Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

"Control de motores en entorno MATLAB sobre plataforma Arduino"

Autor: Federico Pérez Ruiz

Tutor: José Manuel Villadangos Carrizo

2019

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo Fin de Grado

"Control de motores en entorno MATLAB sobre plataforma Arduino"

Autor: Federico Pérez Ruiz

Tutor: José Manuel Villadangos Carrizo

TRIBUNAL

Presidente: Mª Carmen Pérez Rubio

Vocal 1º: Felipe Espinosa Zapata

Vocal 2º: José Manuel Villadangos Carrizo

FECHA: 5 de julio de 2019

Resumen

Este trabajo se basa en la realización de un trabajo de programación, control y movimiento de los principales tipos de motores eléctricos de corriente continua (motores DC estándar, servomotores y motores paso a paso). La programación se realizará en el entorno de desarrollo de MATLAB, en contraposición a la programación convencional tratada en las distintas asignaturas del Grado.

Las aplicaciones se desarrollarán sobre dos plataformas Arduino, los modelos UNO y Due, utilizando dos paquetes de compatibilidad entre estas plataformas de hardware y el entorno MATLAB/Simulink.

Palabras clave

MATLAB, Simulink, Arduino, Motores eléctricos corriente continua.

Abstract

This work is based on the realization of a programming, control and movement work of the main types of DC electric motors (standard DC motors, servo motors and stepper motors). The programming will be done with the development environment of MATLAB, in contrast to the conventional programming treated in the different subjects of the degree.

The applications will be developed on two Arduino platforms, the models UNO and Due, using two compatibility packages between these hardware platforms and the MATLAB / Simulink environment.

Keywords

MATLAB, Simulink, Arduino, Direct current electrical motors.

Resumen extendido

Los motores eléctricos son un elemento fundamental en todas aquellas aplicaciones que requieran la generación de un movimiento mecánico para llevarse a cabo, por lo que son ampliamente utilizados en todos los sectores de la industria.

Estos se dividen principalmente en dos tipos: los motores de corriente continua (DC) y de corriente alterna (AC), según el tipo de corriente que utilicen para funcionar.

Las principales ventajas de los motores DC frente a los motores AC son las siguientes:

- Poseen una mayor versatilidad, debido a que es posible modificar la corriente inducida en el motor y de esta forma variar la velocidad y el torque mecánico. Debido a esto, son muy útiles en aplicaciones que requieran bajas velocidades.
- Producen un gran torque en el momento del arranque, por lo que pueden romper la inercia provocada por grandes cargas e iniciar el movimiento con mayor facilidad, por lo que son idóneos para trabajos pesados.

Por contrapartida, los motores DC son más caros de fabricar y solo pueden ser alimentados de forma monofásica, además de presentar un rendimiento inferior.

Partiendo del motor DC normal, existen otros dos tipos principales: los servomotores, que permiten un control mucho mayor de su posición angular, y los motores paso a paso, que rotan con mayor precisión y no requieren de realimentación para controlar sus movimientos.

Las plataformas de hardware Arduino permiten a sus usuarios diseñar multitud de aplicaciones con estos motores en su entorno de desarrollo integrado (IDE). Sin embargo, empleando este tipo de programación, el potencial de las aplicaciones queda determinado por las librerías y funciones de las que dispone este IDE.

El objetivo de este proyecto es la programación de diversas aplicaciones con los tres tipos de motores, empleando el software de MATLAB y Simulink.

Estas herramientas, ampliamente utilizadas en el campo de la ingeniería, disponen de un amplio abanico de funciones y librerías enfocadas principalmente en la creación y simulación de sistemas, la graficación de datos y la comunicación con otros equipos.

De esta forma, el presente trabajo pretende enseñar acerca de una forma alternativa de diseñar aplicaciones con motores eléctricos en Arduino, para complementar la formación obtenida en el Grado de Ingeniería en Electrónica y Automática Industrial.

La comunicación con las plataformas Arduino y la programación de las aplicaciones se realizará con dos paquetes de compatibilidad que ofrecen MATLAB y Simulink. La programación de las aplicaciones se desarrollará con las herramientas (funciones y bloques) que ofrecen ambos entornos.

De esta forma, MATLAB/Simulink pasa de ser un entorno enfocado a la simulación de sistemas a ejecutarlos.

Por su parte, las aplicaciones se enfocarán principalmente en el manejo y control de los motores, y en su caso, la incorporación de un elemento adicional complementario al movimiento del motor para mostrar las posibilidades de diseño que ofrecen MATLAB y Simulink en relación a los motores eléctricos de corriente continua.

Índice de contenidos

Índice de figuras

Índice de tablas

1 Introducción

1.1 Objetivos del proyecto

Los objetivos a conseguir con este proyecto son los siguientes:

- Conocer los usos y características de las plataformas Arduino.
- Comprender el funcionamiento interno de los motores DC, servomotores y motores paso a paso, y la forma de controlar su movimiento mediante MATLAB y Simulink.
- Conocer las posibilidades que brindan MATLAB y Simulink al diseño de aplicaciones con motores eléctricos, tanto con los paquetes de compatibilidad con Arduino como con las herramientas propias de estos entornos.
- Diseñar una serie de aplicaciones basadas en el manejo y control de los tres tipos de motores.
- Valorar las principales ventajas y desventajas de un entorno u otro a la hora de operar con cada tipo de motor.
- Emplear MATLAB y Simulink para integrar un elemento adicional que complemente el movimiento de los motores o lo emplee para un uso determinado.
- Sugerir futuras líneas de investigación en base a las conclusiones extraídas de este proyecto, de modo que pueda servir de base para el diseño de aplicaciones más complejas.

1.2 Fases del proyecto

Para cumplir con los objetivos antes mencionados, se ha decidido dividir el proyecto en cuatro partes principales.

-**Primera fase**: En esta fase se detallará el funcionamiento del hardware a emplear para el montaje de las aplicaciones. Este hardware comprenderá principalmente las tarjetas Arduino, los tres modelos de motores elegidos y los drivers necesarios para controlar su movimiento en el caso de que se requieran.

-**Segunda fase**: En esta fase se realizará una breve introducción al entorno de MATLAB y Simulink y se detallarán las funciones y bloques que ofrecen sus respectivos paquetes de compatibilidad con Arduino. También se explicará la forma de conectar las tarjetas Arduino a sendos entornos de programación para poder ejecutar las aplicaciones.

-**Tercera fase**: En esta fase se desarrollarán las aplicaciones con los motores y se llevará a cabo su comprobación experimental. También se detallará el funcionamiento de los componentes adicionales a utilizar en cada aplicación, si los hubiera. Aunque en este proyecto MATLAB/Simulink actúa como un entorno de desarrollo de aplicaciones prácticas, también se llevarán a cabo en él todas las simulaciones pertinentes para asegurar el buen estado de los componentes.

-**Cuarta fase**: En esta fase se extraerán las conclusiones extraídas de cada aplicación y se valorarán posibles proyectos futuros en base a las mismas.

2 Hardware

2.1 Arduino

Las plataformas Arduino son un conjunto de tarjetas de desarrollo que se emplean para la implementación de distintas aplicaciones. Cada una de estas tarjetas dispone de un microcontrolador que almacena y ejecuta de forma embebida las distintas funciones que el usuario programa desde el software de Arduino.

Este software se compone de un entorno de desarrollo integrado (IDE) propio basado en el lenguaje de programación Wiring.

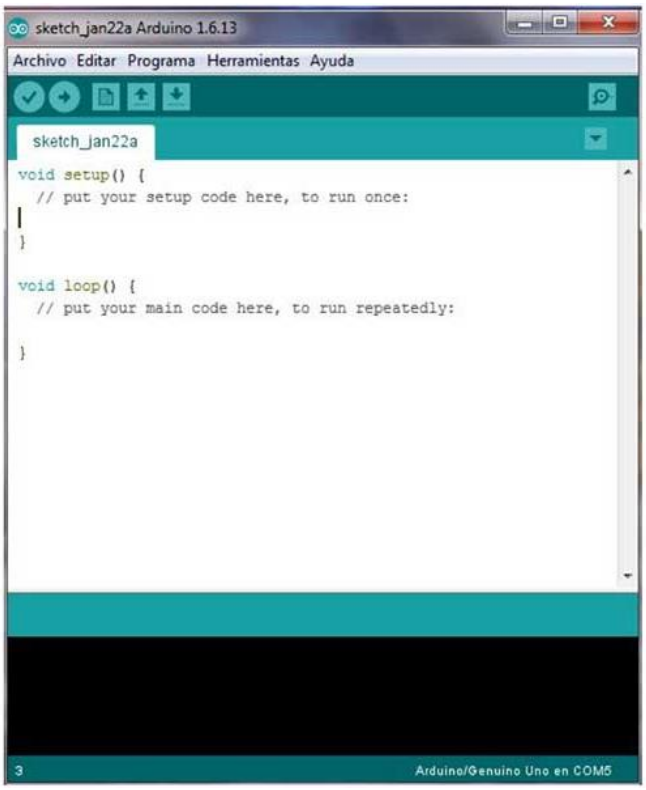


Figura 1. IDE de Arduino

La principal característica de Arduino reside en la libre distribución que permite a cualquier usuario tanto de las tarjetas de desarrollo como de las herramientas de software, lo que lo convierte en una opción muy atractiva para la creación de proyectos pequeños e independientes.

En el presente proyecto emplearemos los paquetes de compatibilidad de Arduino con MATLAB y Simulink, y los dos entornos citados para realizar la programación de los movimientos de los motores, en lugar del IDE de Arduino.

Usaremos dos tarjetas Arduino: la Arduino UNO, la plataforma más básica de la serie, y la Arduino Due, de una potencia y capacidad mucho mayor.

2.1.1 Arduino UNO

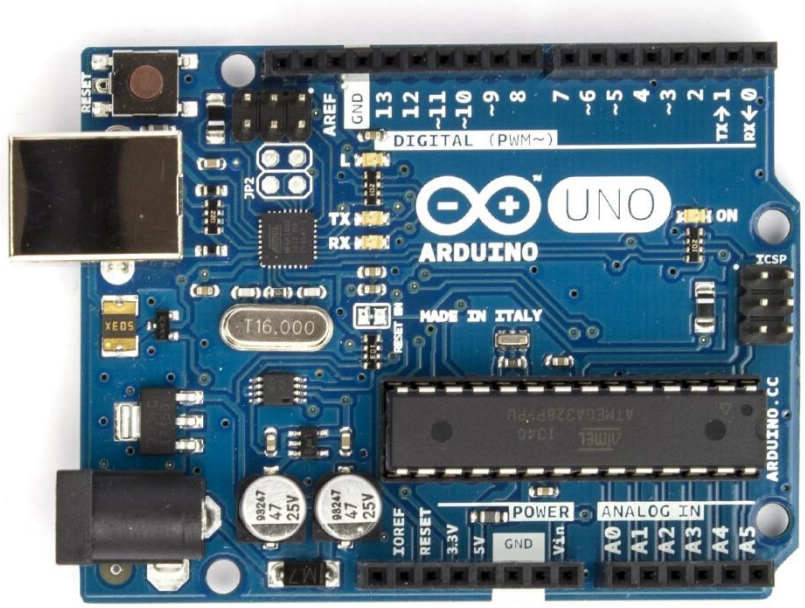


Figura 2. Arduino UNO

La Arduino UNO incorpora un microcontrolador Atmega328P, el cual posee las siguientes características principales.

Memoria Flash	32 kB
Frecuencia máxima de reloj	16 MHz
SRAM	2 kB
EEPROM	1 kB
Voltaje de operación	5 V

Tabla 1. Características técnicas de Arduino UNO

La Arduino UNO cuenta con la siguiente configuración de pines:

- 2 pines de alimentación a 5 V y 3,3 V que suministran la tensión continua correspondiente, con una intensidad máxima de 200 mA en el caso de 5 V y de 150 mA en el caso de 3,3V. También cuenta con dos pines GND referidos a masa y de un pin Vin que proporciona un valor de tensión equivalente a la de una fuente de alimentación externa conectada a la tarjeta.
- 14 pines de propósito general que pueden configurarse como entradas o salidas digitales, con una tensión de operación de 5 V. Algunos de estos pines, además, tienen otras posibles aplicaciones:
    - Pines 0 (RX) y 1 (TX): Se emplean para realizar la comunicación serie (recepción y transmisión de datos, respectivamente) con dispositivos Bluetooth o con el PC a través del cable USB.
    - Los pines 3, 5, 6, 9, 10 y 11 se pueden emplear como salidas de señales de modulación PWM, configurando el ciclo de trabajo de una señal cuadrada periódica.

- Los pines 10, 11, 12 y 13 se pueden emplear para implementar una comunicación SPI síncrona entre el microcontrolador de la tarjeta y los distintos periféricos conectados a la misma.

- 6 entradas analógicas, con un ADC interno de 10 bits de resolución (1024 valores para medir entre masa y 5V). El pin de entrada AREF se emplea para proporcionar el límite superior de tensión leída y de esta forma aumentar la precisión de la lectura realizada.
- 2 series de 2 pines para establecer una comunicación I2C con otros dispositivos.
- 1 bus ISCP, que puede emplearse para establecer comunicaciones de tipo serie o SPI.

2.1.2 Arduino Due

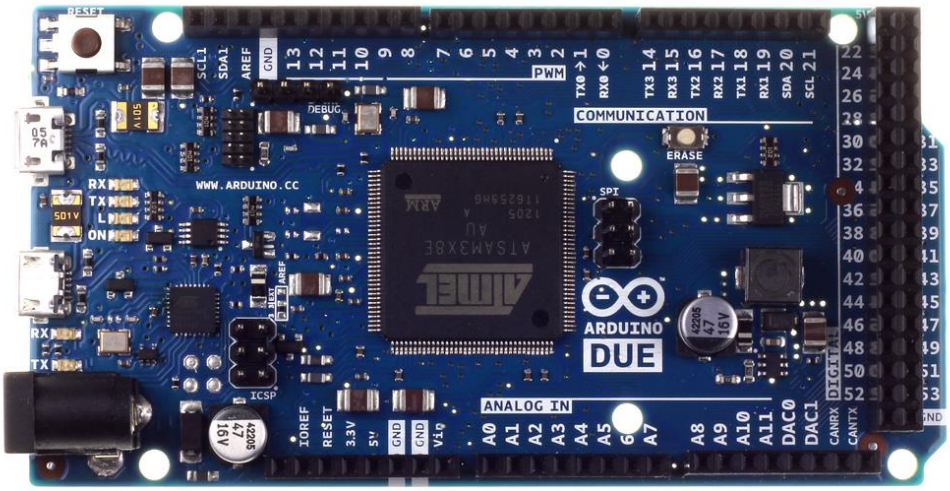


Figura 3. Arduino Due

La Arduino Due es la primera tarjeta de desarrollo de Arduino basada en ARM. Utiliza un potente microcontrolador ARM Cortex M3, el modelo AT91SAM3X8E, con las siguientes características:

Memoria Flash	512 KB
Máxima frecuencia de reloj	84 MHz
SRAM	96 KB
Voltaje de operación	3,3 V

Tabla 2. Características técnicas Arduino Due

Este modelo es el único de Arduino que opera con un voltaje de 3,3 V, en vez de con los 5 V habituales. Alimentar los pines de propósito general con una tensión mayor podría dañar la tarjeta.

La Arduino Due cuenta con la siguiente configuración de pines:

- 2 pines de alimentación a 5 V y 3,3 V, que suministran la tensión continua correspondiente, con una intensidad máxima de 800 mA para ambos casos. También cuenta con pines GND, un pin Vin y un pin IOREF, que proporciona la referencia de tensión con la que opera el microcontrolador.
- 54 pines de propósito general que pueden operar como entradas o salidas digitales, con una tensión de operación de 3,3 V.
- 8 pines de comunicación en serie (4 RX y 4 TX) para recibir y transmitir datos en serie. Los pines 0 y 1 (RX1 y TX1) se emplean para comunicar la tarjeta con un PC vía USB.
- 12 pines que proporcionan salidas PWM.
- 12 entradas analógicas, cuyo ADC interno es de hasta 12 bits de resolución (entre 0 y 3,3 V). La resolución de las lecturas se fija por defecto en 10 bits, para facilitar la conexión con otras plataformas Arduino.
- 2 salidas analógicas con una resolución de 12 bits.
- 1 conector de 6 pines que admiten una comunicación SPI para conectarse con dispositivos de este tipo.
- 2 series de 2 pines (SDA y SCL) para establecer comunicación con dispositivos I2C.
- 2 pines (CANRX y TANRX) que soportan un protocolo de comunicación CAN.

La carga de programas al SAM3X es distinta en comparación a la que se emplea en el resto de microcontroladores AVR de las demás tarjetas Arduino. En este caso el microcontrolador sufre un borrado antes de que la memoria Flash sea reprogramada. El tipo del borrado depende del puerto de conexión de la Arduino Due que se utilice.

Figura 4. Puertos USB de la Arduino Due

- Puerto USB nativo: El puerto USB nativo está conectado directamente a la SAM3X. Abrir y cerrar este puerto nativo a una velocidad de 1200bps desencadena un procedimiento de "borrado ligero": la memoria flash se borra y la tarjeta se reinicia con el bootloader. Abrir y cerrar el puerto nativo a una velocidad de transmisión diferente, no reiniciará el SAM3X.
- Puerto de programación: Este puerto está conectado a un Atmega16U2, que proporciona un puerto COM virtual de software en un ordenador conectado.
  El puerto de programación utiliza el 16U2 como un chip conversor de USB a serie conectado a la UART0 del SAM3X (RX0 y TX0). Abrir y cerrar el puerto de programación conectado a 1200 bps desencadena un procedimiento de "borrado exhaustivo" del SAM3X, activa el borrado y resetea los pines en el SAM3X antes de comunicarse con la UART.

El puerto de programación es el más recomendado de los dos para programar con la Due, debido a la mayor fiabilidad del borrado que ofrece.

Ambas tarjetas pueden alimentarse con un PC a través del cable de conexión USB o mediante una fuente de alimentación externa a través de las clavijas JACK o el pin Vin. En nuestro proyecto, salvo que se indique lo contrario, siempre se alimentarán a través de un cable USB conectado al PC.

2.2 Funcionamiento y modelo de motor DC

Los motores de corriente continua están formados por dos elementos, el rotor y el estator.

- El rotor es un componente móvil formado por un núcleo ferromagnético que opera como eje del motor y una serie de bobinas conductoras enrolladas en torno a él.
- El estator es un componente fijo que envuelve al rotor formado por imanes permanentes que generan un campo magnético que posteriormente influye sobre los conductores anteriores.

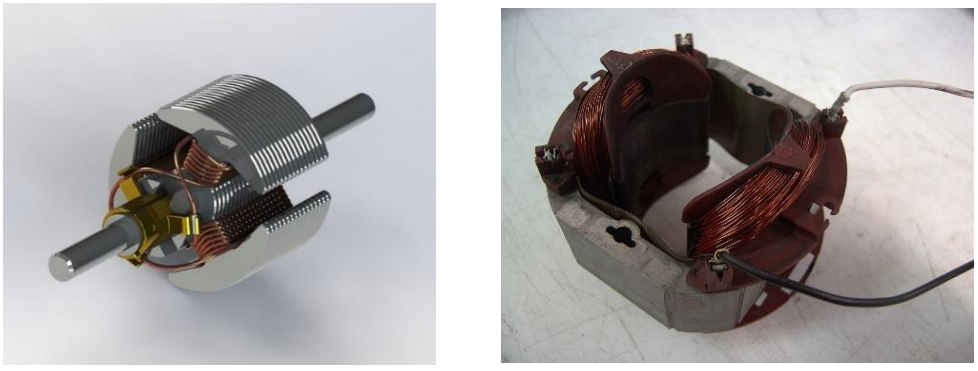


Figura 5. Rotor y estator de un motor DC

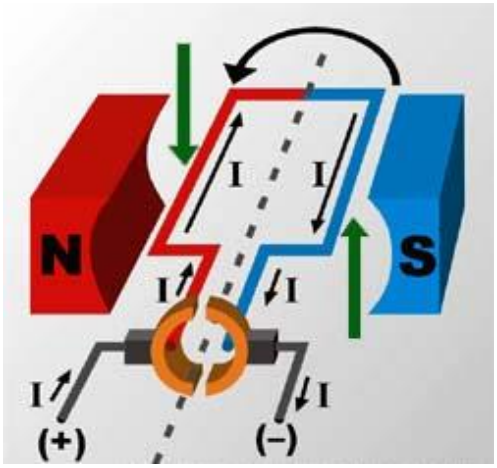


Figura 6. Expira excitada en el interior de un campo magnético (1)

Su principio de funcionamiento es el siguiente.

Al suministrar corriente a los devanados del rotor, se genera un campo electromagnético, cuya dirección depende del sentido de circulación de la corriente inducida.

Este nuevo campo interactúa con el campo magnético creado por los imanes permanentes del estator. Cuando los polos de ambos campos coinciden, se produce una fuerza de rechazo perpendicular a la corriente inducida y al campo magnético generado en el rotor. Esta fuerza es proporcional a la magnitud de dicho campo y al número de conductores por los cuales se transmite la corriente.

De esta forma, se genera un movimiento giratorio por parte del eje del motor a partir de energía eléctrica.

Para obtener un movimiento continuo se emplean conmutadores que cambian el sentido de circulación de la corriente cada vez que el motor recorre media vuelta. De esta manera los polos del campo magnético del rotor coinciden en todo momento con el del estátor, provocando una fuerza de rechazo constante.

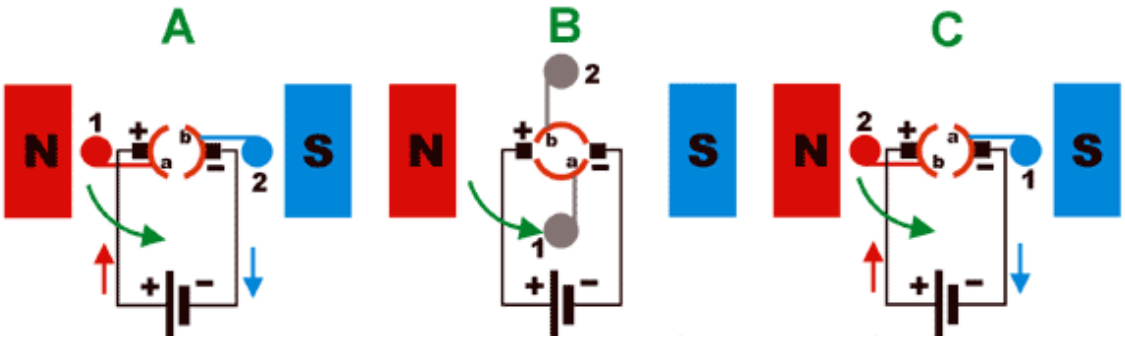


Figura 7. Efecto de conmutaciones en un motor DC (1)

Estas conmutaciones pueden realizarse mediante elementos mecánicos, como el caso de las escobillas en los motores DC, o de forma electrónica, como en los motores paso a paso.

(1) http://www.asifunciona.com/electrotecnia/af_motor_cd/af_motor_cd_6.htm

El motor DC elegido es el modelo GA12-N20, un motor de reducidas dimensiones cuya velocidad máxima sin carga es de 1000 rpm, a una tensión de alimentación de 12V.

Dispone de una reductora que otorga un factor de 1:30 y de un encoder rotativo que detecta la velocidad y el sentido de giro del motor.

Figura 8. Motor GA12-N20 y terminales

El motor se controla mediante los terminales M1 y M2, y las lecturas proporcionadas por el encoder se muestran a través de los terminales C1 y C2. Por último existen dos terminales para alimentar el encoder.

Únicamente alimentando a tensión constante los terminales del motor (M1 y M2) no podemos controlar ni la velocidad ni el sentido de giro del mismo. Para ello necesitamos un elemento auxiliar: un driver basado en un puente en H formado por cuatro transistores bipolares o unipolares.

2.2.1 Driver L9110S

Este driver contiene dos chips independientes L9110 para controlar el sentido de giro y la velocidad de dos motores DC simultáneamente o de un único motor paso a paso unipolar. Soporta tensiones de operación de entre 2,5V y 12V, por lo que podemos manejarlo con ambas plataformas Arduino, y su corriente de operación ronda los 800 mA.

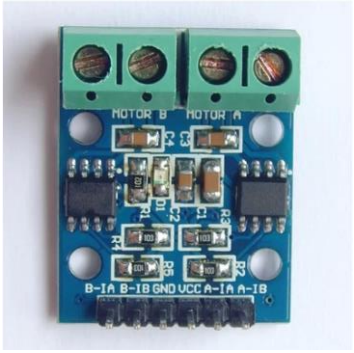


Figura 9. Driver doble puente en H L9110S

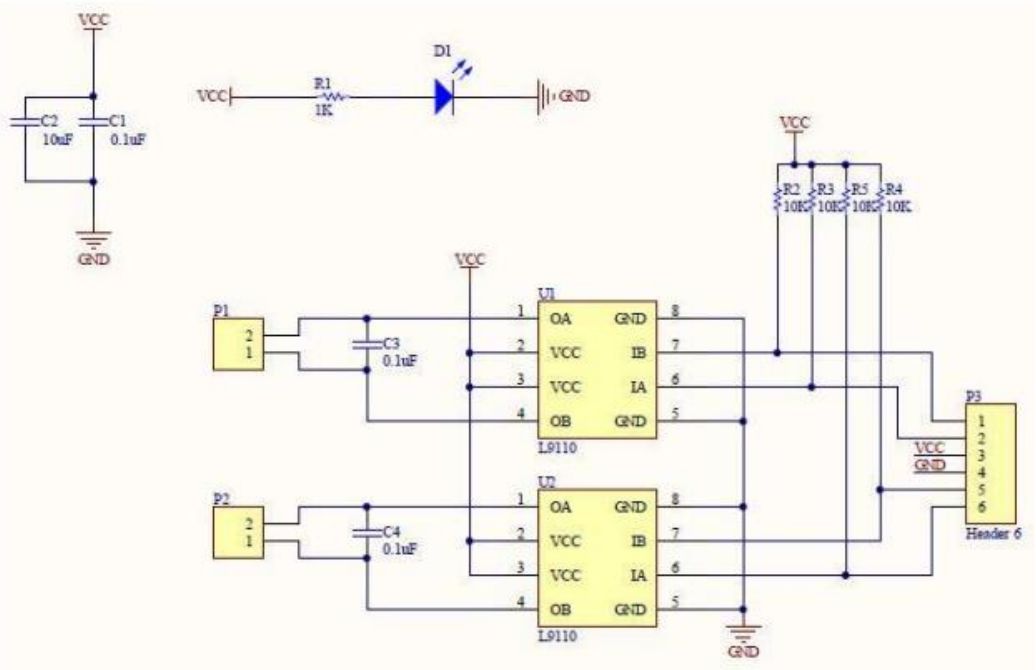


Figura 10. Esquema eléctrico driver L9110S

Cada chip es capaz de controlar un único motor DC (representados en el esquema con los bloques P1 y P2) empleando dos entradas de control digitales (x-IA y x-IB). La siguiente tabla de verdad indica el sentido de giro de un motor en base a los valores de las entradas digitales.

x-IA	x-IB	Movimiento
0	0	Motor parado
0	1	Giro horario
1	0	Giro antihorario
1	1	Motor parado

Tabla 3. Estado del motor en función de las entradas del L9110S

Variamos la velocidad del motor ajustando el ciclo de trabajo de la señal de entrada PWM correspondiente.

Los pines de salida digital de Arduino operan únicamente a 0 y a 5V (a 3,3V en el caso de la Due), por lo tanto no podemos controlar la velocidad del motor DC directamente: es necesario simular una tensión de salida analógica, algo que los pines de propósito general de la Arduino UNO no pueden proporcionar.

2.2.2 Modulación por ancho de pulsos

Como solución disponemos de la modulación por ancho de pulsos (PWM). Esta técnica consiste en transmitir de forma periódica una señal cuadrada de ciclo de trabajo (pulse-width) variable, es decir, el porcentaje de un periodo de la señal donde esta se sitúa a nivel alto.

Esta técnica se emplea principalmente para transmitir información o para controlar la energía suministrada a una carga. En nuestro proyecto la empleamos como convertidor ADC, para simular una salida analógica a partir de una salida digital.

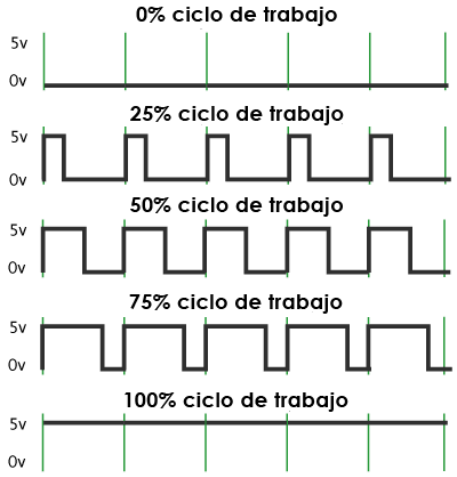


Figura 11. Señales PWM con distintos ciclos de trabajo

La tensión media de la señal PWM es directamente proporcional a su ciclo de trabajo.

$$Vm = Vcc * PW \quad [1]$$

De esta forma podemos variar la tensión media de salida conectada al L9110S, controlando así la velocidad del motor DC.

2.2.3 Encoder incremental

Por último, utilizamos el encoder que viene acoplado de base al motor GA12-N20 para calcular la velocidad del mismo y determinar el sentido de giro cuando se encuentra en movimiento.

Un encoder es un dispositivo de detección que genera señales eléctricas a partir del movimiento al cual se ve sometido. De esta forma, puede informar acerca de la posición o velocidad de un motor con el que esté conectado mecánicamente.

Los encoders rotativos se clasifican en dos grupos:

-Encoders incrementales: Generan una cierta cantidad de pulsos por cada revolución del motor. En base a este valor de resolución podemos calcular cuantos grados corresponden a un pulso y la posición del motor. El inconveniente es que no cuentan con ninguna referencia de posición absoluta.

-Encoders absolutos: Estos encoders generan un código binario de salida que indica la posición angular del motor. Cada valor del código se corresponde con una posición determinada dentro de una misma vuelta, por lo tanto, estos encoders disponen de un punto de referencia absoluta que permanece intacto tras la desconexión del motor.

El encoder acoplado al motor DC es de tipo rotativo incremental, alimentado a 5 V y posee una resolución de 7 pulsos por revolución (ppr). Este encoder funciona en base a las respuestas de dos sensores de efecto Hall.

Debido a que está conectado entre el eje externo del motor y los engranajes, su resolución se multiplica por la reductora 1:30 del motor, elevándose a 210 ppr.

Figura 12. Encoder incremental acoplado al GA12-N20

Los encoders incrementales emplean normalmente dos canales de salida: A y B, por los cuales se transmiten los pulsos, en este caso mediante dos sensores de efecto Hall que reaccionan ante el imán circular que gira junto con el eje del motor.

Estos sensores se sitúan con una separación de 90º respecto al centro de giro.

Debido a esto, un canal siempre se sitúa desfasado o adelantado 90º grados eléctricos respecto al otro, dependiendo del sentido de giro del motor.

De esta forma, podemos conocer el sentido de giro comprobando si la señal del canal A se encuentra adelantada o retrasada respecto de la señal del canal B.

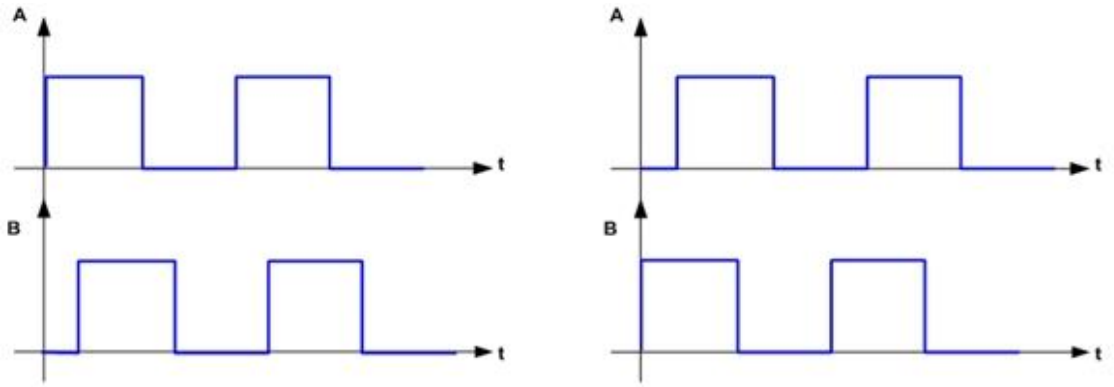


Figura 13. Canales A y B del encoder en sentido horario (izquierda) y antihorario (derecha)

2.3 Funcionamiento y modelo de servomotor

Los servomotores son dispositivos que permiten realizar movimientos angulares de forma controlada, por lo tanto tienen una gran utilidad en multitud de aplicaciones de robótica.

Un servomotor está formado por un motor DC conectado a un sistema de regulación que actúa sobre el mismo y a un sensor que indica su posición angular en tiempo real.

Para explicar el sistema de control de los servos y la regulación de los mismos emplearemos como referencia el servo a utilizar en nuestro proyecto: el modelo SG90.

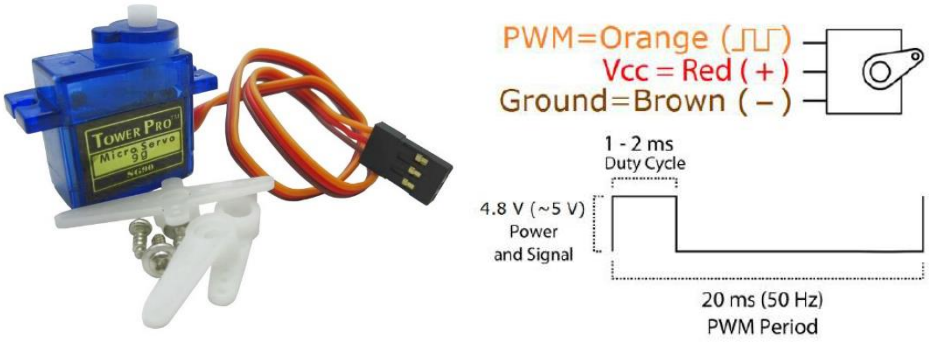


Figura 14. Apariencia y señal de control del servomotor SG90

El modo de control de los servomotores se basa en la anchura de pulsos periódicos a una frecuencia de 50 Hz, dependiendo de la posición angular que queramos adoptar. El SG90 acepta pulsos comprendidos entre 1ms (0º) y 2ms (180º) de duración. La anchura del pulso es directamente proporcional a la posición alcanzada.

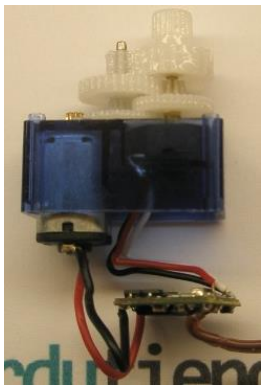


Figura 15. Estructura interna del SG90 (2)

Debido a la limitación de 180º que existe en el rango de movimiento de la mayoría de los servos, la realimentación de los mismos se basa en la tensión proporcionada por un potenciómetro interno, una resistencia variable, que gira solidario al motor DC, gracias a la conexión del tren de engranajes.

La tensión proporcionada por el potenciómetro indica la posición actual del motor, ya que ambos giran a la par.

En reposo, el servomotor se encuentra en un estado de equilibrio, y al aplicar un pulso se produce una descompensación entre la tensión enviada al servo y la lectura del potenciómetro. Debido a esta diferencia de tensión, la tarjeta de control del servo alimenta el motor DC y lo hace girar hasta que la lectura iguala la señal de entrada.

Este método resulta mucho más eficaz y preciso que tratar de aplicar un circuito de control a un motor DC, variando su tensión de entrada.

Por último, existe otro tipo de servos que no serán tratados en este proyecto, los servomotores de rotación continua. Estos pueden girar hasta 360º en ambos sentidos y de forma continuada, y la señal de control determina su velocidad de giro en vez de su posición angular.

(2) http://www.ardumania.es/trucar-servo-a-rotacion-continua/

Es posible transformar el SG90 en un servomotor de rotación continua sustituyendo el potenciómetro interno por un par de resistencias iguales y eliminando los topes mecánicos del tren de engranajes, que restringen su rango de movimiento a 180º.

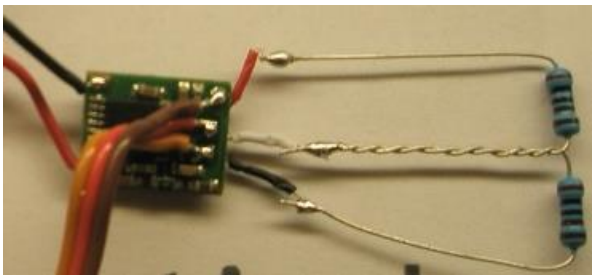


Figura 16. Servo SG90 con rotación continua (2)

Aunque un modelo comercial posee mayor rendimiento, esta modificación convierte al SG90 en un servomotor de rotación continua totalmente funcional.

2.4 Funcionamiento y modelo de motor paso a paso

Los motores paso a paso, a diferencia de los dos tipos anteriores, no emplean escobillas, sino que las conmutaciones se realizan de forma electrónica.

En este tipo de motores, los bobinados del estator se alimentan siguiendo una determinada secuencia, lo que genera un desplazamiento angular del eje del rotor. El desplazamiento provocado por una única conmutación del estado de las bobinas se denomina como paso.

Existen dos tipos principales de estos motores: los de reluctancia variable y los de imanes permanentes. En nuestro proyecto nos centraremos en un modelo perteneciente al segundo tipo.

Los motores paso a paso de imanes permanentes disponen de un imán ubicado en el rotor solidario al eje del motor y de un estator formado por dos bobinas desfasadas 90º entre sí.

Aplicando un suministro de corriente a las bobinas, se consigue que el imán se oriente progresivamente, provocando el movimiento angular del motor.

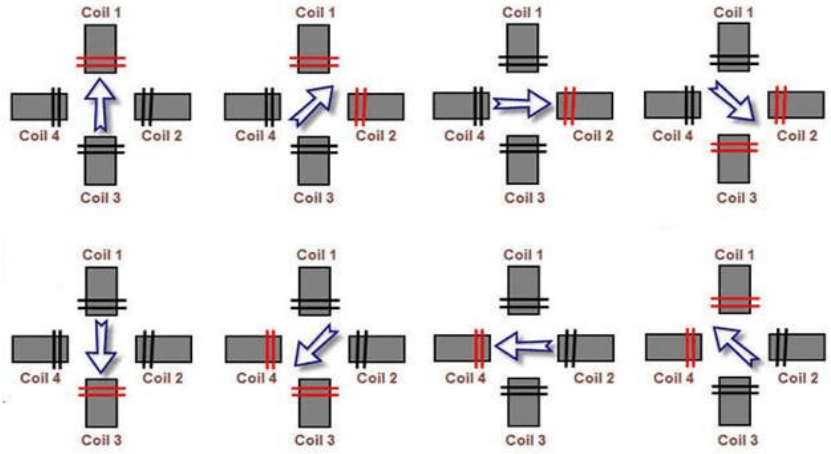


Figura 17. Secuencia de excitación de bobinas en un ciclo

(2) http://www.ardumania.es/trucar-servo-a-rotacion-continua/

Incorporando un tren de engranajes al eje, se obtiene un paso muy reducido, lo cual convierte a este tipo de motores en una opción idónea para aplicaciones que requieran movimientos angulares precisos y de alta resolución y un giro de 360º por parte del motor.

Estos se dividen en bipolares y unipolares:

- Los bipolares solo disponen de cuatro cables de salida, lo cual implica la necesidad de intercambiar el sentido de la corriente que fluye por las bobinas para obtener un movimiento adecuado.
- Los motores unipolares disponen además de terminales de alimentación conectados a los puntos medios de las bobinas. Al poder alimentar solo media bobina es posible realizar movimientos angulares sin invertir el sentido de corriente en ningún momento. Desafortunadamente, esto implica un menor torque que en el caso bipolar, ya que solo se emplea la mitad de tensión en las bobinas y la fuerza inducida es menor.

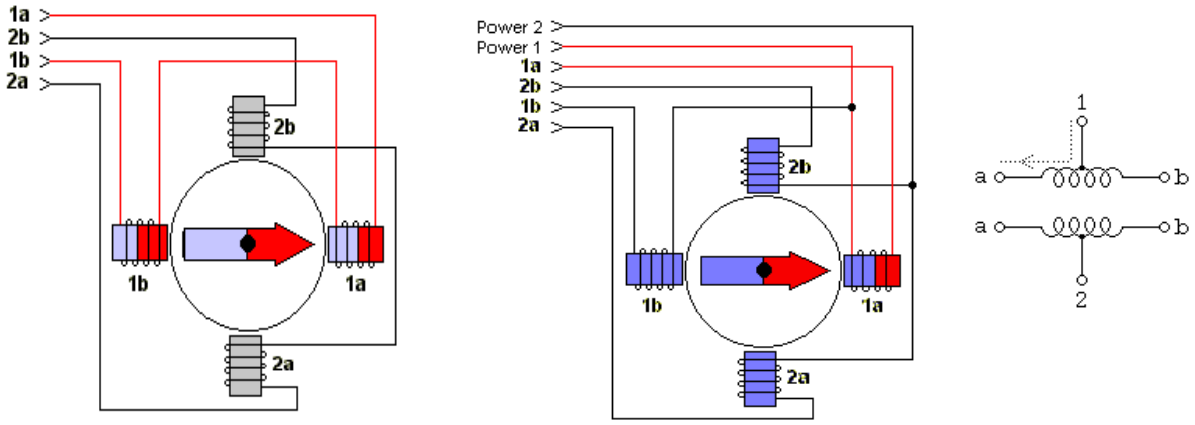


Figura 18. Conexión del bobinado en motores bipolares y unipolares

El motor paso a paso empleado en este proyecto es el modelo 28BYJ-48, un motor paso a paso unipolar de imanes permanentes con cuatro terminales de salida y un quinto común. Se alimenta a 5 V y posee una reductora de velocidad 1:64 y una resolución máxima de 0,088º por paso, superando ampliamente la resolución proporcionada por los dos motores anteriores.

Para controlar el motor utilizaremos el circuito integrado ULN2003.

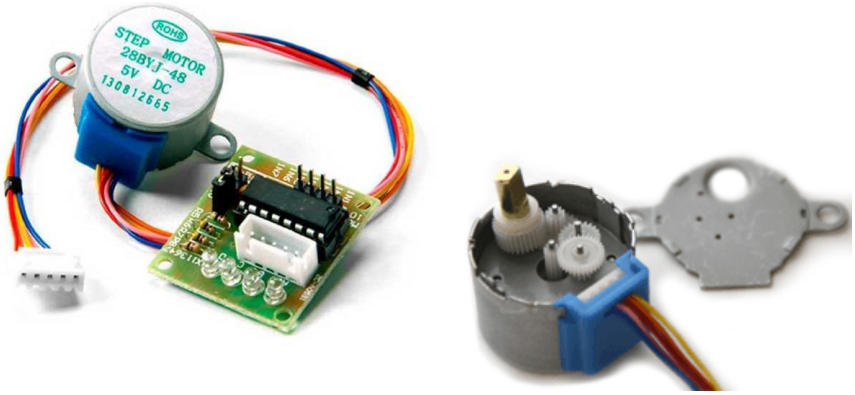


Figura 19. Motor 28BYJ-48 y PCB con driver ULN2003 integrado

El ULN2003 es un driver compuesto por siete transistores Darlington. El objetivo de estos transistores es aumentar de forma significativa la ganancia de corriente obtenida y de esta forma poder controlar el motor con corrientes de entrada pequeñas.

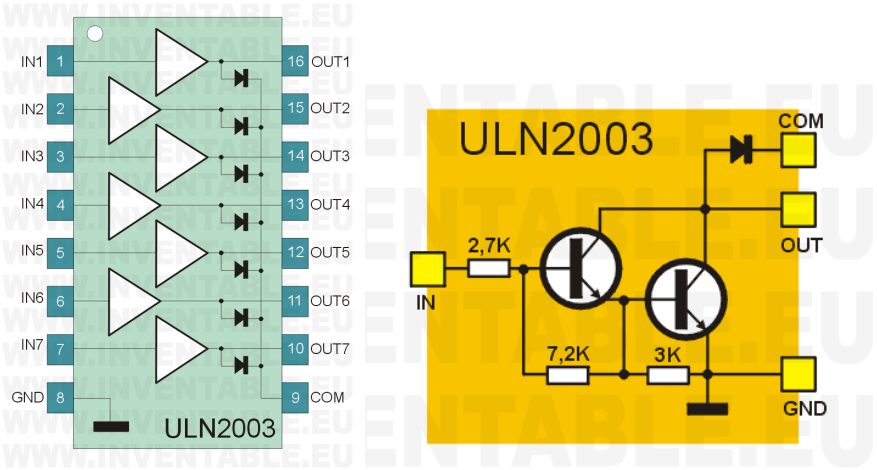


Figura 20. Esquema eléctrico del ULN2003 y de un transistor Darlington

En el caso del motor 28-BYJ48 solo empleamos cuatro transistores, uno por cada fase del motor.

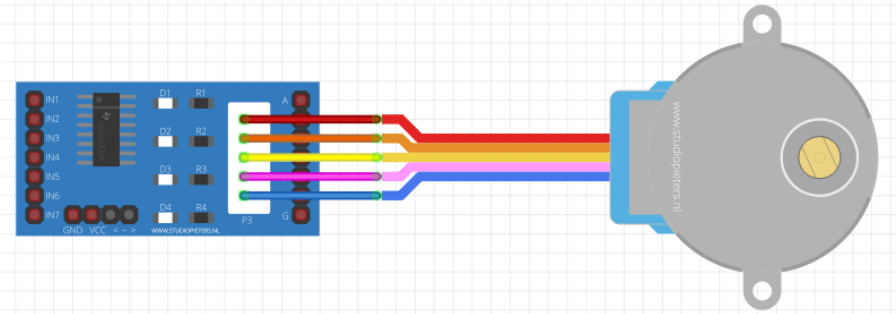


Figura 21. Conexión entre 28-BYJ48 y driver ULN2OO3

Dependiendo de la secuencia elegida para excitar las fases (la mitad de una bobina) en cada paso, este motor dispone de varios modos de funcionamiento.

- Modo de paso completo: En este caso se excita una única fase en cada segmento de la secuencia.
- Modo de paso doble: En este caso siempre hay dos fases excitadas en cada segmento de la secuencia, lo que genera un campo magnético mayor y el par del motor aumenta. Como desventaja, también aumenta el consumo energético.
- Modo de medio paso: En este caso se excitan alternativamente una y dos fases del motor, lo cual proporciona un movimiento más suave por parte del motor y una resolución por paso máxima.

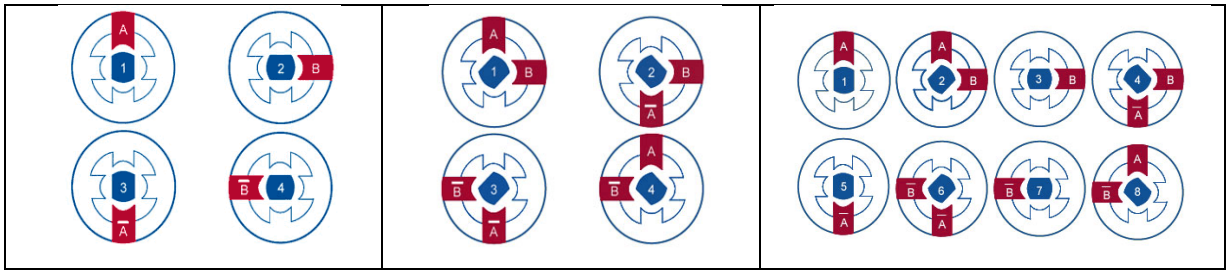


Figura 22. Secuencias de excitación en motores paso a paso unipolares (3)

Si las cuatro fases del motor están conectadas a sendos pines digitales de un microcontrolador, las secuencias de excitación para los tres modos anteriores siguen el siguiente orden.

Nº paso	IN1	IN2	IN3	IN4	Decimal
1	1	0	0	0	12
2	0	1	0	0	6
3	0	0	1	0	3
4	0	0	0	1	9

Tabla 4. Secuencia de excitación de paso completo

Nº paso	IN1	IN2	IN3	IN4	Decimal
1	1	1	0	0	12
2	0	1	1	0	6
3	0	0	1	1	3
4	1	0	0	1	9

Tabla 5. Secuencia de excitación de paso doble

Nº paso	IN1	IN2	IN3	IN4	Decimal
1	1	0	0	0	8
2	1	1	0	0	12
3	0	1	0	0	4
4	0	1	1	0	6
5	0	0	1	0	2
6	0	0	1	1	3
7	0	0	0	1	1
8	1	0	0	1	9

Tabla 6. Secuencia de excitación de medio paso

(3) https://aprendiendoarduino.wordpress.com/2018/10/17/motor-paso-a-paso-con-arduino/

3 Entorno MATLAB de simulación y desarrollo

MATLAB (abreviatura de Matrix Laboratory) es un sistema algebraico computacional compuesto por un entorno de desarrollo integrado que emplea un lenguaje propio de alto nivel. El entorno de MATLAB integra el cálculo numérico, la programación de comandos y la visualización de variables para la creación de distintas aplicaciones.

El elemento básico de operación en MATLAB es la matriz, lo que permite resolver problemas que involucran este tipo de variables de forma mucho más rápida que empleando un lenguaje de programación centrado en variables escalares como C.

Este software es una herramienta que se emplea tanto en cursos de formación de matemáticas e ingeniería como en centros de investigación para el desarrollo de nuevos productos tecnológicos.

Figura 23. Logo de MATLAB

Una de las mayores ventajas de MATLAB es la posibilidad de estudiar el desarrollo de dicho entorno, lo cual ha permitido tanto a desarrolladores como usuarios crear distintos módulos de trabajo para resolver una gran cantidad de problemas concretos, facilitando el uso de este software y extendiendo su rango de utilidad. Esos módulos se agrupan en bibliotecas de funciones llamadas toolboxes.

Simulink es una importante toolbox de MATLAB centrada en la simulación de comportamientos de sistemas dinámicos, tanto lineales como no lineales, mediante el empleo de diagramas de bloques en el entorno de simulación. Esto permite representar un sistema de forma muy intuitiva para el usuario.

Existen muchos tipos de bloques enfocados a distintos usos y asociaciones con hardware real (señales eléctricas, operaciones matemáticas, etc.).

3.1 Paquetes de compatibilidad con Arduino

El principal aspecto de este trabajo es el desarrollo de aplicaciones en plataformas Arduino mediante el entorno de desarrollo de MATLAB. Para ello emplearemos dos paquetes de compatibilidad de descarga gratuita entre los dos elementos.

-Paquete de compatibilidad de MATLAB con Arduino

Este paquete es una toolbox que genera una comunicación entre el entorno de MATLAB y las plataformas Arduino, mediante una serie de funciones agrupadas en librerías.

configurePin	Configura el modo de funcionamiento de un pin de Arduino
readDigitalPin	Lee un dato de un pin digital de Arduino
writeDigitalPin	Escribe un dato en un pin digital de Arduino
writePWMVoltage	Escribe el valor de tensión de un pin digital de Arduino que opera como salida PWM
writePWMDutyCycle	Configura el ciclo de trabajo de una señal PWM emitida desde un pin digital de Arduino
playTone	Provoca un tono de determinada frecuencia en un altavoz conectado a un pin digital de Arduino
readVoltage	Lee la tensión de un pin que opera como entrada analógica

Tabla 7. Funciones de lectura y escritura de datos

i2cdev	Conecta un dispositivo al bus I2C de Arduino
spidev	Conecta el Arduino a un dispositivo SPI
readRegister	Lee un dato del registro del dispositivo I2C
writeRegister	Escribe en el registro del dispositivo I2C
scanI2CBus	Escanea la dirección de un bus I2C en un elemento Arduino
read	Lee un dato desde el bus I2C
write	Escribe un dato en el bus I2C
writeRead	Devuelve un dato escrito a un dispositivo SPI

Tabla 8. Funciones de comunicación por I2C y SPI

shiftRegister	Conecta un registro shift a un hardware Arduino
shiftRegister/read	Lee un dato de un registro shift
shiftRegister/write	Escribe un dato en un registro shift
reset	Borra todas las salidas programadas en un registro shift

Tabla 9. Funciones de comunicación con registros de desplazamiento

Los registros de desplazamiento proporcionan entradas y salidas adicionales al sistema. Los modelos compatibles con el paquete son los siguientes: 74HC165, 74HC595 y 74HC164.

rotaryEncoder	Conecta un encoder incremental al hardware Arduino
resetCount	Reinicializa el contador de pulsos del encoder
readCount	Lee el valor del contador del encoder junto con el tiempo transcurrido desde el inicio del mismo
readSpeed	Lee la velocidad actual percibida por el encoder
servo	Conecta un servomotor al hardware Arduino.
readPosition	Lee la posición actual de un servomotor
writePosition	Mueve un servomotor hasta la posición indicada

Tabla 10. Funciones relacionadas con servos y encoders de cuadratura

Estas funciones permiten trabajar de forma muy sencilla y precisa con estos componentes en MATLAB y haremos uso de ellas en la programación de las aplicaciones.

Por último, existe un grupo de funciones que permiten manejar motores conectados al "Adafruit Motor Shield v2". Esta plataforma de hardware adicional (shield) se acopla a una tarjeta Arduino y permite conectar simultáneamente 2 servos y hasta 4 motores DC bidireccionales y 2 motores paso a paso.

El empleo de este shield y de las funciones asociadas al mismo es la forma más eficaz, intuitiva y potente de controlar varios motores a la vez. Sin embargo, es una opción algo más costosa, ya que requiere de una plataforma adicional.

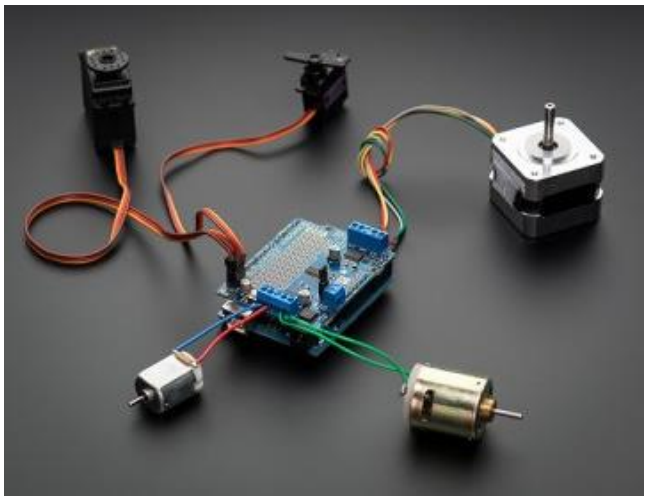


Figura 24. Diversos motores conectados a un Adafruit Motor Shield v2

-Paquete de compatibilidad de Simulink con Arduino

Este paquete permite la ejecución de modelos de bloques de Simulink en las plataformas Arduino.

Este paquete se compone de cuatro librerías de bloques:

1) Elementos comunes: Las principales operaciones de estos bloques son la asociación con entradas y salidas analógicas o digitales con un hardware Arduino, la comunicación externa en serie o con otros dispositivos y la lectura/escritura de posición para un servomotor.

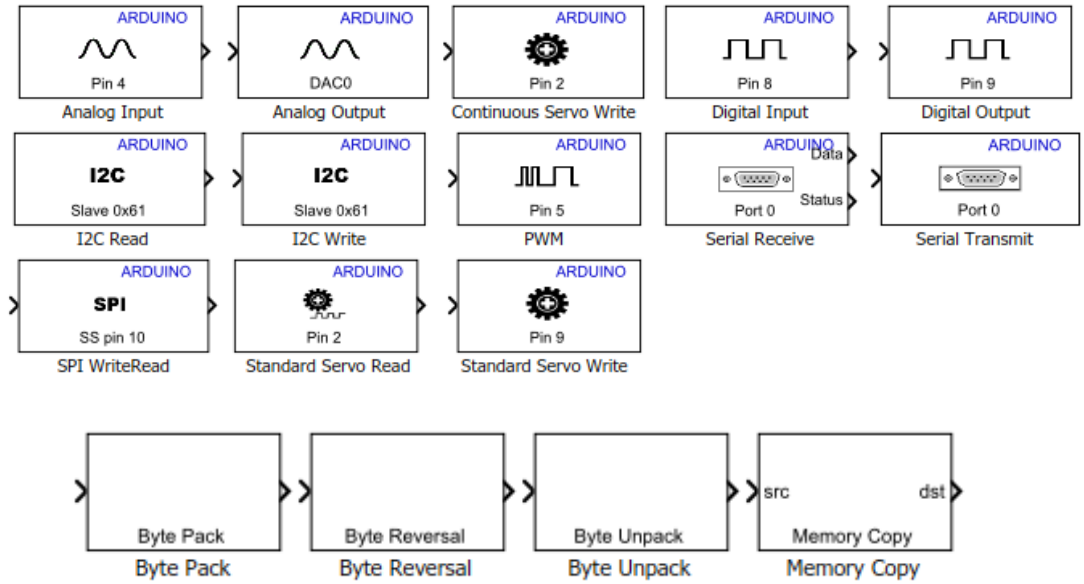


Figura 25. Bloques de elementos comunes

Los últimos cuatro bloques sirven para convertir señales de entrada a formato uint8 y viceversa, copiar hacia o desde un bloque de memoria o cambiar el orden de los bits para que unos datos de formato little endian puedan ser leídos por un procesador de formato big endian.

2) Comunicación con un shield Ethernet: Esta plataforma adicional permite conectar el PC con una tarjeta Arduino vía Ethernet para obtener una conexión más estable y con mayor ancho de banda.

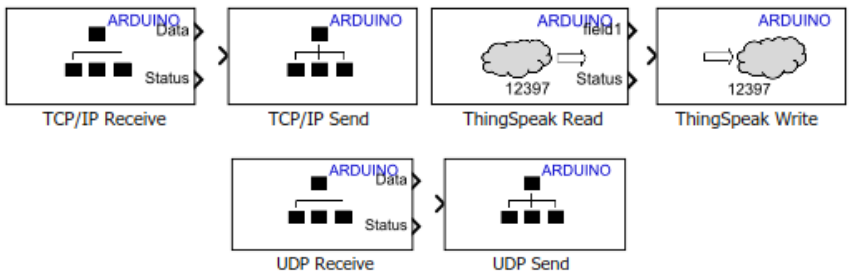


Figura 26. Bloques de comunicación Ethernet

3) Comunicación vía WiFi: Estos bloques permiten mandar y recibir datos de dispositivos conectados a una red de protocolos TCP/IP o UDP o de un canal ThingSpeak.

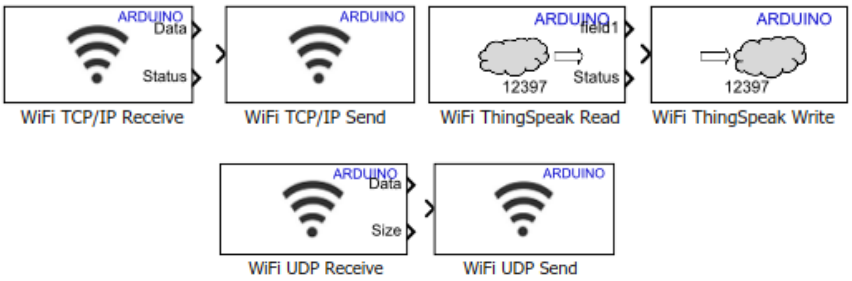


Figura 27. Bloques de comunicación WiFi

3.2 Ejecución de scripts y modelos en Arduino

En MATLAB, un script se ejecuta manteniendo una comunicación en serie con la tarjeta Arduino. Ésta última actúa de forma pasiva, actualizando sus entradas y salidas, mientras que la computadora de MATLAB se encarga de realizar los cálculos necesarios de la aplicación.

De esta forma:

- Se reduce la carga de trabajo del microcontrolador y se facilita la integración de las distintas funciones de MATLAB, al ser en este entorno donde se ejecuta la programación de las aplicaciones.
- Se pueden realizar aplicaciones que requieran cálculos complejos incluso con un microcontrolador de capacidades básicas.
- Algunas formas de conexión con el microcontrolador requieren una conexión en serie permanente entre la tarjeta Arduino y el PC, debido a la dependencia del procesador de MATLAB. Esto impide la construcción de aplicaciones móviles de motores, en las que el motor y la tarjeta Arduino deban poder desplazarse con libertad lejos del PC en el que se ejecuta la aplicación.
- Debido a la transmisión serie, el tiempo de ejecución de las sentencias es mayor en comparación con un script programado en el IDE de Arduino.

Una plataforma Arduino puede conectarse a MATLAB mediante USB, Bluetooth o WiFi, empleando la función “arduinosetup”.

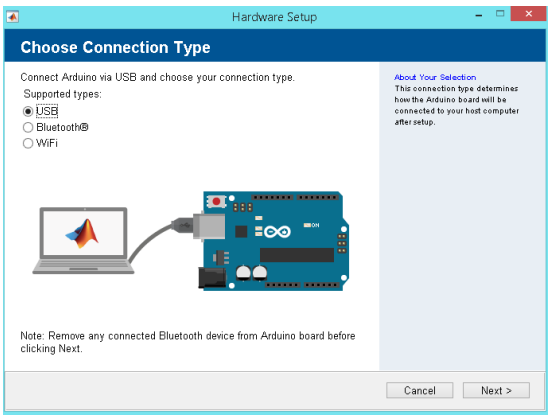


Figura 28. Conexión con hardware Arduino en MATLAB

La conexión por bluetooth puede realizarse con tres dispositivos: dos shields especializados o un módulo HC-05.

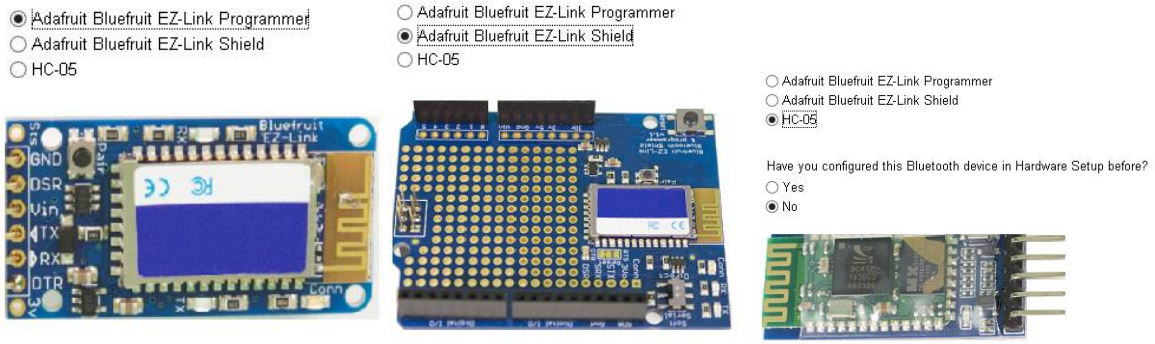


Figura 29. Dispositivos Bluetooth para conectar con Arduino

Sin embargo, únicamente los dos primeros permiten realizar la conexión Bluetooth de manera inalámbrica, que permiten solucionar el inconveniente de la conexión con cable en aplicaciones móviles.

El módulo HC-05 requiere conectarse a un adaptador FTDI de 3,3 V y éste a su vez al PC.

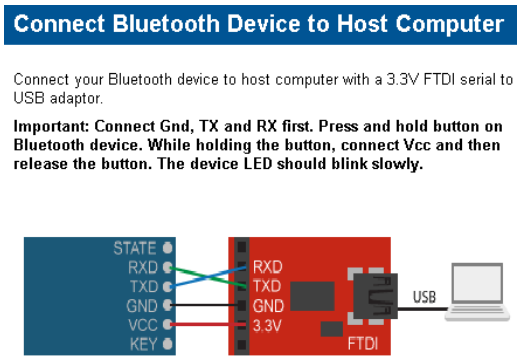


Figura 30. Conexión Bluetooth con HC-05 y driver

Solo se puede conectar una tarjeta Arduino a MATLAB si su puerto UART0 se encuentra disponible.

En nuestro proyecto, salvo que se indique lo contrario, siempre emplearemos una conexión USB para a la hora de realizar las aplicaciones con motores, debido a la estabilidad de la conexión por cable frente al WiFi, al hecho de que no se requiera ningún hardware adicional y al ahorro de una fuente de alimentación externa, al poder usar el PC para dicho propósito.

Por otra parte, la ejecución de las aplicaciones que diseñamos en Simulink es muy distinta. En vez de ejecutar una serie de sentencias de forma secuencial como en el caso de los scripts, en su lugar se ejecuta el modelo de Simulink y se genera un código en C.

Este código ejecuta las acciones programadas en el modelo de Simulink en cada periodo de muestreo, instantes en los cuales la aplicación cambia de estado dependiendo de las directrices programadas en los distintos bloques.

De esta forma, una aplicación en Simulink es similar a una interrupción periódica que altera las entradas y salidas del sistema.

Las principales diferencias con respecto a la programación vía script son las siguientes:

- Gracias el editor de bloques de Simulink las aplicaciones desarrolladas son mucho más fáciles de organizar y de interpretar que mediante la ejecución de sentencias en los scripts de MATLAB. Esto permite programar aplicaciones complejas de forma clara para el usuario.
- A diferencia del código de los scripts, los modelos en Simulink se pueden volcar en las plataformas Arduino y ejecutarse de forma embebida en la tarjeta. De esta forma las aplicaciones se pueden ejecutar sin necesidad de interactuar con el entorno de Simulink. Esta solución es ideal en el caso de aplicaciones que requieren ejecutar un gran número de sentencias en poco tiempo o de aplicaciones móviles (como un vehículo), que necesiten que la plataforma Arduino no se encuentre conectada al PC.
- También puede emplearse el modo External para alterar parámetros de un modelo durante su ejecución y monitorizar en el modelo los resultados del cambio realizado en tiempo real, con el fin de solucionar errores de diseño. En este caso se requiere una comunicación en serie entre Simulink y Arduino.
- Es importante destacar que cuando el modelo se vuelca en la tarjeta, no podremos visualizar la evolución de las distintas variables mediante bloques Scope o Display, a pesar de que estén definidas y tengamos abierto el entorno de Simulink.

En nuestro proyecto utilizaremos los dos modos de ejecución, uno para probar las aplicaciones de forma inalámbrica y el otro para monitorizar las variables del modelo con el objetivo de depurar la aplicación.

Para volcar un modelo en la tarjeta seguimos los siguientes pasos:

-Pinchamos en “Tools” y después en “Run on Tarjet Hardware” y en “Options” para abrir el menú de parámetros de configuración del modelo.

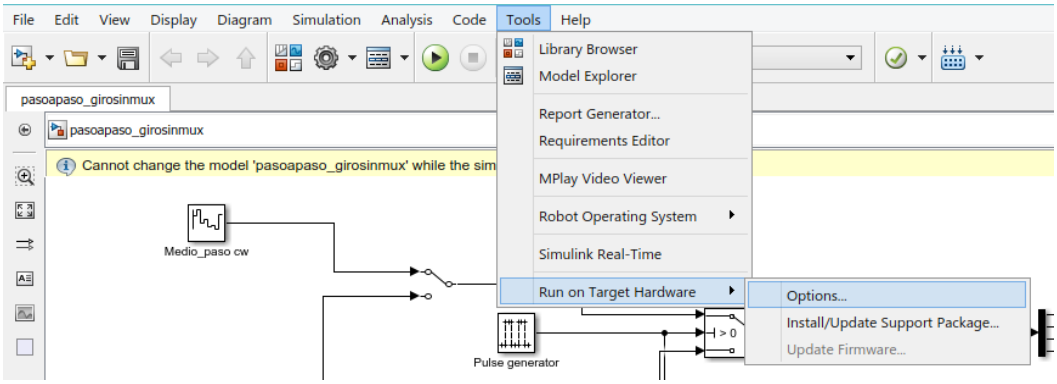


Figura 31. Opción para volcar un modelo de Simulink en una plataforma

-Una vez abierta la ventana de parámetros de configuración, seleccionamos la plataforma que queramos utilizar en el menú desplegable de la pestaña Hardware Implementation.

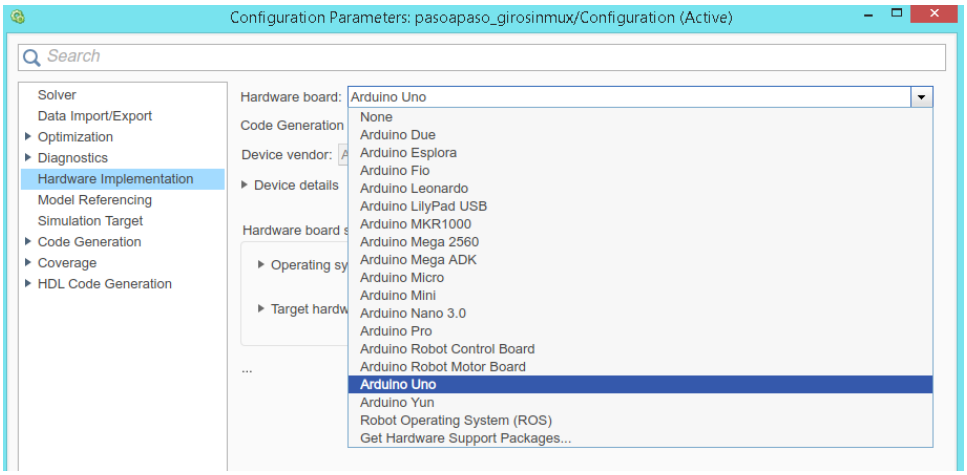


Figura 32. Pestaña Hardware Implementation del menú Options

-Por último, con la tarjeta conectada, el modo normal activado y un tiempo infinito para la ejecución, pinchamos en el ícono Deploy To Hardware.

Figura 33. Ejecución de código embebido en Arduino con Simulink

Una vez realizados los pasos anteriores el código se ejecuta de forma embebida en la tarjeta en el momento en el cual se alimenta el microcontrolador, con el cable USB o mediante alimentación externa.

Si por el contrario queremos monitorizar las variables y alterarlas en tiempo real, es necesario seleccionar el modo External y pinchar en el icono de “Play”.

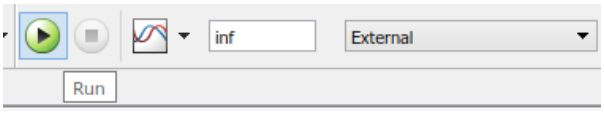


Figura 34. Ejecución para monitorización en tiempo real en Simulink

En ambos casos, el puerto UART0 de la tarjeta Arduino debe estar disponible para la ejecución de los modelos.

4 Aplicaciones con motores

A continuación se exponen las aplicaciones experimentales que se han llevado a cabo para comprobar la efectividad de MATLAB y Simulink y los paquetes de compatibilidad de ambos con Arduino.

Cada aplicación consiste principalmente en conseguir un control eficaz de los distintos motores (DC, servos y motores paso a paso) en ambos entornos (MATLAB y Simulink), señalando si alguno de los dos ofrece una mayor utilidad frente a otro dependiendo del tipo de motor a controlar.

Una vez comprobado el manejo de los motores en ambos entornos se añadirá una característica extra a cada aplicación, con el objetivo de presentar otras funciones de los paquetes de compatibilidad o de complementar dichos paquetes con otras herramientas integradas de base en MATLAB y Simulink.

Por último, se extraerán las conclusiones resultantes de las aplicaciones y se propondrá una idea para el desarrollo de una aplicación más compleja en base a las mismas.

4.1 Aplicación con motor DC GA12-N20

La aplicación a desarrollar en el proyecto con respecto al motor DC consiste en realizar las siguientes acciones:

- Comprobar la relación entre la tensión suministrada al motor y la velocidad alcanzada por el mismo en las condiciones de trabajo del proyecto.
- Manejar el motor mediante dos elementos: un interruptor para configurar el sentido de giro y un potenciómetro para controlar la velocidad.
- Durante el manejo del motor, se registrará periódicamente la velocidad alcanzada por el motor y se representará mediante gráficas en el entorno MATLAB/Simulink y posteriormente, en una pantalla LCD conectada a la plataforma Arduino vía comunicación I2C.

4.1.1 Cálculo de velocidad y manejo del motor en MATLAB y Simulink

Comenzaremos el manejo del motor DC con MATLAB, mediante la programación y ejecución de scripts. Mientras usemos MATLAB, la tarjeta se comunica en serie con el PC.

El montaje hardware construido para calcular las velocidades es el siguiente:

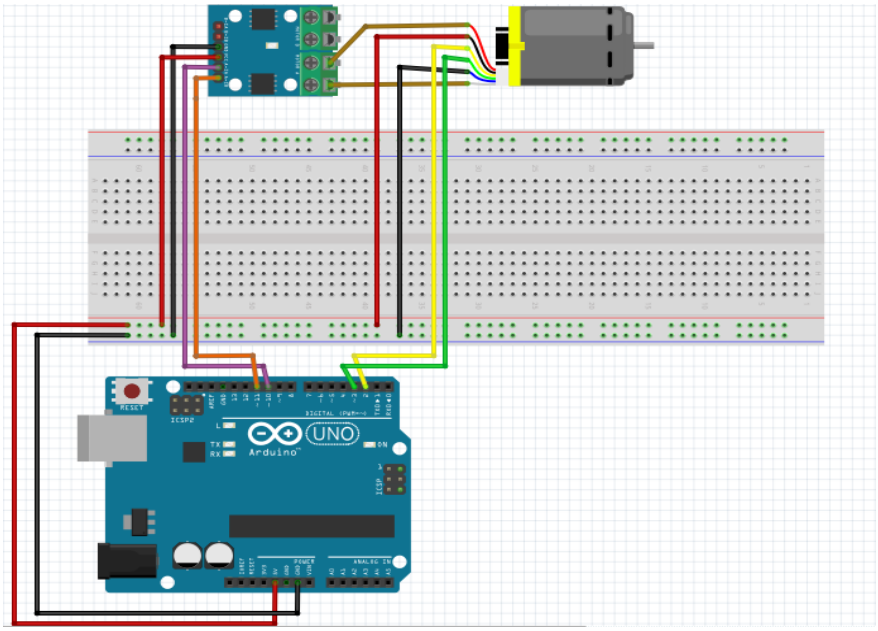


Figura 35. Montaje eléctrico motor DC

Las entradas del driver AIA y AIB (cables morado y naranja, respectivamente), se conectan a los pines D10 y D11, que pueden operar como salidas PWM.

Los canales del encoder C1 y C2 (cables amarillo y verde, respectivamente) se conectan a los pines D2 y D3, que operan como entrada de los canales A y B del encoder.

Es importante definir la librería rotaryEncoder a la hora de conectar la tarjeta Arduino con MATLAB para poder utilizar las funciones relacionadas con los encoders incrementales. Las principales funciones utilizadas son:

>> rotaryEncoder(a, P1, P2, ppr): Esta función crea un objeto de tipo encoder en el espacio de trabajo de MATLAB. Para ello hay que indicar la tarjeta Arduino a la cual se conecta el encoder (a), los pines de entrada de los canales A y B del encoder (P1 y P2) y, de forma opcional, la resolución del dispositivo (ppr).

>> writePWMDutyCycle(a, p, n): Esta función genera una señal PWM en un pin p de un objeto hardware Arduino a. El ciclo de trabajo viene indicado por un porcentaje (por ejemplo: n=0.5 equivale a un ciclo del 50%).

>> readCount(enc): Esta función lee el número actual de pulsos generados por el encoder (el contador aumenta o disminuye en 4 por cada pulso recorrido en sentido horario o antihorario).

>> readSpeed(enc): Esta función analiza brevemente el incremento del contador de pulsos recorridos del encoder enc y proporciona la velocidad actual que registra el mismo de forma inmediata, lo cual permite trabajar con estos dispositivos de una forma muy sencilla en MATLAB.

El script programado para el movimiento del motor es el siguiente:

```
%Lectura de velocidades del motor DC

clear all;
a=arduino('COM5','Uno','Libraries','rotaryEncoder');
encoder = rotaryEncoder(a,'D2','D3',210);
resetCount(encoder);

velocidades_cw = zeros(1,21);   %array de velocidades en sentido horario
velocidades_acw = zeros(1,21);  %array de velocidades en sentido antihorario
i=1; %índice de arrays

AIA = 'D10';   %Salidas PWM conectadas al driver L9110s
AIB = 'D11';
writePWMVoltage(a,AIA,5);  %Ajustamos la amplitud de las señales PWM a 5V
writePWMVoltage(a,AIB,5);

for porcentaje = 0:0.05:1     %El ciclo de trabajo de AIA aumenta en un 5%
    writePWMDutyCycle(a,AIA,porcentaje);  %cada vez
    writeDigitalPin(a,AIB,0);
    pause(3);                 %Tras la pausa se registra la velocidad
    velocidades_cw(i) = readSpeed(encoder);
    i=i+1;
end
i=1;
for porcentaje = 0:0.05:1     %se repite el proceso girando en el otro
    writeDigitalPin(a,AIA,0); %sentido
    writePWMDutyCycle(a,AIB,porcentaje);
    pause(3);
    velocidades_acw(i) = readSpeed(encoder);
    i=i+1;
end

writeDigitalPin(a,AIA,0);     %apagamos el motor
writeDigitalPin(a,AIB,0);

%Gráfica de resultados en valores absolutos
x=0:0.05:1;
plot(x,velocidades_cw);
grid on
hold on
plot(x,-velocidades_acw);
```

En este script se incrementa cada 3 segundos el ciclo de trabajo de las señales PWM que controlan la velocidad del motor en un 5%, primero girando hacia la derecha (AIA=PWM y AIB=0) y después hacia la izquierda (AIA=0 y AIB=PWM).

En cada uno de los ciclos se registra la velocidad del motor gracias al encoder y a la función **>>readSpeed**. Las velocidades se registran con signo negativo cuando el motor gira en sentido antihorario.

Con estos datos, podemos relacionar la velocidad del motor DC (en ausencia de carga) con los ciclos de trabajo de las señales de control del driver. Este estudio se realiza con el driver L9110S alimentado a 5 V desde la tarjeta Arduino.

Se obtienen los siguientes resultados conforme aumenta el ciclo de trabajo de las señales de control.

Ciclo de trabajo	Velocidades en sentido horario (rpm)	Velocidades en sentido antihorario (rpm)
0	0	0
0.05	0	0
0.1	0	0
0.15	0	0
0.2	0	0
0.25	0	-53,6
0.3	64,3	-85,7
0.4	100	-132,1
0.5	139,3	-164,3
0.6	175	-196,4
0.7	203,6	-221,4
0.8	225	-235,7
0.9	239,3	-250
1	260,7	-271,4

Tabla 11. Velocidades del motor DC alimentado a 5V

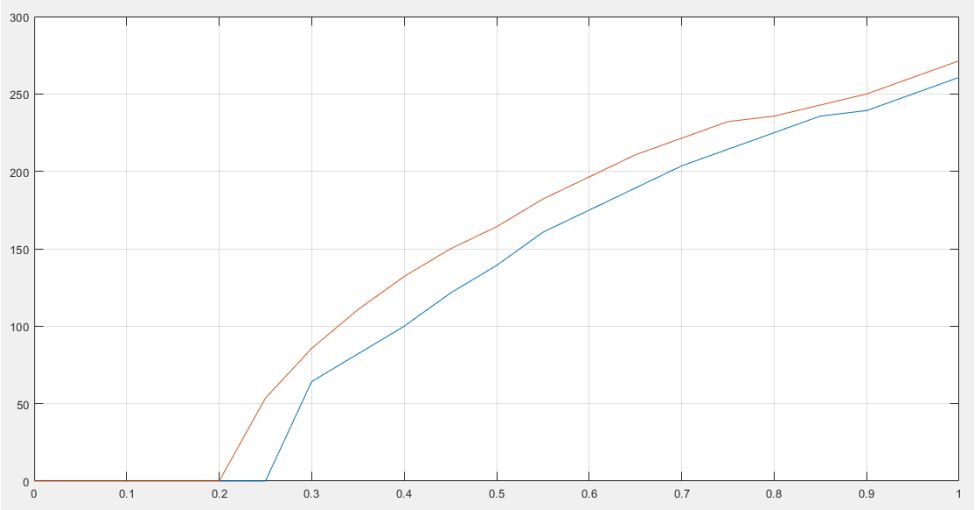


Figura 36. Evolución de la velocidad respecto al ciclo de trabajo

Según los resultados obtenidos, el motor DC gira a mayor velocidad hacia la izquierda (gráfica naranja) y precisa de un menor valor de tensión para comenzar a moverse.

A continuación añadimos al montaje un interruptor y un potenciómetro para poder controlar el motor de forma manual.

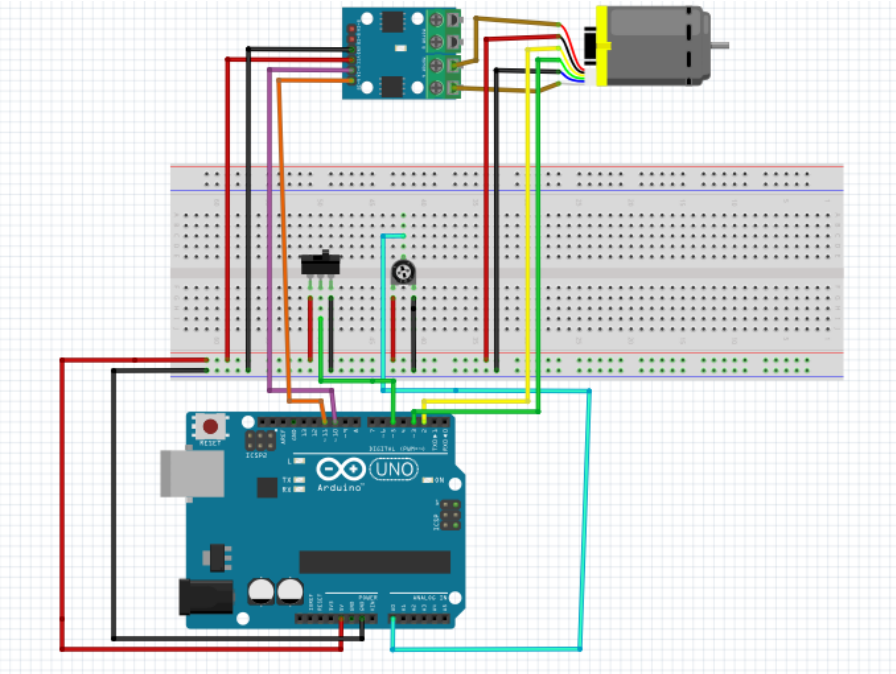


Figura 37. Montaje de control manual de motor DC

El terminal central del potenciómetro (cian) se conecta a una entrada analógica (A0) y el interruptor a una entrada digital (D5).

El script programado para el movimiento manual del motor DC es el siguiente.

```
%Movimiento del motor DC, con potenciómetro e interruptor para controlar la
%velocidad y el sentido de giro

clear all;
a=arduino('COM5','Uno','Libraries','rotaryEncoder');
encoder = rotaryEncoder(a,'D2','D3',210);

AIA = 'D10';    %Salidas PWM conectadas al driver L9110s
AIB = 'D11';
writePWMVoltage(a,AIA,5);  %Ajustamos la amplitud de las señales PWM a 5V
writePWMVoltage(a,AIB,5);

%Entradas del potenciometro y swith
potenciometro = 'A0';
interruptor = 'D5';

%Variables del motor
sentido = 0;    %0: giro a la derecha     1: giro a la izquierda
tension = 0;

velocidades = 0;
muestras = 1;
resetCount(encoder);
```

```
%De forma continua, se actualizan los valores de sentido y velocidad del
%motor y se registra este último parámetro.
for i=1:200
    muestras(i) = i;
    sentido = readDigitalPin(a,interruptor);
    tension = readVoltage(a,potenciometro);
    tension = round((tension/5),2);   %conversión a porcentaje

    if sentido == 0        %dependiendo del estado del interruptor el motor
        writePWMDutyCycle(a,AIA,tension);  %gira en un sentido u otro
        writeDigitalPin(a,AIB,0);
    else
        writePWMDutyCycle(a,AIB,tension);
        writeDigitalPin(a,AIA,0);
    end

    pause(0.03)        %se registra la velocidad y se muestra en una gráfica
    velocidades(i) = readSpeed(encoder);  %que se actualiza en cada ciclo
    plot(muestras,velocidades,'r');
    axis([0 200 -300 300]);
    grid on
    pause(0.02);
end

writeDigitalPin(a,AIA,0);
writeDigitalPin(a,AIB,0);
```

En este caso se toman 200 muestras de velocidad, con un intervalo de 50 ms entre ellas, antes de detener el movimiento del motor.

Las funciones que ofrece el paquete de compatibilidad de Arduino con MATLAB (lectura/escritura y encoders de cuadratura) permiten manejar correctamente un motor DC y medir su velocidad de forma precisa e inmediata mediante un script.

Sin embargo, para realizar esto es necesario mantener una comunicación en serie constante entre el entorno de MATLAB y la tarjeta Arduino mediante USB.

Esto provoca una limitación importante: el motor DC debe permanecer estacionado en un mismo lugar para poder permitir la conexión con cable, es decir, estos scripts no pueden desarrollarse para aplicaciones móviles tales como un vehículo teledirigido, en las cuales el montaje hardware (Arduino y motores) se desplaza.

Podemos solucionar este problema mediante un modelo de Simulink volcado en la tarjeta, al ejecutar el código de forma embebida.

El siguiente modelo está configurado con un periodo de muestreo Ts = 1ms.

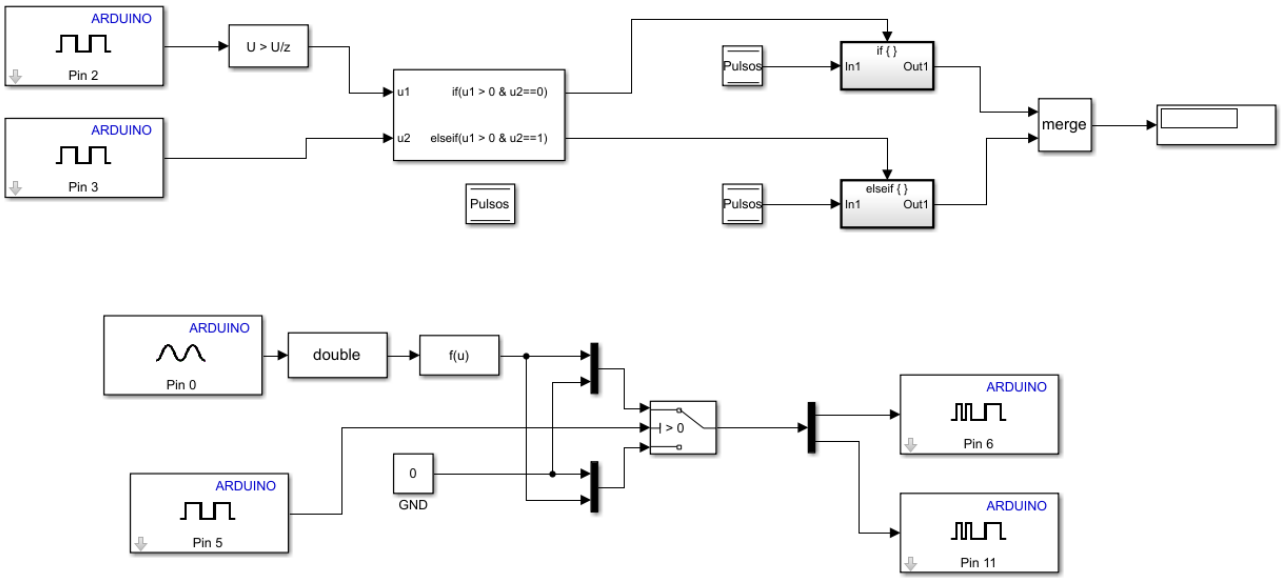


Figura 38. Modelo de movimiento manual de motor DC

Este diagrama se divide en dos partes independientes, que serán estudiadas por separado. El conjunto de bloques superior controla las señales recibidas por el encoder y el conjunto inferior maneja el motor en base a los valores del interruptor y el potenciómetro.

Se mantiene el montaje de la figura 37.

-Manejo del motor

La lógica de esta parte es idéntica a su contraparte en script. En base al sentido de giro se genera una señal PWM en uno de los pines y el otro se sitúa a nivel bajo. La lectura del pin 0 (de 0 a 1023) se convierte a un valor entre 0 y 255, proporcional al ciclo de trabajo de las señales PWM.

Al volcar el modelo sobre la tarjeta, es posible ejecutar la aplicación sin abrir el entorno de Simulink y sin necesidad de una conexión USB (si se emplea una alimentación externa).

El motor responde de manera satisfactoria, demostrando que puede emplearse el entorno de Simulink para el diseño de aplicaciones que requieran movilidad por parte de motores DC.

-Lectura de canales del encoder

El entorno de Simulink ofrece el bloque “Encoder Input” en su librería “Simulink Desktop Real-Time” para mostrar la respuesta de un encoder conectado a un microcontrolador.

Sin embargo, este bloque es incompatible con las plataformas Arduino, por lo que tendremos que diseñar un conjunto de bloques propio que detecte los pulsos de los canales del encoder y distinga el sentido de giro del motor que indican los mismos.

Como se indicó anteriormente, los canales A y B del encoder (conectados a los pines D2 y D3) generan un número determinado de pulsos por vuelta recorrida, y se sitúan desfasados 90º uno de otro dependiendo del sentido de giro.

Con ello, podemos interpretar un flanco de subida en el canal A como un pulso, y midiendo el nivel del canal B (alto o bajo) durante esos flancos podemos determinar el sentido de giro.

Generamos dos ondas cuadradas iguales a las de la figura 13, desfasadas 90º entre sí para simular las señales del encoder y definir el sistema de bloques.

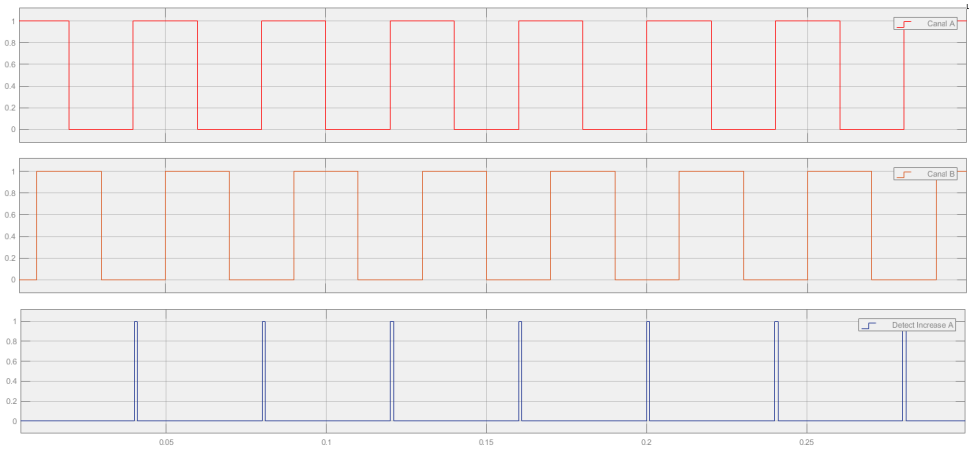


Figura 39. Canales A y B del encoder y flancos de subida de A en sentido horario

En este caso, el canal B se encuentra desfasado respecto al canal A y se sitúa a nivel bajo en cada flanco de subida del canal A.

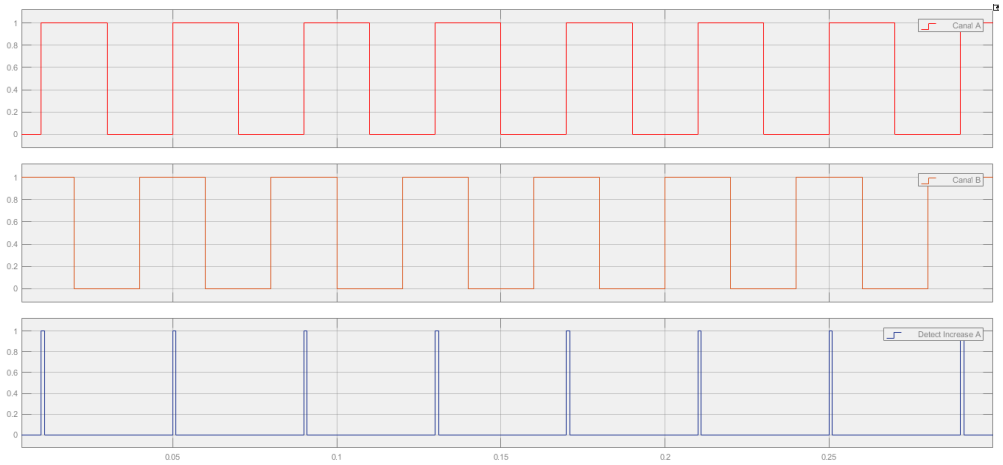


Figura 40. Canales A y B del encoder y flancos de subida de A en sentido antihorario

En este caso, el canal B se encuentra adelantado respecto al canal A y se sitúa a nivel alto en cada flanco de subida del canal A.

El algoritmo del modelo funciona de la siguiente forma: dependiendo del nivel del canal C2 en cada flanco de subida de C1 se lee un pulso positivo (giro a la derecha) o un pulso negativo (giro a la izquierda), modificando una variable del modelo llamada “Pulsos”, que funciona de forma similar a la función **>>readCount(encoder)** de MATLAB, aumentando o disminuyendo en 1 unidad si el motor recorre un pulso girando hacia la derecha o hacia la izquierda.

Según se registran los pulsos, medimos la variación del contador durante el transcurso de un número determinado de muestras para obtener una variable de magnitud pulsos/sg, que después convertimos a rpm.

En el siguiente ensayo, empleamos un periodo de muestreo de 0,5 ms, y aplicamos un retardo de 50 muestras al contador de pulsos para medir su variación. El motor está programado para girar al 40% (102/255) de su velocidad máxima hacia la derecha.

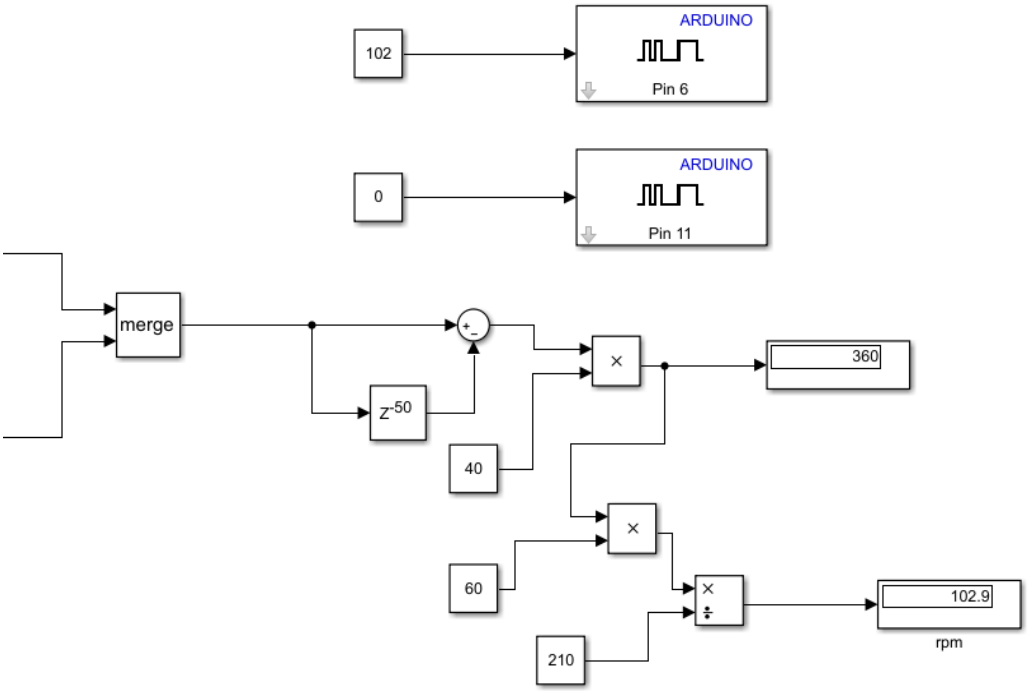


Figura 41. Monitorización de velocidad del motor DC en Simulink

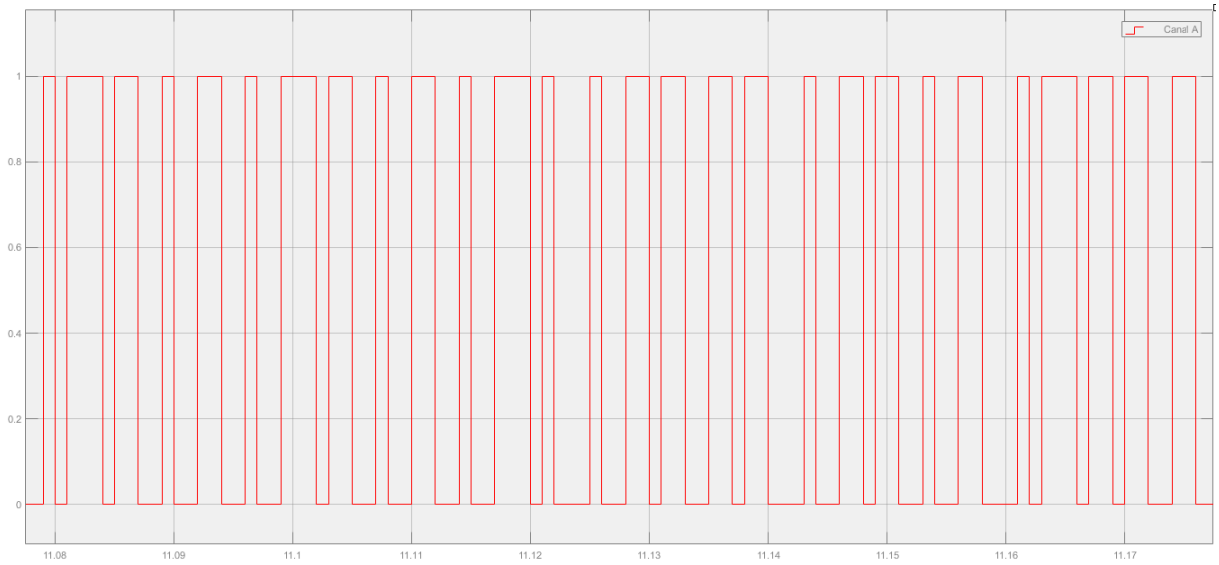


Figura 42. Canal A del encoder durante la aplicación

Los pulsos del encoder se registran de manera uniforme y con un ciclo de trabajo aproximado del 50%, de acuerdo a las especificaciones técnicas del dispositivo.

Según el resultado obtenido:

$$\frac{\Delta Pulsos}{50 * 0{,}5ms} = \frac{\Delta Pulsos}{25\ ms} * \frac{1000ms}{1s} = \Delta Pulsos * \frac{40}{sg} = \frac{360pulsos}{sg} = 102{,}9rpm \qquad [2]$$

Este resultado es muy similar al obtenido en el ensayo de MATLAB de la tabla 11, de 100 rpm, por lo que podemos considerar válido el algoritmo empleado en el modelo.

Sin embargo, se producen errores al aumentar la velocidad del motor y los pulsos se registran con un periodo mayor al esperado (menor velocidad) y con un ciclo de trabajo menos estable. Esto es debido a la alta resolución del encóder, de 210 ppr, y a la imposibilidad por parte del entorno de procesar correctamente los pulsos incluso con un periodo de muestreo reducido como 0,5 ms.

Por ejemplo, suponiendo una velocidad de 200 revoluciones por minuto se generan 700 pulsos por segundo en las salidas del encoder.

$$200rpm * \frac{210\ pulsos}{1\ rev} * \frac{1\ min}{60\ sg} = 700\frac{pulsos}{sg} \qquad [3]$$

Analizando la inversa de este resultado, determinamos que se genera un pulso cada 1,4 ms aproximadamente.

Esto implica que en tan solo tres instantes de muestreo Simulink debe registrar los periodos a nivel alto y bajo de cada pulso y la duración de los mismos, lo cual no es posible. En esta situación disminuir el periodo de muestreo es contraproducente, pues se producen diversos errores: el modelo tarda mucho más en cargar, el motor se mueve de forma intermitente, etc.

Para solucionar dicho problema, es necesario emplear un encoder de resolución inferior.

4.1.2 Representación de datos en pantalla por I2C en Simulink

El I2C (Inter-Integrated Circuit) es un protocolo de comunicación en serie entre dispositivos electrónicos mediante una jerarquía maestro-esclavo.

Este protocolo se compone de dos vías de comunicación: SDA (transmisión de datos) y SCL (señal de reloj).

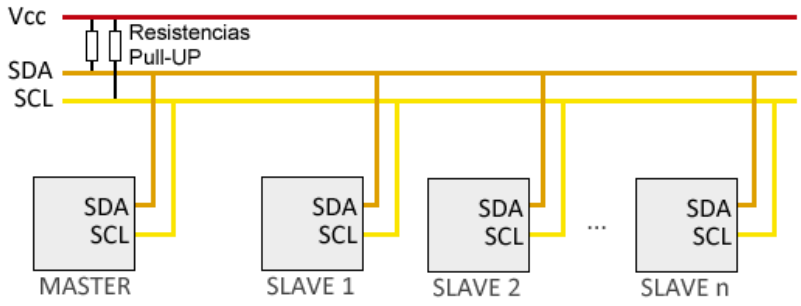


Figura 43. Conexión I2C

I2C opera de forma síncrona: el envío de datos está sincronizado por una misma señal de reloj que comparten todos los dispositivos a través de la vía SCL.

Cada dispositivo posee una dirección I2C única, con el objetivo de comunicarse de forma individual con cada uno de los dispositivos esclavos subordinados.

La transmisión de un mensaje por I2C sigue el siguiente esquema:

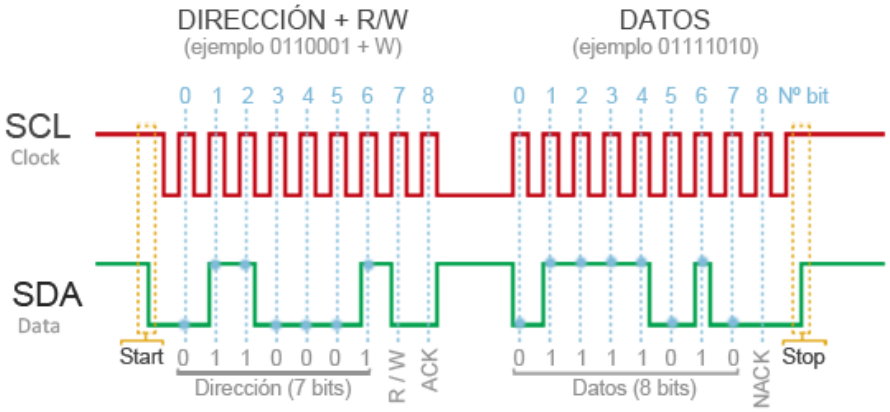


Figura 44. Transmisión de un byte de información por I2C

- El dispositivo maestro escribe a los esclavos tras producirse la condición de inicio: SDA se sitúa a nivel bajo y SCL la sigue poco después.
- Tras el inicio, el maestro escribe la dirección del dispositivo con el que queremos comunicarnos en todos los esclavos conectados, junto con un último bit R/W, que indica si se quiere realizar una operación de escritura o de lectura.
- El esclavo cuya dirección coincide con la escrita por el master devuelve a este un bit de validación ACK, a nivel bajo.
- A continuación el maestro escribe o lee la información dividida en bytes, comenzando por el bit más significativo. Tras cada byte de datos, el dispositivo receptor envía un bit de validación para indicar si la información se ha transmitido correctamente o no.
- Para detener la comunicación el maestro transmite la condición de STOP: sitúa la señal SDA a nivel alto y la parada se realiza en cuando la señal SCL también se encuentra a nivel alto.

Esta comunicación se emplea en multitud de dispositivos y sus principales ventajas son que se puede llevar a cabo únicamente con dos cables y el hecho de que existe una confirmación de un correcto funcionamiento.

Por el contrario, la transmisión de bits ajenos a la información con la que se quiere trabajar provoca que la velocidad de transmisión sea más lenta que con otros protocolos, oscilando esta entre los 100 kHz y los 400 kHz.

En nuestro proyecto utilizaremos una pantalla LCD 16x2 para representar variables de la aplicación.

Este tipo de pantallas requiere de hasta 16 pines para conectarla al Arduino, pero dispone de un adaptador I2C para poder trabajar con ella con tan solo el bus I2C de la tarjeta.

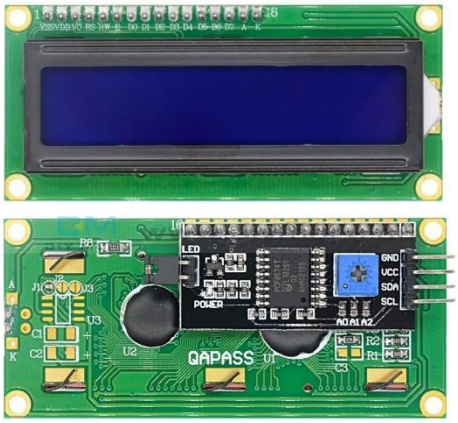


Figura 45. Pantalla LCD 16x2 con adaptador I2C

En la Arduino UNO, el bus I2C lo componen las entradas analógicas A4 y A5 (SDA y SCL).

Con los paquetes de compatibilidad podemos enviar y recibir datos por I2C tanto en MATLAB como en Simulink. Sin embargo, la programación de pantallas LCD no solo requiere de un conocimiento total sobre su funcionamiento interno, sino que en condiciones normales requiere la transmisión de una gran cantidad de comandos por I2C para inicializar la pantalla y para que esta muestre la información que queremos.

Para facilitar su uso, el IDE de Arduino dispone de librerías enfocadas a estos dispositivos y a la propia comunicación I2C. En este caso, se necesitan las librerías “LiquidCrystal_I2C” y “Wire”.

MATLAB y Simulink no disponen de ninguna librería dedicada a trabajar con Arduino más allá de las funciones y bloques que ofrecen los paquetes de compatibilidad, que engloban principalmente la lectura/escritura de datos, varios protocolos de comunicación y algunas librerías dedicadas al funcionamiento de servomotores, encoders y al shield Adafruit v2 que vimos en el apartado 3.2.

Ante este problema tenemos dos opciones:

- Crear una librería personalizada en MATLAB, adaptando el código de las librerías al lenguaje de MATLAB y diseñándola desde cero.

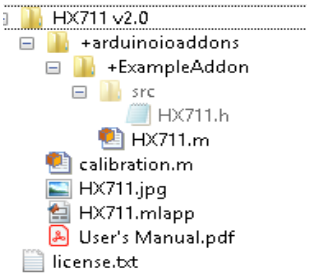


Figura 46. Ejemplo de librería de MATLAB personalizada para el sensor de presión HX711

- Utilizar la función S-function Builder de Simulink, que adapta un script del IDE de Arduino (librerías incluidas) al lenguaje de MATLAB y permite integrar el código de dicho script en un modelo.

En vista a los objetivos de accesibilidad que busca este proyecto, utilizaremos la segunda opción, ya que la primera requiere conocimientos muy avanzados de MATLAB para una correcta implementación, mientras que la otra únicamente requiere de pequeños cambios en el código del propio programa (no de las librerías) para poder utilizarla.

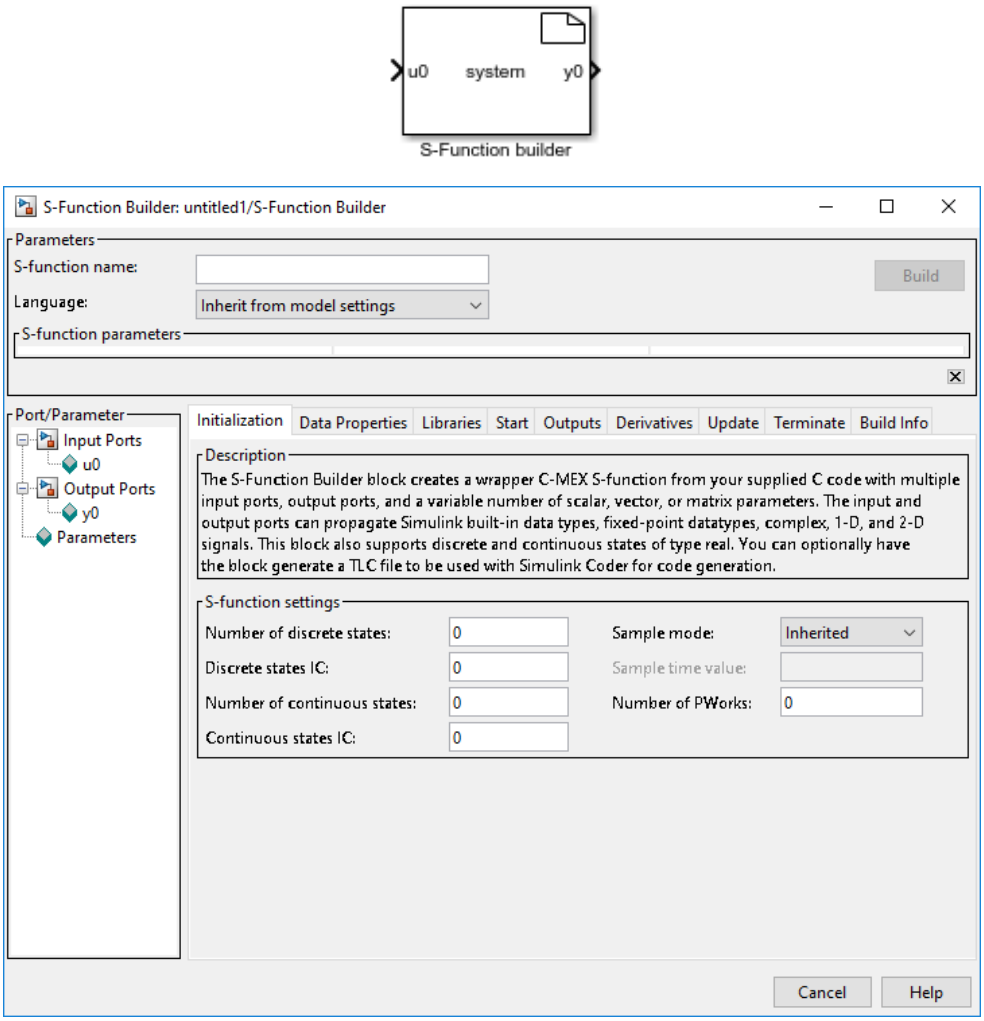


Figura 47. Bloque S-Function Builder

El bloque S-Function Builder genera una función compatible con los modelos de Simulink a partir de código escrito en lenguaje C/C++. Estas funciones son, en esencia, un modelo de entradas y salidas regulado por unas variables denominadas “estados continuos o discretos”. La programación se realiza en base al valor actual de dichos estados.

Un script programado en el IDE de Arduino se divide en tres apartados: inclusión de librerías y definición de variables y dos funciones principales, una de configuración que solo se ejecuta al inicio (setup) y una función periódica en la que se aloja el código principal (loop).

Teniendo en cuenta este esquema seguimos los siguientes pasos para crear una S-Function análoga a un script de Arduino:

- En la pestaña Inicialization seleccionamos 1 estado discreto. Este estado nos permitirá distinguir entre la programación de las funciones “setup” y “loop”. Al inicio de la ejecución el valor de este estado es 0.
- En la pestaña Libraries incluimos los archivos .h y .cpp de las librerías de Arduino que queramos utilizar y definimos las variables estáticas del programa. Estos archivos también deben guardarse en la misma carpeta que el modelo de Simulink. El bloque #ifndef MATLAB_MEX_FILE hasta #endif convierte código C/C++ para que pueda ser interpretado por MATLAB.

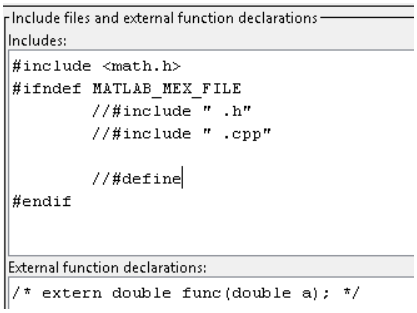


Figura 48. Pestaña Libraries

- En la pestaña Data Properties definimos el nombre y el tipo de las entradas y salidas del modelo.
- En la pestaña Update programamos el código relacionado con la actualización de los estados discretos (referidos como xD[0]...xD[n]). Teniendo en cuenta que nuestro estado inicia en 0 podemos utilizar un bloque if para programar la función setup y después cambiar su valor, para asegurarnos de que no vuelve a ejecutarse otra vez.

```
/*
 * Code example
 *   xD[0] = u0[0];
 */

if (xD[0] != 1)
{
    #ifndef MATLAB_MEX_FILE
    //Función setup
    #endif

    xD[0] = 1;
}
```

Figura 49. Pestaña Update

- En la pestaña Outputs escribimos la función loop de nuestro programa y cómo influye en las salidas del modelo si las hubiera.

```
/* This sample sets the output equal to the input
      y0[0] = u0[0];
 For complex signals use: y0[0].re = u0[0].re;
      y0[0].im = u0[0].im;
      y1[0].re = u1[0].re;
      y1[0].im = u1[0].im;
 */

if (xD[0] == 1)
{
    #ifndef MATLAB_MEX_FILE
    //Función loop
    #endif
}
```

Figura 50. Pestaña Outputs

Por último construimos la función con la opción Build, y de los archivos que se generan, renombramos el archivo "-wrapper.h" a formato .cpp e incluimos extern "C" antes de la definición de las funciones programadas en las pestañas Update y Outputs, para indicar que están escritas en este lenguaje.

La Arduino Due, al poseer un modelo de microcontrolador distinto al resto de plataformas Arduino, puede presentar errores de compatibilidad con ciertas librerías, por lo que es recomendable emplear estas funciones con el resto de tarjetas (UNO, Mega, Leonardo, etc).

Una vez comprendido el funcionamiento del método a emplear, incluimos la pantalla LCD en el montaje de la figura 37.

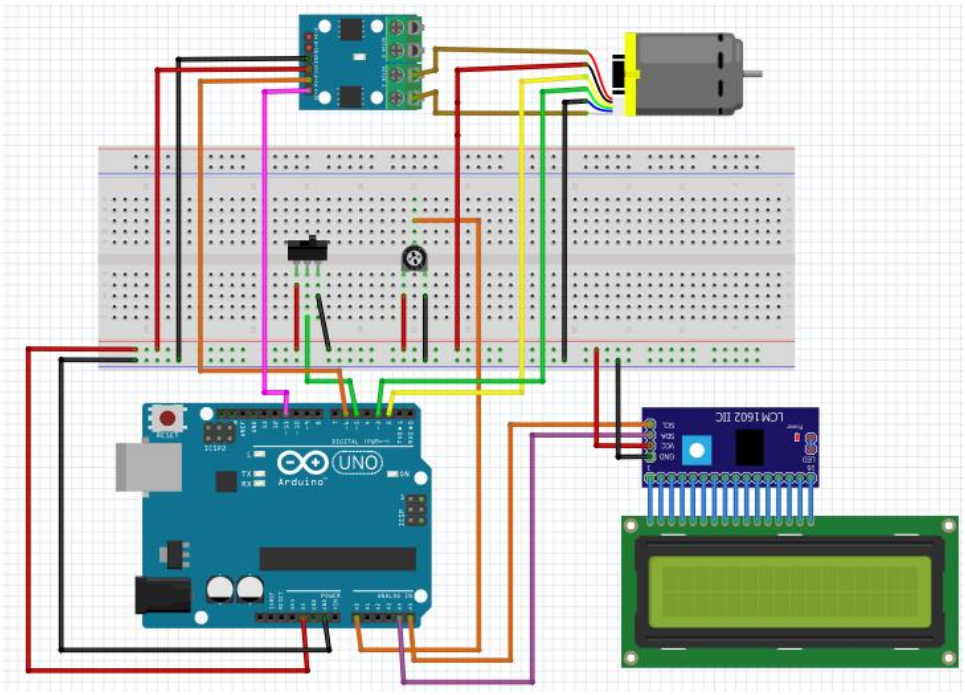


Figura 51. Aplicación motor DC con pantalla LCD 16x2

En vista a la incapacidad por parte de Simulink de reastrear correctamente los pulsos del encóder a velocidades medias y altas, se ha decidido en su lugar representar el ciclo de trabajo de la señal de control PWM del motor DC, y su sentido de giro mediante caracteres.

De esta forma podremos comprobar el funcionamiento de las librerías de Arduino con MATLAB/Simulink midiendo parámetros del modelo que conocemos como correctos.

Creamos el siguiente modelo, con un periodo de muestreo Ts = 1ms.

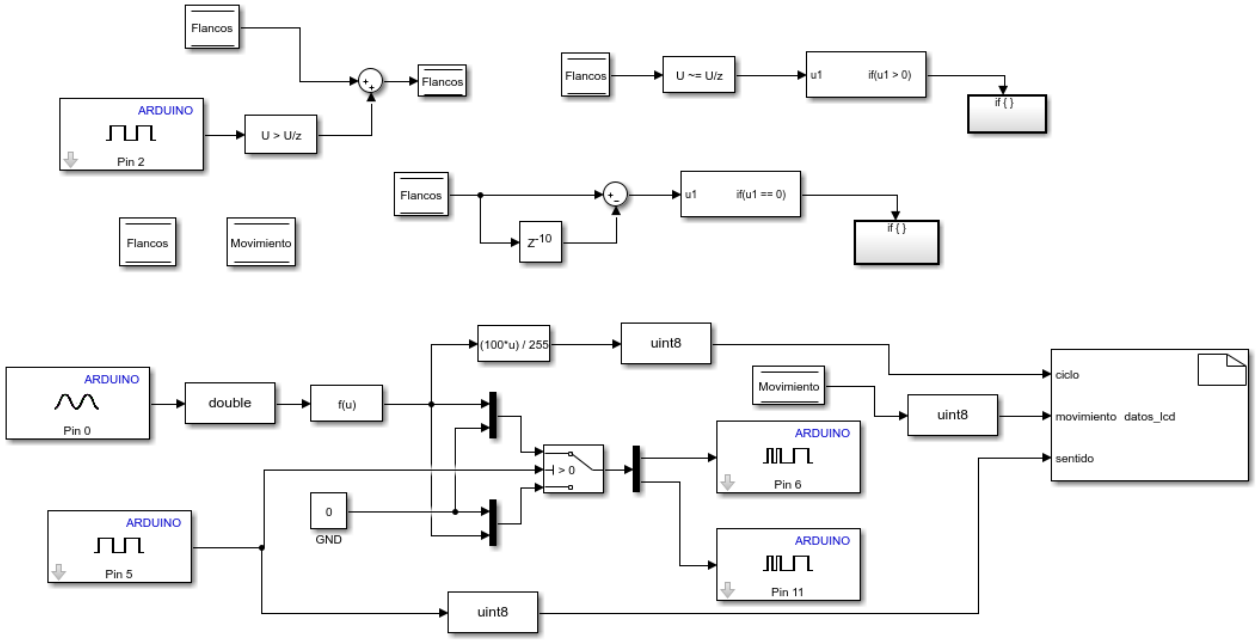


Figura 52. Manejo del motor y representación de datos en pantalla LCD

El bloque datos_lcd muestra el ciclo de trabajo de la señal de control y el sentido de giro del motor. Este último depende de las variables *movimiento* y *sentido*. Esta última entrada está conectada al interruptor de la protoboard, que controla el sentido de giro del motor.

Se mantiene la parte del manejo del motor del modelo de la figura 38.

Para determinar si el motor se encuentra parado o en movimiento empleamos un único canal del encoder, donde se registran flancos de subida y se almacenan en la variable *Flancos*. En base a esta variable:

- Si se detecta una variación en *Flancos*, es decir, se registran flancos de subida por parte del encoder, se interpreta que el motor gira y la variable *movimiento* se sitúa a nivel alto (en el bloque If Action Subsystem).
- Según la tabla , la velocidad mínima del motor es 50 rpm aproximadamente, lo que equivale al registro de 175 pulsos por segundo, es decir, se genera un pulso como mínimo cada 5 ms aproximadamente, o 5 periodos de muestreo. Con esta referencia, provocamos un retardo en la variable *Flancos* de 10 muestras (10 ms). Si la variable no ha cambiado en dicho intervalo de tiempo, es decir, no se registran flancos de subida, se interpreta que el motor se encientra detenido y la variable *movimiento* se sitúa a nivel bajo (en el bloque If Action Subsystem).

La única función del bloque datos_lcd es interactuar con la pantalla LCD 16x2. Diseñamos el siguiente script en el IDE de Arduino, importamos las librerías correspondientes y trasladamos el código a los apartados Update y Outputs siguiendo los pasos anteriores.

```
#include "Wire.h"
#include "LiquidCrystal_I2C.h"
int ciclo, movimiento, sentido;    //En Simulink estas variables se definen como entradas del S-Function Builder
LiquidCrystal_I2C lcd(0x27,16,2);
void setup () {
     lcd.begin(16,2);                  //Creamos el lcd y lo configuramos para que
     lcd.init();                       //empiece a funcionar
     lcd.backlight();
     lcd.setCursor(0,0);
     lcd.print ("Ciclo de trabajo");   //Muestra el nombre de la variable a evaluar
     delay (1000);
}

void loop () {
     lcd.setCursor(0,1);         //Muestra el ciclo de trabajo en %
     lcd.print(ciclo[0]);
     lcd.print(" % ");
     lcd.setCursor(14,1);
     if (movimiento[0] == 1)     //si el motor se encuentra en movimiento
     {
         if (sentido[0] == 1)    //si gira hacia la derecha
            lcd.print("->");
         if (sentido[0] == 0)    //si gira hacia la izquierda
            lcd.print("<-");
     }
     else
         lcd.print("X");
     delay(500);                 //retardo de medio segundo
}
```

Por último, construímos la S-Function y ejecutamos el modelo de forma embebida en la tarjeta.

A continuación se muestran unas capturas de la pantalla LCD durante el transcurso de la aplicación.

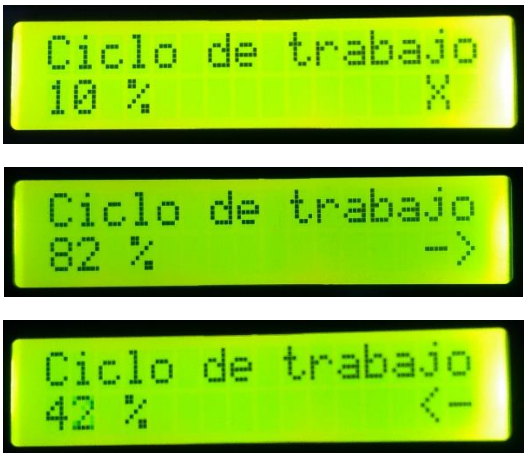


Figura 53. Representación de datos en pantalla LCD

En la primera captura, no se transmite un valor de tensión media suficiente con la señal PWM para arrancar el motor.

Estos resultados muestran la posibilidad de integrar librerías ya programadas para Arduino en los modelos de Simulink, lo que permite desarrollar una amplia variedad de aplicaciones con estas plataformas de hardware sin necesidad de diseñar estas librerías desde cero en el lenguaje de MATLAB.

4.2 Aplicación con servomotor SG90

La aplicación a desarrollar en el proyecto respecto al servomotor consiste en un manejo y control de posición. Para ello:

- El usuario introducirá una posición ángular deseada mediante un elemento de hardware, en este caso un potenciómetro. Esta posición será leída posteriormente por los entornos de programación.
- Una vez leída dicha posición, el servo se desplazará a la misma y después se compararán la posición introducida por el usuario y la posición que ocupe el motor tras el desplazamiento.
- Posteriormente, se implementará la misma aplicación mediante Bluetooth con las funciones de MATLAB/Simulink enfocadas en dicha comunicación.

4.2.1 Obtención de la posición angular del servomotor

El modelo de fábrica del servo solo dispone de 3 entradas: señal PWM, alimentación y masa, y no posee ninguna salida, por lo que sin ningún hardware adicional solo podemos comprobar las posiciones alcanzadas por el servo de manera visual.

El servo, a diferencia de los motores DC y paso a paso, dispone de un punto de referencia para una posición angular de 0º. Por ello, una opción válida para leer la posición alcanzada y cuantificarla mediante señales de tensión sería acoplar al servo un encoder absoluto.

Sin embargo, existe otra opción. En el apartado anterior se explicó que para mover un servo la tarjeta de control del mismo compara la tensión de la señal PWM de entrada con la tensión del potenciómetro interno del servo, que indica su posición debido a que dicho potenciómetro se mueve junto con el motor.

Por tanto, la solución consiste en soldar un cable adicional a la patilla central de dicho potenciómetro y conectarlo a un pin de Arduino que opere como entrada analógica. A esta tensión la denominaremos Vout.

Figura 54. Salida analógica del potenciómetro del servo

Esta opción es mucho más sencilla y precisa que acoplar un encoder, ya que no requiere de hardware adicional y la información es leída desde el propio motor.

El paquete de compatibilidad de MATLAB con Arduino dispone de dos funciones principales respecto a los servos:

>> servo(a, p): Crea un objeto de tipo servomotor en el espacio de trabajo de MATLAB. Para ello hay que indicar la tarjeta Arduino a la cual se conecta el servo (a) y el pin donde se genera la señal de control PWM (p).

>> writePosition(s,n): Escribe la posición n en el servomotor definido con el nombre s, donde n es un valor entre 0 y 1.

>> readPosition(s): Lee la posición del servomotor definido con el nombre s y devuelve un valor correspondiente a esta posición entre 0 y 1. El inconveniente que presenta esta función es que extrae dicho valor de posición de la propia señal PWM del motor, asumiendo de esta forma que la posición introducida y la posición real del motor siempre coinciden. Debido a esto, en nuestro proyecto leeremos la posición del motor a través del cable soldado al potenciómetro interno del servo.

Estas dos funciones operan en torno a un porcentaje en tanto por uno en relación a la máxima posición que puede alcanzar un servo conectado a Arduino. Es decir, en nuestro caso, n=1 equivale en teoría a 180º, mientras que si conectáramos otro servo con un rango de movimiento restringido entre 0º y 90º, ese mismo valor n=1 equivaldría a una posición de 90º.

Para trabajar directamente en grados con estas funciones usamos las siguientes expresiones:
>>writePosition(s,grados/180)
>>grados = writePosition(s) * 180

Para conocer la relación entre la posición del servo y Vout ejecutamos el siguiente script, donde el pin D9 genera la señal PWM del servo y Vout está conectada al pin A0. En este script el servo recorre todo su rango de posiciones y la tarjeta lee Vout cada vez que el servo avanza 1 grado.

```
clear all;

%Creación de hardware Arduino y servo
a=arduino('COM8','Uno');
s=servo(a,'D9');

%Ejes X e Y
grados = 0:1:180;
Vout = 0:1:180;

%Empleando la función writePosition(servo,A):  A = (1/180) * n°grados
for i=0:1:180
    writePosition(s,((1/180)*i));

    %Esperamos 0,1 segundos para asegurarnos de que el servo se encuentra
    %en la nueva posición
    pause(0.1);
    Vout(i+1) = readVoltage(a,'A0');
    pause(0.2);
end

%Representamos la evolución de la tensión asociada a la posición del servo
plot(grados,tension);
```

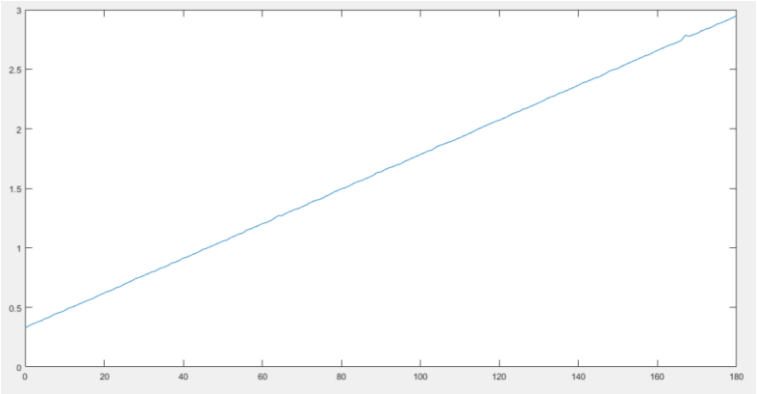


Figura 55.Gráfica de Vout respecto a la posición angular del servo

Vout vale 0,3421 V cuando el servo se sitúa a 0º, y 2,957 V cuando se sitúa a 180º. La gráfica sigue una progresión lineal, por lo que se obtiene la siguiente resolución en la medida.

$$Resolución = \frac{\Delta V}{\Delta º} = \frac{2{,}957V - 0{,}3421V}{180º - 0º} \cong 14{,}5\frac{mV}{grado} \quad [4]$$

Entonces:

$$Vout = 0{,}3421 + 0{,}0145 * grados \quad [5]$$

Despejando la ecuación obtenemos la posición del servo indicada por su potenciómetro interno.

$$grados = \frac{Vout - 0{,}3421}{0{,}0145} \quad [6]$$

Una vez conocida la evolución de Vout respecto a la posición angular podemos programar la aplicación del servo.

4.2.2 Programación del movimiento del servo mediante MATLAB/Simulink

Para indicar la posición que queremos alcanzar con el motor empleamos un potenciómetro de 10 kΩ conectado a otra entrada analógica de Arduino. El valor de esta entrada estará comprendida entre 0 y 5V, que corresponderá a 0º y 180º respectivamente.

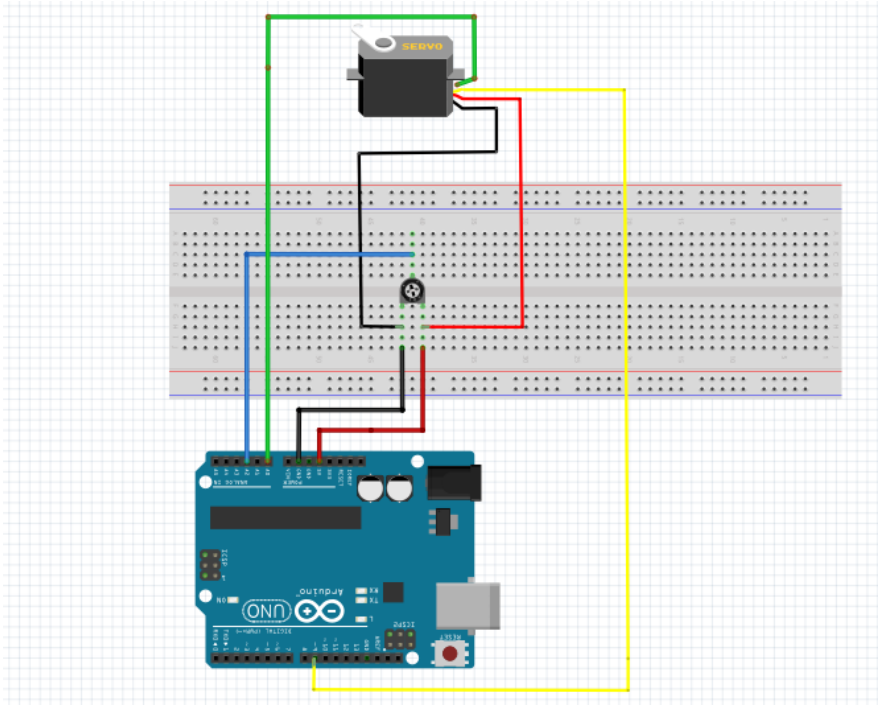


Figura 56. Esquema de control manual de posición del servo

En la aplicación desarrollada, se tomarán 250 lecturas de posición de Vout y del potenciómetro de la protoboard, convirtiéndolas de tensión a grados.

Los valores leídos se actualizarán cada 100 ms, y se representará la segunda variable (las posiciones introducidas por el usuario) en tiempo real con una gráfica mientras el servo gira. Después, se mostrará la gráfica de las muestras tomadas del servo para comparar ambos arrays de posiciones.

```
%Este script controla la posición de un servo en base a un potenciómetro
clear all;

%Creamos los objetos hardware arduino y servo
a=arduino('COM5','Uno','Libraries','Servo');
s=servo(a,'D9');

%Controlamos la posición del servo con un potenciómetro conectado a A2
potenciometro = 'A2';
valor_potenciometro = 0;

muestras = 1;     %Eje X
posiciones = 1;  %Vector que almacena las posiciones mandadas por el usuario
lecturas = 1:250;  %Vector que almacena las posiciones leídas por el pot.
del servo
```

```
for i=1:250
    %La tensión del potenciómetro de la protoboard se sitúa entre 0 y 5V
para indicar entre
    %0° y 180°
    muestras(i)=i;
    valor_potenciometro = readVoltage(a,potenciometro);
    valor_potenciometro = valor_potenciometro/5;
    writePosition(s,valor_potenciometro);

    %Tras escribir, pasamos a grados para facilitar la lectura de la
    %gráfica, redondeando el valor debido a la resolución de 1° del servo
    posiciones(i) = round(valor_potenciometro * 180);
    %Esperamos 25 ms para considerar realizado el cambio de posición
    pause(0.075);
    %Leemos la posición actual del servo
    lecturas(i) = readVoltage(a,'A0');

    %Actualizamos la gráfica de posiciones, ajustando los ejes
    plot(muestras,posiciones,'r')
    axis([0 250 0 180]);
    grid on
    pause(0.025); %Bucle cada 50 ms
end

%Conservamos la gráfica de posiciones
hold on

%Ahora pasamos el array de lecturas de tensiones a grados.
%El pot. del servo lee 0.3421V a 0° y 2.957V a 180° con progresión
lineal.
%La resolución es de 14,5 mV por grado.
lecturas = (lecturas - 0.3421) / 0.0145;
lecturas = round(lecturas);
pause(5);
plot(muestras,lecturas,'b');
```

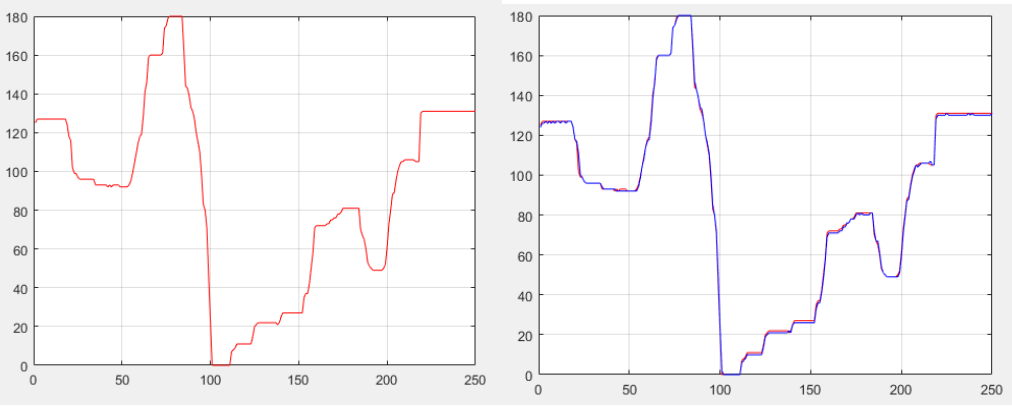


Figura 57. Gráficas de posiciones introducidas (rojo) y de lecturas (azul)

Superponiendo ambas gráficas observamos que son casi idénticas.

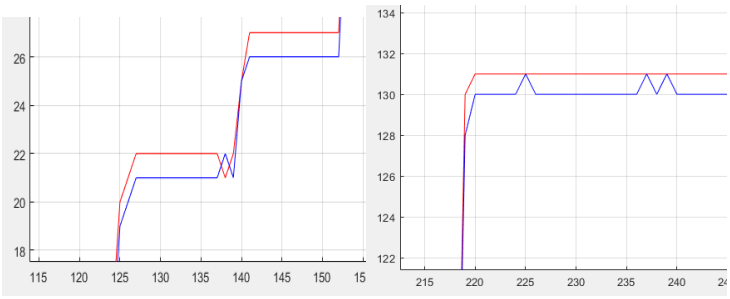


Figura 58. Errores de posición del SG90 en MATLAB

En algunos puntos de la gráfica se muestran pequeñas desviaciones de 1 grado. Esto puede ser debido a la aproximación empleada para determinar la resolución o a un error por parte de la tarjeta de control interna del servo.

En base a los resultados, podemos verificar un control preciso del servomotor empleando las funciones de MATLAB.

Por su parte, el paquete de compatibilidad de Simulink con Arduino ofrece tres bloques relacionados con servomotores:

Figura 59. Bloques de control de servomotores

Standard Servo Read: Lee la posición angular del servo en grados. Este bloque, al igual que la función **>>readPosition** de MATLAB, lee dicha posición en base a la señal de control PWM, por lo que puede no corresponder con la posición angular real.

Standard Servo Write: Mueve el motor según la posición introducida en grados (0 a 180).

Continuous Servo Write: Configura la velocidad de giro de un servomotor de rotación continua variando la señal PWM según el valor de entrada (-90 implica un periodo a nivel alto de 1 ms y una máxima velocidad en sentido antihorario y 90 implica un periodo a nivel alto de 2 ms y una máxima velocidad en sentido horario)

Es necesario emplear estos bloques para mover el servo en vez de generar una señal PWM debido a las limitaciones técnicas de Simulink.

Como se indicó anteriormente, Simulink emplea un periodo de muestreo determinado en sus modelos discretos para analizar las señales de entrada y para generar las señales de salida de los mismos, y el modelo funciona como una interrupción periódica.

El inconveniente de esta situación es que existe un rango de posiciones demasiado amplio (181 posiciones posibles, de 0º a 180º) para un intervalo del tiempo a nivel alto de la señal PWM que controla el servo muy pequeño (1ms).

Esta información se extrae del datasheet del servomotor SG90.

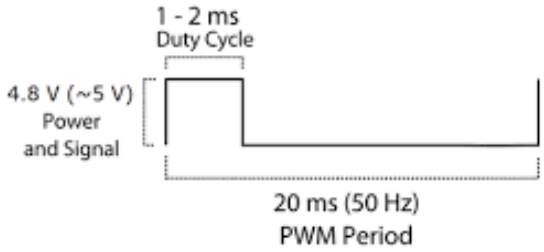


Figura 60. Señal de control en un SG90

$$TH = 1 + \frac{grados}{180} \ [ms] \qquad [7]$$

Por tanto, el incremento de TH es de (1/180) ms, o 5,5 µs por grado recorrido. Esto implica que para gestionar correctamente el control del servo con una señal PWM necesitamos emplear en el modelo un periodo de muestreo del orden de 5 µs, lo que equivale a una frecuencia de operación de 200 kHz.

Los modelos de Simulink presentan errores de procesamiento cuando trabajan a frecuencias tan elevadas, de forma similar a lo ocurrido en la aplicación del motor DC debido a la alta resolución del encoder incremental.

Empleando los bloques de la figura 59 definimos el siguiente modelo, con un periodo de muestreo Ts = 10 ms.

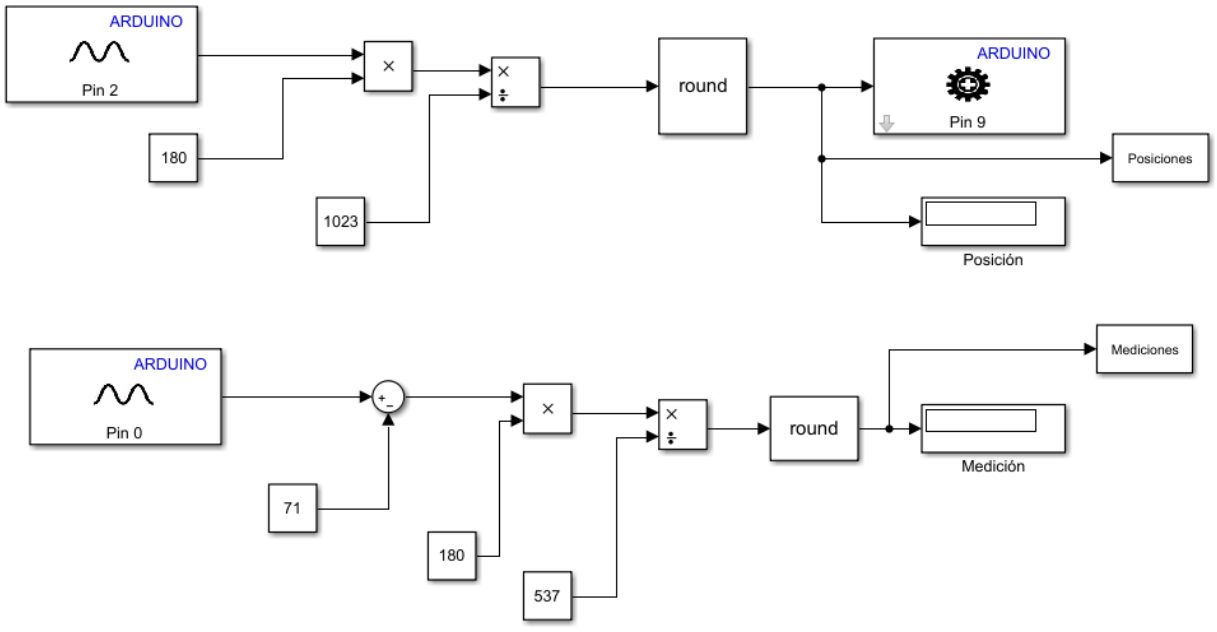


Figura 61. Modelo de control de posición del SG90 con potenciómetro

El único cambio respecto al script anterior es la adquisición de tensiones del potenciómetro de la protoboard y de Vout (conectados a A2 y A0 respectivamente). La lectura de los pines Analog Input oscila entre 0 y 1023, correspondiendo con un intervalo de tensión de entre 0 y 5V en el caso de la Arduino Uno.

Para Vout, se lee un valor de 71 (0,347 V) cuando el servo se sitúa a 0º y 608 (2,97 V) cuando se sitúa a 180º. De forma similar a la ecuación nº 6:

$$grados = (Vout - 71) * \frac{180}{608 - 71} \qquad [8]$$

A continuación ejecutamos el modelo en modo External para monitorizar las variables en los bloques Display y para almacenarlos como arrays en MATLAB.

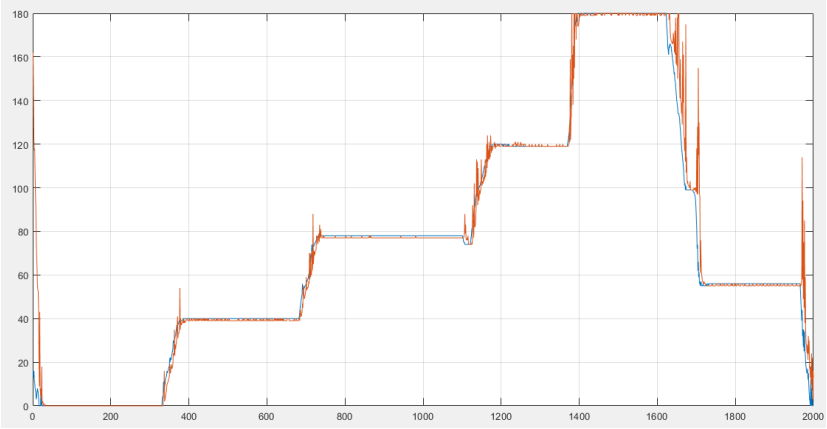


Figura 62. Gráfica de posiciones introducidas (azul) y de lecturas (naranja) en Simulink

Analizando las gráficas compuestas por 2000 muestras tomadas observamos que la señal de mediciones está retardada con respecto a la de posiciones introducidas por el usuario.

Esto es debido a que el servo requiere de un intervalo de tiempo correspondiente a varios periodos de muestreo para cambiar de posición, y el modelo registra a la vez la posición introducida por el usuario y la posición actual del servo en cada muestreo.

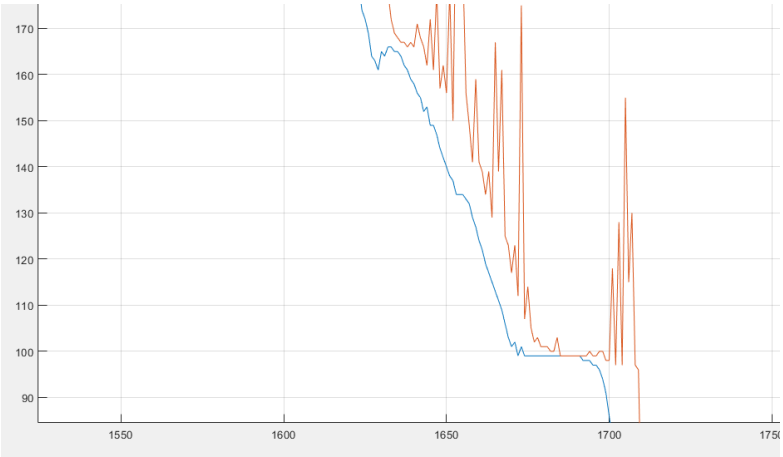


Figura 63. Retardo entre gráficas de posiciones introducidas y lecturas

En los intervalos en los cuales el servo se encuentra detenido, en régimen estacionario, las mediciones coinciden con las posiciones indicadas, con un error de resolución de 1º, al igual que en el caso del script de MATLAB.

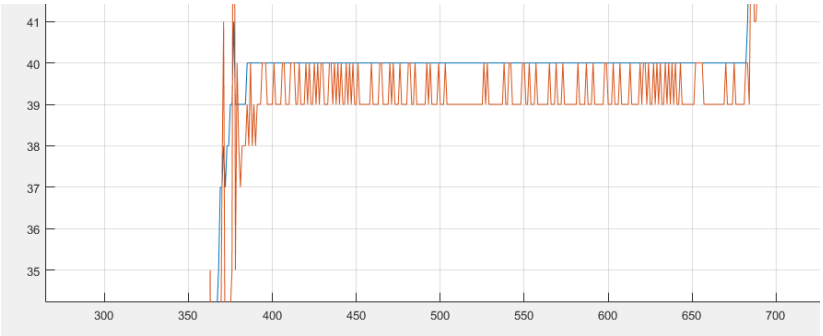


Figura 64. Error de posición del SG90 en Simulink

Según los resultados, la medida de la posición se realiza correctamente. Una vez comprobada la validez del modelo, podemos cambiar a modo de simulación normal y volcar el modelo a la tarjeta.

De esta forma podremos controlar el servo con el potenciómetro sin necesidad de la mediación del entorno de Simulink y almacenar las mediciones, pero no podremos visualizar la posición del servo con los bloques Display.

Los resultados demuestran que tanto MATLAB como Simulink permiten manejar servomotores de forma precisa e inmediata debido a las funciones y bloques que aportan ambos paquetes de compatibilidad. Cada entorno posee una ventaja distinta.

- En Simulink pueden moverse los servomotores con código embebido en la tarjeta Arduino, requiriendo del PC únicamente la alimentación vía USB en ausencia de una fuente de alimentación externa y el volcado de los modelos.
- En MATLAB la posición angular puede medirse con mayor facilidad, gracias al orden secuencial en el que se ejecutan las sentencias del script y al uso de retardos que en Simulink son más complicado de programar.

4.2.3 Control de servomotores por Bluetooth

En este apartado sustituiremos el control del motor con un potenciómetro por comandos transmitidos y recibidos a través de un terminal Bluetooth instalado en un teléfono móvil.

Estos comandos serán gestionados en un modelo de Simulink para desplazar el motor y para representar periódicamente las mediciones de posición en la pantalla del terminal del móvil.

Para realizar la comunicación entre MATLAB y el terminal durante la ejecución de la aplicación emplearemos un módulo Bluetooth HC-05, de 6 pines.

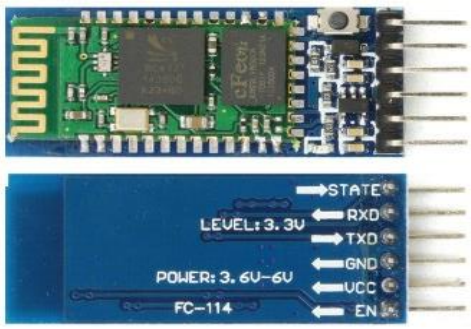


Figura 65. Módulo Bluetooth HC-05

Este módulo se identifica mediante una dirección de 48 bits, y puede operar en dos roles: Master o Slave.

Cuando opera en modo Slave al conectarse a otro dispositivo Master, como un PC o un móvil, únicamente puede recibir y transmitir datos a través de ese dispositivo al que se encuentra subordinado.

Operando en modo Master uno de estos módulos es capaz de conectarse a otros configurados en modo Slave e intercambiar información entre ellos, actuando como un puente.

En nuestro proyecto utilizaremos el módulo en modo Slave conectado al microcontrolador de Arduino.

La función que desempeñan los terminales son las siguientes:

- RXD y TXD: Recibir y transmitir datos, respectivamente, cuando el módulo se encuentra conectado a un dispositivo. Cabe destacar que estos pines funcionan con una lógica de 3,3V, a diferencia de la mayoría de plataformas Arduino, con 5V.
- VCC y GND: Pines de alimentación, entre 3,6V y 6V.
- STATE: Se sitúa a nivel alto cuando el módulo se encuentra conectado y a nivel bajo cuando no hay conexión con ningún dispositivo. Esto también viene indicado con un LED integrado, que parpadea constantemente cuando no hay conexión y dos veces cada 2 segundos cuando sí la hay.
- EN o KEY: Es un pin que permite establecer un modo de configuración de las características de la comunicación con el módulo mediante comandos denominados AT empleando un terminal serie. Para activar este modo, hay que situar este pin a nivel alto cuando se alimente el módulo o pulsar el botón de la esquina superior derecha.

En primer lugar, debemos conocer y configurar las características del módulo (rol de Slave, velocidad de transmisión, etc), empleando los comandos AT. Para ello, conectaremos el HC-05 al PC mediante un puerto COM empleando el driver CP2102, un adaptador de USB a serie RS232.

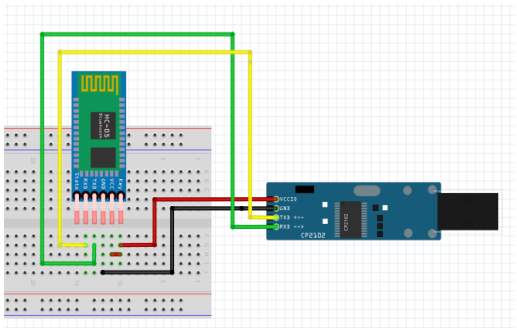


Figura 66. Esquema de conexión del HC-05 a un puerto COM

Para transmitir los comandos AT necesitamos un programa que actúe como un terminal serie virtual. En nuestro prooyecto empleamos el software Termite. Tras conectar el driver al PC, configuramos la comunicación del nuevo puerto COM creado (COM7).

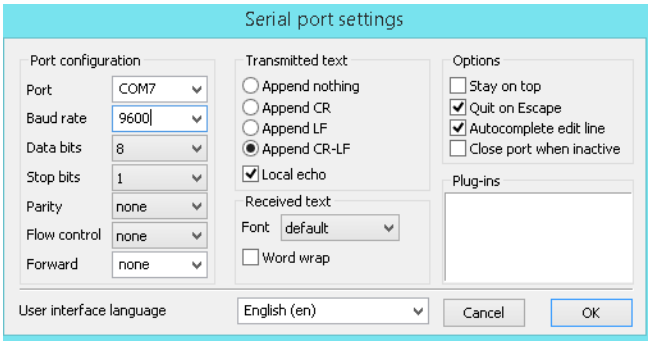


Figura 67. Configuración del puerto COM7 en Termite.exe

Una vez creada la conexión transmitimos los siguientes comandos para comprobar los parámetros del módulo y para aumentar la velocidad de transmisión.

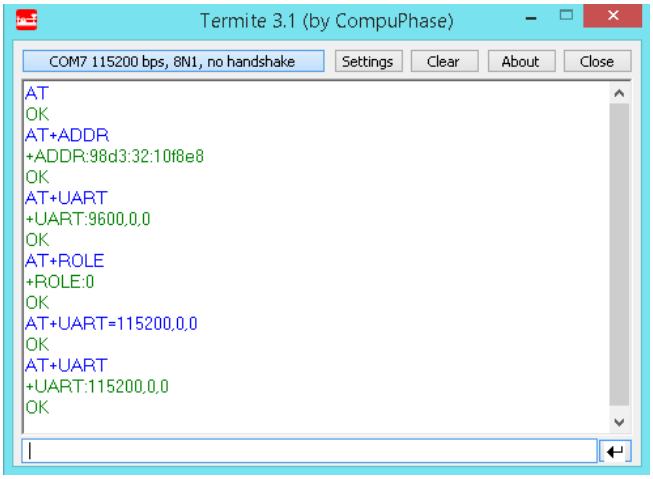


Figura 68. Configuración del HC-05 mediante comandos AT

El terminal obtiene por respuesta a los comandos la dirección del módulo “98d33210f8e8”, y que opera a una velocidad de transmisión de 9600 baudios ejerciendo el rol de esclavo (ROLE:0).

Aumentamos la velocidad de transmisión a 115200 baudios para que la comunicación sea más veloz.

Una vez configurado el módulo HC-05 comprobamos la correcta transmisión y recepción de datos entre el terminal Bluetooth y Simulink, empleando los siguientes bloques del paquete de compatibilidad con Arduino.

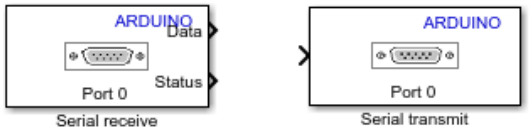


Figura 69. Bloques de comunicación Bluetooth con hardware Arduino

Serial Receive: Recibe un array de datos transmitidos por Bluetooth a través de un puerto UART de la tarjeta Arduino. Dentro del bloque se configura el periodo de actualización de los datos, la longitud del array y de qué tipo se exportan los datos al modelo por la salida Data. El valor Data puedes ser de tipo double, single, int8, uint8, int16, uint16, int32, uint32 o boolean

Pueden transmitirse comandos en hexadecimal o en ASCII.

Cuando se reciben datos, la salida Status vale 1, y 0 si no se recibe nada. Por último, si no se transmiten datos, la salida Data vale también 0.

Serial Transmit: Transmite un dato numérico al terminal Bluetooth acompañado de un mensaje de texto opcional.

Para comprobar la correcta recepción y transmisión de datos, programamos un modelo de prueba.

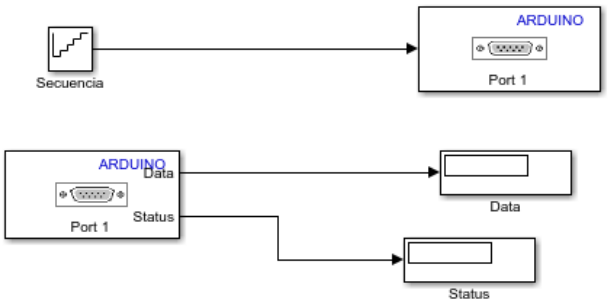


Figura 70. Modelo de recepción/transmisión de datos por Bluetooth

Por un lado, se transmitirá la secuencia de números [1 2 3 4 5] cada 5 segundos, y por otro, se recibirá una cadena de texto en ASCII con salto de línea (\n): “ABCDE”, introducida en el terminal móvil.

En este punto existe un inconveniente: la Arduino Uno solo dispone de un puerto UART (Port 0), que se encuentra ocupado al estar conectada la tarjeta con el PC para su programación, y dado que necesitamos que el puerto 0 se encuentre libre para ejecutar un modelo de Simulink (de forma embebida o en modo External), necesitamos otro hardware que añada al menos otra UART adicional.

La tarjeta Arduino Due dispone de cuatro puertos UART (Port 0 para comunicarse con MATLAB/Simulink y otros tres adicionales), así que la emplearemos para este apartado, empleando la UART1 de la tarjeta (TXD1 y RXD1), ajustando la velocidad de transmisión a 115200 baudios.

Una vez conectado el HC-05 a la Arduino Due, conectamos el módulo al terminal móvil y ejecutamos el modelo en modo External.

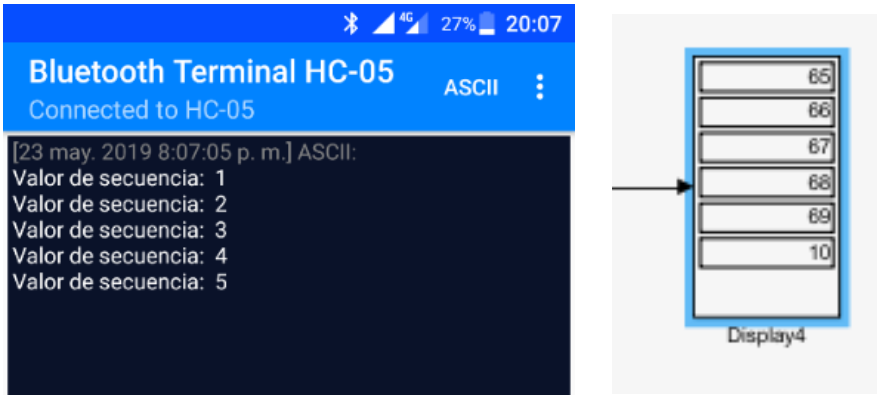


Figura 71. Transmisión de secuencia y recepción de cadena de texto

La secuencia se transmite correctamente y el modelo recibe los caracteres ASCII [65 66 67 68 69 10], que corresponden a las letras A, B, C, D, E y al salto de línea respectivamente.

Una vez comprobado que la comunicación se realiza correctamente integramos los bloques serie en el modelo de la figura 61.

Hay que tener en cuenta que Data vale 0 si no se trasmite ningún número, por lo que solo debe moverse el servo cuando se reciban datos (Status = 1).

Con esto evitamos que el servo se desplace a su posición de origen cuando no se estén transmitiendo comandos.

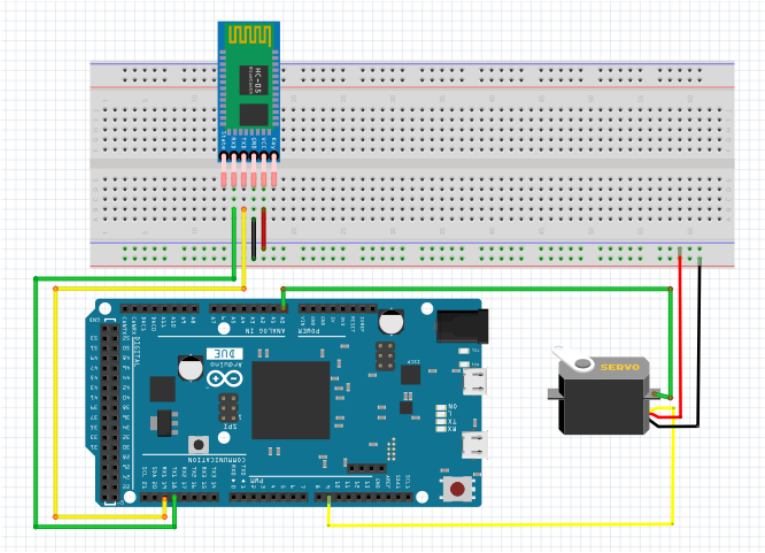


Figura 72. Esquema control de posición de servo por Bluetooth

El siguiente modelo opera con un periodo de muestreo Ts = 5ms.

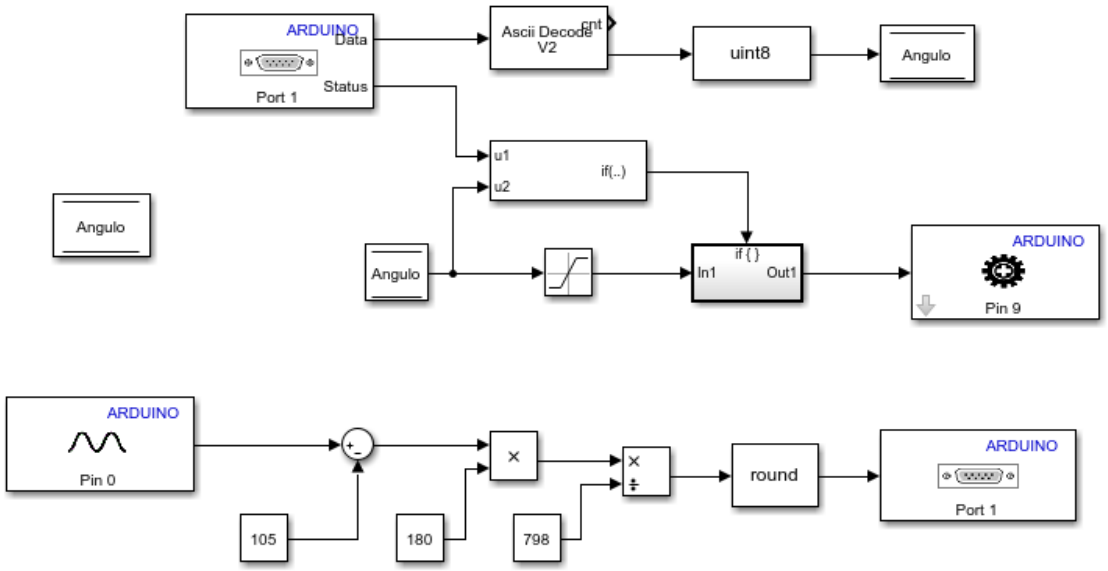


Figura 73. Modelo de control de posición del SG90 por Bluetooth

El modelo realiza las siguientes acciones:

- Se recibe el ángulo introducido en ASCII por el terminal móvil. En el bloque Serial receive definimos la longitud del array en 4 valores (el ángulo máximo es 180º, más el caracter de final de línea). Debido a que el tamaño de los arrays de datos recibidos es fijo, debemos enviar únicamente números de tres cifras (000 a 180) para evitar errores.
  Después convertimos este array de caracteres ASCII a un número decimal con el bloque Ascii Decode V2 ('%d\n'), que se almacena en la variable *Angulo*. Si se reciben datos y el ángulo introducido está comprendido entre 0º y 180º, el servo se mueve a dicha posición.
- Cada segundo se lee Vout, se convierte a grados y la posición del servo se muestra en la pantalla del terminal móvil.

Al emplear la Arduino Due, cambia el intervalo de tensión empleado en los bloques de entradas analógicas (de 0 a 3,3V). Para Vout, se lee un valor de 105 (0,347 V) cuando el servo se sitúa a 0º y 903 (2,97 V) cuando se sitúa a 180º. Entonces:

$$Ángulo\ medido = (Vout - 105) * \frac{180}{903 - 105} \qquad [9]$$

Como no utilizamos bloques display o scope para ver los resultados en el modelo, podemos volcar el código en la tarjeta.

A continuación se muestra una lista de posiciones marcadas en el terminal y de los valores correspondientes a las lecturas de Vout.

Consigna	Posición (Vout)
000	1º
040	40º
095	95º
130	131º
180	181º

Tabla 12. Posiciones alcanzadas por el SG90 ante comandos Bluetooth.

El movimiento del servo se produce de forma inmediata y Vout presenta el mismo margen de error de 1º que en casos anteriores.

De esta forma podemos controlar servomotores y medir sus posiciones de forma totalmente inalámbrica, sin necesidad de la mediación del entorno de MATLAB/Simulink que se requería en el modelo de la figura 61.

Por último, es posible manejar varios servos a la vez con una misma UART. Para ello hay que aumentar el tamaño del array de datos recibidos e indicar en el bloque Ascii Decode v2 que dicho array está compuesto por varios números de tres cifras, cada uno correspondiente a la posición de un servo distinto.

Un ejemplo con tres servos puede ser:

Data length	Ascii Decode v2	Comando introducido	Salidas de Ascii Decode
12	'%3d_%3d_%3d\n'	000 090 164	3 (0, 90 y 164)

4.3 Aplicación con motor paso a paso 28BYJ-48

La aplicación a desarrollar en el proyecto respecto a los motores paso a paso es un sencillo escáner formado por un medidor de distancia por infrarrojos modelo GP2Y0A02YK0F acoplado al motor, de modo que pueda girar en 360º y detectar objetos cercanos situados a un cierto umbral de distancia.

Este medidor proporciona una tensión en función de la distancia a un objeto detectado, y su funcionamiento se detallará posteriormente en el apartado 4.3.2.

A medida que se vayan tomando las muestras de distancias, el motor paso a paso girará de forma continua en sentido horario o antihorario, en intervalos de 10º, deteniéndose 0,5 segundos en cada uno. Si detecta un objeto situado a una distancia inferior al umbral fijado, se indicará iluminando un led.

4.3.1 Programación del giro del motor en MATLAB/Simulink

El primer paso es conseguir un movimiento continuo rápido y preciso por parte del motor paso a paso. Para ello conectamos las entradas del driver a los pines de Arduino.

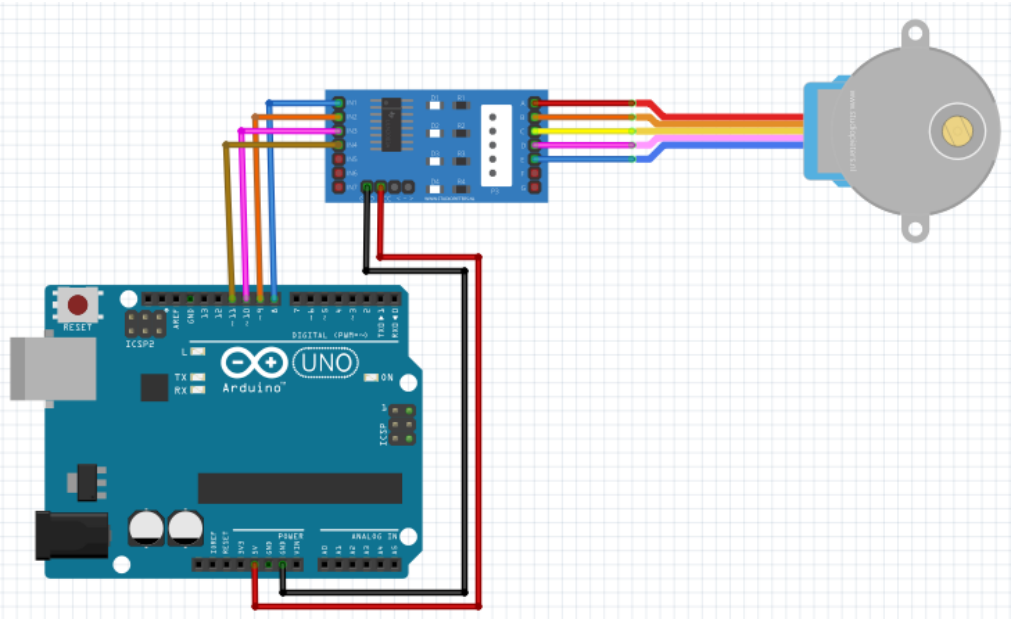


Figura 74. Conexión entre Arduino y motor 28BYJ-48

Las entradas del driver se conectan a cuatro pines digitales (D8, D9, D10 y D11 en nuestro caso).

La secuencia de excitación elegida es la del modo de medio paso, detallada en la tabla 6, debido a que es la que ofrece la mayor resolución que puede ofrecer este motor, de 0,088º por paso recorrido.

Aplicando la secuencia decimal [8 12 4 6 2 3 1 9] a las bobinas de forma periódica se consigue un movimiento ininterrumpido en sentido horario. Para girar en sentido antihorario, empleamos la misma secuencia a la inversa.

Programamos un script en MATLAB para comprobar que el motor gira correctamente.

```
clear all;

%Creación de objeto hardware Arduino
a=arduino('COM5','Uno');

%Configuración de pines como salidas digitales
configurePin(a,'D8','DigitalOutput');
configurePin(a,'D9','DigitalOutput');
configurePin(a,'D10','DigitalOutput');
configurePin(a,'D11','DigitalOutput');

IN1 = 'D8';
IN2 = 'D9';
IN3 = 'D10';
IN4 = 'D11';

%Secuencia de medio paso. 4096 pasos por revolución. (0,088 grados por paso)
secuencia = [1 0 0 0; 1 1 0 0; 0 1 0 0; 0 1 1 0; 0 0 1 0; 0 0 1 1; 0 0 0 1;
1 0 0 1];

while 1
    for i=1:8
    writeDigitalPin(a,IN1,secuencia(i,1));
    writeDigitalPin(a,IN2,secuencia(i,2));
    writeDigitalPin(a,IN3,secuencia(i,3));
    writeDigitalPin(a,IN4,secuencia(i,4));
    pause(0.002)
    end
end
```

En este script se escribe continuamente la secuencia de movimiento en sentido horario en las entradas del driver, con una pausa de 2 ms por paso recorrido, lo que equivale a una frecuencia de operación por parte del motor de 500 Hz. La frecuencia máxima de operación con el motor 28BYJ-48 es de 1 kHz.

Es decir, el motor recorre en teoría 500 pasos por segundo. Debido a la resolución de 4096 pasos por revolución que existe en el modo de operación de medio paso, el tiempo previsto para que el motor recorra una vuelta completa es de:

$$T = \frac{4096\ pasos}{1\ rev} * \frac{1\ sg}{500\ pasos} \cong 8{,}2\ sg \qquad [10]$$

No obstante, se encontraron varios problemas fundamentales a la hora de comprobar el resultado del script:

- El paquete de compatibilidad solo permite escribir en un único pin cada vez, en contraposición a otros entornos de programación, que permiten escribir en todo un grupo de pines a la vez. Esta limitación provoca que necesitemos usar cuatro sentencias de escritura para el desplazamiento de cada paso.

- A la pausa de 2 ms se añade el tiempo de ejecución de estas sentencias, lo cual disminuye la frecuencia de operación (en un margen teórico de unos pocos Hz).
- A pesar de este ligero error, el motor gira mucho más lentamente de lo previsto, tardando en dar una vuelta un tiempo aproximado de 40 segundos, lo cual invalida la aplicación. Esta gran diferencia se debe a la comunicación en serie constante que se requiere con la tarjeta Arduino, ya que la computadora de MATLAB transmite cuatro sentencias de escritura por cada paso a recorrer a través del pin TX.

Este movimiento tan lento por parte del motor es debido al elevado número de sentencias que se necesitan transmitir en serie para mover este motor (4096 pasos * 4 sentencias de escritura por paso equivalen a 16384 sentencias para dar una vuelta completa en medio paso).

En las formas usuales de programación con Arduino, como con su IDE, el código se ejecuta principalmente de forma embebida en la tarjeta, y la comunicación serie queda relegada a situaciones puntuales en los que se necesite de una comunicación directa con el PC, como por ejemplo, la orden de ejecutar una determinada acción por parte del usuario.

Como alternativa existen dos posibles soluciones para utilizar un script de MATLAB: emplear las funciones propias con el Adafruit Motor Shield v2 para controlar el motor paso a paso con unas determinadas funciones o acoplar un Shield Ethernet a la Arduino UNO para aumentar notablemente la frecuencia de transmisión.

Sin embargo, gracias al entorno de Simulink podemos ejecutar código de manera embebida en la tarjeta, resolviendo el problema anterior sin necesidad de emplear ningún hardware adicional.

Por tanto, usaremos esta herramienta para desarrollar la aplicación con el motor paso a paso.

El siguiente modelo permite que el motor gire en ambos sentidos, y el periodo de muestreo es de 1 ms, lo que equivale a una frecuencia de operación de 1 kHz por parte del motor.

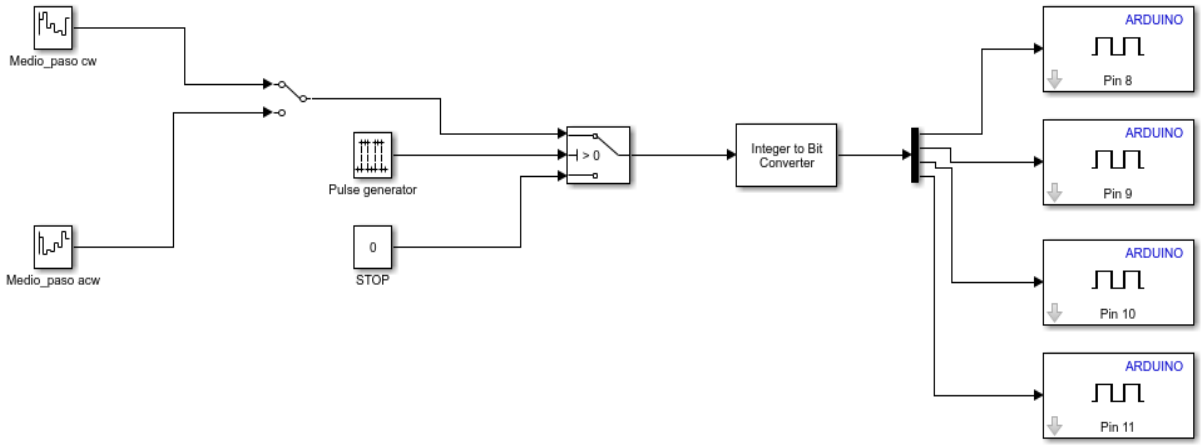


Figura 75. Modelo de giro de un motor paso a paso en Simulink

Definimos las variables del modelo, junto con un umbral de detección de 50 centímetros.

Figura 76. Variables del modelo del escáner

En cada instante de muestreo se escribe un valor de las secuencias decimales, [8 12 4 6 2 3 1 9] en sentido horario y [9 1 3 2 6 4 12 8] en sentido antihorario, en el puerto de cuatro pines de acuerdo a la tabla 6.

El sentido de giro se controla mediante un interruptor manual del modelo.

El movimiento (recorrer 10º + parada de 0,5 segundos) es periódico, por lo que para controlar los tiempos de movimiento y parada del motor empleamos una señal cuadrada basada en muestras que opera en función a las variables *Grados* y *Pausa*.

-Instantes de muestreo a nivel alto

El motor gira un paso, igual a 0,088º, en cada instante de muestro. Entonces:

$$TH = \frac{Grados}{0{,}088º/muestra} = \frac{10º}{0{,}088º/muestra} \cong 114\ muestras \qquad [11]$$

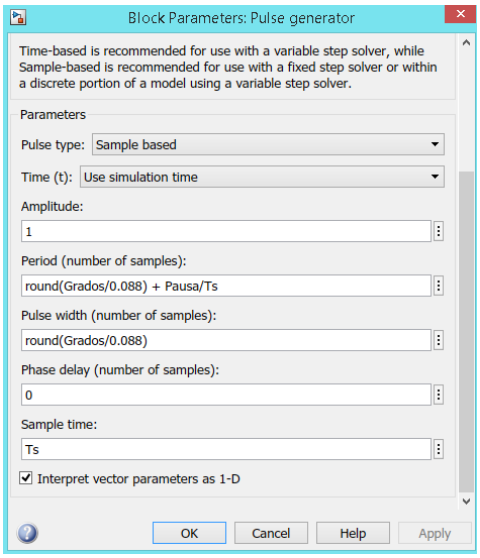
Redondeamos el resultado para obtener un número entero de muestras.

-Instantes de muestreo a nivel bajo

Dividimos la variable Pausa entre el periodo de muestreo.

$$TL = \frac{Pausa}{Ts} \cong \frac{0{,}5\ s}{0{,}001\ s} = 500\ muestras \qquad [12]$$

Con las variables de la figura anterior la onda de control resultante es la siguiente:



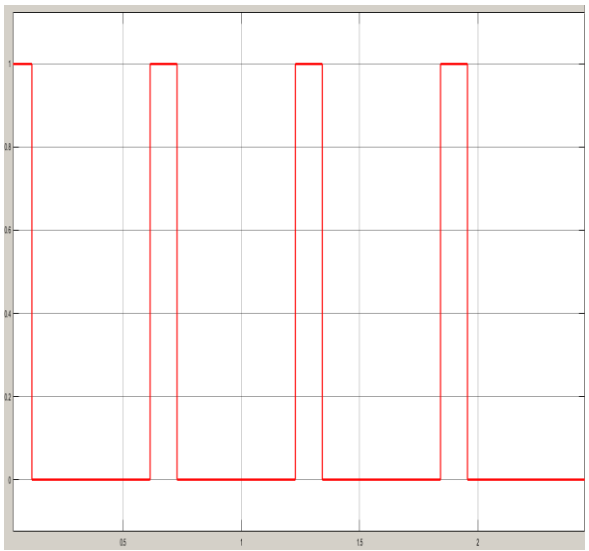


Figura 77. Señal de control del movimiento del motor paso a paso

A continuación ejecutamos el modelo en modo External. En esta ocasión los resultados han sido satisfactorios, obteniéndose una gran velocidad y precisión por parte del movimiento del motor, y una respuesta inmediata al cambio en su sentido de giro.

Además, operando en modo External podemos modificar las variables del modelo, como la frecuencia del motor o el umbral de posición, en tiempo real y sin necesidad de detener la aplicación.

4.3.2 Medidor de distancia por IR

En nuestro escáner emplearemos un sensor de distancia SHARP, sensores ópticos capaces de medir la distancia entre ellos y un determinado objeto, y transmitirla como un valor de tensión.

Para ello emplean un emisor infrarrojo y un receptor basado en un PSD (position sensitive device o dispositivo sensible a la posición). Este sensor calcula la distancia a un objeto en base al ángulo que forman el emisor, el objeto y el receptor, que detecta el ángulo de incidencia del rayo infrarrojo tras impactar con el objeto. Esta técnica se conoce como triangulación.

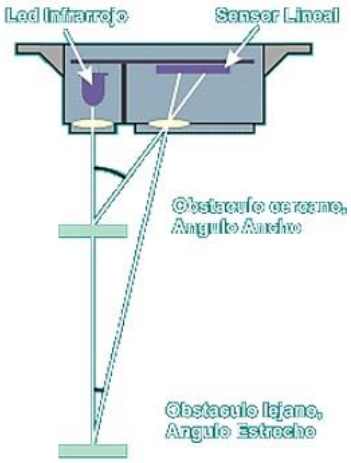


Figura 78. Funcionamiento de sensor SHARP

Las principales ventajas de estos sensores son su reducido precio y el hecho de sus medidas no se ven afectadas bajo una fuerte iluminación ambiental. El receptor detecta únicamente la luz infrarroja provocada por el emisor.

El sensor elegido es el modelo GP2Y0A02YK0F, el cual posee un rango de detección de entre 15 cm y 150 cm. La tensión de salida se actualiza cada 38 ms.

Figura 79. Sensor GP2Y0A02YK0F

Del datasheet del dispositivo, incluido en esta memoria en el apartado de anexos, analizamos la gráfica que relaciona la distancia medida con la tensión de salida del sensor.

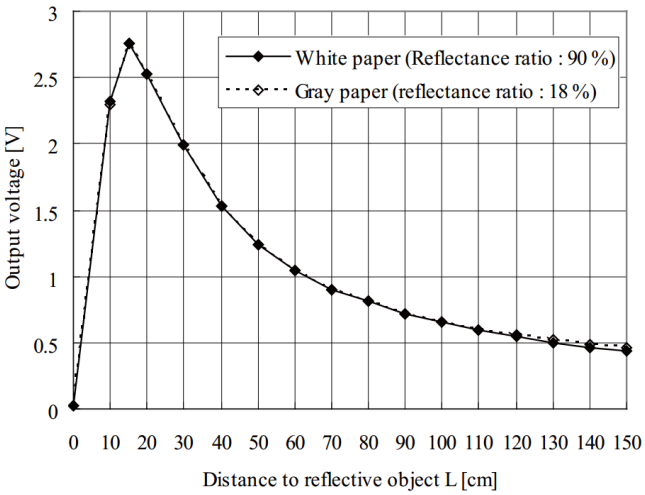


Figura 80. Voltaje de salida del sensor con respecto a la distancia del objeto detectado

Debido a que el sensor presenta el mismo valor de tensión para distancias menores o mayores a 15 cm, la gráfica es de tipo no lineal. Por ello no podemos relacionar directamente la distancia detectada con la tensión de salida.

Sin embargo, limitando el rango de distancias a partir de 15 cm podemos observar que la evolución de la gráfica es potencial y podemos aproximar la distancia de los objetos detectados en base a la tensión de salida del sensor.

$$L = a * V^b$$

Tomamos dos puntos de la gráfica.

$$L1 = 30\ cm => V1 = 2V \quad y \quad L2 = 130\ cm => V2 = 0{,}5V$$

Resolviendo el sistema de dos ecuaciones obtenemos el siguiente resultado.

$$L = 62{,}45 * V^{-1{,}0577} \qquad [13]$$

Por último, conectamos un condensador de 10 µF entre la alimentación y masa para reducir el ruido de las medidas, como indica la hoja de datos del fabricante.

Advice for the power supply

- In order to stabilize power supply line, we recommend to insert a by-pass capacitor of 10µF or more between Vcc and GND near this product.

Figura 81. Consejo para estabilizar la tensión de alimentación

4.3.3 Comprobación experimental del escáner

Una vez linealizada la gráfica del medidor de distancias podemos acoplarlo al eje del motor.

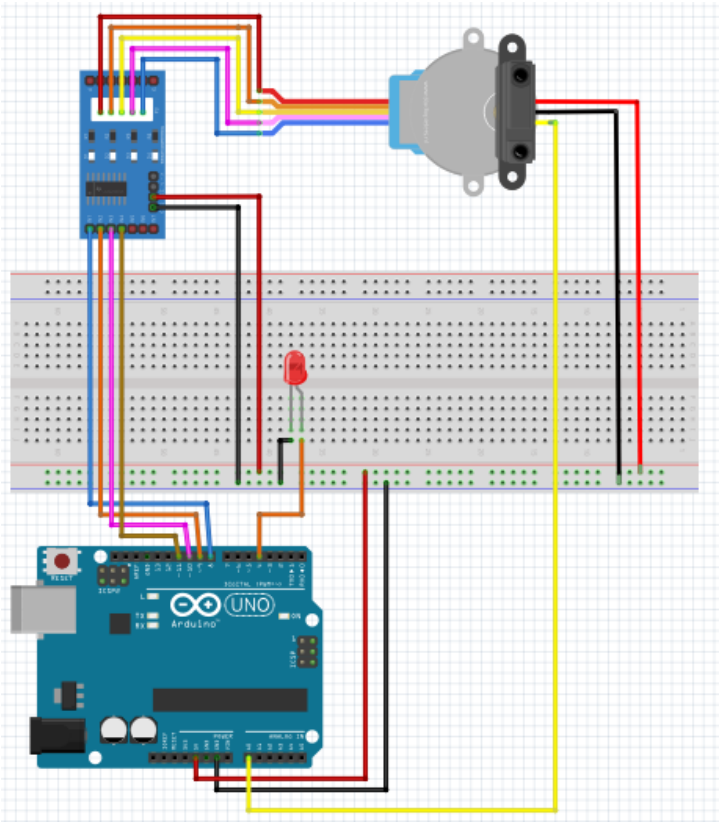


Figura 82. Montaje escáner por infrarrojos

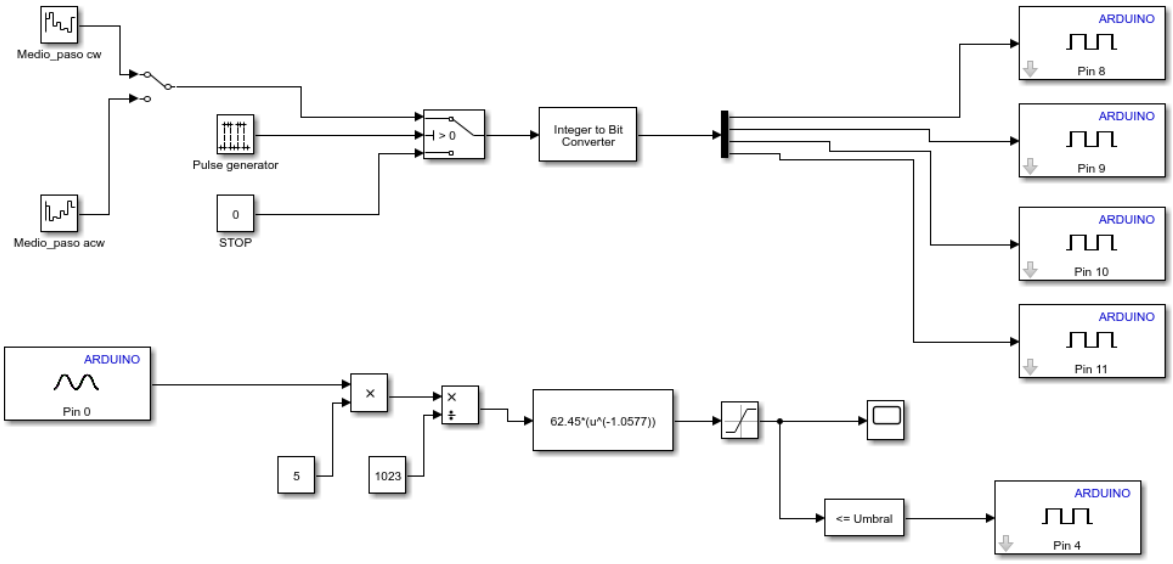


Figura 83. Modelo escáner por infrarrojos

El bloque Analog Input lee la tensión analógica del sensor cada 50 ms y la representa como un número digital de 10 bits de resolución (0 a 1023 para un intervalo de tensión de 0 a 5V). Convertimos este valor a tensión y posteriormente a centímetros empleando la fórmula anterior.

También aplicamos un bloque de saturación para representar únicamente distancias entre 15 cm y 150 cm, el rango útil de nuestro sensor.

Para probar el escáner, este se situará en el siguiente espacio de trabajo, colocando algunos objetos, y se analizarán las distancias tomadas en un barrido de 360º en sentido horario.

Teniendo en cuenta que el escáner tarda 614 instantes de muestreo en recorrer 10º (incluyendo la pausa de 0,5 s), según las ecuaciones 11 y 12, debemos configurar el siguiente tiempo de ejecución para el modelo.

$$Tiempo\ ejecución = \frac{614\ muestras}{10º} * 36 = \frac{22.104\ muestras}{360º} \cong 22{,}1\ segundos \qquad [14]$$

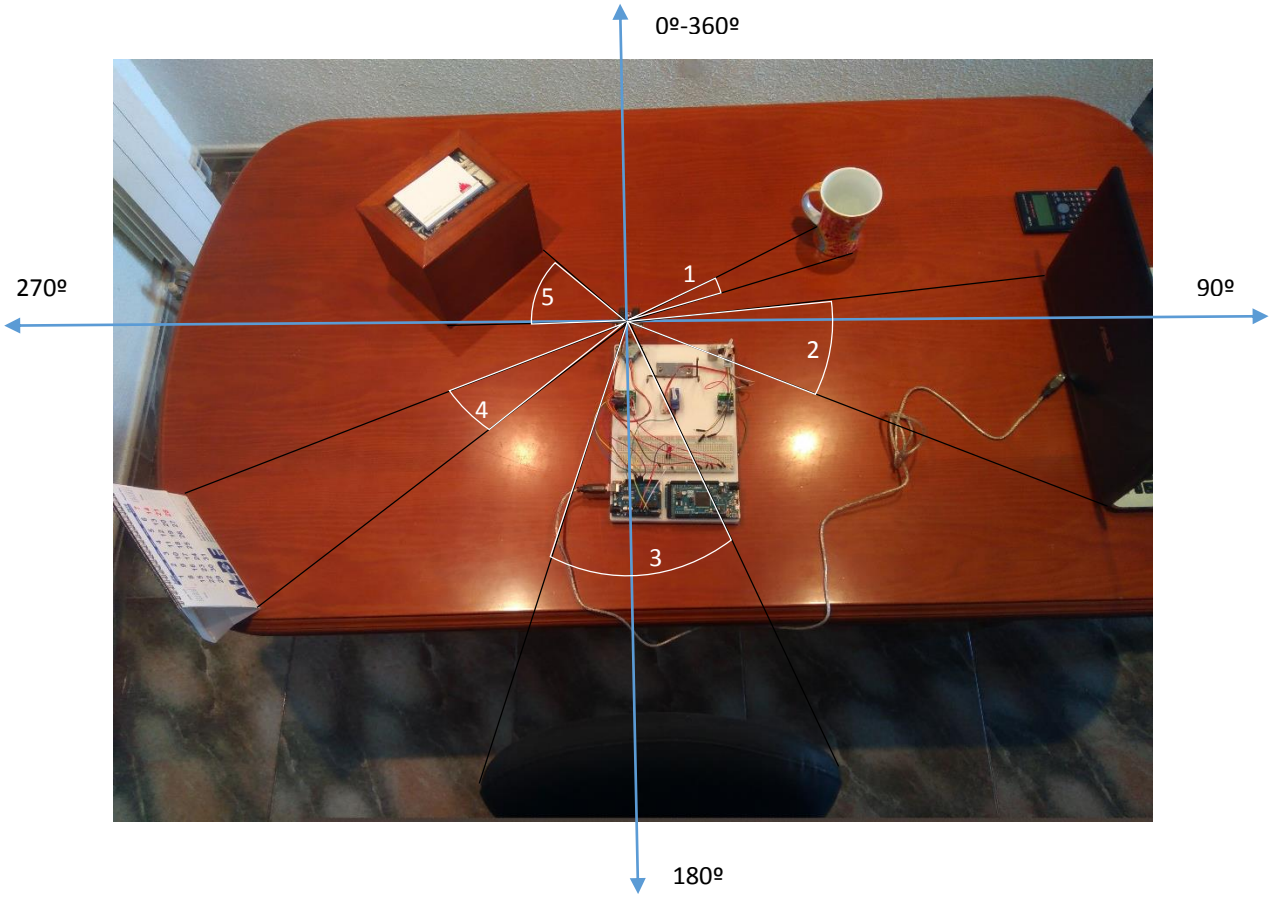


Figura 84. Espacio de trabajo del escáner

A continuación se muestran las medidas tomadas durante el barrido, y los ángulos en los cuales el escáner detecta cada uno de los cinco objetos. Debido a que los bloques Scope siempre usan el tiempo de la simulación para trazar el eje de abcisas, incorporamos un nuevo eje X de posiciones angulares, de 0º a 360º.

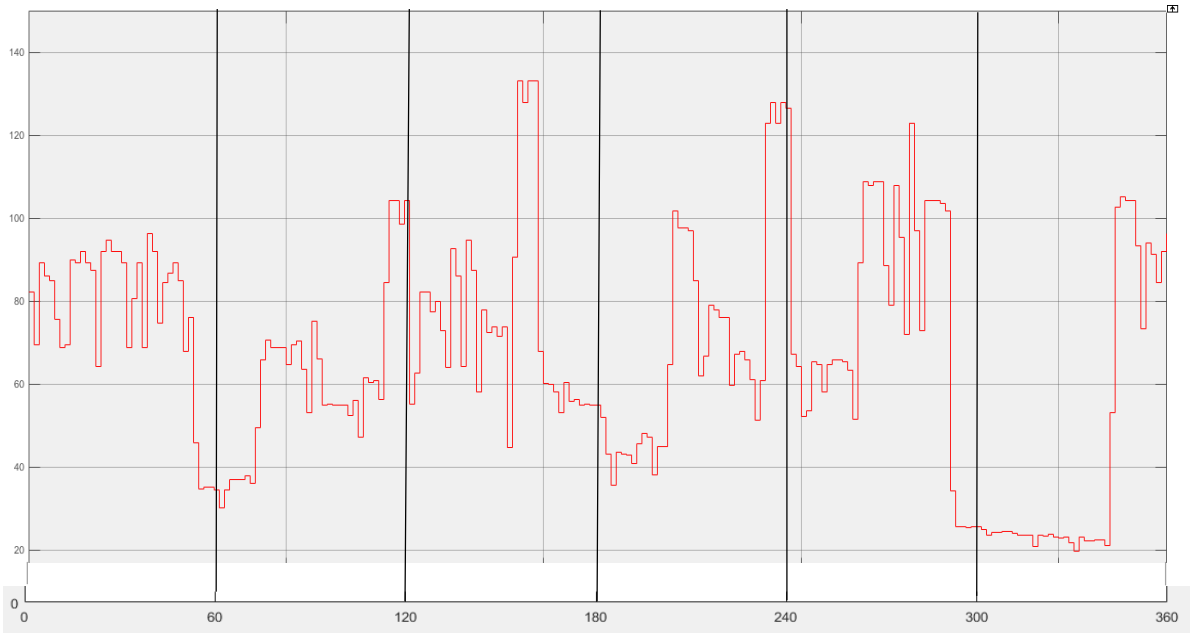


Figura 85. Distancias tomadas por el escáner

Según los resultados, la detección de los objetos cercanos es correcta (se toman medidas de la taza entre x=50º y x=70º, de la pantalla del PC entre x=85º y x=120º aproximadamente, del respaldo de la silla entre x=160º y x=205º, del calendario entre x=240º y x=265º y del bloque entre x=285º y x=340º), y el led permanece iluminado ante la presencia de los objetos más cercanos, la taza y el bloque.

Existe un gran nivel de ruido en las medidas tomadas, de magnitud proporcional a la distancia medida, lo que puede llevar a detecciones erróneas bajo el umbral fijado. Esto puede ser debido a varias causas (el estado o las propias prestaciones del sensor, las condiciones de trabajo o una sensibilidad al movimiento giratorio por parte del dispositivo).

Para mitigar este problema, podemos implementar en el modelo un sistema de medidas medias, tomando N medidas en sucesivos instantes de muestreo y únicamente teniendo en cuenta para la gráfica y la iluminación del led el valor medio de estas medidas.

En ausencia de objetos, el sensor arroja valores de distancia arbitrarios con una amplia diferencia entre muestras sucesivas, por lo que realizando un análisis de la varianza de este conjunto N de muestras podemos determinar si se está detectando un objeto o no.

Aun con estos problemas derivados del medidor de infrarrojos, la aplicación demuestra su eficacia en el movimiento del motor paso a paso y en la detección de objetos situados bajo un umbral pequeño.

5 Conclusiones y futuras líneas de trabajo

En este apartado se detallarán las principales conclusiones extraídas del desarrollo y los resultados de las tres aplicaciones, señalando especialmente las diferencias entre la programación en ambos entornos (MATLAB y Simulink) para cada tipo de motor.

Como último aspecto del proyecto, se utilizarán estas conclusiones para el planteamiento de posibles proyectos futuros de mayor complejidad, para los cuales se utilizará el presente trabajo como base.

-Aplicación con motor DC

En esta aplicación hemos comprobado las siguientes cuestiones:

- Ambos entornos permiten un manejo correcto de los motores DC, si bien en MATLAB se requiere una conexión en serie permanente.
- Las funciones relativas a los encoders del paquete de compatibilidad con MATLAB permiten trabajar con estos dispositivos de forma sencilla y precisa, incluso con resoluciones altas, debido a la potencia del computador de MATLAB.
- Simulink puede presentar problemas de procesamiento al utilizarse en aplicaciones que requieran de un periodo de muestreo muy reducido.
- Es posible integrar librerías de Arduino en los modelos de Simulink, lo que permite diseñar una amplia variedad de aplicaciones sobre estas plataformas con dicho entorno de programación. Esto además se realiza de forma sencilla para el usuario, a diferencia de en MATLAB, que requiere diseñar las liberías desde cero en un nuevo lenguaje.

Un posible proyecto futuro podría ser aprovechar la exactitud de los encoders con MATLAB para desarrollar un trabajo de precisión con motores eléctricos o, por el contrario, manejar un vehículo mediante el entorno de Simulink.

-Aplicación con servomotor

En esta aplicación hemos comprobado las siguientes cuestiones:

- El entorno de MATLAB puede utilizarse para controlar servomotores de forma muy sencilla y precisa, aunque por otro lado se requiere una conexión en serie constante entre el PC y la plataforma Arduino a la cual se conecten dichos motores.
- Este problema puede subsanarse programando el movimiento en el entorno de Simulink empleando los bloques dedicados a los servomotores, a pesar de las limitaciones del periodo de muestreo.
- Es posible integrar una comunicación con módulos bluetooth en Simulink y de esta forma integrarlos en multitud de aplicaciones, incluyendo aquellas relacionadas no solo con servomotores, sino además con los dos tipos restantes: DC y motores paso a paso.

- Usando ambos métodos es posible manejar varios servomotores a la vez con una misma tarjeta, puesto que para el manejo y control de cada uno solo se necesita de un pin PWM y de una entrada analógica. La Arduino Uno posee 6 pines que pueden operar como salidas PWM y 6 entradas analógicas, por lo que puede usarse para manejar un total de 6 servomotores a la vez en una aplicación.

Debido a la comunicación en serie requerida en MATLAB, el control de servomotores mediante script es idóneo para aplicaciones estacionadas, aquellas en las que la plataforma Arduino queda fija en una posición determinada.

Los brazos robóticos responden precisamente a estos dos principios: son controlados mediante un conjunto de servomotores y su base permanece fija en un único lugar, además de ser ampliamente utilizados en multitud de industrias, como la del automóvil.

El número de grados de libertad de un brazo robot es igual al número de servos que controlan su movimiento. Sin incluir pines adicionales, es posible manejar un robot de hasta 6 grados de libertad con la Arduino Uno, la tarjeta más básica de la serie.

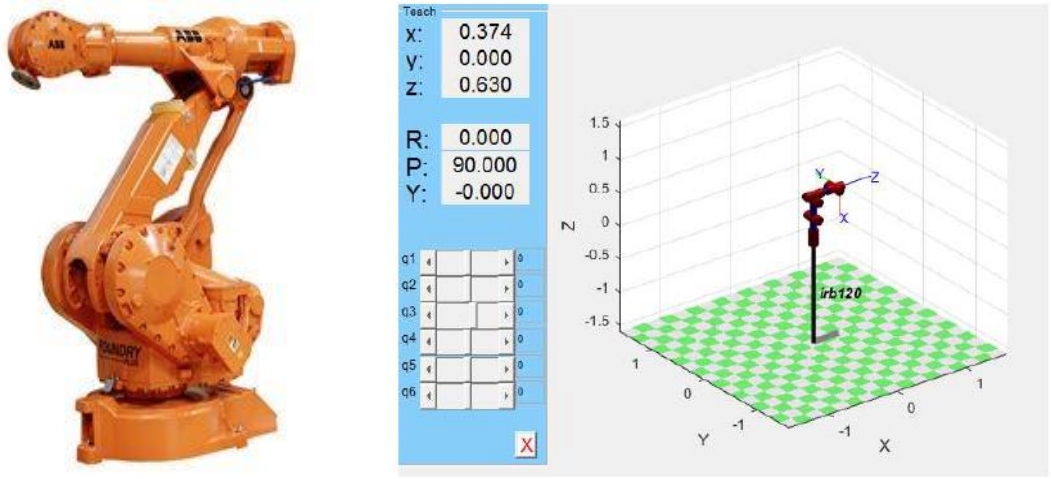


Figura 86. Brazo robot de ABB con 6 GDL y simulación en MATLAB

Un proyecto interesante sería la construcción de un brazo robot casero y la programación del mismo utilizando MATLAB o por Bluetooth en Simulink. MATLAB además, dispone de la Robotics System Toolbox, enfocada en la simulación y programación de brazos robóticos.

De esta forma, puede realizarse tanto las simulaciones pertinentes en base a las características mecánicas del robot construido como el manejo de dicho brazo empleando el mismo entrorno de programación.

-Aplicación con motor paso a paso

En esta aplicación hemos comprobado las siguientes cuestiones:

- En condiciones normales, el entorno de MATLAB no es apropiado para el manejo de los motores paso a paso, debido al gran número de sentencias que deben ejecutarse en serie y a la limitación de escribir en un único pin cada vez. Solo es posible subsanar este problema utilizando un shield Ethernet o las funciones del Adafruit Motor Shield v2,
- Hemos comprobado que Simulink permite manejar correctamente este tipo de motores, debido al esquema de sus modelos de actuar como una interrupción periódica que se corresponde con cada fragmento de la secuencia de movimiento del motor. Esto permite además que el tiempo previsto y el real para la ejecución de una aplicación sean más parecidos entre sí que en el caso de MATLAB, donde el tiempo de ejecución extra de las sentencias puede producir errores inesperados.
- Los bloques de entradas analógicas permiten la integración de sensores de diversa índole en las aplicaciones con MATLAB/Simulink. También es posible crear con facilidad librerías o sistemas de bloques en base a cada tipo de sensor con los bloques S-Funtion Builder o Subsystem.
- Podemos programar movimientos periódicos con exactitud empleando generadores de ondas cuadradas como señales de control.

Un posible proyecto futuro podría consistir en el aprovechamiento de la alta resolución del movimiento del propio motor y de la ejecución en Simulink para un trabajo donde la precisión sea un aspecto fundamental, como el posicionamiento de distintas piezas en una cadena de montaje.

6 Manual de usuario

En este apartado se indicarán los pasos necesarios para realizar la instalación de todos los elementos software necesarios para el desarrollo del proyecto que no se encuentren ya instalados de forma predeterminada en MATLAB/Simulink.

6.1 Guía de instalación de los paquetes de compatibilidad con Arduino

Los paquetes de compatibilidad de Arduino con MATLAB y Simulink pueden descargarse de forma gratuita desde el sitio web de MathWorks tras hacerse una cuenta.

-Tras abrir MATLAB, pinchamos en la pestaña “Add-Ons” y en su apartado “Get Hardware Support Packages”.

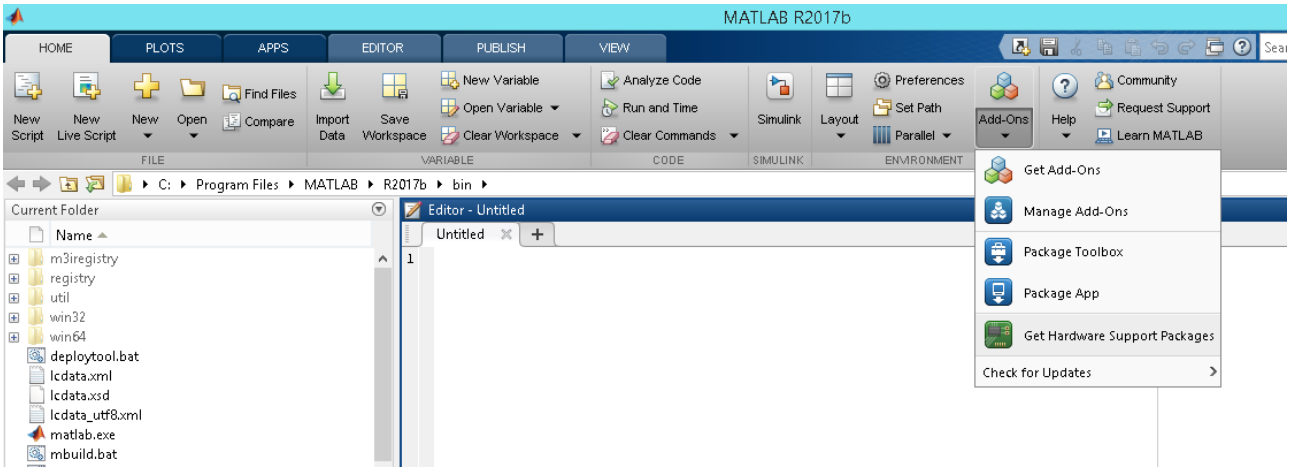


Figura 87. Instalador de paquetes de soporte en MATLAB

-Buscamos los dos paquetes, “MATLAB Support Package for Arduino Hardware” y “Simulink Support Package for Arduino Hardware”, y en sus páginas de presentación seleccionamos la opción Install en lugar de únicamente descargar los paquetes.

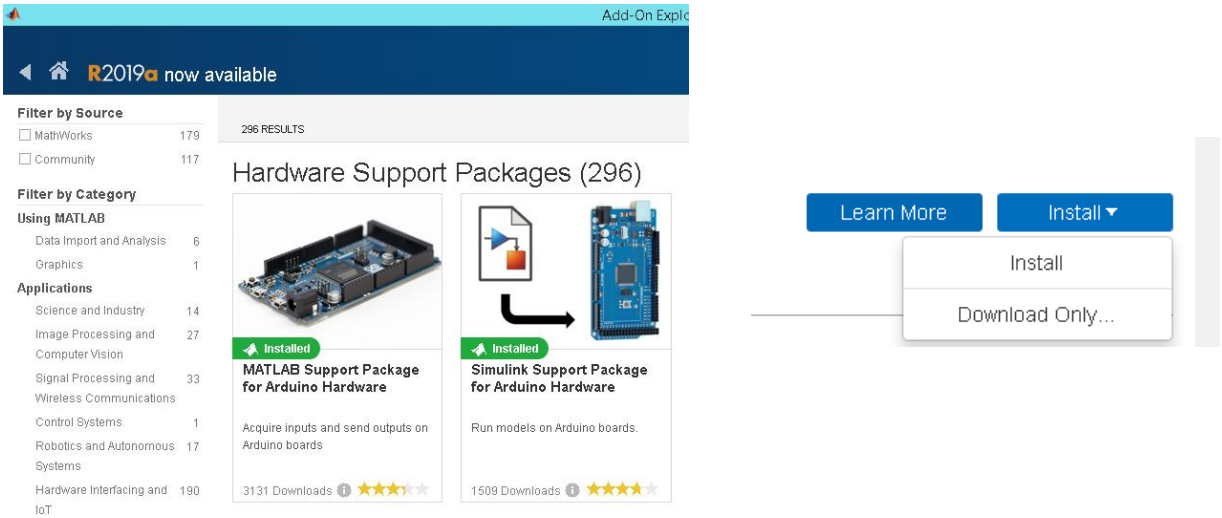


Figura 88. Paquetes de compatibilidad con Arduino

-Tras aceptar el acuerdo de licencia comenzará la descarga y la instalación. Si no se ha iniciado sesión anteriormente también hay que indicar nuestro nombre de usuario y la contraseña.

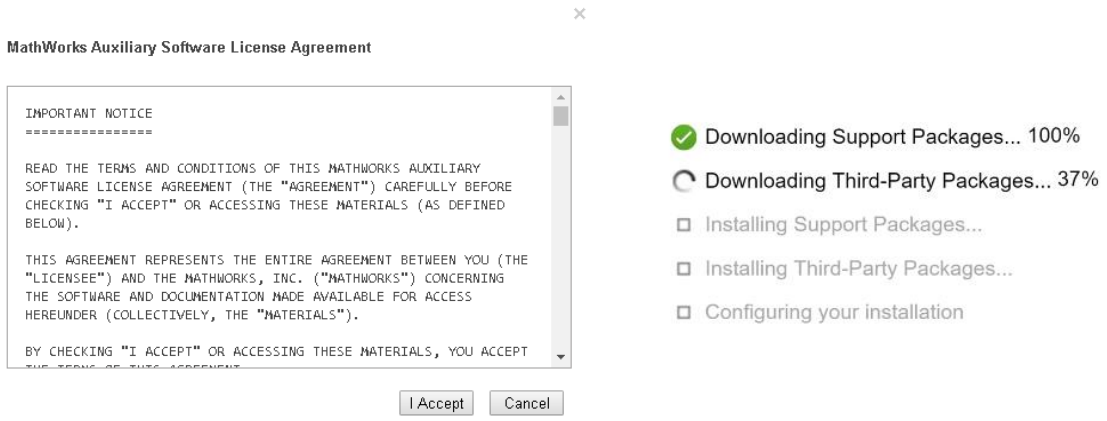


Figura 89. Términos y condiciones y proceso de descarga e instalación

-En el caso del paquete de MATLAB nos saltará un aviso tras finalizar la instalación para permitir la instalación de un driver necesario para conectar las plataformas Arduino. Pinchamos en "Next" y en "Yes" en la siguiente ventana.

Figura 90. Instalación de driver para hardware Arduino

Tras la instalación de los paquetes y del driver nos pedirá preparar la conexión con las plataformas Arduino mediante cable USB o conexiones Bluetooth o Wifi, tal y como se indica en el apartado 3.3 de esta memoria.

7 Presupuesto

A continuación se adjunta un desglose de los costes estimados para la realización de este proyecto. Se ha dividido este presupuesto en costes de elementos hardware, costes de elementos software y costes por mano de obra.

7.1 Costes de elementos hardware

Concepto	Precio	Cantidad	Total
Portátil Intel Core i7-4510U	620 €	1	620 €
Arduino UNO	20 €	1	20 €
Arduino Due	35 €	1	35 €
Motor GA12-N20 con encoder integrado	10 €	1	10 €
Servomotor SG90	7 €	1	7 €
Motor 28BYJ-40 con driver ULN2003	4 €	1	4 €
Sensor GP2Y0A02YK0F	9 €	1	9 €
Driver puente en H L9110S	5 €	1	5 €
Pantalla LCD 16x2	9 €	1	9 €
TOTAL COSTES HARDWARE			**719 €**

Tabla 13. Costes de elementos hardware

Todos los elementos se han comprado con el IVA incluido en el precio.

7.2 Costes de elementos software

Concepto	Precio	Cantidad	Total
Windows 8.1	0 €	1	0 €
MATLAB 2017b	0 €	1	0 €
Microsoft Office 365	0 €	1	0 €
Arduino	0 €	1	0 €
Paquetes de compatibilidad de MATLAB y Simulink con Arduino	0 €	1	0 €
Fritzing	0 €	1	0 €
TOTAL COSTES SOFTWARE			**0 €**

Tabla 14. Costes de elementos software

Esta parte del presupuesto tiene coste cero debido a que el sistema operativo viene instalado de base junto al PC. Por otra parte, MATLAB y Microsoft Office ofrecen una versión de prueba gratuita para los estudiantes y las herramientas de software de Arduino son de libre distribución.

7.3 Costes de mano de obra

Concepto	Precio/hora	Nº horas	Total
Desarrollo del proyecto	30 €/hora	350 horas	10.500 €
Documentación	15 €/hora	120 horas	1.800 €
TOTAL COSTES MANO DE OBRA			**12.300 €**

Tabla 15. Costes de mano de obra

Se ha dividido el trabajo en dos perfiles diferentes. El desarrollo del proyecto corresponde al tiempo empleado en la elección de los componentes, en la instalación de los programas necesarios, en la comprensión del funcionamiento de los distintos motores y de las opciones que ofrecen los paquetes de compatibilidad con Arduino y en la programación de las distintas aplicaciones, junto con las comprobaciones experimentales realizadas. La documentación corresponde al tiempo empleado en la redacción y corrección de esta memoria.

7.4 Coste total

Concepto	Precio	Cantidad	Total
Costes de elementos hardware	719 €	1	719 €
Costes de elementos software	0 €	1	0 €
Costes de mano de obra	12.300 €	1	12.300 €
COSTE TOTAL DEL PROYECTO			**13.019 €**

Tabla 16. Coste total

El importe total del proyecto asciende a la cantidad de:

“Trece mil diecinueve euros, IVA incluido”

8 Bibliografía

[1] Dale Wheat, Arduino Internals, 2011.

[2] Amos Gilat, MATLAB Una intrducción con ejemplos prácticos, Ed. Reverté, 2000.

[3] Jesús Fraile Mora, Máquinas eléctricas, Ed. Mc Graw Hill, 2003.

[4] www.arduino.cc.

[5] es.mathworks.com.

[6] www.mathworks.com/help/pdf_doc/simulink/sfunctions.pdf.

9 Anexos

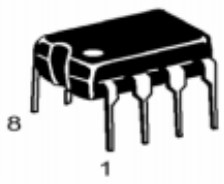
Datasheet chip L9110

Motor control driver chip

L9110

Features:

- Low quiescent current;
- Wide supply voltage range: 2.5V-12V;
- 800mA continuous current output capability per channel;
- Lower saturation voltage;
- TTL / CMOS output level compatible, and can be directly connected to the CPU;
- Output built-in clamp diodes for inductive load;
- Integrated control and drive into a monolithic IC;
- With pin high-voltage protection function;
- Operating temperature: 0 ℃ -80 ℃.



DP Suffix plastic package (DIP8)

SO Suffix plastic package (SOP8)

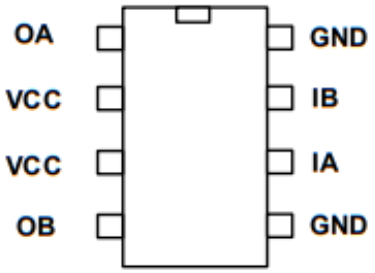
Description：

L9110 The ASIC device control and drive motor design two-channel push-pull power amplifier discrete circuits integrated into a monolithic IC, peripheral devices and reduce the cost, improve the reliability of the whole. This chip has two TTL / CMOS compatible with the level of the input, with good resistance; two output terminals can directly forward and reverse movement of the drive motor, it has a large current driving capability, each channel through 750 ~ 800mA of continuous current, peak current capability up to 1.5 ~ 2.0A; while it has a low output saturation voltage; built-in clamp diode reverse the impact of the current release inductive load it in the drive relays, DC motors, stepper motor or switch power tube use on safe and reliable. L9110 is widely used in toy car motor drives, stepper motor drive and switching power tube circuit.

Pin definitions:

No.	Symbol	Function
1	OA	A road output pin
2	VCC	Supply Voltage
3	VCC	Supply Voltage
4	OB	B output pin
5	GND	Ground
6	IA	A road input pin
7	IB	B input pin
8	GND	Ground

Device pins Figure



Test conditions： Vcc =9V , Iout =750mA

Symbol	Parameters	minimum	Typical	maximum	units
VH_{out}	Output high	7.50	7.60	7.70	V
VL_{out}	Output low	0.35	0.45	0.55	V
VH_{in}	Input high	2.5	5.0	9.0	V
VL_{in}	Input low	0	0.5	0.7	V

Motor control driver chip

L9110

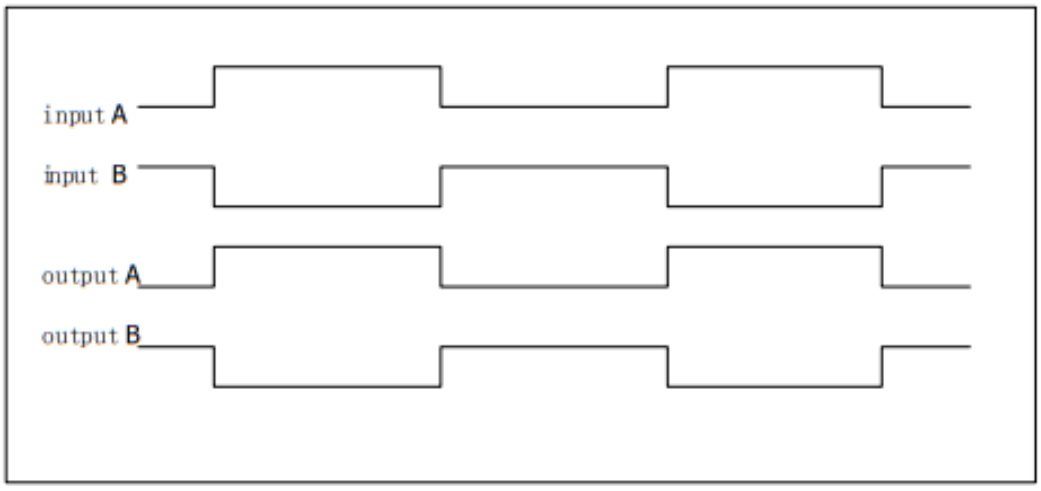
Electrical characteristics:

Symbol	Parameters	Range			Units
		Min	Typical	Max	
VCC	Supply Voltage	2.5	6	12	V
Idd	Quiescent Current	—	0	2	uA
Iin	Operating current	200	350	500	uA
IC	Continuous	750	800	850	mA
IMax	Current peak	—	1500	2000	mA

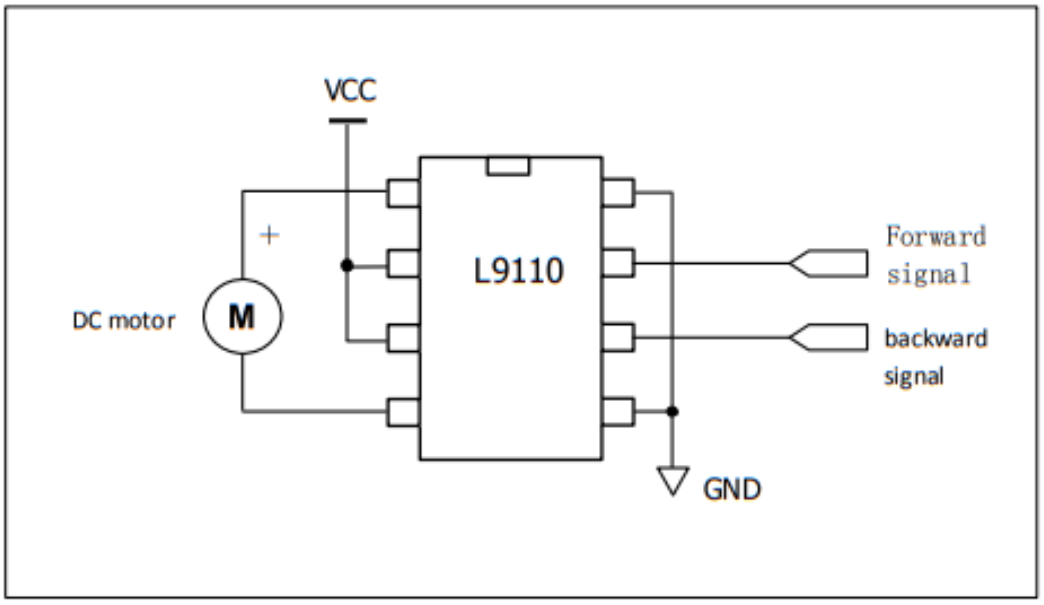
Logical relationship:

IA	IB	OA	OB
H	L	H	L
L	H	L	H
L	L	L	L
H	H	L	L

Pin waveform diagram:



Application Circuit:



Datasheet sensor SHARP GP2Y0A02YK0F

GP2Y0A02YK0F

Distance Measuring Sensor Unit
Measuring distance: 20 to 150 cm
Analog output type

■Description

GP2Y0A02YK0F is a distance measuring sensor unit, composed of an integrated combination of PSD (position sensitive detector) , IRED (infrared emitting diode) and signal processing circuit.
The variety of the reflectivity of the object, the environmental temperature and the operating duration are not influenced easily to the distance detection because of adopting the triangulation method.
This device outputs the voltage corresponding to the detection distance. So this sensor can also be used as a proximity sensor.

■Features

1. Distance measuring range : 20 to 150 cm
2. Analog output type
3. Package size : 29.5×13×21.6 mm
4. Consumption current : Typ. 33 mA
5. Supply voltage : 4.5 to 5.5 V

■Agency approvals/Compliance

1. Compliant with RoHS directive (2002/95/EC)

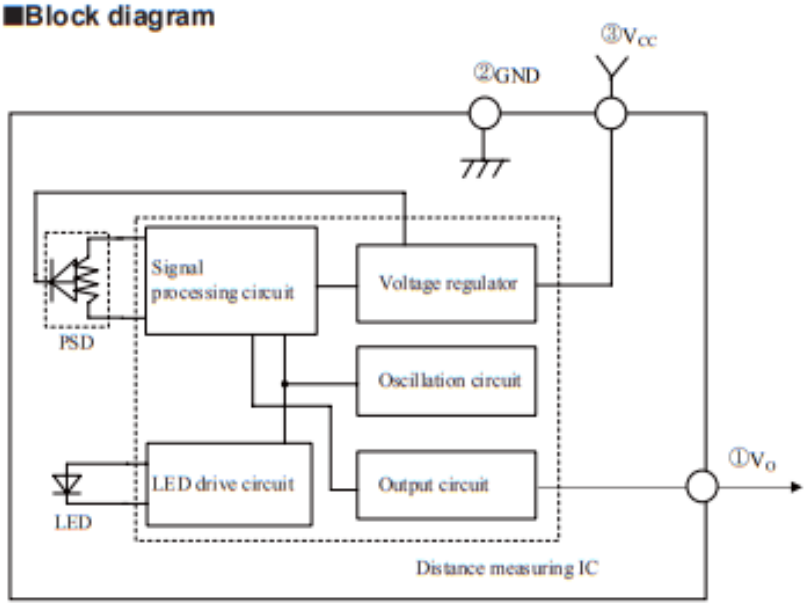
■Applications

1. Touch-less switch
   (Sanitary equipment, Control of illumination, etc.)
2. Sensor for energy saving
   (ATM, Copier, Vending machine, Laptop computer, LCD monitor)
3. Amusement equipment
   (Robot, Arcade game machine)

Sheet No.: E4-A00101EN
Date Dec.01.2006

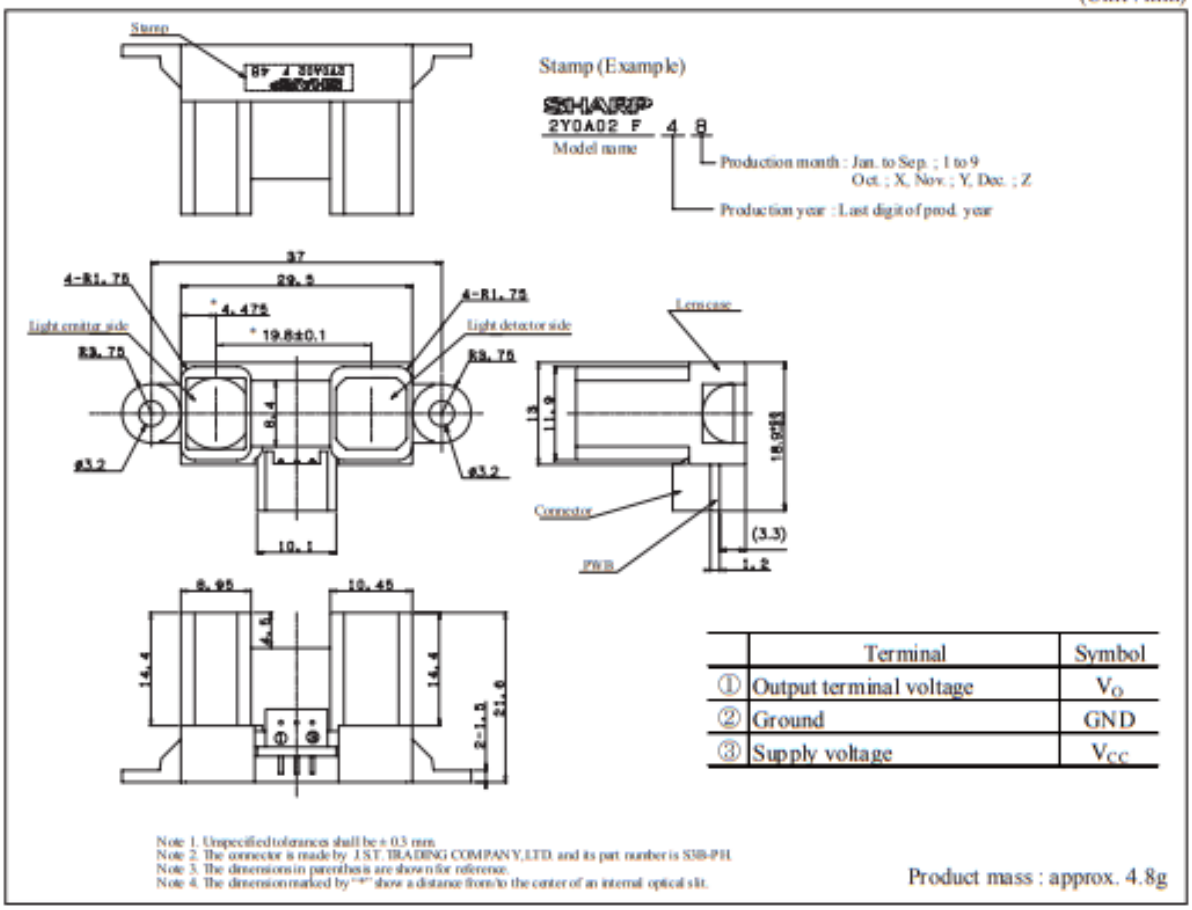
SHARP GP2Y0A02YK0F

■Block diagram



■Outline Dimensions

(Unit : mm)



	Terminal	Symbol
①	Output terminal voltage	V_O
②	Ground	GND
③	Supply voltage	V_{CC}

Note 1. Unspecified tolerances shall be ± 0.3 mm.
Note 2. The connector is made by J.S.T. TRADING COMPANY, LTD. and its part number is S3B-PH.
Note 3. The dimensions in parenthesis are shown for reference.
Note 4. The dimension marked by "*" show a distance from/to the center of an internal optical slit.

Product mass : approx. 4.8g

Sheet No.: E4-A00101EN

2

SHARP GP2Y0A02YK0F

■Absolute Maximum Ratings (Ta=25℃, Vcc=5V)

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	-0.3 to +7	V
Output terminal voltage	V_O	-0.3 to V_{CC}+0.3	V
Operating temperature	T_{opr}	-10 to +60	℃
Storage temperature	T_{stg}	-40 to +70	℃

■Electro-optical Characteristics (Ta=25℃, Vcc=5V)

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Average supply current	I_{CC}	L=150cm (Note 1)	—	33	50	mA
Measuring distance range	ΔL	(Note 1)	20	—	150	cm
Output voltage	V_O	L=150cm (Note 1)	0.25	0.4	0.55	V
Output voltage differential	ΔV_O	Output voltage difference between L=20cm and L=150cm (Note 1)	1.8	2.05	2.3	V

* L : Distance to reflective object
Note 1 : Using reflective object : White paper (Made by Kodak Co., Ltd. gray cards R-27・white face, reflectance; 90%)

■Recommended operating conditions

Parameter	Symbol	Conditions	Rating	Unit
Supply voltage	V_{CC}		4.5 to 5.5	V

Fig. 1 Timing chart

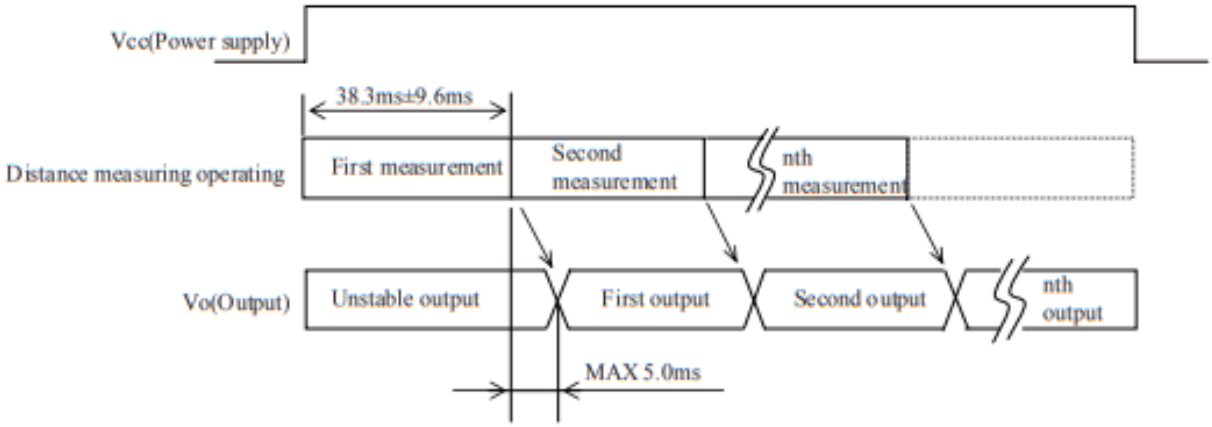
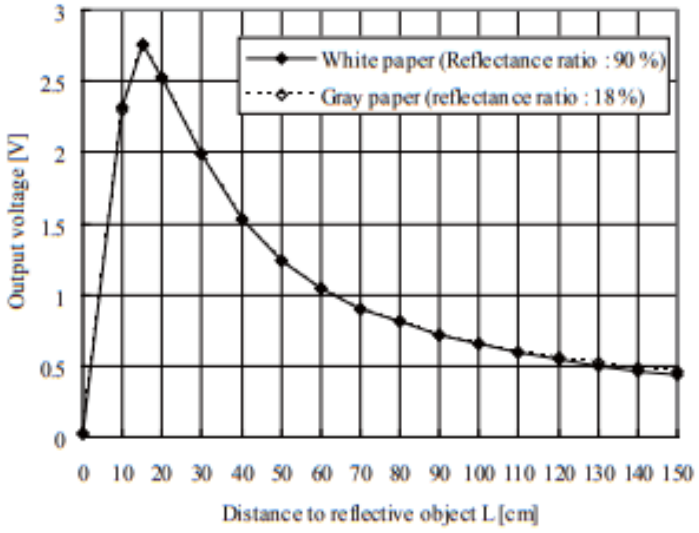
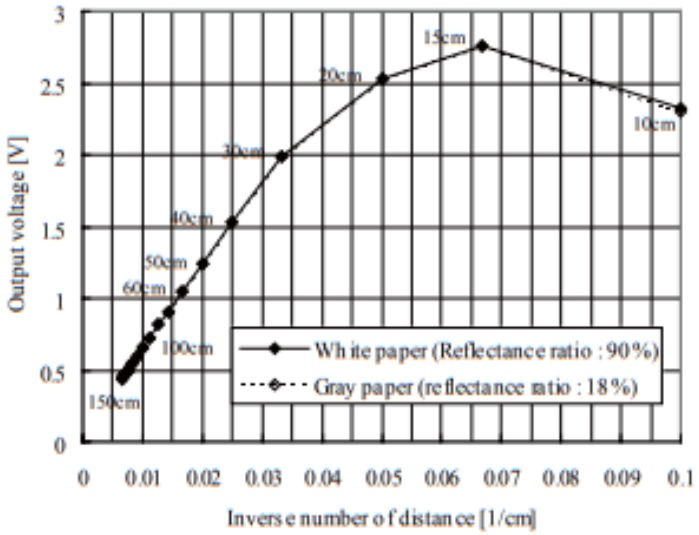


Fig. 2 Example of distance measuring characteristics (output)





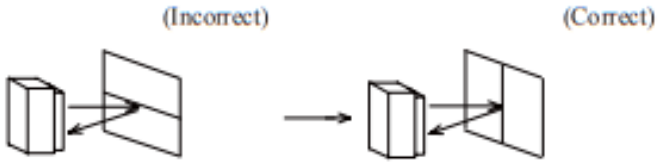
■Notes

●Advice for the optics

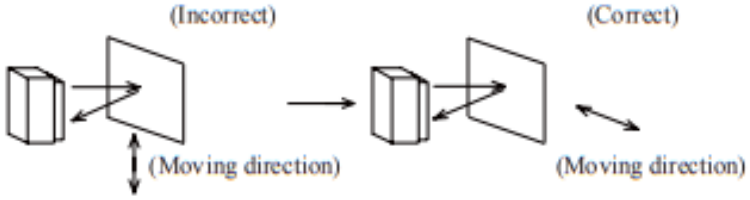
- The lens of this device needs to be kept clean. There are cases that dust, water or oil and so on deteriorate the characteristics of this device. Please consider in actual application.
- Please don't do washing. Washing may deteriorate the characteristics of optical system and so on. Please confirm resistance to chemicals under the actual usage since this product has not been designed against washing.

●Advice for the characteristics

- In case that an optical filter is set in front of the emitter and detector portion, the optical filter which has the most efficient transmittance at the emitting wavelength range of LED for this product ($\lambda = 850 \pm 70$nm), shall be recommended to use. Both faces of the filter should be mirror polishing. Also, as there are cases that the characteristics may not be satisfied according to the distance between the protection cover and this product or the thickness of the protection cover, please use this product after confirming the operation sufficiently in actual application.
- In case that there is an object near to emitter side of the sensor between sensor and a detecting object, please use this device after confirming sufficiently that the characteristics of this sensor do not change by the object.
- When the detector is exposed to the direct light from the sun, tungsten lamp and so on, there are cases that it can not measure the distance exactly. Please consider the design that the detector is not exposed to the direct light from such light source.
- Distance to a mirror reflector can not be sometimes measured exactly. In case of changing the mounting angle of this product, it may measure the distance exactly.
- In case that reflective object has boundary line which material or color etc. are excessively different, in order to decrease deviation of measuring distance, it shall be recommended to set the sensor that the direction of boundary line and the line between emitter center and detector center are in parallel.



- In order to decrease deviation of measuring distance by moving direction of the reflective object, it shall be recommended to set the sensor that the moving direction of the object and the line between emitter center and detector center are vertical.



●Advice for the power supply

- In order to stabilize power supply line, we recommend to insert a by-pass capacitor of 10μF or more between Vcc and GND near this product.

●Notes on handling

- There are some possibilities that the internal components in the sensor may be exposed to the excessive mechanical stress. Please be careful not to cause any excessive pressure on the sensor package and also on the PCB while assembling this product.

● **Presence of ODC etc.**

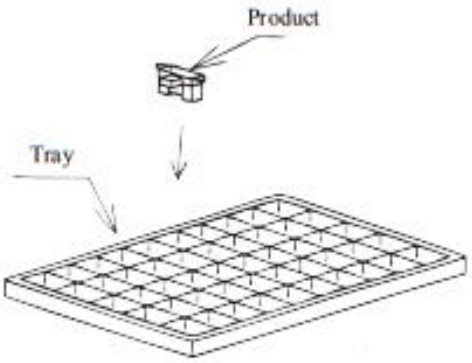
This product shall not contain the following materials.
And they are not used in the production process for this product.
Regulation substances : CFCs, Halon, Carbon tetrachloride, 1.1.1-Trichloroethane (Methylchloroform)

Specific brominated flame retardants such as the PBB and PBDE are not used in this product at all.

This product shall not contain the following materials banned in the RoHS Directive (2002/95/EC).

- Lead, Mercury, Cadmium, Hexavalent chromium, Polybrominated biphenyls (PBB), Polybrominated diphenyl ethers (PBDE).

■Package specification



MAX. 50 pieces per tray

■Important Notices

· The circuit application examples in this publication are provided to explain representative applications of SHARP devices and are not intended to guarantee any circuit design or license any intellectual property rights. SHARP takes no responsibility for any problems related to any intellectual property right of a third party resulting from the use of SHARP's devices.

· Contact SHARP in order to obtain the latest device specification sheets before using any SHARP device. SHARP reserves the right to make changes in the specifications, characteristics, data, materials, structure, and other contents described herein at any time without notice in order to improve design or reliability. Manufacturing locations are also subject to change without notice.

· Observe the following points when using any devices in this publication. SHARP takes no responsibility for damage caused by improper use of the devices which does not meet the conditions and absolute maximum ratings to be used specified in the relevant specification sheet nor meet the following conditions:
(i) The devices in this publication are designed for use in general electronic equipment designs such as:
--- Personal computers
--- Office automation equipment
--- Telecommunication equipment [terminal]
--- Test and measurement equipment
--- Industrial control
--- Audio visual equipment
--- Consumer electronics
(ii) Measures such as fail-safe function and redundant design should be taken to ensure reliability and safety when SHARP devices are used for or in connection with equipment that requires higher reliability such as:
--- Transportation control and safety equipment (i.e., aircraft, trains, automobiles, etc.)
--- Traffic signals
--- Gas leakage sensor breakers
--- Alarm equipment
--- Various safety devices, etc.
(iii) SHARP devices shall not be used for or in connection with equipment that requires an extremely high level of reliability and safety such as:
--- Space applications
--- Telecommunication equipment [trunk lines]
--- Nuclear power control equipment
--- Medical and other life support equipment (e.g., scuba).

· If the SHARP devices listed in this publication fall within the scope of strategic products described in the Foreign Exchange and Foreign Trade Law of Japan, it is necessary to obtain approval to export such SHARP devices.

· Contact and consult with a SHARP representative if there are any questions about the contents of this publication.

Lista de comandos AT para configurar el módulo HC-05

Command	Function
AT	Test UART Connection
AT+RESET	Reset Device
AT+VERSION	Query firmware version
AT+ORGL	Restore settings to Factory Defaults
AT+ADDR	Query Device Bluetooth Address
AT+NAME	Query/Set Device Name
AT+ROLE	Query/Set Device Role
AT+CLASS	Query/Set Class of Device CoD
AT+IAC	Query/Set Inquire Access Code
AT+INQM	Query/Set Inquire Access Mode
AT+PSWDAT+PIN	Query/Set Pairing Passkey
AT+UART	Query/Set UART parameter
AT+CMODE	Query/Set Connection Mode
AT+BIND	Query/Set Binding Bluetooth Address
AT+POLAR	Query/Set LED Output Polarity
AT+PIO	Set/Reset a User I/O pin

Tabla 17. Lista de comandos AT para módulos Bluetooth (4)

(4) https://cdn.instructables.com/ORIG/FKY/Z0UT/HX7OYY7I/FKYZ0UTHX7OYY7I.pdf

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR