Departamento de Ingeniería de Sistemas y Automática

PROYECTO FIN DE CARRERA

INGENIERÍA INDUSTRIAL

Sistemas de Adquisición de Datos basados en la plataforma Arduino. Aplicaciones a Matlab, Simulink y Android.

Autor: MARIO DE LA HORRA KÖLLMER

Director: RAMÓN IGNACIO BARBER CASTAÑO

Leganés, Mayo 2013

Título:

Autor: Mario De la Horra Köllmer

Director: Ramón Ignacio Barber Castaño

Tutor: Ramón Ignacio Barber Castaño.

EL TRIBUNAL

Presidente:_____.

Vocal:_____.

Secretario:_____.

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Escribo éstas líneas gracias a quienes lo han hecho posible, Familia, Amigos, Profesores y Compañeros. Este trabajo es vuestro.

En Madrid, a 7 de Mayo del 2013,

Mario

Resumen

En este Proyecto se desarrolla un sistema de adquisición de datos de bajo coste con el doble objetivo de que por un lado sea muy sencillo de usar para un destino educativo y, por otro lado, de que se pueda conectar mediante la arquitectura cliente-servidor a otros dispositivos de forma inalámbrica o cableada para realizar el control.

En el sistema prima la interfaz microcontrolador-cliente, de forma que busca de forma intensiva diferentes caminos para conectar ambos, bien a través del ordenador o dispositivo móvil.

Además de toda la investigación y el desarrollo del sistema tanto en la parte hardware, que ha consistido en el acondicionamiento de la señal obtenida de los sensores y proporcionada a los actuadores, del cual se ha llevado a cabo su implantación física haciendo uso de microcontroladores (Arduino MEGA y Arduino UNO) y módulos de expansión de comunicaciones (Ethernet Shield); como de la parte software, mediante el lenguaje de programación de la plataforma, muy similar a C, y Simulink, para implementar los sistemas de control. También se han desarrollado pruebas para validar el sistema.

Al final se propone, resuelve y prueba un ejercicio práctico de laboratorio.

Abstract

This Project develops a low cost Data Acquisition System with the double goal that would be easy to use for learners and easy to connect through several systems.

The system highlights the microcontroller-client interface, so propose several ways to connect both, through a computer or a mobile device.

Also it develops on one hand the hardware interface that consist of signal conditioning, from sensors and to actuators, physical implementation using Arduino microcontrollers and shields. On the other hand the mobile interface is developed and several tests are done to verify the whole system.

Finally a practice exercise is proposed, solved and tested.

Índice

Índice de figuras

1. Introducción

1.1. Motivación

El origen de este trabajo se encuentra en la búsqueda de sistemas de adquisición de datos de bajo coste que permitan por un lado integrar la experimentación en el aprendizaje y por otro ampliar las interfaces con dichos sistemas buscando la movilidad y la sencillez de uso.

En relación a la búsqueda de la movilidad, la motivación parte de la necesidad de que si bien la medida de una magnitud física en innumerables ocasiones se realiza en el sitio, los datos adquiridos deben ser ubicuos, es decir estar disponibles en cualquier momento y lugar. De esta manera, ya se esté midiendo el ritmo cardiaco de un paciente, la velocidad de un motor, o la temperatura de un horno, aunque el paciente, el motor o el horno no cambien de ubicación, el médico, la persona o el programa que recoge los datos no sólo podrán no estar en la misma habitación o laboratorio, sino que además los datos se encontrarán desde dónde sean solicitados.

Por otra parte la integración con el entorno educativo requiere unas características especiales, la principal de ellas es la sencillez, debe de tratar con sistemas muy intuitivos de forma que el alumno pueda dirigir toda su energía al aprendizaje mediante la experimentación y desarrollar en el mismo sus habilidades. La segunda característica crucial es el coste, que deberemos minimizar para evitar crear barreras de entrada, por lo que nuestro esfuerzo se dirigirá a crear un lugar para la experimentación muy asequible.

1.2. Objetivos del proyecto

A raíz de las motivaciones surgen los objetivos. Satisfacer las motivaciones implica desarrollar un sistema sencillo, cuya interacción sea intuitiva. Además debe ser modular, para permitir su comunicación con el exterior, y el añadido características con facilidad a coste reducido, de forma que su adopción tenga las mínimas barreras de entrada.

Los objetivos del proyecto se concretan en:

- Realizar un sistema basado en Arduino que permita controlar un sistema físico mediante Matlab o Simulink.
- Implementación de etapa de acondicionamiento de la señal entre el microcontrolador y el mundo físico.
- Implementación de una interfaz móvil sobre la plataforma Android.

En primer lugar será elegida una plataforma que permita cumplir con las especificaciones dadas por la maqueta de un motor que hará de mundo físico del que los datos serán adquiridos, tras esto se realizarán pruebas en el laboratorio con la maqueta del motor de corriente continua. Para ello, desde la tarjeta de adquisición de datos, teniendo en cuenta la plataforma experimental utilizada será realizada una

etapa de acondicionamiento de la señal. Por el lado del ordenador se utilizará una interfaz de conexión.

Para el desarrollo de una solución móvil, se investigarán las posibilidades de la tecnología web, y realizará una aplicación para una plataforma móvil.

1.3. Partes del documento

A continuación se presentan los contenidos de éste documento.

En el Capítulo Segundo trata sobre el estado del arte y de la técnica junto a las soluciones propuestas, en los campos de tarjetas de adquisición de datos, arduino, Matlab y Android.

En el Capítulo Tercero se explica la tarjeta de adquisición de datos empleada basada en Arduino. Se desarrolla el sistema de acondicionamiento entre un sistema físico y el microcontrolador, y se realizan las pruebas de verificación del sistema. También se analiza la validez de Arduino como tarjeta de Adquisición de datos.

En el Capítulo Cuarto se desarrolla e implementa una interfaz móvil sobre la plataforma Android que se comunica con el microcontrolador.

En el Capítulo Quinto se muestran como resultados experimentales una práctica de laboratorio donde se muestra el uso del sistema experimentado.

Por último, en el Capítulo Sexto se comentan las conclusiones y se muestran las posibles líneas de ampliación y mejora del sistema propuesto.

2. Estado del Arte

En este capítulo se presenta el estado tecnológico en relación a las plataformas disponibles e interfaces existentes para la implementación de Tarjetas de Adquisición de Datos de bajo coste.

2.1. Microcontrolador

"Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos" [1].

Dicha plataforma consiste en un soporte hardware formado por una placa que proporciona la alimentación, el oscilador, y la carga del programa en microcontroladores de la serie 8 bit de Atmel y soporte software formado por un entorno de desarrollo basado en Wiring [2], que es un framework abierto para la programación de microcontroladores; y un lenguaje llamado Processing [3], usado para el aprendizaje, realización de prototipos y productos (Figura 1).

En este trabajo se ha optado por adquirir las placas ensambladas, sin embargo gracias a que toda la documentación se encuentra disponible de forma abierta, las placas pueden ser ensambladas manualmente y que sean compatibles con dicha plataforma como se muestra en la Figura 1.

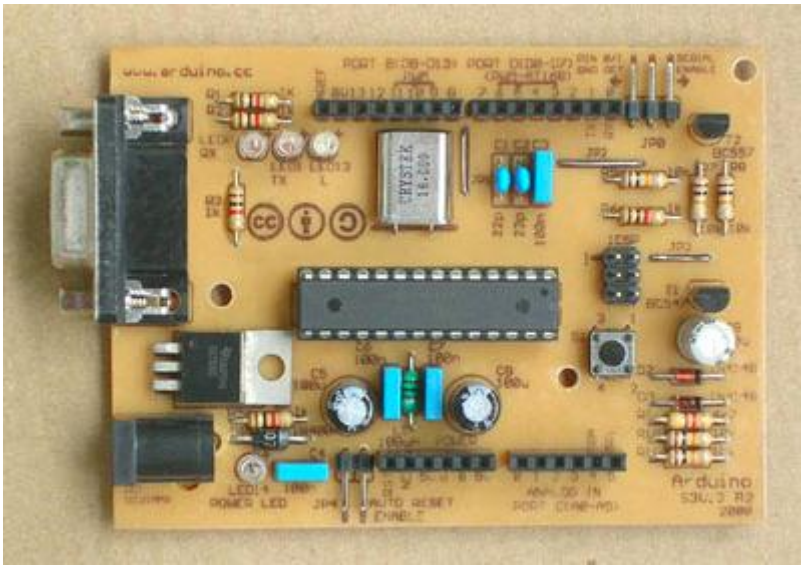


Figura 1 Placa Arduino ensamblada artesanalmente

En la familia de placas oficiales Arduino, además del Arduino MEGA y UNO que se analizarán posteriormente, hay una gran variedad de tarjetas como Arduino Bluetooth o Nano preparadas para usar directamente en placas de desarrollo, (ver Figura 2).

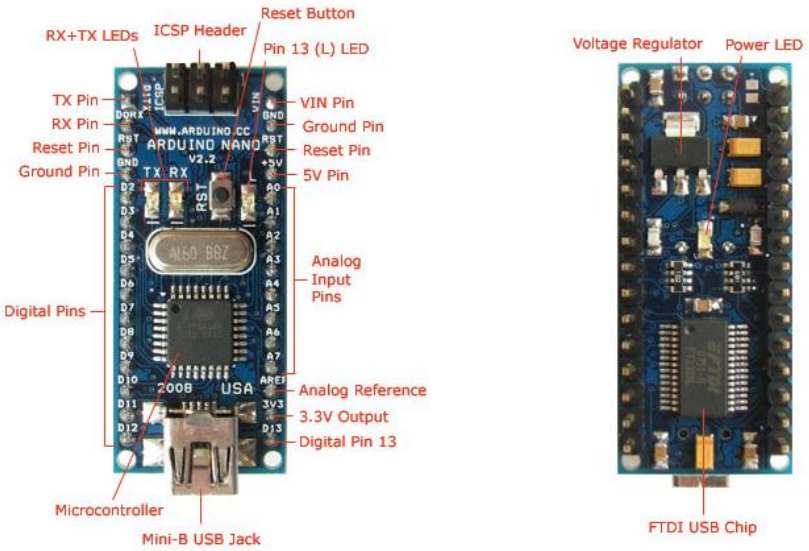


Figura 2 Arduino Nano

Se puede encontrar también placas con características únicas como Arduino Mini, la menor en tamaño; Mega ADK, extensión de Arduino Mega para conectarse directamente por USB a móviles Android; Arduino Lili, para aplicaciones sobre ropa [4], etc.

Complementariamente a las placas, hay módulos de expansión conocidos como 'shields' que añaden funcionalidades a las mismas, como la comunicación Wifi, Bluetooth, o GSM.

2.1.1. Hardware

En primer lugar se describirá la parte del sistema electrónico correspondiente al soporte hardware (Figura 3):

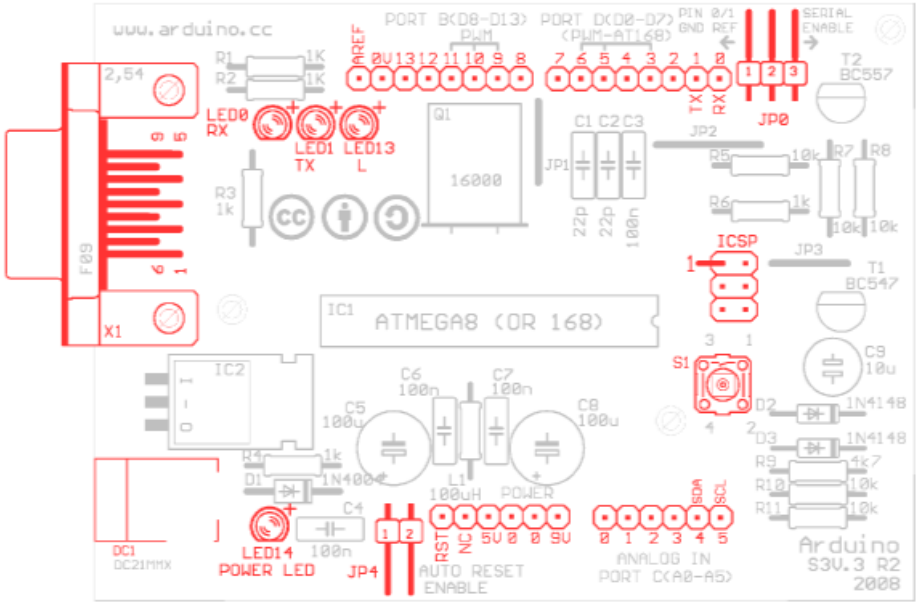


Figura 3 Esquema básico Arduino

Desde la parte izquierda, de arriba hacia debajo se observa con la marca X1 un conector serie que usa el estándar RS-232 que se utiliza para conectar la placa al ordenador y cargar los programas. A continuación, debajo se encuentra la

alimentación por un Jack centro-positivo que puede funcionar entre 7-20V aunque lo recomendado es mantenerse en un rango de 9-12V.

A ambos lados, en la parte superior e inferior de la imagen, están las regletas de pines de entradas/salidas digitales en la parte superior, y analógicas en la inferior.

Debajo de los pines digitales 9-10-11 se puede ver el cristal de 16Mhz, Figura 4.

Figura 4 Cristal 16Mhz

En la parte derecha, de arriba hacia debajo se tiene marcado como JP0, la parte que habilita/deshabilita la comunicación serie. Y a continuación el ICSP que se puede utilizar para grabar un gestor de arranque a través de un programador externo. Por último el interruptor S1 que resetea el Atmega.

2.1.2. Software

Arduino proporciona un entorno de desarrollo basado en Wiring, Figura 5, el cuál se trata de un framework abierto para la programación de microcontroladores y está constituido por un editor de texto para escribir código y debajo de éste un área de mensajes.

En la parte superior se tiene una barra de herramientas con las funciones básicas de verificar/compilar el programa, volcado en la placa Arduino, etc. La herramienta más útil será el monitor serie con el que se podrá estar en contacto continuo con la placa.

Figura 5 Entorno de desarrollo

Antes de volcar un programa, se deberá configurar la placa a usar y el puerto donde se encuentra conectada, para ello en la barra de menús seleccionar Tools > Board se escoge la placa, y después en Tools > Serial Port, el puerto en el que esté conectada (Figura 6).

Al seleccionar la placa que va a ser usada, en realidad se está configurando la CPU, velocidad, el fichero y la configuración del bootloader.

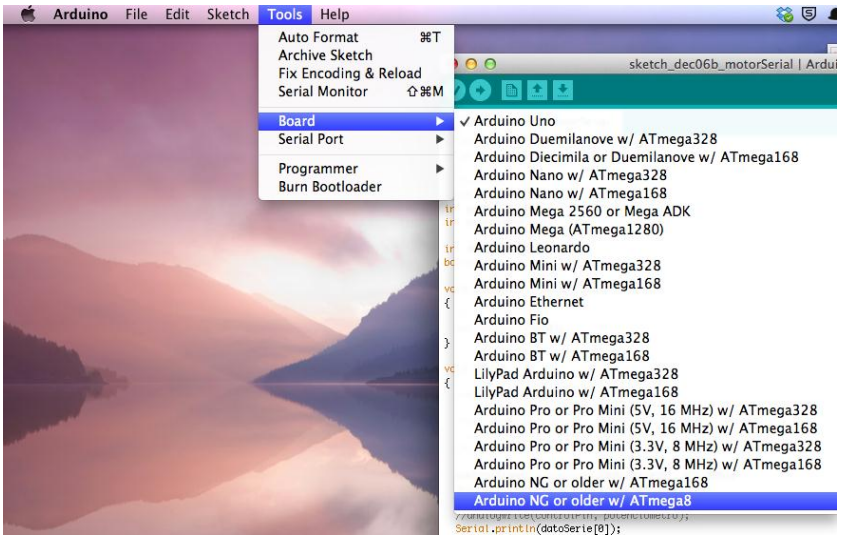


Figura 6 Selección de placa

Para volcar un programa, por defecto se utiliza el bootloader de Arduino, que es un programa precargado en el Arduino y permite subir el código sin utilizar hardware adicional. Este se inicia justo tras el reseteo de la placa y hace parpadear el LED 13. Tras esto se inicia el programa que haya sido cargado en el microcontrolador.

Estructura de un programa

Un programa para Arduino comienza incluyendo las librerías que van a ser utilizadas, esto es la forma de extender la funcionalidad de nuestros programas. Para utilizar una librería simplemente insertamos en nuestro programa la directiva:

```
#include <miLibreria.h>
```

Esto será descrito más tarde, cuándo al conectar el módulo de expansión Ethernet, que como se conecta a través del puerto SPI, se utilizarán las directivas:

```
#include <SPI.h>

#include <Ethernet.h>
```

Otras librerías de las que disponemos oficialmente son la librería EEPROM, para leer/escribir en memorias permanentes; Firmata, para comunicación con aplicaciones en el ordenador a través del protocolo Serial; Servo, para controlar servomotores; LiquidCrystal para controlar pantallas LCD; etc. [5]

A continuación aparece la función *setup()*, cuyo código sólo se ejecutará una vez. Es el lugar dónde inicializar y configurar variables y programas.

Después se encuentra la función *loop()*, que se ejecutará en un bucle continuo mientras la placa esté alimentada.

Tras la función *loop()*, pueden aparecer las funciones llamadas por el programa, si las hay.

De esta forma el esquema de un programa en Arduino, también llamado sketch, se construye a partir de las siguientes líneas:

```
#include <miCabecera.h>

void setup()

{

//

}


void loop()

{

 ... funcion_1();

}

int funcion_1()

{}
```

Otro punto destacable, además de la familia de placas Arduino, es que gracias a su popularidad y a su carácter abierto se ha creado una gran comunidad en torno a estas placas, que proporciona módulos, accesorios para su expansión, librerías y en general mucha documentación y ayuda para realizar las interfaces con otro software y hardware. Estas características de modularidad y expansión harán que esta plataforma sea una opción ideal para éste proyecto.

Entre las placas serán señaladas Arduino UNO, y Arduino MEGA, que son las que utilizadas; y entre los módulos de expansión, el módulo de conexión Ethernet, con el que se proporcionará la interfaz de uso móvil.

2.2. Tarjetas de adquisición de datos

En el mundo real nos encontramos con magnitudes físicas como la temperatura o la humedad en una habitación, el peso de los alimentos que compramos, o la velocidad a la que nos movemos. La forma de interactuar con estas variables ocurre en dos sentidos, por un lado las medimos a través de sensores, como por ejemplo un termómetro, y por otro actuamos sobre ellas mediante actuadores como un aparato de calefacción.

Estos sensores o actuadores, Figura 7, para informar de la magnitud que están midiendo proporcionan una señal en forma de tensión o corriente, en el caso de los actuadores, les es proporcionada. Por otra parte, para comunicar con un ordenador se deben seguir unos protocolos que le sean comprensibles, es decir, que la señal de tensión obtenida de un sensor no puede ser conectada directamente al ordenador, sino que debe ser conectada antes a un intermediario que la presentará al mismo u a otro dispositivo en una forma que pueda ser procesada. Este intermediario es la tarjeta de adquisición de datos, TAD o DAQ en inglés.

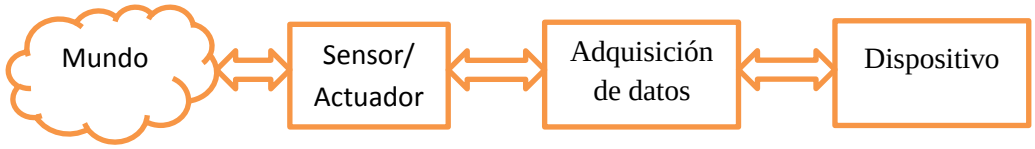


Figura 7 Esquema de adquisición de datos

La etapa de adquisición de los datos se puede subdividir en dos subetapas, una de acondicionamiento de las señales de los sensores y otra la Tarjeta de Adquisición de Datos en sí.

Actualmente en las aulas de laboratorio se encuentran instaladas las tarjetas Advantech 1711S (Figura 8) que consisten en una tarjeta PCI con 16 entradas/salidas analógicas y otras 16 entradas/salidas digitales. Las desventajas de estas tarjetas son, por una parte que para realizar las prácticas con ellas se depende del ordenador de sobremesa en el que se encuentran instaladas, y por otra su coste, actualmente unas 6 veces superior al de la placa Arduino Mega.

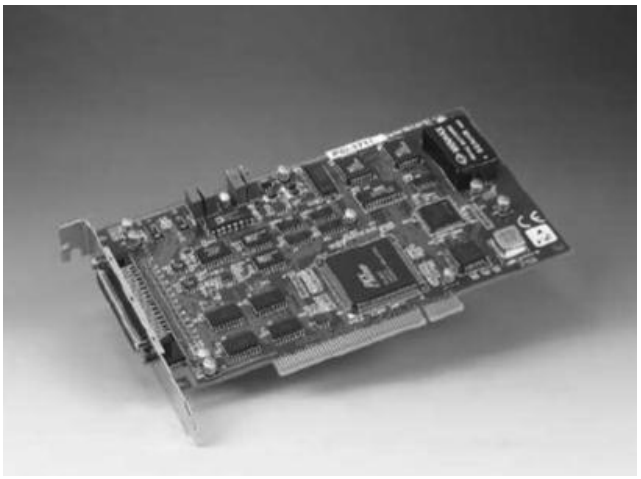


Figura 8 Advantech 1711S

2.2.1. Arduino mega

Esta placa utiliza el microcontrolador ATmeg1280, tiene 54 entradas/salidas digitales, -de las cuáles 14 pueden proporcionar una señal PWM- y 16 pines de entradas/salidas analógicas. En la Figura 9 se muestra la placa Arduino Mega empleada en este trabajo.

Figura 9 Arduino Mega

Entre otras características destacan:

- Intensidad por pin: 40mA
- Memoria Flash: 128KB
- EPROM: 4KB
- Velocidad del reloj: 16Mhz
- Tensión de funcionamiento: 5V
- Tensión de entrada recomendado: 7-12V
- Tensión de entrada límite: 6-20V

Alimentación

Puede realizarse por USB o con una fuente de alimentación externa, que a su vez puede ser mediante un conector macho centro-positivo de 2.1mm o mediante conexión directa a los pines Vin y GND (toma de tierra) de la placa que vienen marcados como POWER (pines en la parte inferior izquierda de la Figura 9).

Desde el pin 'Vin', si se está alimentando a través de un transformador en el conector centro-positivo, se puede acceder al voltaje que proporciona.

Junto a los pines Vin, y GND se encuentran los pines 5V y 3.3V que proporcionan una fuente de tensión a la dicha tensión indicada, siendo la corriente máxima soportada de 50mA.

Memoria

La placa incorpora 128KB de memoria Flash, de los que 4KB son utilizados por el gestor de arranque, y que para acceder a la memoria EEPROM debemos añadir las librerías EEPROM [6] a nuestro código.

Entradas/Salidas Digitales

Operan a 5V y pueden recibir una intensidad máxima de 40mA. Los pines del 0 al 13 pueden funcionar además de generador de onda PWM, y en el pin 13 hay además un LED integrado en la placa.

Los pines 0(Rx0)-1(Tx0), 14(Tx3)-15(Rx3), 16(Rx2)-17(Tx2), 18(Tx1)-19(Rx1) se pueden utilizar para transmitir/Recibir (Tx)/(Rx) datos por el puerto serie.

Los pines de interrupciones externas son el 2(INT0), 3(INT1), 18(INT5), 19(INT4), 20(INT3) y 21(INT2). Para utilizarlos se debe utilizar la función:

```
attachInterrupt(interrupción, función, modo);
```

dónde modo puede ser LOW (dispara la interrupción en cualquier momento que el pin está a nivel bajo), CHANGE(cuando ocurre el cambio de valor), RISING(cambio de valor Bajo a Alto), FALLING(cambio de valor Alto a Bajo).

Entradas/Salidas Analógicas

Las 16 entradas/salidas analógicas tienen una resolución de 10 bits, y por defecto miden de 0 a 5V. Este valor de referencia puede ser cambiado suministrando una tensión en el pin AREF.

2.2.2. Arduino uno

Esta placa, más modesta que la anterior, está basada en el microcontrolador Atmeg328. En la Figura 10 se muestra la placa más sencilla que proporciona Arduino.

Figura 10 Arduino UNO

Presenta 14 entradas/salidas digitales, de las que 6 pueden ser utilizadas para generar ondas PWM, y otras 6 analógicas. Salvo por la memoria, las características son muy similares a las del Arduino Mega:

- Intensidad por pin: 40mA
- Memoria Flash: 32KB
- EEPROM: 1KB
- Velocidad del reloj: 16Mhz
- Tensión de funcionamiento: 5V
- Tensión de entrada recomendado: 7-12V
- Tensión de entrada límite: 6-20V

Puede observarse que es muy similar al Arduino Mega, las características y pines de alimentación son idénticos; y respecto de la memoria, en esta placa el gestor de arranque ocupa 0.5KB frente a los 4KB del MEGA.

En el apartado de entradas/salidas, tienen también las mismas limitaciones de corriente que la placa anterior. Incorpora 14 entradas/salidas digitales, de las cuáles sólo pueden usarse para comunicación serie la 0(Rx) y 1(Tx); y para interrupciones externas el 2(INT0) y el 3(INT1).

2.2.3. Ethernet shield

Se trata de un módulo que proporciona conexión de Arduino con internet [7]. En la Figura 11 se muestra el módulo empleado para realizar la comunicación.

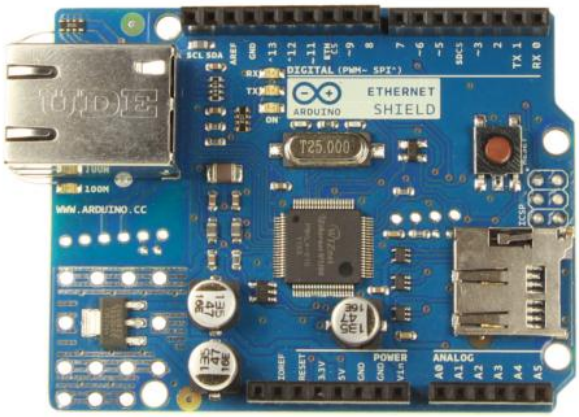


Figura 11 Módulo Ethernet

Este módulo se conecta por un lado a un router a través de un cable RJ45; y por el otro se conecta con Arduino a través del puerto SPI. Viene ya ensamblado con las dimensiones del Arduino, por lo que la conexión con él consiste en acoplarlo sobre la placa Arduino usada. Esta extensión provee de una dirección IP al Arduino y soporta hasta 4 conexiones simultáneas.

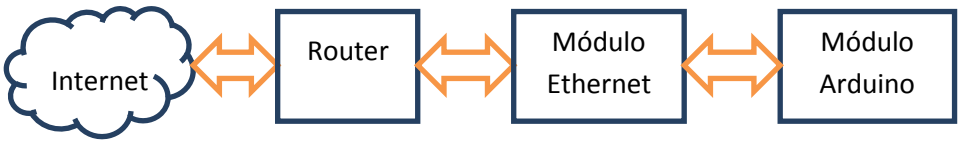


Figura 12 Esquema comunicación módulo Ethernet

El controlador Ethernet de éste módulo es el W5100 y utiliza la librería Ethernet. A éste módulo le puede ser añadido a su vez un módulo PoE (Power over Ethernet). Se ha utilizado el modelo sin PoE, dado que para las necesidades es suficiente y además más barato.

En el módulo de expansión hay un lector de tarjetas SD integrado que se puede utilizar para almacenar archivos, para utilizarlo se debe usar la librería SD.

Además éste módulo contiene varios LEDs de información:

- ON: Indica que al módulo le llega energía.
- LINK: Indica la presencia de una red, parpadea cuando se emiten/reciben datos
- RX: Parpadea al recibir datos
- TX: Parpadea al enviar datos
- 100M: Indica que está conectado a 100Mb/s en lugar de a 10Mb/s

2.3. Matlab

El enlace Matlab-Arduino se puede realizar de tres formas, dos específicas para Arduino y una general utilizando el puerto serie. Se analizarán por separado.

2.3.1. Arduino IO

Esta forma de realizar la interfaz Matlab-Arduino consiste en un programa que, cargado en el microcontrolador por medio del entorno de desarrollo de Arduino, hace que éste se comporte como un servidor que recibe las instrucciones a través del puerto serie, ejecuta los comandos, y si es necesario devuelve un resultado [8].

Este programa, ADIOSRV.pde (Analogic Digital Input Output SeRVer), viene con el paquete de Arduino IO, y está continuamente corriendo en el microcontrolador a la escucha de instrucciones.

Las ventajas de éste paquete son:

- Es compatible con Arduino UNO, Mega 2560, y Duemilanove.
- Es compatible con versiones Matlab anteriores.
- Es compatible con sistemas Mac OS y Linux.

Otra de las ventajas, que se analizará en detalle en el apartado "Estudio de Arduino como Tarjeta de Aquisición de Datos" se basa en que como el microcontrolador se encuentra corriendo siempre el mismo programa, tendremos una frecuencia de muestreo constante e independiente del modelo que estemos simulando.

Sin embargo, el coste es que al no estar ejecutando el modelo sino un servidor, necesitará en todo momento tener conectado un ordenador corriendo Matlab para que le indique los comandos.

Los pasos a seguir para ejecutar nuestro modelo, serán pues en primer lugar cargar el programa adiosrv.pde en el microcontrolador. A continuación, simular nuestro model, o programa Matlab que se conecta a Arduino.

La librería Arduino de Simulink contiene los bloques que permiten la lectura y escritura y otras funcionalidades (Figura 13).

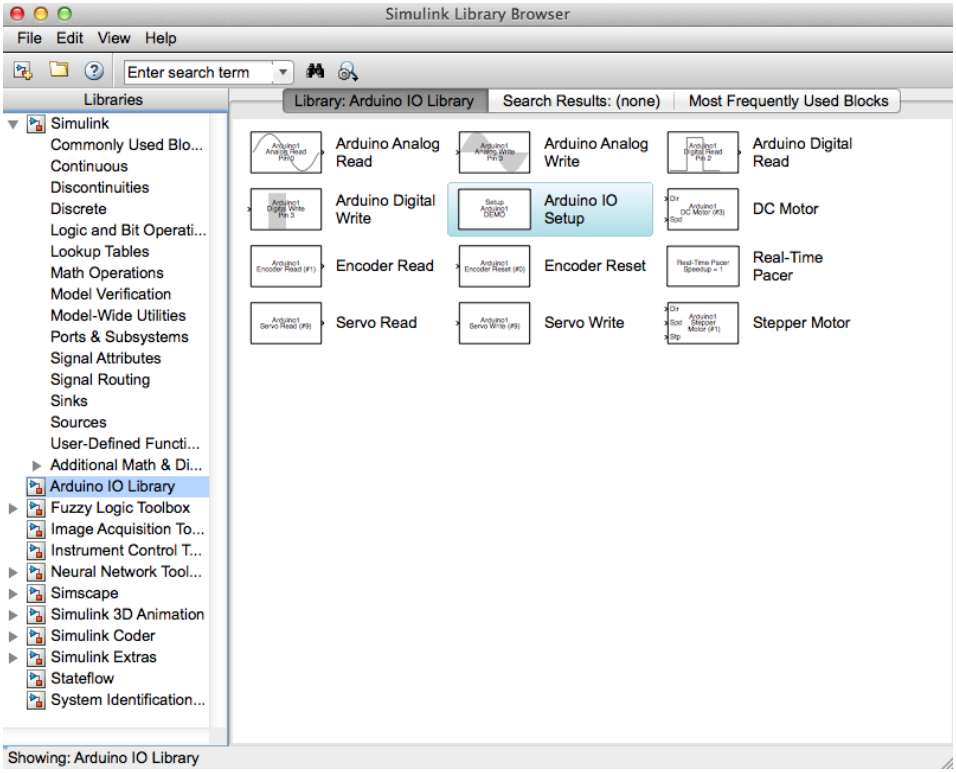


Figura 13 Librería Arduino IO

El siguiente punto analizará la librería del Soporte Arduino Simulink, y dado que ambas tienen un funcionamiento similar, en este punto, sólo será resaltado el bloque, "Arduino IO Setup", que es característico de éste método de comunicación Simulink-Arduino, ya que configura el puerto dónde se encuentra Arduino tal y como se muestra en la Figura 14.

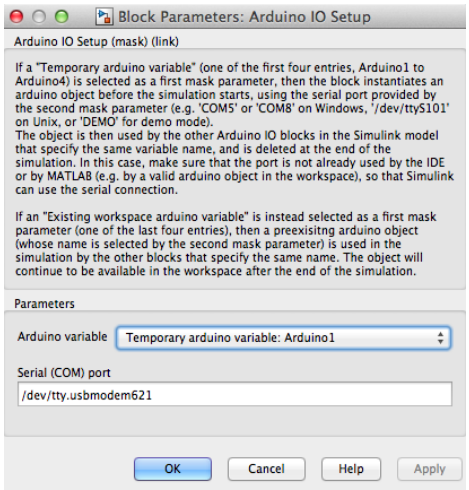


Figura 14 Arduino IO Setup

En esta pantalla se configura la ruta del puerto serie donde se encontrará nuestro Arduino ejecutando el servidor. La ruta será 'COMx' dónde x será el número que identifique el puerto en sistemas Windows, o '/dev/tty…' en sistemas UNIX.

2.3.2. Soporte Arduino Simulink

El soporte oficial de Simulink para Arduino, apareció en la versión Matlab 2012b, permite que los algoritmos desarrollados en Simulink, funcionen en Arduino.

Consiste en un traductor que convierte el esquema Simulink en código ejecutable que se carga en el microcontrolador para que lo ejecute éste directamente [9].

Sus características:

- El código corre directamente en el microcontrolador, luego puede ser desconectado del ordenador.
- Requiere al menos la versión de Matlab 2012b.
- Requiere Windows 7.
- El funcionamiento en modo externo requiere de la placa Arduino Mega 2560, que aunque es más potente, también es más cara. (50€ frente a los 25€ de Arduino UNO).

Para instalar el soporte se indica el destino de instalación (targetinstaller) y selecciona Arduino Simulink para instalar. En la librería se encuentran los bloques mostrados en la Figura 15.

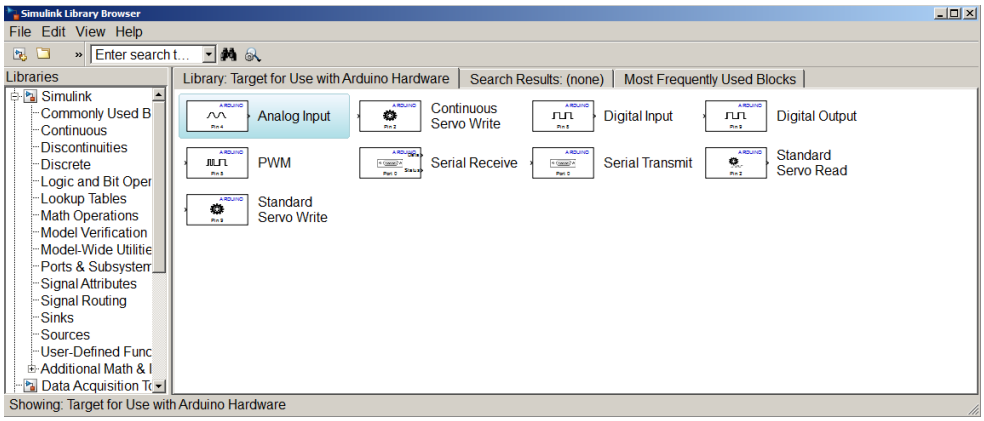


Figura 15 Libreria Simulink de Arduino

A continuación se describie en profundidad las características y limitaciones de los bloques que han sido usados: entrada analógica, digital, salida analógica, y digital.

Analog Input

El voltaje de una entrada analógica se mide en relación al voltaje de referencia de la placa Arduino y se obtiene su valor como 10-bit en un rango de 0-1023, siendo 1023 si la tensión medida es igual a la de referencia, y 0 si es igual a la de tierra. En la Figura 16 se muestra la pantalla de configuración de la entrada analógica.

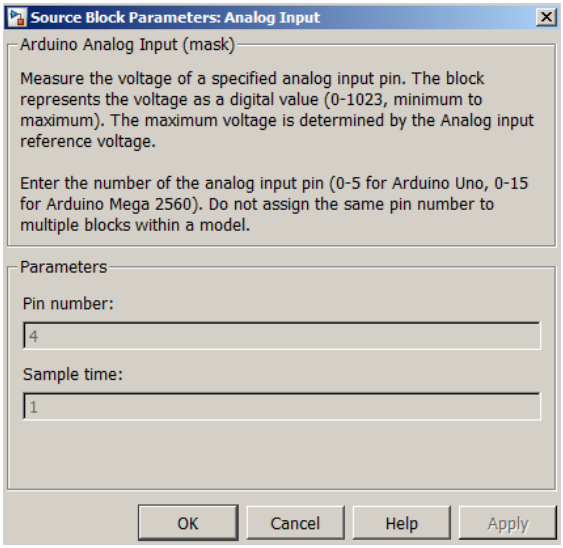


Figura 16 Bloque: Entrada analógica

El valor de referencia para la entrada de voltaje analógico por defecto es de 0 a 5V, sin embargo este valor puede ser cambiado para el modelo, pudiendo ser seleccionada la

referencia por defecto (5V), una interna de 2.56V, otra de 1.1V o una referencia externa. La referencia externa deberá conectarse al pin AREF de la placa, que se encuentra en el mismo lugar para las dos placas utilizadas, es indicado en la Figura 17.

Figura 17 Conexión AREF

En Simulink, para ajustar la referencia externa se deberá ir a Tools > Run On Target Hardware > Options… > Analog input reference voltaje que abrirá la pantalla mostrada en la Figura 18.

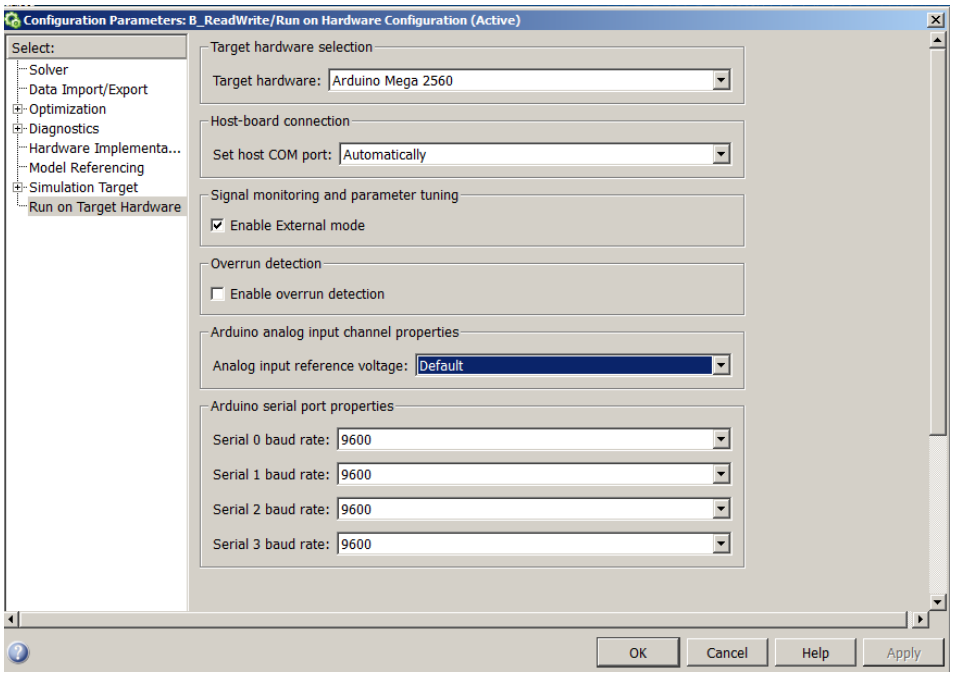


Figura 18 Configuración de la señal de referencia

Dentro del bloque se ha de configurar el pin del que leemos la señal, y la frecuencia de muestreo, que aunque en teoría puede ser un valor tan pequeño como 0.000001s, en

la práctica los valores a partir de 0.0001 congelan el ordenador debido a que utiliza el procesador para procesar más valores en menos tiempo. En la sección de pruebas del Capítulo Tercero se analizará cómo se comporta en este sentido.

Digital Input

Permite configurar el valor de una entrada digital, si está en estado bajo, adquiere un 0, y si se encuentra en estado Alto un 1. El tipo de dato recibido es un entero sin signo de 8 bits. Los parámetros número de pin y tiempo de muestro funcionan de la misma forma que en el bloque anterior.

En la Figura 19 se muestra la pantalla de configuración de las entradas digitales.

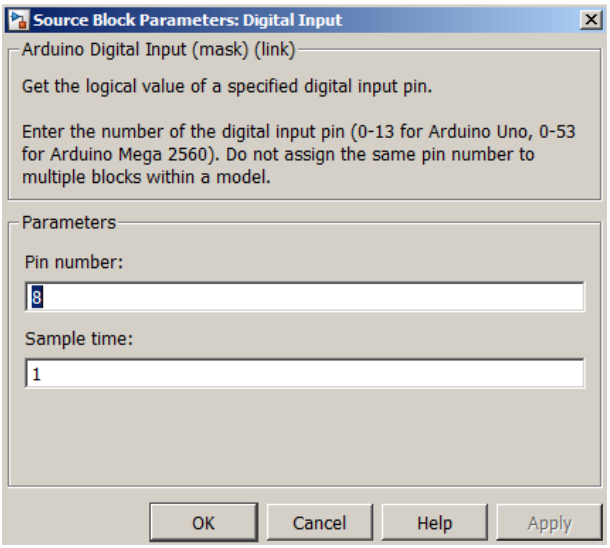


Figura 19 Bloque entrada digital

PWM

Este bloque genera una onda PWM (Modulación por Ancho de Pulso) en el pin especificado, con el porcentaje de ciclo indicado por el bloque anterior en un rango de 0 a 255, siendo 255 el 100% del ciclo a nivel alto. Este bloque hereda el tipo de dato del bloque precedente e internamente lo convierte a entero sin signo de 8 bits.

La pantalla de configuración de la señal PWM es mostrada en la Figura 20.

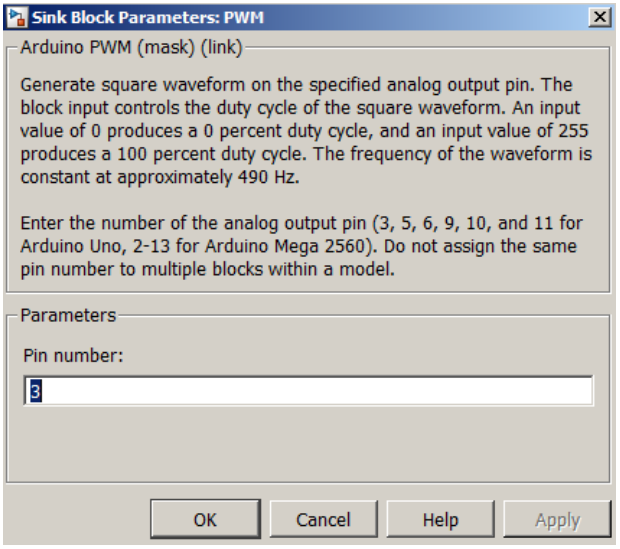


Figura 20 Bloque Salida PWM

Para el Arduino UNO, aunque entre los pines de salida están el 9 y el 10, lo cierto es que si el modelo contiene un bloque Servo no podrá utilizar ninguno de los dos.

Con el Arduino MEGA tiene una limitación más holgada y es que si el modelo contuviera más de doce bloques servo, no podría utilizar los pines 11 o 12 para generar un PWM.

Digital Output

Este bloque establece un pin a un valor lógico. El valor lógico '1' hace que la salida se ponga a nivel Alto, esto son 5V. El valor '0', 0V.

El bloque hereda el tipo de dato del bloque que lo precede y lo convierte a booleano. En la Figura 21 se muestra la pantalla de configuración de las salidas digitales, donde se indica el número de pin a utilizar.

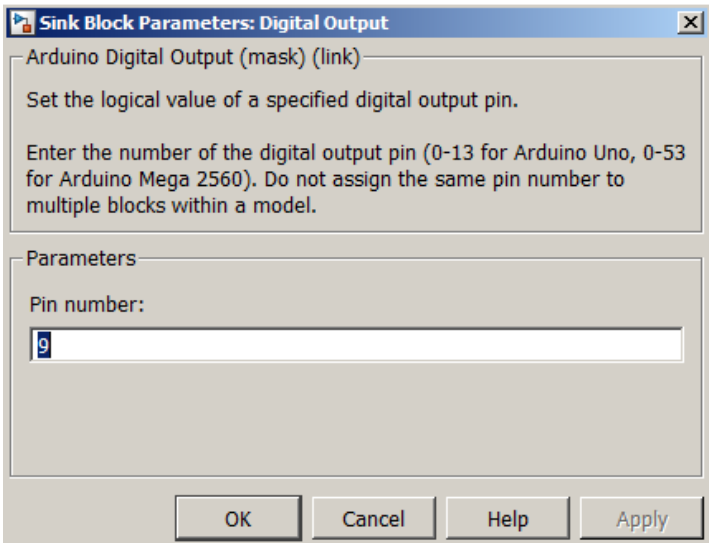


Figura 21 Bloque Salida Digital

De los bloques descritos y del resto de bloques se puede obtener más información y demostraciones de uso escribiendo: 'Target for Use with Arduino' en la consola de Matlab.

2.3.3. Comunicación serie

Esta se realiza por medio de los bloques de comunicación serie de Simulink. La primera ventaja es que es realizable con independencia de la placa Arduino que se encuentre conectada, y también de la versión de Matlab instalada. Por otra parte, al comunicarse con el programa que este corriendo en el microcontrolador, el momento en que los datos se reciben depende de dicho programa. De esta manera la frecuencia de adquisición de los datos dependerá del regulador implementado salvo que mediante un temporizador e interrupciones se obligue al microcontrolador a comunicarse en el tiempo preciso.

Los bloques utilizados vienen en la librería de Simulink y son 'Serial Configuration', 'Serial Send' y 'Serial Receive'. De acuerdo a cómo se establezcan los parámetros de comunicación del microcontrolador habrán de ser configurados los de Simulink.

En el caso desarrollado como ejemplo de envío/recepción de datos vía puerto serie (

Figura 22), se genera una señal de pulsos que convenientemente escalada se convierte a un tipo entero de 8 bits y se envía por el puerto serie. La recepción cuando detecta la llegada de datos, los convierte a un tipo para que sea mostrado en un bloque 'scope'.

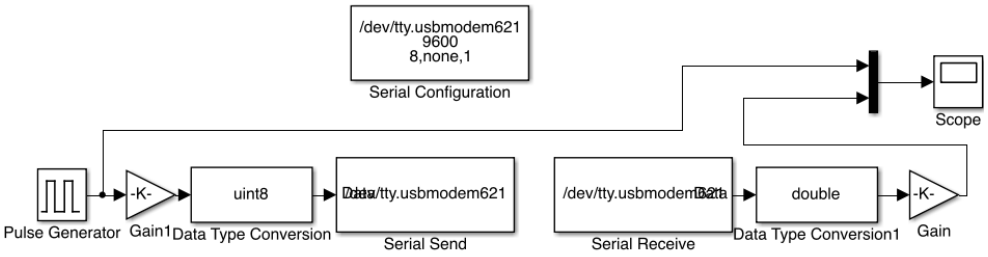


Figura 22 Comunicación serie Matlab-Arduino

La configuración de los parámetros del puerto serie es:

```
Puerto de comunicación: /dev/tty.usbmodem621
Baudios: 9600
Bits por dato: 8
Paridad: no
Bit de parada: 1
Orden: "Fila india"
```

Por su parte, el programa cargado en el microcontrolador leerá una señal analógica simulada por medio de un potenciómetro y enviará dicha lectura a través del puerto serie. Comprobará si ha llegado información en el puerto serie, y si la hay, actualizará un actuador –simulado con un LED- con dicho valor. En la Figura 23 se muestra el Flujograma empleado para realizar la comunicación serie y en el Anexo B.1 se encuentra su código.

Este programa es cargado en el microcontrolador, quién tras configurar el puerto serie realiza la lectura de un sensor, envía su valor al programa que se está ejecutando en el ordenador, si recibe la orden, actualiza el valor de un actuador. En las pruebas, la señal proporcionada por el sensor fue simulada con un potenciómetro mientras que el actuador lo fue con un LED. El sistema funciona de la forma esperada.

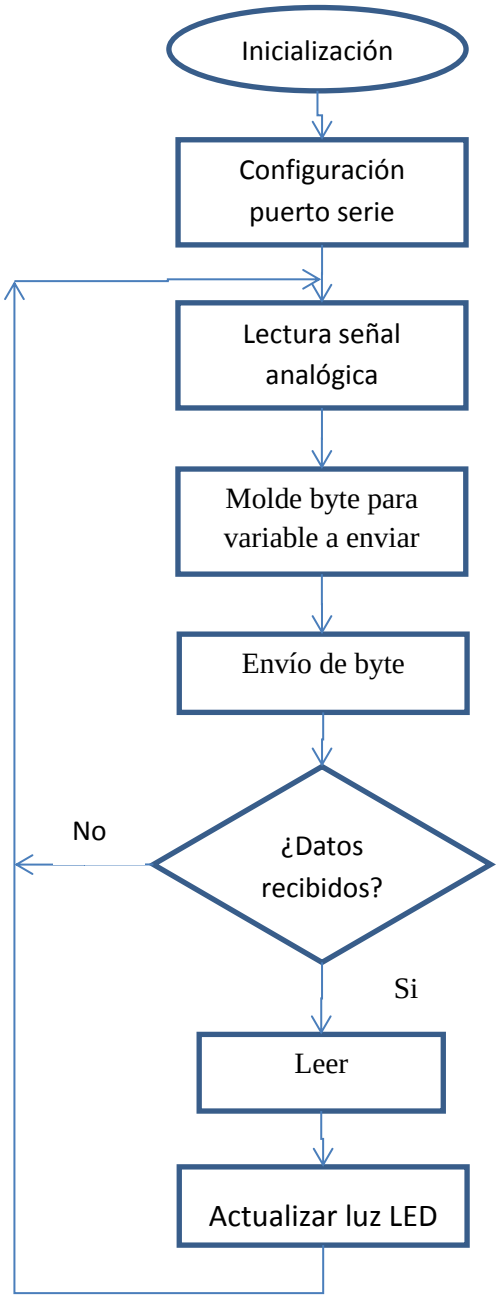


Figura 23 Flujograma comunicación serie

2.4. Interfaz Arduino Android

Actualmente, los métodos oficiales para realizar la conexión entre una aplicación Android, y un microcontrolador Arduino que se encuentran documentados en la wiki de Arduino [13]:

- Bluetooth: Amarino
- Audio: Androino
- Cable USB

Además, aunque no documentado oficialmente, tenemos la posibilidad de programar la placa Arduino como un servidor, y realizar una aplicación cliente que se conecte a Arduino como veremos en el capítulo de Interfaz con Arduino.

2.4.1. Bluetooth: Amarino

Se trata de una herramienta para conectar dispositivos Android a placas Arduino por medio de la tecnología bluetooth [10], Figura 24.

Se basa en una aplicación Android llamada “Amarino” y una librería Arduino llamada “MeetAndroid” y soporta sólo un módulo bluetooth, aunque da referencias de otros 3 módulos que pueden ser utilizados de forma más complicada.

Figura 24 Módulo Bluetooth Mate Gold

El módulo bluetooth en cuestión tiene un precio de 60€, y en un espacio abierto sin interferencias puede llegar a tener un alcance de 100m, que en edificios y en un entorno real se supone quedan en 20m.

En principio esta opción fue descartada por el precio del módulo bluetooth.

2.4.2. Audio: Androino

Se trata de una librería que realiza la comunicación Android-Arduino a través del cable de conexión de audio [11], Figura 25.

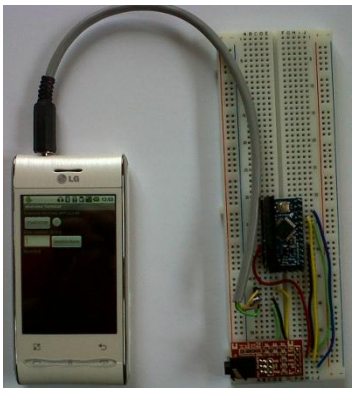


Figura 25 Conexión por cable audio

El proyecto actualmente permite el envío y recepción de enteros del 0 al 31.

Esta opción junto con la siguiente fueron descartadas debido a que requieren en todo momento una conexión por cable con el dispositivo Android, lo que restringe la movilidad.

2.4.3. Cable USB

Se trata de una librería para conectar cualquier hardware por USB a un dispositivo Android. En Arduino existe una placa llamada Arduino Mega ADK que es compatible con Android [12]. Es similar a la placa Arduino Mega. La diferencia está en que al

contrario que cuándo se conecta un accesorio a un ordenador y el ordenador provee la energía, como un teclado, cuando se conecta un accesorio a Android, el accesorio debe proveer la energía para todo el sistema.

3. Diseño de la Tarjeta de Adquisición de Datos

3.1. Descripción del sistema.

El sistema sobre el cuál se desarrollarán las pruebas consta de un motor de corriente continua, una etapa de acondicionamiento de las señales proporcionadas desde y hacia el motor, la tarjeta de adquisición de datos y el ordenador o dispositivo que procesa esta información. El esquema del sistema se muestra en la Figura 26.

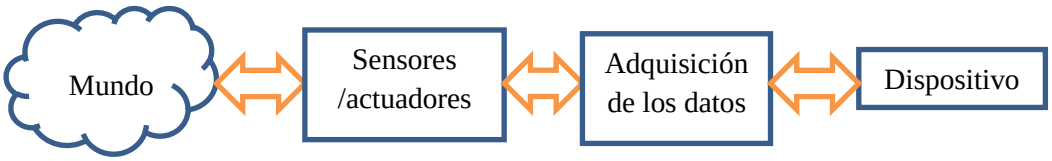


Figura 26 Esquema descriptivo del sistema

En el trabajo con la tarjeta y experimentación, el mundo consiste en una de las plataformas experimentales de laboratorio, formada por un motor de continua con el

correspondiente circuito de alimentación CA-CC, un rectificador de onda, un filtro y un encoder incremental que permite conocer la posición del motor (Figura 27).

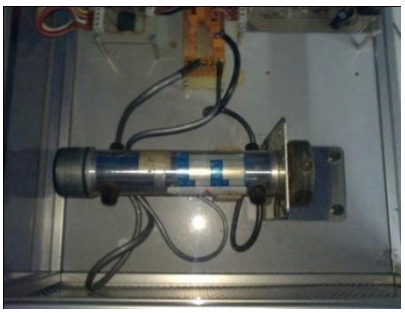


Figura 27 Motor CC

La interfaz física de control de la maqueta de la figura anterior consta del siguiente panel, descrito en la Figura 28.

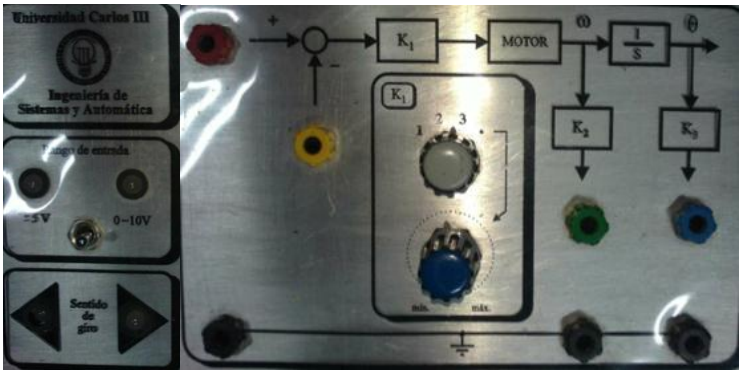


Figura 28 Panel de control de la maqueta

El motor permite dos modos de funcionamiento, a partir de una entrada en un rango ±5V o 0-10V. Este modo se selecciona en la parte izquierda del panel, donde se puede ver el interruptor de selección del rango de entrada. La selección del conmutador depende del rango de la señal conectada.

A continuación se describen el resto de bornes de control:

- Borne rojo: Es la entrada al sistema, a él se conecta la señal de referencia proporcionada por la tarjeta de adquisición de datos, previo acondicionamiento.
- Borne amarillo: Se utiliza para comparar la señal referencia. En este trabajo los reguladores se implementan por software, por este motivo y para evitar interferencias, este borne se conectará a tierra.

- Borne verde: Es la conexión que permite conocer la velocidad de giro del motor.
- Borne azul: Proporciona la posición resuelta por el encoder.
- Bornes negros: Masa de la maqueta.

Además en el centro del panel aparece una rueda que permite ajustar una constante K1 que multiplica la señal de referencia.

3.2. Especificaciones

A la entrada de la maqueta, la señal de referencia utilizada es una señal en tensión continua, de 0-10V. Para controlar esta señal de referencia se parte de una señal PWM del módulo Arduino de 0-5V que ha de ser acondicionada.

A la salida, se tiene una señal que puede llegar a variar entre [-10, +10]V y que será necesario acondicionar a la entrada analógica del Arduino entre [0, 5]V.

3.3. Acondicionamiento de la señal

Es necesario distinguir entre el acondicionamiento de la señal de entrada a la maqueta, y el de la señal de salida de la misma.

A continuación se describirán el acondicionamiento de la señal de entrada y de salida de la maqueta empleada como plataforma experimental.

3.3.1. Acondicionamiento de la señal de entrada a la maqueta

El microcontrolador crea la señal PWM en un rango de [0-5] V, por lo tanto la señal deberá ser filtrada para obtener una señal continua. Posteriormente deberá ser amplificada al rango [0-10] V, rango de entrada de la maqueta. En la Figura 28 se muestra el esquema de acondicionamiento de esta señal.

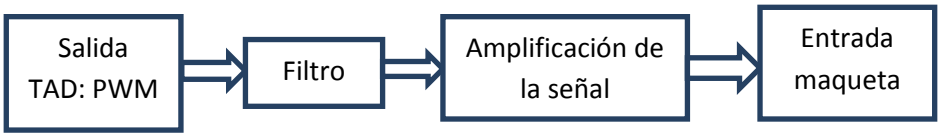


Figura 29 Esquema acondicionamiento entrada a maqueta

3.3.2. Acondicionamiento de la señal de salida de la maqueta

Para acondicionar la señal de salida serán necesarias dos etapas. En la primera mediante un divisor de tensión, se ajusta el rango de variación de la señal de salida de la maqueta al rango de entrada del microcontrolador. En la segunda etapa, con un operacional en configuración de sumador, se añade un offset a la señal para que quede entre [0, 5] V.

En la Figura 30 se muestra el esquema de acondicionamiento de esta señal.

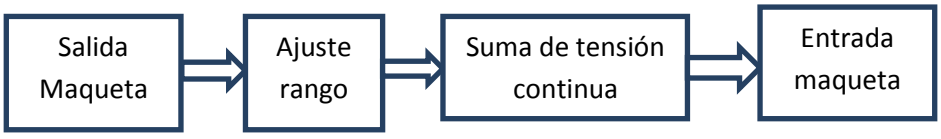


Figura 30 Acondicionamiento de la salida de la maqueta

3.4. Diseño electrónico

A continuación se describe el sistema electrónico desarrollado. Se han implementado tres bloques, un filtro, y dos etapas de potencia.

3.4.1. Filtro

La conversión de la señal PWM en una señal analógica continua es realizada por medio de un filtro paso bajo sencillo (Figura 31).

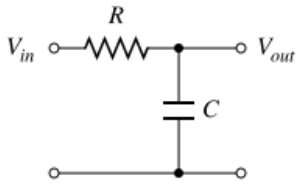


Figura 31 Filtro RC

Cuando se aplica una tensión a la entrada de R, el condensador comienza a cargarse hasta que se encuentra cargado y deja de conducir corriente. Si la carga tiene una impedancia muy alta, la tensión de entrada coincidirá con la de salida. Como un condensador se carga con la corriente variable, mientras la corriente de alta frecuencia será filtrada adecuadamente, la parte de continua pasará limpia. Sin embargo lo que ocurre entre la baja y la alta frecuencia depende de la constante de tiempo.

El problema aparece en el tiempo de respuesta y la tensión de rizado. Ya que un tiempo de respuesta pequeño implicará una tensión de rizado alta y viceversa, como se observa en la Figura 32, que muestra la rectificación de una onda senoidal.

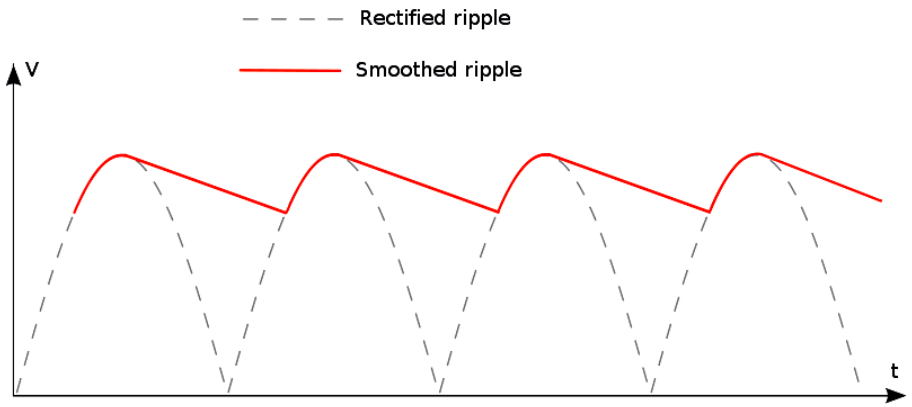


Figura 32 Rectificación de onda senoidal

Idealmente, se elegiría un filtro que dejase pasar la frecuencia baja con ganancia 1 – se trata de un filtro pasivo- para rápidamente dejar caer la ganancia. Para ello sería necesario preparar un filtro con varios polos. Sin embargo esto volvería el circuito más complejo de lo necesario para su propósito. Además de costoso. Como se verá más adelante, un filtro con un polo resuelve bastante bien el problema.

Así se busca un compromiso entre un rizado bajo y una frecuencia de corte también baja. Y para ello se va a utilizar un filtro paso bajo pasivo con polo simple.

Para minimizar el rizado optaremos por un condensador con relativa gran capacidad, el mayor de los condensadores cerámicos es de 0.1 µF, el siguiente con el que podemos probar sería el cerámico multicapa de 1µF. La alternativa sería utilizar un condensador electrolítico que tiene mayor capacidad. Sin embargo debemos recordar una mayor capacidad implica un mayor tiempo de respuesta.

El siguiente paso será elegir una frecuencia de corte baja, pongamos 10Hz. Esto resulta en seleccionar los valores R=15kΩ, C=1µF para nuestra resistencia y condensador respectivamente.

La función de transferencia del filtro quedará:

$$G(s) = \frac{\frac{1}{s \cdot RC}}{1 + \frac{1}{s \cdot RC}} = \frac{66.67}{1 + 66.67}$$

Cuyos parámetros frecuencia de corte, tensión de rizado y tiempo de respuesta son:

- Frecuencia de corte = 10.61Hz

- Tensión de rizado = $V_{pp} = 0.17\text{V}$
- Tiempo de respuesta (tiempo en alcanzar el 90% de la señal) = 0.034s

En la Figura 33 se muestra la respuesta en frecuencia de dicho filtro.

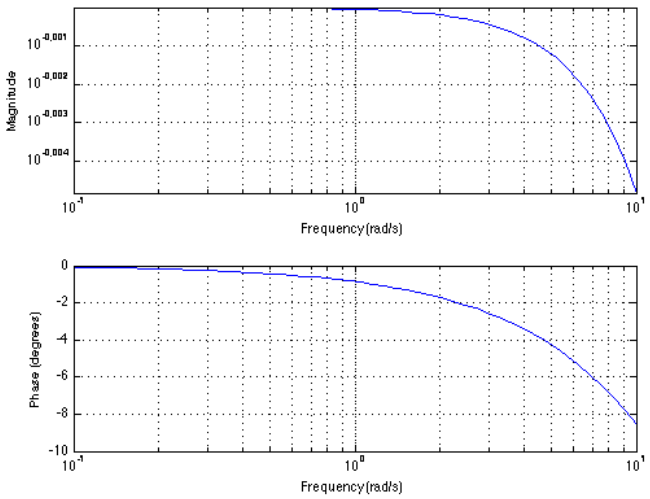


Figura 33 Respuesta en frecuencia del filtro

En la Figura 34 se muestra la simulación de la respuesta a escalón con el PWM cubriendo el 90% del ciclo, que corresponde a 4.5V. Como se puede observar el acondicionamiento de la señal es el esperado.

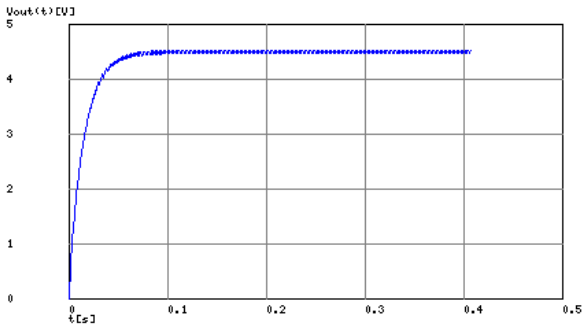


Figura 34 Simulación respuesta filtro a entrada escalón

En la Figura 35 se muestra un solo ciclo de la señal PWM al 50% y la señal acondicionada tras el filtro, mostrada como válida para el trabajo de control.

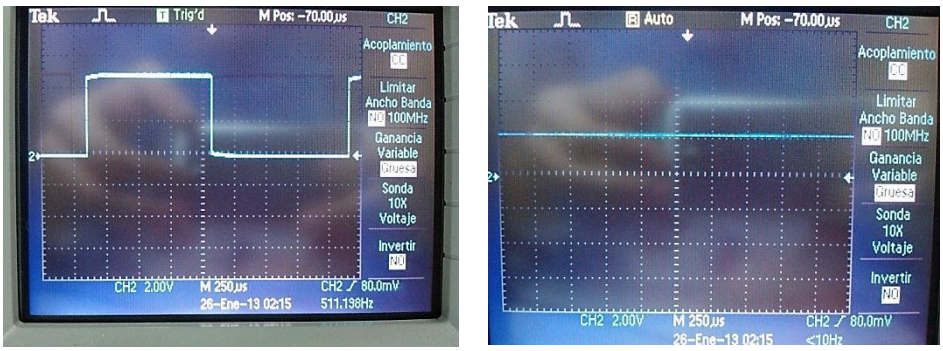


Figura 35 Filtro: Ciclo PWM

En la Figura 36 se observa como la señal PWM produce una onda cuadrada y como queda tras el filtro. En este caso el rizado es algo mayor.

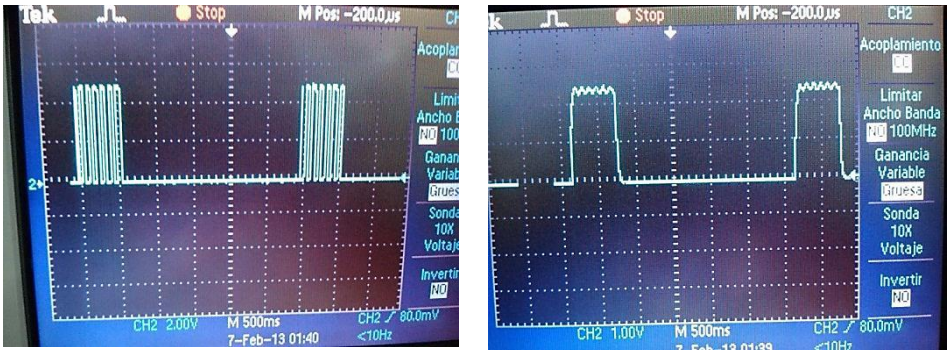


Figura 36 Filtro Onda cuadrada

La última prueba consiste en la producción de una señal sinusoidal, tal y como se muestra en la Figura 37.

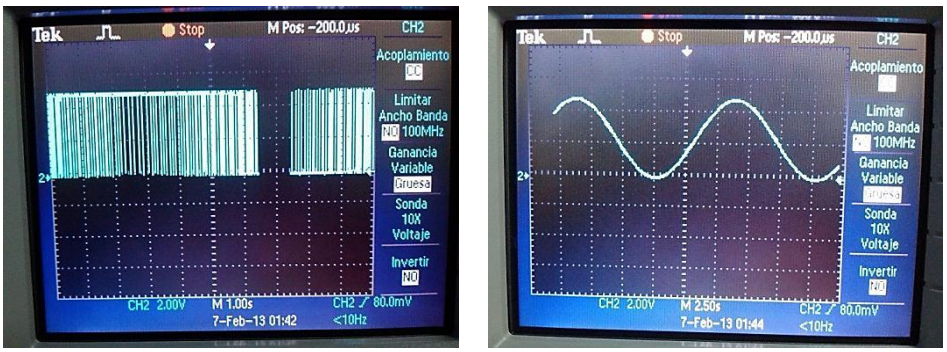


Figura 37 Filtro. Onda Senoidal

Cómo se observa, la respuesta del filtro es adecuada.

3.4.2. Amplificación de la señal

Se realizará mediante un amplificador operacional en configuración no inversora colocado a la salida del filtro (Figura 38).

Cómo la maqueta funciona de 0-10V y la salida PWM se encuentra en el rango 0-5V, ha de ser multiplicada por dos.

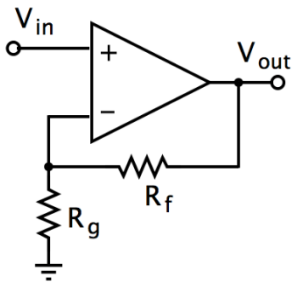


Figura 38 Amplificador no inversor

Con esta configuración el cálculo de la tasa de salida queda:

$$V_{salida} = \left(1 + \frac{R_f}{R_g}\right) V_{entrada} \longrightarrow \{R_g = R_f\} \rightarrow V_{salida} = 2 \cdot V_{entrada}$$

Uniendo el filtro con el amplificador, el siguiente esquema electrónico mostrado en la Figura 39, esquema final implementado.

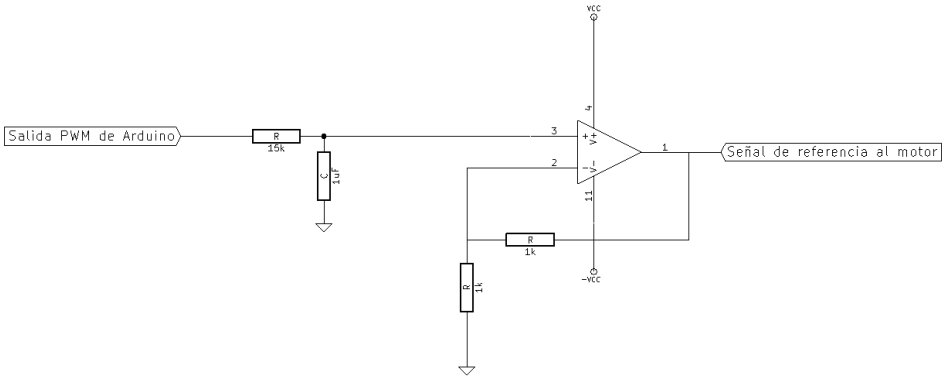


Figura 39 Acondicionamiento de la señal PWM

En la Figura 40 se observa la secuencia completa de acondicionamiento de la señal de referencia a la maqueta: un ciclo de la señal PWM, la misma señal filtrada y en último lugar, la señal amplificada.

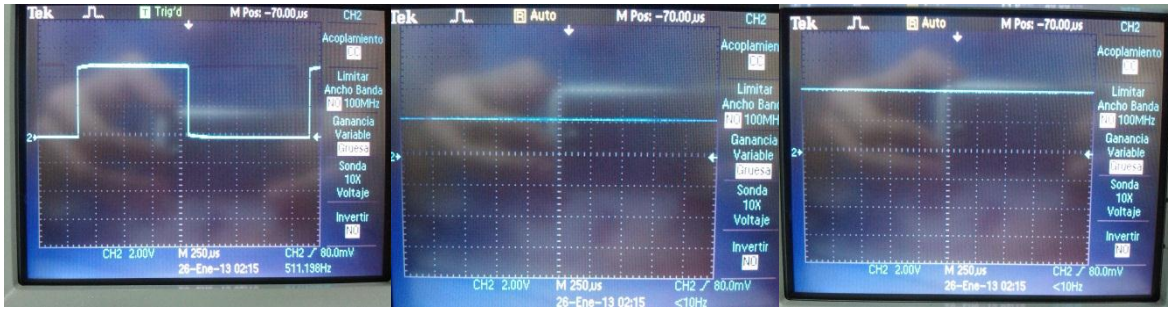


Figura 40 Acondicionamiento salida PWM -> Salida Filtro -> Amplificación: Señal de referencia de la maqueta

3.4.3. Acondicionamiento de la señal de salida del motor

En la Figura 41 se muestra el esquema electrónico implementado para el acondicionamiento de la señal de salida de la maqueta.

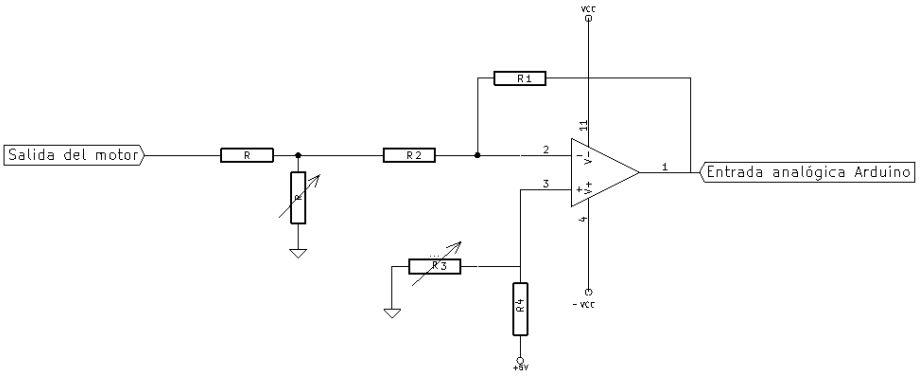


Figura 41 Esquema electrónico acondicionamiento de la señal de salida de la maqueta

La señal de salida del motor pasa a través de un divisor de tensión para ajustar su rango de variación al de los 5V que acepta el microcontrolador. Tras él, se tiene una tensión que varía entre [-2.5, 2.5]V. Se coloca un potenciómetro para facilitar el ajuste.

A continuación se le suma un offset de tensión continua para situar la señal entre los [0, 5]V que serán leídos por el microcontrolador.

El valor de la entrada analógica se calculará a partir del circuito como:

$$V_1 = -\frac{R_1}{R_2}V_{Salida\ maqueta} + \left(1 + \frac{R_1}{R_2}\right)\frac{R_3}{R_3 + R_4} \cdot V_{ref\ 5V} \rightarrow$$
$$\{R_1 = R_2; 3R_3 = R_4\} \rightarrow$$

$$V_1 = -V_{Salida\ motor\ entre\ [-2.5,2.5]} + 2.5V$$

Por tanto se concluye que:

$$V_{Entrada\ Arduino} = \frac{1}{4}(-V_{Salida\ maqueta} + 2.5V)$$
$$\rightarrow \begin{cases} Caso\ 1: V_{salida\ maqueta} = -10V \rightarrow V_{Entrada\ Arduino} = 5V \\ Caso\ 2: V_{salida\ maqueta} = 10V \rightarrow V_{Entrada\ Arduino} = 0V \end{cases}$$

La primera implementación del circuito es la mostrada en la Figura 42.

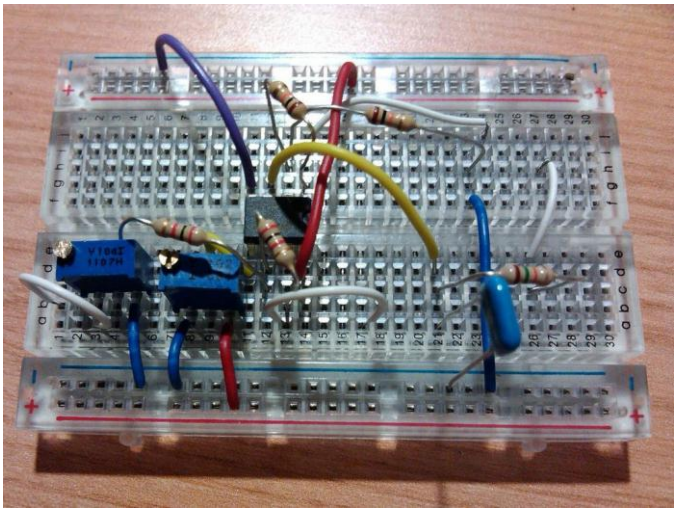


Figura 42 Implementacion de las etapas de acondicionamiento sobre protoboard

La siguiente versión se implementó sobre una placa de baquelita con las dimensiones de la placa de Arduino Mega, con el objeto de que quedará como un módulo de expansión (Figura 43).

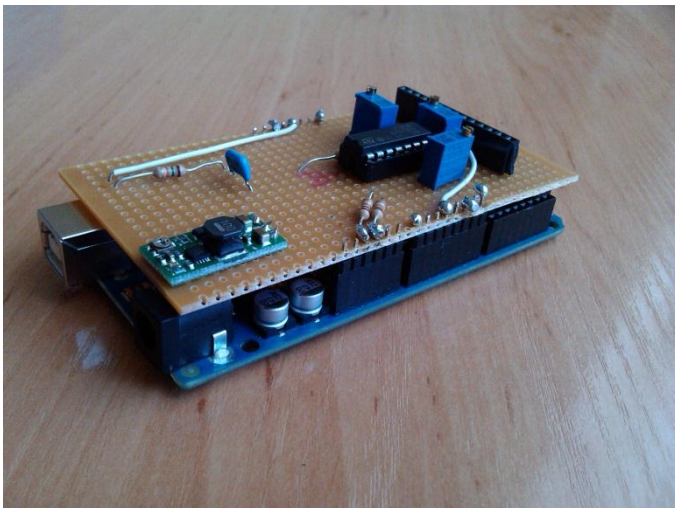


Figura 43 Implementación sobre baquelita

En el siguiente apartado se mostrarán las pruebas realizadas con el conjunto.

3.5. Pruebas del circuito electrónico

A continuación se describen los experimentos realizados para la determinación del funcionamiento de las 3 formas analizadas de implementación Simulink-Arduino.

3.5.1. Prueba realizada con Arduino IO

Las pruebas realizadas con el paquete de Arduino IO, fueron realizadas sobre el sistema operativo Mac OS X, con la placa Arduino UNO.

El objetivo de la primera prueba trata de simular un sistema formado por un sensor y un actuador de manera que según el estado del sensor se comporte el actuador (Figura 44). A través de éste test será comprobada la conectividad Arduino-Matlab.

Para ello nuestro sensor será simulado por una resistencia variable mientras que el actuador consistirá en un LED. Se trata de una prueba sencilla pero fundamental, que confirmará la conectividad tanto en lectura como en escritura.

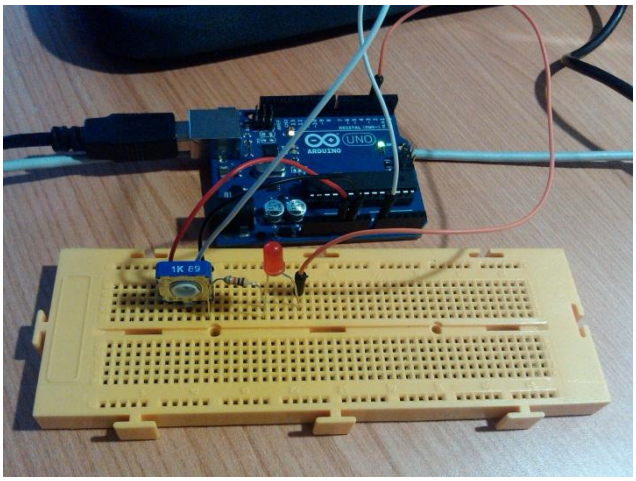


Figura 44 Circuito para el test de conectividad

En la Figura 45 se representa en rojo la señal leída por el potenciómetro y en azul, la señal enviada al LED.

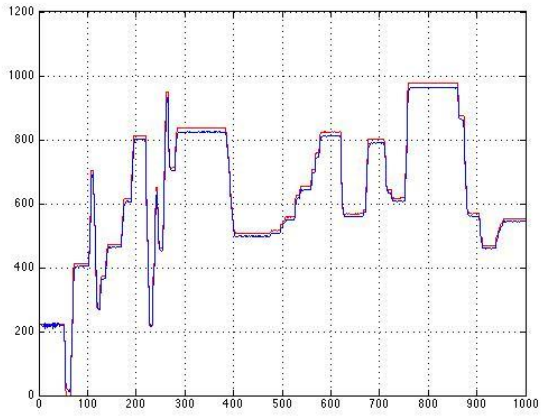


Figura 46 Test de comunicación (Tensión frente a tiempo)

Una vez realizado el test de comunicación se tratará de cerrar el bucle de control añadiendo una resistencia fotosensible (PDV-P9203, Anexo A.1) y haciendo una aproximación lineal a su respuesta. Esta resistencia variable será leída en la entrada analógica 3.

La función Matlab empleada (Anexo B.2), recibe cómo parámetro el número de muestras a tomar como se muestra en el flujograma de la Figura 47.

Tras llamar a la función con un parámetro 'n' referente al número de muestras a tomar, se produce la inicialización de las variables y la conexión. Se tomará una muestra tras otra hasta alcanzar las 'n' muestras solicitadas. En cada una de las muestras proporcionadas por el sensor de luz, se compara ésta con la señal de referencia y se ajusta la señal de control. Finalmente se dibuja el gráfico.

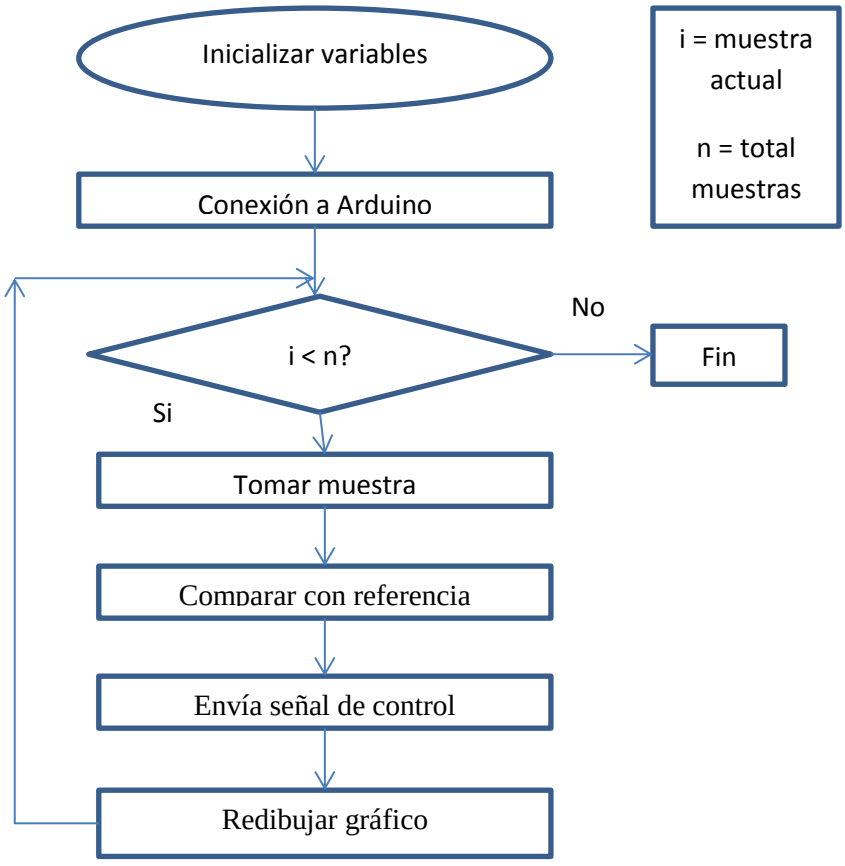


Figura 47 Flujograma control mediante Arduino IO

El resultado puede observarse en la Figura 48, dónde la línea roja muestra la señal de referencia, generada por el con ayuda del potenciómetro; mientras que la línea azul se trata de la señal recogida por el LDR.

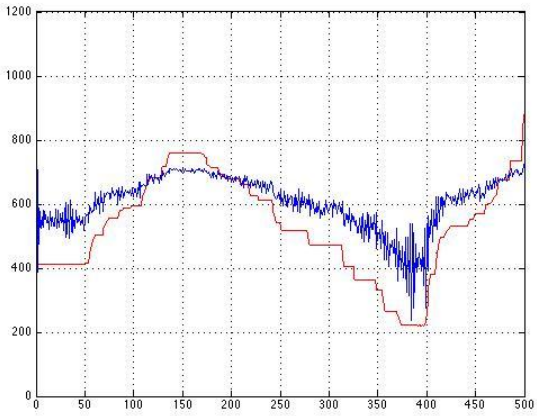


Figura 48 Control luz. Luminosidad frente a tiempo

Como puede observarse, aunque con cierto retraso y margen de error en general se produce un seguimiento de la luz emitida por el LED de acuerdo a una señal de referencia y a la realimentación proporcionada por el sensor de luz.

En la Figura 49 figura se pueden observar una onda senoidal y otra cuadrada adquiridas y representadas mediante un código similar al anterior, tomando unas 100 muestras por segundo.

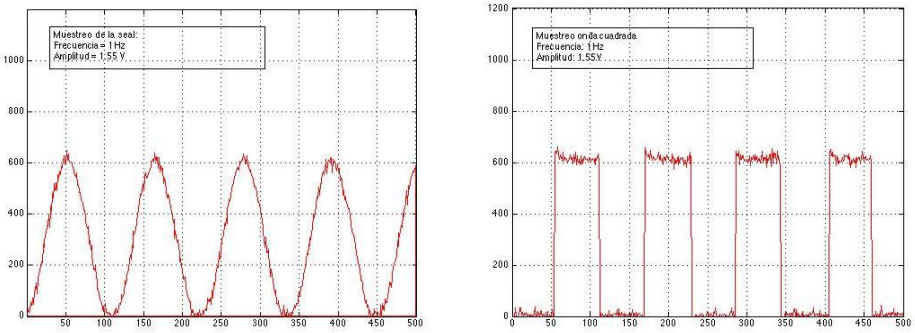


Figura 49 Muestreo de señales mediante Arduino IO. Tensión frente a tiempo

Se concluye que puede utilizarse para la mayoría de los sistemas de control.

3.5.2. Soporte Simulink con Arduino

Las primeras pruebas realizadas tratan de mostrar el funcionamiento y la carga de modelos de Simulink en Arduino. El objeto de las mismas es probar la generación de distintos tipos de ondas con el microcontrolador.

Las pruebas se han realizado con un LED conectado a través de una resistencia de 1kΩ en la salida digital 3 del Arduino, como puede observarse en la Figura 50.

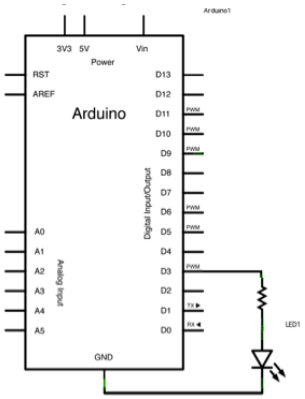


Figura 50 Esquema conexión pruebas Simulink

Se ha controlado el LED con las siguientes características:

- Modo Externo
- Tiempo de integración infinito

Con el microcontrolador se ha generado una onda cuadrada, una señal senoidal y un escalón.

Onda Cuadrada

El esquema de Simulink implementado genera una onda cuadrada en uno de los pines de Arduino mediante una señal PWM como se observa en la Figura 51.

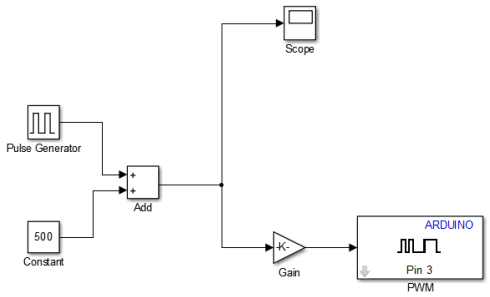


Figura 51 Esquema prueba de generación de onda cuadrada

Este programa que genera una onda cuadrada (Figura 52, izquierda) produce el parpadeo intermitente del LED de acuerdo a la señal recibida (Figura 52, derecha).

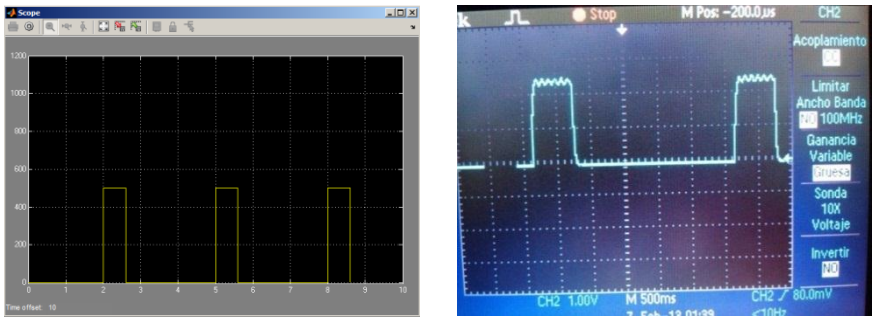


Figura 52 Onda Cuadrada

Onda Senoidal

El esquema de Simulink implementado genera una onda senoidal en uno de los pines de Arduino mediante una señal PWM como se observa en la Figura 53.

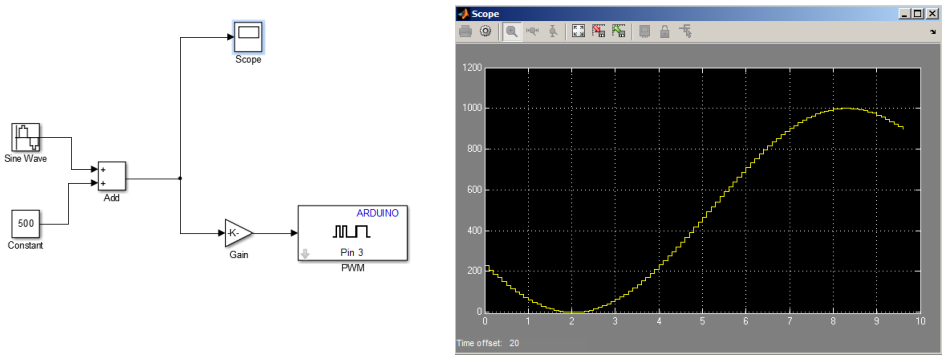


Figura 53 Generación senoide

Esta señal produce ahora un cambio continuo de la iluminación ofrecida por el LED desde el estado apagado hasta el de luz máxima.

Onda Escalón

Por último se implementa la generación de una señal escalón en uno de los pines de Arduino mediante una señal PWM como se observa en la Figura 54.

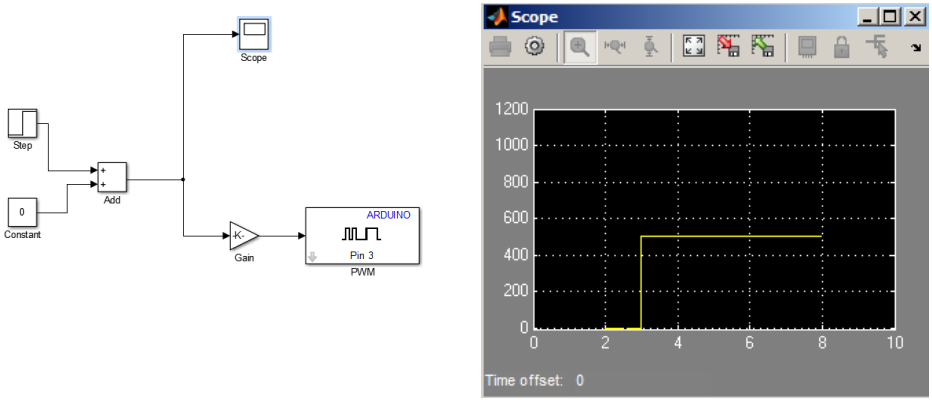
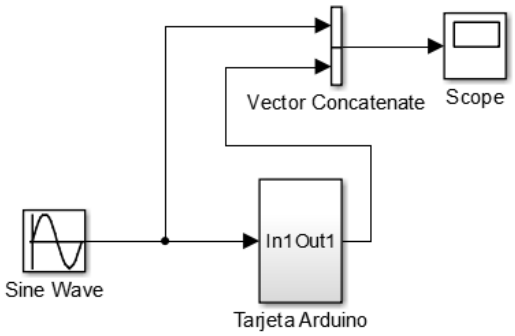


Figura 54 Generación onda escalón

En este caso se obtiene el encendido puntual del LED a los 3 segundos de acuerdo a la señal escalón.

3.5.3. Control de motor de corriente continua en bucle abierto. Prueba de lectura de acondicionamiento de la entrada al Arduino

Esta prueba consiste en la implementación del siguiente esquema en Simulink, donde en el que se genera una señal de referencia senoidal y se lee la velocidad para comprobar que el acondicionamiento a la entrada de la tarjeta Arduino funciona como de acuerdo a lo previsto.



Donde dentro del bloque “Tarjeta Arduino” se hace un ajuste por software de los parámetros de forma que al usuario final le sea transparente todo el proceso; escoja una señal de referencia sobre 5V y y vea la de lectura en los Voltios de la maqueta. La composición del bloque ‘Tarjeta Arduino’ se puede ver en la Figura 55.

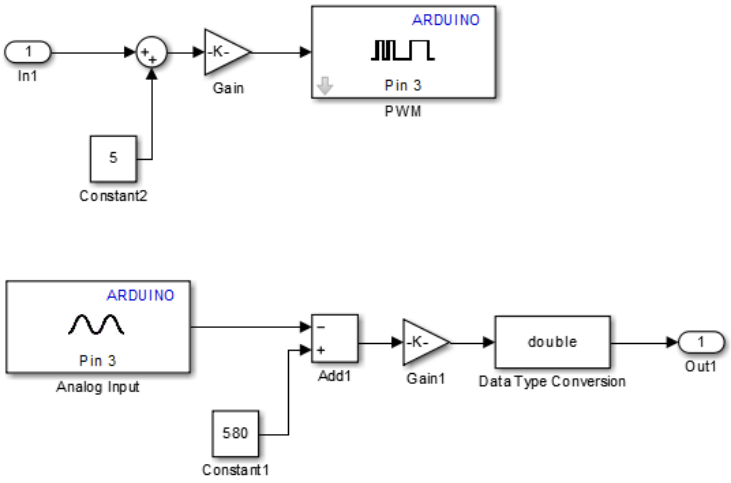


Figura 55 Bloque "Tarjeta Arduino"

El resultado de aplicar la señal de referencia al motor y la lectura de la señal de su estado en velocidad se muestra en la Figura 56, donde se presenta la salida del bloque “Scope” del esquema de Simulink que proporciona la velocidad frente al tiempo, en amarillo se representa la señal de referencia provista al motor, y en rosa la lectura de la velocidad.

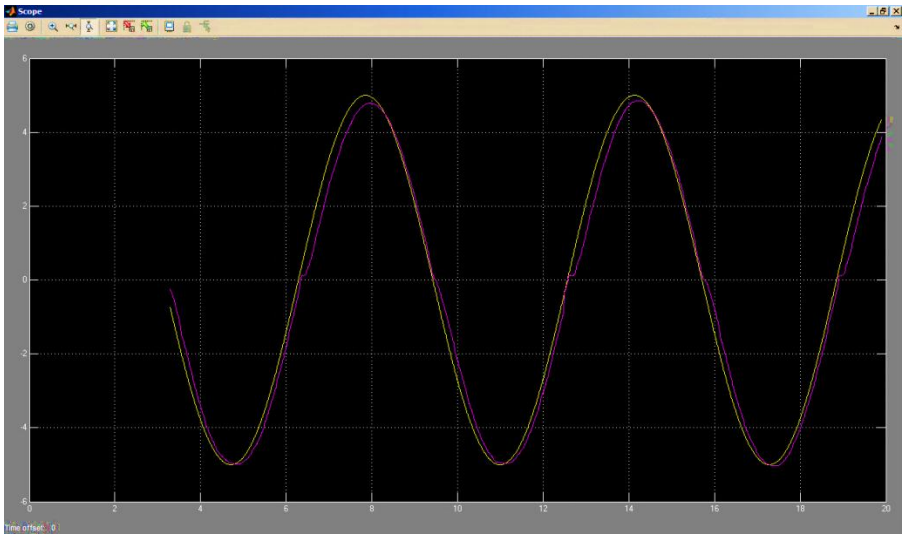


Figura 56 Control de motor con una señal senoidal

Como puede observarse, el motor gira en sincronía con la señal de referencia.

3.6. Estudio de Arduino como Tarjeta de Adquisición de datos

En este apartado es necesario diferenciar los métodos de trabajo para integrar Arduino y Simulink estudiados. Por un lado el soporte Simulink y por otro el paquete Arduino IO. La conexión mediante la comunicación serie explicada en el apartado 2.3.3 depende del programa que ejecute el microcontrolador y las interrupciones que se añadan.

3.6.1. Soporte Simulink

Dado el soporte que traduce un esquema en Simulink a código ejecutable en el microcontrolador y que contiene la librería estudiada en el Capítulo de Estado del Arte.

Este apartado se centra en la frecuencia de muestreo que puede ofrecer la tarjeta Arduino.

Teóricamente y de acuerdo con la documentación, el bloque de "Entrada analógica" de la librería puede configurarse a un valor tan pequeño de tiempo de muestreo como 0.000001s, sin embargo este valor congela la mayoría de los ordenadores. Un valor razonable para el tiempo de muestreo es 0.02s, o lo que es lo mismo, 50 muestras por segundo. Como se deducirá después, tiempos de muestreo menores (o frecuencias de muestreo mayores) no aportan mayor número de muestras, sino que cambian la escala temporal.

De acuerdo al esquema de la Figura 57 se llevan a cabo las pruebas para realizar el estudio.

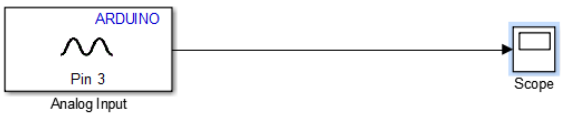


Figura 57 Esquema lectura soporte Simulink

La prueba consiste en la lectura de una onda cuadrada de frecuencia 1Hz en uno de los pines de lectura analógica del Arduino.

Estableciendo el tiempo de muestreo en 0.02s y el tiempo de integración en 10 segundos, se obtiene la señal mostrada en la Figura 58, que representa la señal frente al tiempo.

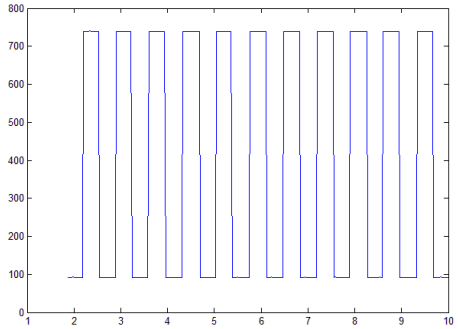


Figura 58 Señal contra tiempo. Tiempo de muestreo = 0.02s

Puede observarse que la señal queda bien definida, para analizar la adquisición de esta señal se representa en la Figura 59 la misma señal, esta vez frente al número de muestra.

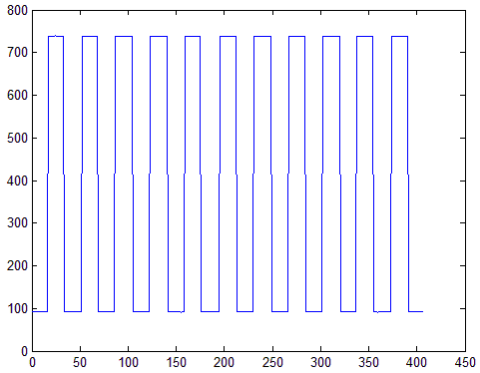


Figura 59 Señal contra número de muestra. Tiempo de muestreo = 0.02s

En esta prueba fueron tomadas exactamente 407 muestras, lo que implica en realidad un tiempo de muestreo de 0.024s. Además se puede observar que no tenemos tiempo real.

Para amplificar éste efecto, se realiza la siguiente prueba con la misma señal cuadrada a 1Hz, pero ésta vez configurando el tiempo de muestreo en 0.002s, con tiempo de integración de 5 segundos. Se representa la señal adquirida frente al tiempo en la Figura 60.

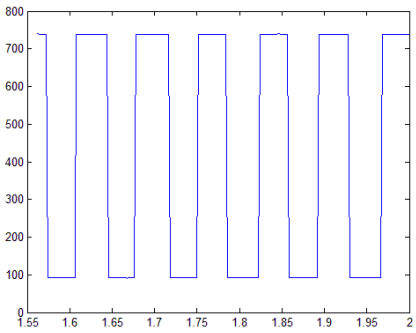


Figura 60 Señal frente a tiempo. Tiempo de muestreo = 0.002s

Como puede deducirse en ésta última figura, la escala temporal del eje 'x' no es real porque no trabaja en tiempo real. Para corroborarlo se representa en la Figura 61 la señal frente al número de muestra.

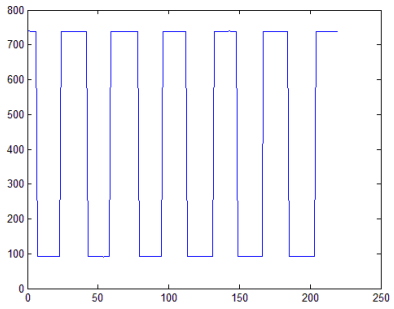


Figura 61 Señal frente a número de muestra. Tiempo de muestreo = 0.002s

Fueron obtenidas 220 muestras en 5 segundos, lo que concluye un tiempo de muestreo esta vez de 0.022s o unas 45 muestras por segundo.

Obviamente no hay tiempo real, y el hecho de incrementar la tasa de muestreo conduce a tomar las mismas muestras, en el mismo espacio de tiempo, pero a representarlas en un período de tiempo cómo si estuvieran siendo tomadas en el tiempo de muestreo configurado.

Mediante éste método de conexión a la tarjeta Arduino, la tasa de muestreo varía según el programa cargado en el microcontrolador.

Esto puede ser debido a que el traductor a código no programa las interrupciones de reloj para realizar la lectura y enviar el dato por el puerto serie, sino que ejecuta el código como un bucle infinito.

No obstante, en las pruebas realizadas, la frecuencia de muestreo siempre ha estado por encima de las 40 muestras por segundo, lo que cumple la especificación de 10 muestras por segundo para realizar el control del motor del laboratorio.

3.6.2. Paquete Arduino IO

En este caso el microcontrolador contiene un servidor que se ejecuta continuamente, por lo que en este caso como será confirmado, la frecuencia de muestreo será independiente del esquema de Simulink.

Para realizar las pruebas, de forma similar al apartado anterior, se utiliza el esquema de la Figura 62.

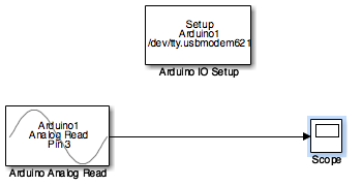


Figura 62 Esquema lectura Arduino IO

Con el generador de señales, se proporciona también una onda cuadrada de frecuencia 1Hz conectada a uno de los pines de lectura analógica de Arduino.

Como puede observarse en las Figura 63 y Figura 64, que representan la señal obtenida frente al tiempo en el primer caso, y frente al número de muestra en el segundo, tampoco se está trabajando en tiempo real. Sin embargo el número de muestras tomadas es superior y la frecuencia de muestreo constante, como se verá posteriormente.

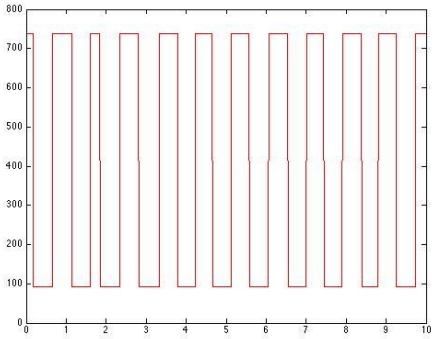


Figura 63 Señal frente a tiempo de muestreo. Tiempo de muestreo = 0.01s

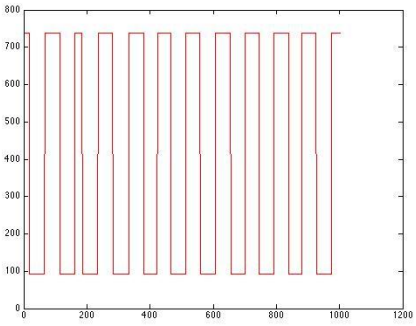


Figura 64 Señal frente a número de muestra. Tiempo de muestreo = 0.01s

Fueron tomadas 1001 muestras en 10 segundos, lo que arroja una tasa de muestreo igual a 100 muestras por segundo. Superior al caso anterior.

A continuación se confirmará la hipótesis realizada sobre que el número de muestras se mantiene constante. Para ello si se aumenta el número de muestras por segundo hasta 1000 se obtiene la señal representada en la Figura 65.

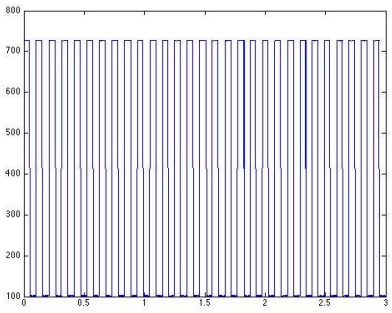


Figura 65 Señal frente a tiempo. Tiempo de muestreo = 0.001s

Como puede observarse, la escala de tiempo sigue sin ser real, ya que se está trabajando con una onda escalón a 1Hz.

Para analizar el número de muestras tomadas se recurre a la Figura 66 que representa la misma señal frente al número de la muestra.

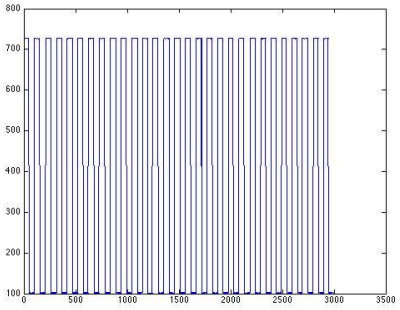


Figura 66 Señal frente a número de muestra. Tiempo de muestreo = 0.001s

En los 30 segundos de duración del experimento fueron tomadas 3001 muestras, lo que confirma la tasa de muestreo de 100 muestras por segundo.

4. Interfaz con Arduino

4.1. Introducción

La manera propuesta se basa en el comportamiento de la placa Arduino como servidor que acepta conexiones de una aplicación cliente en un teléfono Android. Esta forma que no aparece documentada oficialmente por Arduino, está señalada de forma no oficial en algunos blogs en la web [14]. La placa se encuentra conectada a la red local por medio de un cable Ethernet, y el dispositivo Android a través de Wifi, también se encuentra en la misma red local como ilustra el esquema de la Figura 67.

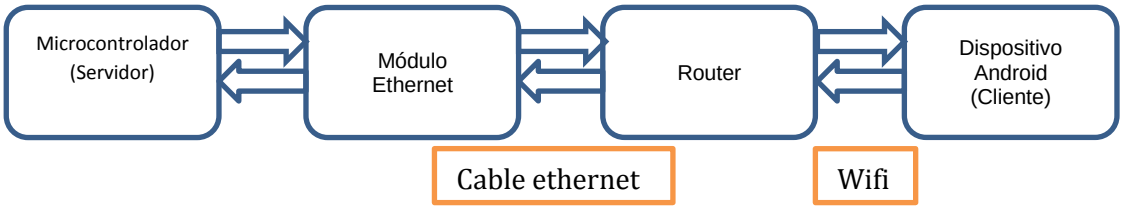


Figura 67 Esquema de conexión

Esta ha sido la opción elegida para realizar la comunicación, por los siguientes motivos:

- El reducido coste del módulo ethernet.

- El alcance dentro de todo un área wifi, mayor que el bluetooth y que la conexión por el cable de audio.

El precio del router no se computa aquí ya que con un router se podría dar servicio varios microcontroladores, por lo que dependiendo del número de puestos de la instalación varía su precio unitario.

4.2. Conexión

Cómo se ilustra en la Figura 67, el microcontrolador se comporta como un servidor al que se puede conectar una aplicación cliente que se encuentra en un dispositivo Android.

La aplicación móvil, a través de una interfaz gráfica, inicia la conexión con el servidor, envía peticiones al mismo y espera su respuesta para mostrarla también de forma gráfica.

Por otra parte, el microcontrolador realiza la tarea de control para la que haya sido programado al tiempo que espera la petición de un cliente. Cuándo ésta llega, la procesa y envía la respuesta.

4.2.1. Servidor

Se trata de la parte que acepta las conexiones, y es ejecutado por el microcontrolador. Cómo se muestra en el flujograma de la Figura 68, tras el inicio o reseteo del microcontrolador, se produce la configuración de la dirección IP, MAC y puerto para a continuación iniciar la conexión. Una vez haya un cliente conectado prosigue con la lectura del sensor, y si ha cambiado respecto de su último valor, envía el valor actual. A

continuación si ha recibido una petición del cliente actualiza el valor del actuador(Código en Anexo B.3).

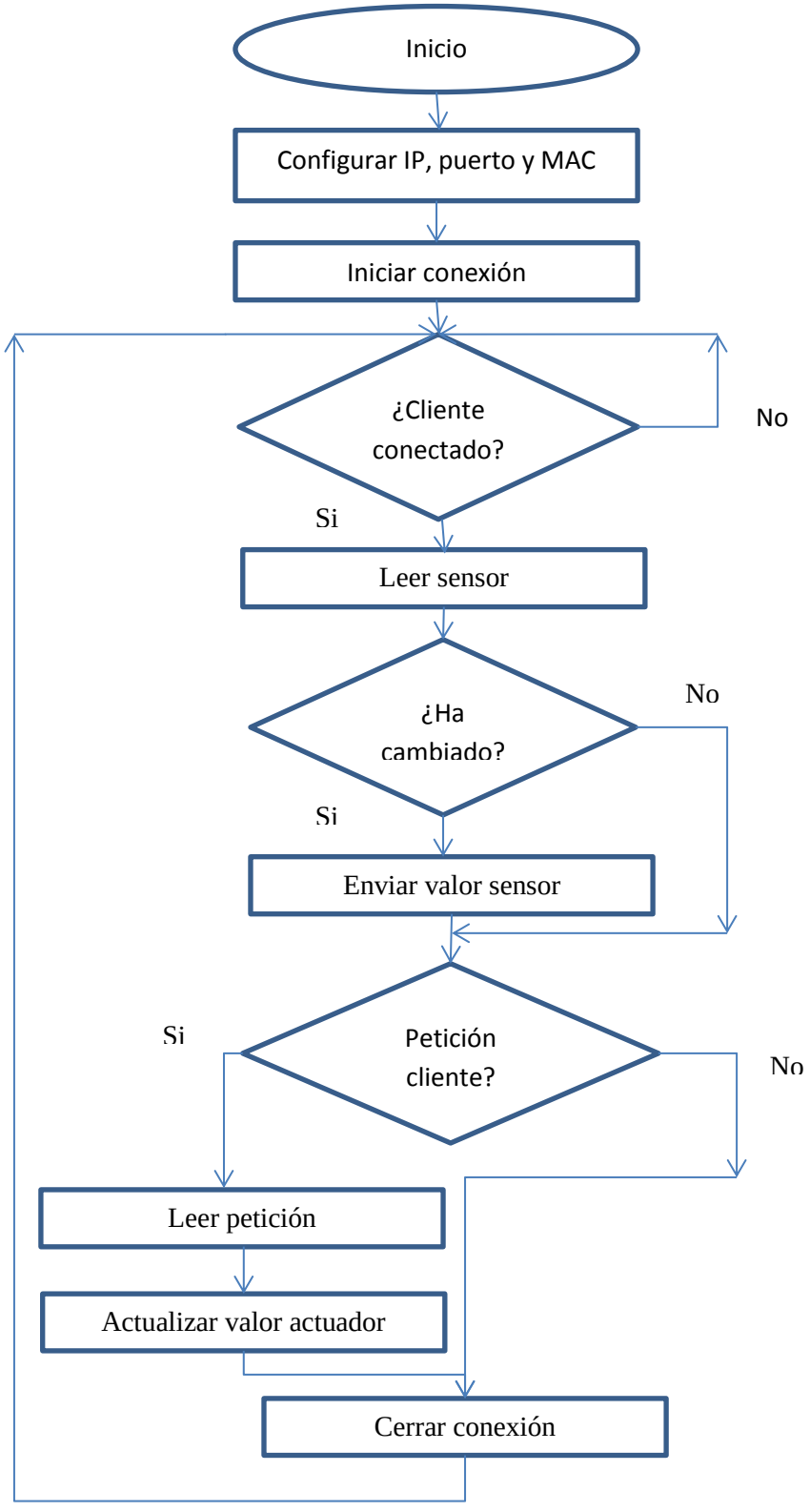


Figura 68 Flujograma programa servidor

4.2.2. Cliente

La aplicación cliente (Figura 69) consiste en una interfaz Android que contiene, de la parte superior a la inferior: un botón para conectar/desconectar al microcontrolador (servidor); una barra con la que se envía una señal de referencia al sistema, es decir, puede controlar un actuador variando el pulso PWM, -en las pruebas, el actuador ha consistido en un LED-; otra barra que representa el estado del sistema, es decir, que muestra la lectura de un sensor; a continuación se encuentra un cuadro de texto que muestra los mensajes de conexión/desconexión; y por último una representación gráfica de la lectura del sensor. El código de la aplicación se encuentra en el Anexo B.4.

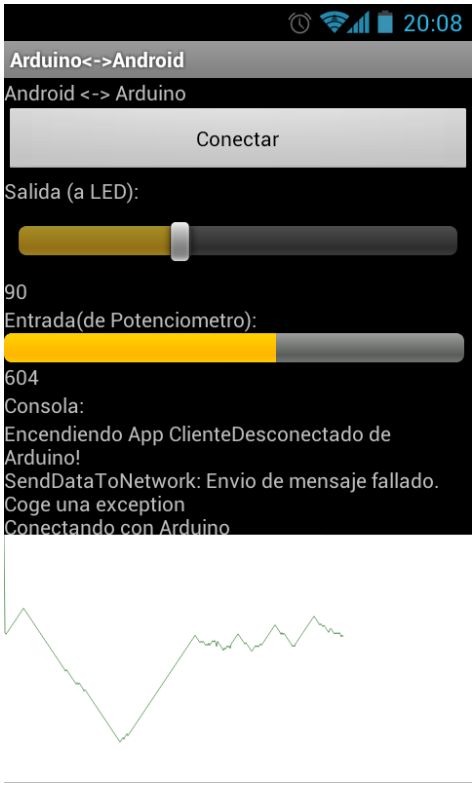


Figura 69 Aplicación cliente

La actividad es el componente de la aplicación encargado de mostrar la interfaz de usuario, es decir, es la ventana y sus elementos se encuentran definidos en el fichero XML con nombre 'Main'.

La aplicación (Figura 70) comienza definiendo los objetos que se muestran en la pantalla (botones, barras, gráfico, cuadros de texto, etc) y conecta dichos objetos a los elementos de la interfaz. A continuación se activan los eventos que indicarán a la aplicación cuándo se ha pulsado el botón conectar, desplazado la barra de control de un actuador o recibido un dato por la red. Una aplicación Android se puede pensar como una interfaz gráfica a la espera de eventos, que son gestionados cuándo se producen.

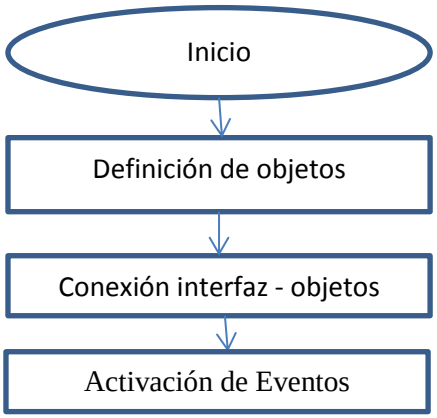


Figura 70 Método Actividad principal, conecta la interfaz de usuario con los objetos

Tras el inicio de la aplicación el usuario hará clic en el botón conexión/desconexión, que dispara el evento encargado de crear una conexión nueva si no hay ninguna establecida o cerrar la actual si hay alguna en proceso (Figura 71).

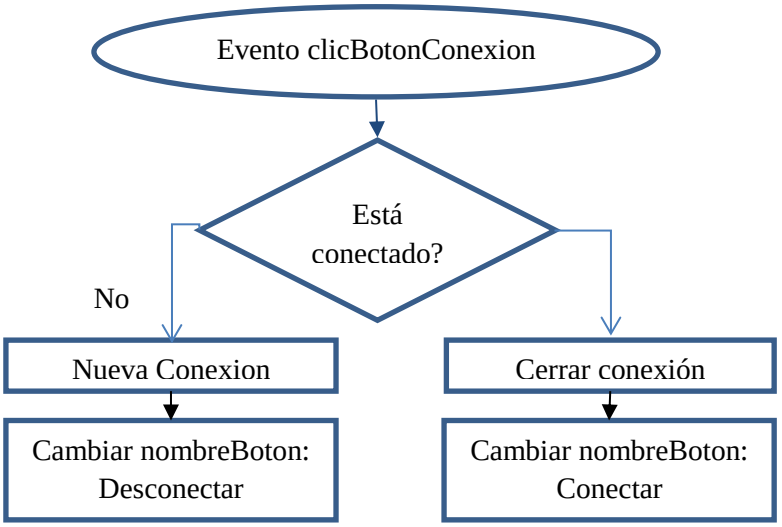


Figura 71 Método clicBotonConexion, llamado cuando el usuario pulsa el botón conectar/desconectar

El siguiente elemento de la interfaz es una barra llamada 'barraActuador' en el programa, que está ligada a un evento (Figura 72) iniciado por el desplazamiento de la misma, una vez que la conexión con el microcontrolador ha sido establecida, este dispara la secuencia de acciones para tomar el valor de la barra, actualizar el cuadro de texto que indica el valor actual de la misma y enviar la información posición al microcontrolador (Figura 74), que ajustará la señal PWM para satisfacer la orden.

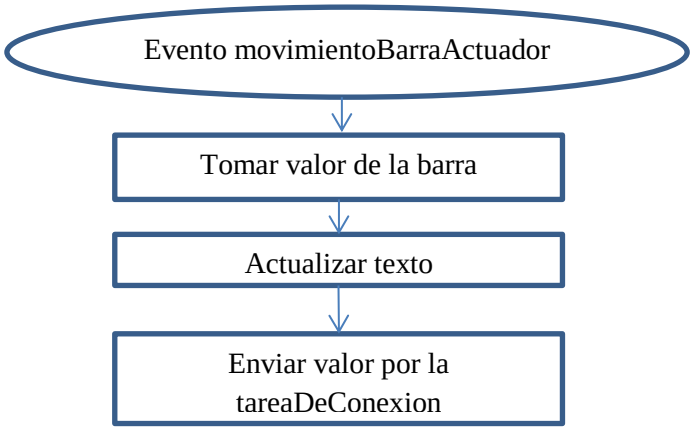


Figura 72 Método movimientoBarraActuador, llamado cuando el usuario desplaza la barraActuador

La barra que representa la información de estado del sistema, señal proporcionada por un sensor y se actualiza junto con el gráfico cada vez que se reciben datos por la red. Para ello la tarea de red una vez inicializada y configurados sus parámetros entra en espera en otro hilo de ejecución y cuando un byte es recibido, se llama a los métodos encargados de actualizar la barra y el gráfico (Figura 73).

Por último, el método que envía los datos por la red (Figura 74) y comprueba que hay una conexión establecida, si existe conexión envía el dato por la red y si no, envía una excepción al cuadro de texto que muestra los mensajes de la aplicación.

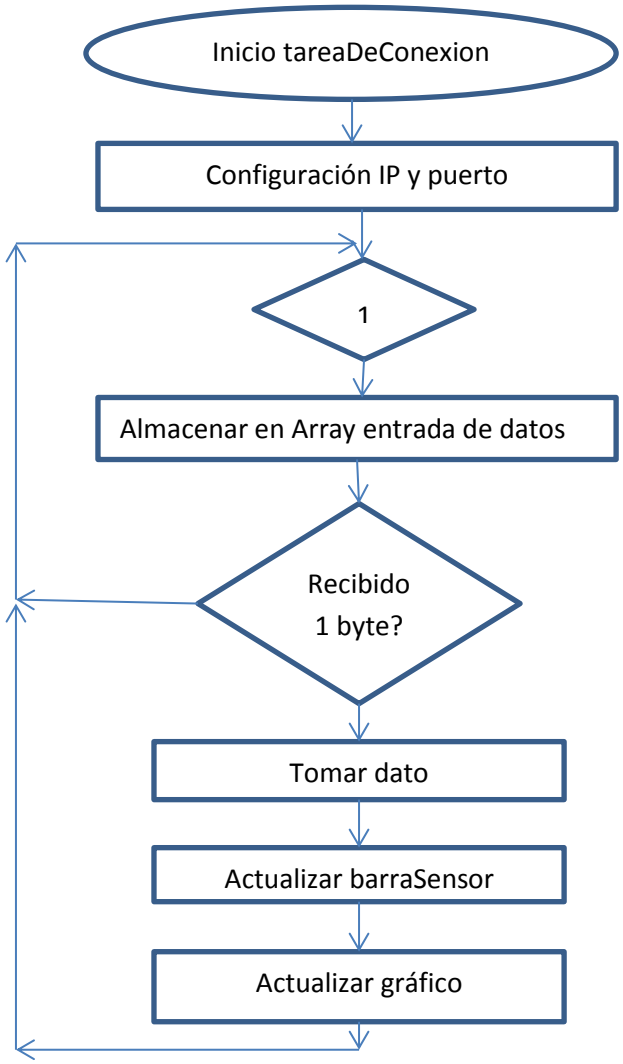


Figura 73 Metodo onProgressUpdate, llamado cada vez que se recibe una cadena

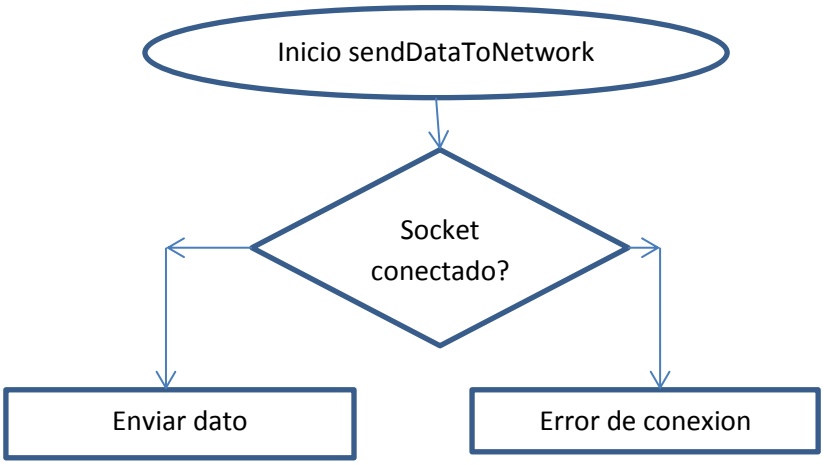


Figura 74 Método sendDataToNetwork, llamado cada vez que se mueve la barraActuador

Tal y como se muestra a lo largo de este capítulo, una de las ventajas inmediatas de este método es su independencia de un sistema de comunicación física, ya que no requiere un cable USB o de audio conectado de forma permanente. También se evitan las limitaciones de distancia del bluetooth. Por otra parte permite la fácil extrapolación de esta arquitectura cliente-servidor a otros sistemas de comunicación como podrían ser la sustitución del módulo Ethernet por uno Wifi y sólo tener que cambiar los parámetros de configuración de la conexión.

5. Resultados experimentales: Control de un motor de Corriente Continua

En este capítulo se procede a la implementación de un regulador PID sobre el microcontrolador para la maqueta del motor de corriente continua. Se realizará el control en posición del motor como propuesta de una práctica docente de Ingeniería de Control.

A priori, la función de transferencia del sistema es desconocida, luego el ajuste de los parámetros del regulador PID se realizará de forma empírica. Dado que el tiempo de respuesta del sistema es inferior a un segundo, se podrá hacer de forma rápida.

5.1. Método de Ziegler-Nichols

Se trata de dos métodos empíricos de ajuste de reguladores PID en los que a partir de algunas características de respuesta del proceso se proponen los parámetros de dicho regulador [19].

Las fórmulas que permiten obtener los parámetros del regulador PID mediante este método, intentan conseguir que el sistema de lazo cerrado se aproxime al valor de referencia con una sobreoscilación inicial dada (no hay un valor máximo concreto sobre este parámetro) y que la siguiente sobreoscilación se reduzca a ¼ de la primera, tal y como se muestra en la figura 75. De esta forma consiguen una solución de compromiso entre el transitorio y el permanente.

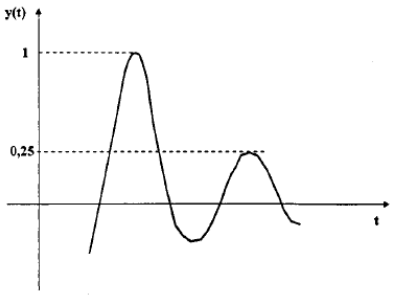


Figura 75 Especificación del método Ziegler-Nichols

Para ello, a partir de la respuesta del sistema en las condiciones que especifique cada método en concreto, se calculan los parámetros K, T_d y T_i de un PID dado por:

$$R(s) = K_p + K_d \cdot s + \frac{K_i}{s}$$

En el siguiente apartado será aplicado el método de estimación de estabilidad límite. Este método se basa en llevar a la estabilidad límite al sistema en cadena cerrada mediante una acción proporcional y medir unos determinados parámetros de la respuesta ante escalón del sistema en cadena cerrada. A partir de ellos se calculan los parámetros del regulador.

5.2. Ajuste empírico del regulador

Para la prueba del sistema se hace el ajuste sobre la plataforma experimental que se muestra en la Figura 76, dónde se puede ver la maqueta de pruebas que consiste en el motor de CC, el panel de control de dicha maqueta, la etapa de acondicionamiento diseñada, la placa Arduino o microcontrolador y un ordenador comportándose como cliente de la placa.

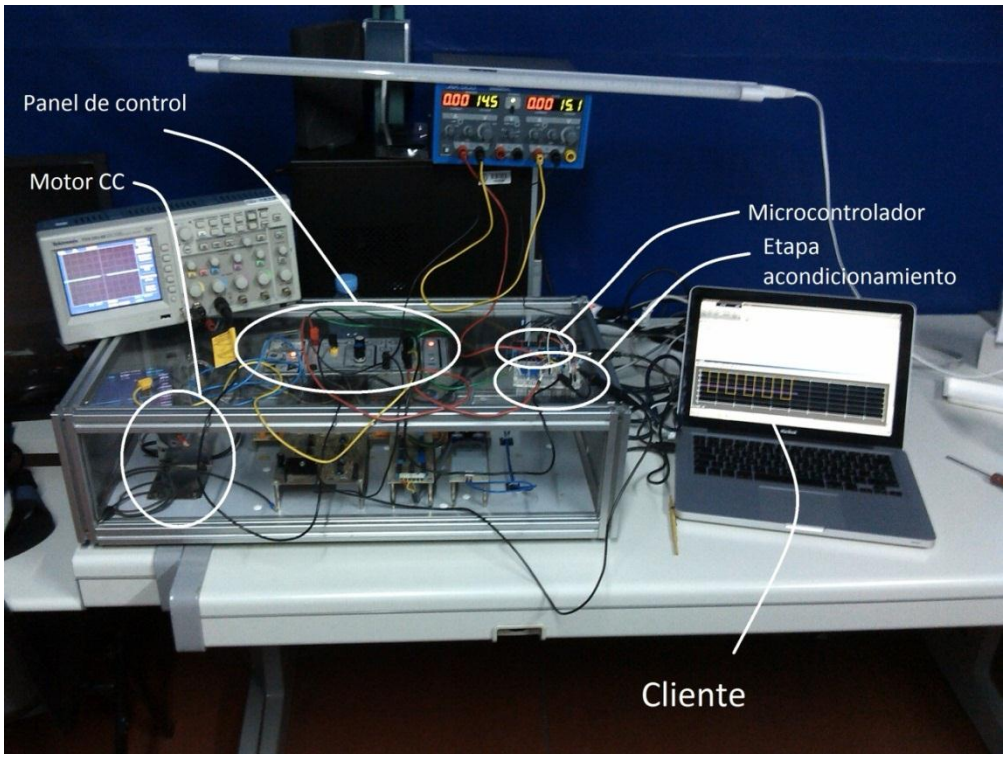


Figura 76 Plataforma experimental

Se utilizará el método de Ziegler-Nichols de la respuesta frecuencial o de cadena cerrada. Para su implementación se realiza el esquema de la Figura 77, en ella se cierra el bucle de realimentación con un regulador de tipo proporcional en Simulink.

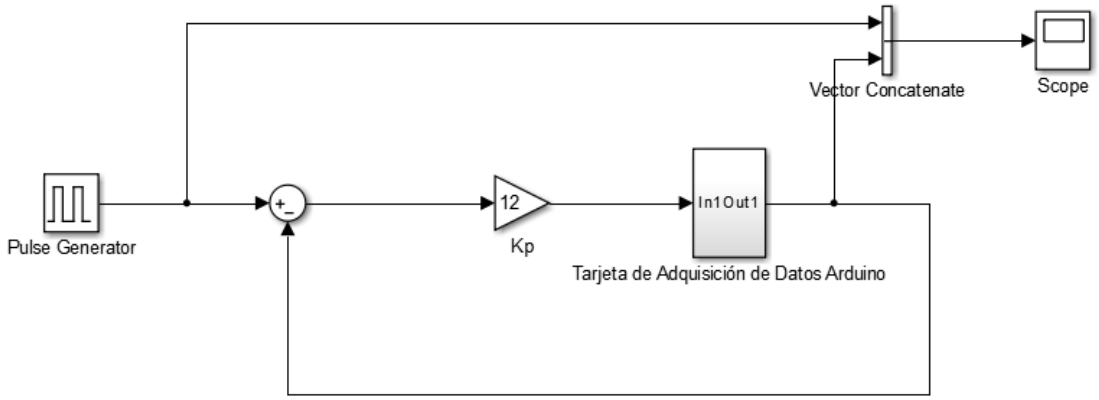


Figura 77 Esquema para ajuste Ziegler-Nichols en bucle cerrado

La señal de referencia será una onda cuadrada con la que se calculan los parámetros del PID. Se aumenta la ganancia de dicho regulador desde ganancia 1 hasta obtener la ganancia crítica en la que el sistema presenta una respuesta oscilatoria sostenida como se ve en la Figura 78, que presenta la salida del bloque "Scope" de Simulink, y muestra la posición del sistema frente al tiempo. Para esta ganancia se determina el período crítico de la señal de salida oscilatoria.

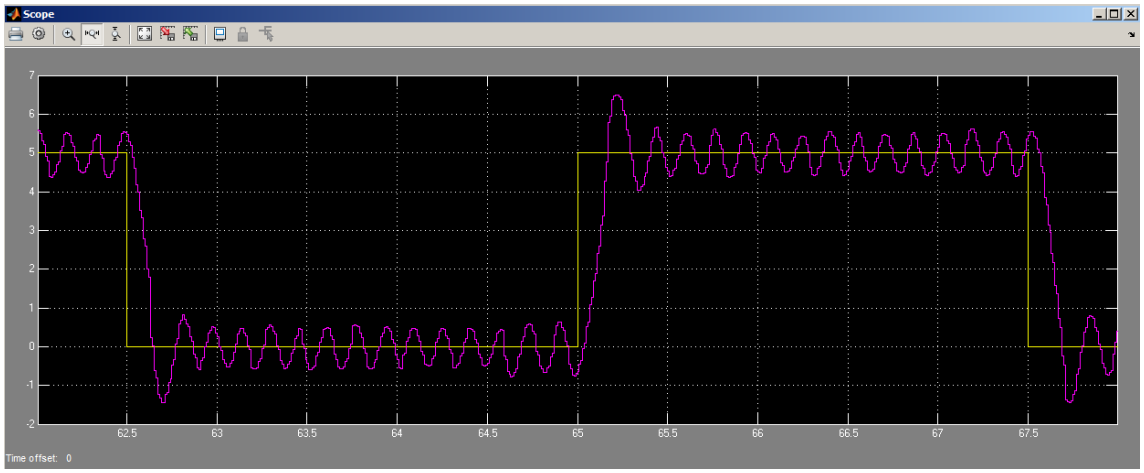


Figura 78 Ganacia crítica.

La salida oscilatoria mantenida se consigue con una ganancia crítica, $K_C = 12$. Para ella, el período de dichas oscilaciones es:

$$T_{crítico} = 0.14\text{s}$$

Con los valores de los parámetros propuestos pos Ziegler-Nichols para el método frecuencial se obtienen los parámetros del regulador:

$$K_p = 0.6 \cdot K_C = 7.2$$

$$K_i = 1.2 \cdot \frac{K_c}{t_c} = 11.9$$

$$K_d = 0.075 \cdot K_c \cdot t_c = 0.126$$

5.3. Prueba del regulador

Una vez obtenidos los parámetros del regulador se realiza la implementación del regulador del sistema en Simulink, como se observa en la Figura 79.

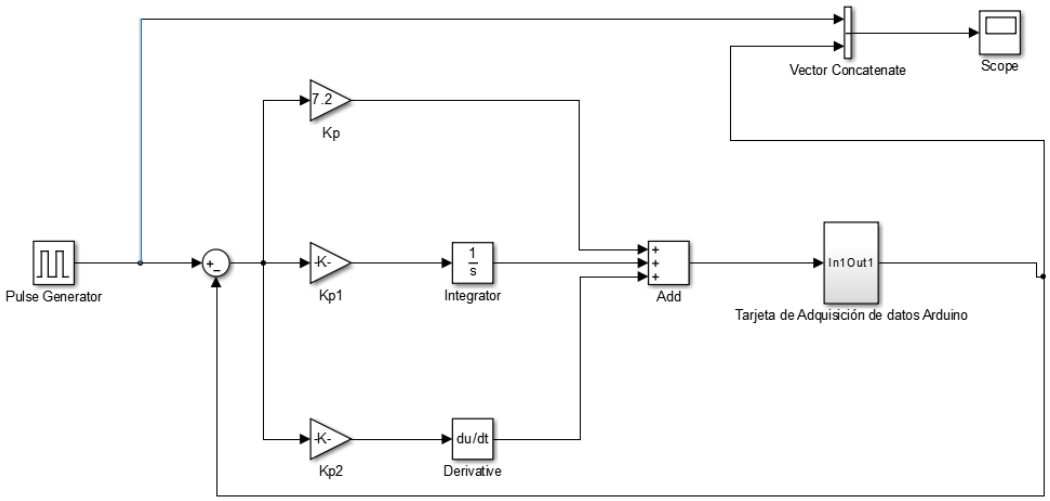


Figura 79 Implementación PID

Ajustando el regulador con los parámetros obtenidos, se obtiene como respuesta del sistema regulado la mostrada por el bloque “Scope” del esquema anterior, que representa la posición del sistema frente al tiempo, Figura 80.

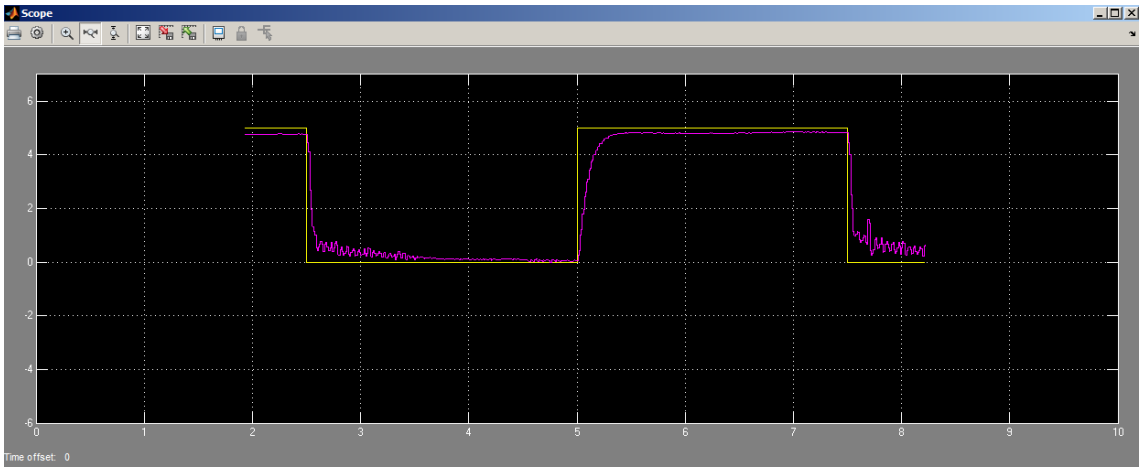


Figura 80 Resultado implementación PID

De esta manera y mediante el sistema propuesto se ha podido realizar de forma fácil y a muy bajo coste el control de un sistema físico mediante un esquema de control implementado a través de Simulink.

Como ejemplo de otros reguladores se propone también la implementación de reguladores P o PI, donde se puede observar experimentando como influyen las distintas variables proporcional, integral y derivativa en el control.

En la Figura 81 se muestra la respuesta de un regulador proporcional en el que no se alcanza el seguimiento de la señal de referencia.

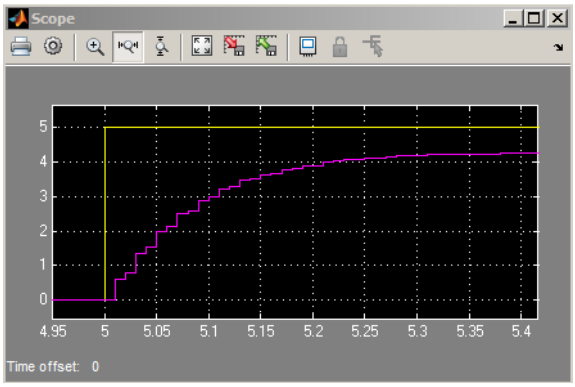


Figura 81 Regulador Proporcional

También se ha realizado la implementación de un regulador Proporcional-Integral cuya respuesta se observa en la Figura 82 donde se representa la posición frente al tiempo. Como se puede ver se evitan las oscilaciones del PID de la Figura 79.

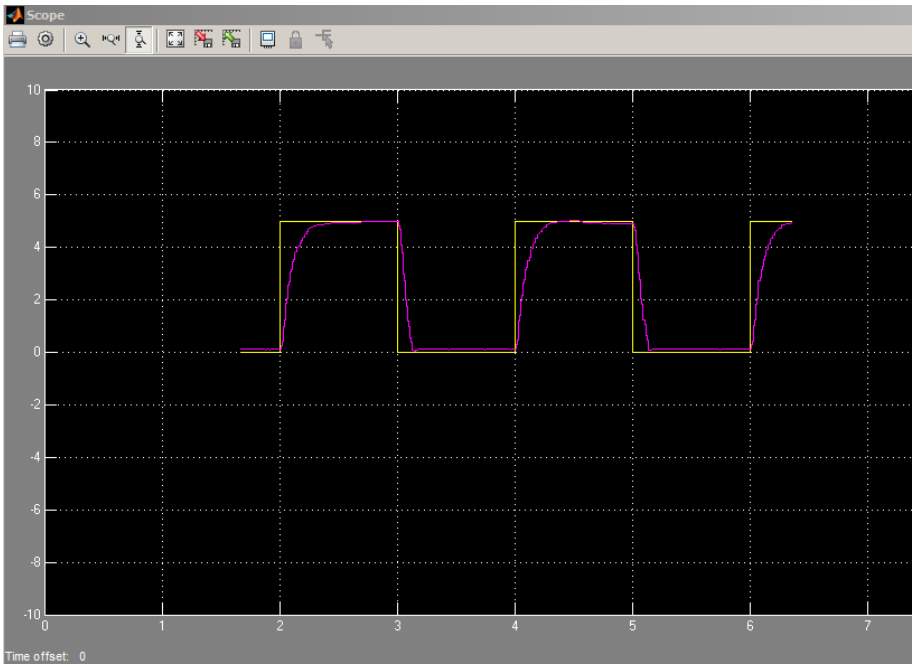


Figura 82 Regulador Proporcional-Integral

Tal y como se demuestra con estos experimentos el sistema permite realizar esquemas de control y prácticas docentes sobre Matlab y Simulink mediante el hardware de bajo coste diseñado.

6. Conclusiones y trabajos futuros

6.1. Conclusiones

En este trabajo se ha desarrollado un sistema de adquisición de datos de bajo coste basado en la placa Arduino que trabaja como DAQ entre Simulink o una aplicación móvil y el mundo físico. Se ha diseñado y desarrollado la interfaz hardware para adaptar las señales entre el microcontrolador y las maquetas de laboratorio, así como una interfaz software para actuar sobre el sistema físico.

Varias pruebas han sido realizadas para validar el sistema completo y se llevado a cabo un estudio de la frecuencia de muestreo para conocer las posibilidades de esta arquitectura. Se ha deducido que puede ser utilizado en la mayoría de los sistemas mecatrónicos.

Finalmente se ha propuesto, resuelto y probado un ejercicio práctico para validar la propuesta.

6.2. Trabajos futuros

En este apartado se proponen algunas posibles líneas de desarrollo para trabajos futuros.

Respecto del uso de la plataforma Arduino como DAQs de bajo coste, tras ver las limitaciones de las tarjetas probadas y para superarlas se propone la prueba de la placa Arduino DUE, que con sus características permitirá realizar una adquisición de datos a gran velocidad.

De manera similar se propone la prueba de Raspberry Pi, un pequeño ordenador con un precio de 35$ capaz de correr Linux y realizar tareas multimedia, se le puede adjuntar una cámara y tiene soporte de Simulink para la carga de algoritmos en los que se capta audio y video.

En relación a la aplicación Android, la siguiente mejora podría ser la implementación de una interfaz gráfica que guíe al alumno en la realización de las prácticas al tiempo que interactúa con la maqueta. Otra opción sería la posibilidad de añadir interfaces de información de sensores y actuadores a la aplicación para el control de sistemas de mayor complejidad.

7. Referencias

Enlaces web visualizados a Mayo 2013.

[1] Enlace web: http://arduino.cc

[2] Enlace web: http://wiring.org.co

[3] Enlace web: http://processing.org

[4] Enlace web: http://arduino.cc/es/Main/Hardware

[5] Enlace web: http://arduino.cc/es/Reference/Libraries

[6] Enlace web: http://arduino.cc/en/Reference/EEPROM

[7] Enlace web: http://arduino.cc/en/Reference/Ethernet

[8] Enlace web: http://www.mathworks.com/matlabcentral/fileexchange/32374-matlab-support-package-for-arduino-aka-arduinoio-package

[9] Enlace web: http://www.mathworks.es/academia/arduino-software/arduino-simulink.html

[10] Enlace web: http://www.amarino-toolkit.net/

[11] Enlace web: http://androino.blogspot.com.es/

[12] Enlace web: http://developer.android.com/tools/adk/index.html

[13] Enlace web: http://playground.arduino.cc//Main/InterfacingWithSoftware

[14] Enlace web: http://www.lauridmeyer.com/category/tutorials/arduino/

[15] Ned Moham, Tore M. Undeland y William P. Robbins "*Power Electronics*". Editorial JOHN WILEY & SONS, INC. ISBN: 0471584088. Fecha de publicación: 2003.

[16] Andrés Barrado Bautista y Antonio Lázaro Blanco *"Problemas de electrónica de Potencia"*. Editorial PEARSON. ISBN: 9788420546520. Fecha de publicación: 2001.

[17] Jorge Pleite Guerra, Ricardo Vergaz Benito y José Manuel Ruiz de Marcos "*Electrónica analógica para ingenieros"*. Editorial MC GRAW HILL. ISBN: 9788448168858. Fecha de publicación: 2009.

[18] Enlace web: http://www.mathworks.es/es/help/simulink/slref/raspberrypiv4l2videocapture.html

[19] Luis Moreno, Santiago Garrido y Carlos Balaguer "Ingeniería de Control". Editorial Ariel Ciencia. ISBN: 8434480557 Fecha de publicación: 2003.

Anexos

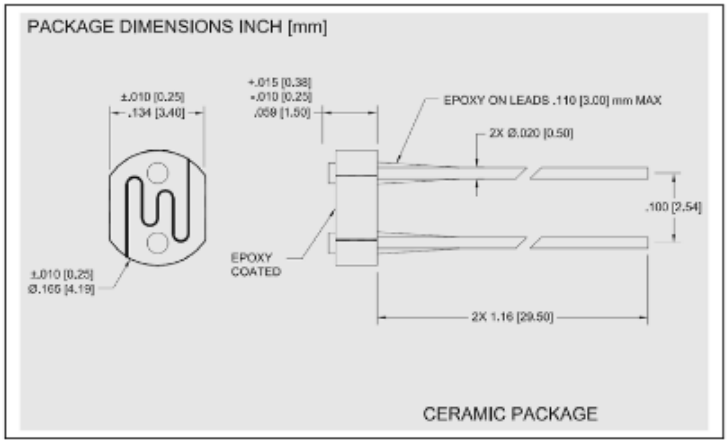
Índice de anexos:

A. Hojas de características de los componentes

A.1. LDR

API
Advanced Photonix, Inc.

CdS Photoconductive Photocells
PDV-P9203



FEATURES
- Visible light response
- Sintered construction
- Low cost

DESCRIPTION
The **PDV-P9203** are (CdS), Photoconductive photocells designed to sense light from 400 to 700 nm. These light dependent resistors are available in a wide range of resistance values. They're packaged in a two leaded plastic-coated ceramic header.

APPLICATIONS
- Camera exposure
- Shutter controls
- Night light Controls

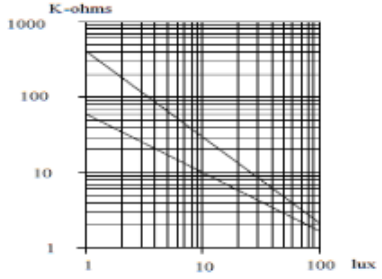
RoHS

ABSOLUTE MAXIMUM RATING (TA)=23°C UNLESS OTHERWISE NOTED

SYMBOL	PARAMETER	MIN	MAX	UNITS
V_{pk}	Applied Voltage		150	V
$P_{d\ above\ 50°C}$	Continuous Power Dissipation		125	mW/°C
T_O	Operating and Storage Temperature	-25	+75	°C
T_S	Soldering Temperature*		+260	°C

* 0.200 inch from base for 3 seconds with heatsink.

CELL RESISTANCE VS. ILLUMINANCE



B. Código

B.1. Comunicación Serie. Código Arduino

```
//Pines
int outPin = 5; //Pin de salida
int inPin = A0; //Pin de entrada


//Variables
//int refPot;
byte deSimulink;
byte aSimulink;


void setup()
{
  //
  pinMode(outPin, OUTPUT);
  pinMode(inPin, INPUT);


  Serial.begin(9600);
}
```

```
void loop()
{
  //Leemos el potenciometro y lo envia
  int refPot = analogRead(inPin);
  aSimulink = map(refPot, 0, 1023, 0, 255);
  Serial.write(aSimulink);



  //Lee la señal para el led
  if(Serial.available()){
    deSimulink = Serial.read();
    analogWrite(outPin, aSimulink);
  }

  delay(500);
}
```

B.2. Arduino IO. Función Matlab

```
function ConceptoTiempoReal(interv)


a = arduino('/dev/tty.usbmodem621');


% Inicializamos variables
%interv = 1000;
paso = 1;
t=1;
```

```
x1=0;
x2=0;
base = 0;

%El bucle se realiza durante el numero de muestras interv
while(t<interv)
    %Realizamos la lectura del potenci?metro que nos da la
referencia
    referencia=a.analogRead(0);
    %Leemos el sensor de luz
    sensor = a.analogRead(3);

    %Normalizamos las magnitudes de ambas lecturas
    %para que puedan ser comparadas
    ReferenciaMap = round((referencia/1023)*254);
    SensorMap = round(((sensor-150)/ 850) * 254);

    %Control del Led
    diferencia = ReferenciaMap - SensorMap;
    salida = ReferenciaMap + diferencia;

    %La salida del LED est? limitada a 250
    if(salida>250) salida = 250;
    elseif(salida < 5) salida = 0;
    end

    a.analogWrite(3, salida);
```

```
    x1=[x1,referencia];
    x2=[x2,sensor];
    base = [base, t];


    plot(base, x1, 'r', base, x2, 'b');


    axis([0,interv,0,1200]);
    grid
    t=t+paso;
    drawnow;
end


delete(a);
 delete(a);
```

B.3. Código Servidor

```
#include <SPI.h>
#include <Ethernet.h>


// Se introduce la direccion IP, MAC, y el puerto del servidor.
// La IP depende del servidor local
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0D, 0x7A, 0x10 };
IPAddress serverIP(192,168,1,177);
int serverPort=8888;
int digPin=3;
```

```
// Se inicializa la libreria ethernet
EthernetServer server(serverPort);

// Variable para almacenar el valor anterior de un sensor
analogico
// para saber si ha cambiado
int preValorSensor=0;

void setup()
{
  // El puerto serie se configura para depurar el programa
  Serial.begin(9600);
  // Se comienza la conexion ethernet
  Ethernet.begin(mac, serverIP);
  server.begin();
  Serial.println("Servidor iniciado. OK");// Para depurar
}

void loop()
{
  // Escucha de conexion entrante
  EthernetClient client = server.available();
  if (client) {
    String commandStr ="";//Almacena los comandos entrantes
// Si un cliente se conecta:
    while (client.connected()) {
      //Lee el sensor
      int valorSensor = analogRead(A0);
      // Comprueba si el valor ha cambiado +5, -5
```

```
      if(valorSensor>(preValorSensor+5) ||
valorSensor<(preValorSensor-5)){
          // Para depurar, envia el valor por el puerto serie:
          Serial.println("El valor del sensor ha
cambiado:"+String(valorSensor));
          // Envia el valor:
          server.println("Sensor:"+String(valorSensor));
          // Actualiza el ultimo valor del sensor:
          preValorSensor=valorSensor;
      }
      // Si el cliente envia una peticion:
      if (client.available()) {
        // Se lee dicha peticion
        char c = client.read();
        commandStr+=c;

        // Si el comando se ha recibido por completo
        if (c == '\n') {
          // Se muestra por el puerto serie para depurar:
          Serial.println("Comando Actuador:"+commandStr);
          // Si el comando comienza por "set:", se establece el
actuador a dicho valor
          if(commandStr.indexOf("Actuador:")==0){
            String valor=commandStr;
            valor.replace("Actuador:", "");
            Serial.println("Establece actuador en:"+valor);
            analogWrite(digPin, convertToInt(valor));
            server.println("Actuador:"+valor);
          }
          // Se resetea el comando
```

```
          commandStr="";
        }
      }
    }
    delay(1);
    // Se cierra la conexion
    client.stop();
  }
}

// Funcion para convertir una cadena en un entero
int convertToInt(String valor)
{
  char buf[valor.length()];
  valor.toCharArray(buf,valor.length());
  return atoi(buf);
}
```

B.4. Código Aplicación Android

Main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    android:layout_width="fill_parent"

    android:layout_height="wrap_content"

    android:orientation="vertical" >


    <TextView

        android:layout_width="fill_parent"

        android:layout_height="wrap_content"

        android:text="@string/hello" />


    <Button

        android:id="@+id/connect"

        android:layout_width="fill_parent"

        android:layout_height="wrap_content"

        android:text="connect" />


    <TextView

        android:layout_width="fill_parent"

        android:layout_height="wrap_content"

        android:text="@string/actuador_nombre" />
```

```
<SeekBar
    android:id="@+id/seekbar"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:max="250"
    android:progress="0" />

<TextView
    android:id="@+id/seekbarvalue"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="0" />

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/sensor_nombre" />

<ProgressBar
    android:id="@+id/progressbar"
    style="@android:style/Widget.ProgressBar.Horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginRight="5dp"
    android:max="1024"
    android:progress="0"
```

```
        android:visibility="visible" />

    <TextView
        android:id="@+id/lecturapotenciometro"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="0" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/console" />

    <TextView
        android:id="@+id/textlog"
        android:layout_width="fill_parent"
        android:layout_height="match_parent"
        android:layout_weight="4.68"
        android:scrollbarStyle="outsideOverlay"
        android:scrollbars="vertical" />

    <com.lauridmeyer.tests.GraphView
        android:id="@+id/graphView1"
        android:layout_width="wrap_content"
        android:layout_height="170dp" />
</LinearLayout>
```

Strings.xml

```
<?xml version="1.0" encoding="utf-8"?>

<resources>


    <string name="hello">Android &lt;-&gt; Arduino</string>

    <string name="console">Consola:</string>

    <string name="app_name">Arduino&lt;-&gt;Android</string>


    <string name="actuador_nombre">Salida (a LED):</string>


    <string name="sensor_nombre">Entrada(de
Potenciometro):</string>


</resources>
```

TAD.java

```
/*
 * Cliente TCP Arduino <-> Android
 *
 *
 * Basado en "BidirectionalAndroidTCPClient":
 * http://www.lauridmeyer.com
 *
 */
package com.hk.mario;


import java.io.BufferedReader;
import java.io.IOException;
```

```
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.SocketAddress;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.SeekBar;
import android.widget.TextView;

public class mainActivity extends Activity {

	int datoRecibido;

	// Log conocer el estado de la app
	TextView textlog;

	// Se definen los objetos: Boton conexion/desconexion,
	// barra para mover el actuador, valor de la barra,
	// barra indicadora del sensor y valor de dicha barra.
	Button botonConexion;
	SeekBar barraActuador;
	TextView valorBarraActuador;
```

```
        ProgressBar barraSensor;
        TextView valorBarraSensor;

        // Se almacena el estado de la conexion
        Boolean bConexion=false;

        // networktask es la clase que gestiona la conexion
        NetworkTask networktask;

        // Grafico:
        private GraphView mGraph;
        //private TextView mValueTV;

        /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

 // Se conectan los objetos con elementos de la interfaz
           botonConexion = (Button)findViewById(R.id.connect);
           textlog = (TextView)findViewById(R.id.textlog);
           barraActuador = (SeekBar)findViewById(R.id.seekbar);
           valorBarraActuador =
(TextView)findViewById(R.id.seekbarvalue);
           barraSensor =
(ProgressBar)findViewById(R.id.progressbar);
           valorBarraSensor =
(TextView)findViewById(R.id.lecturapotenciometro);
```

```
        mGraph = (GraphView)findViewById(R.id.graphView1);



        //Maximo valor del grafico en eje 'y'
        mGraph.setMaxValue(1024);

        // Se comunica al usuario que la app esta funcionando
        // y se pone el estadoConexion en 'desconectado'
          textlog.setText("Encendiendo App Cliente");
          cambiarEstadoConexion(false);

          // Se establecen los metodos que escuchan los eventos,
          // de esta forma la app sabe cuando se ha pulsado un
boton
          // o desplazado una barra

botonConexion.setOnClickListener(eventoClicBotonConexion);

barraActuador.setOnSeekBarChangeListener(eventoMovimientoBarraAc
tuador);


        // Se crea la instancia inicial de NetworkTask
          networktask = new NetworkTask();
    }
    // ---------------------- barraActuador EVENTLISTENER -
Comienzo ----------------------------
    SeekBar.OnSeekBarChangeListener
eventoMovimientoBarraActuador = new
SeekBar.OnSeekBarChangeListener(){
```

```
        // Este metodo se ejecuta cuando la barraActuador es
desplazada
        // en ese momento se guarda el valor de la barra, se
actualiza el numero valorBarraAcuador
        // y se envia al microcontrolador
        //@Override
            public void onProgressChanged(SeekBar seekBar, int
progress, boolean fromUser) {
                String valueOfseekbar = String.valueOf(progress);
                valorBarraActuador.setText(valueOfseekbar);

        networktask.SendDataToNetwork("Actuador:"+valueOfseekbar+'
\n');
            }

            //@Override
            public void onStartTrackingTouch(SeekBar seekBar) {
            }

            //@Override
            public void onStopTrackingTouch(SeekBar seekBar) {
            }
    };
    // ---------------------- barraActuador EVENTLISTENER - fin
----------------------------
// ---------------------- botonConexion EVENTLISTENER -
Comienzo ----------------------------
    private OnClickListener eventoClicBotonConexion = new
OnClickListener() {
        public void onClick(View v){
```

```
            // El botonConexion:
            // Si no hay conexion, se conecta, crea una
instancia de networkTask
            // y comunica al usuario que se esta conectando
            // Si la hay, se desconecta y lo comunica.
            if(!bConexion){
                  outputText("Conectando con Arduino");
                  networktask = new NetworkTask(); //New
instance of NetworkTask
               networktask.execute();
            }else{
                  outputText("Desconectando de Arduino...");
                  if(networktask!=null){
                        networktask.closeSocket();
                        networktask.cancel(true);
                  }
            }
        }
    };
 // ---------------------- botonConexion EVENTLISTENER - fin --
--------------------------



 // ---------------------- NETWORK TASK - Comienzo -----------
    public class NetworkTask extends AsyncTask<Void, byte[],
Boolean> {
```

```
        Socket nsocket; //Network Socket
        InputStream nis; // Flujo de Entrada
        OutputStream nos; // Flujo de Salida
        // Los bytes de entrada son almacenados en el buffer de
entrada,
        BufferedReader inFromServer;

        @Override
        protected void onPreExecute() {
             // Cuando la tarea comienza se cambia el estado de
conexion a conectado
            cambiarEstadoConexion(true);
        }

        @Override
        // En otro hilo de ejecucion se lleva la conexion
        protected Boolean doInBackground(Void... params) {
            boolean result = false;
            try {
             // Se crea la instancia del socket de conexion
                SocketAddress sockaddr = new
InetSocketAddress("192.168.1.177", 8888);
                nsocket = new Socket();
                // Se conecta con un tiempo limite de 5 segundos
                nsocket.connect(sockaddr, 5000);
                // Cuando se conecta obtiene los flujos de
entrada y salida
                if (nsocket.isConnected()) {
```

```
                    nis = nsocket.getInputStream();
                    nos = nsocket.getOutputStream();
                    // El flujo de entrada se adjunta al buffer
de lectura
                    inFromServer = new BufferedReader(new
InputStreamReader(nis));
                    // Mientras este conectado:
                    // Se leen las lineas que entran y almacenan
en un array
                    while(true){
                          String msgFromServer =
inFromServer.readLine();
                          byte[] theByteArray =
msgFromServer.getBytes();
                          publishProgress(theByteArray);
                    }
                }
            // Excepciones
            } catch (IOException e) {
                e.printStackTrace();
                result = true;
            } catch (Exception e) {
                e.printStackTrace();
                result = true;
            } finally {
             closeSocket();
            }
            return result;
        }

        //Metodo de cierre del socket
```

```
        public void closeSocket(){
             try {
                nis.close();
                nos.close();
                nsocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        // Metodo de envio de datos
        public void SendDataToNetwork(String cmd) {
            try {
                if (nsocket.isConnected()) {
                    nos.write(cmd.getBytes());
                } else {
                   outputText("SendDataToNetwork: No se puede
enviar mensaje. Socket cerrado");
                }
            } catch (Exception e) {
             outputText("SendDataToNetwork: Error en envio de
mensaje. Excepcion");
            }
        }
```

```
        // Metodo llamado cada vez que un string es recibido por
el socket de conexion
        @Override
        protected void onProgressUpdate(byte[]... values) {
             // Si el dato recibido tiene al menos un byte
            if (values.length > 0) {
             // Se obtiene la cadena de los valores recibidos
                String command=new String(values[0]);
                // Si la cadena comienza con "Sensor"
                if(command.indexOf("Sensor:")==0){
                    // Se elimina la parte "Sensor:" y se toma el
dato recibido
                    command=command.replace("Sensor:", "");
                    datoRecibido = Integer.parseInt(command);
                    setBarraSensor(datoRecibido);
                    mGraph.addDataPoint(datoRecibido);
                }
            }
        }


        // Si la tarea se cancela, se establece el
EstadoConexion en desconectado
        @Override
        protected void onCancelled() {
            cambiarEstadoConexion(false);
        }
```

```
        // Tras la ejecucion de la conexion se llama a este
metodo
        @Override
        protected void onPostExecute(Boolean result) {
            if (result) {
                outputText("onPostExecute: Completado con un
Error.");
            } else {
             outputText("onPostExecute: Completado.");
            }
            // Por ultimo se cambia el estado de la conexion a
desconectado
            cambiarEstadoConexion(false);
        }
    }
    // ---------------------- NETWORK TASK - Fin --------------
--------------



    // Metodo que efectua el cambio del estado de la conexion
       public void cambiarEstadoConexion(Boolean isConnected) {
             bConexion=isConnected;//change variable
             // Habilita/deshabilita la barra Actuador
             barraActuador.setEnabled(isConnected);
             // Habilita/deshabilita la barra Sensor
             barraSensor.setEnabled(isConnected);
             // Si la conexion se establece o cierra,
```

```
            // Lo muestra en el log, y cambia el texto del
botonConexion


            if(isConnected){
                  outputText("Conectado con Arduino");
                  botonConexion.setText("Desconectar");
            }else{
                  outputText("Desconectado de Arduino!");
                  botonConexion.setText("Conectar");
            }
      }


      // Metodo que cambia la barraSensora y su valor
       public void setBarraSensor(int position) {
            barraSensor.setProgress(position);
            String valorDeDatoRecibido =
String.valueOf(datoRecibido);
            valorBarraSensor.setText(valorDeDatoRecibido);
       }


      // Metodo que publica en el log todas las entradas
      public void outputText(String msg) {
            textlog.append(msg+"\n");
      }
```

```
       // Metodo llamado cuando cierra la aplicacion
    @Override
    protected void onDestroy() {
        super.onDestroy();
        if(networktask!=null){//In case the task is currently
running
                    networktask.cancel(true);//cancel the task
             }
    }
}
```

C. Presupuesto

- Autor: Mario De la Horra Köllmer.
- Departamento: Automática y electrónica.
- Descripción del proyecto: Sistema de adquisición de datos de bajo coste
- Duración del proyecto: 6 meses.
- Desglose presupuestario:

Personal			
Apellidos y nombre	**Categoría**	**Dedicación (hombres mes)***	**Coste por mes (€)**
BARBER CASTAÑO, RAMÓN IGNACIO	Ingeniero Sénior	0,80	4.289,54
DE LA HORRA KÖLLMER, MARIO	Ingeniero	3,00	2.694,39
		Total	**14.946,434 €**

Equipos					
Descripción	**Coste (€)**	**% Uso dedicado proyecto**	**Dedicación (meses)**	**Periodo de depreciación**	**Coste imputable (€)****
Portátil MacBook	1100	5	4	60	9,33
Portátil HP	700	5	2	60	3
Soldador	100	10	2	60	0,33
				Total	**12,67 €**

Costes de materiales				
Descripción	**Empresa**	**unidades**	**coste imputable**	**Total (€)**
Microcontrolador	Arduino Uno	1	26	26
·Entorno de desarrollo (libre)	Arduino Mega	1	50	0,00
Módulo ethernet	Arduino	1	39.325	39.325
·Librerías ethernet.h (libre)	GNU	1	0	0,00
Convertidor DC-DC	sparkfun	1	12.43	12.43
			Total	**127,75 €**

Resumen de costes	
Presupuesto Costes Totales	**Costes**
Personal	14.946,43
Subcontratación	12,67
Costes materiales	127,75
Total	**15.086,85 €**

NOTAS:

Presupuesto estimado para un sistema con 1 puesto.

* 1 Hombre mes = 131.25 horas. Máximo de dedicación de 12 hombres mes (1575 horas).

** Fórmula de cálculo de la amortización:

$$\frac{A}{B} \cdot C \cdot D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado.

B = periodo de depreciación (60 meses).

B = periodo de depreciación (60 meses).

D = % del uso que se dedica al proyecto.

D. Glosario

DAQ	*Data Acquisition*
IO	*Input/Output*
LED	*Light Emitting Diode*
LDR	*Light Dependent Resistor*
USB	*Universal Serial Bus*
RX	*Receive*
TAD	*Tarjeta de Adquisición de Datos*
TX	*Transmit*
WLAN	*Wireless Local Area Network*