PONTIFICIA UNIVERSIDAD
CATOLICA
DE VALPARAISO

Escuela de Ingeniería Eléctrica
PONTIFICIA UNIVERSIDAD CATOLICA DE VALPARAISO

Boris Iván González Zambra

Sistema de adquisición de datos para laboratorio de Física bajo el modelo de hardware y software libre.

Informe Proyecto de Título de Ingeniero Civil Electrónico

Escuela de Ingeniería Eléctrica

PONTIFICIA UNIVERSIDAD
CATOLICA
DE VALPARAISO

Sistema de adquisición de datos para laboratorio de Física bajo el modelo de hardware y software libre

Boris Iván González Zambra

Informe final para optar al título de Ingeniero Civil Electrónico,
aprobada por la comisión de la
Escuela de Ingeniería Eléctrica de la
Pontificia Universidad Católica de Valparaíso
conformada por

Sr. Juan Vignolo Barchiesi
Profesor Guía

Sr. Sebastián Fingerhuth Massmann
Secretario Académico

Sr. Jorge Mendoza Baeza
Director

Valparaíso, 28 de febrero de 2017

Dedicado a mi familia,

sin cuyo apoyo y esfuerzo a pesar de las dificultades, este trabajo no hubiera sido posible

Agradecimientos

Quisiera agradecer primeramente a mi familia cuyos esfuerzos, dedicación y apoyo incondicional y constante me permitió tener tiempo para desarrollar este trabajo sin dificultades.

Además, al profesor Juan Vignolo por su paciencia y tolerancia, a los profesores Francisco Vera y Manuel Ortiz quienes siempre estuvieron dispuestos a hacer un espacio en su agenda para resolver incluso las más puntuales de mis dudas.

Finalmente, quisiera agradecer a mis compañeros de universidad y amigos, por su compañía, consejos y palabras de ánimo.

Valparaíso, 28 de febrero de 2017 B. G

Resumen

Se presenta el diseño y evaluación de sistema de adquisición de datos orientado a complementar, asistir y servir como herramienta para sacar un mayor provecho a las experiencias desarrolladas en un laboratorio de Física de la PUCV.

El proyecto se basa en una interfaz gráfica en computador, una placa Arduino y una serie de distintos sensores.

La función de la placa Arduino es tomar muestras desde los sensores y de entradas digitales, siendo capaz de tomar datos desde seis distintos sensores a la vez, además de dos entradas digitales y un contador.

La interfaz presenta los datos en un gráfico en el cual es posible seleccionar qué entradas, tanto analógicas y digitales, mostrar en pantalla. Además posee diferentes opciones para facilitar el análisis y revisión de los datos en una presentación simple y amigable.

Tanto el código del programa de Arduino como de HMTL5 se encuentran a disposición pública.

Se hizo uso de sensores adquiridos con anterioridad por parte del Instituto de Física, se comprobó el funcionamiento de cada uno de ellos para verificar su utilidad respecto a las necesidades que se tenían y se diseñaron circuitos para sensores de campo magnético y de carga mecánica, circuitos que operan con una sola fuente.

Palabras claves: Arduino, Software libre, HTML5, JavaScript.

Abstract

In this document is presented the design and evaluation of a data acquisition system oriented to compliment, assist, and serve as a tool to take greater advantage of the experiences developed in the physics laboratory of the PUCV.

The Project is based in a graphic interface running in a computer, an Arduino board and a series of sensors.

The use of the Arduino board is to take samples from the sensors and digital pins, being able to take data from six different sensors at the same time, plus two digital inputs and a counter.

The interface shows the data on a graph, in which it is possible to select which inputs, both analog and digital, display on the screen. Also presents different options to facilitate analysis and review of data in a simple and friendly way.

Both the program code of Arduino and HMTL5 are publicly available.

Sensors previously acquired by the physics institute were used, was verified the operation of each of them to check their usefulness with regard to the existing necessities, in view of this, circuits were designed for magnetic field and mechanical load sensors, circuits which operate with a single source.

Key words: Arduino, Open software, HTML5, JavaScript.

Índice general

Introducción

Investigaciones científicas requieren la recolección de datos para estudiar, monitorear, analizar, describir o entender particulares procesos o eventos. La toma de datos usualmente conlleva un compromiso, entre la cantidad y tipo de muestras necesitadas y los recursos disponibles para ello.

El apoyo de herramientas computacionales a las actividades de laboratorio, proporciona un modo más rápido de registrar datos, no solo con mayor exactitud, sino también a un nivel que es difícil para un ser humano. De este modo se aumenta de manera sustancial la complejidad de experimentos realizables.

Uno de los grandes problemas respecto a la adquisición de datos, son los costosos y cerrados sistemas disponibles de forma comercial, y la incapacidad que presentan al usuario de operar dichos sistemas sin adquirir componentes de ese mismo fabricante e imposibilidad de modificar cualquier característica de este, con lo cual se incurren en más y más costos. [1]

Software y *hardware* libre permiten el desarrollo de proyectos en base a esfuerzos colectivos, donde las motivaciones para aportar pueden ser poner en práctica las habilidades, reputación, diversión y entretenimiento, expectativas de reciprocidad, altruismo, ideas respecto al desarrollo de software o para satisfacer una necesidad propia. [2]

Sin embargo y respecto a lo anteriormente señalado, uno de los aspectos que se tienden a caer en abandono en el desarrollo de proyectos de código abierto, son instrucciones específicas, manuales o simplemente comentar el código, dado que esta tarea puede resultar menos gratificante, emocionante, prioritaria o más engorrosa para quienes deseen aportar en el desarrollo de dichos proyectos, con lo cual se puede crear una alta barrera de ingreso a personas menos experimentadas e incluso de uso a los usuarios finales.

El código de los programas desarrollados en este proyecto ha sido liberado al momento de presentar este documento, de este modo puesto a disposición del Instituto de Física así como de cualquier posible futuro interesado. Esto con vistas de posibles modificaciones, mejoras o adiciones, además de permitir que sea de utilidad a quienes lo requieran.

El código puede ser encontrado en https://github.com/DenpaBorisu/datalogger, también en los apéndices A y B.

Los avances realizados en los últimos años respecto al hardware libre, han dado origen a la placa de desarrollo Arduino, la cual es de muy bajo costo y compatible con cualquier componente capaz de entregar una salida de voltaje.

Permitiendo al usuario crear sus propios sensores, modificar los programas acorde a sus necesidades, etc.

Un trabajo de similares características, Liberlab, fue desarrollado entre 2004 y 2006 para promover la experimentación científica mediante la creación y uso de un laboratorio digital a muy bajo costo.

Justificaciones del tema

Los motivos que dieron lugar a este proyecto fue el hecho de que Liberlab, corre en una versión desactualizada de Python forzando a tener los computadores del laboratorio corriendo en Linux y sin poderlos actualizar, la otra opción de correr Liberlab en Windows resulta engorrosa debido a la necesidad de ejecutarlo desde el IDE de Python, teniendo el problema de la designación de puertos seriales por parte de Windows, requeriría además la modificación del código dependiendo que número de puerto COM le fuera asignado al Arduino de forma automática por el sistema operativo.

Por otro lado existe la necesidad de integrar, mejorar o buscar otras opciones, según sea el caso, de sensores que permitan realizar experiencias adicionales, con lo cual puedan ser más provechosos los cursos de laboratorio que imparte el Instituto de Física.

Este proyecto fue propuesto por los profesores Francisco Vera y Manuel Ortiz del Instituto de Física de la PUCV, siendo esta unidad académica la beneficiada y quien financiará el proyecto.

Proyecciones

Los posibles usos del desarrollo de este es proyecto incluyen:

Entregar la posibilidad de realizar experiencias de laboratorio adicionales dadas las capacidades del sistema de adquisición de datos, estas siendo lectura simultánea de más sensores a la vez, información sobre la activación de elementos que podrían estar realizando algún estímulo o modificación a la situación que se intenta registrar, visualización de la información leída en tiempo real y sensores adicionales.

Otro de los puntos sería que al presentar facilidad y rapidez de instalación y uso, permitirían al profesor, tanto como al alumno concentrarse más en la realización de las experiencias y el análisis de los datos, además de ser un sistema de bajo costo y fácil replicación.

Si bien este proyecto estuvo enfocado a cumplir con los requerimientos presentados por los profesores del Instituto de Física, a la vez también sus partes están orientadas a poder ser modificables en caso de ser requerido.

Que sea open source da la posibilidad de que el proyecto sea mejorado, optimizado o modificado de acuerdo a necesidades de diferentes personas o grupos, ya sea por ellos mismos u otro grupo interesado [2]

Objetivos

El objetivo del proyecto fue mejorar el sistema de adquisición de datos que se tiene en el Laboratorio de Física de la PUCV, manteniendo la simplicidad y facilidad de puesta en marcha y los elementos activos actualmente, además expandiendo sus capacidades.

Para ello se propuso:

- Desarrollar un programa para adquirir los datos usando la placa Arduino.
- Desarrollar un programa en HTML5/JavaScript para el manejo y representación gráfica de los datos.
- Adaptar o diseñar circuitos para los diferentes sensores

1 Antecedentes y propuestas

En este capítulo se presenta en detalle la situación que originó el desarrollo de este proyecto y distintas posibilidades que existen para abordarla.

Además se explica qué se pretende en forma general y específica, y se menciona la solución que fue propuesta para el desarrollo de éste y su relación con los objetivos.

1.1 Descripción detallada del problema

El Laboratorio de Física de la PUCV utilizaba el software Liberlab para realizar mediciones en ciertas actividades que requerían el uso de sensores y muestreo de datos respecto al tiempo.

Una dificultad que se presenta al utilizar el software Liberlab, es el requerimiento de una versión desactualizada de Python.

La opción de correr Liberlab en Windows resulta engorrosa, debido a la necesidad de ejecutarlo desde el IDE de Python, además presenta problemas con la designación de puertos seriales por parte de Windows, con lo cual se requiere modificar el código del software dependiendo del número de puerto COM que fuera asignado a la placa Arduino por parte del sistema operativo, con esto entorpeciendo la realización de las experiencias designadas a los alumnos. Por otro lado existía la necesidad de integrar, mejorar o buscar otras opciones, según sea el caso, de sensores que permitan realizar experiencias adicionales, con lo cual puedan ser más provechosos los cursos de laboratorio que imparte el Instituto de Física.

Este proyecto fue propuesto por los profesores Francisco Vera y Manuel Ortiz del Instituto de Física de la PUCV, siendo esta unidad académica la beneficiada y quien financiará el proyecto.

En los laboratorios de Física actualmente se tienen 100 computadores con Linux, pero pueden cambiarse a Windows, 40 Arduinos con sus cajas, 40 Arduinos nuevos y sus cajas compradas y el software “Liberlab”.

Se poseían los siguientes sensores, utilizados en diversas experiencias:

- Sensor de campo magnético
- Sensor temperatura

- Sensor intensidad luminosa
- Potenciómetros para medir posición angular

Experiencias adicionales que se buscaban llevar a cabo eran medición de carga mecánica (peso) y su posible variación respecto al tiempo, como podría ser un recipiente con agua hirviendo y un mechero; carga y descarga de un capacitor; medición de pequeñas variaciones de campo electromagnético, entre otros.

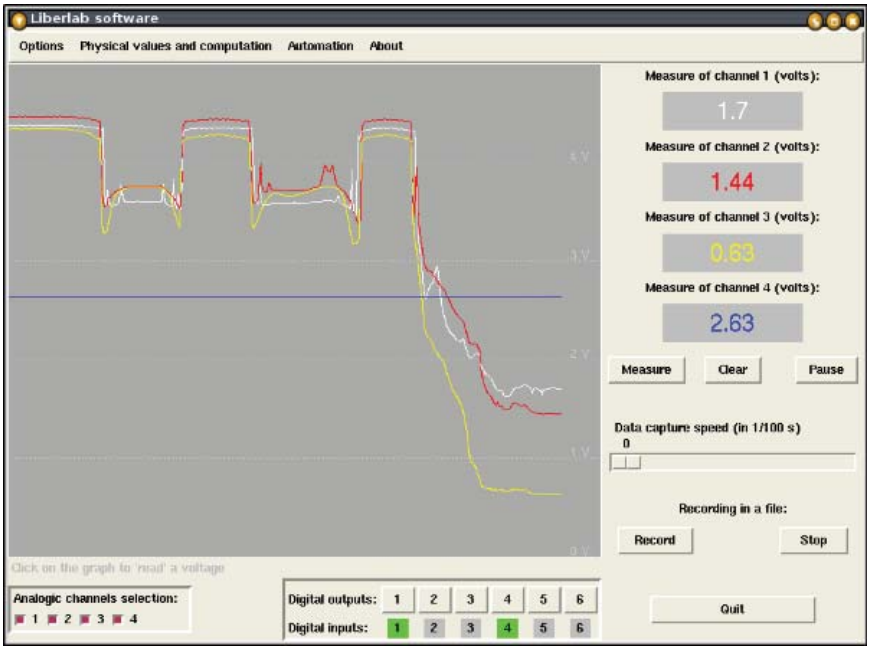

Figura 1-1: Ventana principal del software Liberlab

Entre las posibilidades para solucionar este problema están:

Modificar Liberlab, lo cual hubiera significado hacer una revisión exhaustiva del código, lo cual podría ser una larga labor, además de tener que lidiar con sus limitaciones y posiblemente reescribir secciones enteras de este.

Construir o adaptar sensores, esto es modificar sensores de acuerdo a las necesidades específicas del proyecto, o construir un circuito que adaptará alguno o algunos de los sensores utilizados o a utilizar, para hacerlos compatibles con el rango del conversor análogo digital del Arduino.

Comprar sensores listos, esto siendo la búsqueda y compra de sensores con respectivo circuito de amplificación/adaptación tal de conectarlo al Arduino y estar listos para usar.

Comprar un sistema de adquisición de datos comercial, que si bien de inicio estaba descartada esta opción, podría haber sido la elección en caso de requerirse algo muy rápidamente o en otro tipo de condiciones, si es que se estuviera dispuesto a invertir el dinero teniendo en cuenta lo que ello implicaría.

En base a lo expuesto anteriormente y a los objetivos del proyecto, se decidió tomar acción como se detalla a continuación.

1.2 Solucion propuesta

Desarrollar un programa en Arduino

Desarrollar un programa en HTML5

Revisar el funcionamiento de los sensores y en caso de ser necesario reemplazarlos o modificarlos.

Liberar el código de los programas, dejándolos comentados y utilizables.

1.3 Objetivos Generales y específicos

Objetivos generales

El objetivo del proyecto es mejorar el sistema de adquisición de datos que se tiene en el laboratorio de Física de la PUCV, manteniendo la simplicidad y facilidad de puesta en marcha y los elementos activos actualmente, además expandiendo sus capacidades y manteniendo el bajo costo de este.

Objetivos específicos

Desarrollar un programa en Arduino que permita la lectura de todos los canales analógicos de la placa Arduino a la vez, además de entradas digitales y un contador que registrará los cambios en uno de los pines, adicionalmente que poseyerá la capacidad de modificar la velocidad de muestreo.

El sistema propuesto consistiría de una serie de sensores para diferentes propósitos, conectados a una placa Arduino, la cual a su vez estaría conectada por USB a un computador enviando información sobre los distintos sensores, siendo esta recibida y graficada en una aplicación desarrollada en HTML5, la que permitiría la visualización de los datos haciendo uso de opciones de zoom y de revisión de cada lectura recibida.

El sistema presentará compatibilidad con Windows, Linux y Mac, al ser HTML5 no requeriría de programas que podrían no correr en distintas versiones de estos sistemas operativos.

Reemplazar o adaptar sensores para que concuerden con las necesidades de las experiencias de laboratorio desarrolladas en el Instituto de Física.

Se podría claramente utilizar en otros ámbitos distintos a los experimentos del laboratorio de Física, pero en principio y en lo que respecta a este proyecto se enfocará en ese uso y varios aspectos será definido por éstos.

2 Solución y marco teórico

2.1 Solución Propuesta

En esta sección se explica el modo en que se decidió desarrollar este proyecto.

Para completar este proyecto se hizo necesario abarcar 3 secciones: programa de Arduino, programa de manejo de datos y sensores, los cuales en conjunto constituyen el sistema de adquisición de datos que se desarrolló.

2.1.1 Arduino

Se decidió mantener el uso de las placas Arduino, dado que es lo que se poseía en el laboratorio, además concuerda con la idea de facilidad de uso e instalación.

Lo que se propuso hacer en este ámbito fue escribir un programa en Arduino que leyera las entradas análogas y un par de las digitales, haciendo uso de todas las capacidades que poseen estas placas, dentro de los mismos requerimientos del proyecto.

Otro punto en consideración fue hacer uso de instrucciones de interrupción, a modo de que la placa pudiera mantener a la vez la lectura constante de las entradas, en conjunto con el contador y la posibilidad de recibir instrucciones de control por parte del usuario, enviadas por el programa en HMTL5.

La información leída sería enviada vía USB hacia el computador.

Respecto al programa de Arduino es importante tener en cuenta que lo que se pretendía era cumplir con los objetivos antes mencionados, sin tener eso en mente la información y decisiones presentadas en secciones posteriores pueden parecer confusas e incluso contradictorias. Esto dada la velocidad de toma de datos máxima del Arduino, la cual no pudo ser aprovechada en su totalidad puesto que entra en conflicto con el uso de más de un canal. Información detallada al respecto puede encontrarse en la sección 2.2.1.

2.1.2 Programa de manejo de datos

Para manejar y visualizar los datos se propuso escribir un programa en JavaScript haciendo uso de características de HTML5 para su interfaz.

Se estableció que el programa debía ser capaz de:

- Graficar los datos a medida que son recibidos
- Guardar los gráficos generados
- Modificar el rango de tiempo del gráfico
- Seleccionar las entradas digitales a graficar
- Permitir modificar el valor graficado de los datos mediante operaciones introducidas por usuario
- Mostrar estado de entradas digitales
- Tener una interfaz simple e intuitiva
- Documentar el código.

Este programa recibiría los datos desde el Arduino ordenándolos, separándolos y/o convirtiéndolos en caso de ser necesario, y de forma acorde los graficaría o mostraría en pantalla.

2.1.3 Sensores

En cuanto a los sensores, algunos ya funcionaban de acuerdo a los requerimientos necesarios, en tanto otros requerían encontrar o construir uno acorde al funcionamiento del Arduino y de los requerimientos de las actividades a realizar en el laboratorio de Física.

- Sensor magnético

  Este sensor sería modificado, adaptado o reemplazado acorde a las necesidades de los experimentos del laboratorio de Física.

- Sensor temperatura

  El sensor de temperatura funcionaba bien para los usos que se le daba, de todos modos se procedió a verificar su funcionamiento.

- Sensor intensidad luminosa

  Este sensor no fue modificado.

- Potenciómetro para medir posición angular

  Revisar el funcionamiento de este sensor.

- Sensores de carga

  Para este sensor se quiso buscar una solución fácilmente replicable y de bajo costo.

- Voltaje

  En cuanto a lectura de voltaje, utilizar el conversor ADC del Arduino era suficiente para satisfacer los requerimientos.

- Contador

  Para el contador se quiso utilizar un conteo interno en el Arduino gatillado por una interrupción asociada a un pin en específico.

2.2 Marco Teórico

2.2.1 Arduino

Con el objetivo de generar un programa más estable, de mayor capacidad de muestras por segundo y con fin de aprovechar en su totalidad las capacidades de la placa Arduino, se decidió utilizar comandos de bajo nivel, diferentes a los utilizados en el IDE de Arduino además de hacer uso de instrucciones de interrupciones.

Las interrupciones son señales que le dicen al microcontroladores que detenga la ejecución del loop principal y ejecute ciertas funciones especiales.

Las funciones internas son manejadas mediante registros de 8 bits, donde cada uno controla algo en particular, AVR tiene prioridades establecidas respecto a las interrupciones, es decir en caso de intentar ejecutar dos interrupciones al mismo tiempo, ejecutará primero la que mayor prioridad tenga, capacidad de la cual se decidió tomar ventaja.

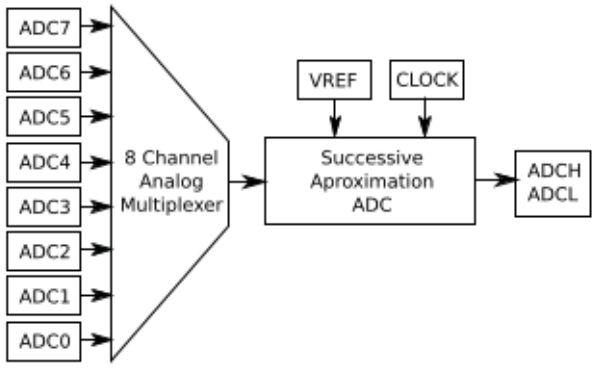

Figura 2-1: Diagrama conversor análogo digital del microcontrolador ATmega328p

El microcontrolador Arduino, ATmega328P, posee un conversor AD de 10bits de aproximación sucesiva, antes del conversor tiene un multiplexor que permite enviar al ADC señales desde diferentes pines, pero solo uno a la vez, además posee un reloj dedicado cuya frecuencia obtiene desde el reloj principal del Arduino luego de ser preescalado, en la figura 2-1 se presenta un diagrama del conversor AD.

El valor de preescalado por defecto es 128, en el caso del Arduino Uno cuyo reloj principal funciona a 16 MHz, el reloj del ADC corre a 16 MHz/128 = 125kHz.

Cada conversión toma 13 ciclos, excepto la primera la cual toma 25 ciclos del ADC, entonces idealmente la placa Arduino uno puede tomar 9600 muestras por segundo o dicho de otro modo tomar muestras cada 104 μs, usando su configuración estándar, esto bajo circunstancias específicas solamente.

Como dato adicional cabe mencionar que es posible reducir la resolución del conversor análogo digital, de modo de usar solo uno de los registros, dicho de otro modo que las lecturas fueran de 8 bits, sin embargo en este proyecto se mantuvo la configuración estándar de 10bits.

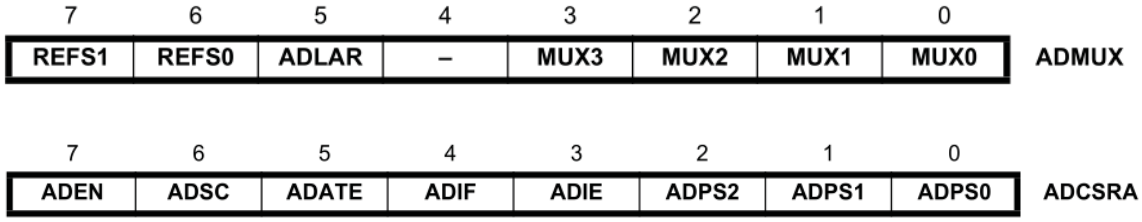

Figura 2-2: Registros del conversor análogo digital

En la figura 2-2 se muestran los registros del conversor AD, de los cuales los relevantes para lo que se pretende son:

- ADEN: Activar el ADC
- ADSC: Iniciar conversión
- ADIF: Interrupt Flag
- ADIE: Interrupt enable
- MUX: Selección de canal

El ADC de AVR posee dos modos de operación; conversión simple y free running.

En el modo de conversión simple es necesario iniciar cada conversión. Cuando esta es completada los resultados son guardados en ADCH y ADCL.

Cada conversión es iniciada activando el bit ADSC en el registro ADCSR, el cual se mantendrá activo mientras haya una conversión en progreso; una vez completada, ese bit es desactivado por el conversor.

Es posible seleccionar el canal que se quiere leer ajustando el valor del registro ADCMUX antes de iniciar una conversión.

En el modo *Free running*, al terminar cada conversión el ADC automáticamente comenzará la siguiente, de modo similar se puede elegir el canal a convertir, pero con la restricción de que no habrá cambio de canal hasta que comience una nueva conversión.

2.2.2 Html5

HTML5 agrupa tres tecnologías distintas, HTML, CSS y JavaScript, siendo cada uno el encargado de la estructura, estilo y presentación, y programación propiamente tal.

HTML (*HyperText Markup Language*) es un lenguaje de etiquetas, creado por W3C (*World Wide Web Consortium)*

CSS (*Cascading Style Sheets*) es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML

JavaScript es un lenguaje de programación orientado a objetos e interpretado, es decir no es compilado, si no que el navegador web lo interpreta directamente, aunque también se usa frecuentemente en entornos sin navegación.

En este aspecto elementos importantes que se decidieron utilizar fueron NW.js, la librería serial de Chrome y la librería grafica Echarts.

NW.js (Node-Webkit)

Node.js es un entorno de ejecución de JavaScript basado en motor de Chrome V8. Webkit es un motor de navegador web de código abierto.

Node-Webkit es un proyecto iniciado en el año 2011 desarrollado por el Open Source Tecnology Center de Intel, es un entorno de ejecución que permite usar tecnologías web como HTML5, CSS y JavaScript para elaborar aplicaciones nativas. Corre aplicaciones escritas en HTML5, CSS3, JavaScript y WebGL, posee completo soporte para APIs de Node.js y módulos de terceras partes, es fácil de compilar, disponible para Linux, Mac y Windows.

Justificación del uso de Node-Webkit

Los navegadores web por motivos de seguridad no tienen acceso a puertos seriales o puertos USB, Node-Webkit por su lado si puede acceder a ellos.

El hecho de que Node-Webkit sea capaz de correr en Windows, Linux y Mac, además de que sea mantenida por una comunidad de software libre, concuerda con el objetivo de que el proyecto sea multiplataforma. Del mismo modo proporciona cierto grado de seguridad en la continuidad de éste, lo cual es uno de los elementos base para este proyecto.

Librería Chrome.Serial

Es una API desarrollada por Google la cual permite lectura y envío de datos hacia un dispositivo conectado mediante algún puerto serial.

Echarts

ECharts es una librería de visualización de datos, de código libre desarrollado por el equipo de Baidu EFE.

Las características más llamativas de esta librería resultaron ser la interactividad que es capaz de entregar respecto a otras librerías similares, además de estar optimizado para la constante actualización de datos, visualización en detalle de estos mismos y posibilidad de generar dos gráficos sincronizados al mismo tiempo sin un mayor uso de recursos.

2.2.3 Sensores

Para los sensores fue necesario que pudieran funcionar con un voltaje de alimentación de 5[V] y que su salida fuera dentro del rango de 0 a 5 [V] que es el rango de entrada para una señal mediante el ADC del Arduino.

3 Desarrollo

En esta sección se presenta el desarrollo y el funcionamiento general de los programas tanto para Arduino como el de HTML5.

El código de los programas puede ser encontrado en sus apéndices respectivos.

También se realiza una revisión de los sensores utilizados, esquemáticos de sus circuitos y decisiones que fueron tomadas al respecto.

3.1 Arduino

Se escribió un pequeño programa de prueba, para comprobar el tiempo que toma realmente una lectura del ADC.

Como se puede notar en la figura 3-1, toma más tiempo que el ideal (104us), 116 a 124 µs por conversión o 8620 muestras por segundo, todo esto claro, sin considerar el envío, procesamiento alguno de las lecturas o la rotación de canales con lo cual esta velocidad de muestreo se vería fuertemente afectada.

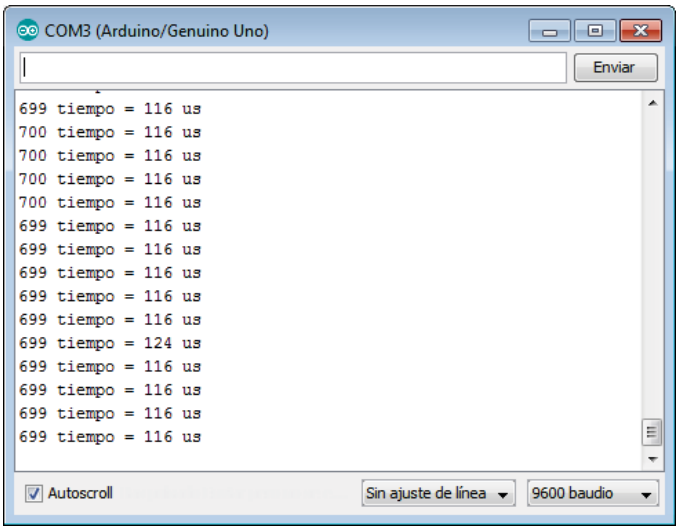

Figura 3-1: Consola serial de Arduino, recibiendo datos correspondientes a una lectura de una entrada analógica y el tiempo que tomó esa lectura.

La idea del programa como fue mencionado anteriormente es aprovechar el tiempo que le toma al ADC completar una conversión

Hay 3 interrupciones en este programa, una encargada del contador, otra activada cuando se completa una conversión del ADC y la tercera asociada al *Timer*, diagramas del proceder de las interrupciones pueden encontrarse en la figura 3-3.

La interrupción asociada al contador se activa cuando se detecta un canto de subida en el PIN2 del Arduino, simplemente aumenta el valor de la variable de conteo.

La interrupción activada al completarse una conversión del ADC, guarda el valor de la última lectura en variables temporales, asigna la próxima lectura al canal siguiente. Además levanta un flag adicional (externo a los registros del ADC) que indica que una lectura ha sido completada. Luego se inicia una nueva conversión, a no ser que el siguiente canal a leer sea el 6, el cual es inexistente en la placa Arduino Uno, y por tanto no inicia una nueva conversión.

En el loop principal se realizarían dos acciones en caso de cumplirse con sus respectivos requisitos, como puede apreciarse en la figura 3-2:

Si ya el flag de lectura completada está activo se procede a dar formato y anexar los datos a un arreglo, por otra parte, en el caso de que hayan sido enviados datos al Arduino se interrumpirá la ejecución del Timer asignándole un nuevo valor, de este modo cambiando la frecuencia a la cual se toman las muestras.

Una vez se cumpla el tiempo asignado por el *Timer*, se procede a enviar el arreglo de datos mediante USB y a comenzar una nueva tanda de lecturas.

El funcionamiento normal del programa si es que no se cambia la frecuencia de muestreo sería como sigue:

- Se inicia conversión del ADC
- Termina conversión del ADC y se da inicio a otra en el canal siguiente.
- Los datos de la primera lectura son anexados a un *string*
- Termina otra conversión del ADC y se da inicio a otra en el canal siguiente
- Los datos de la segunda lectura son anexados a un *string*
- Continúa de este modo hasta que se han leído las 6 entradas analógicas del Arduino
- Al completarse el tiempo del *Timer* se adiciona al mencionado string el valor en el momento del contador y de las dos entradas digitales, a la vez el tiempo del *Timer* comienza a correr nuevamente
- El *string* es enviado vía USB
- Se inicia una nueva tanda de lecturas

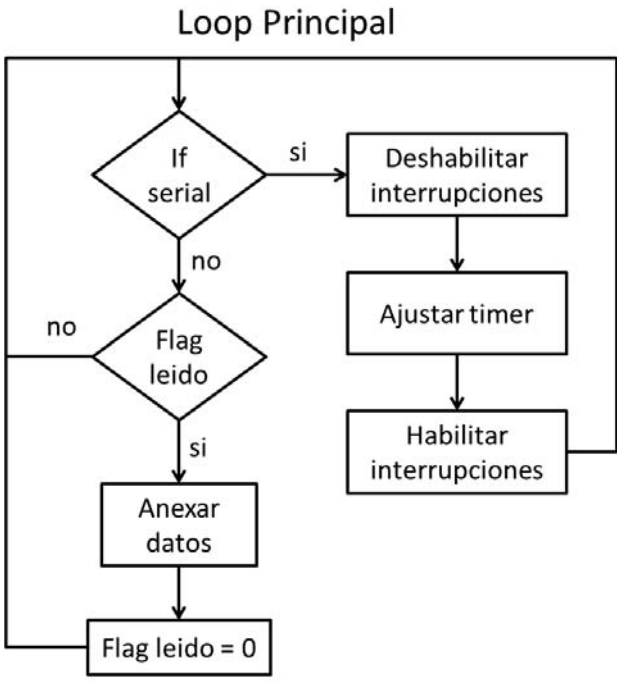

Figura 3-2: *Loop* principal del programa de Arduino

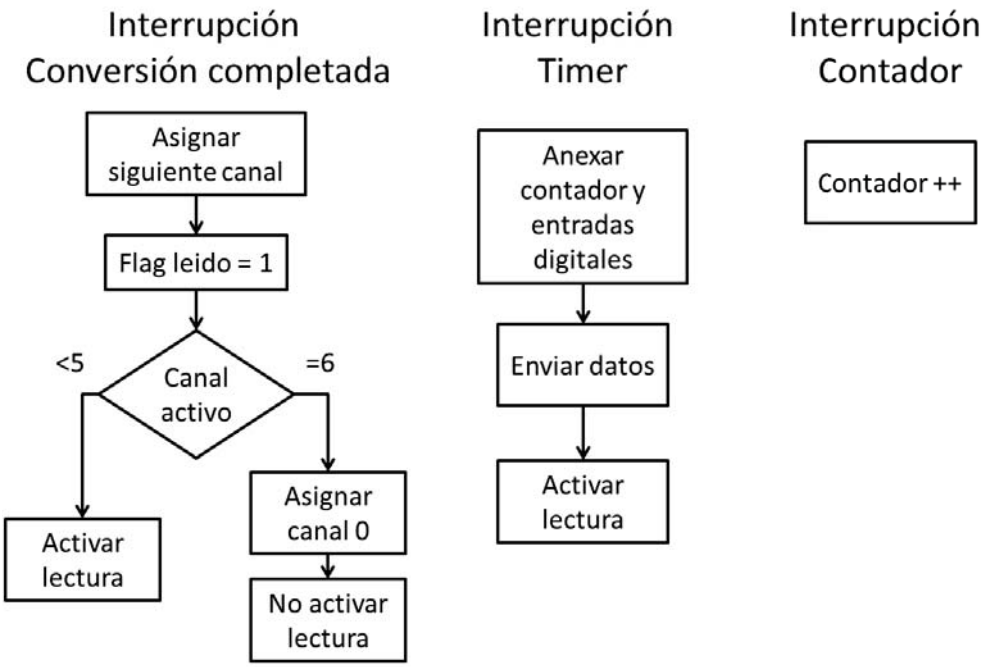

Figura 3-3: Diagramas de bloque de las instrucciones llevadas a cabo luego del llamado de cada una de las tres interrupciones.

3.2 Comunicación

Las lecturas de las entradas analógicas del Arduino son convertidas a una cadena de caracteres o *string* y anexa un salto de línea. Este último para separar un grupo de lecturas de la siguiente. Luego este *string* es enviado por el Arduino hacia el computador.

Un problema encontrado al realizar la comunicación entre el Arduino y el programa en desarrollo fue que los datos eran recibidos de forma segmentada, de modo impredecible, es decir el *string* enviado por el Arduino no era siempre recibido en su totalidad en una sola revisión del puerto serial por parte del computador, en otras ocasiones recibiendo el *string* completo, pero con datos de la medición siguiente. La solución fue crear un *buffer* para ordenar los datos a medida que el programa los recibía.

Al llegar datos, el programa verifica si es que hay algún salto de línea entre ellos, de no ser así los almacena temporalmente en el *buffer*, junto con otros datos que podrían haber estado almacenados allí con anterioridad. En caso de que, si haya un salto de línea, une los contenidos anteriores al salto de línea con el *buffer* y pasan a la siguiente etapa, el *buffer* es vaciado. Los datos posteriores al salto de línea son almacenados en el *buffer*, como es ilustrado en la figura 3-4.

De este modo fue posible lograr una comunicación adecuada entre el Arduino y el programa en HTML5.

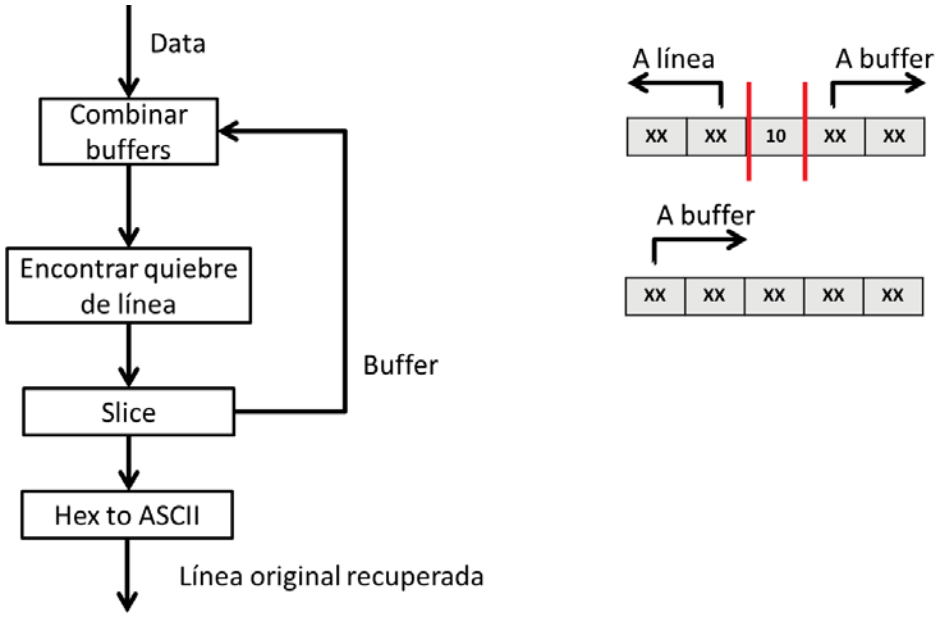

Figura 3-4: Diagrama de bloques del formateo, orden y separación de los datos al ser recibidos por el computador

3.3 HTML5

El funcionamiento general del programa se describe a continuación.

Una vez que ha establecido conexión con el Arduino, el programa genera un gráfico inicial el cual a medida que se van recibiendo datos, los separa de acuerdo al canal al cual pertenezcan, los convierte a un valor listo para ser usado, entonces guarda estos datos en el data set del gráfico, el cual es actualizado, como se muestra en la figura 3-5.

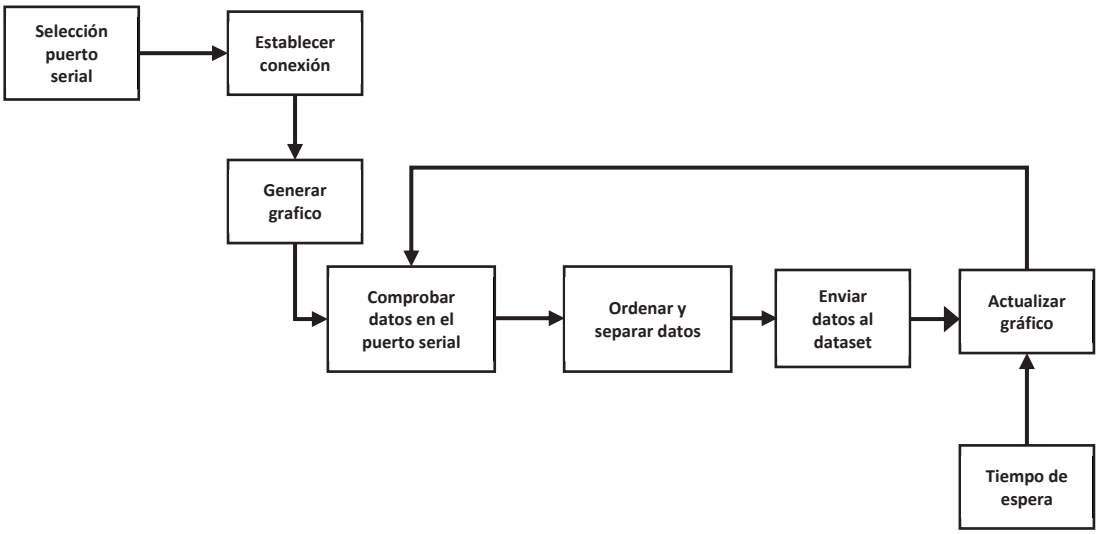

Figura 3-5: Diagrama de bloques del funcionamiento general del programa principal.

Para generar el gráfico es necesario definir opciones y ajustes para ser utilizados por la librería Echart.

Dado que lo que se pretende es representar datos respecto al tiempo, el tipo de gráfico será un gráfico de línea el cual puede mostrar datos de más de un canal a la vez.

Para esto es necesario definir cuantos grupos de datos serán utilizados y el formato de cada uno, es decir color de línea, etiqueta del grupo de datos.

En este proyecto se utilizarán 8 grupos de datos, que corresponderían a las lecturas de las 6 entradas analógicas y 2 digitales del Arduino.

Los datos recibidos desde el Arduino luego de ser ordenados y separados, son asignados al grupo de datos correspondiente a cada entrada, nombradas en el gráfico como “canales”.

La actualización del gráfico se realiza a ritmo diferente de la velocidad de muestreo del Arduino, con esto logrando mayor estabilidad, al no saturar el computador cuando se toman muestras a

una alta tasa por segundo, de todos modos no hace falta actualizar demasiadas veces por segundo el grafico. Se encontró que con 10 veces por segundo acomodaba a la vista.

3.4 Sensores

Se decidió optar por diseñar los circuitos tanto para amplificar la salida de la celda de carga como para el sensor magnético, esto en vista de los siguientes puntos.

Se encontró una empresa ubicada en la Quinta Región, específicamente Valparaíso, la cual construye circuitos a encargo.

El hecho de encontrar esta empresa fue algo positivo, dado que a pesar de que se diseñarán y construirán los circuitos probablemente se daría el caso de que, en periodos posteriores a la realización del proyecto, se requirieran más unidades de los circuitos que se diseñaron, dejando a los profesores de Física con la tarea de armarlos ellos mismos, no siendo este su campo ni ellos teniendo las herramientas necesarias.

3.4.1 Celda de carga

El circuito a final es el mostrado en la figura 3-9, usando un amplificador operacional de instrumentación, el INA125 en conjunto con una celda de carga rectangular para cargas pequeñas, la cual puede verse en la figura 3-6.

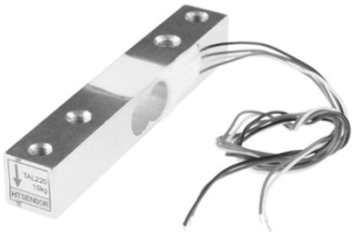

Figura 3-6: Celda de carga rectangular

A partir de los datos de la tabla 3-1 se puede obtener la máxima variación de voltaje que se generara entre las salidas de la celda de carga

$$\Delta Vmax = Supply * Rated\ output$$

$$\Delta Vmax = 5[V] * 1[\frac{mV}{V}]$$

$$\Delta Vmax = 5[mV]$$

Dicho de otro modo cuando la celda de carga este sometida a una carga de 500 [g] y alimentada con 5 [V] la diferencia de voltaje entre sus salidas variará 5[mV] respecto a su estado de reposo

Tabla 3-1: Características celda de carga

Capacidad	500 [g]
Salida nominal	1 [mV/V]
Resistencia de salida	1000±10 [Ω]
Voltaje de excitación recomendado	5 [V]

Así, para una carga de 1 [g], la diferencia en de voltajes en la salida de la celda de carga sería 10[μV].

Para probar la celda de carga se armó una pequeña plataforma como la mostrada en la figura 3-7, la cual es la configuración recomendada para este tipo de celdas [6].

Se realizaron mediciones en diferentes valores dentro del rango de operación de la celda, con lo cual se pudo observar que la relación entre variación entre las salidas de la celda y la carga aplicada poseen una relación lineal como puede apreciarse en la tabla 3-2 y la figura 3-8.

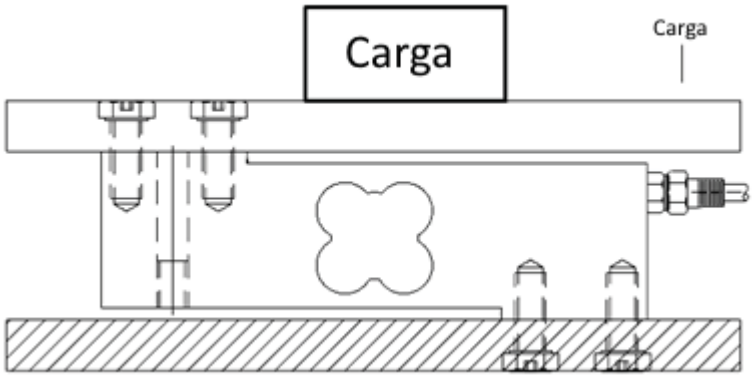

Figura 3-7: Configuración recomendada para celdas de carga rectangulares

Para la amplificación de la celda de carga se decidió utilizar el amplificador INA125 el cual es un amplificador de instrumentación de una sola fuente y resulta ideal para trabajar con celdas de carga.

Un aspecto importante respecto a esta configuración es que no importa el sitio en la plataforma en que se ubique la carga, la salida será la misma para la misma carga.

Respecto al amplificador INA125, este posee una ganancia ajustable mediante la resistencia R_G

$$G = 4 + \frac{40[k\Omega]}{R_G}$$

Tabla 3-2: Diferencia de voltaje a las salidas de la celda al aplicar carga.

Carga [g]	Salida [mV]
500	7,4
450	6,9
400	6,4
350	5,9
300	5,4
250	4,9
200	4,4
150	4,0
100	3,5
50	3,0
0	2,5

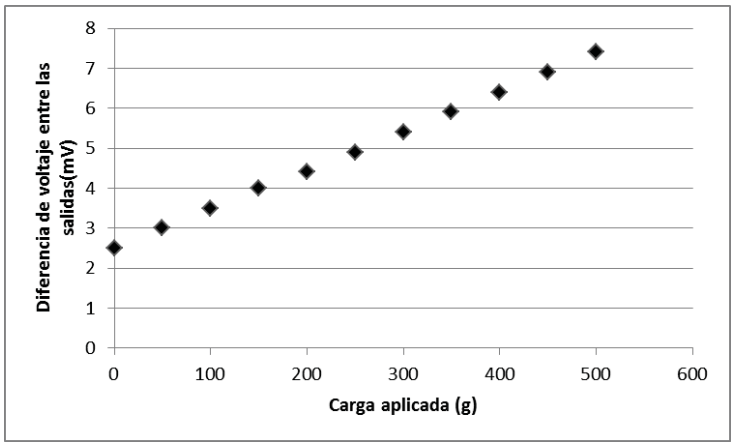

Figura 3-8: Relación entre la salida de la celda y la carga aplicada.

La ganancia seleccionada para este circuito es G = 500, con lo cual la salida en carga cero sería 1,2[V] y con carga máxima (500[gr]) 3,75[V], valores apropiados para la entrada análogo digital del Arduino, todo esto con una resistencia de 121[Ω].

Además posee un factor de rechazo al modo común de 100[dB] con G = 500, con lo cual el amplificador es capaz de bloquear el ruido generado.

Adicionalmente un filtro pasabajo, con R = 4,7 [kΩ] y C = 16 [μF] con lo cual se filtrarían las frecuencias por encima de ~2[Hz].

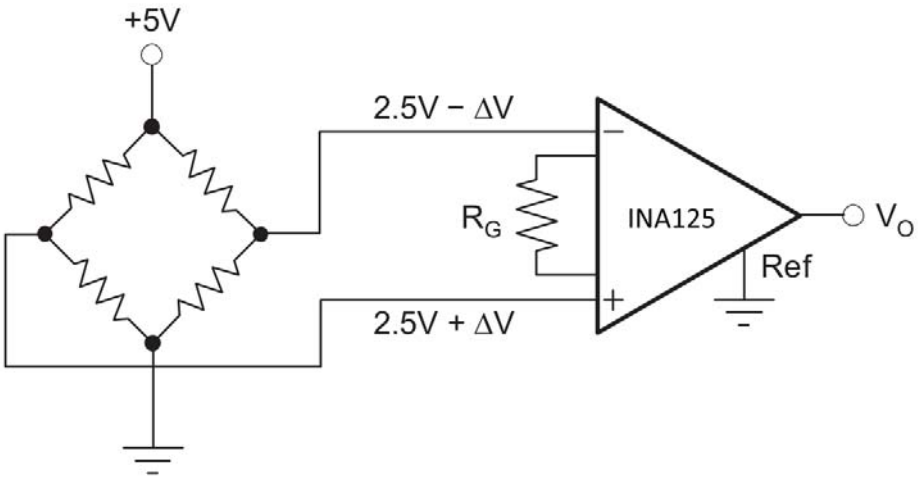

Figura 3-9: Configuración básica puente de Wheatstone con amplificador operacional de instrumentación

3.4.2 Sensor magnético

El sensor magnético utilizado actualmente es el UGN3503, el cual es direccional. Si no hay campo magnético su salida es el punto medio, en caso de ser alimentado con 5V su salida es 2,5V, en la figura 3-10 se muestra un diagrama simplificado del sensor.

La salida del sensor mencionado cambia 1,3mV por Gauss, lo cual resultó ser un problema, dado que el cambio en su salida generado, por ejemplo, por el campo magnético terrestre 0,5 Gauss seria 0,65mV, mucho menor que el cambio más pequeño que puede detectar el conversor análogo digital del Arduino, que es 4,9mV.

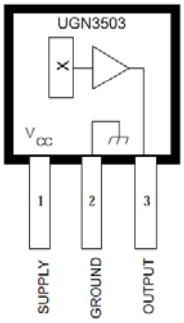

Figura 3-10 Esquema simplificado del sensor magnético UGN3503

La solución elegida fue utilizar dos de estos sensores espalda contra espalda, para amplificar la diferencia de salida entre ellos.

La salida de uno de los sensores apuntando primero hacia el norte y luego hacia el sur respecto a 2,5 [V] se muestra en la tabla 3-3.

Tabla 3-3: Medición de campo magnético de la tierra.

Norte	Sur
2,7 [mV]	[2,2 mV]

Tabla 3-4: Salida típica de los sensores UGN3503 siendo operados a 5[V]

	Min	Typ	Max
B = 0 G	2.25 [V]	2.50 [V]	2,75 [V]

Dada la gran diferencia que pueden tener los sensores en sus salidas típicas, en relación a su salida respecto al campo magnético de la tierra, como se puede apreciar en las tablas 3-3 y 3-4, se optó por construir un circuito que amplificará los cambios en la salida de estos dos sensores, en lugar del valor en todo momento, dado que hacerlo de este modo generaría problemas al ser amplificado, puesto que la diferencia que hay entre sus salidas la cual intentaría medirse sería mucho menor que la diferencia de salida propia de cada sensor.

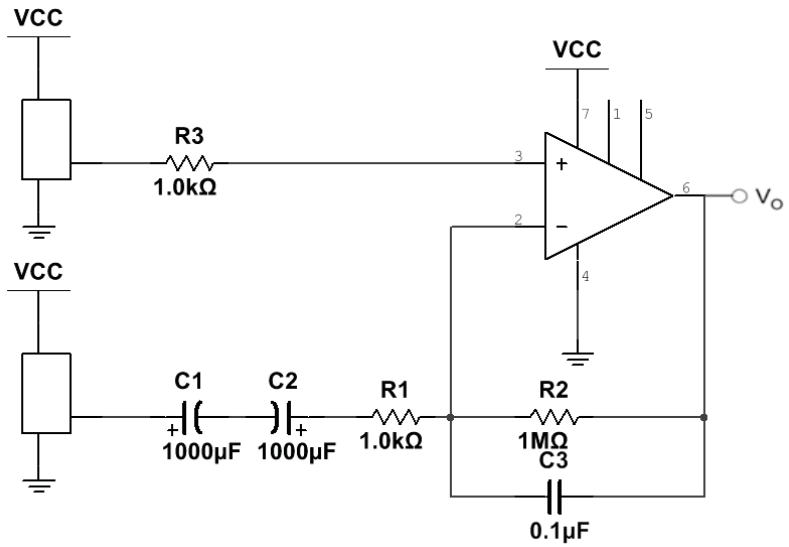

Figura 3-11 Circuito amplificador de campo magnético

Los capacitores C1 y C2 se cargarán según la diferencia entre la salida de los circuitos, tomando un voltaje igual a esa diferencia, al hacer esto la salida del circuito será 2,5 [V].

Al haber un cambio brusco en el campo magnético, las salidas de los sensores cambiará, sin embargo el capacitor mantendrá su voltaje por un breve periodo de tiempo antes de cargarse o descargarse según sea el caso.

Dado que la diferencia entre las salidas de ambos sensores puede ser positiva o negativa respecto a las entradas del amplificador C1 y C2 se encuentran colocados en serie y con polaridad invertida el uno respecto al otro, como se puede apreciar en la figura 3-11.

3.4.3 Sensor temperatura

El sensor de temperatura que se utiliza actualmente es un termistor de tipo NTC o de coeficiente de temperatura negativo, lo cual quiere decir que a medida que sube la temperatura su resistencia disminuye.

Su esquemático es el que se muestra en la figura 3-12.

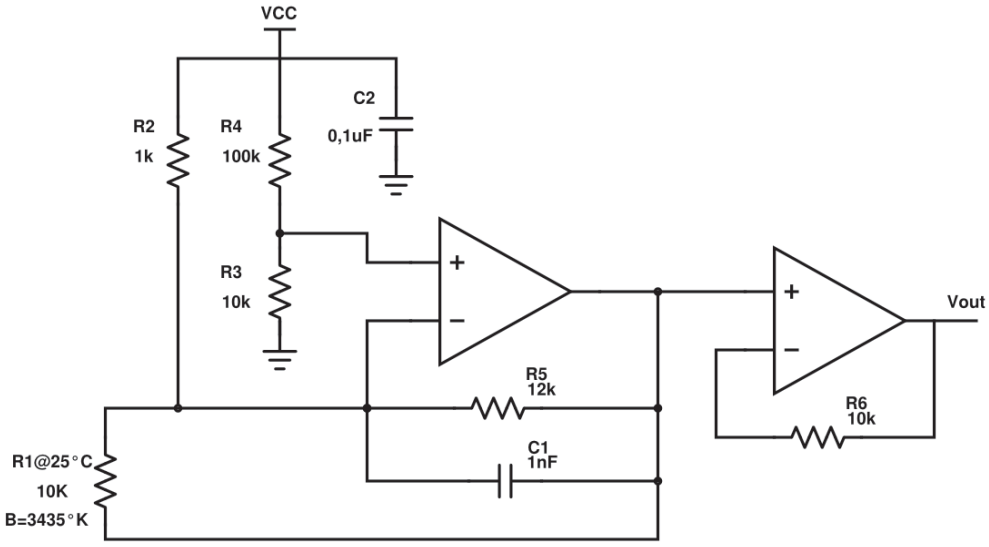

Figura 3-12: Esquemático del sensor ZX-Thermometer

Se puede notar que este circuito amplificador cuenta de dos etapas, la primera un amplificador de corriente constante y la segunda parte corresponde a un seguidor de voltaje.

Se realizaron mediciones para obtener la característica Voltaje-temperatura de este sensor, principalmente con dos propósitos, primero revisar su funcionamiento y segundo comprobar la repetitividad entre distintos sensores del mismo tipo.

Para esto se utilizó un tester con una termocupla y una ampolleta como fuente de calor, la cual se acercó progresivamente tanto a la termocupla como al ZX-Thermometer, las mediciones se muestran en la tabla 3-5 y la figura 3-13.

Tabla 3-5 Mediciones de temperatura y su respectiva salida en voltaje.

Voltaje [V]	Temperatura [°C]
2,02	19
2,38	30
2,62	41
2,99	48
3,19	57
3,42	66
3,58	73

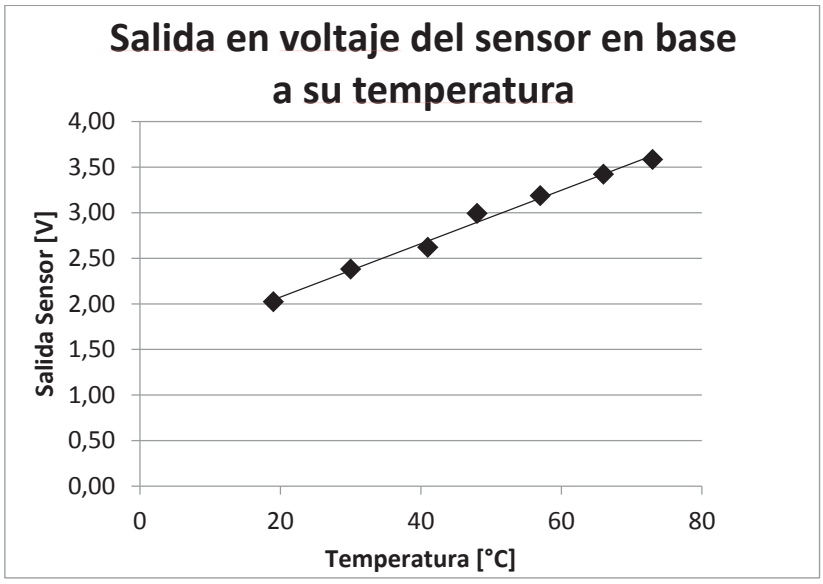

Figura-3-13 Relación entre voltaje y temperatura del sensor ZX-Thermometer

De este modo obtenemos la siguiente relación entre ambas variables

$$Voltaje = 0{,}029 \cdot temperatura + 1{,}511$$

Luego se comprobó esta línea de tendencia con mediciones de dos sensores del mismo tipo.

3.4.4 Sensor intensidad luminosa

Este sensor consiste en un divisor resistivo, cuyo punto medio es la salida que toma el Arduino, siendo una de las resistencias una foto-resistencia.

Este sensor ya cumple con las características requeridas para el proyecto.

3.4.5 Voltaje

Respecto a la medición de voltaje se utilizaron directamente los conversores análogos digitales del Arduino, dado esto no hubieron modificaciones o mayor análisis que hacer al respecto.

Tabla 3-6: Relación entre posición angular y salida del potenciómetro hall

Posición angular [Grados]	Salida potenciómetro [V]
0	0,49
30	0,83
60	1,15
90	1,50
120	1,82
150	2,14
180	2,48
210	2,8
240	3,13
270	3,52
300	3,82
330	4,17
358	4,43

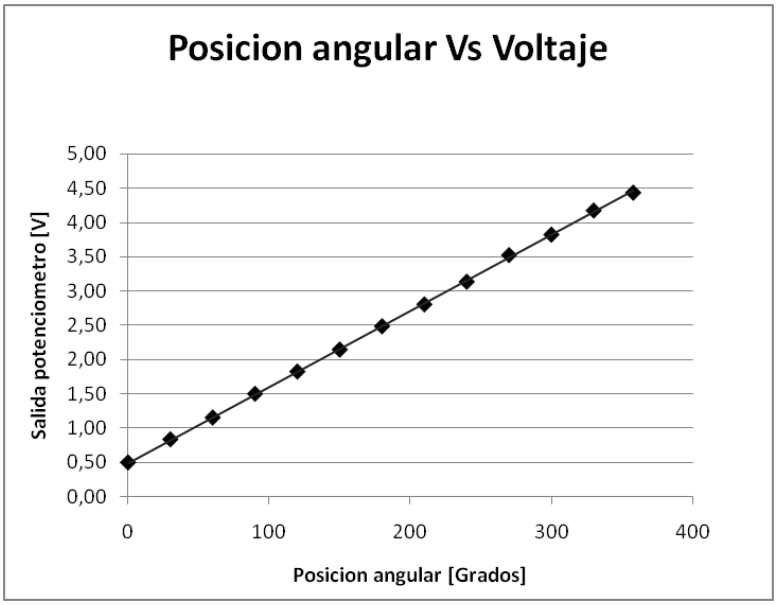

Figura 3-14: Respuesta angular del Potenciómetro hall

3.4.6 Contador.

Respecto al contador, se dejó un pin libre para conectar la salida de algún elemento según sea necesario.

Adicionalmente resulta necesario colocar resistencias *Pull-Down* en el pin del contador, así como en las dos entradas digitales habilitadas del Arduino, a fin de proveer a dichos pines de un nivel definido si es que no se encuentra conectado algún elemento a ellos.

3.4.7 Posición angular

El sensor utilizado es un potenciómetro de efecto Hall con escobillas, se realizó también una comprobación de su característica ángulo vs voltaje de salida, valiéndose de un compás y una aguja, alineando cuidadosamente esta última con el ángulo que se quería medir, la relación entre estas variables se muestra en la tabla 3-6 y la figura 3-14.

4 Resultados

Se presentan los resultados obtenidos tanto en el programa de Arduino, el programa en html5 y los sensores.

Se muestra el formato en el que los datos son enviados desde el Arduino y los tiempos que toma cada subproceso del programa respectivo; la interfaz final del programa en HTML5 y sus distintas funciones; y resultados obtenidos respecto a los sensores.

4.1 Arduino

Se midió el tiempo de ejecución de diferentes partes del programa, obteniendo como resultado

Entradas análogas: 12 μs
Enviar datos: 240 μs
Contador: 4 μs
Anexar datos :80 – 100 μs
Conversión ADC: 116 – 124 μs

Se puede notar que tanto para la conversión ADC, como anexar los datos, los tiempos varían ligeramente, esto presentaría un problema si es que la ejecución fuera lineal, es decir leer ADC; anexar datos; enviar datos; repetir. Lo cual no sucede con el uso del *Timer*, pues este se ejecuta en un tiempo fijo, siendo este tiempo mayor que lo que toman las lecturas de los canales y la anexión los datos, con cierta holgura en el Timer respecto a la toma y acomodo de datos, la velocidad de muestreo será constante.

En la figura 4-1 se aprecia como los datos son enviados por parte del Arduino, siendo los primeros grupos las entradas analógicas, seguido por dos caracteres correspondientes a las entradas digitales y finalmente el contador.

4.2 HTML 5

En la figura 4-2 se puede apreciar la interfaz del programa en HTML5 corriendo en NW.js.

Para esta se instaló dos sensores de intensidad luminosa a diferentes distancias de un LED, este LED a su vez fue accionado por un interruptor el cual a su vez también se conectó a la entrada

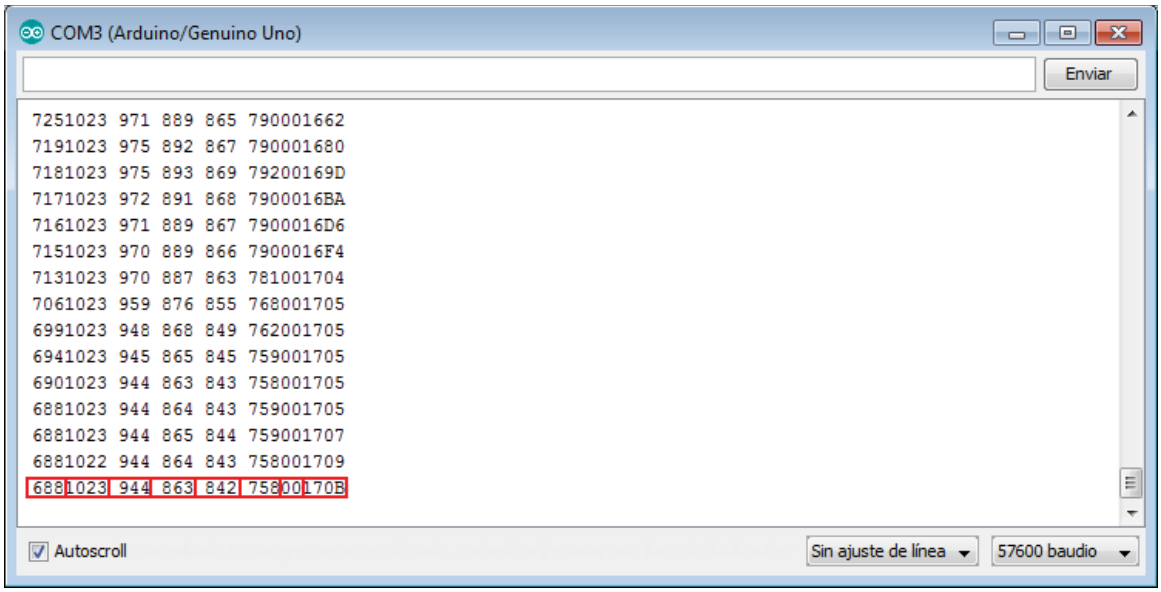

Figura 4-1: Datos enviados desde al Arduino tal como aparecen en la consola del IDE de Arduino.

del contador. No tiene un uso práctico esta situación, pero sirve para aclarar el funcionamiento del programa y de sus diferentes opciones.

1. Selector de canales

   Esta zona indica que canales se encuentran activos o inactivos en el gráfico, estos se pueden activar o desactivar haciendo clic en su nombre o respectivo circulo indicador.

2. *Dataview*, restaurar y guardar imagen

   El icono con forma de documento abre un cuadro dónde es posible ver y modificar los datos de cada canal como puede apreciarse en la figura 4-4, de ser modificados se puede hacer clic en el botón "*save*" y los cambios se verán reflejados en el gráfico.

   Restaurar permite deshacer los cambios hechos mediante *Dataview*.

   Guardar imagen abre una ventana que pregunta donde guardar una imagen del gráfico en el momento actual.

3. Valor instantáneo y función

   Muestra el último valor leído por el Arduino para determinado canal analógico, los cuadros de la derecha permiten la inserción de funciones usando como variable la letra "x", es decir modificará el valor instantáneo mostrado acorde a la función ingresada por el usuario. Para activar esta función se hace necesario hacer clic sobre el botón "f".

4. Gráfico principal

Área principal de la interfaz donde se muestran los datos tomados por las entradas analógicas. Si el puntero es posicionado en cualquier parte del grafico se desplegará un pequeño cuadro que mostrará el valor de todos los canales activos en ese punto.

5. Gráfico digital

   Muestra el valor en el tiempo de las entradas digitales, este gráfico esta sincronizado con el gráfico principal.

6. Contador

   Muestra el valor actual del contador del Arduino

7. Milisegundos entre muestras

   Control de milisegundos entre muestras, es posible insertar un valor numérico o ajustar usando la *slider*, de ambos modos los milisegundos son restringidos a valores preestablecidos, es decir si se ingresa el valor "101" en este cuadro, dicho valor será aproximado a 100.

8. Control de conexión

   Permite desactivar la autoconexión, cambiar los baudios o el puerto serial al que conectarse en caso de ser necesario.

9. Herramienta zoom y mini mapa

   Permite hacer zoom al arrastrar sus extremos con el mouse, además al deslizarlo permite centrar la visualización en cualquier parte de interés.
   Por su parte el mini mapa muestra una pequeña representación del gráfico principal permitiendo ubicarse a momento de hacer *zoom* y revisar otras áreas de interés, como puede verse en la figura 4-3.

10. *Pause*

    Pausa la actualización del gráfico, de modo de permitir una visualización de los datos durante tiempos indefinidamente prolongados, especialmente útil si se usa en conjunto con el *zoom* o la herramienta *Dataview*.
    No detiene la comunicación entre el programa y el Arduino.

Cabe mencionar que la velocidad a la que se actualiza el gráfico no es la misma a la que se reciben los datos son recibidos a medida que son enviados por el Arduino, en tanto el grafico es actualizado 10 veces por segundo, creando independencia entre ambos, dicho de otro modo, la renderización del gráfico en cualquier momento no interfiere con el procesamiento de los datos recibidos.

Otro punto importante es que mientras se tenga activada la autoconexión, el programa volverá a conectarse al Arduino si es presionado su botón de *reset* o reconectado.

Dos aspectos en los que se podría poner atención si es que se pretende realizar cambios en esta interfaz es la optimización del código y en modificar el orden y posición de los elementos si es que se considera confusa.

Respecto a la portabilidad se comprobó su vialidad y funcionamiento como era de esperarse, se probó en dos computadores con diferentes versiones de Windows, por un lado, Windows XP x32 y Windows7 x64 funcionando sin ningún problema, el único requerimiento adicional fueron los *drivers* de la placa Arduino.

En cuanto a Linux se revisó su funcionamiento en dos sistemas uno utilizando Debian y otro Ubuntu, si bien no hicieron falta *drivers*, si fue necesario entregarle permisos de acceso a los puertos seriales a NW.js.

Con esto se cumple con todo lo propuesto para este programa, visualización de datos tomados desde múltiples entradas del Arduino tanto analógicas como digitales, capacidad de revisar estos datos en detalle, posibilidad de ingresar funciones y una interfaz amigable.

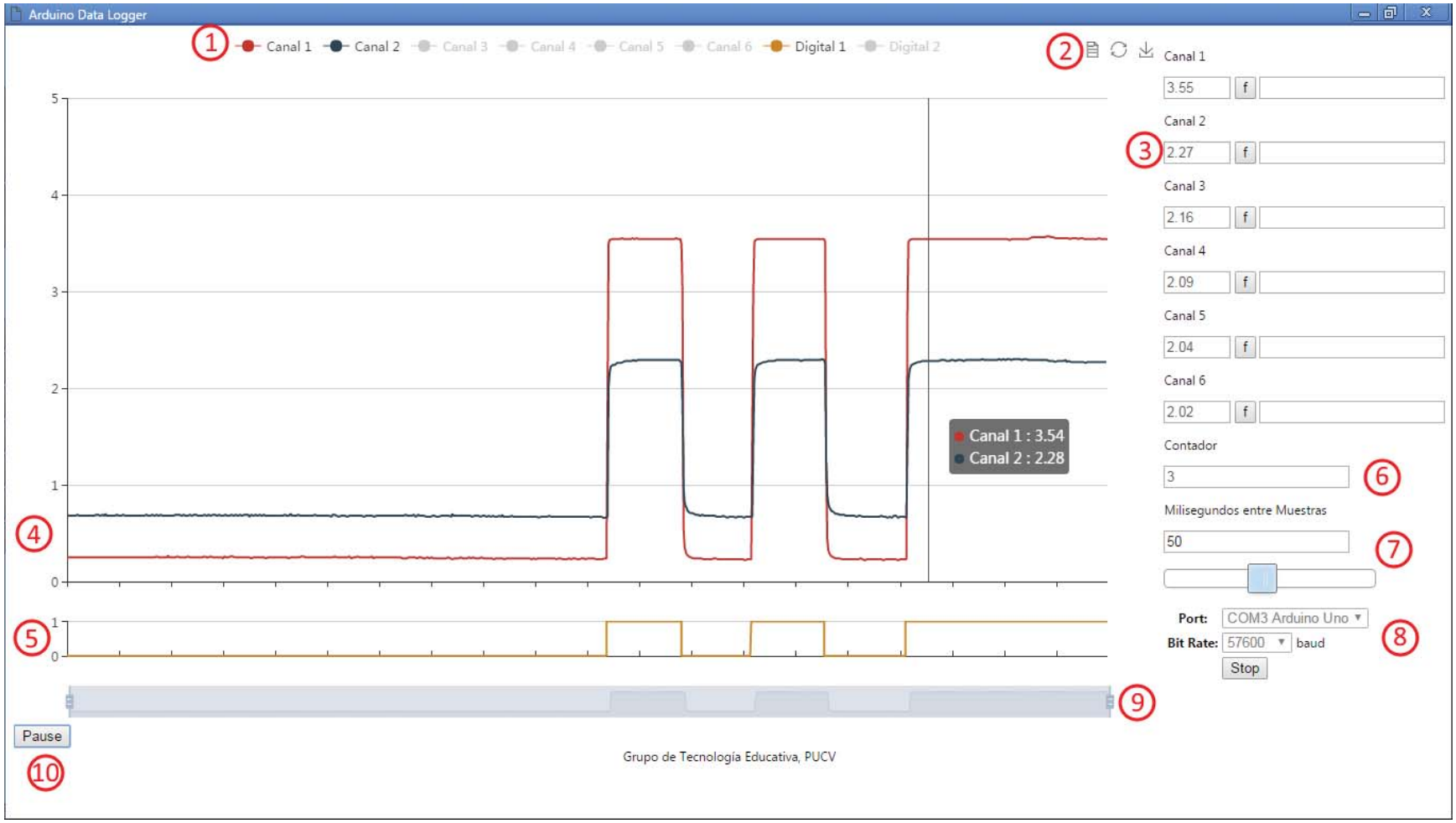

Figura 4-2: Interfaz programa en HTML5, dos señales son leídas desde los puertos análogos del Arduino y mostrados en pantalla en conjunto con una entrada digital, el resto de las entradas se encuentran temporalmente deshabilitadas y por tanto no son visibles.

Figura 4-3: Gráfico de la figura 4-2 al cual se le ha aplicado *zoom*, tal y como se ve al ser guardado en el computador

DataView

Canal 1	Canal 2	Canal 3	Canal 4	Canal 5	Canal 6
0.25	0.69	0.81	0.87	0.93	0.95
0.24	0.69	0.61	0.58	0.59	0.62
0.25	0.68	0.81	0.87	0.93	0.95
0.24	0.69	0.62	0.59	0.59	0.62
0.25	0.68	0.82	0.88	0.94	0.95
0.25	0.69	0.61	0.58	0.58	0.62
0.25	0.68	0.82	0.88	0.94	0.95
0.26	0.69	0.61	0.58	0.58	0.62
0.25	0.68	0.81	0.87	0.94	0.95
0.26	0.68	0.6	0.57	0.58	0.61
0.25	0.69	0.81	0.87	0.94	0.95
0.26	0.69	0.61	0.57	0.58	0.61
0.25	0.68	0.82	0.88	0.95	0.96
0.25	0.69	0.61	0.58	0.58	0.61
0.25	0.68	0.82	0.88	0.95	0.96
0.25	0.69	0.61	0.57	0.57	0.61
0.25	0.68	0.82	0.89	0.95	0.97
0.24	0.68	0.61	0.57	0.57	0.61
0.25	0.68	0.82	0.89	0.95	0.97
0.26	0.68	0.61	0.57	0.57	0.61
0.25	0.68	0.82	0.89	0.95	0.97
0.25	0.68	0.61	0.57	0.57	0.61

Cancel Save

Figura 4-4: Datos en detalle al ser revisados utilizando la opción *Dataview*

4.3 Sensores

Tanto los sensores de intensidad luminosa, posición angular y temperatura ya funcionaban de acuerdo a los requerimientos, sin embargo se comprobó su funcionamiento.

El sensor de posición angular presenta una relación lineal respecto a su posición.

En tanto la celda de carga se montó en una plataforma de acuerdo a lo mencionado en el capítulo anterior, dicha plataforma se puede apreciar en la figura 4-5, se comprobó que sin importar en que punto de la plataforma se ubique una misma carga, la salida será la idéntica.

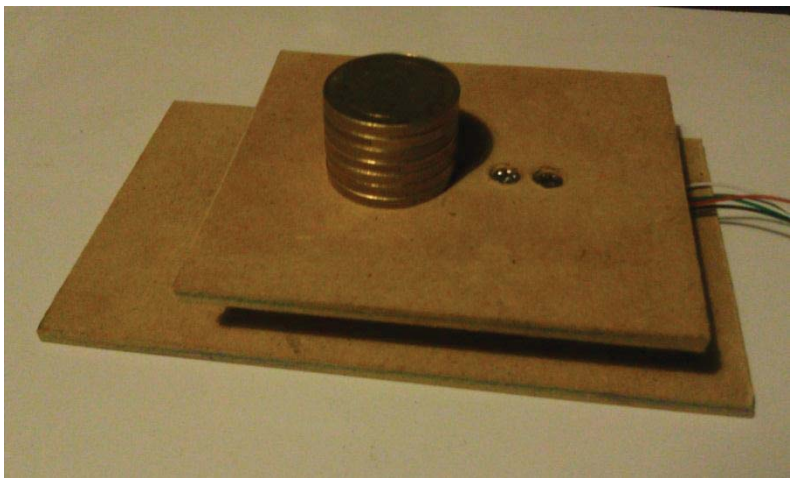

Figura 4-5: Pequeña plataforma utilizando la celda de carga para medir el peso de unas monedas.

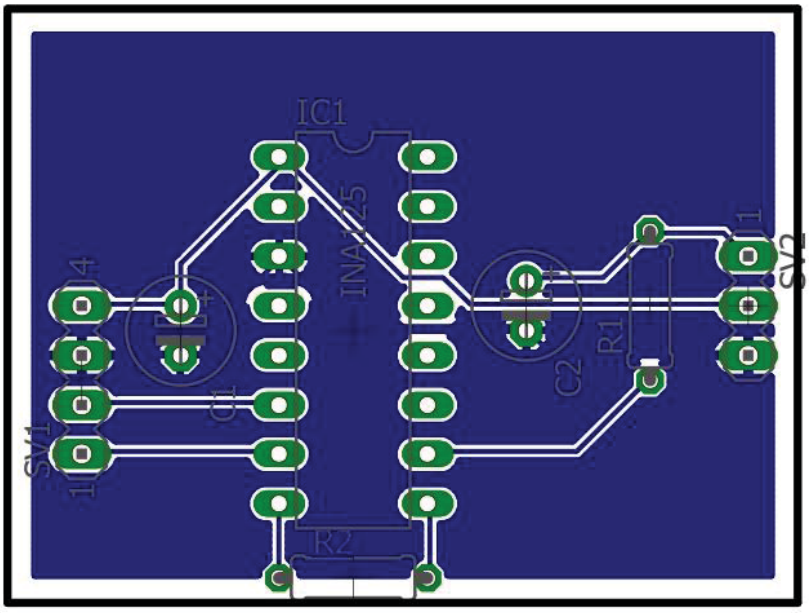

Figura 4-6: Placa del circuito de amplificación de la celda de carga.

Adicionalmente se hizo el diseño de la placa para el circuito de amplificación de la celda de carga, el cual se muestra en la figura 4-6.

En cuanto a los sensores de intensidad luminosa, temperatura y posición angular, se comprobó que funcionaban de acuerdo a los requerimientos establecidos según las actividades a realizar en el laboratorio de Física.

Discusión y conclusiones

La realización de este proyecto permitió una ampliación en las posibilidades y capacidades del sistema de adquisición de datos utilizados por el Instituto de Física en sus laboratorios, haciendo uso de elementos que ya se tenían disponibles y de todas las capacidades de la placa Arduino acorde a los objetivos.

Arduino

La velocidad de muestreo es el tiempo asignado al *Timer* más el tiempo que toma enviar los datos.

De este modo se toman muestras a tiempo fijas, se aumentó la velocidad máxima de muestreo y es posible controlar la velocidad de muestreo directamente desde el computador.

Además, con el programa escrito para el Arduino es posible leer las 6 entradas del ADC, 2 entradas digitales y se integró el funcionamiento de un contador.

HTML5

El programa cumplió con todos los requerimientos necesarios, proporcionando una interfaz gráfica sencilla e intuitiva, que permite analizar las mediciones tomadas con detenimiento y en detalle.

Posee auto-detección y auto-conexión a puertos seriales disponibles, haciendo falta solamente iniciar el programa para que este comience a mostrar las mediciones en pantalla.

La documentación al respecto es fundamental para la facilidad de modificación/optimización de este programa.

Sensores

Se construyó y probó satisfactoriamente un sensor de carga mecánica, con lo cual se podrán realizar actividades adicionales en las cuales la composición de algún elemento pueda cambiar respecto al tiempo.

Esto sumado a la capacidad de modificar la velocidad de muestreo, posibilita tomar una cantidad apropiada de muestras para dicho proceso, si es que es uno lento tomar muestras con una menor frecuencia o si es rápido con mayor frecuencia.

Trabajos Futuros

Como trabajos posteriores queda la optimización del código del *datalogger*, además de la posible adaptación de elementos de este para usos específicos en caso de ser requerido, como podrían ser control de salidas digitales o más entradas digitales.

Adicionalmente la incorporación de sensores extra, tanto sensores listos como la construcción de circuitos acordes para la medición de alguna variable o elemento de interés.

Un punto que puede resultar de interés es la posibilidad de realizar un *datalogger* mucho más veloz en término de muestras por segundo que el logrado aquí. Elementos a dejar de lado podrían ser lectura de múltiples canales con lo cual sería posible habilitar el modo *free running* del ADC; muestreo a 10bits con lo cual cada conversión duraría menos; modificar el factor de escalamiento de frecuencia del ADC con lo cual cada conversión se realizaría más rápido y último, pero bastante significativo podría ser habilitar un *buffer* dentro de la misma memoria del Arduino, evitando así tener que enviar los valores de cada lectura a medida que se vayan obteniendo sus valores con los efectos que ello tendría, esto de importancia, dado que el mayor cuello de botella es justamente el envío de datos mediante el puerto serial, pero al ejecutarlo de ese modo podría perderse la visualización "instantánea" de las lecturas.

Todo lo mencionado en el párrafo anterior no pudo ser explorado en la realización de este proyecto, puesto que entraba en conflicto con los objetivos del mismo, siendo estos la lectura de múltiples canales simultáneamente y mantener la resolución de muestreo.

Por otra parte, los elementos desarrollados en este proyecto podrían ser utilizados por separado si son modificados, es decir solo requerir el Arduino sin conexión al computador guardando los datos en una tarjeta SD, o el programa desarrollado en HTML5 usarlo para graficar datos enviados por un servidor o tomados de alguna otra forma.

Bibliografía

[1] Daniel K. Fisher, Peter J. Gould, Open-Source Hardware Is a Low-Cost Alternative for Scientific Instrumentation and Research, USDA Agricultural Research Service, Stoneville, USA, 2012

[2] Michael Heron , Vicki L Hanson and Ian Ricketts, Open source and accessibility: advantages and limitations. Journal of Interaction Science 2013

[3] C. K. Alexander y M. Sadiku, Circuits, Fundamentals of Electric, McGraw-Hill College, 2003.

[4] Hayt, Jr., William H. Engineering Electromagnetics, Third Edition. McGraw-Hill, 2010.

[5] R Coughlin,F Driscoll, Amplificadores operacionales y circuitos integrados lineales.1999

[6] Steve Patoray, Jeff Robidoux , Load Cell Application and Test Guideline, Scale Manufacturers Association, Columbus, Ohio, USA, 2010

[7] Bruce Carter, A Single-Supply Op-Amp Circuit Collection, Application Report, Texas instruments, 2000.

[8] Marijn Haverbeke, Eloquent JavaScript, Berlin, 2014.

A Programa HTML5

En este apéndice se presenta el código del programa en HMTL5, lo primero en cargar por parte de NW.js es la ventana, seguido por las librerías que no son dependientes de elementos de la interfaz gráfica, luego los diferentes elementos de esta y finalmente las hojas de código escritas además de la librería gráfica.

Listado A-1: Window.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <!-- Librerias y hojas de codigo a utilizar -->
        <link href="nouislider.min.css" rel="stylesheet">
        <script src="jquery-2.1.4.min.js"></script>
        <script src="underscore.js"></script>
        <script src="backbone.js"></script>
        <script src='Chart.js'></script>
        <script src="nouislider.js"></script>
        <script src="wNumb.js"></script>
        <script src="nerdamer/nerdamer.core.js"></script>
        <script src="nerdamer/Calculus.js"></script>

        <link rel="stylesheet" type="text/css" href="style.css">
    </head>

    <body>
        <aside id="side">
        <!-- Barra lateral donde se ubican controles varios -->
        <p> Canal 1 </p>
        <input type="textbox" name="valorCanal1" disabLED size="4">
        <button size="2" id="f1"onclick="fx1()">f</button>
        <input type="textbox" name="fCanal1" >
        <p> Canal 2 </p>
        <input type="textbox" name="valorCanal2" disabLED size="4">
        <button size="2" id="f2"onclick="fx2()">f</button>
        <input type="textbox" name="fCanal2" >
        <p> Canal 3 </p>
        <input type="textbox" name="valorCanal3" disabLED size="4">
        <button size="2" id="f3"onclick="fx3()">f</button>
        <input type="textbox" name="fCanal3" >
        <p> Canal 4 </p>
        <input type="textbox" name="valorCanal4" disabLED size="4">
        <button size="2" id="f4"onclick="fx4()">f</button>
        <input type="textbox" name="fCanal4" >
        <p> Canal 5 </p>
        <input type="textbox" name="valorCanal5" disabLED size="4">
        <button size="2" id="f5"onclick="fx5()">f</button>
        <input type="textbox" name="fCanal5" >
        <p> Canal 6 </p>
```

```
43          <input type="textbox" name="valorCanal6" disabLED size="4">
44          <button size="2" id="f6"onclick="fx6()">f</button>
45          <input type="textbox" name="fCanal6" >
46
47          <p> Contador </p>
48          <input type="textbox" name="contador" disabLED>
49          <p> Milisegundos entre Muestras </p>
50
51          <input type="textbox" id="input-format">
52          <div id="slider-format"></div>
53          <br>
54          <!-- Controles de conexion  -->
55          <div id="connection">
56              <table class="form">
57                  <tbody>
58                      <tr>
59                          <th><label for="port">Port:</label></th>
60                          <td>
61                              <select id="port" name="port">
62                              </select>
63                          </td>
64                      </tr>
65                      <tr>
66                          <th><label for="bitrate">Bit Rate:</label></th>
67                          <td>
68                              <select id="bitrate" name="bitrate">
69                                  <option value="9600">9600</option>
70                                  <option value="19200">19200</option>
71                                  <option value="38400">38400</option>
72                                  <option value="57600" selected>57600</option>
73                                  <option value="115200">115200</option>
74                              </select>
75                              baud
76                          </td>
77                      </tr>
78                      <tr>
79                          <th></th>
80                          <td>
81                              <button id="connect">Connect</button>
82                              <button id="stop-connection">Stop</button>
83                          </td>
84                      </tr>
85                  </tbody>
86              </table>
87          </div>
88          </aside>
89          <!-- Area del grafico -->
90          <div id="main" style="height:650px; width: 80%"></div>
91          <!-- Carga de libreria grafica y de hojas de codigo  -->
92          <script src="echarts.js"></script>
93          <script src="graph.js"></script>
94          <script src="connection.js"></script>
95          <div>
96          <button id="stop"onclick="stopFunction()">Pause</button>
97          </div>
98          <script src="index.js"></script>
99
100 <center>Grupo de Tecnología Educativa, PUCV</center>
101
102     </body>
103 </html>
104
```

Listado A-2: connection.js

```
// Se definen variables para almacenar temporalmente los datos
var buffer =[];
var buffrecibido= [];

var RETRY_CONNECT_MS = 1000;
var cid =0;
var connectionId;
var serialOk= 0;

// Se definen variables necesarias para la conexion del puerto serial
var Connection = Backbone.Model.extend({
    defaults: {
        connectionId: null,
        path: null,
        bitrate: 57600,
        dataBits: 'eight',
        parityBit: 'no',
        stopBits: 'one',
        autoConnect: undefined,
        ports: [],
        buffer: null,
        text: '...',
        error: '',
    },

    initialize: function() {
        chrome.serial.onReceive.addListener(this._onReceive.bind(this));

chrome.serial.onReceiveError.addListener(this._onReceiveError.bind(this));
    },

    enumeratePorts: function() {
        var self = this;
        chrome.serial.getDevices(function(ports) {
            self.set('ports', ports);
            self._checkPath();
        });
    },

    hasPorts: function() {
        return this.get('ports').length > 0;
    },

    autoConnect: function(enable) {
        this.set('autoConnect', enable);
        if (enable) {
            this._tryConnect();
        } else {
            this._disconnect();
        }
    },

    _tryConnect: function() {
        if (!this.get('autoConnect')) {
            return;
        }
        var path = this.get('path');
        if (path) {
            var self = this;
            var opts = {
                bitrate: this.get('bitrate'),
                dataBits: this.get('dataBits'),
                parityBit: this.get('parityBit'),
                stopBits: this.get('stopBits'),
            }

            chrome.serial.connect(path, opts, function(connectionInfo) {
                self.set('buffer', new Uint8Array(0));
```

```
69                  if (connectionInfo) {
70                    self.set('connectionId', connectionInfo.connectionId);
71                    window.cid = (connectionInfo.connectionId);
72                    serialOk= 1;
73                  } else {
74                    self.set('connectionId', null);
75                    self.set('autoConnect', false);
76                    self.set('error'+'</div>');
77                  }
78              });
79          } else {
80              this.enumeratePorts();
81              setTimeout(this._tryConnect.bind(this), RETRY_CONNECT_MS);
82          }
83          var cid = this.get('connectionId');
84      },
85
86      _disconnect: function() {
87          var msg =  '<br/>'
88
89          console.log(arrv);
90          var cid = this.get('connectionId');
91          if (!cid) {
92              return;
93          }
94
95          var self = this;
96          chrome.serial.disconnect(cid, function() {
97              self.set('connectionId', null);
98              self.enumeratePorts();
99          });
100     },
101
102     _checkPath: function() {
103         var path = this.get('path');
104         var ports = this.get('ports');
105
106         if (ports.length == 0) {
107             this.set('path', null);
108             return;
109         }
110
111         for (var i = 0; i < ports.length; ++i) {
112             var port = ports[i];
113             if (port.path == path) {
114                 return;
115             }
116         }
117
118         // Intentar conectarse al primer puerto listado
119         var portIdx = 0;
120         for (var i = 0; i < ports.length; ++i) {
121             var port = ports[i];
122             if (port.path.indexOf('/dev/cu.usbmodem') === 0) {
123                 portIdx = i;
124                 break;
125             }
126         }
127         this.set('path', ports[portIdx].path);
128     },
129
130     _onReceive: function(receiveInfo) {
131         var cid = this.get('connectionId');
132
133         var data = receiveInfo.data;
134
135         data = new Uint8Array(data);
136         // Junta el buffer con los ultimos datos recibidos
137         var result = catBuffers(buffer,data)
138         buffer = result;
139         // Busca un quiebre de linea
140
```

```
141         var lbr = findLineBreak(result);
142 
143         if (lbr !== undefined) {
144         // Si es que hay un quiebre de linea divide los datos entre una frase
145         //  correcta y lo sobrante
146             var txt = result.slice(0, lbr);
147             buffer = result.slice(lbr +1);
148             var correcto = [];
149             // Convierte los datos de ASCII decimal a ASCII hex
150             for(var i = 0; i < txt.length; ++i){
151                 correcto += txt[i].toString(16);
152             }
153             // Convierte los datos desde Hex a characters o string
154             correcto = hex_to_ascii(correcto);
155             // Envia los datos para ser graficados
156             graficar(correcto);
157         }
158     },
159 
160     _onReceiveError: function(info) {
161         this._disconnect();
162         this.set('error', info.error);
163         this.enumeratePorts();
164     }
165 });
166 
167 $(function() {
168     var connection = new Connection();
169     connection.on('change:text', function(c) {
170         var text = c.get('text');
171     });
172     connection.on('change:error', function(c) {
173         var text = c.get('error');
174     });
175     connection.on('change:ports', function(c) {
176         var ports = c.get('ports');
177         var $port = $('#port');
178         $port.empty();
179         for (var i = 0; i < ports.length; ++i) {
180             var port = ports[i];
181             $('<option value="' + port.path + '">' +
182               port.path + ' ' + (port.displayName || '') +
183 '</option>').appendTo($port);
184         }
185         if (ports.length == 0) {
186             $('<option value="">[no device found]</option>').appendTo($port);
187             $port.prop('disabLED', true);
188         } else {
189             $port.val(c.get('path'));
190         }
191     });
192     connection.on('change:autoConnect', function(c) {
193         var autoConnect = !!c.get('autoConnect');
194         $('#stop-connection').toggle(autoConnect);
195         $('#connect').toggle(!autoConnect);
196         $('#port').prop('disabLED', autoConnect || !c.hasPorts());
197         $('#bitrate, #dataBits, #parityBit, #stopBits').prop('disabLED',
198 autoConnect);
199     });
200     connection.on('change:path', function(c) {
201         var path = c.get('path');
202         $('#port').val(path);
203     });
204     connection.on('change:connectionId', function(c) {
205         var connected = !!c.get('connectionId');
206         $('.btn-connection').toggleClass('connected', connected);
207     });
208     $('#connect').click(function(e) {
209         e.preventDefault();
210         connection.send(true);
211     });
```

```
212     $('#port').change(function(e) {
213         connection.set('path', $(this).val());
214     });
215     $('#bitrate').change(function(e) {
216         connection.set('bitrate', parseInt($(this).val()));
217     });
218     $('#dataBits, #parityBit, #stopBits').change(function(e) {
219         connection.set($(this).attr('name'), $(this).val());
220     });
221     $('#stop-connection').click(function(e) {
222         e.preventDefault();
223         connection.autoConnect(false);
224     });
225     connection.autoConnect(true);
226 });
227
228 // Junta el buffer con los ultimos datos recibidos
229 function catBuffers(a, b) {
230     // Crea una nueva variable del tamaño del buffer y los datos recibidos
231     var result = new Uint8Array(a.length + b.length);
232     // Deposita el buffer en esta variable
233     result.set(a);
234     // Deposita los datos en esta variable
235     result.set(b, a.length);
236     // Devuelve el resultado
237     return result;
238 }
239
240 // Busca un quiebre de linea dado por el valor "10" de ASCII decimal
241 // (fin de linea)
242 function findLineBreak(b) {
243     for (var i = 0; i < b.length; ++i) {
244         if (b[i] == 10)
245             return i;
246     }
247 }
248 // Convierte un grupo de datos hexadecimales en caracteres ASCII
249 function hex_to_ascii(str1)
250  {
251     var hex  = str1.toString();
252     var str = '';
253     for (var n = 0; n < hex.length; n += 2) {
254         str += String.fromCharCode(parseInt(hex.substr(n, 2), 16));
255     }
256     return str;
257  }
258
259 // Enviar datos serial
260  var writeSerial=function(str) {
261   chrome.serial.send(window.cid, convertStringToArrayBuffer(str),
262 function(sendInfo) {
263     if (sendInfo.error) {
264         console.log(sendInfo.error);
265     } else if (sendInfo.bytesSent > 0) {
266         console.log('Message sent!');
267     }
268   });
269 }
270
271 // Convertir string a ArrayBuffer
272 var convertStringToArrayBuffer=function(str) {
273     var buf=new ArrayBuffer(str.length);
274   var bufView=new Uint8Array(buf);
275   for (var i=0; i<str.length; i++) {
276     bufView[i]=str.charCodeAt(i);
277   }
278   return buf;
279 }
280
```

Listado A-3: graph.js

```
// Inicia con pausa desactivada
var stop =true;

// Preparacion de datos iniciales
var axisx =  new Array(1000).fill(null);
var dataDig1= new Array(1000).fill(0);
var dataDig2= new Array(1000).fill(0);
var dataChan1 =  new Array(1000).fill(0);
var dataChan2 =  new Array(1000).fill(0);
var dataChan3 =  new Array(1000).fill(0);
var dataChan4 =  new Array(1000).fill(0);
var dataChan5 =  new Array(1000).fill(0);
var dataChan6 =  new Array(1000).fill(0);

// Asigna la ubicacion que tendra el grafico
var grafico = echarts.init(document.getElementById('main'));
// Define las opciones del grafico
var option = {
        tooltip: {
            trigger: 'axis',
            transitionDuration: 0,
            showDelay: 0
        },
        legend: {
            data: ['Canal 1', 'Canal 2', 'Canal 3', 'Canal 4', 'Canal 5',...
            'Canal 6','Digital 1','Digital 2']
        },
        // Sin animacion para mejorar su desempeño
        animation: false,
        addDataAnimation: false,
        // Activar opciones del toolbox de Echarts, guardar imagen, revision
        // de datos y restauracion
        toolbox: {
            show: true,
            feature: {
                dataView: { show: true, readOnly: false },
                restore: { show: true },
                saveAsImage: { show: true }
            }
        },
        calculable: true,
        grid: [{
        left: 50,
        right: 50,
        height: '70%'
    }, {
        left: 50,
        right: 50,
        top: '85%',
        height: '5%'
    }],
        xAxis: [{
            type: 'category',
            boundaryGap: false,
            show: true,
            splitLine: {
                show: false
            },
            data: axisx
        },
        {
            gridIndex: 1,
            type : 'category',
            boundaryGap : false,
            axisLine: {onZero: true},
            data: axisx,
            position: 'top'
        }],
```

```
69          yAxis: [{
70              type: 'value',
71              min: null,
72              max: 5
73          },
74          {
75                 gridIndex: 1,
76                  type: 'value',
78                  max:1,
79                  splitNumber:1
80              }],
81          series: [{
82              name: 'Canal 1',
83              type: 'line',
84              symbol: 'none',
85              data: dataChan1
86          }, {
87              name: 'Canal 2',
88              type: 'line',
89              symbol: 'none',
90              data: dataChan2
91          }, {
92              name: 'Canal 3',
93              type: 'line',
94              symbol: 'none',
95              data: dataChan3
96          }, {
97              name: 'Canal 4',
98              type: 'line',
99              symbol: 'none',
100             data: dataChan4
101         }, {
102             name: 'Canal 5',
103             type: 'line',
104             symbol: 'none',
105             data: dataChan5
106         }, {
107             name: 'Canal 6',
108             type: 'line',
109             symbol: 'none',
110             data: dataChan6
111         },
112         {
113             name:'Digital 1',
114             type:'line',
115             xAxisIndex: 1,
116             yAxisIndex: 1,
117             symbol: 'none',
118             step: 'start',
119             data:dataDig1
120         },{
121             name:'Digital 2',
122             type:'line',
123             xAxisIndex: 1,
124             yAxisIndex: 1,
125             symbol: 'none',
126             step: 'start',
127             data:dataDig2
128         }],
129         // Activa la capacidad de zoom en el grafico
130         dataZoom: {
131             show: true,
132             realtime : true,
133             xAxisIndex: [0, 1]
134         }
135 };
136
137 // Generar grafico inicial
138 grafico.setOption(option);
139
140 // Controla la opcion de pausar la actualizacion del grafico
```

```
var stopFunction = function() {
    stop = !stop;
}

// Funcion de actualizacion del grafico
setInterval(function() {
    // Si se ha activado la pausa, pausar
    if (stop){
    grafico.setOption(option);
}
}, 100);
```

Listado A-4: Index.js

```
// Asigna un valor por defecto a las funciones
var f1 = "x";
var f2 = "x";
var f3 = "x";
var f4 = "x";
var f5 = "x";
var f6 = "x";

// Convierte el valor entregado por las salidas digitales de arduino a volts
function toVolt(data){
    data = (data*0.00488).toFixed(2);
    return data;
}

function graficar(n){
  var datatemp = toVolt(n.slice(0, 4));
  if (stop){

document.getElementsByName('valorCanal1')[0].value=nerdamer(f1,{x:datatemp})
    .toTeX('decimal');
  }
  dataChan1.push(datatemp);
  dataChan1.shift();
  var datatemp = toVolt(n.slice(5, 8));
  if (stop){

document.getElementsByName('valorCanal2')[0].value=nerdamer(f2,{x:datatemp})
    .toTeX('decimal');
  }
  dataChan2.push(datatemp);
  dataChan2.shift();
  var datatemp = toVolt(n.slice(9, 12));
  if (stop){

document.getElementsByName('valorCanal3')[0].value=nerdamer(f3,{x:datatemp})
    .toTeX('decimal');
  }
  dataChan3.push(datatemp);
  dataChan3.shift();
  var datatemp = toVolt(n.slice(13, 16));
  if (stop){

document.getElementsByName('valorCanal4')[0].value=nerdamer(f4,{x:datatemp})
    .toTeX('decimal');
  }
  dataChan4.push(datatemp);
  dataChan4.shift();
  var datatemp = toVolt(n.slice(16, 20));
  if (stop){

document.getElementsByName('valorCanal5')[0].value=nerdamer(f5,{x:datatemp})
```

```
    .toTeX('decimal');
  }
  dataChan5.push(datatemp);
  dataChan5.shift();
  var datatemp = toVolt(n.slice(20, 24));
  if (stop){

document.getElementsByName('valorCanal6')[0].value=nerdamer(f6,{x:datatemp})
    .toTeX('decimal');
  }
  dataChan6.push(datatemp);
  dataChan6.shift();
  var datatemp = n.slice(24, 25);
  dataDig1.push(datatemp);
  dataDig1.shift();
  var datatemp = n.slice(25,26);
  dataDig2.push(datatemp);
  dataDig2.shift();
  // Convierte el valor del contador a decimal y lo muestra en su cuadro
  document.getElementsByName('contador')[0].value=parseInt(n.slice(26,30),
16);
}

// Define el slider
var sliderFormat = document.getElementById('slider-format');
// Establece propiedades del slider
noUiSlider.create(sliderFormat, {
    start: [ 100],
    range: {
        'min': [     1,1 ],
        '30%': [   10,  10 ],
        '70%': [   100,  100 ],
        'max': [ 1000 ]
    },
    format: wNumb({
        decimals: 0,
        thousand: '',
    }),
});

var inputFormat = document.getElementById('input-format');
sliderFormat.noUiSlider.on('update', function( values, handle ) {
    inputFormat.value = values[handle];

});
// Detecta si el valor del slider ha cambiado si lo ha hecho envia ese nuevo
// valor mediante el puerto serial
inputFormat.addEventListener('change', function(){
    sliderFormat.noUiSlider.set(this.value);
    writeSerial(this.value);
});

// Carga la funcion insertada por el usuario
function fx1(){
    f1 = document.getElementsByName('fCanal1')[0].value;
// Si no hay nada ingresado, no hay funcion y pasa el valor directamente
    if (f1.length <1){
      f1 = "x";
    }
// Se guarda la funcion, la cual será evaluada en cada llegada de datos
    f1 = nerdamer(f1);
}

function fx2(){
    f2 = document.getElementsByName('fCanal2')[0].value;
    if (f2.length <1){
      f2 = "x";
    }
    f2 = nerdamer(f1);
}
function fx3(){
```

```
124     f3 = document.getElementsByName('fCanal3')[0].value;
125     if (f3.length <1){
126       f3 = "x";
127     }
128     f3 = nerdamer(f3);
129 }
130 function fx4(){
131     f4 = document.getElementsByName('fCanal4')[0].value;
132     if (f4.length <1){
133       f4 = "x";
134     }
135     f4 = nerdamer(f4);
136 }
137 function fx5(){
138     f5 = document.getElementsByName('fCanal5')[0].value;
139     if (f5.length <1){
140       f5 = "x";
141     }
142     f5 = nerdamer(f5);
143 }
144 function fx5(){
145     f5 = document.getElementsByName('fCanal6')[0].value;
146     if (f5.length <1){
147       f5 = "x";
148     }
149     f5 = nerdamer(f5);
150 }
```

B Programa Arduino

Listado B-1: Arduino_serial.ino, Programa de Arduino.

```
// Contador
volatile int conteo;
int pin_int = 0;

// Timer
#include <TimerOne.h>
volatile double tiempo;

// ADC
volatile int readFlag;
volatile byte ADL,ADH;
volatile int canal=0;
String datos;
char buff[4];
char dest[30];

// Digital
char D3,D4;


void setup() {

  // ADC
    //ADMUX
      // REFS1..0 seteados para operar el ADC con VCC (01)
      // ADLAR (0) -> ADCL 8 bits, ADCH 2 bits mas significativos
      // | REFS1 | REFS0 | ADLAR | â€“ | MUX3 | MUX2 | MUX1 | MUX0 |
  ADMUX = B01000000;
    // ADCSRA
      // ADEN: settear para activar el ADC
      // ADPS2..0 todos seteados para tener el prescaler en 120 => 125khz
      // ADIE: Habilitar las interrupciones por ADC
      // ADSC: Iniciar conversion
      // | ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 |
  ADCSRA = B11001111;

  // Digital
        // Define ambos pines como entradas digitales
  pinMode(3, INPUT);
  pinMode(4, INPUT);


  // Serial
        // Inicializa el puerto serial del arduino
  Serial.begin(57600);
    // Contador
    // pin_int: pin con el cual se contara
    // contador: funcion llamada por la interrupcion
    // rising: solo los cantos de subida activan la interrupcion
  attachInterrupt(pin_int, contador, RISING);
```

```
51      // Timer
52      // Define un tiempo inicial para la interrupcion con timer (µs)
53      // Funcion que será llamada mediante esta interrupcion
54    Timer1.initialize(100000);
55    Timer1.attachInterrupt(timer);
56
57      // Habilitar interrupciones globales
58    sei();
59  }
60
61  void contador() {
62
63    // Aumenta en una unidad el conteo
64    conteo++;
65  }
66
67  void timer(){
68    // Lee los pines digitales
69    D3= bitRead(PIND,3);
70    D4= bitRead(PIND,4);
71    // Anexa todos los datos
72    sprintf( buff,"%1i%1i", D3,D4);
73    strcat(dest,buff);
74    sprintf(buff,"%04X",conteo);
75    strcat(dest,buff);
76    memset(buff,0,sizeof(buff));
78
79    //Envia los datos mediante el puerto serial
80    Serial.print(dest);
81    Serial.write(0x0A);
82
83    // Inicia nuevamente el ciclo de lecturas de las entradas analogas
84    ADCSRA |=B01000000;
85
86    // Vacia el arreglo de datos
87    memset(dest,0,sizeof(dest));
88  }
89
90  void loop() {
91
92    // Revisa si hay datos esperando en el puerto serial
93    // De ser asi los convierte a su valor numerico
94    // Y cambia el tiempo del Timer
95    if(Serial.available()){
96      tiempo = Serial.readString().toInt();
97      noInterrupts();
98      Timer1.initialize(1000*tiempo);
99      interrupts();
100   }
101   // Si un dato ya fue leido
102   if (readFlag ==1){
103     // Desactiva el flag de lectura
104     readFlag = 0;
105     // Anexar datos
106     unsigned int word = ((unsigned int)ADH << 8) + ADL;
107     sprintf( buff,"%4i", word);
108     strcat(dest, buff);
109     // Vacia el arreglo temporal
110     memset(buff,0,sizeof(buff));
111     }
112 }
113
114
115 ISR(ADC_vect){
116
117   // Guarda la lectura ADC
118   ADL=ADCL;
119   ADH=ADCH;
120   // Si el canal es menor a 6
121   if (canal <6)
122   {
```

```
123     // Siguiente canal
124     canal++;
125     ADMUX++;
126    }
127    // Si el canal a llamar la siguiente vuelta es el 6
128    // asignar al canal 0 y pausar el ciclo de conversiones
129    if (canal ==6){
130      canal = 0;
131      ADMUX = B01000000;
132    }
133    // Si el canal a leer a continuacion no es superior al 5
134    // iniciar otra conversion del ADC
135    if (canal <5){
136    ADCSRA |=B01000000;
137    }
138    // Activa el Flag de lectura
139    readFlag = 1;
140  }
```