COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

IMPLEMENTACIÓN DE ALGORITMOS DE NAVEGACIÓN EN UN TURTLEBOT3

Autor: Francisco Barragán Castro

Directores:

Jaime Boal Martín-Larrauri

Ramón Rodríguez Pecharromán

Madrid

Junio de 2022

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Implementación de algoritmos de navegación en un Turtlebot3

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2021/22 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Francisco Barragán Castro Fecha: 04/ 07 /22

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado digitalmente por BOAL MARTIN LARRAURI JAIME - 05304600H
Fecha: 2022.07.04 12:02:20 +02'00'

Fdo.: Jaime Boal Martín-Larrauri Fecha: 04/ 07 /22

RODRIGUEZ PECHARROMAN RAMON - 02617129M

Firmado digitalmente por RODRIGUEZ PECHARROMAN RAMON - 02617129M
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-02617129M, givenName=RAMON, sn=RODRIGUEZ PECHARROMAN, cn=RODRIGUEZ PECHARROMAN RAMON - 02617129M
Fecha: 2022.07.04 12:54:19 +02'00'

Fdo.: Ramón Rodríguez Pecharromán Fecha: 04/ 07 /22

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

IMPLEMENTACIÓN DE ALGORITMOS DE NAVEGACIÓN EN UN TURTLEBOT3

Autor: Francisco Barragán Castro

Directores:

Jaime Boal Martín-Larrauri

Ramón Rodríguez Pecharromán

Madrid

Junio de 2022

Agradecimientos

En primer lugar, quisiera agradecer mis ambos directores, Jaime Boal y Ramón Rodríguez, por brindarme la oportunidad de realizar este proyecto. También me gustaría apreciar su apoyo, paciencia y constancia a la hora de ayudarme a desarrollarlo.

Finalmente agradecer a mi familia, compañeros y amigos, que han estado a mi lado y me han ayudado en mi desarrollo tanto académico como personal durante esta etapa de mi carrera.

IMPLEMENTACIÓN DE ALGORITMOS DE NAVEGACIÓN EN UN TURTLEBOT3

Autor: Barragán Castro, Francisco.
Directores: Boal Martín-Larrauri, Jaime y Rodríguez Pecharromán, Ramón.

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El Turtlebot3 [1] es un pequeño robot programable diseñado para facilitar el desarrollo de redes en ROS. Su bajo coste unido a su gran comunidad lo hacen perfecto para proyectos educacionales. En este proyecto se han desarrollado y validado algoritmos de navegación usando dicho robot. Los modelos desarrollados permiten validar el flujo de trabajo de controles usando MATLAB, para ser después desplegados en entornos de producción.

Palabras clave: Turtlebot3, ROS2, Robot móvil, Modelado de planta, vehículo diferencial, Navegación, PID

1. Introducción

Este proyecto ha sido desarrollado con el objetivo de crear una plataforma de robótica para la nueva asignatura de robótica avanzada del iMAT. Los algoritmos diseñados en este proyecto serán desplegados como algoritmos de bajo nivel sobre los cuales los alumnos desarrollarán otros de nivel superior. Además, se pretende usar el robot para los laboratorios de las asignaturas de control del grado donde se enseña el diseño de controles PID.

2. Definición del proyecto

El proyecto se puede dividir en tres partes principales:

- Control PWM de los motores mediante ROS2.
- Diseño e implementación de un algoritmo de control de velocidad de avance.
- Diseño e implementación de un algoritmo de seguimiento de la pared.

El control PWM de los motores se trata de una parte crucial del proyecto, ya que el desarrollo de los controles depende de poder mandar consignas desde la red de ROS2. Es por ello por lo que esta parte ha sido la primera en tratar. Por otra parte, los algoritmos de velocidad y seguimiento de pared se pueden diseñar en paralelo, ya que el usuario puede cambiar fácilmente entre ellos.

3. Descripción del modelo/sistema/herramienta

3.1. Arquitectura

Hay varias maneras de controlar un actuador desde un ordenador dependiendo del número de sensores y actuadores que uno desee controlar [3]. En este caso, se usan unos servos motores digitales Dynamixel. Para controlarlos se puede usar un solo bus de comunicación con un microcontrolador conectándolos en serie. La tarjeta OpenCR [2] ha sido escogida para este propósito, ya que tiene su propio transductor integrado para comunicarse con los motores por TTL (transistor-transistor logic). El microcontrolador también se comunicar con el ordenador por USB para retrasmitir la información obtenida. El ordenador elegido ha sido una Raspberry Pi 3B, sobre la que se ha desplegado una red de ROS. Este se ocupa de hacer las tareas de alto nivel del sistema.

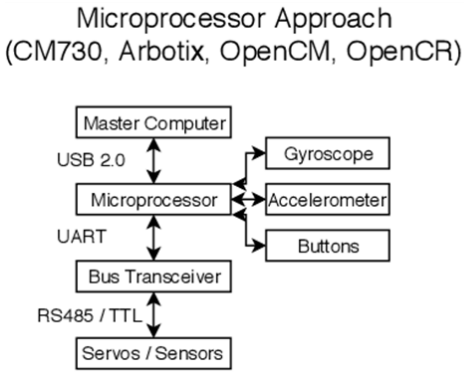

Figura 1 Diagrama de la arquitectura del sistema [3].

3.2. Modelo de la planta

Para poder diseñar correctamente los algoritmos propuestos, se han modelado la planta del sistema a controlar.

Para el algoritmo de control de velocidad se ha identificado un sistema de segundo orden, con la dificultad extra de un retraso marcado por las comunicaciones.

$$P\left(\frac{rad/s}{PWM\ \%}\right) = \frac{88.92}{(s+57)(s+24)} * e^{-\,0.09s}$$

Ecuación 1 Planta de velocidad.

Por otro lado, para el algoritmo de seguimiento de pared, se ha escogido un control en cascada. Por tanto, la planta del lazo externo depende del control escogido para el lazo interno.

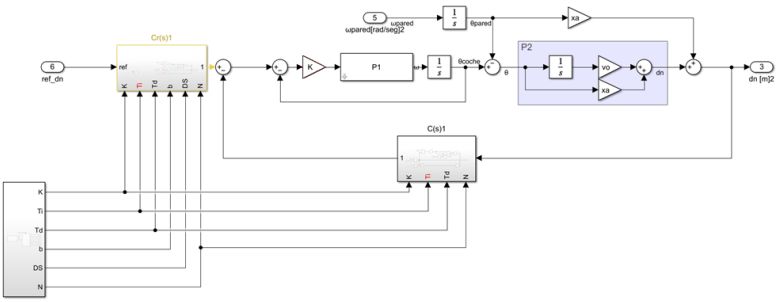

Figura 2 Estructura del control de seguimiento de pared.

Esta estructura ha sido elegida para mejorar la resiliencia del sistema, ya que la doble integración de la planta dificulta el diseño de controles robustos con un solo lazo. Se han identificado las plantas P1 y P2 de la Figura 2, obteniendo las siguientes funciones de transferencia.

$$P1(ud/\theta) = \frac{-2}{\left(\frac{s}{1.2}+1\right) * s} \qquad P2(d_n/\theta\,) = \frac{0.0395\ s\ +\ 0.1}{s}$$

Ecuación 2 Plantas del control de seguimiento de pared.

3.3. Modelo de la planta

Los algoritmos de lazo cerrado han sido desarrollados con la ayuda de la MATLAB ROS Toolbox [4]. Esta herramienta permite al usuario iterar y desplegar los controles con facilidad usando sus diferentes modos de trabajo.

El control de velocidad ha sido desarrollado con un simple lazo de control PID. Este control se ha creado en Simulink y luego se ha convertido en código C++ para ser ejecutado como nodo en la Raspberry Pi. Para monitorizar y procesar la información generada durante ensayos, se ha creado un segundo archivo de Simulink que archiva esta información. Estos datos pueden ser después comparados con una simulación en Simulink para validar los ensayos.

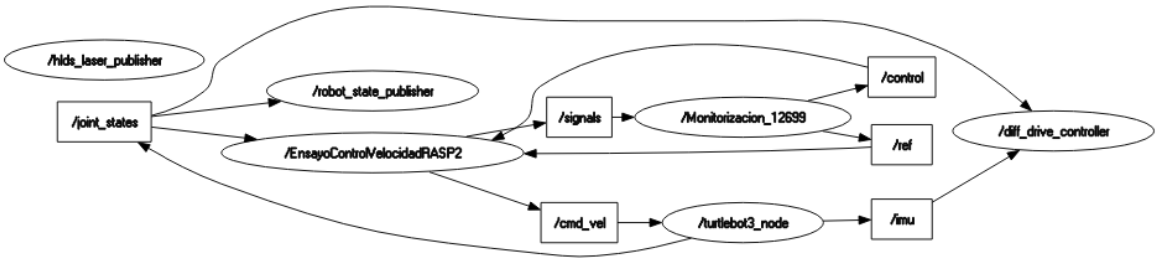

Figura 3 Red de ROS para el control de velocidad.

El algoritmo de seguimiento de pared por su parte se ha creado usando las capacidades de conexión con ROS de la Toolbox. En este método, Simulink hace todos los cálculos del control, y manda y recibe la información de la red mediante una serie de publishers and subscribers.

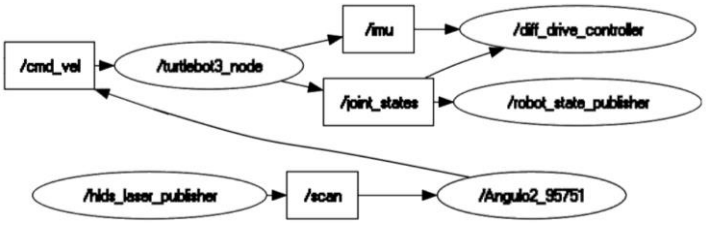

Figura 4 R Red de ROS para el control de seguimiento de pared.

4. Resultados

Durante el desarrollo del proyecto se han realizado una serie de ensayos para validar los algoritmos. A continuación, se muestra una gráfica de un ensayo de cada control.

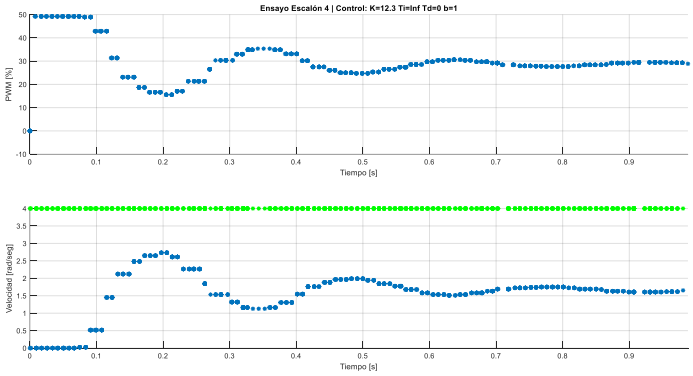

Figure 5 Velocity step on the velocity algorithm.

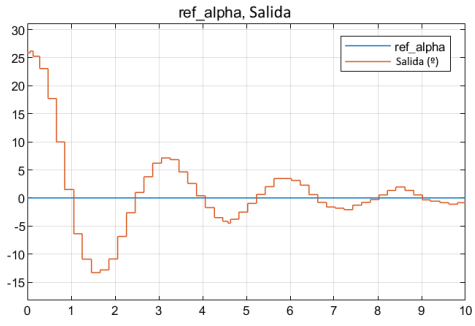

Figure 6 Angle correction on the wall-following algorithm.

5. Conclusiones

El resultado del proyecto ha sido la creación de dos algoritmos que podrían usarse para cumplir con nuestro objetivo principal, ejecutarse como algoritmos de bajo nivel en el curso de robótica avanzada del iMAT. Por lo tanto, se cumplieron todos los objetivos planteados en el proyecto.

1. Referencias

[1] «TurtleBot3». https://emanual.robotis.com/docs/en/platform/turtlebot3/bringup/#bringup (accedido 2 de abril de 2022).

[2] *OpenCR: Open Source Control Module for ROS*. ROBOTIS, 2022. Accedido: 25 de junio de 2022. [En línea]. Disponible en: https://github.com/ROBOTIS-GIT/OpenCR

[3] M. Bestmann, J. Guldenstein, y J. Zhang, «High-Frequency Multi Bus Servo and Sensor Communication Using the Dynamixel Protocol», p. 14.

[4] «ROS in Simulink - MATLAB & Simulink - MathWorks España». https://es.mathworks.com/help/ros/ros-in-simulink.html?s_tid=CRUX_lftnav (accedido 24 de junio de 2022).

IMPLEMENTATION OF NAVIGATION ALGORITHMS ON A TURTLEBOT3

Author: Barragán Castro, Francisco.
Supervisors: Boal Martín-Larrauri, Jaime, Rodríguez Pecharromán, Ramón.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

The Turtlebot3 [1] is a small programmable robot designed to ease the development of ROS networks. It's low cost and extensive community make it perfect for educational projects.

This thesis develops and validates navigation controllers using MATLAB to be deployed later on the aforementioned platform using ROS.

Keywords: Turtlebot3, ROS2, Mobile robots, Plant modeling, Differential drive, Navigation, PID

2. Introduction

This project has been developed with the aim of creating a robotics platform for the newly created robotics course on the iMAT degree. The algorithms developed on this project will be deployed as low-level algorithms for the students to build upon. In addition, this robot will be used in the laboratories of control courses to teach PID algorithms.

3. Project definition

The project can be clearly divided into three main parts:

- PWM motor control over the ROS2 network.
- Design and implementation of a velocity control algorithm.
- Design and implementation of a wall following algorithm.

The PWM motor control was a crucial part for the development of the project, as it enabled creating closed loop controls over the ROS2 network. This task had to be carried out first as the designed algorithms rely on it. Meanwhile, the velocity and wall following algorithms can be designed simultaneously, as the user can easily switch between both.

4. Model description

4.1. Architecture

There are multiple ways of controlling an actuator from a computer. Depending on the total number of sensors and actuators one would like to coordinate [3]. In this case, as we are using Dynamixel digital actuators, we use a microcontroller with a single communication bus with the motors. The OpenCR [2] board has been chosen for this purpose as it has an integrated transducer to communicate with the motors with TTL. The microcontroller also communicates with the master computer over USB. This computer is a Raspberry Pi 3B, which runs ROS2 and enables us to create the controls.

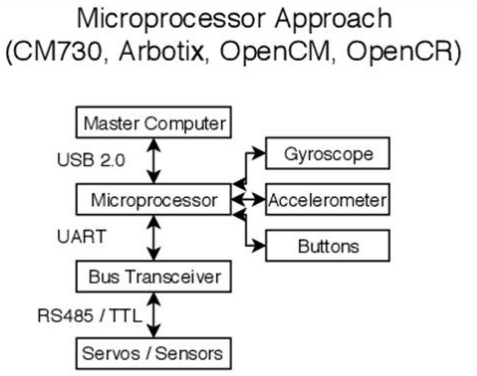

Figure 1 Communication diagram between devices[3]

4.2. Plant Model

In order to design the algorithms described on the project, a plant model of the system to be controlled was developed. For the velocity control algorithm, a simple second order model was identified, with the added complexity of a delay caused by the communications.

$$P\left(\frac{rad/s}{PWM\ \%}\right) = \frac{88.92}{(s+57)(s+24)} * e^{-0.09s}$$

Equation 4 Velocity plant.

On the other hand, for the wall following algorithm, a cascade control was chosen. Therefore, the plant of the external loop depends on the controller chosen for the internal loop.

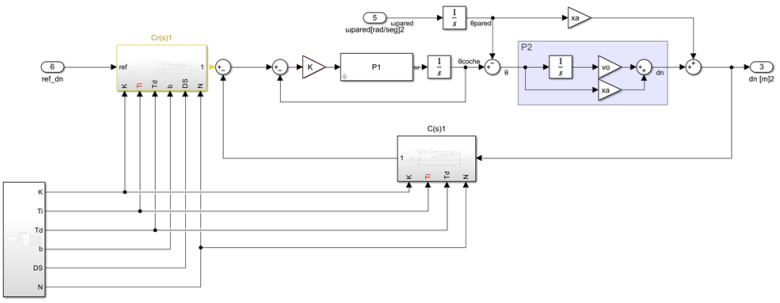

Figure 2 Structure of the wall following algorithm.

This structure improves the resiliency of the system, as the double integration on the main plant would make it hard to create a robust control with a single loop. The P1 and P2 plants, shown on Figure 2, where identified. The resulting transfer functions are shown below.

$$P1(ud/\theta) = \frac{-2}{\left(\frac{s}{1.2}+1\right) * s} \qquad P2(d_n/\theta\ ) = \frac{0.0395\ s\ +\ 0.1}{s}$$

Equation 5 Plant of wall following algorithm

4.3. Model deployment

The closed loop algorithms were developed with the help of the MATLAB ROS Toolbox [4]. This tool allows iterating and deploying our controls with ease using its different workflows.

The velocity control was developed as a simple closed loop PID control. This control was created in Simulink and later converted into C++ code to be run as a node on the Raspberry Pi. To monitor and process the data generated during trials, a second Simulink model was developed. This data can be later compared to the simulated model in Simulink to validate it.

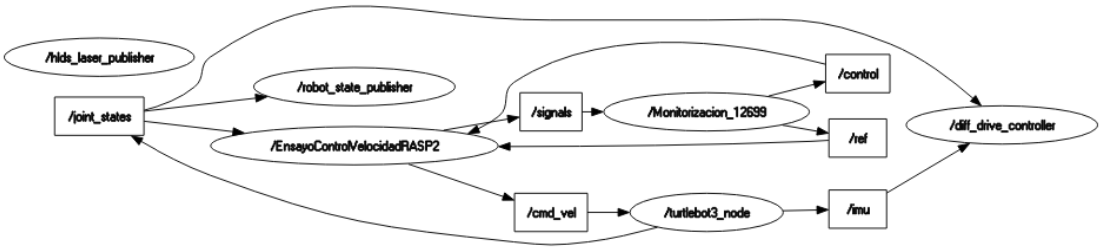

Figure 6 ROS2 network of the velocity control

The wall following algorithm, on the other hand, was created using the ROS connection capabilities of the Toolbox. In this method, Simulink does all the calculations of the PID control, and it sends and shares data to the network through a series of publishers and subscribers.

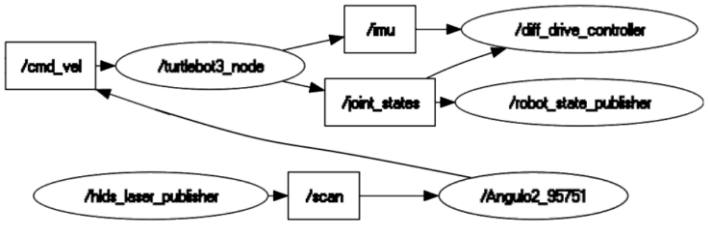

Figure 4 ROS2 network of the wall following control

5. Results

During the development of the project, a series of tests were carried out to validate the algorithms. Below is a graph of one trial of each control.

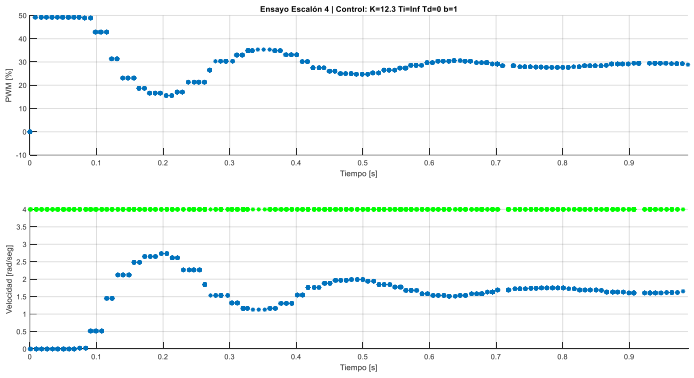

Figure 5 Velocity step on the velocity algorithm.

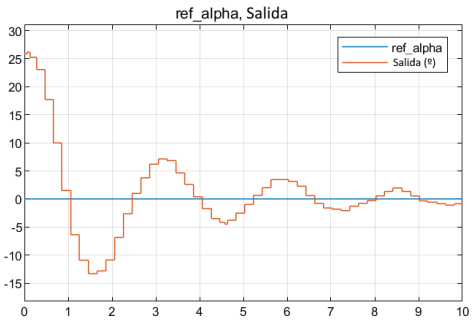

Figure 6 Angle correction on the wall following algorithm.

6. Conclusion

The project resulted in the creation of algorithms which could be used to fulfill our main objective of being run as low-level algorithms on the advanced robotics course of the iMAT. Therefore, all the set objectives of the project have been fulfilled.

7. References

[1] «TurtleBot3». https://emanual.robotis.com/docs/en/platform/turtlebot3/bringup/#bringup (accedido 2 de abril de 2022).

[2] *OpenCR: Open Source Control Module for ROS*. ROBOTIS, 2022. Accedido: 25 de junio de 2022. [En línea]. Disponible en: https://github.com/ROBOTIS-GIT/OpenCR

[3] M. Bestmann, J. Guldenstein, y J. Zhang, «High-Frequency Multi Bus Servo and Sensor Communication Using the Dynamixel Protocol», p. 14.

[4] «ROS in Simulink - MATLAB & Simulink - MathWorks España». https://es.mathworks.com/help/ros/ros-in-simulink.html?s_tid=CRUX_lftnav (accedido 24 de junio de 2022).

ÍNDICE DE LA MEMORIA

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

ABREVIATURAS

SBC	Single Board Computer
OPENCR	Open-source Control Module for ROS
FPU	Floating point unit
PCB	Printed board circuit
BOM	Bill of materials
QoS	Quality of service
ROS2	Robot Operating System 2
TTL	Transistor-transistor Logic
USB	Universal Serial Bus
SLAM	Simultaneous Localization and Mapping
LIDAR	Laser Imaging Detection and Ranging
PWM	Pulse-Width Modulation
RAM	Random Access Memory
EEPROM	ROM programable y borrable eléctricamente
IAE	Integral Absolute Error

1 Introducción

Este trabajo de fin de grado consiste en desarrollar todas las herramientas necesarias para poder ajustar los algoritmos de control de velocidad y navegación de un robot móvil con cinemática diferencial, Turtlebot3 Burger[1], para su uso en el laboratorio de las asignaturas de control de la universidad. Por tanto, el robot ha de ser capaz de percibir el entorno que le rodea, e implantar controles que le permitan navegar a una velocidad razonable sin chocarse. El ajuste de los controles se realiza usando MATLAB y Simulink. La implementación de los controles por su parte se realiza en ROS2[2].

1.1 Motivación del Proyecto

En el laboratorio de control de ICAI se usan diferentes plataformas para enseñar los diferentes aspectos de los sistemas de control. Entre ellos se encuentra un robot móvil, dicho robot se usa para enseñar varios algoritmos de control de velocidad como el PID, o el *deadbeat*. La plataforma actual está basada en LEGO MINDSTORMS, el cual tiene la gran ventaja de su simplicidad y facilidad para reparar. Aunque tiene a su vez la desventaja de que está limitado ya que solo se pueden usar módulos oficiales. Por ello, con el fin de mejorar la experiencia de los alumnos, se ha decidido apostar por una nueva plataforma.

Además, se pretende usar un robot usando la tecnología de ROS2 para las asignaturas de robótica avanzada del iMAT. Es conveniente, por tanto, usar el mismo robot para ambos laboratorios que los alumnos estén familiarizados con este.

1.2 Objetivos del Proyecto

Los objetivos de este proyecto son diseñar e implementar un sistema de control para un robot móvil. El sistema de control se implementará en MATLAB/Simulink y se utilizará para controlar la posición y la velocidad del robot. El proyecto también incluirá un estudio de las dinámicas del robot y el diseño de un controlador para lograr el rendimiento deseado y una interfaz para poder interactuar

fácilmente con los parámetros del robot en un entorno didáctico. Los objetivos del proyecto son, por tanto:

1. Determinar si el robot es apropiado para su uso en asignaturas de control de GITI y robots móviles en el nuevo grado iMAT.
2. Desarrollar y ensayar un control de velocidad para el robot.
3. Desarrollar y ensayar un control de navegación en una aplicación de seguimiento de pared o similar.

1.3 Estado del arte

Existen diferentes modelos de vehículos para el uso académico en entornos de modelado y control, pero la elección de un modelo no es fácil ya que tiene que cumplir nuestras condiciones de diseño. Debido a que querremos realizar el control de velocidad del robot, es necesario que el modelo que elijamos posea alguna forma de medir la velocidad de este. La manera más fiable de conseguir dichos datos son tener enconder en las ruedas del coche, por lo que se buscaran modelos que tengan dichos sensores. Además, es necesario que el robot tenga alguna forma fácil de conectarse con Matlab, ya que se quiere realizar el diseño de los controles desde allí. A continuación, se muestra una serie de modelos que están disponibles en el mercado, cada uno con sus ventajas e inconvenientes, que finalmente han sido descartados.

- QBot 2e: De la famosa empresa de productos educativos Quanser, trae varias ventajas, como su gran calidad y el gran soporte por parte de la empresa con guías. Además, posee multitud de sensores, incluidos *encoders* y la posibilidad de usar el *software* QLabs Virtual QBot 2e para interactuar con MATLAB®/Simulink®. Desgraciadamente, este software tiene dos inconvenientes decisivos. Primero de todo, se trata de una plataforma muy cerrada y que requiere una suscripción. En segundo lugar, dicho programa está pensado para realizar control a más alto nivel, por lo que no es posible acceder a los valores que nos dan los *encoders* a través de Matlab.
- Alphabot2: Una alternativa *open-source* de la empresa WaveShare trae varias ventajas como su bajo coste y su modularidad. Además, dispone de un sensor encoder para medir la velocidad de cada motor. Dicho sensor es de un *enconder* óptico de 20 ranuras, por lo que nos dará una precisión muy baja en el control. Además, no posee ninguna herramienta directa para comunicarse con MATLAB, aunque existe documentación de usuarios que lo han conseguido previamente.
- Turtlebot3: Es un pequeño robot programable pensado para ser usado en ROS. Su bajo coste y gran comunidad lo hacen perfecto para proyectos de educación, investigación y prototipado. El objetivo de la creación del Turtlebot3, respecto a iteraciones anteriores de la gama, era reducir drásticamente el tamaño y precio sin reiniciar a calidad y funcionalidad. Sus motores son unos Dynamixel servo, los cuales permiten tener fácilmente las medidas de posición y velocidad. Por otra parte, está basado en la tecnología ROS (Robot Operating System), que se describe en el capítulo 2.4. Esta tecnología es la buscada para el laboratorio

de robots móviles autónomos del iMat, por otra parte, permite conectar fácilmente el robot a MATLAB usando la ROS Toolbox.

Finalmente se ha elegido el Turtlebot3 debido a que es el que más se acerca a nuestros objetivos. Ya que es el único que corre ROS2 y nos permite conectarse a MATLAB desde un entorno *open-source.*

1.4 Objetivos de Desarrollo Sostenible

Las metas de este proyecto se alinean con las metas de desarrollo sostenible propuestos a los lideres mundiales en septiembre de 2015[3]. Algunos de los objetivos tratados son:

1.4.1 Objetivo 4: Garantizar una educación inclusiva, equitativa y de calidad y promover oportunidades de aprendizaje durante toda la vida para todos

El objetivo número cuatro de los ODS se refiere a la necesidad de asegurar educación inclusiva, equitativa y de calidad, y promover oportunidades de aprendizaje para todos a lo largo de la vida. Este proyecto tendrá como objetivo el desarrollo de un sistema de control para un robot móvil que pueda ser utilizado en entornos educativos. El uso de este robot en la enseñanza ayudará a promover la inclusión de estudiantes en estas áreas. También ayudará a mejorar las habilidades de estudiantes y prepararlos para el trabajo en estas áreas. Esto ayudará a mejorar la sostenibilidad de la fuerza laboral en el futuro.

1.4.2 Objetivo 9: Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación

El objetivo número nueve de los ODS se refiere a la necesidad de construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación. El desarrollo de un sistema de control para un robot móvil requiere el uso de métodos e innovadoras tecnologías. El resultado será un sistema que pueda ser utilizado en entornos educativos y que ayudará a mejorar la enseñanza de los sistemas de control. Esto ayudará a promover la industrialización sostenible mediante la capacitación de una nueva generación de ingenieros.

2

Descripción de tecnologías

En este capítulo se describen las tecnologías utilizadas en el proyecto para familiarizar al lector con estas y servir como referencias.

2.1 Turtlebot3

El Turtlebot3 es un pequeño robot programable pensado para ser usado en ROS. Su bajo coste y gran comunidad de usuarios lo hacen perfecto para proyectos de educación, investigación y prototipado. El objetivo de la creación del Turtlebot3, respecto a iteraciones anteriores de la gama, era reducir drásticamente el tamaño y precio sin renunciar a la calidad y funcionalidad.

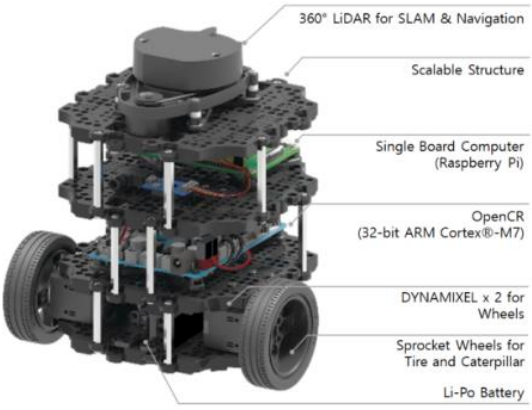

Figura 1 Turtlebot3 Burger

Su diseño modular, basado en niveles unidos por separadores, le ofrece una gran capacidad de modificación y expansión. Además, todo el *software* y planos del robot son *open-source*, para incentivar la colaboración y personalización.

El TurtleBot3 se puede personalizar de varias maneras, dependiendo de cómo se reconstruyan las partes mecánicas y se utilicen partes opcionales como un ordenador y sensores.

En la Figura 1 se pueden ver los principales componentes del robot. Estos son:

- Sensor LDS-01: Sensor 360° LiDAR para ayudar en tareas de SLAM y navegación.
- SBC: En nuestro caso se trata de una Raspberry Pi 3B, aunque los nuevos modelos vienen con la Raspberry Pi 4.
- OpenCR: Un microcontrolador de código abierto para conectar los motores y sensores con el SBC.
- 2x servomotores digitales Dynamixel XL430-W250-T.
- Batería Li-Po de 11,1V 1800mAh.

2.2 Dynamixel XL430-W250-T

El Dynamixel XL430-W250-T es un *smart-servo* desarrollado por la empresa ROBOTIS. Se trata de un actuador de la gama de bajo coste de la empresa.Aun así, se puede apreciar en la siguiente gráfica que tiene un par considerable.

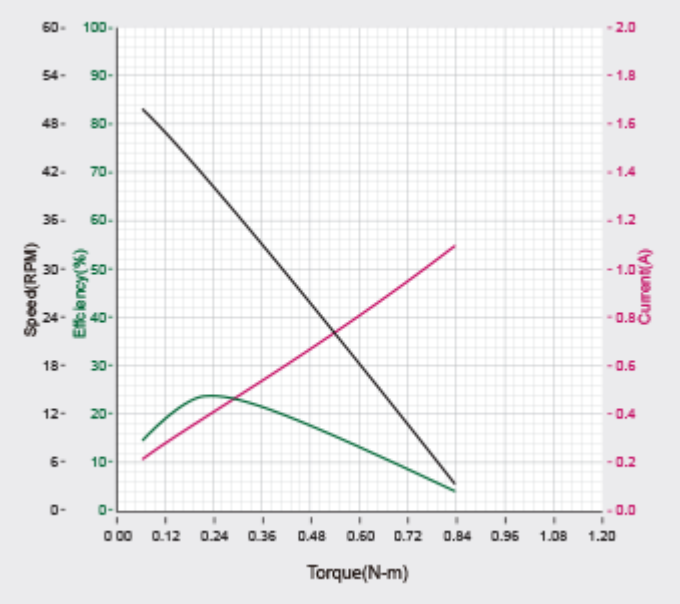

Figura 2 Características del Dynamixel XL430-W250

Al ser un *smart-servo*, el actuador tiene un microcontrolador ARM CORTEX-M3 el cual controla el desempeño del motor, en este caso un motor DC con núcleo. La interacción con el motor se hace usando el protocolo Dynamixel 2.0 para leer y escribir en la tabla de control interna del dispositivo.

La última letra del nombre del motor indica el protocolo de comunicación, en este caso se trata de TTL (Transistor-transistor Logic).

En la tabla de control existen diferentes direcciones para cambiar los ajustes del dispositivo. Entre ellas se encuentra el modo de operación. El valor de esta dirección de memoria(11) determinará el tipo de control interno que ejecutará el motor.

- [1] Control de velocidad: Permite controlar la velocidad del motor.
- [3] Control de posición (por defecto): Permite controlar la posición del robot entre 0 y 360º, esto limita el radio de acción del servo a una revolución.
- [4] Control de posición extendido. Permite controlar la posición pudiendo recorrer más de una vuelta, hasta un máximo de 512 vueltas.
- [16] Control PWM: Desactiva todos los controles del dispositivo y aplica un voltaje al motor directamente.

Para este proyecto se ha usado el modo control PWM para poder controlar el motor de forma externa.

2.3 OpenCR[4]

La placa OpenCR (Open-source Control Module for ROS) es una placa *open-source* desarrollada por la empresa ROBOTIS para el uso en sistemas integrados de ROS. Tanto los planos, PCB (Printed board circuit), como el BOM (Bill of materials) y *firmware* son de libre acceso dentro en GitHub[5], [6].

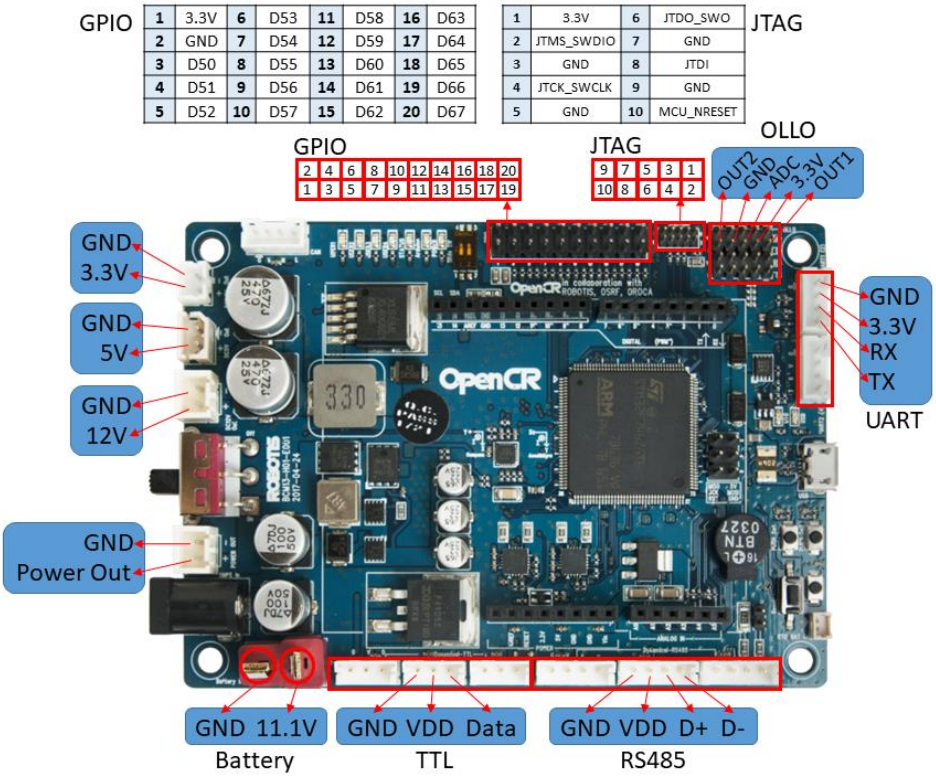

Figura 3 OpenCR[6]

Está basada en el microcontrolador STM32F7[7], el cual contiene el procesador ARM Cortex®-M7, un procesador de 32-bit con FPU (Floating point unit). Con él nos podemos conectar a los sensores internos y a los motores con facilidad.

Cuenta con un acelerómetro y giroscopio cuyos datos son mandados en nuestro robot al SBC. Por otra parte, ofrece la posibilidad de conectar y controlar hasta seis motores DYNAMIXEL, tres por puertos TTL y otros tres por RS485.

La carga del firmware al dispositivo se puede hacer mediante micro USB usando la IDE de Arduino. En el momento de la redacción del documento hay un error en la versión `v1.4.19` del *firmware* que obliga a bajar la librería DYNAMIXEL2Arduino a la versión `v0.3.0`.

2.4 ROS2[2]

El software Robot Operating System (ROS) es un *middleware* que permite la comunicación entre diferentes sistemas de control de un robot. Está compuesto por una compilación de librerías y herramientas de código abierto que permiten diseñar e implementar todas las tecnologías necesarias para un proyecto de robótica. Incluye varias herramientas de alto nivel para realizar tareas de control, navegación, visualización y simulación entre otras, además de elementos de más bajo nivel como capas de abstracción de *hardware* y algoritmos.

El entorno ROS fue desarrollado originalmente en 2007 por la empresa Willow Garage para el robot PR2. Sin embargo, no fue diseñado con muchos algoritmos y características necesarias para las necesidades actuales del diseño de robots. Es por eso que surgió ROS2, la versión usada en este proyecto, que fue rediseñada desde cero para hacer frente a los retos presentes y futuros de la industria de la robótica.

2.4.1 Conceptos generales[8], [9]

ROS2 está basado en un sistema descentralizado de procesos, los cuales se comunican usando un mecanismo anónimo de publicación y suscripción. Estos procesos se despliegan sobre una red de ROS compuesta por los diferentes subsistemas del robot. Esta red puede estar compuesta por varios equipos dentro de una red local.

El concepto general sobre el cual recae el resto de los elementos en una red de ROS2 es el ROS graph. Este se refiere al sistema de nodos y el sistema de conexión sobre el que estos se comunican.

Una forma muy intuitiva de entender una red de ROS es usar la herramienta incluida por defecto de `rqt`, en concreto el node_graph. En la figura siguiente se muestra un diagrama de una red de ejemplo, esta será referenciada en adelante para explicar los elementos de la red.

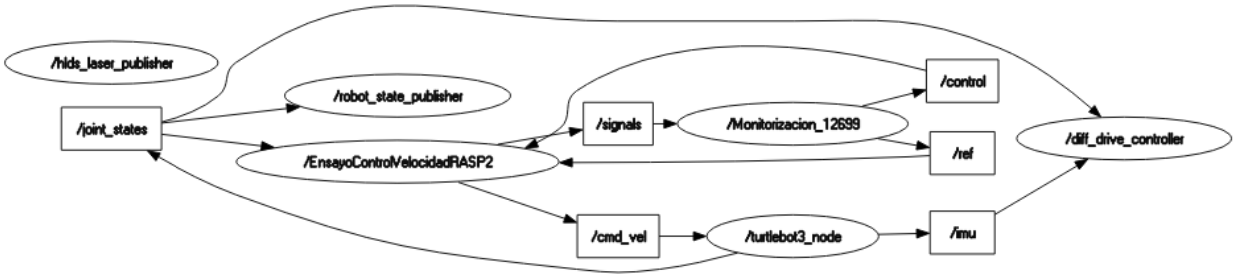

Figura 4 Red de ROS en `rqt`.

A continuación, se explican algunos conceptos básicos para entender una red de ROS.

2.4.1.1 ***Nodos***

Un nodo es un elemento que participa dentro de la red del ROS graph. Los nodos pueden pertenecer al mismo programa, a programas diferentes o incluso a máquinas diferentes.

Cada nodo ha de ser responsable de una sola tarea de alto nivel del robot. Cada nodo puede mandar y recibir información de otros nodos con ajustes QoS (Quality of service) compatibles. Esta información puede venir en forma de otro nodo en usando los mecanismos de topic, servicios, acciones y parámetros.

Dentro del diagrama de `rqt` se representan dentro de unos óvalos.

2.4.1.2 ***Topics***

Los topics son un elemento básico dentro de una red, ya que actúan como bus donde pueden intercambiar mensajes los nodos. Se suelen usar para mandar mensajes que se tienen que retransmitir constantemente.

Un nodo puede mandar a y recibir información de varias fuentes usando un solo topic. También puede comunicarse usando varios topics a la vez.

2.4.1.3 ***Mensajes***

Son la estructura básica con la que los nodos intercambian información en ROS. Cada mensaje de ROS tiene un tipo de mensaje asociado donde se definen los tipos de datos y la estructura de la información que se intercambia.

Por ejemplo, los datos enviados por un LIDAR en nuestro robot se mandan usando el tipo de mensaje *sensor_msgs/LaserScan*. Donde se manda la distancia recorrida por cada rayo, el momento de llegada del mensaje, la mínima distancia medida, etc.

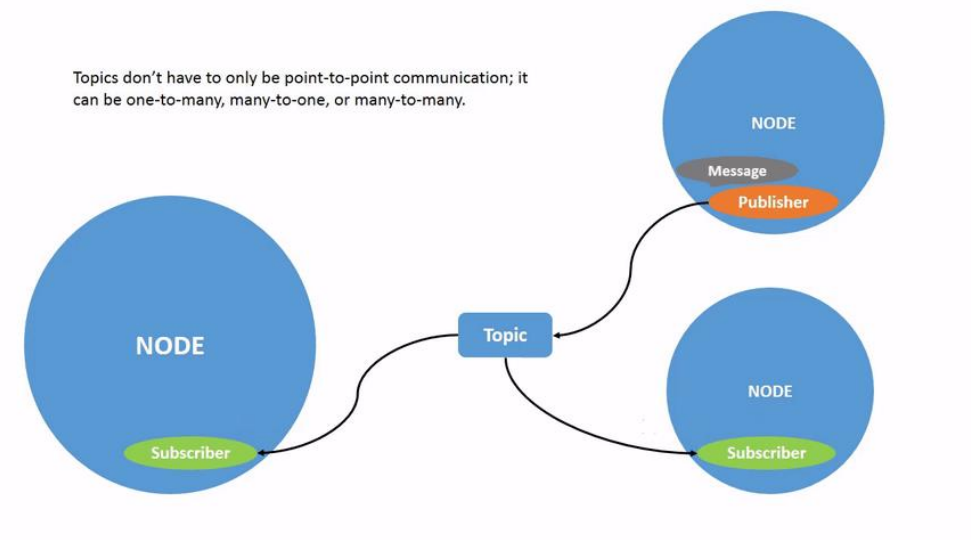

Figura 5 Comunicación entre nodos usando topics[8]

Dentro del diagrama de `rqt` se representan usando un rectángulo. Un ejemplo dentro de nuestro robot es el topic `/scan`, donde se publican todos los mensajes de tipo *sensor_msgs/LaserScan* que recibe del LIDAR.

2.4.1.4 ***Publishers***

Un publisher, es un elemento que manda mensajes a un topic. Este mensaje lo recibirán todos los nodos que tengan un subscriber a ese topic. Puede haber varios mensajes dentro de un topic, aunque hay que tener en cuenta que no hay ninguna trazabilidad de la procedencia de un mensaje una vez que se publica dentro de un topic, ya que todos van al mismo pool donde los recogen los suscriptores.

Dentro del diagrama de RQT se representan con una flecha que parte de un nodo y llega a un topic. Un ejemplo en nuestra red puede encontrarse en el nodo `/EnsayoControlVelocidad`, nodo que contiene un control de lazo cerrado de velocidad. Esta manda la consigna PWM de los motores en el topic `/cmd_vel` donde se publican mensajes de tipo *sensor_msgs/Twist con la tensión común y diferencial para los motores*.

2.4.1.5 ***Subscribers***

Un *subscriber* por su parte, es un elemento que recibe los mensajes que se publican en un topic. Puede haber varios dentro de un topic y todos ellos pueden leer una copia del mismo mensaje.

Dentro del diagrama de `rqt` se representan con una flecha que parte de un topic y llega a un nodo. En la red de nuestro robot un ejemplo se puede encontrar el nodo `/turtlebot3_node`, nodo que contiene el programa principal. Este recibe la información del topic `/cmd_vel`, donde se publican todos los mensajes de tipo *sensor_msgs/Twist,* que contienen la consigna PWM para los motores.

2.4.1.6 ***Servicios***

Un servicio constituye la segunda manera principal de retrasmitir información entre nodos, además del topic. A diferencia del sistema de conexión unidireccional del publisher/subscriber, un servicio se basa en sistema de petición-respuesta.

Los servicios no están pensados para mandar un flujo de información constante, sino para enviar datos siempre que un cliente lo pida. Además, a diferencia de un topic, funcionan con una comunicación punto a punto. Es decir, solo recibe la información del servidor el cliente que ha hecho la petición.

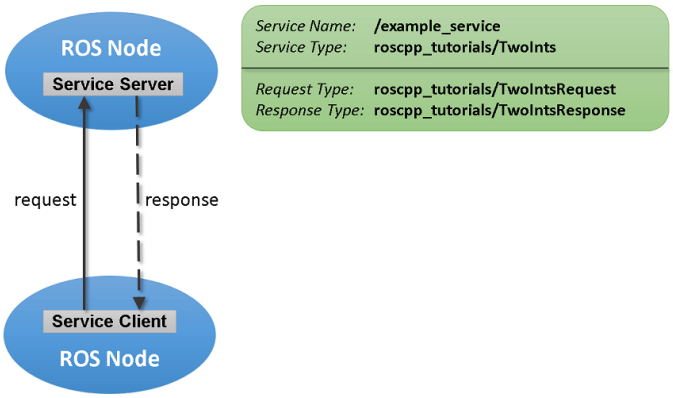

Figura 6 Comunicación entre nodos usando services[8]

La comunicación empieza cuando un cliente del servicio manda un mensaje de petición al servidor de servicios y espera su respuesta. Entonces, usando la información de la petición, el servidor crea una respuesta a la petición y se la devuelve al cliente.

2.4.2 ROS2 Foxy

La distribución de ROS2 que se ha usado en este proyecto es ROS2 Foxy Fitzroy. Esta es la sexta versión de producción de ROS2, que fue publicada en junio de 2020. Fue elegida para elaborar este proyecto ya que se trataba de la última versión LTS al inicio del proyecto. Sin embargo, durante el desarrollo de este ha salido la versión Humble Hawksbill.

2.5 MATLAB ROS2 Toolbox

MATLAB incluye una librería que proporciona una interfaz entre MATLAB y Robot Operating System (ROS y ROS2) para crear una red de nodos.

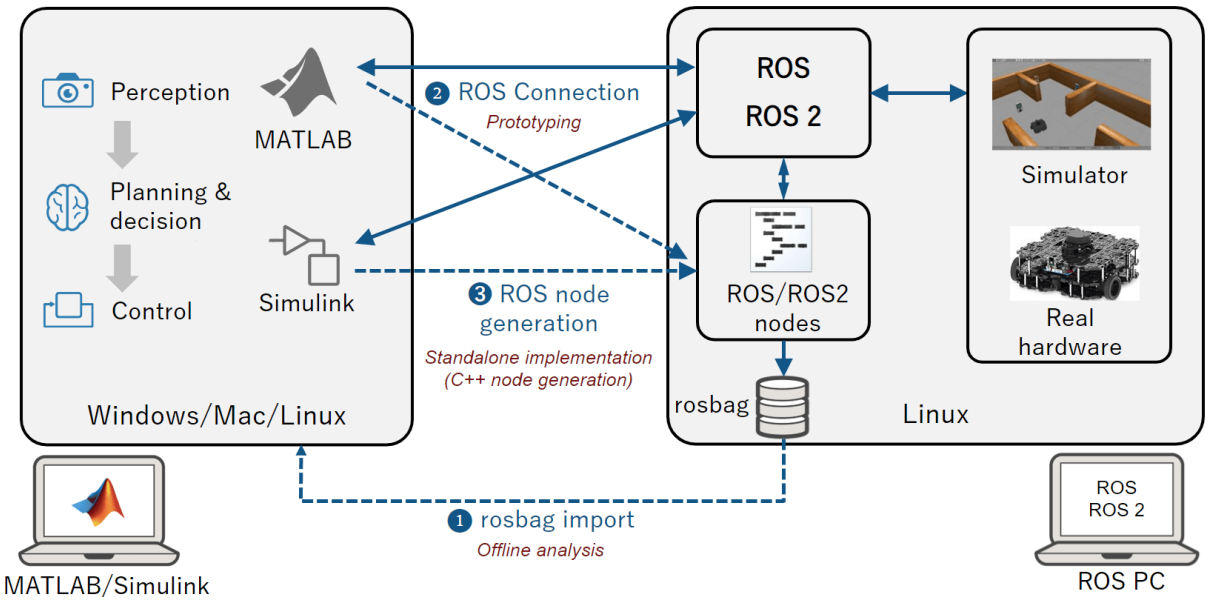

Figura 7 Esquema de comunicación de ROS Toolbox[10]

La Toolbox permite la conexión con ROS usando tres métodos diferentes, tal y como se muestra en la Figura 7 Esquema de comunicación de ROS Toolbox. En los siguientes apartados se detallará su funcionamiento.

Una ventaja importante de esta Toolbox es su fácil instalación y uso, ya que permite a un usuario poco experimentado en ROS trabajar con este con una capa de abstracción superior. Es incluso posible trabajar sin hacer una instalación tradicional de ROS en el ordenador.

Existe actualmente en la versión 2022a de MATLAB un error que obliga a usar dentro de la Toolbox y en el resto de los nodos de la red el middleware *Cyclone DDS* para interactuar con versiones de ROS2 desde Foxy. [11]

2.5.1 Rosbag import

Existe la opción de importar datos de ensayos pasados en MATLAB usando el objeto de rosbag. Este tiene la ventaja de que permite revisitar los ensayos pasados, ya que guarda el contenido que pasa por los topics de interés en un archivo.

Actualmente, en la versión de MATLAB 2022a solo es posible leer rosbag en ROS2. La creación de rosbag está limitada solamente a ROS. Por tanto, para poder crear una rosbag en la actualidad es necesario ejecutar el siguiente comando en una consola de comandos de ROS2, fuera de MATLAB.

```
ros2 bag record <topic_name>
```

2.5.2 Conexión con ROS

La ROS Toolbox permite verificar los nodos ROS diseñados a través de la simulación de escritorio. De esta forma, existe una comunicación fluida entre ambos *softwares*. Esta comunicación permite recibir datos y mandar consignas a los diferentes nodos en tiempo real mientras que se hacen todos los cálculos necesarios dentro del entorno de MATLAB, teniendo así acceso al uso de las otras librerías para simplificar algunas operaciones como el procesamiento de imagen o audio.

2.5.2.1 ***MATLAB[12]***

Para comunicarse con ROS desde MATLAB es necesario crear manualmente el nodo dentro del código. Allí se pueden configurar todos los *publishers* y *subscribers* necesarios para la comunicación. Este método tiene la ventaja de que permite crear con facilidad nuevos tipos de mensajes y que permite tener un mayor control sobre la creación de la red que en la conexión desde Simulink.

2.5.2.2 ***Simulink[13]***

Existe también la posibilidad de establecer la comunicación utilizando Simulink. Este método tiene la ventaja de que permite ver en tiempo real los mensajes recibidos desde ROS. Estos son tratados como señales en buses y se pueden usar para generar lazos de control u otro tipo de algoritmos. Cada archivo de Simulink genera su propio nodo al pulsar el botón de Run. Para redes más complejas se pueden usar varias instancias de Simulink en paralelo para generar una red de nodos.

2.5.3 Despliegue de ROS

La ROS Toolbox soporta también la generación de nodos de ROS en C++. Este método tiene grandes ventajas para entornos de producción ya que permite el uso y distribución del código de forma completamente independiente a MATLAB. Además, el código generado es más eficiente que el usado en el apartado de Conexión con ROS, por lo que permite ejecutar el código a mayores frecuencias de muestreo.

El código C++ puede ser generado tanto para sistemas creados en MATLAB como Simulink usando MATLAB Coder™ y Simulink Coder™. El uso de bloques y funciones de otras toolboxes está limitado por su soporte de generación de código.[14]

2.6 Reguladores PID [3–5]

El control PID es dl control más utilizado en la industria, gracias a su simplicidad, facilidad de ajuste y robustez. De esta forma, se ha convertido en el control por defecto para sistemas mono-variables, siendo la mayoría de estos controles con acción integral.

Las siglas de control PID vienen de control proporcional, integral y derivativo. Cada una de estas acciones del control se dedican a mejorar la respuesta. De esta forma, podemos controlar el error en régimen permanente a la vez que se adecua el amortiguamiento y velocidad, ajustando los parámetros de la acción proporcional, derivativa e integral.

A continuación, se van a explicar las principales funciones de cada parámetro.

- La acción proporcional actúa, como bien dice su nombre, de forma proporcional al error de la salida respecto a la referencia. Así, al aplicar una ganancia a dicho error intentamos acercarnos al valor de referencia más rápidamente. De esta forma, al aumentar la ganancia se corre el error de crear grandes sobrepasos o incluso sistemas inestables. Un problema importante de este control es que tiene error en general en régimen permanente.

- La acción integral consigue arreglar el problema del error en régimen permanente que tienen las otras dos acciones. Como su nombre indica, actúa calculando la integral de la señal error e intenta compensar el error acumulado respecto a la referencia. De esta forma a medida que pasa el tiempo, la señal se va acercando a la referencia hasta conseguir error cero en régimen permanente. La principal desventaja de esta acción es que se añade una inercia al sistema, y por tanto lo hace más lento, perjudicando otros parámetros de la respuesta. Es por ello, que se evita poner esta acción en sistemas donde la planta ya contiene una acción integral (si no hay perturbaciones relevantes), ya que esta realizará la misma función que la componente integral del control.

- Por último, la acción diferencial actúa según la 'velocidad' del error. De esta forma, medimos el ritmo según se va acercando la respuesta a la referencia para poder frenarlo antes de que medida llegue a la referencia deseada. Así se consigue disminuir el sobrepaso y aumentar la velocidad de la respuesta.

La acción conjunta de estos tres controles ha de ajustarse para afinar la ganancia de cada uno y así conseguir el control más rápido, robusto y preciso posible.

2.6.1 Tipos de reguladores PID

Regularmente se usa el termino PID para referirse al regulador del control sin especificar el tipo de control en concreto que se ha usado. Esto es un gran error, ya que dependiendo de la formulación existen diferentes parámetros para el ajuste. Las dos formulaciones más populares son:

- PID interactivo (serie) $C(s) = K_p * \frac{1+Is}{Is} * \frac{1+Ds}{1+fDs}$
- PID no interactivo (paralelo) $C(s) = K * (1 + \frac{1}{T_i s} + \frac{T_d s}{1+\frac{T_d}{N}s}$

Ecuación 7 Modelos PID.

Ambos modelos presentan diferencias, pero para un margen de trabajo habitual, D≤I, f <1 y μ ∈ [1,2]. Para ello se pueden usar las siguientes fórmulas de paso: [16]

$$\mu = 1 + (1 - f) * \frac{D}{I}$$

$$\mathrm{K} = \mu\, K_p \quad T_i = \mu\, \mathrm{I} \qquad T_d = (\frac{1}{\mu} - \mathrm{f}) * \mathrm{D} \qquad N = \frac{1}{\mu \mathrm{f}} - 1$$

$$K_I = \frac{K_p}{I} = \frac{K}{T_i}$$

Ecuación 8 Conversión entre modelos PID.

2.6.2 Ajuste por respuesta en frecuencia

Mediante el estudio de la respuesta en frecuencia del lazo abierto, G(s), se pueden determinar la estabilidad del sistema y realizar varias medidas para medir la robustez de la estabilidad y el amortiguamiento de la respuesta en lazo cerrado. Ya que hay una gran correlación entre la respuesta temporal y la repuesta en frecuencia de un sistema. Los procedimientos de diseño tienen un origen gráfico, aunque también son aptos para un tratamiento analítico.

Los dos procedimientos más comunes son el gráfico de Nyquist y el de Black.

- El gráfico de Nyquist es un gráfico en coordenadas polares que indica amplitud y fase de la respuesta de un sistema en el rango de frecuencias. Existen distintos indicadores que se pueden sacar desde este gráfico, pero no resulta fácil trabajar sobre él, por ello su uso principal es el estudio de estabilidad gracias a la aplicación del criterio de Nyquist.
- También llamado gráfico de Nichols, en él se pueden ver en coordenadas cartesianas la fase en grados y la magnitud en dB a lo largo de la frecuencia. Esta será nuestra principal herramienta en el desarrollo de controles ya que las acciones del control PID se traducen en simples translaciones de la gráfica.

3

Control de motores mediante ROS

El control de los motores se realiza de forma externa, sin utilizar el microcontrolador integrado. El objetivo de esta sección es crear un *firmware* para la placa OpenCR que permita controlar la tensión de los motores desde ROS2.

3.1 Dynamixel

Como se ha explicado en el apartado de descripción de tecnologías, el Dynamixel XL430-W250-T[18] es un actuador inteligente controlado por un microcontrolador interno. Existen diferentes modos de funcionamiento dependiendo de la aplicación, en nuestro caso se va a utilizar el modo PWM. Este nos permite controlar la tensión de los motores de forma externa en lazo abierto mediante unas referencias de PWM, tal y como se muestra en la Figura 8. En este caso, al ser un motor de tipo T(ultima letra del nombre) se comunica usando el protocolo serial TTL. Los paquetes que se mandan están en el protocolo propio de la compañía, el Dynamixel Protocol 2.0.

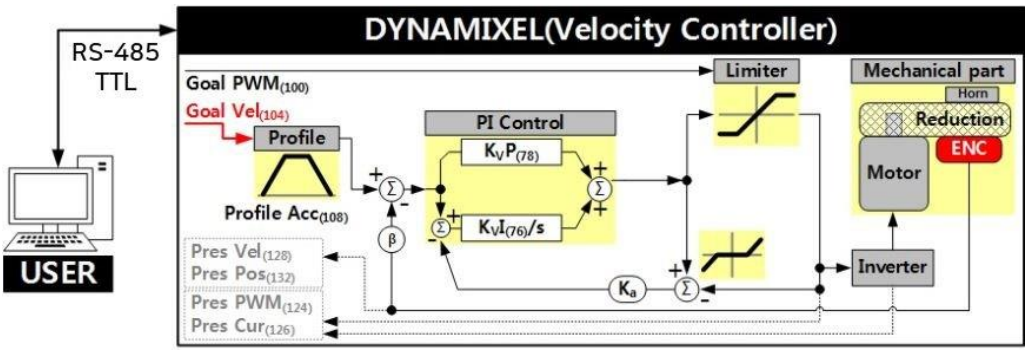

Figura 8 Esquema del control interno de velocidad del motor[19]

A la hora de controlar un robot, es muy importante tener un lazo de control de alta frecuencia. De esta forma, se consigue mejorar la respuesta de los controles y se obtiene una mayor velocidad y sensibilidad. Uno de los principales obstáculos para generar dicho lazo es la velocidad con la que recibimos las medidas del sensor, por lo que es muy importante pensar el protocolo de comunicaciones.

3.2 OpenCR

Hay varias maneras de controlar un motor desde un ordenador, dependiendo del número y tipo de sensores y actuadores que se quiera controlar[20]. En este caso, el fabricante ha decidido de serie un microcontrolador con un solo canal de comunicaciones con los motores. Se trata de una tarjeta de código abierto, el OpenCR, descrito con detenimiento en el apartado de descripción de tecnologías.

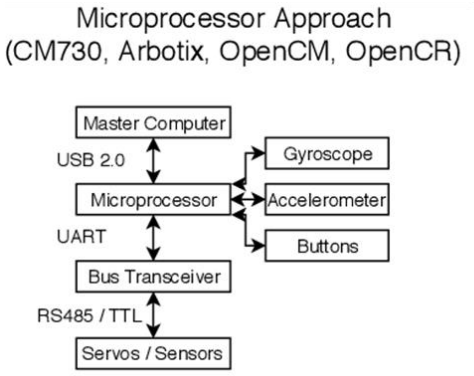

Figura 9 Esquema de comunicaciones entre dispositivos[20]

En el esquema podemos ver que el microcontrolador se comunica con la SBC por USB. Si el ordenador le pide un paquete, el microcontrolador le responde directamente si se trata de un sensor integrado en el mismo. En caso contrario, para aparatos externos como nuestros motores, recibe el dato por UART. Como nuestros motores están pensados para funcionar por TTL, existe un transceptor que transcribe los datos mediante un sencillo circuito integrado en la placa.

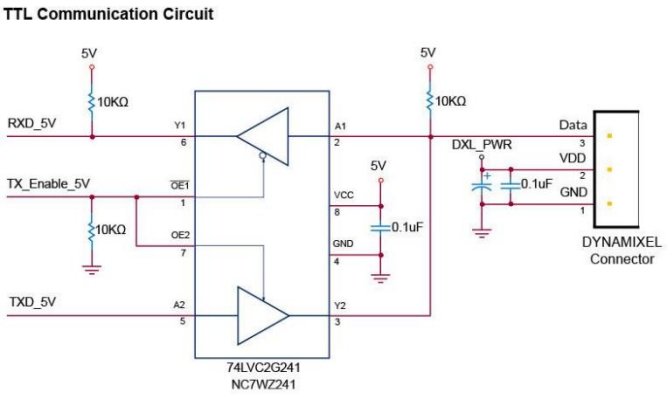

Figura 10 Esquema de comunicaciones TTL[18]

3.3 Implementación serie del fabricante

El fabricante incluye en la documentación oficial un código para controlar el robot desde ROS2. Anteriormente se ha visto que se va a utilizar un microcontrolador OpenCR y una Raspberry Pi 3B como SBC. Ambos elementos se comunican ente sí usando el protocolo Dynamixel 2.0 tal y como se muestra en la siguiente imagen.

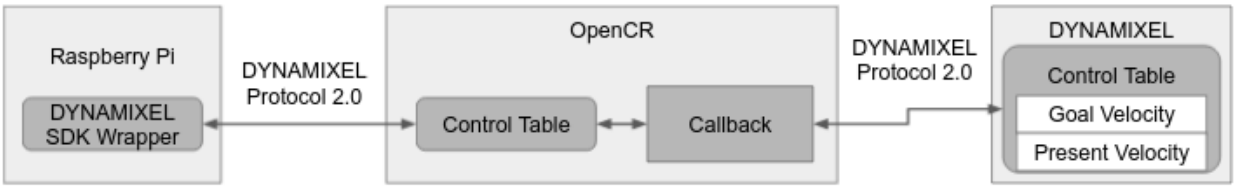

Figura 11 Estructura de comunicaciones[21]

Es importante resaltar que, entre el motor y el microcontrolador, se está obligado a usar este protocolo, pero entre el OpenCR y el SBC se pueden usar otros métodos. A continuación, se procede a explicar las estructura y objetivos del código de ambas placas.

3.3.1 SBC

En el ordenador del robot se está ejecutando ROS 2 Foxy Fitzroy. Como se puede ver en el siguiente diagrama, los datos de los sensores y actuadores del microcontrolador llegan a la placa, y son publicados por el nodo `turtlebot_node`. No se realiza ningún tipo de operación ni comprobación de límites con dichos datos antes de ser publicados.

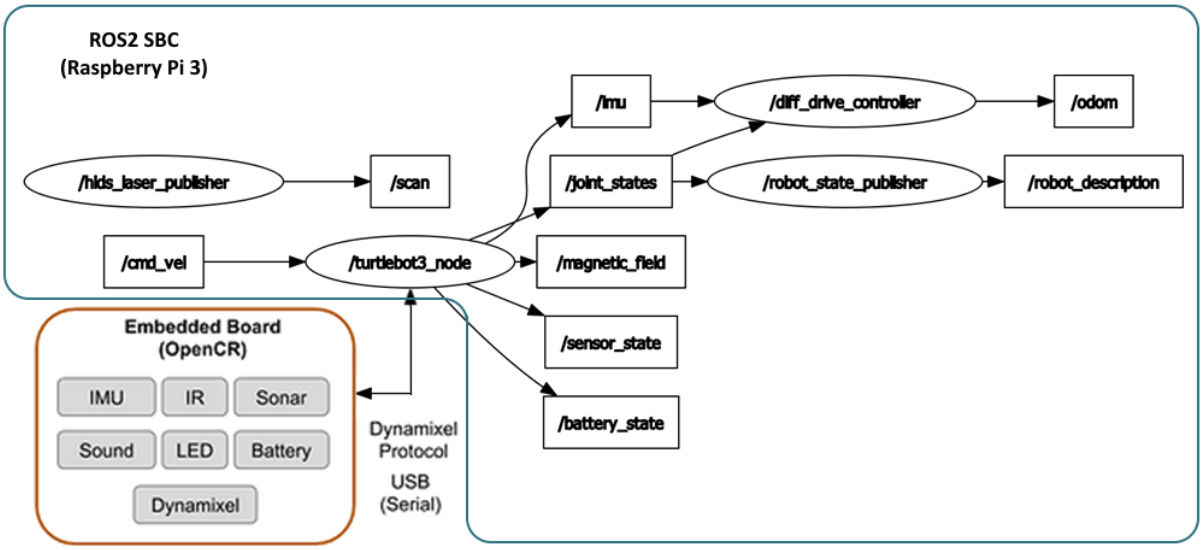

Figura 12 Mapa de comunicación entre ROS y OpenCR[22]

En el diagrama anterior se pueden ver una serie de *topics* o temas, marcados con un cuadrado, que representan la información que hay dentro de ROS2 para realizar los controles. A continuación, se muestra el tipo de mensaje que hay en cada tema y su función.

- `/battery_state`: Da información sobre el nivel de batería actual.

- `/imu`: Aceleración lineal y angular del robot.
- `/joint_states`: Velocidad angular y ángulo recorrido por cada motor.
- `/magnetic_field`: Orientación y fuerza del campo magnético en el espacio.
- `/odom`: Posición y orientación estimadas del robot.
- `/scan`: Distancia de los objetos alrededor del robot usando el LIDAR.
- `/cmd_vel`: Consigna de velocidad del robot. La Tabla 1 muestra la estructura de este.

Twist.msg	/ cmd_vel
`Vector3 linear`	Velocidad lineal [m/s]
`float64 x`	
`float64 y`	
`float64 z`	
`Vector3 angular`	Velocidad angular [rad/s]
`float64 x`	
`float64 y`	
`float64 z`	

Tabla 1 Estructura del topic /cmd_vel[23]

El robot viene por defecto con cuatro nodos, marcados con un círculo, que crean y consultan la información de los temas previamente descritos. Por la naturaleza de ROS, se pueden añadir más para interactuar con ellos según las necesidades del usuario. Los nodos principales cumplen las siguientes funciones:

- `/turtlebot3_node`: Envía la consigna de velocidad a los motores en `/cmd_vel` y publica la información de los sensores del microcontrolador.
- `/hids_laser_publisher`: Publica la información del LIDAR mediante `/scan`.
- `/diff_drive_controller`: Publica la posición estimada del robot en `/odom` mediante las medidas del `/imu` y el `/joint_states`.
- `/robot_state_publisher`: Publica el modelo urdf del robot en `/robot_description` si se pide el parámetro con un servicio.

3.3.2 OpenCR

El código del microcontrolador está escrito en C y consta de cinco archivos principales. A continuación, se va a explicar la función de cada uno de ellos.

3.3.2.1 ***turtlebot_controller***

Permite controlar el robot con el mando RC-100B fabricado por ROBOTIS. Para usarlo hay que conectar por el puerto de UART de la placa el módulo BT-410 y asegurarse de que el mando tiene colocado el módulo BT-100 para poder realizar la comunicación entre ambos.[24]

1.1.1.1 ***turtlebot_diagnosis***

Realiza todas las tareas que tengan que ver con el diagnóstico del robot y prevención de problemas del robot. Mide el nivel de la batería, en caso de que la batería este por debajo de voltage_ref_warn enciende un pitido para avisar al usuario y en caso de que la batería pase el valor critico de voltage_ref desactiva el robot para evitar una sobre descarga.

También ajusta los LED de la placa para informar al usuario sobre su correcto funcionamiento. A continuación, se muestra un diagrama del mensaje que simboliza cada led.

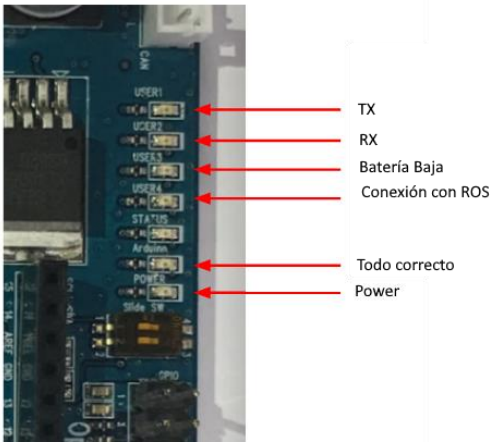

Figura 13 Indicadores LED de la OpenCR.

Por último, comprueba si se ha pulsado el pulsador más cercano al conector micro USB para realizar una prueba de motores.

3.3.2.2 ***turtlebot_motor_driver***

Se comunica con los motores, habilita el par, lee la posición, velocidad, corriente y perfil de aceleración de cada motor y escribe la velocidad y perfil de aceleración deseados en cada motor.

3.3.2.3 ***turtlebot_sensor***

Realiza las medidas de los sensores. Además, muestra la dirección de avance mediante los pines GPIO 4,6,8,10.

3.3.2.4 ***turtlebot***

Función principal del código, realiza la comunicación tanto con el motor como con el SBC a través de una tabla de control y el protocolo Dynamixel 2.0.

3.4 Firmware modificado

Como se ha visto, el robot está diseñado para funcionar estableciendo la consigna de velocidad del motor. Esta se manda desde ROS por el comando `/cmd_vel` y después es mandado al OpenCR, que lo escribe en la tabla de control.

Sin embargo, en este proyecto se prefiere, controlar el robot mandando directamente el valor de PWM para cada motor. Por ello, tenemos que poner los motores en el modo de control PWM. En ese modo, el motor solo puede recibir consignas de tensión, por lo que no se podrán usar consignas de velocidad por defecto a la vez. Las consignas de velocidad se darán a nuestro control usando ROS. De esta forma, la comunicación de los motores desde ROS se hará con el comando `/cmd_vel`, de forma que:

```
msg->linear.x = Tensión común
msg->angular.z = Tensión diferencial
```

Este mensaje actualmente puede resultar contraintuitivo, ya que el formato Twist está pensado para velocidades y aceleraciones. Un posible trabajo futuro puede ser migrar a un mensaje propio, en este mensaje además se podría añadir un *timestamp* para tener constancia del momento en el cual se genera e

Como se ha explicado en el apartado anterior, la comunicación con los motores se da en el archivo `turtlebot_motor_driver.c`. A continuación, se muestran los cambios en dicho archivo.

3.4.1 OpenCR

La consigna de tensión que usamos en ROS ha de introducirse en el motor cambiando el valor de GOAL PWM[25] en la tabla de control en RAM. Este valor se ve limitado por el valor PWM Limit[26] en la tabla de control en la memoria EEPROM.

Este valor está limitado entre 885 para el 100% de PWM y 0 para 0% de PWM con una escala lineal con una unidad de 8.85[1/%]. Para no mandar valores demasiado altos al motor lo vamos a limitar también desde el código usando la constante `LIMIT_X_MAX_PWM`. Por otro lado, también tenemos la constante `PWM_CONSTANT_VALUE` para cambiar de unidad en porcentaje a las unidades internas del motor.

```
25  const uint16_t LIMIT_X_MAX_PWM = 885;
...
31  const float PWM_CONSTANT_VALUE = 8.85;
```

Código 1 Escala PWM OpenCR.

Por otra parte, usamos la función `control_motors` para combinar las consignas de tensión común y tensión diferencial para mandarlas a los motores con la función `write_pwm`. Para mandarlo, la convertimos a las unidades de la tabla de control y la limitamos con `LIMIT_X_MAX_PWM`.

```
244 bool Turtlebot3MotorDriver::control_motors(const float wheel_separation,
float linear_value, float angular_value)
```

```
245 {
246   bool dxl_comm_result = false;
247
248   // float wheel_velocity[MortorLocation::MOTOR_NUM_MAX];
249   float wheel_pwm[MortorLocation::MOTOR_NUM_MAX];
250   float lin_vel = linear_value;
251   float ang_vel = angular_value;
252
253   wheel_pwm[LEFT] = lin_vel - (ang_vel);
254   wheel_pwm[RIGHT] = lin_vel + (ang_vel);
255
256   wheel_pwm[LEFT] = constrain(wheel_pwm[LEFT] * PWM_CONSTANT_VALUE, -
LIMIT_X_MAX_PWM, LIMIT_X_MAX_PWM);
257   wheel_pwm[RIGHT] = constrain(wheel_pwm[RIGHT] * PWM_CONSTANT_VALUE, -
LIMIT_X_MAX_PWM, LIMIT_X_MAX_PWM);
258
259   dxl_comm_result = write_pwm((int16_t)wheel_pwm[LEFT],
(int16_t)wheel_pwm[RIGHT]);
260   if (dxl_comm_result == false)
261     return false;
262
263   return true;
264 }
```

Código 2 Turtlebot3MotorDriver::control_motors

La función `write_pwm`, finalmente manda los valores de consigna de tensión a los motores.

```
210 bool Turtlebot3MotorDriver::write_pwm(int16_t left_value, int16_t
right_value)
211 {
212   bool ret = false;
213
214   sync_write_param.addr = 100;
215   sync_write_param.length = 2;
216   memcpy(sync_write_param.xel[LEFT].data, &left_value,
sync_write_param.length);
217   memcpy(sync_write_param.xel[RIGHT].data, &right_value,
sync_write_param.length);
218
219   if (dxl.syncWrite(sync_write_param))
220   {
221     ret = true;
222   }
223
224   return ret;
225 }
```

Código 3 Turtlebot3MotorDriver::write_pwm

Por otra parte, también se han hecho cambios en el archivo `turtlebot.c`. En este archivo se realiza la comunicación con el nodo ROS2 a través de la función `dxl_slave_write_callback_func`. El valor se manda a la placa OpenCR como un numero entero sin decimales. El número que se recibe es necesario dividirlo por 0.01 ya que el número ha sido previamente multiplicado por 100 para poder

mandar el valor con dos decimales de precisión en la función `TurtleBot3::cmd_vel_callback()` en el SBC. Este valor, además, se limita con las variables `max_linear_velocity` y `max_angular_velocity`.

```
544 static void dxl_slave_write_callback_func(uint16_t item_addr, uint8_t
&dxl_err_code, void *arg)
545 {
546   (void)arg;
547
548   switch (item_addr)
549   {
…
572   case ADDR_CMD_VEL_LINEAR_X:
573     goal_velocity_from_cmd[VelocityType::LINEAR] = constrain((float)
(control_items.cmd_vel_linear[0] * 0.01f), min_linear_velocity,
max_linear_velocity);
574     break;
575
576   case ADDR_CMD_VEL_ANGULAR_Z:
577     goal_velocity_from_cmd[VelocityType::ANGULAR] = constrain((float)
(control_items.cmd_vel_angular[2] * 0.01f), min_angular_velocity,
max_angular_velocity);
578     break;
…
585   }
586 }
```

Código 4 dxl_slave_write_callback_func

También se ha cambiado la función que prueba los motores, `test_motors_with_buttons`, para funcionar con PWM ya que resulta útil para diagnosticar problemas de conexión de los motores con el microcontrolador.

3.4.2 SBC

En el SBC el único cambio significativo es el cambio de la frecuencia de muestreo a 20 Hz a 100Hz. De esta forma, se consigue publicar cinco veces más muestras por segundo en los topics generados por el código implementado por el fabricante.

```
void TurtleBot3::run()
{
  RCLCPP_INFO(this->get_logger(), "Run!");

  publish_timer(std::chrono::milliseconds(10)); //Cambiado de 50ms
  heartbeat_timer(std::chrono::milliseconds(100));

  parameter_event_callback();
  cmd_vel_callback();
}
```

Código 5 TurtleBot3::run

4

Modelado de la planta del motor para el control de velocidad

En esta sección se desarrolla un modelo basado en los principios físicos del movimiento del motor, incluyendo el motor Dynamixel XL430-W250-T. Para ello se usan las especificaciones de la ficha técnica del motor, además de una serie de ensayos en lazo abierto y cerrado.

4.1 Modelo matemático[27]

Un requisito fundamental que ha de tratarse a la hora de diseñar un regulador o planta es crear un modelo matemático para simular la dinámica del sistema a controlar. Este modelo ha de representar la relación entre la variable de mando (el valor PWM) y la variable de salida (la velocidad de rotación de avance del robot).

El actuador que se usa para las ruedas es un Dynamixel XL430-W250-T, un servomotor. Contiene un motor de corriente continua con núcleo con un controlador integrado de posición absoluta de 12 bits. El modelo concreto de motor DC que se ha usado no es especificado por el fabricante, por lo que no podremos crear un modelo numérico del sistema, aunque nos ayudará a entender las variables que afectan al control y el tipo de respuesta que vamos a encontrarnos.

En el artículo[28] se ha modelado la dinámica del Dynamixel MX-28AT, un modelo superior del mismo fabricante. Ambos modelos de motor son muy similares, y tienen un funcionamiento parecido. Ambos usan el mismo sensor de posición, pero difieren en algunos parámetros claves de la dinámica, al tener un modelo de motor diferente. Se trata de el mismo tipo de motor, un motor sin núcleo y con bornas de corriente continua, lo que nos indica que el tipo de respuesta a esperar es igual. Por tanto, se han incluido sus resultados matemáticos para ilustrar el orden de la función de transferencia que nos vamos a encontrar.

En la publicación, el autor llega a la siguiente función de transferencia entre la entrada (el voltaje en las bornas del motor), y la salida (el ángulo de salida del eje de la carga, θ_l),

$$\frac{\theta_l(s)}{U(s)} = \frac{N\,\eta\,K_t}{s\left\{L(J_l + J_m N^2\,\eta)s^2 + [R(J_l + J_m N^2\,\eta) + L\,b_m N^2\,\eta]s + R\,b_m N^2\,\eta + \frac{K_t N^2\,\eta}{K_\omega}\right\}}$$

Ecuación 9 Función de transferencia ϑ_l (s)/(U(s)

donde L representa la inductancia de la bobina de la armadura y R la resistencia, N la relación de transmisión, η la eficiencia de los engranajes, K_ω la constante de velocidad, K_t la constante de par, J_m la inercia y b_m la fricción.

Podemos ver que esta función de transferencia no es la que estamos buscando, ya que las variables de entrada y salida son diferentes. Por suerte, es fácil realizar una transformación a la ecuación de transferencia buscada usando las relaciones que se detallan continuación.

La entrada de nuestro control es dada como una señal de ancho modulado. El fabricante no especifica la frecuencia a la que funciona el PWM, pero se supone suficientemente alta para que el motor vea un voltaje constante siendo este un porcentaje del voltaje de operación Us (12V).

$$U = Us * PWM$$

Por otra parte, el ángulo θ_l representa la posición angular de la rueda en cada momento. Se puede aplicar la derivada de dicho ángulo para conseguir la velocidad angular y por último multiplicar esta por el radio de la rueda para conseguir la velocidad de avance. Siempre que se suponga que hay buen agarre entre la rueda y el suelo, y por tanto no hay deslizamiento.

$$\Omega l = \theta l * s \qquad v = \omega l * r_R$$

Realizando las transformaciones indicadas nos queda la siguiente función de transferencia.

$$\frac{v(s)}{PWM(s)} = \frac{N\,\eta\,K_t U_s}{r_R L(J_l + J_m N^2\,\eta)s^2 + r_R[R(J_l + J_m N^2\,\eta) + L\,b_m N^2\,\eta]s + r_R R\,b_m N^2\,\eta + \frac{r_R K_t N^2\,\eta}{K_\omega}}$$

Ecuación 10 Función de transferencia v/pwm

Finalmente, vemos que nos queda un sistema de segundo orden sin polos o ceros adicionales. Ello nos ayudará a encontrar más fácilmente los valores numéricos de la función de transferencia.

4.2 Ganancia de la planta

El primer lugar, vamos a modelar la ganancia de la planta. Según hemos visto en el modelado numérico, el modelo es lineal ya que lo hemos podido expresar con una función de transferencia.

Para comprobarlo, se ha enviado una señal triangular de PWM de rango ±100% con la intención de ver cómo aumenta la velocidad según se aumenta la tensión. El periodo de esta onda es de 20 Hz, de

esta forma, el motor tiene tiempo de variar su velocidad sin sufrir grandes aceleraciones que puedan alterar el ensayo.

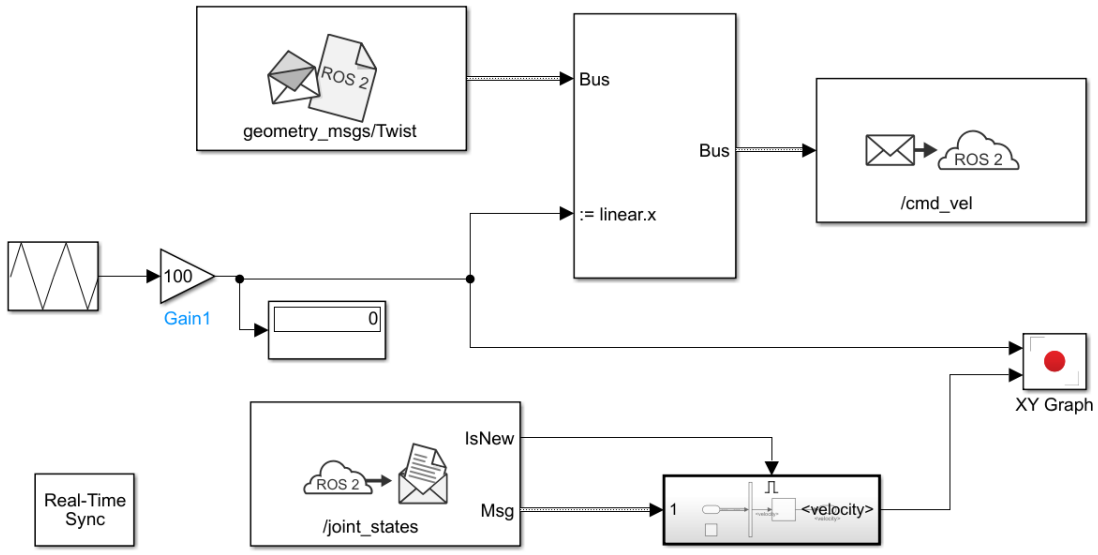

Figura 14 Modelo del ensayo de ganancia.

Al ser lineal, esperamos ver al dibujar la gráfica entre la tensión y velocidad lineal una pendiente constante, ya que la ganancia la planta es constante. En la siguiente gráfica se puede ver el resultado del ensayo.

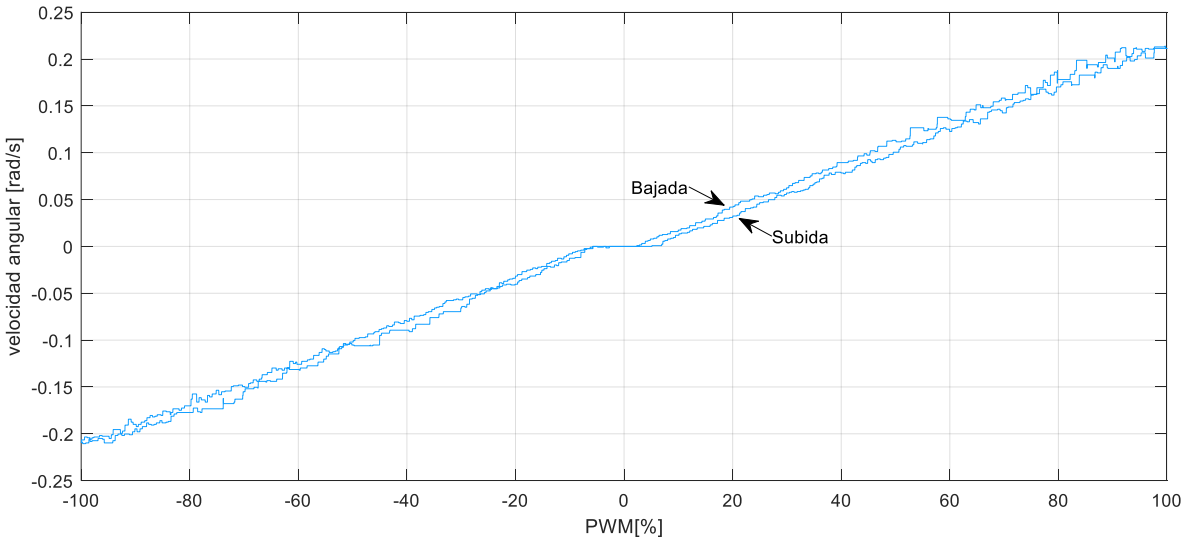

Figura 15 Resultado de un periodo de una onda triangular en la tensión al motor

Se puede ver un carácter muy lineal en la ganancia, como se esperaba. Por otra parte, también se puede observar que en la zona cercana al cero existe una zona donde la tensión es constante. Esto se debe a que antes de poder empezar a moverse hay que sobrepasar la fricción estática. Se puede ver

un efecto similar en los extremos, cerca del 95% empieza a disminuir la pendiente de la recta, esto se debe a que a medida que aumentamos la velocidad disminuimos el par que puede dar el motor, y para torques bajos disminuye la eficiencia del motor tal y como se muestra en la siguiente gráfica.

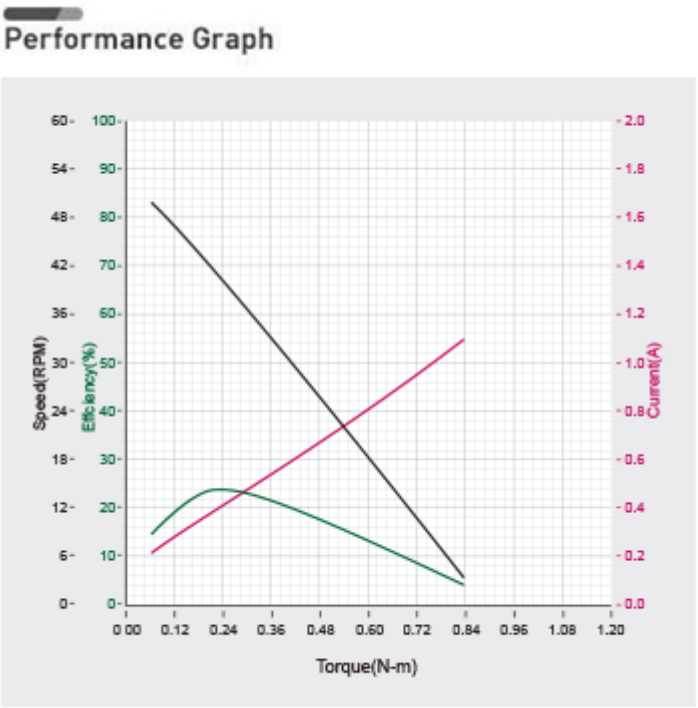

Figura 16 Gráfica de características del motor[1].

Por último, podemos ver una cierta histéresis entre la pendiente de subida y la de bajada. Esto puede deberse a la fricción que sufre el motor. Al acelerar desde parada, este ha de vencer a la fricción estática, y al desacelerar, este se parará cuando la fricción dinámica gane al torque del motor. La fricción dinámica siempre es menor que la estática, esto produce la histéresis que estamos viendo.

Para conseguir finalmente el valor de ganancia estática entre la velocidad de avance y la tensión PWM se va a hacer una aproximación lineal de la recta vista en la Figura 15 Resultado de un periodo de una onda triangular en la tensión al motoren las zonas rectas. Para poder filtrar el ruido de la medida se han hecho múltiples periodos de la onda triangular. De esta forma se obtiene la siguiente gráfica.

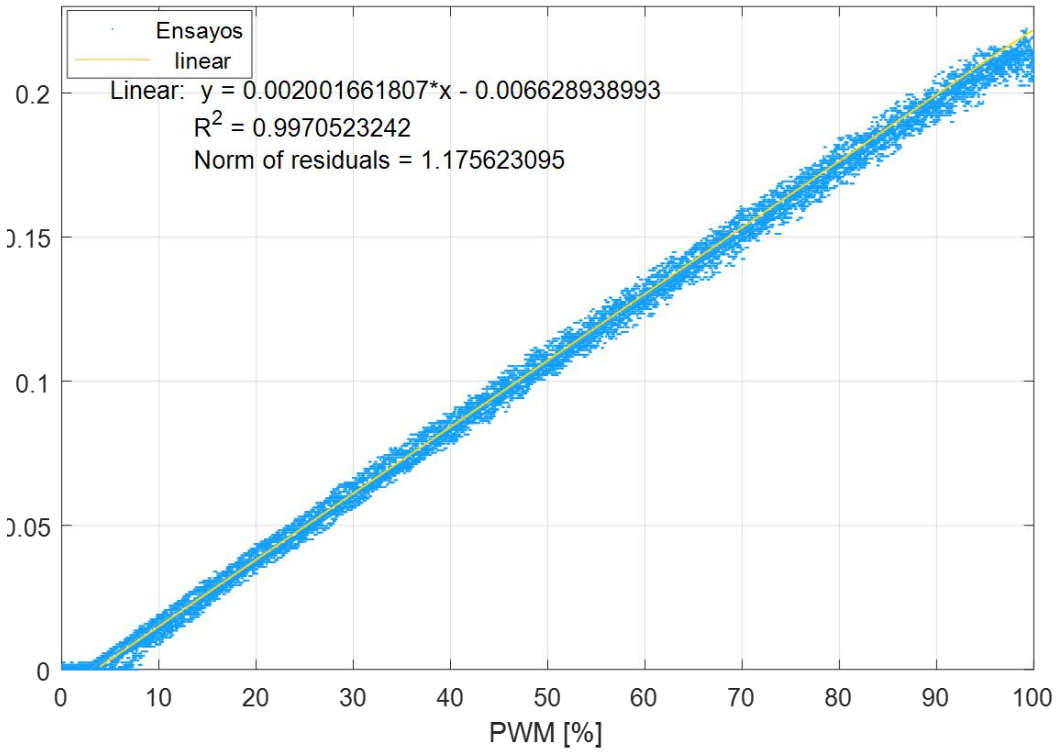

Figura 17 Ensayo onda triangular semiplano positivo.

Se consigue finalmente una pendiente de 0.0020016[%/(m/s)]. La aproximación es muy buena, ya que el coeficiente de determinación(R^2) del 99.70%.

La planta del motor que estamos modelando es entre la velocidad de avance del robot y la tensión PWM, pero una ganancia del control tan baja nos obligará a usar controles de lazo cerrado con ganancias muy altas. Por ello, es conveniente realizar el lazo entre la velocidad angular y PWM. Se divide por tanto la velocidad lineal por el radio de la rueda para conseguir nuestra ganancia de 0.0661 respecto a la velocidad angular.

Por otra parte, se tomará como nulo cualquier mando menor del 5%, ya que se considerará que no conseguirá sobrepasar a la fricción estática.

4.3 Determinación de los polos

Para modelar la dinámica de la planta se ha ajustado un control proporcional en lazo cerrado. Dicho control ha de tener una ganancia suficientemente alta para que tengamos un sobrepaso sustancial. De esta forma podemos modelar fácilmente el sistema usando los valores tabulados para sistemas de segundo orden[16].

4.3.1 Ensayo en ROS

A continuación, se muestra en control en Simulink que se va a desplegar usando ROS.

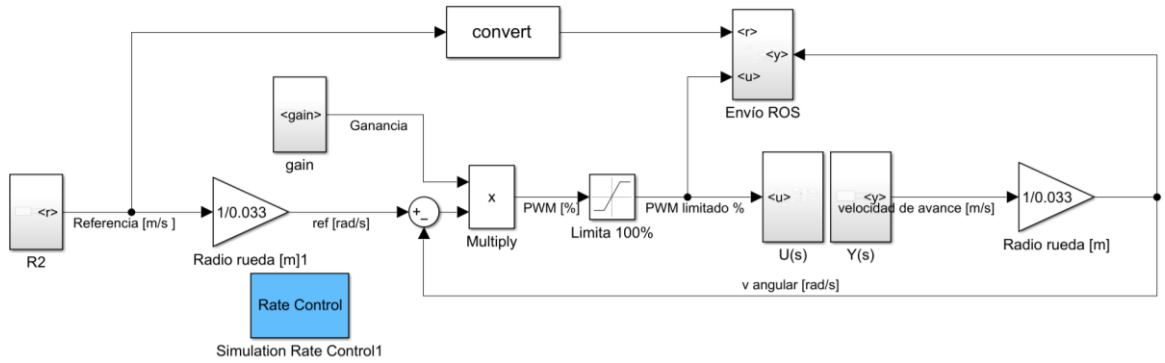

Figura 18 Ensayo proporcional en lazo cerrado.

El bloque U representa el *publisher* del mando en el topic `/cmd_vel`, el bloque Y, el *subscriber* al *topic* `/joint_states` y el bloque R un *subscriber* al topic `/ref`, donde se publicará la velocidad lineal deseada.

Por otro lado, se ha creado el bloque Envío ROS donde hay un *publisher* del topic `/signal`, que contiene las tres señales y una marca de tiempo. La frecuencia de envío de datos de este bloque es igual a la frecuencia del control, 100 Hz. Esta frecuencia es mayor que los 20Hz del topic /joint_states por lo que se verá una retención en las muestras de la salida.

Se han realizado varios ensayos para describir la planta. A continuación, se muestra un ensayo con un escalón de referencia 0.132 m/s, es decir 4 rad/s y ganancia del control (K = 12.3).

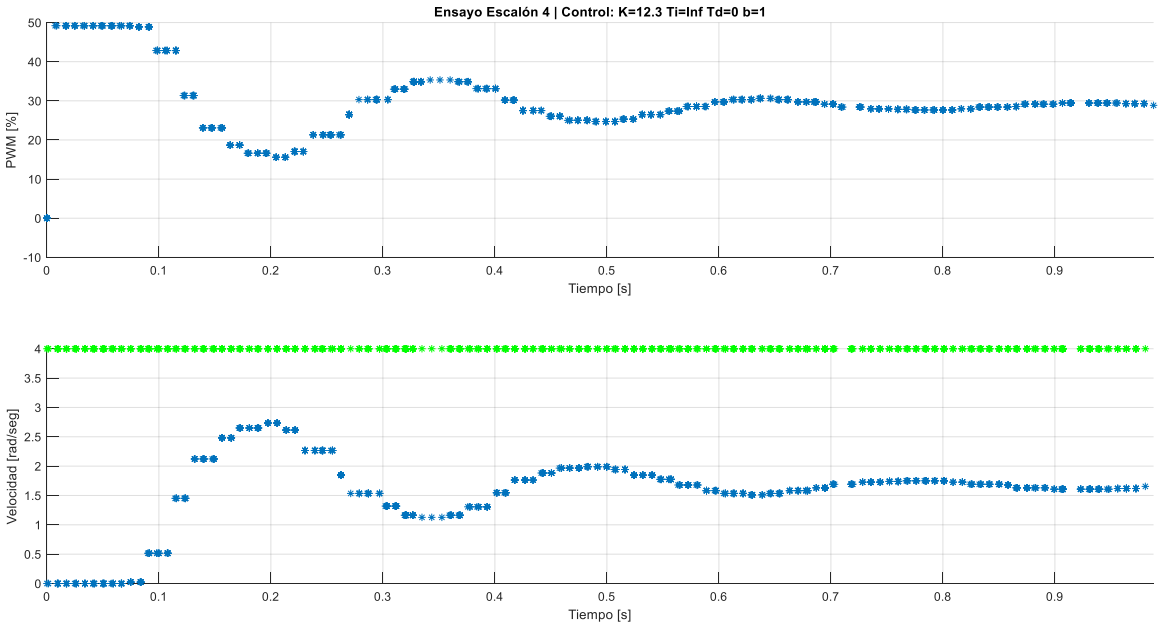

Figura 19 Ensayo control P

Se puede ver que hay un retraso en la respuesta del control respecto a la referencia. Se podría modelar esta característica con un retraso en la planta. Para asegurarnos del valor de este retraso se ha repetido muchas veces el ensayo, para comprobar si hay una gran variabilidad entre ensayos.

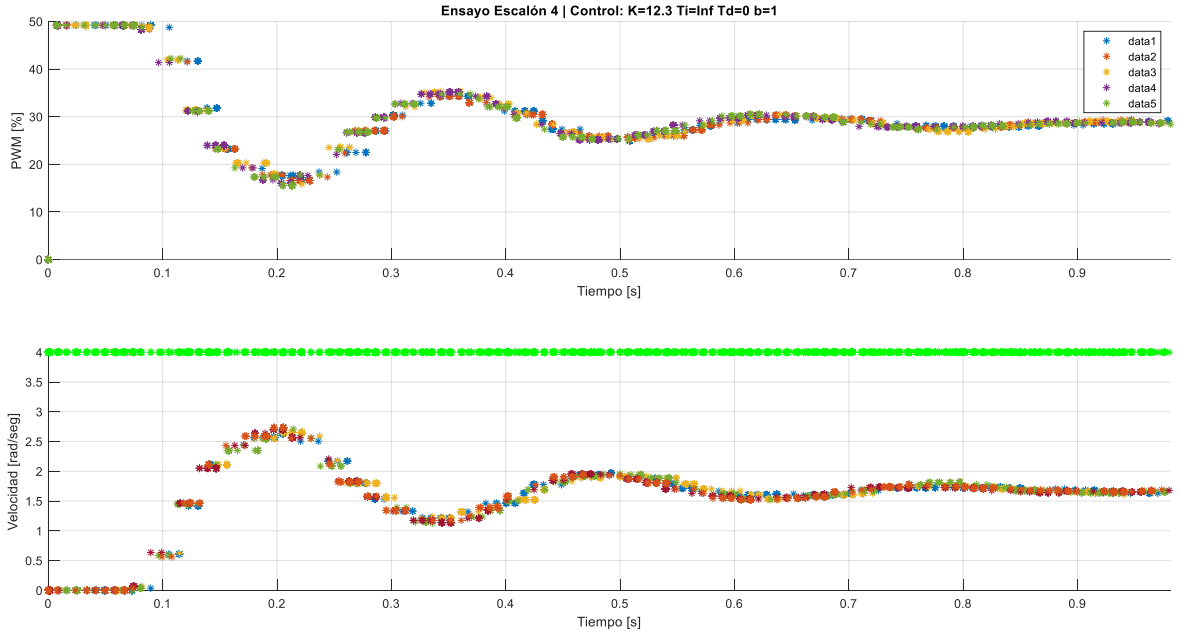

Figura 20 Múltiples ensayos control P

Al repetir el ensayo se puede ver que el periodo de muestreo es bajo en comparación con la velocidad de la respuesta. Por otra parte, se puede ver que la variabilidad entre ensayos es baja., aunque existe una mayor variabilidad en los primeros ensayos después de reiniciar el equipo.

Como se ha descrito en el capítulo 3, la comunicación entre el motor y la Raspberry se hace por el protocolo Dynamixel 2.0. Primero el motor se comunica con el microcontrolador, y después con la SBC. Este protocolo ha de ser el principal factor para contribuir al retraso que se aprecia, ya que la transmisión de mensajes en ROS2 se puede considerar instantánea en una red local[30].

Para intentar disminuir la incertidumbre y modelar el retraso producido por la comunicación del motor se ha vuelto a realizar el control, esta vez desde Matlab.

4.3.2 Ensayo en Matlab

Para comprobar la hipótesis de la causa del retraso en ROS, se ha realizado la conexión con el motor directamente desde Matlab, sin usar ROS2. De esta forma se espera emular la comunicación entre el motor y el microcontrolador. La interfaz para conectar el motor con Matlab es la placa U2D2, aunque también se podría hacer cambiando el *firmware* de la placa OpenCR.

Para realizar los ensayos se ha programado una librería para facilitar en control de los motores desde Matlab. Dicha librería se puede encontrar en el Apéndice M.

Utilizando esta librería se han repetido los ensayos para escalones de referencia de 0.1515 m/s, es decir 5 rad/s y ganancia del control (K = 10).

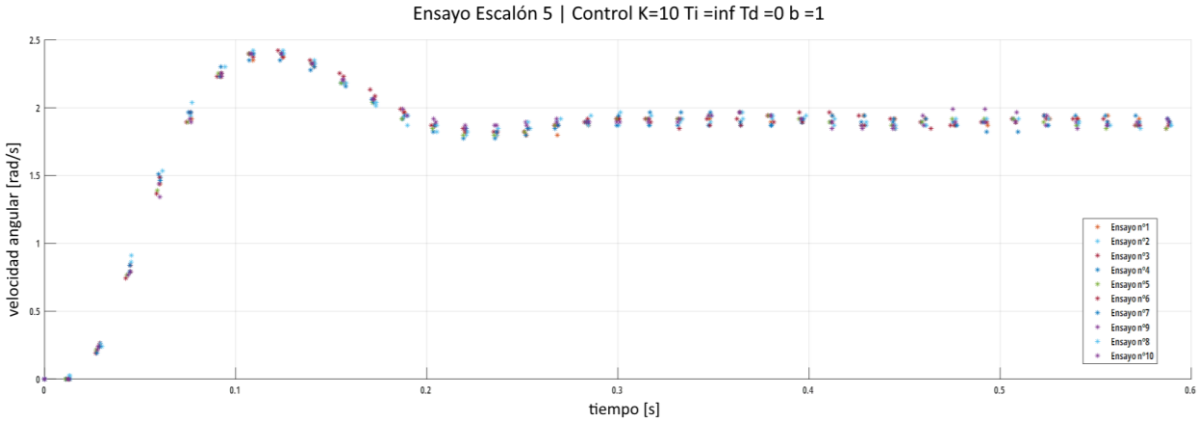

Figura 21 Ensayo control proporcional de velocidad en MATLAB.

En este caso se puede ver mucho mejor la forma de la respuesta, ya que vemos mucha menos variabilidad entre ensayos y un periodo de muestreo muy similar. Por otro lado, podemos ver un retraso de 13 ms en la respuesta. Durante el tiempo entre muestras, se reciben los datos de salida, se calcula el mando y se registran los datos. Para registrar el tiempo que se emplea para cada operación se ha usado el Matlab Time Profiler.

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
Ensayo_main_LA	1	5.265	2.038	
ParamRead>ParamRead.read	104	1.338	0.013	
groupSyncReadTxRxPacket	104	1.312	1.312	
Ensayo_header	1	1.108	0.014	
loadlibrary	1	0.787	0.078	
dos	3	0.626	0.626	
plot_results	1	0.378	0.119	
ParamWrite>ParamWrite.write	104	0.331	0.083	
ADDR_TABLE	111	0.270	0.270	
perl	1	0.198	0.006	
loadlibrary>getLoadlibraryCompilerConfiguration/thunkBuilder	1	0.185	0.001	
setBaudRate	1	0.146	0.146	
legend	1	0.141	0.003	
legend>make_legend	1	0.136	0.004	
Ensayo_footer	1	0.123	0.004	
closePort	1	0.117	0.117	
newplotwrapper	3	0.097	0.001	
newplot	3	0.096	0.013	
Legend.Legend>Legend.Legend	1	0.053	0.008	

Figura 22 MATLAB Time Profiler.

Analizando el tiempo usado en cada función se ha podido ver que el motor tarda 12.86 milisegundos en enviar la muestra en la operación *groupSyncReadTxRxPacket* y 0.15 milisegundos en enviar el mando. Por tanto, juntando el tiempo de envío del mando y el tiempo de lectura de la salida se consigue el retraso de 13ms que hemos observado.

4.3.3 Simulación en MATLAB

La dinámica del control se ha modelado con el siguiente modelo en Simulink.

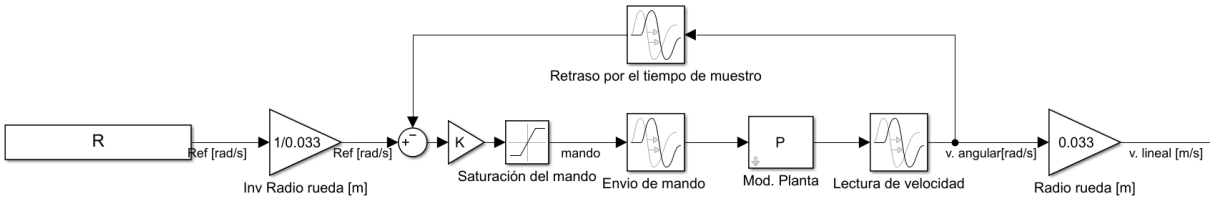

Figura 23 Modelo de la Simulación del control de velocidad en MATLAB

Se ha incluido un retardo en la realimentación, ya que en la Figura 21 Ensayo control proporcional de velocidad en MATLAB se puede ver un periodo de muestreo pequeño para con la velocidad de la respuesta que obtenemos. En nuestro lazo se ha usado el modelo analógico modificado, aunque no se cometería un gran error si se usa el modelo analógico puro sin retardo.

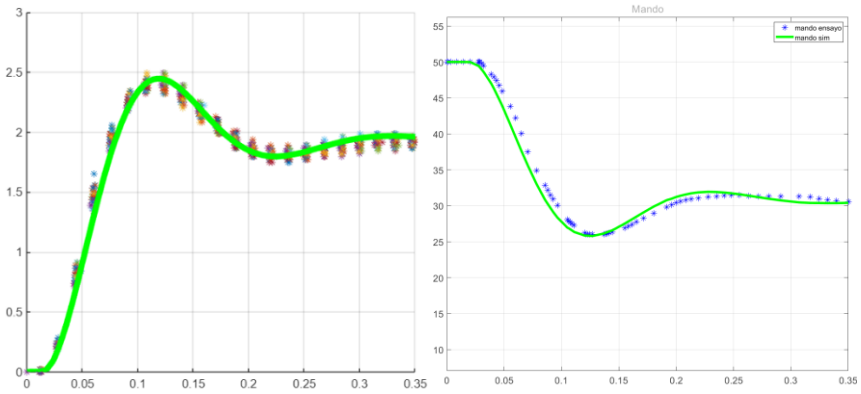

Figura 24 Comparación ensayo simulación MATLAB

Se puede ver en las gráficas anteriores que el modelo del que hemos partido se ajusta muy bien a los ensayos, por lo que podemos concluir que el modelado es correcto.

$$P(s) = \frac{v(s)}{PWM(s)} = \frac{0.065}{\left(\frac{s}{24}+1\right)\left(\frac{s}{57}+1\right)}$$

4.3.4 Simulación en ROS

Con la información sobre la planta del sistema adquirida en los apartados anteriores se va a retomar el modelado de la planta en ROS, que es el objetivo principal de este capítulo. Nuestra planta modela solo la dinámica interna del motor, por lo que se puede usar también para modelar la dinámica de los ensayos en ROS, pero en este caso, para modelar el lazo habrá que hacer algunos ajustes para tener en cuenta las comunicaciones.

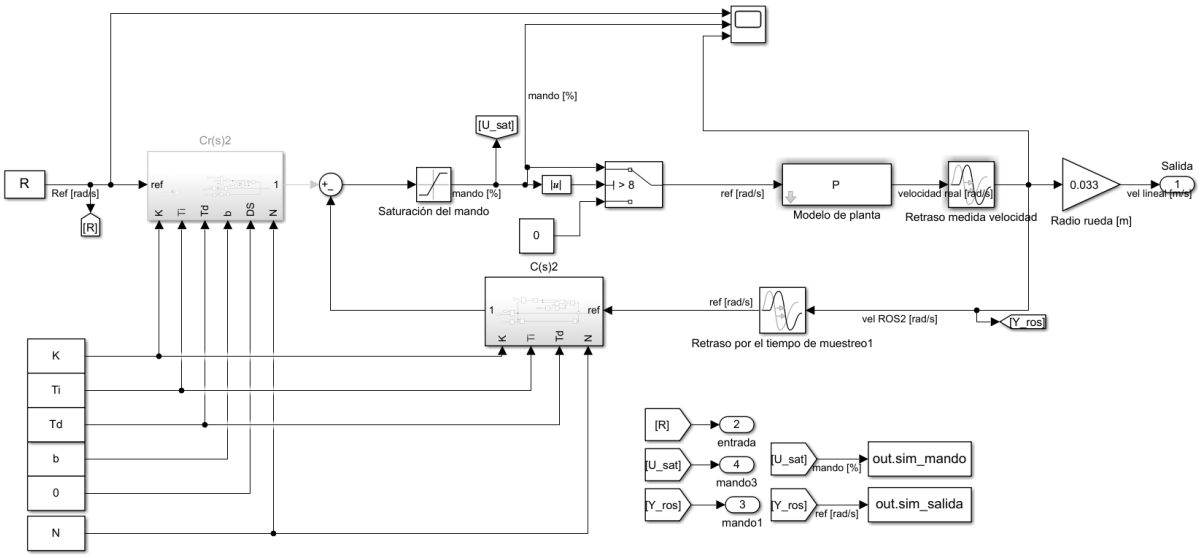

Figura 25 Ensayo del control en MATLAB.

En primer lugar, se ha añadido un retraso de la mitad del periodo de muestreo del control (100Hz) para crear un modelo analógico modificado. Por otra parte, se ha modelado el retraso de la planta con un bloque de *Trasport Delay* de 90ms.

Para comprobar que los polos de la planta modelados en MATLAB representan realmente al sistema, se han usado estos mismos polos para una simulación del control en ROS.

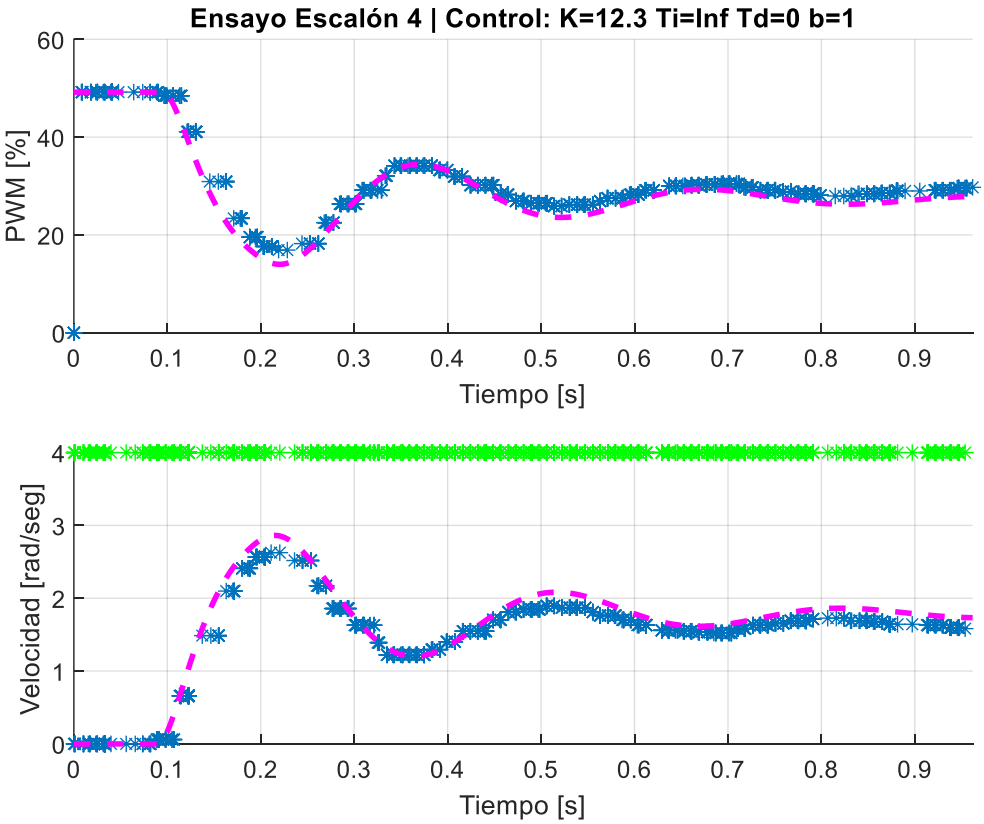

Figura 26 Comparación simulación ensayo control velocidad ROS2

Se puede ver que el ajuste es bastante bueno, por lo que se usará para todas las simulaciones futuras y el diseño de controles más complejos. Existe una pequeña diferencia en la ganancia estática, pero se puede atribuir a la disminución del voltaje de alimentación a los motores al descargarse la batería.

4.4 Modelo analógico modificado

Por otro lado, estudiamos si finalmente es necesario realizar el modelo analógico modificado. Realizamos una gráfica de Black de G(s) para ver la pulsación de cruce.

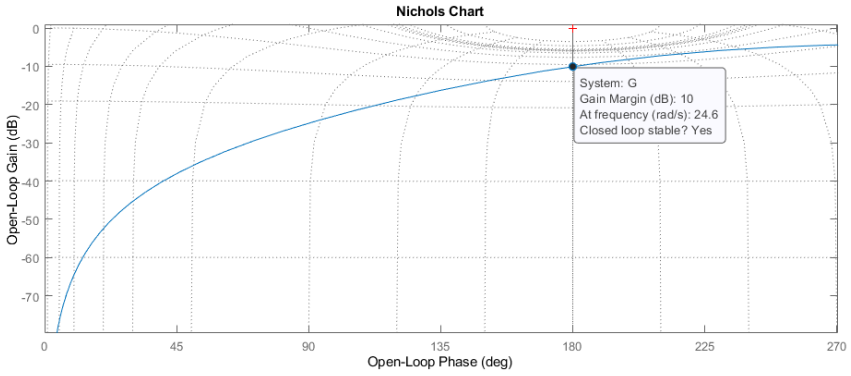

Figura 27 Gráfica de Black de control proporcional de velocidad.

Se puede ver que nuestro control tiene la pulsación de cruce en 24.6 rad/s, por lo que multiplicada por nuestro periodo de muestreo nos da un valor de woTs cercano a 0.4.

T_s	Recomendación	Muy pequeño	Pequeño	Mediano	Grande	Muy grande
$\omega_o T_s$	0.1 a 0.5	0.1	0.2	0.5	1	2
$\varphi_s = \frac{\omega_o T_s}{2}\frac{180^\circ}{\pi}$	5 a 15	2.9	5.7	14.3	28.6	57.3
$\frac{\omega_s}{\omega_o} = \frac{2\pi}{\omega_o T_s}$	10 a 30	62.8	31.4	12.6	6.3	3.1

Figura 28 Tabla de tiempos de muestreo[16]

Por tanto, podemos considerar que tenemos un periodo de muestreo mediano, por lo que es recomendable mantener el retraso en la realimentación para obtener un modelo más aproximado para implantación digital.

5

Ajuste del control de velocidad

Se diseñarán reguladores de tipo PID por orden creciente de complejidad: P, PI, PD, PID.

En el capítulo anterior se ha llegado a la conclusión de que la planta del control se puede representar mediante la siguiente función de transferencia.

$$P = \frac{88.92}{(s+57)(s+24)} * e^{-0.09s}$$

Ecuación 11 Planta del control de velocidad.

Aplicando una aproximación de Padé de primer orden para eliminar la componente exponencial:

$$P_{pade}(s) = \frac{-88.92\ (s-22.22)}{(s+57)(s+24)(s+22.22)}$$

Ecuación 12 Aproximación de Padé de la planta de velocidad.

Se puede ver que el polo de -57 está bastante alejado de los otros dos, por lo cual la respuesta la dictarán sobre todo el polo en -24 y el polo del retraso. Por otra parte, el valor del cero comparado con los polos nos indica que influye en gran medida en las dinámicas de la planta.

Por tanto, la respuesta en lazo cerrado será:

$$F(s) = \frac{C * P_{pade}}{1 + C * P_{pade} * H(s)}$$

Ecuación 13 Respuesta en lazo cerrado del control de velocidad

Donde C representa el control PID que usemos en lazo cerrado. Esta aproximación no es perfecta, aunque su exactitud aumentaría según se sube el grado de aproximación de Padé. Para ilustrarlo, se ha repetido la simulación con los mismos parámetros que en la identificación de la planta.

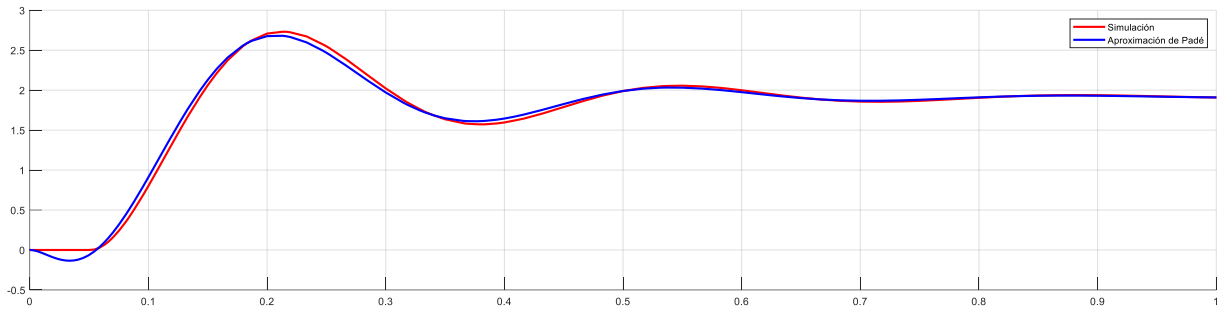

Figura 29 Comparación aproximación de Padé

Se puede ver que la aproximación es suficientemente buena para el diseño de controles, ya que tenemos un amortiguamiento y velocidad parecidos. La diferencia principal se da en los primeros instantes, donde los ceros que se añaden provocan una bajada inicial de la respuesta.

5.1 Reguladores PID

En el capítulo 2 se ha explicado brevemente el funcionamiento de un control PID y sus variantes de diseño. En este caso se va a usar la formulación no interactiva. Ya que, aunque la interactiva resulta más cómoda para hacer un diseño del control por respuesta en frecuencia, las limitaciones expuestas en la sección 6.5 obligan a usar esta configuración. Se ha decidido hacer este tipo de diseño, ya que, como hemos visto en la ecuación de respuesta en lazo cerrado Ecuación 12, la planta es de tercer orden. Además, tiene un cero adicional. De esta forma, el orden total de la respuesta será tal que no se podrán usar valores fácilmente tabulados para prever su respuesta.

Al hacer un estudio de la respuesta en frecuencia de la planta con una gráfica de Black obtenemos la siguiente figura:

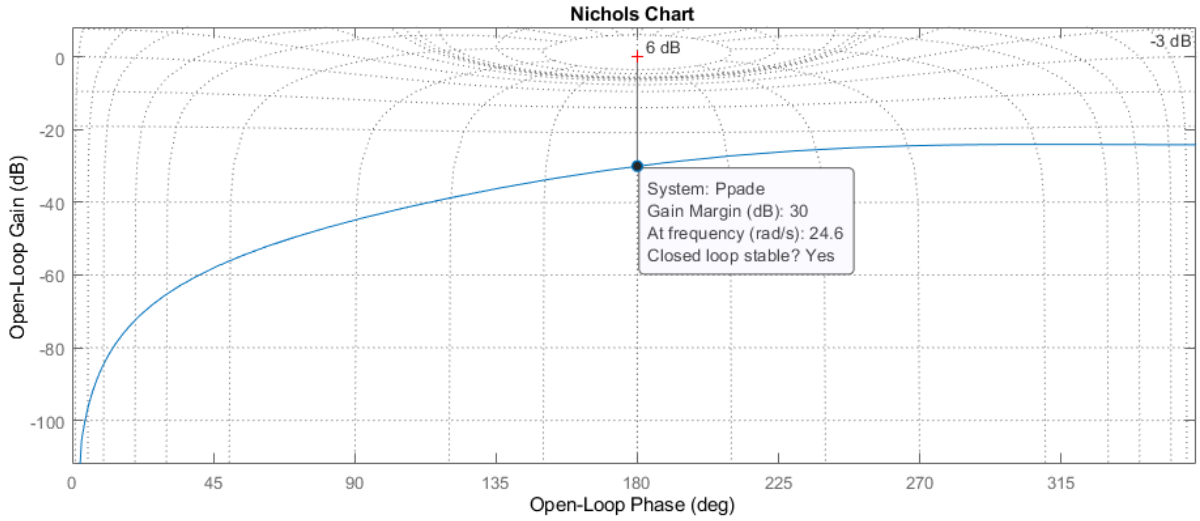

Figura 30 Gráfica de Black de la planta del control de velocidad.

Se puede ver que la planta (lazo abierto con K=1) tiene un margen de ganancia de 30dB con una frecuencia de oscilación de 24.6 rad/s, mostrándonos tal y como hemos visto que el sistema es estable. Por otro lado, se puede ver que el sistema no tiene margen de fase ya que la ganancia es tal que no llega nunca al valor de 0dB. Esto se debe a que el retraso de la planta produce que el gráfico de Black sea muy horizontal. Por ello, se ha decidido usar como especificación para diseñar los controles el margen de ganancia.

5.1.1 Regulador proporcional (P)

En un control proporcional solo tenemos un grado de libertad, ya que el único parámetro a variar es Kp. Al variar el valor de este podemos alterar el margen de ganancia del sistema el alzo abierto. De esta forma, si incluimos una Kp mayor que 1 aumentaremos la ganancia de G, subiendo la gráfica de Black y reduciendo el margen de ganancia. Al reducir el margen de ganancia hacemos más rápida la respuesta, pero también hacemos el sistema menos armotiguado, por lo que aumentan el sobrepaso y el tiempo de establecimiento. Por ello, se busca un punto intermedio para ajustar el margen de ganancia.

Teniendo este balance en cuenta se ha diseñado el control en un punto intermedio obteniendo el siguiente resultado.

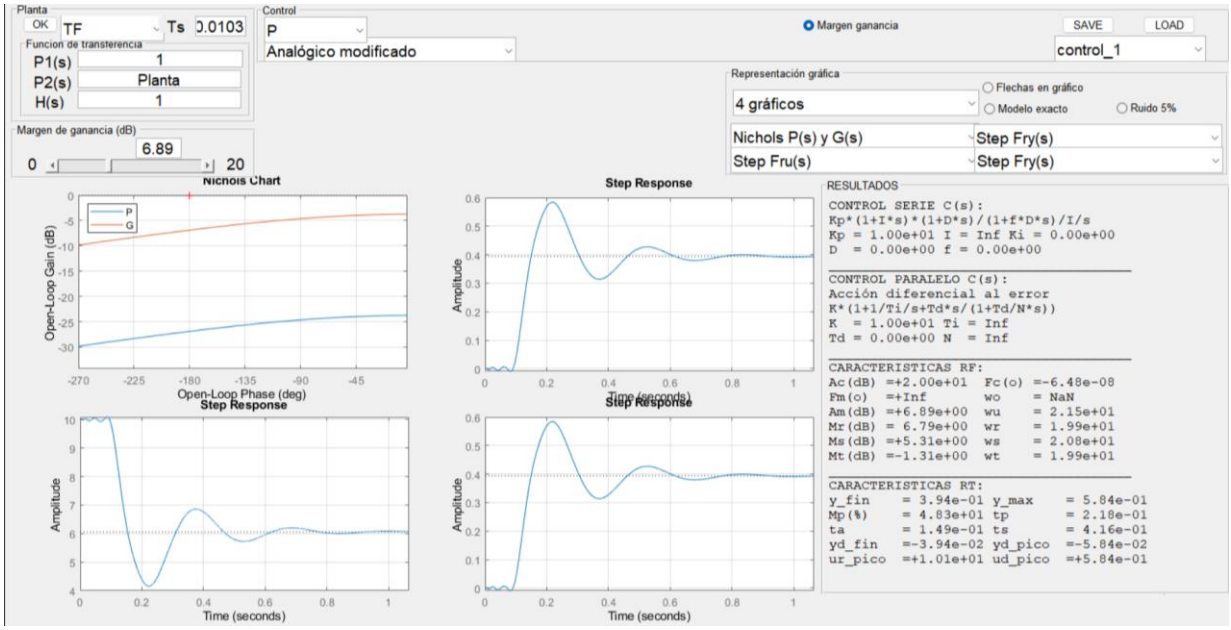

Figura 31 Control Proporcional de velocidad

El control escogido tiene un sobrepaso bastante alto, del 48.3% pero se ha escogido este valor para no sacrificar la ganancia del control. En este caso se consigue un error en régimen permanente del 60%.

5.1.2 Regulador integral (PI)

Al incluir la acción integral, se consigue eliminar el error en régimen permanente que sufríamos con el control proporcional. Por otra parte, este control reduce la velocidad de la respuesta, por lo que no se debe de abusar de esta acción. A la hora de elegir el control PI, es necesario saber cuál es el objetivo. Para este caso, nuestro objetivo es conseguir el menor IAE (Integral Absolute Error) posible, aunque, sin sacrificar en otros parámetros claves de la respuesta para conseguirlo.

Por ello, se ha estudiado cómo varían los diferentes parámetros para las combinaciones de los parámetros de entrada margen de fase (Fm), ponderación (b) y la velocidad del control (wn). Se observa que el tiempo de establecimiento, Ts, no varía mucho para nuestros diferentes valores por lo que no le damos mucho peso a la hora de elegir la combinación.

En general, se puede ver que, a mayor margen de fase, menor sobrepaso y, además, que a partir de una velocidad limite empieza a bajar cada vez más el sobrepaso para un margen de fase dado.

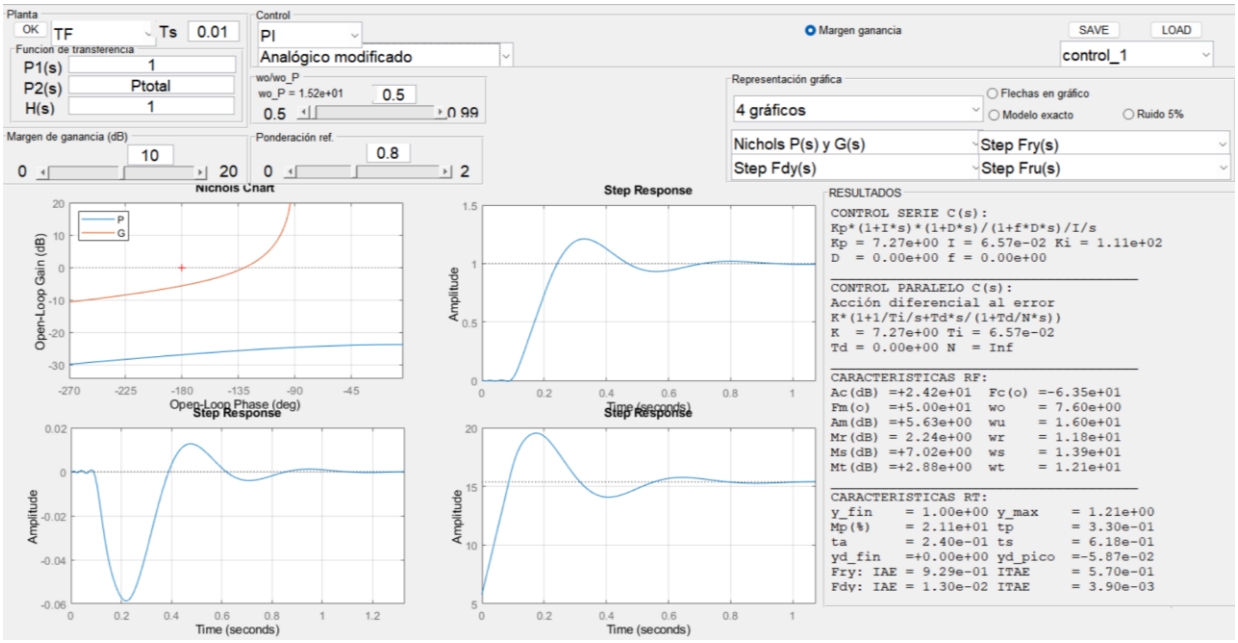

Figura 32 Control PI de velocidad

Se puede observar que la respuesta del control es buena, consiguiendo eliminar el error en régimen permanente. Además, el sobrepaso es mediano, un 21%. Por desgracia, el control en los primeros instantes es más lento que el proporcional, consiguiendo un tiempo de pico de 0.33 segundos.

5.1.3 Regulador derivativo (PD)

Al tener una planta con retraso, la acción diferencial no tiene un gran efecto, ya que el desplazamiento lateral de la gráfica de P no produce un gran cambio en el margen de ganancia.

Lo que se busca principalmente con la acción diferencial es aumentar la rapidez de la respuesta. Para diseñar este control vamos a fijar el margen de ganancia (por coherencia con el P, K=6.89), la pulsación de cruce se buscará tal que sea mayor a la del proporcional y que sea lo más alta posible para el factor de filtrado típico de 0.1.

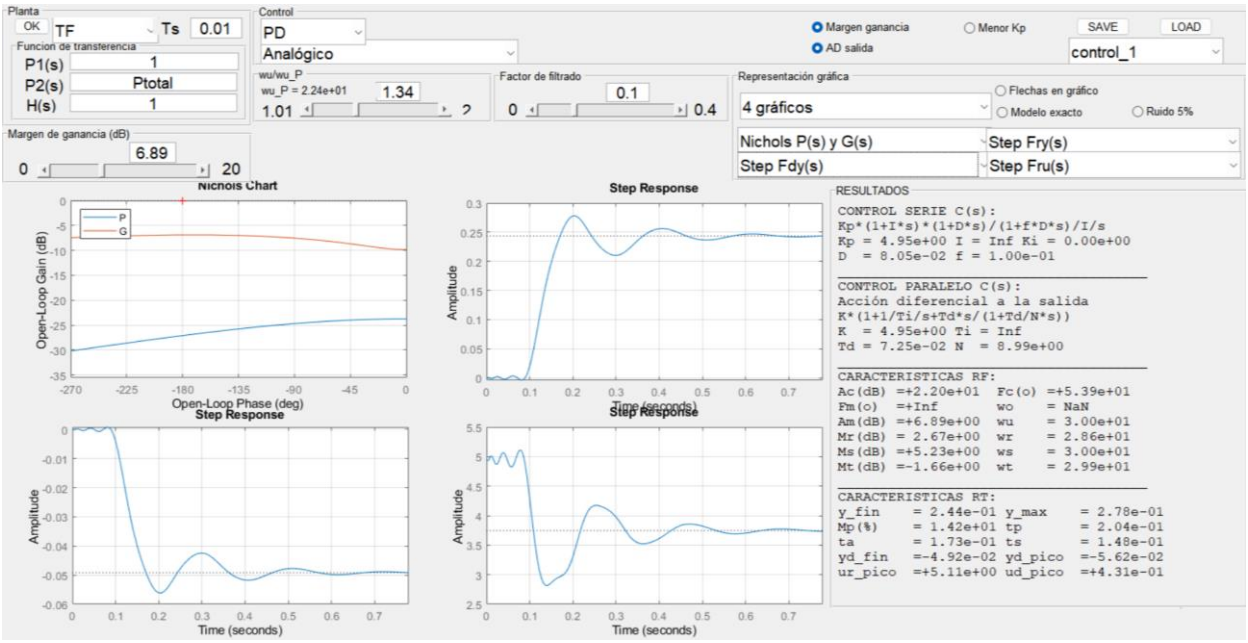

Figura 33 Control PD de velocidad

Se ha decidido usar acción diferencial a la salida ya que la respuesta, aunque más lenta, mejora notablemente respecto a la acción en el error. Se puede ver que aplicando la acción diferencial aumentamos la velocidad del control respecto a su equivalente proporcional. Aunque no disminuye el tiempo de pico notablemente sí lo hace el tiempo de establecimiento, siendo este casi tres veces más rápido que el proporcional. Por último, se puede ver que hay un gran error en régimen permanente, estabilizándose solo al 25% de la referencia.

5.1.4 Regulador PID

El objetivo de diseño de un control PID es combinar las ventajas del control integral con las del diferencial. Es decir, aumentar la velocidad del control y anular el error en régimen permanente.

Una vez más, se va a fijar el margen de ganancia (por coherencia con el P, K =6.89) y el factor de filtrado a 0.1.

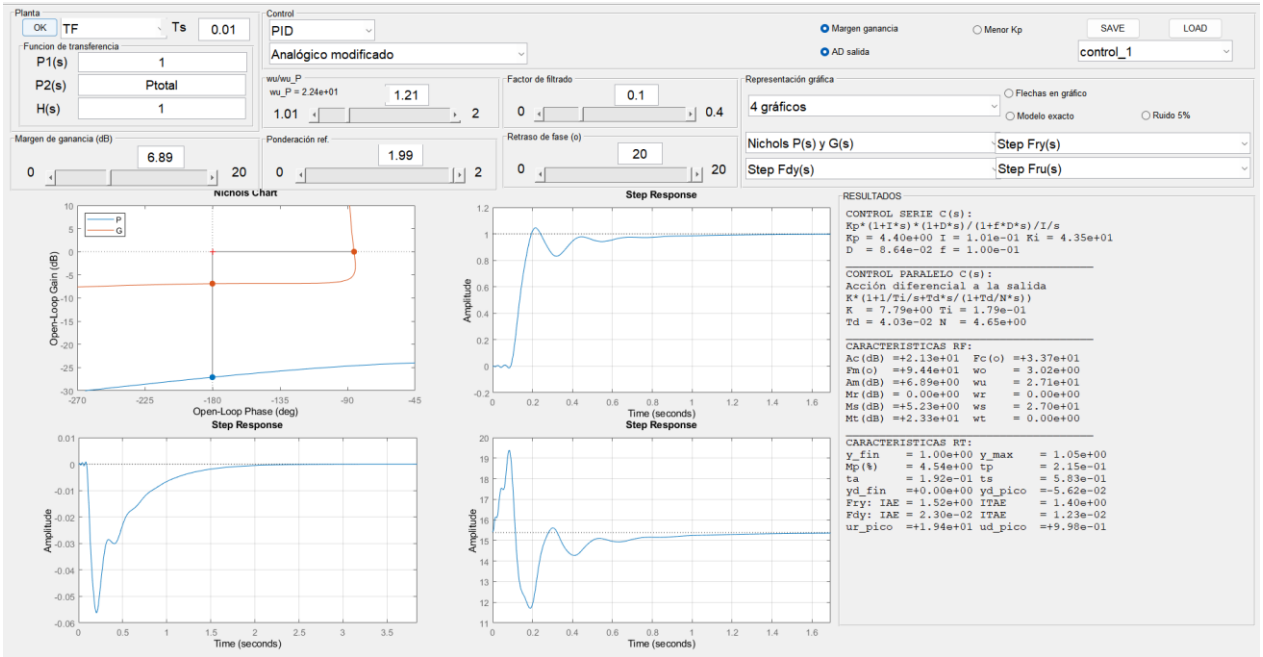

Figura 34 Control PID de velocidad

Se puede ver en el gráfico del escalón de la parte superior derecha que se consigue una buena respuesta, con un sobrepaso y tiempo de establecimiento bajos. Por último, el tiempo de pico de la respuesta ha descendido notablemente respecto al del control PID.

6

Implementación del control de velocidad de avance en ROS2

El objetivo de esta sección es comentar los detalles técnicos asociados a la implementación del control de velocidad en el robot usando las capacidades de despliegue de controles desde MATLAB a ROS2.

Para realizar el control de velocidad en ROS2 se han usado las capacidades de creación de controles en Simulink y el despliegue de este usando las capacidades de generación de código C++ de ROS la MATLAB ROS Toolbox. Para la lectura de los controles se ha usado un archivo de Simulink diferente, con él se leen *topics* usando la característica de conexión de ROS2 para prototipado en el equipo. Se describe el funcionamiento de la Toolbox en más detalle en la sección 2.5.

6.1 Esquema del control

A continuación, se muestra el esquema en Simulink del control implementado.

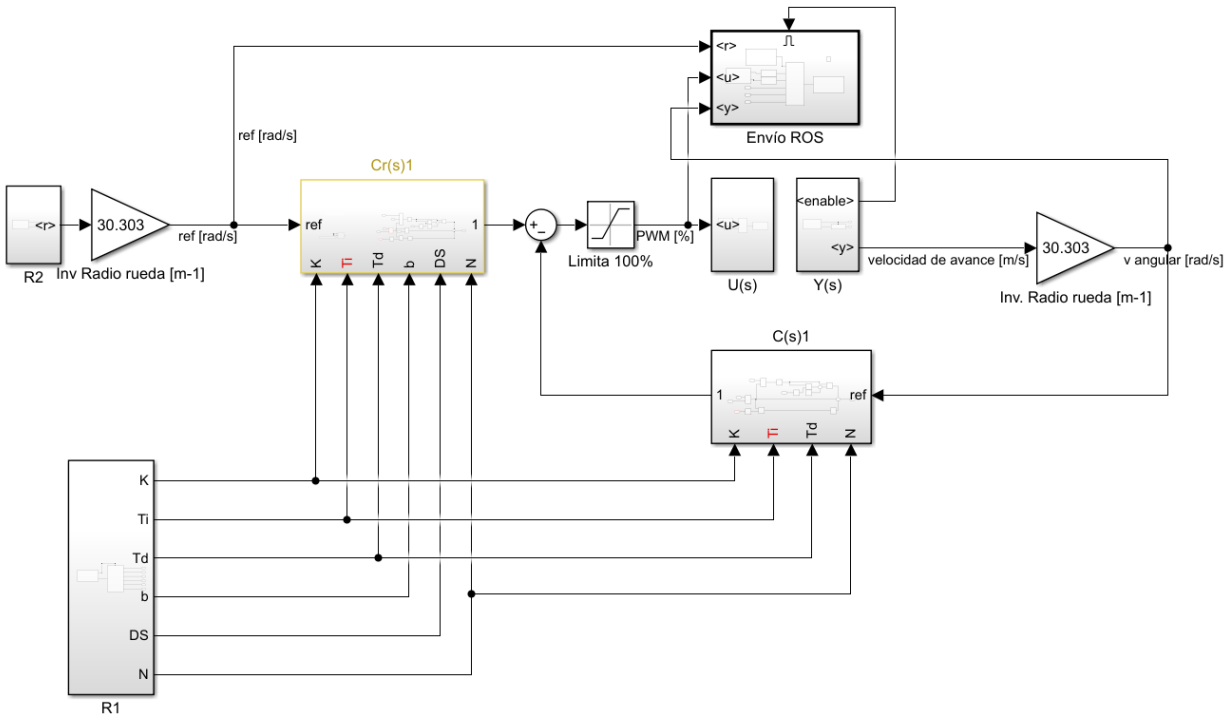

Figura 35 Esquema del control de velocidad a desplegar.

Este control se implementa en la Raspberry Pi 3B usando las herramientas de generación de código. Este código, una vez generado y subido al robot, es completamente independiente de MATLAB, y puede ser usado y editado para entornos de producción sin necesidad de licencia alguna ya que se trata de código en C++. Al ser generado automáticamente tiene un alto nivel de abstracción y es difícil de ser interpretado en detalle por una persona. Para hacer de puente en Matlab para recibir los datos de los ensayos y mandar los parámetros y consignas del control hace falta tener un segundo archivo de Simulink.

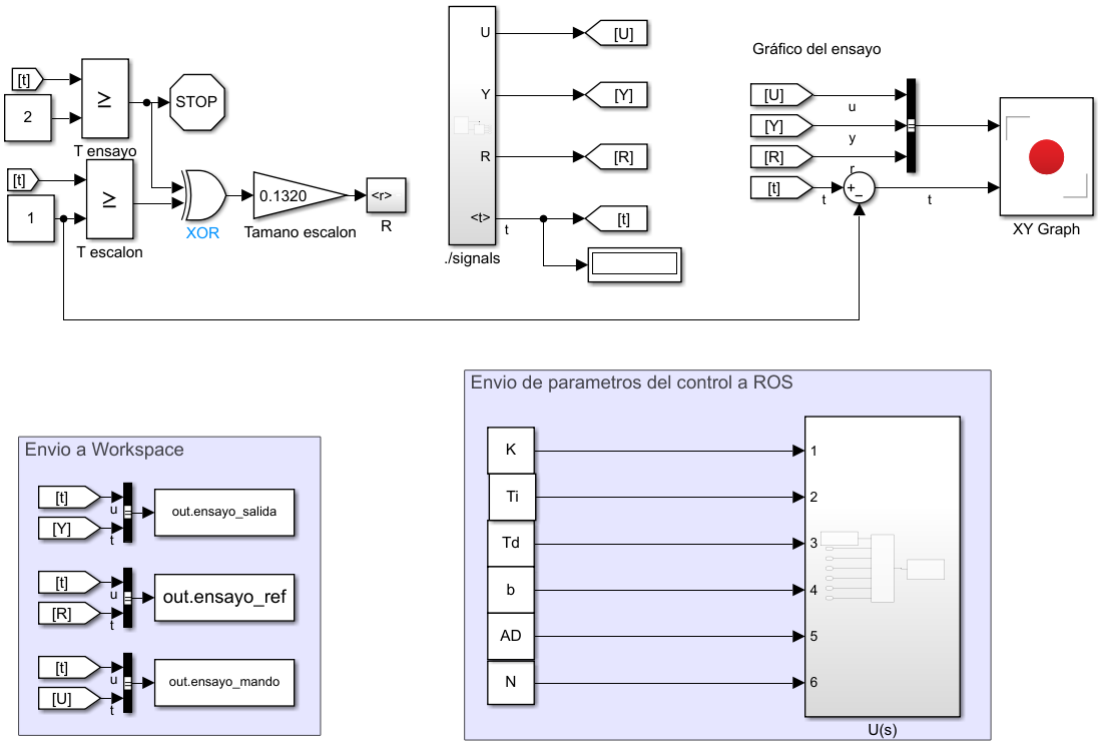

Figura 36 Sistema de monitorización y envío de señales del control de velocidad.

Este archivo posee además la capacidad de monitorizar las señales en tiempo real usando el bloque XY Graph. Se usa este bloque para usar la escala de tiempos recibida en el topic `/signals` como eje X. De esta forma no se tiene que usar el bloque *Simulation Rate Control* como En el apartado 10.3. Esto lleva a que las medidas de tiempo sean más exactas ya que el bloque puede llevar a derivas con el tiempo real para simulaciones muy largas.

En la Figura 36 Sistema de monitorización y envío de señales del control de velocidad se puede ver que las señales recibidas se envían a Matlab para ser procesadas si es necesario. Por último, en la parte superior izquierda se observa el sistema por el cual se genera la referencia para el control. En este caso se ha generado un escalón en el un segundo después de empezar a monitorizar. La señal de referencia puede ser de cualquier tipo, aunque en caso de que se cree con un bloque de generación interno de Simulink, como un step, hay que usar el bloque de *Simulation Rate Control*. El funcionamiento de este bloque se explica más adelante en el apartado 10.3.

6.2 Intercambio de información con ROS2

Dado que la generación de código obliga a usar dos archivos de Simulink, tal y como se ha descrito en los apartados anteriores, la estructura de las comunicaciones en ROS2 se complica un poco.

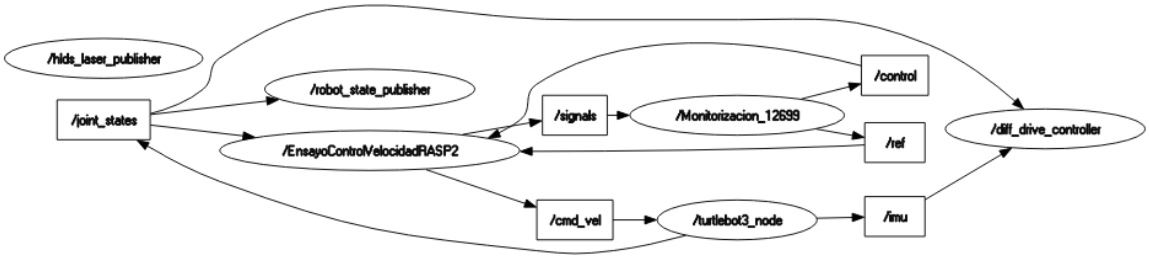

Figura 37 Gráfico de nodos en ROS2 del control de velocidad.

En la Figura 37 Gráfico de nodos en ROS2 del control de velocidad, existen seis nodos, marcados por óvalos en el diagrama, y cinco *topics*, marcados por rectángulos, sobre los cuales los nodos intercambian información.

Los nodos de `/turtlebot3_node`, `/diff_drive_controller`, `/robot_state_publisher`, son creados por el robot y forman parte de la estructura básica del robot descrita en el apartado de Implementación serie del fabricante. Estos nodos no han sido alterados en absoluto, salvo la modificación del topic de `/cmd_vel` para llevar en las consignas de tensión explicadas en el apartado de Firmware modificado.

Este topic, `/cmd_vel`, se publica por el nodo `/EnsayoControlVelocidadRASP`, este contiene el control de la Figura 35 Esquema del control de velocidad a desplegar.. Dentro de este, se publica en concreto por el bloque U(s). Dentro de él se usa un bloque de publisher de la ROS Toolbox donde se introduce el mando dentro del componente `linear.x` de un mensaje tipo Twist.

Para monitorizar la señal de mando aplicada, se podría suscribir dentro del nodo `/Monitorizacion_XXXX`, creado por el archivo de monitorización de la Figura 36 Sistema de monitorización y envío de señales del control de velocidad. Se ha decidido no hacerlo usando esta técnica finalmente, ya que el mensaje tipo Twist no tiene una marca de tiempo asociada. Esto puede llevar a una desincronización con el resto de las señales del control a la hora de visualizarlas. Para ello se ha diseñado el topic `/signals` que contiene todas las señales del ensayo. La implementación de dicho topic se explica en el apartado de Sincronización de señales.

Por último, los parámetros del control se mandan desde el nodo `/Monitorizacion_XXXX` mediante el topic `/control` hasta el nodo `/EnsayoControlVelocidadRASP`. En el apartado de Conclusiones y futuros desarrollos

Estructura del control PID se explica en detalle el topic y la estructura del control.

6.3 Retraso asociado a suscriptores

Existe además una particularidad de los controles generados usando la herramienta de generación de nodos de Simulink. Tal y como se ha expuesto en el capítulo de Modelado de la planta del motor para el control de velocidad, existe un retraso considerable en el control, en este caso de 90ms.

Se ha llegado a la formula empírica haciendo ensayos con diferente número de suscriptores en Simulink de que cada subscriptor que se añada al lazo provoca un retraso de 30ms en la respuesta del control. Esto se debe a que el código generado lee todos los topics antes de empezar a hacer los cálculos y publicar los topics salientes cada iteración.

Por eso, tal y como se puede ver en la Figura 35 Esquema del control de velocidad a desplegar., al tener 3 suscriptores, R1, R2 e Y(s), nuestro sistema tiene 90 ms de retraso.

6.4 Sincronización de señales

Dentro del programa de monitorización del ensayo existe el problema de la sincronización de señales. Si se enviase cada señal por separado se correría el riego de que se desincronizasen las señales entre sí para ensayos largos. Esto se debe a que al mandarse y recibirse cada señal individualmente, por problemas de comunicación al estar el programa de monitorización en el PC y el control en el SBC, pueden perderse muestras o retrasarse su llegada. Esto se puede arreglar en gran medida añadiendo una marca de tiempo asociada a cada mensaje. Para simplificar la estructura y evitar hacer cálculos innecesarios se ha decidido mandar las señales dentro de un mensaje `TwistStamped` en el *topic* `/signal`. Dentro de este se muestra el estado de todas las señales internas del nodo `/EnsayoControlVelocidadRASP` en un instante de tiempo. Se ha decido usar este tipo de mensaje para permitir monitorizar más señales de las actuales, ya que se podrían añadir hasta tres señales más sin coste adicional si fuese necesario. En la siguiente Figura se muestran los bloques usados para crear el publisher del topic `/signals`.

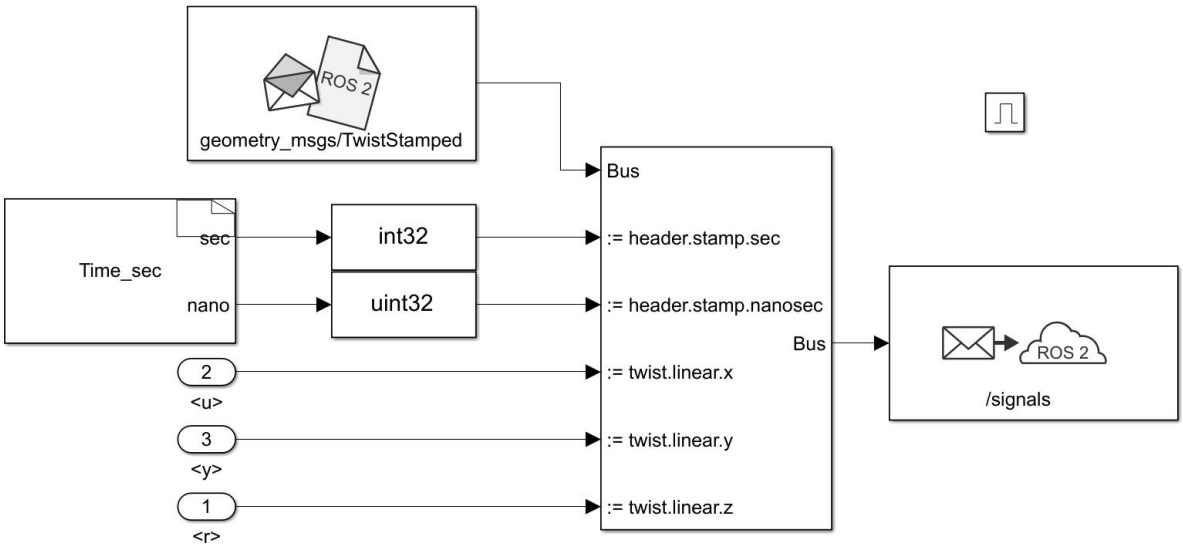

Figura 38 Publisher del mensaje de topic /signals.

En la Figura 38 Publisher del mensaje de topic /signals, se puede ver la estructura de bloques usada para generar el mensaje. Se añaden las señales de mando, salida y referencia al componente twist del mensaje y se añade la marca de tiempo, dividida en segundos y nanosegundos, en el encabezado del mensaje. Para generar las señales de segundos y nanosegundos se usa el siguiente código en C.

```
/* Includes_BEGIN */
#include <math.h>
#include <time.h>
/* Includes_END */

void Time_sec_Start_wrapper(void)
{
/* Start_BEGIN */
/* Start_END */
}

void Time_sec_Outputs_wrapper(real_T *sec, real_T *nano)
{
/* Output_BEGIN */
    sec[0] = (int)time(NULL);
    struct timespec ts;
    timespec_get(&ts, TIME_UTC);
    nano[0]=ts.tv_nsec;
/* Output_END */
}

void Time_sec_Terminate_wrapper(void)
{
/* Terminate_BEGIN */
/* Terminate_END */
}
```

Código 6 Bloque de adquisición de tiempo.

Se puede observar que se trata de la típica técnica de adquisición de medidas en C usando `time.h`.

Por último, existe una particularidad en el *topic* `/signal`. Tal y como se muestra en la Figura 35 Esquema del control de velocidad a desplegar.el topic se actualiza cada vez llega un mensaje del topic `/joint_states`, momento en el cual se manda el valor actual de todas las señales. Esto provoca que la señal de mando esté una muestra retrasada, ya que no le da tiempo a la señal Y a propagarse hacia delante para calcular el nuevo mando. Para solucionarlo se puede simplemente adelantar la señal de mando una muestra al mostrar las señales al usuario.

6.5 Estructura del control PID

A la hora de crear la estructura para el control, es importante tener en cuenta que para cambiar los valores numéricos del control es necesario volver a hacer una compilación de este, tal y como se explica en el apartado de despliegue del control. Este proceso depende de varios factores, pero suele tardar entre 5 y 7 minutos. Por tanto, dificulta en gran medida la experimentación a la hora de elegir el control PID deseado. En un entorno de producción esto no sería un problema, ya que se desarrollaría un control que se mantendría constante usando el modelo de la planta.

En este caso, dado que se busca un uso académico en el Laboratorio de Control, se ha decidido configurar el control pasando los parámetros por medio de un topic de ROS, de forma que se puedan cambiar en tiempo real. Esto añade un retraso al que se ha presentado en el apartado 6.3. A continuación, se muestra la estructura del control desplegado.

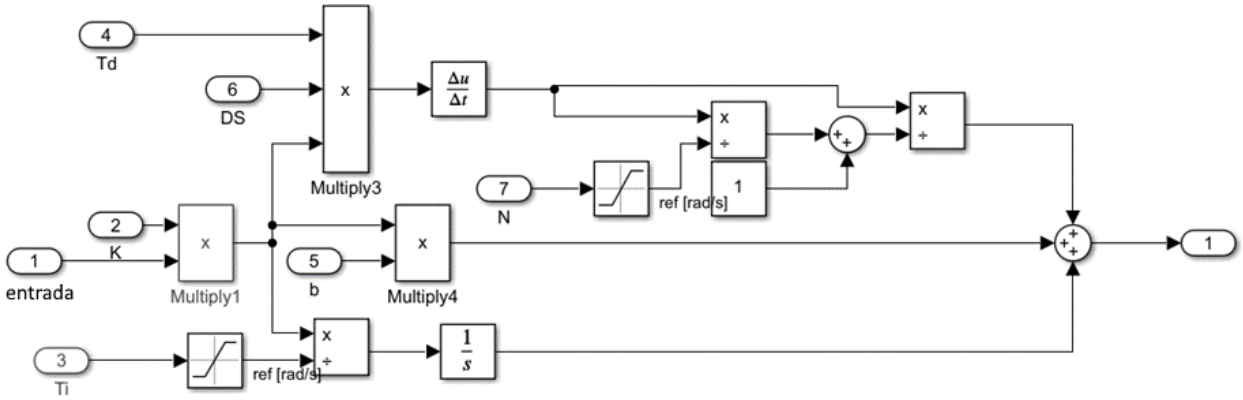

Figura 39 Control PID paralelo.

En la Figura 38 se puede ver que se trata de un control PID en paralelo con la particularidad de que todos los parámetros son señales, y por tanto no se pueden usar bloques de funciones de transferencia.

La configuración de la acción diferencial tal y como está expuesta es problemática y pueden darse grandes errores numéricos grandes errores numéricos si se usan ganancias muy agresivas y satura el mando. Una solución para ello que se podría explorar en futuros proyectos es usar el bloque predefinido PID de Matlab, ya que permite ajustar sus parámetros con señales.

6.6 Despliegue del control

Este modelo está pensado para ser usado en MATLAB 2022a utilizando la librería ROS Toolbox. Los detalles sobre el despliegue de controles en ROS2 usando las herramientas de generación se incluyen en el Anexo IV Uso del control de velocidad.

Es de vital importancia realizar el paso de configuración del middleware `rmw_cyclonedds_cpp` para las versiones de MATLAB 2022a y ROS2 Foxy. En un futuro cercano, la compañía creadora de MATLAB, Mathworks, se ha comprometido a arreglar este inconveniente para que se pueda usar el middleware por defecto.

7

Controles de navegación

El objetivo de esta sección es elegir un sistema de navegación para nuestro robot. De esta forma, el robot ha debe ser capaz de reconocer su posición y orientación en el espacio respecto a un sistema de referencia relativo o absoluto. Para cualquier vehículo la habilidad de navegar el entorno es crucial, ya que le confiere la habilidad de evitar obstáculos y colisiones.

7.1 Sistemas de navegación

Existen diferentes modelos de navegación dependiendo de las necesidades. A continuación, se van a listar algunos tipos:

- Navegación libre: En este control se basa en el seguimiento de consignas de velocidad lineal y angular. Estas se pueden actualizar en tiempo real, usando por ejemplo una emisora de radio frecuencia, o pueden ser previamente planificadas y desplegadas en el momento de uso.
- Navegación para seguimiento de trayectorias: Se realiza un control de las velocidades en lazo cerrado del robot, usando como referencia la posición deseada. Para ello ha de realizarse una localización del robot en el entorno mediante sensores.
- Navegación mediante seguimiento de pared: Se controla la distancia y el ángulo del robot con respecto a la pared en un lateral del vehículo. De esta forma, si esta pared forma parte un circuito cerrado se realiza una navegación continua, dando vueltas por este.
- Localización y mapeo simultáneos: Se realiza un control de posición, al mismo tiempo que se realiza una exploración del entorno del vehículo, sin que este haya sido registrado de antemano. Esta técnica comúnmente conocida por su acrónimo en ingles SLAM (*Simultaneous Localization and Mapping*) es apta para su aplicación fuera de entornos controlados como los coches autónomos.

A la hora de decidir el modo de navegación, se ha tenido en cuenta el entorno en el que se va a implementar. En asignaturas de control de ICAI se realizan ensayos en un recinto cerrado con rampas como se muestra en la imagen siguiente.

Figura 40 Circuito del Laboratorio de Control.

Al tratarse de un entorno cerrado y controlado, sabemos el tipo de obstáculos que se va a encontrar nuestro robot. En este caso, el control ha de ser capaz de girar en el momento que se encuentre la curva para continuar el circuito.

Por ello, se ha decidido implementar un control de navegación mediante seguimiento de pared, ya que se adapta a nuestras necesidades y tiene además un valor didáctico al poder ser desarrollado con controladores PID. Para realizar este control es necesario usar un sensor para medir la distancia a la pared. En este caso se va a usar el LIDAR que viene de serie en el Turtlebot3.

7.2 Sensor LIDAR

La tecnología LIDAR permite obtener una representación de la distancia variable entre el sensor y un objeto. Su funcionamiento se basa en la emisión de un láser hacia la superficie de un objeto y en la medida del tiempo que pasa hasta que la luz vuelve al receptor por reflexión, tal y como se muestra en la Figura 41. La aplicación de esta tecnología se ve por tanto limitada por las propiedades físicas del material. En general, esta tecnología puede ser usada para materiales que sean opacos a dicha longitud de onda, que en nuestro caso es de 785nm.

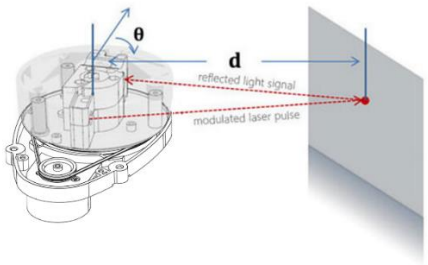

Figura 41 Funcionamiento básico de un LIDAR.

Nuestro robot incluye un sensor LIDAR para realizar tareas de navegación y SLAM. Se trata de un LDS-01[31]. A continuación se muestran sus principales características.

Items	Specifications
Distance Range	120 ~ 3,500mm
Distance Accuracy (120mm ~ 499mm)	±15mm
Distance Accuracy (500mm ~ 3,500mm)	±5.0%
Distance Precision (120mm ~ 499mm)	±10mm
Distance Precision (500mm ~ 3,500mm)	±3.5%
Scan Rate	300±10 rpm
Angular Range	360°
Angular Resolution	1°

Tabla 2 Especificaciones del LDS-01

Podemos ver que el sensor proporciona 360 medidas con una resolución angular de 1 grado, obteniendo así una vista cenital de los alrededores del robot en el plano del sensor. Esta medida se actualiza a una frecuencia variable de aproximadamente 5 Hz según la velocidad del motor.

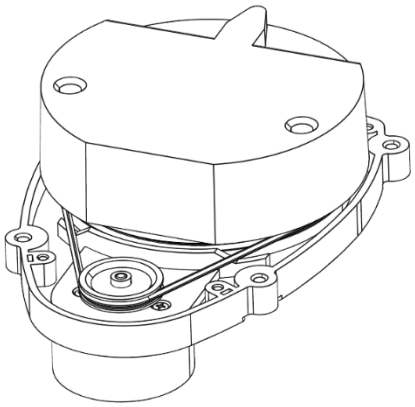

Figura 42 LDS-01

El sensor se comunica con nuestro SBC a través de la placa interfaz USB2LDS, habilitando así la comunicación por la interfaz USB. Los datos recibidos son recibidos y publicados por el paquete de ROS2 del sensor creado por el fabricante. La información del sensor se publica en la estructura de mensaje estándar `LaserScan`. A continuación, se muestra la estructura de dicho mensaje.

LaserScan.msg	
`std_msgs/Header header`	Timestamp y frame id
`float32 angle_min`	Angulo inicial [rad]

LaserScan.msg	
`float32 angle_max`	Angulo final [rad]
`float32 angle_increment`	Distancia angular entre muestras[rad]
`float32 time_increment`	Tiempo entre muestras [s]
`float32 scan_time`	Tiempo entre escaneos [s]
`float32 range_min`	Distancia mínima [m]
`float32 range_max`	Distancia máxima [m]
`float32[] ranges`	Distancia de en la dirección [m]
`float32[] intensities`	Intensidad del haz devuelto

Tabla 3 Estructura del mensaje LaserScan en ROS2

Dependiendo de la aplicación, diferentes elementos del mensaje serán de utilidad, pero para nuestro uso, los parámetros principales serán el vector ranges para ver el entorno alrededor del robot y *scan_time* y el *timestamp* del *header* para poder llevar un registro de cuándo llega el paquete.

8

Modelado de la planta para el control de seguimiento de pared

El objetivo de esta sección es modelar la planta del robot de manera tanto matemática como numérica, de forma que se disponga de un modelo para diseñar el control de seguimiento de pared.

8.1 Descripción de la planta[32], [33]

El vehículo por controlar en este proyecto es el Turtlebot3, que es un robot móvil con cinemática diferencial. En el capítulo 2 se han explicado en detalle los componentes del mismo.

Las dinámicas de este robot se pueden modelar como un sólido rígido moviéndose por un entorno 2D. Por tanto podemos modelar completamente el coche usando el sistema de coordenadas (x,y,θ). Donde (x,y) representa la posición cartesiana del entorno del robot desde su centro de rotación O, y θ representa el ángulo que forma la pared medida con el eje X solidario al robot. En la Figura 43 se muestra un dibujo mostrando nuestro sistema de referencia.

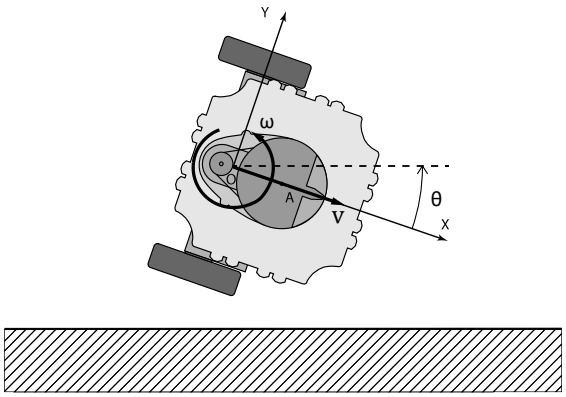

Figura 43: Sistema de referencia del robot

8.1.1 Medida de distancia y orientación

Para realizar el control de seguimiento de pared es necesario saber la distancia y orientación relativa del robot respecto a esta, para lo que se va a usar el LIDAR previamente descrito. En la Figura 43 se puede ver que el LIDAR, punto A, se encuentra desplazado una distancia x_a respecto al centro de rotación O. Esto afectará a los controles que se hagan ya que las medidas estarán adelantadas al punto de giro del robot.

Para medir el entorno se va a hacer una aproximación lineal de la forma de la pared. Para ello se van a tomar dos medidas separadas por el ángulo Φ, la primera de ellas a -90º del eje X, Como se muestra en la Figura 44. Para conseguir una mayor resiliencia de las medidas, se va a hacer la media de las dos medidas contiguas brindadas por el láser en ambos lados. Estos dos haces nos darán los puntos B y C respectivamente. La distancia AB la llamaremos d_1 y la distancia AC la llamaremos d_2.

Existe un límite inferior del ángulo Φ de 5º, causado por las medidas contiguas usadas. Por otra parte, el límite máximo viene definido por cómo de adelantada se quiere que esté la medida respecto al coche, $d_{2t}= d_2 \sin(\Phi)$ sin rebasar el ángulo máximo teórico de 90º. Es interesante ajustar este parametro ya que dependiendo de su valor pueden cambiar drásticamente las dinámicas del control.

La medida del ángulo del robot con la pared se puede calcular como $\theta = \text{atan}((d_1-d_2 \cos(\Phi)) /d_1)$ siendo d1 y d2 las medidas del primer y segundo haz respectivamente. La distancia a la pared, d_n, se define como la menor distancia entre la pared, BC, y el punto A, es decir $d_n = d_1 \cos(90 - \theta)$. A continuación, se muestra un esquema mostrando las medidas.

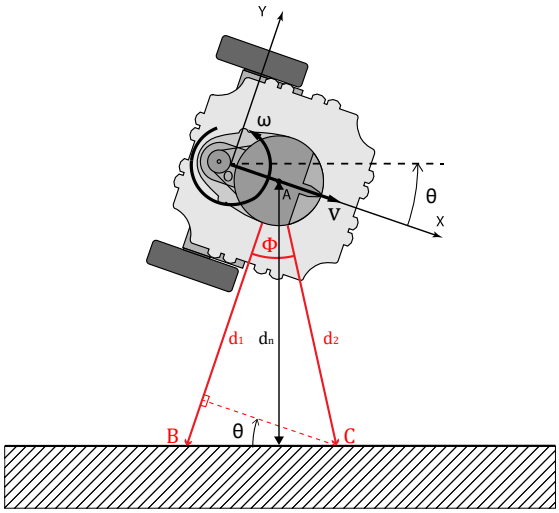

Figura 44 Medidas del LIDAR

8.1.2 Modelado del sistema

Consideramos las siguientes ecuaciones de cinemática, donde v representa a la velocidad de avance del coche y ω a la velocidad de rotación de este. Nótese que θ representa el ángulo relativo del coche con la pared.

$$\begin{cases} \dot{x} = v \\ \dot{y} = 0 \\ \dot{\theta} = \omega - \omega pared \end{cases}$$

Ecuación 14 Cinemática del control de pared.

Se traslada la velocidad del coche respecto al punto A trasladando las velocidades a ese punto.

$$\begin{bmatrix} v_{ax} \\ v_{ay} \end{bmatrix} = \begin{bmatrix} v \\ 0 \end{bmatrix} + \omega * \begin{bmatrix} 0 \\ x_a \end{bmatrix} = \begin{bmatrix} v \\ \omega * x_a \end{bmatrix}$$

Ecuación 15 Velocidad del punto A.

Sobre esta base podemos aplicar una matriz de rotación para pasar a una base solidaria a la pared.

$$\begin{bmatrix} v_{an} \\ v_{at} \end{bmatrix} = \begin{bmatrix} \sin\theta & \cos\theta \\ \cos\theta & -\sin\theta \end{bmatrix} * \begin{bmatrix} v \\ \omega * x_a \end{bmatrix} = \begin{bmatrix} v * \sin\theta + \omega * x_a * \cos\theta \\ v * \cos\theta - \omega * x_a * \sin\theta \end{bmatrix}$$

Ecuación 16 Velocidad del punto A en base solidaria a la pared.

Para modelar el ángulo entre el coche y la pared podemos aplicar que la velocidad es la integral del ángulo, con la precaución de que el ángulo θ se refiere al ángulo relativo entre el coche y la pared. Para ello debemos tener en cuenta que la pared no es siempre recta. Para modelarlo trataremos la pared como otro cuerpo con velocidad de rotación ωpared.

$$\frac{d\theta}{dt} = \omega - \omega pared$$

Ecuación 17 Derivada del ángulo girado.

Se puede simplificar el cálculo si aplicamos la curva a secciones circulares. Podemos modelar la velocidad de giro de la curva, y el tiempo que pasa en ella con las siguientes ecuaciones:

$$\omega_{pared} = \frac{v}{r_{curva}} \; T_{curva} = \frac{\theta_{curva} * r_{curva}}{v}$$

Ecuación 18 Modelado de curva.

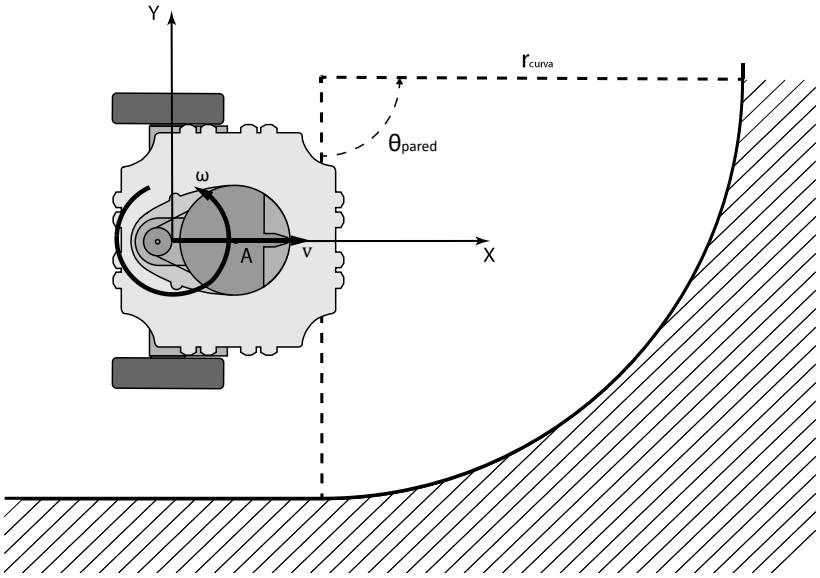

Figura 45 Robot ante perturbación de curva.

Dicha velocidad angular ω_{pared} se aplicará como un escalón mientras dure la curva el tiempo T_{curva}.

Por otra parte, se puede modelar la relación entre la velocidad angular, w [rad/s], y el factor de pwm diferencial, ud [%], con una ecuación diferencial de segundo orden,

$$\frac{d\omega}{dt} = -\frac{1}{Tm}\omega + \frac{Km}{Tm}ud$$

Ecuación 19 Relación entre velocidad angular y tensión.

siendo Tm la constante de tiempo de la función y Km la ganancia estática del sistema de primer orden entre ambas.

Usando estas ecuaciones podemos desarrollar un modelo espacio de estado para modelar el sistema, ya que contamos con tres entradas, por lo que no es posible la representación con una única función de transferencia.

$$X = \begin{bmatrix} \omega \\ \theta \\ d_n \end{bmatrix} U = \begin{bmatrix} ud \\ v \\ \omega_{\text{pared}} \end{bmatrix} \quad Y = d_n$$

$$\frac{dX}{dt} = \begin{bmatrix} -\frac{1}{Tm} * \omega + \frac{Km}{Tm} * ud \\ \omega - \omega_{\text{pared}} \\ v * \sin\theta + \omega * xa * \cos\theta \end{bmatrix} \quad Y = I * X + 0 * U$$

Ecuación 20 Modelo de estado del sistema.

Podemos ver que se trata de un modelo no lineal, ya que incluye funciones trigonométricas y producto de variables. Vamos a linealizarlo en el punto de operación v = v_0, u_d = u_{do}, ω_{pared} =ω_{pared0}

$$\Delta X = \begin{bmatrix} \Delta\omega \\ \Delta\theta \\ \Delta d_n \end{bmatrix} \Delta U = \begin{bmatrix} \Delta ud \\ \Delta v \\ \Delta\omega_{\text{pared}} \end{bmatrix} \quad \Delta Y = \begin{bmatrix} \Delta\omega \\ \Delta\theta \\ \Delta d_n \end{bmatrix}$$

$$\frac{dX}{dt} = \frac{d\Delta X}{dt} \cong F(Xo, Uo) + \begin{bmatrix} -\frac{1}{Tm} & 0 & 0 \\ 1 & 0 & 0 \\ xa\cos\theta o & vo\ \cos\theta o - w0\sin\theta o & 0 \end{bmatrix} \Delta X + \begin{bmatrix} \frac{Km}{Tm} & 0 & 0 \\ 0 & 0 & -1 \\ 0 & \sin\theta o & 0 \end{bmatrix} \Delta U$$

$$\Delta Y \cong Yo + I * \Delta X + 0 * \Delta U$$

Ecuación 21 Aproximación lineal del modelo de estado.

A continuación, convertimos la representación de espacio de estado en funciones de transferencia.

$$\begin{cases} \dot{X} = A\,X + B\,U \\ Y = C\,X + D\,U \end{cases} \quad \frac{Y(s)}{U(s)} = C[Is - A]^{-1}B + D$$

$$\frac{Y(s)}{U(s)} = \begin{bmatrix} \frac{\text{Km}}{Tm} & 0 & 0 \\ \frac{\text{Km}}{s\,(Tm * s + 1)} & 0 & -\frac{1}{s} \\ \frac{\text{Km}\,(v_0\cos(\theta_0) - w_0\sin(\theta_0) + s\,\text{xa}\cos(\theta_0))}{s^2\,(Tm * s + 1)} & \frac{\sin(\theta_0)}{s} & -\frac{v_0\cos(\theta_0) - w_0\sin(\theta_0)}{s^2} \end{bmatrix}$$

Ecuación 22 Respuesta del sistema.

Las funciones que nos interesan en este caso son las que relacionan nuestras las medidas que determinan la posición de nuestro robot, θ_0 y dn, con nuestro mando de los motores, ud, la tensión PWM diferencial.

$$\frac{\theta}{ud} = \frac{\text{Km}}{s\,(Tm * s + 1)} \qquad \frac{d_n}{ud} = \frac{vo * Km * \cos(\theta_0) * \left(1 + \frac{xa}{vo}s\right) - w_0\sin(\theta_0)}{s^2\,(Tm * s + 1)(Tm * s + 1)}$$

Ecuación 23 Respuesta a las variables medidas.

8.2 Obtención de valores numéricos de la planta

En primer lugar, se va a definir el punto de operación.

$$v_0 = 0.1\ \left[\frac{m}{s}\right]\quad ud_0 = 0[\%]\quad \omega_{pared_0} = 0\left[\frac{rad}{s}\right]$$

$$\omega_0 = 0\ \left[\frac{rad}{s}\right]\quad \theta_0 = 0[º]\quad d_n = 0.2\ [m]$$

Ecuación 24 Punto de linealización.

Nótese que en el punto de operación la distancia a la pared, la tensión diferencial y la velocidad relativa de la pared son variables de entrada y salida de la planta, por lo que cambiarlos no modificarán las dinámicas del control en lazo cerrado, pero si la amplitud de la respuesta. Por tanto, se verá la misma planta aun si se cambian sus valores en el punto de operación.

La velocidad de avance por su parte sí que cambia las dinámicas de la planta, ya que la ganancia estática y la constante de tiempo del cero de la función entre la tensión y la distancia a la pared dependen de esta.

El punto de operación de este sistema es tal que el ángulo θ es cero. Esto nos facilita mucho las ecuaciones ya que quita el término que depende de la velocidad de la pared. La función de transferencia resulta:

$$\frac{d_n}{ud} = \frac{vo * Km * \left(1 + \frac{xa}{vo}s\right)}{s^2\ * (Tm * s + 1)}$$

Ecuación 25 Función de distancia linealizada

La planta anterior se puede dividir en dos usando como variable intermedia del ángulo del coche, θcoche, tal como se describe en la Ecuación 22.

$$\frac{d_n}{ud} = P1 * P2 = \frac{Km}{s\ * (Tm * s + 1)} * \frac{vo\left(1 + \frac{xa}{vo}s\right)}{s}$$

Ecuación 26 Separación en variables intermedias.

Para identificar el valor de la Planta P1, un método posible para este caso es hacer un ajuste por mínimos cuadrados a un ensayo con un control proporcional en lazo cerrado. De esta forma, podemos ajustar los parámetros del sistema de primer orden hasta que se minimice el error. Es importante resaltar que para que la identificación sea buena es necesario tener en cuenta el retraso producido por los *subscribers* en ROS2, y este parámetro ha de ser constante y no ajustable por el algoritmo de mínimos cuadrados.

En este caso, dada la naturaleza del control, el ensayo más cómodo resulta de ejecutar el control poniendo referencia cero y ejecutando el control desde un ángulo medianamente alto. A continuación, se muestran unas gráficas de la respuesta en ensayo y simulación con condiciones iniciales de 26º, K=-1.83 y referencia 0.

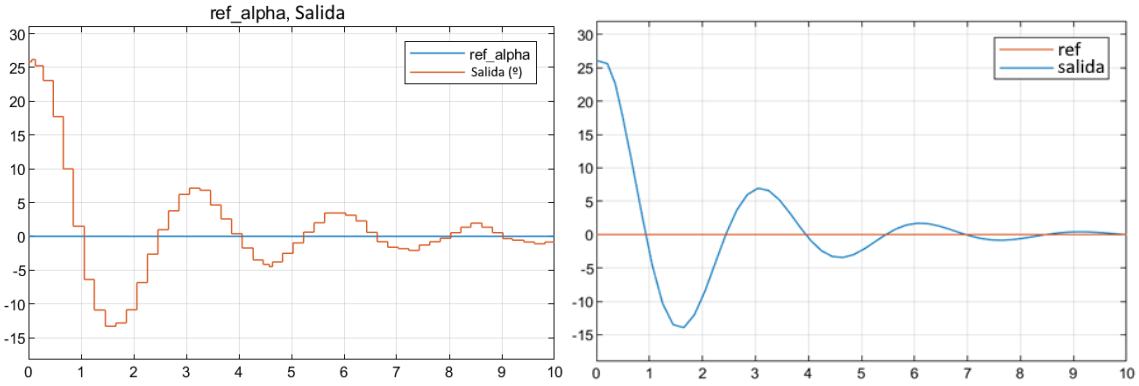

Figura 46 Comparación perturbación inicial en ángulo.

Se puede ver en las figuras que el ajuste de modelo de la plantaes bastante bueno, aunque la respuesta en la realidad es ligeramente menos amortiguada. La planta que se ha conseguido para la simulación vista tiene la siguiente función de transferencia.

$$\frac{\theta}{ud} = \frac{-2}{(s/1.2 + 1)}$$

Ecuación 27 Respuesta mediad ante ángulo.

De esta forma, obtenemos un valor de ganancia estática, Km, de -2 y una constante de tiempo, Tm, de 0.833. Finalmente, se ha visto que había una relación inversa entre las variables que da lugar a una ganancia negativa. Ello nos obligará a usar controles con ganancias también negativas. Juntando todos los valores numéricos comentados se obtiene finalmente las siguientes funciones de transferencia como planta para el punto de operación estudiado.

$$P1 = \frac{\theta}{ud} = \frac{-2}{(s/1.2 + 1)} \qquad P2 = \frac{d_n}{\theta} = \frac{0.1(1 + 1.975s)}{s^2}$$

Ecuación 28 Valores numéricos de las plantas.

Nótese que la función de transferencia de esta planta tiene un doble integrador, lo cual dificulta la aplicación de un control PID, aunque la presencia de un cero negativo (xA >0) mitiga en parte este problema. [16]

9

Control de seguimiento de pared

El objetivo de esta sección es plantear un ejemplo de posible implementación del control de seguimiento de pared.

El objetivo de nuestro control es el seguimiento de la distancia a la pared dn, por medio de la tensión PWM diferencial a los motores ud. Hoy otras dos entradas que afectan a la respuesta del sistema, velocidad de avance y la velocidad de rotación de la pared.

$$\frac{d_n}{v} = \frac{sin(\theta_0)}{s} \qquad \frac{d_n}{\omega_{pared}} = -\frac{v_0\, cos(\theta_0) - w_0\, sin(\theta_0)}{s^2} = -\frac{v_0}{s^2}$$

Ecuación 29 Perturbaciones al control

Vemos que estas funciones de tranferencia comparten una gran parte de la estructura con la planta por lo que será fácil incluir estas entradas como perturbaciones.

Además de estas entradas es recomendable modelar también una perturbación para tener en cuenta la asimetría de los motores. Es muy probable exista una pequeña deriva de nuestro robot hacia el lateral al aplicar la misma tensión a ambos motores. Para compensarlo se aplicará un pequeño voltaje extra a uno de los motores. Se pueden incluir estas entradas como perturbaciones dentro de la planta dentro del modelo de planta linealizada.

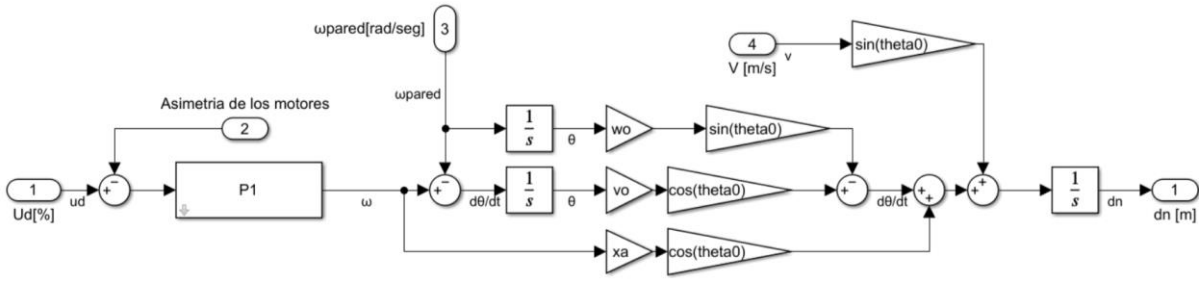

Figura 47 Planta completa

Este sería el modelo de planta completo para realizar nuestro control, pero se pueden realizar algunas simplificaciones dado nuestro punto de trabajo. El efecto de las perturbaciones de la velocidad de avance se va a despreciar ya que queremos trabajar para ángulos pequeños y por tanto la ganancia de la función de transferencia será muy baja. Sin embargo, el efecto de la velocidad de rotación relativa de la pared, no se puede considerar despreciable. De esta forma obtenemos el siguiente diagrama, véase que se ha reducido notablemente la complejidad del modelo.

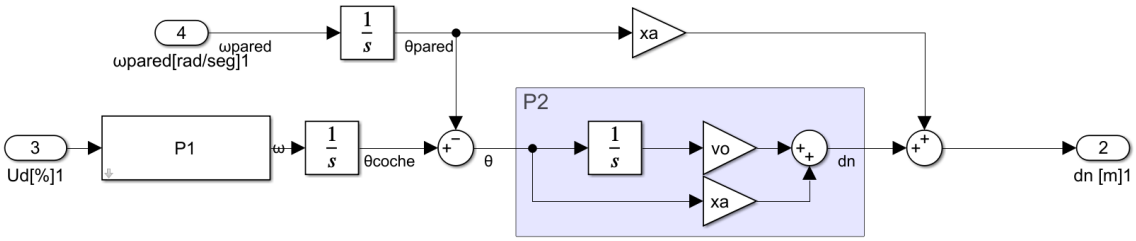

Figura 48 Planta simplificada

Para controlar esta planta podemos aplicar la estructura básica de un sistema de control mediante la realimentación con un solo lazo de la variable controlada. Sin embargo, tal y como se ha comentado en el apartado 8.2, la presencia de un doble integrador complica el diseño de este tipo de control.

9.1 Control en cascada

Por tanto, se ha decidido implementar un control en cascada con dos lazos, uno interno, y otro externo. El control externo contiene el controlador primario o maestro. El lazo interno, por su parte, contiene un controlador secundario o esclavo, que usa como consigna el error del primario.

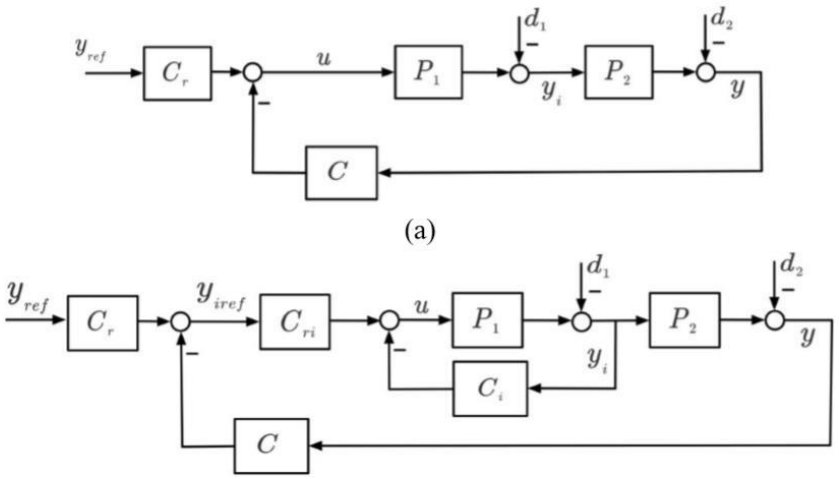

Figura 49 Comparación entre control de lazo cerrado y control en cascada

[Laboratorio de Control Avanzado, ICAI, curso 2019-2020]

A la hora de diseñar el control, para su buen funcionamiento es necesario que el control interno sea más rápido que el externo para que los ajustes que realiza el lazo interno tengan el efecto deseado sobre la variable de salida.

En nuestro caso, tenemos acceso a la variable theta, el ángulo relativo a la pared del coche. Esta variable controlará la tensión de los controles gracias al control interno, la consigna de ángulo será controlada a su vez por el lazo externo.

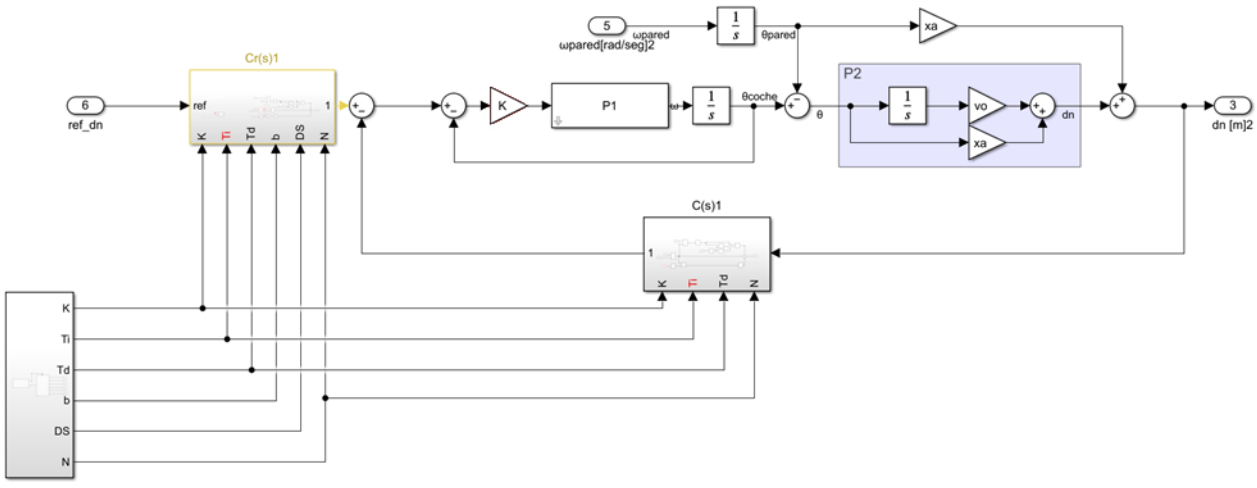

Figura 50 Implementación del control

9.2 Desarrollo del control interno.

Dado que el lazo interno posee ya una integración, no es necesario que nuestro control tenga acción integral. Por tanto, la decisión quedaría entre un control PD y P. Finalmente se ha decidido usar un control PD para obtener la respuesta más rápida posible.

Al mismo tiempo, para que el control funcione correctamente es importante que el lazo interno no tenga un gran sobrepaso, ya que las oscilaciones del ángulo de salida pueden provocar a su vez que la medida de distancia a la pared varíe en gran medida, especialmente en las curvas. Esto nos puede llevar a que el robot choque con una pared.

Por otra parte, este control nos ayuda también a mejorar el lazo externo, ya que las dinámicas de este control afectarán también al exterior. Para conseguir que la velocidad del control de distancia sea lo más rápida posible es necesario que el control de ángulo sea también rápido.

Se ha diseñado un control PD con parámetros en el control en paralelo de K=2.36 Td=0.166 y N=9. Para evitar que el mando sature en los primeros instantes se ha decidido usar la acción diferencial en la salida.

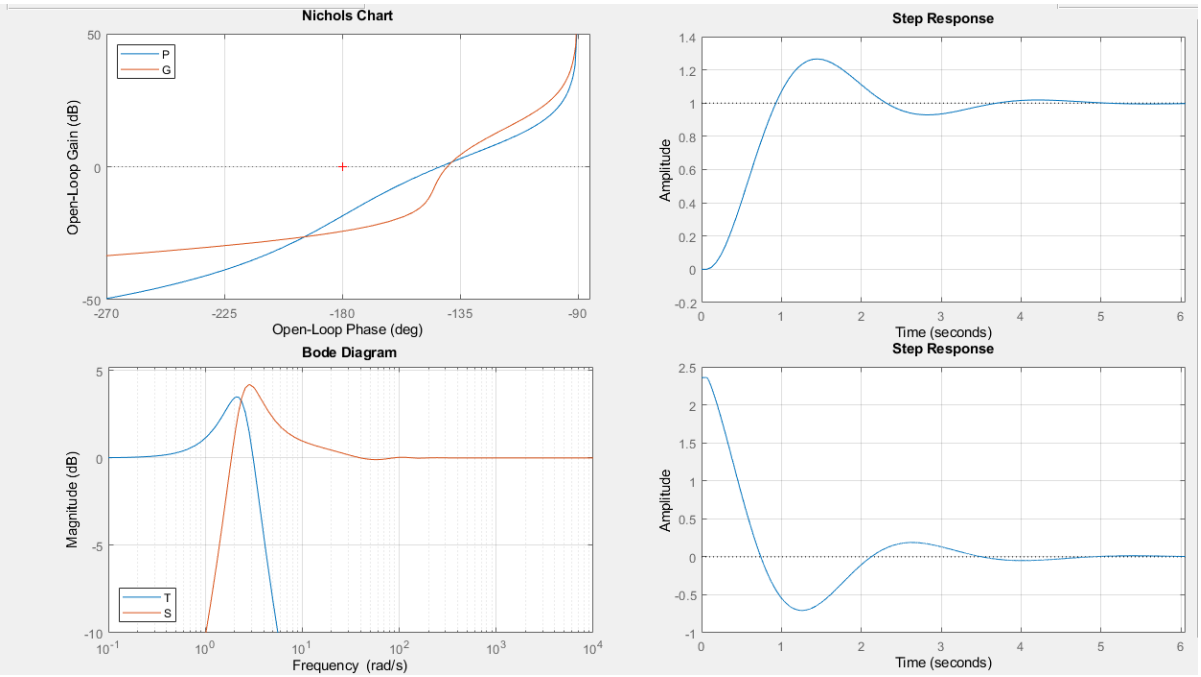

Figura 51 Control PD de lazo cerrado de ángulo

Se puede ver en las siguientes gráficas que el control responde de un forma bastante proxima a lo esperado.

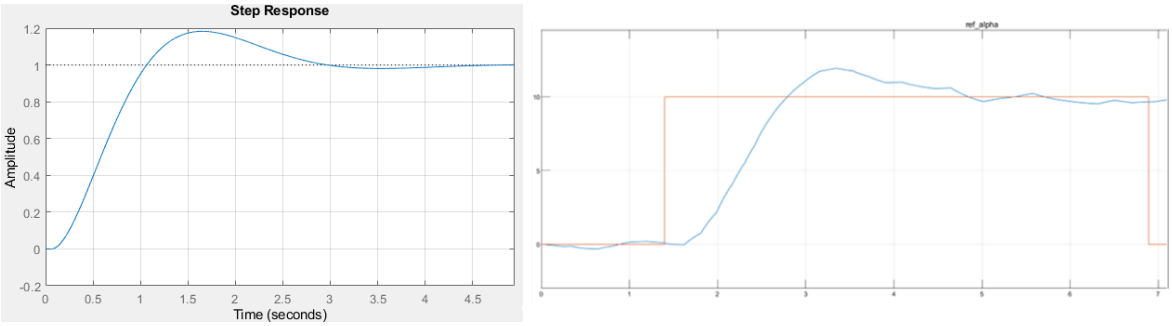

Figura 52 Comparación control PD con la simulación

La función de lazo cerrado de este control se usará como parte de la planta del control exterior. Su función de transferencia no es nada trivial, ya que el retardo de la planta impide hacer una buena representación. Por ello, para el lazo encerrado nos vemos obligados a hacer el control usando técnicas de respuesta en frecuencia con aproximaciones de Padé. Haciendo una aproximación de Padé de primer order al lazo interior obtenemos la siguiente función de transferencia.

$$Fry = \frac{-18.12\,s^2 + 573\,s + 1038}{s^4 + 38.22\,s^3 + 269.1\,s^2 + 896.8\,s + 1038}$$

Ecuación 30 Función en lazo cerrado del lazo interno.

9.3 Desarrollo del control externo.

El lazo de externo de nuestro control de seguimiento de pared controla la distancia a la pared. Para ello, el mando respecto a la distancia de referencia se mandará al control secundario como consigna. La planta de este lazo estará compuesta por la respuesta en lazo cerrado del control interno junto a otra planta que llamamos P2. De esta forma la planta del control será la siguiente.

$$P = Fry * P2 = \frac{-18.12\,s^2 + 573\,s + 1038}{s^4 + 38.22\,s^3 + 269.1\,s^2 + 896.8\,s + 1038} * \frac{0.0395\,s + 0.1}{s}$$

Ecuación 31 Planta total del control.

La planta P2, según se ha visto en el capítulo 8, depende del punto de operación, más en concreto, de la velocidad de avance del coche. Esta relación es fácilmente entendible ya que la variación de la distancia a la pared depende del seno del ángulo respecto a esta. La acción integral de la planta, por su parte, viene de que mientras que se mantenga un ángulo no nulo, la distancia irá variando progresivamente.

Para hacer la elección del control, debido al gran número de polos y ceros de la planta, nos vemos obligados a usar técnicas de respuesta en frecuencia. A continuación, se muestra un gráfico de Black de nuestra planta.

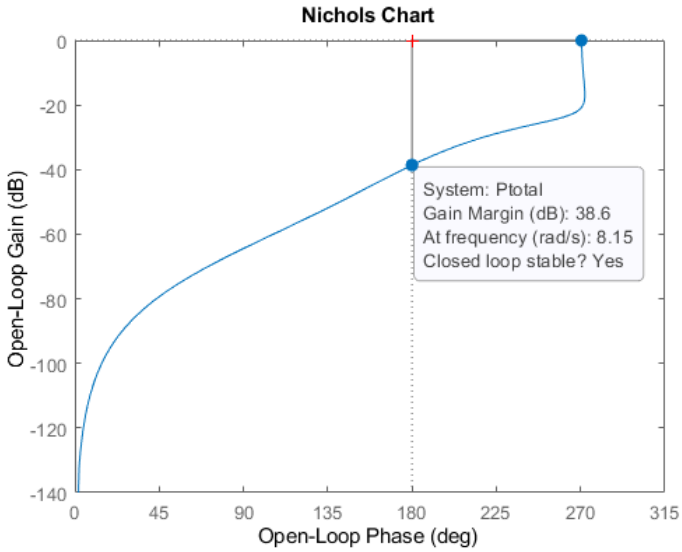

Figura 53 Gráfica de Black de la Planta del lazo exterior

Se puede ver en la Figura 53 Gráfica de Black de la Planta del lazo exterio que el margen de ganancia es muy alto con K=1 por lo que tendremos que usar ganancias muy altas en nuestro control. La causa física está las del control el mando del control interno, la referencia de ángulo y el mando. Las unidades de la medida de mando son grados, mientras que la referencia de distancia está en metros, por lo que para que un escalón de 10cm provoque una diferencia significativa en el ángulo hace falta una gran ganancia.

El control elegido para el lazo exterior es un proporcional, con ganancia K=35 de esta forma desplazamos hacia arriba el gráfico de Black hasta conseguir un margen de fase de 7.72dB.

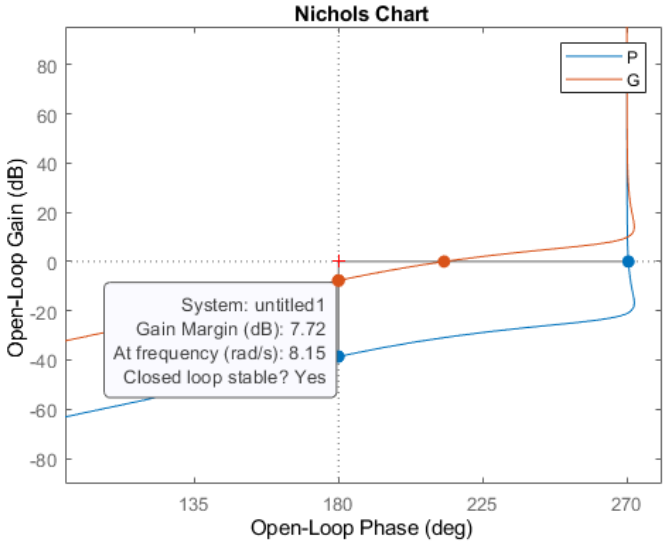

Figura 54 Diagrama de Black de K=35 para el lazo exterior

Con este control se obtiene la siguiente dinámica ante un escalón en la referencia y ante una perturbación por una curva interna en la pared.

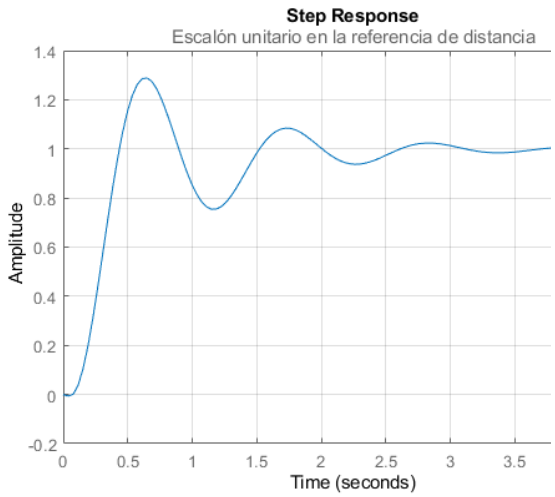

Figura 55 Escalón de distancia

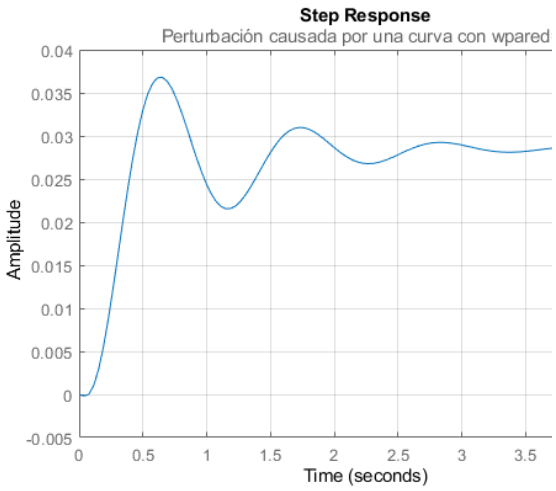

Figura 56 Respuesta ante curva

En la Figura 55 Escalón de distancia se puede ver que existe un sobrepaso cercano al 30% y que existe un error nulo en régimen permanente gracias a la integración de la P2. En la Figura 56 Respuesta ante curva se puede ver que el control es incapaz de mitigar completamente la perturbación, existiendo un error de distancia. Esto causará que la distancia no se mantenga durante las curvas. En el apartado de Medida de distancia y orientación, se explicó que la velocidad de la curva viene marcada por la velocidad de avance del coche y el radio de la curva. Es peligroso por tanto tomar curvas interiores muy cerradas ya que el robot puede llegar a chocarse. Por otra parte, para curvas exteriores, el problema causado por la perturbación es menor, aunque en ese caso corremos el peligro de perder para curvas muy abiertas por culpa del método de adquisición de las medidas de distancia y orientación.

10

Implementación en ROS2 del control de seguimiento de pared

El objetivo de esta sección es comentar los detalles técnicos asociados a la implementación del control de seguimiento de pared en el robot usando las capacidades de conexión de Matlab con ROS para el prototipado de controles.

Existe una segunda forma de implantar los controles en ROS2 desde MATLAB, usando la herramienta interna de prototipado en el equipo. Esta nos permite realizar simulaciones en Simulink y comunicarlas enviando y recibiendo datos en ROS2. Se describe su funcionamiento más en detalle en el apartado 2.5. Para comprobar su funcionamiento, y documentar sus peculiaridades, se ha decido implantar el control de seguimiento de pared usando esta tecnología.

10.1 Esquema del control

A continuación, se muestra el esquema en Simulink del control implementado.

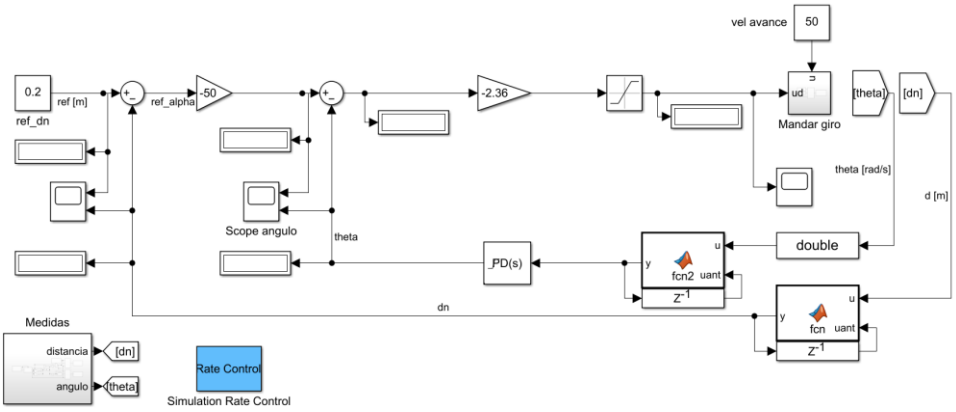

Figura 57 Simulink del control de seguimiento de pared

En la Figura 57 Simulink del control de seguimiento de paredse puede ver que el control no difiere en gran medida de cualquier otro control creado en Simulink, salvo algunas particularidades que se van a describir a continuación en las siguientes secciones.

10.2 Intercambio de información con ROS2

Al igual que en el control de código C++ generado descrito en el capítulo Implementación del control de velocidad de avance en ROS2, se intercambia información usando los bloques de Subscriber y Publisher de la MATLAB ROS2 Toolbox.

En este caso existe dentro del bloque medidas un subscriptor al topic */scan* para recibir los datos del LIDAR y con esos datos se obtener las medidas de distancia y ángulo, tal y como se detalla en el apartado de Medida de distancia y orientación. Estos cálculos tienen la limitación de que se tienen que hacer usando manipulación de señales de Simulink.

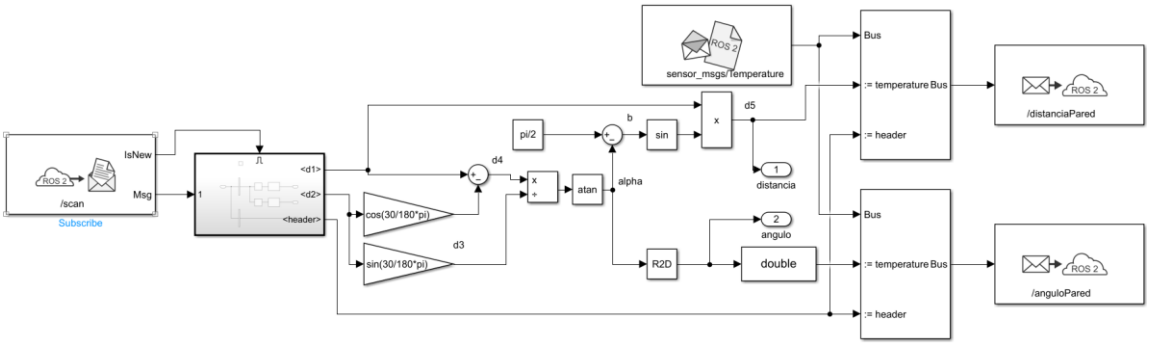

Figura 58 Adquisición de medidas de distancia y orientación

En la parte izquierda de la Figura 58 Adquisición de medidas de distancia y orientación se puede ver el *subscriber* que recibe los datos del LIDAR. Estos se mandan a Simulink usando las salidas del bloque. Por último, hay unos bloques de Publisher que publican en ROS las medidas que queremos ya procesadas. Este último paso es opcional, ya que no tiene ninguna interacción con nuestro control.

Por otro lado, está el bloque mandar giro cuya labor consiste en mandar las consignas de velocidad de avance y giro al robot mediante el topic `/cmd_vel`.

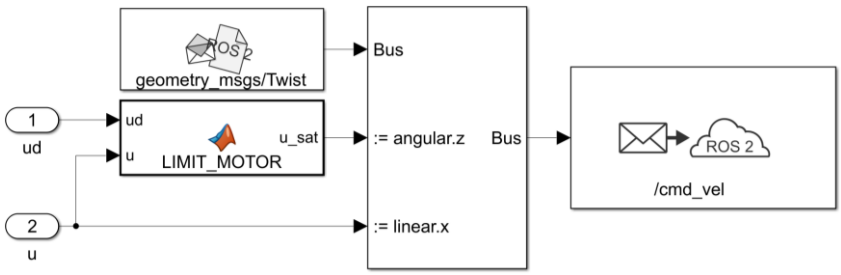

Figura 59 Envío de medidas de PWM común y diferencial.

Antes de mandarla se asegura de que la suma de la tensión común y diferencial no satura y en caso contrario limita la tensión diferencial para evitarlo.

10.3 Sincronización de reloj de simulación

Para poder ejecutar el control usando Desktop prototyping y poder analizar posteriormente la respuesta de este, es importante que la escala de tiempos de Simulink sea correcta. Para ello, hay que añadir un bloque llamado `Simulation Rate Control` para limitar la velocidad de ejecución interna del control para que esta sea a tiempo real. Este bloque no hace falta construirlo manualmente, ya que llama a una función de Matlab bastante compleja. Para conseguirlo es recomendable a ir al ejemplo oficial [34].

Un detalle importante para que el bloque funcione correctamente es ajustar bien la tasa de muestreo deseada. Para ello tenemos que cambiar la tasa de muestreo en el bloque, en el código asociado y hacerlas coincidir con el tamaño de *Fixed-Step* elegido para la simulación. En este caso, el bloque se ha ajustado a 0.01, ajustarlo a valores mucho menores puede no dar los resultados esperados ya que por debajo de este valor ya empieza a estar limitado por el *hardware* del equipo.

10.4 Filtrado de medidas de suscriptores

Es posible que el sensor que se coloque en ROS pierda datos al mandarlos a Matlab. Esto se puede dar de dos formas. En primer lugar, puede ser que se pierda la medida y no llegue ningún dato al lazo de control. En ese caso la mejor solución es cambiar los ajustes QoS del bloque de Subscriber para evitar que se pierdan medidas.

Otro problema que puede haber es que llegue alguna medida rellenada por 0, o sin valor numérico, NaN. Para arreglar este problema lo mejor es realizar un filtro, se puede hacer un filtro en el espacio de Laplace, pero ya que estamos trabajando con muestras discretas es mucho mejor hacer una función que detecte y filtre los errores.

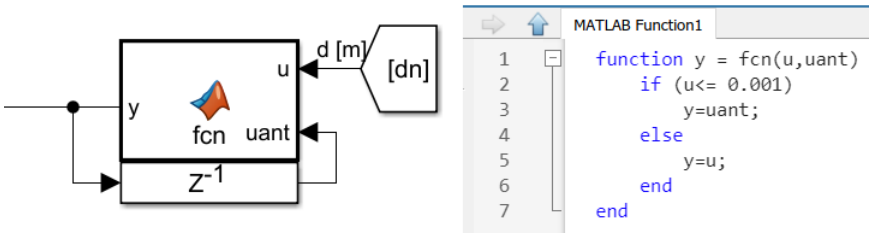

Figura 60 Filtro de medidas perdidas

En la Figura 60 Filtro de medidas se puede ver la lógica que se ha usado para filtrar medidas. Si la medida es muy próxima a cero o inferior, no ha de pasarse al control ya que es una muestra perdida. En su lugar se pasará la última muestra válida que ha pasado por el control. Este filtro tan sencillo se puede hacer ya que por las propiedades del LIDAR en la Tabla 2 Especificaciones del LDS-01, no puede llegar medidas menores a 12cm. Para sensores más complejos puede hacer falta una lógica también más compleja.

11

Conclusiones y futuros desarrollos

11.1 Conclusiones

Finalmente se ha conseguido validar un *workflow* para la implementación de algoritmos en ROS2. Para ello, se ha modificado el modo de operación del motor Dynamixel para convertirlo en un motor controlado por PWM. De esta forma se puede trabajar directamente con las consignas de tensión en ROS. Al sacar el control de velocidad del motor, se consigue una peor respuesta. Ya que se está introduciendo un retraso en la planta del control que ralentiza el control. Aun así, este método nos da mucha flexibilidad y permite usar la estructura de control deseada.

Por otro lado, han propuesto dos métodos para la implementación de los algoritmos de navegación. Aunque ambos métodos cumplen su función, su aplicación está condicionada a las necesidades del usuario.

11.1.1 Despliegue en ROS

Para demostrar el método de despliegue en ROS, se ha realizado el control de velocidad. Este método se basa en crear un control interno en ROS y conectarnos a este para monitorizar desde MATLAB. El nodo de ROS al estar desplegado en la Raspberry Pi, hace que el control sea más estable ya que se elimina la posibilidad de que se pierdan paquetes en la red.

Por otra parte, este método es perfecto para las aplicaciones de robótica de alto nivel, ya que se puede dejar el algoritmo desplegado e interactuar con él a través de otros nodos desplegados.

11.1.2 Conexión con ROS

Para demostrar el método de conexión en ROS, se ha realizado el control de seguimiento de pared. De esta forma se crea el control dentro de Simulink que se comunica con ROS mediante subscribers y publishers. Al estar el algoritmo de control desplegado en el ordenador, puede haber problemas por culpa de las comunicaciones, por tanto, se recomienda usar solo con modelos que no tengan dinámicas excesivamente rápidas.

Este tiene la gran ventaja de que se pueden ajustar las señales y algoritmos con facilidad por el usuario con las herramientas de Simulink, con las que los alumnos están muy familiarizados. Por tanto, es perfecto para su aplicación en el laboratorio de la asignatura de regulación automática.

A la vista de los resultados, se puede ver que se han cumplido todos los objetivos propuestos para el proyecto.

11.2 Futuros desarrollos

A raíz de la finalización proyecto actual, y de lo observado durante realización de este, existen una serie de trabajos futuros cuya realización podría mejorar y ampliar las capacidades del robot.

11.2.1 Modelado en detalle de las plantas

Este proyecto no tenía como objetivo principal el modelado en detalle de las plantas de velocidad y posición para los controles. El objetivo de este era la investigación de los métodos de implementación de controles en el Turtlebot3, por tanto, estas no representan una parte importante del proyecto. Aunque se han identificado unas plantas más que aceptables para el desarrollo del proyecto, se podría hacer un modelado más exhaustivo para identificar, por ejemplo, las causas de los retrasos.

11.2.2 Uso de bloques PID de MATLAB

Se incluye en Simulink unos bloques muy avanzados de PID, estos no se han aplicado en nuestros controles ya que usan un tipo de estructura que no se suele explicar en ICAI, por tanto, resultaría contraproducente para nuestro fin didáctico. Aun así, la generación de código asociada a estos es mucho más eficiente que la usada en este proyecto, además de que la acción diferencial es menos propensa a errores de cálculo. Por ello, podría resultar interesante explorar el uso de estos bloques para mejorar los controles si se continúa usando la generación de código.

11.2.3 Mejoras en la monitorización de señales internas

Actualmente el control de velocidad al estar desplegado se comunica con MATLAB usando el topic `/signals`. Con la implementación actual, la muestra de mando llega una muestra retrasada. Al ser esta solo una señal de monitorización y no afectar en absoluto al control, no se le ha dado ninguna importancia y se ha sincronizado manualmente por código. Podría resultar interesante arreglarlo sacando la señal de mando del topic `/signals` y creando un topic aparte.

11.2.4 Uso de topic personalizado para PWM

Actualmente se está usado para mandar las señales PWM a los motores el mismo topic que se usaba para mandar las consignas de velocidad, el topic `/cmd_vel` en un mensaje es de tipo *Twist*. Este mensaje es un estándar para controlar la velocidad de los robots. Aunque en este caso solo estamos usando dos ejes del mensaje, este sirve también para controlar otros robots con más niveles de libertad como un dron.

11.2.5 Migración del control desplegado a C++

El control de velocidad usado en este proyecto ha sido creado usando las características de generación de código de MATLAB. Esto provoca que el código generado sea más ineficiente que si se hubiese codificado manualmente. El código generado una gran cantidad de punteros innecesarios y realiza el control sin usar los algoritmos clásicos de control digital.

12
Bibliografía

[1] «TurtleBot3». https://emanual.robotis.com/docs/en/platform/turtlebot3/bringup/#bringup (accedido 2 de abril de 2022).

[2] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, y W. Woodall, «Robot Operating System 2: Design, architecture, and uses in the wild», *Sci. Robot.*, vol. 7, n.º 66, p. eabm6074, may 2022, doi: 10.1126/scirobotics.abm6074.

[3] «Objetivos y metas de desarrollo sostenible - Desarrollo Sostenible». https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/ (accedido 29 de junio de 2022).

[4] ROBOTIS, «OpenCR», *ROBOTIS e-Manual*. https://emanual.robotis.com/docs/en/parts/controller/opencr10/ (accedido 25 de junio de 2022).

[5] *OpenCR: Open Source Control Module for ROS*. ROBOTIS, 2022. Accedido: 25 de junio de 2022. [En línea]. Disponible en: https://github.com/ROBOTIS-GIT/OpenCR

[6] *OpenCR HARDWARE: Open Source Control Module for ROS*. ROBOTIS, 2022. Accedido: 25 de junio de 2022. [En línea]. Disponible en: https://github.com/ROBOTIS-GIT/OpenCR-Hardware

[7] «ARM®-based Cortex®-M7 32b MCU+FPU, 462DMIPS, up to 1MB Flash/320+16+ 4KB RAM, USB OTG HS/FS, ethernet, 18 TIMs, 3 ADCs, 25 com itf, cam & LCD», p. 227.

[8] «Concepts — ROS 2 Documentation: Foxy documentation». https://docs.ros.org/en/foxy/Concepts.html (accedido 24 de junio de 2022).

[9] «Tutorials — ROS 2 Documentation: Foxy documentation». https://docs.ros.org/en/foxy/Tutorials.html (accedido 24 de junio de 2022).

[10] «Get Started with ROS - MATLAB & Simulink - MathWorks España». https://es.mathworks.com/help/ros/ug/get-started-with-ros.html (accedido 24 de junio de 2022).

[11] «ROS2 Simulink Toolbox not publishing -». https://es.mathworks.com/matlabcentral/answers/607606-ros2-simulink-toolbox-not-publishing?s_tid=mlc_lp_leaf (accedido 24 de junio de 2022).

[12] «Exchange Data with ROS 2 Publishers and Subscribers - MATLAB & Simulink - MathWorks España». https://es.mathworks.com/help/ros/ug/exchange-data-with-ros-2-publishers-and-subscribers.html (accedido 24 de junio de 2022).

[13] «ROS in Simulink - MATLAB & Simulink - MathWorks España». https://es.mathworks.com/help/ros/ros-in-simulink.html?s_tid=CRUX_lftnav (accedido 24 de junio de 2022).

[14] «Funciones». https://es.mathworks.com/help/referencelist.html?type=function&capability=codegen&listtype=alpha (accedido 24 de junio de 2022).
[15] «Controlador PID - Control Automático - Picuino». https://www.picuino.com/es/control-pid.html (accedido 24 de junio de 2022).
[16] F. L. Pagola, *Regulación Automática*.
[17] «Apuntes de Control Pid | PDF | Sistema de control | Teoría de sistemas», *Scribd*. https://es.scribd.com/doc/23817781/Apuntes-de-Control-Pid (accedido 24 de junio de 2022).
[18] Y. Name, «ROBOTIS e-Manual ttl-communication», *ROBOTIS e-Manual*. https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/ (accedido 24 de junio de 2022).
[19] «velocity_controller_pi_gain.jpg (869×293)». https://emanual.robotis.com/assets/images/dxl/velocity_controller_pi_gain.jpg (accedido 24 de junio de 2022).
[20] M. Bestmann, J. Guldenstein, y J. Zhang, «High-Frequency Multi Bus Servo and Sensor Communication Using the Dynamixel Protocol», p. 14.
[21] «Custominzing TurtleBot3 OpenCR with one more additional Dynamixal Motor · Issue #744 · ROBOTIS-GIT/turtlebot3», *GitHub*. https://github.com/ROBOTIS-GIT/turtlebot3/issues/744 (accedido 24 de junio de 2022).
[22] «[TB3] ROS 2 Dashing Release - TurtleBot», *ROS Discourse*, 21 de agosto de 2019. https://discourse.ros.org/t/tb3-ros-2-dashing-release/10364 (accedido 24 de junio de 2022).
[23] *common_interfaces*. ROS 2, 2022. Accedido: 24 de junio de 2022. [En línea]. Disponible en: https://github.com/ros2/common_interfaces/blob/37ebe90cbfa91bcdaf69d6ed39c08859c4c3bcd4/geometry_msgs/msg/Twist.msg
[24] ROBOTIS, «Rc-100», *ROBOTIS e-Manual*. https://emanual.robotis.com/docs/en/parts/communication/rc-100/ (accedido 24 de junio de 2022).
[25] ROBOTIS, «ROBOTIS e-Manual Goal-pwm», *ROBOTIS e-Manual*. https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/ (accedido 24 de junio de 2022).
[26] ROBOTIS, «ROBOTIS e-Manual PWM Limit», *ROBOTIS e-Manual*. https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/ (accedido 24 de junio de 2022).
[27] M. E. El-Hawary, *Principles of Electric Machines with Power Electronic Applications*. IEEE, 2002. doi: 10.1109/9780470545645.
[28] M. Maximo, C. H. C. Ribeiro, y R. J. Afonso, «MODELING OF A POSITION SERVO USED IN ROBOTICS APPLICATIONS», 2017. https://www.semanticscholar.org/paper/MODELING-OF-A-POSITION-SERVO-USED-IN-ROBOTICS-Maximo-Ribeiro/f1c5a53cbb44188f35955416e4b6d9a240550f96 (accedido 24 de junio de 2022).
[29] Robotis, «ROBOTIS e-Manual», *ROBOTIS e-Manual*. https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/ (accedido 23 de junio de 2022).
[30] T. Kronauer, J. Pohlmann, M. Matthe, T. Smejkal, y G. Fettweis, «Latency Analysis of ROS2 Multi-Node Systems». arXiv, 11 de junio de 2021. Accedido: 23 de junio de 2022. [En línea]. Disponible en: http://arxiv.org/abs/2101.02074
[31] *ROBOTIS wiki e-Manual*. ROBOTIS, 2022. Accedido: 23 de junio de 2022. [En línea]. Disponible en: https://github.com/ROBOTIS-GIT/emanual
[32] T. Shafa, «Design, Model, and Control of a Low-Cost 3 Degree of Freedom Balancing Laminate Leg with an Actively Controlled Ankle Using Fundamental Controls Concepts», p. 55.

[33] Lab. Regulación, Automática, y 3ºGITI, *Proyecto 2 – Control PID diseñado por respuesta en frecuencia para navegación mediante seguimiento de pared*. 2021.

[34] «Feedback Control of a ROS-Enabled Robot Over ROS 2 - MATLAB & Simulink - MathWorks España». https://es.mathworks.com/help/ros/ug/feedback-control-of-a-ros2-robot.html (accedido 30 de junio de 2022).