UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Evaluación de Raspberry3 para adquisición de datos en entornos de laboratorio

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELÉCTRICA

Autor: **Guillermo Silvente Niñirola**

Director: Ángel Molina García

Cartagena, 8 de septiembre de 2017

Índice de contenido

Índice de ilustraciones 3
Objetivos y motivación 5
Raspberry Pi y Arduino. Características y descripción 6
Raspberry Pi 6
Introducción 6
Descripción 7
Hardware 7
Software 10
Usos 14
Arduino 16
Introducción 16
Descripción 17
Hardware 17
Software 19
Usos 22
Diferencias Raspberry PI – Arduino 23
Arduino y Raspberry juntos 25
Montaje y programación 28
Hardware 28
Software y código 33
Código de Arduino 33
Código de Raspberry Pi 43
Resultados 57
Conclusiones 66

Índice de ilustraciones

Ilustración 1: Raspberry PI 3 Modelo B 6
Ilustración 2: Pines Raspberry Pi modelo B 8
Ilustración 3: Numeración de pines según BOARD O BCM 9
Ilustración 4: Escritorio de Raspbian 10
Ilustración 5: Escritorio de Fedora 11
Ilustración 6: Escritorio de Arch Linux 11
Ilustración 7: Escritorio Kano OS 12
Ilustración 8: Escritorio de Windows IoT Core 12
Ilustración 9: OSMC 13
Ilustración 10: OpenElec 13
Ilustración 11: Arduino UNO 16
Ilustración 12: PWM 17
Ilustración 13: IDE de Arduino 19
Ilustración 14: Ejemplos de IDE de Arduino 20
Ilustración 15: Librerías de Arduino 21
Ilustración 16: Código de Arduino 26
Ilustración 17: Código en Raspberry Pi 27
Ilustración 18: Montaje general 28
Ilustración 19: Conexión Raspberry - Arduino 29
Ilustración 20: Esquema de montaje Arduino 30
Ilustración 21: Conexión Arduino con el circuito 30
Ilustración 22: Conexión Arduino con el circuito 31
Ilustración 23: Fuente de alimentación 31
Ilustración 24: Osciloscopio 32
Ilustración 25: Configurar IDE de Arduino 33
Ilustración 26: Código ejemplo de lectura de tensión 34
Ilustración 27: Muestreo de datos 35
Ilustración 28: Código de lectura de datos sin delay 36
Ilustración 29: Código de Arduino con TimerOne 1/2 39
Ilustración 30: Código de Arduino con TImerOne 2/2 39

Ilustración 31: Código final de Arduino 1/2 41
Ilustración 32: Código final de Arduino 2/2 41
Ilustración 33: Código Raspberry en Python 1/9..... 43
Ilustración 34: Código Raspberry Pi en Python 2/9..... 45
Ilustración 35: Código Raspberry Pi en Python 3/9..... 47
Ilustración 36: Código Raspberry Pi en Python 4/9..... 48
Ilustración 37: Código Raspberry Pi en Python 5/9..... 51
Ilustración 38: Código Raspberry Pi en Python 6/9..... 53
Ilustración 39: Código Raspberry Pi en Python 7/9..... 54
Ilustración 40: Código Raspberry Pi en Python 8/9..... 55
Ilustración 41: Código Raspberry Pi en Python 9/9..... 56
Ilustración 42: Menú de resultados..... 57
Ilustración 43: Datos..... 58
Ilustración 44: Menú de gráficas 58
Ilustración 45: Gráfica de la muestra 1..... 59
Ilustración 46: Gráfica de la muestra 2..... 60
Ilustración 47: Gráfica de la muestra 3..... 61
Ilustración 48: Gráfica de la muestra 4..... 62
Ilustración 49: Gráfica de la muestra 5..... 63
Ilustración 50: Gráfica de todos los datos 64
Ilustración 51: Valor eficaz 65
Ilustración 52: Valor medio 65
Ilustración 53: Tensión de pico..... 65

Objetivos y motivación

La motivación de este estudio es medir de forma precisa una onda senoidal de tensión dada por un generador de funciones en un entorno de laboratorio.

El objetivo de este proyecto es conseguir que la lectura sea precisa y fiable, la forma de medición será en un determinado tiempo que se irá ajustando a lo largo del estudio, esperando para realizar la siguiente lectura y volviendo a medir así sucesivamente para n muestras, este proceso se hará mediante el microcontrolador Arduino Uno y el computador de placa reducida Raspberry Pi 3 model B.

En el presente documento se van a analizar las características generales de cada dispositivo, detallar el método seguido para lograrlo, así como las incidencias habidas, los resultados obtenidos, las limitaciones y ventajas de los dispositivos, así como las conclusiones extraídas de este estudio.

Raspberry Pi y Arduino. Características y descripción

Raspberry Pi

Introducción

Raspberry Pi es un computador de placa reducida, computador de placa única o computador de placa simple (SBC) de bajo costo.

Este modelo es Raspberry Pi 3 model B cuenta con un procesador de 4 núcleos a 1,2 GHz, 1 Gb de memoria RAM, 4 puertos USB, salida HDMI, RJ-45, Bluetooth, entrada de vídeo con conector MIPI, almacenamiento para SD y una serie de pines GPIO que se explicarán a continuación.

Ilustración 1: Raspberry PI 3 Modelo B

En la imagen se muestra la Raspberry Pi, quedando a la derecha los puertos USB y el RJ-45, en la parte inferior la salida HDMI, a su derecha la entrada de vídeo y a su derecha la salida de audio, en la parte superior los pines.

Descripción

Hardware

Además de las características generales antes mencionadas, Raspberry Pi cuenta con pines GPIO (General Purpose Input/Output), son pines de entrada y salida digitales que son su interacción con el mundo exterior a través de sensores y controladores.

Estos pines no están protegidos, por lo que hay que tener cuidado con las tensiones que se manejan para no dañar la placa.

En general se pueden dividir en 4 grupos:

- **Pines de alimentación:** Se trata de pines de 3,3V y 5V limitados a 50mA en continua (si se superan estos valores se puede dañar) para alimentar los circuitos con los que se trabaje o también se pueden utilizar otras fuentes de alimentación externas.

- **Pines de tierra:** Estos pines están conectados eléctricamente entre sí, por lo tanto, no importa cual se escoja para la conexión del circuito, aunque no es significativo se recomienda utilizar el más próximo a la fuente de alimentación, pero no supone ninguna diferencia.

- **GPIO:** Estos pines son los utilizados para la conexión de sensores, envío de señales, conexión de LEDs etc.
  Se pueden programar en una gran cantidad de lenguajes que se explicará más adelante para que realicen la función deseada.

- **GPIO especiales:** Éstos tienen doble función además de la mencionada anteriormente, se encuentran varias categorías:

  - UART, con conexiones TXD y RXD que sirven para comunicaciones en serie, por ejemplo, para conectar con una placa Arduino.

- I2C que son utilizados para comunicación con periféricos externos como puede ser un MCP23017, que sirve para ampliar el número de pines que tiene la placa.
- SPI tiene funciones muy parecidas a I2C, pero utiliza protocolos distintos siendo este más simple, pero menos potente

Normalmente se utilizan como GPIO a secas, su distribución es la siguiente

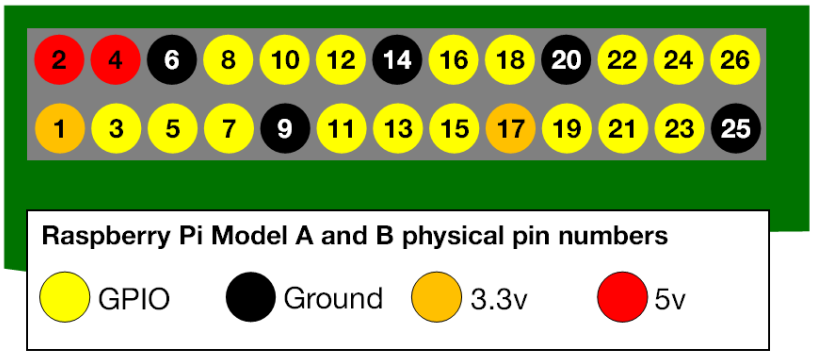

Ilustración 2: Pines Raspberry Pi modelo B

En la imagen se muestra la numeración que siguen los pines si a la hora de programar se utiliza el modo BOARD, esto es importante ya que en esta opción especifica se está refiriendo a los pines por su número, es decir los números impresos en nuestra Raspberry Pi, suele ser la opción más utilizada y las más intuitiva.

También está la opción BCM se refiere a los pines por su número de "Broadcom SOC channel", estos no son correlativos como en el modo BOARD

En esta imagen se muestra la diferencia entre los dos modos

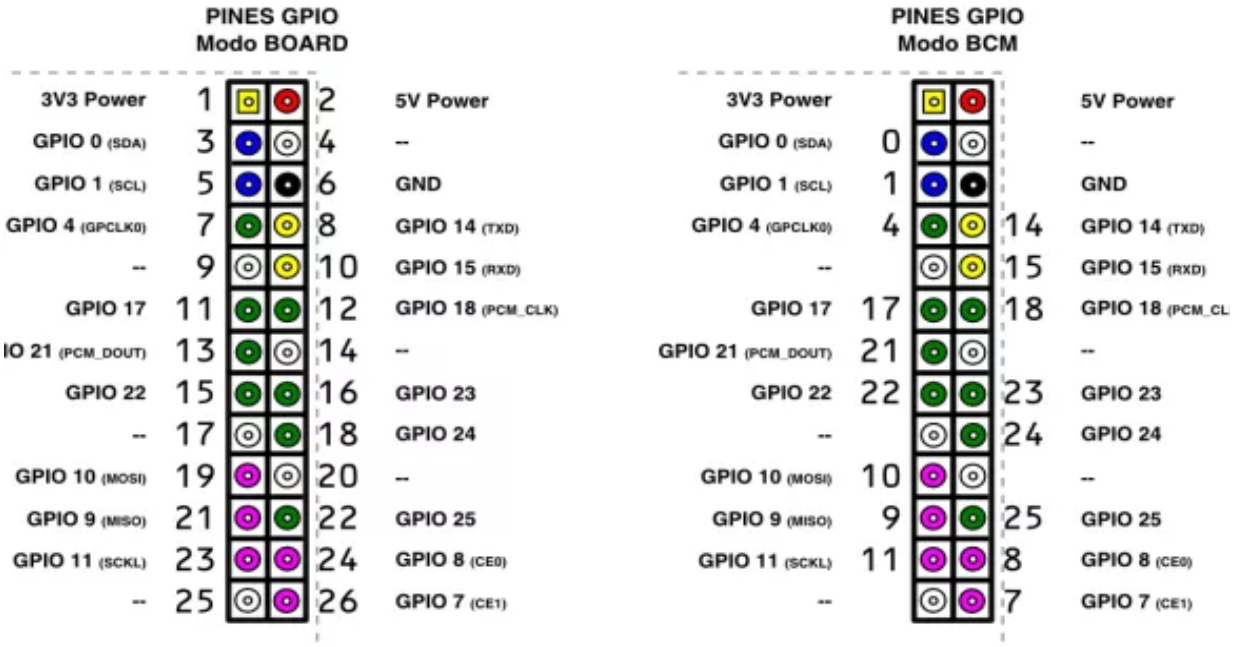

Ilustración 3: Numeración de pines según BOARD O BCM

Software

En primer lugar, Raspberry Pi necesita una tarjeta SD para poder instalar el sistema operativo, análogamente con un PC sería el disco duro sin él no puede funcionar y como cualquier otro PC, puede tener una gran variedad de S.O. según la función que se quiera desempeñar con la PI.

Los sistemas operativos más utilizados son:

- **Raspbian:** es el S.O. oficial, por excelencia de Raspberry PI y el más completo, se basa en un entorno de Linux relativamente sencillo de utilizar, también cuenta con una serie de programas por defecto como LibreOffice, Mathematica, herramientas para programar, incluso algún que otro juego.

  Este es el sistema elegido para el proyecto programándolo en Python 2.7 a través de la herramienta que se muestra en la imagen, se verá en profundidad más adelante.

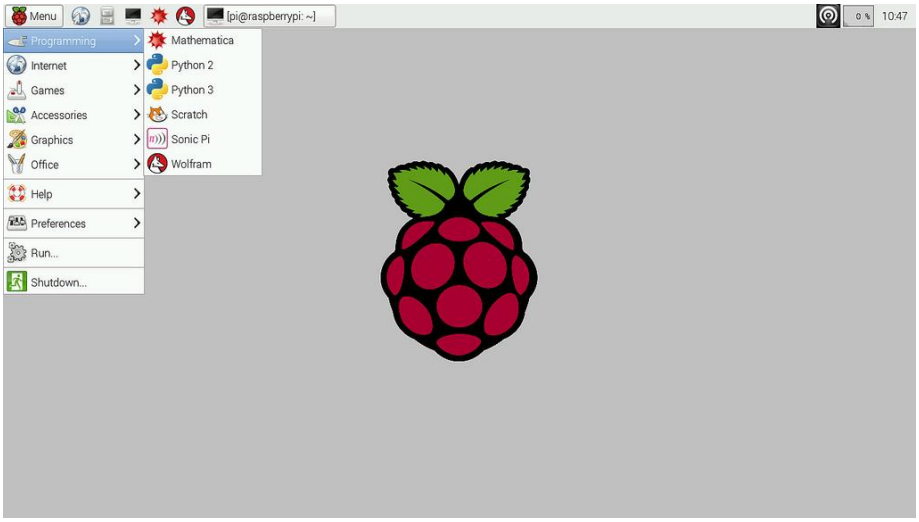

Ilustración 4: Escritorio de Raspbian

- **Fedora:** también basado en Linux sencillo y con una amplia documentación online para todos los niveles, también es una buena opción

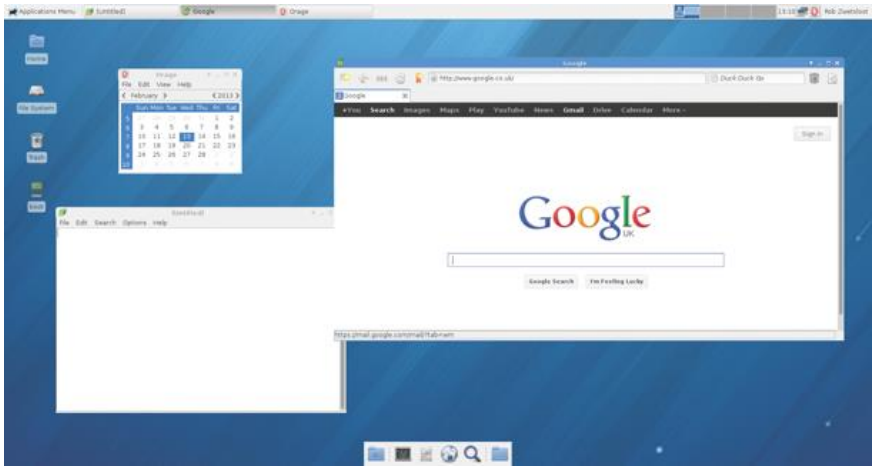

Ilustración 5: Escritorio de Fedora

- **Arch Linux:** es otro S.O. basado en Linux solo que más ligero y simple que los anteriores en caso de querer probar un Linux distinto.

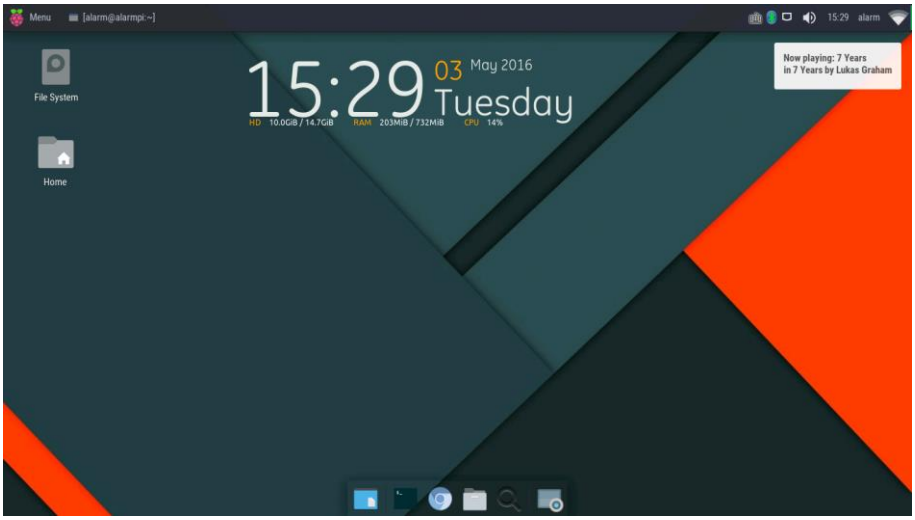

Ilustración 6: Escritorio de Arch Linux

- **Kano OS:** este sistema está orientado más a niños, como se ha comentado según el uso que se le dé a la PI, puede ser una opción interesante.

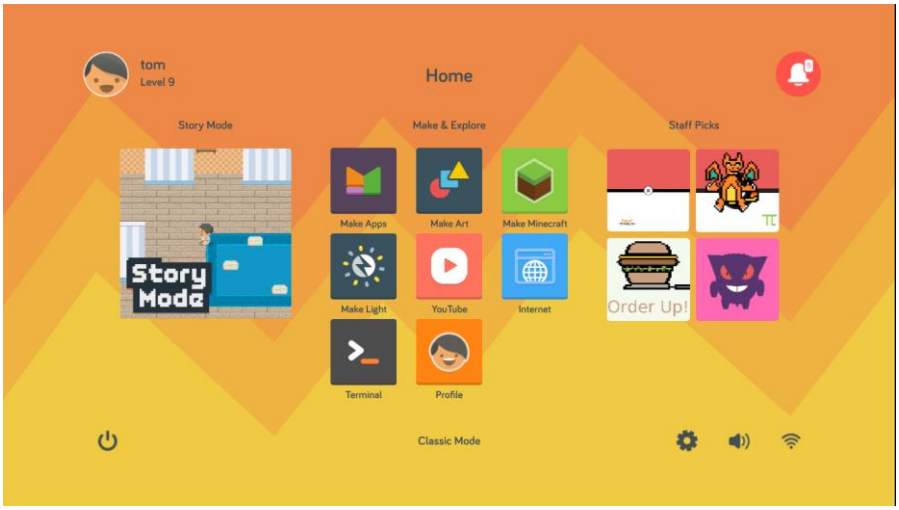

Ilustración 7: Escritorio Kano OS

- **Windows IoT Core:** se trata de un sistema operativo de Windows, pero es más bien una plataforma de desarrollo para que los programadores experimenten con dispositivos conectados a Internet.

Ilustración 8: Escritorio de Windows IoT Core

En cambio, si lo que se busca es un gestor multimedia, como pueda ser para ver películas, series o escuchar música, funciona como un media center y se puede controlar a través del smartphone. Los más populares son OSMC y OpenElec.

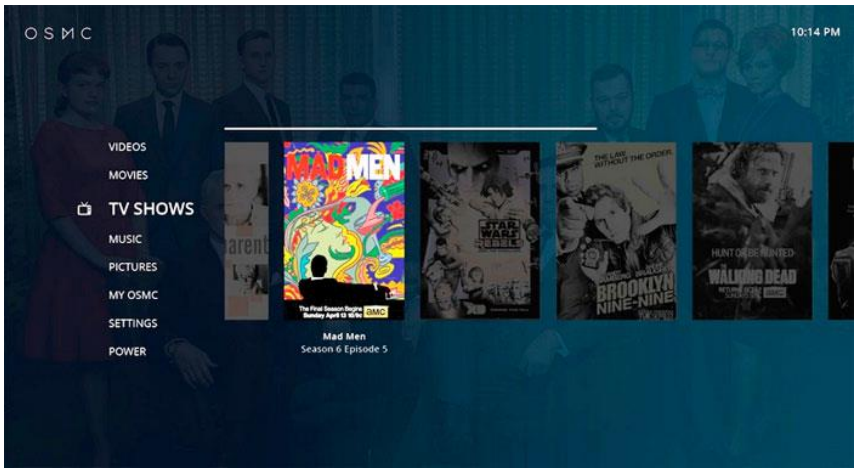

Ilustración 9: OSMC

Ilustración 10: OpenElec

Usos

La cantidad de usos que se le puede dar a este dispositivo es inimaginable, debido a la gran cantidad de sensores que puede utilizar además de ser ampliables y que es como un PC, por lo que puede transmitir datos por internet, bluetooth, comunicación serial, etc.
A continuación, se mencionarán brevemente algunos usos que se pueden dar

- **Maestro de otros dispositivos:** para la realización de un proyecto más complejo que la PI no pueda abarcar sola o como en éste para controlar Arduino y realizar otras funciones con él como tomar datos analógicos.

- **Estación meteorológica:** con una pantalla que muestre la previsión, sensores de humedad, temperatura, presión del aire niveles de luz y radiación ultravioleta, niveles de monóxido de carbono o de dióxido de nitrógeno, etc. Todos estos datos se podrían enviar al teléfono móvil, PC, web…

- **Servidor:** es posible también utilizarlo como todo tipo de servidor, web, correo, descargas, además de conservar el anonimato en la red a través de Tor.

- **Domótica:** automatizar una serie de funciones de la vivienda como encender las luces con los dispositivos móviles, un detector de movimiento con una cámara como seguridad, controlar la alimentación de la mascota cuando se está de viaje

- **Punto de acceso Wifi:** en caso de que no llegue bien la señal a algún punto de la vivienda se puede utilizar de esta forma

- **GPS:** se puede utilizar como GPS con una pantalla para utilizarlo en el coche

- **Escáner e impresora 3D:** este proyecto es más ambicioso ya que necesita al menos 40 Raspberry Pi, aunque es posible y muy interesante ya que se pueden escanear objetos de gran tamaño incluso personas.

Esto son solo algunos ejemplos generales de lo que se puede hacer, con esto se quiere demostrar la versatilidad que tiene ya que se le puede conectar casi cualquier tipo de sensor para hacer lo que se requiera.

Arduino

Introducción

Arduino es un microcontrolador electrónico de placa reducida de código abierto que tiene entradas y salidas analógicas y digitales, cuyo coste es ideal para la realización de pequeños proyectos de automatización y electrónica, procesar información y ejecutar órdenes.

El modelo elegido es el Arduino UNO, este cuenta con 14 pines digitales de los cuales 6 son PWM (pulse-width modulation), 6 entrada analógicas, 32Kb de memoria flash y 16 MHz de velocidad de reloj.

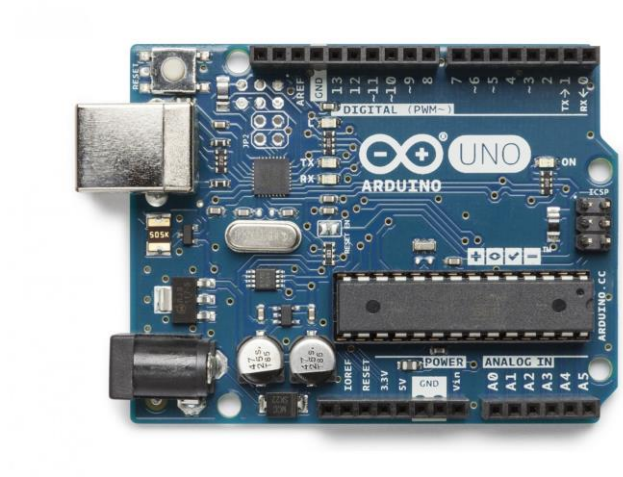

Ilustración 11: Arduino UNO

En la imagen se ve una placa Arduino UNO típica, los pines analógicos abajo a la derecha numerados por A0, A1, … A su lado los pines de alimentación y tierra al igual se tienen en Raspberry PI, arriba los pines digitales, el puerto USB y la alimentación.

Descripción

Hardware

El hardware de Arduino en comparación con el de Raspberry Pi es más simple, a continuación, se explicarán más a fondo los pines antes mencionados.

- **Analógicos:** los cuales le da gran versatilidad al poder leer este tipo de señales, no se le puede introducir más de 5V y 20mA de lo contrario la placa podría dañarse, así como no introducir valores de tensión negativos, arrojará un resultado 0 o nulo además de que también puede dañarse. Estos pines pueden tomar datos a una velocidad de algo más de 0.1 ms en una escala de 0-1023 que corresponde a 0-5 V, para obtener el valor en voltios hay que multiplicar por 5 y dividir por 1023.

- **PWM:** pulse-width modulation conocido por sus siglas en inglés, que quiere decir modulación por ancho de pulsos, esta técnica en lo que consiste es simular una señal periódica analógica (como una onda senoidal) mediante una serie de pulsos de una señal digital a través de la modificación de la frecuencia y el ciclo de trabajo.

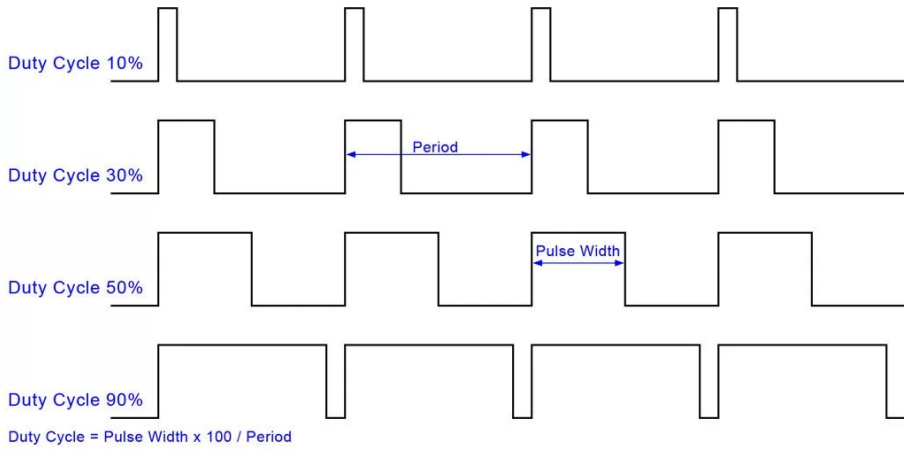

Ilustración 12: PWM

- **Digitales:** Se trata de simples entradas/salidas digitales, reciben o envían señales de todo/nada, true/false, 0/1, high/low, o la designación que se quiera.

- **Alimentación:** son muy parecidos a los de Raspberry, se dividen en los siguientes:
  - **Ground:** los dos pines de tierra.
  - **5V y 3,3V:** por donde suministra esa tensión en continua al circuito que se quiera alimentar en caso de necesitarla.
  - **Vin:** es otra entrada de alimentación a la placa en lugar de la convencional que está al lado del puerto USB.
  - **Reset:** para reiniciar la placa, también se puede utilizar el botón de reset al lado del USB.
  - **AEREF:** sirve para referenciar a otra escala de tensión el voltaje que se mide en la entrada analógica, por ejemplo, que se reescale de 0-1.5V, pero nunca fuera del rango 0-5V.
  - **IOREF:** es el pin que nos suministra la tensión para el estado alto de los pines digitales. Por norma general si conectamos el Arduino a 5V este pin nos dará 5V, en caso contrario serán 3.3V.

Software

Arduino de por sí no puede tener un sistema operativo, necesita de un elemento externo para introducir su programación en él, normalmente se utiliza el IDE (*Integrated Development Environment*) de Arduino de código abierto que se programa en un lenguaje basado en C++, es el método utilizado en este proyecto.

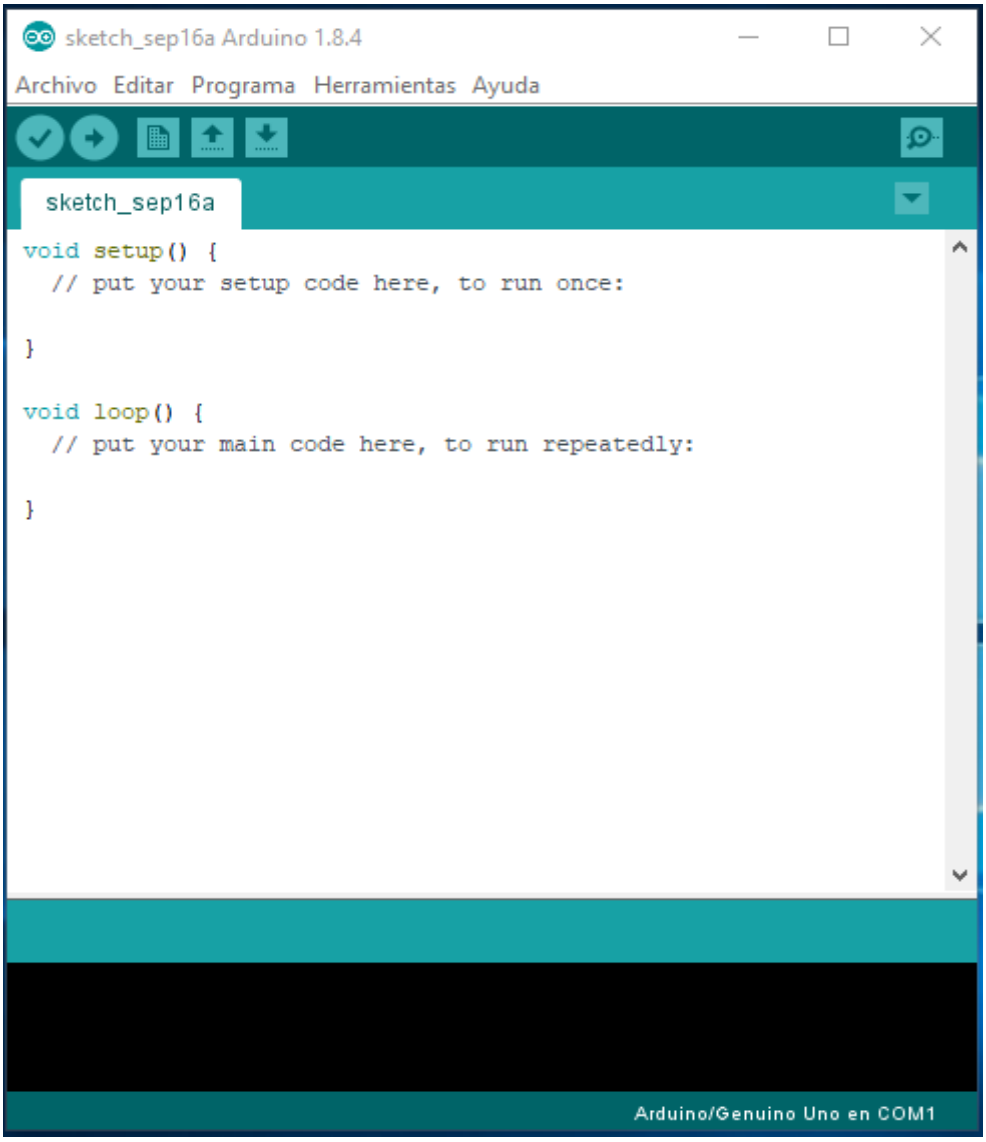

Ilustración 13: IDE de Arduino

La estructura básica del IDE de Arduino cuenta con un editor de texto donde se programa, una consola de errores debajo donde notifica los posibles conflictos o información, una barra superior de archivos, edición, herramientas y ayuda y otra barra debajo con las órdenes básicas como verificar el código, cargar programación, nuevo archivo, cargar archivo y una opción de monitorizar los datos que salen de Arduino.

En la opción Archivo cuenta con una serie de ejemplos predeterminados muy interesantes, desde hacer parpadear un led, un servidor web, lecturas analógicas, punto de acceso Wifi y muchos más ejemplos que pueden ser de utilidad a la hora de realizar un proyecto propio. Para poder utilizarlos, evidentemente hay que tener los sensores y dispositivos que así lo requiera cada ejemplo.

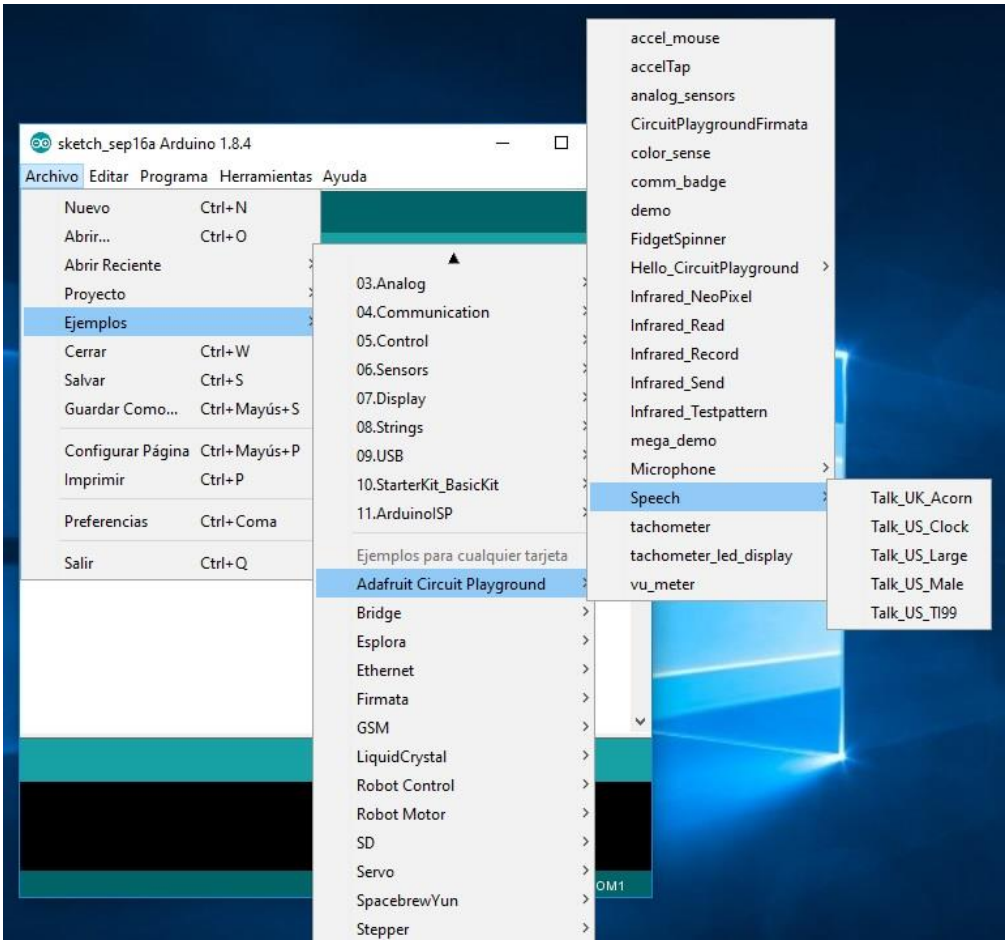

Ilustración 14: Ejemplos de IDE de Arduino

En la pestaña de programa se puede encontrar la opción de añadir una serie de librerías predeterminadas, además de poder descargar y añadir otras que se quieran utilizar, como TimerOne que se verá en la parte del código desarrollado.

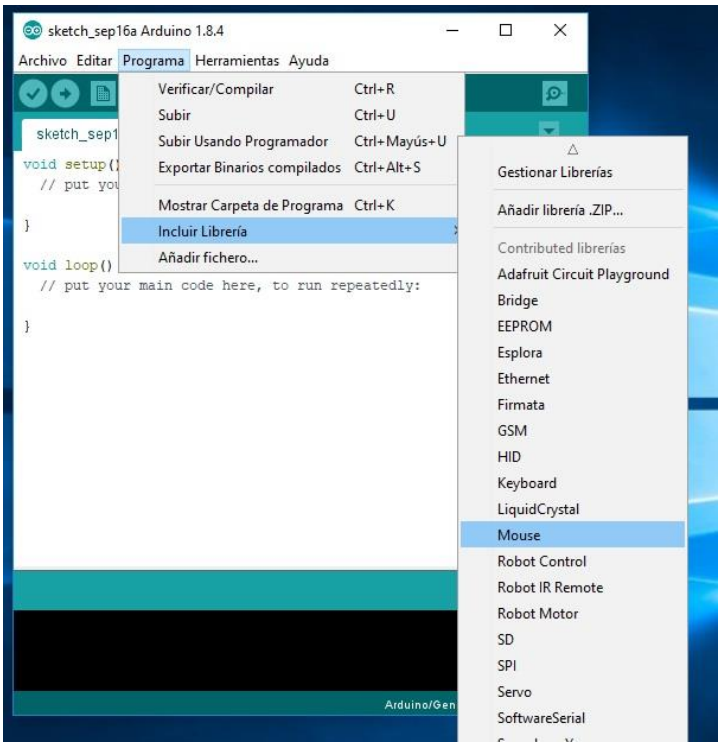

Ilustración 15: Librerías de Arduino

Usos

Arduino tiene unos usos similares a Raspberry, pero también otros distintos, a continuación, se verán unos ejemplos:

- **Mediciones analógicas:** con los pines analógicos de Arduino se pueden medir multitud de señales como por ejemplo una onda de tensión, que es el objeto de este proyecto y se desarrollará más adelante.
- **Con PWM:** con esta técnica vista anteriormente, se puede controlar el ventilador de un PC, para mantener la temperatura de la CPU sin que sobrepase un cierto nivel, sin esta técnica el ventilador estaría funcionando a máxima velocidad continuamente, lo que es un desaprovechamiento de energía si el PC no está funcionando a máxima potencia. Con PWM se puede regular la velocidad de forma sencilla para una mejor optimización. Este es un ejemplo sencillo para entender su utilidad ya que los PCs lo incorporan de serie, este método en general se puede utilizar para todo tipo entradas digitales que requieran regulación o simular una señal analógica.
- **Alarma para vivienda o local:** con ayuda de sensores de movimiento se puede programar para que cuando detecten algo, activar sonidos, luces, capturar fotografías…
- **Sistema de riego:** es posible programar un sistema de riego mediante Arduino para una explotación agrícola o para un uso más doméstico.
- **Control domótico de la vivienda:** al igual que Raspberry es posible hacer una vivienda domótica de bajo coste.

En general tiene muchas aplicaciones parecidas a Raspberry, además de las vistas en los ejemplos del IDE, pero Arduino es más sencillo de utilizar y está más orientado a proyectos electrónicos, se explicarán a continuación las diferencias entre cada uno.

Diferencias Raspberry PI – Arduino

A continuación, se mostrará una tabla resumiendo las características más importantes de cada uno y se analizarán

Característica / Dispositivo	Raspberry Pi 3 Modelo B	Arduino Uno
Memoria RAM	1 Gb	2 Kb
Velocidad de reloj del procesador	1.2 GHz con 4 núcleos	16 MHz
Memoria flash	Tarjeta SD	32 Kb
Voltaje de entrada	5 V	7 a 12 V recomendado
Tamaño	8.6cm x 5.4cm x 1.7cm	7.6 x 1.9 x 6.4 cm
Sistema Operativo	Raspbian (entre oros)	-
Entradas analógicas	-	6
Entradas/Salidas digitales	17	14 (6 PWM)
Puertos USB	4	1

A simple vista Raspberry es mucho más potente que Arduino, su velocidad de reloj del procesador es 75 veces más rápida además de tener 4 núcleos, la memoria RAM es 512000 veces mayor, además de soportar salida HDMI, teclado, ratón, 4 puertos USB y entrada para cámara.

Raspberry tiene que mover un sistema operativo multitarea y se beneficia de 30 años de desarrollo de Linux, lo que le aporta una gran ventaja en lo que a aplicaciones mediante software se refiere, en esto es claramente mejor. En cambio lo que puede ser el punto débil de Arduino, el no contar con sistema operativo, es sin embargo su punto fuerte para otras funciones, debido a la simplicidad y flexibilidad, hace que éste sea mejor para los proyectos de hardware, ya que se trata simplemente de conectar y programar.

Arduino tiene la capacidad analógica y en tiempo real que la PI no, esto le da una flexibilidad que le permite trabajar con casi cualquier tipo de chip o sensor que Raspberry no tiene y requiere de hardware adicional.

A la hora de la programación el Arduino IDE es más sencillo de utilizar que Linux, para hacer parpadear un LED, con unas 10 líneas de código es suficiente, en cambio en la PI, es necesario instalar un sistema operativo, descargar e incluir una serie de librerías correspondientes si no están ya descargadas y programarlo, el resultado final es el mismo, pero más sencillo con Arduino.

Como se ha dicho Raspberry PI se beneficia del SO de Linux, por lo que es multitarea, es decir, que puede estar ejecutando múltiples programas a la vez, como un servidor de impresión y uno web, mientras parpadean unos leds, en cambio Arduino está limitado a un proceso al mismo tiempo, se le carga el programa y ejecutará sólo eso, sólo puede realizar una única tarea a la vez.

No se puede decir que uno sea mejor que el otro, como siempre, depende siempre del proyecto que se vaya a llevar a cabo.

Si se trata de simple lectura de datos de sensores, accionamiento de motores, control de LED… es mejor Arduino. En cambio, si es un proyecto más complejo donde es necesaria la multitarea, mayor complejidad de cálculo, múltiples procesos y envío de datos, Raspberry PI es más adecuada ya que aporta una gran capacidad computacional y un entorno gráfico.

Raspberry tiene un hardware mucho más potente, lo que hace que no sea fácil utilizarla mediante baterías de forma autónoma en cambio Arduino consume mucho menos y se puede alimentar sin problemas con una batería.

También es posible utilizar Raspberry PI como controlador y Arduino como actuador, lo que haría que el proyecto se beneficie de las ventajas que aportan los dos, incluso en proyectos más complejos es posible conectar varios Arduino que procesen una serie de señales y la PI como maestro recibiendo los datos, procesándolos y controlando los Arduinos.

Arduino y Raspberry juntos

La conexión Arduino – Raspberry puede ampliar el rango de posibilidades de lo que pueden hacer por separado, la idea de la comunicación entre los dos es que sea Arduino el que se encargue de la parte de hardware y Raspberry sea el maestro que controle el proceso, para ello hay multitud de formas para su conexión como pueden ser utilizar un módulo Ethernet, mediante el propio cable USB, por Bluetooth, etc.

La forma elegida es por USB ya que suele ser la más sencilla y para este objetivo es la mejor, para ello primero hay que preparar los dos dispositivos.

La parte de Arduino es más sencilla, se descarga el IDE de Arduino en la PI, en el PC o incluso en el móvil, se escribe el código y se carga en Arduino, lo ideal es descargarlo en la Raspberry Pi y programar desde ahí. Se elige la placa utilizada y el puerto de conexión, normalmente se hace automáticamente, pero es necesario revisarlo.

Para la parte de Raspberry previamente habrá que descargar e instalar la librería que permite la comunicación serial entre los dos dispositivos, para ello se entra en la consola y se escribe el siguiente código:

apt-get install python-serial

Con esto ya está descargada la librería que luego hay que importar en el script, a continuación, se mostrará un ejemplo muy simple del control de unos LEDs.

El código de Arduino

```
const int led = 13;

void setup(){
  pinMode(led, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  if (Serial.available()){
    char c = Serial.read();
    if (c=='H'){
      digitalWrite(led,HIGH);
    }else if(c=='L'){
    digitalWrite(led,LOW);
    }
  }


}
```

Ilustración 16: Código de Arduino

Su función es declarar la variable led como el número 13, la función void setup() es la función de inicio donde se declara el pin 13 como salida, para enviar señales y se establece la comunicación serial a 9600 baudios.

La función void loop() se va a estar repitiendo hasta que haya algún comando que la detenga, en ese caso no hay ninguno. Esta función lo que hace es que lee del puerto serie y cuando se recibe desde Raspberry Pi una “H”, el LED se encenderá, si se envía una “L” el LED se apagará, sin Raspberry tendrían que enviarse los comandos desde el PC o poner algún temporizador, pero ya no estaría controlado.

El código de Raspberry está en Python y es el siguiente:

```
import serial

arduino = serial.Serial('/dev/ttyACM0',9600)


while True:
        comando = raw_input('Introduce un comando: ')
        arduino.write(comando)
        if comando == 'H':
                print('LED ENCENDIDO')
        elif comando == 'L':
                print('LED APAGADO')

arduino.close()#FINALIZA LA COMUNICACION
```

Ilustración 17: Código en Raspberry Pi

En primer lugar, se importa la librería para la comunicación serial, sin ella no se puede hacer, después se establece la comunicación serial con Arduino en el puerto especificado y a 9600 baudios, es importante que sea la misma velocidad o no funcionará correctamente.

En este caso está funcionando continuamente y pide un valor de entrada, se puede escribir cualquier valor y se enviará, pero Arduino no hará nada a no ser que sea “H” o “L”.

Este es un ejemplo sencillo a modo de introducción de lo que se puede hacer con los dos combinados, más adelante se explicará en profundidad el funcionamiento del código empleado.

Montaje y programación

A continuación, se explicará el método de montaje de los dispositivos, fuente de alimentación y resistencia, así como la programación empleada para llegar al objetivo de medir una onda de tensión.

Hardware

El montaje general consiste en que la Raspberry Pi alimenta a Arduino a través de un cable USB y éste mide la tensión en un circuito de una resistencia de 100 Ohmios.
También se utiliza un adaptador VGA a HDMI alimentado por el PC debido a que el monitor utilizado no tiene entrada HDMI.

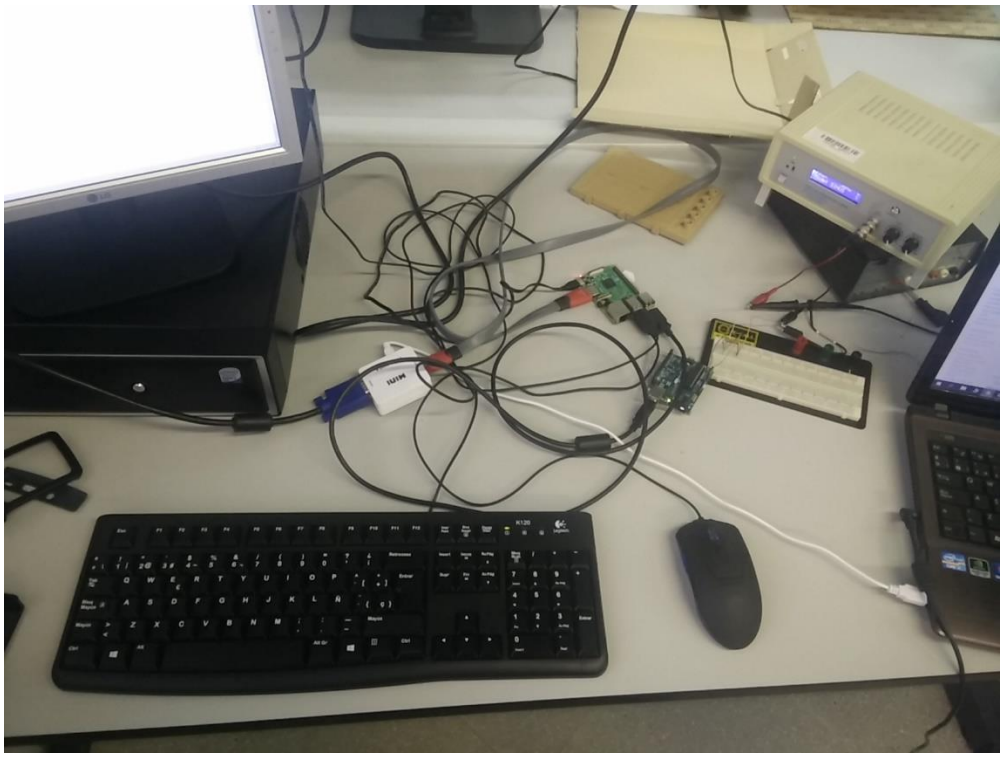

Ilustración 18: Montaje general

A la Raspberry PI además del cable USB que alimenta al Arduino también se le conecta un teclado, un ratón y un cable HDMI

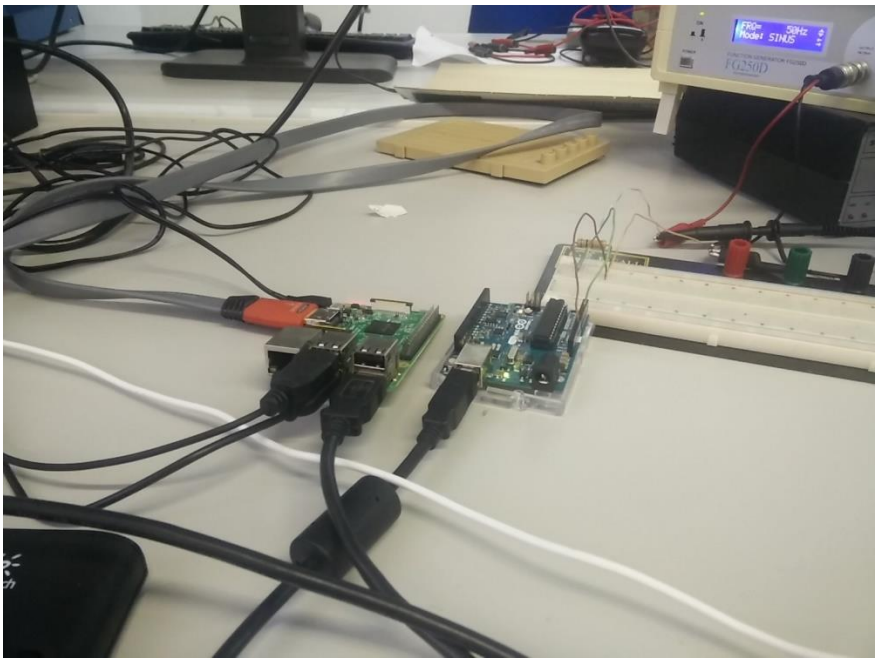

Ilustración 19: Conexión Raspberry - Arduino

El circuito de se trata tan sólo de una resistencia de 100 Ohmios, donde el pin A0 se conecta al positivo del circuito y la tierra al negativo.

La fuente de alimentación aporta una onda senoidal de 50 Hz, como se ha comentado Arduino no soporta tensiones negativas, arroja un valor nulo, 0 o incluso podría dañarse, para evitar este problema lo que se ha hecho es añadir una señal de corriente continua de 1,5 V que el mínimo de la onda senoidal esté por encima de 0, esto se ha hecho mediante la función Offset de la propia fuente de alimentación, como resultado se obtiene una onda de casi 3 V de tensión de pico que tampoco supera los 5 V de máximo.

En el esquema se representa el circuito con la fuente de alimentación y el osciloscopio realizado con el programa Fritzing, y a continuación están las fotos

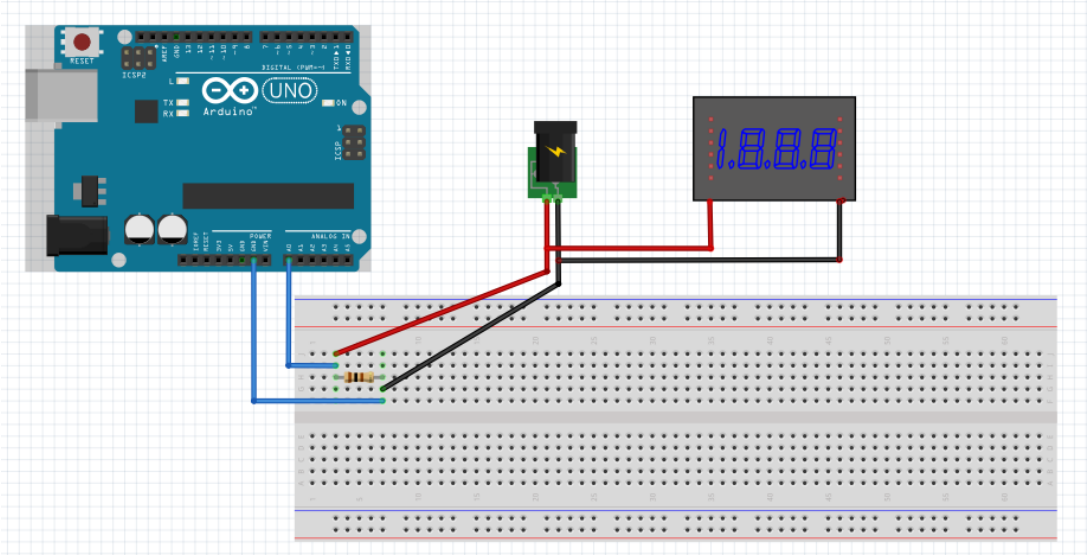

Ilustración 20: Esquema de montaje Arduino

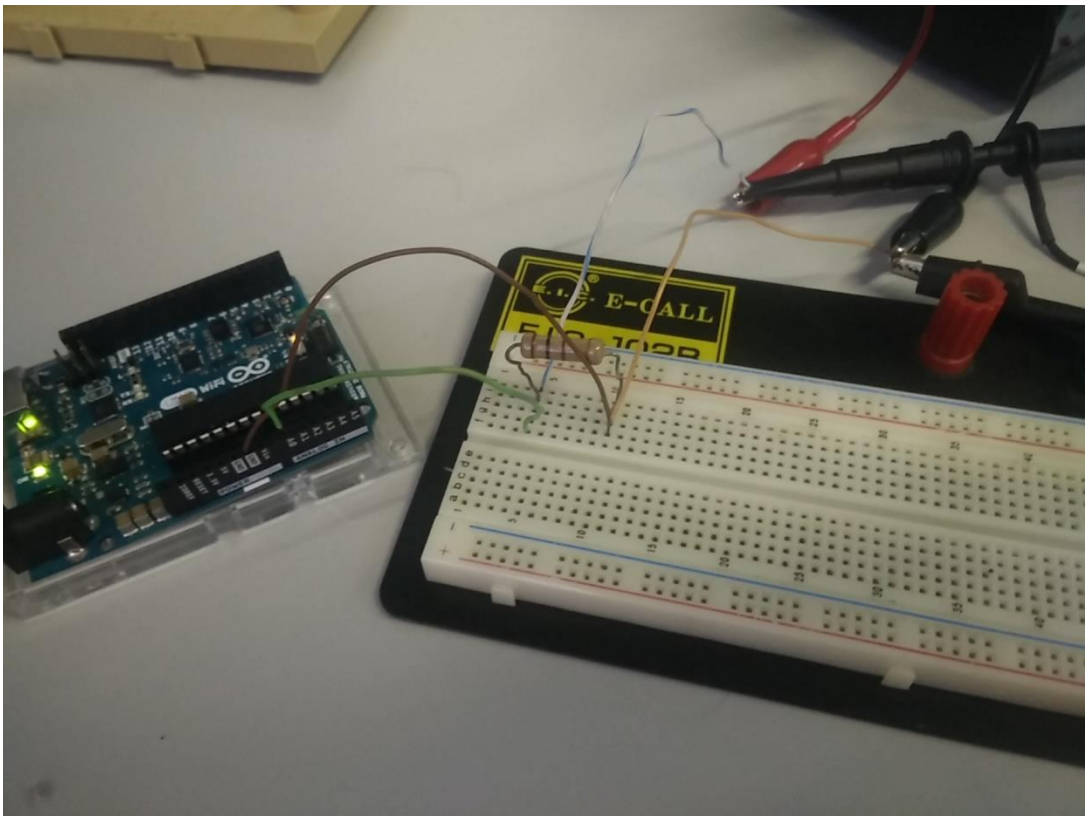

Ilustración 21: Conexión Arduino con el circuito

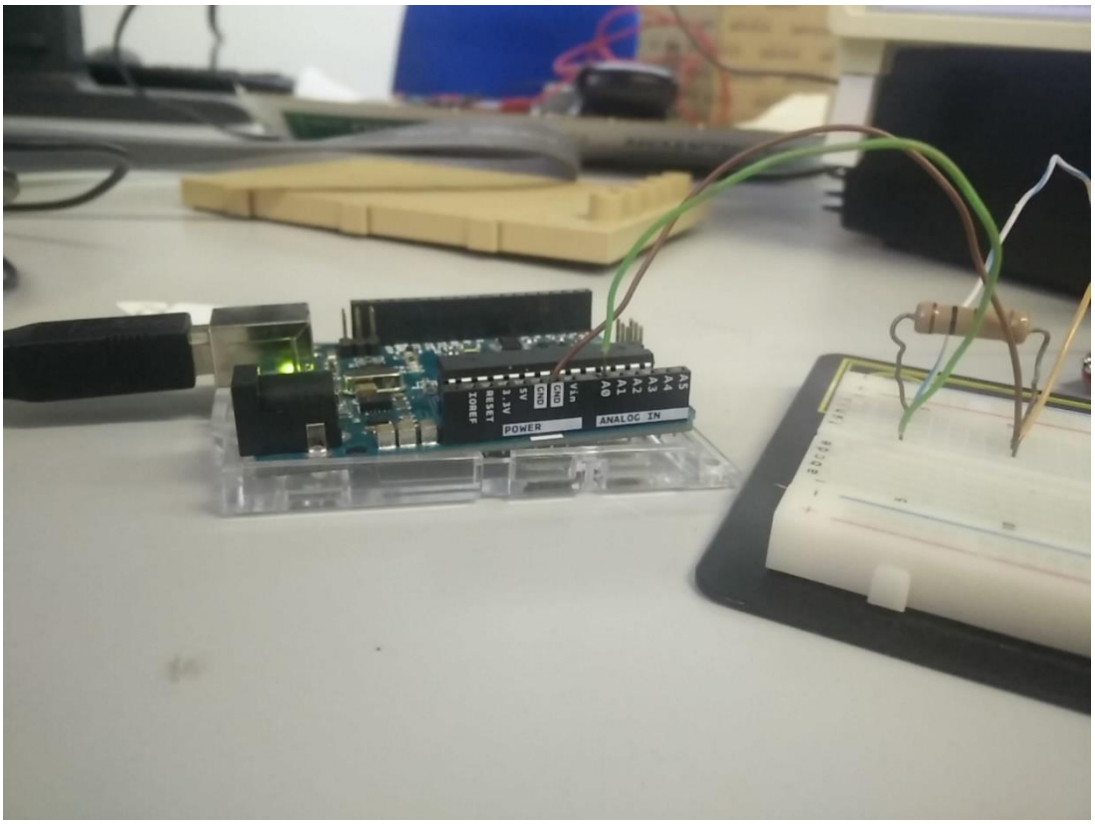

Ilustración 22: Conexión Arduino con el circuito

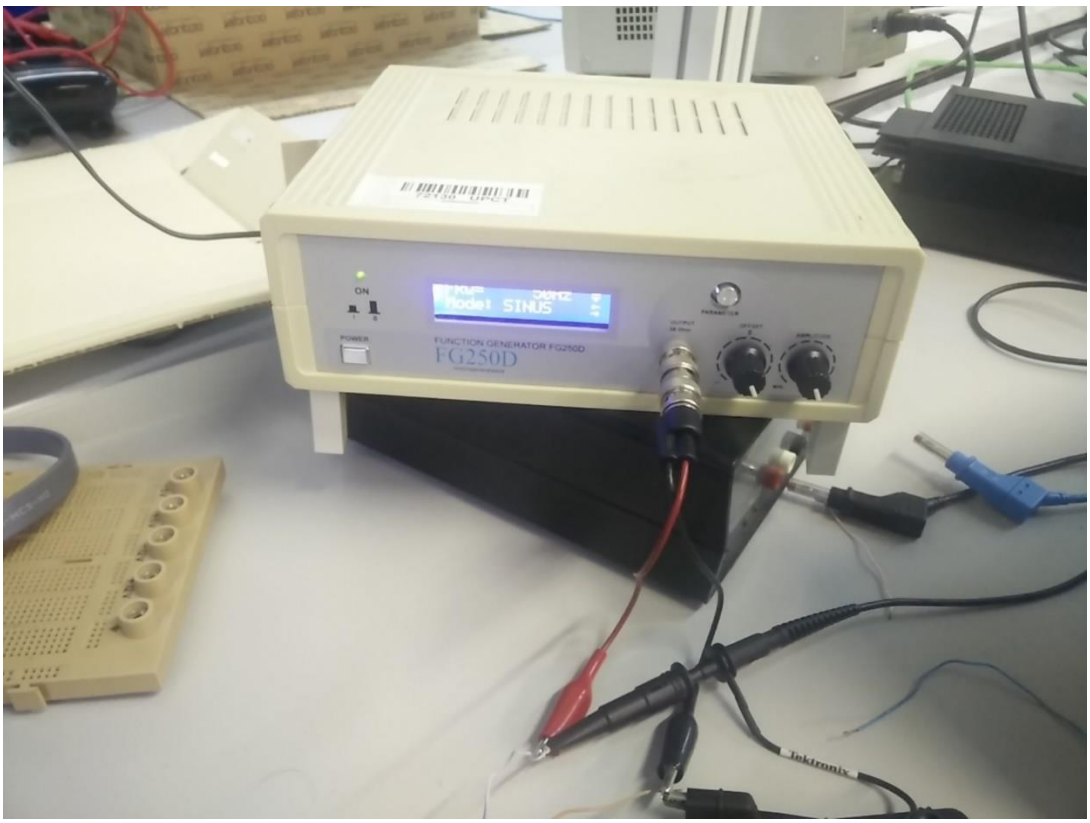

Ilustración 23: Fuente de alimentación

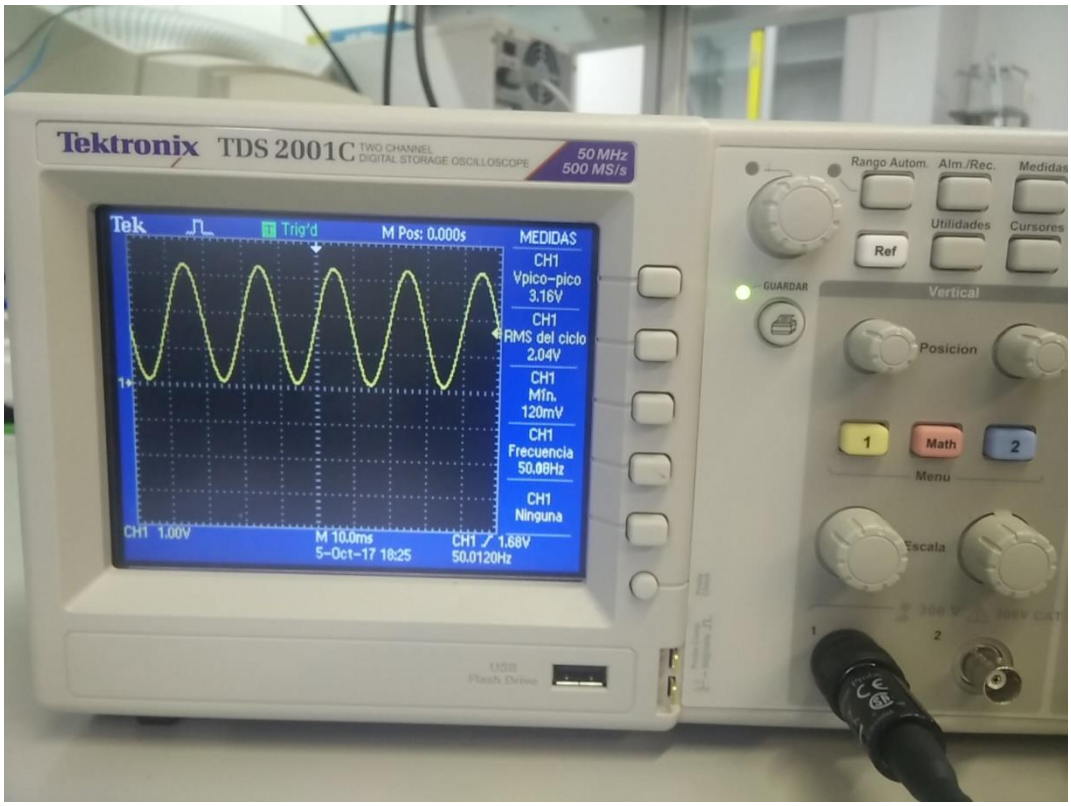

Ilustración 24: Osciloscopio

Software y código

Mediante el código programado, lo que se ha conseguido es tomar 5 muestras de 90ms con un tiempo de espera de 5s cada una, a continuación, se explicará el proceso seguido para llevar a cabo esta medida, así como los problemas que han surgido a la hora de tomar medidas.
En ese apartado sólo se explicará el código, la presentación de resultados será en su apartado correspondiente.

Código de Arduino

Para poder comunicarse con Arduino hay que seleccionar la placa que se está usando, en este caso Arduino Uno y el puerto correspondiente situado justo debajo, normalmente se configura automáticamente, pero hay que comprobarlo.

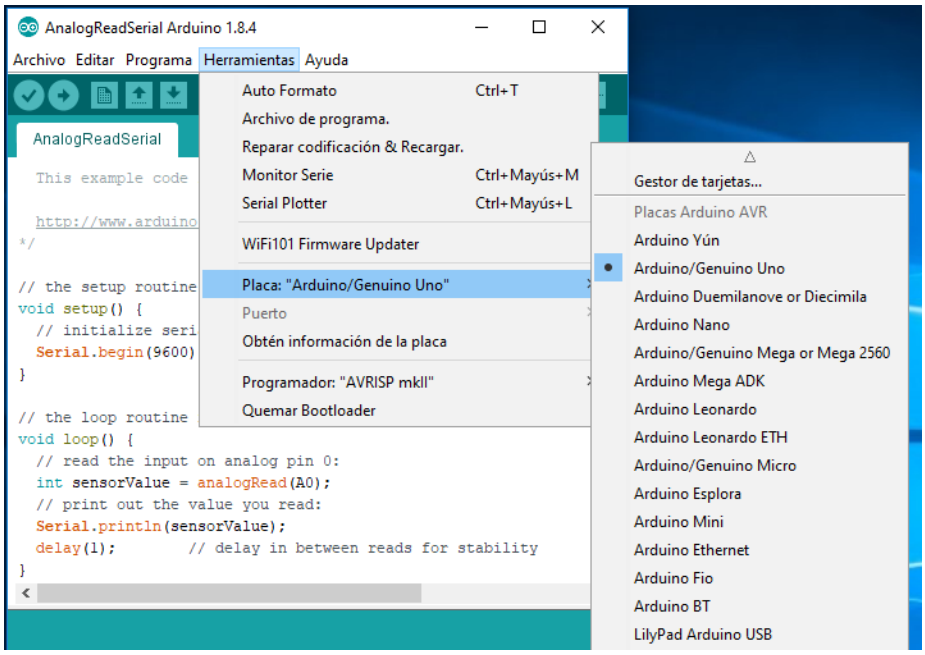

Ilustración 25: Configurar IDE de Arduino

Al principio se tomó como base el código que viene por defecto en el IDE de Arduino para medir tensión

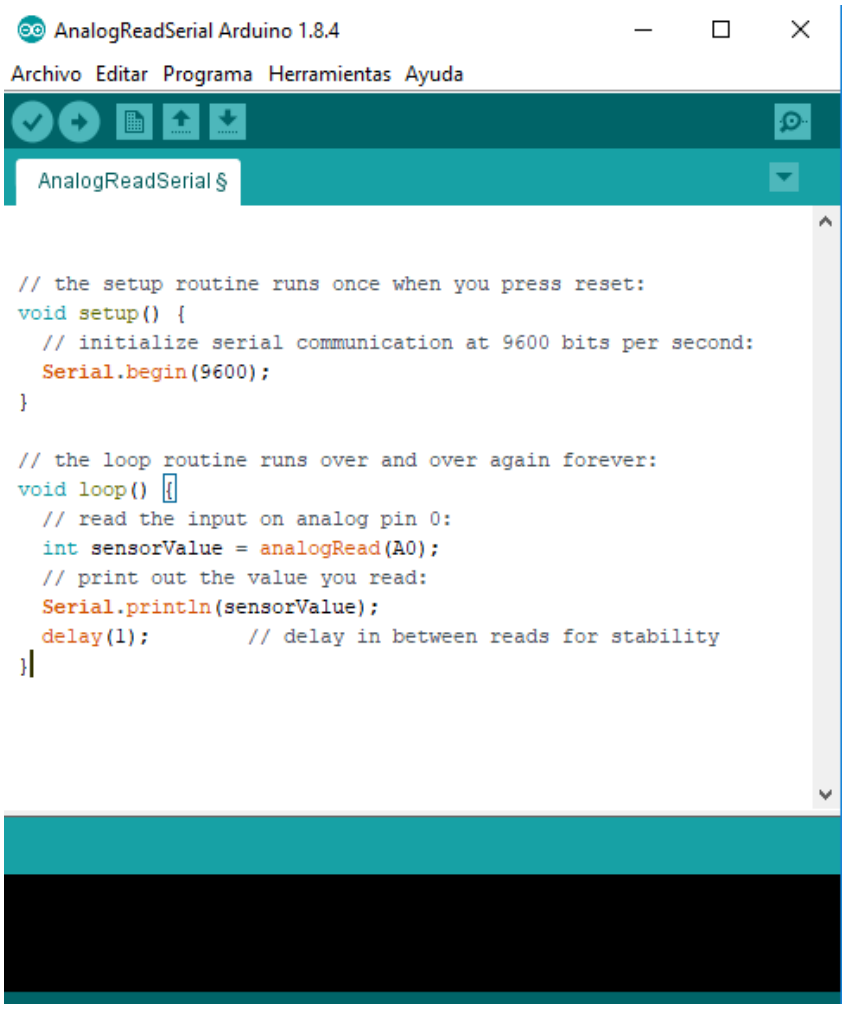

Ilustración 26: Código ejemplo de lectura de tensión

En la función void setup() abre la comunicación serial por USB a 9600 baudios.

La función void loop(), lo que hace es repetir continuamente el siguiente proceso:
Con el comando analogRead se le comunica a Arduino que tome un dato en el pin especificado entre paréntesis en este caso A0 y lo guarda en la variable sensorValue, tras esto envía el dato con el comando Serial.print o Serial.println, con ln al final indica un salto de línea, es el que se utilizará por defecto, finalmente, con la función delay() se espera el tiempo especificado en ms para volver a hacer el ciclo.

Al probar este método lo que se hizo fue poner un contador y cuando llegara a 100 se detuviera, de ese modo una lectura cada 1ms serían 100 ms, para mayor seguridad, se envió el tiempo de medida junto con el dato.

Los resultados no eran correctos, se saltaba muchos datos del ciclo por lo que se optó por subir la velocidad de transmisión notablemente de 9600 a 115200 baudios y se repitió la prueba. Esta vez todos los datos eran consecutivos y no se saltaba ninguno, en 100 ms tomaba 250 medidas, lo que es sensiblemente más lento a las especificaciones de Arduino, además en lugar de durar 100 ms, el proceso duraba alrededor de 140-150 ms y como se puede apreciar en los datos a continuación al principio de cada muestra, en los primeros ms se tomaban más datos de al final.

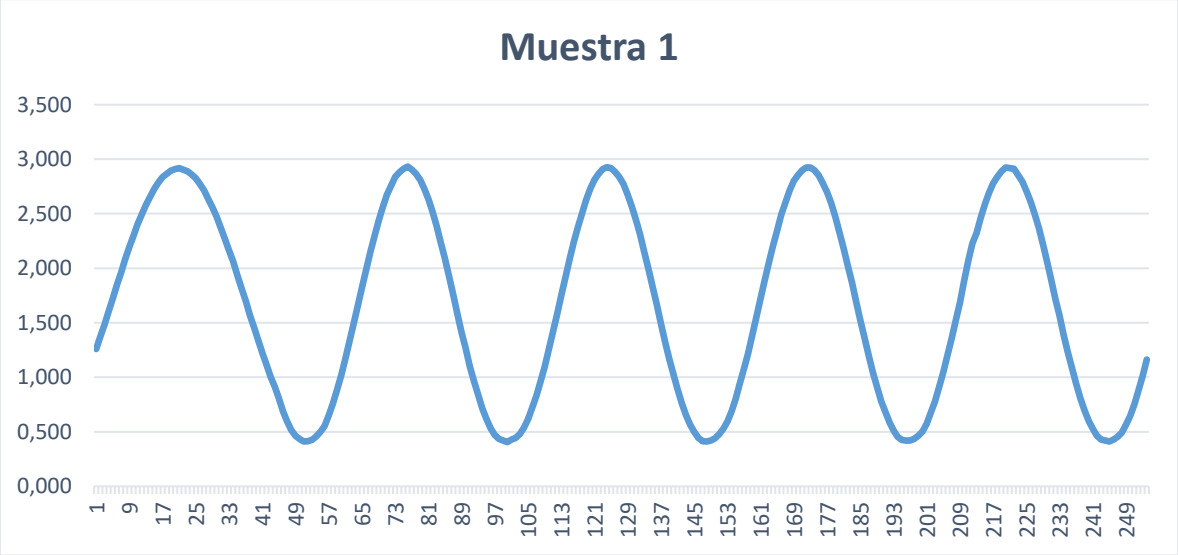

Ilustración 27: Muestreo de datos

Como se aprecia el primer ciclo es sensiblemente mayor a los siguientes, de unas 60 muestras, cuando los demás son de 48-50.

Este problema en parte es causado por la función delay() que como se ha comentado lo que hace esperar un determinado tiempo, esta espera lo que realmente hace es bloquear el Arduino y que no pueda realizar ninguna función durante un tiempo, esto con ciclos tan rápidos y tan seguidos a la larga retrasa el proceso.

Para solucionarlo se puso una pausa con tiempo, donde se omite el principio del código ya que es igual que el anterior, sólo que, con 115200 baudios, el código es el siguiente:

```
void loop(){
      while(cont<muestras){
         while(tiempo-t_actualizado<t_lectura){
           tiempo=millis();
           int sensorV = analogRead(A0);
           Serial.println(sensorV,DEC);
           }

         }

        tiempo=millis();
        t_actualizado=tiempo;
        while(tiempo-t_actualizado<sampletime){
           tiempo=millis();
        }
        cont=cont+1;
        tiempo=millis();
        t_actualizado=tiempo;
        }
    }
```

Ilustración 28: Código de lectura de datos sin delay

Lo que hace este código es:

Un ciclo while que está activo mientras la variable cont (el contador) sea menor que el número de muestras, la variable cont se define como en 0 previamente y repetirá el código tantas veces como muestras se hayan puesto.

En la siguiente línea hay otro while con una resta entre las variables tiempo y t_actualizado que se inicializan a 0 y t_lectura es el tiempo que está midiendo, por lo tanto se repetirá el ciclo hasta que la diferencia entre esas dos variables sea mayor que t_lectura, en la siguiente línea se tiene que tiempo=millis(), esto indica que la variable tiempo toma el tiempo y se va actualizando e incrementando mientras t_actualizado permanece en 0, por lo que cuando llegue a 100 ms ese ciclo while se detendrá.

Las siguientes dos líneas ya se han visto que es la lectura y almacenamiento en una variable y el envío de la información en decimal.

Al terminar el ciclo while se actualizan las dos variables de tiempo para igualarlas y aquí es donde entra el sustitutivo de la función delay, se crea otro while igual que el anterior, pero esta vez que no haga nada, simplemente cuando supere el tiempo de espera de 5000 ms continua con el proceso, de esta forma Arduino no se bloquea en ningún momento, al terminar la espera, se vuelve a actualizar las variables del tiempo y se incrementa el contador.

Con este método los resultados salían muy parecidos a los mostrados antes 250 medidas, muy lejos de las aproximadamente 1000 que debería tomar, ya que en teoría toma alrededor de 0.1 ms por lectura y el primer ciclo seguía saliendo distinto.

Tras probar con otro tipo de soluciones como limpieza de buffer y haber aumentado del envío de la velocidad de datos, que también son significativas, pero no definitivas, se dio con la solución, la función Serial.println() que es la que envía los datos, es lo que se considera como una interrupción por software y actualmente Arduino no las soporta.

Una interrupción es en este caso una llamada por software para realizar una acción sí o sí, interrumpe el funcionamiento normal de Arduino y lo ejecuta, es lo que se llama Interrupt Service Handler o ISH (Servicio de gestión de interrupción).

Cuando el ISH finaliza, el procesador vuelve tranquilamente al punto donde lo dejó y sigue con lo que estaba como si no hubiese pasado nada.

Realmente lo que está haciendo Arduino es leer constantemente los datos de tensión del pin A0 y tras cada lectura se le está interrumpiendo para que envíe datos lo que retrasa mucho el proceso final, además que el envío es más lento que la lectura.

En un ejemplo de la vida cotidiana, que puede no ser el más acertado, pero sirve para hacerse una idea sobre las interrupciones

Sería como si una persona está viendo la televisión (lectura del código) y tiene que hacer otra tarea (interrupción), es persona no se está levantando cada 2 minutos a hacer un poco de la tarea, vuelve a ver a televisión, vuelve a hacer la tarea, vuelve a ver la televisión, así sucesivamente, análogamente con Arduino es leer la señal, enviarla por el puerto serie, volver a leerla, enviarla…Al final esa persona no ha prestado atención al programa de televisión que estaba viendo y no hará la tarea correctamente, lo que debe hacer es ver la televisión y en los anuncios hacer la tarea, lo que en Arduino sería recolectar todos los datos, almacenarlos y después enviarlos.

Como solución a esto se encontró la librería Timer1 o TimerOne, lo que hace esta librería es causar una interrupción por software voluntaria, es decir, el código es el mismo, solo que se quita Serial.println() y se introduce en otra función que durante el tiempo de espera entre muestra y muestra donde no está leyendo datos, TimerOne interrumpe la función principal y llama a la de envío de datos, de esta forma la recogida de datos no se ve afectada.

```
#include <TimerOne.h>
volatile int a = 0;
volatile int n = 0;
volatile int cont = 0;
int muestras = 3;
int sampletime = 5000;
volatile int datos[810];
unsigned long tiempo = 0;
unsigned long t_actualizado = 0;
unsigned long t_lectura = 90;

void setup(){
  Serial.begin(115200);
  Timer1.initialize(4500000);
  Timer1.attachInterrupt(envio);
}

void loop(){
  while(cont<muestras)
  while(tiempo-t_actualizado<t_lectura){
    tiempo=millis();
    int sensorV = analogRead(A0);
    datos[n] = sensorV;
    n = n+1;
  }
```

Ilustración 29: Código de Arduino con TimerOne 1/2

```
  tiempo=millis();
  t_actualizado=tiempo;
  while(tiempo-t_actualizado<sampletime){
    tiempo=millis();
  }
  cont=cont+1;
  tiempo=millis();
  t_actualizado=tiempo;
}

void envio(){
  if(a<muestras){
    for (int i = 0; i<=810; i++){
        Serial.println(datos[i],DEC);
    }
  }
  a=a+1;
  n=0;
}
```

Ilustración 30: Código de Arduino con TImerOne 2/2

Como muestra el código lo primero es incluir la librería TimerOne, seguidamente se declaran las variables utilizadas, el comando volatile se recomienda al utilizar esta designación de variables por utilizar distintas funciones y se mantendrán a lo largo del estudio.

En la función void.setup() se inicializa la comunicación serial, esta vez notablemente más rápida, también se ordena que ejecute TimerOne cada 4,5 s ya que se espera 5s en cada muestra y se muestra cuál es la función a la que tiene que llamar.

La toma de datos es casi igual que los anteriores, pero esta vez no se incluye Serial.println, lo que se hace es leer el dato y almacenarlo en una lista de 810 huecos, se incrementa el valor de n para guardar el siguiente dato en el siguiente hueco de la lista. También se reduce el tiempo de lectura de 100 ms a 90 ms, debido a que el gran tamaño de la lista que ocupa el 89% de la memoria dinámica de Arduino y es el límite para que sea estable.

El tiempo de espera es igual que el anterior.

Cuando el tiempo llega a 4,5 s TimerOne se ejecuta, interrumpe void.loop() y llama a void.envio() donde mediante un bucle for se envían los datos, al terminar, se inicializa la variable n para volver a tomar los datos y se incrementa la variable a, que cumple la función para que se ejecute sólo las veces que se muestreen, como en void.loop().

Con este método los resultados salían perfectos, gran precisión debido a las casi 800 muestras en 90 ms y ningún ciclo mayor que otro (los cuales se muestran en el apartado de resultados), pero tiene un problema.

En el código mostrado, se toman 3 muestras cada 5 s y TimerOne interrumpe cada 4,5 s lo que al final son 15 s y TimerOne 13,5 s lo que es un desfase entre las dos de 1,5 s. Si se quieren tomar 10 muestras, el ciclo de recogida tardaría 50 s y el de interrupción 45 s lo que daría lugar a una interrupción durante la medida y para 11 muestras serían 55 y 49,5 s con lo que quedaría una muestra sin enviar.

Por lo que se llegó a la conclusión de que TimerOne no es la mejor opción para tomar muestras de esta forma, para evitar este problema en el caso de tomar más muestras, se desarrolló un código muy parecido y más sencillo.

```
volatile int i = 0;
volatile int n = 0;
volatile int cont = 0;
int muestras = 5;
int sampletime = 5000;
volatile int datos[809];
unsigned long tiempo = 0;
unsigned long t_actualizado = 0;
unsigned long t_lectura = 90;
void setup(){
  Serial.begin(115200);
}
```

Ilustración 31: Código final de Arduino 1/2

```
void loop(){
      Serial.flush();
        while(cont<muestras){
          while(tiempo-t_actualizado<t_lectura){
            tiempo=millis();
            int sensorV = analogRead(A0);
            datos[n] = sensorV;
            n = n+1;
            }
          while(i<=n){
          Serial.println(datos[i],DEC);
          i=i+1;
          }
        tiempo=millis();
        t_actualizado=tiempo;

        while(tiempo-t_actualizado<sampletime){
           tiempo=millis();
        }
        cont=cont+1;
        tiempo=millis();
        t_actualizado=tiempo;
        n=0;
        i=0;
        Serial.flush();
        }
}
```

Ilustración 32: Código final de Arduino 2/2

En este código se las variables son las mismas, se ha excluido la librería TimerOne y por lo tanto ya no está en la función void.setup().
Se ha añadido la función Serial.flush() al principio y al final para limpiar el buffer, el resto del código es igual que el anterior, excepto que la información se envía después de tomar las muestras, en un bucle while, tras esto se espera los 5 s y se inicializan las variables n para el ciclo de recogida de datos y la i para el ciclo de envío de datos.

Este es la programación definitiva que se ha utilizado, los datos son iguales que con TimerOne, pero en ese caso se pueden poner 5, 10 o 20 muestras, las que se quieran, pero no en más de 90 ms ya que se necesitaría una lista mayor y satura la memoria de Arduino, no por la complejidad del programa en sí, sino por la lista creada de 809 huecos donde se guardan los datos para enviarlos.

Código de Raspberry Pi

Ahora se explicará el código de Raspberry Pi escrito en Python 2.7, no es Python 3.4 que es la última versión de este lenguaje, aun así, apenas cambia de una versión a otra, sólo ciertos comandos sencillos.

Se mostrará el código por partes y se irá analizando

```
# -*- coding: utf-8 -*-
import pickle
from matplotlib.widgets import Cursor
import matplotlib.pyplot as plt
from matplotlib.ticker import EngFormatter
import serial
import time
import os

ajuste = 0
contador = 0
Vmax = 0
Vef = 0
Vmedio = 0
Vtotal = 0
n = 0
indices = []
indices2 = []
data = []
arduino = serial.Serial(
    port='/dev/ttyACM0',
    baudrate = 115200,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=0.2
)
arduino.setDTR(False)
time.sleep(0.2)
arduino.flushInput() #Limpiar el buffer
arduino.setDTR(True)
tension = []
datos = 0
tiempo=time.time()
tiempototal=30
t_actualizado=tiempo
```

Ilustración 33: Código Raspberry en Python 1/9

En primer lugar, se importan las librerías necesarias que se describirán a continuación:

- Pickle sirve para guardar y tomar los datos de un archivo de texto donde se guardan
- Matplotlib son módulos para la representación gráfica, donde el primero corresponde a un cursor para las gráficas, el segundo las gráficas como tal y el tercero para la leyenda del eje Y.
- Serial es el módulo que permite la comunicación USB con Arduino.
- Time es la librería que permite importar el tiempo y poder trabajar con él.
- OS es el módulo para poder trabajar y comunicarse con el sistema operativo Raspbian, en este caso para poder crear y tomar información de los ficheros de texto.

Se inicializan una serie de variables que se verán más tarde cuando al llegar a ellas.
En la siguiente línea se abre la comunicación serial con Arduino donde se define el puerto que por defecto será el que está indicado, luego se definen 3 variables que son la paridad, stopbits y el tamaño del byte, que por defecto se deja así a no ser que se quiera utilizar para aplicaciones más específicas, también está el timeout que es el tiempo que tarda en iniciar la comunicación serial, se ha elegido 0,2 s.

Las cuatro líneas siguientes sirven para limpiar el buffer en caso de que haya algún dato basura, ruido o distorsión.

Se inicializan otras 2 variables y finalmente el tiempo, como se comentó en la parte de Arduino, éste tiene la capacidad de trabajar sobre tiempo real, esto quiere decir que, al iniciar el programa, Arduino empieza a contar el tiempo desde 0, por eso en su código se han inicializado las variables de tiempo y t_actualizado a 0 donde la variable tiempo se iba incrementando hasta que la diferencia entre tiempo y t_actualizado llegara al tiempo de lectura.

En cambio, Raspberry Pi no tiene esa capacidad de ejecutar el programa y medir el tiempo desde 0, para ello lo que se hace es tomar el tiempo del sistema (para ello la inclusión de la librería time) para las dos variables de tiempo de esa forma ese será el “0”, también se define el tiempo de lectura que se ha estimado mayor que en Arduino para tener un margen, si en éste era de 25 s en total aquí se ha puesto de 30 s.

```
print ("Espere mientras se toman los datos, cuando finalice se mostrará un menú\n")

while ((tiempo-t_actualizado)<tiempototal):
        tiempo=time.time()
        read=arduino.readline().rstrip();
        tension.append(read)
for dato in range(0,len(tension)):
        tension[dato] = str(tension[dato]) #Convertir a string
        if tension[dato] == '':
                tension[dato]='0'
for i in range(0,len(tension)-1):
    tension[i] = float(tension[i])
    tension[i] = tension[i] * 5.0/1023.0
    tension[i] = str(tension[i])

fichero = file("Final5m.dat", "w")

pickle.dump(tension, fichero)

fichero.close()

def cls(): print "\n" * 100 #Limpiar pantalla
while True:
    print ("1. Para mostrar todos los datos")
    print ("2. Para mostrar los datos gráficamente")
    print ("3. Para mostrar el valor eficaz")
    print ("4. Para mostrar el valor medio")
    print ("5. Para mostrar el valor de pico")
    print ("6. Para salir")
    opcion = raw_input("Escoja una opción\n")
```

Ilustración 34: Código Raspberry Pi en Python 2/9

En la primera línea se indica que hay que esperar un tiempo mientras se toman los datos y al final \n, que es para un salto de línea.

El ciclo while es la misma idea que en Arduino, estar leyendo del puerto serial durante un determinado tiempo, se han igualado las variables al mismo tiempo, el del sistema y sólo se actualiza la de tiempo, así cuando sea mayor de 30 s la diferencia se detendrá.

Dentro de while se tiene la línea read=arduino.readline().rstrip(), esto lo que hace es leer del puerto serial llamado arduino con arduino.readline(), el nombre de la función seguido de la acción.
Este dato se guarda en la variable read y rstrip() es para limpiar los datos, en Arduino se puso que se enviara los datos con Serial.println que era con salto de línea para diferenciar los datos y no con Serial.print que es sin salto de línea, como se acaba de comentar en Python los saltos de línea son \n, por lo tanto si se envían los números 653, 688 y 699, se leería 653\n, 688\n y 699\n, para evitar eso se utiliza rstrip().
Finalmente, la variable read se añade a una lista vacía llamada tensión con el comando append(read).

El bucle for recorre la lista donde se han guardado las variables desde 0 hasta la longitud de la lista y convierte a cadena de caracteres cada dato de la lista.
En el tiempo que no se envía nada desde Arduino, Raspberry sigue leyendo, que lo interpreta como un hueco vacío “ “, para evitar problemas más adelante se añade un if, el cual sustituye el hueco “ “ por un “0” indicando que no hay medida. Esto se hace porque cuando hay que calcular, se tienen que convertir de cadena de caracteres a integradores, es decir, a números, Python no puede convertir un hueco en blanco a integrador en cambio un “0” sí.

En el siguiente bucle for se recorre la lista y se convierte en variable float, que es como un integrador, pero con decimales, se multiplican por 5 y dividen por 1023 y se vuelven a convertir a cadena de caracteres.
Esto se hace porque Arduino lee voltaje en una escala de 0-1023 y envía los datos así, que corresponde a 0 – 5 V.

Se crea la variable fichero = file(“Final5m.dat” , “w”), esto crea un archivo de texto de nombre Final5m.dat y la “w” indica que se ha abierto en modo escritura.
Con módulo importado pickle y con el comando pickle.dump(tension.fichero) se vuelcan los datos de la lista tension en el fichero creado y con la orden fichero.close() se cierra el archivo, de esta forma ya están los datos en un archivo de texto.

Se define una función que lo que hace es repetir 100 veces un salto de línea para poder utilizarla para limpiar la pantalla, ya que en Raspbian no se dispone de ningún comando específico para esto.

Se define un ciclo while a modo de menú para la presentación de los datos y cálculos realizados, que estará siempre activo a no ser que se pulse el 6 para salir, la opción elegida almacenada se guarda con el comando raw_input.

```
if opcion == '1':
    cls()
    fichero = None
    fichero = file("Final5m.dat")   #cargar fichero de datos
    tension2 = pickle.load(fichero)
    fichero.close()
    for i in range(0,len(tension2)): #redondear
        tension2[i] = float(tension2[i])
        tension2[i] = round(tension2[i],3)
    tension2 = zip(*[iter(tension2)]*8) # dividir en grupos de 8
    tension2 = [[str(x) for x in tup] for tup in tension2] #quitar tuple
    for row in tension2:
         output = [row[0].ljust(5)]
         for col in row[1:]:
             output.append(col.rjust(10))
         print ''.join(output)
         print '-------------------------------------------------------------------
```

Ilustración 35: Código Raspberry Pi en Python 3/9

Si se pulsa el 1 para mostrar los datos, se limpia la pantalla, se limpia la variable fichero, se abre el archivo donde se han guardado los datos, se almacenan en otra lista mediante pickle.load() y se cierra.

En el bucle for, se convierte a float, ya que estaba guardado como string (cadena de caracteres) y se redondea a 3 decimales.
En la siguiente línea, lo que se hace es dividir en grupos de 8 los datos para que su presentación sea más vistosa, ya que si los muestra directamente si imprimen por pantalla en lista, todos seguidos y es confuso. El problema es que se crea una tupla (tuple en inglés), que es la misma lista, pero los datos son fijos, no se pueden manipular, interesa poder modificarlos para poder mostrarlos en columnas para eso en la siguiente línea, con ese comando se puede quitar la tupla.

En el siguiente bucle for, se reordenan los datos para mostrarlos en columnas de 8 y se imprimen por pantalla seguido de una línea de guiones para facilitar su lectura al usuario.

```
elif opcion == '2':
    fichero = None
    graficas = None
    indices = None
    indices2 = None
    indices2 = []
    ajuste = None
    ajuste = 0
    n = None
    fichero = file("Final5m.dat")
    graficas = pickle.load(fichero)
    fichero.close()
    #Hallar los índices de la lista donde es 0
    indices = [i for i, x in enumerate(graficas) if x == '0.0']
    for i in range(0, len(indices)-1):
        if indices[i]-indices[i-1] > 1:
            indices2.append(indices[i-1])
            indices2.append(indices[i])
    n = len(indices2)/2 #Conocer el tamaño de la muestra
    while True:
        cls()
        for x in range(0,n):
            print(str(x+1) + ". Para representar la gráfica con la muestra " + str(x+1))
        print (str(n+1) + ". Para representar la muestra completa")
        print (str(n+2) + ". Atrás")
        comando = raw_input("Escoja una opción\n")
```

Ilustración 36: Código Raspberry Pi en Python 4/9

Si se pulsa 2 para ver las gráficas, se limpian todas las variables que se utilizan en esa opción para que al salir al menú y al volver a entrar no se sobrescriban y salgan las gráficas distorsionadas, además se carga el fichero de los datos, esta vez en la variable graficas.

En la siguiente línea se recorre la lista buscando en qué posiciones es 0, como está guardado en un string que previamente era un float, será con decimales “0.0”.
Con esto lo que se busca es ver en qué momento de la lista no se está midiendo, es decir es 0, la lista indices tendrá un valor de la forma [0,1,2,3,813,814,815,1622,1623] (en la realidad la lista es mucho más extensa, es un ejemplo), esto indica que en esa posición de la lista principal graficas donde están todos los datos, los valores son 0, no que en esa posición sean esos valores.
Por ejemplo, en la lista de graficas en la posición 3 el valor es 0, pero en la posición 4 que no aparece en la lista, el valor puede ser 1,334V, en la posición 812 el valor es 2,556V y en la posición 813 vuelve a ser 0, lo que se guarda es dónde es 0.
Esto es importante ya que determina los puntos que delimitan las gráficas, ya que en la posición 4 empezaría la primera muestra y acabaría en 812.

El bucle for lo que hace es recorrer la lista desde 0 hasta la longitud de la lista indices – 1, el -1 es para que no se salga de rango porque Python enumera las listas desde 0, si una lista va desde 0 hasta 7, la longitud de la lista es 8, si se recorre de 0 a 8 son 9 posiciones lo que se sale de rango.
El bucle recorre la lista con la condición de encontrar un valor de la lista que sea mayor que 1 respecto su anterior.
En la lista de indices se tiene que [0,1,2,3,813,814,815,1622,1623], de 0 a 1 no es mayor que 1 por lo que no se ejecuta, ni de 1 a 2 etc. Se ejecuta cuando llega a 813 donde 813-3 es mayor que 1, lo que hace es guardar las posiciones en una nueva lista cuando es mayor que 1 mediante indices2.append(i -1) en esta línea guarda el anterior, el 3, en la línea siguiente indices2.append(i) guarda el valor sobre el que se está iterando, 813.
De esta forma lo que se obtiene es una lista llamada indices2 que se compone de [3,813,815,1622], esta lista lo que tiene almacenado es el punto de inicio y final de la primera muestra en la lista graficas, el 3 y el 813 y el punto de inicio y final de la segunda muestra, el 815 y 1622 que serán los valores que correspondan.

Este algoritmo es crucial para automatizar, conocer el tamaño de la muestra y hallar las gráficas. En este caso hay 4 puntos, lo que corresponde a 2 muestras, si hubiera 30 puntos, correspondería a 15 muestras.
En la siguiente línea se define esa variable n, que es la mitad de longitud de la lista indices2, para saber el número de muestras y por consiguiente el número de gráficas que hay que representar.

El ciclo while define otro submenú para las gráficas, comienza con un ciclo for desde 0 hasta n, que es el tamaño de la muestra.
La siguiente línea es para imprimir una línea de texto para cada gráfica, para ello se imprime str(x+1), x es el valor iterado que comienza en 0 y se suma 1 para que empiece en 1, no en 0, seguidamente el texto y el número de la muestra, con lo que para la primera iteración quedaría
1.Para representar la gráfica con la muestra 1
Para la siguiente iteración sería igual, pero con 2, la siguiente con 3 así sucesivamente hasta el tamaño de la muestra, de esta forma se automatiza el menú para un tamaño de muestras que sea.
La siguiente línea se utiliza str(n+1) para una gráfica con todos los datos, es n+1 para que sea la siguiente al acabar la muestra y finalmente con str(n+2) para volver al menú principal, de esta forma también se automatiza y en el caso de tener 7 muestras, la gráfica completa será el 8 y salir el 9.

```
for op in range(0,n):
    if comando == str(op+1):
            x = None #Inicializar las variables
            y = None
            formatter = None
            x = [i for i in range(indices2[0+ajuste]+1,indices2[1+ajuste])]
            #+1 para el siguiente índice después del 0
            y = graficas[indices2[0+ajuste]+1:indices2[1+ajuste]]
            #+1 fuera del indice porque es un rango de valores
            #Comandos para el cursor y representación gráfica
            fig = plt.figure()
            ax = fig.add_subplot(111)
            ax.plot(x,y)
            formatter = EngFormatter(unit='V')
            ax.yaxis.set_major_formatter(formatter)
            plt.xlabel('Numero de muestras en 90ms')
            plt.title('Muestra ' + str(comando))
            plt.ylabel('Voltaje (V)')
            # set useblit = True on gtkagg for enhanced performance
            cursor = Cursor(ax, useblit=True, color='red', linewidth=1)
            plt.show()
    ajuste = ajuste + 2
```

Ilustración 37: Código Raspberry Pi en Python 5/9

Para representar las gráficas de las muestras individualmente, se define un bucle for que itere desde 0 hasta n que es el número de muestras.
La sentencia if discrimina la opción elegida, si se escoge la 1, comprueba si el comando introducido es igual a la iteración+1, como empieza en 0, al sumarle 1 se ejecutará, en el ciclo siguiente la iteración será 1 pero como se le suma 1 será 2 por lo que no se ejecuta.
Una vez que se verifica la condición, se inicializan las listas ‘x’, así como la variable formatter,

La línea de la lista ‘x’ define el número de muestras que se toman iterando sobre la lista indices2 desde el principio de la muestra hasta el final, ahora mismo es 0 la variable ‘ajuste’ se verá más adelante, recordemos que esta lista es [3,813,815,1622], donde el 3 corresponde a la posición de la lista graficas que es un 0, por lo que se añade el 1 fuera de la lista para que sea un 4 en la lista graficas que es donde están todos los datos y corresponde con un valor, por eso es fuera de la lista, si fuera dentro, lo que se indica es que se está utilizando la siguiente posición de la lista indices2, lo que sería el número 813, es por eso que se busca es el siguiente número no la siguiente posición.

El final de la lista es la segunda posición de la lista indices2, es decir la posición 1, recordando que Python numera empezando por 0.
Por lo tanto, se halla una lista ‘x’ iterando desde 4 hasta 813, siendo 4, 5, 6, 7 ,8 etc, que será el eje x y corresponderá con un valor de ‘y’.

La línea de la lista ‘y’ se sigue el mismo razonamiento, pero esta vez no se itera, lo que se hace es en la lista graficas, tomar un rango de valores que se indica con los dos puntos, en este caso es:
y = graficas(indices2[0]+1:indices2[1])
Que con números es
y = graficas(4:813)
Con esto se indica que se tomen todos los datos entre la posición 4 y la 813, así tiene correspondencia con la lista ‘x’ y se han eliminado los ‘0’ para la representación gráfica.

Las siguientes líneas son comandos específicos para la representación gráfica, excepto la de la variable formatter y la siguiente que son de una librería para representar mejor las unidades en el eje y.

La línea de cursor es una librería para que según donde se ponga el cursor encima de la gráfica se representen dos líneas, una horizontal y otra vertical para facilitar el corte con los ejes para ver una medida.
Finalmente el comando de representación gráfica plt.show()

La variable ajuste que está indentada a nivel del for y fuera de if, representa la corrección para los siguientes opciones y se incrementa de 2 en 2, es decir, si la gráfica que se desea ver es la 2, se pulsará el botón 2, el funcionamiento es el mismo al ya explicado, sólo que se escogen las dos posiciones siguientes, si tenemos la lista de indices2 [3,813,815,1622], se necesitan tomar los dos siguientes datos para la representación de la muestra 2, para ello en la variable ‘y’ esta vez se tiene:

y = graficas(indices2[0+ajuste]+1:indices2[1+ajuste])

Donde ajuste es 2, lo que se queda

y = graficas(indices2[2]+1:indices2[3])

Que corresponde con

y = graficas(815:1622)

Para la muestra 4 o 5 sigue el mismo razonamiento, así como en la variable 'x', de esta forma como ya se ha comentado, es indiferente el número de muestras que se vayan a medir, el programa siempre mostrará las gráficas necesarias, así como sus menús.

```
if comando == str(n+1):
    x = None
    y = None
    formatter = None
    x = [i for i in range(len(graficas))]
    y = graficas
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(x,y)
    formatter = EngFormatter(unit='V')
    ax.yaxis.set_major_formatter(formatter)
    plt.xlabel('Numero de muestras en total')
    plt.title('Muestra completa')
    plt.ylabel('Voltaje (V)')
    # set useblit = True on gtkagg for enhanced performance
    cursor = Cursor(ax, useblit=True, color='red', linewidth=1)
    plt.show()

elif comando == str(n+2):
        break
ajuste = 0
```

Ilustración 38: Código Raspberry Pi en Python 6/9

En la opción de representación de todas las muestras la idea global es la misma, se inicializan las variables y en este caso la 'x' y la 'y' toman todo el espectro de los datos.

Si se pulsa la opción de salir se rompe el bucle del menú de las gráficas y se vuelve al principal.

Finalmente, la variable ajuste vuelve a 0 para que no se acumule, sin esto al elegir una gráfica el valor de ajuste sería mayor que el de la lista y produciría un error.

```
elif opcion == '3':
    cls()
    fichero = None
    Veficaz = None
    fichero = file("Final5m.dat")
    Veficaz = pickle.load(fichero)
    fichero.close()
    for i in range(len(Veficaz)-1,-1,-1):
        if Veficaz[i] == '0.0':
            Veficaz.pop(i)
    for i in range(0,len(Veficaz)):
        Veficaz[i]= float(Veficaz[i])
        if Veficaz[i] > Vmax:
            Vmax = Veficaz[i]
    Vef = round(Vmax*0.707, 3)
    print ("El valor eficaz es: " + str(Vef) + " V")
    raw_input ("Pulse para continuar...\n")
```

Ilustración 39: Código Raspberry Pi en Python 7/9

En la opción 3 se calcula el valor eficaz de la muestra, para empezar como siempre se limpia la pantalla con cls(), se inicializan las variables y se carga el fichero de datos.

En este caso lo que hay que hacer es quitar los 0 para que no distorsionen la medida, para ello se recurre al bucle for que recorra la lista en sentido inverso de ahí los '-1' para la longitud de la lista, si el dato coincide con '0.0' mediante el comando pop() se elimina de la lista.
Es necesario recorrerla inversamente ya que la longitud de la lista al eliminar los 0 varía constantemente, de hacerlo en sentido normal se sale de rango y se detiene el programa, de forma inversa no ocurre.

Una vez eliminados los 0 se convierte la lista en float y si el dato en el que se está es mayor que la variable Vmax que es 0 en principio, esta variable toma ese valor, esto se hace para hallar el valor máximo.

Cuando termina, se redondea a 3 dígitos y se multiplica por 0,707 para determinar el valor eficaz y muestra por pantalla.

```
elif opcion == '4':
    cls()
    fichero = None
    Vpromedio = None
    fichero = file("Final5m.dat")
    Vpromedio = pickle.load(fichero)
    fichero.close()
    for i in range(len(Vpromedio)-1,-1,-1):
        if Vpromedio[i] == '0.0':
            Vpromedio.pop(i)
    for i in range(0,len(Vpromedio)):
        Vpromedio[i]= float(Vpromedio[i])
        if Vpromedio[i] > 0:
            Vtotal = Vtotal+Vpromedio[i]
            contador = contador + 1
    Vmedio = Vtotal/contador
    Vmedio = round(Vmedio, 3)
    print ("El valor de medio es: " + str(Vmedio) + " V")
    raw_input ("Pulse para continuar...\n")
```

Ilustración 40: Código Raspberry Pi en Python 8/9

Se comienza a analizar desde el segundo for, ya que es igual que el anterior.

En el for lo que se hace es convertir a float y si es mayor que 0 que debe ser todos los datos ya que se han eliminado con anterioridad, en la variable Vtotal se van acumulando los valores y en el contador se suma cada iteración para al final sacar la media, redondear a 3 y presentar el valor por pantalla.

```
elif opcion == '5':
    cls()
    fichero = None
    Vmaximo = None
    fichero = file("Final5m.dat")
    Vmaximo = pickle.load(fichero)
    fichero.close()
    for i in range(len(Vmaximo)-1,-1,-1):
        if Vmaximo[i] == '0.0':
            Vmaximo.pop(i)
    for i in range(0,len(Vmaximo)):
        Vmaximo[i]= float(Vmaximo[i])
        if Vmaximo[i] > Vmax:
            Vmax = Vmaximo[i]
    Vmax = round(Vmax, 3)
    print("El valor de pico es: " + str(Vmax) + " V")
    raw_input ("Pulse para continuar...\n")
```

Ilustración 41: Código Raspberry Pi en Python 9/9

El proceso para hallar el valor de pico es el mismo que para el valor eficaz, pero se presenta el valor máximo que se obtiene directamente.

Resultados

A continuación, se mostrarán los resultados del código desarrollado con capturas del programa en funcionamiento.

Como se ha visto en el código, primero hay que esperar 30 s para que se reciban los datos de Arduino, pasados este tiempo se mostrará el menú

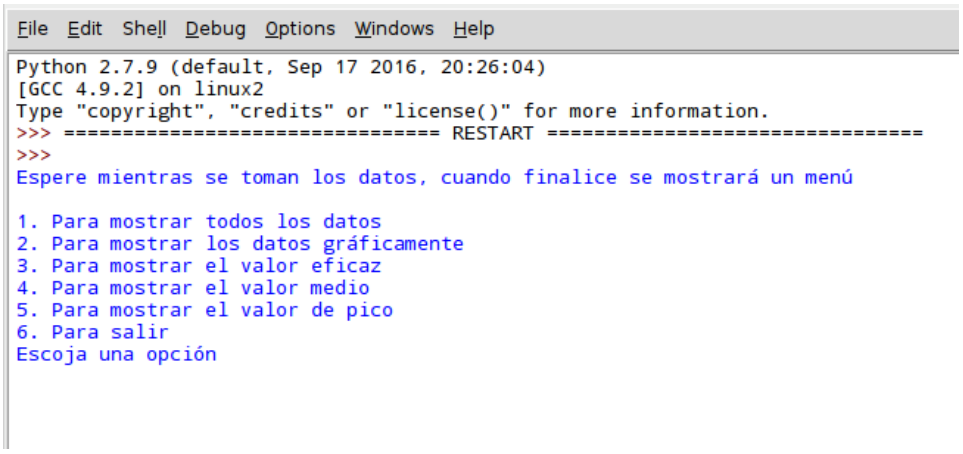

Ilustración 42: Menú de resultados

Tras esperar se muestra el menú, se pulsa 1 para mostrar todos los datos

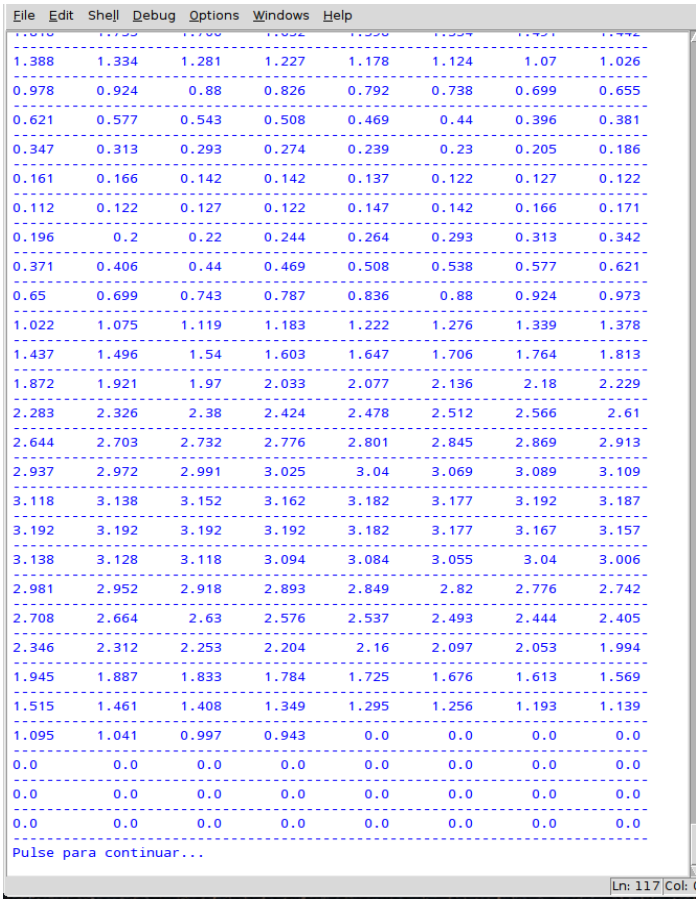

Ilustración 43: Datos

En esta imagen no aparecen todos los datos ya que son alrededor de 4000, se opta por enseñar los últimos donde se puede apreciar la variación de datos y la gran precisión que se consigue.

Al pulsar intro se vuelve al menú y se pulsa 2 para entrar al menú de las gráficas.

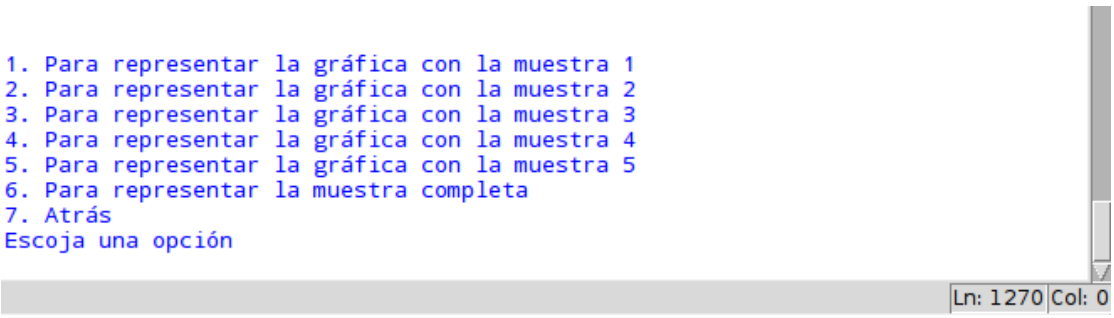

Ilustración 44: Menú de gráficas

Aquí se muestran las gráficas que se pueden representar, se irán mostrando una a una hasta la muestra completa, después se pulsará el 7 para volver al menú principal.

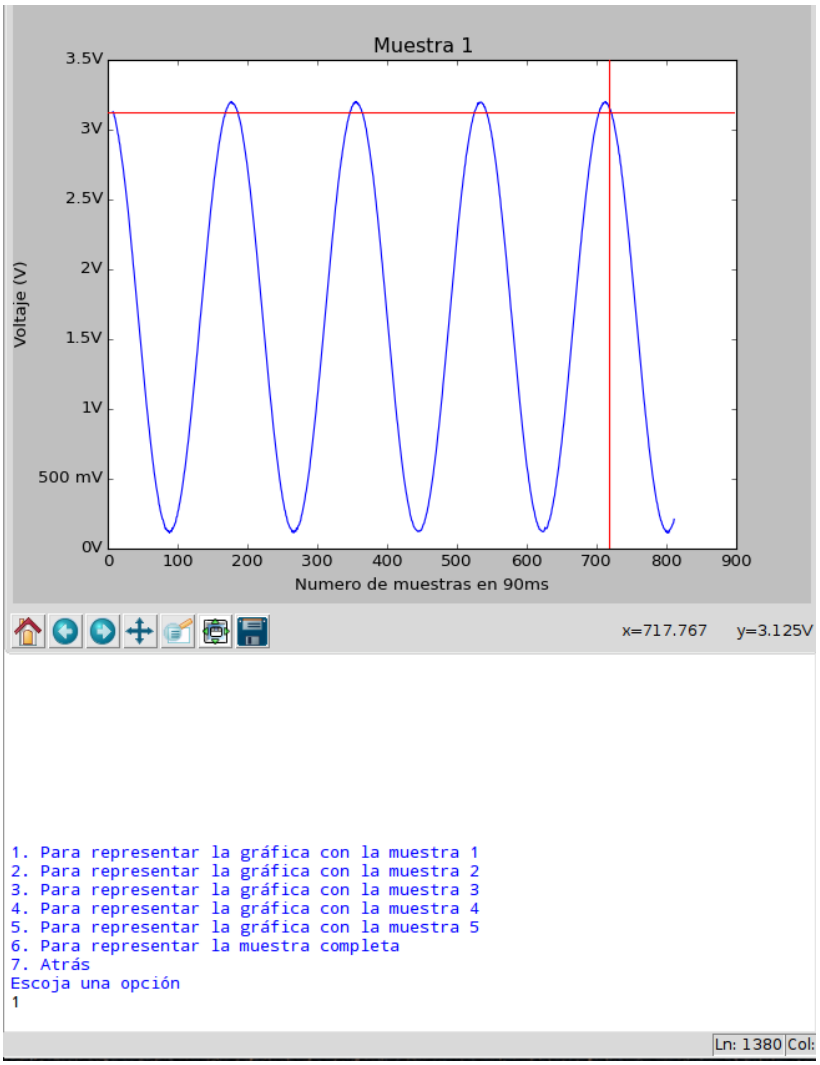

Ilustración 45: Gráfica de la muestra 1

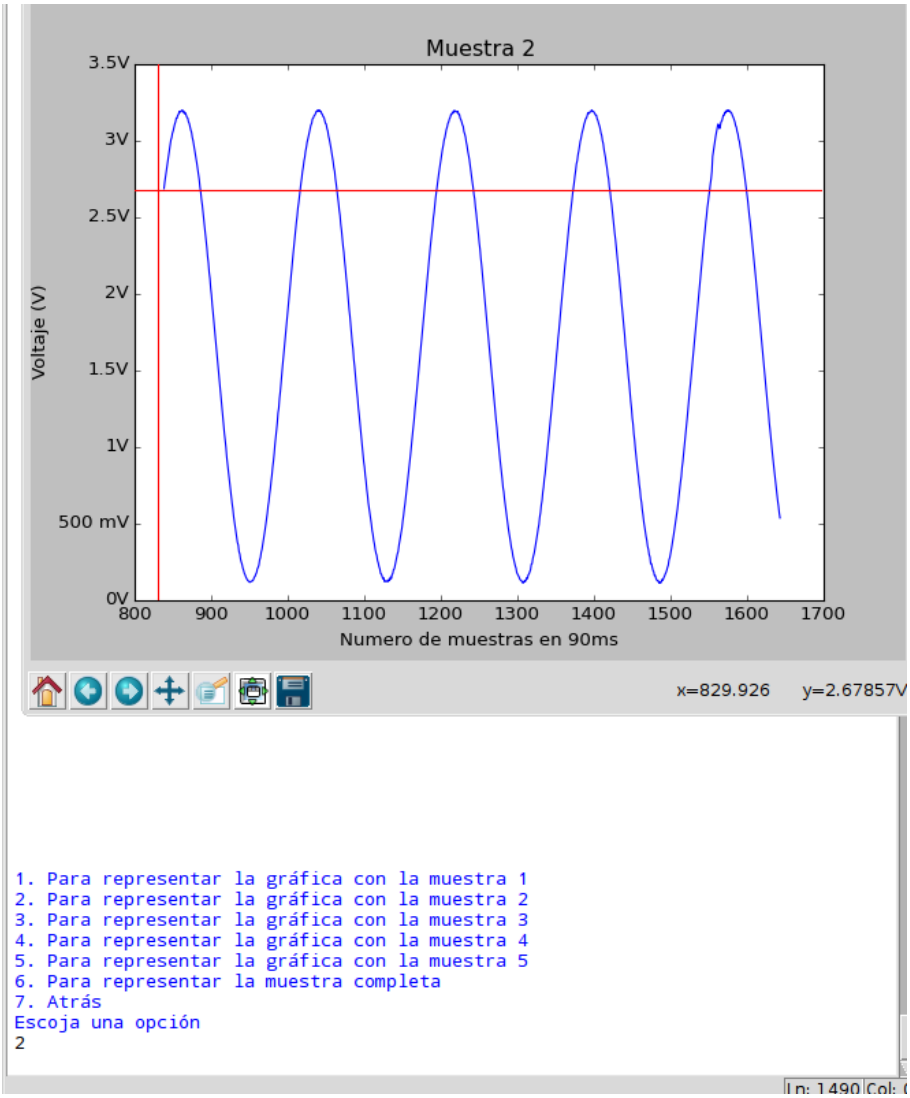

Ilustración 46: Gráfica de la muestra 2

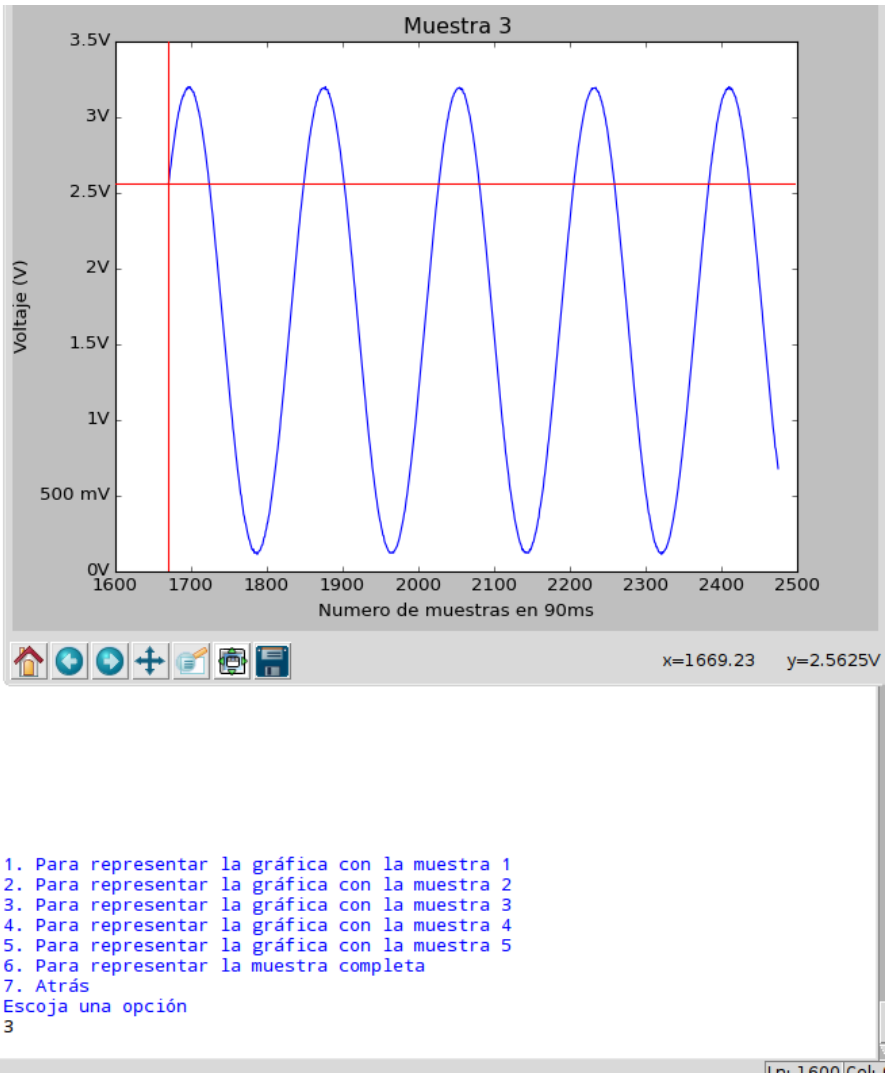

Ilustración 47: Gráfica de la muestra 3

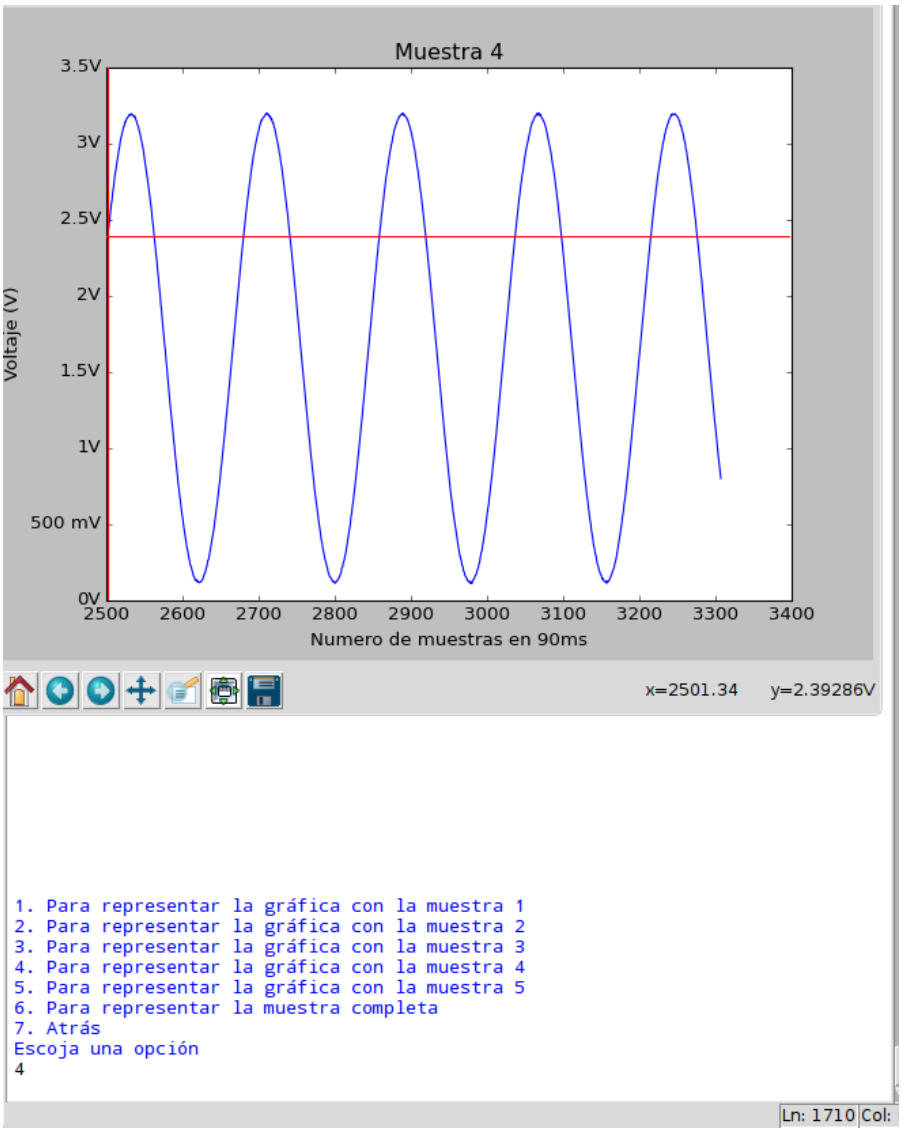

Ilustración 48: Gráfica de la muestra 4

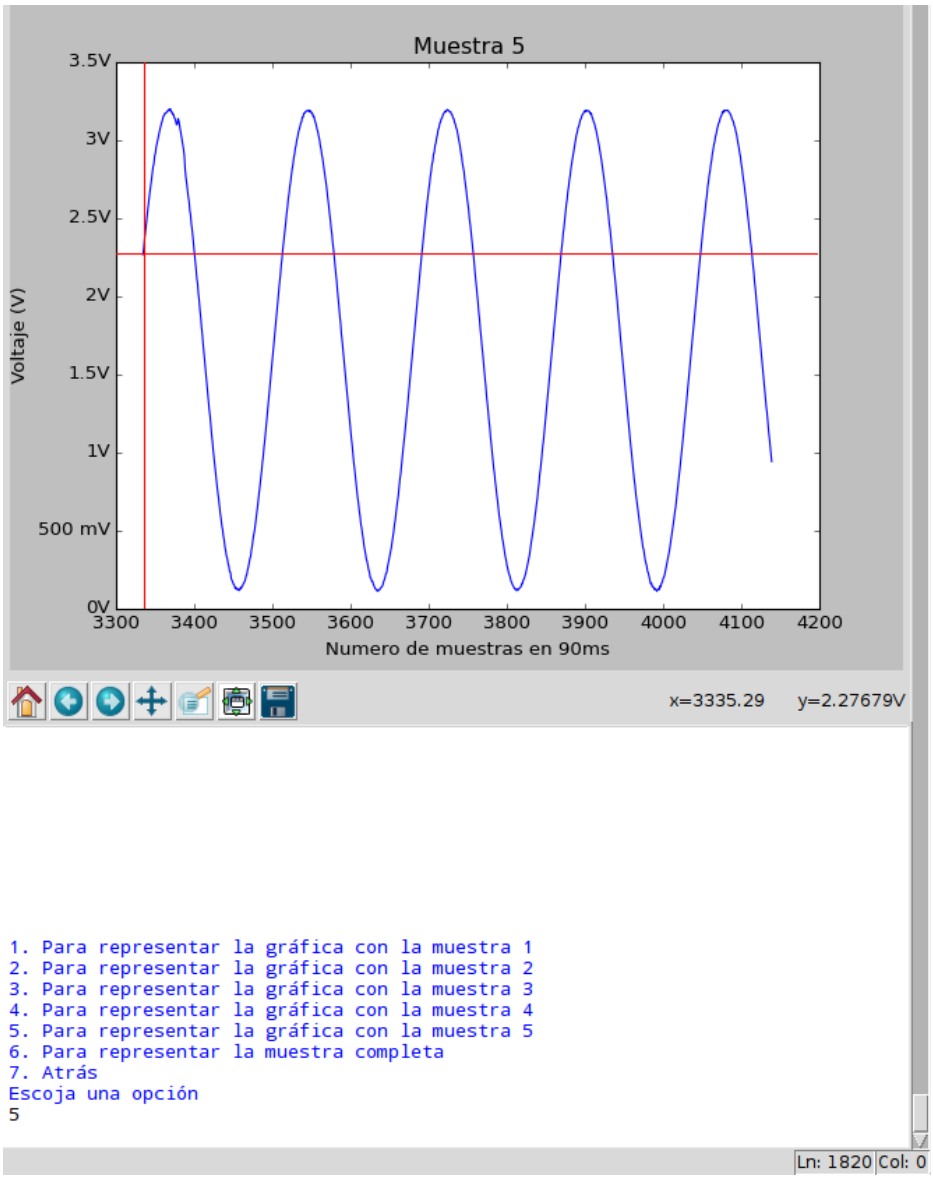

Ilustración 49: Gráfica de la muestra 5

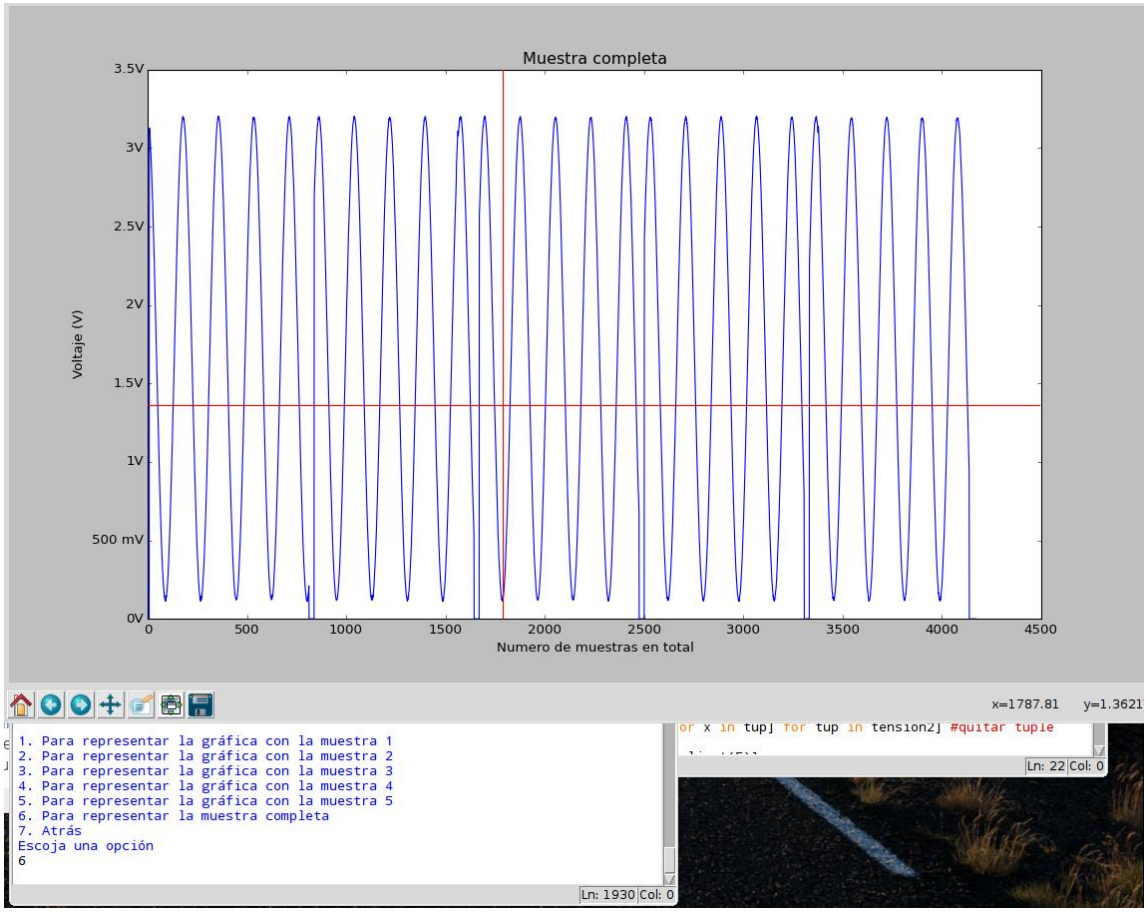

Ilustración 50: Gráfica de todos los datos

En esta gráfica se decide dejar los 0 para que se note dónde empieza cada muestra, ya que hay muchos ciclos y datos y no es la representación más clara, es mejor ver las gráficas una a una.

Comentar que las gráficas tienen opciones para guardar la imagen, ampliar en una sección y resetear la vista original en caso de ampliar, además abajo a la derecha están las coordenadas para saber exactamente en qué punto está el cursor.

Después de esto, se vuelve al menú principal y se pulsa el 3 para ver el valor eficaz, después 4, para el valor medio, 5 para la tensión de pico y finalmente 6 para salir.

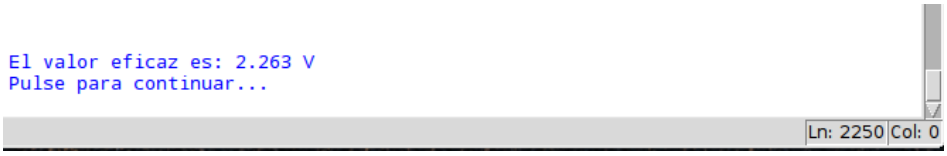

Ilustración 51: Valor eficaz

Ilustración 52: Valor medio

Ilustración 53: Tensión de pico

Estos son los resultados que se obtienen mediante Arduino y almacenándolos en Raspberry para su posterior lectura y análisis para 5 muestras de 90 ms esperando 5 s entre ellas, midiendo un total de 25 s

Conclusiones

Los resultados obtenidos determinan que Arduino tiene una buena precisión para la medida de tensión senoidal a 50 Hz debido a la gran velocidad de muestreo que posee que es de algo más de 0.1 ms por dato, esto hace si un ciclo son 20ms se puedan tomar casi 200 medidas por ciclo lo que garantiza una precisión realmente buena además una vez programada la base ya se puede adaptar al uso que se quiera, como tiempo entre muestras más corto, más muestras, que esté leyendo y enviando datos indefinidamente etc.
La parte negativa es si se quiere una lectura continua, tiene la limitación a 90 ms de lectura que se puede solucionar en parte acortando el tiempo entre muestras, pero no llegará a ser continua del todo ya que tiene que parar para enviar los datos. Se puede programar para que lea constantemente como se ha visto pero no con ese grado de precisión logrado.

La parte negativa en Rapspberry ha sido la complejidad para desarrollar el código, pero una vez que se tiene la base, igual que Arduino, se puede adaptar a los usos que se requieran además de ampliar sus funciones, importando otras librerías y módulos, como monitorizar una serie de consumos, enviar los datos a un servidor por internet, almacenarlos en una base de datos, transmitirlos a una web, enviarlos por bluetooth, desarrollar aplicaciones móviles para enlazarlas con la Raspberry…
En definitiva, las funciones que se pueden desarrollar con estos dos dispositivos son casi ilimitadas y puede ser considerado un buen punto de partida para aplicaciones más complejas.