo de filtrado

desarrollado en un Trabajo de Fin de Grado anterior, se consiguió implementar un sistema de filtrado eficaz que garantiza una nivelación robusta y fiable.

- Creación de protocolo de comunicación entre dispositivos (PC, ESP32, Arduino MKR WiFi 1010) con identificación de fallos. Debido al elevado volumen de datos intercambiados entre los distintos dispositivos, se ha diseñado un protocolo de comunicación que permite el envío y recepción de mensajes de forma estructurada, escalable y eficiente; preparado para futuras modificaciones e integraciones en Horu.

9.3 Impacto personal del TFG

Este proyecto ha supuesto la puesta en práctica de conceptos fundamentales de mi formación en Electrónica, Control y Comunicaciones. A lo largo del desarrollo, he integrado teoría y práctica para diseñar sistemas de medida, controlar actuadores y establecer protocolos de comunicación robustos, consolidando así los conocimientos adquiridos en la universidad.

La implementación de los algoritmos de control en Simulink y Arduino me ha permitido profundizar en la programación de sistemas robóticos, mientras que el manejo de Fritzing y otras herramientas de diseño electrónico ha facilitado el cableado y la documentación de esquemas. Asimismo, la experiencia de montaje de baterías a medida fortaleció mis competencias en ensamblaje, y configuraciones de BMS.

Colaborar con el equipo de RoboRescue UMA ha sido clave para aprender a coordinar tareas multidisciplinares. Desde la planificación conjunta hasta la resolución de incidencias en el laboratorio, he perfeccionado mi capacidad para comunicar requerimientos, repartir responsabilidades y sincronizar esfuerzos con mis compañeros.

Desde mi incorporación en septiembre de 2024, he sido testigo de la evolución de Horu, desde cuatro orugas sueltas hasta un robot plenamente funcional. Ver el avance tangible del proyecto día a día ha sido enormemente gratificante y ha reforzado mi motivación por la robótica de rescate.

Este trabajo no solo consolida mi formación, sino que también proyecta mi trayectoria profesional hacia el ámbito de la robótica. Confío en continuar contribuyendo a proyectos similares, donde mi pasión por la ingeniería y la innovación encuentre nuevos retos y oportunidades de mejora.

9.4 Ampliaciones futuras

En futuras ampliaciones del proyecto se prevé la sustitución de componentes como los drivers de los motores paso a paso, dado que presentan limitaciones en términos de entrega de par, así como la actualización de los propios motores paso a paso por servomotores, lo que permitirá mejorar la precisión y la capacidad de respuesta del sistema.

Asimismo, se contempla la integración de ambos microcontroladores en un entorno de programación unificado, como ROS (Robot Operating System), lo que facilitaría la coordinación y sincronización de las distintas tareas del robot.

Adicionalmente, está previsto implementar sensores LiDAR y unidades IMU de mayor precisión, así como cámaras 3D que posibiliten la navegación autónoma del robot en entornos no estructurados y de elevada complejidad.

Por último, se dará mayor importancia al mecanismo de detección de errores mediante la implementación de un protocolo más robusto, que permita al usuario identificar gráficamente el origen del error. Para ello, se propone programar un Sunton ESP32 dotado de pantalla TFT (Thin Film Transistor) táctil capacitiva que facilite la localización de fallos de manera visual. Ya que, siendo críticos, esta parte del trabajo es la que se ha quedado algo más básica, y sería en la que reforzaría si continuase con la programación de este robot.

Bibliografía

[1] Y. Morimoto, T. Tomiyama, and R. Michikawa, “RoboCup Rescue 2022 Team Description Paper SHINOBI”, Accessed: May. 09, 2025. [Online]. Available: https://tdp.robocup.org/wp-content/uploads/tdp/robocup/2022/robocuprescue-robot/shinobi-356/robocup-2022-robocuprescue-robot-shinobiAw37ofmjWF.pdf

[2] “Capra Robotics | Outdoor Mobile Robotics | Multiple Applications.” Accessed: May. 01, 2025. [Online]. Available: https://capra.ooo/

[3] “GitHub- MatVo1992/Simulink-XInput-Controller: Simulink Block / S-Function for Windows XInput API (XBOX Controller).” Accessed: May. 09, 2025. [Online]. Available:https://github.com/MatVo1992/Simulink-XInput-Controller

[4] “User’s Manual For DM542 Fully Digital Stepper Drive”. Feb. 25, 2025. [Online]. Available: https://kitaez-cnc.com/f/dm542.pdf

[5] “Arduino MKR WiFi 1010 — Arduino Official Store.” Accessed: Apr 14, 2025. [Online]. Available: https://store.arduino.cc/products/arduino-mkr-wifi-1010

[6] “Controlling MY ACTUATOR motor with CAN-BUS Shield- Using Arduino / Motors, Mechanics, Power and CNC- Arduino Forum.” Accessed: Apr. 14, 2025. [Online]. Available: https://forum.arduino.cc/t/controlling-my-actuator-motor-with-can-bus-shield/1042204

[7] “Wire.h,” Arduino Documentation, Apr. 14, 2025. [Online]. Available:

https://docs.arduino.cc/language-reference/en/functions/communication/wire/

[8] “WiFiUdp.h,” GitHub - espressif/arduino-esp32, Apr. 29, 2025. [Online]. Available: https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/src/WiFiUdp.h

[9] “Magnetic Rotary Position Sensor AS5600 Datasheet,” Apr. 02, 2025. [Online]. Available: https://files.seeedstudio.com/wiki/Grove-12-bit-Magnetic-Rotary-Position-Sensor-AS5600/res/Magnetic

[10] “ESP32 Overview,” Espressif Systems, Mar. 08, 2025. [Online]. Available: https://www.espressif.com/en/products/socs/esp32

[11] “Arduino MCP2515 CAN Bus Interface Tutorial- CAN Protocol.” Accessed: Mar. 12, 2025. [Online]. Available: https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/

[12] L. Suzhou Micro Actuator TechnologyCo., “Servo Motor Control Protocol V4.01.” Accessed: Mar. 20, 2025. [Online]. Available:https://www.myactuator.com/_files/ugd/cab28a_4396d7519700437c979530e3acdb4bd7.pdf

[13] L. Llamas, “Usar Arduino con los IMU de 9DOF MPU-9150 y MPU-9250,” Apr. 14, 2025. [Online]. Available: https://www.luisllamas.es/usar-arduino-con-los-imu-de-9dof-mpu-9150-y-mpu-9250/

[14] D. Pablo Arroyo Suárez, “Control de motores de un vehículo robótico de orugas basculantes” 2024

[15] D. Jesús Muñoz Martínez , “CONTROL AUTO-BALANCEADO TIPO PÉNDULO INVERTIDO PARA EL ROBOT MÓVIL PIERO ” 2016

[16] ”HDCF32700-6000mAh-3.2V LiFePO4 Battery Datasheet,”Haidi Energy Technology Co., Ltd. [Online]. Available: https://docs.tuyap.online/FDOCS/39736.pdf. [Accessed: 01-May-2025].

[17] ”Smart BMS 12S,”Daly BMS, [Online]. Available: https://es.dalybms.com/smart-bms-12s/. [Accessed: 01-May-2025].

[18] .ªbout Simulink States,”MathWorks, [Online]. Available: https://es.mathworks.com/help/stateflo simulink-states.html. [Accessed: 01-May-2025].

Apéndice A

Modelo del control PC versión simplificada

En este capítulo, se muestra un modelo de control del PC algo más sencillo que el desarrollado en la memoria. Este se basa de nuevo en una máquina de estados usando un *Subsystem Chart*, pero solamente con tres estados. El subsistema es el siguiente (ver figura A.1):

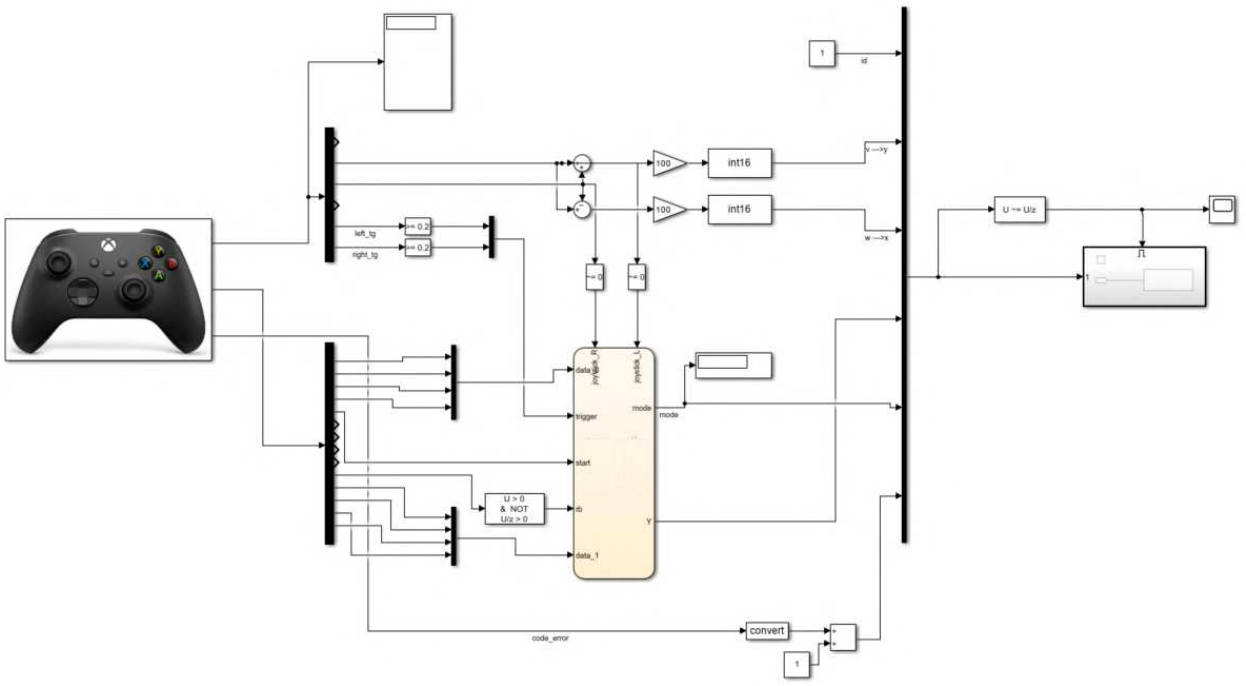


Figura A.1: Subsistemas del control PC con tres estados

Las entradas del subsistema son las mismas que las del modelo del capítulo anterior. La máquina de estados es la de la siguiente figura (ver figura A.2):

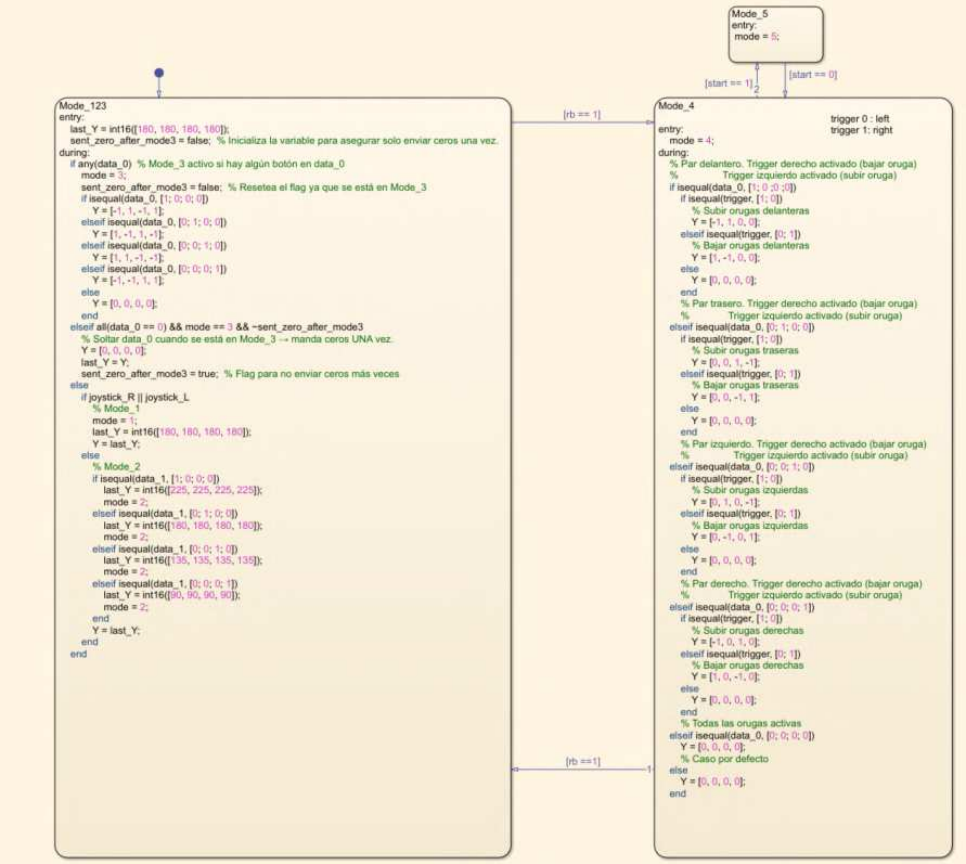


Figura A.2: Diagrama de estados del control PC simplificado

El primer estado tiene el siguiente código donde se unifican los modos de funcionamiento 1, 2, 3 del robot (ver código A.1):

Código A.1: Control del modo 1, 2 y 3 para el robot Horu

```
1  Mode_123
2  entry:
3      last_Y = int16([180, 180, 180, 180]);
4      sent_zero_after_mode3 = false;  % Inicializa la variable para asegurar solo enviar ceros una ...
           vez.
5  during:
6      if any(data_0)  % Mode_3 activo si hay algún botón en data_0
7          mode = 3;
8          sent_zero_after_mode3 = false;  % Resetea el flag ya que se está en Mode_3
9          if isequal(data_0, [1; 0; 0; 0])
10             Y = [-1, 1, -1, 1];
11         elseif isequal(data_0, [0; 1; 0; 0])
12             Y = [1, -1, 1, -1];
13         elseif isequal(data_0, [0; 0; 1; 0])
14             Y = [1, 1, -1, -1];
15         elseif isequal(data_0, [0; 0; 0; 1])
16             Y = [-1, -1, 1, 1];
17         else
18             Y = [0, 0, 0, 0];
19         end
20     elseif all(data_0 == 0) && mode == 3 && ¬sent_zero_after_mode3
21         % Soltar data_0 cuando se está en Mode_3 → manda ceros UNA vez.
22         Y = [0, 0, 0, 0];
```

```
        last_Y = Y;
        sent_zero_after_mode3 = true;  % Flag para no enviar ceros más veces
    else
        if joystick_R || joystick_L
            % Mode_1
            mode = 1;
            last_Y = int16([180, 180, 180, 180]);
            Y = last_Y;
        else
            % Mode_2
            if isequal(data_1, [1; 0; 0; 0])
                last_Y = int16([225, 225, 225, 225]);
                mode = 2;
            elseif isequal(data_1, [0; 1; 0; 0])
                last_Y = int16([180, 180, 180, 180]);
                mode = 2;
            elseif isequal(data_1, [0; 0; 1; 0])
                last_Y = int16([135, 135, 135, 135]);
                mode = 2;
            elseif isequal(data_1, [0; 0; 0; 1])
                last_Y = int16([90, 90, 90, 90]);
                mode = 2;
            end
            Y = last_Y;
        end
    end
```

El siguiente estado (ver código A.2) es el correspondiente al modo de funcionamiento 4. Este estado, sirve de estado intermedio entre el primer estado y el tercer estado, que se encarga del modo de nivelado horizontal automático (modo 5):

Código A.2: Control de modo 4 para el robot Horu

```
Mode_4

entry:
    mode = 4;
during:
    % Par delantero. Trigger derecho activado (bajar oruga)
    %                 Trigger izquierdo activado (subir oruga)
    if isequal(data_0, [1; 0 ;0 ;0])
        if isequal(trigger, [1; 0])
            % Subir orugas delanteras
            Y = [-1, 1, 0, 0];
        elseif isequal(trigger, [0; 1])
            % Bajar orugas delanteras
            Y = [1, -1, 0, 0];
        else
            Y = [0, 0, 0, 0];
        end
        % Par trasero. Trigger derecho activado (bajar oruga)
        %              Trigger izquierdo activado (subir oruga)
    elseif isequal(data_0, [0; 1; 0; 0])
        if isequal(trigger, [1; 0])
            % Subir orugas traseras
            Y = [0, 0, 1, -1];
        elseif isequal(trigger, [0; 1])
            % Bajar orugas traseras
            Y = [0, 0, -1, 1];
```

```
27          else
28              Y = [0, 0, 0, 0];
29          end
30          % Par izquierdo. Trigger derecho activado (bajar oruga)
31          %                 Trigger izquierdo activado (subir oruga)
32      elseif isequal(data_0, [0; 0; 1; 0])
33          if isequal(trigger, [1; 0])
34              % Subir orugas izquierdas
35              Y = [0, 1, 0, -1];
36          elseif isequal(trigger, [0; 1])
37              % Bajar orugas izquierdas
38              Y = [0, -1, 0, 1];
39          else
40              Y = [0, 0, 0, 0];
41          end
42          % Par derecho. Trigger derecho activado (bajar oruga)
43          %              Trigger izquierdo activado (subir oruga)
44      elseif isequal(data_0, [0; 0; 0; 1])
45          if isequal(trigger, [1; 0])
46              % Subir orugas derechas
47              Y = [-1, 0, 1, 0];
48          elseif isequal(trigger, [0; 1])
49              % Bajar orugas derechas
50              Y = [1, 0, -1, 0];
51          else
52              Y = [0, 0, 0, 0];
53          end
54          % Todas las orugas activas
55      elseif isequal(data_0, [0; 0; 0; 0])
56          Y = [0, 0, 0, 0];
57          % Caso por defecto
58      else
59          Y = [0, 0, 0, 0];
60      end
```

Y por último, para pasar al estado tres y ejecutar el modo de nivelado se tiene el siguiente código (ver código A.3):

Código A.3: Control de modo 5 para el robot Horu

```
1 Mode_5
2 entry:
3       mode = 5;
```

Como se puede observar, es mucho más simple y útil tener varios modos de funcionamiento combinados en un mismo estado (estado 1). Esto facilita al operador saber en qué modo se encuentra, ya que, aunque existen tres estados, los principales son solo dos. Uno de estos modos permite operar con el robot en los modos de tracción, control de posición e incremental, mientras que el otro permite nivelar el robot moviendo el par de orugas de forma manual. Sin embargo, si lo desea, el operador puede activar el modo automático manteniendo presionado un botón.