Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

INTEGRACIÓN DE SENSORES Y ESTRATEGIA DE CONTROL PARA ROBOT MÓVIL BASADO EN MICROCONTROLADOR

Alumno: Rubén Maestro Huesa

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: Integración de sensores y estrategia de control para robot basado en microcontrolador, que presenta Rubén Maestro Huesa, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Octubre de 2018

El alumno: Los tutores:

Rubén Maestro Huesa Pablo Cano Marchal Diego Martínez Gila

Índice

Índice de figuras

Índice de cuadros de código

1. INTRODUCCION

El presente trabajo final de grado se desarrollará mediante los conocimientos adquiridos a lo largo de nuestro periodo docente en el grado de Ingeniería Electrónica Industrial. Dicho proyecto tratará sobre la integración de una variedad de sensores en un robot móvil y el diseño de controladores de bajo nivel.

Si nos basamos en la historia de los robots, observamos que estos, cuando empezaron a desarrollarse, se basaron la mayor parte en desarrollar robots para el ámbito industrial. Ocurrió de esta manera debido a que era el entorno más rentable, y, con el cual, existía una mayor demanda. Por ejemplo, podemos hablar sobre los robots manipuladores, dedicados al ámbito de la producción, y programados para realizar una acción de manera repetitiva.

La creación de estos robots manipuladores fue una idea que revolucionó la mayor parte de las empresas, aunque el problema llega cuando dichas empresas progresan y amplían su entorno, de manera que, dicho robot no podría trabajar en un rango mayor debido a la limitación de sus articulaciones. De esta manera, se vio necesario desarrollar robots con movimiento automático, los cuales, realizaban la tarea de los manipuladores, y a su vez, se desplazaban de manera repetitiva siguiendo una trayectoria.

El problema era que el robot no se desplazaba por razonamiento propio, sino porque automatizaba movimientos, y esto, conllevaba a que, si el robot se encontraba con obstáculos en su entorno, él seguiría su trayectoria indicada, y no realizaría de manera correcta su función. A partir de este momento, se desarrolló la creación de los **robots móviles**, los cuales no realizarían tareas de manera repetitiva, sino que trabajarían de manera razonada a partir de la información que sus propios sensores suministran.

Nuestro robot móvil está basado en un microcontrolador en el cual se implementarán las diversas estrategias de control. Para poder abarcar las distintas tareas de nuestro trabajo multidisciplinar, haremos uso de un hardware de bajo coste, basado en diferentes sensores, como pueden ser de proximidad o inerciales, y, a su vez, de software libre como es Arduino, el cual trabaja con un lenguaje basado en C++, o de plataformas, como Matlab, las cuales nos servirán de gran ayuda a la hora de entablar comunicaciones con nuestro robot.

Este proyecto tiene de finalidad el uso docente, de tal forma que el alumnado integrará diferentes sensores para poder implementar nuevas estrategias de control.

1.1. Motivación

El principal aspecto que resulta motivador para realizar este trabajo es poder adentrarnos en el mundo de los sistemas de control, ya que es un tema que resultó bastante interesante cuando obtuvimos algunos conocimientos básicos durante el periodo lectivo. Esta rama de la ingeniería es una de la más importantes y básicas para cualquier alumno de esta titulación.

Por otra parte, este trabajo permite al alumnado desarrollar las cualidades básicas para el manejo de este tipo de hardware/software, y poder involucrarse en diferentes proyectos de este tipo de manera más eficiente.

Por último, un motivo también importante es obtener experiencia a la hora de hacer frente a un proyecto real de este tipo, tener la capacidad de poner a prueba los conocimientos adquiridos, y poder resolver los problemas que éste puede albergar en todo su desarrollo.

1.2. Objetivos

El objetivo principal de este proyecto es lograr la implementación de diversas estrategias de control en un robot móvil, de tal forma que desarrolle un comportamiento deseado en un entorno controlado. Asimismo, servirá para reflejar los límites de la electrónica de bajo coste dentro del marco docente.

Para realizar dicho objetivo debemos de saber qué información necesitamos conocer sobre nuestro robot y el entorno. Es por ello que el siguiente paso será estudiar todas las alternativas de hardware de bajo coste propuestas, las limitaciones que poseen cada una de ellas, y en función de estos requisitos, realizar una elección para desarrollar de forma correcta el proyecto.

Por último, una vez dispongamos de los sensores seleccionados, y obtengamos la información necesaria, debemos estudiar e implementar diferentes estrategias de control para conseguir un correcto funcionamiento de nuestro robot móvil.

1.3. Metodología

Para poder llevar a cabo el exhaustivo desarrollo de este proyecto, el alumnado debe seguir una metodología, la cual se compone de diversas pautas, y, de esta manera, permitir la consecución de los distintos objetivos que se han propuesto para finalizar nuestro proyecto.

A continuación, mencionaremos las distintas pautas a seguir de una forma resumida, para hablar posteriormente sobre ellas de manera más íntegra.

- Estudio previo y determinación del tipo de información que vamos a requerir para permitirnos controlar de forma regulada nuestro robot móvil con una estrategia adecuada.
- Documentación y estudio sobre los sensores propuestos, los cuales serán empleados para captar la información necesaria.
- Aprendizaje de las distintas plataformas que utilizaremos para el desarrollo del proyecto. En nuestro caso, las plataformas elegidas serán Matlab, junto a la toolbox de Simulink, y Arduino.
- Repaso teórico sobre los fundamentos de los sistemas de control, profundizando sobre los aspectos relacionados con el controlador.
- Análisis completo sobre las posibles alternativas de estrategias de control que se pueden implementar basándonos en el hardware/software seleccionado.

- Implementación de las estrategias de control finalmente adoptadas. También, se realizará una comprobación de su funcionamiento y se expondrán los resultados finales.

1.4. Organización de la memoria

Este apartado está compuesto por las diferentes secciones que se tratarán a lo largo de la memoria. Los capítulos que estudiaremos son los siguientes:

- **Introducción**: En este primer apartado se introducirá el trabajo que vamos a elaborar, definiendo los objetivos finales y la metodología que hemos de seguir para poder ejecutar el proyecto correctamente. También, se incluyen los aspectos que han resultado motivadores para la realización del proyecto.

- **Determinación de necesidades de información proveniente de sensores:** Este apartado está basado en el análisis completo sobre los distintos tipos de información que se necesitan para controlar a nuestro robot.

- **Comparativa entre alternativas de tecnologías de sensores y elección:** Realizaremos una comparación entre una variedad de sensores que se han propuesto para la realización del proyecto.

- **Integración de sensores en el robot:** En este capítulo hablaremos sobre la elección final de los sensores y sobre las alternativas de control que nos ofrece cada uno de ellos.

- **Diseño e implementación de estrategia de control para el robot:** Se analizarán los fundamentos teóricos de los sistemas de control y serán contrastados con nuestro robot móvil. También, se pondrán en marcha las diferentes estrategias de control, y se expondrán las correspondientes pruebas y resultados.

- **Resultados y conclusiones:** Estudiaremos los resultados obtenidos una vez hemos completado los distintos objetivos de nuestro proyecto y hablaremos sobre las conclusiones adquiridas a lo largo de su realización.

- **Bibliografía y recursos web:** Se expondrá la bibliografía en la cual nos hemos basado para informarnos sobre conceptos básicos e ideas.

- **Anexos:** Se encontrará la correspondiente programación que se ha utilizado para la realización de diversas tareas.

2. DETERMINACION DE NECESIDADES DE INFORMACION PROVENIENTE DE SENSORES

Hoy en día, existe una gran variedad de sensores en el mercado que nos ofrecen cualquier tipo de información, ya sea como la temperatura, la velocidad o cualquier dato que se requiera. Un requisito es que sea hardware de bajo coste, por lo que esto reduce considerablemente el número de sensores en nuestra búsqueda.

La finalidad que perseguimos con nuestro robot móvil es la de controlarlo para que realice tareas como puede ser la de seguir una trayectoria, y, por tanto, la información que se necesita va a tener relación con el espacio y el tiempo. Analizando todas las variables de este tipo de información, se ha llegado a la conclusión de que las necesarias, y más importantes dentro de todas ellas son las expuestas a continuación:

- **Proximidad:** Este tipo de información puede ser de gran utilidad ya que nos advierte sobre el cambio de posición que está efectuando nuestro robot, y, por consiguiente, poder obtener de manera aproximada dónde nos encontramos respecto al punto de partida.
- **Posición lineal o angular:** Con este tipo de información, vamos a obtener la variación de la posición de nuestro robot respecto al tiempo que va transcurriendo, es decir, podremos obtener la velocidad lineal o angular de nuestro robot móvil.

- **Orientación:** Gracias a esta variable podremos obtener constantemente el ángulo de giro de nuestro robot móvil, lo cual nos permite desarrollar estrategias de control relacionadas con el seguimiento de trayectorias.

2.1. Comparativa entre alternativas de tecnologías de sensores y elección

Para nuestro proyecto se nos ha propuesto una variedad de robots móviles, los cuales están compuestos por materiales físicos para su montaje, y su correspondiente hardware. A su vez, se nos permitió elegir libremente en el mercado distintos sensores de bajo coste para satisfacer nuestras necesidades.

A continuación, hablaremos sobre los tipos de sensores que son necesarios y sobre los modelos de cada tipo que finalmente han sido seleccionados para el desarrollo del proyecto.

2.1.1. Sensor de proximidad

Un sensor de proximidad se basa en un transductor que nos permite obtener información acerca de la variación física que existe de manera autónoma. En el caso de estos sensores, la variación física es la distancia que existe entre un punto de referencia y la superficie más cercana.

Dentro de este tipo de sensores, se pueden encontrar una gran variedad de modelos, ya sea dependiendo del rango de medida con el que trabajaremos, las condiciones de trabajo que disponemos, e incluso, el tipo de material sobre el cual obtendremos la medida.

Los tipos de sensores de proximidad más comunes son los capacitivos, los inductivos, y también los de ultrasonidos. Estos últimos son idóneos para nuestro proyecto ya que son bastante sencillos de utilizar y disponen de un precio bajo.

2.1.1.1. HC – SR04

De todos los sensores de proximidad que hay en el mercado se ha decidido por la elección de un sensor ultrasónico, y más concretamente, el modelo HC-SR04.

Figura 1: Módulo ultrasónico HC-SR04

Como podemos observar de su apariencia, se compone dos cilindros, de los cuales, uno se encargará para enviar la señal ultrasónica y el otro para recibirla. Obteniendo el tiempo que se tarda en enviar y recibir la señal, y mediante una serie de operaciones matemáticas, podemos obtener la medida de proximidad aproximadamente.

Debido a que se trata de un sensor de bajo coste, éste nos ofrece la información con baja resolución, aunque nos ofrece una gran exactitud.

2.1.2. Sensor inercial

Un sensor inercial se trata de un sensor compuesto por diversos elementos, y, por tanto, nos proporciona una importante diversidad de información. La mayor parte de estos sensores están compuestos por acelerómetros, giróscopos y magnetómetros, aunque, ocasionalmente, nos podemos encontrar con módulos que dispongan de menos elementos.

Este tipo de sensores nos ofrecen información acerca de la temperatura, fuerza magnética y muchas variables más, aunque para nuestro trabajo solamente utilizaremos la información de la orientación.

2.1.2.1. BNO – 055

Este módulo está basado en una unidad de medida inercial compuesta por 9 grados de libertad. En concreto, este modelo, sigue la arquitectura de sistemas típica en este tipo de sensores, es decir, está compuesto por un acelerómetro, un giróscopo y magnetómetro.

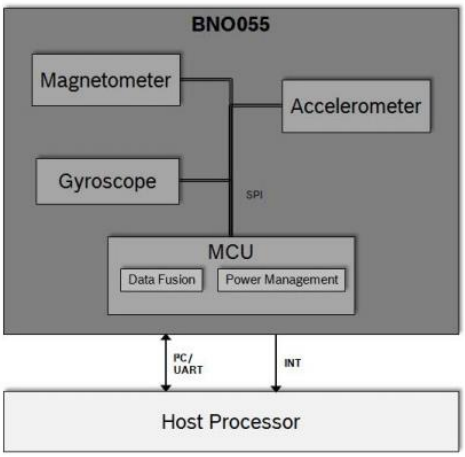

Figura 2: Arquitectura de los sistemas del módulo BNO-055

Gracias a la integración y fusión de todos estos componentes en una pequeña placa electrónica, este módulo será idóneo para nuestro robot, ya que éste posee un cableado sencillo y dimensiones pequeñas, y, por tanto, el sensor no albergará demasiado espacio en la superficie posterior.

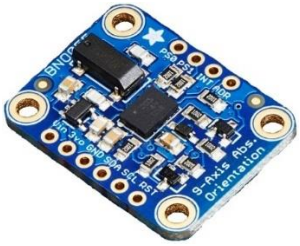

Figura 3: Módulo inercial BNO-055

2.1.2.2. MPU – 6050

El módulo MPU-6050 se trata de otra unidad de medida inercial diferente a la anterior, ya que dispone solamente de 6 grados de libertad. Es decir, dispondremos tan solo de un acelerómetro y un giróscopo.

Figura 4: Módulo inercial MPU-6050

Al igual que el BNO-055, este módulo también está basado en una pequeña placa, donde se encuentran todos los componentes integrados, por lo que dispondrá de las mismas ventajas que el módulo anterior, en cuanto a montaje y cableado se refiere.

2.2. Integración de sensores en el robot

Este capítulo tratará sobre el estudio de los módulos que finalmente han sido seleccionados. Dicho estudio estará basado en conocer el funcionamiento de nuestros sensores, y la manera para establecer conexión con el microcontrolador Arduino.

Para finalizar el estudio de los sensores, se procederá a realizar numerosas pruebas sobre su puesta en marcha y funcionamiento, para posteriormente, exponer los resultados obtenidos con cada sensor de nuestro robot móvil.

2.2.1. Análisis previo del robot disponible

En este capítulo hablaremos sobre los módulos que hemos seleccionado para nuestro proyecto de robot móvil. Finalmente, hemos escogido un sensor de cada tipo, y así disponer de más alternativas para trabajar con el robot en un futuro.

En cuanto a la elección de los **sensores de proximidad**, no nos supondrá un problema ya que solo disponíamos de un modelo de este tipo. Se trata del módulo HC-SR04, un sensor de proximidad ultrasónico. Al ser un sensor de bajo coste debemos de asumir que no nos ofrecerá información con una buena exactitud.

En cuanto a los **sensores inerciales** se ha seleccionado finalmente el módulo BNO-055, ya que posee cierta ventaja respecto al MPU-6050 al disponer de un mayor número de componentes, y por tanto será capaz de ofrecernos una mayor variedad de información.

2.2.2. Software

En este apartado analizaremos el software con el cual hemos trabajado durante la elaboración del proyecto. Las plataformas que usaremos para desarrollar el trabajo serán Arduino y Matlab.

Aparte, hablaremos sobre las distintas herramientas que disponen ambas plataformas y que nos han servido para poner en marcha los sensores seleccionados y entablar comunicación con nuestro robot móvil de manera inalámbrica.

2.2.2.1. Arduino

En primer lugar, la plataforma principal de este proyecto es **Arduino**. El principal motivo para su empleo es que dicha plataforma se trata de un software libre. A su vez, nuestro robot móvil dispone de un microcontrolador Arduino Uno, por lo que, gracias a este software seremos capaces de poner en marcha los sensores e implementar las distintas estrategias de control.

Un aspecto importante a considerar sobre el uso de esta plataforma es que, durante nuestro periodo lectivo, hemos trabajado en alguna ocasión con este software y su correspondiente lenguaje de programación, el cual es C++, y, por tanto, será sencillo trabajar con Arduino.

A continuación, vamos a analizar la placa con la que hemos desarrollado nuestro proyecto. Se trata de un microcontrolador Arduino Rev 3, una placa electrónica basada en un chip microcontrolador, cuyo nombre es ATmega328P.

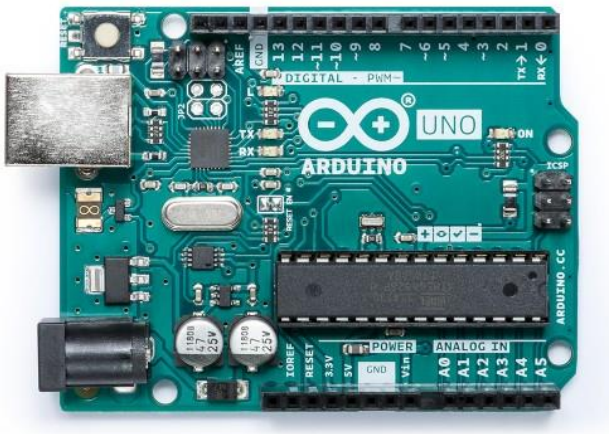

Figura 5: Microcontrolador Arduino Uno Rev 3

La placa Arduino Uno Rev 3 es una de las más conocidas y básicas. Es usada en la mayoría de proyectos básicos de electrónica, ya que dispone de las características básicas que se necesitan para desarrollar un trabajo. En concreto, esta placa consta de 14 entradas digitales, de las cuales 6 pueden ser usadas como salidas de PWM, y de 6 entradas analógicas.

Gracias a la cantidad de entradas/salidas que dispone esta placa Arduino, nos será suficiente para llevar a cabo el conexionado de los distintos sensores que disponemos.

2.2.2.2. Matlab

Una diferente plataforma usada es Matlab. El primer motivo para usarla es debido a que necesitamos una plataforma con la cual entablar comunicación en nuestro proyecto, y así, enviar referencias a nuestro robot móvil. Un ejemplo de uso será el mandarle un ángulo de referencia, y, que nuestro robot móvil gire los grados indicados de manera autónoma.

Para entablar comunicación con nuestro robot, necesitaremos realizarla de manera inalámbrica, ya que, por puerto serie debemos de mantener conectado un cable al robot, e interferiría en el funcionamiento de éste. Las conexiones inalámbricas serán establecidas mediante un dispositivo bluetooth, y más concretamente, con el módulo HC-06.

Figura 6: Módulo Bluetooth HC-06

Este tipo de módulos inalámbricos, como el Bluetooth, disponen de un inconveniente, y es que disponen de tiempo de envío, y, por tanto, nuestro robot no captará las referencias de manera inmediata. Esto sería un problema grave si mediante este módulo, recibiéramos información para controlar a nuestro robot, pero, en nuestro caso, solo será usado para enviar referencias al robot.

Otro motivo realmente interesante para el uso de Matlab es disponer de una gran variedad de Toolbox y herramientas. La principal Toolbox que se ha usado para este proyecto sobre el control es Simulink, y, sobre ella, hablaremos en su apartado correspondiente.

2.2.2.3. Simulink

Simulink es una herramienta que dispone Matlab que puede ser de gran utilidad debido a la cantidad de elementos que podemos manejar en la plataforma.

Puede ser útil en muchos campos, como, por ejemplo, en robótica o para el procesamiento de señales. En nuestro caso, siendo un trabajo sobre control, también nos servirá de gran utilidad trabajar con esta herramienta, ya que contiene una gran cantidad de elementos para poder trabajar sobre el control de nuestra planta, e implementar bucles de realimentación de manera gráfica.

2.2.3. Hardware

En este apartado hablaremos sobre el hardware que compone a nuestro robot móvil, como pueden ser los motores y el controlador que se trata de ponerlos en marcha.

Hablaremos también sobre su funcionamiento y su propia programación en Arduino para hacer desplazar a nuestro robot móvil.

2.2.3.1. Driver L298N

Se trata de un driver dual para motores DC, con el cual podemos controlar el sentido y la velocidad a la que giran nuestros motores. La velocidad de giro será proporcionada mediante un valor de PWM entre 0 y 255.

A continuación, mostramos la diferente programación de Arduino para poner en marcha nuestros motores:

```
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);
}
```

Cuadro de código 1: Programación de Driver L298N

De esta manera, gracias a los pines “IN” conseguiremos desplazar nuestros motores en un sentido u otro, según esté programado, y, seleccionar un valor de PWM a cada motor para que nuestro robot realice la tarea que nosotros deseamos.

2.2.4. Alternativas de uso para cada sensor

En este apartado hablaremos sobre las diferentes alternativas de uso que nos ofrecen los sensores que hemos seleccionado, mencionando a su vez las limitaciones que estas pueden albergar.

Por parte del sensor de proximidad, nuestras alternativas serán un poco más limitadas ya que la información que nos ofrece es exclusivamente sobre la proximidad a una superficie.

La primera alternativa que se nos ocurre es sobre su seguridad. Es decir, con el uso de este sensor sabremos a qué distancia se encuentra la superficie más próxima, y, mediante una específica estrategia de control, nuestro robot conseguirá nunca estrellarse con una superficie. Esta alternativa será imprescindible en la solución final debido a que evitamos que nuestro robot sea dañado.

Esta estrategia de control conlleva un problema, y es que, el robot solamente dispone de un módulo ultrasónico en la parte delantera de éste, por lo que, si en cualquier ocasión, nuestro robot móvil se desplaza hacia atrás, éste colisionará.

Por tanto, para solucionar dicha limitación, nuestro robot se deberá mover siempre hacia delante para que dicha estrategia de control siempre sea efectiva.

Una diferente estrategia de control es que, aparte de evitar estrellarse con una superficie, el robot esquive dicho obstáculo eligiendo un nuevo camino, y todo esto de manera autónoma. Para la implementación de esta funcionalidad, haremos uso de los dos sensores que vamos a utilizar, ya que, aparte de necesitar la información de la proximidad, también necesitaremos la orientación para ser capaces de regularla.

Aplicando la estrategia de control mencionada anteriormente, y trabajando en un entorno elaborado por nosotros mismos, nuestro robot móvil será capaz de seguir una trayectoria.

La última alternativa fue la de la obtención de la velocidad lineal del robot, ya que sabíamos la distancia que se había aproximado a una superficie en un tiempo definido. Gracias a esta información, y mediante operaciones matemáticas, podríamos obtener la velocidad lineal que posee nuestro robot en cada instante de tiempo, y así poder disponer de más alternativas para trabajar en el proyecto.

Esta última alternativa fue probada mediante el puerto serie (mediante el cable USB) antes de disponer de módulo Bluetooth, y se desarrolló en Simulink debido a la facilidad que nos ofrece la programación gráfica de la plataforma.

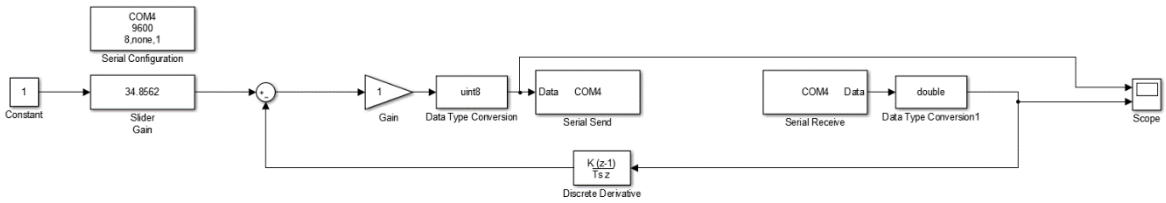

Figura 7: Modelo de bloques para la obtención de la velocidad en Simulink

De esta manera, mediante puerto serie, podemos enviar un valor de PWM a nuestros motores y recibir la proximidad a la pared. Derivando esta proximidad mediante un derivador discreto obteníamos la velocidad, pero debido a que nuestro robot no variaba mucho la velocidad con distintos valores de PWM nos salían velocidades exactamente iguales, y, por tanto, esta alternativa no se llevó a cabo exitosamente.

En cuanto al sensor inercial, éste nos ofrece una gran variedad de alternativas debido a la variedad de información que nos ofrece.

La primera alternativa que decidimos desarrollar tiene relación con la estabilidad de nuestro robot móvil. Es decir, nuestro robot no puede seguir una trayectoria recta por sí solo debido a una falta de homogeneidad en la distribución de la masa provocada por el posicionamiento de los distintos componentes. Por tanto, gracias a un control específico, podremos equilibrar a nuestro robot para que pueda seguir el camino deseado sin desviación alguna.

Otra alternativa posible con este sensor sería la de mandar una referencia de ángulo a nuestro robot mediante el módulo Bluetooth, y que nuestro robot, mediante dicha estrategia de control, regule sus motores para conseguir girar hasta dicha referencia.

Por último, y como hemos mencionado en la sección del sensor ultrasónico, mediante éste y el módulo inercial, el robot será capaz de desarrollar una trayectoria en un entorno elaborado por el usuario

2.2.5. Prueba de sensores

La primera tarea a realizar antes de poner a prueba ambos sensores, será estudiar las distintas hojas de características específicas sobre cada sensor. En ellas, podremos obtener información relevante, como, por ejemplo, el voltaje de alimentación, o el rango de información que nos ofrecen. Dichas hojas de características serán expuestas en el apartado de anexos.

La siguiente tarea es elaborar el correcto cableado de nuestros módulos en la placa. Para ello, debemos de informarnos sobre sus características eléctricas y, sobre todo, de los pines de conexión que disponen nuestros sensores.

Una vez realizadas las tareas anteriores, el siguiente paso a realizar es el de establecer una comunicación con nuestro microcontrolador Arduino. Cabe destacar que nuestros sensores seleccionados poseen diferentes librerías compatibles con Arduino, y, por tanto, será bastante sencillo establecer conexión con nuestro microcontrolador. Dichas librerías contenían ejemplos de uso de los distintos sensores, en los cuales nos basamos para ver qué utilidades nos ofrecían.

Para dar como finalizada la prueba de cada sensor, deberemos de obtener medidas, y, comprobar si es información real y coherente con el entorno en el que se encuentra nuestro robot móvil. A su vez, se representarán dichos datos frente al tiempo de muestro, y así comprobar también gráficamente el correcto funcionamiento de ambos sensores.

2.2.5.1. Módulo ultrasónico HC-SR04

En este apartado hablaremos sobre cómo hemos establecido conexión con nuestro sensor ultrasónico, y a su vez, para exponer los datos obtenidos y las limitaciones que albergamos con dicho módulo.

Dicho sensor funciona de la siguiente manera: éste transmite un pulso y tras incidir en la superficie más próxima, se refleja hacia el sensor y éste capta el pulso reflejado. De esta manera, conociendo la velocidad de dicho pulso, que es la velocidad de la luz, y el tiempo que tarda en enviar y recibir el pulso, podremos obtener aproximadamente la proximidad a la superficie más próxima.

Gracias a la plataforma Arduino, y la compatibilidad que tienen estos sensores con el microcontrolador, nos será sencillo el hecho de programar y poner en marcha el sensor. A continuación, se muestra la función para obtener la proximidad en Arduino:

```
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 2: Función de sensor ultrasónico HC-SR04

Por lo tanto, una vez se ha elaborado la programación, podemos dar paso a la obtención de información con nuestro sensor.

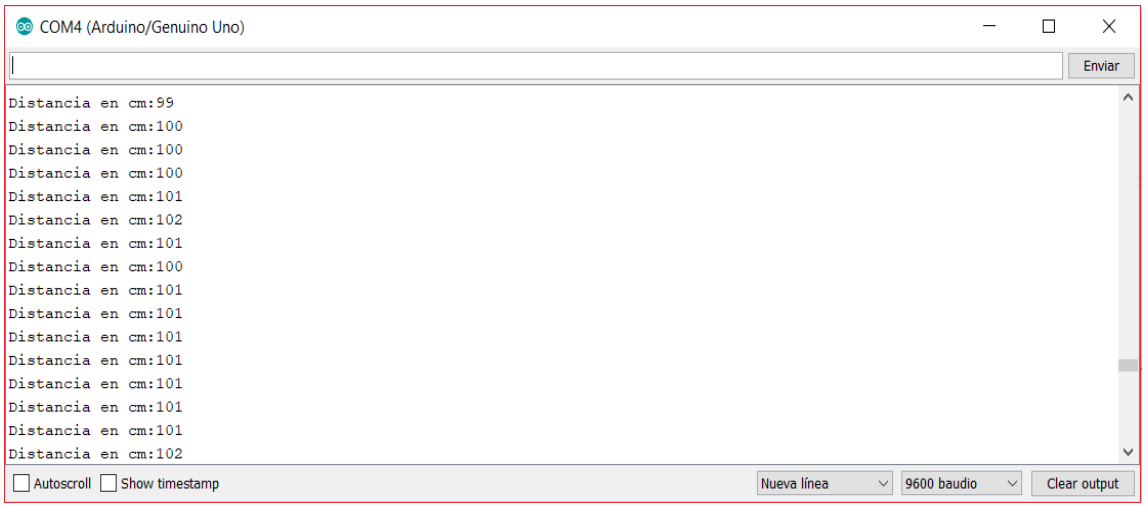

Figura 8: Consola de Arduino obteniendo información de la proximidad

Para obtener los datos que se observan en la figura 8, colocamos nuestro robot a 1 metro aproximadamente de una superficie, y como podemos observar, nuestro sensor nos ofrece una información bastante exacta, aunque a su vez, nos ofrece algo de ruido.

Dicho sensor nos ofrece la mayoría de las veces información exacta, aunque en ocasiones, y como vemos en la siguiente figura, nos ofrece medidas que son erróneas, como medidas de 30 metros (longitud que no es capaz de medir nuestro sensor)

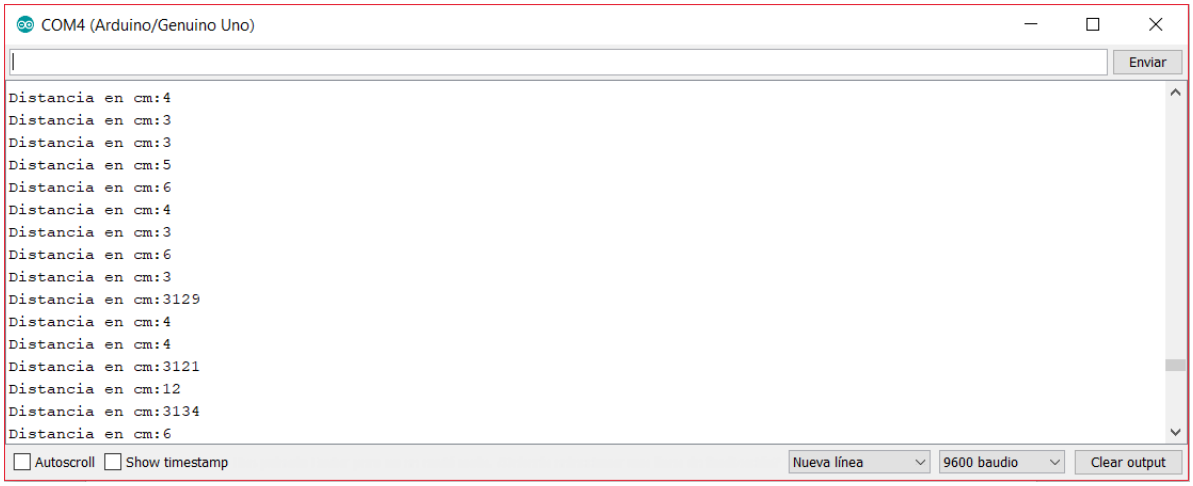

Figura 9: Información errónea proveniente del sensor HC-SR04

Analizando el error que nos ofrece y el funcionamiento de dicho sensor, dimos con el problema. Dicho problema es que el sensor no capta bien la medida si aparece y desaparece un obstáculo de manera instantánea, como, por ejemplo, colocar nuestra mano delante del sensor e inmediatamente quitarla. Otra ocasión en la que aparecía el error es cuando trabajamos con el robot en un espacio abierto con muchos obstáculos irregulares, ya que se puede dar el caso en el nuestro sensor no capta el pulso de vuelta, y nos ofrece este mismo error.

Por tanto, sabiendo el motivo del error, debemos trabajar en un entorno que disponga de paredes sin irregularidades y sin obstáculos de por medio para que realice de manera correcta la estrategia de control.

Otro problema que surge con el uso de este sensor, y es cuando nuestro robot realiza una curva. Sabemos que nuestro robot envía el pulso en línea recta, y que por uno de sus cilindros envía un pulso, y por otro lo recibe. Por tanto, cuando se encuentra en dicha situación problemática, dichos cilindros se encuentran uno más cerca que otro de la superficie de la curva y el pulso tardará un tiempo diferente al enviarse y al recibirse, y de esta manera, nuestro sensor nos proporcionará información errónea.

Es debido a esto por lo que para las pruebas hemos operado con el robot desplazándolo linealmente, de manera que avance hacia una superficie dos segundos y retroceda dos segundos. Cabe destacar que cada dos segundos, aplicábamos a nuestros motores aleatorios valores de PWM.

Una vez sabemos las limitaciones de nuestro sensor, lo podemos poner en marcha y recoger información sobre la proximidad. En la siguiente figura podemos cómo varía dicha información conforme pasa el tiempo.

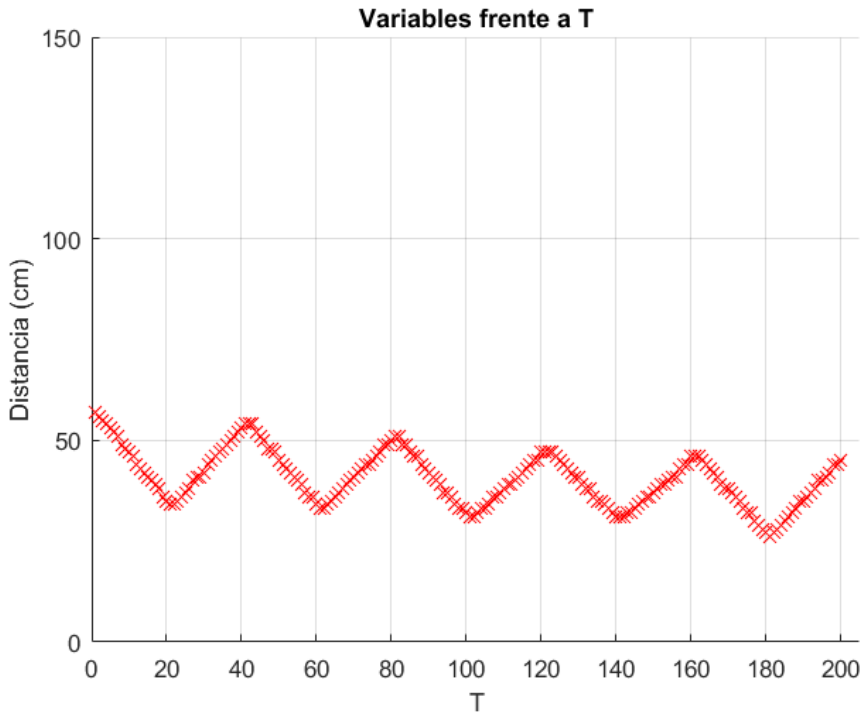

Figura 10: Información de la proximidad frente al tiempo

Hay que señalar un aspecto que se aprecia en la figura 10 que tiene relación con la velocidad que lleva el robot, y es la pendiente que tienen las rectas que observamos. Analizando la gráfica, junto al valor de PWM que le damos a los motores en cada ciclo del recorrido, vemos que cuando el robot va a mayor velocidad, la inclinación de la recta es mayor, y en caso contrario cuando el robot va a menor velocidad.

2.2.5.2. Módulo inercial BNO-055

Al igual que con el sensor ultrasónico, operaremos de la misma manera con el sensor inercial para verificar su correcto funcionamiento.

Antes de dar paso a la realización de las pruebas, debemos establecer la conexión del módulo inercial con nuestro microcontrolador. Gracias a las librerías que nos ofrece este sensor inercial, no necesitaremos programar ninguna función para obtener la información que necesitamos. Por ejemplo, para obtener el vector de Euler (x, y, z) operaríamos de la siguiente manera en Arduino:

```
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 3: Función para obtener información con sensor BNO-055

Para obtener el ángulo que necesitamos (Yaw), el cual nos indica la dirección que sigue nuestro robot, solo necesitamos escribir las siguientes líneas de código:

```
sensors_event_t event;
bno.getEvent(&event);
orientation = (event.orientation.x);
```

Cuadro de código 4: Función para la obtención directa de la orientación

Aparte, gracias a este módulo inercial podremos obtener el ángulo Pitch, con el cual podríamos controlar el balanceo si nuestro robot dispusiera solamente de dos ruedas.

Una vez hemos establecemos conexión con nuestro sensor, mostramos en la siguiente figura los datos que nos ofrece cuando se gira hacia la derecha hasta alcanzar los 90 grados. A modo informativo, este sensor nos ofrecerá la información en un rango de 0 a 360 grados, y al superar el máximo valor del rango, nuestro sensor nos devuelve de nuevo el valor de 0 grados.

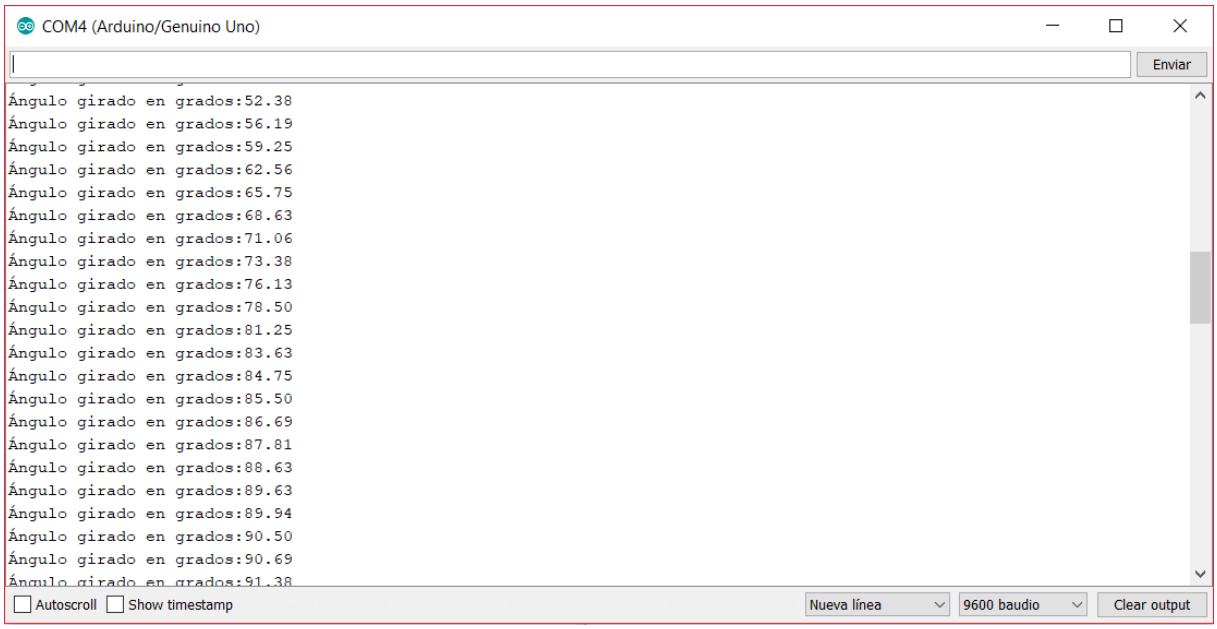

Figura 11: Consola de Arduino obteniendo información de la orientación

Como podemos observar en la figura 11, la información que nos ofrece es siempre correcta y con buena resolución, y no nos ofrece ningún tipo de valor extraño o erróneo.

Antes de dar paso a la representación gráfica, cabe destacar que nuestro módulo inercial nos permite realizar una calibración de los datos antes de poner en marcha su tarea. De esta manera, nos aseguramos que la información que recibimos será exacta. Para realizar dicha calibración, usaremos la siguiente función en Arduino:

```
1. void displayCalStatus(void)
2. {
3.   /* Get the four calibration values (0..3) */
4.   /* Any sensor data reporting 0 should be ignored, */
5.   /* 3 means 'fully calibrated" */
6.   uint8_t system, gyro, accel, mag;
7.   system = gyro = accel = mag = 0;
8.   bno.getCalibration(&system, &gyro, &accel, &mag);
9.
10.  /* The data should be ignored until the system calibration is >
   0 */
11.  Serial.print("\t");
12.  if (!system)
13.  {
14.    Serial.print("! ");
15.  }
16.
17.  /* Display the individual values */
18.  Serial.print("Sys:");
19.  Serial.print(system, DEC);
20.  Serial.print(" G:");
21.  Serial.print(gyro, DEC);
22.  Serial.print(" A:");
23.  Serial.print(accel, DEC);
24.  Serial.print(" M:");
25.  Serial.println(mag, DEC);
26.}
```

Cuadro de código 5: Función para la calibración de sensor BNO-055

Para terminar la verificación representaremos gráficamente los datos operando de manera similar que, con el módulo ultrasónico, ya que, en vez de desplazar linealmente a nuestro robot, lo haremos siguiendo una trayectoria circular, para observar cómo varía la orientación. Por tanto, nuestro robot avanzará a la derecha dos segundos, y retrocederá dos segundos, para siempre volver al punto de partida. Al igual que con el anterior módulo, cada dos segundos cambiaremos aleatoriamente el PWM de nuestros motores.

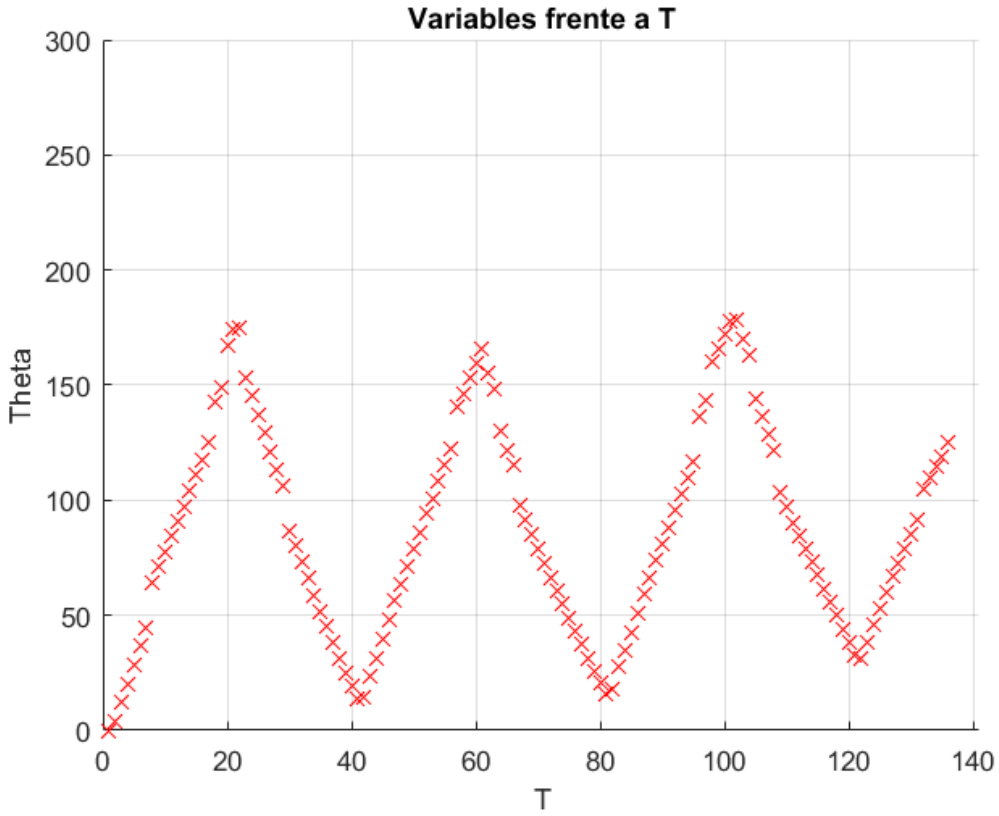

Figura 12: Información de la orientación frente al tiempo

Como podemos observar en la figura 12, obtenemos de nuevo rectas con distintas pendientes, ya que cuando el robot avanza girando hacia la derecha aumenta el ángulo de giro, y lo contrario ocurre cuando se desplaza hacia atrás.

Una última observación que se aprecia es la longitud de las rectas, que son diferentes cada dos segundos. Esto es debido a la velocidad de giro de nuestro robot cada dos segundos, ya que cuanta mayor es la velocidad, mayor ángulo recorrerá, y menor si circula a menor velocidad.

3. DISEÑO E IMPLEMENTACIÓN DE ESTRATEGIA DE CONTROL

Una vez realizada la tarea de verificación del correcto funcionamiento de nuestros sensores, es el momento de exponer las diferentes alternativas de control que se mencionaron previamente.

Antes de abordar el tema sobre las diferentes estrategias de control que se han propuesto para nuestro robot, se procederá a realizar una recopilación sobre los fundamentos básicos de control y analizar cómo se pueden llevar a cabo en la práctica.

Una vez recordada la teoría básica sobre ingeniería de control, conociendo el sistema físico de nuestro robot móvil, y disponiendo del robot con los sensores correctamente funcionando, se procederá a plantear las diferentes estrategias de control que somos capaces de llevar a cabo, las limitaciones de cada una y, por último, los resultados y conclusiones (exponiendo todos los datos obtenidos durante los ensayos y sus respectivas gráficas) sobre cómo funciona nuestro robot con la implementación de dichas estrategias.

Por último, y antes de dar paso a los fundamentos de control, se debe aclarar que este robot está diseñado para aplicaciones docentes, y, por tanto, el alumno debe de estar informado de cómo funciona nuestro robot con una estrategia seleccionada y un entorno establecido.

3.1. Fundamentos de modelado de sistemas de control

Este apartado está dedicado al repaso teórico de los fundamentos básicos de control, empezando por un aspecto muy importante y útil: la función de transferencia. Además, hablaremos brevemente sobre cuál es la que define el sistema físico de nuestro robot móvil.

Para que el siguiente apartado cobre importancia, tenemos que partir de la base de que un problema real prácticamente nunca va a responder de manera lineal, con lo cual el concepto de linealización y su aplicación a nuestro problema será realmente importante para el modelado del mismo.

Para terminar el repaso teórico, profundizaremos sobre todo con lo relacionado a la estrategia de control debido a que es el punto más importante de este proyecto. Se hablará sobre distintos conceptos, como pueden ser la definición de controlador, el control en lazo cerrado o lazo abierto, los tipos de controlador que existen, etc.

3.1.1. Función de transferencia

La función de transferencia es uno de los pilares fundamentales de los sistemas de control, ya que nos permite conocer el desarrollo del sistema físico a estudiar. Ésta, consiste en el cociente entre la transformada Laplaciana de una entrada, y una salida, también elaborada mediante la transformada de Laplace. Así, mediante la función de transferencia podremos conocer la salida de un sistema a partir de una señal de entrada aplicada.

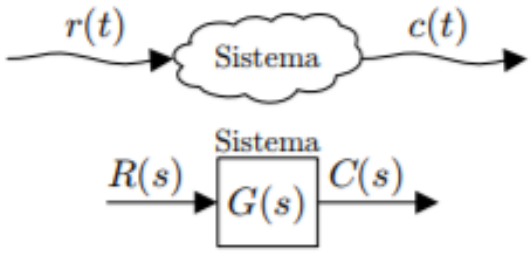

Figura 13: Función de transferencia

Hay que resaltar que, dicha función de transferencia es un invariante del sistema, y, por tanto, ante un valor de entrada que un usuario propone al sistema, la salida dependerá siempre de la función de transferencia del sistema propuesto.

Un ejemplo de sistema físico de nuestro robot podría ser el de los motores de corriente continua, con los cuales hacemos mover a nuestro robot. Este tipo de motores funcionan a partir de un voltaje proporcionado, y de esta manera el motor gira a una cierta velocidad angular. Por tanto, de dicho sistema físico podemos obtener una función de transferencia, la cual nos relaciona la velocidad angular del motor a partir de un voltaje de entrada.

Mediante la aplicación de las ecuaciones generales del robot, tanto como las eléctricas, magnéticas y mecánicas, se demuestra que la función de transferencia anterior de un motor de corriente continua es de primer orden aproximadamente, y su estructura es la siguiente:

$$F(s) = \frac{k}{\tau s + 1}$$

Figura 14: Estructura de función de transferencia de primer orden

Para finalizar, hay que señalar que existen más tipos de función de transferencia con distintas características, como puede ser, con tiempo de retardo. A su vez, existen con orden superior a uno, como por ejemplo puede ser, la función de transferencia que relaciona la variación de la posición angular del motor con un voltaje de entrada, la cual es de segundo orden.

3.1.2. Linealización del sistema

Un sistema no lineal se considera aquel en el que no se puede aplicar la propiedad de superposición o homogeneidad. De esta manera, trabajando con sistemas no lineales, albergaremos diversos problemas, como, por ejemplo, obtener relaciones matemáticas no lineales. Por tanto, deberemos recurrir a la linealización para obtener una aproximación de un sistema lineal.

Para conseguir dicha aproximación lineal, aplicaremos una expansión en serie de Taylor alrededor del punto en el que nos basamos. Dicho modelo lineal tendrá validez para un rango de puntos próximos al punto de linealización.

$$y = f(x) = f(x_0) + \left.\frac{df}{dx}\right|_{x=x_0} (x - x_0) + \left.\frac{1}{2!}\frac{d^2 f}{dx^2}\right|_{x=x_0} (x - x_0)^2 + \ldots$$

Figura 15: Expansión en serie de Taylor

3.1.3. Sistemas de control en lazo abierto y cerrado de un sistema

En este apartado hablaremos sobre los tipos de sistemas de control que existen, en lazo abierto y lazo cerrado. En ambos tipos, dispondremos de una entrada o valor de referencia, y, una función de transferencia con la cual obtendremos el valor de salida de nuestro sistema

Se dice que un sistema de control se elabora en lazo abierto cuando la entrada no sufre una variación debido al valor que nos proporciona la salida de dicho sistema.

Figura 16: Sistema de control en lazo abierto

Como podemos observar en la figura 4, su estructura consta de un controlador, con el cual vamos a regular el valor de la entrada para obtener la salida deseada en dicho proceso.

Estos sistemas de control tienen la desventaja de no disponer una realimentación en su estructura, y, por tanto, al no obtener información sobre la salida de dicho sistema, no conseguiremos alcanzar nuestro valor de referencia con una gran exactitud.

Un ejemplo sobre un sistema de control en lazo abierto es el control de la intensidad de la luz. Nuestro controlador sería el potenciómetro, y de esta manera, según cuánto lo giremos, tendremos más o menos intensidad lumínica.

Ahora damos paso a los sistemas de control en lazo cerrado. Al contrario de los sistemas en lazo abierto, este tipo de sistemas nos permite variar el valor de la entrada a partir del valor de la salida que obtenemos.

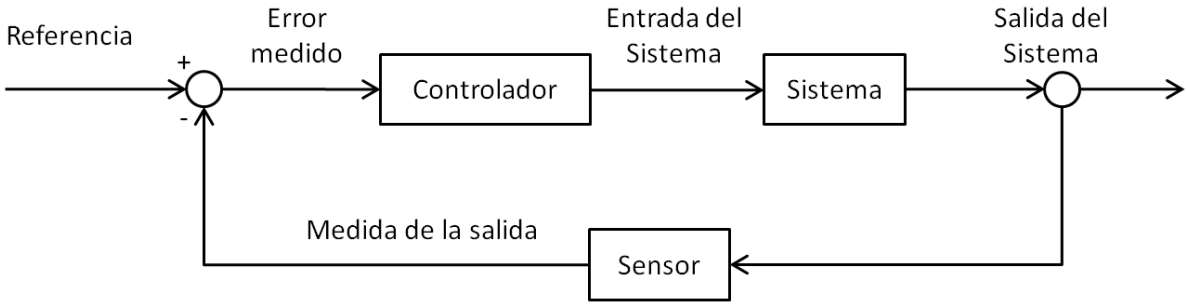

Figura 17: Sistema de control en lazo cerrado

Como vemos en la figura 5, la estructura de este tipo de sistemas es parecida a la anterior, pero con la diferencia de que disponemos de una realimentación con la cual obtenemos el valor de la salida de nuestro sistema.

De esta manera, obteniendo la diferencia que hay entre el valor de salida y el valor de referencia, y, mediante el desarrollo de un controlador óptimo, podremos variar el valor de la entrada del sistema para obtener un valor de salida con mayor exactitud. Por tanto, para que nuestro robot opere correctamente, utilizaremos nuestros sensores para aplicar este tipo de sistemas en nuestro proyecto.

Un ejemplo sobre este tipo de sistema de control puede ser el de un aire acondicionado de un coche. En este caso, nuestro sensor es el termómetro disponible en el coche, el cual nos ofrecerá el valor de temperatura en cada instante.

3.1.4. Diseño y compensación de sistemas de control

En este apartado hablaremos sobre todo lo relacionado con el concepto de controlador, lo cual tiene una gran importancia en nuestro trabajo debido a que gran parte consistirá en la implementación de dicho controlador.

Un controlador se trata de un elemento de nuestro sistema que se encarga de variar el valor de nuestra entrada con el fin de conseguir el valor de salida deseado. Éste trabajará con mayor exactitud si nuestro sistema de control es en lazo cerrado.

A continuación, damos paso a los diversos tipos de controlador que podemos elaborar:

- Acción proporcional, o tipo P.
- Acción integral, o tipo I.
- Acción PI

Empezaremos hablando sobre el controlador de acción proporcional, con el cual regularemos el valor de nuestra entrada al sistema, aplicando una proporción al error entre el valor medido y el valor de referencia. De esta manera, si el valor del error es considerable, nuestro valor de entrada aumentará, y ocurre justo lo contrario cuando el valor del error es mínimo.

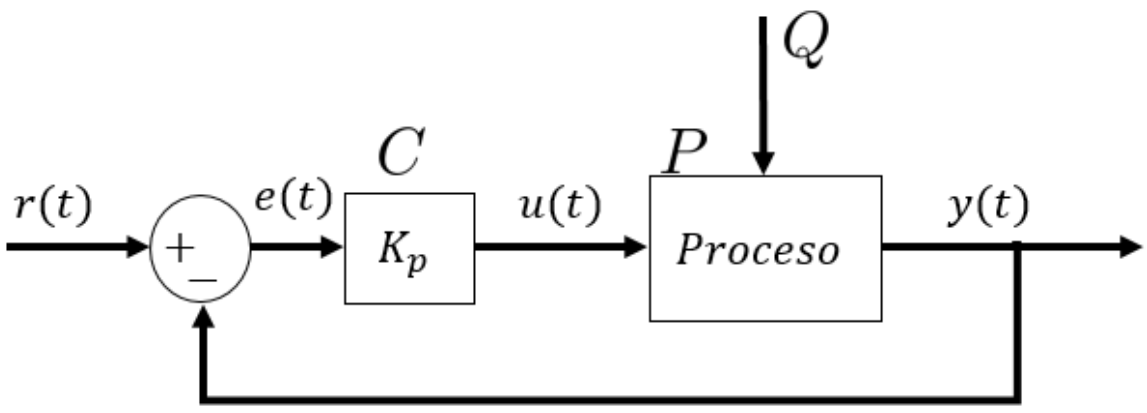

Figura 18: Estructura de controlador de acción proporcional

Respecto al controlador de acción integral, hay que señalar que el valor de la salida va a depender de la variación de ella misma y del tiempo en el que se somete dicha variación, o dicho de manera distinta, este controlador es proporcional a la integral de la señal de error que comete dicho sistema. Gracias a este tipo de acción podemos desarrollar una compensación de atraso en nuestro sistema, de manera que, podemos reducir el error de nuestro robot en régimen estacionario.

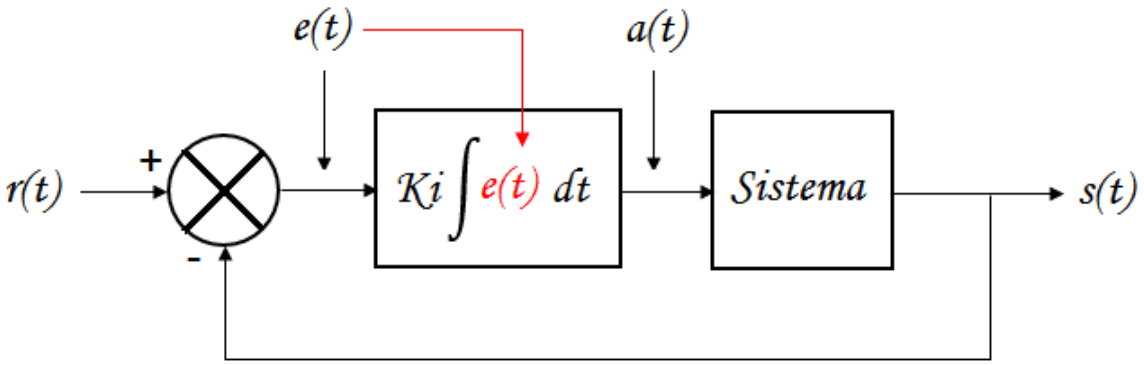

Figura 19: Estructura de controlador de acción integral

Por tanto, juntando ambas acciones (proporcional e integral) podemos conseguir un controlador PI que nos permita regular el valor de la entrada al sistema para conseguir dicha salida, y a su vez, nos corrija el error en régimen estacionario de dicha salida.

3.2. Implementación de los controladores

En este apartado hablaremos sobres las diferentes estrategias de control que vamos a implementar en nuestro robot móvil, mencionando a su vez, el objetivo que perseguimos, las posibles alternativas junto a las limitaciones que éstas conllevan para realizar dicha estrategia y para finalizar, se expondrán las diferentes pruebas y resultados que obtenemos mediante la implementación de cada una de ellas.

También, hablaremos sobre la programación de los distintos controladores que implementamos y expondremos los diferentes diagramas de bloques para entender cómo se realiza cada estrategia de control.

Antes de dar paso a los diferentes controladores, cabe destacar que la mayoría de ellos fueron mencionados anteriormente en el apartado de las alternativas de los sensores, y finalmente, se han podido llevar a cabo.

3.2.1. Control de ángulo

Antes de dar paso a los objetivos que perseguimos gracias a dicho control, debemos mencionar de manera resumida su funcionamiento. Este control se basa en la regulación de los valores de PWM de cada uno de nuestros motores para conseguir variar la orientación de nuestro robot y conseguir los objetivos deseados.

3.2.1.1. Objetivos

Como primer objetivo de este control de ángulo es el estabilizar a nuestro robot. Como sabemos, nuestro robot tiene colocados de manera aleatoria los diferentes sensores y demás elementos en la parte superior, y, por tanto, no dispone una homogeneidad en su masa.

Esto quiere decir que, si un lado de nuestro robot alberga más peso, el motor de dicho lado alberga una velocidad angular menor que la del otro motor con el mismo valor de PWM.

Este control recibe gran importancia ya que, si ambos motores reciben un valor de PWM igual, nuestro robot no sería capaz de seguir una línea recta. A su vez, si el robot móvil fuera desplazado por cualquier circunstancia y varía su orientación, él podría retomar dicho ángulo de partida de manera autónoma.

Por último, podremos enviar a nuestro robot una referencia de ángulo de manera inalámbrica, y éste mediante dicho control consiguiera regular su orientación hasta alcanzar la de dicha referencia.

3.2.1.2. Alternativas para el control

En este apartado hablaremos sobre las alternativas que hemos llevado a cabo para desarrollar la programación de dicho control. A su vez, mencionaremos las limitaciones que suponen cada una y si finalmente se han llevado a cabo.

Como ya sabemos, por falta de homogeneidad de masa en nuestro robot debido a la disposición del hardware, el peso que recaía en una rueda era mayor que el de la otra. Esto daba lugar a que, para un mismo valor de PWM en ambos motores, nuestro robot no seguía una trayectoria en línea recta.

Es por este motivo que lo primero que nos planteamos fue calibrar esta descompensación. Para corregir este desequilibrio, pensamos en obtener una proporción entre los valores de PWM. Hicimos dos ensayos en los que hacíamos girar a nuestro robot sobre una rueda (ambos con el mismo PWM) y obtener el ángulo total que había girado. De esta forma podíamos obtener la proporción entre ambas ruedas, según la ecuación:

$$K_{motores} = \frac{\theta_{total\ A}}{\theta_{total\ B}}$$

Nuestro siguiente paso fue comprobar que esa proporción entre los valores de PWM de una y otra rueda cumplía nuestras expectativas, así que nos decidimos a realizar otro ensayo aplicando un valor de PWM al motor A, y ese mismo valor multiplicado por dicha K al motor B.

Lamentablemente seguíamos teniendo los mismos resultados: el robot hacía una ligera curva hacia un lado. Tras esto repetimos varias veces el ensayo anterior, y ahí nos percatamos de que para un mismo valor de PWM, las ruedas no siempre giraban lo mismo. La variación era pequeña, pero suficiente para imposibilitar la obtención de una proporción entre los valores de PWM de cada motor.

El problema de esta alternativa se acrecentaba cuando reducíamos los valores de PWM, es decir, para valores pequeños la constante de proporción era considerablemente mayor que para PWM cercanos al máximo. Es por todo esto que tuvimos que descartar esta posibilidad. A continuación, mostramos un cuadro de código donde reflejamos esta variación de las constantes de proporción según el valor de PWM:

```
if (pwm_valueA<100){ //el motor A se considera el izquierdo
      kpwm=1.45;
    }else if (pwm_valueA>=100 && pwm_valueA<180){
      kpwm=1.3;
    }else if (pwm_valueA>=180){
      kpwm=1.1;
    }
    pwm_valueB=int(pwm_valueA*kpwm); //el motor B se considera el
derecho
```

Cuadro de código 6: Rangos de constantes proporcionales

La siguiente alternativa se basa en un controlador proporcional algo más complejo que el anterior. Debido a que en el momento de llevar a cabo esta alternativa no disponíamos de módulo Bluetooth, decidimos implementar dicho control en Simulink antes que, en Arduino, ya que esta plataforma nos ofrecía un mayor número de herramientas y utilidades.

La limitación de implementarlo en Simulink es que tendríamos que transmitir información mediante puerto serie de manera alámbrica, y nuestro robot se podría desestabilizar.

El funcionamiento de nuestro control proporcional es el siguiente: en primer lugar, obtenemos la diferencia entre el valor de referencia que queremos (en nuestro caso 0, ya que queremos ir en línea recta) y nuestro valor actual proporcionado por el sensor. Dicho error será multiplicado por una constante y será añadido (o reducido, esto depende del signo de nuestro error) al valor de PWM de la rueda A.

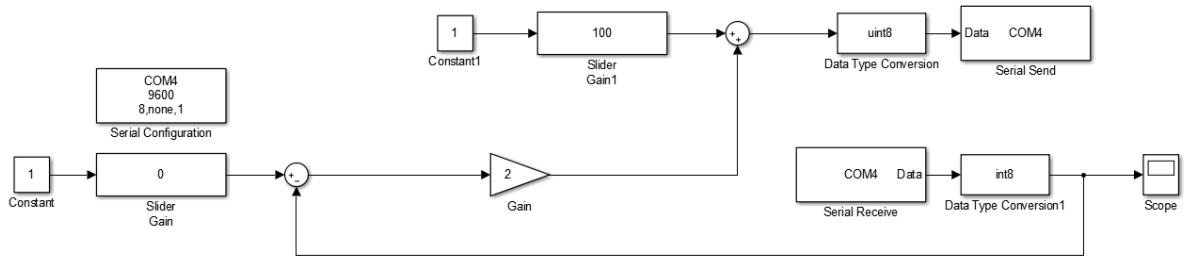

Figura 20: Diagrama de bloques para control de ángulo en Simulink

Para recibir la variación que debemos añadir o reducir nuestro PWM en Arduino se realiza de la siguiente manera:

```
//CONTROL DE PWM (MEDIANTE SIMULINK)
 if (Serial.available()){
   in=Serial.read();
 }
 inA=inA+in; //inA se trata del valor de PWM del motor A
```

Cuadro de código 7: Obtención por puerto serie en Arduino para el control en Simulink

Gracias a Simulink podemos obtener una representación de cómo se comportaba nuestro sistema ante un ángulo de referencia de 90 grados mediante la implementación del control.

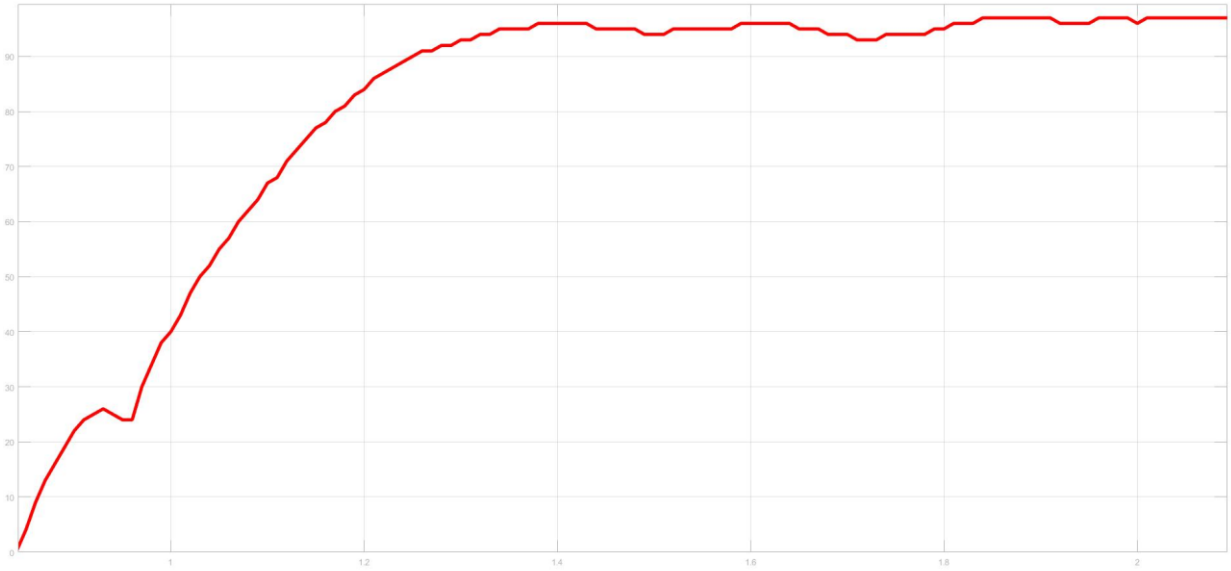

Figura 21: Respuesta del sistema en Simulink

Este control funcionaba regular porque disponíamos del problema de tener un cable permanentemente conectado a nuestra placa Arduino. Como podemos apreciar en la figura anterior, la orientación de nuestro robot varía demasiado cuando ha llegado a la referencia de 90 grados.

Por tanto, una vez entendido el funcionamiento de un controlador, implementamos este mismo control en nuestro microcontrolador, para así poder prescindir de cable. De esta manera, la función para el control del ángulo en Arduino sería de la siguiente manera:

```
//FUNCION CONTROL ANGULO
void programaControlAngulo() {
  e0 = ref - orientation;
  inA = KpA * e0;
}
```

Cuadro de código 8: Función para el control de ángulo con una rueda

Para aplicar dicha función sobre nuestro robot de la misma manera que en Simulink sería mediante las siguientes líneas de código:

```
    //CONTROLAR PWM PARA ANGULO
    programaControlAngulo();

    pwm_valueA = pwm_valueA + inA;

    //SATURACIÓN DE PWM
   if (pwm_valueA > 255) {
      pwm_valueA = 255;
    }
    if (pwm_valueA < 0) {
      pwm_valueA = 0;
    }
```

Cuadro de código 9: Aplicación del control de ángulo con una rueda a nuestro robot

La limitación que encontramos es que, al realizar dicho control, damos por supuesto que uno de los dos motores está funcionando constantemente a un valor de PWM, y, por tanto, cuando realizamos un giro hacia el lado de dicho motor, la curva se realizará lentamente. Si se realiza el giro hacia la rueda que controlamos, el valor de PWM de este motor variaría hasta 0 y la curva se realiza de manera más eficiente.

Por tanto, para mejorar dicho control anterior en Arduino, se realizó aplicándose a ambos motores en vez de solamente a uno. De esta manera, si nuestro robot debe de hace un giro, a la vez que un valor de PWM aumenta el otro disminuye, y el control se realizaría de una forma más eficiente.

Para realizar dicha tarea, tan solo debemos de ejecutar dicha función para ambos motores de la misma manera:

```
void programaControlAngulo() {
  e0 = ref - orientation;
  inA = KpA * e0;
  inB = KpB * e0;
}
```

Cuadro de código 10: Función para el control P de ángulo con ambas ruedas

Para aplicar esta función a nuestro robot, operaremos de manera similar a como lo hicimos para el control sobre una rueda:

```
//CONTROLAR PWM PARA ANGULO
 programaControlAngulo();

 pwm_valueA = pwm_valueA + inA;
 pwm_valueB = pwm_valueB - inB;

 //SATURACION
 if (pwm_valueB > 255) {
   pwm_valueB = 255;
 }
 if (pwm_valueB < 0) {
   pwm_valueB = 0;
 }
 if (pwm_valueA > 255) {
   pwm_valueA = 255;
 }
 if (pwm_valueA < 0) {
   pwm_valueA = 0;
 }
```

Cuadro de código 11: Aplicación del control P de ángulo con ambas ruedas nuestro robot

Como podemos apreciar en la programación de la función para el control, en cada rueda aplicamos una distinta constante proporcional, aspecto importante para contrarrestar la no homogeneidad de la masa de nuestro robot. Posteriormente, probaremos distintas constantes proporcionales para ver cómo se comporta nuestro sistema.

A continuación, vamos a exponer el diagrama de bloques en el cual se aplica el control anterior:

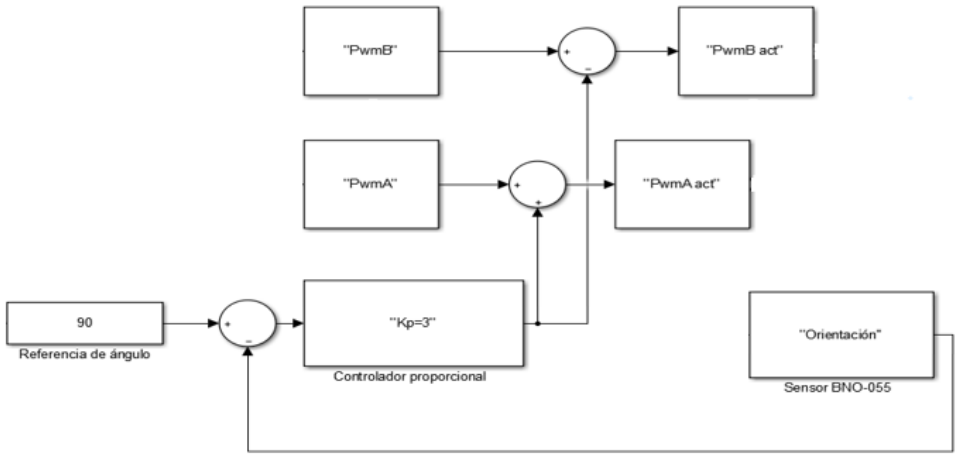

Figura 22: Diagrama de bloques con controlador proporcional a ambas ruedas para la orientación

Una vez elaborado correctamente el control de nuestro ángulo, lo único que nos queda es intentar mejorar su funcionamiento. Como sabemos, nuestro robot es bastante irregular, y mediante un control proporcional no podremos regular si nuestro sistema dispone de un pequeño offset en su respuesta.

Para mejorar nuestro controlador, podemos añadir la acción integral a él, para eliminar cualquier tipo de error en el régimen permanente de nuestro sistema. Para añadir dicha acción en nuestro controlador para ambas ruedas realizamos la siguiente modificación en nuestro código:

```
//CONTROL PI ANGULO
void programaControlAngulo() {
  e0 = ref - orientation;
  inA = KpA * e0;
  inB = KpB * e0;
  Ia=Ia+qi1*e0;
  Ib=Ib+qi2*e0;
}
```

Cuadro de código 12: Función para el control PI de ángulo con ambas ruedas

Donde “Ia” e Ib” son las acciones integrales que añadimos al valor de PWM de nuestros motores para aplicar dicho controlador.

```
//CONTROLAR PWM PARA ANGULO
 programaControlAngulo();
 pwm_valueA = pwm_valueA + inA + Ia;
 pwm_valueB = pwm_valueB - inB - Ib;
 if (pwm_valueB > 255) {
   pwm_valueB = 255;
 }
 if (pwm_valueB < 0) {
   pwm_valueB = 0;
 }
 if (pwm_valueA > 255) {
   pwm_valueA = 255;
 }
 if (pwm_valueA < 0) {
   pwm_valueA = 0;
 }
```

Cuadro de código 13: Aplicación del control PI de ángulo con ambas ruedas nuestro robot

El cálculo de los parámetros de la acción integral se realizará en la función “setup” de Arduino, ya que éstos no varían a lo largo del programa:

```
 qi1=(KpA*(Ts/1000000))/Ti;
 qi2=(KpB*(Ts/1000000))/Ti;
```

Cuadro de código 14: Cálculo de los parámetros para la acción integral

Para no saturar la acción integral debemos de usar un “Anti-WindUp” de la siguiente forma en Arduino:

```
if ((pwm_valueA > 255) | pwm_valueA < 0) { //anti wind-up

  Ia = Ia - qi1 * e0;

}

if ((pwm_valueB > 255) | pwm_valueB < 0) { //anti wind-up

  Ib = Ib - qi2 * e0;

}
```

Cuadro de código 15: Aplicación de Anti-WindUp a nuestra acción integral

Posteriormente, realizaremos diversas pruebas con distintos proporcionales y con distintos Ti para la acción integral, y se analizará cómo varía dicha respuesta de nuestro robot móvil.

Este último control se considera el más completo y con el cual nuestro robot se ajusta más a lo que queremos. La programación completa de Arduino para el correcto funcionamiento de este controlador será expuesta posteriormente en el apartado de Anexos.

Para comprender el funcionamiento de dicho control, vamos a exponer el diagrama de bloques correspondiente:

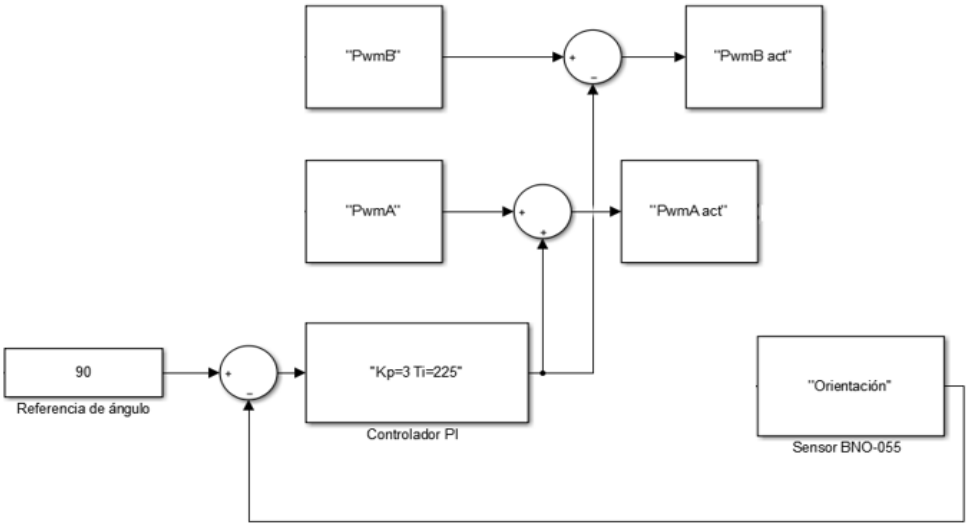

Figura 23: Diagrama de bloques con controlador PI a ambas ruedas para la orientación

Por último, hay que mencionar un problema que nos ofrece este control, y es cuando la referencia es de 0 grados. Como sabemos, nuestro sensor nos ofrece la información desde 0 hasta 360 grados, y, por tanto, si la referencia es de 0 grados y nuestro sensor ofrece una orientación de 359.9, el control se desarrolla y hace dar a nuestro robot una vuelta hasta alcanzar de nuevo la referencia 0, y realizar el mismo problema.

La solución para el problema anterior es obtener la orientación diferenciando entre girar a la derecha e izquierda, es decir, de 0 a 180 sería hacia la derecha y de 0 a (-180) hacia la izquierda. De esta forma, para una referencia de 0 grados no tendremos ese salto de 0 a 360 grados, y no se desarrollará dicho problema. La programación para que nuestro sensor nos ofrezca de esta manera la información es la siguiente:

```
//OBTENER ÁNGULO
 orientation = (event.orientation.x);

 if (ref == 0) {
   if (orientation > 180 && orientation <= 360) {
     orientation = orientation - 360;
   }
 }
```

Cuadro de código 16: Transformación de la orientación

3.2.1.3. Pruebas y resultados

Antes de dar paso a la representación de los resultados obtenidos con cada controlador explicaremos la forma de establecer conexión con nuestro robot. Para establecerla nos basaremos en el módulo Bluetooth que disponemos y la plataforma

Matlab, debido a que establece conexión correctamente con nuestro módulo y nos permite almacenar los datos recibidos en matrices. Gracias a dichas matrices y la cantidad de herramientas que dispone Matlab podremos representar la diferente información que recibimos, y así exponer los resultados de la respuesta de nuestro sistema ante los diferentes controladores que serán probados.

Como sabemos, el módulo HC-06, funciona como si éste se tratara de un puerto serie, pero inalámbrico. Por tanto, para el envío de datos desde Arduino se realizará de la misma manera, con la diferencia de que los datos deben ser enviados transformados a “String” y separados mediante barras para poder separar los datos correctamente en Matlab.

```
String distancia = String(cm);
String pwm = String(pwm_motor);
Serial.println(pwm + "/" + distancia + "/");
```

Cuadro de código 17: Envío de datos desde Arduino

Un aspecto importante para el envío de datos es el tiempo de muestreo de éstos. En nuestro caso, éste debe de cumplirse en cada envío para poder representar correctamente dichos datos. A su vez, recobra más importancia debido a que implementaremos una GUI en la cual representaremos en vivo toda la información que podemos obtener mediante los sensores. El funcionamiento de dicha GUIDE será explicado posteriormente en los anexos.

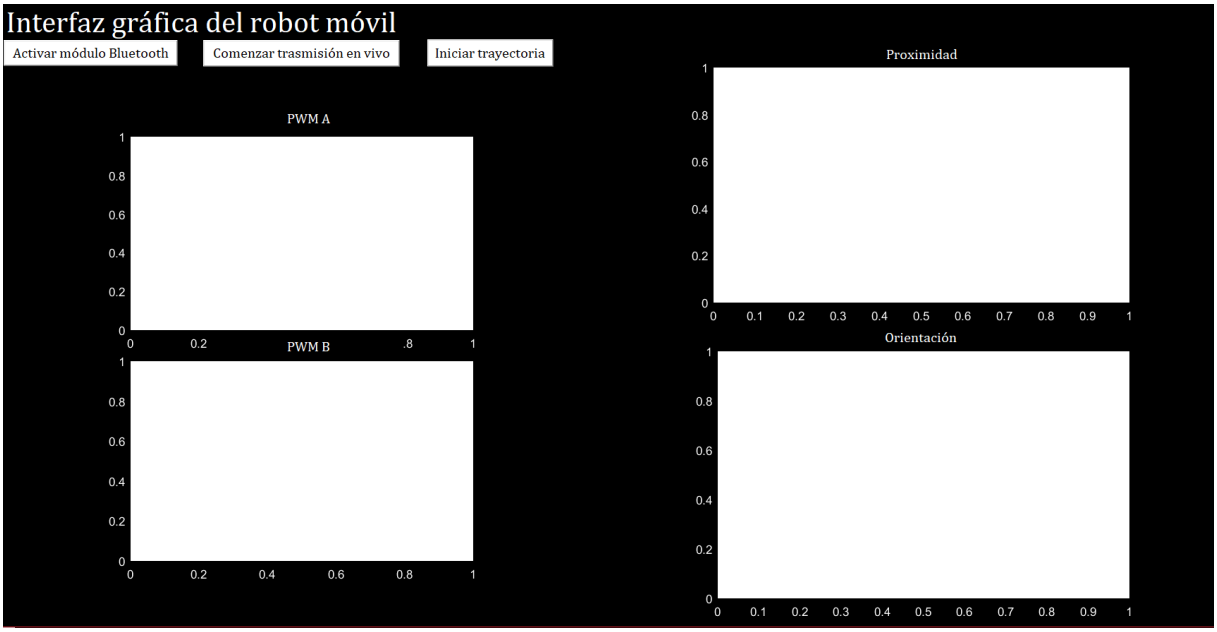

Figura 24: GUI para representar en vivo

En dicha GUI (Graphic User Interface) representaremos los valores de PWM de ambos motores y la orientación y proximidad de nuestro robot.

Para enviar los datos en cada iteración (una iteración dura un definido tiempo de muestreo) lo realizaremos de la siguiente manera en Arduino:

```
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 18: Programación para establecer un tiempo de muestreo en Arduino

Una vez hemos obtenido el vector de datos desde Arduino, hemos de ejecutar un programa en Matlab encargado de separar dicha información y almacenarla en matrices. Se trata del siguiente programa:

```
%Líneas de comando para conectar módulo Bluetooth
s=Bluetooth('HC-06',1);
fopen(s);

num_muestras=6200;
muestra=1;
num_columnas=1;
j=1;

%En esta parte leemos el vector de datos que nos viene de
arduino
%Separamos y guardamos 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;
```

Cuadro de código 19: Función de Matlab para almacenar información recibida

En la siguiente figura podemos observar la utilidad de dicha función de Matlab. En esta matriz tenemos recogidos en la primera columna los datos del PWM aplicados a los dos motores y en la siguiente columna la proximidad.

	1	2	3	4	5	6	7	8
1	232	57						
2	232	56						
3	232	55						
4	232	54						
5	232	53						
6	232	52						
7	232	51						
8	232	49						
9	232	48						
10	232	47						
11	232	46						
12	232	44						
13	232	43						
14	232	42						
15	232	41						
16	232	40						
17	232	39						
18	232	38						
19	232	36						
20	158	35						
21	158	34						

Figura 25: Datos almacenados en matriz

Por tanto, para finalizar la representación de los datos, lo último que debemos de hacer es crear una gráfica donde representarlos. Gracias la plataforma Matlab tenemos la posibilidad de variar el tipo de línea, el grosor de la misma y añadir más variedad de elementos. Para realizar la representación hemos recurrido a la siguiente función de Matlab.

```
    [filas columnas]=size(matriz);
    figure; hAxes(1) = gca;
    hAnimatedLine(1) = animatedline('Parent',gca);
    hAnimatedLine(1).Color = 'red';
    hAnimatedLine(1).LineWidth=3;
    muestra=1;
    while muestra<=filas
        hAxes(1).XLim = [0 muestra]
        hAxes(1).YLim = [0 380]
        addpoints(hAnimatedLine(1),muestra,matriz(muestra,2));
        drawnow
        muestra=muestra+1;
    end
```

Cuadro de código 20: Representación de datos en Matlab

Si aplicamos esta función a los datos del ejemplo de la proximidad obtendríamos la siguiente representación gráfica:

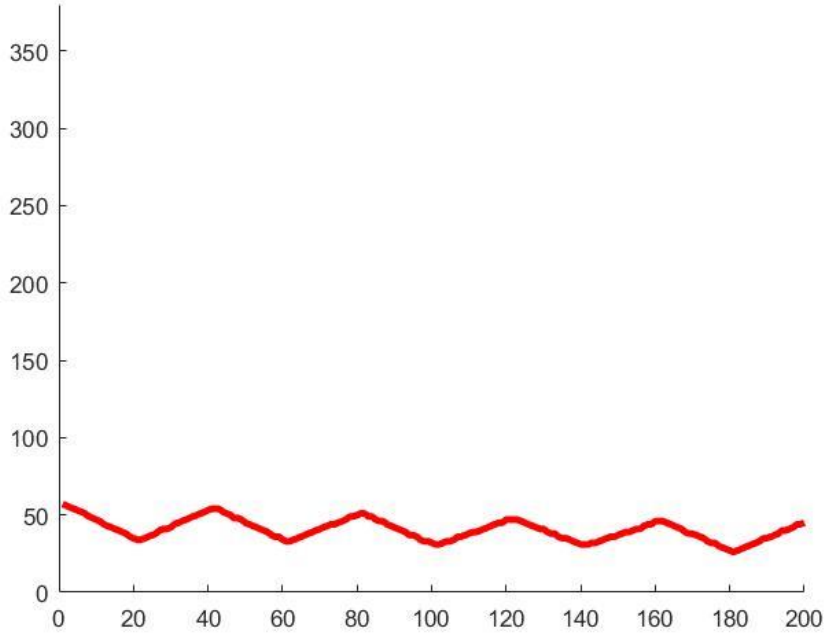

Figura 26: Representación gráfica de los datos

Una vez disponemos de las herramientas necesarias para representar nuestra información, procederemos a realizar las distintas pruebas de nuestro controlador implementado.

Para realizar dichas pruebas lo haremos de la siguiente manera: nuestro robot tendrá de referencia inicial un ángulo de 0 grados, y mediante Bluetooth le mandaremos una referencia de ángulo (en nuestro caso serán 90 grados) a nuestro microcontrolador para que nuestro robot varíe su orientación e iremos recopilando la información de nuestro sensor para representar la respuesta de nuestro sistema.

A continuación, mostramos la programación para enviar y recibir la referencia que queremos conseguir con nuestro robot:

//Mandar referencia mediante Matlab

```
%Conectamos dispositivo Bluetooth
s = Bluetooth('HC-06',1);
fopen(s);
fwrite(s,90);
```

//Recibir referencia con Arduino

```
if (Serial.available()) {
   ref = Serial.read(); //referencia de  ángulo
}
```

Cuadro de código 21: Activación de módulo Bluetooth. Envío y recibo de información mediante Bluetooth

A continuación, damos paso a la variedad de pruebas que realizaremos con este control:

- **Control P de un motor con Kp=4 y ref=90**

En este caso implementamos un control proporcional con una constante de 3, y en la siguiente figura, observamos la respuesta de nuestro robot al variar su ángulo hasta obtener el ángulo de referencia de 90 grados. También, representaremos el valor de PWM del motor que controlamos y del que no varía, cuyo valor de PWM es de 80 constantemente.

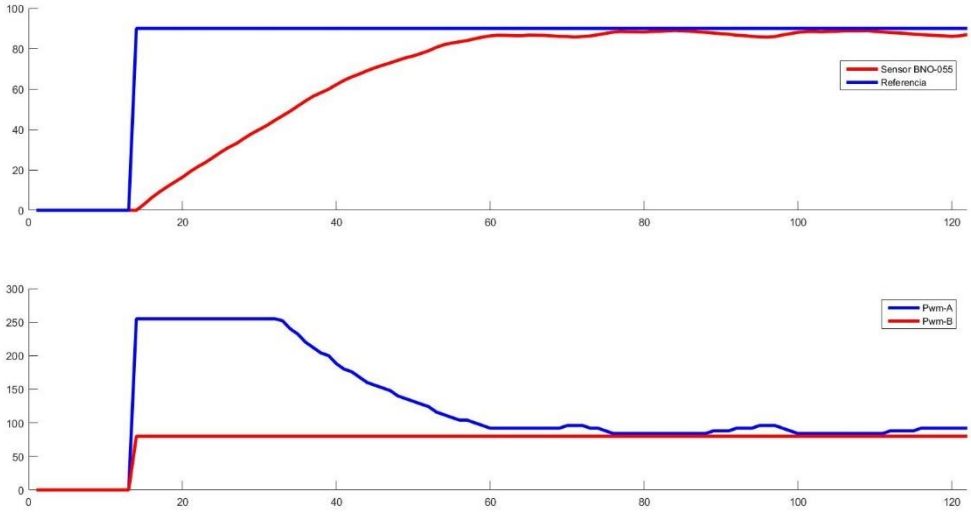

Figura 27: Respuesta de orientación y valores de PWM con control proporcional aplicado a una rueda

Como podemos comprobar, nuestro robot gira relativamente lento por la limitación que este control conllevaba. Además, una vez hemos obtenido el valor de referencia disponemos de un offset medianamente grande comparado con los controladores expuestos posteriormente.

- **Control P de ambos motores con KpA=3 (motor A) y KpB=3 (motor B)**

Se trata de un controlador proporcional con la misma constante proporcional para ambos motores, con valor de 3. A continuación, se muestra la respuesta de la orientación de nuestro robot y la de ambos valores de PWM:

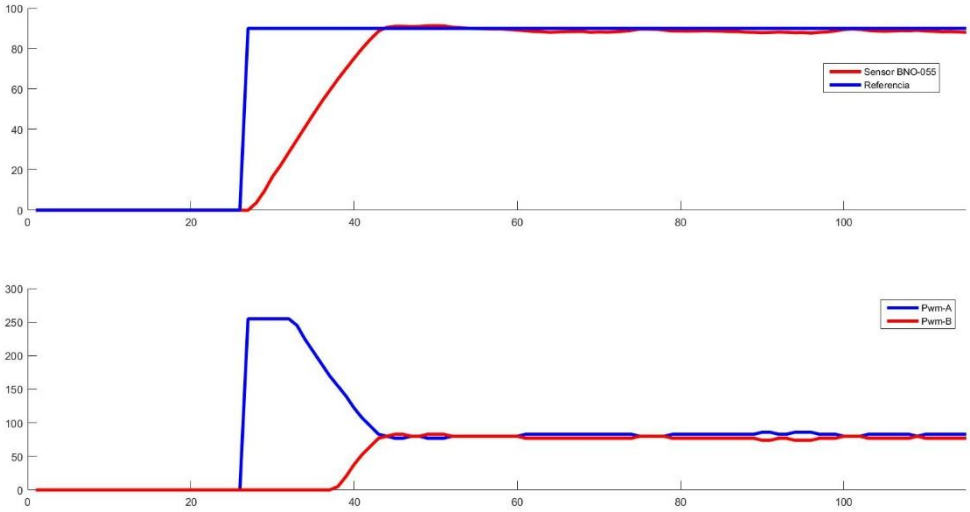

Figura 28: Respuesta de orientación y valores de PWM con control proporcional aplicado a ambas ruedas

Gracias a dicho controlador, obtenemos una respuesta bastante precisa y sin oscilaciones bruscas. En menor medida que el anterior controlador, éste nos ofrece también un offset, aunque se tratan de uno o dos grados arriba o abajo de la referencia que deseamos.

- **Control PI sobre ambos motores con KpA=3, KpB=3 y Ti=225**

Este controlador se ha implementado para corregir la respuesta anterior, en concreto, para corregir el pequeño offset que conlleva dicho controlador. Tras la realización de diversas pruebas, hemos llegado a la conclusión de añadir una acción integral con un Ti=225:

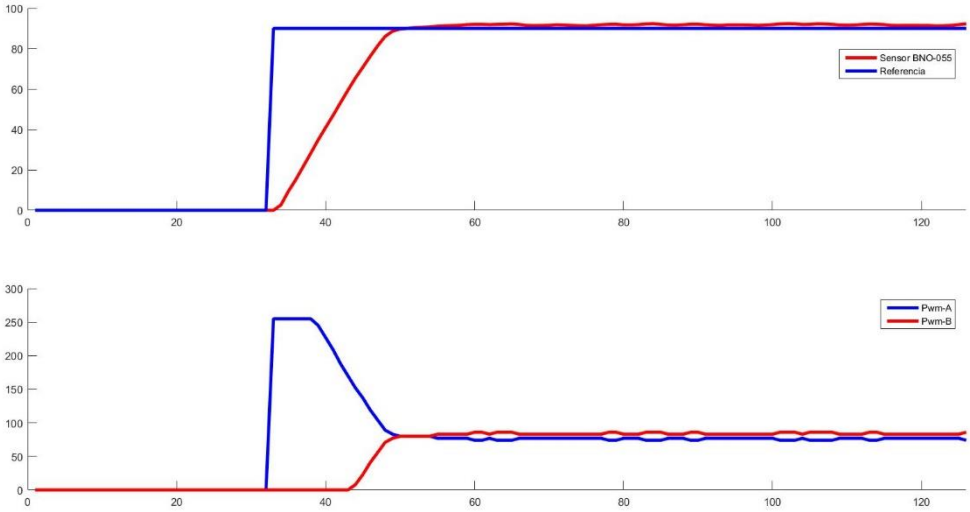

Figura 29: Respuesta de orientación y valores de PWM con control PI aplicado a ambas ruedas

Al aplicar este controlador, nuestra respuesta reduce considerablemente el pequeño offset que se producía con el controlador anterior. Finalmente, para controlar la orientación de nuestro robot, nos hemos decantado por implementar este controlador.

3.2.2. Control de proximidad

Dicho control funcionará de la misma manera que el control anterior. Es decir, nuestro controlador variará el valor de los PWM de nuestros motores para conseguir el valor de referencia de proximidad que queremos alcanzar.

3.2.2.1. Objetivos

Los objetivos que buscamos con este control tienen relación con la seguridad de nuestro robot. Es decir, gracias a este control evitaremos que el robot colisione con la superficie más próxima que tenga enfrente, y, por tanto, no peligrará de ser dañado.

Como en el control anterior, también podremos enviar referencias a nuestro robot. En este caso, las referencias serán sobre la distancia a la que éste se debe inmovilizar. Como es obvio, cuanto mayor sea el valor de la referencia enviada, menos se aproximará nuestro robot a la superficie y nos aseguraremos en mayor medida.

Por último, cabe destacar sobre este control que gracias a él vamos a regular la velocidad de nuestro robot. Es decir, si el robot se encuentra lejos de la referencia de proximidad, el robot se desplazará a la velocidad límite que se impone. Justo lo contrario ocurrirá cuando se aproxime a la referencia, ya que disminuirá la velocidad de nuestro robot.

3.2.2.2. Alternativas para el control

En este apartado hablaremos sobre las alternativas que hemos seguido para elaborar dicho controlador. La realización de éste nos será más sencillo ya que anteriormente realizamos el control de ángulo, y éste no va a ser muy distinto en cuanto a programación se refiere.

Al contrario que para el control anterior, en este no podemos aplicar el control en una sola rueda, ya que para que el robot quede inmovilizado por completo deben de estar los dos motores parados. Por tanto, tuvimos que implementar un control que accionara en las dos ruedas de nuestro robot. A continuación, exponemos la función en Arduino para realizar este control de proximidad:

```
void programaControlProximidad() {
  e1 = cm - refd;
  in1 = Kpd * e1;
}
```

Cuadro de código 22: Función para controlar la proximidad de nuestro robot

Para realizar este control, hemos usado tan solo una constante proporcional para ambos motores, debido a que este control solamente será utilizado cuando se aproxime a la superficie. Además, si en cualquier caso una rueda frenara más que la otra (por el tema de falta de homogeneidad de masa), hemos insertado en nuestra programación el control de ángulo para que nuestro robot no se desvía a medida que se va acercando a dicho obstáculo.

Dicha constante proporcional nos variará cuánto de rápido se detendrá nuestro robot a medida que se va desplazando, es decir, cuanto mayor sea dicha constante, más rápido se detendrá éste. Posteriormente, en los resultados que hemos obtenido aplicando dicho controlador, observaremos el efecto que conlleva el aplicar una constante u otra.

Por tanto, una vez tenemos la función para realizar dicho control, lo siguiente que debemos de hacer es insertarlo en la programación del funcionamiento del robot móvil de la siguiente manera:

```
//CONTROLAR PWM PARA PROXIMIDAD
 programaControlProximidad();
 pwm_valueA = pwm_valueA + in1;
 pwm_valueB = pwm_valueB + in1;

 //EN CASO DE QUE SATURE
 if (pwm_valueA > 255) {
   pwm_valueA = 255;
 }
 if (pwm_valueA < 0) {
   pwm_valueA = 0;
 }

 if (pwm_valueB > 255) {
   pwm_valueB = 255;
 }
 if (pwm_valueB < 0) {
   pwm_valueB = 0;
 }
```

Cuadro de código 23: Aplicación de control de proximidad en el funcionamiento de nuestro robot

Una vez hemos expuesto la programación para dicho control de proximidad, a su vez, mostraremos el diagrama de bloques que muestra su funcionamiento:

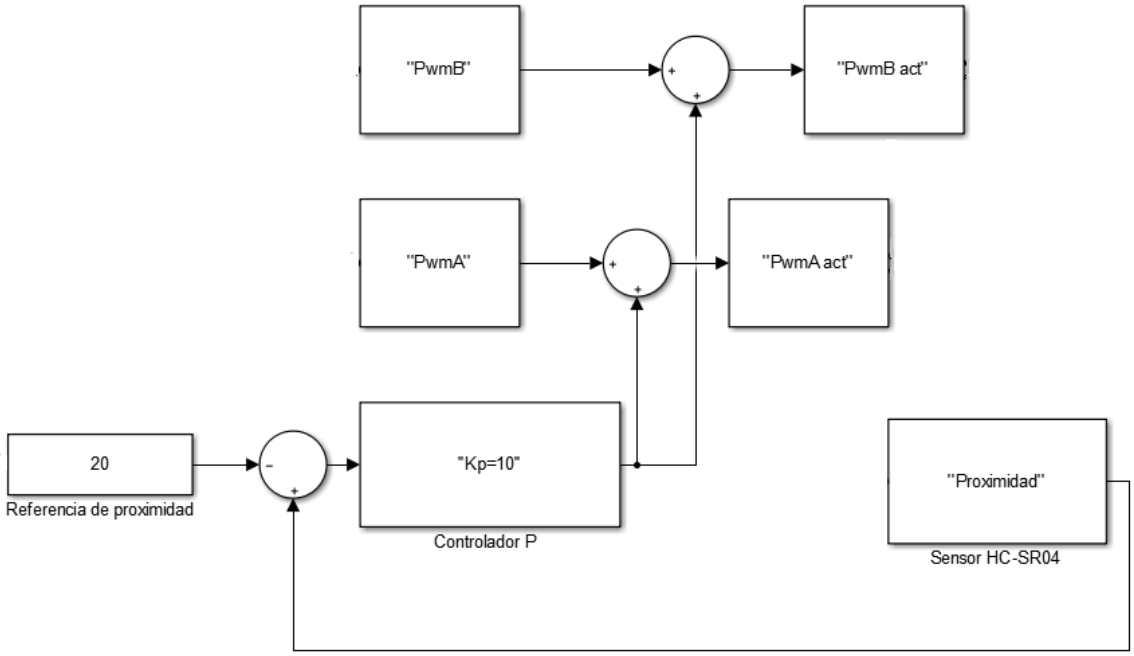

Figura 30: Diagrama de bloques con controlador P a ambas ruedas para proximidad

3.2.2.3. Pruebas y resultados

En este apartado mostraremos los resultados obtenidos mediante la prueba que realizamos con nuestro robot para comprobar que dicho controlador funciona a la perfección. Dicha prueba consistirá en colocar a nuestro robot a unos 90 centímetros de distancia, y definir como referencia unos 20 centímetros de la superficie más próxima. En nuestro controlador de proximidad, la constante proporcional tendrá un valor de 10.

A continuación, representamos el valor de la proximidad de nuestro robot, y a su vez, los valores de PWM de cada uno de los motores:

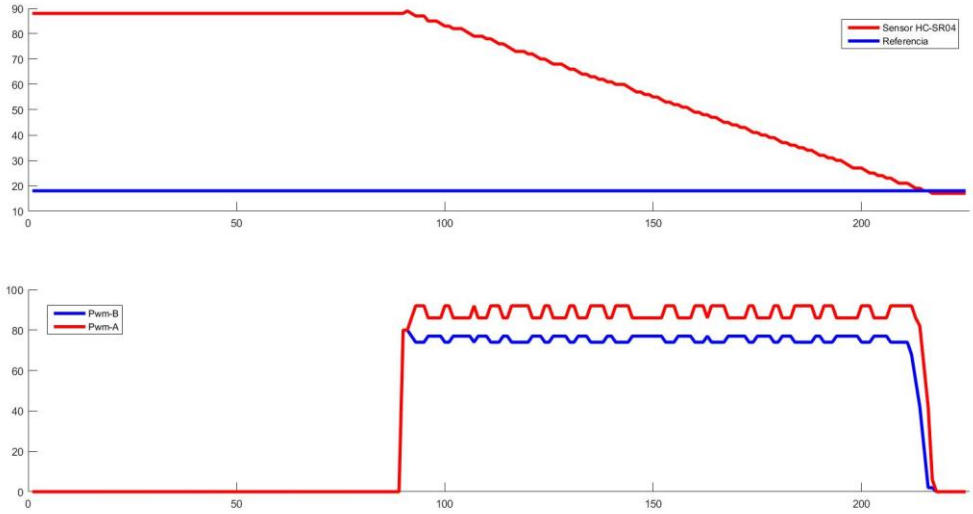

Figura 31: Respuesta de la proximidad y PWM con controlador para proximidad

Como podemos observar en las figuras, mientras el robot se desplaza linealmente el control de ángulo se realiza y varía los valores de PWM para no desviarse, y cuando alcanza la referencia de proximidad, se reducen dichos valores hasta llegar a un valor de 0.

3.2.3. Seguimiento de trayectorias

Esta se trata de la última estrategia de control que se ha implementado en nuestro robot móvil. Se trata de la más completa debido a que en ella aprovechamos la utilidad de los controles anteriormente mencionados para realizar una trayectoria propuesta por nosotros.

Cabe destacar que para seguir cualquier trayectoria que deseemos ambos controles expuestos anteriormente son de vital importancia. Para empezar, el control de ángulo nos servirá en primer motivo para no desviarse de la trayectoria si ésta es lineal, y, en segundo lugar, nos servirá para realizar giro si la trayectoria lo requiere, como puede ser un triángulo.

Por otro lado, usamos el control de proximidad para disponer de seguridad, es decir, si nuestro robot debe realizar un giro que lo efectúe con menor velocidad si éste se encuentra aproximado a una superficie. O, si por algún casual, nuestro robot se aproxima demasiado a la superficie, que éste se detenga y así evitar el choque.

Un último aspecto antes de dar paso a la trayectoria que hemos propuesto es que ésta se realizará a partir de un entorno controlado, y, nuestro robot, de manera autónoma, se deberá desplazar en dicho entorno para formar la trayectoria deseada.

3.2.3.1. Trayectoria cuadrada

La trayectoria cuadrada es la propuesta para realizar con nuestro robot móvil. Como hemos dicho anteriormente, nuestro robot realizará dicha trayectoria a partir de un entorno controlado. En nuestro caso, para realizar una trayectoria cuadrada, hemos realizado el siguiente montaje:

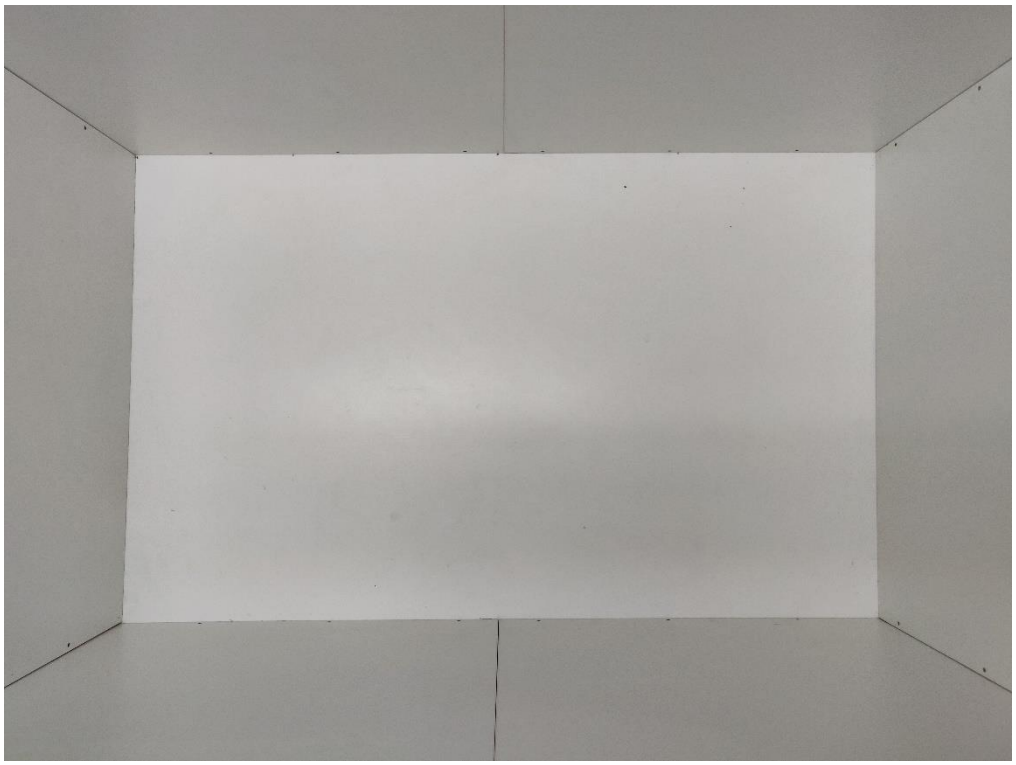

Figura 32: Recinto controlado para realizar trayectoria cuadrada

Como podemos observar se trata de una estructura formada por 4 paredes, formando en este caso, un rectángulo. Así de esta manera, cuando nuestro robot llegue a cualquiera de las superficies, tendrá que girar 90 grados cada vez que ocurra, y finalmente, formar un cuadrado como trayectoria.

Antes de dar paso a mostrar la programación para realizar dicha trayectoria, debemos aplicar los controles de ángulo y de proximidad a nuestro robot, cuyas funciones son las vistas siguientes.

```
//CONTROL ANGULO
void programaControlAngulo() {
  e0 = ref - orientation;
  inA = KpA * e0;
  inB = KpB * e0;
  Ia=Ia+qi1*e0;
  Ib=Ib+qi2*e0;
}
//CONTROL PROXIMIDAD
void programaControlProximidad() {
  e1 = cm - refd;
  in1 = Kpd * e1;
}
```

Cuadro de código 24: Funciones de control de ángulo y de proximidad

Solamente con la aplicación de ambas funciones conseguiremos que nuestro robot varíe su velocidad conforme se aproxima a una superficie y no desviarse de la referencia inicial, que en nuestro caso es de 0 grados. Para aplicar en nuestra programación y que tenga la utilidad que hemos mencionado lo realizaremos de la siguiente manera:

```
 //CONTROLAR PWM PARA PROXIMIDAD
  programaControlProximidad();
  pwm_valueA = pwm_valueA + in1;
  pwm_valueB = pwm_valueB + in1;
  //EN CASO DE QUE SATURE
  if (pwm_valueA > 80) {
    pwm_valueA = 80;
  }
  if (pwm_valueA < 0) {
    pwm_valueA = 0;
  }
  if (pwm_valueB > 80) {
    pwm_valueB = 80;
  }
  if (pwm_valueB < 0) {
    pwm_valueB = 0;
  }
  //CONTROLAR PWM PARA ANGULO UNA VEZ SE HA ESTABLECIDO LA VELOCIDAD
DE NUESTRO ROBOT
  programaControlAngulo();
  pwm_valueA = pwm_valueA + inA + Ia;
  pwm_valueB = pwm_valueB - inB - Ib;
  if (pwm_valueB > 255) {
    pwm_valueB = 255;
  }
  if (pwm_valueB < 0) {
    pwm_valueB = 0;
  }
  if (pwm_valueA > 255) {
    pwm_valueA = 255;
  }
  if (pwm_valueA < 0) {
    pwm_valueA = 0;
  }
```

Cuadro de código 25: Aplicación de control de ángulo y de proximidad en nuestro robot

La siguiente tarea es la de aplicar la programación para que nuestro robot realice la trayectoria que deseamos. En nuestro caso, al tratarse de una trayectoria cuadrada, lo realizaremos de la siguiente manera: en primer lugar, empezaremos con referencia 0, y a continuación, trabajaremos siempre de la misma manera. Es decir, cuando nuestro robot se acerque a una distancia corta de la siguiente superficie, cambiaremos el valor de la referencia sumando 90 grados, y así sucesivamente hasta llegar a 360 que equivale a 0 grados.

Una vez obtenemos una referencia de 360 o 0 grados quiere decir que hemos realizado la trayectoria satisfactoriamente. La programación en Arduino para dicho control sería la siguiente:

```
   if (e1 < (10) && (e1 >= 1) && giro == false && completo == false)
{  //siendo e1 la diferencia entre referencia y prox. actual
      ref = ref + 90;
      if (ref == 360) {
        ref = 0;
        completo = true;
      }
      giro = true;
    }
    if (giro == true && e1 >= 30) {
      giro = false;
    }
```

Cuadro de código 26: Programación para el desarrollo de una trayectoria cuadrada

Cuando se realizaba el giro teníamos el problema de que nuestro robot seguía estando cerca de la superficie, y, por tanto, mientras se daba esta situación, nuestra referencia sumaba constantemente 90 grados hasta que realizaba dicho giro. Por tanto, insertamos un booleano en nuestra programación para que mientras se desarrollaba el giro, no permitiera variar la referencia.

Para que nuestro robot no realice una trayectoria cuadrada infinita, hemos tenido que insertar una variable booleana para controlar que no se llegue a esta situación. De esta forma, cuando nuestro robot alcance la referencia de 0 grados por segunda vez (ya ha realizado un cuadrado), no sumará 90 grados a la referencia cuando se aproxime a la superficie más próxima.

3.2.3.2. Resultados obtenidos

En este apartado expondremos los resultados que hemos obtenido una vez se ha implementado esta estrategia de control. Hemos realizado varias pruebas al realizar dicha trayectoria. En las siguientes pruebas, el control de proximidad será el mismo en ambas pruebas, debido a que su función será la de regular la velocidad a medida que se aproxima a las superficies.

- **Trayectoria con control P con KpA=3 y KpB=3:** De primeras, hemos realizado dicha trayectoria con el proporcional con una K de valor 3 para el motor izquierdo y de valor 6 para el motor derecho. A continuación, se muestra la respuesta de la orientación, los valores de PWM y los valores de la proximidad a las distintas superficies:

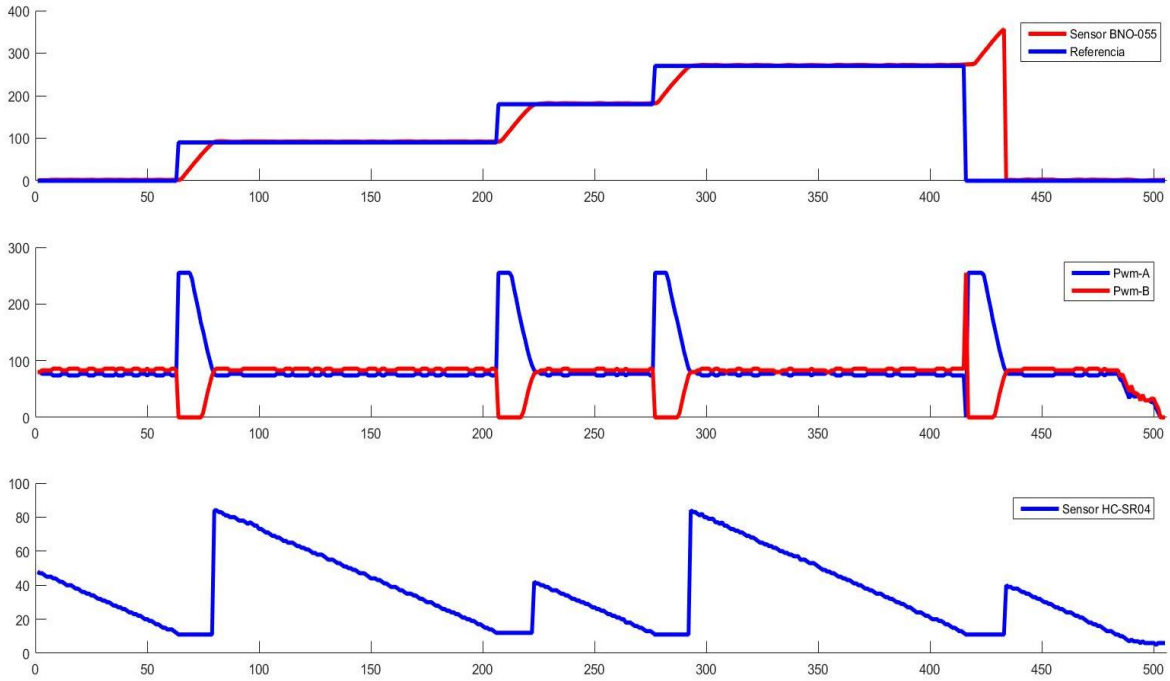

Figura 33: Respuestas de las diversas variables ante un control P para el seguimiento de una trayectoria

Como vemos, gracias a este control realizamos la tarea correctamente, ya que cada vez que nos aproximamos a la superficie más próxima, nuestro robot cambia de referencia. Si observamos más de cerca la figura anterior, se puede observar la limitación que nos conllevaba el uso de este controlador, es decir, el offset de dos o tres grados respecto a la referencia.

Como vemos en la tercera gráfica podemos afirmar que estábamos tratando en un entorno rectangular, donde los picos más altos corresponden a los lados más largos, y los picos bajos a los lados cortos.

- **Trayectoria con control PI con KpA=3, KpB=3 y Ti=225**

Con el uso de este controlador intentaremos regular el offset que albergamos al realizar la trayectoria. A continuación, mostramos la respuesta del sistema:

Como se puede comprobar, el offset se elimina en gran parte, y nuestro robot alcanza correctamente las referencias en todo momento. Para completar la comprobación de dicho controlador, mostramos también la respuesta de las variables más importantes:

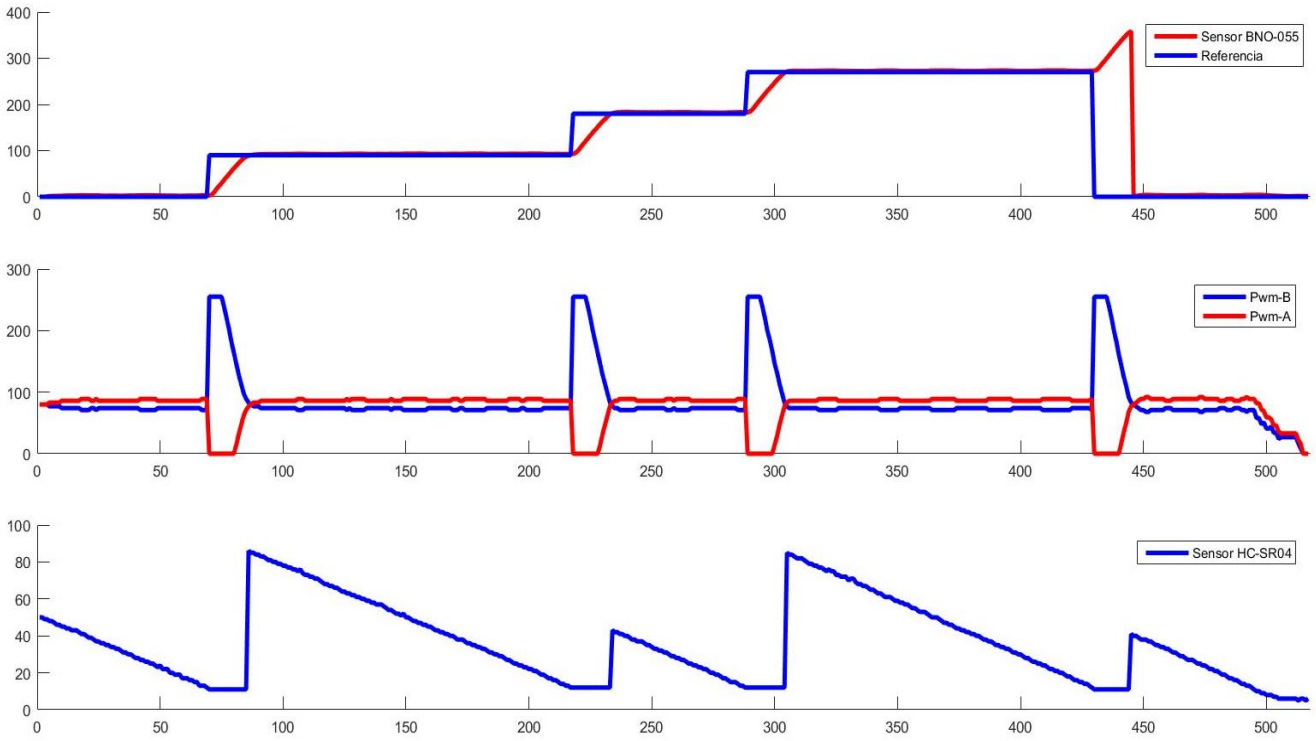

Figura 34: Respuestas de las diversas variables ante un control PI para el seguimiento de una trayectoria

3.2.4. Control de velocidad

Este control sigue el mismo funcionamiento que los controles anteriores, es decir, en variar los valores de PWM de nuestros motores para conseguir la velocidad lineal deseada.

3.2.4.1. Objetivos

Como hemos dicho anteriormente, el control de proximidad nos varía la velocidad a medida que se aleja de la pared o se acerca, de tal forma que nuestro robot se desplazaba a su velocidad máxima cuando se encontraba lejos de la superficie.

Gracias a este controlador podremos mandarle a nuestro robot una referencia de velocidad máxima que queremos alcanzar, y que éste la consiga para realizar la tarea que nosotros deseamos, como puede ser el seguimiento de trayectorias.

3.2.4.2. Alternativas para el control

Anteriormente, en el apartado de las alternativas de uso para el sensor de proximidad, hablamos de que podíamos conseguir la velocidad lineal de nuestro robot derivando la proximidad respecto al tiempo. La primera prueba la realizamos enviando información por puerto serie (cable USB) y nuestro robot se desestabilizaría y los datos de velocidad serían erróneos.

La siguiente prueba que realizamos fue directamente aplicando la derivada de la proximidad en nuestro microcontrolador, y así evitar el uso de cable. Como sabemos, nuestro sensor ultrasónico nos ofrecía información coherente, pero a su vez nos ofrecía bastante ruido, por lo que, al derivar dicha información, este ruido aumentará en el valor de la velocidad. Para derivar dicha proximidad hemos procedido de la siguiente forma en Arduino:

```
 cm = ping(TriggerPin, EchoPin);
 velocidad = (cm_anterior - cm) / 0.1; //0.1 equivale al tiempo de
muestreo
 cm_anterior = cm;
```

Cuadro de código 27: Función para obtener la velocidad lineal

En la siguiente figura, representamos los valores de dicha velocidad obtenida cuando nuestro robot se desplaza linealmente con el máximo valor de PWM para sus motores.

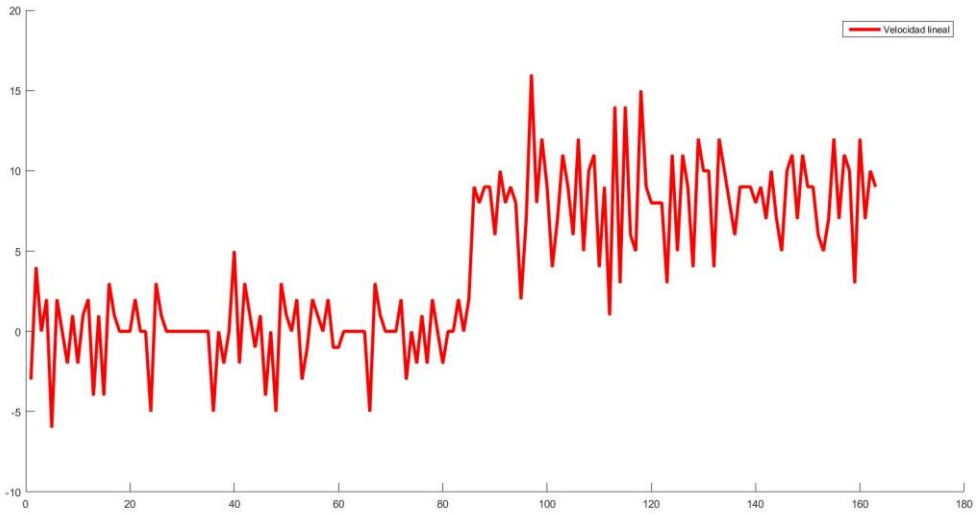

Figura 35: Representación de la velocidad lineal

Como podemos apreciar, la velocidad obtenida mediante la derivada de la proximidad tiene bastante ruido, por lo que nos dificultará el proceso de control. Por tanto, para intentar disminuir dicho ruido, aplicamos un filtro pasa-baja a la velocidad. Para aplicar dicho filtro en Arduino, lo hicimos mediante la siguiente función:

```
float LPvelocidad(float value)
{
  EMA_LPv = ALPHA * value + (1 - ALPHA) * EMA_LPv;
  return EMA_LPv;
}
```

Cuadro de código 28: Función para filtrar la velocidad

Gracias a la aplicación de dicho filtro conseguimos disminuir el ruido de la velocidad, aunque como vemos en la siguiente figura, no disminuye lo suficiente como para conseguir controlar a nuestro robot.

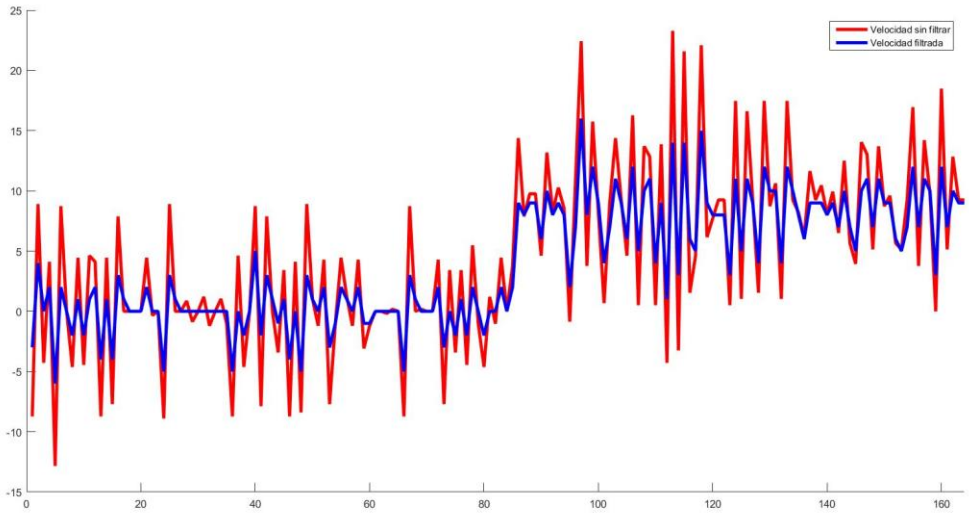

Figura 36: Comparación entre velocidad filtrada y sin filtrar

Debido a que de esta manera no podemos controlar la velocidad, vamos a trabajar de forma distinta con nuestra información para intentar disminuir aún más el ruido. Dicha forma se basa en aplicar un filtro pasa-baja a la información de la proximidad, derivarla para obtener la velocidad y aplicar de nuevo un filtro pasa-baja a dicha velocidad obtenida.

El filtro que aplicamos para disminuir el ruido de la proximidad es el siguiente:

```
float LPproximidad(float value)
{
  EMA_LPp = ALPHA * value + (1 - ALPHA) * EMA_LPp;
  return EMA_LPp;
}
```

Cuadro de código 29: Función para filtrar la proximidad

En la siguiente figura podemos observar la función de dicho filtro en la proximidad obtenida, y cómo disminuye el ruido que nos ofrece nuestro sensor ultrasónico:

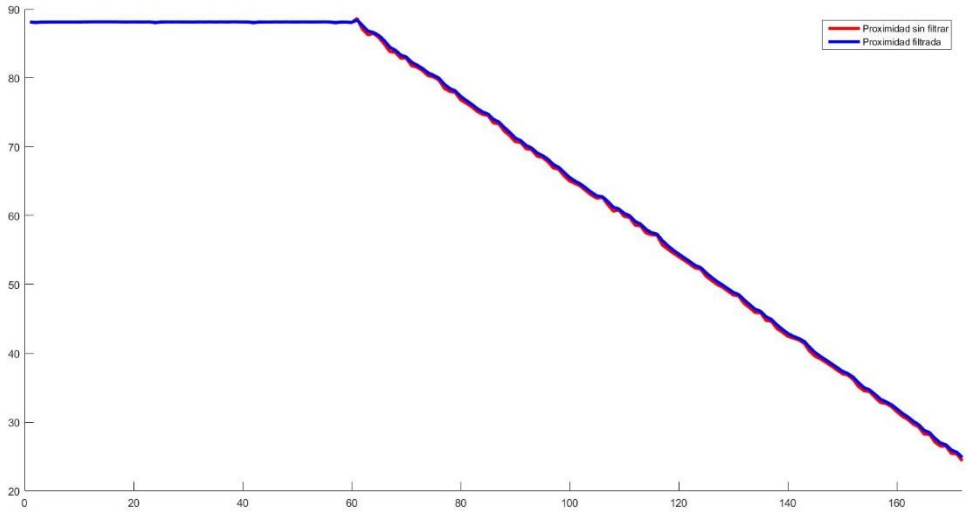

Figura 37: Comparación entre proximidad filtrada y sin filtrar

El siguiente paso que vamos a realizar es el de derivar dicha proximidad filtrada para obtener la velocidad, y filtrar ésta para disminuir aún más dicho ruido. En la siguiente figura podemos apreciar la diferencia entre la velocidad calculada aplicando filtro y sin aplicar.

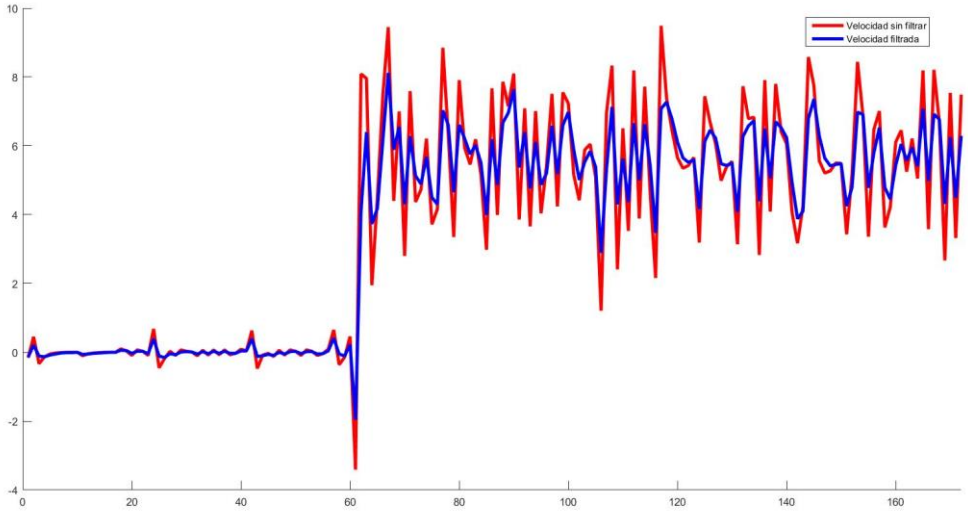

Figura 38: Comparación entre velocidad filtrada y sin filtrar tras aplicar filtro a la proximidad

De esta manera, la velocidad de nuestro robot filtrada disminuye bastante el ruido, por lo que intentaremos aplicar un control de velocidad con dicha información. El control aplicado funciona de manera similar al control de ángulo y proximidad. A continuación, se exponen la función en Arduino para dicho control:

Para dicho control hemos seleccionado una constante proporcional de valor 40 y un valor de referencia de 6 centímetros por segundo. Debido al ruido de la velocidad, el robot no funciona de manera correcta con dicho control. A continuación, mostramos la velocidad de nuestro robot comparada con la referencia y los valores de PWM de los motores de nuestro robot:

```
void programaControlVelocidad() {
  e2 = ref_vel - velocidad_filt;
  inv = kpv * e2;
}
```

Cuadro de código 30: Función para control de la velocidad

Como observamos en la figura anterior, los valores de PWM se ven alterados debido al ruido de la señal de la velocidad, y, por tanto, nuestro robot no funciona correctamente. Cabe destacar que a la vez que aplicamos el control de la velocidad también aplicamos el control de ángulo para que nuestro robot se desplazara de manera lineal. Debido a esto, podemos observar en la figura que los valores de PWM para cada motor son distintos.

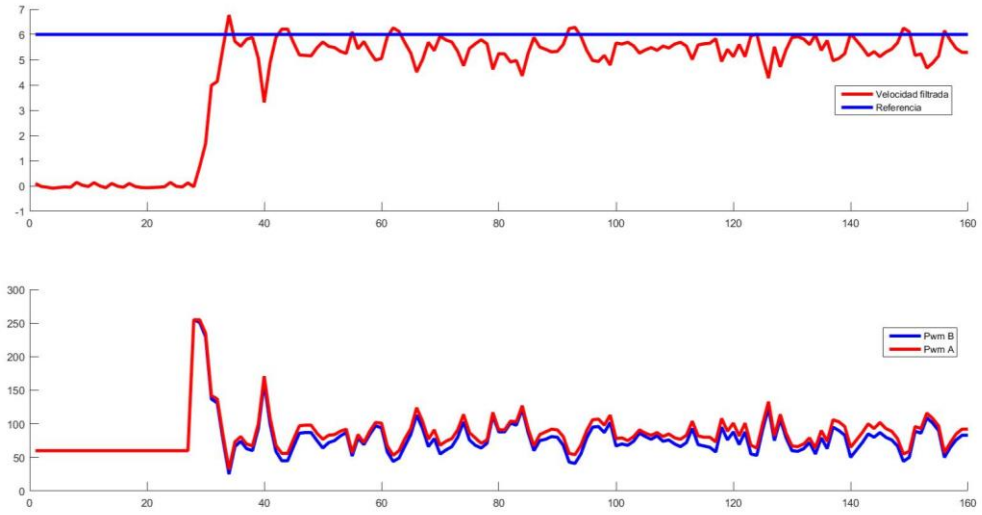

Figura 39: Representación de la velocidad y valores de PWM de nuestro robot

Debido a que este control anterior no tuvo éxito, vamos a proceder a aplicar un filtro pasa-baja al error entre la velocidad de referencia y la que lleva nuestro robot, y aplicar de nuevo el control con dicho error filtrado. La programación para el filtro pasa-baja para el error es la siguiente:

```
float LPerror(float value)
{
  EMA_LPe = EMA_ALPHA * value + (1 - EMA_ALPHA) * EMA_LPe;
  return EMA_LPe;
}
```

Cuadro de código 31: Función para filtrar el error en nuestro control

Por tanto, una vez aplicamos también dicho filtro al error, mostramos de nuevo la velocidad de nuestro robot comparada con la referencia y los valores de PWM de los motores de nuestro robot aplicando el control:

Como vemos en la figura anterior, la respuesta de la velocidad tiene menos ruido que la anterior, pero los valores de PWM de nuestros motores siguen siendo alterados, por lo que nuestro robot no funciona de manera correcta.

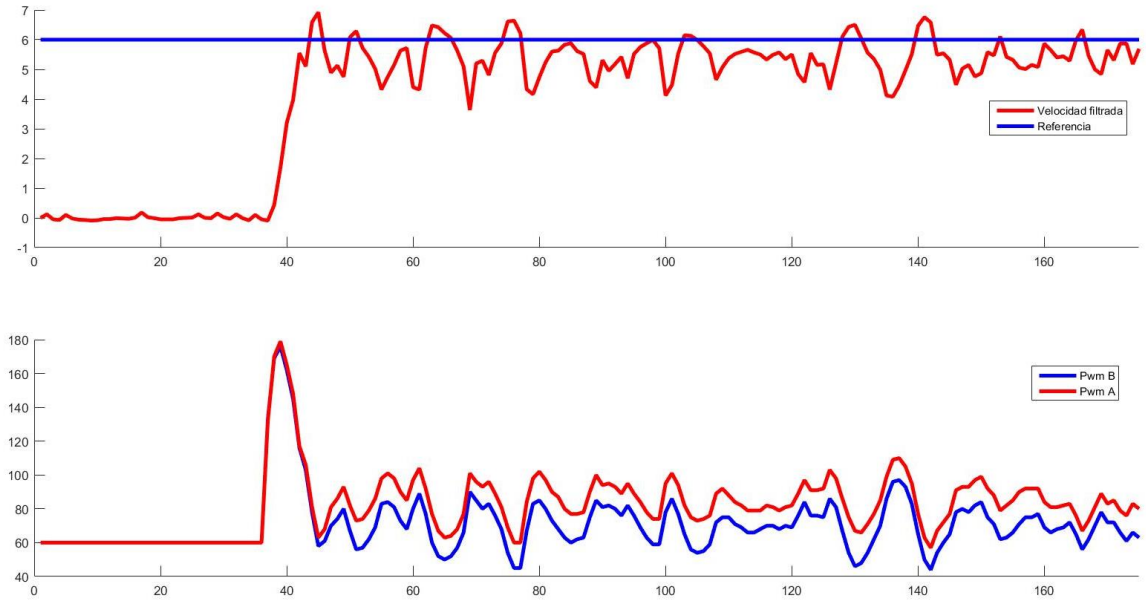

Figura 40: Representación de la velocidad y valores de PWM de nuestro robot

Por tanto, este control no se ha podido llevar a cabo debido al ruido que nos proporcionaba la medida de nuestro sensor de proximidad, y a su vez, el aumento de ruido que provoca el derivar algún tipo de información.

4. CONCLUSIONES

En este apartado hablaremos sobre las conclusiones que hemos obtenido tras todo el trabajo realizado con el robot móvil.

En cuanto a la orientación se refiere conseguimos controlar esta variable gracias a la exacta información que nos ofrece nuestro módulo BNO-055. Con un controlador proporcional-integral conseguíamos alcanzar la referencia deseada.

Por otro lado, con el sensor de ultrasonidos conseguimos que nuestro robot no colisionara con las superficies más próximas, asegurando así la integridad de los componentes.

Gracias a ambos controles anteriores, conseguimos que nuestro robot realizara la trayectoria mediante un entorno controlado. En nuestro caso, dicha trayectoria se trató de un cuadrado.

Otra utilidad que encontramos al sensor de ultrasonidos es la de obtener la velocidad lineal del robot (conociendo el tiempo de muestro y como varía la proximidad a una superficie). El problema es que conseguíamos una información con bastante error, imposibilitando así el control de la velocidad.

Una opción para conseguir dicho control sería la compra de un nuevo sensor de proximidad que proporcione la información con mayor precisión. De esta manera, derivando dicha información podríamos obtener la velocidad con menos ruido que con el sensor actual.

Otra opción para aplicar dicho control es intentar trabajar con la aceleración que nos proporciona el módulo inercial BNO-055. De esta manera, trabajaríamos de manera inversa a como lo hemos realizado actualmente, es decir, integrando dicha información. La integración de la información posee una ventaja frente a la derivación, y es el hecho de añadir menos ruido a nuestra señal.

Bibliografía

[1] https://www.arduino.cc/

[2] https://es.mathworks.com/products/simulink.html

[3] https://es.mathworks.com/help/matlab/index.html

[4] https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview

[5] https://www.zonamaker.com/arduino/modulos-sensores-y-shields/ultrasonido-hc-sr04

[6] https://es.wikipedia.org/wiki/Filtro_paso_bajo

[7] Ogata, Katsuhiko. Ingeniería de Control Moderna. Quinta edición. Editorial Prentice Hall.

[8] Fundamentos de control con MATLAB. Enrique Pinto Bermúdez y Fernando Matía Espada.

[9] Modern Control System. Richard C. Dork – Robert H. Bishop. Editorial Prentice Hall. Twelfth edition.

Anexos

1. Control de ángulo (Arduino)

- **Void setup ():**

```
 Serial.begin(9600); //Inicia la comunicación por puerto serie en
9600/15200 baudios
  temporizador = micros(); //Se asigna la variable Timer a un
temporizador en microsegundos
  qi1 = (KpA * (tm / 1000000)) / Ti;
  qi2 = (KpB * (tm / 1000000)) / Ti;
  //PARTE DEL MOTOR
  pinMode (ENA, OUTPUT);
  pinMode (ENB, OUTPUT);
  pinMode (IN2, OUTPUT);
  pinMode (IN1, OUTPUT);
  pinMode (IN3, OUTPUT);
  pinMode (IN4, OUTPUT);


  //pinMode HC - SR04
  pinMode(LedPin, OUTPUT);
  pinMode(TriggerPin, OUTPUT);
  pinMode(EchoPin, INPUT);

  /* Inizializamos el sensor */
  if (!bno.begin())
  {
    /*Aparecerá en caso de que no reconozca el módulo BNO - 055*/
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or
I2C ADDR!");
    while (1);
  }
```

- **Void loop ():**

```
 func_calibracion();
  if (calibracion == true) {
    temporizador += tm;

    //FUNCION PARA OBTENER DISTANCIA CON HC-SR04

    //int cm = ping(TriggerPin, EchoPin);
    //String distancia = String(cm);
    /*Serial.print("Distancia: ");
      Serial.println(cm);*/

    //PARTE DEL BNO055
    sensors_event_t event;
    bno.getEvent(&event);

    //CONVERSION (0 A 180 Y DE -180 A 0)
```

```
    orientation = (event.orientation.x);

    if (ref == 0 || ref == 90) {
      if (orientation > 180 && orientation <= 360) {
        orientation = orientation - 360;
      }
    }

    //PARTE PARA CAPTAR LA REFERENCIA POR BLUETOOTH

    //NO MOVEMOS MOTORES HASTA QUE NO CAPTEMOS INFORMACION POR
MATLAB Y BLUETOOTH
    if (Serial.available()) {
      ref = Serial.read();
      start = true;
    }
    if (start == false) {
      //PARTE DEL MOTOR
      pwm_valueA = 80;
      pwm_valueB = 80;

      //CONTROLAR PWM PARA ANGULO
      programaControlAngulo();
      /*
          pwm_valueA = pwm_valueA + inA;
          pwm_valueB = pwm_valueB - inB;
      */

      pwm_valueA = pwm_valueA + inA + Ia ;
      pwm_valueB = pwm_valueB - inB - Ib;

      if (pwm_valueB > 255) {
        pwm_valueB = 255;
      }
      if (pwm_valueB < 0) {
        pwm_valueB = 0;
      }
      if (pwm_valueA > 255) {
        pwm_valueA = 255;
      }
      if (pwm_valueA < 0) {
        pwm_valueA = 0;
      }

      analogWrite(ENA, pwm_valueA);
      analogWrite(ENB, pwm_valueB);
      digitalWrite (IN2, HIGH);
      digitalWrite (IN1, LOW);
      digitalWrite (IN4, HIGH);
      digitalWrite(IN3, LOW);
    }
    while (micros() < temporizador + tm) { //micros es el
temporizador interno
```

```
}
    /*
        String orientacion360 = String(event.orientation.x);
        String ref_str = String(ref);
        String pwma = String (pwm_valueA);
        String pwmb = String (pwm_valueB);
        Serial.println(orientacion360 + "/" + ref + "/" + pwma + "/"
+ pwmb + "/"); //Este bucle while se encarga de mantener constante
el tiempo   //    independientemente del tiempo que tarde el resto
de la ejecu
    */
  }
```

2. Seguimiento de trayectoria (Arduino)

- **Void setup ():**

```
  start = false;
  Serial.begin(9600); //Inicia la comunicación por puerto serie en
9600/15200 baudios
  temporizador = micros(); //Se asigna la variable Timer a un
temporizador en microsegundos
  qi1 = (KpA * (tm / 1000000)) / Ti;
  qi2 = (KpB * (tm / 1000000)) / Ti;
  //PARTE DEL MOTOR
  pinMode (ENA, OUTPUT);
  pinMode (ENB, OUTPUT);
  pinMode (IN2, OUTPUT);
  pinMode (IN1, OUTPUT);
  pinMode (IN3, OUTPUT);
  pinMode (IN4, OUTPUT);

  //pinMode HC - SR04
  pinMode(LedPin, OUTPUT);
  pinMode(TriggerPin, OUTPUT);
  pinMode(EchoPin, INPUT);

  /* Inizializamos el sensor */
  if (!bno.begin())
  {
    /*Aparecerá en caso de que no reconozca el módulo BNO - 055*/
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or
I2C ADDR!");
    while (1);
  }
```

- **Void loop ():**

```
func_calibracion();
  if (calibration == true) {


    if (Serial.available()) {
```

```
       start = true;
    }
    temporizador += tm;
    //FUNCION PARA OBTENER DISTANCIA CON HC-SR04

    cm = ping(TriggerPin, EchoPin);
    distancia = String(cm);

    /*Serial.print("Distancia: ");
      Serial.println(cm);*/

    //PARTE DEL BNO055
    sensors_event_t event;
    bno.getEvent(&event);


    //CONVERSION (0 A 180 Y DE -180 A 0)
    orientation = (event.orientation.x);
    if (start == false) { //COMENZAMOS CUANDO RECIBIMOS INFORMACION
DE MATLAB


      //PARTE DEL MOTOR
      if (completo == true) {
        pwm_valueA = 0;
        pwm_valueB = 0;
      }
      if (completo == false) {
        pwm_valueA = 80;
        pwm_valueB = 80;
      }


      if (e1 < (10) && (e1 >= 1) && giro == false && completo ==
false) {
        ref = ref + 90;
        if (ref == 360) {
          ref = 0;
          completo = true;
        }
        giro = true;
      }
      if (giro == true && e1 >= 30) {
        giro = false;
      }

      if (ref == 0 || ref == 90) {
        if (orientation > 180 && orientation <= 360) {
          orientation = orientation - 360;
        }
      }

      //CONTROLAR PWM PARA PROXIMIDAD
```

```
        programaControlProximidad();

        if (completo == true) {
         pwm_valueA = pwm_valueA + in1;
         pwm_valueB = pwm_valueB + in1;

         //EN CASO DE QUE SATURE
         if (pwm_valueA > 80) {
           pwm_valueA = 80;
         }
         if (pwm_valueA < 0) {
           pwm_valueA = 0;
         }

         if (pwm_valueB > 80) {
           pwm_valueB = 80;
         }
         if (pwm_valueB < 0) {
           pwm_valueB = 0;
         }
       }

       //CONTROLAR PWM PARA ANGULO
       programaControlAngulo();
       pwm_valueA = pwm_valueA + inA + Ia ;
       pwm_valueB = pwm_valueB - inB - Ib;

       if (pwm_valueB > 255) {
         pwm_valueB = 255;
       }
       if (pwm_valueB < 0) {
         pwm_valueB = 0;
       }
       if (pwm_valueA > 255) {
         pwm_valueA = 255;
       }
       if (pwm_valueA < 0) {
         pwm_valueA = 0;
       }


       analogWrite(ENA, pwm_valueA);
       analogWrite(ENB, pwm_valueB);
       digitalWrite (IN2, HIGH);
       digitalWrite (IN1, LOW);
       digitalWrite (IN4, HIGH);
       digitalWrite(IN3, LOW);

     }
     while (micros() < temporizador + tm) { //micros es el
temporizador interno
     } //Este bucle while se encarga de mantener constante el tiempo
de muestreo en cada iteración
```

```
  //    independientemente del tiempo que tarde el resto de la
ejecución.
    /*
        String orientacion360 = String(event.orientation.x);
        String ref_str = String(ref);
        Serial.println(orientacion360 + "/" + distancia + "/" + ref
+ "/" + pwmA + "/" + pwmB + "/"); //SI QUIERO HACERLO EN VIVO QUITAR
LA ULTIMA BARRA
    */
  }
```

3. GUIDE para representación en vivo (Matlab)

Como mencionamos anteriormente, se ha elaborado una GUIDE para representar en vivo toda la información sobre nuestro robot. A continuación, se exponen la utilidad de sus elementos y su correspondiente programación.

- **Botón para activar módulo Bluetooth:**

```
function pb_bt_Callback(hObject, eventdata, handles)
global s
 s = Bluetooth('HC-06',1);
 fopen(s);
```

- **Botón para comenzar la transmisión en vivo y representar datos en sus correspondientes gráficas:**

```
function pb_start_Callback(hObject, eventdata, handles)

global s
    hAnimatedLine(1) = animatedline('Parent',handles.axes1);
    hAnimatedLine(2) = animatedline('Parent',handles.axes2);
    hAnimatedLine(3) = animatedline('Parent',handles.axes1);
    hAnimatedLine(4) = animatedline('Parent',handles.axes3);
    hAnimatedLine(5) = animatedline('Parent',handles.axes4);
    hAnimatedLine(1).Color = 'red';
    hAnimatedLine(2).Color = 'blue';
    hAnimatedLine(3).Color = 'green';
    hAnimatedLine(4).Color = 'red';
    hAnimatedLine(5).Color = 'blue';
    hAnimatedLine(1).LineWidth=3;
    hAnimatedLine(2).LineWidth=3;
    hAnimatedLine(3).LineWidth=3;
    hAnimatedLine(4).LineWidth=3;
    hAnimatedLine(5).LineWidth=3;
    muestra=1;
    while true
```

```
        vector_datos=fscanf(s);
        datos=strsplit(vector_datos,'/');

        orientacion_str=datos{1};
        orientacion = str2double(orientacion_str);
        distancia=datos{2};


        distancia = str2double(distancia);
        referencia=datos{3};
        referencia=str2double(referencia);
        pwma=datos{4};
        pwma=str2double(pwma);
        pwmb=datos{5};
        pwmb=str2double(pwmb);
        handles.axes6.XLim = [0 muestra];
        handles.axes6.YLim = [0 380];
        handles.axes7.XLim = [0 muestra];
        handles.axes7.YLim = [0 150];
        handles.axes11.XLim = [0 muestra];
        handles.axes11.YLim = [0 255];
        handles.axes12.XLim = [0 muestra];
        handles.axes12.YLim = [0 255];
        addpoints(hAnimatedLine(1),muestra,orientacion);
        addpoints(hAnimatedLine(2),muestra,distancia);
        addpoints(hAnimatedLine(3),muestra,referencia);
        addpoints(hAnimatedLine(4),muestra,pwma);
        addpoints(hAnimatedLine(5),muestra,pwmb);
        drawnow
        muestra=muestra+1;
    end
```

- **Botón para iniciar trayectoria:**

```
function pb_inicio_Callback(hObject, eventdata, handles)
% hObject    handle to pb_stop (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see
GUIDATA)
global s
fwrite(s,10); % recibimos un valor en Arduino y
                empieza a realizar el control de la trayectoria
```