Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

CONSTRUCCIÓN DE ROBOT MÓVIL BASADO EN MICROCONTROLADOR Y PLATAFORMA DE SIMULACIÓN

Alumno: Andrés Liébana Cazalla

Tutores: Prof. D. Pablo Cano Marchal
Prof. D. Diego Martínez Gila
Dpto: Ingeniería Electrónica y Automática

Octubre, 2018

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Ingeniería Electrónica y Automática

Don Pablo Cano Marchal y Don Diego Martínez Gila, tutores del Proyecto Fin de Carrera titulado: Construcción de un robot móvil basado en microcontrolador y plataforma de simulación, que presenta Andrés Liébana Cazalla, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Junio de 2018

El alumno: Los tutores:



Andrés Liébana Cazalla Pablo Cano Marchal Diego Martínez Gila

Índice

1. INTRODUCCIÓN

Este trabajo de fin de grado (TFG) consiste en la construcción de un robot móvil basado en microcontrolador, el modelado cinemático del mismo y su correspondiente plataforma de simulación. A lo largo del proyecto se irán comentando los requisitos, tanto de software como hardware, el diseño del robot, así como las herramientas y entorno utilizados para el desarrollo de este.

Es necesario aclarar que este robot servirá para el TFG de nuestro compañero Rubén Maestro Huesa, quien se encargará de la estrategia de control de este, con lo cual habrá partes en las que se mencionen las cooperaciones emprendidas para ciertas tomas de decisiones.

1.1. Motivación

Este proyecto resulta interesante ya que al ser un robot basado en el microcontrolador Arduino (una plataforma de código libre y de bajo coste), nos va a permitir ver el abanico de posibilidades que la electrónica de bajo coste puede ofrecernos. Por ejemplo, para la docencia los alumnos podrán ver el funcionamiento de un robot real y como aplicar la teoría que se ve en clase, además de la posibilidad de añadir nuevos sensores y probar nuevas estrategias de control, todo ello por un coste muy bajo.

Por otro lado, también podemos hablar de la plataforma de simulación, para la cual emplearemos una herramienta software, EJSS, que tiene un gran potencial para probar todo lo que pensemos antes de ponerlo en marcha en el robot.

Además, estas simulaciones son el primer paso para una posible implementación en plataformas virtuales/remotas en un futuro. Son herramientas con un gran potencial docente, que nos permiten visualizar el funcionamiento de distintos sistemas físicos.

1.2. Objetivos

Como hemos mencionado anteriormente, el robot servirá para varios TFG, con lo cual nuestro primer objetivo es construirlo en base a un diseño que permita hacer modificaciones a nuestro compañero (y cualquiera que lo use) de la forma más sencilla posible, facilitando la futura incorporación de otros componentes (también a nivel de cableado).

En segundo lugar, después del montaje, se deberá hacer una identificación experimental del sistema, es decir, se realizarán diversos experimentos y pruebas en las que se recogerán datos que proporcionen la información suficiente para describir matemáticamente cómo se comporta el sistema ante una entrada dada.

Por último, se implementará el modelo teórico de dicho robot en una plataforma de simulación que permita representar y prever el comportamiento del mismo.

1.3. Metodología

El primer punto a tener en cuenta es la elección del robot. Se nos ha propuesto, a mí y a mi compañero, un catálogo de robots de bajo coste con distintas características, estructuras y sensores. Juntos, plantearemos las distintas posibilidades e intentaremos llegar al robot más favorable para ambos.

El siguiente punto que tratar sería la identificación del sistema. Para ello hemos de aprender cuál es el modelo teórico y cómo funciona el sistema, de tal forma que podamos identificar las variables que nos proporcionarán la información necesaria. Para este paso será necesario implementar algunos sensores y hacer uso de ellos, con lo cual se harán colaboraciones con Rubén Maestro para esta tarea.

Una vez terminado ese trabajo toca paso a la simulación, para la cual tendremos que empezar un aprendizaje del software que vamos a utilizar: “Easy JavaScript Simulations” (EJSS) y Matlab (junto a su toolbox: Simulink).

Todas estas tareas, como se comentará más adelante, irán acompañadas de los correspondientes estudios teóricos y de la autodidáctica necesaria para procurar que todo funcione lo mejor posible.

1.4. Contenido de la memoria

A continuación, exponemos un pequeño resumen de los contenidos que tendrá este TFG:

- **Capítulo 1: Introducción**. Se hablará en términos generales de que trata este proyecto, cuáles son los objetivos a conseguir, la metodología a seguir y la motivación que lo impulsa.

- **Capítulo 2: Comparativa entre robots de bajo coste y elección.** Se contrastarán todos los robots de un catálogo y se seleccionará el más adecuado en cuanto a diseño y componentes.

- **Capítulo 3: Montaje y puesta en marcha del robot.** Se hablará un poco del hardware que disponemos y el software a utilizar. Después se expondrá el procedimiento seguido en el montaje, con sus correspondientes fotografías, comentando los problemas y el diseño final adoptado.

- **Capítulo 4: Fundamentos de modelado de sistemas.** Aquí se hará un breve recordatorio teórico de modelado de sistemas, introduciendo así como se procederá para obtener el modelo del robot.

- **Capítulo 5: Modelado cinemático de un robot móvil.** Partiendo de la cinemática del robot, conseguiremos su modelo matemático, obteniendo las variables de utilidad y su modelo de espacio de estados.

- **Capítulo 6: Ensayos de identificación y obtención de modelos.** Después de saber la información que se requiere se harán experimentos con los que
  trataremos de obtener dichos datos, con el objetivo de conseguir el modelo matemático experimental.

- **Capítulo 7: Simulación del modelo en EJSS y Simulink.** En esta parte de hará uso del software Easy JavaScript Simulations para introducir el modelo de nuestro robot y poder simular su comportamiento.

- **Capítulo 8: Índice de figuras.**

- **Capítulo 9: Índice de cuadros de código.**

- **Capítulo 10: Bibliografía y recursos web.**

2. COMPARATIVA ENTRE ROBOTS DE BAJO COSTE Y ELECCIÓN

Como ya hemos mencionado anteriormente, partíamos de un pequeño catálogo predispuesto que incluía los elementos mecánicos y electrónicos con los que tendríamos que construir el robot, aunque abierto a sugerencias de robots con prestaciones parecidas. Comenzaremos mencionando cada robot, ilustrándolo y comentando los componentes de los que disponía.

2.1. Robot de dos ruedas con control de balanceo

Figura 1: Robot con control de balanceo

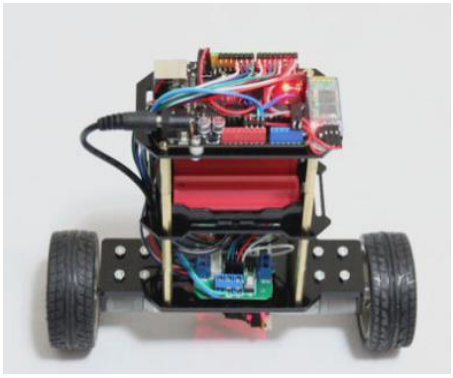


Figura 2: Robot con control de balanceo 2

Este kit robótico incorpora todos los elementos mecánicos necesarios para su montaje, incluyendo: tableros acrílicos, tachones de cobre, tuercas y tornillos, los ejes, las ruedas y sus correspondientes motores.

En cuanto a los módulos electrónicos, trae consigo un modelo de Arduino UNO, un controlador para los motores de corriente continua (modelo L298N), un convertidor DC/DC (modelo LM2596), un acelerómetro (MPU6050) y un par de baterías recargables de 2600mAh con su correspondiente soporte.

Otro dato interesante es que se puede añadir un módulo de Bluetooth para control remoto por una aplicación de Android.

Más adelante discutiremos las ventajas y desventajas de este robot frente al resto, llegando a la elección y final y su por qué.

2.2. Robot de dos ruedas. Chasis. SODIAL

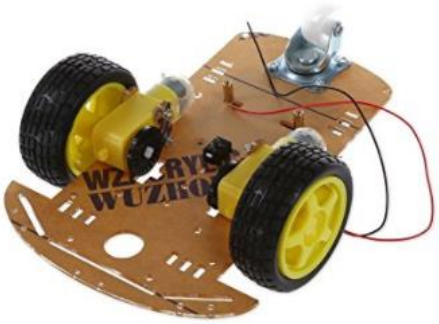


Figura 3: Robot chasis 2

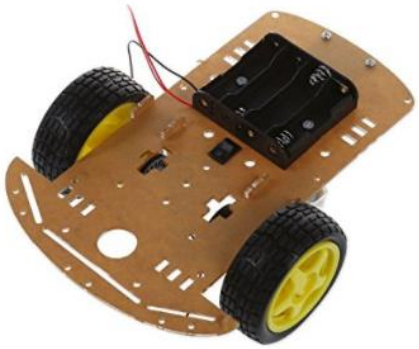


Figura 4: Robot chasis 1

Este robot tan solo incorpora los materiales mecánicos, los necesarios para su montaje: chasis, dos motores de corriente continua, dos ruedas, dos encoder para controlar la velocidad de los motores, una rueda loca o universal y la caja de la batería.

La ventaja que presentaba era su bajísimo coste, y la desventaja que había comprar aparte los componentes electrónicos.

2.3. Robot de dos ruedas. Chasis. Robot elegante

Figura 5: Robot elegante 2

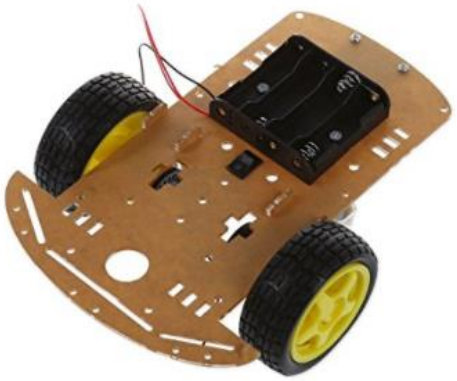


Figura 6: Robot elegante 1

Este se trata del mismo kit que el anterior, con la diferencia de que este si incorpora algunos módulos electrónicos, así que vamos a mencionar cuales son:

Este robot trae consigo un modelo de Arduino UNO R3 con una placa "Shield V5" que añade más pines para facilitar y ampliar el número de conexiones que podamos hacer al Arduino. Además, trae consigo otro motor de corriente continua junto con un servomotor (SG90) y un controlador para los motores (L298N). También trae consigo una cámara emisora y receptora de video.

2.4. Robot tortuga de dos ruedas

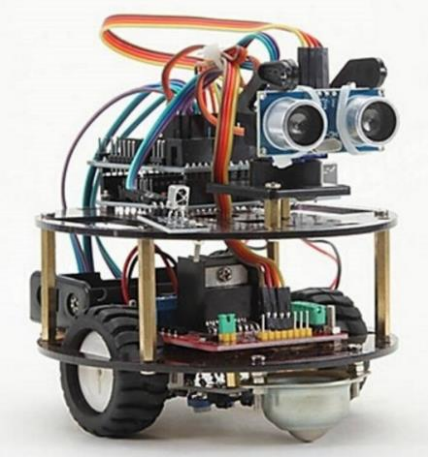


Figura 7: Robot tortuga 1

Figura 8: Robot tortuga 2

Empezando por los elementos mecánicos cabe mencionar los dos chasis, los separadores de placas, los neumáticos de las ruedas, dos motores, dos ruedas locas, el soporte para la batería junto con un cargador, un mando de control remoto y toda la tornillería necesaria.

Por otro lado, incorpora muchos elementos electrónicos: el driver L298N para controlar los motores de corriente continua, un Arduino UNO R3 junto con la placa “Shield V5”, un módulo Bluetooth HC-06, un servomotor SG90, un sensor de ultrasonidos HC-SR04, un sensor de infrarrojos y un módulo de rastreo TCRT5000.

2.5. Comparativa y elección

Empezando por el primer modelo: el robot con control de balanceo, nos parecía llamativo. La construcción sería sencilla y un control de balanceo podría ser interesante como estrategia de control. La principal desventaja era que no incorporaba más sensores con los que poder trabajar, habría que comprarlos a parte y además no teníamos la certeza de poder incorporarlos adecuadamente debido a su estructura.

En cuanto al segundo y tercer modelo, son el mismo solo que uno incorporaba módulos de electrónica y el otro no. Un montaje sencillo que se reducía a un chasis y dos ruedas, y además la incorporación de sensores se podría hacer sobre el chasis.

Por último, el robot tortuga de dos ruedas. Sin duda el más completo que vimos de bajo coste. Además de incorporar la mayoría de los sensores de los anteriores, añadía unos pocos más. Con este robot teníamos más posibilidades: más espacio donde colocar sensores y la facilidad de añadir una placa board para el conexionado en la parte superior del mismo. Por otro lado, incorporaba muchos sensores que podrían servir para futuras aplicaciones: módulo de rastreo para seguir líneas, modulo bluetooht para comunicaciones, ultrasónico para esquivar obstáculos, etc.

Por todos estos motivos, el robot tortuga fue nuestra elección.

3. MONTAJE Y PUESTA EN MARCHA DEL ROBOT

En este capítulo hablaré sobre todo lo necesario para poner en marcha el robot: empezando por sus componentes hardware, el software necesario para la identificación y para su simulación, y terminando con el desarrollo, paso a paso, de su montaje.

3.1. Hardware

En este apartado empezaremos mostrando y comentando todos los sensores y demás dispositivos hardware que incorpora el robot tortuga.

3.1.1. Arduino UNO R3

Arduino es una plataforma cuyo objetivo es hacer que la electrónica llegue a todo el mundo. Se basa en una fuente de código abierto y un hardware y software fáciles de usar, además de ser muy barato. Existen muchos modelos de placas de Arduino, en nuestro caso utilizaremos el Arduino UNO R3:

Figura 9: Arduino UNO R3

3.1.2. Driver L298N

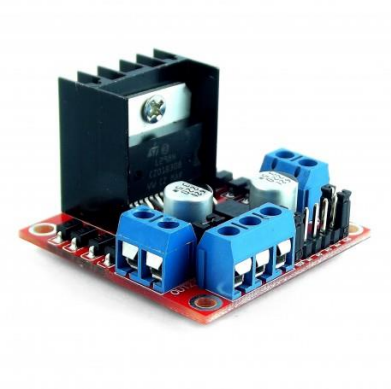


Figura 10: Driver L298N

Este driver nos permitirá controlar la velocidad de nuestros dos motores DC desde la plataforma Arduino. Añade la ventaja que permite la modulación por ancho de pulso (PWM) para el control de la velocidad.

3.1.3. Servomotor SG90

Este servo (motor que podemos controlar por velocidad o posición) nos será útil para girar el sensor de ultrasonidos, siendo así capaces de detectar obstáculos en distintos ángulos. Da la facilidad que funciona con la misma alimentación de la propia placa de Arduino.

Figura 11: Servomotor SG90

3.1.4. Sensor ultrasonidos HC-SR04

Figura 12: Sensor ultrasonidos HC-SR04

Con este sensor podremos calcular la distancia en centímetros en línea recta hasta una pared u obstáculo. Pudiendo medir esta proximidad actuaremos de una forma u otra según los objetivos.

3.1.5. Módulo de rastreo TCRT5000

Se trata de un módulo de rastreo que es capaz de detectar la intensidad luminosa reflejada, gracias a los sensores infrarrojos que incorpora.

Esto lo hace útil para el seguimiento de líneas (negras o blancas sería lo ideal).

Dispone de una salida digital y otra analógica y de un potenciómetro que nos permite ajustar su sensibilidad.

El rango de detección es bastante pequeño (hasta 1 cm), con lo cual deberá estar en la parte inferior del robot orientado hacia el suelo.

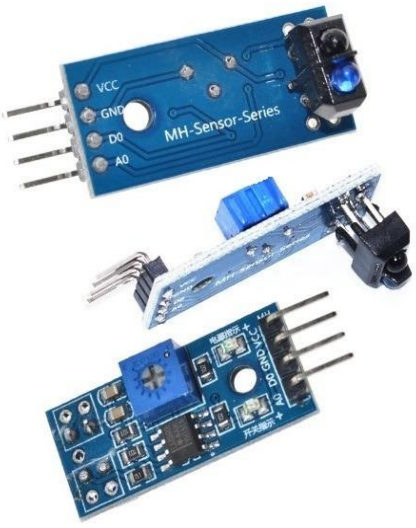
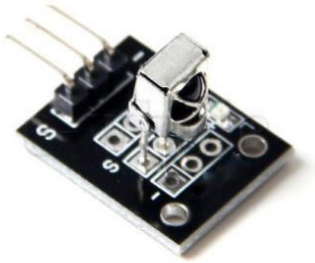


Figura 13: Módulo de rastreo

3.1.6. Sensor de infrarrojos



Se trata de un receptor de infrarrojos que podremos utilizar para captar las señales de un mando a distancia.

Figura 14: Sensor infrarrojos

3.1.7. Módulo Bluetooth HC-06

Este sensor Bluetooth nos servirá para comunicarnos con Matlab en el envío y recibo de datos.

Trabaja como un puerto serie en Arduino, así que podremos mandar la información necesaria para que haga una trayectoria, por ejemplo, y recibir todos los datos de los sensores con los que conseguiremos la identificación del sistema.

Figura 15: Sensor Bluetooth

3.1.8. Sensor inercial BNO-055

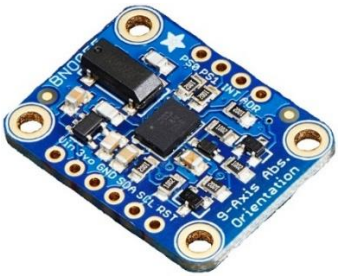


Figura 16: Sensor inercial BNO055

Con este sensor inercial podemos obtener una gran cantidad de datos, entre otros:

- Datos de orientación en los tres ejes
- Datos de aceleración en los tres ejes
- Velocidad angular
- Detector de campo magnético en los tres ejes

En nuestro caso haremos uso principalmente de la orientación en el eje X, ya que como veremos más adelante en el capítulo de la dinámica del robot es clave. Es posible que también nos sea útil conocer la aceleración en los ejes X e Y, así como la velocidad angular. No haremos uso del resto de datos que podemos obtener de este sensor.

3.2. Software

En este apartado vamos a mencionar todas las plataformas software que vamos a utilizar, sus lenguajes de programación y demás elementos de interés para este proyecto.

3.2.1. Arduino

Como hemos mencionado antes, Arduino es una plataforma de código libre cuyo fin es hacer llegar la electrónica a todos los públicos. A nivel de hardware existen distintas placas, cada una con sus particularidades, pero todas ellas se programan en el mismo lenguaje, que recibe el mismo nombre que la plataforma: Arduino.

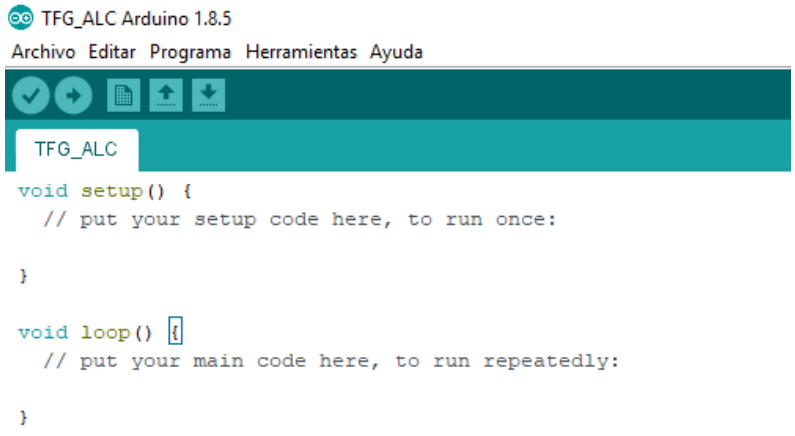


Figura 17: Compilador Arduino

Aunque el propio lenguaje se llame Arduino, está basado en el lenguaje C++, de hecho, podemos usar muchos de los comandos de C++ en el propio Arduino.

Como podemos observar en la figura 17 hay dos apartados donde poder programar:

Setup	**Loop**
En esta parte se programará la configuración inicial, es decir, si queremos que en un inicio tome ciertas referencias o las variables tomen ciertos valores, aquí deberemos ponerlo.	Este código se estará ejecutando continuamente en un ciclo infinito. Aquí ira toda la programación de los sensores: tanto el de ultrasonidos, como el driver de los motores, como el Bluetooth para hacer las comunicaciones.

Algo importante que debemos tener en cuenta es que para obtener la información con la que identificaremos el sistema deberemos lograr que el programa siempre tarde en ejecutarse el mismo tiempo, es decir, conseguir que el código se ejecute y envíe la información en un tiempo de muestreo determinado. En el capítulo de identificación hablaremos de esto.

3.2.2. Matlab

Podemos decir que Matlab es un entorno preparado para ingenieros de muy distintos campos. Usa un lenguaje propio mucho más sencillo que otros como C o C++. La peculiaridad de Matlab es que permite hacer operaciones con vectores y matrices que otros lenguajes no son capaces de hacer.

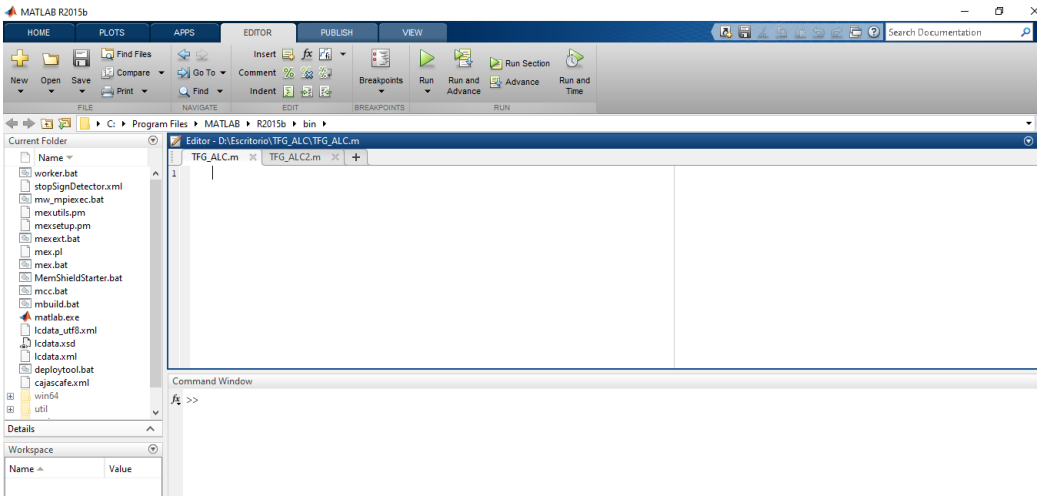


Figura 18: Matlab

Podemos distinguir entre distintas secciones dentro de Matlab, a continuación, iremos explicando lo más básico:

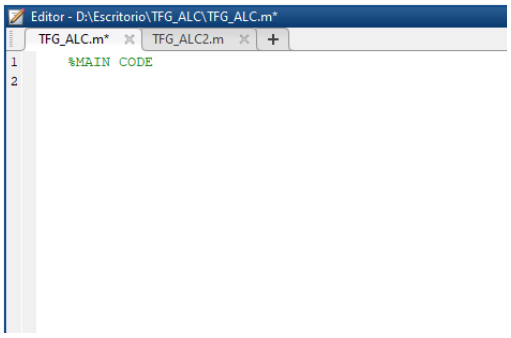


Figura 19: Editor de Matlab

<u>Editor</u>

Es el lugar donde escribimos el código de nuestro programa principal.

Se pueden hacer llamadas a funciones.

Funciones

Aquí programamos las acciones repetitivas u operaciones que tengamos que hacer con distintos datos, en vez de usar el mismo código varias veces en el programa principal, usamos una función.

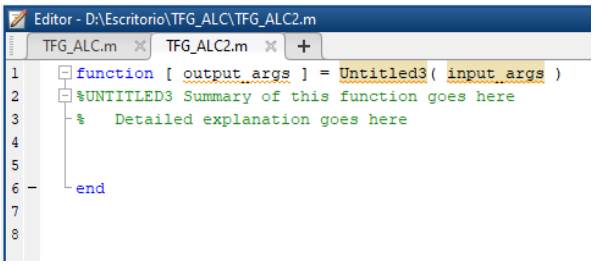


Figura 20: Funciones de Matlab

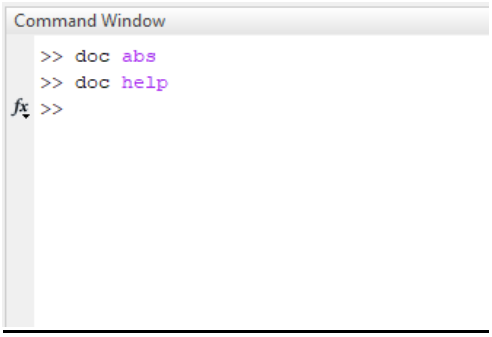


Figura 21: Ventana de comandos de Matlab

Ventana de comandos

Como bien indica su nombre es una ventana donde podemos probar los distintos comandos de Matlab, ya sea para comprobar su funcionamiento o información de la documentación.

Toolbox de Matlab

Otra herramienta muy útil para ingenieros es el gran catálogo de aplicaciones del que dispone.

Muchas de ellas se agrupan en Toolbox que sirven para propósitos en concreto, por ejemplo: la de visión por computador y la de diseño y análisis de sistemas de control.

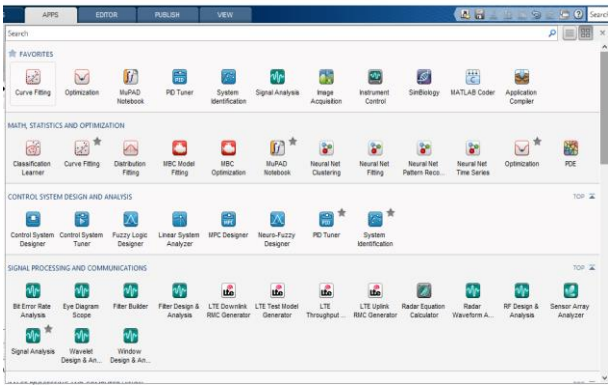


Figura 22: Toolbox de Matlab

3.2.2.1. Conexión con Arduino para la obtención de datos

La conexión en sí será sencilla, tan solo deberemos ejecutar dos comandos: uno será para crear el objeto Bluetooth y el otro para iniciar la conexión, pero antes deberemos configurar el módulo Bluetooth. Lo haremos gracias a una Toolbox de Matlab: “Instrument Control”.

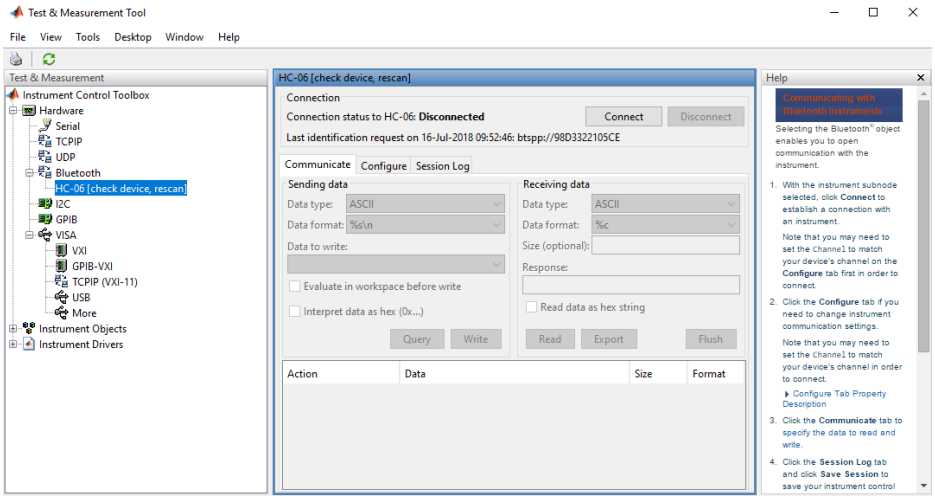


Figura 23: Toolbox de Bluetooth

En esta Toolbox tenemos la posibilidad de escanear en busca de módulos Bluetooth, configurarlos (nombre, ID, contraseña) y guardarlos para poder conectarnos posteriormente. Una vez creado el objeto tan solo nos hacen falta dos comandos para poder conectarnos con Arduino y recibir la información como si de un puerto serie se tratase:

Command Window

$\gg s = Bluetooth('HC-06', 1);$

$\gg fopen(s);$

Una vez hecho esto, nuestro módulo estará conectado y podremos recibir (o mandar) la información entre Arduino y Matlab.

3.2.2.2. System Identification Toolbox

Esta aplicación es la que nos va a permitir hacer la identificación del sistema. Pasándole como entradas el valor de PWM (que nos señala indirectamente la tensión que le proporcionamos) y como salida la posición angular podremos estimar varios modelos, de los cuales nos quedaremos con el que menos porcentaje de error tenga.

En este apartado tan solo voy a comentar el funcionamiento de la aplicación, más adelante, en el capítulo de identificación, hablaré y mostraré nuestros datos y caso en concreto.

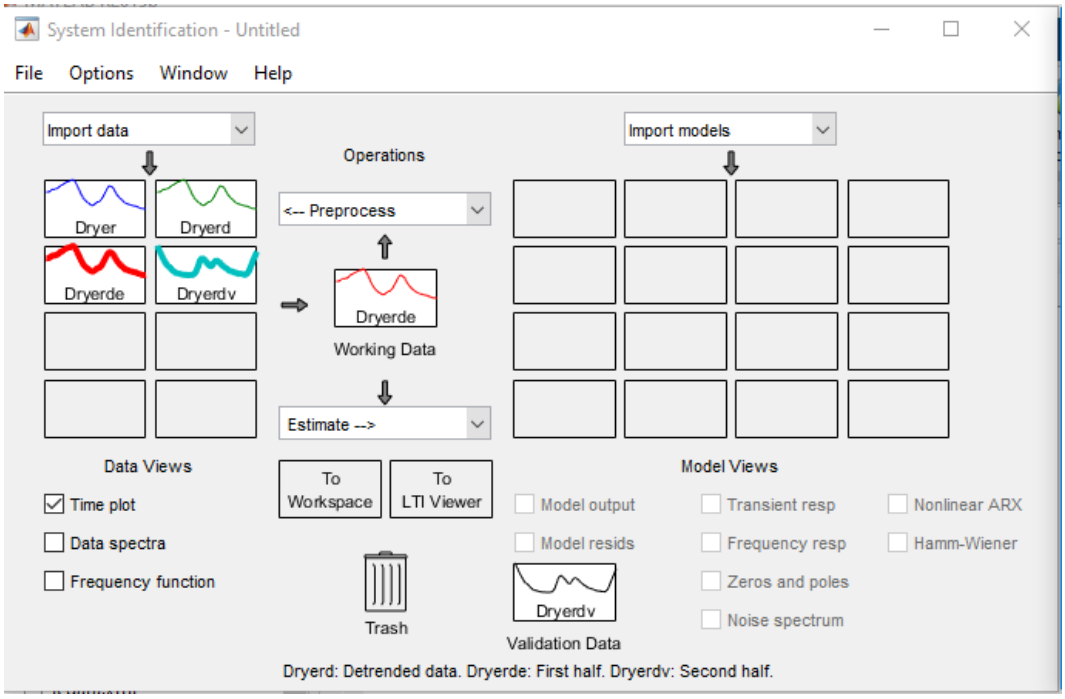


Figura 24: Toolbox System Identification

Si nos fijamos, arriba a la izquierda aparece una pestaña desplegable desde la que podemos importar datos o modelos, en este caso he cargado un ejemplo de temperatura y potencia.

El siguiente paso a seguir sería el preprocesado: utilizamos ciertas operaciones para crear nuevos conjuntos de datos basados en los importados. Podemos filtrar o escoger un rango, por ejemplo. Automáticamente esta aplicación coge, aproximadamente, la mitad de las muestras para hacer la identificación y la otra mitad para comprobar si el sistema estimado es capaz de evaluar correctamente y obtener el menor error posible. A continuación, se muestra en la siguiente figura:

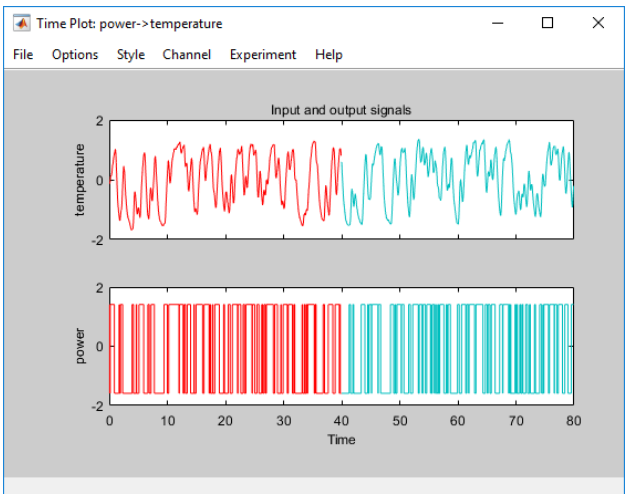


Figura 25: Muestras para la identificación

Después habría que ir a la pestaña de estimación, donde podemos elegir entre distintos métodos, entre los cuales se encuentran: modelo de función de transferencia, modelo de espacio de estados, modelo no lineal, etc. Para un mismo conjunto de datos podemos hacer varias estimaciones, para poder compararlas entre sí y elegir la que mejor defina el comportamiento del sistema. A continuación, muestro varias gráficas del ejemplo de temperatura-potencia:

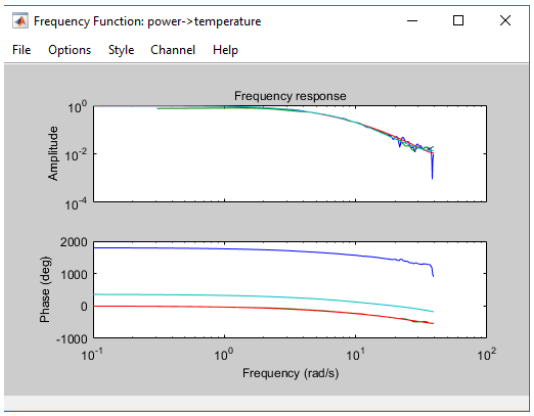


Figura 26: Respuesta en dominio de la frecuencia

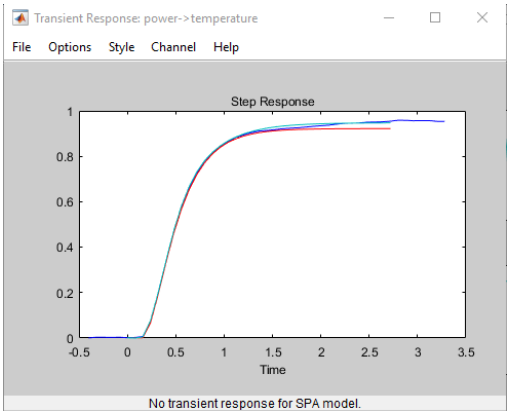


Figura 27: Respuesta en dominio temporal

Para terminar, tan solo nos quedaría exportar los modelos obtenidos tras la identificación al espacio de trabajo de Matlab y operar con ellos.

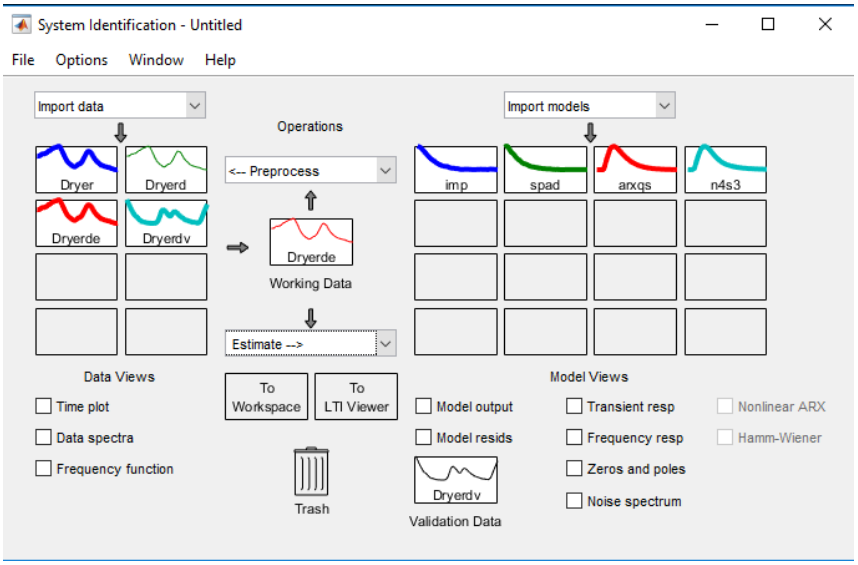


Figura 28: Modelos obtenidos de la identificación

3.2.3. Easy JavaScript Simulations (EJSS)

EJSS es una herramienta software creada por un profesor de la Universidad de Murcia, formaba parte de un proyecto: *Open Source Physics.* La finalidad de esta herramienta es la de ser capaces de crear simulaciones de distintos procesos físicos.

El potencial de esta herramienta es enorme para el uso en docencia: con el simple dibujo de un muelle y un par de ecuaciones de energía, los alumnos podrían ver el verdadero comportamiento de distintos sistemas físicos y dejarlos fuera de la imaginación. En el caso de ingeniería, más concretamente electrónica, podríamos observar distintos procesos de control desde una simulación de EJSS.

Otra gran ventaja de este software es que, gracias a que se puede programar en Java Script, podemos alojar la simulación en un servidor, de tal forma que se podría ejecutar desde nuestra propia casa, viéndola cuantas veces necesitáramos y cambiando los valores de las variables para ver el nuevo comportamiento del sistema.

3.2.3.1. Estructura de EJSS

A continuación, vamos a ver la estructura que tiene este software, describiendo cada una de las partes con más detalle en posteriores apartados.

3.2.3.1.1. Consola de EJSS

Digamos que la consola de EJSS es el punto de entrada, desde aquí se ejecuta el programa y se pueden añadir otras opciones como el idioma, aspecto gráfico, lenguaje de programación, etc.

Otra cosa interesante de la consola es que dispone de un área de mensajes en la cual podemos ver los errores de la simulación creada por EJSS, con lo cual, al abrir una simulación, podremos saber de dónde provienen los fallos.

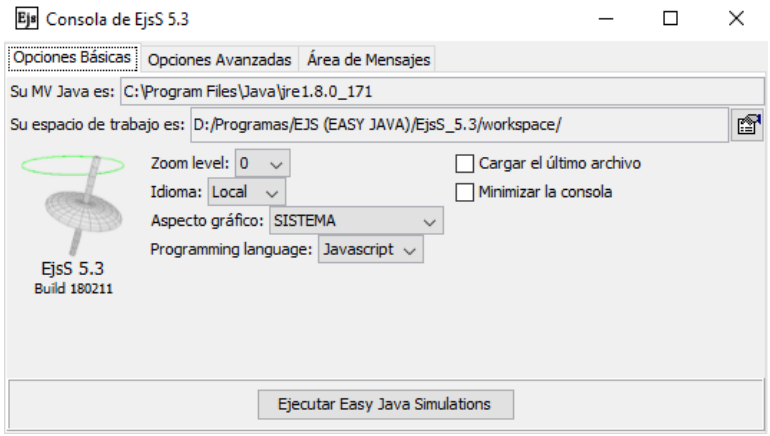


Figura 29: Consola de EJSS

3.2.3.1.2. Interfaz de EJSS

La interfaz dispone de tres pestañas principales: descripción (donde podemos redactar una página que describa el proceso físico que simulamos y las ecuaciones que lo rigen), modelo (aquí iría todo lo concerniente a la programación del mismo: sus ecuaciones diferenciales, la inicialización, las variables) y la vista (donde crearíamos los dibujos, gráficos o cualquier cosa que fuera necesaria para su visualización). A continuación, expondremos ejemplos de cada una.

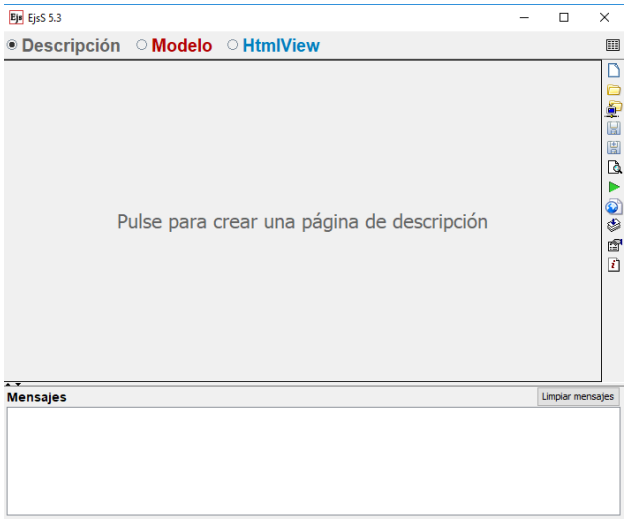


Figura 30: Interfaz EJSS

Descripción

Como he mencionado anteriormente, son páginas donde podemos describir el sistema físico que vamos a simular. En la siguiente figura, podemos ver un ejemplo cargado de EJSS donde muestra un sistema con una masa y un muelle.

También hay que decir que se pueden añadir varias páginas de descripción. Dentro de la docencia puede ser útil para añadir ejercicios o teoría que ayuden a entender el sistema.

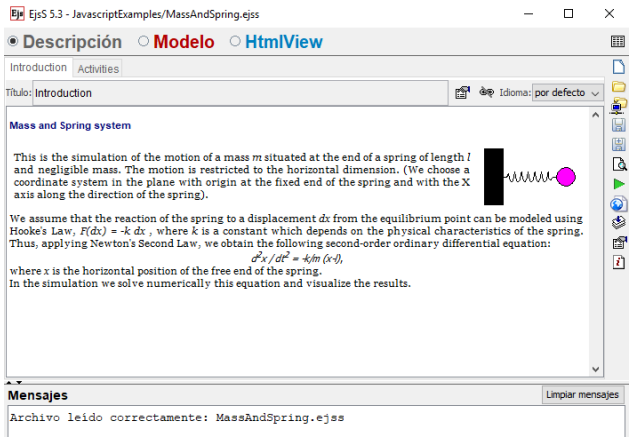


Figura 31: Descripción

Modelo

De esta página hay que comentar varias cosas, porque a su vez se divide en otras muchas pestañas, empecemos por la de variables:

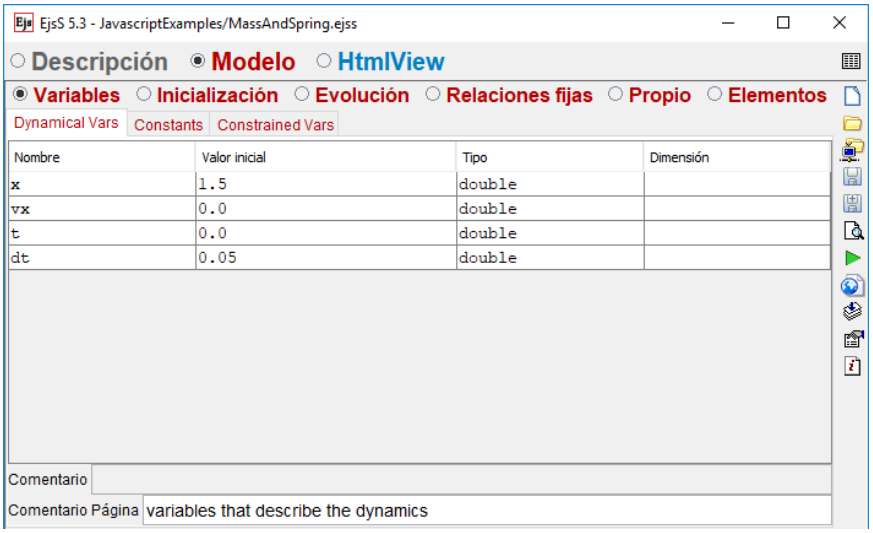


Figura 32: Modelo: variables

Como podemos ver, en esta página crearemos todas las variables que vayamos a utilizar, indicando de que tipo son (entero, con decimal, un carácter o cadena de caracteres, booleana) e incluso inicializarlas con el valor que queramos. Además, se pueden crear varias páginas por si queremos tener las variables más organizadas.

La siguiente pestaña es la de inicialización, la cual parece que no tiene mucho sentido teniendo en cuenta que se pueden inicializar en la propia pestaña de variables. Se pueden crear más de una página para tener separadas las variables según nos convenga.

Figura 33: Modelo: inicialización

La siguiente pestaña trata sobre la evolución del sistema, aquí tenemos dos opciones: programación o el uso de las ecuaciones diferenciales que rigen el sistema.

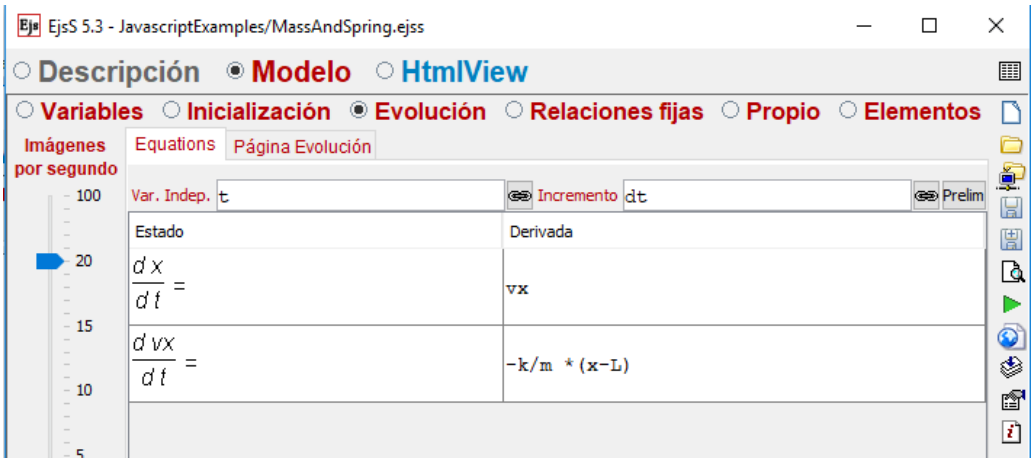


Figura 34: Evolución: ecuaciones diferenciales

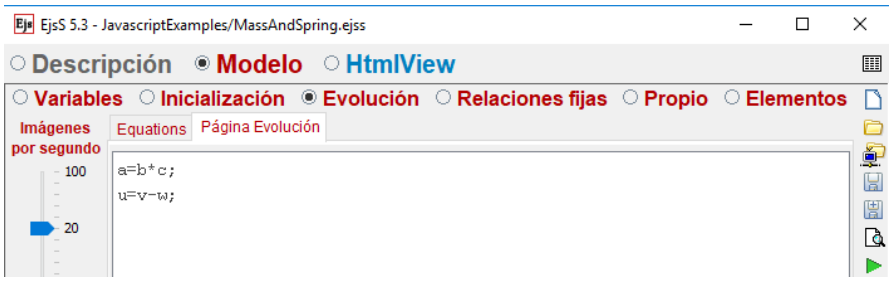


Figura 35: Evolución: código programado

El siguiente del que hablar se trata de las relaciones fijas, que como bien indica su nombre será una página donde podremos indicar las relaciones que tienen unas variables con otras.

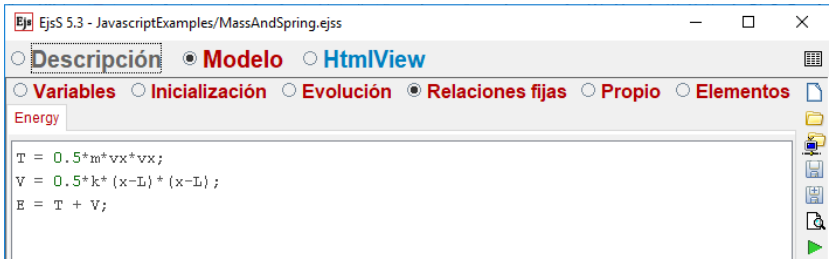


Figura 36: Relaciones fijas

El último punto del que hablaremos de esta parte de la herramienta es “Propio”, donde podremos crear páginas con funciones propias a las que podremos llamar tanto en relaciones fijas como en cualquier página de evolución.

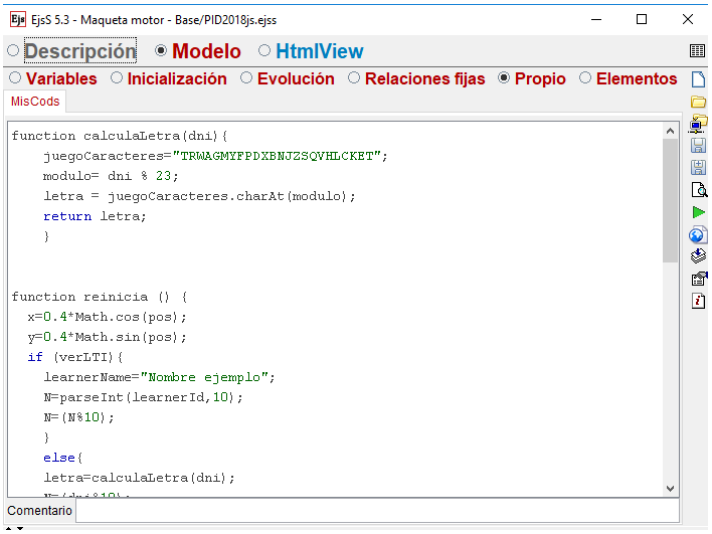


Figura 37: Propio

Vista-HTML View

En esta última pestaña de EJSS es donde podremos crear la interfaz para nuestra simulación y dibujar en 2D o 3D el sistema físico. Podemos hacer que todos los elementos evolucionen con una variable, que se muestren o no dependiendo del punto de la simulación, podemos poner cuadros de texto o campos numéricos donde poder cambiar los valores de ciertas variables, etc. Un gran repertorio de elementos que nos permiten una infinidad de posibilidades.

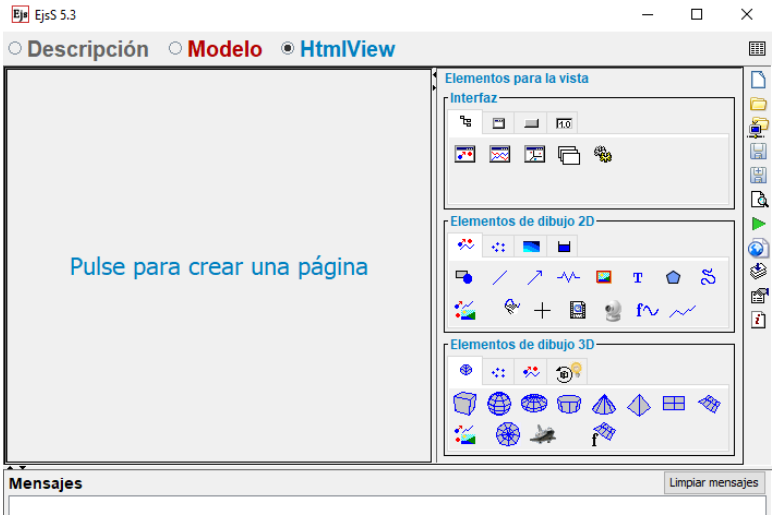


Figura 38: Vista de EJSS

A continuación, muestro un ejemplo de otro trabajo con las variables editadas de dos elementos distintos:

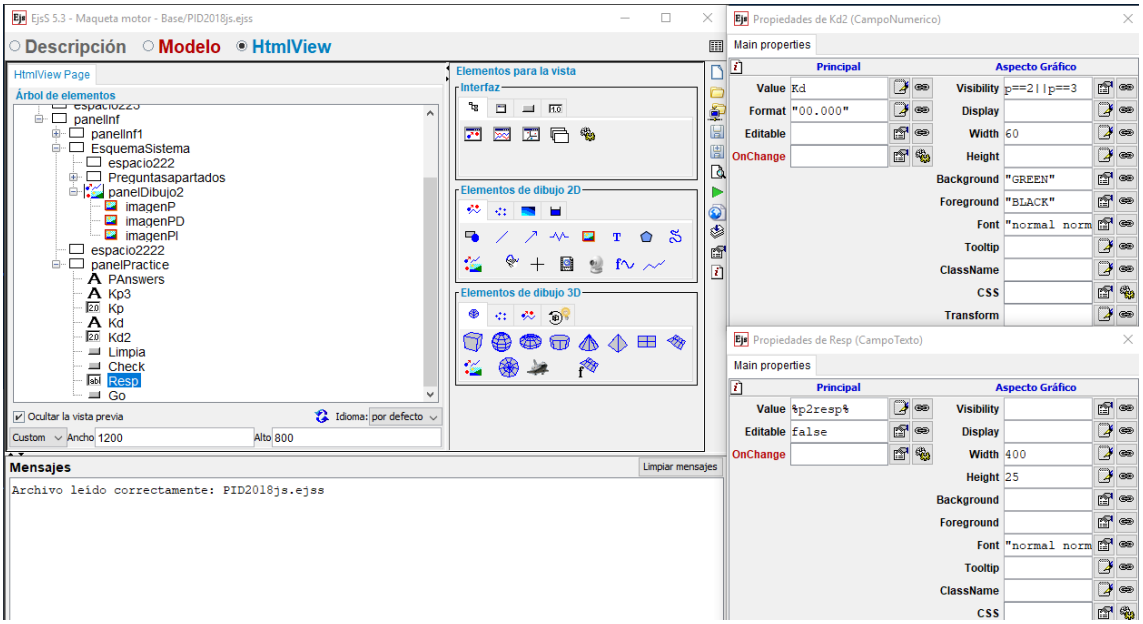


Figura 39: Vista

3.3. Montaje mecánico

En esta sección vamos a mostrar paso a paso el montaje del robot, los problemas que fuimos encontrando y todos los cambios que hicimos para llegar a la solución final.

3.3.1. Montaje siguiendo manual

Una vez comprado el robot le pedimos al fabricante el manual para montarlo. Por desgracia era muy escueto y tan solo venían imágenes, cada una con un par de piezas más montadas que el anterior. Hemos de añadir que tuvimos que improvisar en varias ocasiones debido a la falta de información del manual.

Aun así, fue suficiente para conseguir un buen montaje. Comenzamos mostrando todos los materiales de los que disponía el kit robótico:

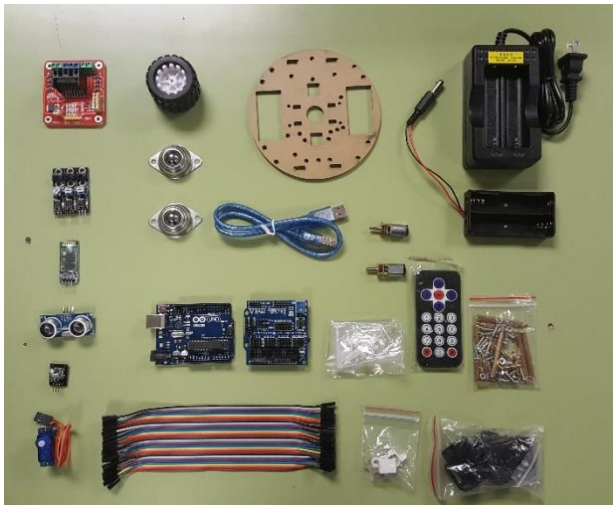


Figura 40: Piezas del kit robótico

El primer paso será soldar los cables de los motores y acoplarlos a ambas ruedas, como mostramos en las siguientes figuras:

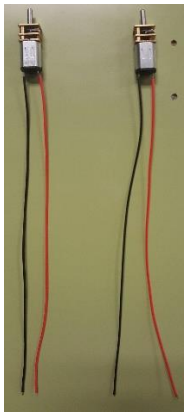


Figura 41: Montaje paso 1

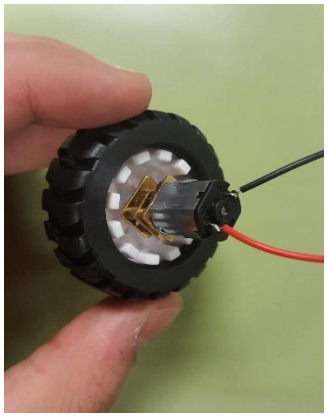


Figura 42: Montaje paso 2

Lo siguiente que vamos a hacer es unir una pieza a cada motor que es la que nos va a permitir atornillar los motores a la base del robot:

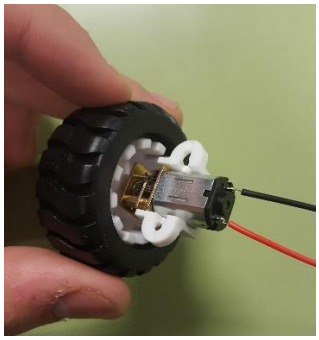


Figura 43: Montaje paso 3

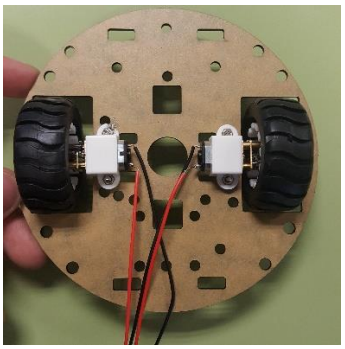


Figura 44: Montaje paso 4

Una vez hecho esto nos faltaría añadir las dos ruedas locas para tener la base del robot completamente estable, así que el siguiente paso a seguir sería atornillar ambas al robot. Como las ruedas locas y las ruedas de goma tienen tamaños distintos, ajustaremos todas las ruedas a la misma altura añadiendo o quitando tuercas a las ruedas locas, como mostramos en las siguientes figuras:

Figura 45: Montaje paso 5

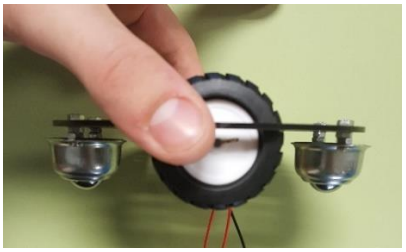


Figura 46: Montaje paso 6

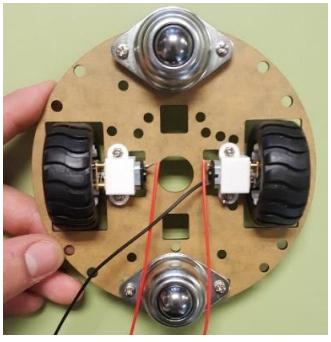


Figura 47: Montaje paso 7

En la figura 38 mostramos como quedaría finalmente la base del robot con las cuatro ruedas incorporadas. Aunque posteriormente se añadirán más elementos a la parte inferior y se tendrán que hacer modificaciones sobre las ruedas locas para ajustar la altura de todas en conjunto.

Antes de seguir montando otros componentes, es conveniente incorporar el módulo de rastreo en la cara inferior de la base ya que hay que atornillarlo y más adelante hay que añadir otros componentes que dificultarían su instalación. Así que las dos siguientes figuras van a mostrar el módulo de rastreo, así como su incorporación en la parte inferior del robot:

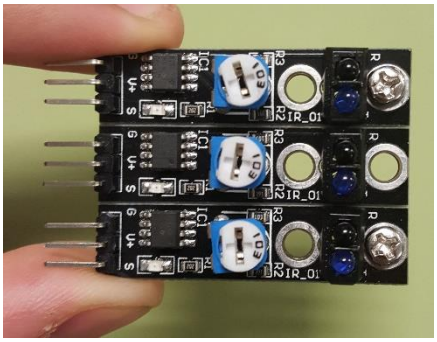


Figura 48: Montaje paso

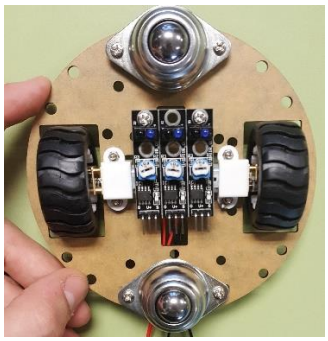


Figura 49: Montaje paso 9

El siguiente paso que vamos a seguir es la instalación del driver de los motores y el correspondiente conexionado con los mismos. Hay que aclarar que en un principio no incorporamos ninguna puntera ni ningún otro terminal al cable, queríamos hacer las pruebas necesarias para comprobar su funcionamiento en un principio. Así que no se descarta que el montaje final las incorpore si es que fuese necesario. A continuación mostramos las dos figuras que representan estos pasos: el montaje del driver en la cara superior de la base del robot, y el conexionado de los motores con este.

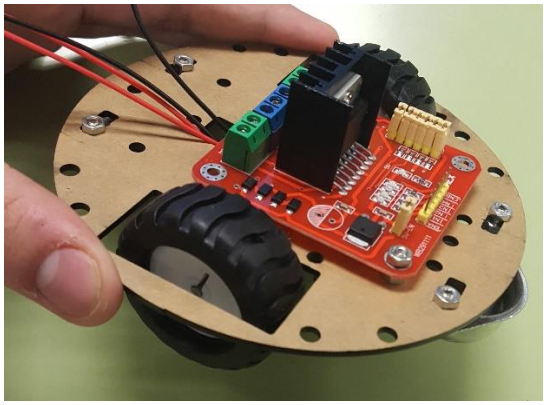


Figura 50: Montaje paso 10

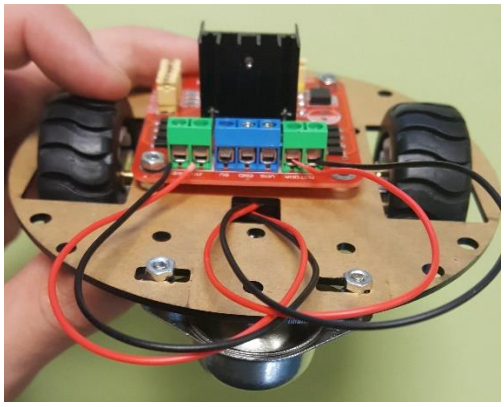


Figura 51: Montaje paso 11

A continuación, procedemos al montaje del compartimento de las baterias (que vienen a ser dos pilas que proporcionan una tensión de hasta 6V, más que suficiente para alimentar los motores y Arduino). Lo primero que tuvimos que hacer es soldar unos cables extra, porque el conector servía para Arduino, pero además teníamos que alimentar los motores. Luego el hecho de atornillarlo fue complicado, porque según el manual iba cerca de una de las ruedas locas, y físicamente había ciertas complicaciones a la hora de enroscar las tuercas y apretarlas. Muestro unas figuras ilustrando estos pasos.

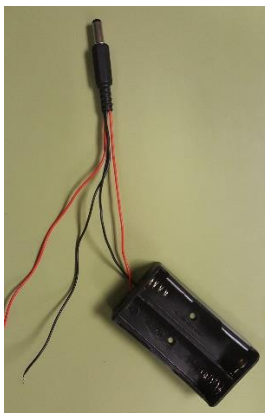


Figura 52: Montaje paso 12

Figura 53: Montaje paso 13

Figura 54: Montaje paso 14

Una vez instalado el compartimento de baterias, hemos terminado (a priori) con todos los elementos que van montados entre la base inferior y la superior, así que es hora de atornillar los separadores de placa y montar la base superior para proseguir con el resto de elementos, como mostramos en la siguiente figura:

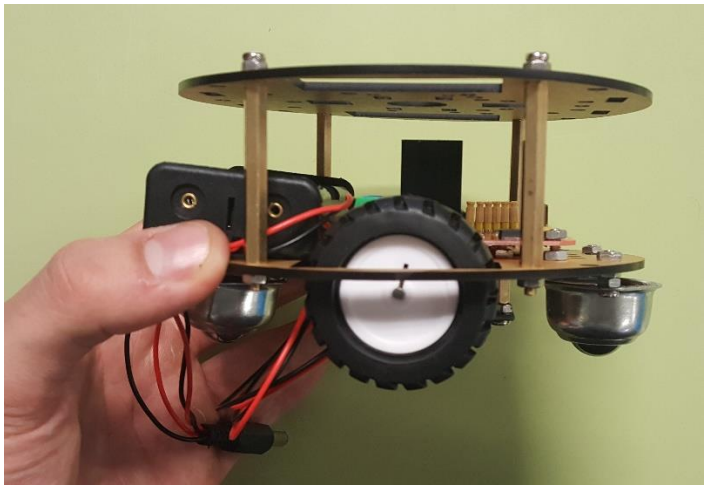


Figura 55: Montaje paso 15

El siguiente paso que vamos a seguir es montar el sensor de ultrasonidos. Como dijimos en el apartado de hardware, disponemos de un servomotor que servirá para girar este sensor para poder ver en distintas direcciones la distancia a un obstaculo o pared. Para hacer un ensamblaje del servo y del sensor de ultrasonidos, disponemos de una serie de piezas pequeñas de plástico. Vamos a mostrar directamente el resultado final, ya que tan solo se trata de encajar una serie de piezas alrededor del servo y unir (en nuestro caso lo hemos hecho con bridas) el sensor de ultrasonidos en la parte superior. Después atornillaremos el ensamble a la base superior del robot.

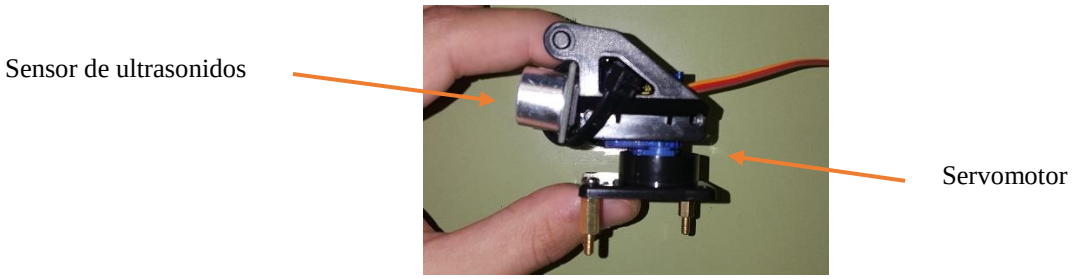


Figura 56: Montaje paso 16

A continuación, mostramos varias imágenes donde esta pieza quedaría ensamblada en nuestro robot:

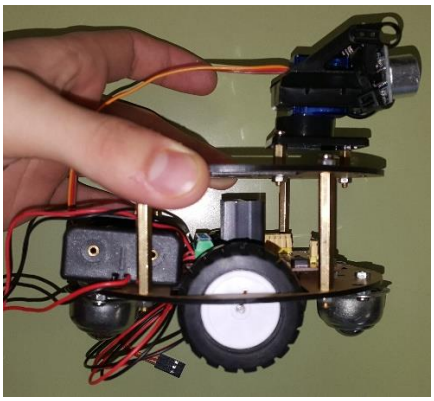


Figura 57: Montaje paso 17

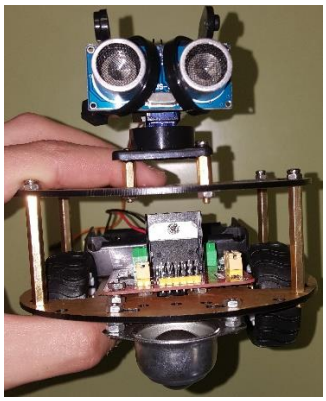


Figura 58: Montaje paso 18

Ya como último paso de esta primera etapa del montaje, quedaría atornillar la placa de Arduino utilizando los mismos separadores con los que instalamos la pieza que incluía el sensor de ultrasonidos y el servo. Una vez puesto el Arduino ya tendríamos los elementos indispensables para que el robot pudiera moverse, programando todo lo concerniente al driver de los motores, e incluso se parase ante un obstáculo si quisiéramos usar el sensor de ultrasonidos.

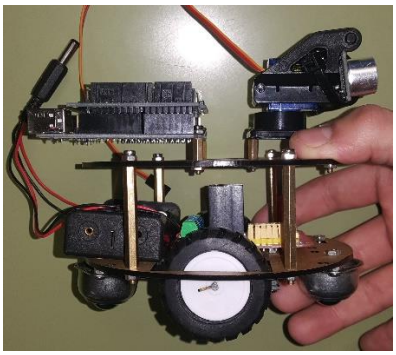


Figura 59: Montaje paso 19

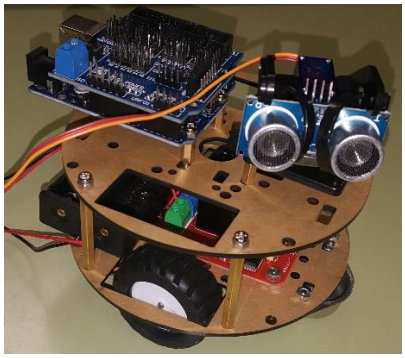


Figura 60: Montaje paso 20

Para la identificación, sin embargo, necesitamos saber el comportamiento teórico de nuestro robot, para obtener las variables esenciales con las que poder hacerlas. Una vez estudiado el modelo teórico (que explicaremos en los siguientes capítulos), ya sabíamos qué variables eran necesarias para la identificación, además de otros sensores que debíamos instalar. Por todo esto y otros inconvenientes que fuimos encontrando, hubo que plantear alternativas y cambiar varias cosas del robot.

3.3.2. Problemas encontrados

Como hemos mencionado antes, fuimos encontrando varios inconvenientes que tuvimos que ir solucionando, en este apartado los trataremos uno por uno, incluyendo la solución final por la que optamos.

Baterías

El primer problema que tuvimos es que el kit robótico no traía las pilas para el mismo, así que teníamos que comprar unas o usar otro tipo de batería. Por circunstancias de disponibilidad en el laboratorio en el que trabajamos, tuvimos que usar otra batería de las mismas características, así que el primer paso fue desmontar el compartimento de las baterías.

Esto tuvo un punto a favor, y es que como mencionamos en el montaje, la instalación de este compartimento era físicamente complicado y molestaba a la rueda loca de ese lado.

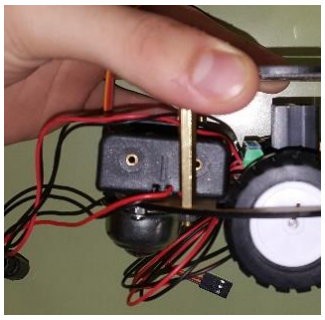


Figura 61: Problemas de montaje 1

Aunque no se ve con claridad, el compartimento no está del todo apoyado, hay cierta inclinación, no solo del propio compartimento, si no de la rueda loca también. Con lo cual era un problema.

Después de eso tuvimos que diseñar una pequeña caja en SolidWorks (un programa de diseño) e imprimirla en 3D. Después de este cambio había que buscar otro lugar para colocar la batería, ya que no cabía en el sitio donde estaba el compartimento, así que se le dio más altura (con un par de separadores pequeños) a la base superior del robot y se colocó la nueva batería (dentro de su caja) en la cara inferior de dicha base. A continuación, mostramos el resultado:

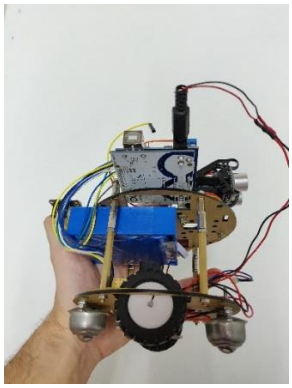


Figura 62: Batería y compartimento imprimido en 3D

Placa board y Arduino

Una vez estudiado el modelo teórico del robot, nos percatamos de que había que añadir otros sensores para obtener la información necesaria para su identificación, y posteriormente para el control que haría mi compañero Rubén Maestro.

No había muchos más sitios donde colocar sensores, además de que no todos los que necesitábamos se podían acoplar con un tornillo, a diferencia de los que habíamos montado hasta ahora. Es por eso por lo que en primer lugar pensamos cambiar el Arduino de posición a un lateral, dejando más espacio para incorporar otros sensores nuevos. Pero el problema principal seguía permaneciendo: no todos los elementos se podían atornillar.

Entonces se nos ocurrió, aprovechando el nuevo espacio que habíamos dejado libre al mover Arduino, colocar una placa board, de forma que todos esos sensores que no podíamos instalar antes, ahora sí que podíamos. Con la ventaja de que podíamos hacer más conexiones y organizar mejor el cableado, además de la posibilidad de añadir nuevos sensores/elementos en un futuro.

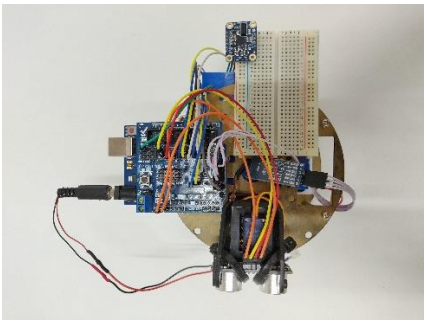


Figura 63: Vista final superior

3.3.3. Solución final

Bueno, tras haber hecho los cambios pertinentes y haber comprobado el correcto funcionamiento de todos los elementos, este es el resultado final del montaje:

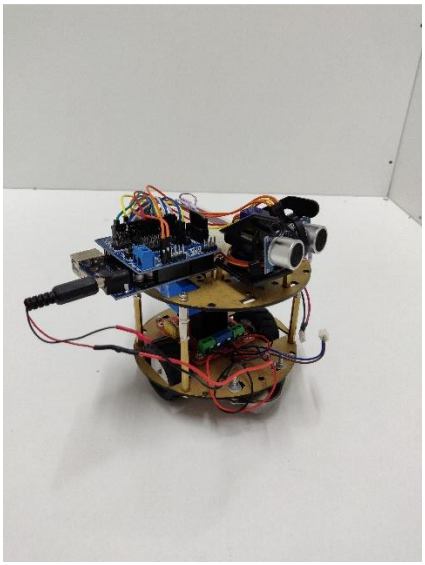


Figura 64: Montaje final

3.4. Montaje eléctrico

A continuación, vamos a mostrar el conexionado eléctrico de cada uno de los componentes que han sido necesarios para la identificación del sistema. Hemos de aclarar que no todas las conexiones se corresponderán con las que mostramos, e incluso puede que en los esquemas usemos una salida para varios elementos. Lo único que queremos reflejar es el tipo de salida que necesita cada sensor (salida PWM, salida normal o en el caso del Bluetooth las RX y TX).

3.4.1. Driver L298N

Empezaremos mostrando la conexión del controlador con los motores, Arduino y la alimentación de la batería:

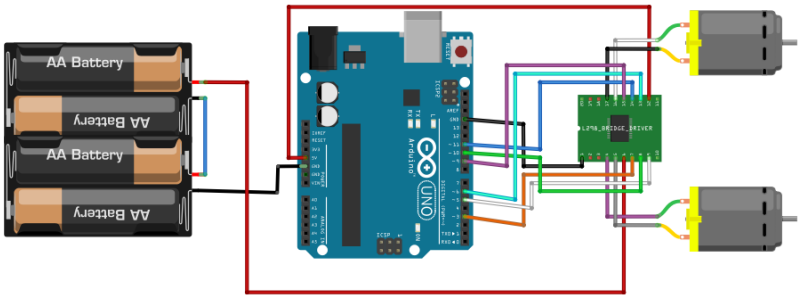


Figura 65: Conexionado del controlador L298N

En esta figura no se aprecian bien las conexiones debido al gran número de componentes, a continuación, mostramos otra figura más simplificada tan solo con el controlador de los motores y comentaremos posteriormente para que sirven cada uno de los pines:

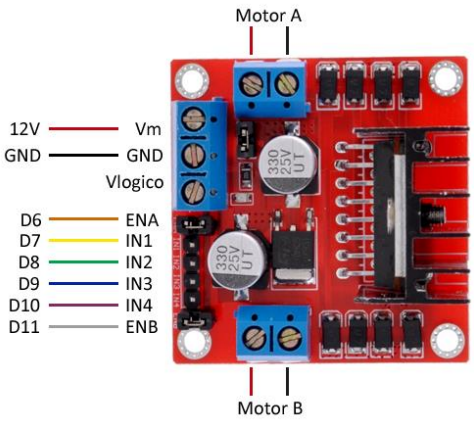


Figura 66: Pinout del controlador L298N

El controlador dispone de seis pines (sin tener en cuenta los de la alimentación): ENA, IN1, IN2, IN3, IN4 y ENB.

- ENA y ENB son los pines de “Enable”, o sea los pines que habilitan que se activen los motores A y B respectivamente. Estos pines son los que admiten modulación por ancho de pulso (PWM), con lo cual estos dos deberemos conectarlos a salidas digitales de Arduino que dispongan de PWM.

- El resto sirven para hacer que los motores giren en un sentido, otro o se queden parados. Los pines que controlan el primer motor junto con ENA son IN1 e IN2, y los que controlan el segundo motor junto con ENB son IN3 e IN4.

3.4.2. BNO-055

A continuación, adjunto dos figuras que muestran el conexionado del BNO-055 y de su esquemático correspondiente:

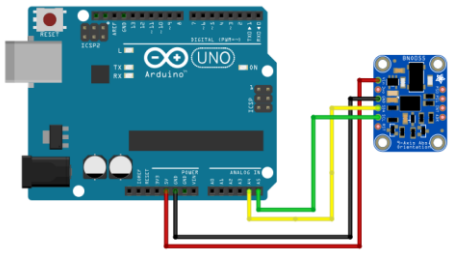


Figura 67: Conexionado BNO-055

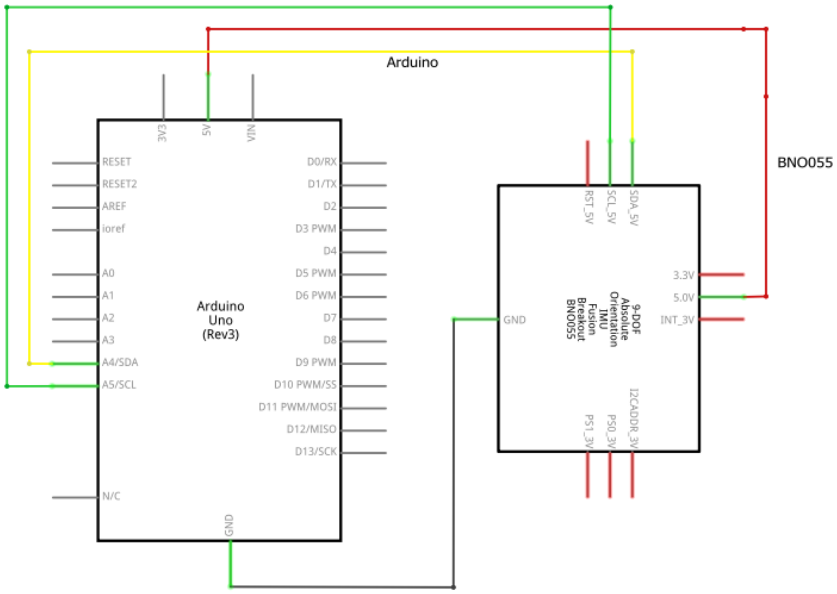


Figura 68: Circuito esquemático BNO-055

En este caso tan solo tenemos los dos pines de alimentación y los pines SDA y SCL que los conectaremos a las entradas analógicas de nuestro Arduino para recibir la información. El pin SCL es un reloj I2C y el pin SDA es el encargado de enviar la información. El resto de los pines sirven para reiniciar el sensor, para cambiar el modo de uso, cambiar la dirección predefinida de I2C, etc.

3.4.3. Bluetooth HC-06

Ahora mostramos las conexiones del módulo Bluetooth con Arduino, y posteriormente su circuito esquemático:

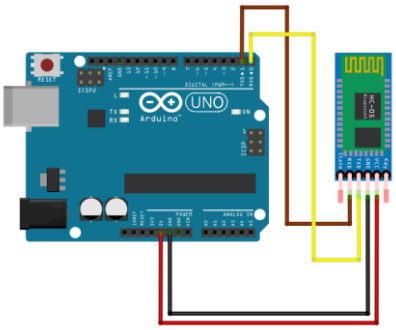


Figura 69: Conexionado HC-06

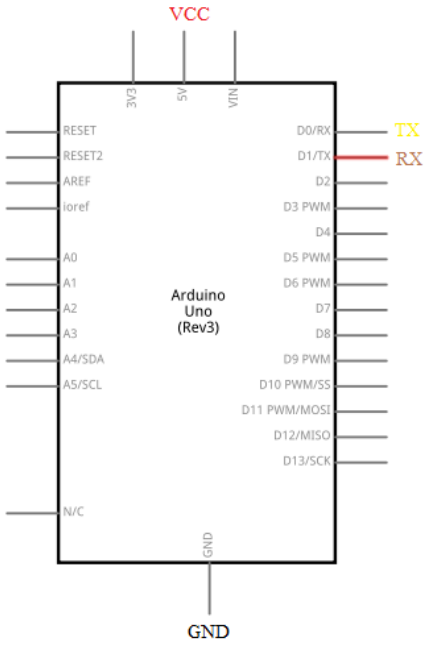


Figura 70: Circuito esquemático HC-06

En este caso también tenemos los dos pines de alimentación y otras dos conexiones: los pines TX y RX que son de envío y recibo de datos respectivamente. Lo único que tenemos que hacer es conectar el pin de TR del módulo HC-06 al pin RX de Arduino y viceversa con los otros, de forma que puedan mandarse y recibir información el uno al otro.

3.4.4. Sensor ultrasonidos HC-SR04

Por último, añadimos las figuras que corresponden al sensor de ultrasonidos:

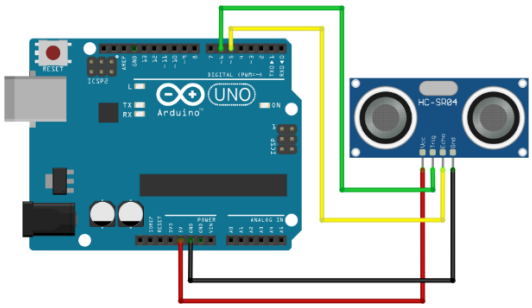


Figura 71: Conexionado sensor de ultrasonidos

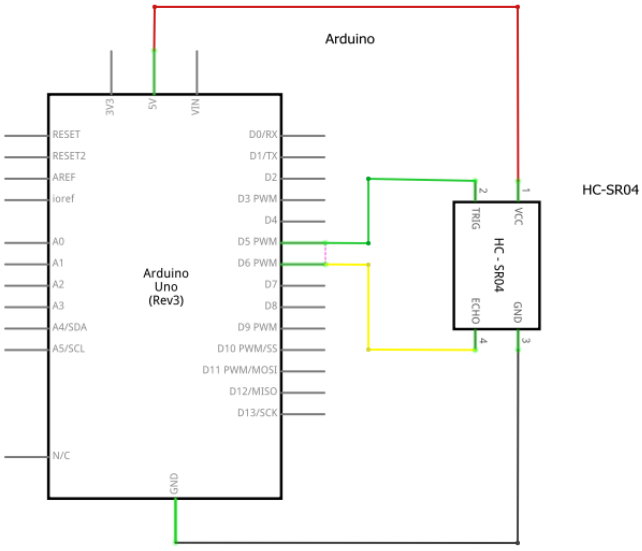


Figura 72: Circuito esquemático sensor de ultrasonidos

Este sensor, aparte de la alimentación, dispone de dos pines. El primero es el “Trigger” o disparador, es el pin que se encarga de que el sensor mande un pulso eléctrico, del cual recoge la información que recibe y la manda a través del pin “Echo”. Con esos datos y unas operaciones sencillas podremos calcular la distancia a un objetivo en cm.

4. FUNDAMENTOS DE MODELADO DE SISTEMAS

Antes de ir directamente al modelo matemático que representa el comportamiento de nuestro robot, conviene repasar brevemente toda la teoría de modelado de sistemas. Durante este capítulo ojearemos la teoría básica para dar paso, en el siguiente capítulo, a nuestro problema en concreto.

El objetivo de la ingeniería de control es llegar a conseguir que un sistema actúe de la manera deseada dentro de un entorno. Pero para conseguir el sistema debe ser estudiado y modelado, entendiendo el proceso físico que conlleva. Una vez obtenido su modelo, se puede controlar para que actúe de la manera que prefiramos.

4.1. Función de transferencia de la planta

Para definir lo que es una función de transferencia de la planta, vamos a partir de la idea de que todos los sistemas responden a unas ecuaciones integro-diferenciales que los rigen, las cuales incluyen todas las variables de las que depende el sistema. Una vez obtenidas dichas ecuaciones, deberemos aplicarles la transformada de Laplace.

Una función de transferencia se define como el cociente entre las transformadas Laplacianas de una salida (variable que queremos controlar) y una entrada (valor de referencia). Si conocemos esta función de transferencia, podremos controlar el sistema.

$$\frac{\text{Output}}{\text{Input}} = G(s) = \frac{Y(s)}{R(s)} = \frac{1}{Ms^2 + bs + k}.$$

Figura 73: Función de transferencia

4.2. Representación interna

Esta representación se basa en el concepto de *estado*. Podemos decir que lo que para definir el estado del sistema tan solo necesitamos saber el valor de ciertas variables en un momento (a estas variables se las denomina de estado). Esto además nos sirve para predecir su evolución a corto plazo.

Por tanto, podemos concluir afirmando que la representación interna es un proceso que simula la respuesta del sistema para estimar su evolución.

La principal diferencia con la función de transferencia, es que con este método no tenemos una ecuación de orden *n,* si no que tenemos *n* ecuaciones de primer orden que describen el comportamiento del sistema.

A continuación, mostramos el ejemplo de sistemas dinámicos lineales invariantes en el tiempo.

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t)$$

Figura 74: Ecuaciones de representación interna y salida

Donde:

A: matriz dinámica (*n* x *n*)

B: matriz de control (*n* x *p*)

C: matriz de salida (*m* x *n*)

D: matriz que muestra la influencia de la entrada sobre la salida (*m* x *p*), solo existe cuando *m=n*

Aquí tendríamos el diagrama de bloques que representa este espacio de estados:

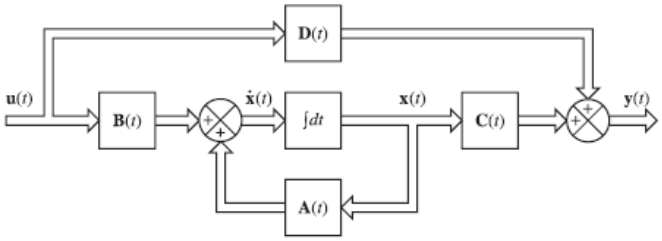


Figura 75: Diagrama de bloques representación interna

4.3. Linealización

Un sistema lineal es aquel al que podemos aplicar el principio de superposición. Esto quiere decir que no podemos estudiar las salidas/entradas por separado y sumar los resultados, con lo cual el problema se complica bastante a la hora de modelar.

En la vida real la mayoría de los problemas no son lineales, incluso los que teóricamente lo son, no responden exactamente como esperábamos, o lo hacen dentro de cierto rango de valores. Es por esto por lo que debemos linealizar los modelos matemáticos no lineales.

Para ello, podemos partir de la premisa de que el sistema va a ser lineal en torno a un punto, al cual llamamos de operación. Con esta base, podemos linealizar un modelo aplicando las series de Taylor en dicho punto de operación. Simplificando de esta forma el modelado.

$$f(x) = \sum_{n=0}^{\infty} \frac{1}{n!} \frac{d^n f(x)}{dx^n}\bigg|_{x=x_0} (x - x_0)^n$$

$$f(x) = f(x_0) + \frac{df(x)}{dx}\bigg|_{x=x_0} (x - x_0) + \frac{d^2 f(x)}{2!\, dx^2}\bigg|_{x=x_0} (x - x_0)^2 + \ldots$$

Figura 76: Serie de Taylor

5. MODELADO CINEMÁTICO DE UN ROBOT MÓVIL

Como hemos mencionado en capítulos anteriores, debemos analizar el problema físico ante el que estamos para poder modelarlo correctamente.

En nuestro caso deberemos estudiar el modelado cinemático diferencial de un robot móvil para saber cuál es la información que necesitamos, tanto para la identificación del sistema como para usar las ecuaciones diferenciales que lo rigen en las herramientas software de simulación.

5.1. Cinemática diferencial

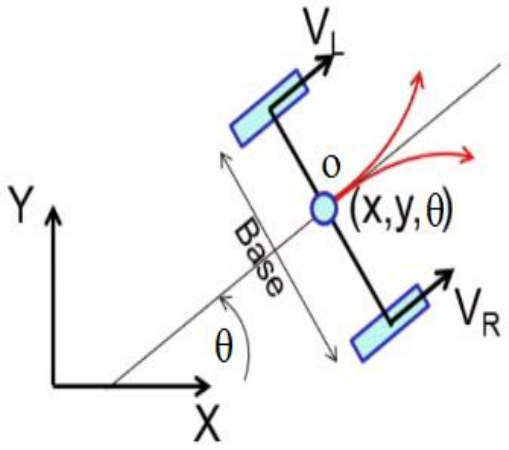
En este apartado vamos a hablar brevemente sobre que es la cinemática diferencial, para posteriormente llegar a obtener su correspondiente modelo matemático.

La cinemática diferencial de un robot móvil va a tratar de buscar la relación que existe entre la velocidad de las ruedas y la velocidad del propio robot. En nuestro caso contamos con dos ruedas motrices paralelas.

Esto nos va a proporcionar gran información y nos va a permitir girar el robot en un sentido, en otro, que siga cierta trayectoria, etc. En definitiva, manipularlo para conseguir que haga lo que deseemos.

5.2. Modelo matemático

Nuestro objetivo es desarrollar la solución al problema cinemático para un vehículo diferencial compuesto de dos ruedas motrices paralelas:



Definimos el punto ***O(x,y,θ)***, situado en el punto central entre las dos ruedas, como el sistema de referencia de nuestro vehículo.

Figura 77: Robot diferencial

Para resolver el problema de la cinemática debemos conocer la posición de este punto, es decir, debemos conocer las coordenadas ***x*** e ***y*** respecto al plano que estamos utilizando, y además el ángulo ***θ*** que lo referimos respecto al eje x, por ejemplo.

Además, podemos definir otra serie de parámetros que nos ayudarán a describir el modelo:

$\omega_R \rightarrow velocidad\ angular\ de\ la\ rueda\ derecha\ (rad/s)$

$\omega_L \rightarrow velocidad\ angular\ de\ la\ rueda\ izquierda\ (rad/s)$

$R_R \rightarrow radio\ de\ la\ rueda\ derecha$

$R_L \rightarrow radio\ de\ la\ rueda\ izquierda$

Como ambas ruedas son iguales

$$R_R = R_L = R$$

$Base \rightarrow distancia\ entre\ ruedas$

Con estas variables podemos empezar a plantear las ecuaciones que definen el movimiento del robot:

$$v_R = R_R * \omega_R \rightarrow velocidad\ lineal\ rueda\ derecha$$

$$v_L = R_L * \omega_L \rightarrow velocidad\ lineal\ rueda\ izquierda$$

Pero a nosotros nos interesa la velocidad del punto ***O*** que hemos elegido como referencia, y que resulta ser la semisuma de ambas:

$$v_o = \frac{v_R + v_L}{2} \rightarrow velocidad\ lineal\ del\ punto\ O$$

Con estas ecuaciones y para unas velocidades angulares dadas, podemos definir la nueva posición de ***O***:

$$x' = x + v_o * Cos(\theta) * t$$

$$y' = y + v_o * Sen(\theta) * t$$

Hay que tener en cuenta que esta ecuación solo es válida para un tiempo muy pequeño, es aquí donde aplicamos el concepto de linealización de una forma muy simplificada.

Pero aún nos falta otra componente para terminar de definir la nueva posición de nuestro vehículo: la orientación. Para ello vamos a aplicar unas velocidades lineales a las ruedas, y teniendo la distancia base entre ellas, podemos aproximar la rotación que sufre el robot como la siguiente velocidad angular:

$$\omega_v = \frac{v_R - v_L}{Base}$$

Y a partir de esta es fácil obtener el nuevo ángulo ***θ:***

$$\theta' = \theta + \omega_v * t$$

De nuevo, se vuelve a comentar que esta ecuación solo es válida para un instante muy pequeño de tiempo.

Luego, podemos definir de una forma más correcta las siguientes ecuaciones de forma matricial:

$$\begin{bmatrix} x(t+\Delta t) \\ y(t+\Delta t) \\ \theta(t+\Delta t) \end{bmatrix} \cong \begin{bmatrix} x(t) + v(t) * Cos\big(\theta(t)\big) * \Delta t \\ y(t) + v(t) * Sen\big(\theta(t)\big) * \Delta t \\ \theta(t) + \omega_v(t) * \Delta t \end{bmatrix}$$

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} \cong \begin{bmatrix} Cos\big(\theta(t)\big) & 0 \\ Sen\big(\theta(t)\big) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega_v(t) \end{bmatrix}$$

Todas estas ecuaciones vienen de hacer muchas simplificaciones: el suelo está liso, no hay amortiguadores, las ruedas se mantienen fijas, etc.

6. ENSAYOS DE IDENTIFICACIÓN Y OBTENCIÓN DE MODELOS

El siguiente paso a seguir sería la identificación y obtención de modelos de nuestro robot. Para ello tendremos que hacer previamente unas comprobaciones, como asegurarnos del correcto funcionamiento de los sensores y demás elementos que vayamos a utilizar. Así que en este capítulo hablaremos de la prueba y puesta en marcha de los sensores, los distintos ensayos y pruebas realizadas, de los problemas encontrados durante el proceso y de las soluciones que dimos a dichos obstáculos.

6.1. Prueba y puesta en marcha del robot y los sensores

En este apartado vamos a hablar de la puesta en marcha del robot con el controlador de los motores de corriente continua, las pruebas realizadas con los sensores de ultrasonidos, inercial y bluetooth, los problemas encontrados con cada uno de ellos y como arreglamos dichos inconvenientes para lograr un correcto funcionamiento.

6.1.1. Driver L298N

Lo primero es la puesta en marcha del robot, comprobar si el controlador funciona correctamente y si podemos controlar mediante PWM la velocidad de las ruedas. Para ello tan solo tenemos que introducir un poco de código en Arduino y ponerlo en marcha.

A continuación, escribimos el breve código con el que hicimos esta prueba y su correspondiente explicación.

```
void setup() {
 pinMode (ENB, OUTPUT);
 pinMode (ENB2, OUTPUT);
 pinMode (IN2, OUTPUT);
 pinMode (IN1, OUTPUT);
 pinMode (IN3, OUTPUT);
 pinMode (IN4, OUTPUT);
}

void loop() {

  //Preparamos la salida para que el motor gire en un sentido
  digitalWrite (IN2, HIGH);
  digitalWrite (IN1, LOW);
  digitalWrite (IN4, HIGH);
  digitalWrite(IN3, LOW);

  // Aplicamos PWM a los pines para que el robot se mueva
  analogWrite(ENB,pwma);
  analogWrite(ENB2,pwmb);
  delay(1000);
}
```

Cuadro de código 1: Prueba del controlador de los motores

En primer lugar, mostramos como en el “Setup” de Arduino configuramos los pines correspondientes al driver como de salida, ya que somos nosotros quienes vamos a disponer que estén en valor alto o bajo, o variar el valor del PWM (será un valor entre 0 y 255).

En la segunda figura simplemente configuramos en el “Loop” los pines para que el robot vaya hacia delante y le damos cierta velocidad con un valor de PWM.

Por supuesto que después de esta prueba, también probamos que fuera hacia atrás, rotara y distintos valores de PWM para comprobar que ambos motores respondían correctamente.

6.1.2. Sensor ultrasonidos HC-SR04

Para este sensor vamos a seguir el mismo procedimiento que para el anterior, vamos a programar el código en Arduino y a comprobar su funcionamiento antes de usarlo con el robot y el resto de los sensores puestos en marcha.

```
void setup()
{
   Serial.begin(9600);
   pinMode(LedPin, OUTPUT);
   pinMode(TriggerPin, OUTPUT);
   pinMode(EchoPin, INPUT);

}

void loop()
{
   int cm = ping(TriggerPin, EchoPin);
   Serial.print("Distancia: ");
   Serial.println(cm);

}
```

Cuadro de código 2: Prueba del sensor de ultrasonidos

Al igual que con el controlador de los motores, configuramos los pines del sensor para que actúen de entrada o de salida en la parte de “Setup” de Arduino. Posteriormente en el “Loop” llamamos a una función (que comentaremos posteriormente) y mostramos la distancia en centímetros por puerto serie.

Para este sensor nos encontramos con diversos contratiempos, así que antes de nada vamos a explicar cómo funciona, y a raíz de eso encontrar las causas de dichos problemas y las soluciones. Para explicar el funcionamiento nos vamos a ayudar de la función que mencionábamos anteriormente.

```
int ping(int TriggerPin, int EchoPin) {
   long duration, distanceCm;
   digitalWrite(TriggerPin, LOW);  //para generar un pulso limpio
ponemos a LOW 4us
   delayMicroseconds(4);
   digitalWrite(TriggerPin, HIGH);  //generamos Trigger (disparo)
de 10us
   delayMicroseconds(10);
   digitalWrite(TriggerPin, LOW);
   duration = pulseIn(EchoPin, HIGH);  //medimos el tiempo entre
pulsos, en microsegundos

   distanceCm = duration * 10 / 292/ 2;   //convertimos a
distancia, en cm
   return distanceCm;
}
```

Cuadro de código 3: Función del sensor de ultrasonidos

El funcionamiento de este sensor es el siguiente: envía un pulso de alta frecuencia que, tras rebotar en un obstáculo, es reflejado hacia el sensor, el cual dispone de un receptor capaz de captarlo. Conocida la velocidad de la transmisión de dicho pulso por el aire (la velocidad del sonido) y midiendo el tiempo entre el envío y la recepción de pulsos, seremos capaces de estimar la distancia a la que se encuentra el obstáculo. A continuación, mostramos el razonamiento con el que conseguiremos la distancia en centímetros:

$$velocidad\ del\ sonido = 343\frac{m}{s} * \frac{1}{10^6}\frac{s}{\mu s} * 100\frac{cm}{m} = \frac{1}{29.2}\frac{cm}{\mu s}$$

Esta ecuación nos muestra que el pulso tarda casi treinta microsegundos en recorrer un centímetro de distancia. De forma que podemos despejar la distancia según:

$$distancia\ (cm) = \frac{tiempo\ (\mu s)}{29.2 * 2}$$

Ese multiplicador por dos que aparece en el denominador de la ecuación se debe a que nosotros medimos el tiempo que tarda el pulso en ir y volver, por tanto, hay que dividirlo entre dos para obtener la distancia real entre el sensor y el obstáculo.

Como vemos en el código adjunto anteriormente, el pin de “Trigger” es el encargado de disparar el pulso, y el pin de “Echo” es el receptor. En este caso mandamos un pulso de microsegundos con el disparador, y cuando el receptor del sensor capta la señal que ha rebotado en el obstáculo, pasa de un estado a nivel bajo, o “Low”, a un estado de nivel alto, o “High”. A partir de ahí, con la ayuda de la función *pulseIn,* que nos dice el tiempo que pasa desde que un pin pasa de un valor bajo a uno alto o viceversa, somos capaces de medir el tiempo entre pulsos. Una vez con esta información y las ecuaciones anteriores, somos capaces de obtener la distancia en centímetros.

Un problema que surge a raíz de cómo funciona este sensor, es que si aparece un obstáculo “instantáneamente” (como poner la mano de golpe delante del sensor, o agitarla hacia delante y hacia atrás muy rápido), la información que registra es errónea y nos proporciona una distancia falsa (generalmente una distancia de tres mil centímetros). A continuación, añadimos una figura que demuestra este comportamiento:

Figura 78: Datos erróneos sensor de ultrasonidos

En este caso hemos puesto un obstáculo delante del sensor de ultrasonidos, acercándolo y alejándolo lentamente, y de vez en cuando agitábamos la mano para ver que da la distancia errónea de tres mil centímetros.

Otro dato importante que comentar, es que este sensor de ultrasonidos emite y recibe los pulsos en línea recta (o con un ángulo muy pequeño de no más de diez grados de amplitud). Con lo cual en las curvas no va a ser muy preciso, pudiendo incluso dar resultados incoherentes en algunos casos. Además de no recibir bien los pulsos al estar en una curva, debemos añadir que es posible que la distancia a un obstáculo cambie bruscamente con el giro, con lo cual la fórmula con la que obtenemos la distancia ya sería incorrecta, ya que no tarda el mismo tiempo en llegar al primer obstáculo antes del giro y en volver del segundo obstáculo tras dar la curva.

6.1.3. Sensor inercial BNO-055

Este es el sensor más complejo y de mejor calidad del que disponemos. Es capaz de proporcionarnos mucha información, como por ejemplo: un vector de orientación absoluta, con los ángulos que giras en cada uno de los tres ejes de un sistema cartesiano, un vector de rotación absoluto con las tres velocidades angulares de dichos ejes, un vector con las aceleraciones de gravedad en el eje Z y las correspondientes a los ejes X e Y, etc.

Para usarlo dispondremos de varias librerías proporcionadas por **Adafruit Industries,** donde además viene una guía con toda la información que hay que saber de este sensor. Gracias a estas librerías seremos capaces de utilizar ciertas funciones que nos proporcionarán la información directamente, sin necesidad de hacer ningún tipo de cálculo o de plantear ecuaciones como el caso del sensor anterior.

Ponemos como ejemplo el siguiente código, obtenido directamente de la página web de Adafruit, en el que solo se dispone a obtener la orientación en los tres ejes espaciales:

```
imu::Vector<3> euler =
bno.getVector(Adafruit_BNO055::VECTOR_EULER);

/* Display the floating point data */
Serial.print("X: ");
Serial.print(euler.x());
Serial.print(" Y: ");
Serial.print(euler.y());
Serial.print(" Z: ");
Serial.print(euler.z());
Serial.println("");
```

Cuadro de código 4: Posición angular en los tres ejes espaciales

En principio este parecía el sensor más sencillo de usar, puesto que todas las funciones y librerías ya venían predefinidas y listas para su uso, tan solo debíamos seguir instrucciones y obtener la información que necesitáramos de dicho sensor.

Hay que añadir, además, que en un principio se había pensado obtener tanto posición como aceleración (ambas obtenibles con este sensor), para después poder aplicar un filtro de Kalman y conseguir identificar el sistema con más precisión que con una sola de las dos medidas (más adelante hablaremos sobre este tema).

El problema vino cuando nos dimos cuenta de que algunos datos como eran los de aceleración eran erróneos. Lo primero que debíamos hacer era calibrar el sensor, para ello nos ayudamos de la función *getCalibration* que nos proporcionaban las librerías de Adafruit Industries.

```
/* Get the four calibration values (0..3) */
/* Any sensor data reporting 0 should be ignored, */
/* 3 means 'fully calibrated" */
uint8_t system, gyro, accel, mag;
system = gyro = accel = mag = 0;
bno.getCalibration(&system, &gyro, &accel, &mag);
 /* The data should be ignored until the system calibration is > 0 */
  Serial.print(" Aceleración:");
  Serial.println(accel, DEC);
```

Cuadro de código 5: Calibración de la aceleración

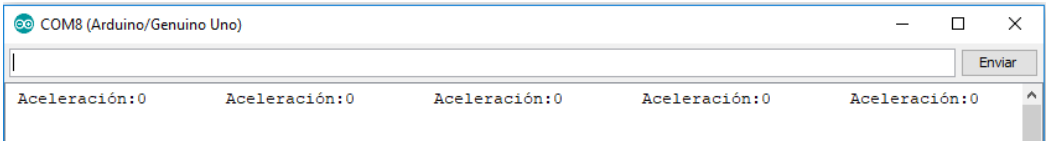


Figura 79: Datos erróneos de calibración de la aceleración

Aquí mostramos como no conseguíamos que el BNO se calibrase, por eso los valores de aceleración resultaban incoherentes. A continuación, mostramos una matriz donde utilizando la función de aceleración lineal obteníamos datos erróneos.

```
 imu::Vector<3> acceleration =
bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
```

Cuadro de código 6: Obtención del vector aceleración lineal

matriz

6200x5 double

	1	2	3	4	5
3004	139	0	-167.3800	-1.4600	2.3000
3005	139	0	-164.0600	0.6300	-0.1100
3006	139	0	-168.8100	0	0
3007	139	0	-174.9400	-0.2800	0.3100
3008	139	0	178.6900	0.1800	-0.2000
3009	139	0	172.5600	0.1900	0.0600
3010	139	0	166.5000	-0.2300	0.4500
3011	139	0	160.5600	0.1100	-0.3900
3012	139	0	154.5600	0.2100	-0.0700
3013	139	0	148.7500	0	0.3900
3014	139	0	143.0600	0.1700	0.0400
3015	139	0	137.1300	0.2400	-0.1600
3016	139	0	131.1300	-0.1600	0.3800
3017	139	0	125.3800	-0.1600	-0.0400
3018	139	0	119.5000	-0.0100	-0.4700
3019	139	0	113.4400	-0.0100	-0.5300
3020	139	0	107.2500	0.1800	0.1000
3021	139	0	101.1900	0	-0.0800
3022	139	0	95.2500	-0.1300	0.2100
3023	139	0	89.1900	0.1600	-0.2400

Figura 80: Datos de aceleración del BNO

Las dos primeras columnas representan el valor del PWM de cada rueda, en este caso he traído un fragmento de veinte muestras de un ensayo en el que recogimos mas de seis mil. La columna tres nos indica la posición angular del robot, y las columnas cuatro y cinco representan las aceleraciones en los ejes X e Y respectivamente. Lo que queremos reflejar es que en este tiempo (dos segundos) el robot estaba girando sobre una rueda en un único sentido, y que no es normal que de un muestreo a otro pasemos de aceleraciones negativas a positivas, o cambios bruscos como vemos en las dos primeras filas de ambos ejes.

Además de estos resultados incoherentes disponíamos de la posición angular del robot, con lo cual podríamos prever como iban a variar las aceleraciones dependiendo de dicha posición.

Por supuesto también probamos otros ensayos en los que hacíamos que el robot solo fuera hacia delante, con distintas velocidades en ambas ruedas, hacia atrás, etc. Y los resultados seguían siendo incoherentes.

Así que descartamos la posibilidad de obtener la aceleración con este sensor, y pensamos en otra forma de conseguir la identificación del sistema, un tema que ya comentaremos más adelante.

6.1.4. Módulo Bluetooth HC-06

Antes de comentar el funcionamiento y programación de este módulo, hay que hablar de las causas que llevaron a su instalación y los requisitos que debe cumplir para satisfacer nuestras necesidades.

Antes de disponer de este módulo, ya intentábamos identificar el sistema, obteniendo los datos por puerto serie gracias a un cable USB. El problema que teníamos era que necesitábamos de un cable lo suficiente extenso como para permitir al robot moverse con libertad.

El problema que esto conlleva es que el cable lastraba al robot de un lado más que de otro, con lo cual no se movía igual con él, que sin él. El resultado: se falseaban las medidas. Por este motivo hubo que pensar en hacer una comunicación inalámbrica con Matlab y utilizamos el módulo Bluetooth para ello.

Antes de seguir con la recogida de muestras de los distintos ensayos, teníamos que asegurarnos de una cosa: para que la herramienta de "System Identification" funcionara correctamente, había que tener la certeza de que el tiempo de muestreo era correcto, y que la información nos llegaba era válida.

Este paso también lo habíamos tenido en cuenta para la comunicación por USB, pero es de vital importancia comentar como se ha conseguido y mostrar que funciona correctamente. Para ello adjuntamos la breve programación en Arduino y una figura con los resultados de una matriz de muestras en Matlab.

```
unsigned long temporizador = 0; //variable de temporizador
unsigned long tm = 100000; //esta cifra es en microsegundos, el
tiempo de muestreo es de 100ms=0.1s

void setup() {

 temporizador = micros();
}

void loop() {

  temporizador += tm;

  //aquí pondríamos el resto del código

  while (micros() < temporizador + tm) {
  }
  //este while se encarga de mantener constante el tiempo de
muestreo para cada iteración

}
```

Cuadro de código 7: Ciclo para asegurar el tiempo de muestreo

Aunque todo está comentado en el código, hay que dejar claro que ese *while* es el encargado de que el programa "espere" el tiempo necesario para cada iteración.

matriz

6200x4 double

	1	2	3	4
3712	135	0	-5	425002
3713	135	0	-10.5000	425102
3714	135	0	-16.5600	425202
3715	135	0	-22.4400	425302
3716	135	0	-28.4400	425402
3717	135	0	-34.3100	425502
3718	135	0	-40.2500	425602
3719	135	0	-46.0600	425702
3720	135	0	-51.6900	425802
3721	135	0	-57.5000	425901
3722	135	0	-63.3800	426002
3723	135	0	-69.1300	426102
3724	165	0	-74.9400	426202
3725	165	0	-80.8800	426302
3726	165	0	-87.1900	426402
3727	165	0	-93.7500	426502
3728	165	0	-100.1900	426602
3729	165	0	-106.5600	426702
3730	165	0	-113.2500	426802

Figura 81: Datos de tiempo de muestreo

En esta figura vemos una matriz con cuatro columnas, las dos primeras representan los valores de PWM de cada rueda (hay que aclarar que a veces utilizábamos el mismo valor de PWM para ambas ruedas, con lo cual no quiere decir que siempre hiciéramos el mismo ensayo de hacer girar al robot en un sentido).

La tercera columna se corresponde a la posición angular en un rango de -180º a 180º. Y la cuarta corresponde con el tiempo en el que se envía cada muestra en milisegundos. Como vemos cada muestra se envía cada 100 ms que es el tiempo de muestreo que hemos utilizado en nuestros experimentos.

Por último, añadir que todo el proceso de conseguir conectar el módulo Bluetooth con Matlab ya fue explicado en el capítulo tres cuando hablamos del software de Matlab y las Toolbox que íbamos a utilizar, con lo cual es innecesario repetir dicha información.

Una vez nos hemos asegurado del correcto funcionamiento y de las limitaciones de cada uno de los elementos necesarios para la identificación, vamos a dar paso a todo el trabajo que hicimos para la correcta identificación del sistema.

6.2. Identificación del sistema

Antes de mostrar los distintos ensayos realizados, los errores cometidos y las soluciones a cada uno de los problemas, tenemos que plantearnos que esperamos obtener y como, con los datos de nuestros sensores, podemos llegar a identificar correctamente el sistema y obtener la función de transferencia que nos diga el comportamiento de cada motor. Así que en esta sección vamos a ver todos los conocimientos previos y pasos seguidos hasta poder llegar al uso de la Toolbox: System Identification de Matlab.

6.2.1. Función de transferencia de un motor de CC

Empezaremos pues, ilustrando cual es la función de transferencia de un motor de corriente continua controlado por tensión y sin despreciar los efectos de fricción, inductancia, resistencia del bobinado e inercia mecánica. A continuación, mostramos su circuito esquemático:

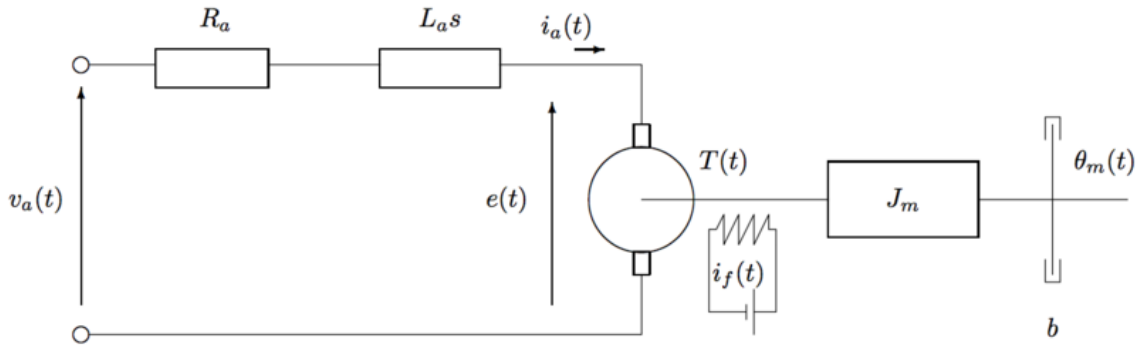


Figura 82: Circuito esquemático de un motor de CC

Hay que analizar tanto el circuito eléctrico como el circuito mecánico para llegar a nuestra solución. Empezaremos obteniendo las ecuaciones de ambos circuitos, les aplicaremos la transformada de Laplace y operaremos con ellas hasta obtener la relación entre la salida y la entrada del motor, es decir, la función de transferencia.

Circuito eléctrico

$$V_a(t) = R_a * i_a(t) + L * \frac{di}{dt} + e(t) \quad \rightarrow \quad V(s) = R * I(s) + L * s * I(s) + E(s) \quad [1]$$

$$T(t) = k_t * i(t) \quad \rightarrow \quad T(s) = k_t * I(S) \quad [2]$$

$$e(t) = k_e * \frac{d\theta}{dt} \quad \rightarrow \quad E(s) = k_e * \theta * s \quad [3]$$

Circuito mecánico

$$T(t) = J\frac{d^2\theta}{dt^2} + B\frac{d\theta}{dt} \longrightarrow T(s) = J * s^2 * \theta + B * s * \theta \quad [4]$$

Antes de operar con las ecuaciones vamos a comentar que es cada parámetro:

V_a: tensión de alimentación
R: resistencia
L: bobina
i_a: corriente que circula por el rotor
e: tensión en el rotor
k_i: constantes de proporcionalidad
T: par motor
J: inercia mecánica
B: pérdidas por rozamiento
θ: posición angular

Comenzamos a operar sustituyendo [3] en [1], pudiendo despejar posteriormente la corriente (en dominio de Laplace):

$$V = RI + LsI + k_e s\theta$$

$$I = \frac{V - k_e s\theta}{R + Ls} \qquad [5]$$

Ahora vamos a igualar las ecuaciones [2] y [4]:

$$k_t I = Js^2\theta + Bs\theta \quad [6]$$

Si ahora sustituimos [5] en [6] obtenemos una ecuación de la cual podemos obtener nuestra función de transferencia, quedando así:

$$k_t \frac{V - k_e s\theta}{R + Ls} = Js^2\theta + Bs\theta$$

$$G(s) = \frac{\theta(s)}{V(s)} = \frac{k_e}{(s^2 J + sB)(R + Ls) + k_e k_t s}$$

Aunque se tenga esa función de transferencia, hay que tener en cuenta que, por los típicos valores que suelen tomar dichas variables, se puede simplificar a una función de transferencia de primer orden y a un integrador. A continuación, mostramos un ejemplo:

Parámetro	**Valor (unidades)**
J	0.01 (kg*m^2)
B	0.1 (N*m*s)
Ke	0.01 (V/(rad/s))
Kt	0.01 N*m/A
R	1 Ω
L	0.5 H

Tabla 1: Valores típicos de los parámetros de un motor CC

$$G(s) = \frac{\theta(s)}{V(s)} == \frac{0.01}{(0.01s^2 + 0.1s)(1 + 0.5s) + 0.0001s}$$

$$G(s) = \frac{\theta(s)}{V(s)} = \frac{0.01}{0.005s^3 + 0.05s^2 + 0.01s^2 + 0.1s} = \frac{0.01}{s(0.005s^2 + 0.06s + 0.1)}$$

$$G(s) = \frac{\theta(s)}{V(s)} = \frac{0.01}{s(0.06s + 0.1)} = \frac{0.1}{s(0.6s + 1)}$$

Después de esta demostración queda claro que buscamos una función de transferencia de primer orden y un integrador, quedando así nuestro diagrama de bloques:

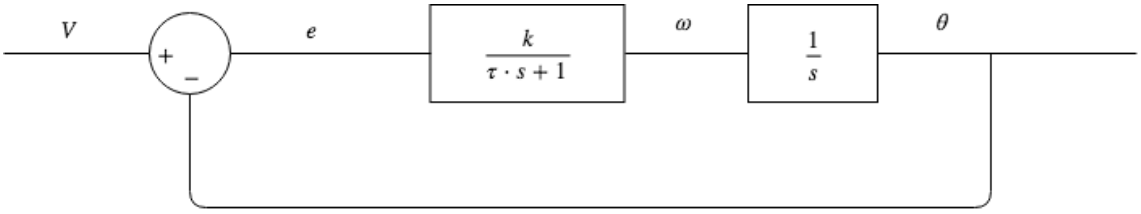


Figura 83: Diagrama de bloques motor CC

6.2.2. Tratamiento de la información

Una vez sabemos la función de transferencia que rige el comportamiento de un motor de corriente continua, el siguiente paso va a ser establecer la relación entre los datos de nuestros sensores y los datos que necesitamos para obtener la función de transferencia, además de todo el tratamiento de dicha información en Matlab para poder usar la herramienta de “System Identification”.

6.2.2.1. Relación entre datos obtenidos y requeridos

Tensión

Este parámetro es el más sencillo de obtener, ya que nosotros controlamos la velocidad de los motores desde Arduino con un valor de PWM que oscila entre cero y doscientos cincuenta y cinco. Con lo cual para obtener el valor de tensión que estamos aplicando tan solo debemos hacer una proporción teniendo en cuenta la tensión de alimentación:

$$V_{aplicada} = \frac{PWM_{aplicado}}{255} * V_{alimentación}$$

Posición angular de las ruedas

Este otro dato, sin embargo, es algo más complejo de conseguir. Hay que partir de la base de que no disponemos de un tacómetro que nos proporcione datos de posición o velocidad angular de los motores, tan solo sabemos la posición angular del robot, con lo cual hay que razonar mediante relaciones geométricas la relación que existe entre el giro de las ruedas y el giro del robot.

En principio, si pensamos en el robot moviéndose aleatoriamente (con ambas ruedas a distintas velocidades, pudiendo cambiar incluso de sentido), es prácticamente imposible que podamos establecer una relación entre la posición angular del robot y las vueltas que ha dado cada rueda, más aun teniendo en cuenta que pueden cambiar de sentido. Es un problema muy complicado de resolver.

Para que podamos solventar esta dificultad, hay que simplificar al máximo el problema: vamos a hacer que el robot gire sobre una rueda, es decir, vamos a dejar una quieta y la otra con cierta velocidad.

De esta forma podemos afirmar dos cosas: la distancia lineal que recorre la rueda en movimiento va a ser la propia longitud de esa circunferencia; el movimiento que va a seguir el robot va a ser un arco de circunferencia, cuyo radio podemos medir (la base entre ambas ruedas). Conociendo estos hechos y aplicando las ecuaciones necesarias podemos resolver este problema:

Distancia recorrida por una rueda

$$\frac{360º}{2\pi R} = \frac{\alpha}{x}$$

Teniendo en cuenta esa relación, podemos decir que:

$$x = 2\pi * R * \frac{\alpha}{360}$$

Siendo:

x = distancia lineal recorrida por la rueda

α = ángulo girado por la rueda

Arco realizado por el robot

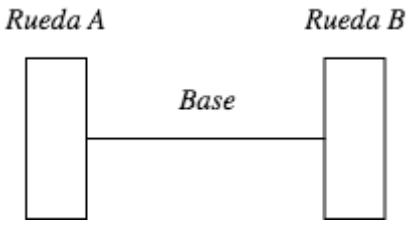


Figura 84: Arco realizado por el robot 1

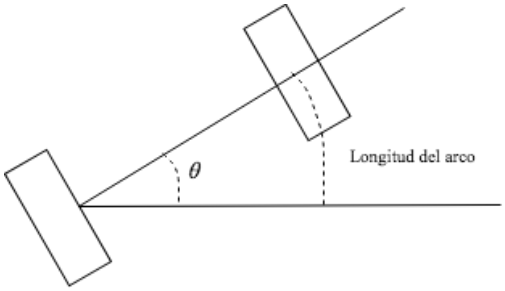


Figura 85: Arco realizado por el robot 2

Ecuación de la longitud del arco de una circunferencia:

$$L = 2\pi * Base * \frac{\theta}{360}$$

Ahora lo único que hay que pensar es que la longitud de ese arco de circunferencia es la distancia lineal que ha recorrido la propia rueda, luego se puede establecer la siguiente igualdad:

$$2\pi * Base * \frac{\theta}{360} = 2\pi * R * \frac{\alpha}{360}$$

$$Base * \theta = R * \alpha$$

Esta ecuación nos muestra una relación lineal entre el ángulo que gira nuestro robot y el ángulo que gira la rueda. Conocidos los datos de base del robot y radio de la rueda podemos obtener la posición angular de la rueda a partir de la posición angular del robot (que es el dato que el sensor BNO-055 nos proporciona).

6.2.2.2. Funciones de Matlab

El primer paso es crear una función con la que consigamos descodificar la cadena de texto que mandamos desde Arduino hasta Matlab gracias al módulo Bluetooth, de forma que podamos guardar los datos de interés en una matriz para después operar con ellos de forma sencilla. En primer lugar, mostramos como enviamos los datos desde Arduino a Matlab:

```
  pwmA = String(pwm_valueA);
  pwmB = String(pwm_valueB);
  orientacion = String(orientation);
  String tiempo=String(micros()/1000); //tiempo de envío en
milisegundos
  Serial.println(pwmA+"/"+pwmB+"/"+orientacion+"/"+tiempo+"/");
```

Cuadro de código 8: Envío de información desde Arduino por Bluetooth

La variable “tiempo” la enviamos tan solo para tener la certeza de que el tiempo de muestreo era correcto y enviábamos las muestras cada 100ms. A continuación, mostramos la función que consigue interpretar estos datos:

```
//Función de Matlab
num_muestras=6200;
muestra=1;
num_columnas=1;
j=1;
%En esta parte leemos la cadena de String que proviene de Arduino y
obtenemos los datos numéricos para guardarlos en una matriz

while muestra<=num_muestras
    vector_datos=fscanf(s);
    longitud=length(vector_datos);
    for k=1:longitud
        if (vector_datos(k)=='/')
            j=1;
            numero = str2double(num);
            num='';
            matriz(muestra,num_columnas)=numero;
            num_columnas=num_columnas+1;
        else
            num(j)=vector_datos(k);
            j=j+1;
        end
    end
    %Aumento del contador de la muestra e inicialización
    muestra=muestra+1;
    num_columnas=1;
    j=1;
end
```

Cuadro de código 9: Lectura e interpretación de datos en Matlab

Tras conectar el módulo Bluetooth tan solo teníamos que ejecutar este código para convertir las cadenas de texto que leíamos y crear una matriz con toda la información.

El siguiente paso es operar con el dato de orientación: hay que conseguir que los datos no estén entre 0 y 360º, si no que representen una curva del ángulo total girado por el robot, para poder identificar correctamente el sistema.

```
nvueltas=0;
matriz(1,5)=matriz(1,4);
for i=2:6200
    if (abs(matriz(i,4)-matriz(i-1,4))>100) %detecto una vuelta
        if(matriz(i-1,4)-matriz(i,4)>0) %significa que va hacia
delante
            nvueltas=nvueltas+1;
        else %va hacia atras y da una vuelta
            nvueltas=nvueltas-1;
        end

    end
    matriz(i,5)=matriz(i,4)+360*nvueltas;
end

pwm=matriz(:,1);
angulo=matriz(:,5);
```

Cuadro de código 10: Creación de la curva de posición angular

A continuación, mostramos una figura donde conseguimos la deseada curva en un ensayo en el que el robot tan solo se movía hacia delante:

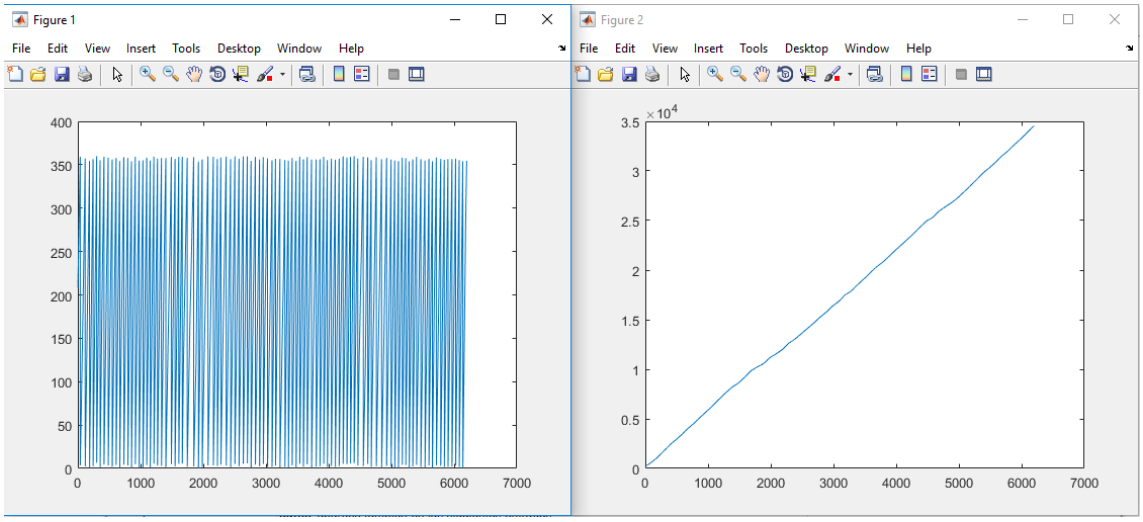


Figura 86: Curva de orientación

Una vez recogidos los datos y aplicadas estas funciones, estamos listos para usar los datos en la “System Identification Toolbox”.

6.2.3. Ensayos de identificación

En este apartado vamos a hablar sobre los distintos ensayos y pruebas que han sido realizadas con el robot, cuales han sido descartados y la solución final a la que llegamos.

En un principio se pensó en aplicar un “Pseudo-Random Signal” al robot, es decir, que se moviera con total libertad y variando, independientemente, la velocidad y sentido de cada una de las ruedas. El problema que este ensayo traía consigo, como ya he mencionado en el apartado anterior, es que no disponíamos de los datos de posición o velocidad angular de las ruedas y es prácticamente imposible obtener información de ambas ruedas con tan solo la posición angular del robot (dato que nos proporciona el sensor).

De hecho, este primer ensayo fue el precursor de la búsqueda de una relación entre el ángulo girado por la rueda y la posición del robot (lo hablado en la sección anterior). Es por eso por lo que fue descartado. Si hubiéramos dispuesto de un tacómetro en cada motor quizá hubiera sido viable este proceso para la identificación del sistema, pero debida a la gran dificultad que suponía hubo que estudiar otras alternativas.

Después de haber reflexionado y conseguir llegar a una relación lineal entre la posición angular de la rueda y el robot cuando hacíamos que girara sobre una rueda, llegamos a la conclusión que ese sería el mejor ensayo para identificar nuestro sistema: aplicar un PWM aleatorio a una rueda mientras manteníamos la otra quieta. De esta forma podíamos obtener el valor de la tensión gracias a que teníamos el valor de PWM y la posición angular gracias a la ecuación del apartado anterior. No obstante, también nos tropezamos con diversos obstáculos siguiendo este procedimiento, pero era el primer paso para lograr la identificación del sistema.

Primer ensayo

Para este primer ensayo decidimos aplicar un valor de PWM aleatorio a cada rueda con un valor entre 60 y 255 (el valor mínimo se debe a que, con un valor inferior a ese, el robot apenas se movía debido al peso del mismo), a la vez que podía que cambiar, también aleatoriamente, de sentido. Manteníamos el mismo valor de PWM durante un escalón de dos segundos, de forma que el robot tuviera el tiempo suficiente para moverse para una velocidad en concreto. Estos fueron los resultados:

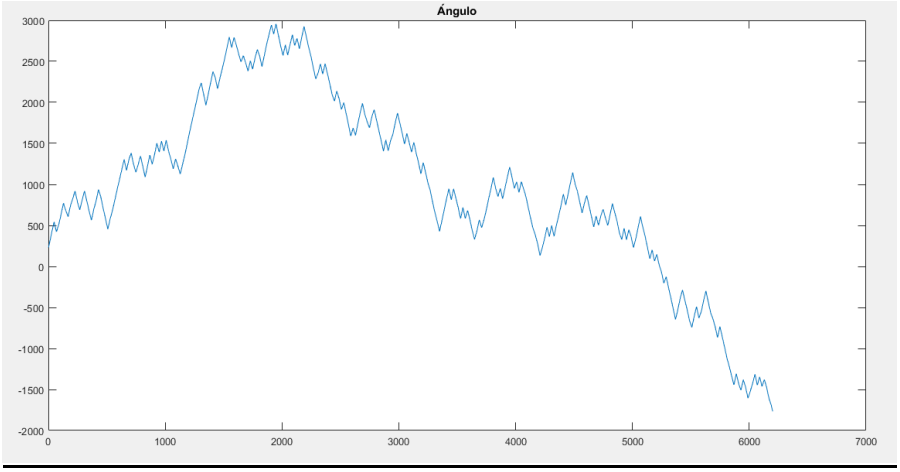


Figura 87: Posición angular, primer ensayo

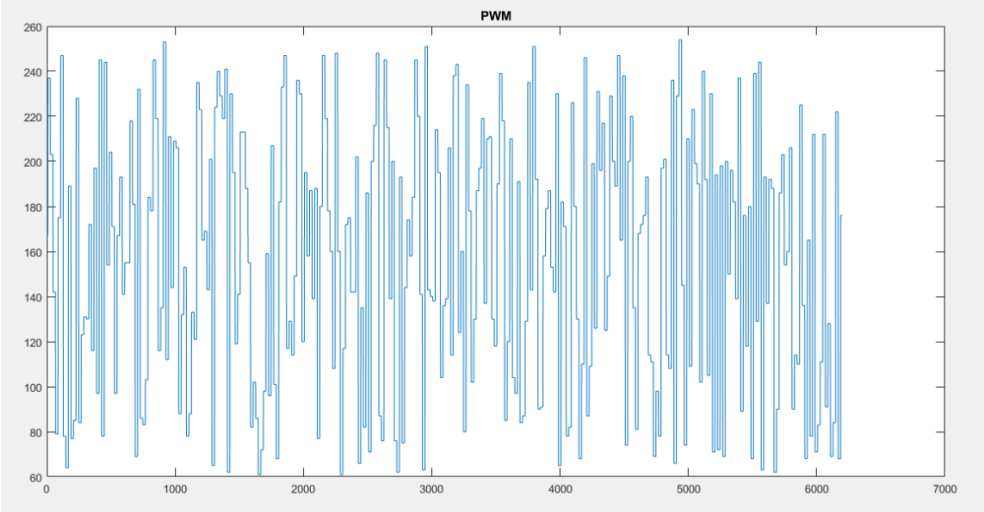


Figura 88: Valores de PWM, primer ensayo

Una vez conseguidos todos los datos es hora de identificar el sistema gracias a la Toolbox de Matlab, siguiendo los pasos que mencionamos en el capítulo tres. A continuación, mostramos las figuras que corresponden a la lectura de los datos de entrada y salida, y a dos estimaciones realizadas con dichas muestras:

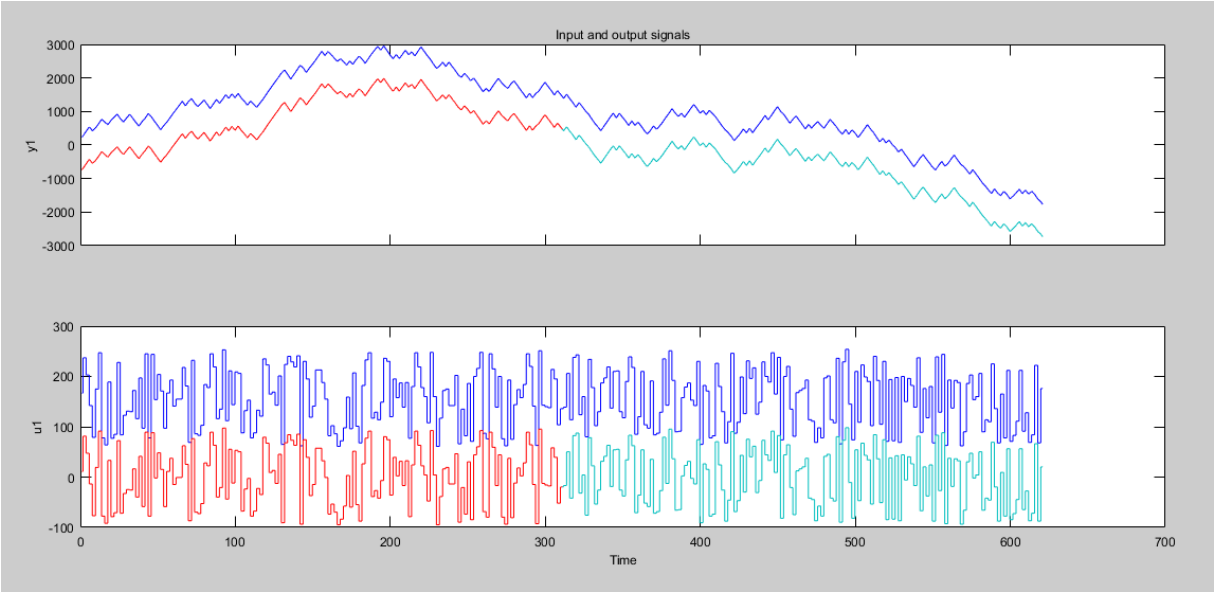


Figura 89: Datos de entrada y salida, primer ensayo

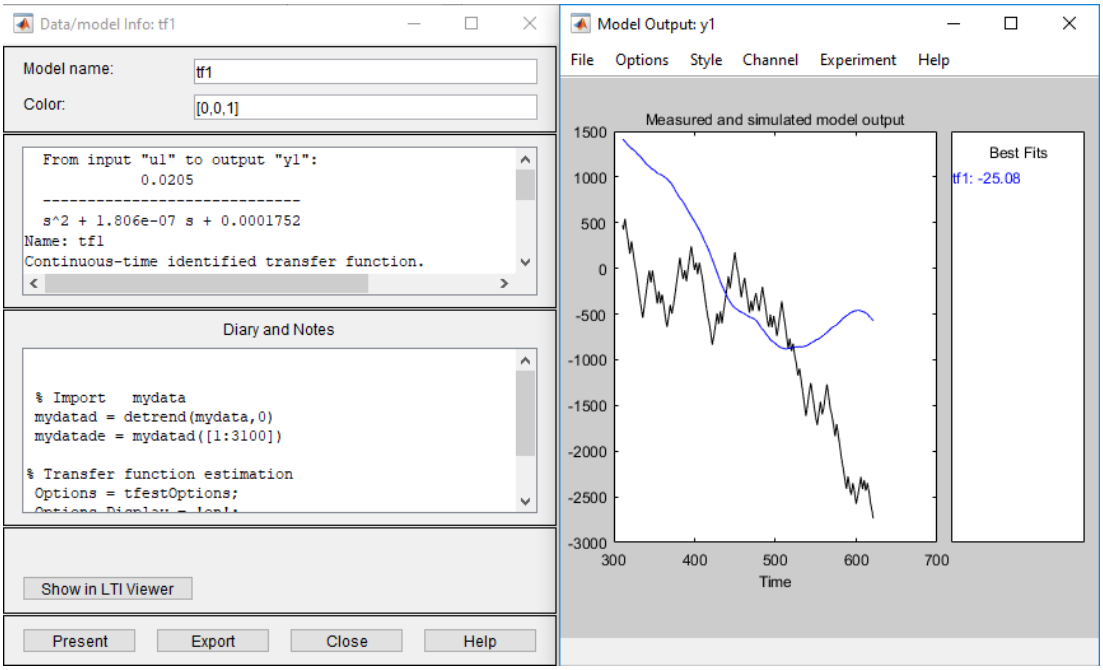


Figura 90: Estimación de modelos función de transferencia, primer ensayo

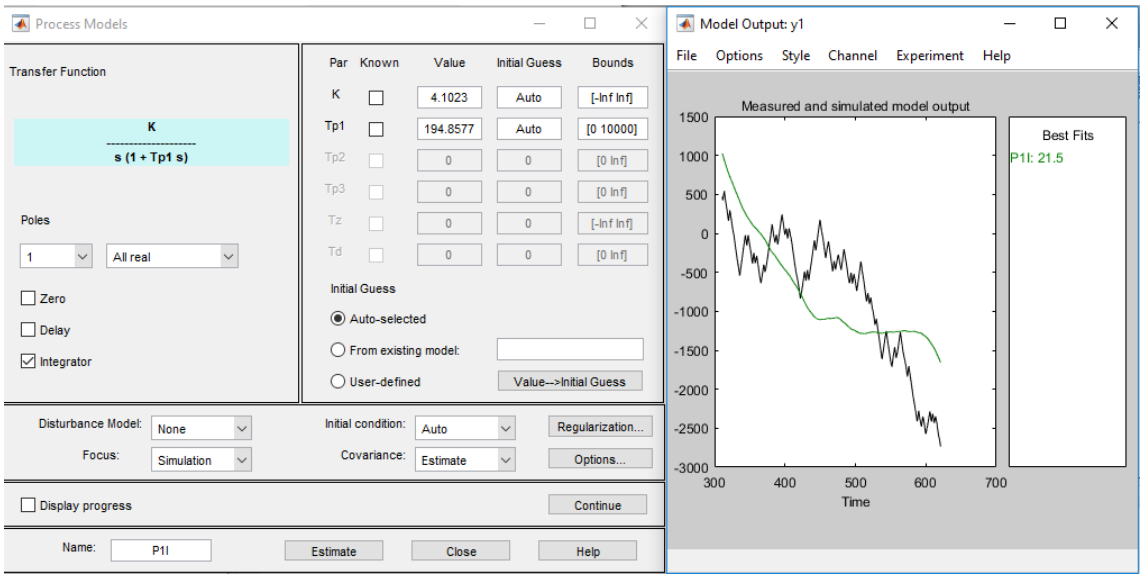


Figura 91: Estimación de modelo del proceso, primer ensayo

Como se puede ver en ambos ensayos, salen funciones de transferencia sin sentido. En ambos casos salen unas constantes de tiempo enormes, lo que nos indicaría que hay dos integradores, lo cual no tiene sentido físico para nuestro problema.

Segundo ensayo

Después de la primera identificación fallida, nos dedicamos a observar los datos detenidamente, percatándonos de que había pequeños fallos en las primeras muestras de cada escalón de dos segundos de PWM cuando la rueda cambiaba de sentido.

43	203	0	146.6900	146.6900	506.6900
44	203	0	153.0600	153.0600	513.0600
45	203	0	159.6300	159.6300	519.6300
46	203	0	166.1300	166.1300	526.1300
47	203	0	172.5600	172.5600	532.5600
48	142	0	179	179	539
49	142	0	-177.5600	182.4400	542.4400
50	142	0	176	176	536
51	142	0	170.3100	170.3100	530.3100
52	142	0	163.8800	163.8800	523.8800
53	142	0	158.4400	158.4400	518.4400
54	142	0	152.3800	152.3800	512.3800

Figura 92: Datos erróneos del primer ensayo

Las dos primeras matrices representan los valores de PWM de cada rueda, la tercera y cuarta columna son la posición angular del robot en los intervalos de -180º a 180º y de 0º a 360º respectivamente. La última es el resultado de operar con dicha posición para obtener la curva para la identificación.

Una vez aclarado esto, podemos observar como para el valor 203 de PWM el robot va hacia delante y de como para el valor 142 va hacia atrás, pero las primeras muestras en las que el robot cambia de sentido, por la inercia del robot, no son correctas.

Pensamos entonces que un problema sería la corta duración de esos escalones de PWM, por lo que el siguiente paso sería probar una identificación con escalones de diez segundos, asegurando que no se falsearan tantísimas medidas cada dos segundos.

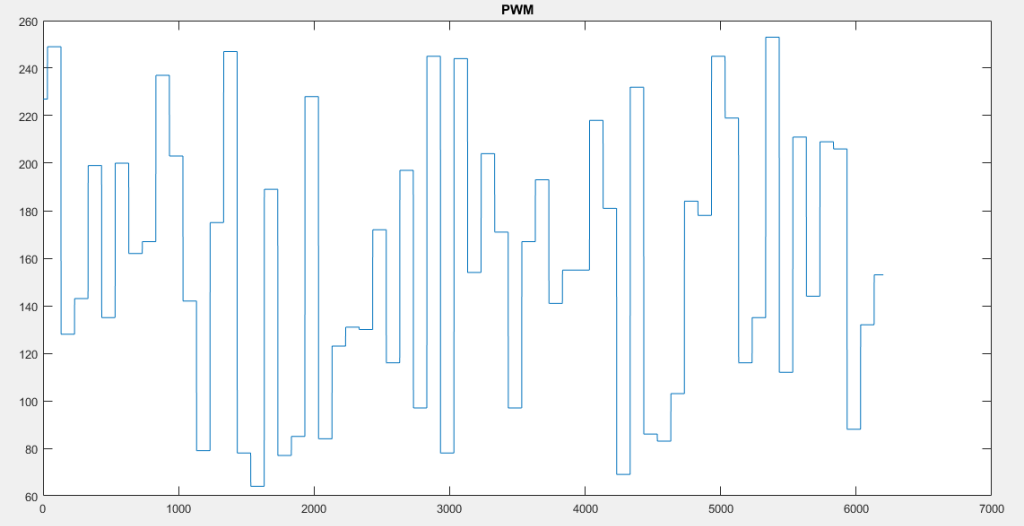


Figura 93: Valores de PWM, segundo ensayo

Aunque ya se aprecia la menor cantidad de escalones respecto al ensayo anterior, añadiré una figura más donde se ve la duración de cada escalón más claramente:

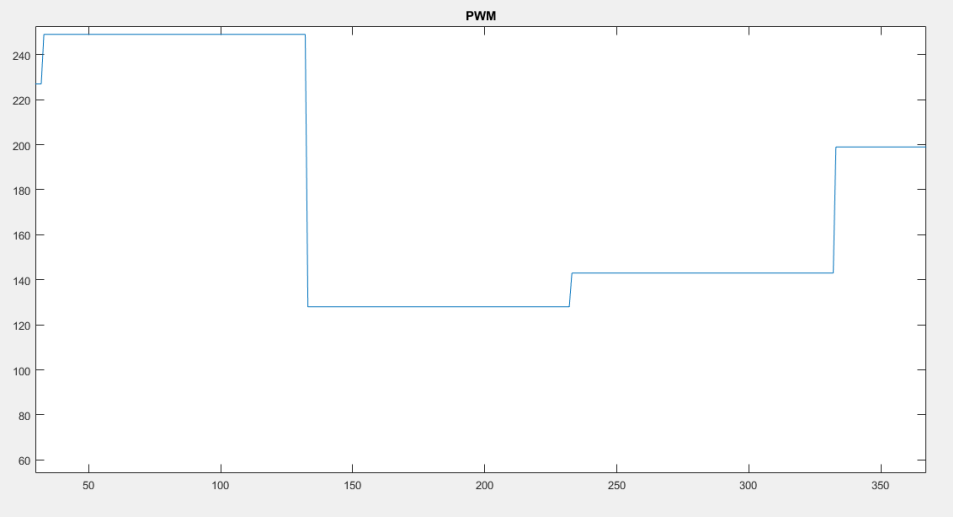


Figura 94: Duración de PWM, segundo ensayo

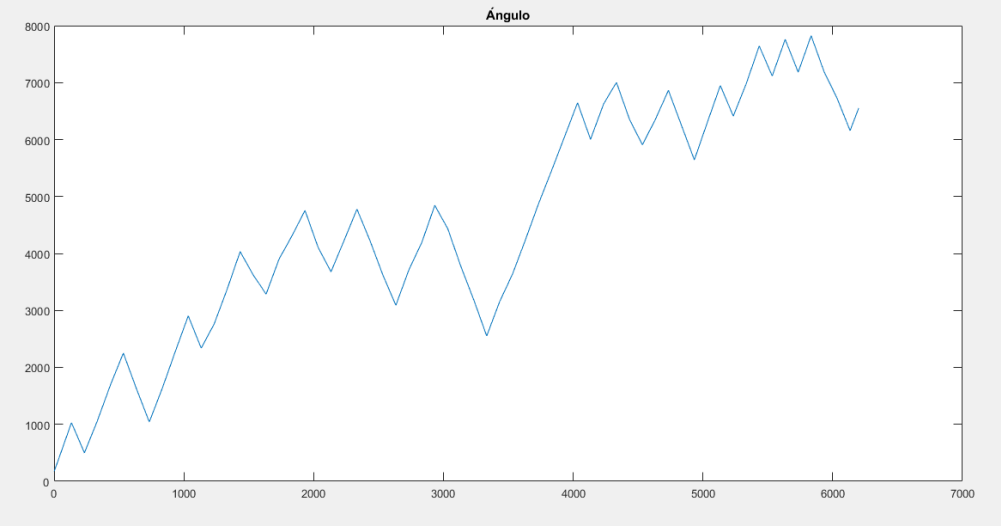


Figura 95: Posición angular, segundo ensayo

Poco más que añadir, de nuevo volvemos a utilizar la herramienta de Matlab que nos permite identificar el sistema, así que damos paso a las figuras que nos muestran los resultados de esta segunda prueba:

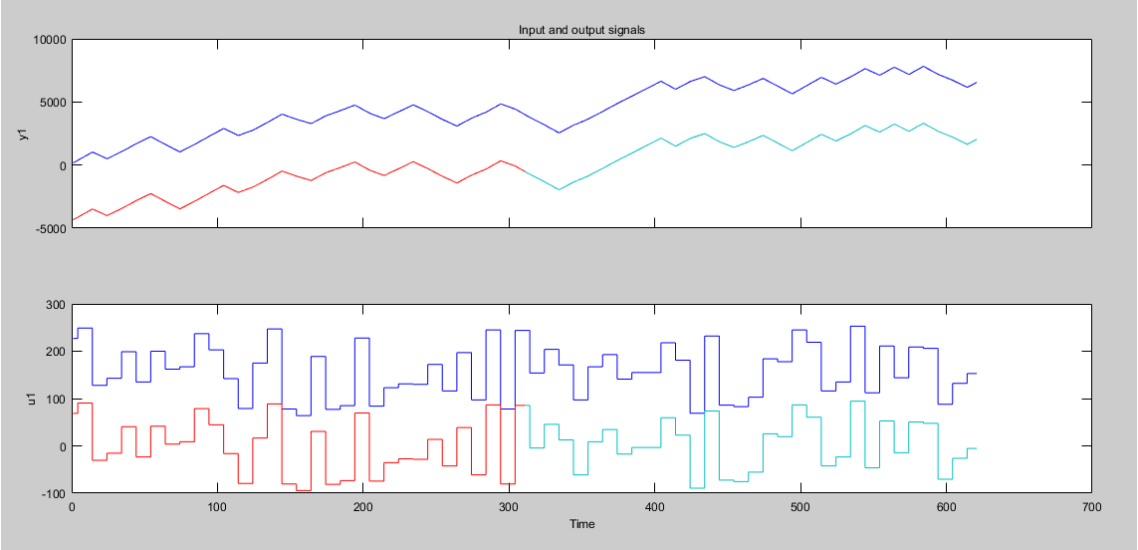


Figura 96:Datos de entrada y salida, segundo ensayo

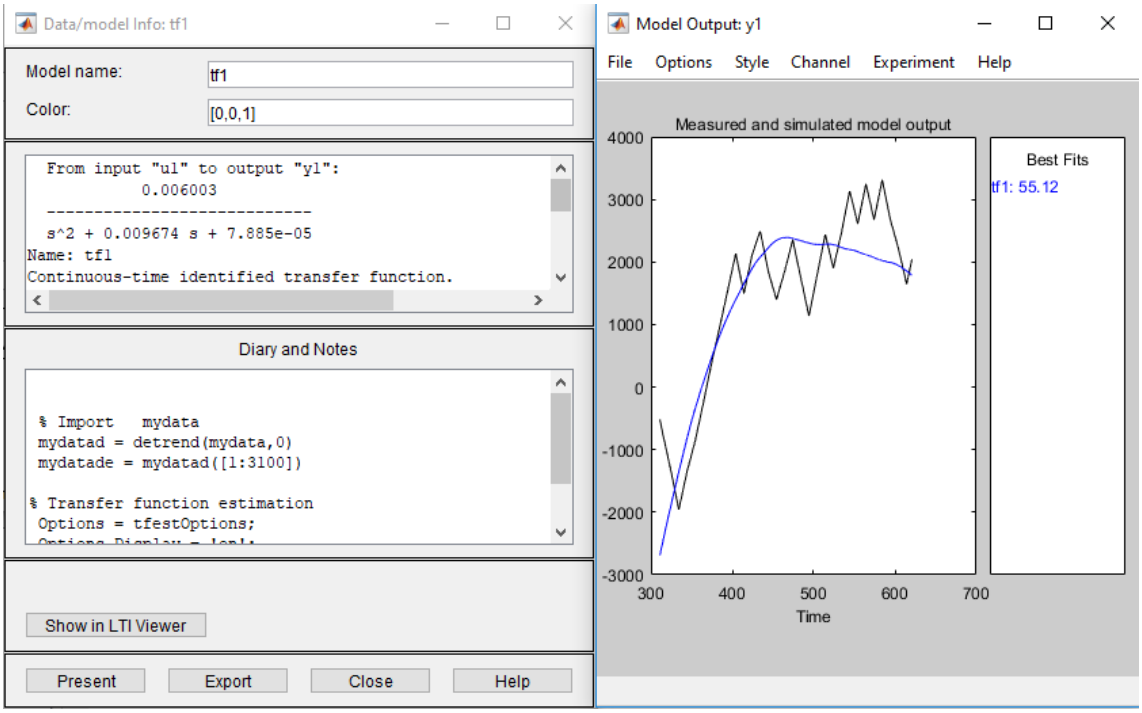


Figura 97: Estimación de modelos función de transferencia, segundo ensayo

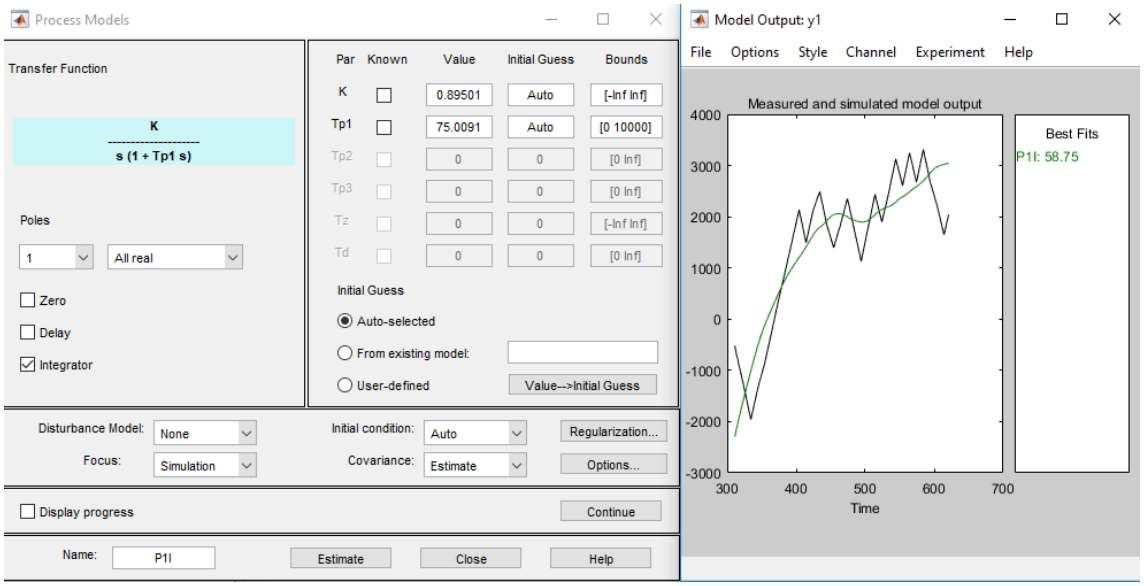


Figura 98: Estimación de modelo del proceso, segundo ensayo

Como se observa, hay una notable mejoría con el único cambio de la duración de los escalones de PWM, aun así siguen saliendo constantes de tiempo bastante grandes, hay que seguir pensando en otras posibilidades para obtener un buen resultado.

Tercer ensayo

Este último ensayo contempla varios cambios respecto a sus predecesores. Hasta ahora habíamos intentado obtener una función de transferencia de primer orden más un integrador, pero también existía la posibilidad de derivar la posición angular (con la función “diff” de Matlab) e intentar obtener una función de transferencia de primer orden, así que ese es el siguiente paso:

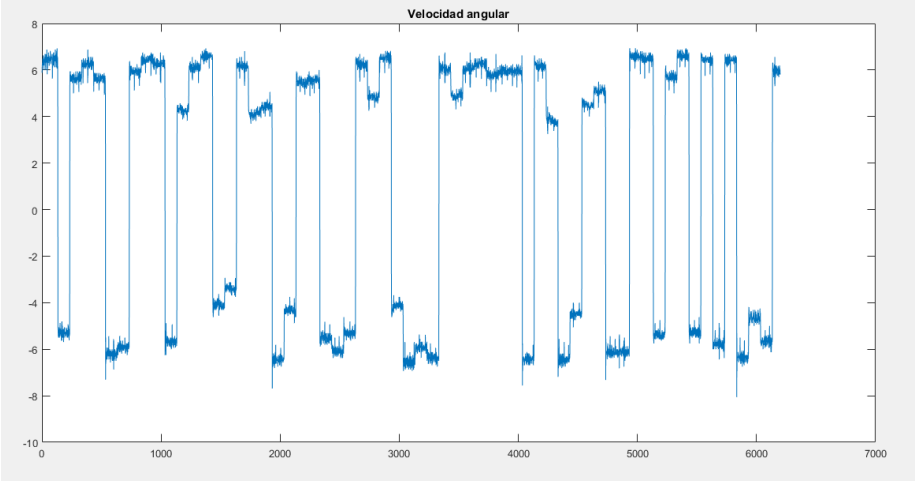


Figura 99: Velocidad angular, tercer ensayo

Una vez obtenida la velocidad angular, de nuevo probamos la identificación:

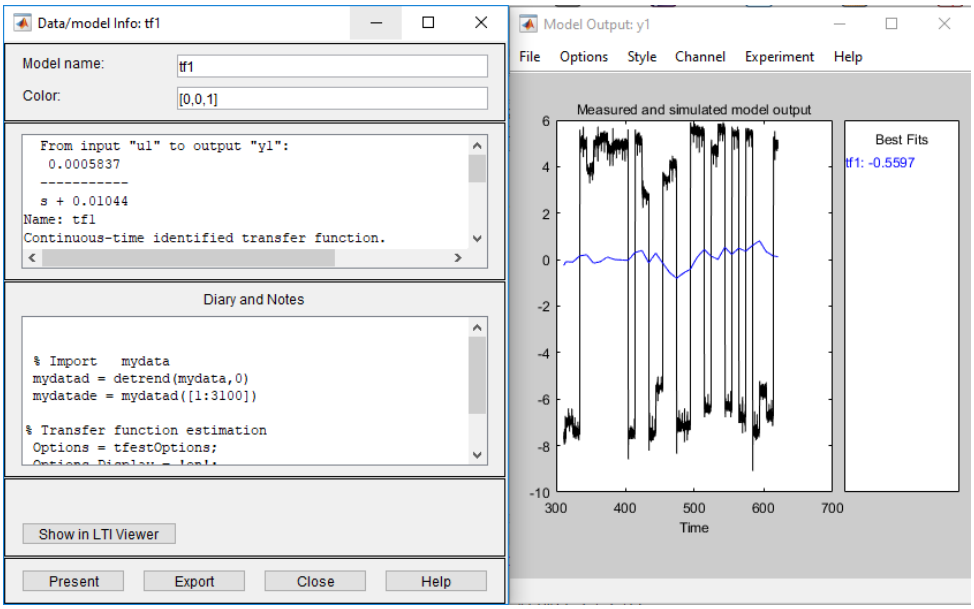


Figura 100: Estimación de modelos función de transferencia, tercer ensayo

Aunque la identificación deje mucho que desear, se aprecia un cambio importante: empieza a aparecer un polo, cosa que antes no ocurría.

Después de esta prueba nos dimos cuenta de varias cosas: la primera es que el dato de la velocidad angular tenía mucho ruido, con lo cual la aplicación de un filtro pasa baja podría ser una buena idea para tener una mejor aproximación; la segunda es que el hecho de que el robot fuese hacia delante y hacia atrás hacía aparecer velocidades positivas y negativas, lo cual podría complicar la identificación. Así que, teniendo en cuenta que el robot lleva la misma velocidad cuando se mueve hacia delante y cuando se mueve hacia detrás, decidimos que ponerlo solo en un sentido sería lo más adecuado.

```
dangulo = diff(angulo);
filtro = zpk([],[-10 -10],100); %filtro pasa baja
fdangulo = lsim(c2d(filtro,0.1), dangulo); %aplicar el filtro
```

Cuadro de código 11: Derivada y filtrado de la posición angular

Este código representa lo comentado en el párrafo anterior: en primer lugar, utilizamos la función “diff” para aplicar la derivada a nuestra posición angular, en segundo creamos un filtro pasa baja de ganancia uno, cuya función de transferencia pondremos a continuación, y por último, aplicamos dicho filtro a la velocidad angular.

$$G_{filtro}(s) = \frac{100}{(s+10)^2}$$

El resultado tras aplicar el filtro a la velocidad es el siguiente:

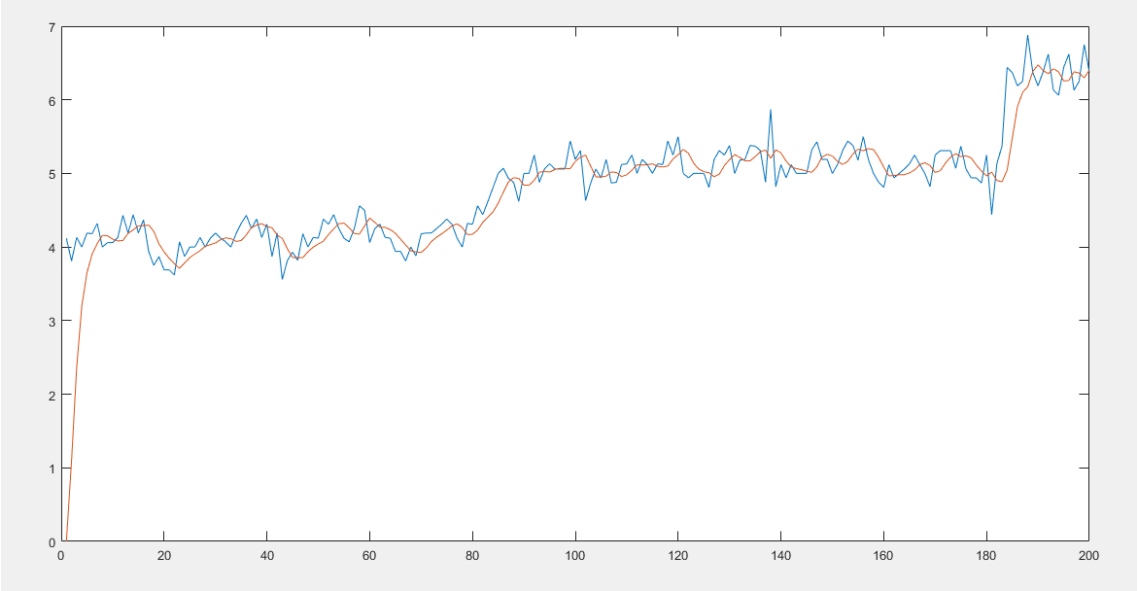


Figura 101: Velocidad angular y velocidad angular filtrada, tercer ensayo

En azul vemos la velocidad angular y en naranja el resultado tras aplicarle el filtro pasa baja. El ruido se reduce, así que, a priori, los resultados deberían mejorar. Vemos a continuación los resultados con la Toolbox de Matlab:

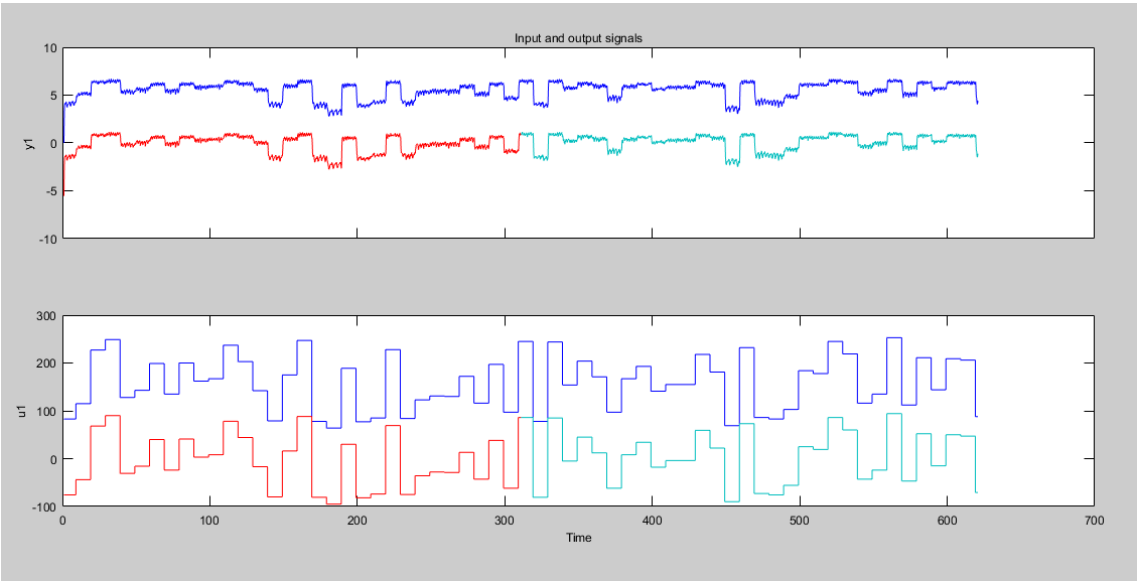


Figura 102: Datos de entrada y salida, tercer ensayo

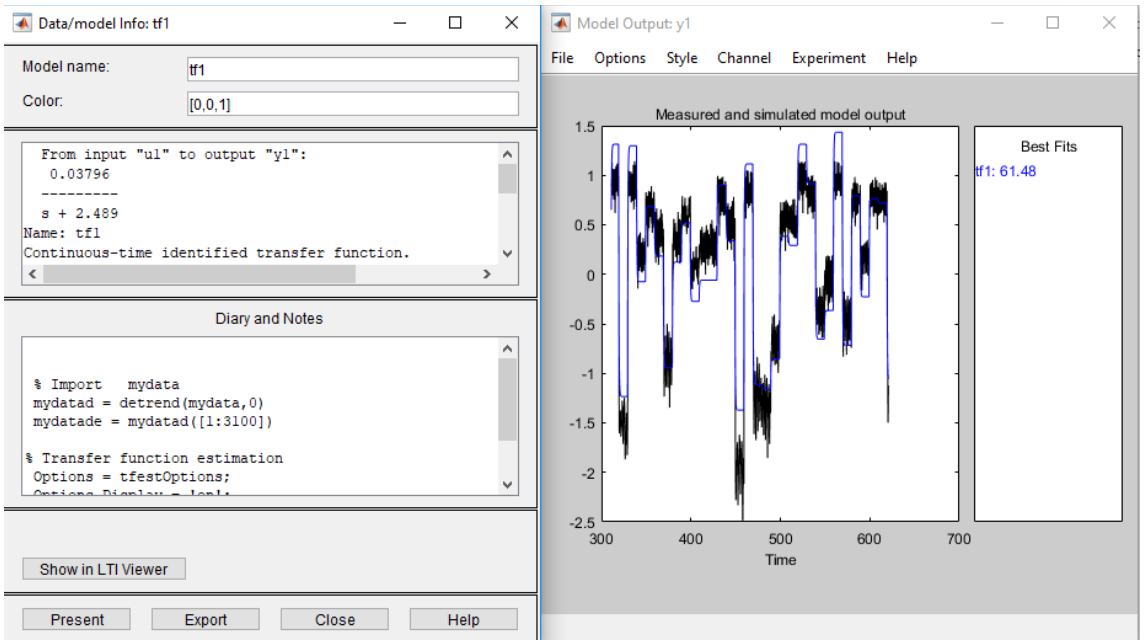


Figura 103: Estimación de modelos función de transferencia, tercer ensayo

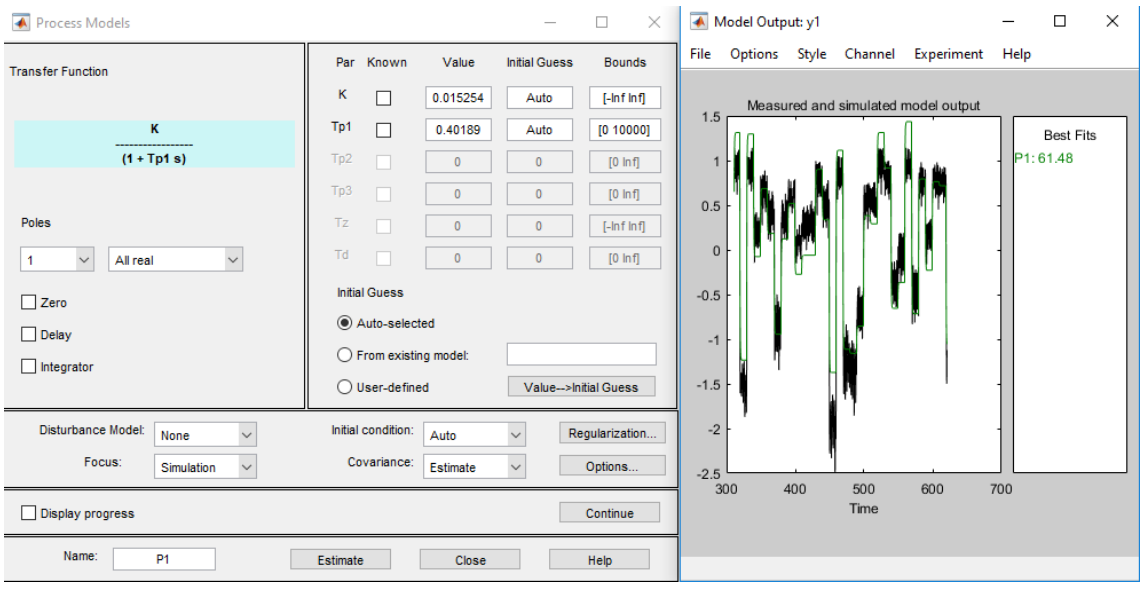


Figura 104: Estimación de modelo del proceso, tercer ensayo

Es la primera vez que obteníamos un polo con algo de sentido físico, y además el mismo polo con dos estimaciones distintas. La única pega es que la ganancia no tiene mucho sentido, pero podríamos justificarlo ya que al tener ruido y aplicar un filtro (aunque de ganancia uno), estamos introduciendo ciertos elementos que podrían alterar este valor.

Así que se llegó a la conclusión que lo ideal sería quedarnos con el polo que nos proporcionaba la Toolbox y calcular la ganancia manualmente gracias a las gráficas, con lo cual eso fue lo que se hizo.

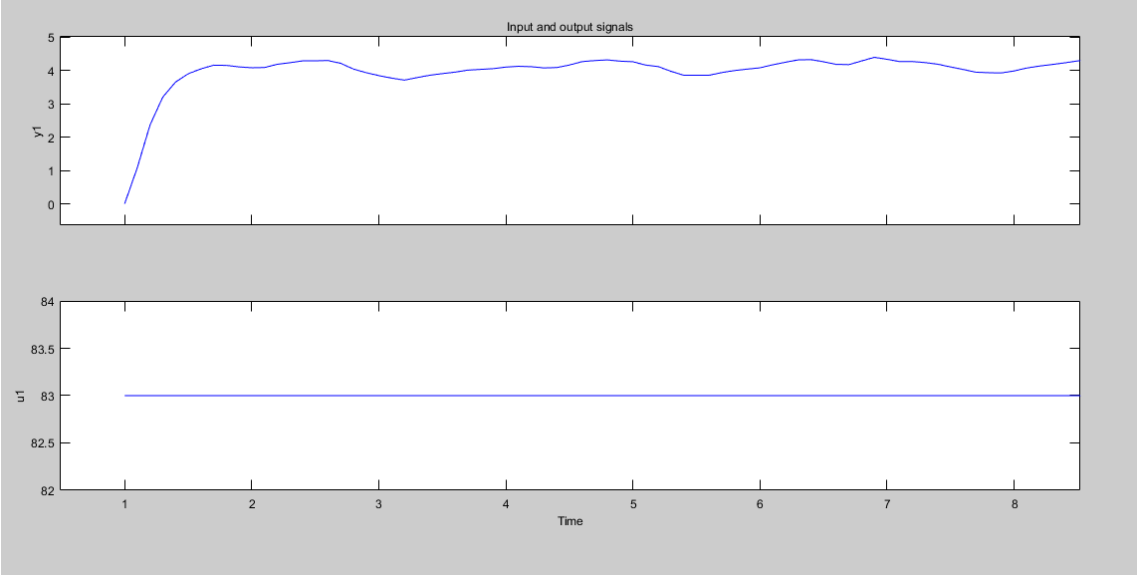


Figura 105: Datos de entrada, tercer ensayo

Sabemos que para $t = \tau$ la ganancia toma el 63% de su valor, con lo cual:

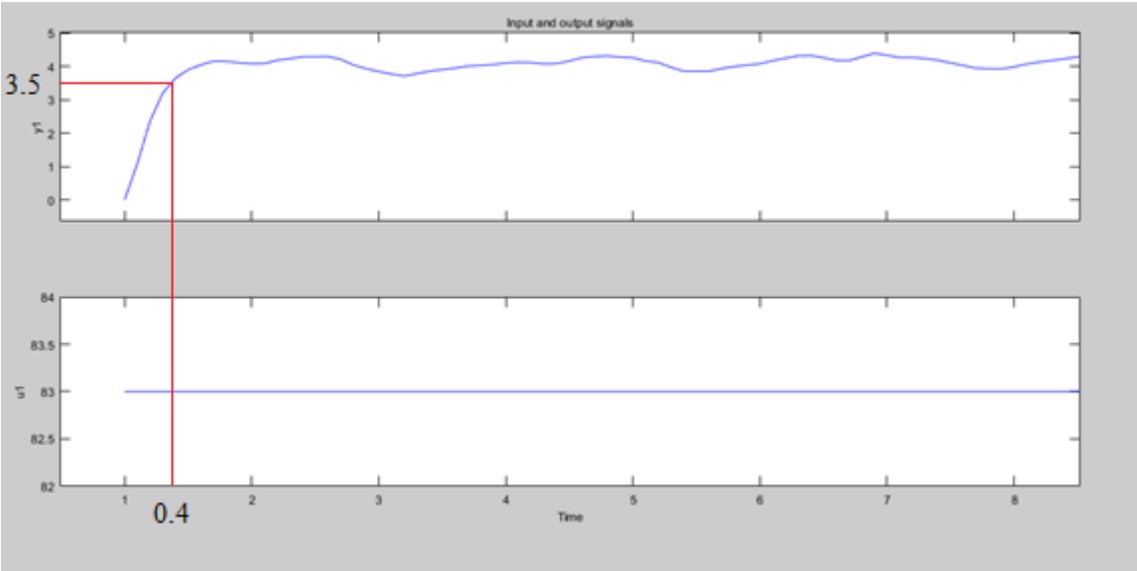


Figura 106: Obtención de la ganancia

Para concluir, podemos decir que, aproximadamente, la función de transferencia es:

$$G_{motor\ A}(s) = \frac{5.56}{s + 2.5}$$

De la misma forma podemos aplicar el mismo procedimiento para obtener la función de transferencia del otro motor:

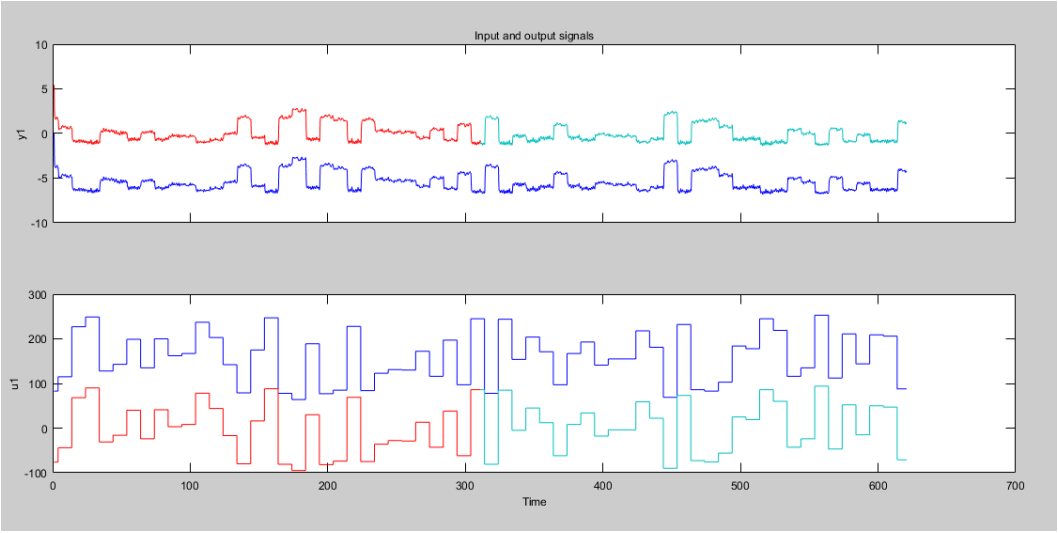


Figura 107: Datos de entrada y salida, tercer ensayo (2)

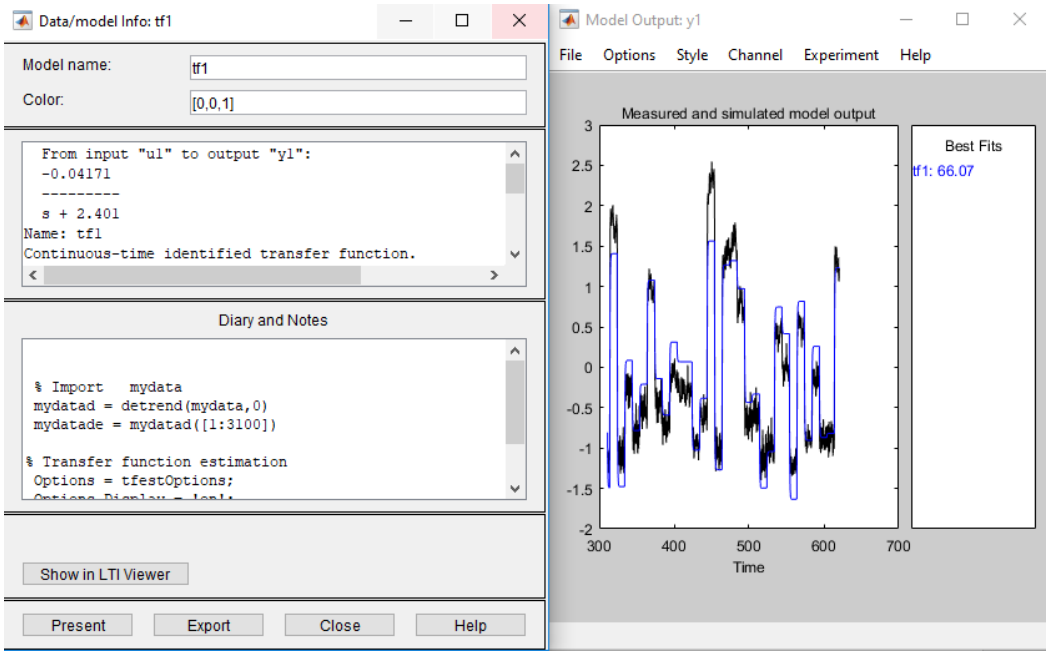


Figura 108: Estimación de modelos función de transferencia, tercer ensayo (2)

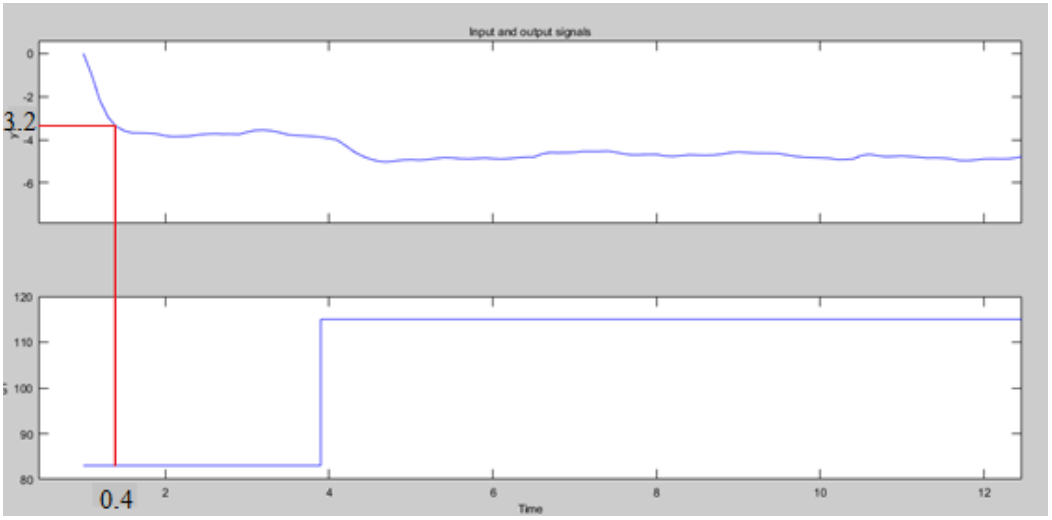


Figura 109: Obtención de la ganancia (2)

$$G_{motor\ B}(s) = \frac{5.08}{s + 2.4}$$

Hay que aclarar que el cálculo de la ganancia ha sido medido con precisión en la propia gráfica, aumentando el tamaño de la misma. Se han omitido dichas figuras por no contemplar el sistema de primer orden que se obtiene, y que si podemos observar en las figuras 106 y 109.

Además, hay que comentar que esta rueda giraba en sentido contrario a la otra, por eso la Toolbox y la propia gráfica muestran ganancias negativas, pero tan solo tenemos que conocer el significado de estos datos e interpretarlos de manera correcta.

6.2.4. Limitaciones y mejoras de la identificación

Como ya hemos comentado, la falta de un tacómetro nos ha limitado mucho esta tarea. De hecho, con los ensayos realizados podemos observar como las velocidades angulares máximas de cada rueda son, aproximadamente, unos 7 rad/s, lo cual se traduce en 14 cm/s de velocidad lineal.

En otro experimento con el sensor de ultrasonidos (el cual comentaremos más adelante), hemos comprobado como la velocidad máxima a la que va el robot es de 9-10 cm/s, con lo cual, ya existe un error en la medida de velocidad, si añadimos además el error provocado por el filtro pasa-baja, podemos darnos cuenta de que la identificación se podría mejorar si contásemos con un tacómetro o encoder para poder controlar a la perfección las variables de velocidad y posición angular de los motores.

7. SIMULACIÓN DEL MODELO EN EJSS Y SIMULINK

En este capítulo vamos a tratar la simulación de nuestro robot móvil en diversas plataformas. Vamos a visualizar como varía la posición del robot frente a los datos de entrada (valores de PWM) a cada uno de los motores gracias a las ecuaciones de modelado que obtuvimos en el capítulo cinco.

7.1. EJSS

Antes de mostrar el código de cada uno de los paneles y la interfaz gráfica, es necesario aclarar que las ecuaciones de modelado del capítulo cinco van a ser distintas a las que mostraremos en la programación de EJSS. Esto es debido a que EJSS toma un sistema de referencia distinto al nuestro. A continuación, explicaremos, con la ayuda de una figura, como hemos obtenido las nuevas ecuaciones para que el robot se mueva adecuadamente en la simulación:

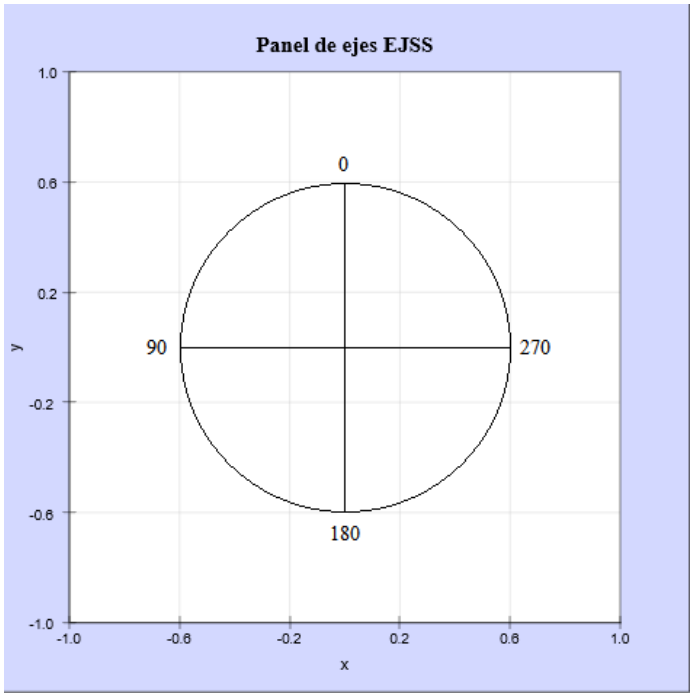


Figura 110: Sistema de referencia de EJSS

Como vemos, la referencia que toma EJSS es diferente a la habitual. El principal problema a la que conlleva esta referencia, es que los signos de las operaciones matemáticas seno y coseno no concuerdan con el propio esquema anterior. Voy a exponer un caso práctico para que se entienda mejor, partiendo de las ecuaciones del capítulo cinco y ayudándonos de otra figura:

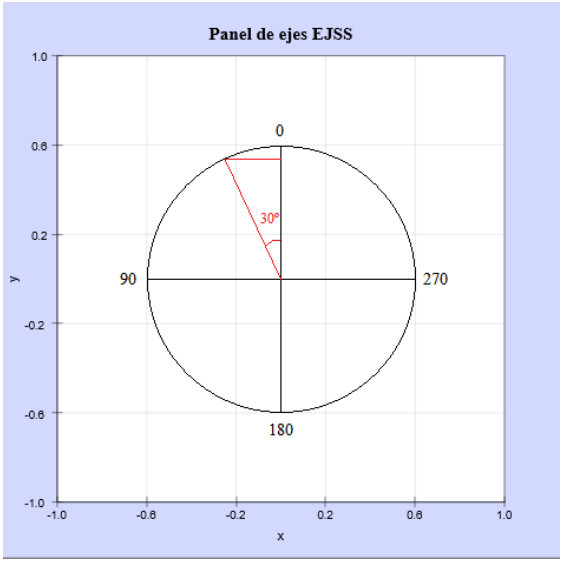


Figura 111: Ejemplo práctico para explicar modelado en EJSS

$$x' = x + v_o * Cos(\theta) * t$$

$$y' = y + v_o * Sen(\theta) * t$$

Imaginemos la situación anterior: comenzamos en un punto inicial ***O* (0,0)** y queremos desplazarnos en una orientación de 30º a una cierta velocidad lineal **v_0**. Según las ecuaciones del capítulo cinco, las nuevas posiciones serían:

$$x' = 0 + 10 * Cos(30) * 0.1 = 0.866$$

$$y' = 0 + 10 * Sen(30) * 0.1 = 0.5$$

Vemos como, aparte de ir en otra dirección, la distancia que avanza en cada eje no concuerda con lo que en realidad queremos. Debido al sistema de coordenadas el avance correspondiente al eje X se va a ver influenciado por el seno del ángulo, mientras que el avance en el eje Y va a depender del coseno del ángulo. De esta forma podemos reescribir las ecuaciones que utilizaremos en EJSS:

$$x' = x \pm v_o * Sen(\theta) * t$$

$$y' = y \pm v_o * Cos(\theta) * t$$

El signo de suma o resta dependerá del cuadrante en el que estemos. Por ejemplo, en el primer cuadrante las ecuaciones tomarían la forma:

$$x' = x - v_o * Sen(\theta) * t$$

$$y' = y + v_o * Cos(\theta) * t$$

En dicho cuadrante las operaciones seno y coseno son positivas, así que a la ecuación correspondiente al eje X le restamos el término del seno (para que vaya hacia la izquierda) y en cuanto a la del eje Y, le sumamos el término del coseno (para suba hacia arriba). Las ecuaciones cambiarán para cada cuadrante, teniendo en cuenta para donde se tiene que desplazar el robot y el signo del seno y coseno de los ángulos en cada cuadrante.

7.1.1. Variables

Para mostrar todas las variables que hemos utilizado, vamos a crear una tabla que indique el nombre (y una breve descripción si es necesario), valor inicial y tipo (String, Int, Double, Boolean).

Variable	**Valor inicial**	**Tipo**
t	0	double
dt	0.05	double
pwmA	0	double
pwmB	0	double
wmotorA (vel angular)	0	double
wmotorB (vel angular)	0	double
vmotorA (vel lineal)	0	double
vmotorB (vel lineal)	0	double
alpha (orientación)	0	double
vrobot (vel lineal del robot)	0	double
wrobot (vel ang del robot)	0	double
posx	0	double
posy	0	double
posxanterior	0	double
posyanterior	0	double
alphaanterior	0	double
radio (de la rueda)	0	double
base (dist entre ruedas)	0	double
kp	5	double
ki	0	double
kd	0	double
error	0	double
derror	0	double
ierror	0	double
referencia	0	double
k	0	double
velocidad	0	double

7.1.2. Relaciones fijas

Como ya comentamos en el capítulo tres, en esta pestaña vamos a establecer las relaciones que existen entre las variables. Dichas relaciones no son más que las ecuaciones de la cinemática del robot.

```
wmotorA=pwmA/51;
wmotorB=pwmB/51;

vmotorA=wmotorA*radio;
vmotorB=wmotorB*radio;

vrobot=(vmotorA+vmotorB)/2;

wrobot=(wmotorA-wmotorB)/base;
```

Cuadro de código 12: relaciones fijas

La única que conviene mencionar es la de velocidad angular de cada uno de los motores, ya que hay que establecer una relación entre el PWM y los valores de velocidad angular para conseguir que la simulación se ejecute correctamente.

Para ello, llevamos a cabo un ensayo con el robot acercándose a una pared a máxima velocidad y gracias al sensor de ultrasonidos vamos a poder calcular la velocidad lineal a la que va con un valor máximo de PWM. En este ensayo necesitamos la ayuda de nuestro compañero Rubén Maestro, ya que tuvimos que implementar un control para que el robot fuera en línea recta.

Aunque el cálculo de la velocidad con el sensor de ultrasonidos dejó mucho que desear (ya que había mucho ruido en los datos obtenidos), pudimos obtener que la máxima velocidad era, aproximadamente, de unos 10 cm/s. Así que tan solo teníamos que aplicar una serie de proporciones para obtener la relación entre las velocidades angulares y los valores de PWM de cada motor.

Además del código del cuadro anterior, hay otra parte que he obviado, ya que lo único que hace es que algunas variables no sobrepasen ciertos valores (por ejemplo, que no podamos introducir un valor de velocidad superior al máximo).

7.1.3. Propio

En este apartado vamos a comentar algunas funciones que hemos diseñado para facilitar la simulación. Todo el código de dichas funciones se podría haber implementado directamente en la pestaña de evolución, pero por simplificación y orden se ha decidido hacerlo así.

```
function saturacionPWM(){
  if (pwmA>255){
    pwmA=255;
  }
  if (pwmB>255){
    pwmB=255;
  }
  if (pwmA<0){
    pwmA=0;
  }
  if (pwmB<0){
    pwmB=0;
  }
}
```

Cuadro de código 13: Función de saturación de PWM

Como su propio nombre indica, controlamos que el valor de PWM no sobrepase el valor máximo que podemos darle en el sistema físico real.

```
function controlador(){
  error=referencia-alpha;
  k=kp*error+kd*derror+ki*ierror;
  pwmA=velocidad/100*(51/0.02);
  pwmB=velocidad/100*(51/0.02);
  pwmA=pwmA+k;
  pwmB=pwmB-k;
}
```

Cuadro de código 14: Función del controlador

Esta función se carga de obtener el error, aplicar la acción de nuestro controlador y variar los valores de PWM de cada motor en consecuencia para llegar a las referencias de velocidad y posición angular que le indiquemos.

7.1.4. Evolución

Como última parte del apartado de modelo: la pestaña de evolución, donde escribimos las líneas de código y las ecuaciones diferenciales que rigen nuestro sistema. Empezaremos por las ecuaciones diferenciales que hemos utilizado:

$$\frac{d(ierror)}{dt} = error \qquad [1]$$

$$\frac{d(error)}{dt} = derror \qquad [2]$$

Gracias a las páginas que nos permiten implementar ecuaciones diferenciales, podemos obtener de una manera muy sencilla la integral del error y la derivada del mismo, tal que podamos usar estos datos para el controlador PID.

A continuación, mostramos un cuadro de código con la página de programación de nuestro sistema:

```
if(alpha>=0 && alpha<90){
posx= posxanterior - vrobot*dt*Math.sin(alpha*Math.PI/180);
posy= posyanterior + vrobot*dt*Math.cos(alpha*Math.PI/180);
alpha=alphaanterior + wrobot*dt;
}

posxanterior=posx;
posyanterior=posy;
alphaanterior=alpha;

//CONTROL
controlador();
saturacionPWM();
```

Cuadro de código 15: Código de evolución del sistema

En primer lugar, hay que aclarar que tan solo hemos puesto las ecuaciones cinemáticas correspondientes al primer cuadrante, el resto son iguales cambiando el signo correspondiente al término de las operaciones seno y coseno.

Después de eso, guardamos las nuevas posiciones en los ejes X e Y, así como la posición angular, para la siguiente iteración. Finalmente llamamos a la función del controlador y posteriormente a la de saturación de PWM para controlar el movimiento del robot según nuestras referencias.

7.1.5. Interfaz gráfica

Hemos diseñado una interfaz que dispone de un panel superior en el que podemos iniciar, pausar y reiniciar la simulación. Además, hay varios campos numéricos donde podemos introducir nuestras referencias de posición y velocidad, y otros donde podemos ver los valores numéricos de los valores que están tomando algunas variables como PWM de cada motor u orientación. La mostramos en la siguiente figura:

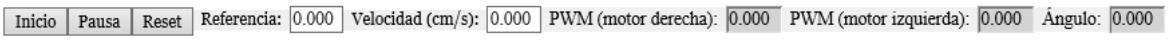


Figura 112: Panel superior de la interfaz

Dicha interfaz cuenta además con cinco gráficas en las que podemos ver la trayectoria seguida y la posición actual del robot, curvas de representación de velocidad y orientación del robot frente a las referencias y otras en las que vemos las velocidades angulares y el valor de PWM de cada uno de los motores:

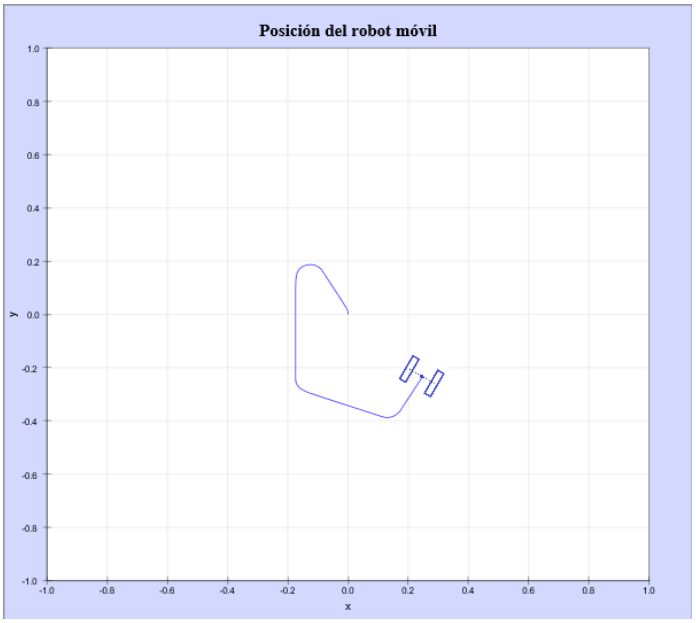


Figura 113: Posición y trayectoria del robot

En esta gráfica podemos ver como se mueve el robot cuando cambiamos las referencias de posición y velocidad, además de toda la trayectoria que ha seguido el mismo.

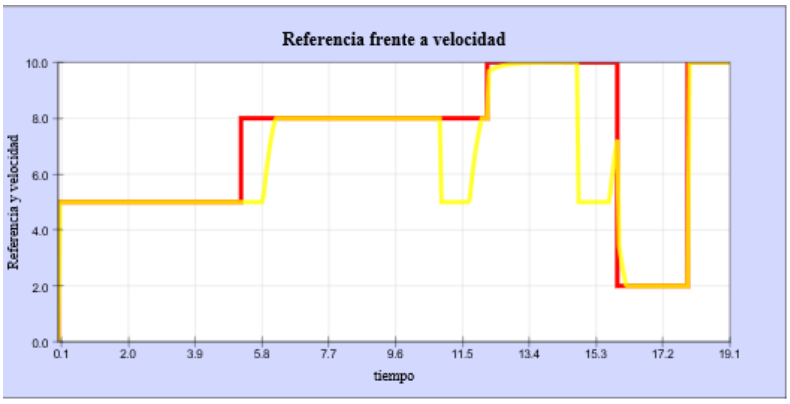


Figura 114: Referencia frente a velocidad del robot

Aquí vemos representado en rojo la referencia de velocidad en cm/s y en amarillo la velocidad lineal del robot en cada momento.

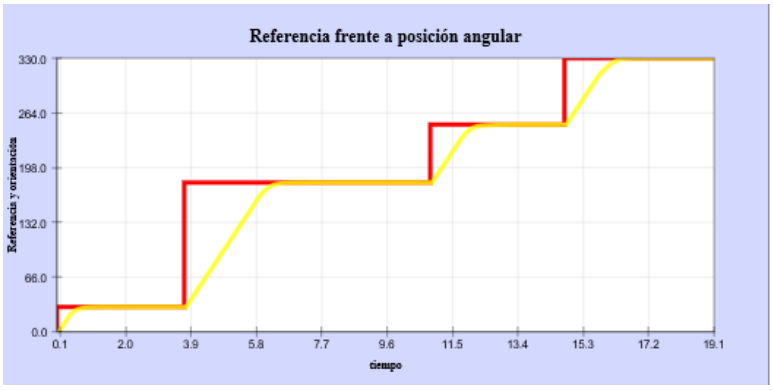


Figura 115: Referencia frente a posición angular del robot

Al igual que en la figura anterior, mostramos una referencia frente a cómo va variando dicha característica del robot hasta llegar a ella, pero en este caso es la orientación.

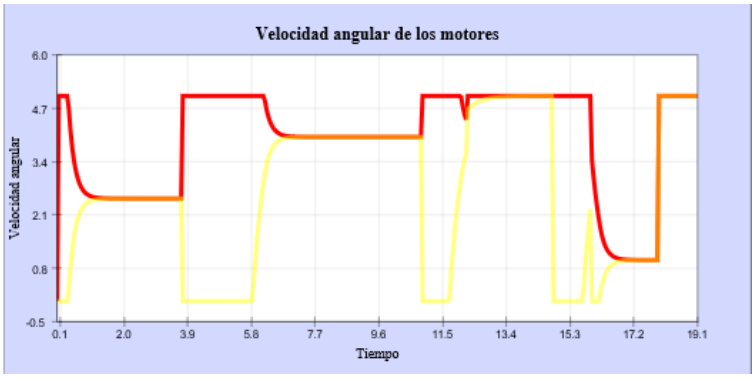


Figura 116: Velocidad angular de los motores

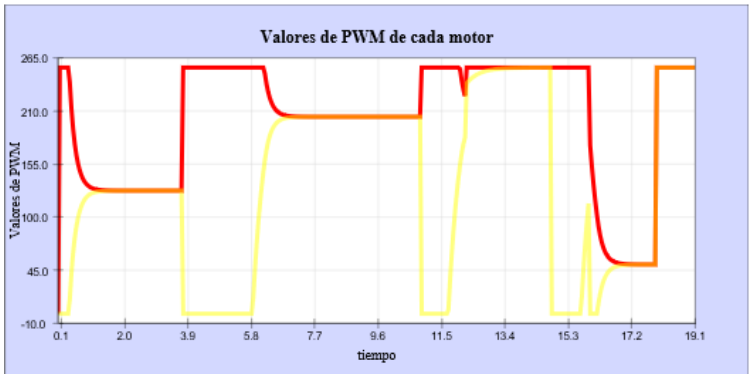


Figura 117: PWM de ambos motores

En estas dos últimas gráficas representamos la velocidad angular y valores de PWM de ambos motores.

7.1.6. Posibles mejoras que implementar

Una de las diferencias más notorias entre el sistema real y el simulado, en cuanto al control se refiere, es que en este último tan solo hace falta una acción proporcional en el controlador para que alcance, sin error, los valores de velocidad y posición que deseemos. Por otro lado, en el sistema real es necesaria la implementación de un controlador con las acciones proporcional e integral para conseguir buenos resultados.

Esto se debe a que en la simulación no hemos incluido las perturbaciones que influyen en el sistema real. Por ejemplo, que recae más peso sobre una rueda que sobre la otra (debido a la disposición del hardware), el rozamiento de las ruedas sobre el terreno puede provocar leves variaciones del comportamiento ideal, etc.

Una posible mejora de la simulación podría haber sido la inclusión de estas perturbaciones, de tal forma que hubiéramos podido observar un comportamiento no ideal en la simulación, y por tanto forzar la incorporación de la acción integral al controlador para llegar a las referencias.

7.2. Simulink

En esta sección vamos a mostrar todo lo concerniente a la simulación en el entorno de Simulink. Para ello nos ayudaremos de figuras que contienen todos los diagramas de bloques que hemos realizado para poder incluir la cinemática y el control de nuestro sistema. Algunos de estos bloques son funciones del propio Matlab, así que también añadiremos cuadros de código con algunas de estas funciones. Empezaremos por el diagrama de bloques que engloba al sistema y después iremos analizando cada parte.

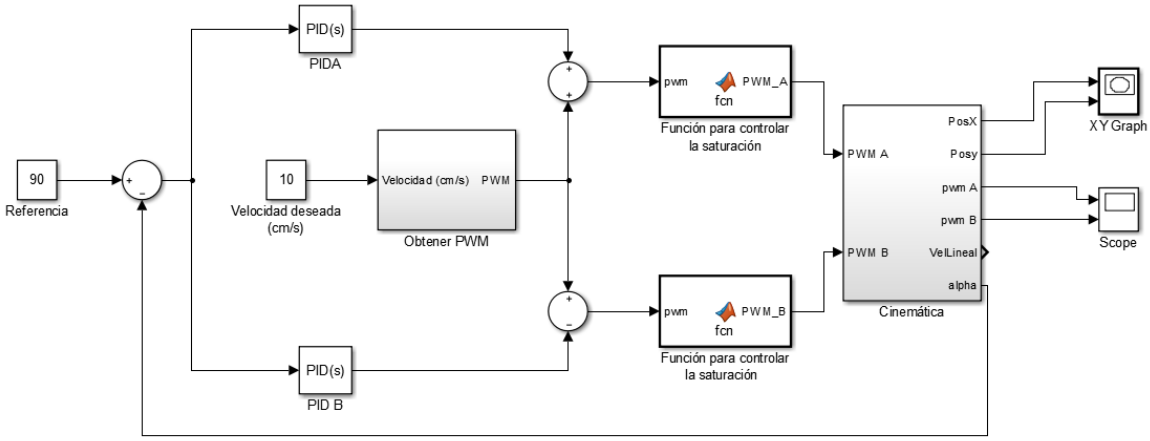


Figura 118: Diagrama de bloques del sistema

Este es el diagrama de bloques del sistema completo, en el que podemos observar: cómo introducimos los valores de referencia angular y la velocidad deseada, los controladores de ambos motores, un par de funciones para controlar el valor de PWM y dos subsistemas: uno correspondiente a la cinemática y otro de conseguir el valor de PWM a partir de la velocidad.

Empezaremos mostrando los valores del controlador PID. Al igual que en EJSS, en esta simulación no hemos tenido en cuenta las perturbaciones del sistema real, con lo cual tan solo ha hecho falta la acción proporcional para llevar a cabo el control del robot.

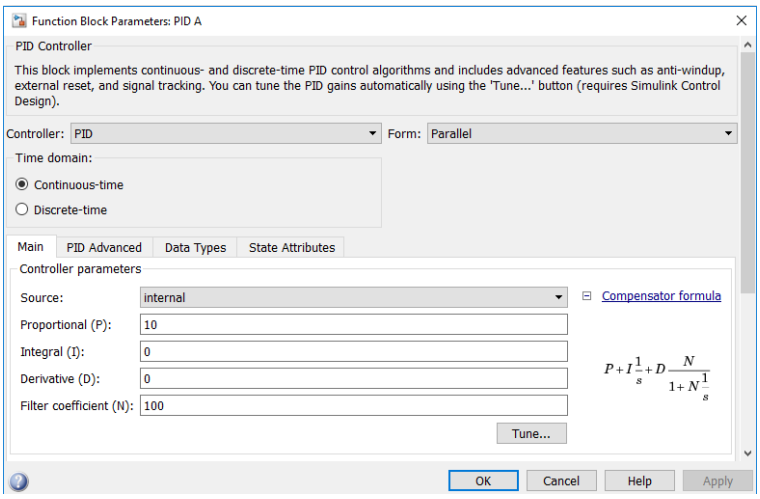


Figura 119: Controlador PID

A continuación, mostramos el subsistema de “Obtener PWM”:

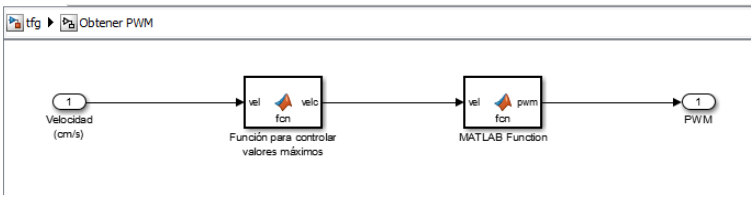


Figura 120: Subsistema "Obtener PWM"

Como vemos se trata de dos funciones de Matlab: la primera se encarga de controlar los valores mínimos y máximos de velocidad que podamos aplicar, la segunda obtiene la relación entre la velocidad y el valor de PWM que debemos aplicar. Esta última función es exactamente igual a la mencionada anteriormente en el entorno de EJSS.

```
function pwm = fcn(vel)

pwm = vel*51/0.02;
```

Cuadro de código 16: Relación Velocidad-PWM

La siguiente función que vemos, siguiendo el diagrama de bloques, se corresponde con la de control de saturación de los valores de PWM. En definitiva, se resume en unas pocas líneas que controlan que esté dentro de los márgenes de 0 a 255.

```
function PWM_A = fcn(pwm)
if (pwm>255)
    pwm=255;
end
if(pwm<0)
    pwm=0;
end
PWM_A = pwm;
```

Cuadro de código 17: Función de saturación de PWM

El siguiente subsistema es el correspondiente a la cinemática. Es un diagrama de bloques que se ciñe, paso a paso, a la cinemática explicada en el capítulo cinco. En las siguientes dos figuras veremos el subsistema entero y lo iremos analizando de principio a fin.

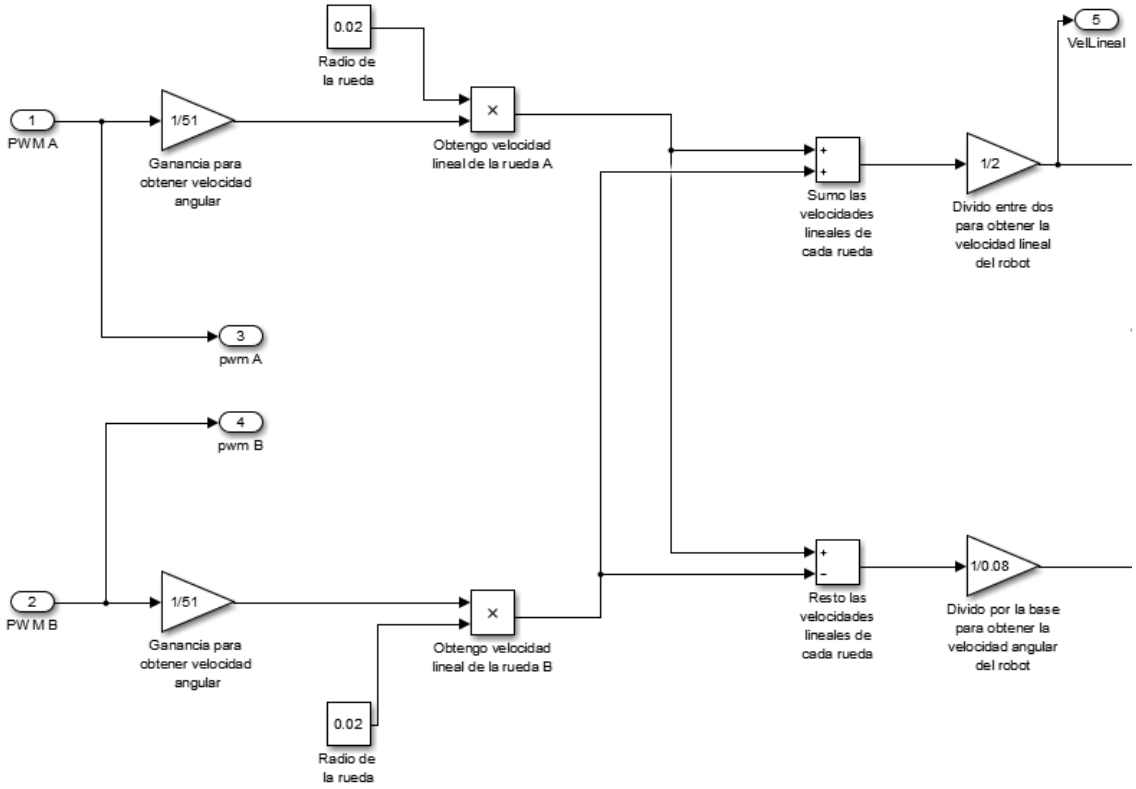


Figura 121: Bloque de cinemática 1

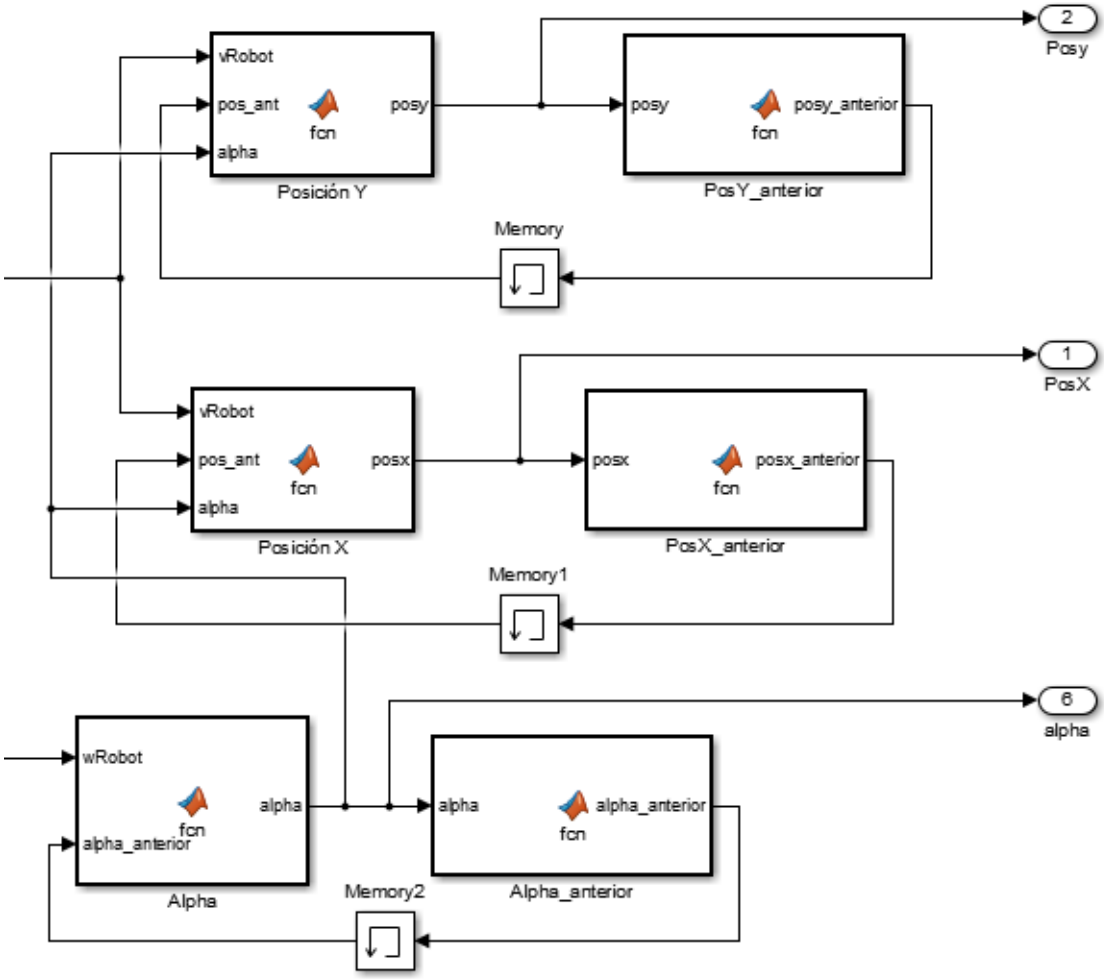


Figura 122: Bloque de cinemática 2

Hay que comenzar aclarando que hemos intentado ser muy descriptivos e ir paso a paso, por ejemplo, en la figura 121 vemos como a partir de los valores de PWM vamos obteniendo, en orden, velocidad angular y luego velocidad lineal de cada rueda, para luego poder saber las mismas del robot. Aun pudiendo haber hecho esas operaciones con una simple función de Matlab, hemos preferido mostrarlo así para que fuera lo más representativo posible de nuestra cinemática.

En cuanto a la figura 122, tan solo se trata de funciones que contienen las ecuaciones posición y orientación:

```
function posy = fcn(vRobot,pos_ant,alpha)
posy = pos_ant + vRobot*sin(alpha*pi/180)*0.01;

function posx = fcn(vRobot,pos_ant,alpha)
posx = pos_ant + vRobot*cos(alpha*pi/180)*0.01;

function alpha = fcn(wRobot,alpha_anterior)
alpha = alpha_anterior + wRobot*0.01;
```

Cuadro de código 18: Ecuaciones de posición y orientación

Por otro lado, cabe mencionar que Simulink no nos permitía hacer una realimentación tal que así:

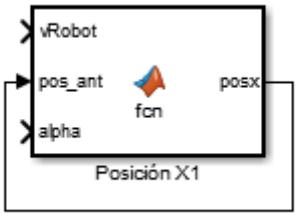


Figura 123: Realimentación

Es por ese motivo que incluimos otra función y añadimos un bloque de memoria para poder guardar el dato de posición u orientación y volverlo a utilizar en la siguiente iteración.

Después de haber comentado todos los diagramas de bloques, solo queda mostrar las gráficas que hemos visto en la sección anterior de EJSS:

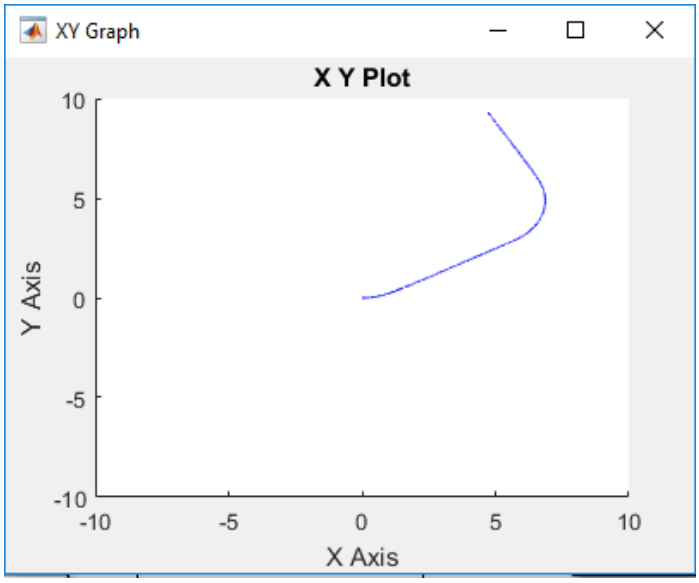


Figura 124: Posición del robot

En esta primera figura podemos ver la posición del robot. Hay que decir que Simulink tan solo nos permite representar un cierto número de puntos, cuya cantidad no podemos modificar. Si la simulación es demasiado extensa, irán desapareciendo las primeras posiciones de la gráfica.

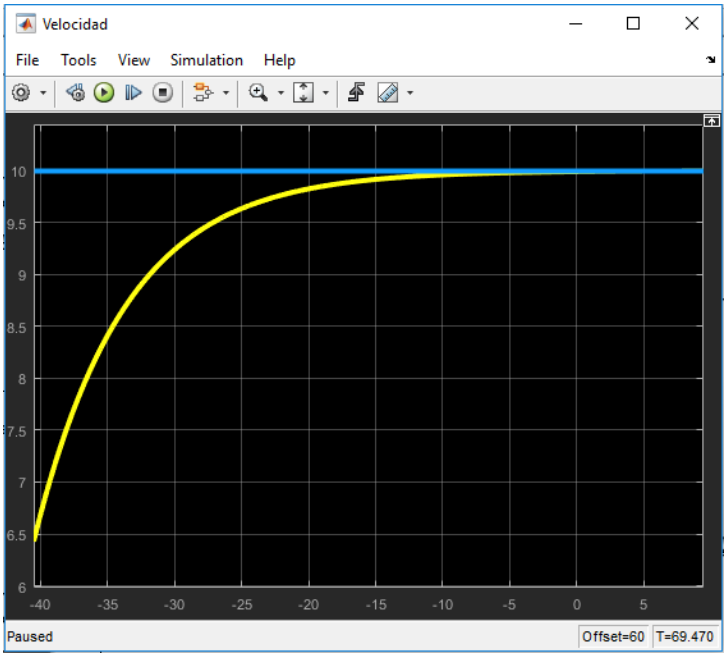


Figura 125: Curva de velocidad de referencia y robot

En esta gráfica vemos representada la referencia de velocidad en azul y la velocidad lineal del robot en amarillo.

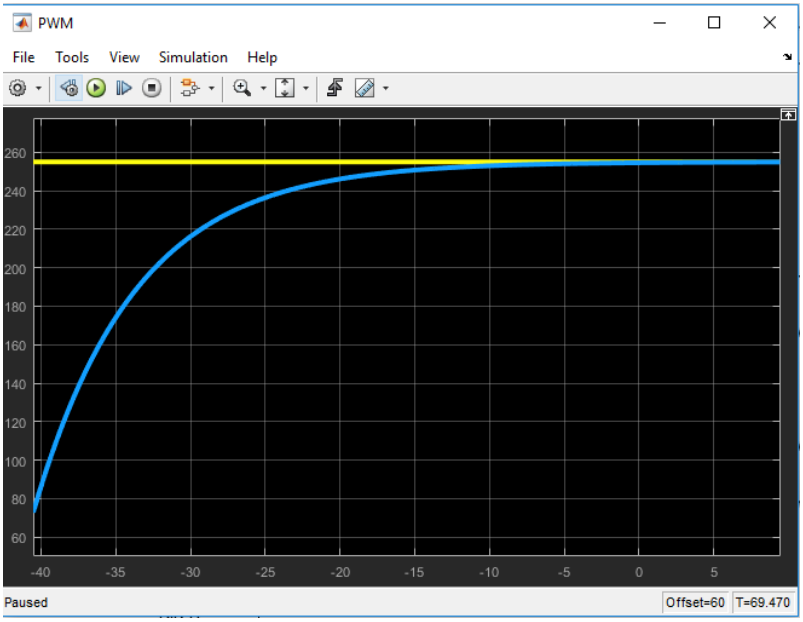


Figura 126: Curva de PWM de ambos motores

En esta otra observamos como varían los PWM hasta alcanzar el mismo valor.

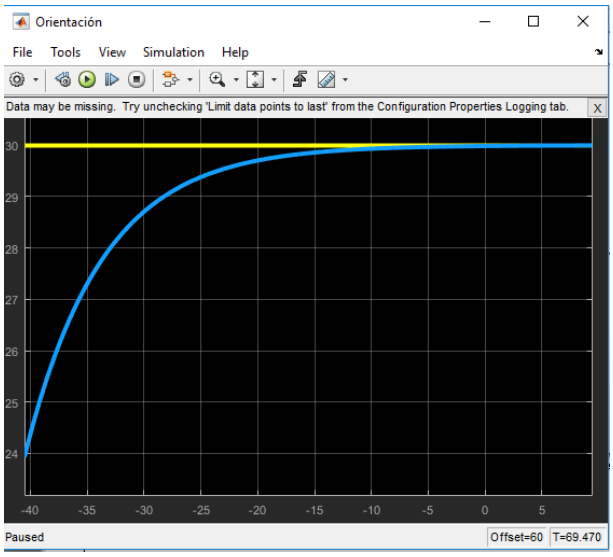


Figura 127: Curva de referencia y posición angular del robot

En esta última podemos ver en amarillo la referencia de posición angular (30º) y en azul como varía la posición angular del robot hasta conseguirla.

Cabe mencionar que todas estas figuras provienen del mismo movimiento (hacer girar al robot 30º a máxima velocidad). Es por esto que la velocidad y el valor de PWM son máximos y es uno de los motores el que permanece a máxima velocidad mientras que el otro disminuye su valor para hacer el giro.

Por último, y al igual que en el modelo de EJSS, también se podrían haber incluido mejoras en cuanto a las perturbaciones se refiere. El modelo simulado es ideal, por eso se puede realizar un control simplemente con una acción proporcional.

8. ÍNDICE DE FIGURAS

9. ÍNDICE DE CUADROS DE CÓDIGO

10. BIBLIOGRAFÍA Y RECURSOS WEB

Bibliografía

- Ingeniería de control moderna, Katsuhiko Ogata, 5ª edición.
- Modern Control Systems, Richard C. Dorf & Robert H. Bishop, 12th edition.

Recursos web

- Página oficial de Adafruit: https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview
  Aquí podemos encontrar toda la información concerniente al sensor BNO-055.
- Página oficial de MathWorks: https://es.mathworks.com/
  Para ayudarnos con todas las herramientas utilizadas de Matlab.
- Página oficial de Arduino: https://www.arduino.cc/
  Aquí buscábamos información sobre las distintas funciones que podíamos utilizar.
- Página oficial de EJSS: https://www.um.es/fem/EjsWiki/?userlang=es
  Se trata de una "Wiki" de la Universidad de Murcia donde explica que es EJSS y como debemos utilizarlo.
- A todo esto, añadir diversos foros y otras plataformas como Youtube donde poder consultar dudas o buscar tutoriales de cómo usar algunas funciones o herramientas, como la Toolbox de System Identification de Matlab.