UNIVERSIDAD
DE MÁLAGA

ESCUELA DE
INGENIERÍAS
INDUSTRIALES

ESCUELA DE INGENIERÍAS INDUSTRIALES

Departamento: Ingeniería de Sistemas y Automática

Área de Conocimiento: Ingeniería de Sistemas y Automática

TRABAJO FIN DE GRADO

CONTROL A BAJA VELOCIDAD DEL ROBOT MÓVIL RAMBLER BASADO EN LA PLATAFORMA ARDUINO

Grado en

Ingeniería Electrónica, Robótica y Mecatrónica

Autor: ANTONIO M. CUADROS RODRÍGUEZ

Tutor: ANTONIO J. MUÑOZ RAMÍREZ

Cotutor JAVIER SERÓN BARBA

MÁLAGA, septiembre de 2018

Resumen | **CONTROL EN BAJA VELOCIDAD DEL ROBOT MÓVIL RAMBLER MEDIANTE ARDUINO**

Autor: Antonio M. Cuadros Rodríguez
Tutor: Antonio J. Muñoz Ramírez.
Cotutor: Javier Serón Barba

Ingeniería Electrónica, Robótica y Mecatrónica
TRABAJO FIN DE GRADO

Palabras clave | Rambler, HBL1660, BLDC, Encoder, Arduino, Simulink, CAN, PID.

En el presente trabajo de fin de grado se desarrolla el control a baja velocidad del robot móvil Rambler. Su movimiento se realiza a través de cuatro motores sin escobillas de accionamiento directo (BLDC), los cuales se accionan a través de controladores HBL1660 de Roboteq. El esquema de control se diseña en el entorno Simulink, exportándose a una placa *Arduino due* encargada de comunicarse con los controladores mediante bus CAN.

Para ello, se partirá de un trabajo previo realizado sobre un banco de pruebas, el cual dispone de un motor BLDC y controlador HBL1660 idénticos a los del vehículo. En este entorno se desarrolla un primer esquema de control, comparando la viabilidad de los sensores hall incorporados en cada motor respecto a un encoder con transmisión por correa, visible en la Figura 1.

A continuación, se adaptan los esquemas desarrollados para utilizarse en el Rambler, con especial énfasis en la comunicación CAN. Con el fin de evitar daños al vehículo, se realiza un primer control sin estar este en contacto con el suelo. Una vez verificado el control del vehículo en el aire, se repite el procedimiento sobre el suelo del taller, desarrollándose un control final y comprobándose su respuesta. Durante todas las pruebas se mantiene activo un sistema de seguridad encargado de detener al vehículo en caso de ocurrir un fallo en la comunicación.

Por último, se realizan pruebas en distintos terrenos y situaciones, mostrándose especial interés en terrenos abruptos (Figura 2). Con estos datos se comparan los efectos del rozamiento y otras perturbaciones sobre el vehículo y se establecen unas directrices para adaptar el esquema de control a cada situación.

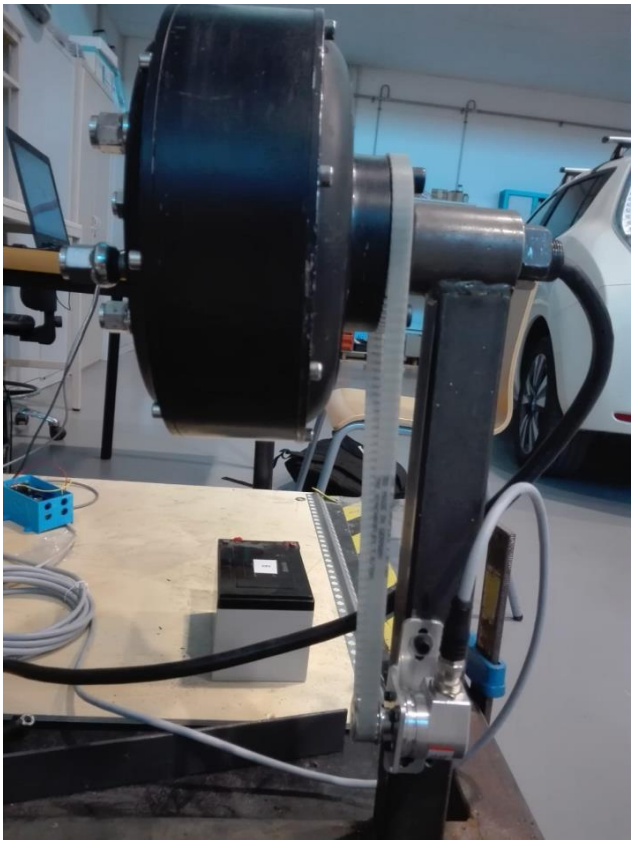

Figura 1: Banco de pruebas con encoder.

Figura 2: Realización de pruebas sobre terreno abrupto.

Abstract | **LOW-SPEED CONTROL OF THE RAMBLER MOBILE ROBOT USING ARDUINO.**

Author: Antonio M. Cuadros Rodríguez
Tutor: Antonio J. Muñoz Ramírez.
Cotutor: Javier Serón Barba

Electronics, Robotics and Mechatronics Engineering
End of Degree Project

Palabras clave | Rambler, HBL1660, BLDC, Encoder, Arduino, Simulink, CAN, PID.

In the present End-Of-Grade work, the low-speed control of the Rambler mobile robot is developed. Its movement is made through four brushless direct drive motors (BLDC), which are driven by Roboteq HBL1660 controllers. The control scheme is designed in the Simulink environment and exported to an Arduino board, which is responsible for communicating with the controllers via CAN bus.

To accomplish this, the starting point relies on previous work carried out on a test bench, which disposes of a BLDC motor and HBL1660 controller identical to those of the vehicle. In this environment, shown in Figure 1, a first control scheme is developed, testing the performance of the hall sensors versus an encoder with belt drive.

The diagrams are then adapted for the Rambler, with special emphasis on the CAN communication. To avoid any damage, a first control scheme is carried out with the vehicle elevated from the ground. Once the control in the air has been verified, the procedure is repeated with the Rambler on the workshop floor, developing the final control scheme and testing its response. For all these experiences, a safety system is kept active to stop the vehicle in the event of a communication failure.

Finally, tests are carried out in different terrains and situations, with a special interest in abrupt terrain (Figure 2). The data from these experiences is used to measure the effect of friction between other interferences and to establish guidelines to adapt the control scheme to each situation.

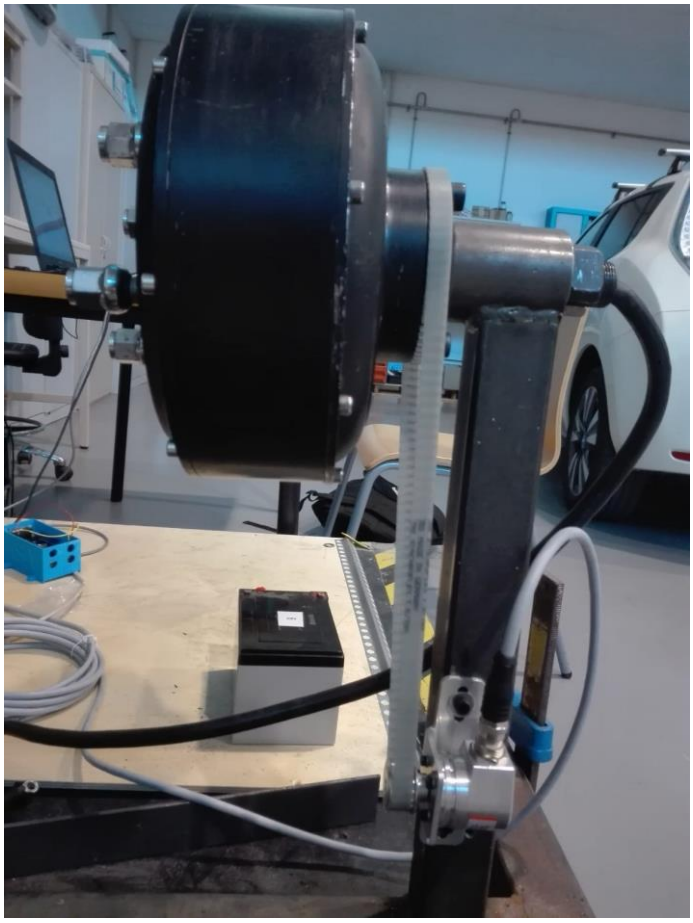

Figure 1: Test bench with encoder sensor.

Figure 2: Performing tests on abrupt terrain.

Índice

Índice de Figuras

Índice de tablas

1. INTRODUCCIÓN

1.1. Objeto

En este trabajo de fin de grado (TFG en adelante), se busca aplicar los conocimientos adquiridos en la mención de Robótica y Automatización del grado en ingeniería Electrónica, Robótica y Mecatrónica, haciendo hincapié en la identificación de sistemas y su posterior control. Para ello, se trabajará sobre el robot móvil Rambler, desarrollado por el departamento de Ingeniería de Sistemas y Automática (ISA) de la Universidad de Málaga [1].

Al trabajar sobre un sistema real se expondrán y desarrollarán habilidades tales como el análisis y resolución de problemas, críticas para llevar a cabo proyecto. Además, será necesaria la investigación y comprensión de nuevas tecnologías.

Se trabajará principalmente sobre el software Simulink de Mathworks, exportando los diseños a una placa Arduino [2]. Para el accionamiento de los motores se utilizarán controladores HBL1660 de Roboteq [3], configurándolos mediante el software Roborun+ proporcionado por su mismo fabricante.

1.2. Antecedentes

Al inicio de este TFG, el vehículo autónomo Rambler, tiene problemas para moverse de manera controlada a baja velocidad, especialmente al intentar realizar giros. Se han llevado a cabo distintos TFGs relacionados con este problema [4], [5], [6], en los cuales se han completado las siguientes tareas:

- Estudio e implementación de la comunicación vía CAN bus mediante Arduino para un único motor.
- Identificación de sistemas aplicado a un motor de pruebas externo al Rambler, para posteriormente implementar un control de velocidad en Simulink, basado en un PI de bucle cerrado.
- Modelado y simulación del vehículo en el entorno Simscape.
- Diseño e impresión de una caja protectora para la placa Arduino y su correspondiente Shield para la comunicación en bus CAN.
- Estudio e implementación de un sistema odométrico basado en el modelo cinemático, obteniéndose la posición y orientación a través de la velocidad de las ruedas.
- Inclusión de la odometría en el esquema de control, incluyendo el avance en metros y la orientación del robot.
- Comprobación de los dos puntos anteriores mediante simulaciones con un modelo 3D del vehículo en Simscape.
- Estudio y calibración externa al robot de la unidad inercial MPU-9250.

1.3. Objetivo

El objetivo principal de este proyecto es controlar el movimiento del Rambler a baja velocidad, consiguiéndose maniobrar de manera controlada.

Como objetivos complementarios, se buscará mejorar la odometría mediante el estudio de los sensores codificadores de posición angular ya presentes en las ruedas y la posible

incorporación de otros encoders. Además, se estudiará el efecto del terreno en el control implementado.

Para cumplir estos objetivos, se instalará una placa Arduino due, la cual se comunicará con los diversos sensores y con los controladores de las ruedas mediante CAN bus.

1.4. Metodología

Inicialmente se trabajará con la rueda del banco de pruebas:

1. Comunicación y control básico de una rueda mediante el software Roborun+.
2. Comunicación vía CAN.
3. Control básico de una rueda a través de Simulink.
4. Ecualizado de la rueda y esquema de control realimentado en Simulink.
5. Incorporación de los encoders al modelo y diseño del esquema final de control.

Una vez se obtengan resultados satisfactorios, se comenzará a trabajar sobre el robot:

7. Adaptación de la comunicación CAN a las 4 ruedas.
8. Estudio y resolución vía software de casos singulares.
9. Ecualizado y control de una rueda con el vehículo en el aire.
10. Adaptación del control a las 4 ruedas con el vehículo en el aire.
11. Identificación del sistema y control con el vehículo en el suelo del taller.
12. Pruebas en distintos terrenos y situaciones.

1.5. Estructura

Este trabajo se estructura en 9 capítulos, incluyendo el presente capítulo de introducción, la bibliografía consultada y un capítulo de anexos con el código desarrollado. El resto de los capítulos cumplen la siguiente función:

- En el capítulo dos se realiza una breve descripción de los componentes que forman el sistema y sus características técnicas.
- Las comunicaciones son tratadas en el capítulo tres, dándose unas nociones básicas del protocolo CAN. Se detalla el trabajo previo existente y las adaptaciones realizadas al mismo para su uso en el Rambler, tales como un filtrado de direcciones y la incorporación de un sistema de seguridad.
- En el cuarto capítulo se realiza la identificación y control del motor del banco de pruebas, comparándose el proceso y los resultados obtenido con el sensor hall respecto a el encoder instalado.
- A continuación, en el capítulo quinto se lleva a cabo la identificación y control del Rambler, llevando a cabo un primer control con el vehículo en el aire y otro posterior en el suelo del taller, detallándose las diferencias entre ambas situaciones.
- Una vez conseguido el control, en el capítulo seis se realizan pruebas en distintos terrenos y situaciones, estudiándose el efecto de realizar giros y transitar sobre distintos terrenos en la respuesta del control desarrollado.
- Por último, en el capítulo siete aparecen reflejadas las conclusiones y líneas para continuar el trabajo.

2. DESCRIPCIÓN DE LAS PARTES DEL SISTEMA

2.1. Motor sin escobillas CC

Los motores sin escobillas de corriente continua (BLDC), no necesitan las escobillas de los motores tradicionales, reemplazándolas por un microcontrolador, siendo este el encargado de realizar el cambio de polaridad secuencial en las bobinas, provocando el giro del rotor.

Este tipo de motores, disponen de imanes permanentes en el rotor, relegando el bobinado únicamente al estator. Además, suelen incorporar tres sensores de efecto Hall, los cuales al estar desfasados 120º sirven para determinar la posición del motor. En la Figura 2.1, se puede apreciar el motor en conjunto.

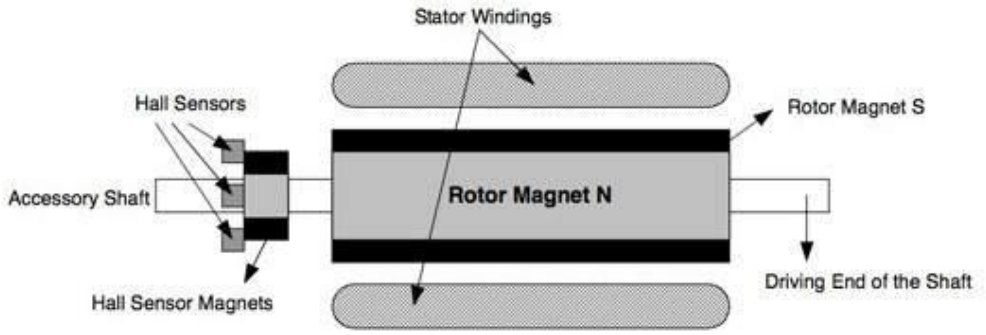

Figura 2.1: Partes de un motor BLDC.

Al no disponer de escobillas, estos motores presentan ciertas ventajas respecto a los motores tradicionales:

- Mayor eficiencia y rendimiento al disminuir perdidas.
- Mayor rango de velocidad al no tener el rozamiento propio de las escobillas.
- Menor mantenimiento y mayor vida útil.
- Menor tamaño para la misma potencia.

Sin embargo, suelen ser más caros y necesitan de un microcontrolador para su correcto control.

Para hacer girar el eje, se hace pasar una corriente eléctrica por los bobinados del estator, dicha corriente genera un campo electromagnético el cual genera un par de fuerzas electromagnéticas al interaccionar con el campo magnético del rotor.

Para conseguir una revolución completa, el microcontrolador debe realizar una secuencia de conmutación de seis pasos, la cual sigue el orden dispuesto en la Figura 2.2. Durante este proceso, los sensores de efecto Hall proporcionaran información del giro, dándose un cambio de polaridad en uno de ellos cada 60º. Durante el giro, el controlador utilizara dicha información para conocer la posición actual del rotor y continuar la secuencia correctamente. En la Figura 2.3 se puede apreciar el proceso completo.

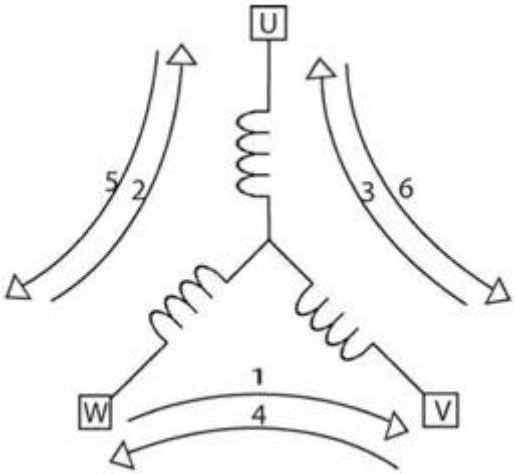

Figura 2.2: Secuencia de alimentación del bobinado.

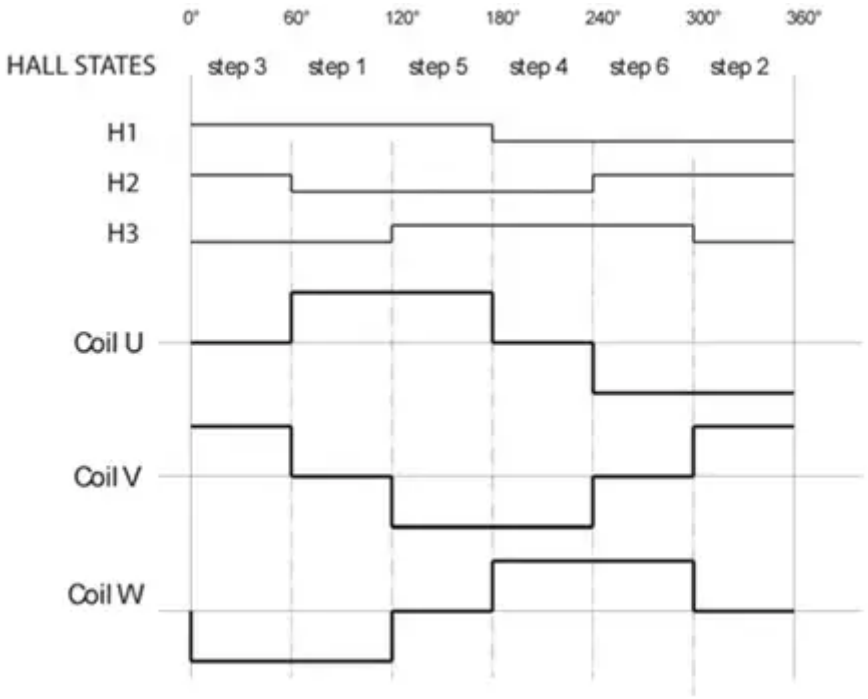

Figura 2.3: Secuencia de los sensores Hall.

El motor utilizado en este proyecto es un motor BLDC trifásico de corriente continua de 23 polos magnéticos. Dispone de dos juegos independientes de sensores Hall, cada juego de 3 sensores Hall desfasados 120º, sin embargo, únicamente se utilizará uno de ellos.

2.2. Controladores Roboteq HBL1660

El dispositivo Roboteq HBL1660 es un controlador de alto rendimiento para motores DC sin escobillas equipados con sensores Hall. El controlador utiliza la información de

los sensores Hall para llevar a cabo el control del motor, además calcula la velocidad y la distancia recorrida por el mismo.

El controlador puede ser configurado, monitorizado y sintonizado en tiempo real a través del software Roborun+, proporcionado por Roboteq. Para comunicar el HBL1660 con programa en el PC, se utiliza el protocolo serie, ya sea mediante USB o por comunicación en serie (RS232).

Figura 2.4: Controlador Roboteq HBL1660.

El voltaje máximo admitido de alimentación es de 60 V y la corriente máxima estable proporcionada al motor es de 100 A, permitiéndose picos de hasta 150 A. Con este modelo se puede controlar un único motor, ya que solo tiene un canal de salida trifásica.

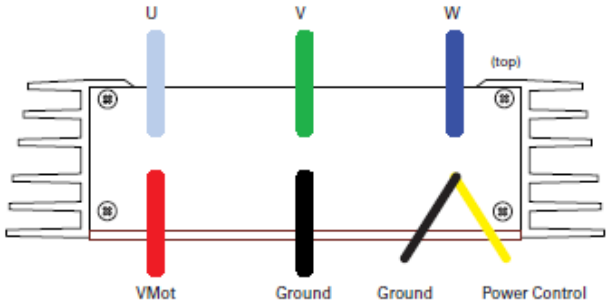

Figura 2.5: Alimentación y etapa de potencia en la parte posterior del controlador.

Para la comunicación y control del motor, dispone de entradas de distintos tipos: analógicas, digitales, PWM, incluyendo soporte para encoders de cuadratura, los cuales permiten algoritmos de control más complejos, ya sea en lazo abierto o cerrado.

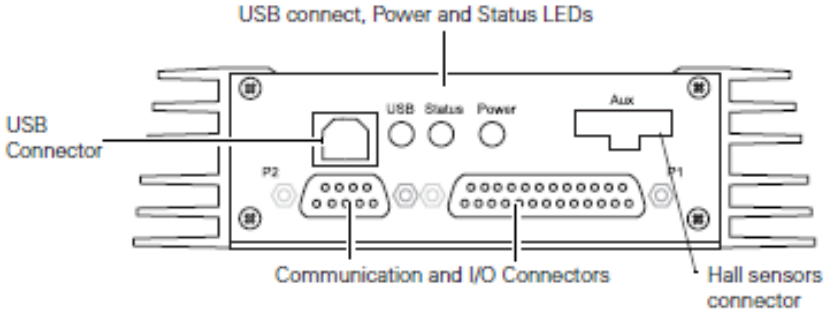

Figura 2.6: Puertos de comunicación en la parte frontal del controlador.

Para conectar los sensores de efecto Hall, se utiliza un conector Molex Microfit 3.0, ref. 43645, de 5 pines, como el de la Figura 2.7**.** En la Figura 2.8 se indica el patillaje esperado por el controlador.

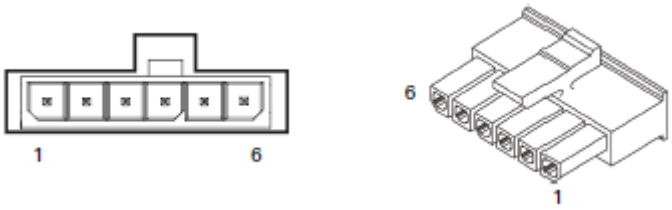

Figura 2.7: Conector de los sensores Hall.

Pin Number	**1**	**2**	**3**	**4**	**5**	**6**
Signal	5V	Reserved	Hall C	Hall B	Hall A	Ground

Figura 2.8: Patillaje de los sensores Hall

Para la comunicación por puerto serie (RS232), se utilizará el conector principal, presente en la Figura 2.9, mientras que la comunicación por bus CAN se llevará a cabo mediante el puerto secundario, Figura 2.10. Más adelante se indicarán los puertos usados en cada conector, así como su función.

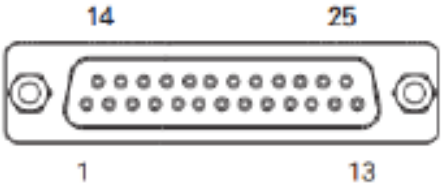

Figura 2.9: Puerto principal.

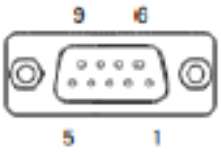

Figura 2.10: Puerto secundario.

Por último, incluye distintas medidas de protección para el motor, permitiendo cortar la corriente al motor en cualquier momento, ya sea manualmente, mediante un script o externamente a través de cualquiera de sus entradas digitales. En caso de error, el controlador informa a través de la aplicación y mediante el parpadeo del LED de estado, siguiendo el patrón representado en la Figura 2.11 para cada mensaje.

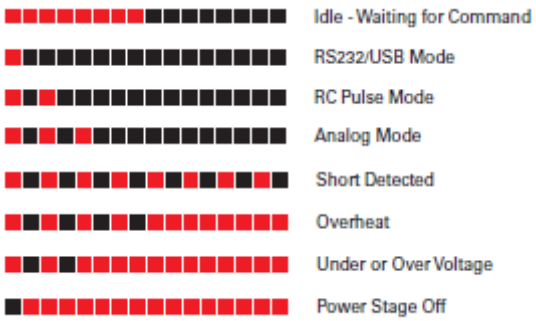

Figura 2.11: Codificación del LED de estado.

2.2.1. Modos de entrada

El HBL1660 acepta comandos procedentes de:

- Comunicación en serie: USB o RS232.
- Por pulsos: radio RC o PWM
- Entrada analógica (0 a 5 V).
- Bus CAN.

Este software permite al usuario establecer prioridades entre los distintos tipos de entrada. Durante el funcionamiento, el controlador utiliza la instrucción procedente del protocolo activo de mayor prioridad. En caso de no detectar ningún modo activo, envía un valor por defecto, establecido previamente por el usuario.

Los protocolos se consideran activos siguiendo los siguientes criterios:

- Modo serie: El comando debe comenzar con el carácter "!" y debe llegar dentro del tiempo de espera.
- Modo pulso: Se recibe un tren de pulsos válidos y mantenido en el tiempo.
- Modo analógico: Se considera siempre activo, se debe deshabilitar manualmente a través de la aplicación Roborun+. Si la se recibe un valor que supere el rango de seguridad establecido, el modo se deshabilita.
- Modo CAN: Se habilita manualmente mediante la aplicación. Debe tenerse en cuenta que el protocolo USB y el bus CAN no pueden operar al mismo tiempo, teniendo prioridad el USB en todo momento.

2.3. Arduino Due

Arduino Due es un microcontrolador basado en el chip Atmel SAM3X8E. Dispone de 54 pines E/S digitales, de los cuales 12 pueden usarse como salidas PWM. Además, incluye 12 entradas analógicas, 4 puertos UARTs y 2 puertos DAC (Digital a analógico).

Debe tenerse en cuenta que la placa trabaja a 3.3 V, por lo que aplicar voltajes mayores a cualquiera de sus pines puede provocar daños en la placa. Además, cada pin puede proporcionar entre 3 mA – 15 mA, o recibir de 6 mA – 9 mA.

Figura 2.12: Placa Arduino Due.

La placa Arduino incorpora dos puertos USB, tal y como se muestra en la Figura 2.13**.** El primer puerto (*Native USB*) está conectado directamente al microcontrolador SAM3X. El segundo puerto (*Programming*) va al chip ATMEGA16U2, el cual actúa como un conversor USB a Serie, se utilizará este puerto para cargar los programas y comunicar el PC con la placa Arduino.

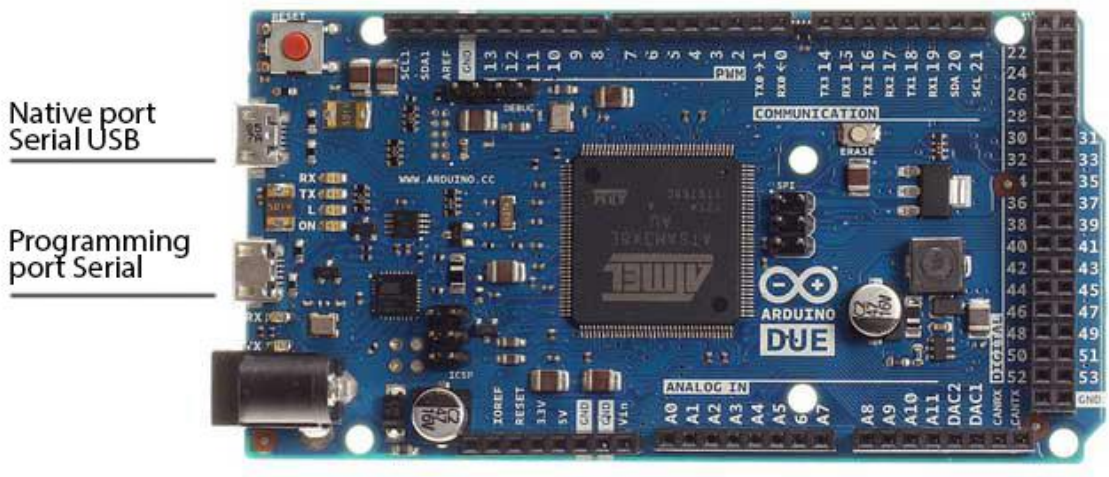

Figura 2.13: Puertos USB.

Por último, la placa dispone de un conector de 6 pines en el centro de la placa para la comunicación SPI (Serial Peripheral Interface). Este protocolo se utiliza para trasferir datos con los distintos circuitos integrados disponibles para Arduino.

En la Figura 2.14 se muestra un esquema completo de la palca [7], indicándose la respectiva función de cada pin.

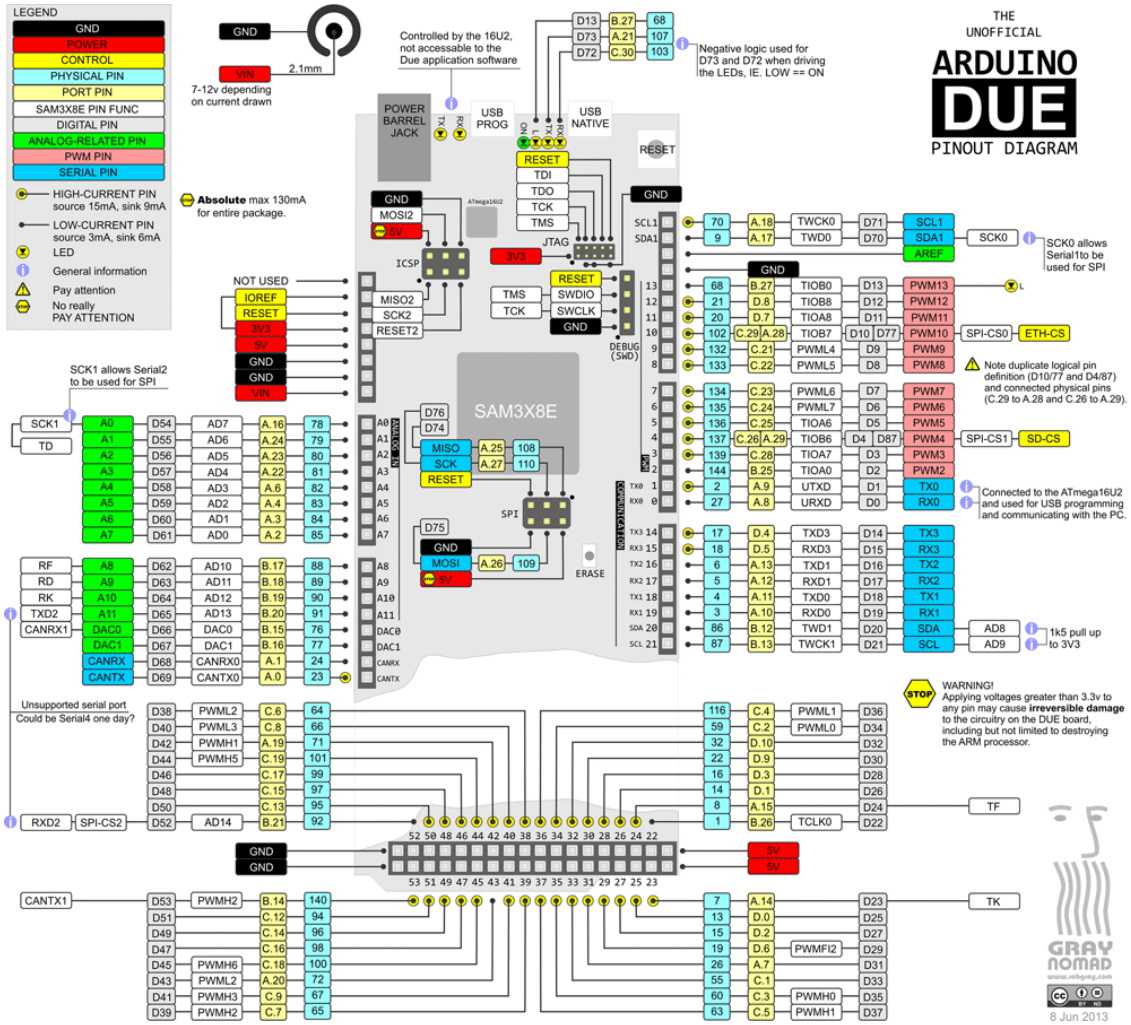

Figura 2.14: Diagrama de pines de Arduino Due.

2.4. Can Bus Shield 1.2V

Para realizar la comunicación por bus CAN se ha recurrido al circuito modular externo CAN-BUS Shield V1.2 de Seeed Studio, el cual incorpora el chip MCP2515, encargado de controlar el bus CAN. Sus características son las siguientes:

- Implementa CAN V2.0B de hasta 1Mb/s.
- Filtrado de direcciones: 2 máscaras y 6 filtros.
- Interfaz SPI de hasta 10 MHz.
- Tramas de datos estándar, con identificador de 11 bits, y extendida con 29 bits de identificador.
- Conector industrial estándar DB-9.
- LEDs indicadores.

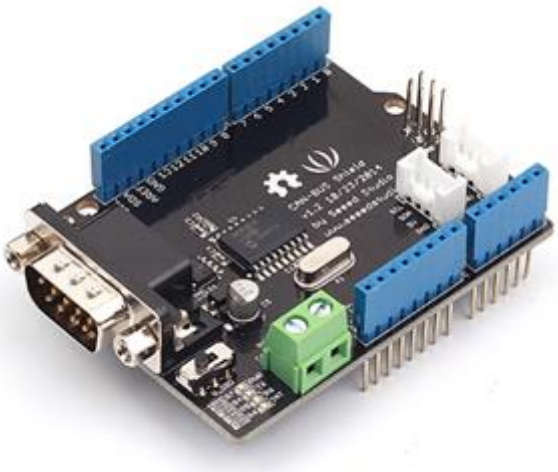

Figura 2.15: CAN Bus Shield V1.2.

Este módulo se acopla en la parte superior del Arduino, realizando la comunicación con el mismo a través del bus SPI. Este bus se utiliza principalmente para comunicar circuitos integrados entre sí, mediante 4 pines:

- MOSI (Master Out Slave In): Envío de datos del maestro.
- MISO (Master In Slave Out): Envío de datos del esclavo.
- CLK: Señal de reloj.
- CS (Chip Select): Activa un esclavo determinado.

A la hora de programar la comunicación, hay que tener en cuenta que el pin 9 del SPI de Arduino se corresponde con el pin de Chip Select, encargado de activar el módulo CAN.

Para la comunicación CAN, se utilizarán únicamente los pines CAN_H, CAN_L y GND del conector DB9 del *shield*, los cuales se aprecian en la Figura 2.16. En este proyecto se utilizará inicialmente para conectar la placa Arduino al controlador HBL1660, y posteriormente al bus CAN del Rambler, al cual están conectados los controladores de las 4 ruedas.

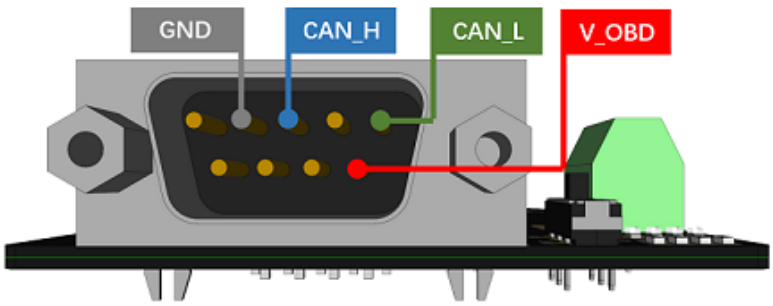

Figura 2.16: Patillaje del conector DB9.

3. COMUNICACIÓN

3.1. Protocolo CAN

Para llevar a cabo la comunicación entre Arduino y los controladores Roboteq, utilizaremos el protocolo CAN, estandarizado y desarrollado por Bosch. Es un protocolo serie, bidireccional de dos hilos, asíncrono, de acceso múltiple y con detección de colisiones.

Su característica más importante es la prioridad de mensajes, dándose mayor importancia a los mensajes de ciertos dispositivos frente a otros, para conseguir esto, el protocolo asigna a cada nodo un identificador (ID), incluyendo esta información en cada mensaje enviado por el mismo. Cada identificador dispone de un nivel de prioridad en el bus, por lo que, en caso de colisión, el nodo de mayor prioridad toma el control del bus, inhibiendo cualquier mensaje proveniente de un nodo de menor prioridad. Además, los identificadores permiten filtrar mensajes, ya que, aunque los mensajes llegan a todos los nodos, estos pueden comprobar el ID al que va dirigido el mensaje, decidiendo en consecuencia si procesarlo o desecharlo.

A nivel físico, la comunicación se realiza a través de dos hilos: CAN_L (*Low*) y CAN_H (*High*). Según el nivel de tensión de los dos cables se definen dos estados: dominante y recesivo. En el estado dominante, ambos hilos tienen el mismo nivel de tensión, en el recesivo, CAN_H aumenta su tensión y CAN_L la disminuye, generando entre ambos una diferencia de potencial de al menos 1.5V. En la Figura 3.1 se representan ambos estados. Para evitar

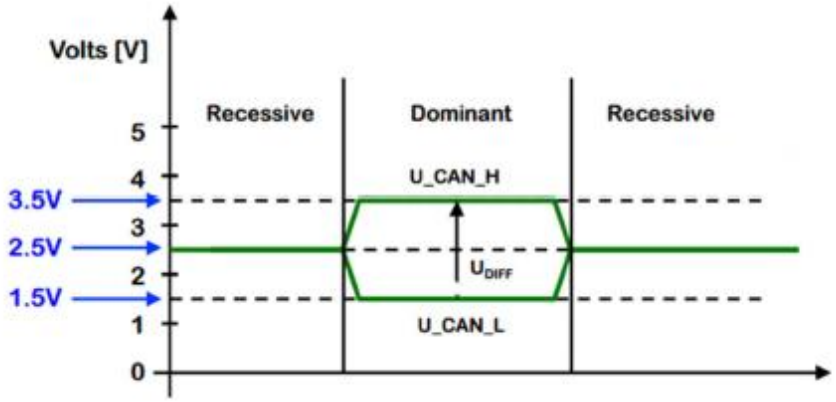

Figura 3.1: Estados del bus CAN.

Los mensajes siguen un patrón establecido, diferenciándose únicamente en la longitud de sus identificadores, existiendo dos tipos de tramas:

- Estándar: con identificador de 11 bits.
- Extendida: con identificador de 29 bits.

En este proyecto se utilizarán tramas de tipo estándar, ya que el número de identificadores requeridos es muy reducido. Los mensajes siguen la estructura indicada en la Figura 3.2.

Figura 3.2: Estructura de una trama CAN.

Cada sección cumple con una función específica:

- Comienzo del mensaje.
- Arbitraje: Contiene la identificación del mensaje, así como su prioridad.
- Control: Indica el número de bytes que se envía en la trama.
- Datos: Datos del mensaje, con un máximo de 8 bits por mensaje.
- CRC: Indica el correcto envío de los datos.
- ACK: Confirma la correcta recepción de los datos.
- Final del Mensaje.

3.1.1. Protocolo CAN en los controladores

El controlador HBL1660 incorpora el protocolo CAN, permitiendo controlar hasta 127 dispositivos. Para ello dispone de 3 modos distintos de operación, configurables en la aplicación Roborun+:

- RawCAN.
- MiniCAN.
- CANopen.

Se utilizará por su sencillez el modo MiniCAN, el cual es una simplificación del estándar CANopen. La configuración se muestra en la Figura 3.3.

- Bit Rate: Velocidad de transmisión, configurada a 1000 kbits/s.
- Node ID: Dirección que utiliza la controladora a la hora de enviar mensajes.
- Listen Node ID: Dirección que utiliza la controladora para recibir tramas, si se utiliza el valor 0 aceptara todas las tramas del bus.
- MiniCAN Send Rate (ms): Tiempo de muestreo de los mensajes en milisegundos, debe ser igual utilizado en Simulink, fijado en 10 milisegundos.

Para una comunicación efectiva, es importante que los valores de *Node ID* y *Listen Node ID* coincidan, de modo que cada controlador tenga una dirección única y diferenciada.

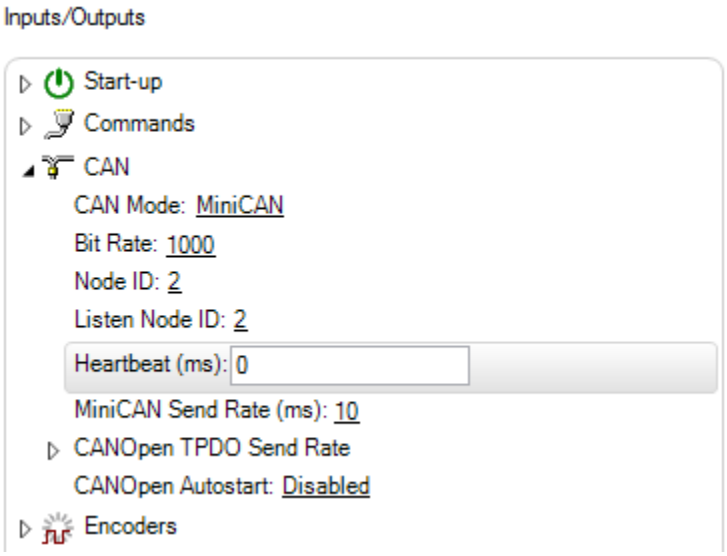

Figura 3.3: Configuración MiniCAN.

El controlador dispone de distintas variables de tipo entero para almacenar datos, cada variable dispone de 11 bits de dirección (4 bits de cabecera y 7 bits de NodoID), los cuales coinciden con tramas del estándar CANopen. El formato de los datos viene predefinido por esta dirección, debiéndose diferenciar entre variables de trasmisión o recepción de datos.

3.1.1.1. TRANSMISIÓN DE DATOS

Para transmitir datos, se dispone de 8 variables enteras y 32 booleanas, repartidas en 4 tipos de tramas del modo que se indica en la Tabla 1, cada una con una dirección predefinida:

- TPDO1: 0x180 + NodoID.
- TPDO2: 0x280 + NodoID.
- TPDO3: 0x380 + NodoID.
- TPDO4: 0x480 + NodoID.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
TPDO1	VAR1				VAR2			
TPDO2	VAR3				VAR4			
TPDO3	VAR5		VAR6		VAR7		VAR8	
TPDO4	BVar 1-8	BVar 9-16	BVar 17-24	BVar 25-32				

Tabla 1: Tramas de transmisión.

Para transmitir datos desde el controlador, se utiliza en el script la función setcommand(_VAR, n) o setcommand(_B, n), donde 'n' es el número de la variable. Nótese que se envía primero el bit menos significativo.

3.1.1.2. RECEPCIÓN DE DATOS

La filosofía es exactamente igual a la de trasmisión, cambiando únicamente las variables reservadas, presentes en la Tabla 2, y las direcciones de sus respectivas tramas:

- RPDO1: 0x200 + NodoID.
- RPDO2: 0x300 + NodoID.
- RPDO3: 0x400 + NodoID.
- RPDO4: 0x500 + NodoID.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
RPDO1	VAR9				VAR10			
RPDO2	VAR11				VAR12			
RPDO3	VAR13		VAR14		VAR15		VAR16	
RPDO4	BVar 33-40	BVar 41-48	BVar 49-56	BVar 57-64				

Tabla 2: Tramas de recepción.

En este caso, se pueden almacenar datos utilizándola función getvalue(_VAR, n) o getvalue(_B, n).

3.2. Trabajo previo – Banco de pruebas.

3.2.1. Comunicación Arduino – Controlador.

La comunicación entre Arduino y el motor del banco de pruebas mediante bus CAN en Simulink, se estudió e implemento en un TFG anterior [4]. Para llevar a cabo esta tarea se ha programado dos S-Function en Simulink, utilizando la librería proporcionada por el fabricante del Shield CAN [8].

En estos scripts, se codifican y decodifican las tramas y variables según el protocolo ya explicado en el capítulo 3.1.1, utilizándose las variables VAR9 y VAR10 en el envío (*SendMiniCANVAR9VAR10*) y VAR1, VAR2 VAR3 Y VAR4 en la recepción de datos (*ReceiveMiniCANVAR1VAR2VAR3VAR4*). Inicialmente se trabaja únicamente con las variables VAR9, VAR1 y VAR2, siendo el esquema de simulink el presente en la Figura 3.4.

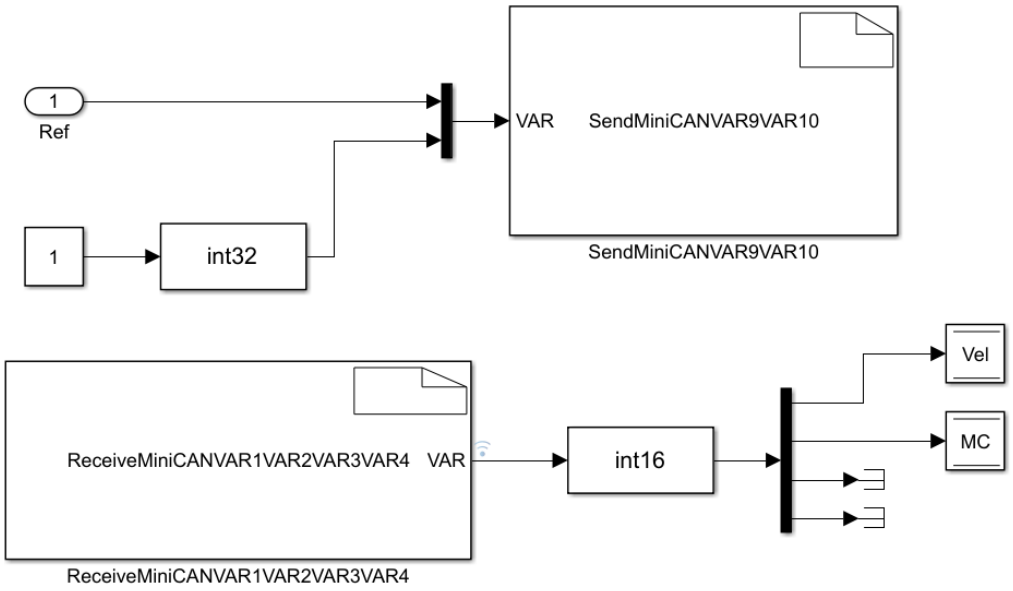

Figura 3.4: Bloque para la comunicación CAN - Banco de pruebas.

Además, se dispone de un script para el propio controlador HBL1660, encargado de recibir y aplicar los datos recibidos por el bus y posteriormente enviar los datos obtenidos por el mismo cada 10 ms.

3.2.2. Comunicación PC – Arduino.

Para realizar la comunicación entre el PC y Arduino se dispone de un esquema de simulink para cada uno de ellos, presentados en las Figura 3.5 y Figura 3.6 respectivamente:

- El PC adapta la referencia a 8 bits, proporcionando un rango de ±126 valores. A continuación, los envía a Arduino a través del puerto serie, mediante el cual recibe los resultados obtenidos.

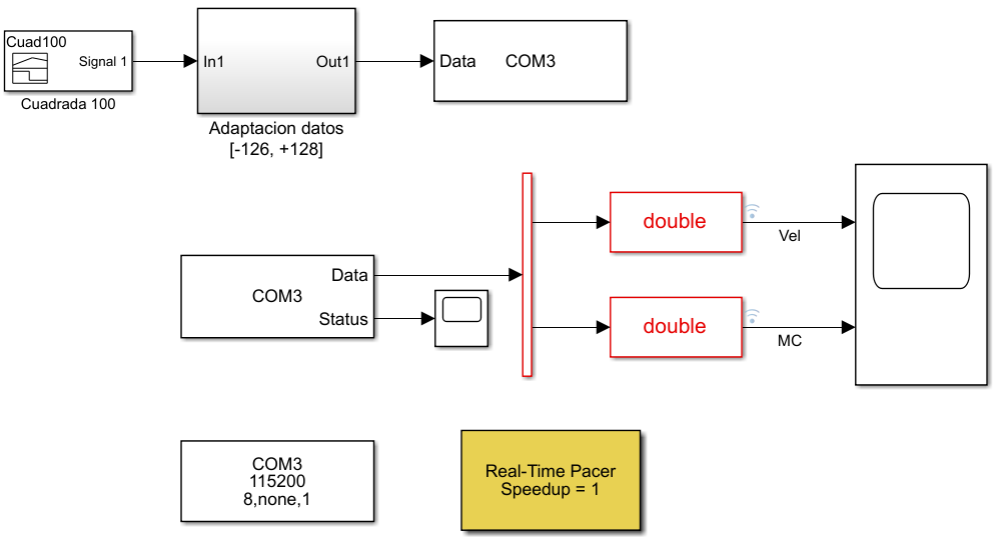

Figura 3.5: Esquema Simulink PC - Identificación.

- Arduino recibe los datos de 8 bits y los decodifica para obtener la referencia original. Estos datos se envían al controlador mediante el bus CAN, el cual responde con la velocidad y el comando motor utilizado para posteriormente, enviarlos al PC por puerto serie, mediante el bloque detallado en la Figura 3.7.

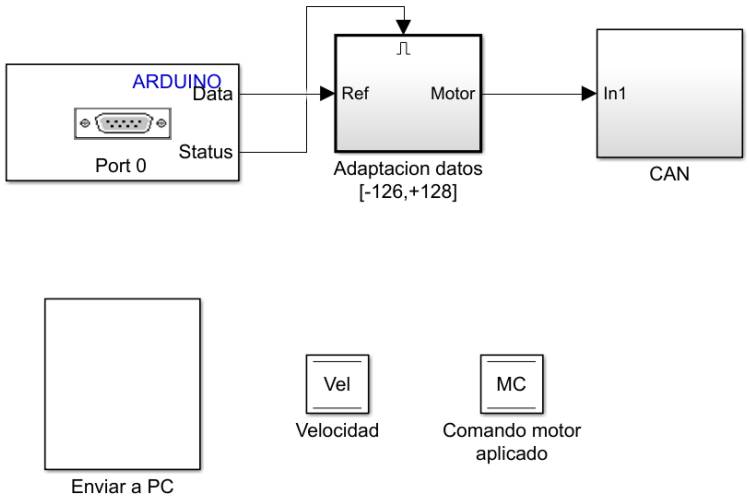

Figura 3.6. Esquema Simulink Arduino - Identificación.

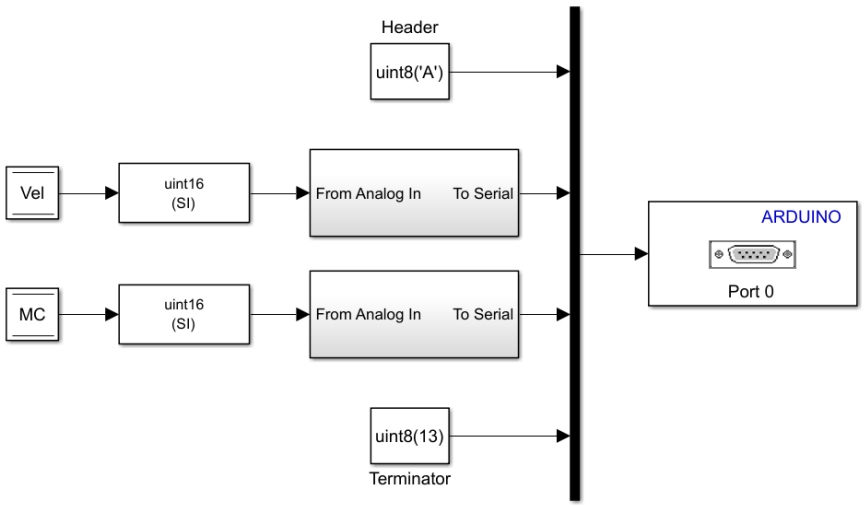

Figura 3.7: Detalle bloque "Enviar a PC".

3.3. Rambler

3.3.1. Adaptación a 4 ruedas.

Adaptar la comunicación del banco de pruebas al Rambler no es trivial, ya que pasamos de comunicarnos con un único motor aislado a trabajar con varios dispositivos al unísono. La solución más obvia sería replicar 4 veces las S-Funtions de Simulink, sin embargo, Matlab trata las variables utilizadas en el código de estos bloques como globales, por lo que esta opción genera múltiples errores de compilación. Disponemos de distintas alternativas:

- Comunicarnos solo con un controlador y que este se comunique con el resto de los controladores.
- Implementar las variables mediante DWork vectors, los cuales permiten tener variables estáticas dentro de cada función.
- Adaptar los bloques y su código para trabajar con 4 direcciones distintas.

Se llevará a cabo esta última opción, ya que apenas requiere modificaciones sobre el código existente y funcional. Se han realizado las siguientes modificaciones:

- Envió de datos: El nuevo bloque utiliza un bucle *for* para enviar los datos a cada dirección. El código completo se encuentra en el anexo SendNode2.
- Recepción de datos: Al recibir un mensaje comprueba el tipo de trama y su procedencia comparando su dirección mediante bucles *if-else*. En caso de no coincidir con los mensajes esperados, se descarta la información. El código completo se encuentra en el anexo SendNode2ReceiveNode2.

Las direcciones se pueden modificar fácilmente desde el propio bloque, ya que están configuradas como un parámetro del bloque. Para asegurar el funcionamiento del código, las direcciones deben tener un valor numérico ascendente, tal y como se indica en la Figura 3.8.

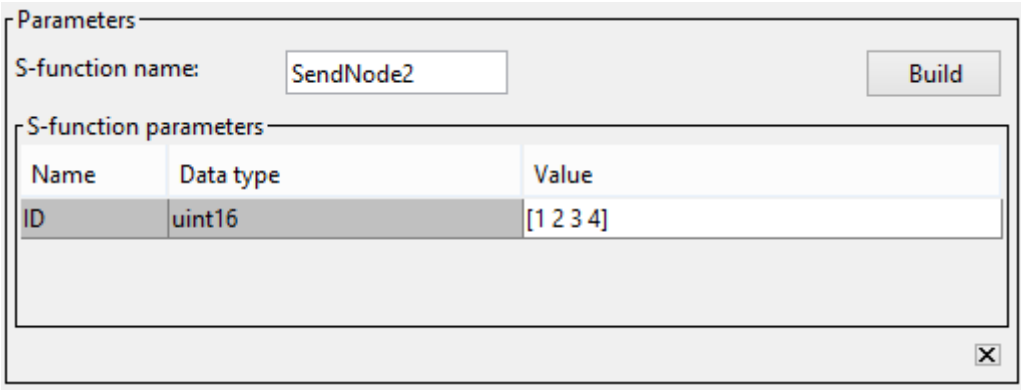

Figura 3.8: Configuración de direcciones CAN en Simulink.

Debe tenerse en cuenta que los motores de cada lado están enfrentados, lo que implica sentidos de giro distintos. Para que el vehículo avance al aplicar una referencia positiva, se invierte el signo de la entrada y salida de los motores del lado derecho. El esquema de direcciones para cada rueda es el siguiente:

- Nodo 1: Derecha trasera.
- Nodo 2: Derecha delantera.
- Nodo 3: Izquierda delantera.
- Nodo 4: Izquierda trasera.

En la Figura 3.9 se dispone el esquema final del bloque “Bus can”, encargado de enviar y recibir los datos del bus CAN en Arduino, incluyendo la entrada del contador de seguridad explicado en el capítulo Sistema de seguridad.

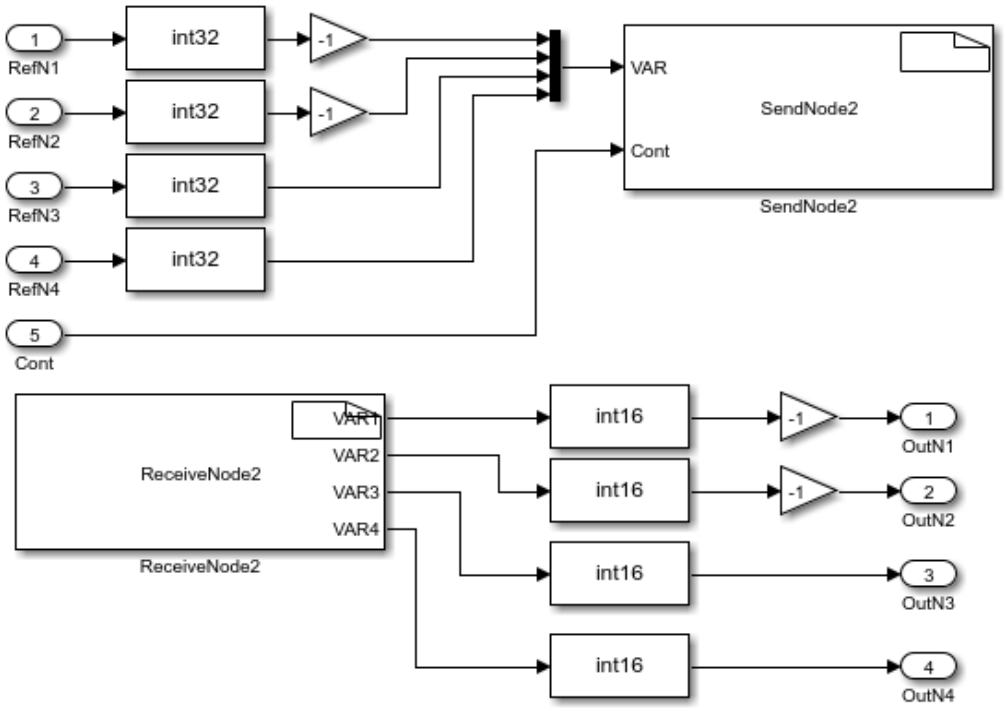

Figura 3.9: Bloque para la comunicación CAN - Rambler.

3.3.2. Conexionado Rambler – Arduino.

El vehículo Rambler utiliza un conversor "USB to CAN compact" para adaptar el bus CAN a USB y realizar las labores de control desde el PC integrado, esta conexión deja de ser necesaria al realizar el control a través de Arduino. Para comunicar la placa con los controladores se dispone de un conector adaptado a este caso, el cual sigue el esquema de la Figura 3.10.

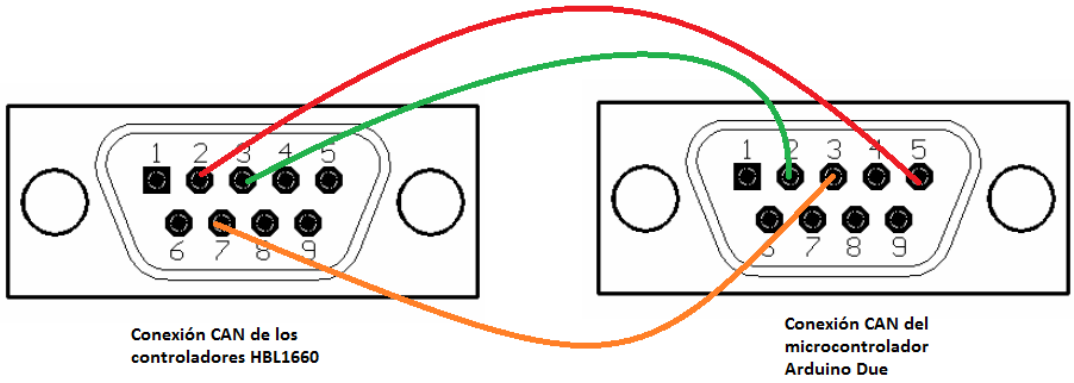

Figura 3.10: Conexionado del conector.

Sin embargo, al probar los nuevos esquemas no se consigue hacer llegar ningún mensaje a los controladores. Al comprobar el cable, se descubre que se han soldado los pines equivocados, no coincidiendo en el conector el cableado de ambos extremos, apreciándose en detalle en la Figura 3.11. Este error surge probablemente de realizar la

soldadura utilizando el esquema de la Figura 3.10 con el conector al revés. Una vez soldado correctamente se consigue enviar y recibir datos sin problema.

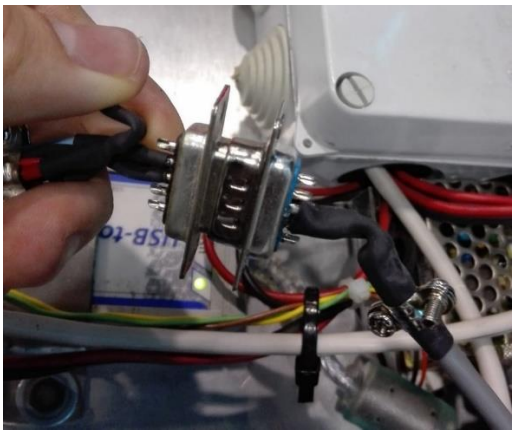

Figura 3.11: Conexionado erróneo del conector.

3.3.3. Filtrado de direcciones.

A la hora de probar la comunicación, esta falla al recibir los datos de cada controlador en Arduino. La referencia llega correctamente a la controladora Roboteq, siendo esta la presente en la Figura 3.12, sin embargo, en Arduino se reciben por el bus CAN los datos de la Figura 3.13, donde se aprecia perdida de datos durante periodos cortos de tiempo.

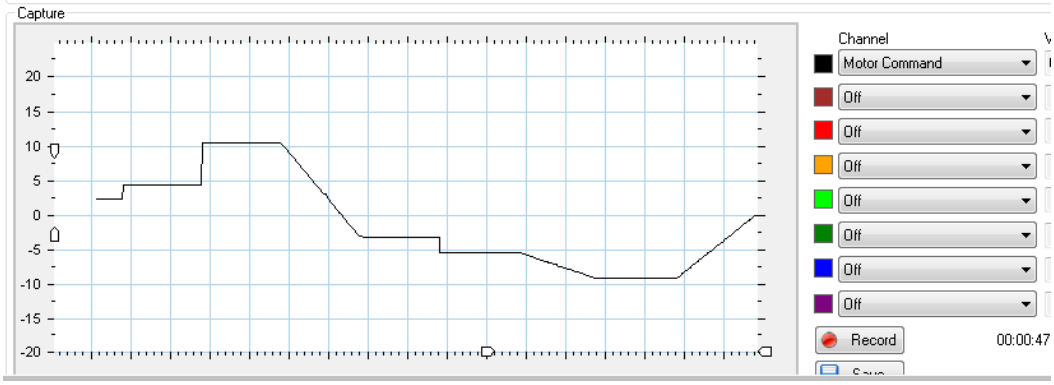

Figura 3.12: Comando motor en la controladora.

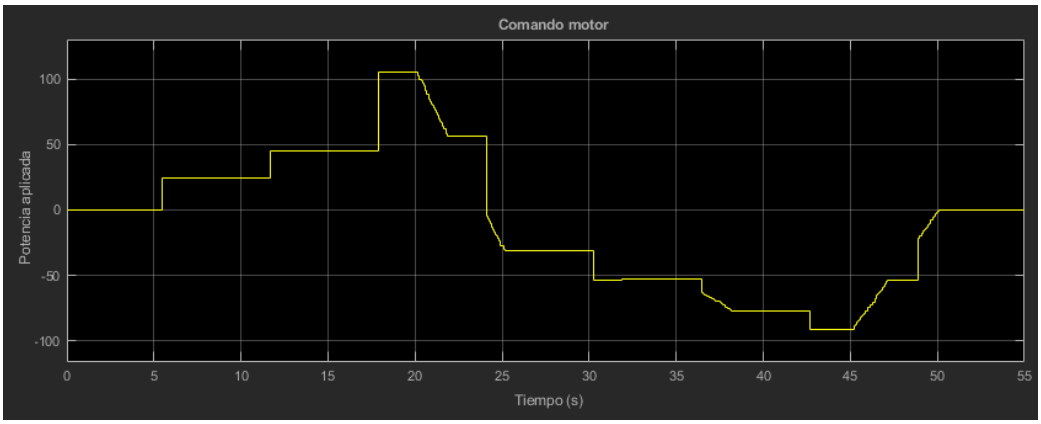

Figura 3.13: Comando motor leído en Simulink.

El problema reside en el script *ReceiveNode2*, el cual se mantiene a la escucha en el bus CAN, recibiendo todo mensaje enviado por el mismo, para posteriormente compararlo las direcciones esperadas y descartando los datos en caso de no coincidir con las mismas. Como en el Rambler hay varios dispositivos conectados al bus, el script es propenso a perder tiempo con datos de otros dispositivos, perdiéndose en el proceso los mensajes procedentes de las controladoras.

Para evitar este tipo de problemas, se utilizarán las máscaras y filtros del shield CAN, configurándose en ambos scripts de Matlab con el fin de leer únicamente los mensajes procedentes de las controladoras:

```
    /*Las máscaras y los filtros deben inicializarse en ambos códigos
(SendNode y ReceiveNode)*/

    CAN.init_Mask(0, 0, 0x7fc);
    CAN.init_Mask(1, 0, 0x7fc);

    /* Estos filtros permiten leer las variables VAR1 y VAR2*/
    CAN.init_Filt(0, 0, 0x180);
    CAN.init_Filt(1, 0, 0x180+NodoID[3]);

    /* Estos filtros permiten leer las variables VAR3 y VAR4*/
    //CAN.init_Filt(2, 0, 0x280);
    //CAN.init_Filt(3, 0, 0x280+NodoID[3]);
```

Se deben inicializar las dos máscaras, las cuales indican que bits de la dirección se deben comparar con los filtros. Los filtros indican que direcciones se reciben, pasando solo aquellas que coincidan los bits del filtro con los bits puestos a 1 de la máscara. Para mayor comprensión, se ilustra un breve ejemplo utilizando los valores del código implementado, suponiendo que el nodo 3 tiene la dirección 4:

$$Máscara: 0x7fc = 0111\ 1111\ 1100$$
$$Filtro\ 1: 0x180 = 0001\ 1000\ 0000$$
$$Filtro\ 2: 0x184 = 0001\ 1000\ 0100$$

Los dos últimos bits en 0 de la máscara permiten además leer mensajes de las direcciones 0x181, 0x182 y 0x183, dejando fuera la 0x184, razón por la que se añade en un nuevo filtro.

3.3.4. Sistema de seguridad.

Se debe tener en cuenta la posible pérdida de comunicación entre Arduino y el PC integrado del Rambler, ya sea por desconexiones del bus o el caso de que uno de los dispositivos deje de responder. En caso de que ocurriese, las ruedas seguirían moviéndose con el ultimo comando recibido por el bus, por lo que se perdería el control del vehículo mientras este sigue desplazándose. Esta situación debe evitarse a toda costa, por lo que se ha diseñado un sistema de seguridad encargado de supervisar la comunicación en todo momento.

Este sistema comienza con un contador discreto de 8 bits, el cual se inicializa en el diagrama de Simulink del propio PC, del modo que se indica en la Figura 3.14. Este contador comparte el tiempo de muestreo del bus CAN (0.01 segundos), por lo que la cuenta asciende de 0 a 255 en 2.55 segundos de manera cíclica, su comportamiento se puede observar en la Figura 3.15.

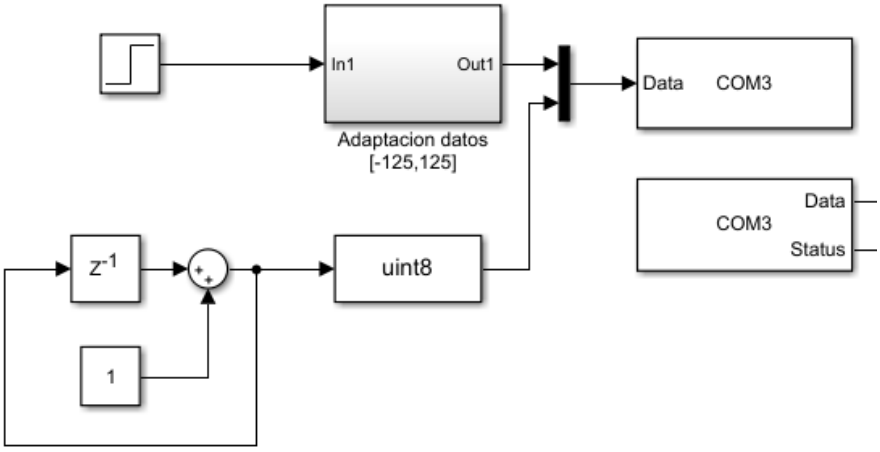

Figura 3.14: Contador discreto en Simulink.

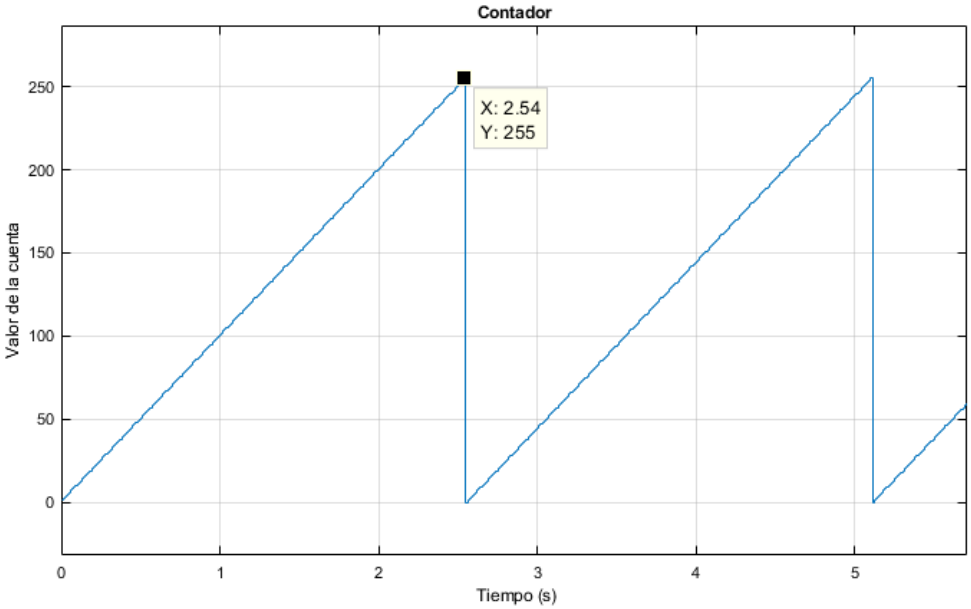

Figura 3.15: Salida del contador.

El valor del contador se envía a Arduino junto a la referencia por USB, para trasmitirla directamente por el bus CAN, esta modificación se detalla en la Figura 3.16. Nótese que no es necesario comprobar la cuenta en el propio Arduino, ya que este deja de responder, no llegara a actualizar el valor en el bus CAN.

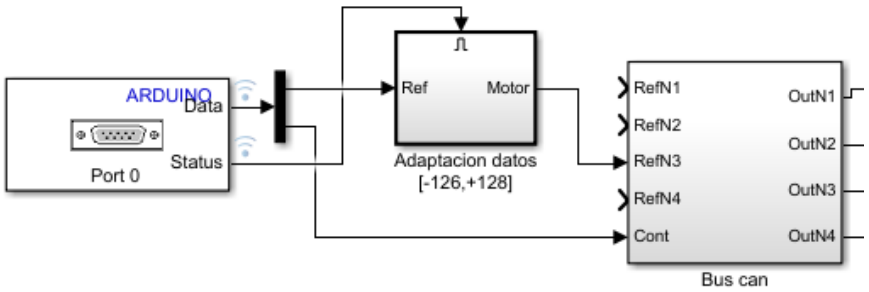

Figura 3.16: Recepción del contador en Arduino.

Para enviar la cuenta por el bus, se usa la VAR10, añadiendo la siguiente línea a la función *SendNode2:*

```
MiniCANtrans.VAR10=Cont[0]; //Guardamos el contador de seguridad.
```

Por último, se debe modificar el script de Roborun de cada controlador, para que reciba y compruebe el valor del contador en cada iteración. En la Figura 3.17 se dispone la modificación final del script, en la cual se añade un ciclo de retraso cada vez que la cuenta falla, en caso de tener varios ciclos de retraso seguidos, se anula la velocidad de las ruedas, deteniendo el vehículo. El script al completo se encuentra en el anexo Script controladores Roboteq.

```
'Lee el comando motor, en la VAR9.
    motor_command=getvalue(_VAR,9)

'------------------------------ Comprobación comunicación ------------------------------
'Guarda valor anterior del contador para poder compararlo con el nuevo valor.
    comunicacionant = comunicacion

'Leemos el contador proveniente de arduino, en la VAR10
    comunicacion = getvalue(_VAR, 10)

'Si el valor del contador actual es mayor que el anterior, la comunicación es correcta.
    if (comunicacion > comunicacionant) then
        ciclosretraso = 0

'Si por el contrario el valor es igual, la comunicación se ha detenido.
    else
        ciclosretraso++ 'Aumentamos el contador de seguridad

'Si el problema persiste, asumimos un fallo en la tramisión de datos.
        if (ciclosretraso > 4) then
            motor_command = 0    'anulamos el comando motor, deteniendo las ruedas.
        end if
    end if
'------------------------------  Fin de comprobación ------------------------------
```

Figura 3.17: Modificación del script de Roborun+.

Al ser necesarios varios ciclos de retraso, no afecta que el contador sea cíclico. Además, si la comunicación se reestablece, el sistema seguiría funcionando correctamente, reiniciándose los ciclos de retraso y retomando la velocidad de referencia.

4. BANCO DE PRUEBAS

4.1. Sensor Hall

Se realizará una primera identificación y posterior control usando los sensores de efecto Hall, con el objetivo de documentar los errores que se obtienen al bloquear la rueda.

4.1.1. Identificación

Para realizar la identificación se utilizará una señal escalonada simple de amplitud 100 como comando motor en el controlador. La respuesta obtenida, visible en la Figura 4.1, sigue el patrón de entrada.

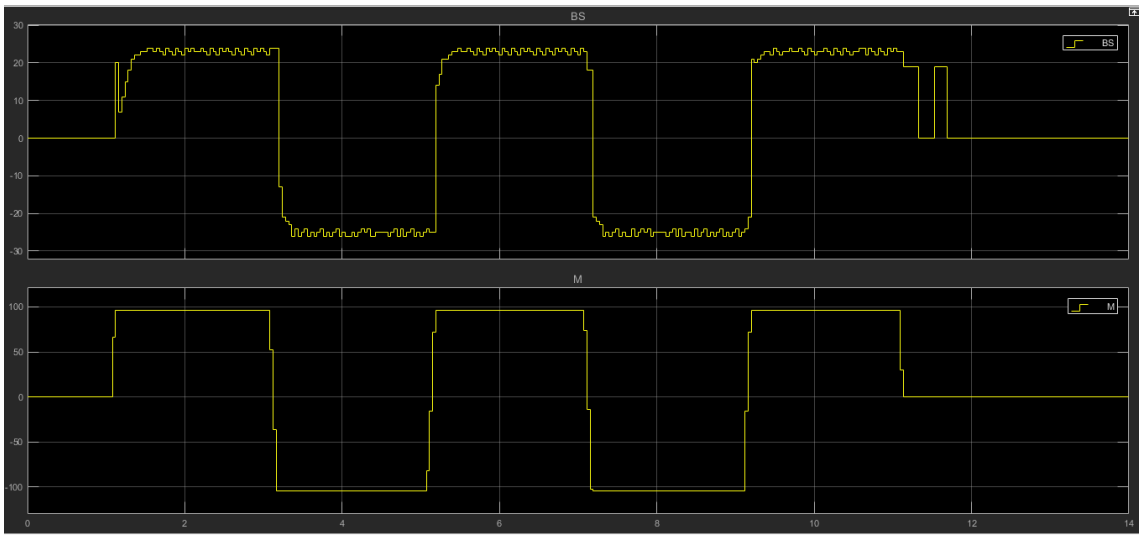

Figura 4.1: Respuesta del motor - Hall.

A continuación, se deben adaptar los datos de Simulink a Matlab. Para ello se utiliza el siguiente código extraído de [4], el cual procesa los datos como un Data object con tiempo de muestreo de 10 milisegundos:

```
%Accede a la información de las señales de Simulink
logsout

%Guarda esa información en variables del workspace
M=logsout.getElement(1).Values.Data;
BS=logsout.getElement(2).Values.Data;

%Transforma las variables para que sean compatibles con la función
iddata
BBS=BS(1,:);
MM=M(1,:);
BBS=BBS';
MM=MM';

%Crea un objeto con el comando del motor como entrada y la
%velocidad como salida, con un tiempo de muestre de 0.01 seg.
Cuad100 = iddata(BBS,MM,0.01);
```

Una vez ejecutado, se utiliza la herramienta de identificación de sistema, la cual proporciona la función de transferencia presente en la Ecuación 1, equivalente al modelo real con una aproximación del 86.53 %.

$$Tf(z^{-1}) = \frac{0.008862}{1 - 1.633\, z^{-1} + 0.6998\, z^{-2}} \qquad \text{Ecuación 1}$$

4.1.2. Control

Para poder sintonizar el controlador PID, se incluye el modelo en paralelo al sistema real utilizando el bloque *Environment controller*, este bloque dará la salida real en caso de exportar el programa a Arduino o ejecutarlo externamente, mientras que al realizar una simulación o trabajar en Simulink trabajará con la función de transferencia. El esquema de la planta es el indicado en la Figura 4.2.

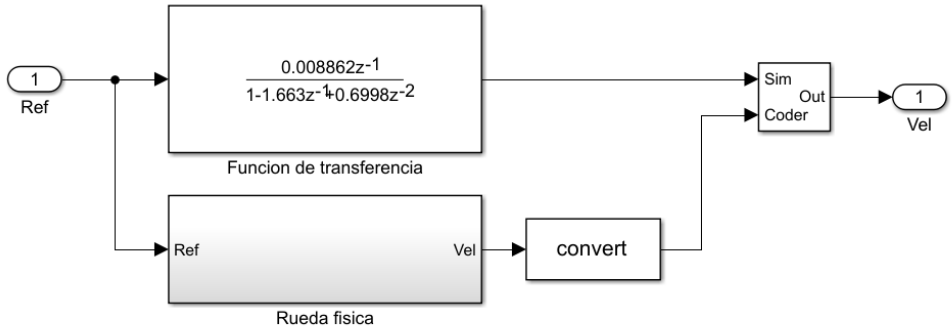

Figura 4.2: Detalle de la planta.

Además, se ha incluido en la salida del controlador el sistema Anti-retroceso [9] de la Figura 4.3, para evitar el envío de comandos al motor de distinto signo a la referencia, debidos a posibles fallos de lectura del sensor.

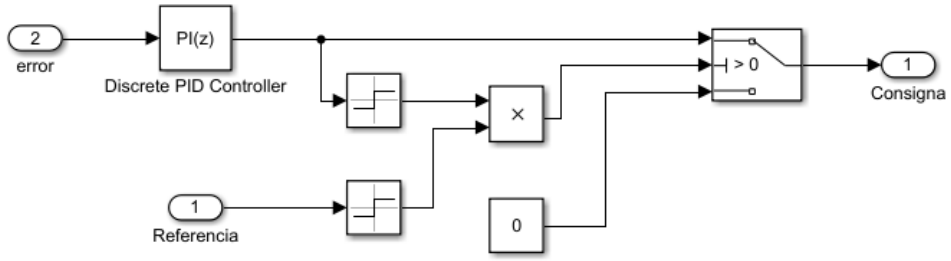

Figura 4.3: Detalle bloque "Anti-retroceso".

Por último, se procede a sintonizar el controlador PID mediante la herramienta de *autotune* del propio bloque. Tras probar distintas configuraciones, se obtiene la respuesta de la Figura 4.4, fruto de un control PI con los siguientes parámetros:

- Ganancia proporcional: 1.1
- Ganancia integral: 20.5

Al no existir ninguna carga sobre la rueda al estar en el aire, el control recae en la acción integral.

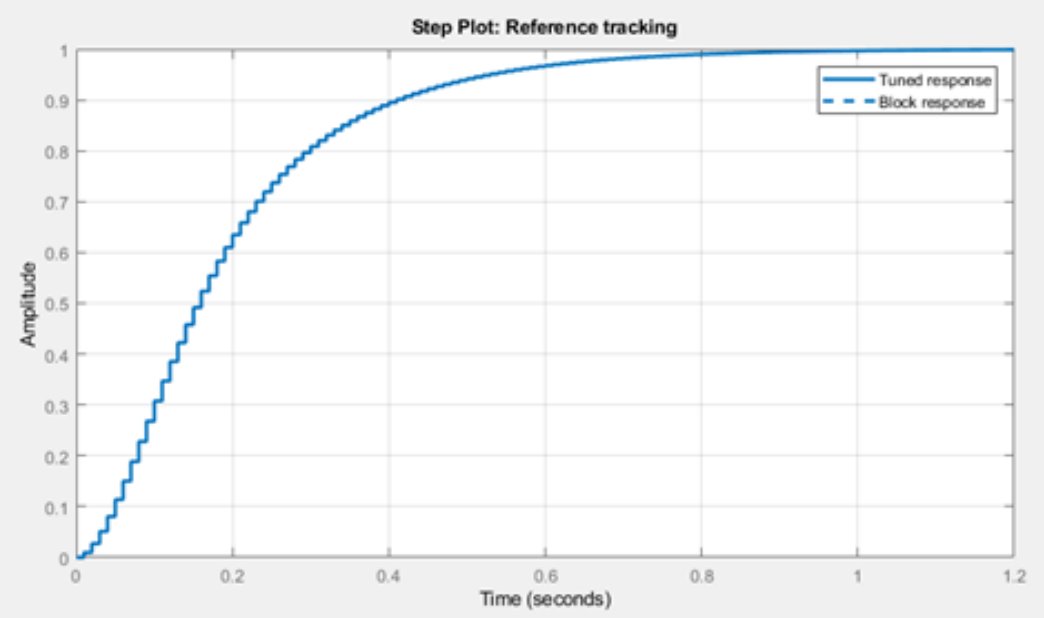

Figura 4.4: Respuesta sintonizada - Hall.

4.1.3. Resultados

Una vez establecido el sistema de control realimentado, mostrado en la Figura 4.5, se procede a realizar pruebas sobre el mismo.

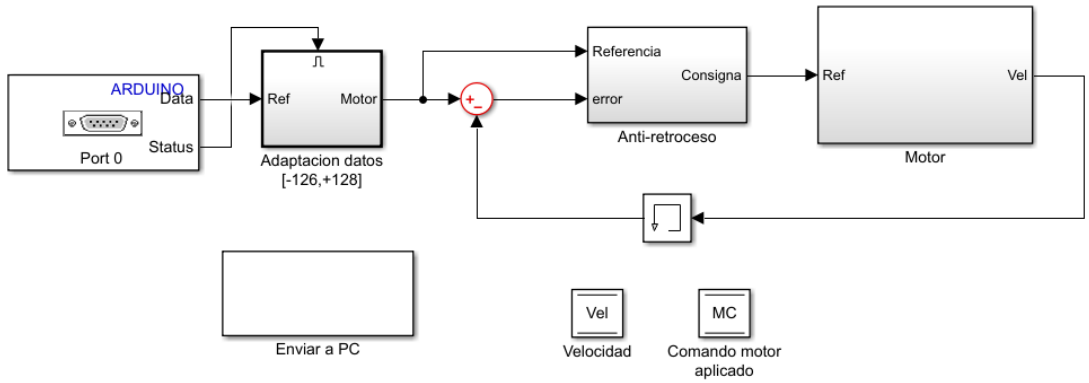

Figura 4.5: Esquema final de control en Arduino - Hall.

El sistema es robusto para velocidades medias y altas, sin embargo, el objetivo es conseguir controlar a bajas velocidades, por lo que trabajamos con una referencia de 10 rpm, consiguiendo la respuesta de la Figura 4.6. A esta velocidad, el sistema responde correctamente ante pequeñas perturbaciones, pero resulta muy sencillo bloquear la rueda al aumentar el rozamiento, provocando errores en la lectura de los sensores Hall de ±1000 rpm para una velocidad real nula.

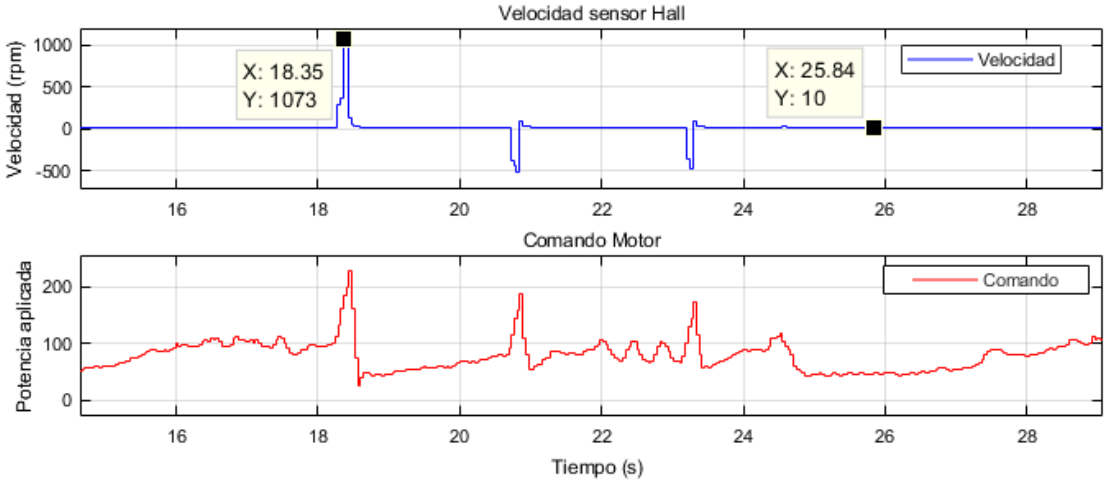

Figura 4.6: Respuesta a bloqueo con sensor Hall.

4.2. Encoder

Una vez completado el control con el sensor Hall, se instala en el banco un sensor tipo encoder, con el cual se repite todo el proceso de manera más exhaustiva, ya que se pretende utilizar este tipo de codificadores en el Rambler.

4.2.1. Instalación

Una vez adquiridos los encoders, deben ser instalados tanto en el banco de pruebas como en el Rambler. Tras consultar los *datasheets* del encoder [10] y del controlador [11], se llega al conexionado de la Tabla 3 y al esquema de conexionado de la Figura 4.7[1].

Cable encoder	Función	Pin Controlador
Marrón	Vcc	25
Blanco	GND	13
Verde	Encoder A	12
Gris	Encoder B	24

Tabla 3: Conexionado del encoder a HBL1660.

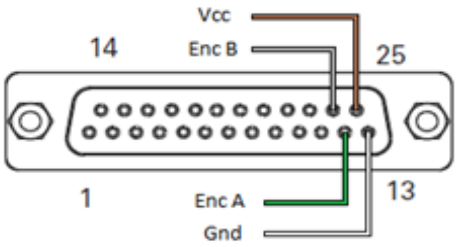

Figura 4.7: Conexión encoder a HBL1660

A la hora de realizar las soldaduras, se debe tener especial cuidado con el cable de alimentación, el cual se aísla con termo retráctil para evitar cortos.

[1] Para este esquema, Roborun considera positivo el sentido de giro antihorario. En caso de querer invertirlo, bastaría con intercambiar los cables *Enc A* y *Enc B* entre sí.

Figura 4.8: Soldado del Encoder.

Para trabajar con el encoder en Matlab, se modifica el código de Roborun para que transmita la velocidad leída por el bus CAN. Para ello se cambia la referencia del sensor Hall "_BS" por la del encoder "_S" [12]. Enviándola posteriormente por la variable 2, estas modificaciones se aprecian en la Figura 4.9.

```
'Lee las velocidades del motor en rpm
    'motor_speed=getvalue(_BS,channel)   'Sensor Hall
    enc_speed=getvalue(_S,1)             'Encoder

'Coloca en el motor en VAR1 la velocidad del sensor Hall y en VAR2 la del encoder
    'setcommand(_VAR,1,motor_speed)      'Sensor Hall
    setcommand(_VAR,2,enc_speed)         'Encoder
```

Figura 4.9: Modificación script.

Por último, se debe configurar el encoder en Roborun, en este caso basta con indicar los pulsos por vuelta, tal y como se indica en la Figura 4.10.

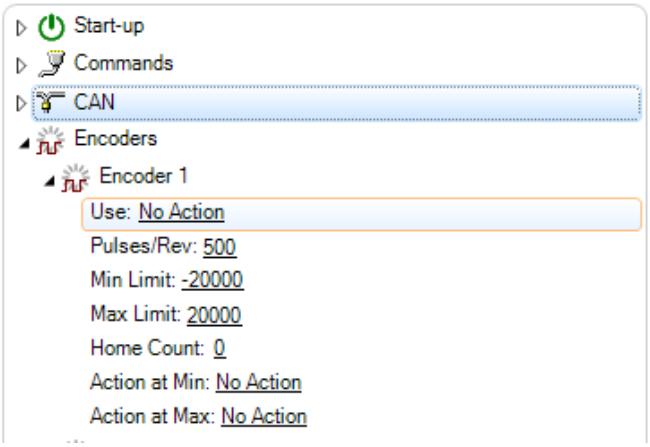

Figura 4.10: Configuración del encoder – Roborun+.

En este caso la velocidad se transmite al encoder mediante una correa y un sistema de engranajes. La relación de engranajes es de 60 dientes para el motor y 14 para el encoder, aumentando así la cantidad de pulsos por vuelta del encoder, indicados en la Ecuación 2.

$$500 \cdot \frac{60}{14} = 2142.85\ pulsos/vuelta$$ Ecuación 2

Debe tenerse en cuenta que el software Roborun trabaja únicamente con números enteros, lo que llevaría a utilizar una aproximación a 2143 pulsos por vuelta, al no ser este valor exacto. Por esta razón se aplica el factor de conversión en Simulink, mediante la ganancia presente en la Figura 4.11, ganando resolución al poder apreciar decimales.

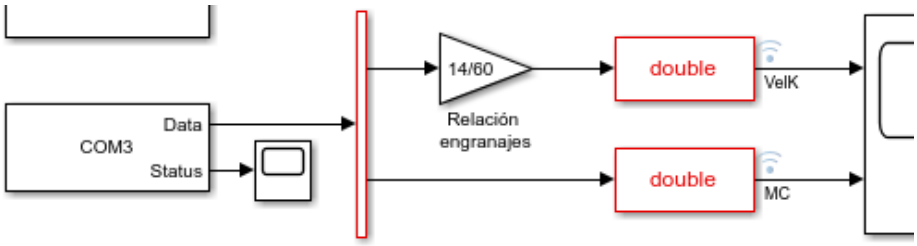

Figura 4.11: Implementación relación de engranajes – Identificación PC.

4.2.1.1. RESOLUCIÓN.

Al visualizar la respuesta del encoder en Simulink, se obtiene una resolución de 0.7 rpm, apreciable en la Figura 4.12. Este valor es demasiado elevado para un control en baja velocidad, por lo que se intenta mitigar disminuyendo los Pulsos por vuelta en la configuración de Roborun+, pero no se consigue disminuir los saltos entre valores.

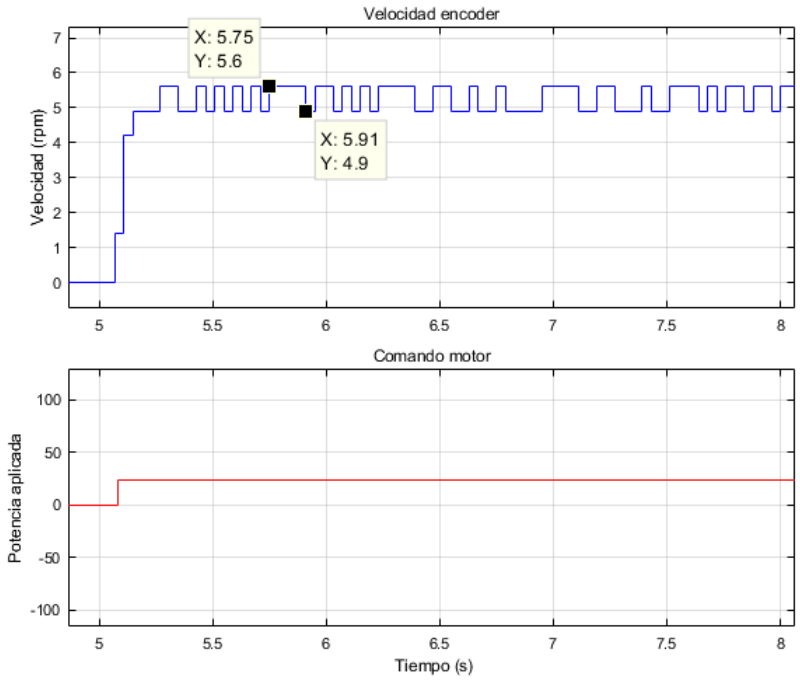

Figura 4.12: Detalle respuesta encoder.

Tras investigar el manual de usuario de Roboteq [12], se descubre que el controlador procesa los datos del encoder con un tiempo de muestreo fijo de 10 ms, lo que resulta en una limitación en la resolución del encoder. En este caso, la resolución mínima para un encoder de cuadratura sigue la expresión de la Ecuación 3 [13]. Teniendo en cuenta la relación de engranajes y los 500 pulsos por vuelta del encoder, se obtienen los 0.7 rpm presenciados en la respuesta.

$$\frac{60}{4 \cdot PPR \cdot 0.01} = \frac{60}{4 \cdot 500 \cdot \frac{60}{14} \cdot 0.01} = 0.7\ RPM \qquad \text{Ecuación 3}$$

Este valor puede afectar a la hora de realizar el control en baja velocidad, pero la única solución para reducirlo consiste en adquirir nuevos encoders de mayor resolución, opción inviable actualmente.

4.2.2. Identificación

En esta ocasión se utilizará la señal de la Figura 4.13 como referencia, mediante la cual se obtendré un modelado más fiable, ya que incluye rampas y escalones de distinta magnitud. La respuesta del motor, presente en la Figura 4.14, es prácticamente idéntica a la entrada.

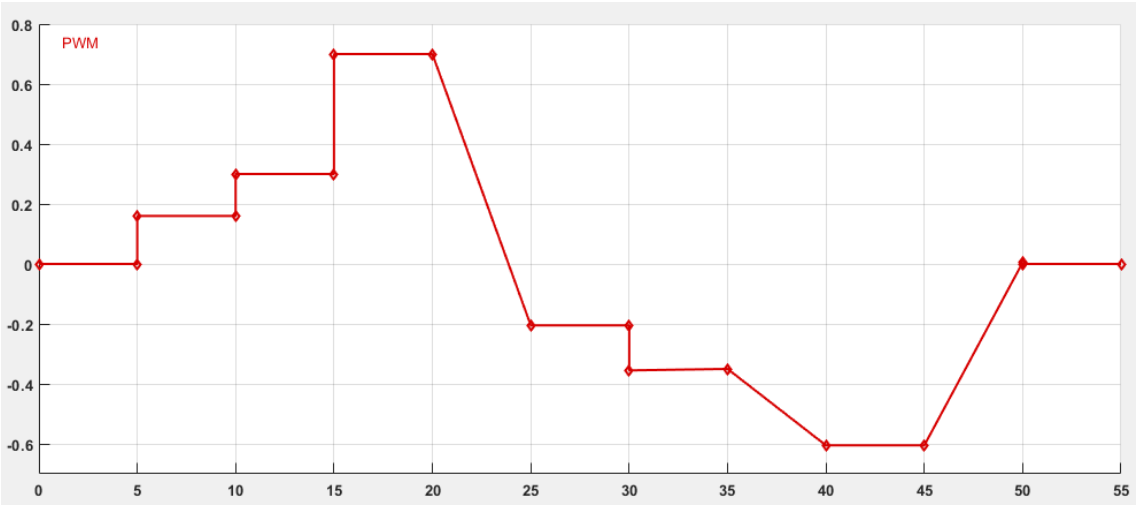

Figura 4.13: Referencia utilizada.

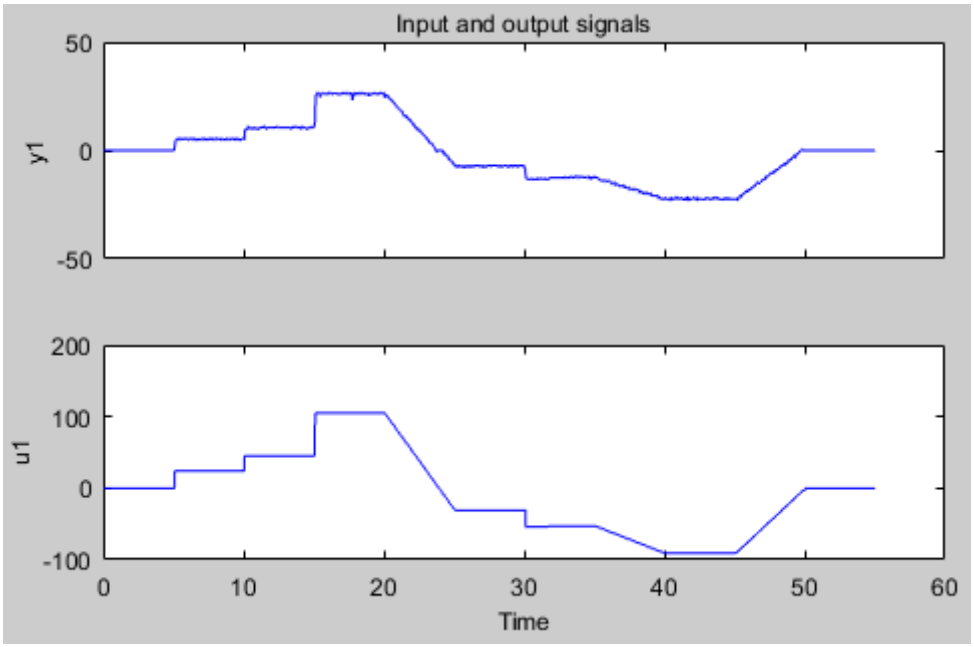

Figura 4.14: Respuesta obtenida.

Además, se trabaja con funciones de transferencia de distinto tipo, con el objetivo de comparar sus respuestas entre sí. Dichas funciones siguen la siguiente terminología:

- tf1: 2 polos y 1 cero.
- tf2: 2 polos.
- tf3: 1 polo.

Para la comparación, Matlab simula la misma referencia utilizada en cada modelo, colocando posteriormente la respuesta encima de la real, obteniendo la gráfica de la Figura 4.15. A priori los 3 modelos parecen ajustarse a la salida del encoder. Sin embargo, al observar en detalle en la Figura 4.16, se aprecia una mejor respuesta de la función de transferencia de un único polo.

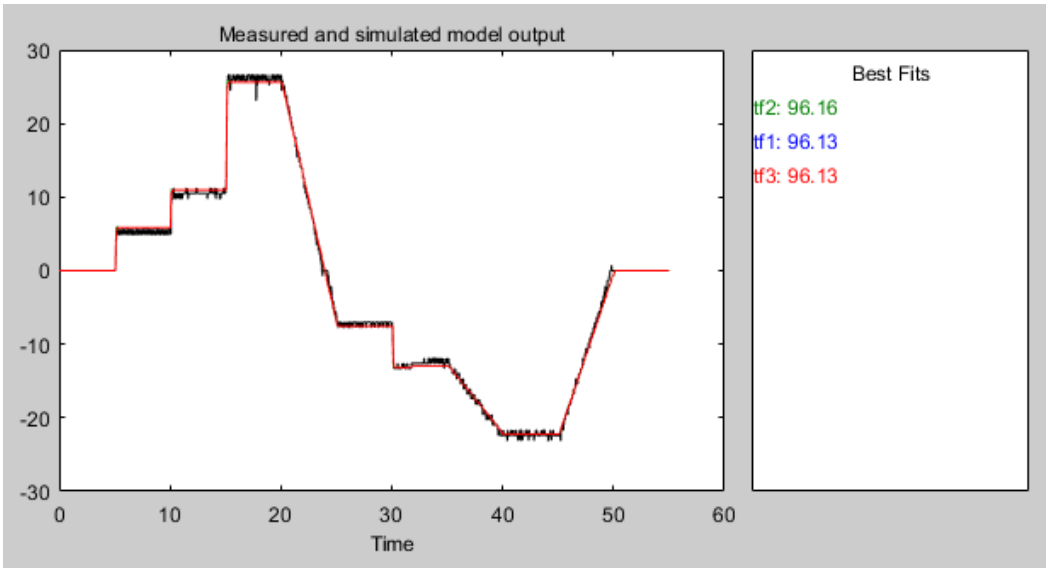

Figura 4.15: Respuesta de los modelos.

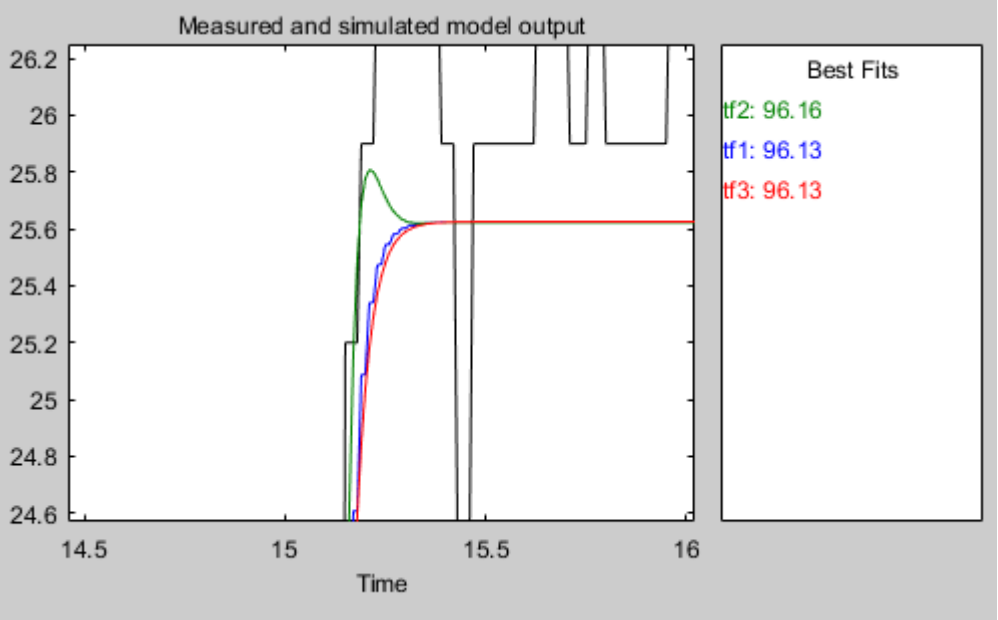

Figura 4.16: Detalle de la respuesta.

En la Ecuación 4 se presenta el modelado escogido, con una aproximación al modelo real del 96.13 %, mejorando considerablemente la fiabilidad conseguida con el sensor Hall.

$$Tf(z^{-1}) = \frac{0.05979}{1 - 0.755\, z^{-1}} \qquad \text{Ecuación 4}$$

4.2.3. Control

El esquema de control sufre algunas modificaciones, debiéndose incluir la relación de engranajes en la salida del bus CAN, como se indica en la Figura 4.17. Además, se prescinde del bloque de Anti-retroceso, ya que el encoder no es propenso a errores de lectura. Se presenta el esquema final en la Figura 4.18.

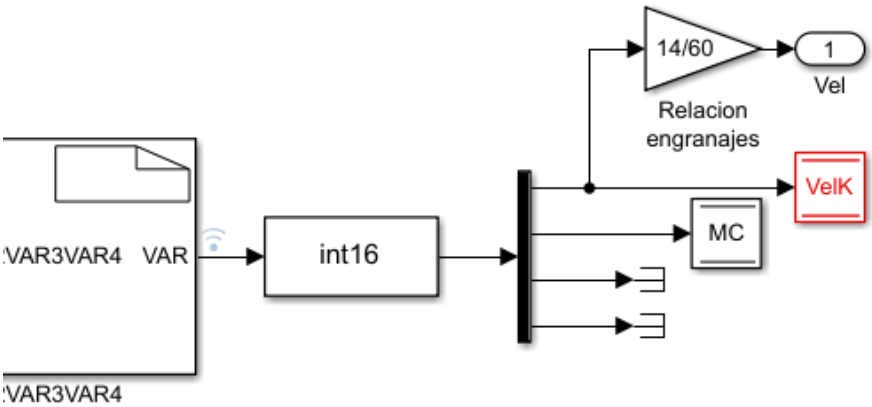

Figura 4.17: Implementación de la relación de engranajes en el control.

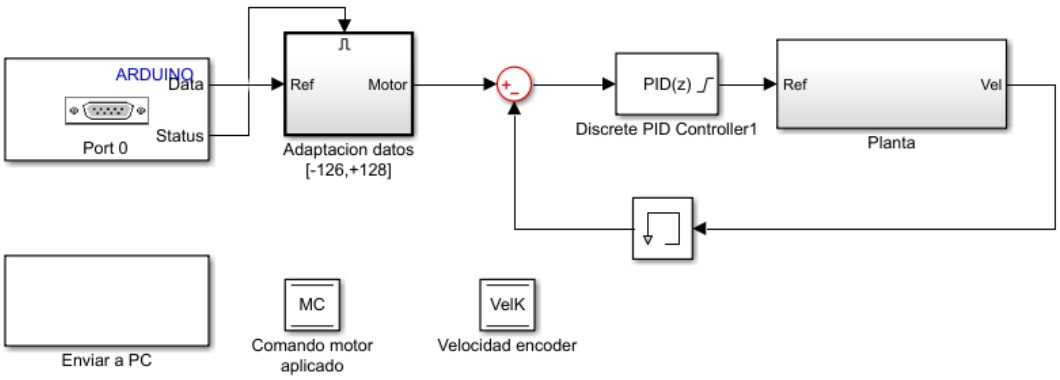

Figura 4.18: Esquema final de control en Arduino - Encoder.

Se mantiene la estrategia de sintonizar un controlador PID mediante la herramienta *Autotune*, en este caso se busca un tiempo de establecimiento muy corto para superar rápidamente cualquier perturbación. En la Figura 4.19 se encuentra la respuesta sintonizada, para unas ganancias muy similares a las obtenidas con el sensor Hall:

- Ganancia proporcional: 2.1
- Ganancia integral: 20.5
- Ganancia derivativa: -0.01

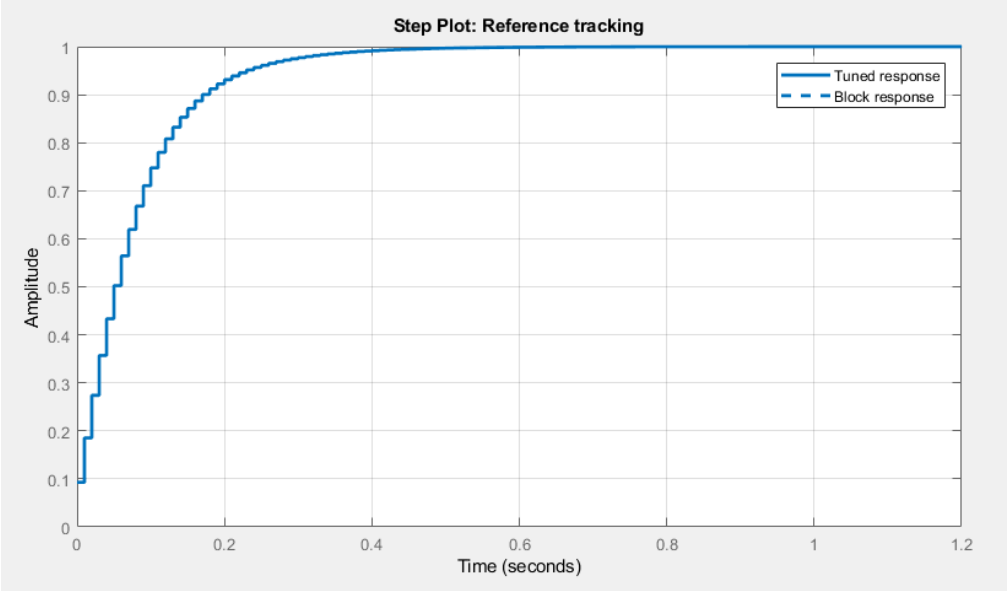

Figura 4.19: Respuesta sintonizada - Encoder.

Además, se aplica una saturación al controlador para evitar valores fuera del rango establecido en el controlador. Para reducir las sobre oscilaciones, se activa la herramienta *Anti-windup* en su modalidad *back-calculation*, de este modo el integrador se descargará al superarse los umbrales de saturación, siguiendo el ciclo realimentado de la Figura 4.20. Esta configuración avanzada se muestra aplicada en la Figura 4.21[2].

[2] Se testeo y descarto el *Tracking mode*. Esta opción descarga el integrador utilizando la respuesta del sensor, eliminando la Sobreoscilación al superar un bloqueo, a costa de un control excesivamente lento.

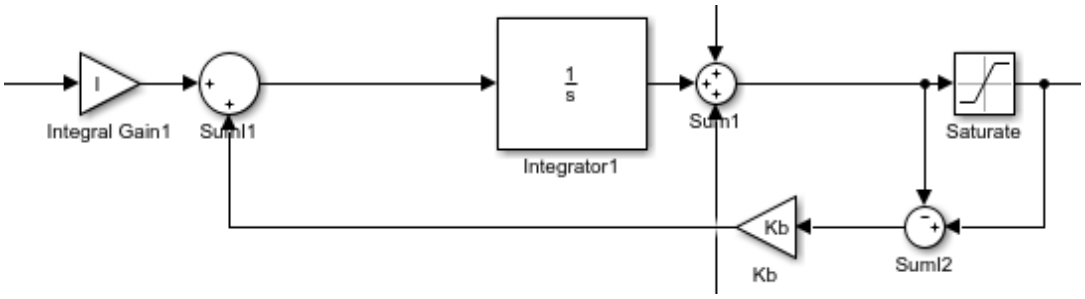

Figura 4.20: Funcionamiento back-calculation.

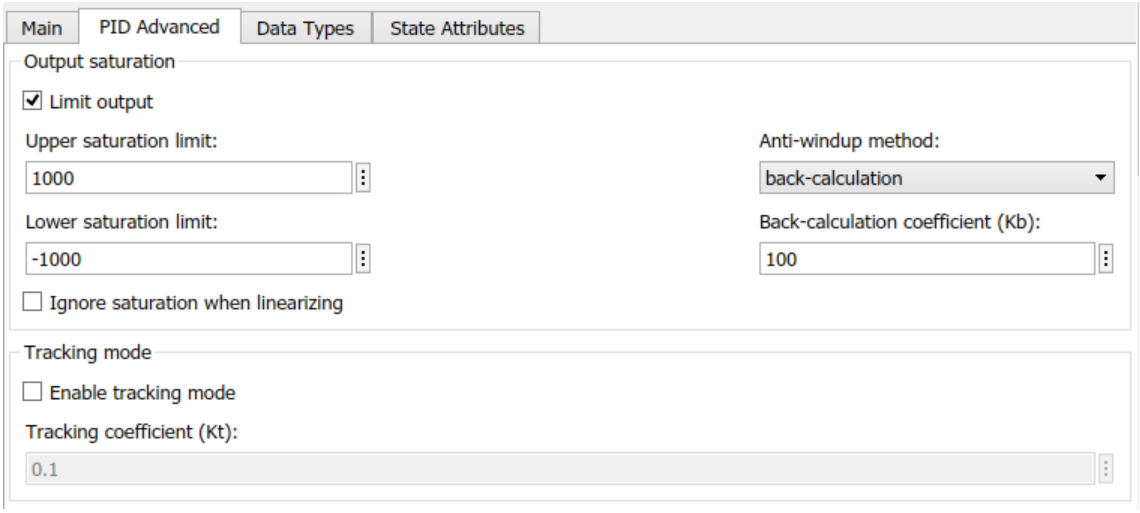

Figura 4.21: Configuración avanzada PID.

4.2.4. Resultado

Se pone a prueba el control aplicando perturbaciones con una referencia de 10 rpm. Debe tenerse en cuenta que a tan baja velocidad resulta muy fácil bloquear la rueda, pero con el control desarrollado la respuesta al superar el bloqueo es buena, avanzando apenas 40 grados y estabilizándose.

En la Figura 4.22 se observa el test realizado, notándose a simple vista las zonas en las que se ha producido un bloqueo, caracterizadas por una fuerte pendiente en el comando motor. Al liberarse la rueda, se produce un pico de velocidad para a continuación estabilizarse en el valor de referencia.

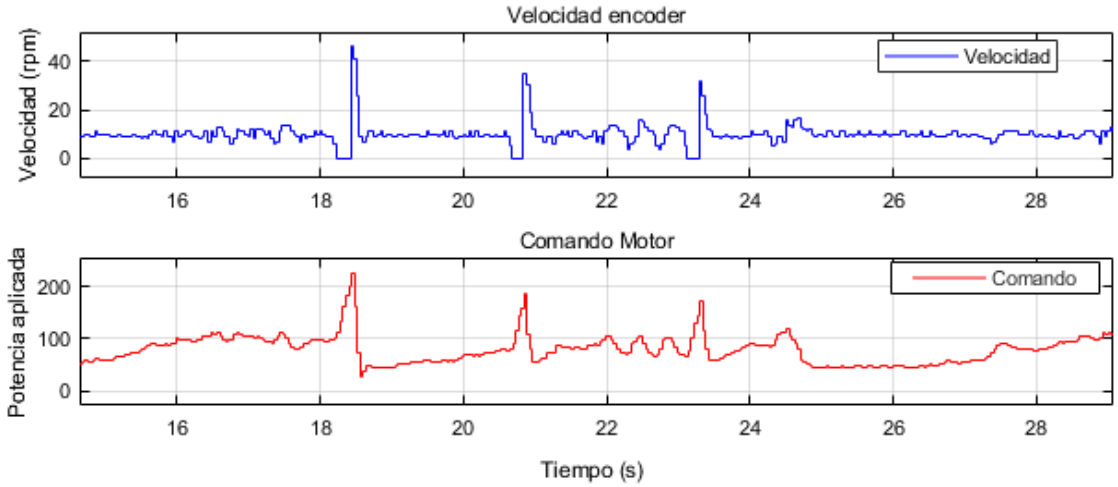

Figura 4.22: Respuesta a 4 rpm - Encoder.

4.3. Comparación sensores.

Ante los resultados obtenidos, queda claro que los encoders son el tipo de sensor más adecuado para trabajar a bajas velocidades. Los sensores Hall pueden resultar interesantes para velocidades mayores, pero los errores que proporciona al ocurrir un bloqueo resultan inaceptables a la hora de desarrollar un control. Por ello, al trabajar en el robot móvil Rambler se utilizarán únicamente los encoders recientemente instalados.

5. RAMBLER

5.1. Test inicial.

Antes de comenzar la identificación, se realizará un pequeño test sobre todas las ruedas, siendo necesario realizar algunas modificaciones a los esquemas en Simulink. Para facilitar la toma de datos, se identifica cada rueda en el esquema del PC y se aplican los factores de conversión a la velocidad, primero el de engranajes y posteriormente a metros por segundo, estas modificaciones se encuentran aplicadas en la Figura 5.1.

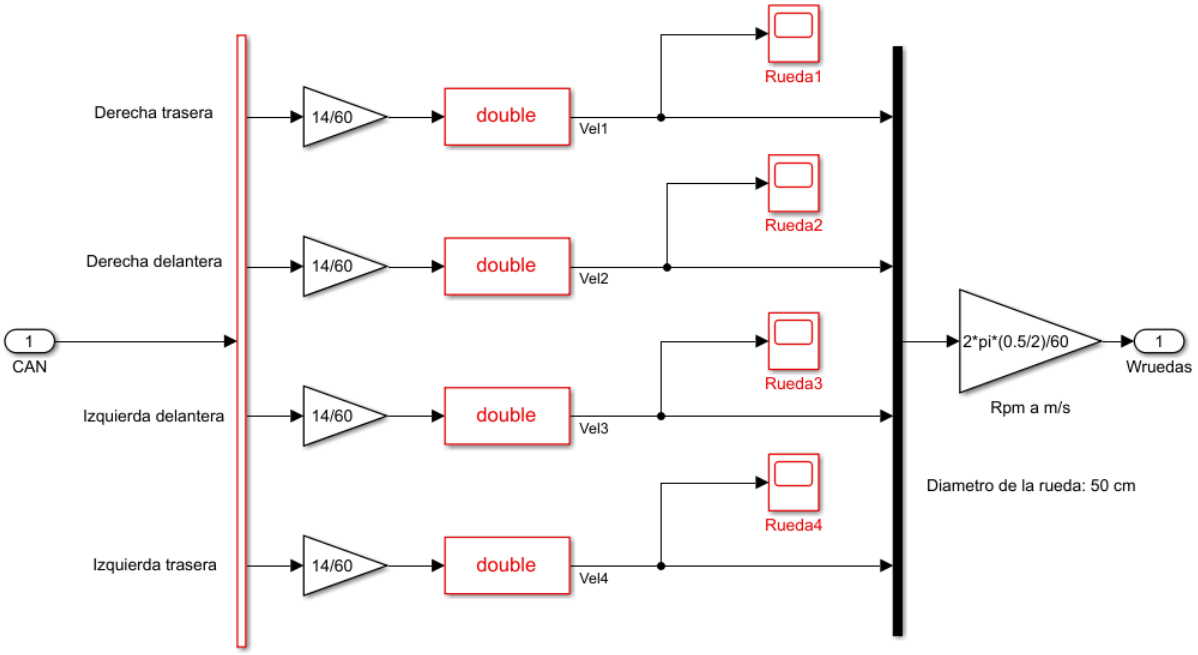

Figura 5.1: Tratamiento de los datos.

Además, se incluye el modelo odométrico del Rambler [5], [14], tomándose el eje X como el avance lineal del vehículo. Este modelo servirá para estudiar su comportamiento, ya que al estar levantado no será visible a simple vista.

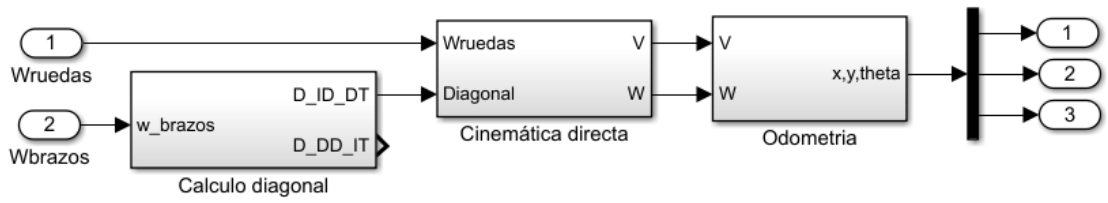

Figura 5.2: Bloque de odometría.

El test consistirá en aplicar una referencia constante a todas las ruedas, deteniéndose la simulación al completar 2 metros en el eje X mediante el switch de la Figura 5.3. Para el cálculo del giro se supondrá que los brazos de la suspensión tienen un ángulo de 0 grados.

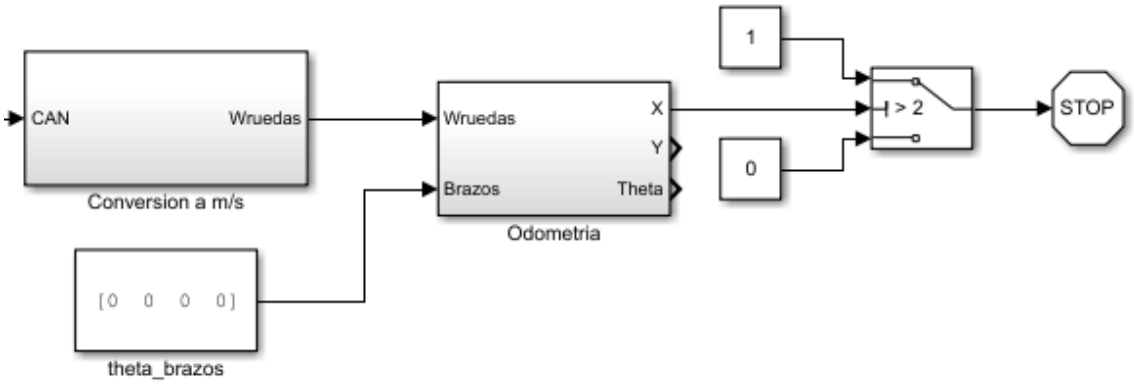

Figura 5.3: Condición para finalizar el test.

Al estar el vehículo levantado, se utiliza la odometría[3] para detallar el recorrido en el transcurso de la identificación. En las Figura 5.4, Figura 5.5 y Figura 5.6 se comprueba como la velocidad lineal sufre fuertes variaciones y el vehículo llega a girar hasta $0.43\ radianes \cong 22º$, generando un desvió de hasta 40 cm en el eje Y en los 2 metros recorridos.

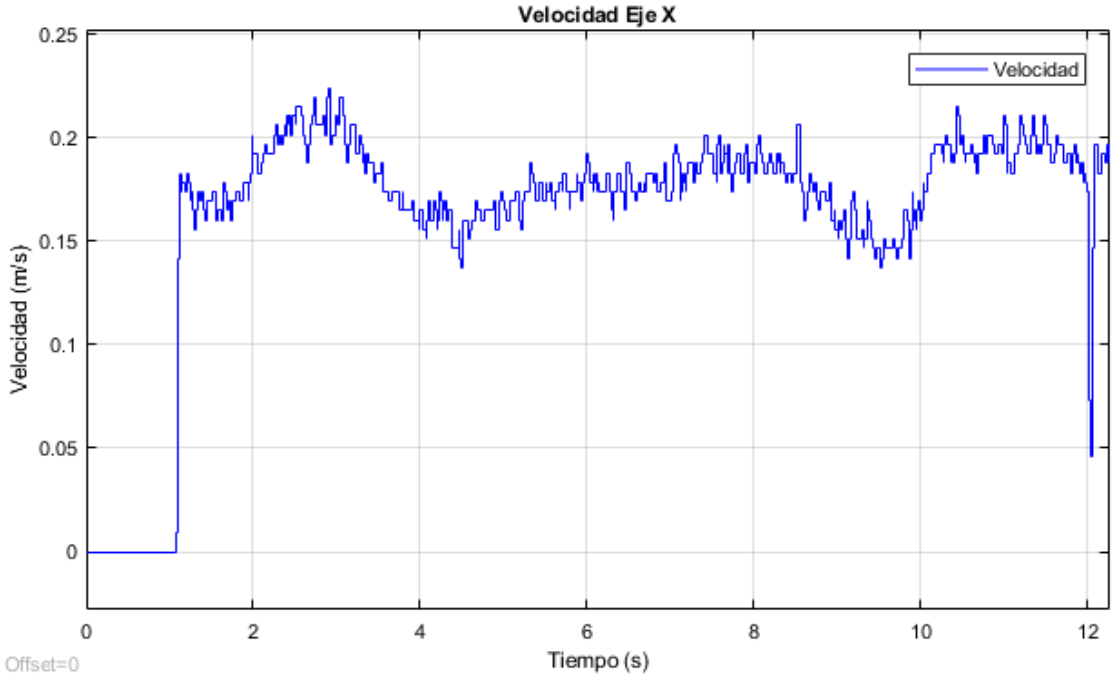

Figura 5.4: Velocidad Eje X - Test.

[3] Debe tenerse en cuenta que la odometría utilizada está basada en el modelo matemático y no es real, ya que no se tiene en cuenta la interacción entre ruedas que se daría en caso de estar en contacto con el suelo.

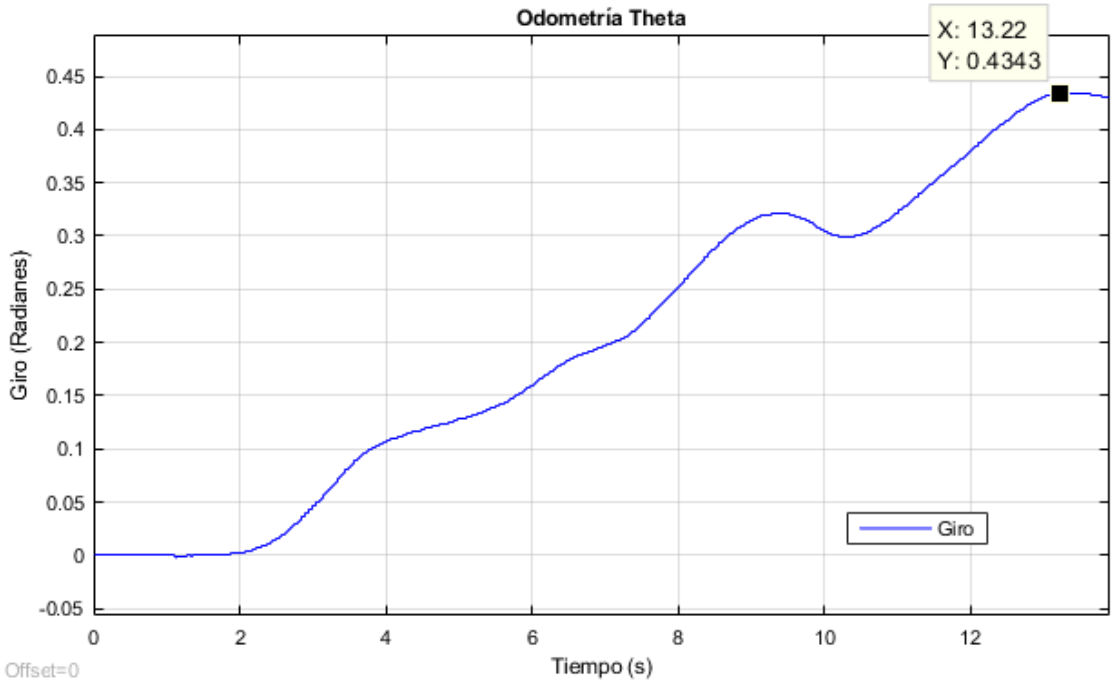

Figura 5.5: Angulo de giro - Test.

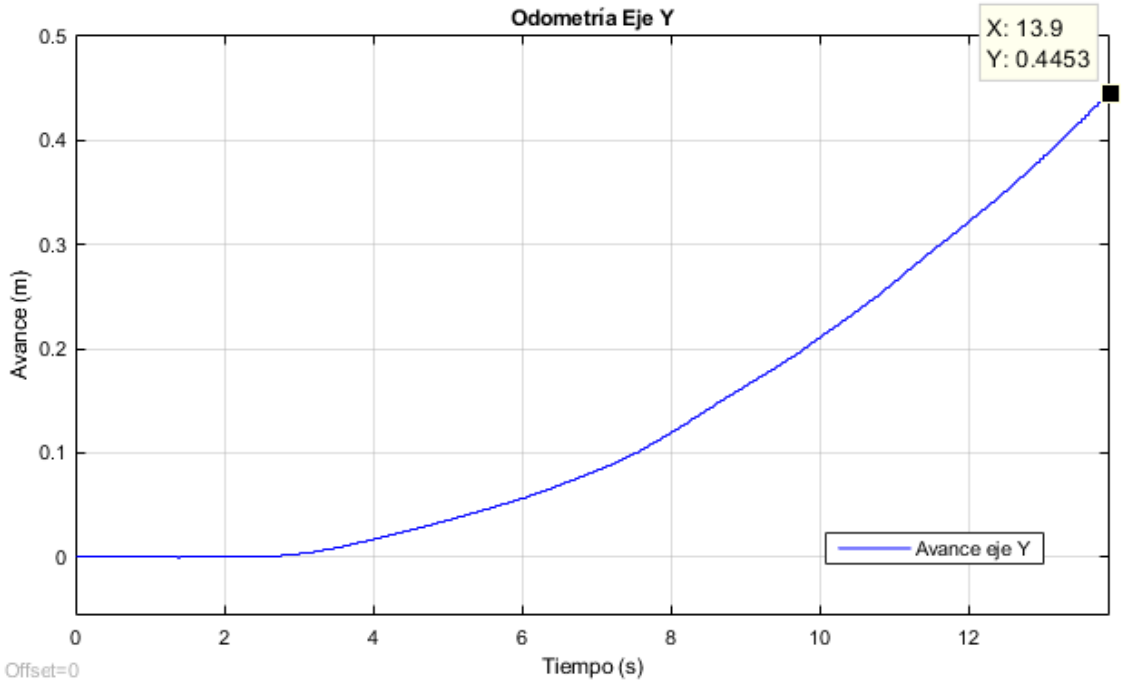

Figura 5.6: Avance eje Y - Test.

5.2. Control de 1 rueda sin contacto con el suelo.

5.2.1. Interferencias a baja velocidad.

Se realiza una primera identificación utilizando la misma señal utilizada con el encoder, obteniendo la respuesta de la Figura 5.7. No obstante, al continuar el proceso la respuesta de los modelos difiere del comportamiento real, tal y como se aprecia en la Figura 5.8.

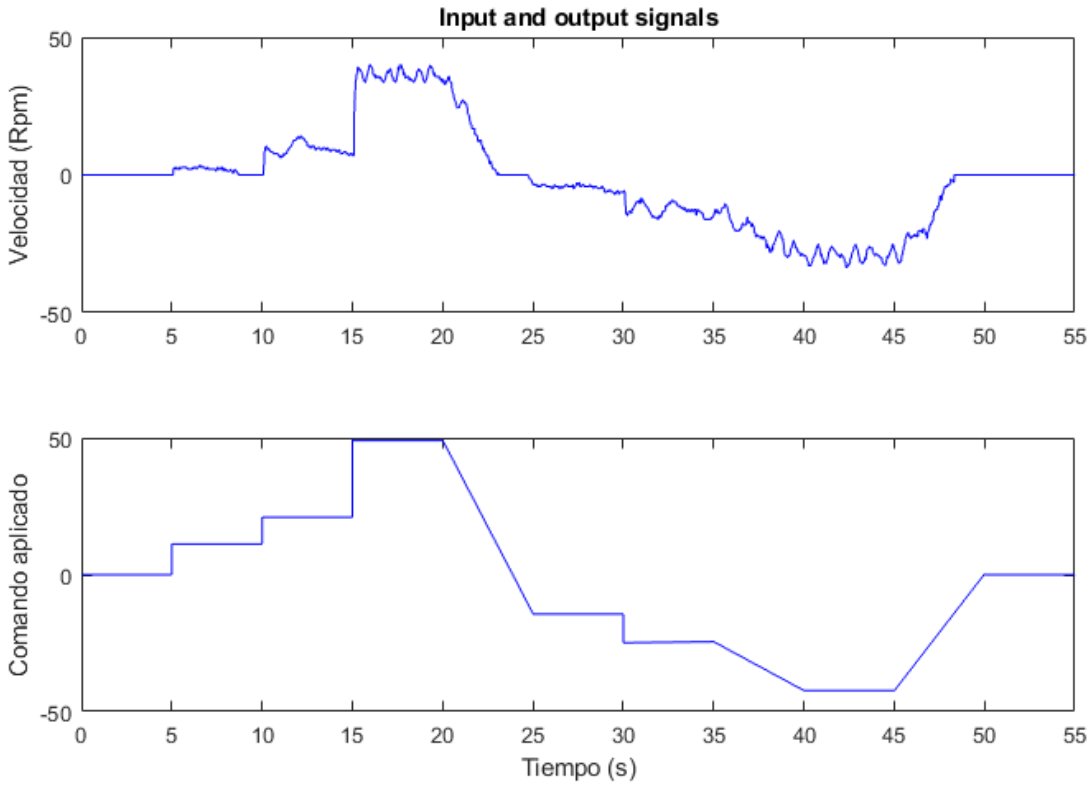

Figura 5.7: Respuesta Rambler - Levantado.

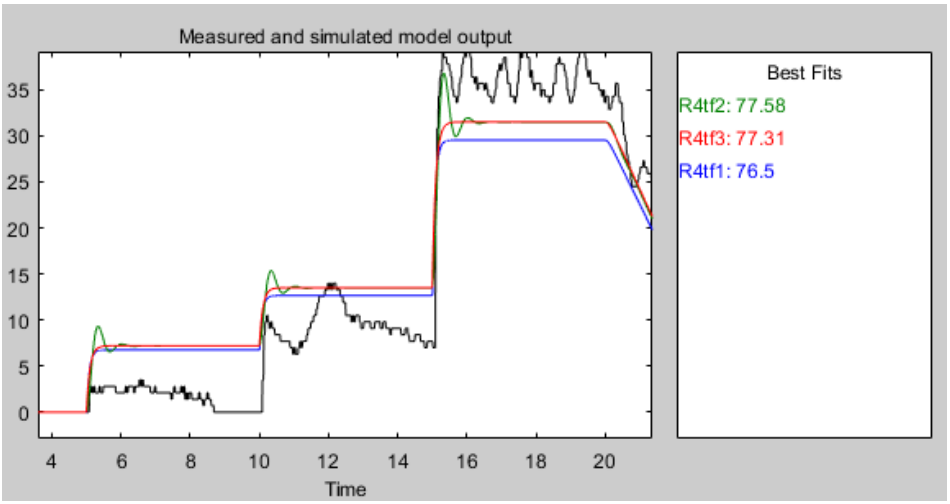

Figura 5.8: Respuesta funciones de transferencia - Levantado.

Este error en el modelado se debe a que la respuesta a baja velocidad no está siendo fiable. En la Figura 5.9 se comprueba que el comportamiento del motor no es real para una velocidad inferior a 30 rpm.

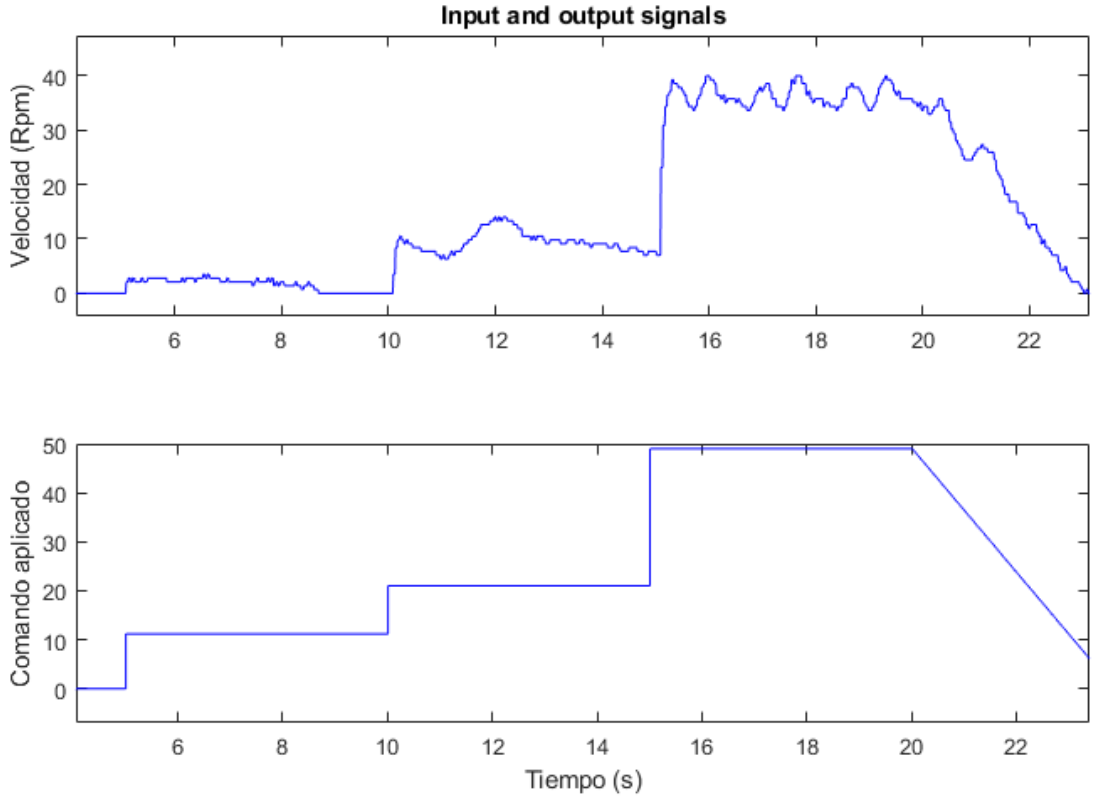

Figura 5.9: Detalle identificación - Levantado.

Para buscar la causa, en las Figura 5.10 y Figura 5.11 se contrasta la respuesta del encoder en el banco de pruebas y en una rueda del vehículo:

- En el banco de pruebas sin la rueda instalada, la velocidad es prácticamente estable, con variaciones de apenas 3 rpm.
- En el caso del Rambler se observa un patrón senoidal en cada vuelta de rueda, presentándose una diferencia de 33 rpm entre el valor mínimo y máximo obtenido.

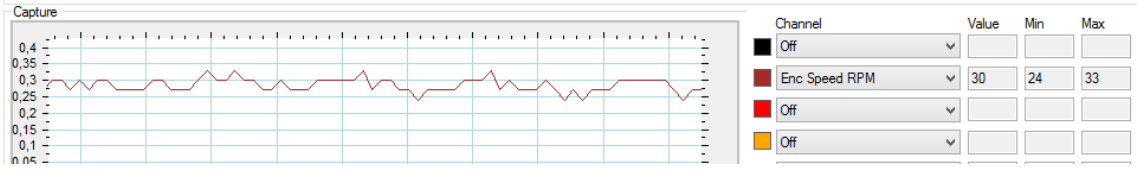

Figura 5.10: Respuesta encoder – Banco.

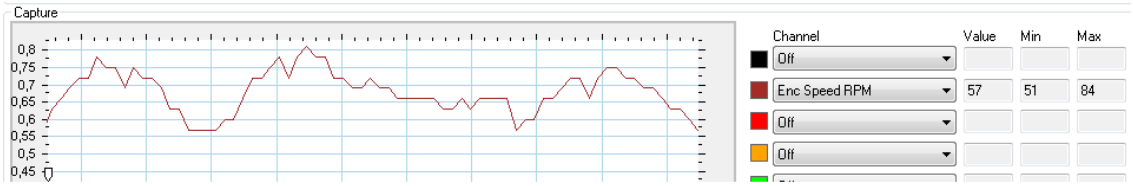

Figura 5.11: Respuesta encoder - Rambler.

A la vista de estos datos, se concluye que la resistencia al giro no es uniforme a lo largo de una revolución. Esta diferencia viene causada por el rozamiento con el disco de

freno y la utilización de ruedas no equilibradas, interferencias no presentes en el banco de pruebas.

Estas perturbaciones no son de interés, ya que al colocar el Rambler en el suelo pasaran a ser despreciables. Por lo tanto, se realizará una segunda identificación a mayor velocidad, a fin de conseguir una respuesta más estable.

5.2.2. Identificación

Se modifica el comando motor a una señal escalón de amplitud 40, obteniéndose una respuesta estable a 30 rpm, presente en la Figura 5.12. En esta ocasión la respuesta del modelo se ciñe a la del sistema real, comparándose ambas en la Figura 5.13.

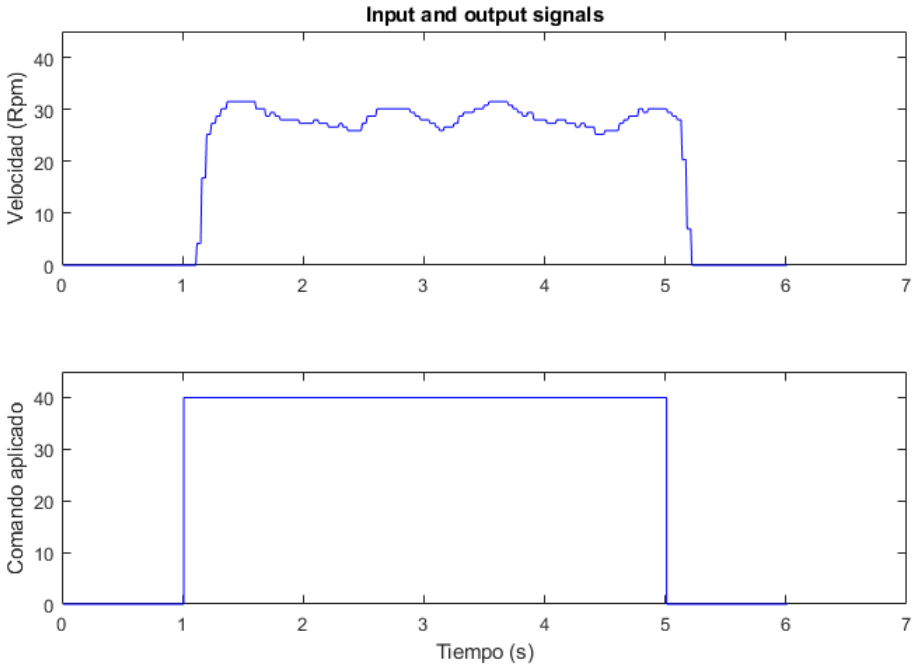

Figura 5.12: Respuesta identificación - Levantado.

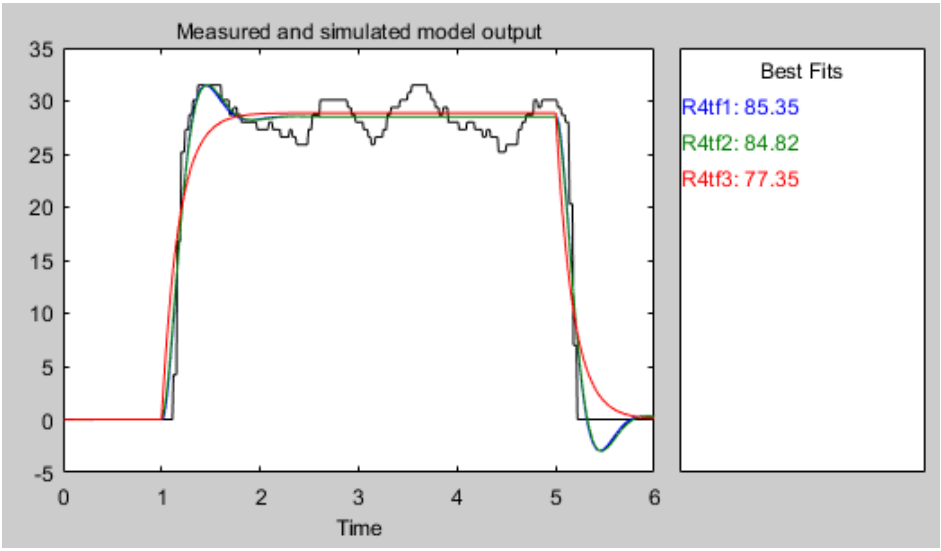

Figura 5.13: Respuesta del modelo – Levantado.

Para realizar el control se utilizará la señal R4tf2, de 2 polos y ningún cero, presente en la Ecuación 5, con un 84.52 % de aproximación al sistema real.

$$Tf(z^{-1}) = \frac{0.004641}{1 - 1.902\,z^{-1} + 0.908\,z^{-2}} \qquad \text{Ecuación 5}$$

5.2.3. Control

Utilizando la función de transferencia obtenida, se sintoniza mediante *Autotune* la respuesta de la Figura 5.14 para un controlador PI con las siguientes ganancias:

- Ganancia proporcional: 0.8
- Ganancia integral: 16.9

La estrategia de control es muy similar a la utilizada en el banco de pruebas, con una acción integral mucho mayor a la proporcional. Además, es similar al control presente en los controladores de Roboteq, ya que la marca aconseja utilizar principalmente la acción integral.

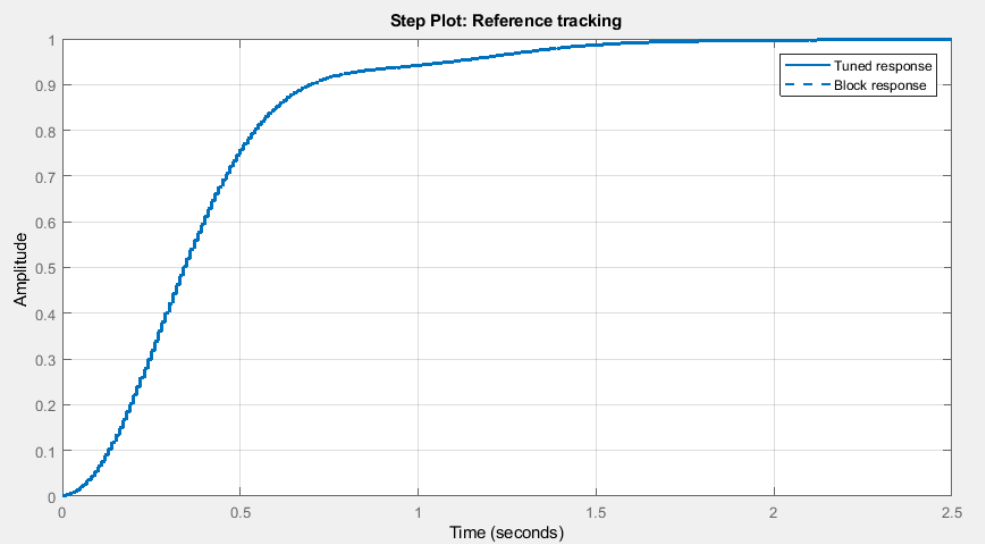

Figura 5.14: Respuesta sintonizada.

5.2.4. Resultados

Para probar el control se utiliza una referencia multi-escalón, encargada de comprobar el comportamiento a baja velocidad del sistema. En la Figura 5.15 se comprueba que la respuesta es óptima, estabilizándose rápidamente y mitigando las interferencias previamente explicadas.

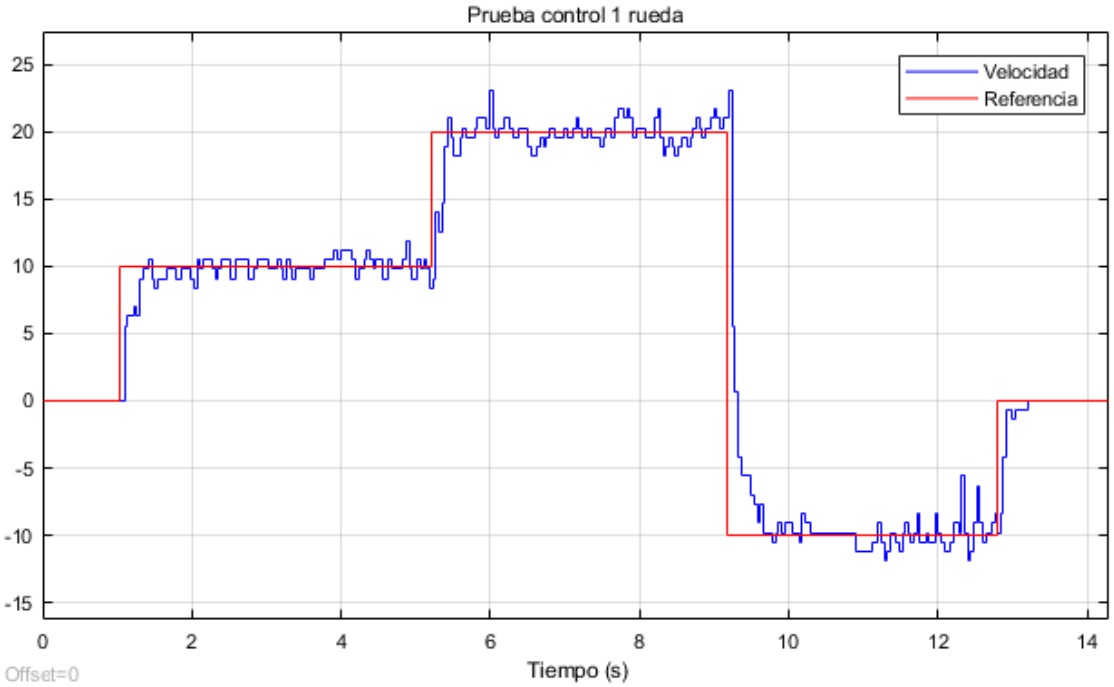

Figura 5.15: Prueba multi referencia - Levantado.

A continuación, se pone a prueba la respuesta ante perturbaciones, llegando a detener la rueda totalmente, el resultado del test se presenta en la Figura 5.16. En este caso supera el bloqueo sin problema para posteriormente estabilizarse tras un pequeño avance, representado por el pico de velocidad.

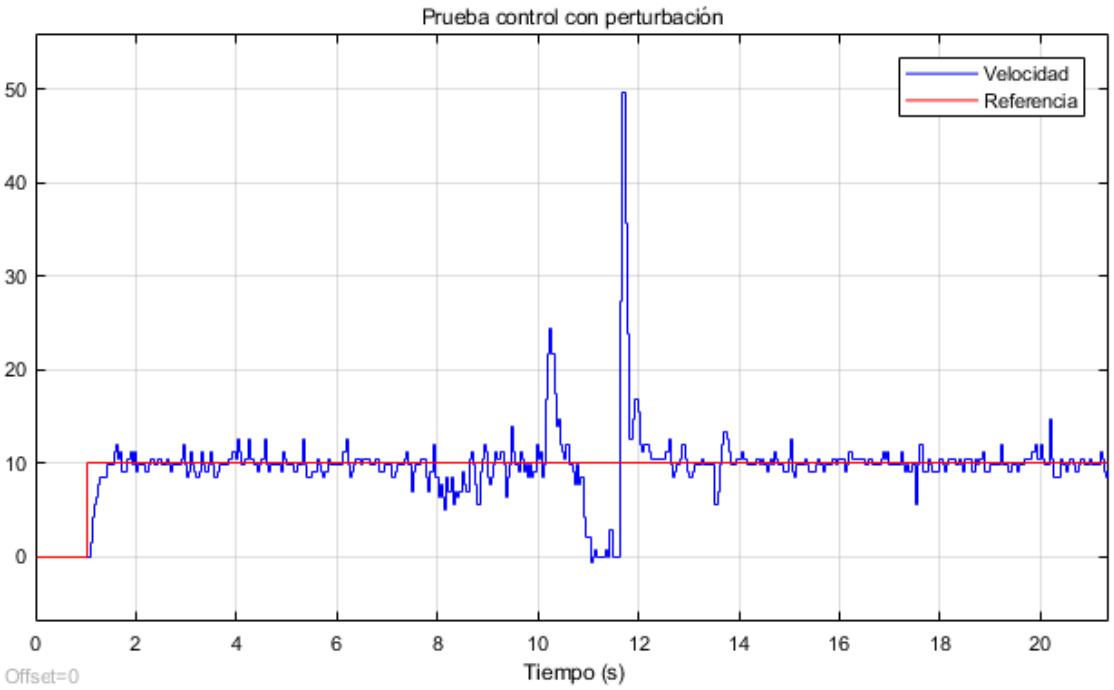

Figura 5.16: Prueba bloqueo - Levantado.

Debe tenerse en cuenta que estas pruebas se han realizado a muy baja velocidad. Al aumentar la misma, se vuelve prácticamente imposible bloquear la rueda, teniéndose un comportamiento mucho más robusto. Se concluye que la respuesta de la rueda es buena, por lo que se procede a repetir el proceso con el resto de las ruedas.

5.3. Control completo sin contacto con el suelo.

5.3.1. Identificación

Se repite el proceso del capítulo anterior para el resto de las ruedas, obteniéndose la respuesta de la Figura 5.17, siendo los datos adecuados para trabajar con ellos.

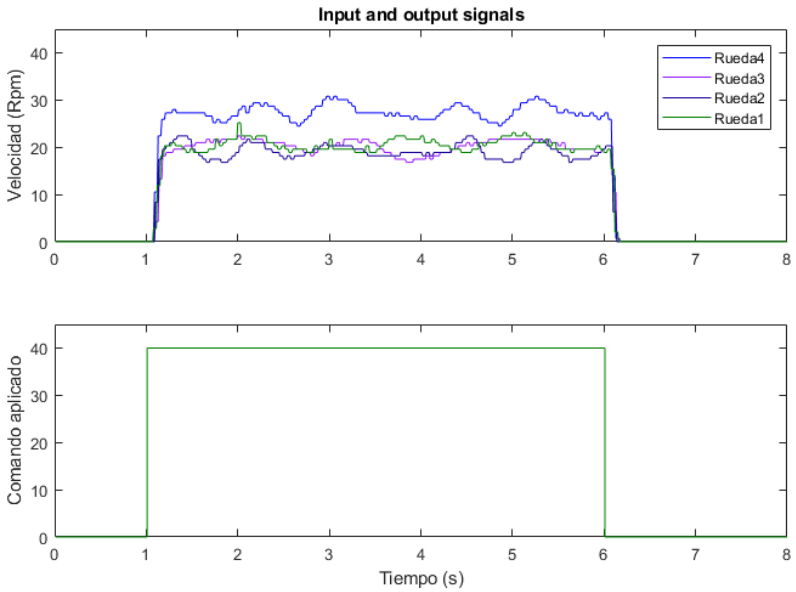

Figura 5.17: Respuesta identificación 4 ruedas - Levantado.

Tras procesar los datos se obtienen las funciones de transferencia de la Figura 5.18, todas ellas con un porcentaje de aproximación entorno al 85%. Se ha optado por las funciones de 2 polos, ya que han demostrado ser eficaces a la hora de sintonizar el controlador.

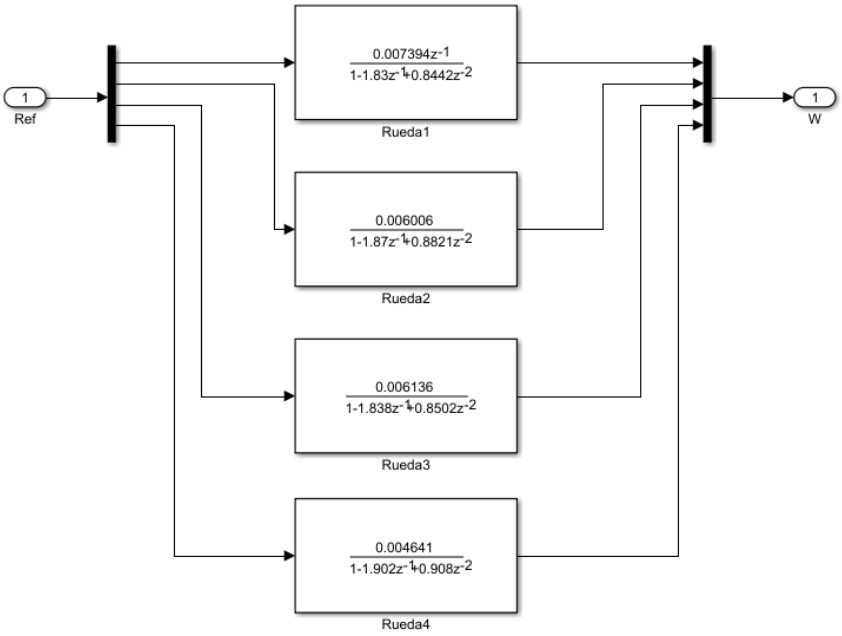

Figura 5.18: Funciones de transferencia sin contacto con el suelo.

5.3.2. Resultados

El esquema de control no sufre modificaciones, dándose una parte integral alta y una proporcional muy baja. Para comprobar su respuesta, se aplica una referencia de 10 rpm hasta recorrer 2 metros en el eje X, en la Figura 5.19 se comprueba que la velocidad de todas las ruedas se estabiliza en la referencia dada sin sobre oscilaciones.

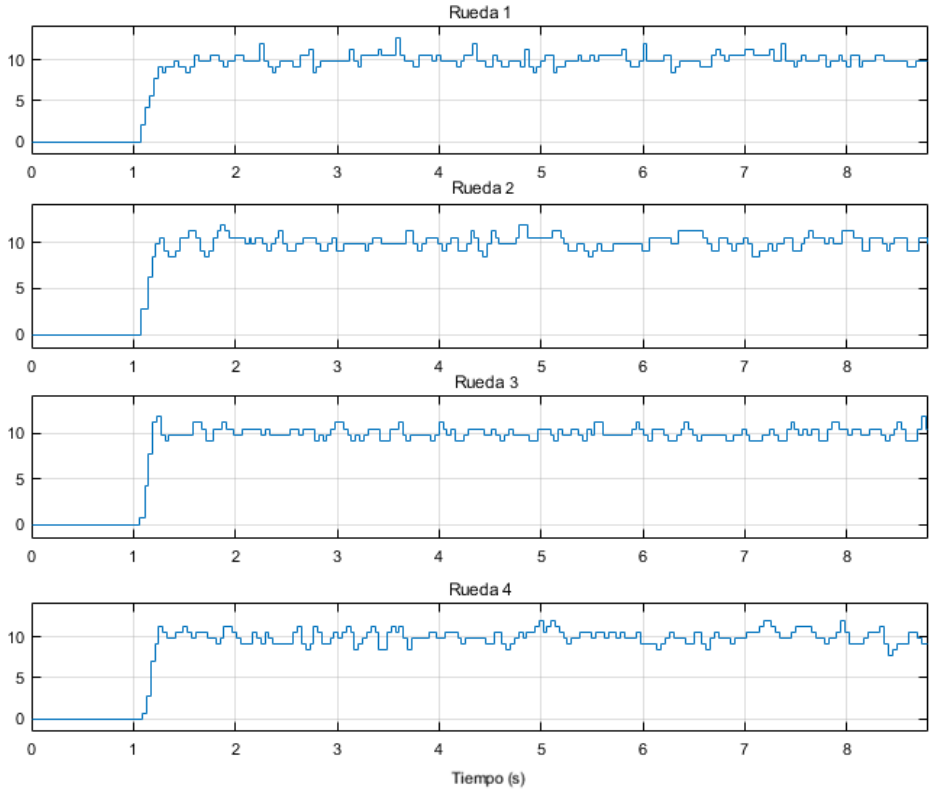

Figura 5.19: Respuesta de las 4 ruedas - Levantado.

Al comprobar la odometría, se advierte una gran mejoría respecto al test inicial. La velocidad en el eje X, en la Figura 5.20, se mantiene estable y en la Figura 5.21 se comprueba que apenas se desvía 1 cm en el eje Y. A la vista de estos resultados se concluye que el control es óptimo.

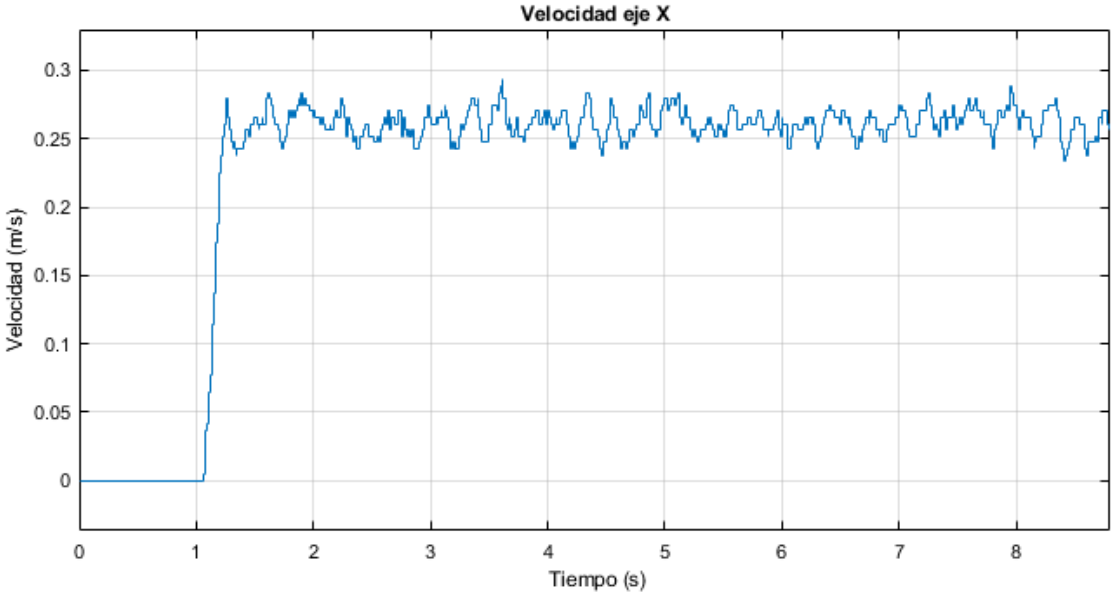

Figura 5.20: Velocidad eje X – Control.

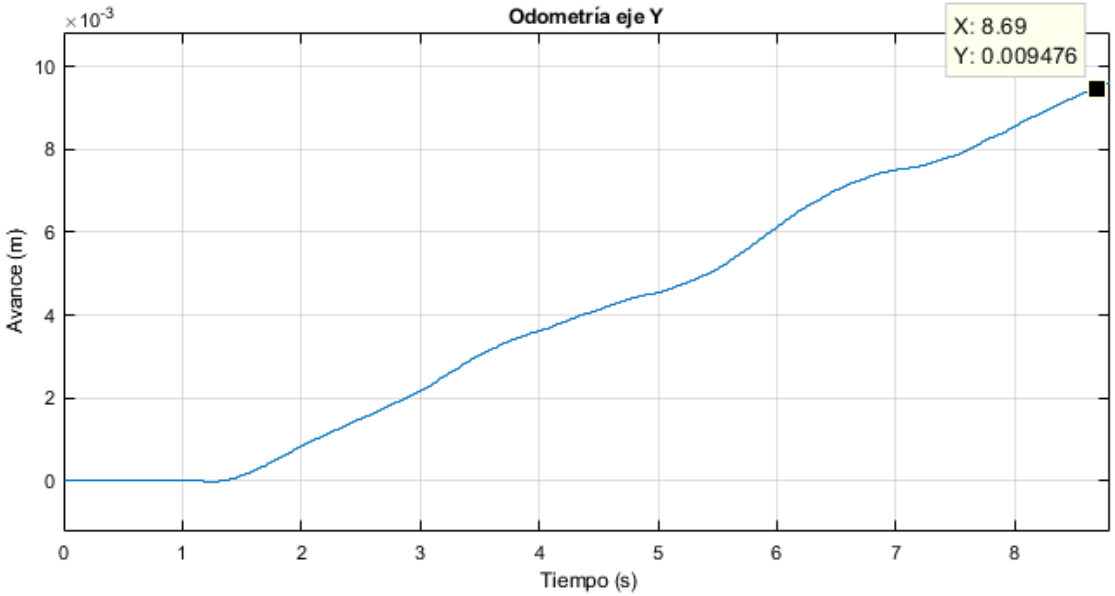

Figura 5.21: Avance eje Y - Control.

5.4. Control en el suelo del taller.

Tras conseguirse un control optimo en el aire, se va a realizar el mismo proceso con el Rambler en el suelo del taller. Al entrar la inercia del vehículo en juego, se van a plantear dos enfoques con el fin de obtener el modelo más fiable posible:

- Referencia escalón con cambio de signo directo.
- Referencia escalón con una parada entre avance y retroceso.

5.4.1. Identificación

5.4.1.1. SIN PARADA

En primer lugar, se realiza el test con un cambio brusco de sentido, dándose la respuesta de la Figura 5.22.

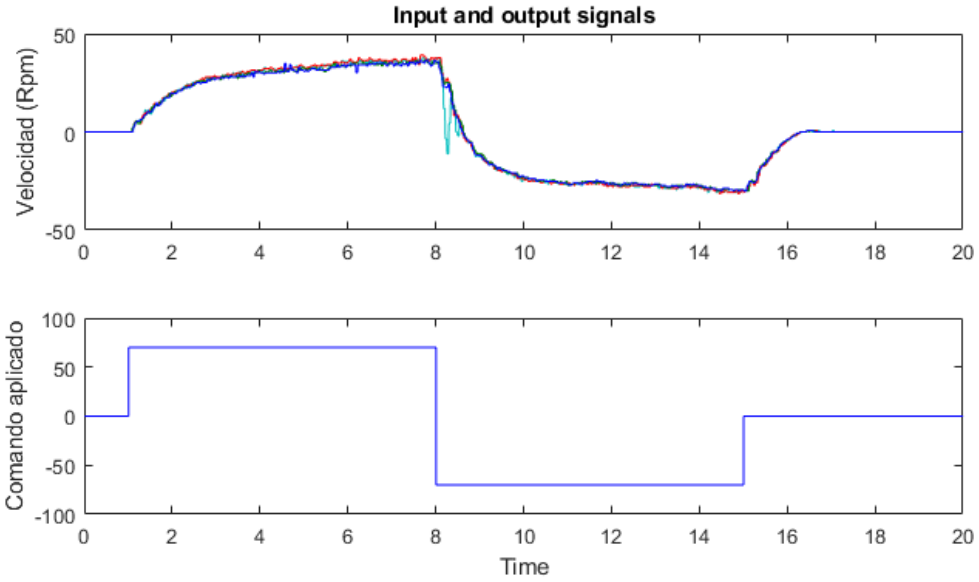

Figura 5.22: Respuesta sin parada.

Al observarse en detalle la transición en la Figura 5.23, se comprueba un comportamiento extraño en las ruedas, especialmente en la rueda trasera izquierda. Este comportamiento se puede apreciar en el video Identificación taller: Sin parada.

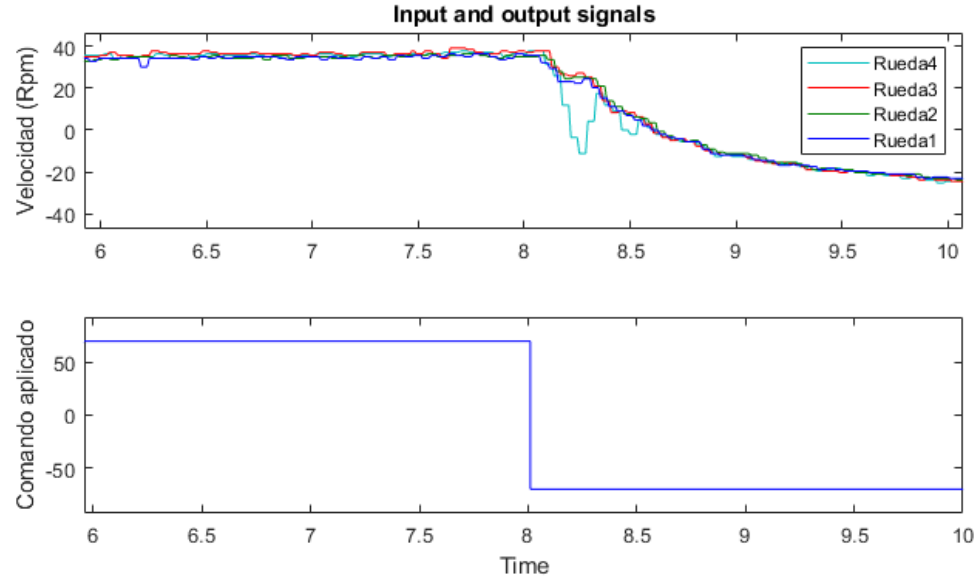

Figura 5.23: Detalle cambio de sentido.

Al realizar la identificación, se obtiene una respuesta con una aproximación del 88% visible en la Figura 5.24.

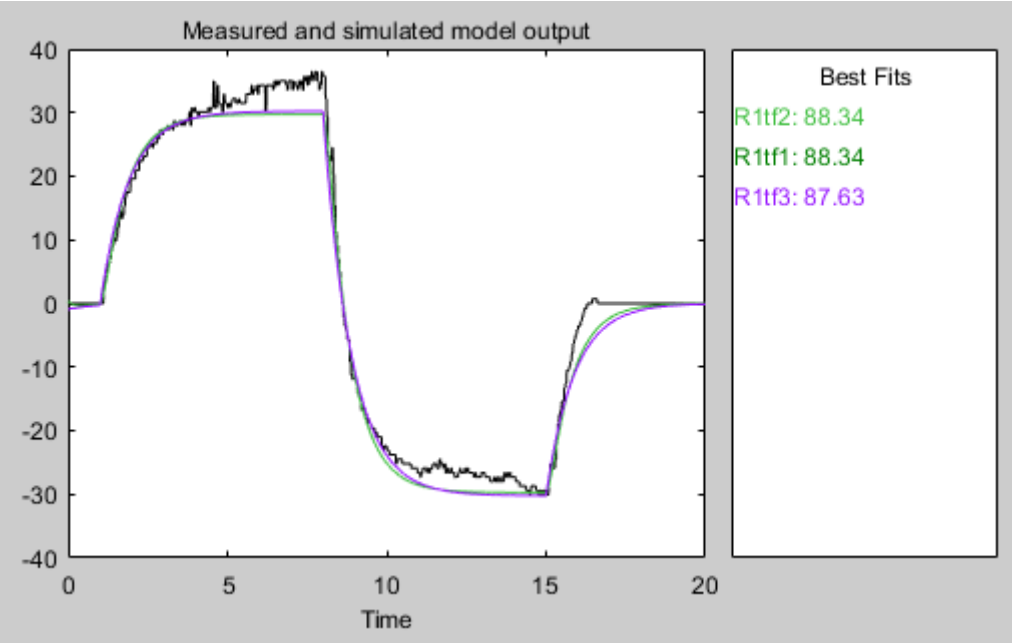

Figura 5.24: Respuesta de los modelos - Sin parada.

5.4.1.2. CON PARADA

A continuación, se modifica la referencia utilizada, dejando dos segundos en punto muerto para que el vehículo frene antes de cambiar de sentido, resultando en la respuesta de la Figura 5.25.

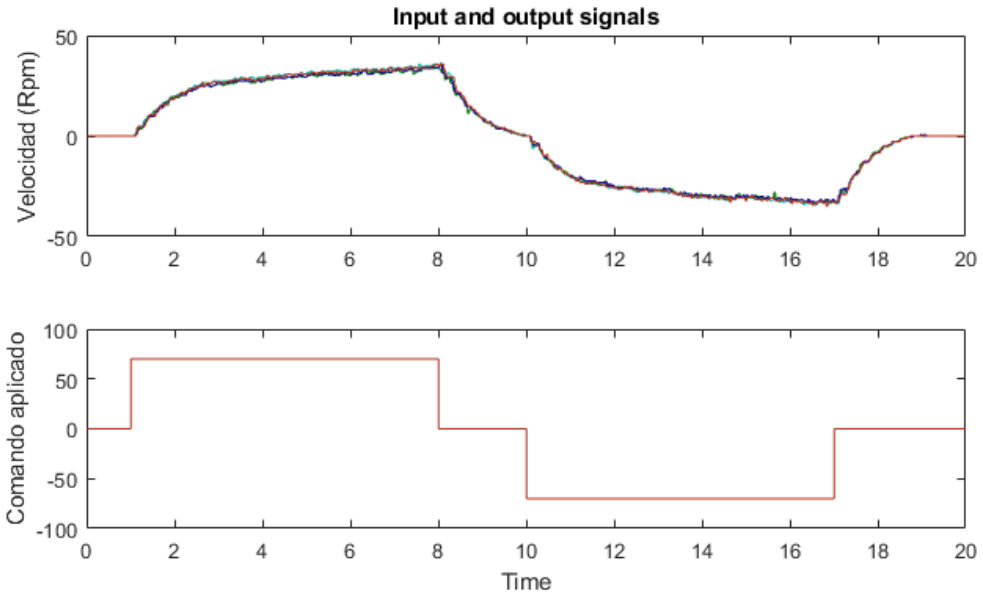

Figura 5.25: Respuesta con parada.

En este caso la transición es mucho más suave tal y como se aprecia en la Figura 5.26 y en el video Identificación Taller: Con parada.

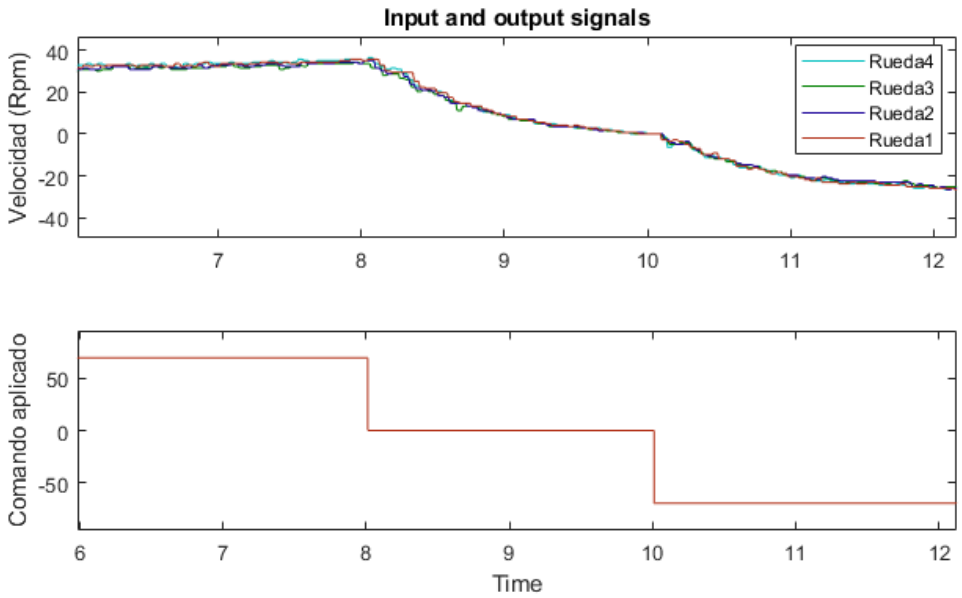

Figura 5.26: Detalle frenado.

Además, al anular la anomalía al cambiar de sentido la identificación supera a la del capítulo anterior, consiguiéndose los 92% de aproximación de la Figura 5.27.

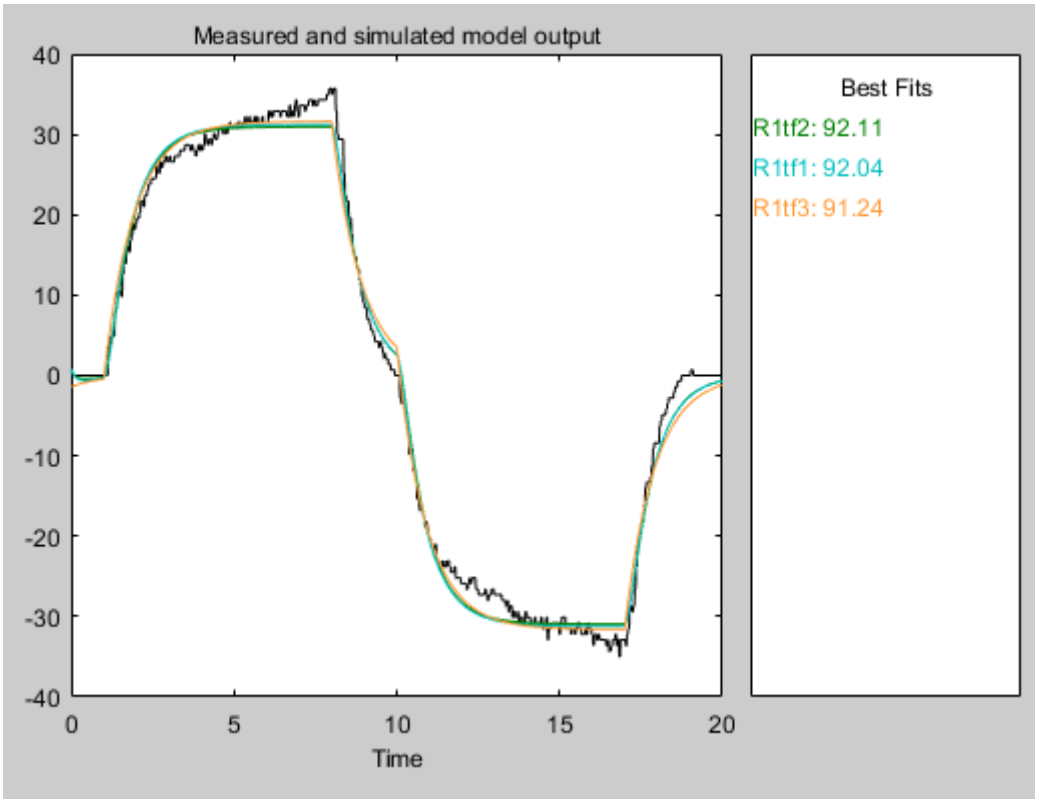

Figura 5.27: Respuesta de los modelos - Con parada.

En la Figura 5.28 se presentan las funciones de transferencia resultantes, disponiendo todas ellas de una fiabilidad entorno al 92 % ya mencionado. Estos modelos servirán para llevar a cabo el control, ya que han demostrado ser más fieles al sistema real.

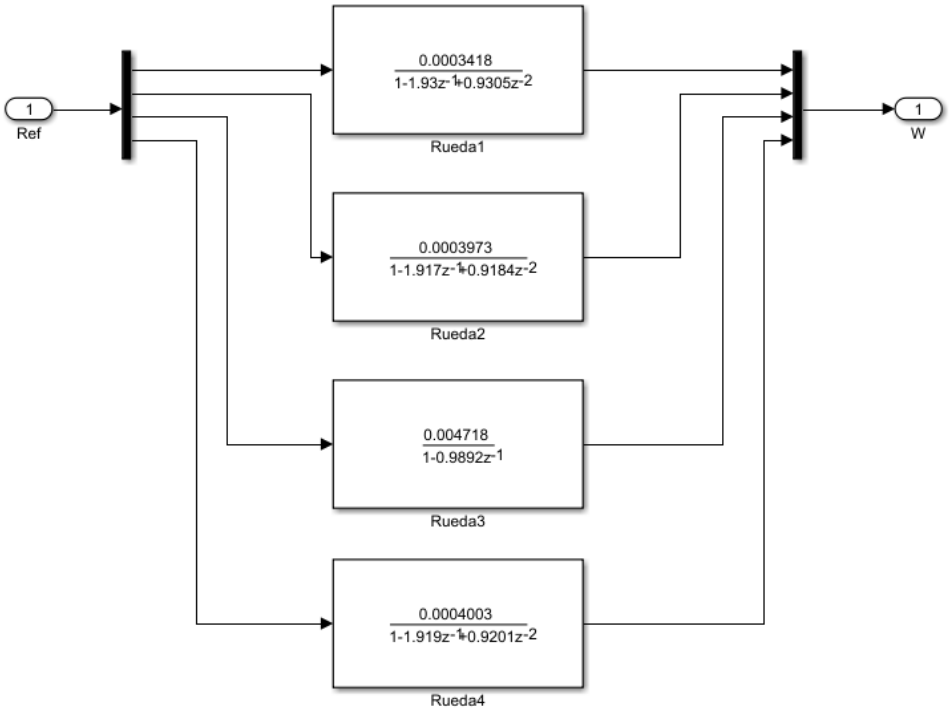

Figura 5.28: Funciones de transferencia - Taller.

5.4.1.3. PRESIÓN NEUMÁTICOS

Al realizar la prueba anterior, se observó que dos de los neumáticos tenían una presión menor a la recomendada por el fabricante. Tras corregirlo, se repite la experiencia y se comparan las respuestas (Figura 5.29), mejorando la velocidad en hasta 8 rpm. Esta mejoría se debe a que al disminuir la presión del neumático aumenta el rozamiento en el mismo, al aumentar la superficie de contacto.

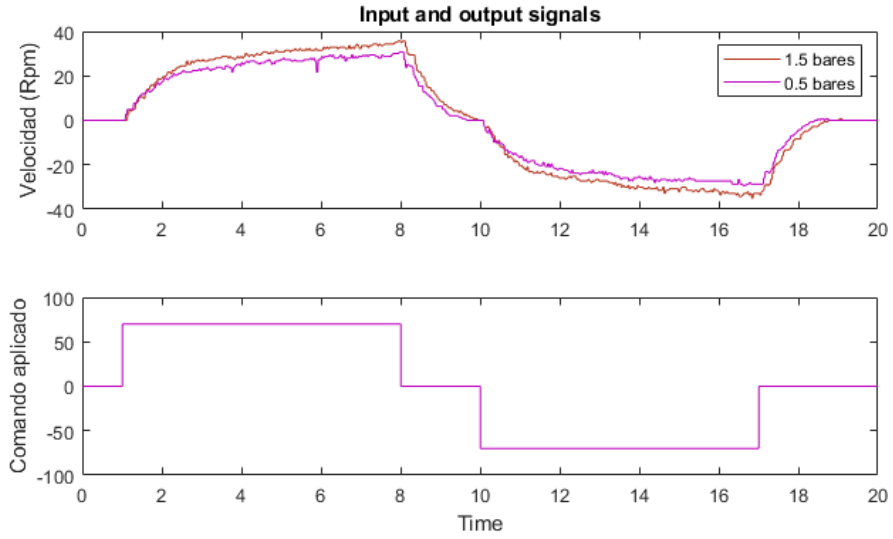

Figura 5.29: Efecto de la presión de las ruedas.

5.4.2. Control

Se repite la estrategia de sintonizar un controlador PI, tras calibrar experimentalmente el controlador se opta por una respuesta lenta ya que respuestas más rápidas tienden a una ganancia proporcional muy alta, lo que produce sobre oscilaciones. En la Figura 5.30 se presenta la respuesta sintonizada, para unos parámetros totalmente apuestos al control desarrollado con el vehículo en el aire, ya que en este caso la acción proporcional es mucho mayor a la integral.

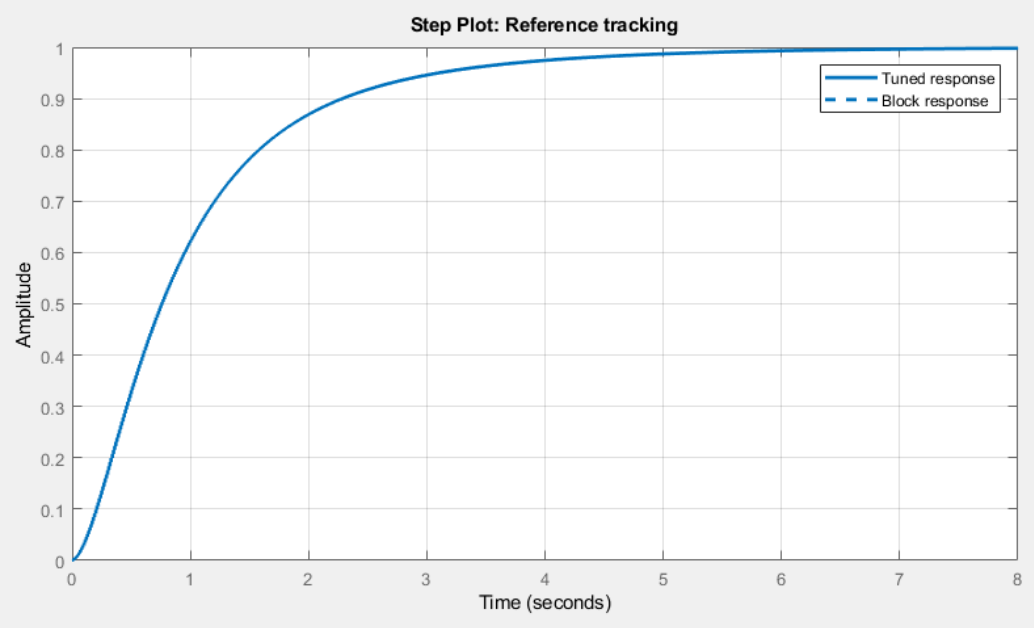

Figura 5.30: Respuesta sintonizada - Taller.

Debe tenerse en cuenta la importancia de tener una respuesta similar en todos los motores, ya que la velocidad de cada uno de ellos va a tener repercusión en el resto, facilitando la aparición de sobre oscilaciones. Para asegurar un comportamiento similar, se dispone el mismo factor proporcional en todas las ruedas, modificando únicamente la acción integral, dando lugar a los parámetros finales de la Tabla 4.

	Rueda 1	Rueda 2	Rueda 3	Rueda 4
Ganancia Proporcional	8	8	8	8
Ganancia Integral	0.72	1.52	1.26	1.38

Tabla 4: Parámetros PI - Taller.

5.4.3. Resultados

Para probar el control se va a utilizar una referencia en rampa, puesto que en su conducción se utilizan señales de este tipo. Sin embargo, al no incluir el termino derivativo el controlador no es capaz de eliminar por completo el error en este tipo de señales, tal y como se aprecia en la Figura 5.31. En principio esto no es un problema, ya que el Rambler dispone de una capa de control superior encargada de la navegación. Esta capa se encarga de corregir estos pequeños errores, por lo que basta con conseguir un movimiento fluido y estable, siendo esta una respuesta aceptable.

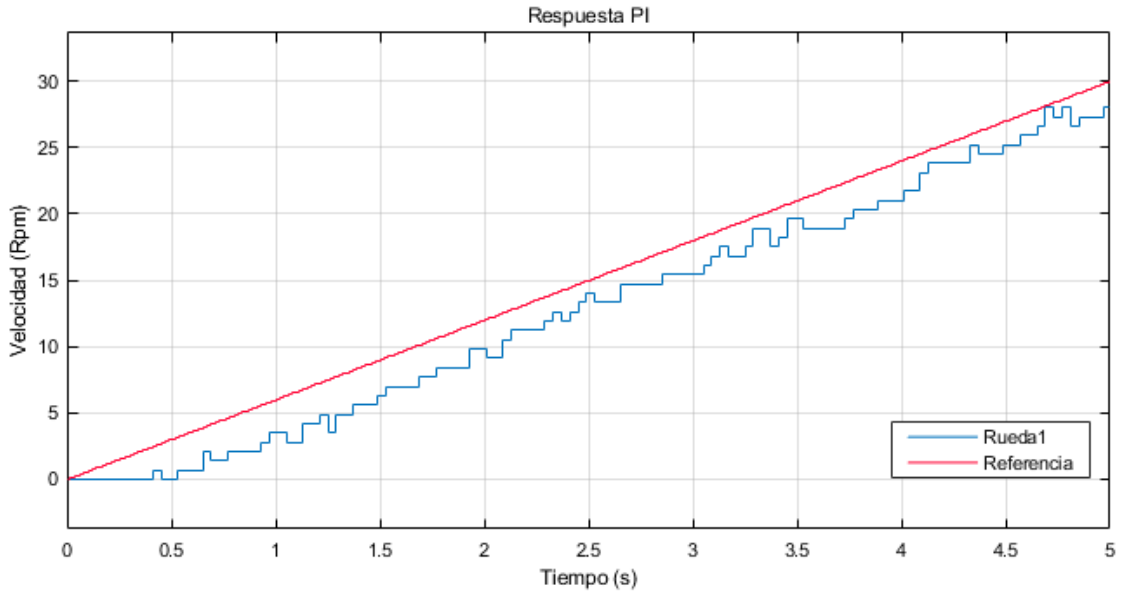

Figura 5.31: Prueba control PI - Taller.

De todos modos, se prueba a implementar un control PID, pero aparecen oscilaciones en la respuesta, visibles en la Figura 5.32. Hay que destacar que no se ha insistido en afinar este control, por lo que tiene margen de mejora, no quedando descartado en caso de ser necesaria una mejor respuesta.

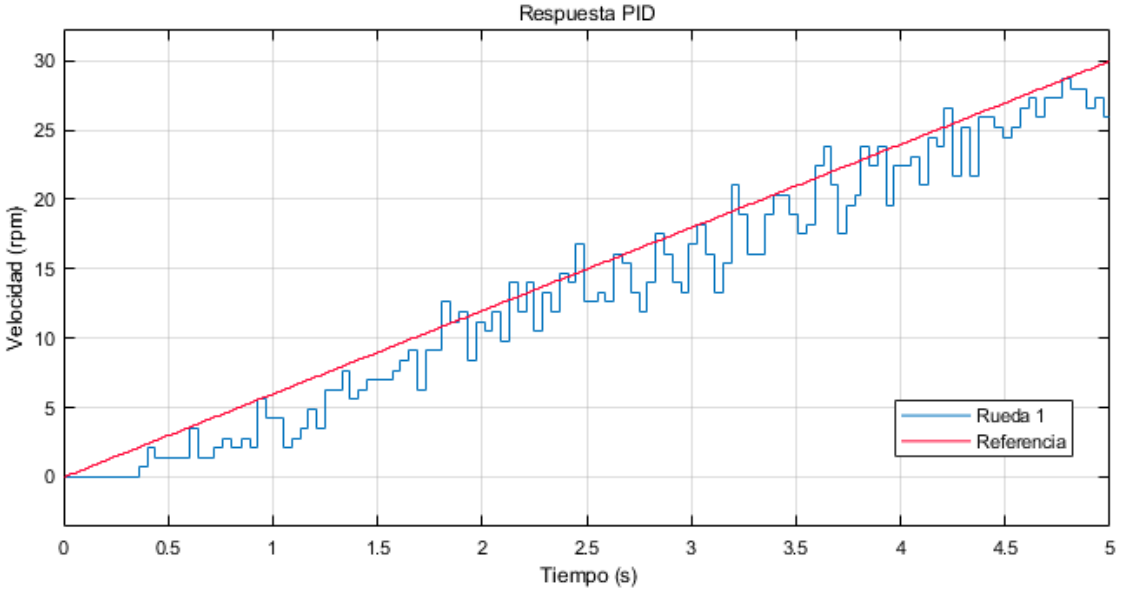

Figura 5.32: Prueba control PID - Taller.

6. EFECTOS DEL TERRENO

Tras conseguirse un control satisfactorio con el vehículo en el pavimento industrial pulido del taller, se van a realizar pruebas en distintos terrenos y situaciones, con el fin de establecer unas pautas de control. Debe tenerse en cuenta que las estrategias de control presentadas se basan únicamente en los modelos conseguidos, por lo que deben tomarse como una guía para su posterior prueba y ajuste experimental.

6.1. Asfalto

El primer test se lleva a cabo en el parking de la propia facultad, proporcionando el asfalto un terreno con mayor rozamiento respecto al taller.

6.1.1. Identificación

Se utiliza como referencia la señal escalón con parada, con el fin de comparar la respuesta con la obtenida en el taller, obteniéndose la respuesta de la Figura 6.1.

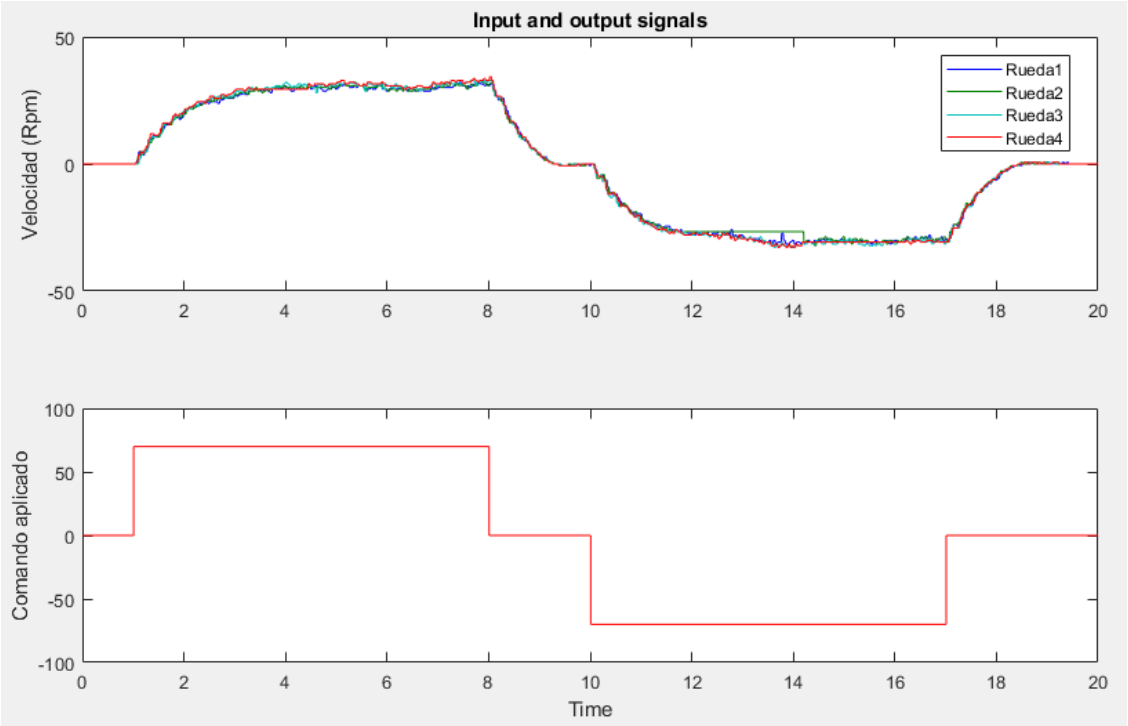

Figura 6.1: Respuesta identificación - Asfalto.

En este caso la respuesta de los modelos es incluso mejor a la del taller, tal y como se aprecia en la Figura 6.2.

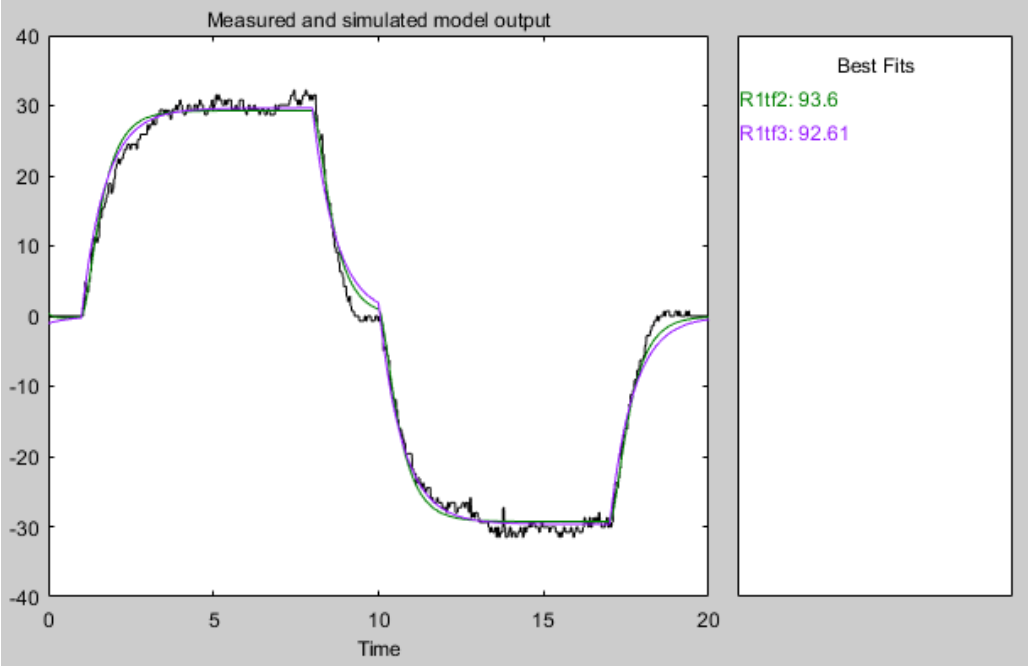

Figura 6.2: Respuesta modelo - Asfalto.

En la Figura 6.3 se presentan las funciones de transferencia obtenidas con una aproximación entorno al 93.5%. Debe destacarse que son muy similares a las calculadas en el pavimento industrial pulido.

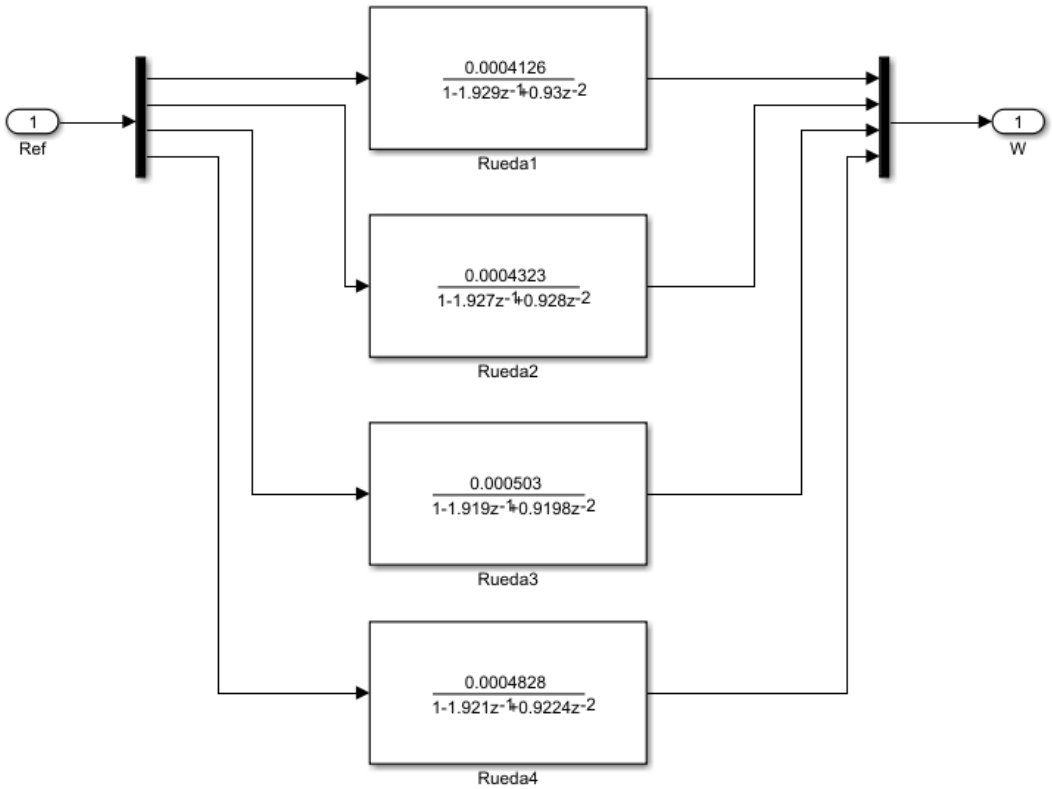

Figura 6.3: Funciones de transferencia - Asfalto.

6.1.2. Comparación terrenos

Antes de desarrollar una estrategia de control especifica se debe comprobar si realmente es necesaria. Para ello, se compara en la Figura 6.4 la respuesta de la rueda trasera izquierda en este terreno frente a la obtenida en el pavimento industrial pulido.

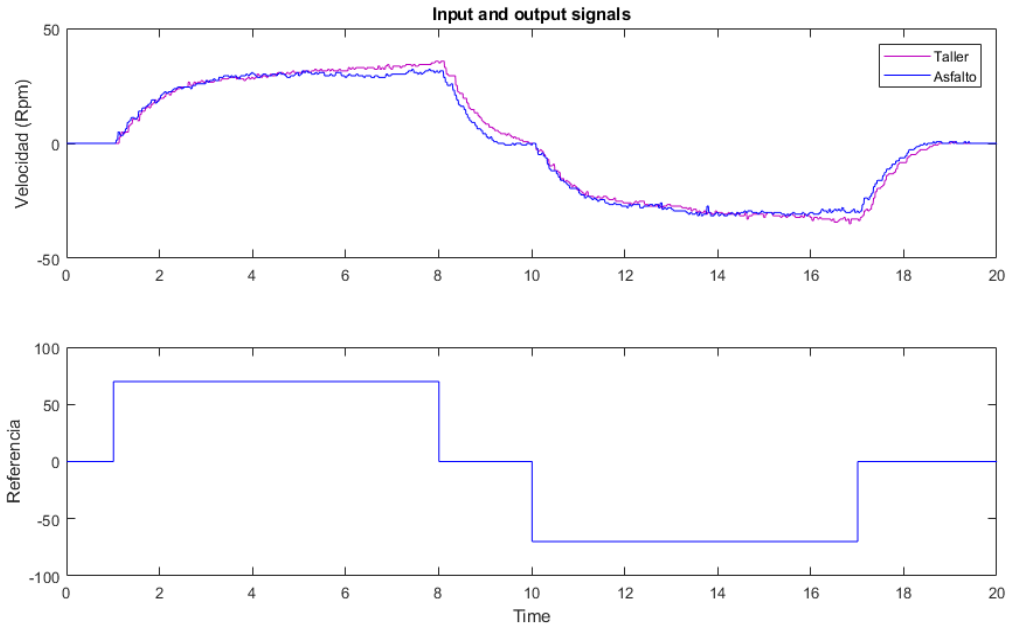

Figura 6.4: Comparación Taller - Asfalto.

Inicialmente el comportamiento no resulta muy distinto, sin embargo, se aprecian diferencias importantes en la velocidad alcanzada y en el tiempo de frenado. En la Figura 6.5 se detallan estas diferencias:

- En el taller el vehículo sigue acelerando lentamente, superando en hasta 5 rpm a la respuesta en el asfalto tanto en el avance como en el retroceso, lo que supone una diferencia del 16%.
- En el asfalto la velocidad de frenada es mucho mayor, debe tenerse en cuenta que parte de una velocidad menor, pero la pendiente es más pronunciada, reduciendo el tiempo de frenada de 2 segundos a 1.2, casi la mitad.

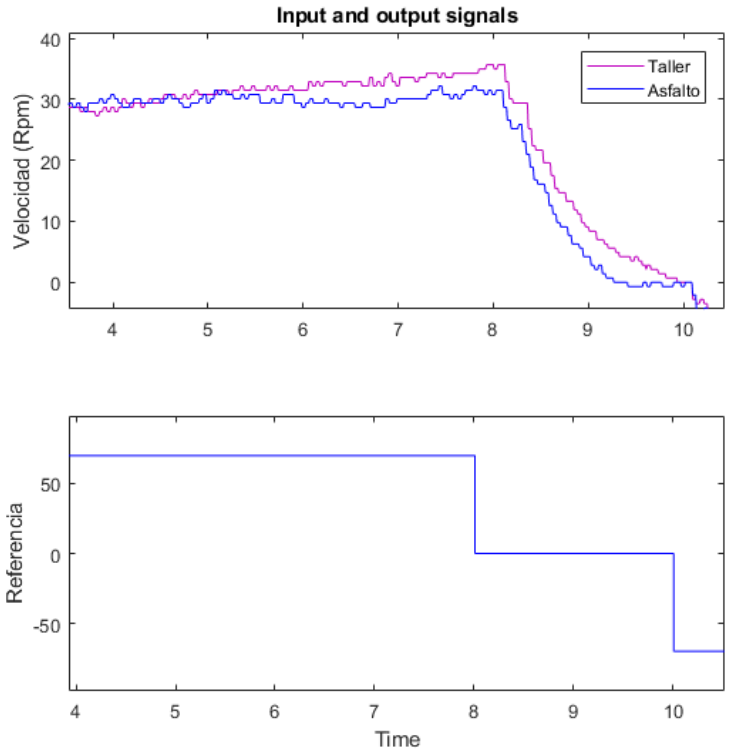

Figura 6.5: Detalle comparación.

A la vista de estas diferencias se van a realizar ajustes en el control, con el objetivo de suplir el efecto causado por el mayor rozamiento del terreno.

6.1.3. Estrategia de control

Se ha configurado un nuevo control, para obtener una respuesta muy similar a la usada en el taller. Al compararlo con el control previo, la respuesta resulta más lenta en las ruedas traseras, y más rápida en las delanteras.

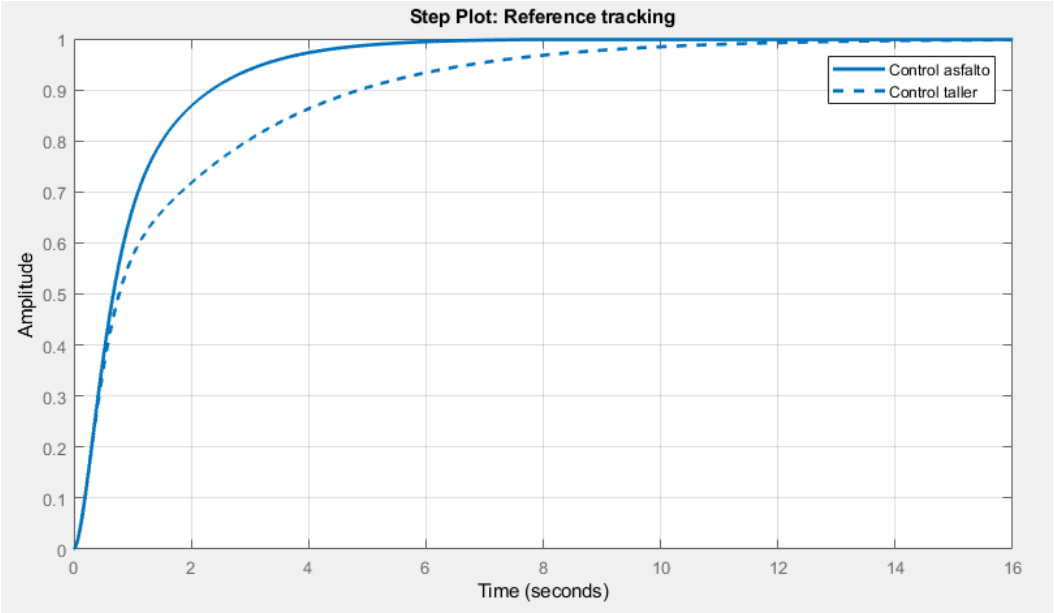

Figura 6.6: Comparación Asfalto – Taller, rueda trasera.

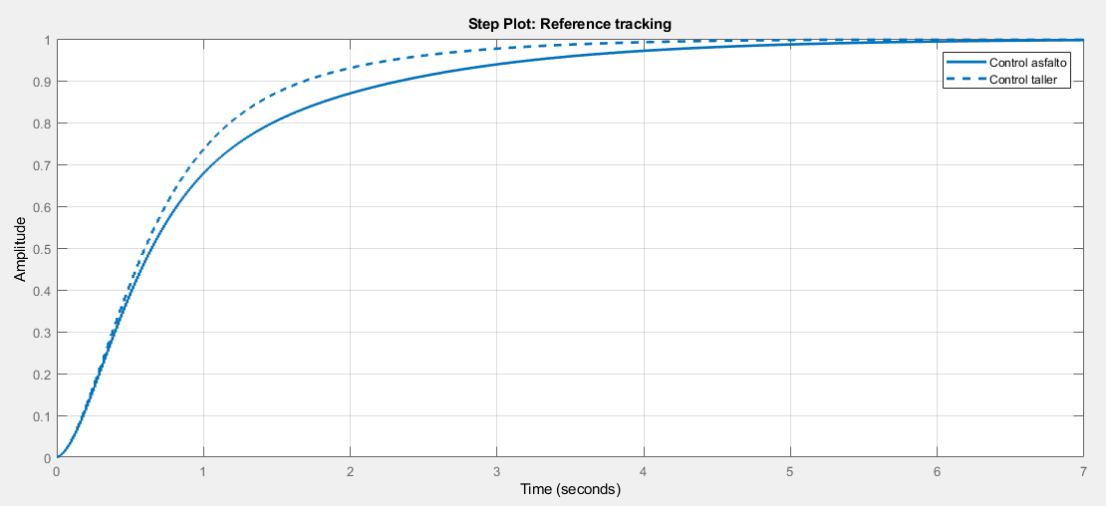

Figura 6.7: Comparación Asfalto – Taller, rueda delantera.

Los parámetros resultantes tienen una ganancia proporcional idéntica, pero se observa que acción integrativa aumenta en las ruedas traseras y disminuye en las delanteras, esto puede darse por existir una pequeña pendiente, los parámetros teóricos se encuentran en la Tabla 5. El control sintonizado resulta muy similar al desarrollado en el taller, por lo que se recomienda probarlo en el Asfalto y ajustar experimentalmente en caso de ser necesario.

	Rueda 1	Rueda 2	Rueda 3	Rueda 4
Ganancia Proporcional	8	8	8	8
Ganancia Integral	1.3	1.2	0.85	1.7

Tabla 5: Parámetros PI teóricos - Asfalto.

6.2. Campo

Las últimas pruebas se llevan a cabo en el terreno arenoso a las afueras de la escuela. Este terreno presenta menor rozamiento que el asfalto, pero resulta interesante debido a la presencia de desniveles, baches y vegetación.

6.2.1. Identificación

Se comienza con la misma consigna utilizada en los otros terrenos, aunque en este caso la respuesta presenta interferencias importantes, apreciables en la Figura 6.8, causadas por lo abrupto del terreno.

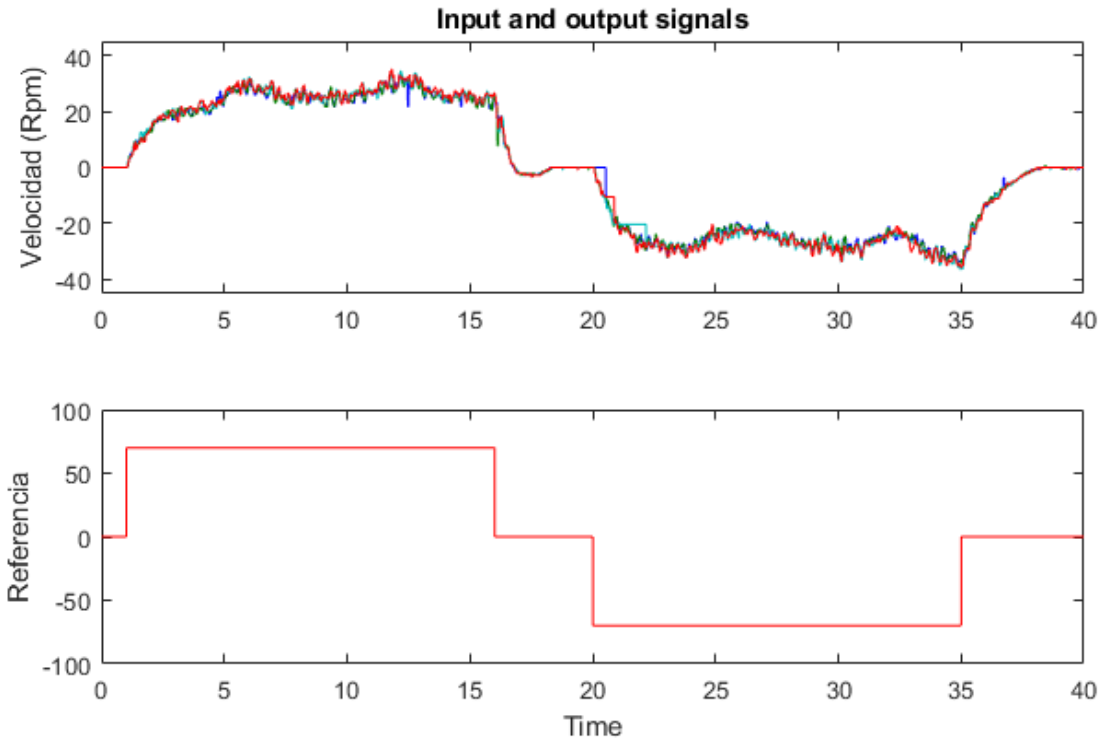

Figura 6.8: Prueba escalón - Tierra.

A la vista de estos resultados, se modifica la referencia añadiendo escalones de distinto valor, para conseguir una identificación más fiable al disponer de más datos. En la Figura 6.9 está el resultado. Hay que destacar las interferencias arbitrarias debidas al terreno, como las presentes en el detalle de la Figura 6.10. El movimiento del vehículo está registrado en el video Multiescalon tierra.

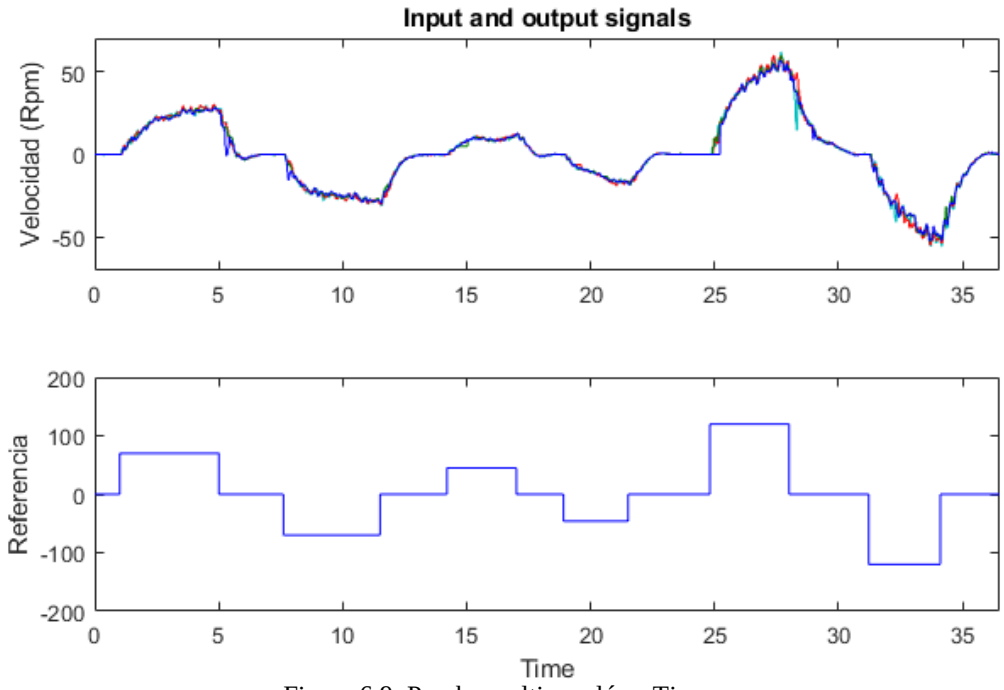

Figura 6.9: Prueba multi escalón - Tierra.

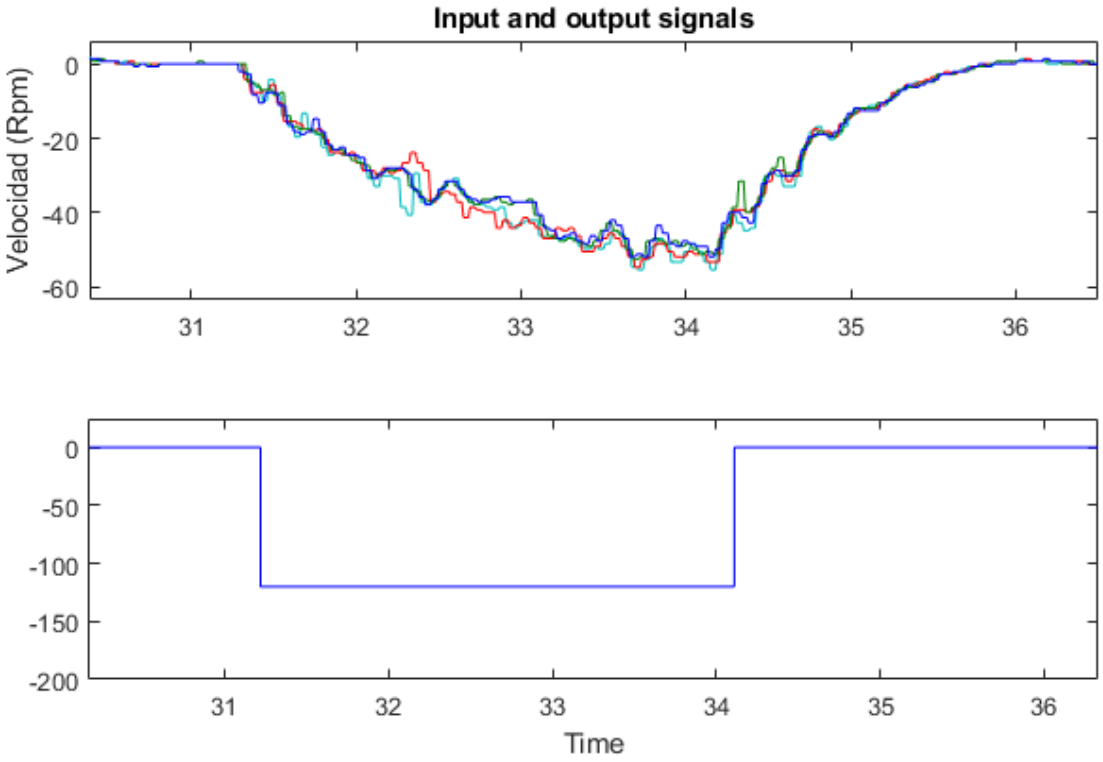

Figura 6.10: Detalle prueba multi escalón - Tierra.

En la Figura 6.11 se aprecia que la respuesta de las funciones de transferencia no acaba de ceñirse al sistema real, ya que no son capaces de modelar el comportamiento correctamente debido a las perturbaciones. De todos modos, en la Figura 6.12 se presentan las funciones obtenidas, las cuales rondan el 82% de aproximación.

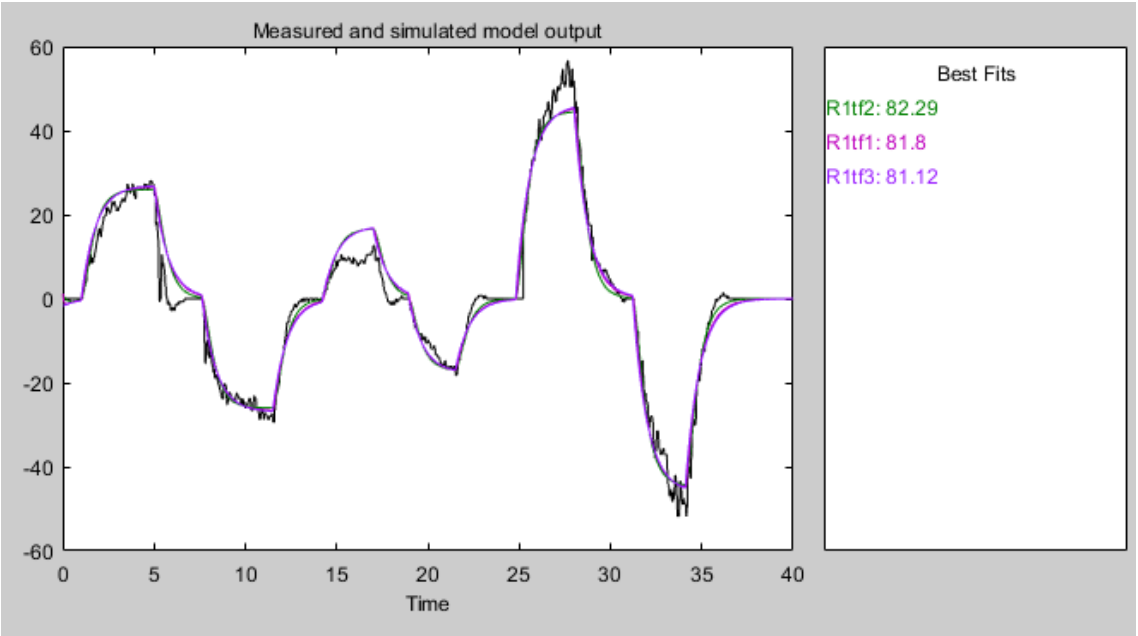

Figura 6.11: Respuesta del modelo - Tierra.

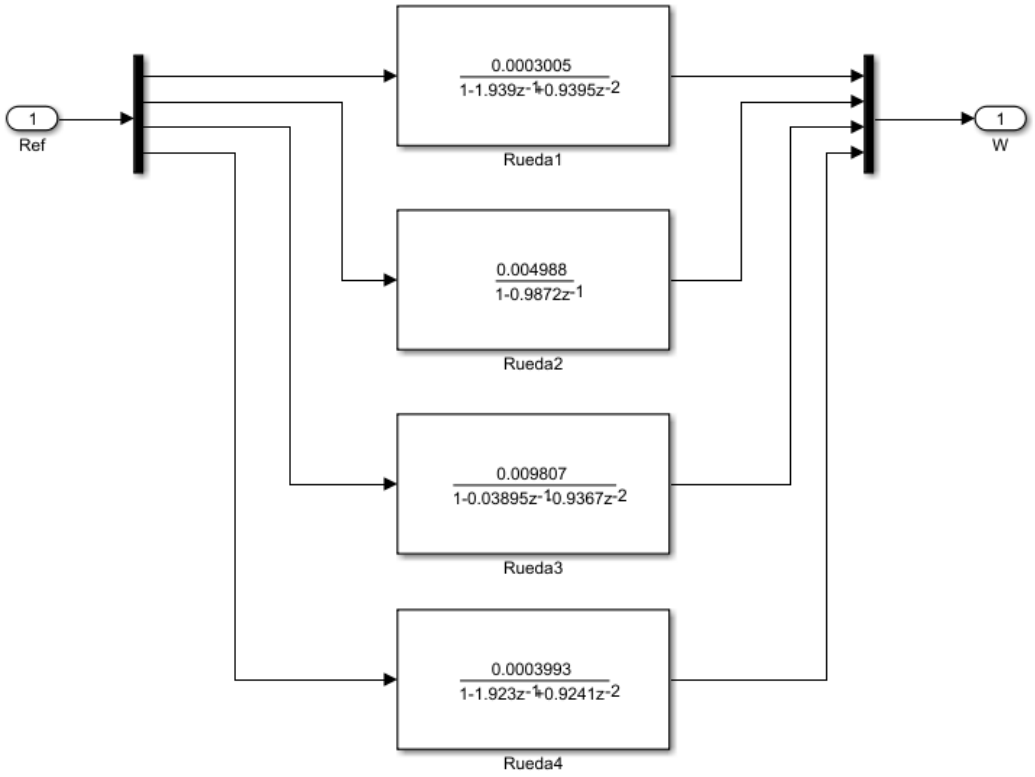

Figura 6.12: Funciones de transferencia - Tierra.

6.2.2. Prueba control

Se va a probar el control desarrollado para el taller en este terreno, para comprobar su respuesta ante las perturbaciones y comprobar si es posible adaptarlo de manera sencilla. Se utiliza como consigna una rampa de pendiente suave que se termina estabilizando. En la Figura 6.13 se observan los resultados de la prueba.

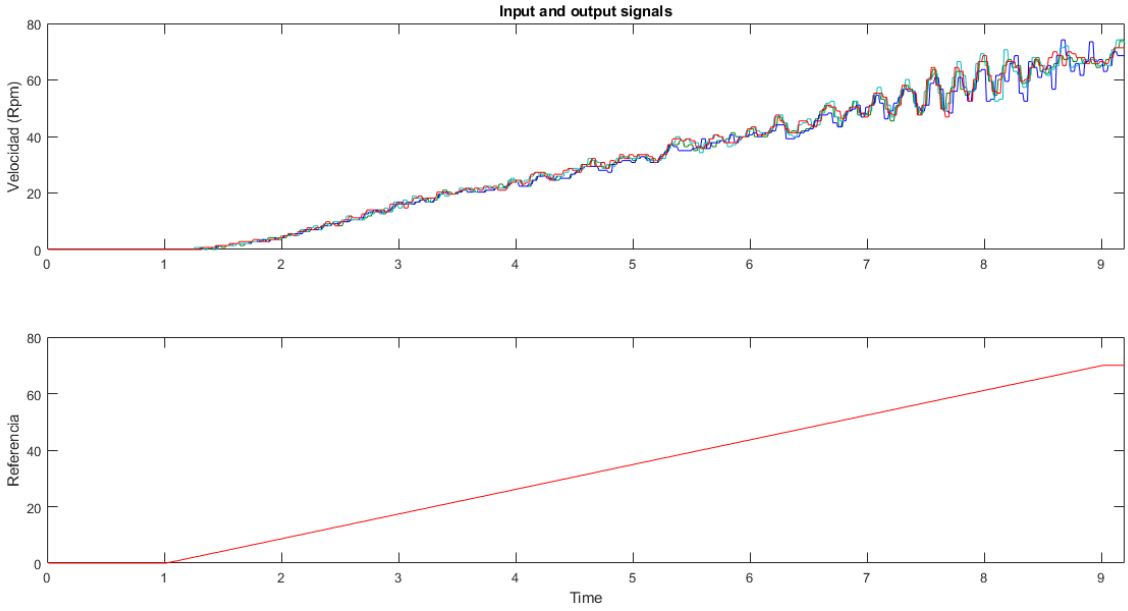

Figura 6.13: Prueba control taller – tierra

Al comenzar la prueba el vehículo se encuentra en terreno llano a baja velocidad, progresando correctamente. Sin embargo, al aumentar su velocidad y encontrar el primer desnivel del terreno, el sistema comienza a sobre oscilar. Esta experiencia se encuentra documentada en el video Prueba control tierra.

6.2.3. Estrategia de control

En este caso el mayor problema reside en las grandes perturbaciones del terreno. La herramienta *Autotune* propone un control similar al del taller, con una constante proporcional mucho mayor a la integral. En la Figura 6.14 se presenta una propuesta de control para la rueda derecha trasera (Rueda1), para las siguientes ganancias:

- Proporcional: 10
- Integral: 0.6

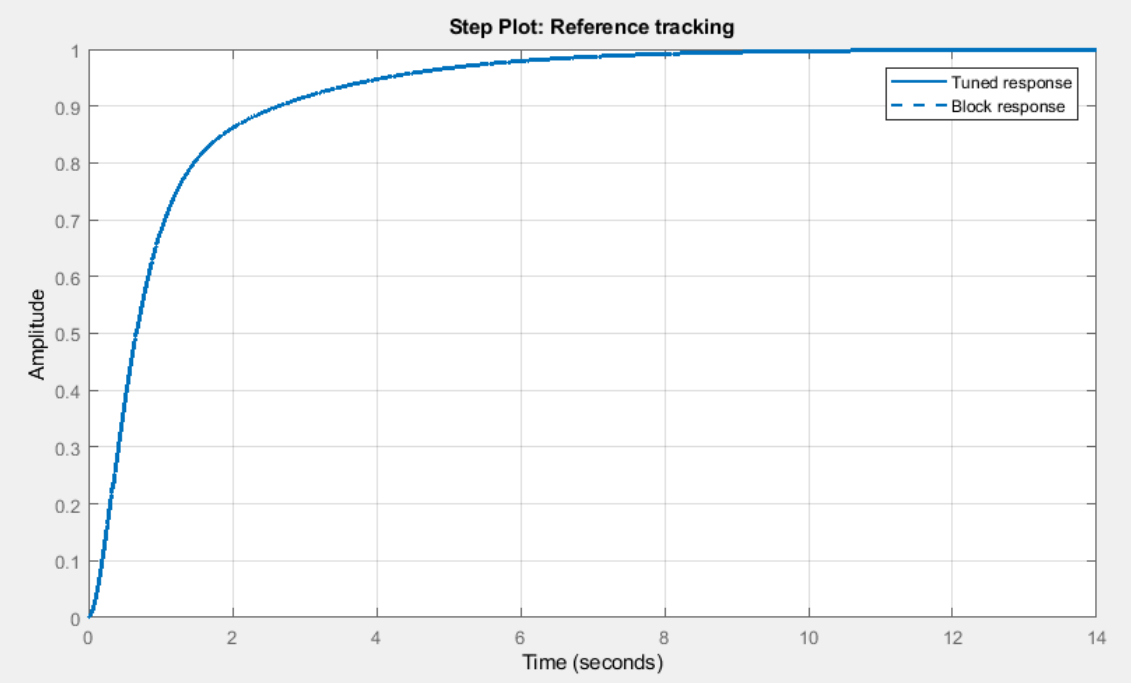

Figura 6.14: Respuesta PI.

Sin embargo, al tener la ganancia proporcional tan elevada el sistema será propenso a volverse inestable, como ya ocurrió en el capítulo anterior. En este caso cobra gran importancia ajustar los valores experimentalmente y visualizar la respuesta ante distintas perturbaciones, también se resultaría interesante incluir el termino derivativo, ya que la respuesta va a ser propensa a cambios repentinos.

6.3. Giro

Por último, se va a estudiar el comportamiento del vehículo al girar en el mismo terreno abrupto del capítulo previo. Primero se intenta que el Rambler realice un giro sobre si mismo desde el reposo total, pero es incapaz de superar la fricción incluso aplicando a los motores el 80% de su potencia máxima. En el video Intento giro se aprecia el intento, no llegando a superar el rozamiento estático.

Se decide realizar un giro en movimiento, aplicando una referencia 6 veces mayor a los motores de la parte izquierda, modificando el esquema de Arduino tal y como se indica en la Figura 6.15. Hay que destacar que para ganancias de menor valor el giro apenas era apreciable.

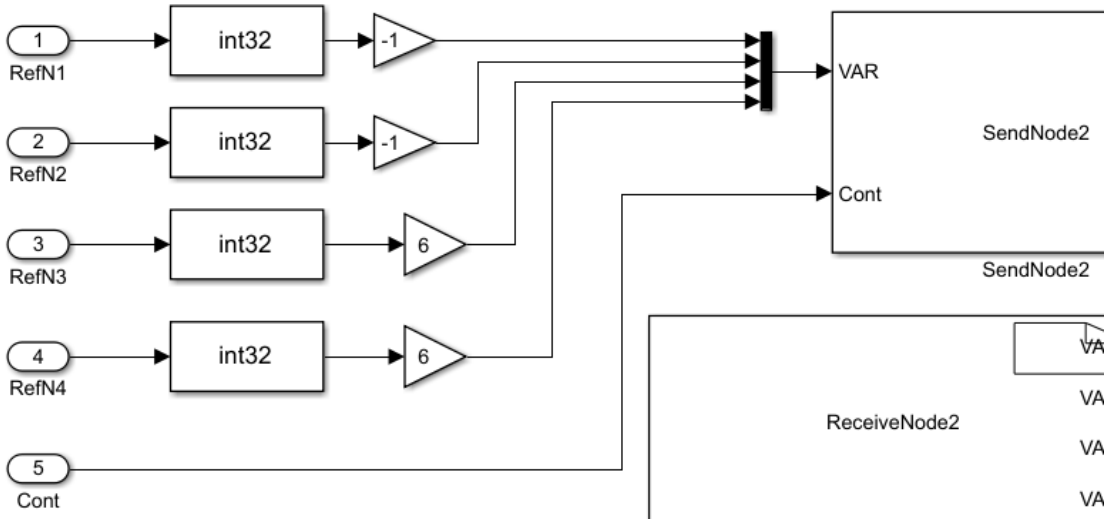

Figura 6.15: Aumento de la referencia.

6.3.1. Identificación

Con esta modificación se consigue un giro completo bastante abierto con una velocidad línea muy elevada. Debido a esta velocidad el vehículo realiza un recorrido extenso, encontrando en su camino numerosas pendientes y desniveles. Esto resulta en una respuesta muy irregular, en la cual se presentan numerosas perturbaciones de gran magnitud, que en muchos casos afectan únicamente a una de las ruedas. En la Figura 6.16 se presentan los datos recogidos, además de poder visualizar el recorrido en el video Giro tierra.

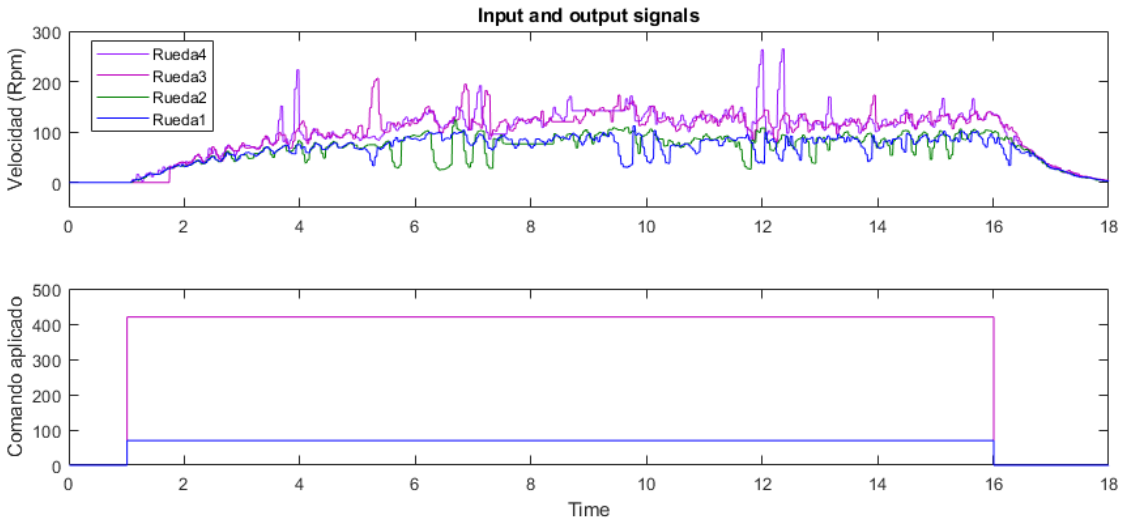

Figura 6.16: Respuesta obtenida - Giro.

En la Figura 6.17 se detalla la diferencia en la respuesta de ambos pares de ruedas. Las velocidades de cada lado del vehículo no presentan la diferencia esperada al aplicar un comando 6 veces mayor. Esto se debe a que las ruedas de la izquierda giran con gran velocidad, y aumentan la velocidad lineal del Rambler, consiguiendo mayor velocidad con menor comando en las ruedas del lado opuesto. Este efecto explica porque al aplicar comandos más similares el vehículo apenas giraba.

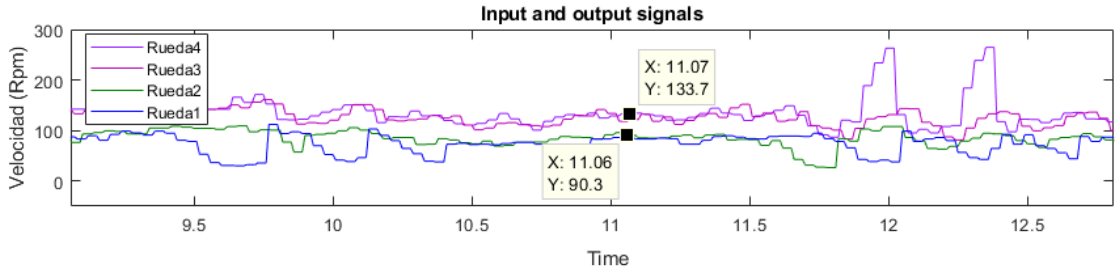

Figura 6.17: Detalle de la respuesta - Giro.

Con una respuesta tan irregular resulta muy complicado obtener un modelo fiable. En la Figura 6.18 se puede observar que el porcentaje de aproximación conseguido es bastante pobre. Aun así, se continua el proceso y se calculan las funciones de transferencia, indicadas en la Figura 6.19.

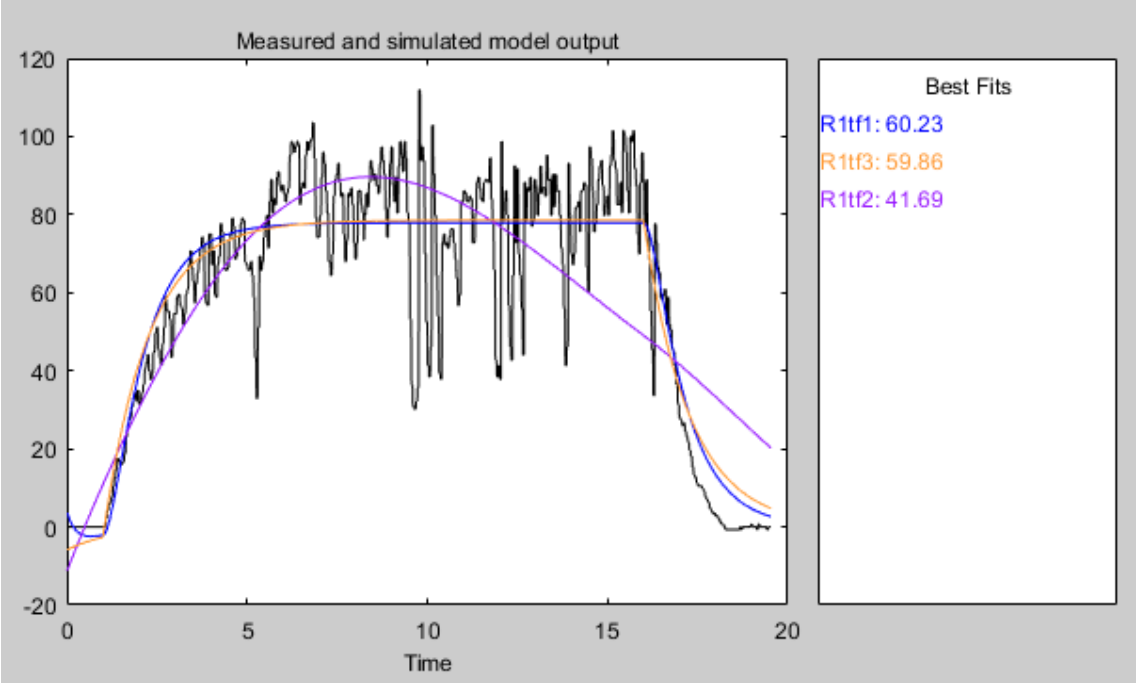

Figura 6.18: Respuesta modelos.

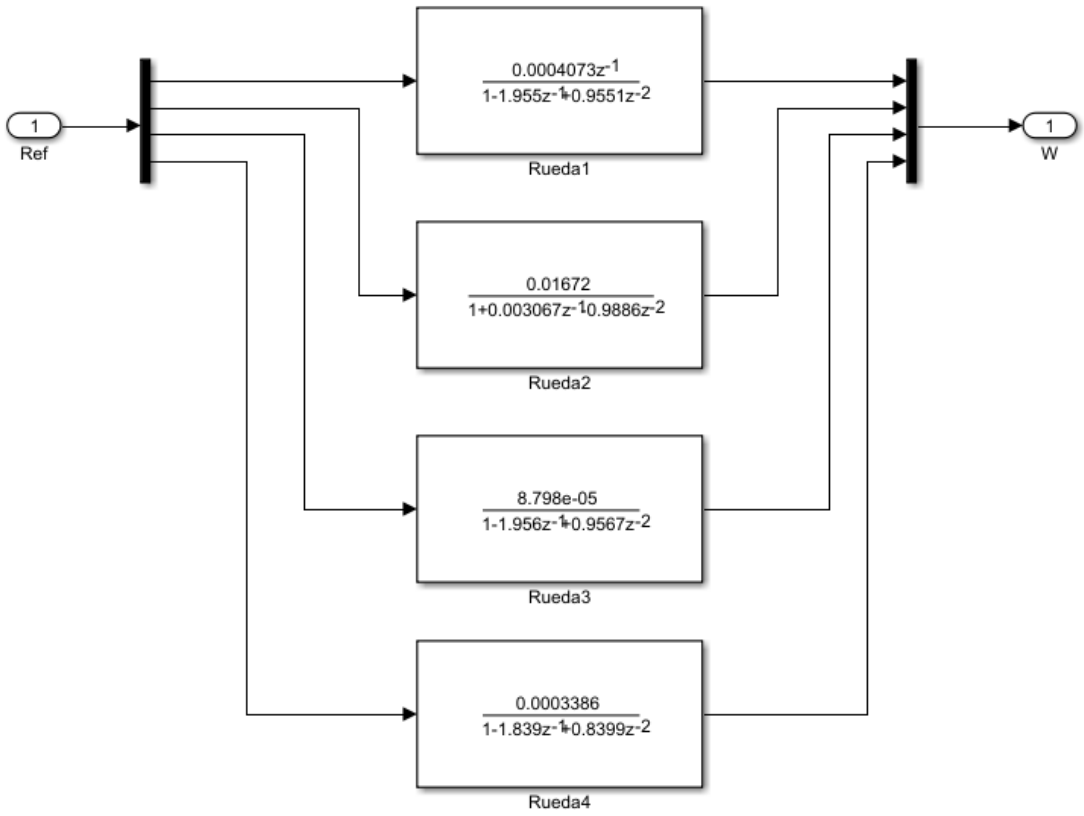

Figura 6.19: Funciones de transferencia Tierra – giro.

6.3.2. Estrategia de control

A pesar de la pobre aproximación conseguida, se procede a trazar las directrices a seguir en el control. Este caso resulta distinto a los demás, ya que se dan dos tipos de control distinto según el lado en el que se situé la rueda: En el lado interno al giro, se obtiene una ganancia proporcional mucho menor a la del lado opuesto, las ganancias integrales también resultan algo menores en el lado del giro, pero no tienen un cambio significativo.

Este esquema se debe a que las ruedas que giran más rápido aumentan la velocidad de las ruedas del lado contrario como ya se explicó anteriormente. En consecuencia, el control del lado interno debe frenarlas para mantener la velocidad adecuada, mientras que en el otro lado el control debe forzar el giro aumentando la velocidad.

7. CONCLUSIONES

En este capítulo se van a exponer los logros conseguidos a lo largo del proyecto, así como las conclusiones sacadas de los resultados obtenidos. Por último, se indican posibles líneas de continuar el trabajo aquí realizado.

7.1. Conclusiones.

El objetivo principal de este proyecto era desarrollar un control a baja velocidad para el robot móvil Rambler. Para ello se comenzó comparando los codificadores angulares instalados recientemente frente a los sensores hall propios de los motores en un banco de pruebas de características similares al vehículo. Tras probar el control con ambos sensores, queda demostrada la mejor respuesta del encoder, ya que los sensores tipo hall muestran picos de velocidad inexistentes al bloquearse la rueda, resultando inviables para velocidades bajas, especialmente en entornos con grandes perturbaciones.

Para el control del Rambler se utilizan los encoders con trasmisión por correa recientemente instalados, consiguiéndose un control aceptable con el vehículo sin contacto con el suelo, detallándose diferencias entre un sistema aislado como es el banco de pruebas con uno real y complejo como es el robot móvil Rambler.

Una vez conseguido el control en el aire, se repite el proceso con el vehículo en el pavimento pulido del taller, observándose el efecto de la inercia al comparar un cambio de sentido con y sin frenada. Debe recalcarse el efecto del peso del robot en la respuesta, obteniéndose controles muy distintos en ambos casos. El control desarrollado se adapta a las especificaciones requeridas, por lo que se consigue el objetivo del proyecto de desarrollar un control funcional para el robot móvil Rambler.

Por último, se han realizado pruebas en distintos terrenos, comprobándose el efecto del rozamiento entre otras perturbaciones en la identificación del sistema. Con estos datos se establecido unas directrices para desarrollar el control, resultando especialmente interesante el caso del giro, en el cual se establecen una estrategia de control específica para cada lado del vehículo.

Para llevar a cabo las experiencias comentadas, ha sido necesario adaptar la comunicación CAN para los 4 controladores del vehículo, debiéndose evitar interferencias con el resto de los nodos del bus mediante un filtrado de direcciones. Además, se incorpora un sistema de seguridad, para evitar perder el control del vehículo en caso de fallar la comunicación.

Como conclusión, el objetivo principal se ha completado con éxito, consiguiéndose un control optimo del vehículo a baja velocidad. A título personal, me ha resultado un proyecto muy interesante, ya que he tenido la oportunidad de trabajar sobre un robot real con componentes y tecnologías actuales, con los retos y aprendizaje que esto supone.

7.2. Líneas de desarrollo futuras

Una vez presentadas los resultados obtenidos, se proponen las siguientes líneas de desarrollo:

- Estudio y desarrollo del control multi-terreno. Necesario para un robot de apoyo en situaciones de emergencia, el cual estará expuesto a distintas situaciones y terrenos.
- Calibración de una unidad inercial y su posterior inclusión en el esquema de control, resultando en un control de orientación mucho más fiable y proporcionando información sobre el movimiento del vehículo.
- Adaptación de los esquemas para ser compatible con el software actual del Rambler, desarrollado en LabView. Este software se encarga de transmitir ciertos mensajes de configuración a los controladores, entrando en conflicto con el control desarrollado en Arduino.
- Desarrollar un control para la suspensión e incluirlo en el control de velocidad, ya que esta tiene un efecto crucial en el giro del robot, siendo necesario conocer el ángulo de cada brazo para maniobrar correctamente.

8. BIBLIOGRAFIA

[1] "Departamento de Ingeniería de Sistemas y Automática - Departamento de Ingeniería de Sistemas y Automática - Universidad de Málaga." [Online]. Available: https://www.uma.es/isa. [Accessed: 03-Sep-2018].

[2] "Arduino - Página principal." [Online]. Available: https://www.arduino.cc/. [Accessed: 03-Sep-2018].

[3] "Brushless DC Motor Controllers : HBL1660." [Online]. Available: https://www.roboteq.com/index.php/roboteq-products-and-services/brushless-dc-motor-controllers/256/hbl1660-detail. [Accessed: 03-Sep-2018].

[4] J. M. L. Bedmar, "Control de una rueda con motor sin escobillas y de accionamiento directo," 2016.

[5] D. R. Gómez, "Control cinemático del Rambler," 2017.

[6] E. Corral Gordillo, "Trabajo fin de grado: simulación del robot móvil rambler," 2017.

[7] Graynomad, "Due pinout diagram." [Online]. Available: https://forum.arduino.cc/index.php?topic=132130.0. [Accessed: 24-Jun-2018].

[8] "Seeed-Studio: CAN BUS Shield." [Online]. Available: https://github.com/Seeed-Studio/CAN_BUS_Shield. [Accessed: 10-Jun-2018].

[9] A. Muñoz Ramírez, J. Luque Bedmar, and J. Serón, "Actas XXXVIII Jornadas de Automática."

[10] Kubler, "Incremental encoder Datasheet."

[11] Roboteq, "HBL16xx Motor Controller Datasheet," pp. 1–14.

[12] "Advanced Brushed and Brushless Digital Motor Controllers User Manual."

[13] "Minimal encoder resolution for a low speed rotation - Roboteq Online Forum." [Online]. Available: https://www.roboteq.com/index.php/forum/14-general-issues/29529334-what-is-the-minimal-encoder-resolution-for-a-low-speed-rotation. [Accessed: 12-Aug-2018].

[14] T. Wang, Y. Wu, J. Liang, C. Han, J. Chen, and Q. Zhao, "Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor.," *Sensors (Basel).*, vol. 15, no. 5, pp. 9681–702, Apr. 2015.

9. ANEXOS

En este capítulo se recogen las funciones y scripts desarrolladas a lo largo del proyecto, incluyéndose comentarios tanto en el código como en una breve descripción para facilitar su compresión.

9.1. ReceiveNode2

Script de Matlab encargado de recibir y clasificar los mensajes del bus CAN del Rambler.

9.1.1. Código S-Funtion.

```
//Include Files

#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
# ifndef MATLAB_MEX_FILE

#include <Arduino.h>
#include <SPI.h>
#include <mcp_can.h>

// El pin de CS (SPI) del shield es el 9
const int SPI_CS_PIN = 9;
unsigned int NodoID1[4];    // Variable auxiliar para las direcciones

MCP_CAN CANN(SPI_CS_PIN);   // Creamos la variable CANN inicializando
el pin CS

typedef struct{
  long VAR1, VAR2,VAR3,VAR4;
}MiniCANr;

volatile MiniCANr MiniCANrecibir;
unsigned char buf1[8];

int decodificarCANBus( unsigned int canId ){
  int error=0;

  if (canId>=0x180 && canId<=0x180+NodoID1[3]){

        MiniCANrecibir.VAR1 = ((long )buf1[3]) << 24;
        MiniCANrecibir.VAR1 |= ((long )buf1[2]) << 16;
        MiniCANrecibir.VAR1 |= ((long )buf1[1]) << 8;
        MiniCANrecibir.VAR1 |= buf1[0];

        MiniCANrecibir.VAR2 = ((long )buf1[7]) << 24;
        MiniCANrecibir.VAR2 |= ((long )buf1[6]) << 16;
        MiniCANrecibir.VAR2 |= ((long )buf1[5]) << 8;
        MiniCANrecibir.VAR2 |= buf1[4];}
```

```
/*  else if (canId>=0x280 && canId<=0x280+NodoID1[3]){

        MiniCANrecibir.VAR3 = ((long )buf1[3]) << 24;
        MiniCANrecibir.VAR3 |= ((long )buf1[2]) << 16;
        MiniCANrecibir.VAR3 |= ((long )buf1[1]) << 8;
        MiniCANrecibir.VAR3 |= buf1[0];

        MiniCANrecibir.VAR4 = ((long )buf1[7]) << 24;
        MiniCANrecibir.VAR4 |= ((long )buf1[6]) << 16;
        MiniCANrecibir.VAR4 |= ((long )buf1[5]) << 8;
        MiniCANrecibir.VAR4 |= buf1[4];}*/

  else
  {
      error=-1;
  }

  return error;
}

# endif
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define y_width 1

// Output functions
extern "C" void ReceiveNode2_Outputs_wrapper(int32_T *VAR1,
            int32_T *VAR2,
            int32_T *VAR3,
            int32_T *VAR4,
            const real_T *xD,
            const uint16_T *ID1, const int_T p_width0)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
if (xD[0]==1) {
        /* don't do anything for mex file generation */
    # ifndef MATLAB_MEX_FILE


    unsigned char len = 0;
    int error;

    if(CAN_MSGAVAIL == CANN.checkReceive())            // check if
data coming
    {
        CANN.readMsgBuf(&len, buf1);    // read data,  len: data
length, buf: data buf

        unsigned int canId = CANN.getCanId();

        error=decodificarCANBus(canId);
        if (!error) {
            if(canId==0x180+NodoID1[0]){

                    VAR1[0]=MiniCANrecibir.VAR1;
                    VAR1[1]=MiniCANrecibir.VAR2;
```

```
            }

            else if(canId==0x180+NodoID1[1]){
                    VAR2[0]=MiniCANrecibir.VAR1;
                    VAR2[1]=MiniCANrecibir.VAR2;
            }

            else if(canId==0x180+NodoID1[2]){
                    VAR3[0]=MiniCANrecibir.VAR1;
                    VAR3[1]=MiniCANrecibir.VAR2;
            }

            else if(canId==0x180+NodoID1[3]){
                    VAR4[0]=MiniCANrecibir.VAR1;
                    VAR4[1]=MiniCANrecibir.VAR2;
            }

/*DESCOMENTAR EN CASO DE RECIBIR 4 VARIABLES (Ampliar el tamaño de las
salidas VAR si se usan)

            else if(canId==0x280+NodoID1[0]){
                    VAR1[2]=MiniCANrecibir.VAR3;
                    VAR1[3]=MiniCANrecibir.VAR4;
            }

            else if(canId==0x280+NodoID1[1]){
                    VAR2[2]=MiniCANrecibir.VAR3;
                    VAR2[3]=MiniCANrecibir.VAR4;
            }

            else if(canId==0x280+NodoID1[2]){
                    VAR3[2]=MiniCANrecibir.VAR3;
                    VAR3[3]=MiniCANrecibir.VAR4;
            }

             else if(canId==0x280+NodoID1[3]){
                    VAR4[2]=MiniCANrecibir.VAR3;
                    VAR4[3]=MiniCANrecibir.VAR4;
            }

 */

        }


    }
# endif

}
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

// Updates function
extern "C" void ReceiveNode2_Update_wrapper(int32_T *VAR1,
            int32_T *VAR2,
            int32_T *VAR3,
```

```
            int32_T *VAR4,
            real_T *xD,
            const uint16_T *ID1, const int_T p_width0)
{

//Inicializacion
if (xD[0]!=1) {

    /* don't do anything for MEX-file generation */
    # ifndef MATLAB_MEX_FILE

    //Las direcciones deben ir en orden ascendente, siendo ID1[3] la
de mayor valor número
    NodoID1[0]=ID1[0];
    NodoID1[1]=ID1[1];
    NodoID1[2]=ID1[2];
    NodoID1[3]=ID1[3];

    while (CAN_OK != CANN.begin(CAN_1000KBPS))              // init
can bus : baudrate = 500k
    {
        delay(100);
    }

    # endif

    /*La mascara decide que bits de los filtros hay que comparar (los
1, los 0 los obvia)
      Los filtros deciden que direcciones se leen, solo se reciben
aquellas que coincidan
      los valores del filtro con los bits puestos a 1 de la mascara*/

    /*Las mascaras y los filtros deben inicializarse en ambos codigos
(SendNode y ReceiveNode)*/

    CANN.init_Mask(0, 0, 0x7fc);
    CANN.init_Mask(1, 0, 0x7fc);

    /* Estos filtros permiten leer las VAR1 y VAR2*/
    CANN.init_Filt(0, 0, 0x180);
    CANN.init_Filt(1, 0, 0x180+NodoID1[3]);

    /* Estos filtros permiten leer las VAR3 y VAR4*/
    //CANN.init_Filt(2, 0, 0x280);
    //CANN.init_Filt(3, 0, 0x280+NodoID[3]);

    /* initialization done */
    xD[0]=1;

}
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

9.2. SendNode2

Script de Matlab encargado de codificar y transmitir los mensajes a los controladores Roboteq del Rambler a través del bus CAN.

9.2.1. Código S-Funtion.

```
/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
# ifndef MATLAB_MEX_FILE

#include <Arduino.h>
#include <SPI.h>
#include <SPI.cpp>
#include <mcp_can.h>
#include <mcp_can.cpp>


// El pin de CS (SPI) del shield es el 9
const int SPI_CS_PIN = 9;
unsigned int NodoID[4];     // Variable auxiliar para las direcciones

MCP_CAN CAN(SPI_CS_PIN);    // Creamos la variable CAN inicializando
el pin CS

typedef struct{

  long VAR9,VAR10;

}MiniCANt;
volatile MiniCANt MiniCANtrans;


unsigned char buf[8];

void codificarCANBus(){

     buf[0] = (byte) MiniCANtrans.VAR9;
     buf[1] = (byte) (MiniCANtrans.VAR9 >> 8);
     buf[2] = (byte) (MiniCANtrans.VAR9 >> 16);
     buf[3] = (byte) (MiniCANtrans.VAR9 >> 24);

     buf[4] = (byte) MiniCANtrans.VAR10;
     buf[5] = (byte) (MiniCANtrans.VAR10 >> 8);
     buf[6] = (byte) (MiniCANtrans.VAR10 >> 16);
     buf[7] = (byte) (MiniCANtrans.VAR10 >> 24);
```

```
}

# endif
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 4

// Output functions
extern "C" void SendNode2_Outputs_wrapper(const int32_T *VAR,
                const uint8_T *Cont,
                const real_T *xD,
                const uint16_T *ID, const int_T p_width0)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
if (xD[0]==1) {
        /* don't do anything for mex file generation */
    # ifndef MATLAB_MEX_FILE

    MiniCANtrans.VAR10=Cont[0]; //Guardamos el contador de seguridad.
for(int i = 0; i < 4; i = ++i)
{
    MiniCANtrans.VAR9=VAR[i];   //Guardamos las consignas del motor

    codificarCANBus();          //Codificamos los datos en la variable
"buf"

    CAN.sendMsgBuf(0x200+NodoID[i], 0, 8, buf); //Mandamos el mensaje
a cada dirección
}

# endif

}
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

// Updates function

extern "C" void SendNode2_Update_wrapper(const int32_T *VAR,
                const uint8_T *Cont,
                real_T *xD,
                const uint16_T *ID, const int_T p_width0)
{

//Inicializacion
if (xD[0]!=1) {

    /* don't do anything for MEX-file generation */
    # ifndef MATLAB_MEX_FILE

    //Las direcciones deben ir en orden ascendente, siendo ID[3] la de
mayor valor numérico
    NodoID[0]=ID[0];
    NodoID[1]=ID[1];
    NodoID[2]=ID[2];
    NodoID[3]=ID[3];
```

```
    while (CAN_OK != CAN.begin(CAN_1000KBPS))              // init can
bus : baudrate = 500k
    {
        delay(100);
    }

          # endif

    /*La mascara decide que bits de los filtros hay que comparar (los
1, los 0 los obvia)
      Los filtros deciden que direcciones se leen, solo se reciben
aquellas que coincidan
      los valores del filtro con los bits puestos a 1 de la mascara*/

    /*Las mascaras y los filtros deben inicializarse en ambos codigos
(SendNode y ReceiveNode)*/

    CAN.init_Mask(0, 0, 0x7fc);
    CAN.init_Mask(1, 0, 0x7fc);

    /* Estos filtros permiten leer las VAR1 y VAR2*/
    CAN.init_Filt(0, 0, 0x180);
    CAN.init_Filt(1, 0, 0x180+NodoID[3]);

    /* Estos filtros permiten leer las VAR3 y VAR4*/
    //CAN.init_Filt(2, 0, 0x280);
    //CAN.init_Filt(3, 0, 0x280+NodoID[3]);

    /* initialization done */
    xD[0]=1;
}
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

9.3. Script controladores Roboteq

Script de los controladores, encargado de recibir y transmitir los datos de los motores. Incluye un sistema de seguridad en caso de fallar la comunicación.

9.3.1. Código Roborun+.

```
'CREACION DE VARIABLES
'Indica la velocidad leída del motor
Dim motor_speed As integer
'Indica la velocidad leída del encoder
Dim enc_speed As integer
'Indica el nivel de potencia leído del motor
Dim power_level As integer
'Indica el comando a enviar al motor
Dim motor_command As integer
'Canal en el que se encuentra el motor
Dim channel As Integer
'Variables auxiliares para comprobar la comunicación
Dim comunicacion As Integer
Dim comunicacionant As Integer
Dim ciclosretraso As Integer
```

```
channel=1

top: 'Etiqueta de inicio del script

'Coloca el temporizador a 0 ms
      SetTimerCount(1,0)

'Lee el comando motor, en la VAR9.
      motor_command=getvalue(_VAR,9)

'------------------- Comprobación comunicación --------------------
'Guarda valor anterior del contador para poder compararlo con el
nuevo valor.
      comunicacionant = comunicacion

'Leemos el contador proveniente de arduino, en la VAR10
      comunicacion = getvalue(_VAR, 10)

'Si el valor del contador actual es mayor que el anterior, la
comunicación es correcta.
      if (comunicacion > comunicacionant) then
            ciclosretraso = 0

'Si por el contrario el valor es igual, la comunicación se ha
detenido.
      else
            ciclosretraso++ 'Aumentamos el contador de seguridad

'Si el problema persiste, asumimos un fallo en la trasmisión de
datos.
            if (ciclosretraso > 4) then
                  motor_command = 0      'anulamos el comando motor,
deteniendo las ruedas.
            end if
      end if
'---------------------- Fin de comprobación -----------------------

'Coloca en el motor la referencia
      setcommand(_G,channel,motor_command)

'Lee las velocidades del motor en rpm
      'motor_speed=getvalue(_BS,channel)      'Sensor Hall
       enc_speed=getvalue(_S,1)               'Encoder

'Coloca en el motor en VAR1 la velocidad del sensor Hall y en VAR2
la del encoder
      'setcommand(_VAR,1,motor_speed)         'Sensor Hall
       setcommand(_VAR,2,enc_speed)           'Encoder

'Lee la potencia del motor y la guarda en power_level
      power_level=getvalue(_P,channel)

'Envia el comando power_level por VAR1
      setcommand(_VAR,1,power_level)

'Espera de 10 ms
```

```
    wait(10-GetTimerCount(1))
goto top
```

9.4. Adaptación datos – Matlab

Función de Matlab encargada de adaptar los datos del *workspace* al trabajar con el Rambler. Proporciona un *Data object* con la velocidad y referencia de cada rueda, preparado para ser utilizado en la herramienta *System Identification*..

9.4.1. Código.

```
%Accede a la información de las señales de Simulink
logsout

%Guarda esa información en variables del workspace
Vel1=logsout.getElement(1).Values.Data;
Vel2=logsout.getElement(2).Values.Data;
Vel3=logsout.getElement(3).Values.Data;
Vel4=logsout.getElement(4).Values.Data;
Ref=logsout.getElement(5).Values.Data;

%Transforma las variables para que sean compatibles con la función
iddata
Ref1=Ref(1,:);
Vel11=Vel1(1,:);
Vel11=Vel11';

Vel22=Vel2(1,:);
Vel22=Vel22';

Vel33=Vel3(1,:);
Vel33=Vel33';

Vel44=Vel4(1,:);
Vel44=Vel44';

%Crea un objeto con el comando del motor como entrada y la
%velocidad como salida, con un tiempo de muestreo de 10 ms.
Rueda1 = iddata(Vel11,Ref,0.01);
Rueda2 = iddata(Vel22,Ref,0.01);
Rueda3 = iddata(Vel33,Ref,0.01);
Rueda4 = iddata(Vel44,Ref,0.01);
```