UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Manual de prácticas para la asignatura de Instrumentación para la carrera de Mecatrónica de la Facultad de Ingeniería

MATERIAL DIDÁCTICO

Que para obtener el título de

Ingeniero en Mecatrónica

P R E S E N T A

Enrique Ugarte Hernández

ASESOR(A) DE MATERIAL DIDÁCTICO

M. en C. Arturo Ronquillo Arvizu

Ciudad Universitaria, Cd. Mx., 2018

Agradecimientos

La realización del presente manual de prácticas ha sido posible gracias al proyecto PAPIME PE109217, ya que con los recursos autorizados fue posible la adquisición del material necesario para el diseño de las prácticas y la adquisición del software LabVIEW.

Gracias a la Dra. Elva Escobar directora del Instituto de Ciencias del Mar y Limnología (ICML) y a la Dra. María Adela Monreal secretaria académica de este mismo instituto, quienes promueven y motivan nuestra participación en proyectos de esta naturaleza y dan seguimiento para su buen fin. Cabe mencionar que dicho proyecto fue desarrollado en las instalaciones del laboratorio de Instrumentación Oceanográfica de este instituto.

Este trabajo fue desarrollado en equipo con la participación de los estudiantes Jorge Farfán, Adrián Juárez y especialmente Enrique Ugarte. Es difícil mencionar a todos los alumnos que de manera indirecta participaron en este proyecto, debido a que durante los últimos 10 años han hecho aportaciones interesantes cada vez que han desarrollado algunas de estas prácticas, lo que ha dado forma a muchas de ellas.

Gracias al Ingeniero Moisés Eugenio Rueda Gutiérrez, al Maestro Luis Yair Bautista Blanco, al Maestro Rigel Gámez Leal y al Maestro Ulises Peñuelas Rivas por su colaboración en el desarrollo de este proyecto y revisión del manual, al profesor Mauricio Iván Escalante Camargo, quien nos apoyó durante la realización de la práctica del tubo de Pitot en el laboratorio de Termofluidos y a la Dra. Sara Ivonne Franco responsable del laboratorio de Geodesia Satelital del Instituto de Geofísica de la UNAM, por su apoyo en la práctica del GPS.

Finalmente gracias a la Maestra Janete Mejía Jiménez, quien amablemente nos procuró un espacio en el laboratorio de cómputo para la docencia donde ha sido posible desarrollar estas prácticas cada semestre con los grupos de instrumentación e instrumentación y control.

M en C. Arturo Ronquillo Arvizu

Agradecimientos

He logrado terminar este trabajo gracias al apoyo de muchas personas quienes han creído en mí y que de algún modo han estado allí, brindándome su confianza y las herramientas necesarias para seguir adelante y no declinar bajo ninguna circunstancia. Agradezco a mi familia por estar presente en cada momento de necesidad y por haberme guiado hacia este camino, el cual me siento muy contento de haber elegido.

Gracias a mi hermana Nadia Kalina Ugarte Hernández por haber sido una gran compañera y ejemplo de fortaleza. Siempre contará conmigo en cualquier situación. Gracias a mi padre Enrique Ugarte Amador por haberme vuelto un hombre fuerte y responsable y gracias a mi madre Diana Hernández González por haberme cuidado siempre, trabajando día y noche con ayuda de mi padre para darnos estudios a mi hermana y a mí.

Quiero agradecer a la Facultad de Ingeniería por haberme dado la preparación requerida para realizar este proyecto me siento contento por haber tenido tan buenos maestros. Y me siento especialmente agradecido con mis compañeros de carrera, quienes han sido buenos amigos y me han apoyado mucho, especialmente a la hora de realizar proyecto. Me enorgullece ver que haya tanto talento en la UNAM.

Agradezco al Maestro Arturo Ronquillo Arvizu por haberme permitido participar en este proyecto, he aprendido mucho de él en muchos aspectos y para mi representa un buen ejemplo a seguir. Gracias al Instituto de Ciencias del mar y Limnología por haberme brindado el material para elaborar el manual y haberme apoyado con una beca, la cual fue de mucha ayuda. Finalmente agradezco al Maestro Yahir Bautista Blanco por haberme dado muchas ideas para la elaboración de las prácticas y al Maestro Mauricio Iván Escalante Camargo, maestro de mecánica de fluidos, de quien apoyo incondicional para la elaboración de la práctica sobre el Tubo de Pitot.

Enrique Ugarte Hernández

Prólogo

Este manual de prácticas proporciona al estudiante una guía para desarrollar aplicaciones en el área de la instrumentación, para lo cual básicamente se requiere una computadora con el software LabVIEW e IDE de Arduino y algunos sensores y/o actuadores de bajo costo.

El presente manual no tiene instruye al alumno en temas de programación; sin embargo, se explican paso a paso los programas (en LabVIEW e IDE de Arduino) que se presentan en las prácticas. Es importante que se consulte otras fuentes para complementar el conocimiento en el área de programación.

La asignatura de instrumentación, al igual que muchas otras asignaturas de las carreras de ingeniería, necesita la elaboración de prácticas por parte de los alumnos para adquirir conocimiento más significativo: ejercicios que les permita a los alumnos desarrollar habilidades en el manejo de software y hardware, así como resolver problemas similares a los que se presentan realmente en la industria.

La asignatura de instrumentación debe impartirse en un laboratorio dotado con computadoras con software afín a la instrumentación, sensores, tarjetas de adquisición de datos, etcétera; sin embargo, no contar con tal infraestructura no debe ser un limitante para impartir dicha asignatura de manera práctica, ya que actualmente casi todos los estudiantes tienen acceso a alguna computadora, pueden adquirir una tarjeta Arduino (que hace las veces de una tarjeta de adquisición de datos) y algunos sensores y/o actuadores de bajo costo para diseñar sus propios prototipos.

La idea de desarrollar un manual de prácticas para la asignatura de instrumentación nace justamente de la necesidad de una guía para los alumnos y el profesor donde esté documentado los objetivos, el material a utilizar y el procedimiento paso a paso para desarrollar cada una de las prácticas.

Casi todas las prácticas incluyen software y hardware, excepto las prácticas donde se propone descargar datos de internet, por ejemplo: datos de una estación mareográfica, datos de boyas oceanográficas, etcétera. Vale la pena que el profesor motive a los estudiantes para que utilicen su ingenio y dar un enfoque o aplicación diferente a cada una de las prácticas.

Mi experiencia de casi diez años como técnico académico del Instituto de Ciencias del Mar y Limnología, encargado de la instrumentación especializada de investigación oceanográfica a bordo de los buques de la UNAM (El Puma y Justo Sierra), me ha permitido conocer las aplicaciones de la instrumentación en este ámbito, y a pesar de que la instrumentación se encuentra en una gran variedad de disciplinas, los protocolos de comunicación, las necesidades de software y hardware son muy similares.

Cabe mencionar que en este manual se ha propuesto utilizar la tarjeta Arduino UNO como tarjeta de adquisición de datos, por ser una de las tarjetas más económicas y al alcance de los estudiantes; sin embargo, se hace hincapié en la mayoría de las prácticas en la comunicación serie, ya que es uno de los protocolos más utilizados en la industria, de tal manera que utilizando comunicación serie el estudiante pueda desarrollar proyectos para comunicarse con muchos instrumentos que tengan disponible un puerto RS-232 o USB, así como con otros dispositivos utilizando dicho protocolo.

También se propone utilizar lenguaje de programación gráfico LabVIEW para la adquisición y procesamiento de los datos de diferentes sensores. Con la observación de que no siempre es necesario tener una interfaz de usuario en LabVIEW; esto dependerá de las necesidades del proyecto.

Algunas prácticas que se presentan en esta manual pueden ser muy útiles para estudiantes de otras carreras, por ejemplo: Geofísica, Computación, Robótica, Ciencias de la tierra, Mecánica, etcétera.

M en C. Arturo Ronquillo Arvizu

Manual de prácticas para la asignatura de Instrumentación para la carrera de Mecatrónica de la Facultad de Ingeniería UNAM

Proyecto PAPIME PE109217

Índice

INTRODUCCIÓN

El concepto instrumentación es utilizado en varias disciplinas, como la música, la medicina, la electrónica, entre otras áreas [1]. Para este caso se maneja el concepto de instrumentación electrónica para trabajar en las prácticas. La instrumentación electrónica es una parte de la electrónica que se encarga de medir magnitudes de cualquier clase, aunque no sean eléctricas [2]. Es importante tanto en la industria como en la investigación, ya que permite medir las variables físicas de interés e implementar control para los procesos [3].

Los procesos industriales de la actualidad están constituidos de tres partes: sensores, actuadores y un controlador. Los actuadores son los dispositivos con la capacidad de generar una fuerza que ejerce un cambio de posición, velocidad o estado de algún tipo sobre un elemento mecánico, a partir de la transformación de energía [4]; estos se encargan de realizar las tareas en los procesos. Existen cuatro tipos de actuadores: manuales, eléctricos, hidráulicos y neumáticos [4, 5].

Los controladores son aquellos capaces de procesar las señales de entrada para compensar el error entre el valor deseado de alguna variable física y el valor real de la misma. En la industria se ocupan principalmente los PLC (controladores lógicos programables), aunque también pueden usarse distintas tarjetas de adquisición (por ejemplo: DAQ, Texas Instruments LaunchPad, Arduino, etcétera) [6,7].

Un sensor es un dispositivo que está capacitado para detectar acciones o estímulos externos y responder en consecuencia [8]. Los sensores son los instrumentos de entrada que proveen una salida manipulable de la variable física medida [4]. Los sensores están íntimamente relacionados con un elemento muy importante llamado transductor. Se denomina transductor a todo dispositivo que convierte una señal física en una señal de otro tipo, por lo general transforman cualquier magnitud física a una magnitud eléctrica. EN otras palabras el sensor tiene una propiedad, la cual se altera su magnitud en función del medio, mientras que el transductor recibe esa magnitud y la convierte en una magnitud eléctrica.

En algunos casos el transductor y el sensor resultan ser el mismo dispositivo, como es el caso del sensor piezoeléctrico, ya que produce tensión en función de la deformación de forma directa. Existen otros casos en que el transductor resulta ser parte del sensor, por ejemplo, en los sensores piezorresistivos, ya que el sensor recibe una señal, la cual altera la resistencia del material, mientras que el transductor son los demás componentes, los cuales traducen la resistencia en una diferencia de potencial. Hay sensores que no utilizan transductores, como el termómetro de mercurio. Cabe decir que los transductores no se utilizan solamente para medir magnitudes, también se utilizan para el funcionamiento de muchos dispositivos, por ejemplo, en los micrófonos [9].

A continuación se muestra una clasificación de los tipos de sensores según su principio de transducción [4]:

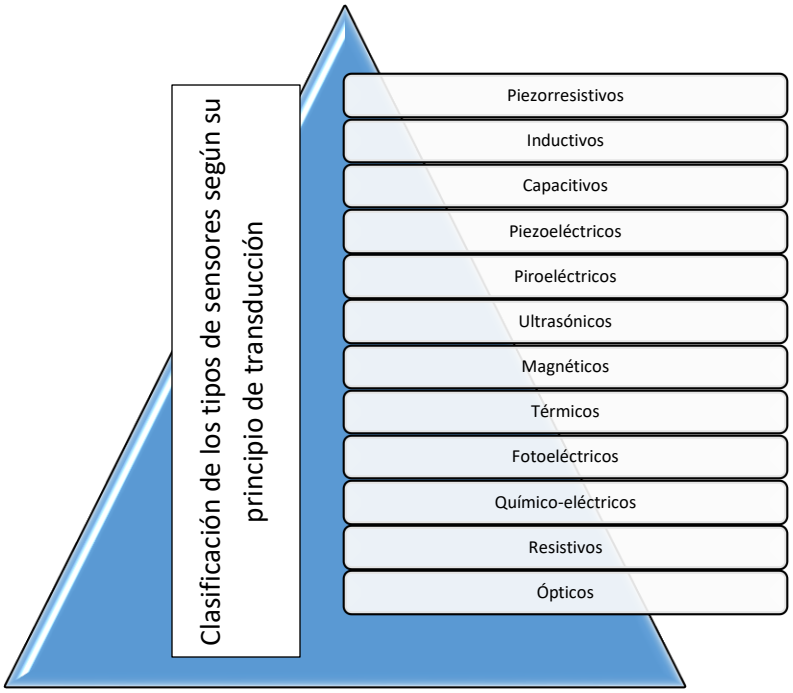


Figura 1. Clasificación de los tipos de sensores.

Muchas veces es conveniente poder visualizar los datos medidos de alguna variable física; para esto existen distintos sistemas eléctricos de medida: los instrumentos de propósito general (como los volmetros, multímetros, osciloscopios, etc.), los sistemas de adquisición de datos (como las tarjetas de adquisición de datos) y los instrumentos virtuales; estos últimos están basados en sistemas de adquisición de datos y software (Lenguaje de programación gráfico LabVIEW , Visual Studio, Embarcadero Delphi, entre otros) [2, 10, 11].

Este manual ocupa dos herramientas principales:

- Tarjeta Arduino (como DAQ)
- Software LabVIEW (versión 2012 o superior)

Arduino es una plataforma de código abierto dirigida a estudiantes sin conocimientos previos de electrónica. Las placas constan de varias terminales, las cuales se pueden configurar como entradas o salidas, ya sea digitales o analógicas. Arduino ha sido utilizado ampliamente por los estudiantes, ya que ofrece muchas ventajas: son económicos, el entorno de desarrollo integrado (IDE) de Arduino es gratuito y corre en distintos sistemas operativos, software de código abierto y extensible, así como hardware extensible para tareas específicas [4,12].

LabVIEW es un lenguaje de programación gráfico pensado para ingenieros y científicos para diseñar aplicaciones de pruebas, permite diseñar sistemas gráficos y generar sistemas embebidos, combina la programación gráfica abierta con hardware para facilitar el trabajo, además, se puede descargar una biblioteca especial para Arduino (LIFA), o bien, utilizar el puerto serie para comunicar el software con la tarjeta [13].

FUENTES DE CONSULTA

[1] Pérez P., Julián y Merino, María. (2008. Actualizado: 2012). Definición de instrumentación. *Definicion.de*. Recuperado de https://definicion.de/instrumentacion/ .

[2] Granda, M., Mercedes y Mediavilla B., Elena. (2010). Instrumentación electrónica: Transductores y acondicionadores de señal. Cantabria, España. Editorial Universidad de Cantabria.

[3] FRANCOR. (2018). INSTRUMENTACIÓN INDUSTRIAL. *francor.com*. Recuperado de http://francor.com.mx/instrumentacion-industrial/ .

[4] Corona Rodríguez, L. G. (2014). Sensores y Actuadores: Aplicaciones con Arduino. Grupo Editorial Patria.

[5] Valvias. (2007-2013). Tipos de Actuadores. *Valvias.com*. Recuperado de http://www.valvias.com/tipos-de-actuadores.php .

[6] Logicbus S.A. de C.V. (2015). Tarjetas para Adquisición de datos (DAQ). *Logicbus.com*. Recuperado de http://www.logicbus.com.mx/adquisicion-de-datos.php .

[7] ColdFire Electronica. (s.f.) Tarjetas de Desarrollo > Texas Instruments. Gdl, Jal. México. *Coldfire-electronica.com*. Recuperado de https://www.coldfire-electronica.com/esp/item/110/44/tarjeta-ti-hercules-launchpad-launchxl-rm42 .

[8] Definicion.de. (2008-2018). DEFINICIÓN DE SENSOR. *https://definicion.de*. Recuperado de https://definicion.de/sensor/ .

[9] GA. (2003). Transductores. *Ehu.eus*. Recuperado de http://www.ehu.eus/acustica/espanol/electricidad/transes/transes.html .

[10] Embarcadero Technologies, Inc. (2018). Delphi - Información General. *Embarcadero.com*. Recuperado de https://www.embarcadero.com/es/products/delphi .

[11] Microsoft (2018). Visual Studio. *Visualstudio.com*. Recuperado de https://www.visualstudio.com/es/ .

[12] Arduino (2018). What is Arduino?. *www.arduino.cc*. Recuperado de https://www.arduino.cc/en/Guide/Introduction .

[13] National Instruments Corporation. (2014). ¿Por qué usar LabVIEW?. *www.ni.com*. Recuperado de https://www.ni.com/academic/why_labview/esa/ .

Introducción al entorno de desarrollo de LabVIEW

En esta sección se presentan algunos bloques y funciones importantes para comenzar a programar en LabVIEW sin conocimientos previos. Esta breve introducción fue realizada en LabVIEW 2016. Los requerimientos para la instalación del software se explican en la práctica "Instalación de software".

I. GENERACIÓN DE UN DE PROGRAMA DE LABVIEW

Para crear una aplicación ir a File → New VI.

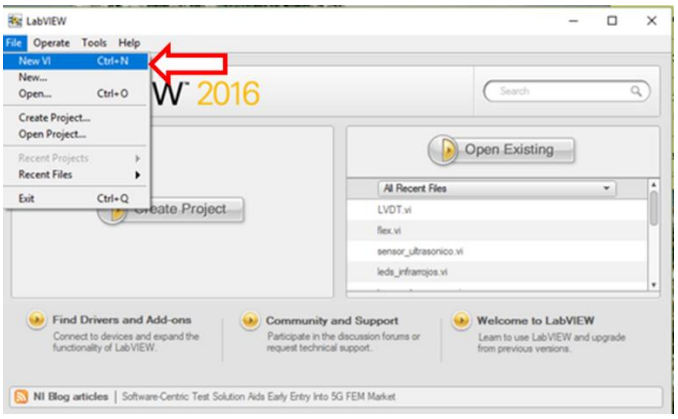


Figura 1. Creación de una aplicación.

Una vez dentro, aparecen dos ventanas: una es el panel frontal y la otra es el diagrama de bloques. El panel frontal es donde se diseña la interfaz gráfica, allí se encuentran los indicadores y controladores. El diagrama de bloques es donde se desarrolla toda la programación. En caso de que no aparezca la ventana de diagrama de bloques, ubicarse en el panel frontal y seleccionar *Windows → Show Block Diagram*. Las ventanas se pueden ajustar de forma manual o allí mismo en *Windows* hay opciones para acomodar las ventanas al gusto del usuario.

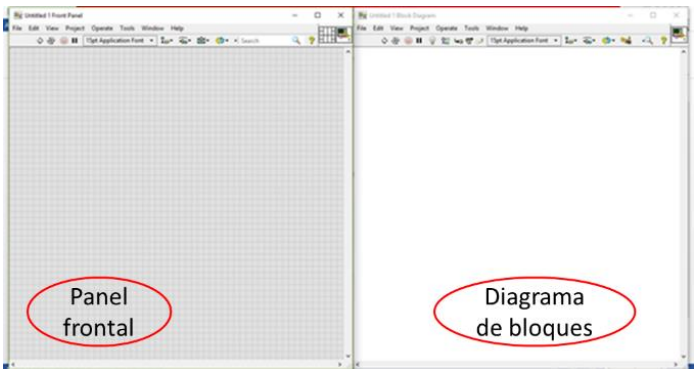


Figura 2. Panel frontal y Diagrama de bloques.

Al hacer clic derecho con el mouse dentro del panel frontal se muestra una nueva ventana donde se encuentran todas las formas, controladores e indicadores para diseñar la interfaz gráfica. Existen varios diseños para los elementos*: Modern, Silver, Classic* y *Express*.

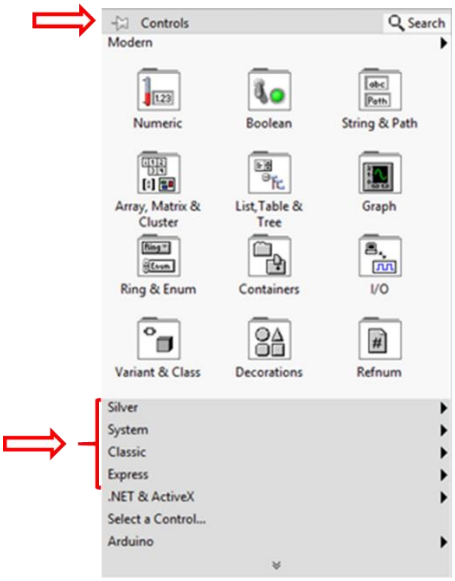


Figura 3. Elementos dentro del panel frontal.

Al hacer clic derecho dentro de la ventana del diagrama de bloques se mostrará una nueva ventana, dentro de la cual se ubican todas las funciones para realizar los programas.

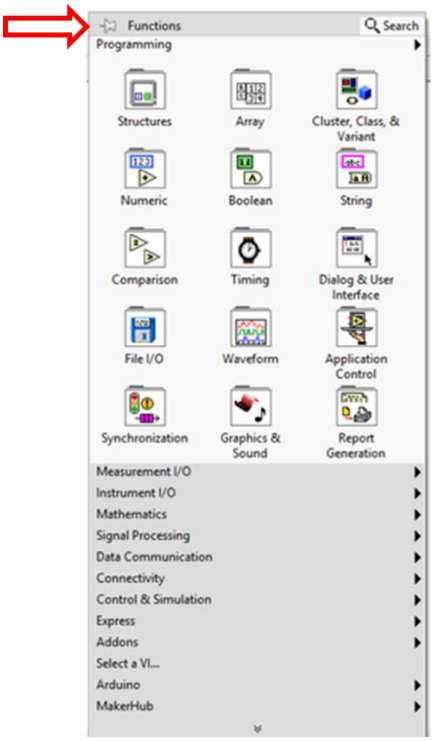


Figura 4. Funciones dentro de la ventana de Diagrama de Bloques.

En la esquina superior izquierda se muestran unos botones, los cuales sirvan para correr el programa una vez, correrlo cíclicamente, detenerlo y para pausarlo. Se recomienda no usar el botón para correr el programa de forma continua y en su lugar utilizar un ciclo *While* dentro del diagrama de bloques.

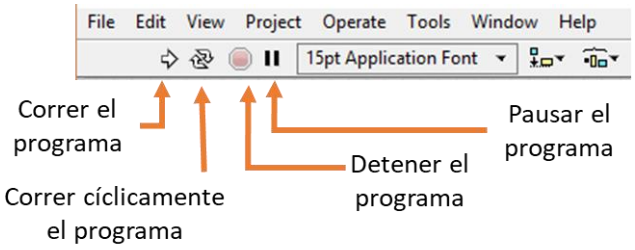


Figura 5. Opciones para correr el programa.

En caso que exista algún error en el programa, LabVIEW no permitirá correr el programa reemplazando la flecha por una flecha rota.

Figura 6. Programa de LabVIEW con algún error.

Al hacer clic sobre la flecha rota aparece un cuadro, el cual señala los errores.

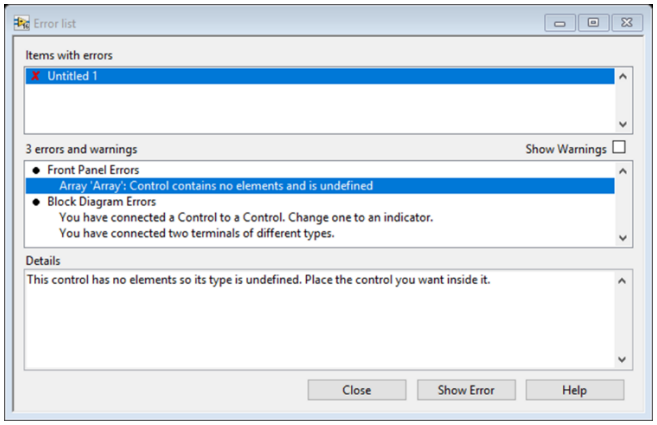


Figura 7. Cuadro de errores.

En el panel frontal y diagrama de bloques hay opciones para acomodar los componentes y verlos ordenados. En el panel frontal, muchas veces es necesario mover un elemento enfrente de otro. En el diagrama de bloques hay una opción llamada *Clean Up Diagram*, la cual ordena el diagrama de forma automática.

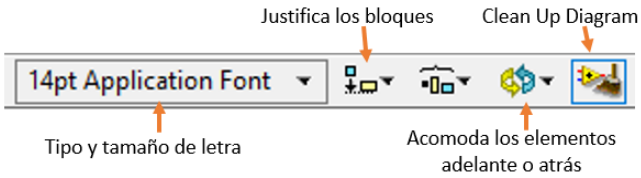


Figura 8. Opciones para ordenar los componentes.

Una forma rápida de copiar elementos es ubicar el apuntador sobre uno, presionar *Ctrl*, mantener presionada la tecla y mover el apuntador hasta la ubicación deseada (clic y arrastre).

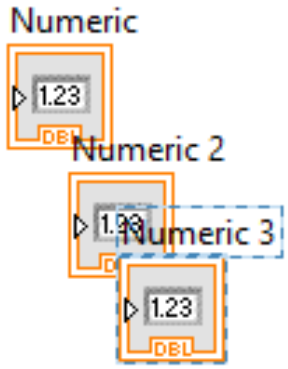


Figura 9. Copiado rápido.

Para eliminar todos los alambres sin conexión, basta con presionar Ctrl + B.

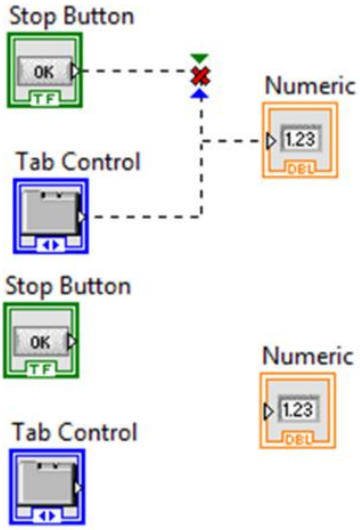


Figura 10. Eliminación rápida de alambres sin conexión.

Normalmente al usar el apuntador, este se adapta a las acciones que se desean realizar; sin embargo, si se quiere cambiar la acción a realizar, el usuario debe ubicarse en el panel frontal y entrar a *View* → *Tools Palette*.

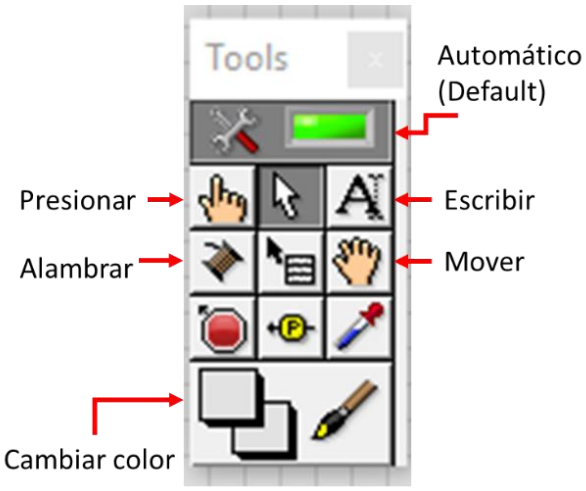


Figura 11. Paleta de herramientas.

II. CONTENEDORES, CONTROLES E INDICADORES

En varias prácticas del manual se muestran ejemplos de interfaces utilizando contenedores con pestañas, este se llama *Tab Control* y se ubica en panel frontal dentro de *Controls* → *Modern* → *Containers.*

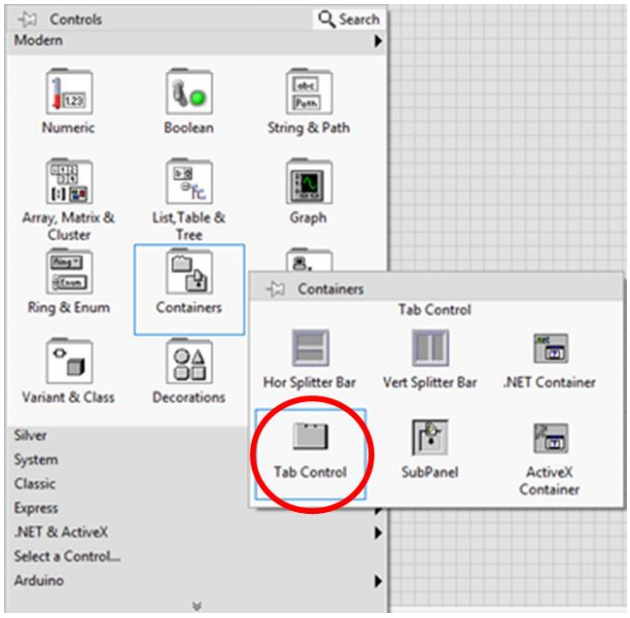


Figura 12. Ubicación del contenedor con pestañas.

Existen varios tipos de controles e indicadores numéricos, por ejemplo: *Numeric Control*, *Numeric Indicator*, *Vertical Pointer Slide*, *Knob*, *Dial*, *Tank*, *Meter*, *Gauge*, *étcetera.*

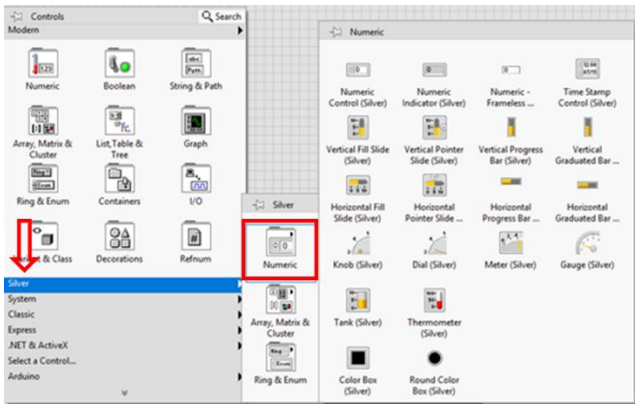


Figura 13. Ubicación de indicadores numéricos con diseño Silver.

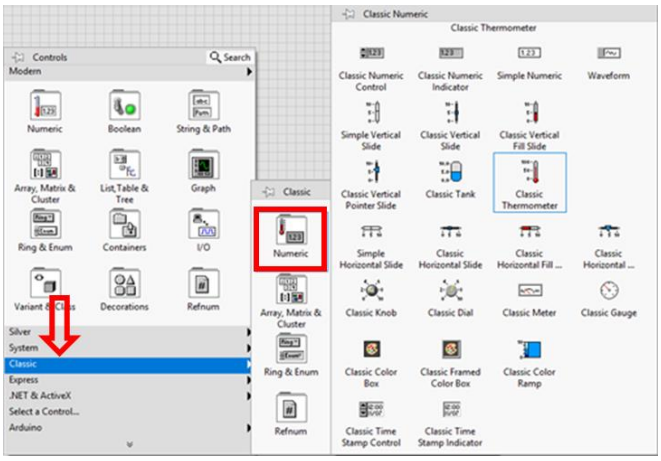


Figura 14. Ubicación de indicadores numéricos con diseño Classic.

También es posible graficar los datos. En algunas prácticas se utiliza el bloque *Waveform Chart*.

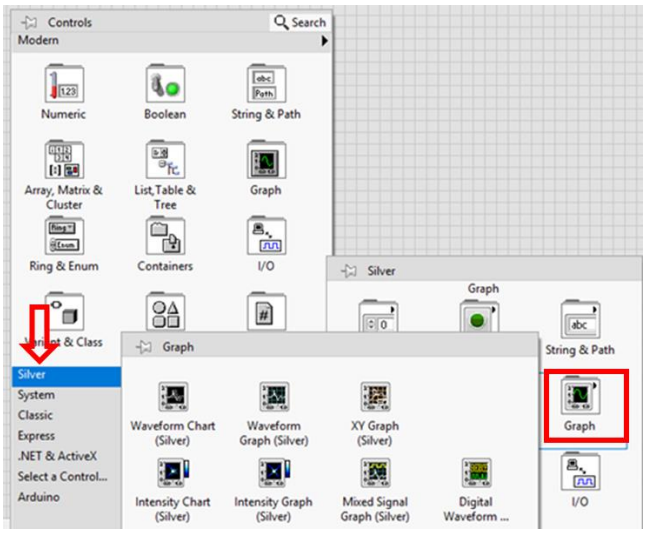


Figura 15. Ubicación de las gráficas con diseño Silver.

Además hay indicadores lógicos, por ejemplo: *Push button y LEDS.*

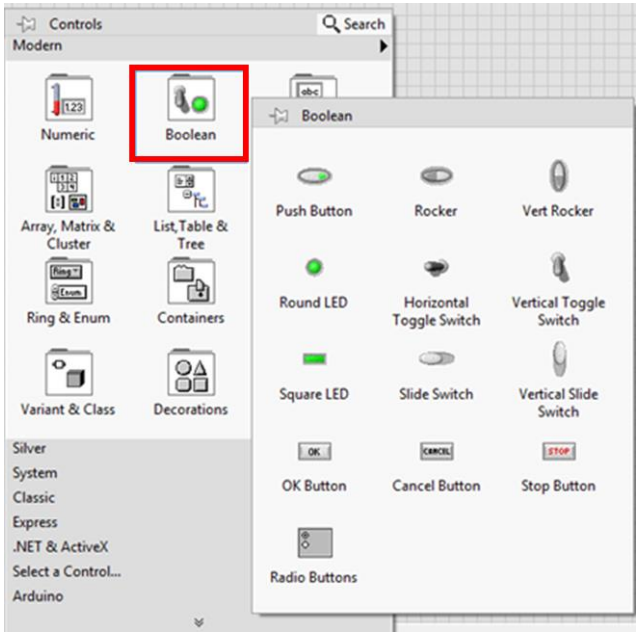


Figura 16. Ubicación de los indicadores lógicos con diseño Modern.

III. FUNCIONES DENTRO DEL DIAGRAMA DE BLOQUES

A continuación se muestran las estructuras, allí es donde se encuentran los bloques: *While Loop*, *Flat secuence*, *Case Structure* y los bloques para generar variables locales y globales.

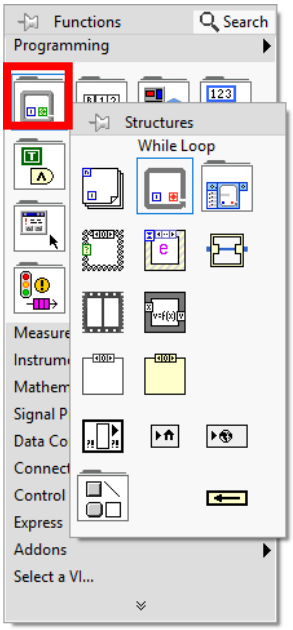


Figura 17. Estructuras.

Dentro de *Programming* → *String*, se encuentran todas las operaciones con datos tipo *String;* allí se encuentra el bloque *Format Into String*, el cual se ocupa para construir el texto que posteriormente se guarda dentro de un documento del bloc de notas.

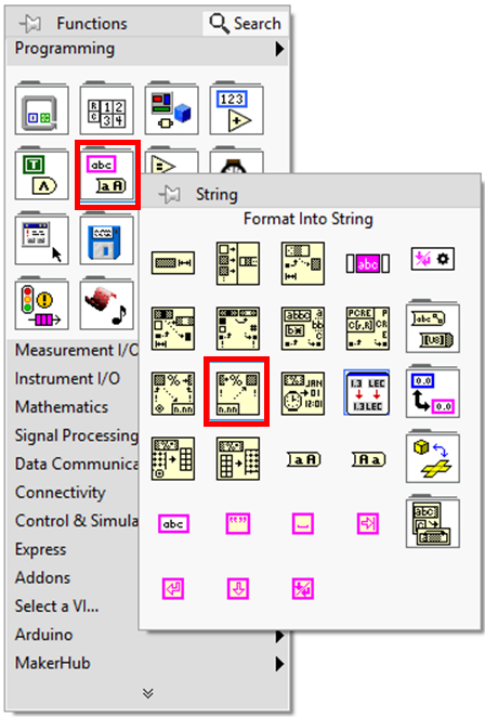


Figura 18. Bloques dentro de Programming → String.

Cuando LabVIEW lee información del Arduino a través del puerto serie, la información es de tipo *String*, por lo que es necesario cambiar el tipo de dato para poder aplicar alguna operación matemática. Los bloques para cambiar el tipo de dato se encuentran en: *Programming → String → Number/String Conversion*.

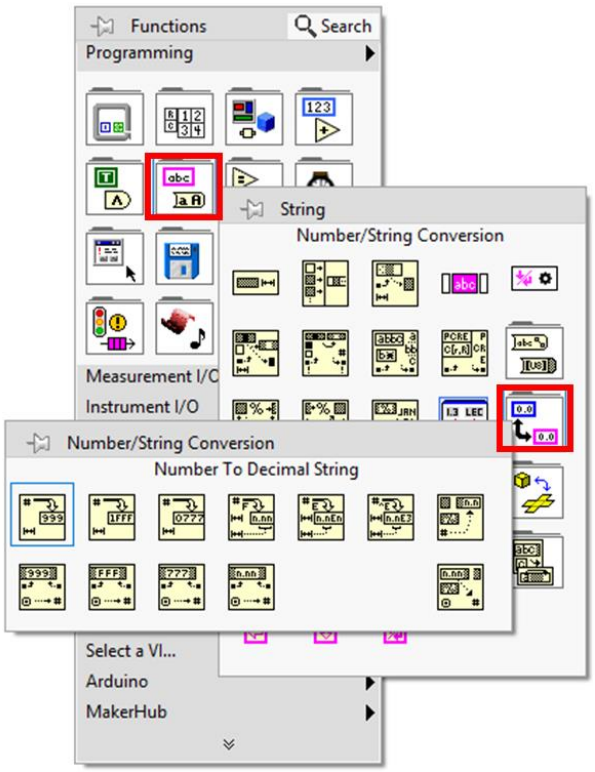


Figura 19. Bloques para cambiar el tipo de dato.

Dentro de: *Programmng → Timing* se encuentran los bloques para generar retardos de tiempo; estos sirven para ajustar el tiempo de muestreo de cada programa. Además allí se encuentra el bloque *Get Date/Time String*, el cual sirve para insertar la fecha dentro de los documentos de texto cuando guardamos la información.

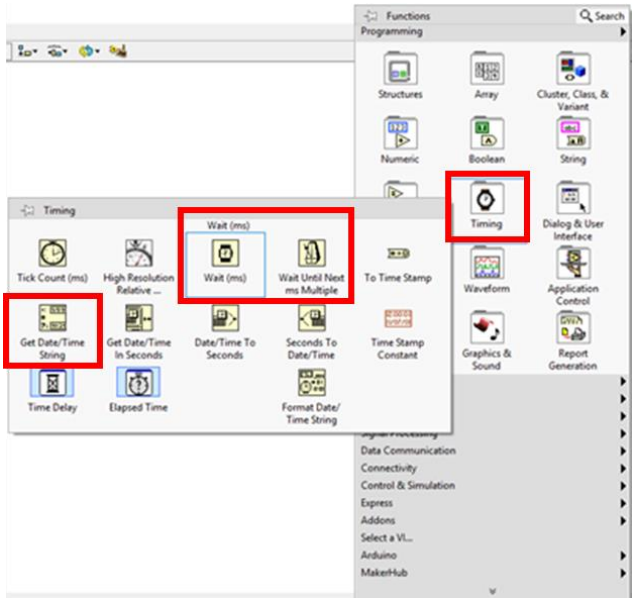


Figura 20. Bloques dentro de Programming → Timing.

Adentrandose en: *Programming → Comparation* se encuentran los bloques que permiten comparar algún dato con otro. Allí se encuentra el bloque *Select*, el cual sirve como *If*, sin tener que recurrir a alguna estructura.

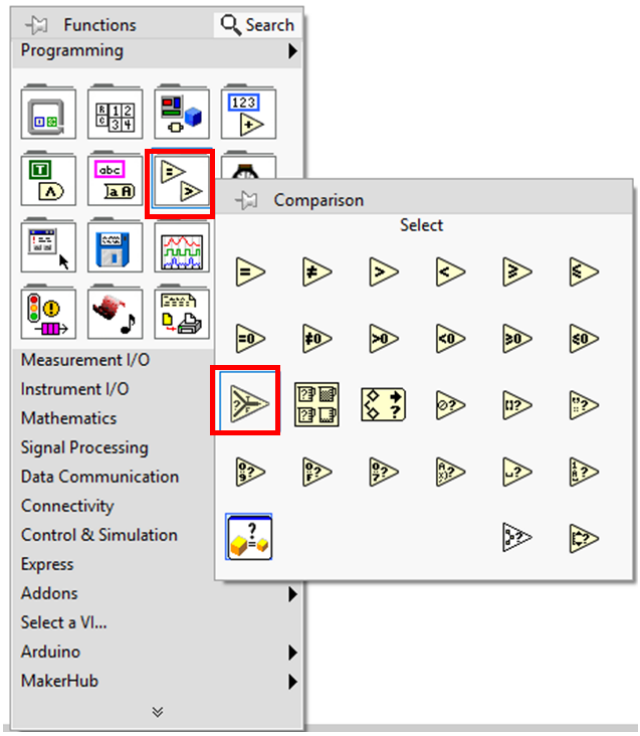


Figura 21. Bloques dentro de Programming → Comparation.

A veces se desconoce la ubicación de algún bloque, por ejemplo: si no sabe la ubicacción del bloque *Boolean Crossing*, se escribe su nombre dentro del cuadro de busqueda, alli aparecen varias opciones.

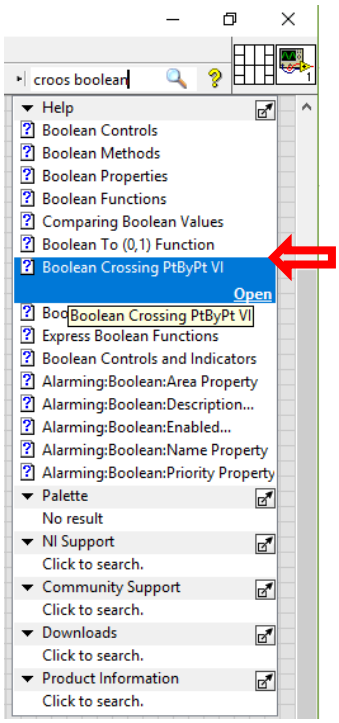


Figura 22. Cuadro de búsqueda.

Al encontrar el bloque que se buscaba, aparece un cuadro donde se da información sobre cómo utilizarlo, ubicación y ejemplos. También se puede llegar a esa ventana haciendo clic derecho sobre algún bloque y seleccionar la opción *Help*.

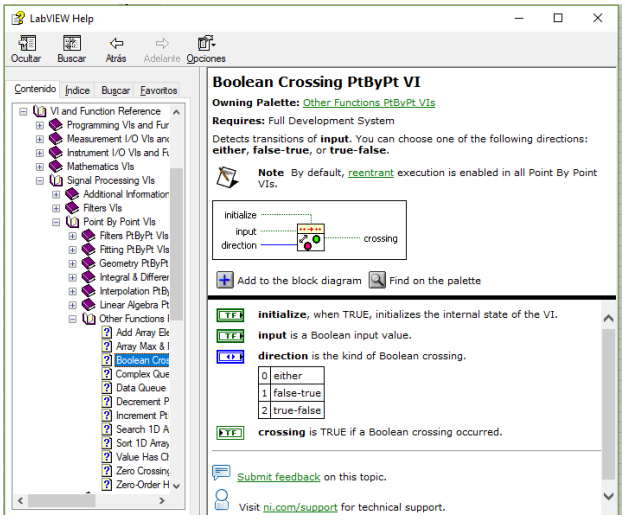


Figura 23. Cuadro de ayuda.

IV. VARIABLES LOCALES

Para crear una variable local es necesario primero haber creado un indicador con el nombre de la variable local, posteriormente

hacer clic derecho y seleccionar la opción *Create → Local Variable.*

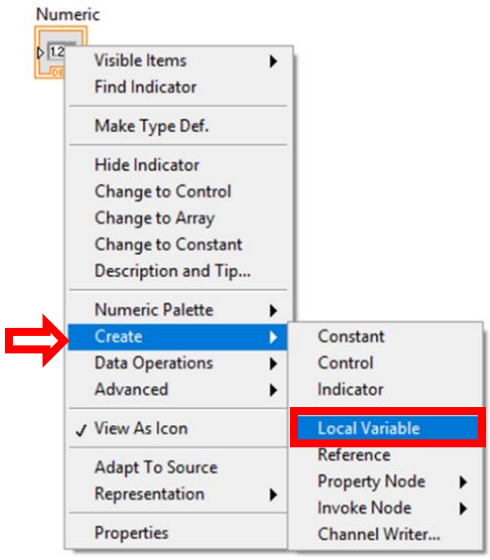


Figura 24. Creación de una variable local.

Una vez creada la variable local, se pueden modificar sus propiedades, ya sea para cambiar de tipo escritura a lectura, cambiar de variable local, ocultarla, etcétera.

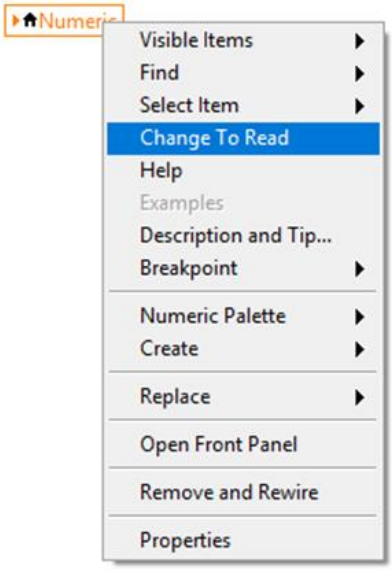


Figura 25. Opciones para modificar la variable local.

V. COMUNICACIÓN SERIE

En la mayoría de las prácticas se utiliza la comunicación serie para que haya un intercambio de datos entre el Arduino y LabVIEW. Los bloques para la comunicación serie se ubican en *Functions → Instrument I/O*.

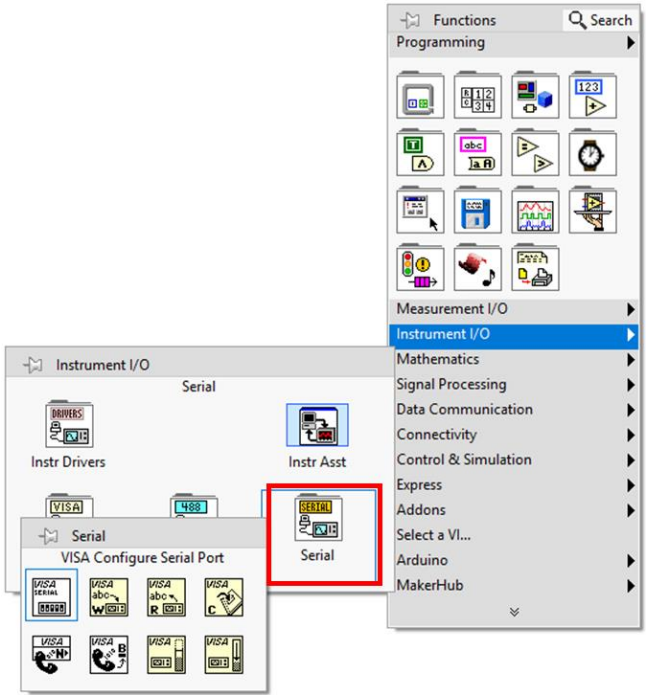


Figura 26. Ubicación de los bloques para la comunicación serie.

Para leer datos a través del puerto serie se utiliza el diagrama de bloques mostrado a continuación.

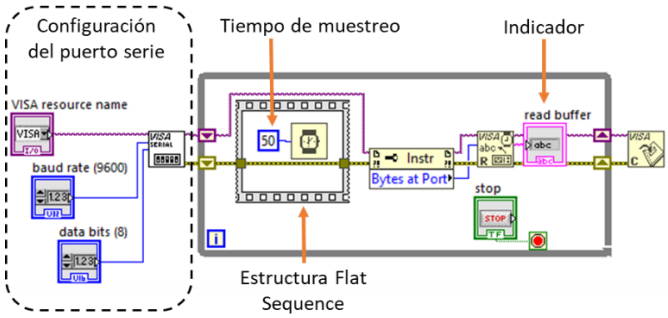


Figura 27. Diagrama de bloques para la lectura de datos a través del puerto serie.

Los datos recibidos del Arduino se muestran en *read buffer*. Los bloques para la comunicación serie son: *VISA Configure Serial Port*, *VISA Read*, *VISA Close* y *VISA Bytes at Serial Port*, estos se encuentran en*: Instrument I/O → Serial*.

En ocasiones es necesario generar alguna realimentación en las conexiones, al hacerlo en automático aparece el bloque *Feedback Node*, el cual indica que está sucediendo una realimentación.

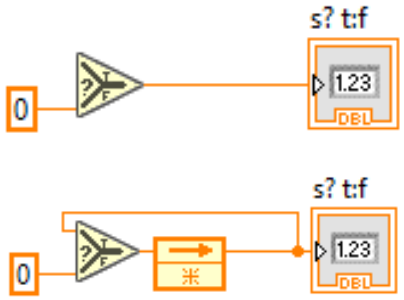


Figura 28. Realimentación.

Varios programas cuentan con unos bloques que eliminan el parpadeo de la entrada, El parpadeo es un efecto resultado de una sincronización con fallas entre la tarjeta de adquisición de datos y la interfaz gráfica que provoca que la visualización de algún dato aparezca y desaparezca de manera alternada generando dificultad para ver la información de manera continua.

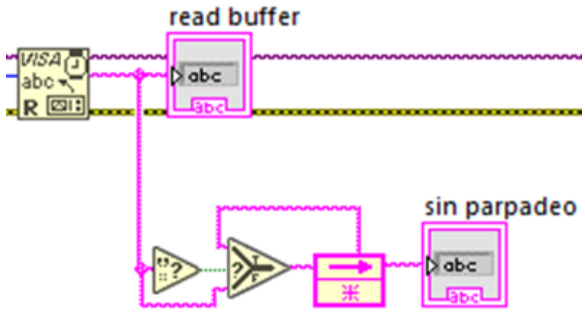


Figura 29. Bloques para eliminar el parpadeo.

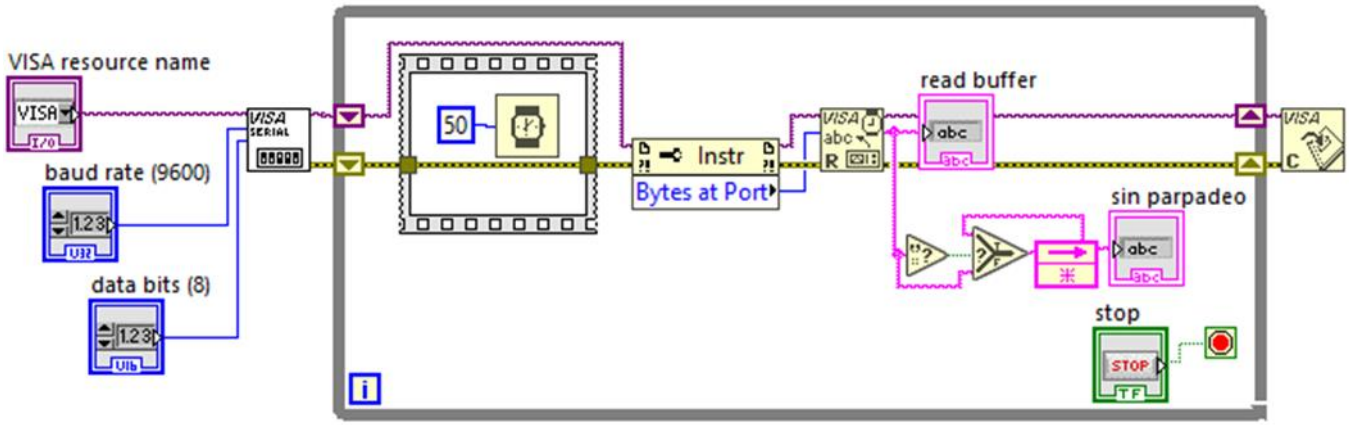


Figura 30. Diagrama de bloques para leer datos a través del puerto serie con bloques para eliminar el parpadeo.

Para leer y escribir a través del puerto serie de agrega el bloque *VISA Write*, ubicado en *Instrument I/O* → *Serial*.

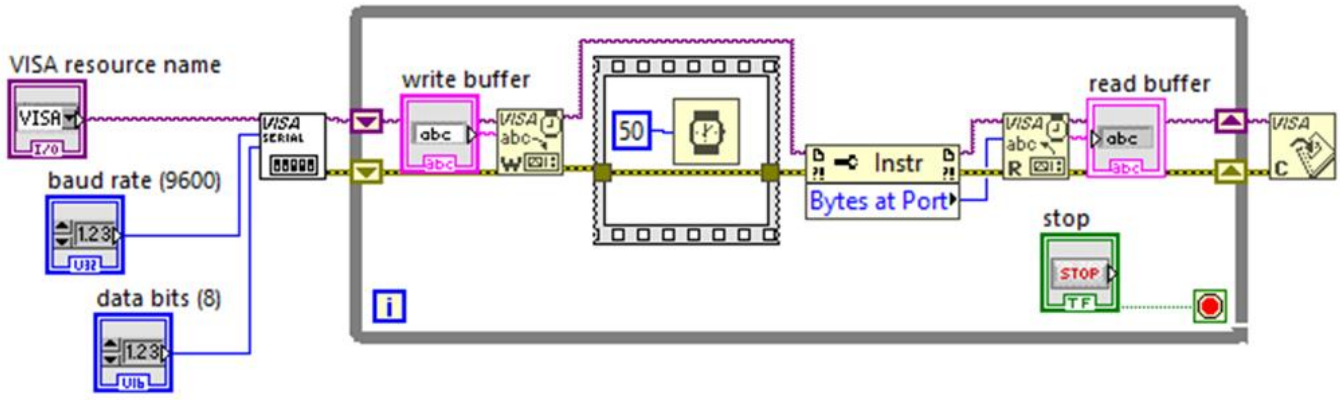


Figura 31. Diagrama de bloques para escribir y leer a través del puerto serie

VI. ALMACENAMIENTO DE DATOS DENTRO DE UN DOCUMENTO DE TEXTO

Los bloques para guardar los datos son: *Open/Create/Replace File*, *Close File* y *Write Text File*, estos se encuentran en *Programming* → *File I/O*. Se utiliza el bloque: *Format Into String* para acomodar la información de manera ordenada, este se encuentra *en Programming* → *String*.

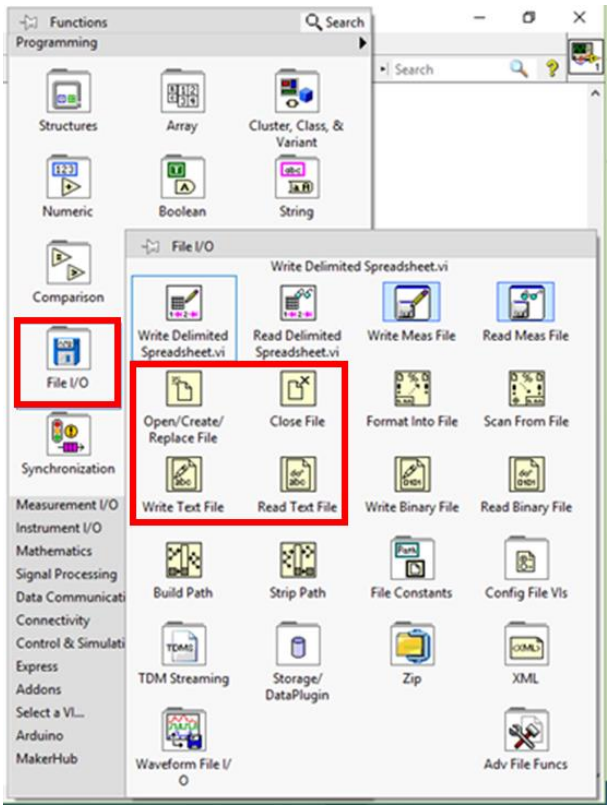


Figura 32. Ubicación de los bloques para guardar datos dentro de un documento de texto.

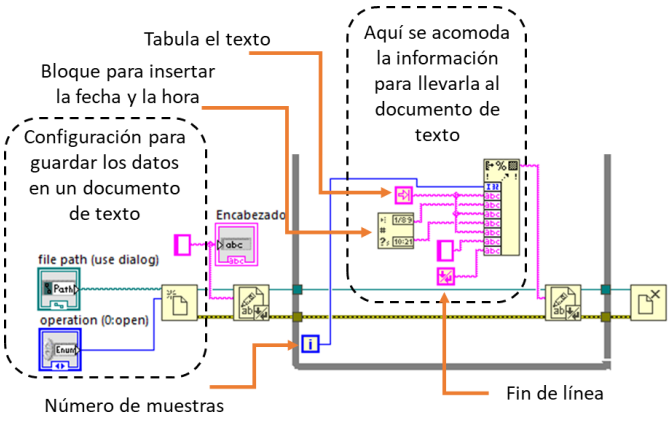


Figura 33. Bloques para guardar la información en un documento de texto.

Para guardar los datos, es necesario crear previamente un documento de texto. Se recomienda usar el bloc de notas.

Práctica 1. Instalación de software

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En esta práctica se describe el procedimiento y requerimientos para la instalación del software necesario para realizar las prácticas de este manual: LabVIEW, IDE de Arduino y los controladores necesarios.**

I. INTRODUCCIÓN

A. LabVIEW

LabVIEW (acrónimo de *Laboratory Virtual Instrument Engineering Workbench*) es una plataforma y entorno de desarrollo para diseñar sistemas, usando un lenguaje de programación gráfico. Este lenguaje de programación es recomendado para sistemas de hardware y software de pruebas, control y diseño, simulado o real y embebido, pues acelera la productividad. El lenguaje que usa se llama lenguaje G, donde la G simboliza que es un lenguaje Gráfico.

Este software fue creado por National Instruments para funcionar sobre máquinas MAC, salió al mercado por primera vez en 1986. Ahora está disponible para las plataformas Windows, UNIX, MAC y GNU/Linux.

Los programas desarrollados con LabVIEW se llaman Instrumentos Virtuales o VIs, cuyo origen provenía del control de instrumentos, aunque hoy en día se ha expandido ampliamente no sólo a la Instrumentación electrónica, sino también a la programación embebida, comunicaciones, matemáticas, etcétera. Un lema tradicional de LabVIEW es: "La potencia está en el software", que con la aparición de los sistemas multinúcleo se ha hecho aún más potente. Entre sus objetivos están el reducir el tiempo de desarrollo de aplicaciones de todo tipo (no sólo en ámbitos de pruebas, control y diseño) y el permitir la entrada a la informática a profesionales de cualquier otro campo. LabVIEW consigue combinarse con todo tipo de software y hardware, tanto del propio fabricante (tarjetas de adquisición de datos, visión, instrumentos y otro hardware), como de otros fabricantes [1].

B. Arduino

Arduino (anteriormente conocido como **Genuino** a nivel internacional hasta octubre 2016) es una compañía *open source* y *open hardware*, así como un proyecto y comunidad internacional que diseña y manufactura placas de desarrollo de hardware para construir dispositivos digitales y dispositivos interactivos que puedan recibir datos de sensores y controlar objetos del mundo real.

Open Source o código abierto, es la expresión con la que se conoce al software distribuido y desarrollado libremente [2]. Se califica como *open source*, por lo tanto, a los programas informáticos que permiten el acceso a su código de programación, lo que facilita modificaciones por parte de otros programadores ajenos a los creadores originales del software en cuestión [3]. *Open hardware* hace referencia a las especificaciones de diseño de un objeto físico que están autorizadas de tal manera que cualquier persona puede estudiar, modificar, crear y distribuir dicho objeto [4].

Arduino se enfoca en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos en proyectos multidisciplinarios. Los productos que vende la compañía son distribuidos como hardware y software libre, bajo la Licencia Pública General Reducida de GNU (LGPL) o la Licencia Pública General de GNU (GPL), permitiendo la manufactura de las placas Arduino y distribución del software por cualquier individuo. Las tarjetas Arduino están disponibles comercialmente en forma de placas ensambladas o también en forma de kits *hazlo tú mismo* (DIY, por sus siglas en inglés de *"Do It Yourself"*).

Los diseños de las placas Arduino usan diversos microcontroladores y microprocesadores. Generalmente el hardware lo integra un microcontrolador Atmel AVR conectado bajo la configuración de "sistema mínimo" sobre una placa de circuito impreso a la que se le pueden conectar tarjetas de expansión (*shields*) a través de la disposición de los puertos de entrada y salida presentes en la placa seleccionada.

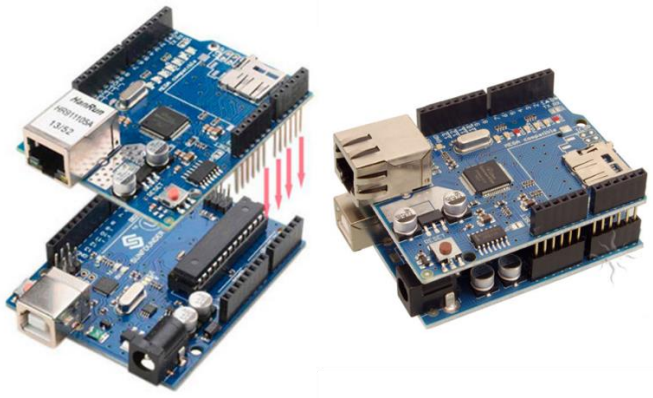


Figura 1. Shield ethernet montada sobre una tarjeta Arduino.

Las shields complementan la funcionalidad del modelo de placa empleada, agregando circuitería, sensores y módulos de

comunicación externos a la placa original. La mayoría de las tarjetas Arduino pueden ser energizadas a través de un puerto USB o un jack de 2.5 mm. La mayoría de las tarjetas Arduino pueden ser programadas a través del puerto serial que incorporan haciendo uso del *bootloader* (gestor de arranque) que traen programado por defecto. Las tarjetas Arduino se programan mediante una computadora, usando comunicación serie [5].

C. ¿Por qué Arduino y LabVIEW?

Frecuentemente es necesario realizar mediciones para las cuales hoy en día es indispensable ser auxiliado de una computadora, esto implica programar la computadora para darle las instrucciones de cada cuanto tiempo se quiere que realice una medición (tiempo de muestreo) entre otras características que se desean en las mediciones. También se necesita darle las instrucciones para que se comunique con algún dispositivo (tarjeta de adquisición de datos) donde se tenga conectados los sensores y/o actuadores.

En este manual de prácticas se utiliza básicamente el siguiente software: IDE de Arduino y LabVIEW.

Se puede utilizar la tarjeta Arduino como tarjeta de adquisición de datos en LabVIEW pero no siempre es así, depende de las necesidades del proyecto.

Algunas veces basta con trabajar únicamente con una tarjeta Arduino y un programa instalado en la misma tarjeta junto con algunos sensores y/o actuadores, sin la necesidad de una interfaz de usuario en una computadora.

Para otras aplicaciones es indispensable tener una interfaz de usuario en una computadora para tener la opción de visualizar algunas variables en un monitor y al mismo tiempo controlar actuadores desde el mismo panel y es aquí donde la tarjeta Arduino hace las veces de tarjeta de adquisición de datos (DAQ).

II. OBJETIVOS

- ✓ Instalar el lenguaje de programación gráfico LabVIEW versión 2012 o superior.
- ✓ Descargar e instalar el IDE de Arduino.
- ✓ Descargar e instalar las paqueterías NI VISA, JKI, así como el toolkit de Arduino y LINX.
- ✓ Instalar el controlador para el Arduino.

III. MATERIALES Y SOFTWARE

- ➢ LabVIEW versión 2012 o superior.
- ➢ IDE de Arduino.
- ➢ Drivers NI VISA.
- ➢ VIPM (VI Package Manager).
- ➢ Tarjeta Arduino UNO.
- ➢ Computadora con Windows 7 o superior.
- ➢ LabVIEW Interfaz for Arduino (LIFA_Base).

IV. DESARROLLO

A. Instalación de LabVIEW

Paso 1:

El primer software a instalar en la computadora es LabVIEW, se recomienda instalar la versión 2012 o superior. Se puede instalar una versión de evaluación de la página de National Instruments, únicamente se tiene que registrar el correo electrónico y listo. Ver figura 1 [6].

http://www.ni.com/es-mx/shop/labview/download.html

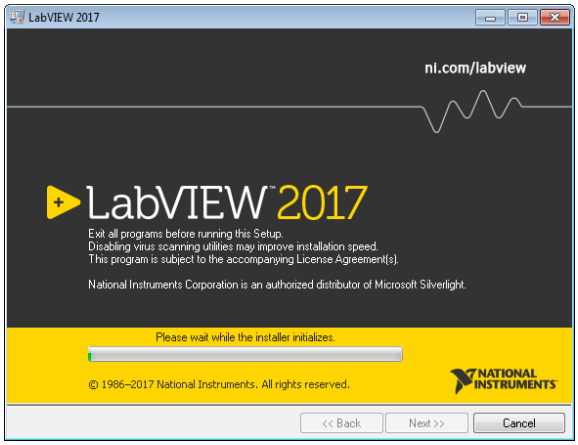


Figura 2. Instalación de LabVIEW.

Paso 2:

El siguiente paso es instalar el controlador NI-VISA, para lo cual se debe tomar en cuenta la compatibilidad de versiones entre el NI-VISA y la versión de LabVIEW que se esté usando. Ver figura 2 [7].

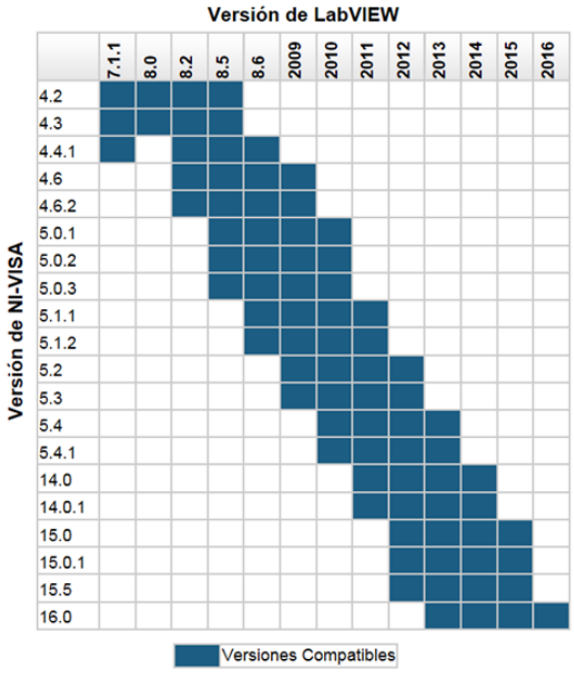


Figura 3. Tabla de compatibilidad entre NI-VISA y LabVIEW.

También se tiene que cuidar que la versión seleccionada del NI-VISA sea compatible con la versión del sistema operativo, para este caso Windows.

La versión de NI-VISA 17 se puede descargar del siguiente enlace:

http://www.ni.com/download/ni-visa-17.0/6646/en/

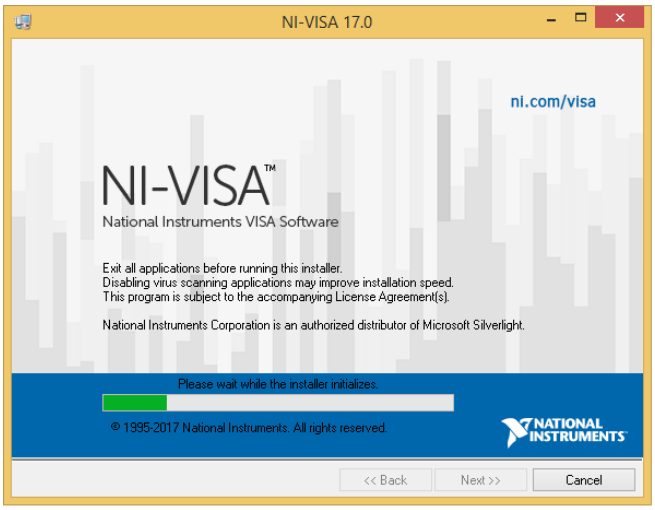


Figura 4. Instalación de NI-VISA 17.

B. Instalación del VIPM

Después se instala el gestor de paquetes VI (VIPM). Se puede descargar la versión de la comunidad (free) de la siguiente página:

- https://vipm.jki.net/get
- http://www.ni.com/tutorial/12397/en/

Este programa es el que descarga e instala los VI de Arduino en LabVIEW. Ver figuras 4 y 5 [8].

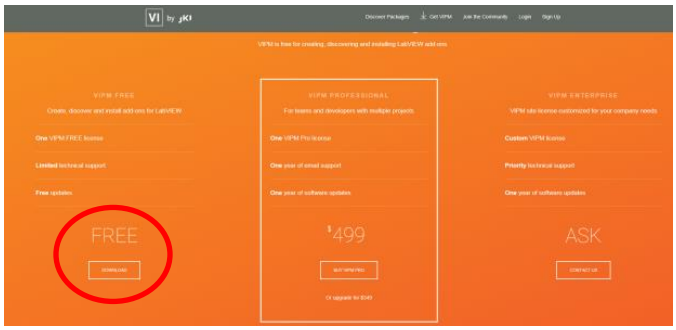


Figura 5. Página de internet para descargar el JKI VI Package Manager.

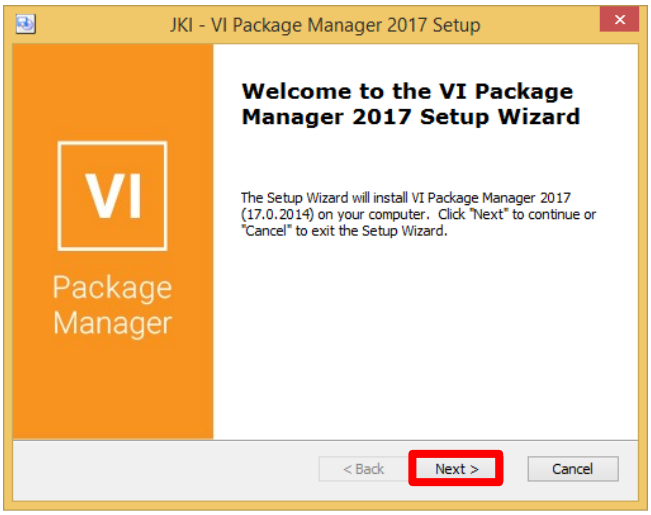


Figura 6. Instalación del JKI VI Package Manager.

C. Instalación del toolkit de Arduino

Se ejecuta el gestor de VI (VIPM) y se busca *LabVIEW Interface for Arduino*, después se selecciona la opción de *Install Package*. Ver figura 5 [9].

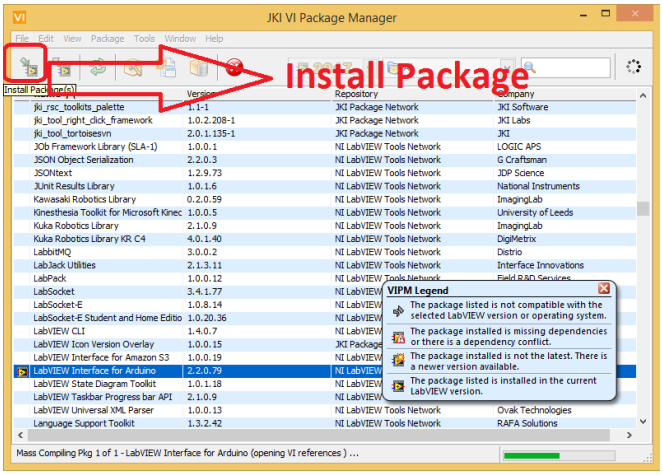


Figura 7. Instalacion del LabVIEW Interface for Arduino.

D. Instalación de LINX

Para instalar LINX, primero se ejecuta el VIPM y después se busca LINX. A continuación aparece una ventana como la que se muestra a continuación [9].

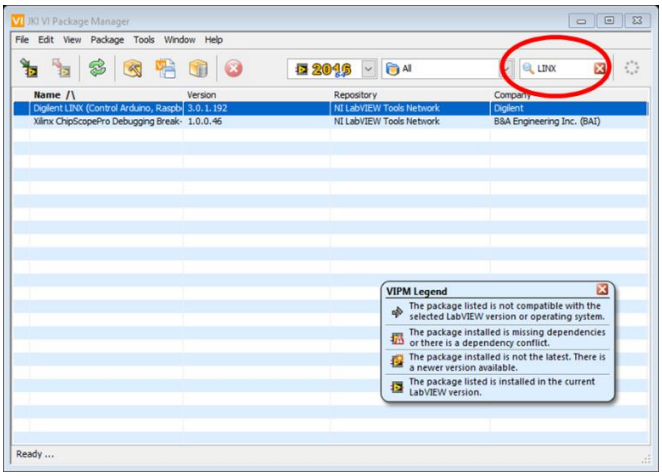


Figura 8. Búsqueda de LINX.

Se hace clic en *Digilent LINX* (Control Arduino, Raspberry Pi, BeagleBone and more) para que se muestre la siguiente ventana. Ver figura 7.

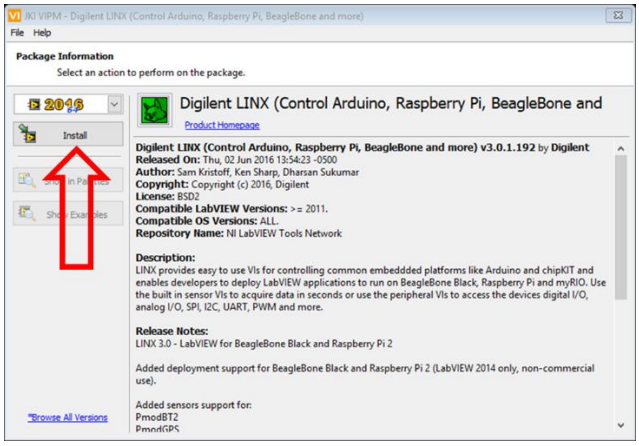


Figura 9. Ventana de instalación.

Una vez dentro se selecciona *Install*. Después se hace clic en *Continue*.

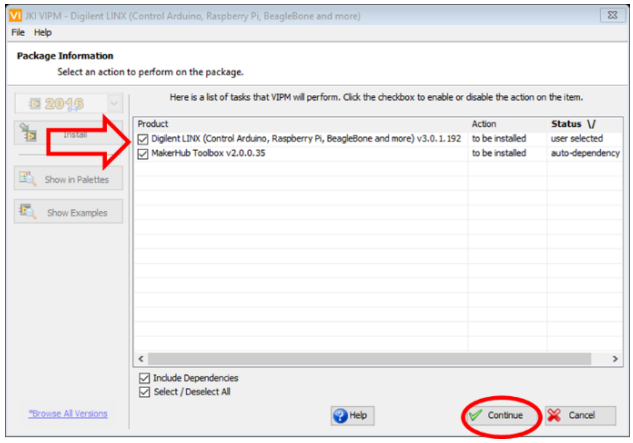


Figura 10. Instalación de los paquetes.

Se aceptan los términos.

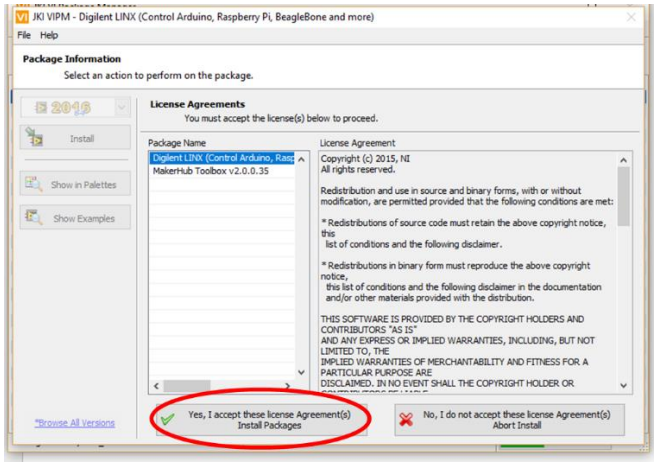


Figura 11. Aceptación de términos.

Y finalizar.

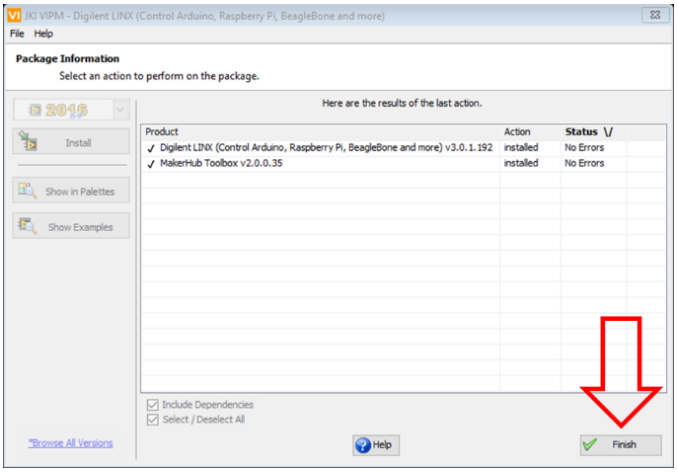


Figura 12. Fin de la instalación.

Al iniciar LabVIEW y crear un nuevo VI, en el diagrama de bloques debe aparecer una nueva opción dentro de *Functions*, llamada *MakerHub*, allí se encuentran todos los bloques relacionados con LINX.

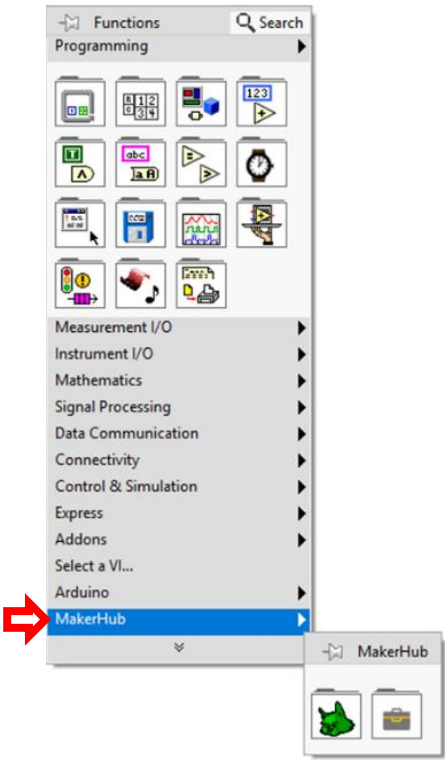


Figura 13. Paleta de funciones de LINX.

E. Instalación del IDE de Arduino

Paso 1:

El IDE de Arduino se descarga directamente de la página arduino.cc, en la opción de *Software* → *Download* seleccionar *Windows Installer* [10,11, 12].

Figura 14. Descarga del IDE de Arduino

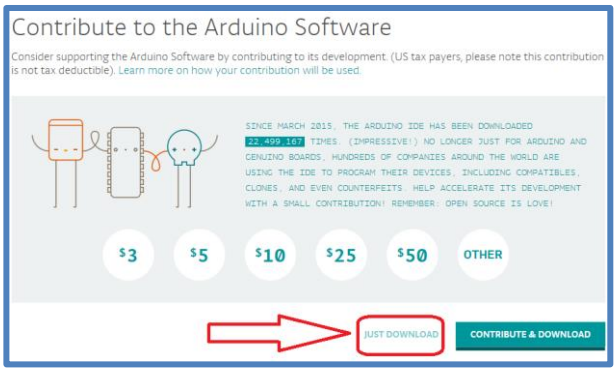


Figura 15. Selección de *Just Download* para iniciar la descarga del IDE de Arduino.

Se recomienda ejecutar la aplicación; sin embargo, también es recomendable guardar dicha carpeta en una memoria USB, ya que se necesitan los controladores si se desea utilizar el Arduino en otra computadora.

Paso 2:

Conectar la tarjeta Arduino UNO a la computadora utilizando el cable USB. Se entra en el administrador de dispositivos de Windows, si es la primera vez y aún no están instalados los controladores la computadora no reconocerá la tarjeta, como se muestra en la figura 13.

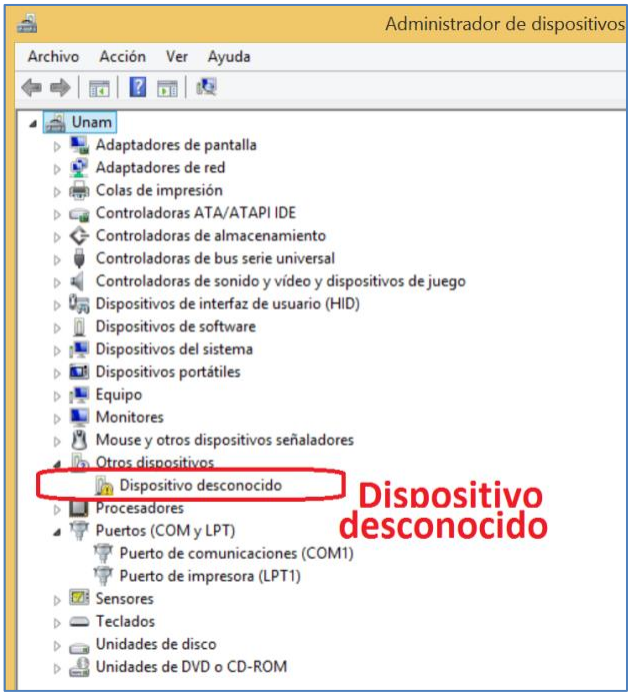


Figura 16. Administrador de dispositivos.

Algunas veces cuando se instala el IDE de arduino, en una ventana de dialogo pregunta si se desea instalar los controladores; si se selecciona la opción de instalar, se instalarán automaticamente.

Figura 17. Instalación del IDE de arduino y del controlador de la tarjeta de arduino.

Paso 3:

Si aún no están instalados los controladores y el sistema operativo no lo identifica, se hace clic derecho sobre la opción de *Dispositivo desconocido* y se selecciona la opción de *Actualizar software del controlador*, como se muestra en la figura 15.

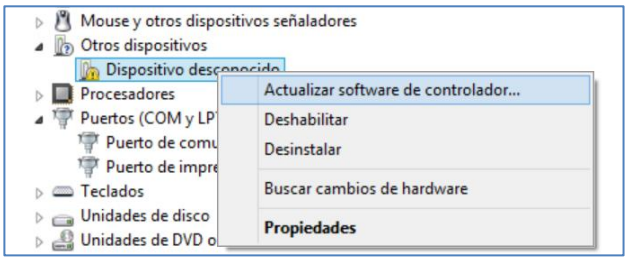


Figura 18. Actualización el software del controlador.

En la ventana que se muestra en la figura 16 se selecciona la opción de *Buscar software del controlador en el equipo*.

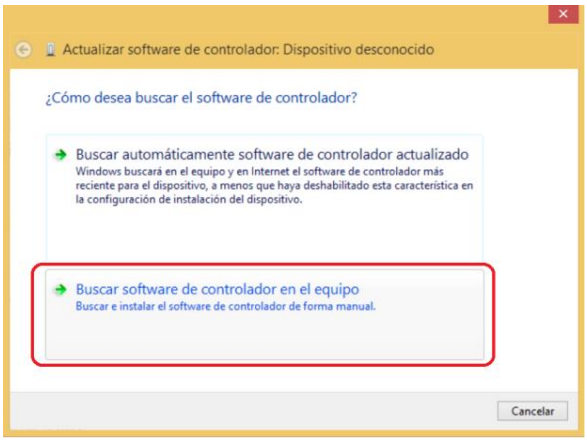


Figura 19. Búsqueda del software del controlador en el equipo.

Seleccionar la ruta donde tenemos almacenado el controlador, que normalmente está en:

C:→ArchivosdelPrograma(x86) →Arduino →Drivers. (Ver figura 17).

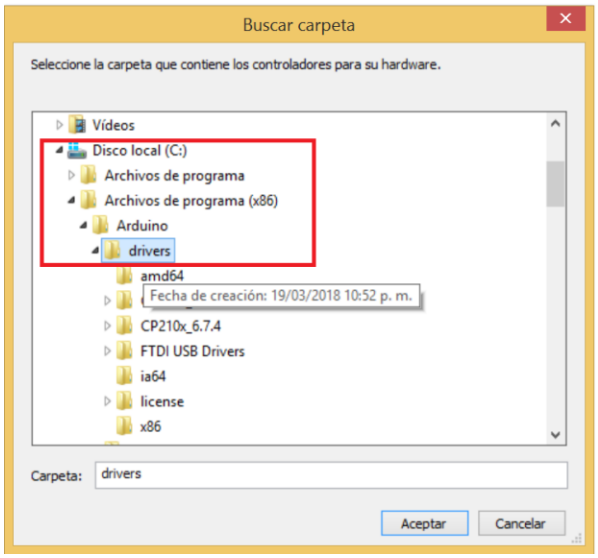


Figura 20. Carpeta de controladores del Arduino.

Finalmente se instalarán los controladores y la computadora reconocerá la tarjeta Arduino asignándole un puerto COM. Ver figura 18.

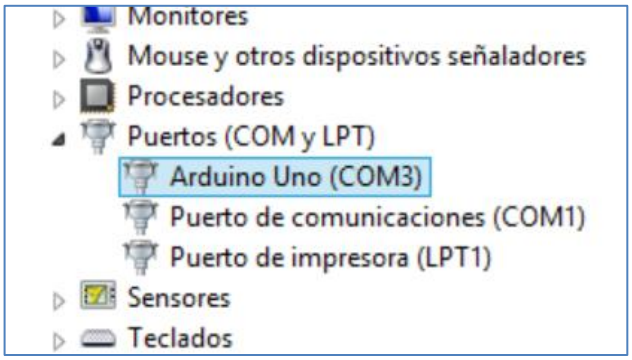


Figura 21. COM 3 asignado a nuestra tarjeta Arduino.

Paso 4:

Cuando ya todo esté listo para trabajar, se puede comprobar que al dar clic derecho en el diagrama de bloques de un nuevo VI de LabVIEW podemos visualizar la paleta del Toolkit de Arduino. Ver figura19.

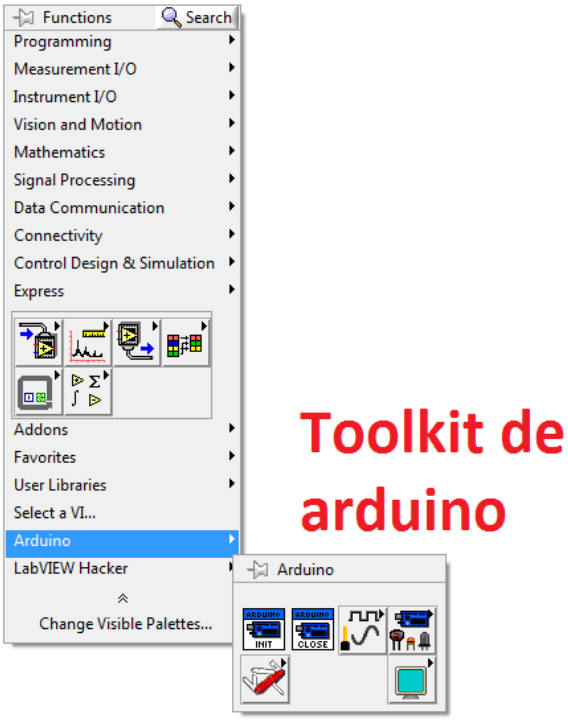


Figura 22. Toolkit de Arduino en el diagrama de bloques de un nuevo VI de LabVIEW.

Paso 5:

Siempre que se use el toolkit de Arduino se tiene que cargar el LIFA_Base a nuestro Arduino, este se encuentra en la siguiente dirección:

C:\Program Files\National Instruments\LabVIEW 2017\vi.lib\LabVIEW Interface for Arduino\Firmware\LIFA_Base.

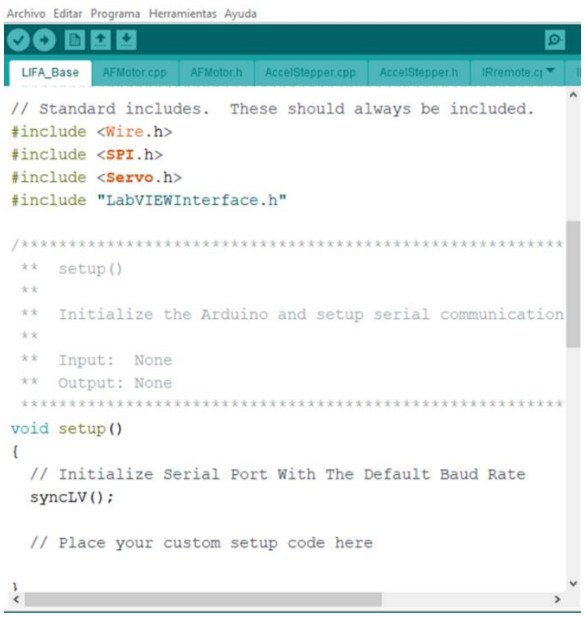


Figura 23. LIFA_Base.

V. EJERCICIO PARA PRINCIPIANTES (opcional)

A. Objetivo

- ✓ Desarrollar un programa sencillo en LabVIEW que permita sumar y multiplicar dos números usando controladores e indicadores numéricos y un contenedor con pestañas.

B. Desarrollo

Se crea un contenedor con pestañas, se le asigna nombre a las páginas y dentro de cada una se generan los controladores numéricos y el indicador correspondiente. El alumno debe ser capaz de encontrar los componentes por sí mismo.

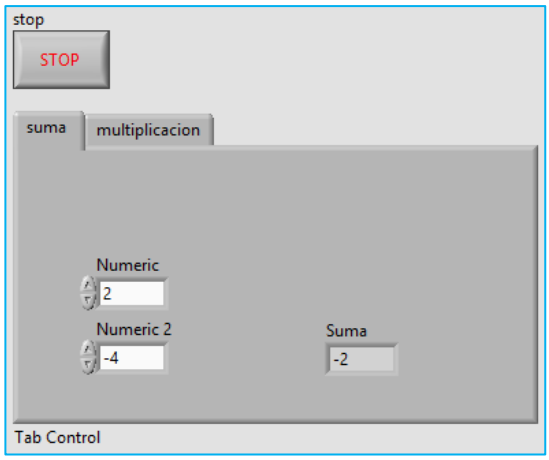


Figura 24. Primera pestaña del panel frontal.

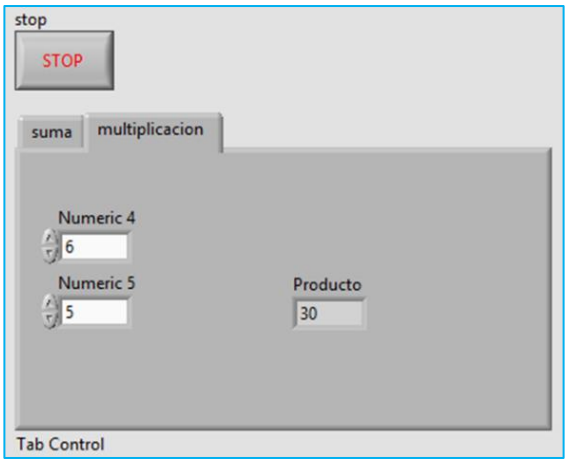


Figura 25. Segunda pestaña del panel frontal.

Se utilizan los bloques *Add* y *Multipy* ubicados en *Functions* → *Programming* → *Numeric* para realizar las operaciones.

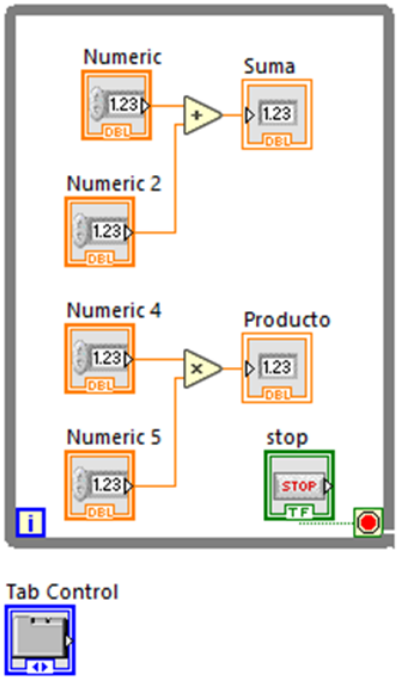


Figura 21. Diagrama de bloques.

Finalmente correr el programa y hacer pruebas con él.

VI. FUENTES DE CONSULTA

[1] Wikipedia. (2018). LabVIEW. *Wikipedia.org*. Recuperado de https://es.wikipedia.org/wiki/LabVIEW .

[2] andrearrs. (2014). Diferencias entre Software Libre y Open Source. *https://hipertextual.com* . Recuperado de https://hipertextual.com/archivo/2014/05/diferencias-software-libre-y-open-source/ .

[3] Definicion.de. (2008-2018). DEFINICIÓN DE OPEN SOURCE. *https://definicion.de* . Recuperado de https://definicion.de/open-source/ .

[4] Red Hat, Inc. (s. f.). What is open hardware?. *https://opensource.com* . Recuperado de https://opensource.com/resources/what-open-hardware .

[5] Wikipedia. (2018). Arduino. *Wikipedia.org*. Recuperado de https://es.wikipedia.org/wiki/Arduino.

[6] Bastian. (2012). Usando Arduino como DAQ en Labview. *Robotica Ludica*. Recuperado de http://roboticaludica.com/?p=763 .

[7] National Instruments Corporation. (2014). Compatibilidad de Versiones de LabVIEW y NI-VISA para Windows. *digital.ni.com*. Recuperado de http://digital.ni.com/public.nsf/allkb/595ADC11442888398625796C0064F2B9 .

[8] © 2018 JKI. Recuperado de https://vipm.jki.net/get .

[9] National Instruments Corporation. (2018). VI Package Manager. *digital.ni.com*. Recuperado de http://www.ni.com/tutorial/12397/en/ .

[10] Arduino. (2018). Home. *arduino.cc*. Recuperado de https://www.arduino.cc/ .

[11] Arduino. (2018). Software. *arduino.cc*. Recuperado de https://www.arduino.cc/en/Main/Software .

[12] Arduino. (2018). Contribute to the Arduino Software. *arduino.cc*. Recuperado de https://www.arduino.cc/en/Main/Donate .

Práctica 2. Sensor Flex

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En este trabajo se explica cómo utilizar el sensor Flex de manera práctica, fabricando un guante especial con el sensor montado sobre un dedo, cuya resistencia cambia en función de su flexión, usando la tarjeta Arduino UNO para la adquisición de datos y el entorno de desarrollo de LabVIEW para la generación de una interfaz gráfica con el fin de visualizar los datos de manera clara para el usuario utilizando la comunicación en serie para la comunicación entre el Arduino y LabVIEW.**

I. INTRODUCCIÓN

A. Características y funcionamiento del sensor

El sensor Flex es un sensor piezorresisitivo, el cual tiene la cualidad de cambiar su resistencia al ser flexionado, por eso es muy usado para hacer guantes inteligentes [1]. Debido a que prácticamente es una resistencia, sus terminales no tienen polaridad. Para su uso se requiere una resistencia para armar un divisor de voltaje.

II. OBJETIVOS

- ✓ Aprender a usar el sensor Flex de una manera práctica.
- ✓ Fabricar un guante que permita utilizar el sensor al flexionar un dedo.
- ✓ Aprender a utilizar el toolkit de Arduino y LINX para comunicar la tarjeta Arduino con LabVIEW.
- ✓ Crear una aplicación en LabVIEW que permita observar la salida del sensor. Deberá ser generada dentro de un contenedor con pestañas, una para la comunicación serie, las necesarias para mostrar los datos y una para los componentes que permitan guardar los datos.

III. MATERIAL

- 1 Sensor Flex.
- 1 Resistor de10 kΩ 1/4 W.
- 1 par de Guantes.
- Materiales para sujetar el sensor al guante.
- Tarjeta Arduino UNO.
- Computadora con el software LabVIEW e IDE de Arduino.
- Protoboard.
- Alambre telefónico para el alambrado en la protoboard.
- Material para soldar componentes electrónicos (cautín, soldadura, etcétera).

IV. DESARROLLO

A. Construcción del guante

Primero se fabrica el guante. Se deja al criterio del alumno los materiales a utilizar. Se debe flexionar el sensor de acuerdo con la figura 1 para no romperlo, flexionando solo dentro del área con líneas horizontales.

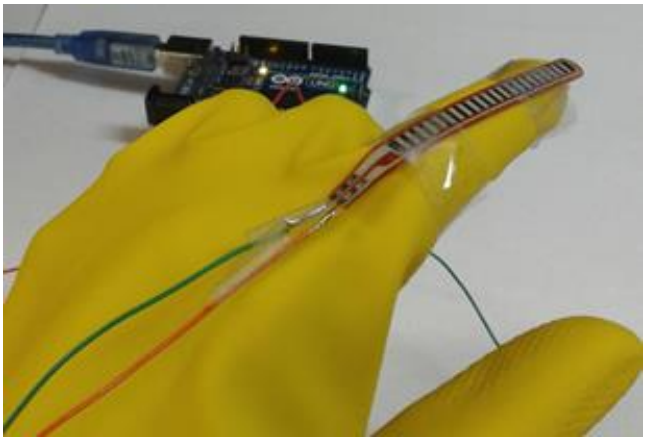


Figura 1. Armado del guante

B. Conexión

Realizamos la siguiente conexión:

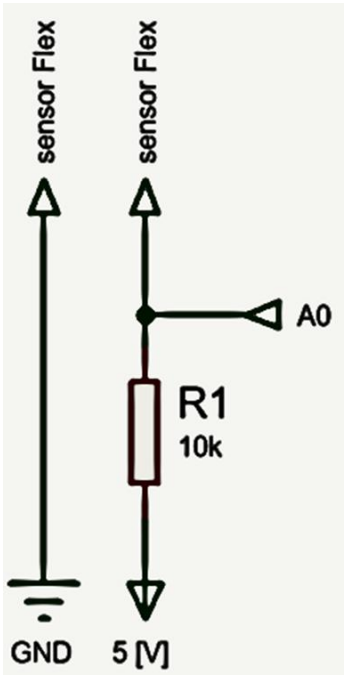


Figura 2. Diagrama esquemático.

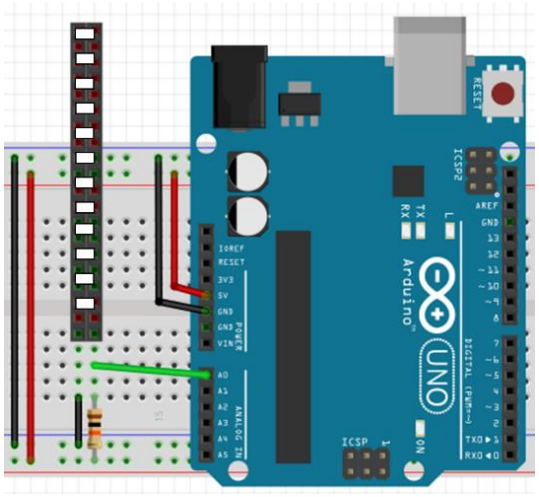


Figura 3. Conexión clásica del sensor Flex [2].

C. Código de Arduino

Se genera el código en Arduino.

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.println(analogRead(A0));
  delay(100);
}
```

Este código realiza una lectura analógica a través de A0 y manda el valor al puerto serie.

D. Programa en LabVIEW usando comunicación en serie

El programa de LabVIEW recibe los datos del Arduino a través de la comunicación en serie, estos datos se muestran en *read buffer* y posteriormente ocupa una serie de bloques para eliminar el parpadeo (ver "Introducción al entorno de desarrollo de Arduino").

El programa cuenta con unos bloques que eliminan el parpadeo de la entrada, después convierte el dato tipo *String* a fraccionario para posteriormente mostrarlo con tres indicadores*: Indicadores numérico*, *Meter* y *Waveform chart*. Cabe decir que existen varios diseños de indicadores, en este ejemplo se usan los indicadores dentro de *Silve*r, pero no importa si se usa algún otro diseño.

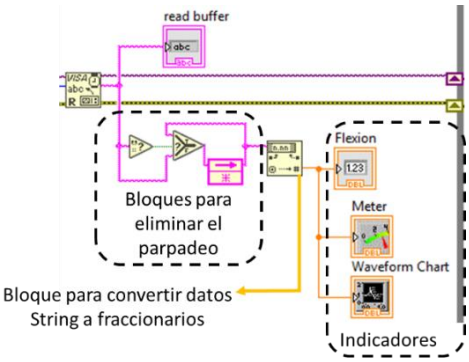


Figura 4. Eliminación del parpadeo e indicadores

E. Diseño final

A continuación se muestra el diagrama de bloques completo.

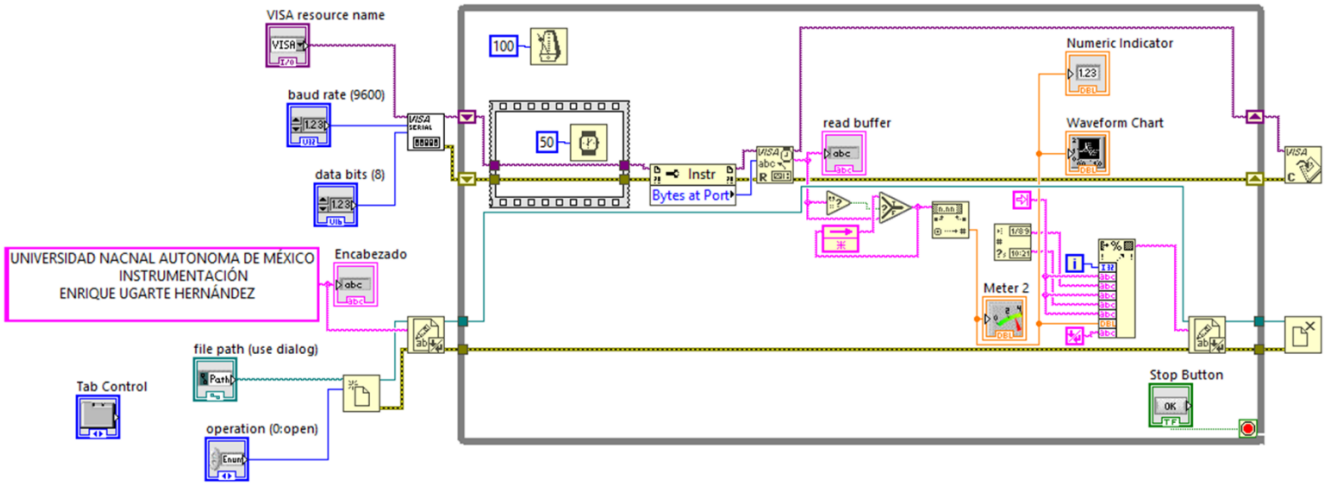


Figura 5. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación en serie se encuentran en la primera pestaña, la visualización de los datos en la segunda y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

Los datos mostrados en el *Meter* y el *Wave Chart* es el voltaje leído por el Arduino, el cual esta escalado de 0 a 1023, donde 1023 corresponde a 5 [V]. Este valor cambiará al flexionar el sensor.

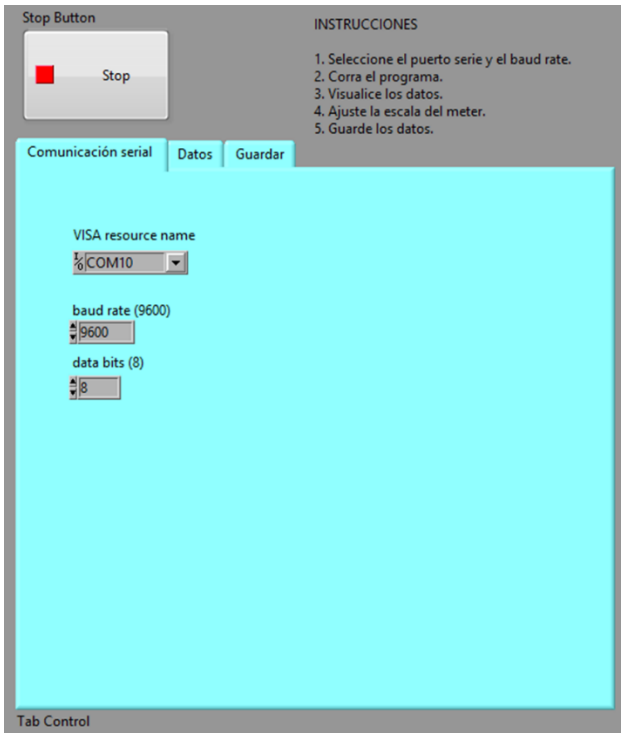


Figura 6. Primera página del contenedor con pestañas.

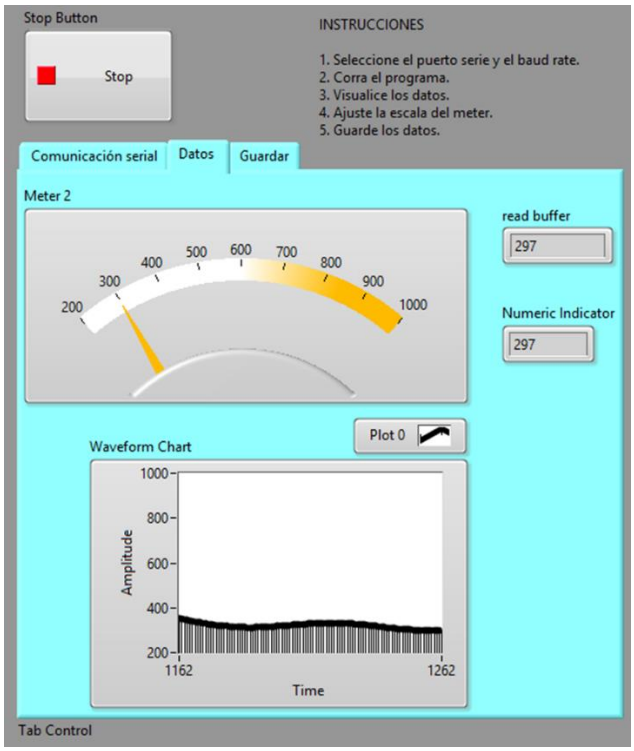


Figura 7. Segunda página del contenedor con pestañas.

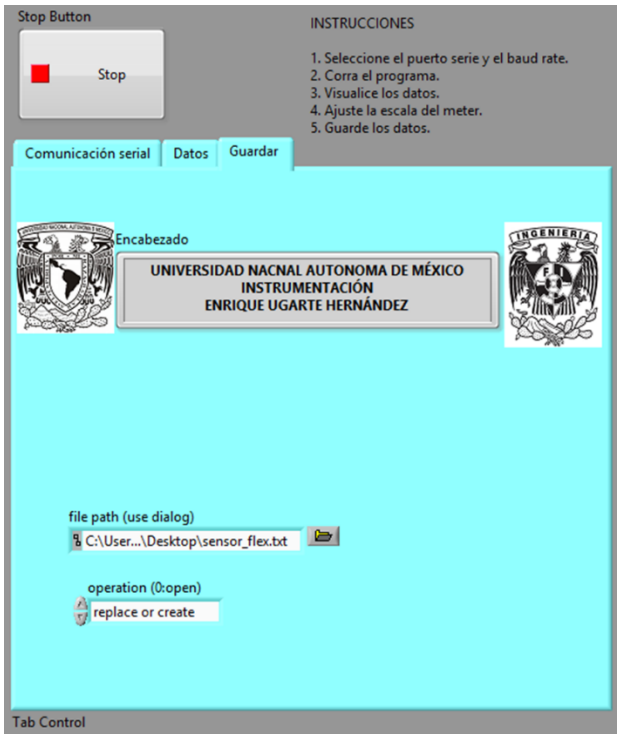


Figura 8. Tercera página del contenedor con pestañas.

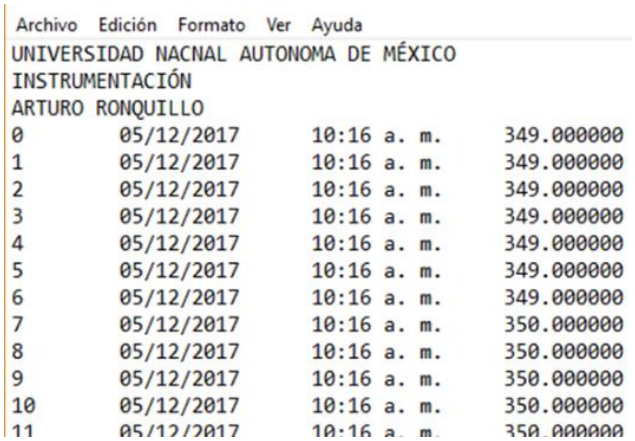


Figura 9. Datos dentro del documento de texto.

V. PROGRAMA DE LABVIEW USANDO LINX

A. Configuración de conexión

Para realizar un programa con LINX, primero hay que ir a *Tools* → *MakerHub* → *LINX* → *LINX Frimware Wizard*. Una vez dentro aparece una ventana donde se ingresan los datos de la tarjeta de adquisición de datos a utilizar.

Figura 10. Ajustes para la comunicación con el Arduino

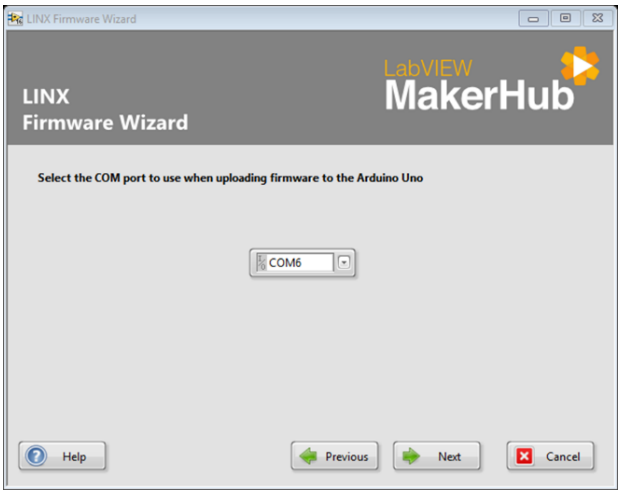


Figura 11. Selección del puerto serie.

Una manera sencilla de conocer el puerto en el que se encuentra conectado el Arduino es revisando el IDE.

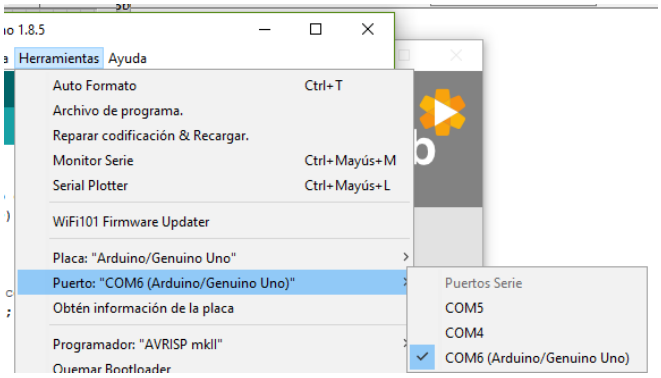


Figura 12. Puerto donde se encuentra conectado el Arduino.

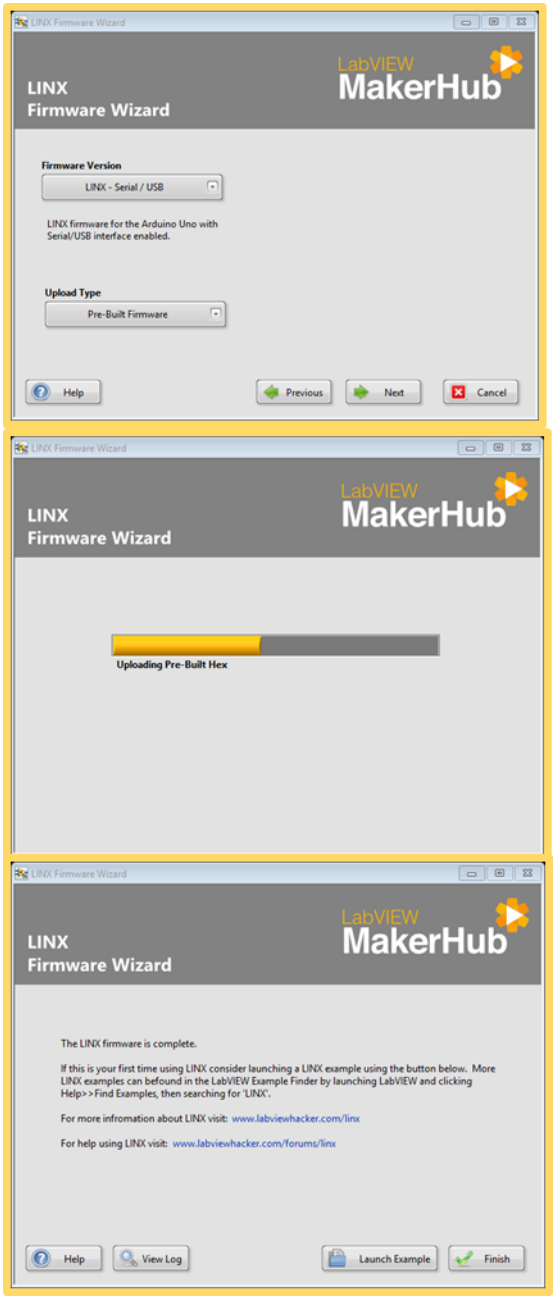


Figura 13. Programa cargando.

B. Comunicación en serie

Para abrir y cerrar la comunicación se necesitan los bloques *Open.vi y Close.vi*, ubicados *en Functions → MakerHub → LINX*. Para leer la información usar el bloque *Analog Read*, ubicado en *Functions → MakerHub → LINX → Peripherals*. Realizar las conexiones dentro de un ciclo *While* y agregar una función *OR* conectada al icono *Loop Condition* para cerrar el programa al presionar el botón *Stop* o al haber alguna falla con la comunicación en serie. Se agrega un *Meter* y *un Waveform Chart* para visualizar la información.

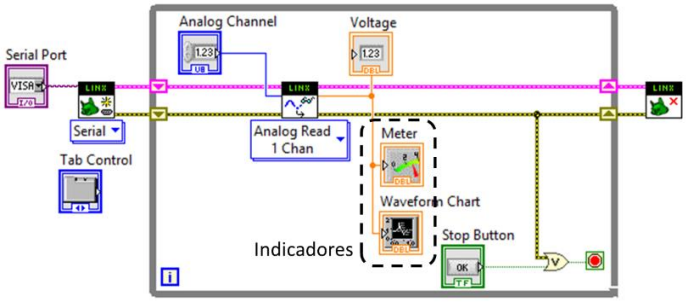


Figura 14. Comunicación en serie e indicadores.

Finamente se utilizan agregar los bloques necesarios para guardar los datos en un documento de texto.

C. Diseño final

A continuación se muestra el diagrama de bloques completo.

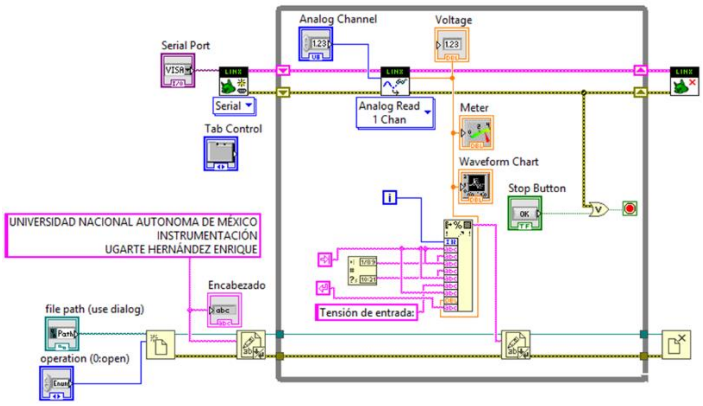


Figura 15. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña, la visualización de los datos en la segunda y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña). Nótese que la lectura analógica se encuentra en volts.

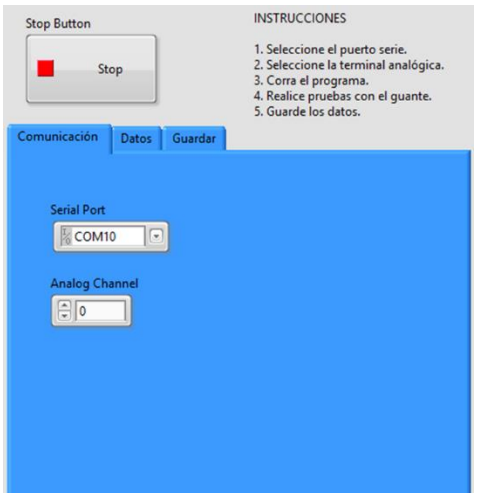


Figura 16. Primera pestaña de la interfaz.

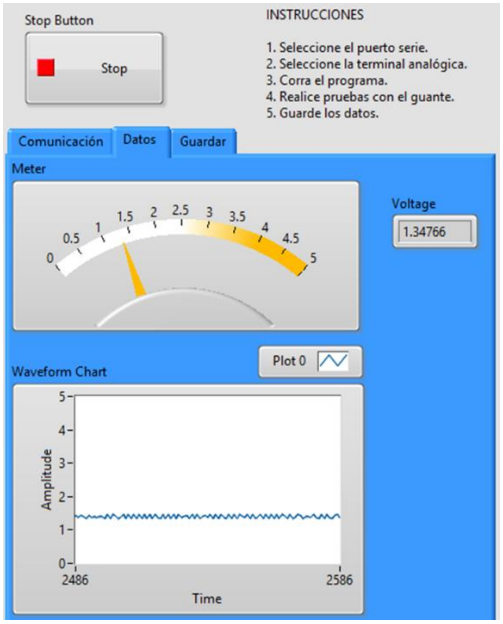


Figura 17. Segunda pestaña de la interfaz.

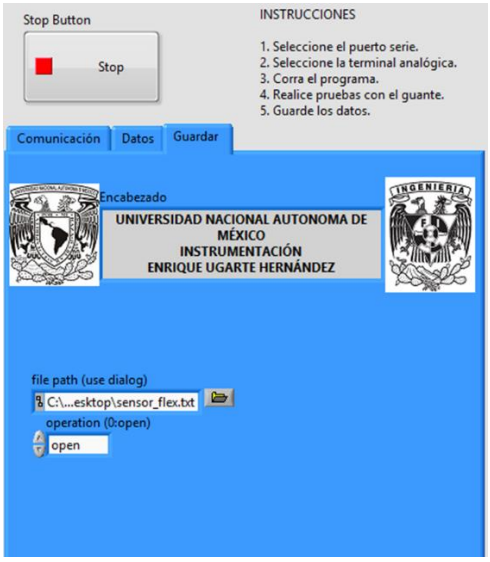


Figura 18. Tercera pestaña de la interfaz.

Normalmente la gráfica se va auto escalando, lo cual en ocasiones no permite visualizar adecuadamente los datos. Para eliminar el auto escalado hacer clic derecho sobre la gráfica y ubicarse en *Y Scale* → *Properties*. Una vez dentro, seleccionar *Amplitude (Y-Axis)* y asignar los valores máximos y mínimos.

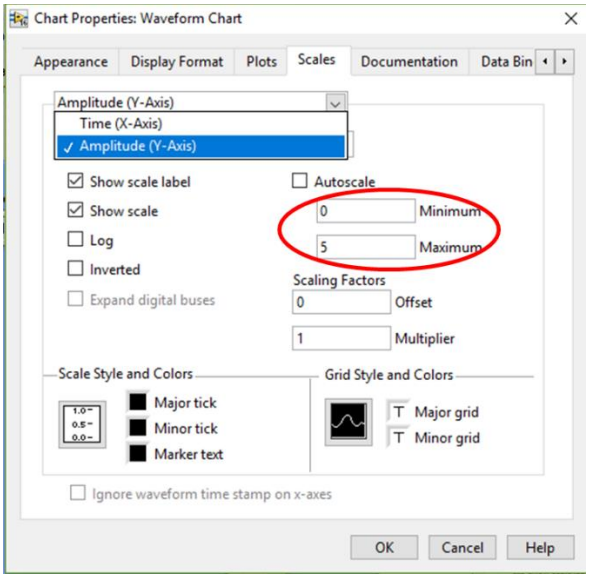


Figura 19. Ajustes de la gráfica.

VI. PROGRAMA USANDO EL TOOLKIT DE ARDUINO

A. Código de Arduino

Cargar el código ubicado en ***C:\Program Files\National Instruments\LabVIEW 2016\vi.lib\LabVIEW Interface for Arduino\Firmware\LIFA_Base*** dentro de la Tarjeta Arduino.

B. Comunicación en serie

Para abrir y cerrar la comunicación con el Arduino se necesitan los bloques *Init* y *Close*, ubicados en *Functions* → *Arduino*. Para leer la información utilizar el bloque *Analog Read Pin*, ubicado en *Functions* → *Arduino* → *Low Level*. Realizar las conexiones dentro de un ciclo *While*. Hacer clic derecho sobre el bloque *Analog Read Pin* para crear un indicador y se agrega un *Waveform Chart* para visualizar la información.

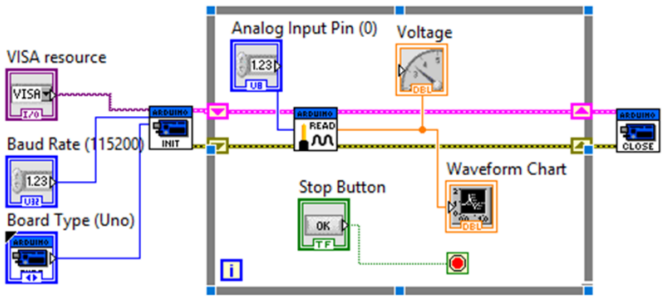


Figura 20. Diagrama de bloques para la comunicación y lectura analógica.

C. Diseño final

A continuación se muestra el diagrama de bloques completo.

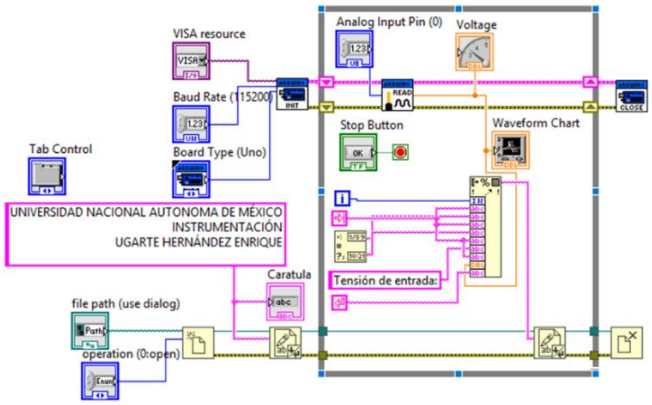


Figura 21. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación en serie se encuentran en la primera pestaña, la visualización de los datos en la segunda y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

Los datos mostrados en el *Meter* y el *Wave Chart* es el voltaje leído por el Arduino, el cual esta escalado de 0 a 1023, donde 1023 corresponde a 5 [V]. Este valor cambiará al flexionar el sensor.

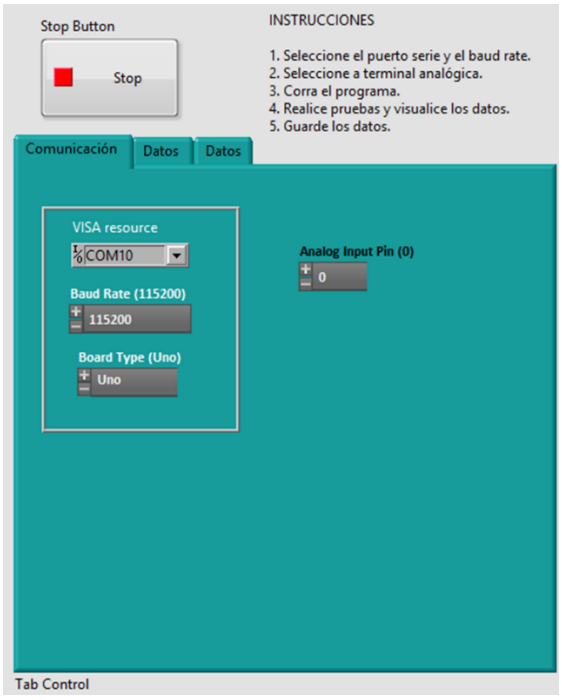


Figura 22. Primera pestaña de la interfaz.

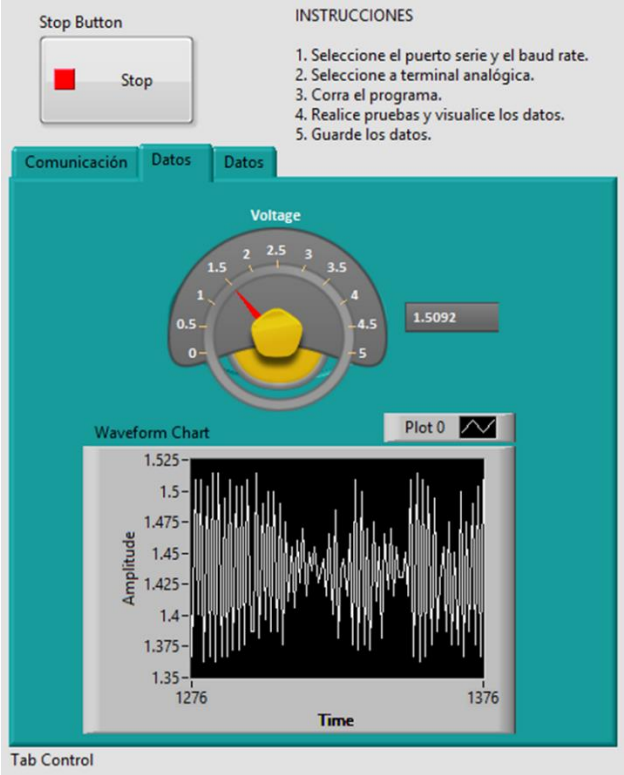


Figura 23. Segunda pestaña de la interfaz.

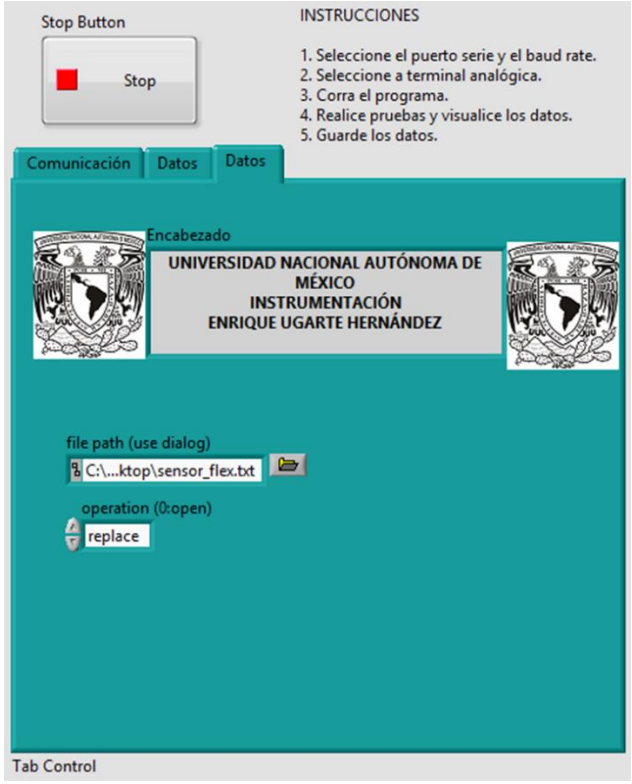


Figura 24. Tercera pestaña de la interfaz.

VII. FUENTES DE CONSULTA

[1] E-Pulse. Sensor Flex 7cm. *BricoGeek*. Recuperado de http://tienda.bricogeek.com/sensores/432-sensor-flex-7cm.html .

[2] Techmake Solutions S.A. de C.V. (2017). Controla un Servomotor con un sensor Flex. Monterrey, Mexico. *Techmake Electronics*. Recuperado de http://www.techmake.com/tutorialflexsensor.

Práctica 3. Motor a pasos

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En este trabajo se explica de forma general el funcionamiento del motor a pasos y se realizan dos programas para manipular uno, uno usando LINX y otro el toolkit de Arduino. Para este propósito se utiliza la tarjeta Arduino UNO como microcontrolador y el entorno de desarrollo de LabVIEW para generar los programas.**

I. INTRODUCCIÓN

A. Generalidades

Un motor a pasos es un dispositivo electromecánico que convierte una serie de pulsos eléctricos en desplazamientos angulares, lo que significa que es capaz de girar una cantidad de grados (paso o medio paso) dependiendo de sus entradas de control [1]. Son muy utilizados en la actualidad para el desarrollo de mecanismos que requieren de una alta precisión. Este tipo de motores poseen cualidades especiales por el hecho de poderlos mover desde un paso hasta una secuencia interminable de pasos dependiendo de la cantidad de pulsos que se les aplique [2].

B. Funcionamiento

Para manipular la velocidad y sentido del motor a pasos se energizan los embobinados internos del motor de manera alternada, los cuales producen un campo magnético que atrae o repele los polos del imán ubicado en el rotor.

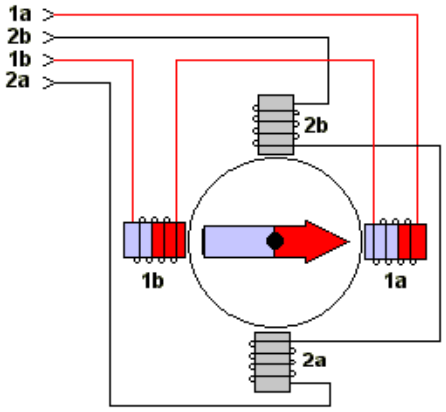


Figura 1. Modelo conceptual del motor a pasos unipolar [3].

C. Clasificación

Según sus características físicas los motores a pasos se pueden clasificar en 3 tipos [5]:

- Reluctancia variable:
  Su rotor está fabricado por un cilindro de hierro dentado y el estator está formado por bobinas. Este tipo de motor trabaja a mayor velocidad que los de imán permanente.
- De imán permanente:
  En los motores tipo imán permanente, su rotor es un imán que posee una ranura en toda su longitud y el estator está formado por una serie de bobinas enrolladas alrededor de un núcleo o polo. Este tipo de motores son los más utilizados y más sencillos de utilizar.
- Híbridos:
  Son una combinación de los de reluctancia variable y de imán permanente con el fin de lograr mayor rendimiento a una buena velocidad.

Según su forma de conexión y excitación se pueden clasificar en 2 tipos [5]:

- Bipolares:
  Este tipo de motores tienen generalmente cuatro cables de salida. Necesitan ciertas manipulaciones para ser controlados, debido a que requieren del cambio de dirección del flujo de corriente a través de las bobinas en la secuencia apropiada para realizar un movimiento. Es necesario además un puente H por cada bobina del motor paso a paso de 4 cables (dos bobinas) [2]. Para su funcionamiento se requieren dos puentes H.

- Unipolares:
  Estos motores suelen tener 6 o 5 cables de salida dependiendo de su conexionado interno, que suelen ser comúnmente 4 cables por los cuales se reciben los pulsos que indican al motor la secuencia y duración de los pasos y los restantes sirven como alimentación al motor [2]. Son más fáciles de trabajar que los bipolares, ya que no requiere de un driver complicado, ni invertir el sentido de la polaridad de alimentación de las bobinas internas.

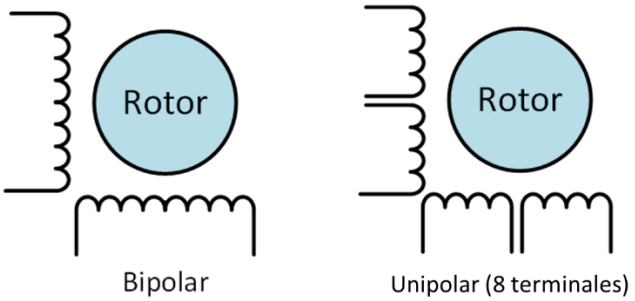


Figura 2. Comparación entre motor a pasos bipolar y unipolar.

D. Driver ULN2003

Es un arreglo de transistores Darlington que soporta hasta 500 mA y dispone de un conector para el motor y de unas terminales (IN1 – IN4) para conectar al Arduino. Sirve para controlar de manera sencilla de motor unipolar 28BYJ-48, tiene 4 terminales para controlar el motor y 2 terminales de alimentación [6].

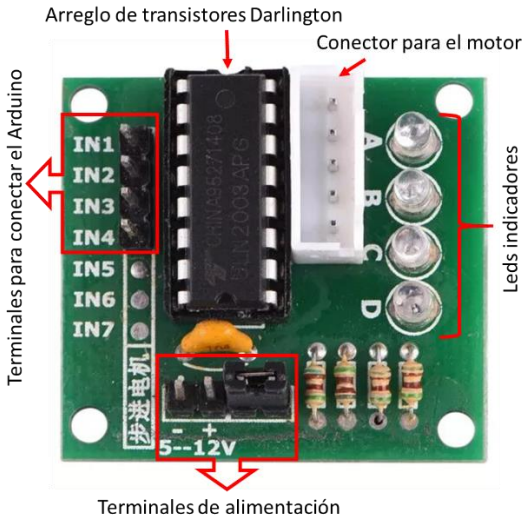


Figura 3. Driver ULN2003

E. Secuencias para motor unipolar

Para este caso particular utiliza un motor unipolar, cuya secuencia puede ser programada de 3 diferentes modos, las cuales se presentan a continuación. La secuencia del motor bipolar no se va a tratar en esta práctica.

1. Secuencia normal

Paso	Bobina A	Bobina B	Bobina C	Bobina D
1	+V	+V	-V	-V
2	-V	+V	+V	-V
3	-V	-V	+V	+V
4	+V	-V	-V	+V

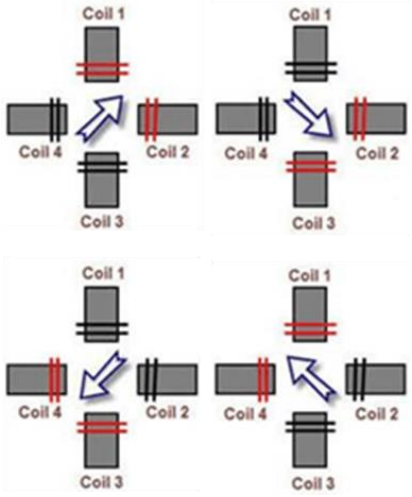


Figura 4. Secuencia normal del motor a pasos.

2. Secuencia tipo wavedrive

Paso	Bobina A	Bobina B	Bobina C	Bobina D
1	+V	-V	-V	-V
2	-V	+V	-V	-V
3	-V	-V	+V	-V
4	-V	-V	-V	+V

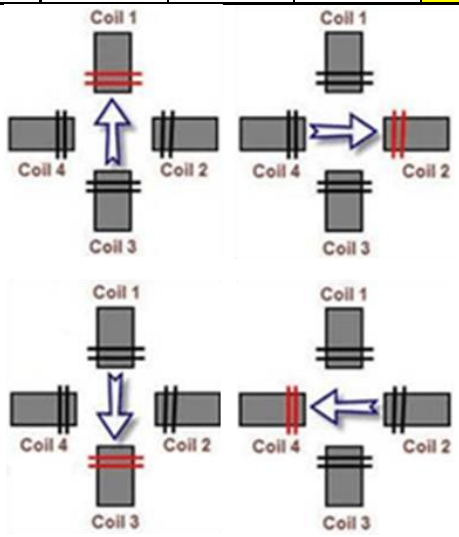


Figura 5. Secuencia tipo wavedrive.

3. Secuencia de medios pasos

Paso	Bobina A	Bobina B	Bobina C	Bobina D
1	+V	-V	-V	-V
2	+V	+V	-V	-V
3	-V	+V	-V	-V
4	-V	+V	+V	-V
5	-V	-V	+V	-V
6	-V	-V	+V	+V
7	-V	-V	-V	+V
8	+V	-V	-V	+V

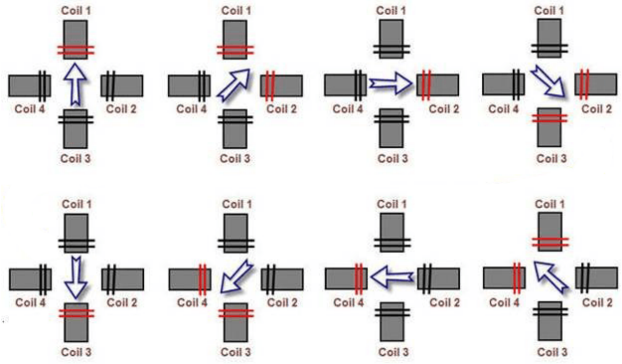


Figura 6. Secuencia de medios pasos [4].

II. OBJETIVOS

- ✓ Conocer los principios básicos de funcionamiento de un motor a pasos y aprender a utilizar uno.
- ✓ Aprender a utilizar la herramienta LINX y el toolkit de Arduino para generar interfaces gráficas.
- ✓ Desarrollar una interfaz gráfica que permita manipular un motor a pasos, cambiando su rapidez angular, sentido de giro y que cuente con un botón de paro.

III. MATERIAL

- Motos a pasos 28BYJ-48.
- Driver ULN2003.
- 6 Jumpers H-M.
- Fuente (opcional).
- 2 cables banana-caimán (opcional).
- Tarjeta Arduino UNO.
- Computadora con el software IDE de Arduino y LabVIEW 2012 o superior.

IV. DESARROLLO

A. Conexión

Conectar el motor al driver.

- 1N1 → terminal 2.
- 1N2 → terminal 3.
- 1N3 → terminal 4.
- 1N4 → terminal 5.
- (-) → GND o 0V.
- (+) → 5V o (5-12) V.
- Fuente (opcional).

Nota: Es recomendable conectar las terminales de alimentación a una fuente externa, puenteando las tierras para incrementar la potencia del motor.

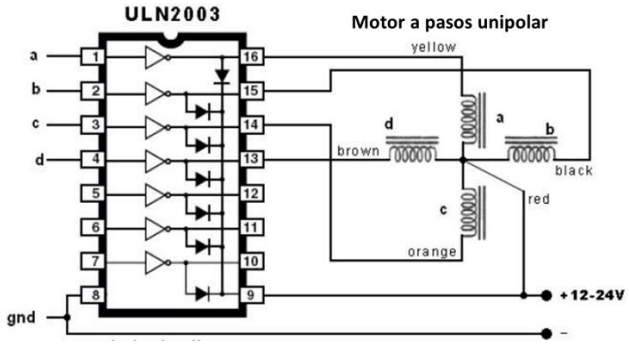


Figura 7. Esquema del driver ULN2003 y el motor a pasos [7].

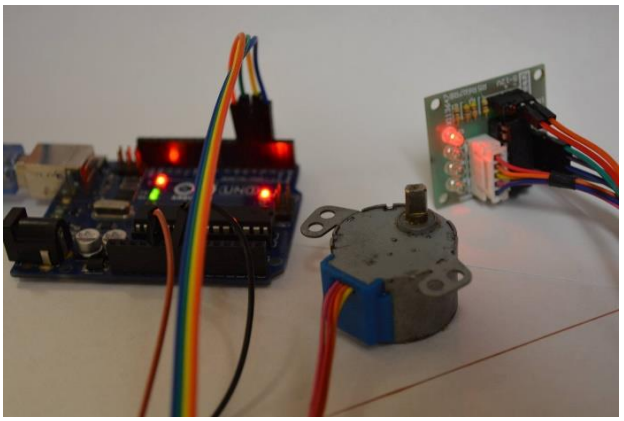


Figura 8. Motor a pasos trabajando.

B. Programa usando LINX

1) Configuración para la conexión

Para realizar un programa con LINX, primero nos vamos a *Tools → MakerHub → LINX → LINX Frimware Wizard*. Una vez dentro nos aparecerá una ventana donde tenemos que ingresar los datos de la tarjeta de adquisición de datos a utilizar.

Figura 9. Ajustes para la comunicación con el Arduino.

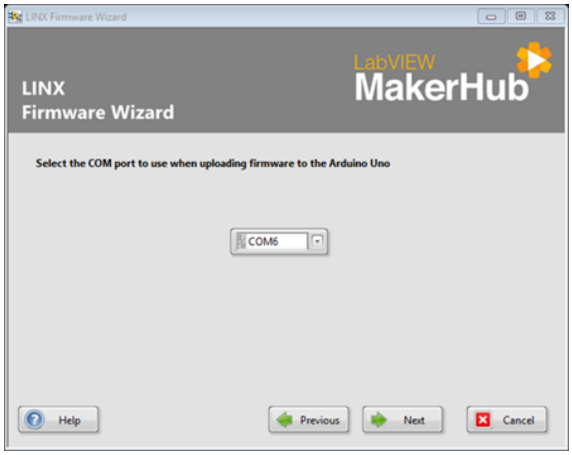


Figura 10. Selección del puerto serial.

Una manera sencilla de conocer el puerto en el que se encuentra conectador el Arduino es revisando el IDE.

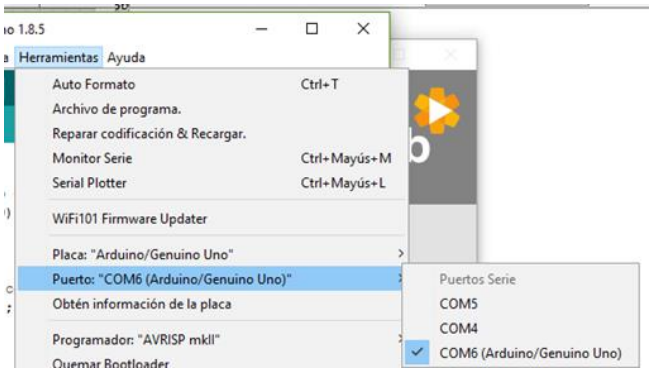


Figura 11. Puerto donde se encuentra conectado el Arduino.

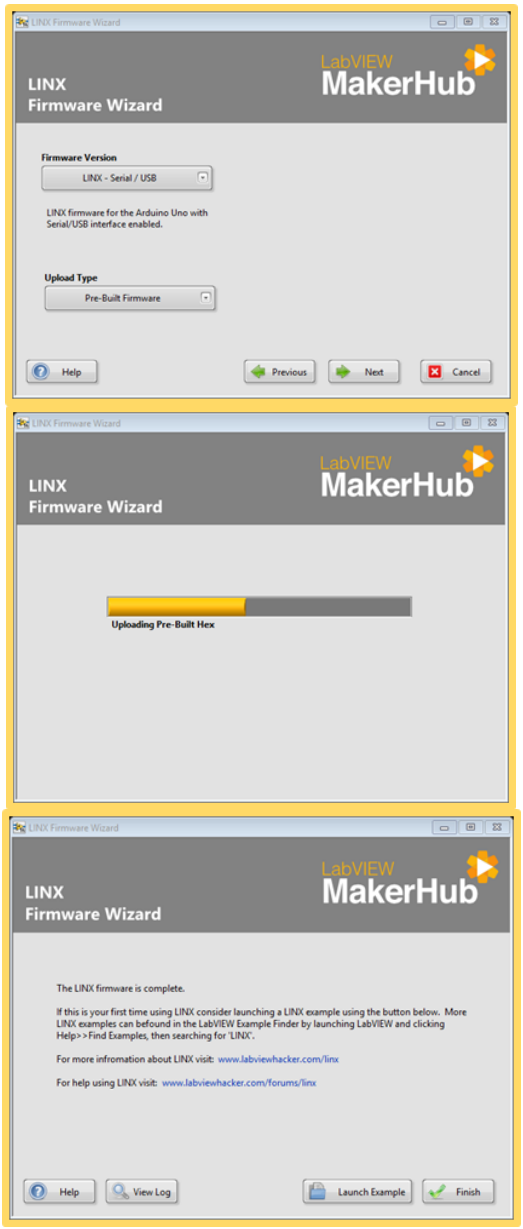


Figura 12. Programa cargando.

2) Bloques para la comunicación

Para abrir y cerrar la comunicación se necesitan los bloques *Open.vi y Close.vi* ubicados *en Functions → MakerHub → LINX*. Para escribir la secuencia usamos el bloque *Digital Write.vi*, ubicado en *Functions → MakerHub → LINX → Peripherals* y realizamos las conexiones dentro de un ciclo *While*. Se crea una variable local, llamada *Secuencia*, esta se va a asociar al paso en que se encuentra el motor. Agregar un slider en el panel frontal para seleccionar los casos en un Case para establecer 10 magnitudes de velocidad diferentes. Cuando se coloca el bloque Digital Write.vi por default dice *Digital Write 1 Chan*, se tiene que hacer clic sobre él para qué cambie a *Digital Write N Chan*.

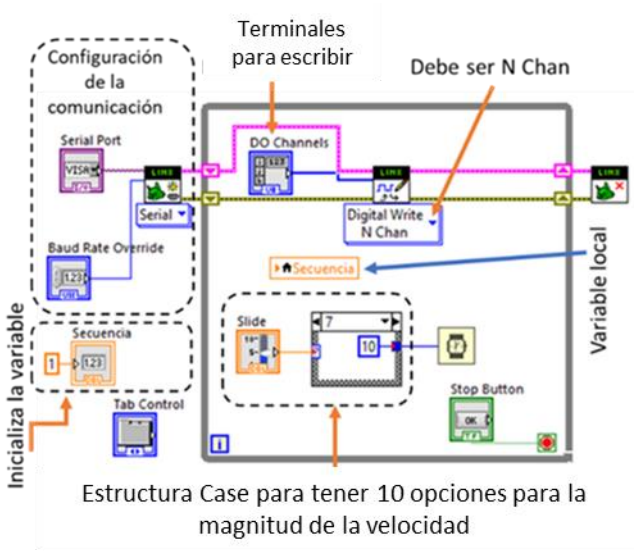


Figura 13. Bloques para la comunicación, variable local y el *Case* para controlar la rapidez.

Valores de *Case* para la velocidad:

Numero de caso	Valor de retardo [ms]
0, Default	500
1	40
2	35
3	30
4	25
5	20
6	15
7	10
8	5
9	2

Para cambiar el tipo de dato del slider y del indicador *Secuencia* hacer clic derecho sobre el elemento y dirigirse a *Propierties → Data Type*. Una vez dentro, hacer clic en el icono *Representación*, allí se desplegan todas las opciones.

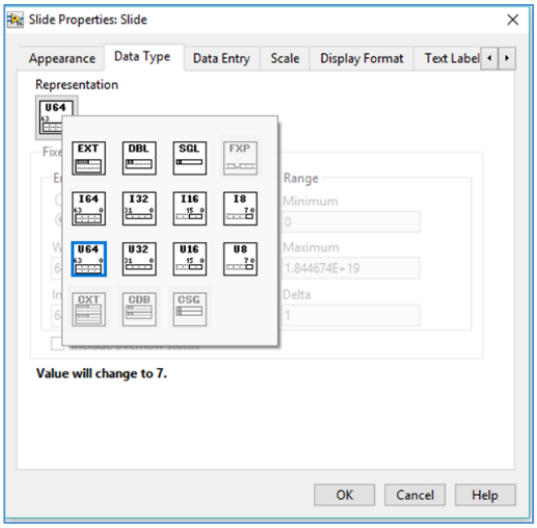


Figura 14. Cambio de tipo de dato.

3) Manejo de datos

Para este programa se utiliza una secuencia llamada *Wavedrive*, cada uno de los 4 pasos, está representado por un valor en la variable *Secuencia*, del 1 al 4. Por esto, se programa un *Case* con otros bloques para que la variable vaya incrementando o decrementando dependiendo de un botón para el sentido.

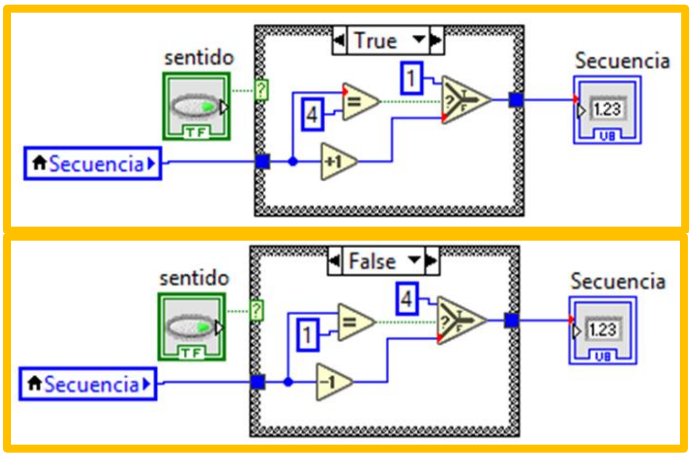


Figura 15. Estructura *Case* para definir el sentido.

Después de tener lista la variable que decide la secuencia, crear una estructura *Case*, controlada por la variable *Secuencia*, el cual manda los comandos de encendido y apagado. Añadir un botón y usar la función AND para que la salida también este en función de este botón, el cual servirá para quitar la energía de las bobinas internas del motor. Finalmente usar el bloque *Build Array*, ubicado en *Functions* → *Programming* → *Array*, para crear un arreglo, el cual posteriormente se usará para visualizar con un indicador.

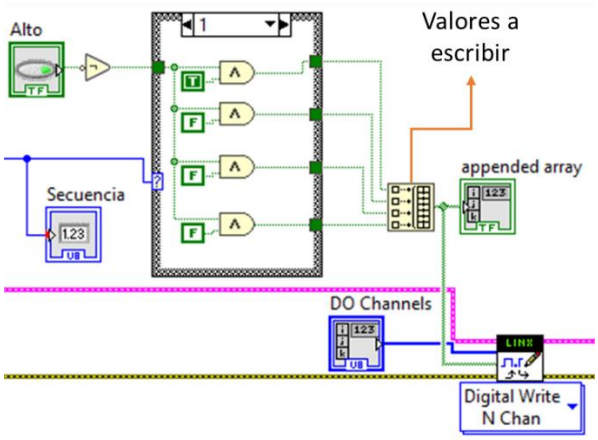


Figura 16. Salidas digitales.

A continuación se muestra la salida, correspondiente a cada paso:

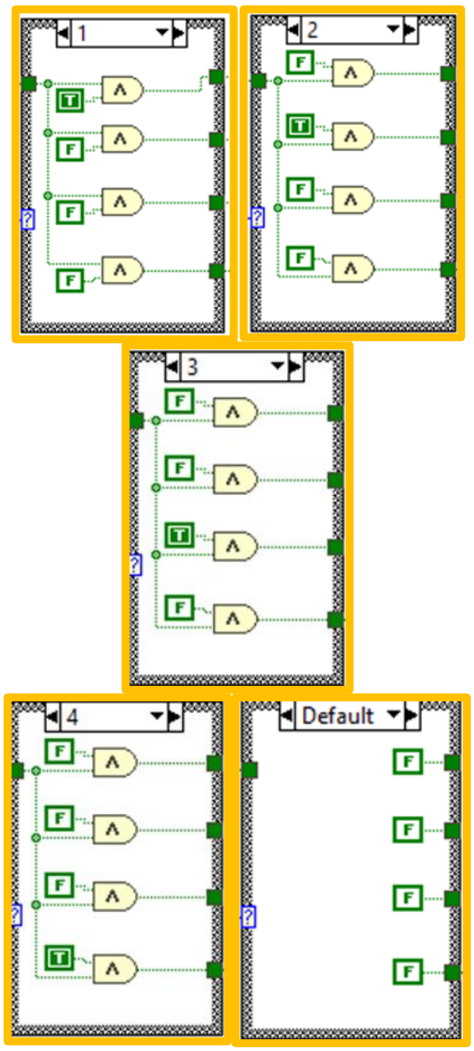


Figura 17. Casos para la secuencia tipo Wavedrive.

4) Diseño final

A continuación se muestra el diagrama de bloques completo.

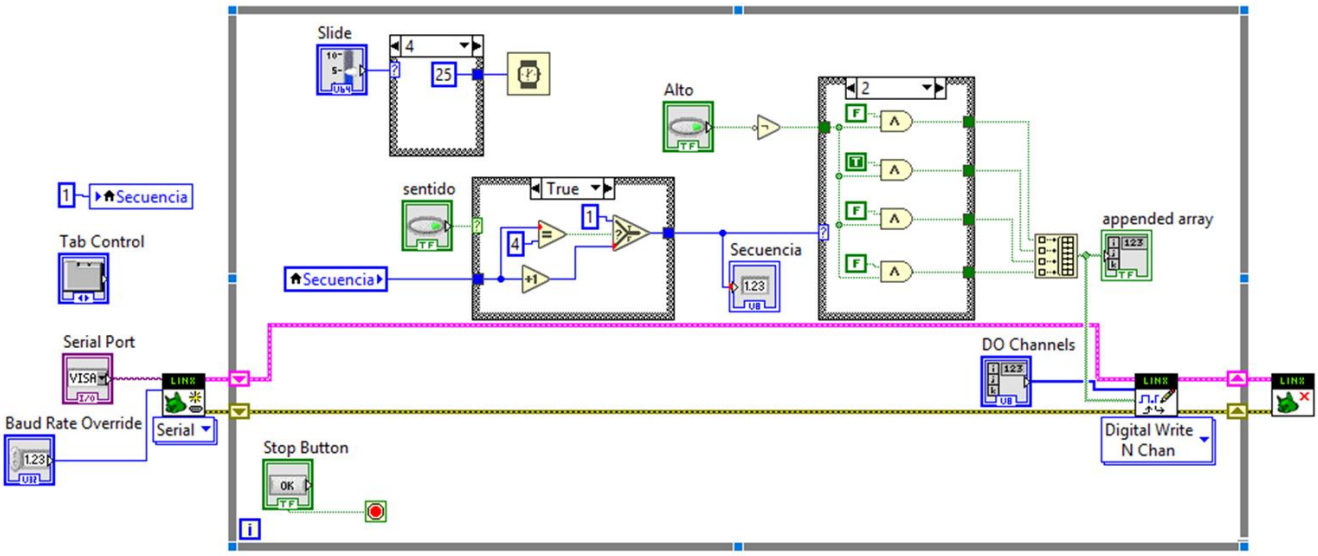


Figura 18. Diagrama de bloques completo.

En el panel frontal se encuentra un contenedor con dos pestañas. En la primera se encuentran los elementos para definir el puerto serie y la velocidad de la comunicación. En la segunda pestaña se encuentra el slider para manipular la rapidez del motor, un botón para detener el motor, otro para cambiar el sentido, un control para establecer las terminales de control e indicadores para visualizar la secuencia.

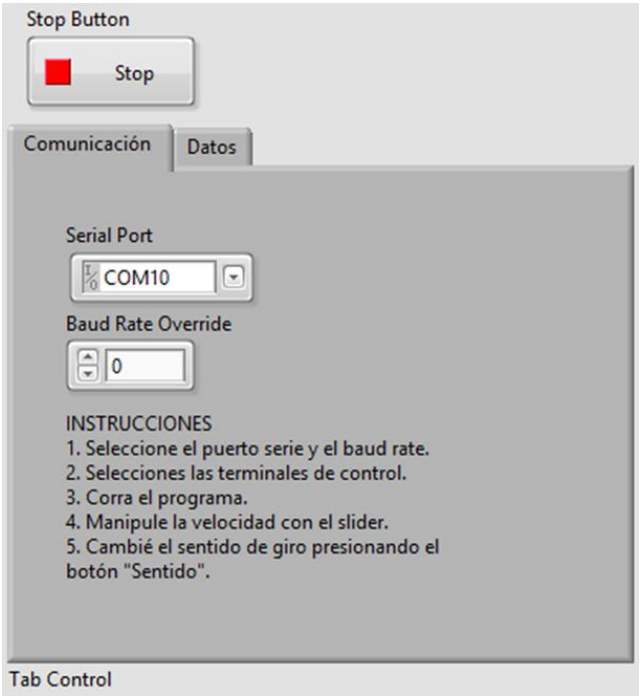


Figura 19. Primera pestaña del panel frontal.

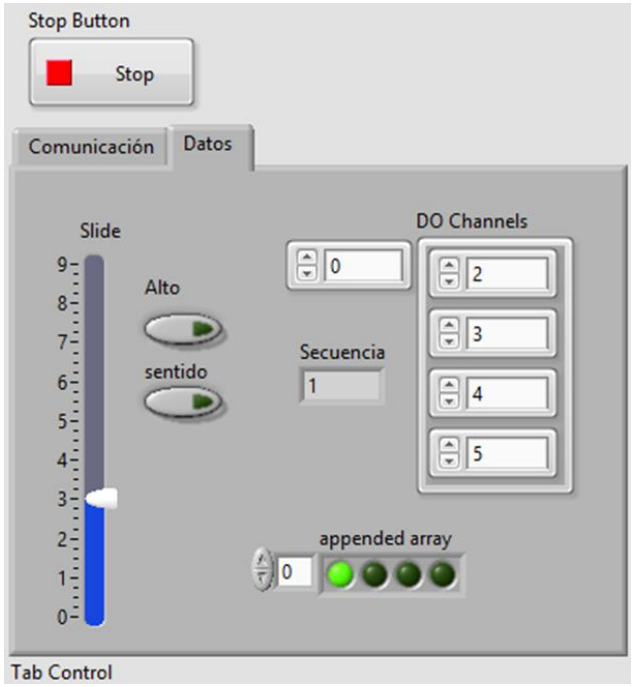


Figura 20. Segunda pestaña del panel frontal.

C. Programa usando el toolkit de Arduino

1) Código de Arduino

Cargar el código ubicado en ***C:\Program Files\National Instruments\LabVIEW 2016\vi.lib\LabVIEW Interface for Arduino\Firmware\LIFA_Base*** dentro de la Tarjeta Arduino.

2) Bloques para la comunicación

Para abrir y cerrar la comunicación con el Arduino se necesitan los bloques *Init* y *Close*, ubicados en *Funtions* → *Arduino*. Para

escribir la secuencia se utilizan 4 bloques *Digital Write Pin*, este se ubica en *Funtions* → *Arduino* → *Low Level*. Se realizan las conexiones dentro de un ciclo *While*. Se genera un *Case*, como se describe en el programa de LINX, para manipular la rapidez utilizando un slider y creamos una variable local llamada *Secuencia*.

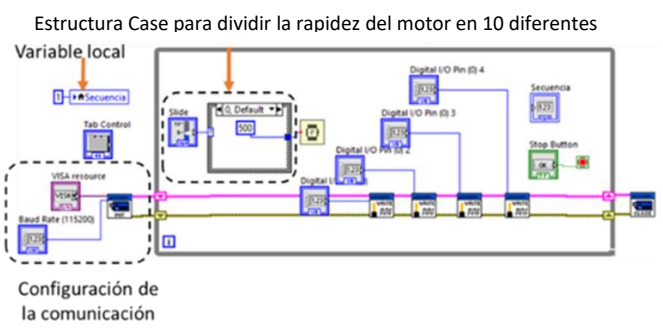


Figura 21. Bloques para la comunicación, variable local y la estructura *Case* para controlar la rapidez del motor.

3) Manejo de datos

Para este programa se utilizó una secuencia llamada *Wavedr*ive, cada uno de los 4 pasos, está representado por un valor en la variable *Secuencia*, del 1 al 4. Por esto, se programó un *Case* con otros bloques para que la variable vaya incrementando o decrementando dependiendo de un botón para el sentido.

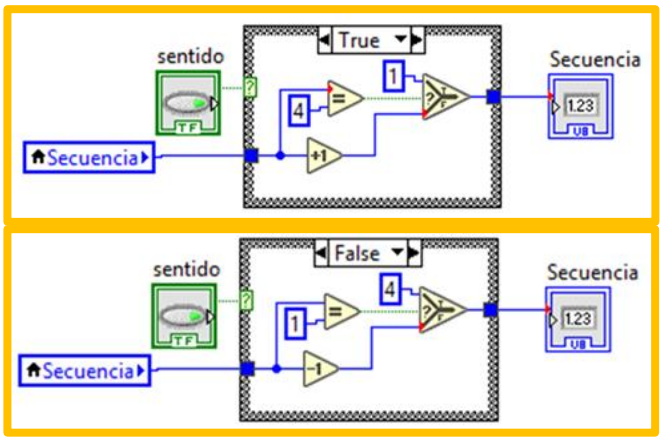


Figura 22. Estructura *Case* para definir el sentido.

Se programa un *Case* (al igual que el programa de LINX), el cual manda los comandos de encendido y apagado en función del valor de la variable *Secuencia*, se integra el *Case* de la secuencia y de las salidas dentro de otro *Case*, el cual será controlado por un botón *Apagado*, con el fin de detener la secuencia.

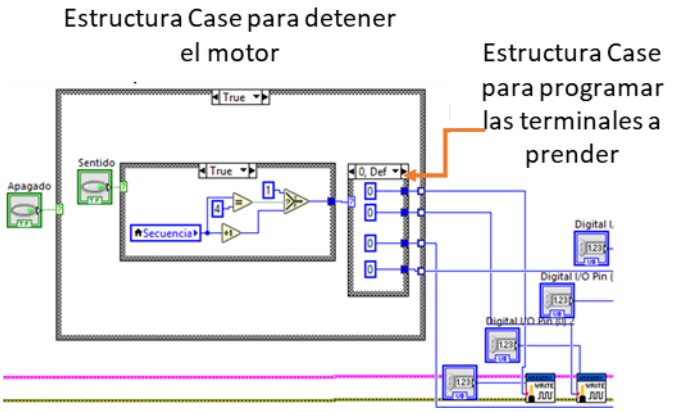


Figura 23. Case para definir el Alto del motor.

En el caso *False*, se quita la energía a los embobinados del motor. Se puede agregar una compuerta NOT para que se detenga el motor presionar el botón Alto, o dejar solo el NOT para que el motor arranque al presionarlo.

Figura 24. Caso FALSE para detener el motor.

Para visualizar la secuencia con indicadores tipo lógicos (*boolean*) se debe convertir las salidas en binario. Es posible sacar el botón *Sentido* del *Case*, para optimizar el espacio.

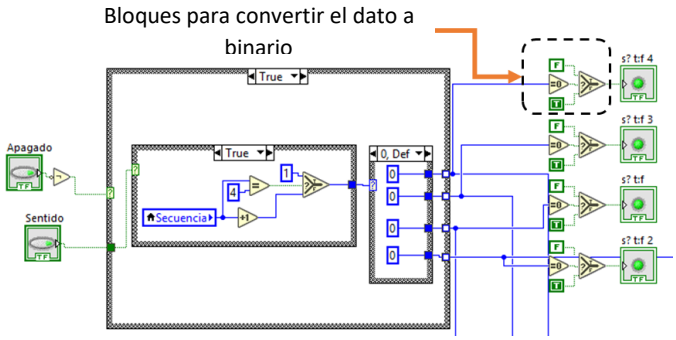


Figura 25. Generación de los indicadores.

4) Diseño final

A continuación se muestra el diagrama de bloques completo.

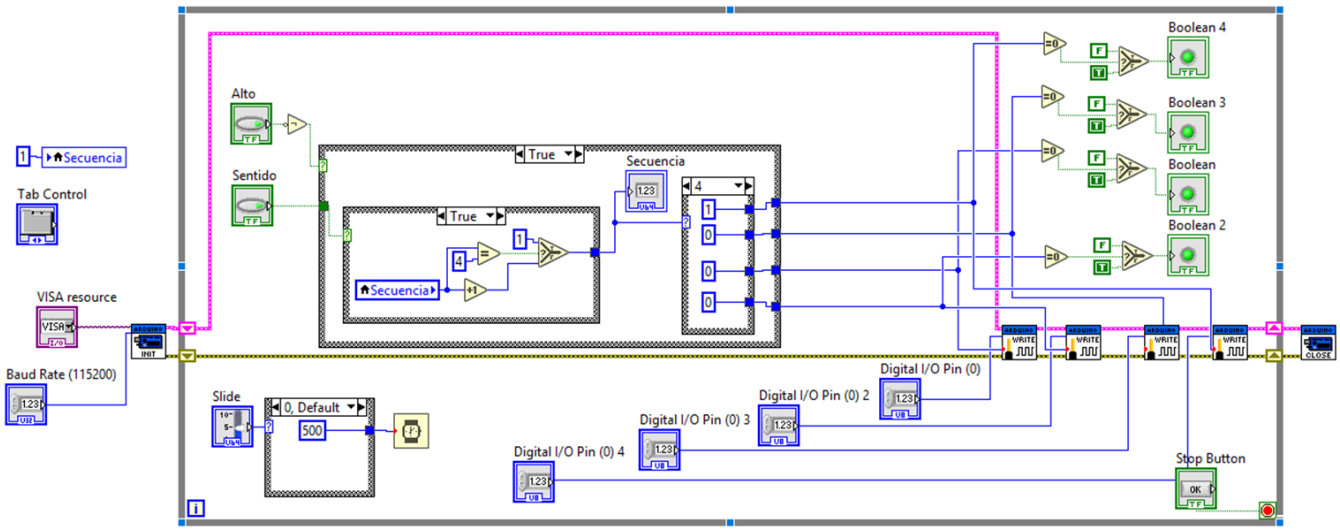


Figura 26. Diagrama de bloques completo.

En el panel frontal, se encuentra un contenedor con dos pestañas. En la primera se encuentran los elementos para definir el puerto serie y la velocidad de comunicación. En la segunda pestaña se encuentra el slider para manipular la rapidez del motor, un botón para detener el motor, otro para cambiar el sentido, un control para establecer las terminales a controlar e indicadores para visualizar la secuencia.

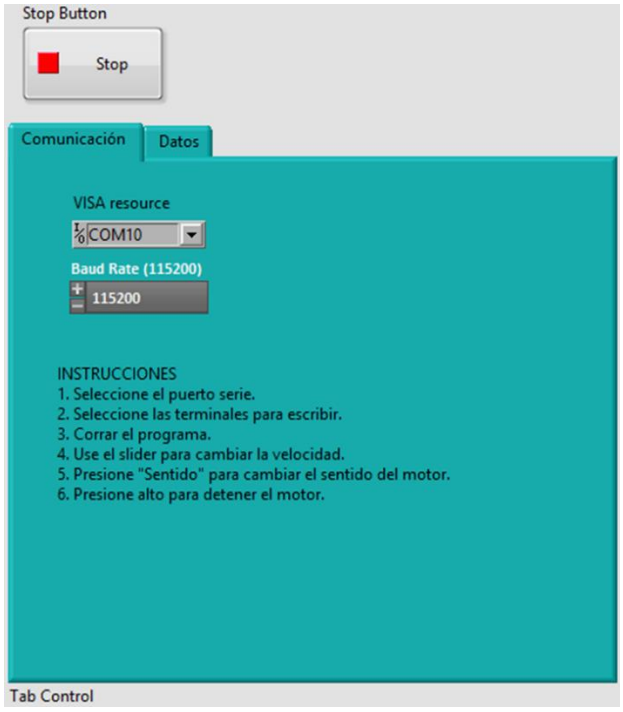


Figura 27. Primera pestaña del panel frontal.

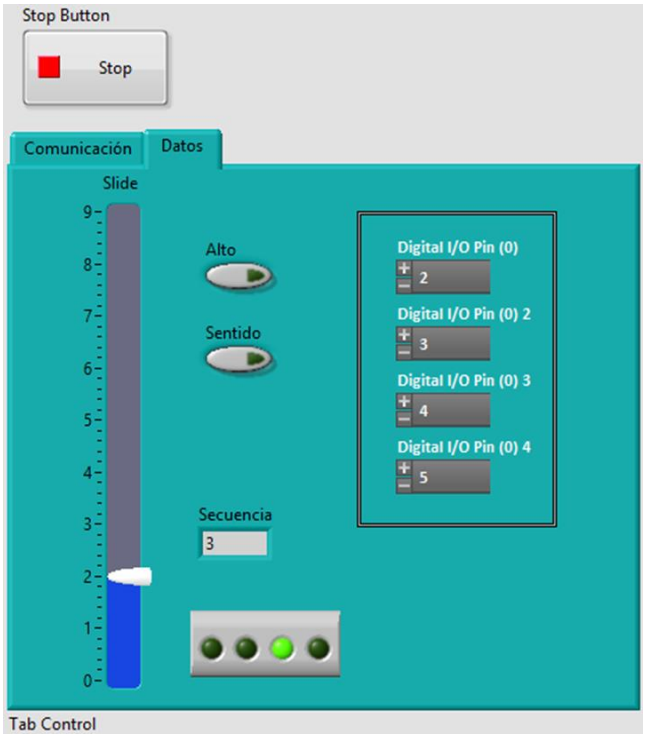


Figura 28. Segunda pestaña del panel frontal.

V. FUENTES DE CONSULTA

[1] Ingeniería mecafenix (2008). Motor paso a paso ¿qué es y cómo funciona?. *Ingmecafenix.com*. Recuperado de http://www.ingmecafenix.com/electricidad-industrial/motor-paso-a-paso/ .

[2] 3. SECCIÓN DE MOTORES A PASOS. Recuperado de http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/hernandez_b_ii/capitulo3.pdf .

[3] 330ohms (2016). Motores a pasos... ¿unipolares o bipolares?. *330ohms.com*. Recuperado de https://www.330ohms.com/blogs/blog/85507012-motores-a-pasos-unipolares-o-bipolares .

[4] Llamas, Luis. (20016). TIPOS DE MOTORES ROTATIVOS PARA PROYECTOS DE ARDUINO. *Luisllamas.es*. Recuperado de https://www.luisllamas.es/tipos-motores-rotativos-proyectos-arduino/

[5] Departamento de Ingeniería Electrónica. (1996-2003). TIPOS DE MOTORES PASO A PASO. Escuela Politécnica Superior de Alcoy. *http://server-die.alc.upv.es/*. Recueprado de http://server-die.alc.upv.es/asignaturas/lsed/2002-03/MotoresPasoaPaso/tipos.htm

[6]. Llamas, Luis. (20016). MOTOR PASO A PASO 28BYJ-48 CON ARDUINO Y DRIVER ULN2003. *Luisllamas.es*. Recuperado de https://www.luisllamas.es/motor-paso-paso-28byj-48-arduino-driver-uln2003/ .

[7] Bibin, John (s.f.). DRIVING STEPPER MOTORS. *www.Geocities.ws* . Recuperado de http://www.geocities.ws/njbibin/robo16i.html .

Práctica 4. Transformador diferencial de variación lineal (LVDT)

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En este trabajo se explica cómo construir un transformador diferencial de variación lineal (LVDT) de dos maneras: una utilizando un transformador para excitar el embobinado primario y otra con una tensión de entrada DC para generar un oscilador que excite la bobina primaria. Se utiliza la tarjeta Arduino UNO para la adquisición de datos y el entorno de desarrollo de LabVIEW para la generación de una interfaz que permita visualizar el desplazamiento del núcleo ferromagnético a través del LVDT.**

I. INTRODUCCIÓN

A. Sensores inductivos

Son sensores que aprovechan el cambio de la inductancia de una bobina en presencia de un conductor [1]. Conforme el objeto se acerca al sensor, aumenta el flujo de corriente de inducción (corrientes eddy o de Foucault) debido a la inducción electromagnética, lo cual provoca que la carga en el circuito de oscilación crezca; de esta manera la amplitud de la señal se incrementará o reducirá en función de la distancia entre el sensor y el objeto [2].

B. Funcionamiento del LVDT

El LVDT (*Linear Variable Differential Transformer)* es un sensor no invasivo que permite medir posición; está formado de 3 embobinados concéntricos y un núcleo de un material ferromagnético, el cual al irse desplazando produce un cambio de diferencia de potencial en los embobinados secundarios.

Un sensor no invasivo es un sensor ubicado a cierta distancia del objeto a medir, de tal manera que no afecta la integridad del objeto a medir.

El LVDT requiere de una excitación de corriente alterna en el embobinado primario, el cual induce una corriente en los secundarios para posteriormente restar las tensiones de salida de estos. Para poder conectar la salida del LVDT al Arduino es necesario rectificar la señal de salida. Cabe decir que se obtiene el mismo resultado si primero se rectifica y después se restan las tensiones de salida de los embobinados secundarios.

Cuando se acerca el núcleo ferromagnético a algún embobinado secundario el núcleo va incrementando el número de vueltas que son afectadas por el núcleo incrementado la tensión de salida de la bobina por lo que se debe utilizar un núcleo que tenga al menos el largo de dos bobinas del LVDT.

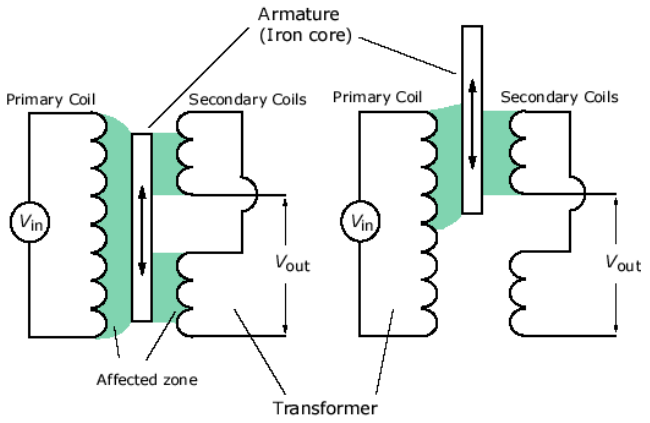


Figura 1. Funcionamiento del LVDT [3].

C. Cálculo de las bobinas

Para analizar las bobinas se va a dividir el circuito en tres etapas: La primera etapa incluye los cálculos de la bobina primaria; la segunda etapa, los cálculos de las bobinas secundarias y la tercera, el efecto del puente rectificador.

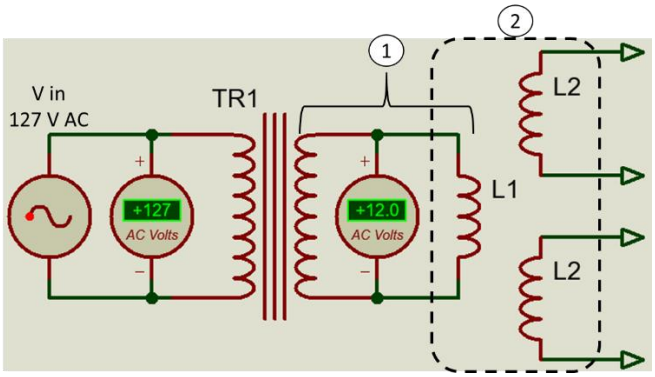


Figura 2. Bobinas del LVDT con excitación de transformador. Se muestran dos etapas para realizar los cálculos. La bobina primaria está representada con L1 y as bobinas secundarias con L2.

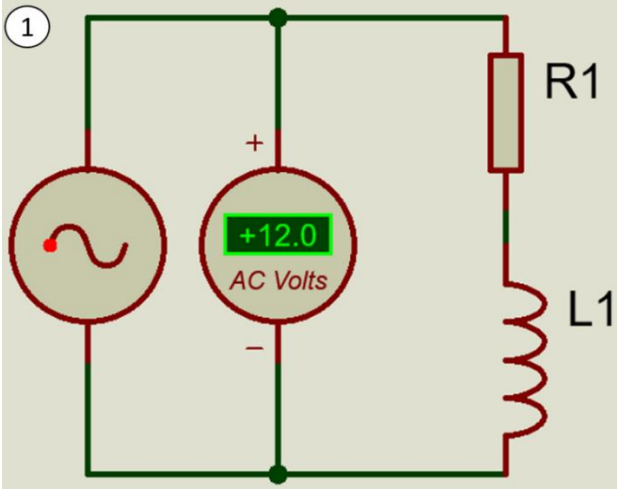


Figura 3. Representación del circuito de la bobina primaria (L1). Nótese que hay un resistor, el cual sirve para indicar que la bobina posee cierta resistencia.

Análisis la bobina primaria

Partiendo de la Ley de Ohm en términos de fasores:

$$V = R \cdot I \qquad (1)$$

Donde:

$V = Tensión\ de\ la\ bobina\ [V]$

$R = Resistencia\ de\ la\ bobina\ [\Omega]$

$I = Intensidad\ de\ corriente\ de\ la\ bobina\ [A]$

La resistencia de una bobina se llama reactancia inductiva.

$$X_L = 2\pi \cdot f \cdot L \quad (2)$$

Donde:

$X_L = Reactancia\ inductiva\ [\Omega]$

$f = frecuencia\ [Hz]$

$L = Inductancia\ [H]$

Para obtener la inductancia:

$$L = \mu \frac{N^2 \cdot A}{l} \qquad (3)$$

Donde:

$\mu = permeabilidad\ magnética\ del\ núcleo\ \left[\frac{Wb}{A \cdot m}\right]$

$N = Número\ de\ vueltas\ de\ la\ bobina$

$A = Área\ del\ núcleo\ ferromagnético\ [m^2]$

Si el diámetro del núcleo de hierro es $1.9\ x\ 10^{-2}[m]$ y la permeabilidad magnética del hierro es $1.8\ x\ 10^{-3}\ \left[\frac{Wb}{A \cdot m}\right]$

$$A = \frac{\pi}{4}(1.9\ x\ 10^{-2})^2$$

$$\therefore A = 2.8352\ x\ 10^{-4}\ [m^2]$$

Considerando la bobina primaria con:

$$N = 500$$

$$l = 5\ x\ 10^{-2}\ [m]$$

Sustituyendo N y l en (3):

$$L = \frac{(1.8\ x\ 10^{-3})\ (500^2)\ (2.8352\ x\ 10^{-4})}{5\ x\ 10^{-2}}$$

$$\therefore L1 = 2.551\ [H]$$

Sustituyendo L en (2) y considerando $f = 60\ [Hz]$ debido a que es la frecuencia de la tensión de la red eléctrica.

$$X_L = 2\pi\ (60)(2.551) = 961.7\ [\Omega]$$

$$\therefore R = 961.7\ [\Omega]$$

Sustituyendo V y R en (1)

$$I = \frac{V}{R} = \frac{12}{961.7} = 0.0124779\ [A] = 12.4779\ [mA]$$

Por lo tanto la corriente que circula en la bobina primaria es $12.4779\ [mA]$

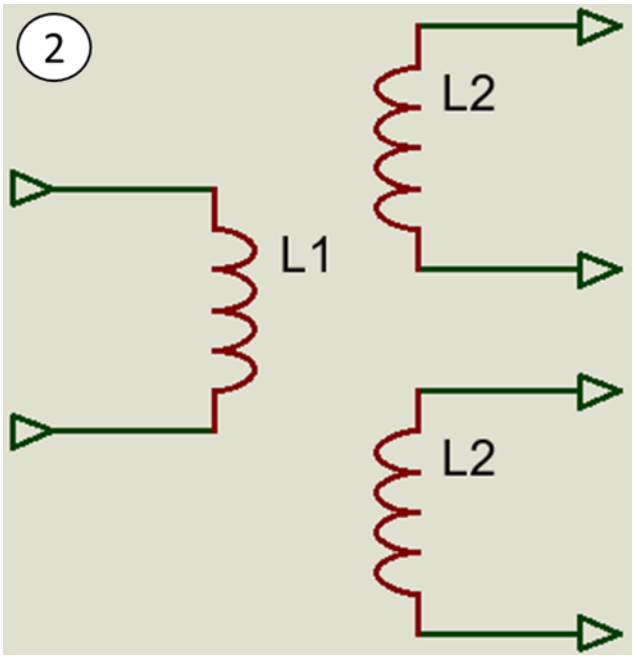


Figura 4. Representación de la bobina primaria (L1) y las bobinas secundarias (L2).

Análisis las bobina secundarias

Tomando en cuenta que están hechas del mismo alambre de cobre y que tienen 400 vueltas cada una:

$$\frac{V_{L2}}{V_{L1}} = \frac{N_{L2}}{N_{L1}} = \frac{400}{500} = \frac{4}{5} \qquad (4)$$

Donde:

$V_{L1} = Tensión\ en\ la\ bobina\ primaria\ [V]$

$V_{L2} = Tensión\ en\ cada\ bobina\ secuendaria\ [V]$

$N_{L1} = Número\ de\ vueltas\ de\ la\ bobina\ primaria$

$N_{L2} = Número\ de\ vueltas\ de\ cada\ bobina\ secuendaria$

Despejando V_{L2} de (4)

$$V_{L2} = \frac{4}{5}V_{L1} = \frac{4}{5}(12) = 9.6\ [V]$$

Igualando la potencia de entrada y de salida:

$$W_{L1} = W_{L2} \qquad (5)$$

Donde:

$W_{L1} = Potencia\ de\ la\ bobina\ primaria\ [W]$

$W_{L2} = Potencia\ de\ cada\ bobina\ secuendaria\ [W]$

Se obtiene

$$V_{L1} \cdot I_{L1} = V_{L2} \cdot I_{L2} \qquad (6)$$

$I_{L1} = Intensidad\ de\ corriente\ en\ la\ bobina\ primaria\ [A]$

I_{L2}
$= Intensidad\ de\ corriente\ en\ cada\ bobina\ secuendaria\ [A]$

$$\therefore\ W_{L1} = (12)(0.0124779) = 0.1497\ [W]$$

Despejando I_{L2} de (5)

$$I_{L2} = \frac{W_{L1}}{V_{L2}} = \frac{0.1497}{9.6} = 0.015583\ [A] = 15.58\ [mA]$$

Por lo tanto la tensión máxima en cada embobinado secundario es de 9.6 [V] AC.

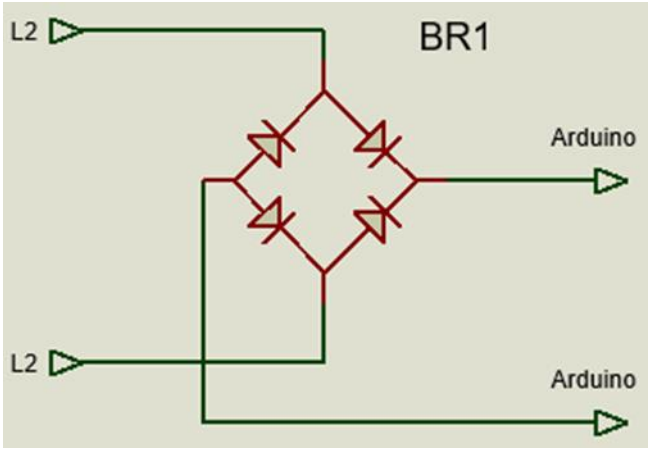


Figura 5. Puente rectificador (BR1) de la diferencia de tensión de las bobinas secundarias (L2).

Efecto del puente rectificador

Rectificando la diferencia de tensión de las bobinas secundarias:

$$V_{rms} = \frac{1}{\sqrt{2}} \cdot V = (0.707106)(9.6) = 6.7882\ [V]$$

Restando la caída de tensión provocado por los diodos del puente rectificador

$$V_{in} = 6.7882 - 1.4 = 5.388\ [V]$$

Por la tanto la máxima tensión que llega al Arduino es $5.388\ [V]$ y la corriente de entrada es $15.58\ [mA]$. Debido a que la potencia de cada bobina es $0.1497\ [W]$ se puede ocupar una resistencia que soporte una potencia máxima de 1/4 W para conectar en paralelo a la entrada del Arduino.

Nota: Debido a perturbaciones, pérdidas y elementos no modelados; las tensiones y corrientes pueden cambiar.

D. Oscilador de puente de Wien

Es un oscilador generado a partir de la combinación de un circuito adelanto-atraso, el cual se encarga de generar la señal senoidal y un amplificador no inversor. La frecuencia está dada por la siguiente fórmula [4]:

$$\boldsymbol{f = \frac{1}{2\pi RC}} \qquad (7)$$

Donde:

$f = Frecuencia\ [Hz]$

$R = Resistencia\ [\Omega]$

$C = Capacitancia\ [F]$

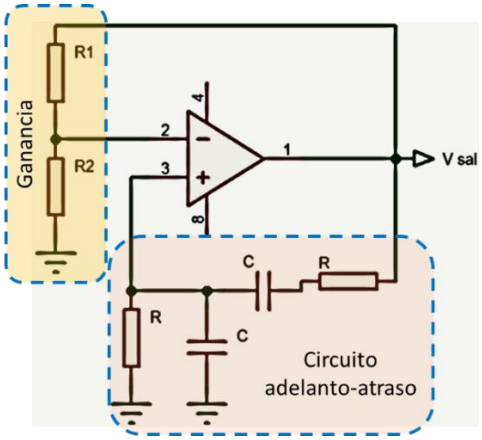


Figura 6. Oscilador de puente de Wien.

Inicialmente el oscilador está apagado y no hay tensión de salida, por lo que la excitación del circuito es resultado del ruido de banda ancha producido térmicamente en los resistores u otros componentes transitorios al encender la fuente de alimentación [4].
Debido a que el circuito adelanto-atraso provoca una ganancia de 1/3 en el voltaje de salida, se requiere una ganancia mayor a 3 para ir incrementando la señal de salida hasta saturar el amplificador operacional.
Existe una configuración con diodos Zener, los cuales permiten usar resistencias en un acomodo para obtener una ganancia

mayor a 3; sin embargo, al llegar al voltaje Zener, estos diodos permiten el paso de la corriente a través de ellos (evadiendo la R3) provocando que la ganancia se mantenga en 3 (R1 = 2*R2) generado una ganancia autoajustable [4].

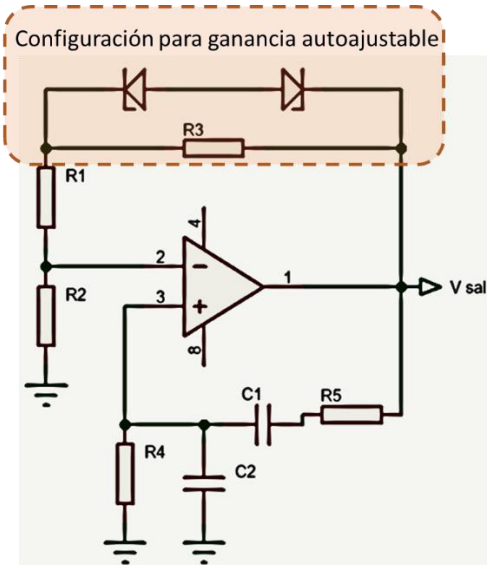


Figura 7. Oscilador de puente de Wien con ganancia autoajustable.

Donde:

$$Ganancia = 3 + \frac{R3}{R2} \; y \; R1 = 2 \cdot R2$$

II. OBJETIVOS

- ✓ Conocer los conceptos básicos sobre el LVDT y construir uno (se recomienda que los componentes se suelden sobre una placa).
- ✓ Realizar una interfaz gráfica con un contenedor con pestañas, la cual deberá tener una pestaña para almacenar todos los elementos relacionados con la comunicación serie, otra para visualizar los datos del LVDT y otra para guardar los datos en un documento de texto.

III. ELABORACIÓN DE LAS BOBINAS

1) Material

- ➢ Alambre magneto #23 AWG.
- ➢ 3 carretes con 2.1 cm de diámetro interno y 5 cm de largo.
- ➢ 1 Encendedor.
- ➢ Cinta de aislar.
- ➢ Alambre para conectar las bobinas a los demás componentes (se deja al criterio del alumno).
- ➢ Material para soldar componentes electrónicos (cautín, soldadura, etcétera).

2) Armado

Enrollar el alambre de cobre sobre los carretes hasta generar tres bobinas: la primaria de 500 vueltas y las secundarias de 400 vueltas cada una, después prender fuego sobre las puntas del alambre para quitarle el esmalte y finalmente usar el alambre, soldadura y cinta de aislar para conectar los embobinados al circuito correspondiente.

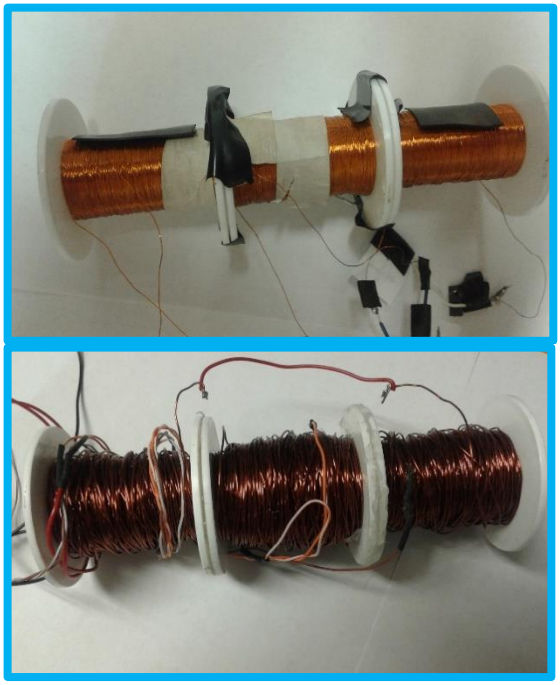


Figura 8. Ejemplos de embobinados de LVDT.

IV. LVDT CON EXCITACIÓN DE TRANSFORMADOR

1) Materiales

- ➢ Transformador de 127 V AC a 12 V AC y 300 mA.
- ➢ 1 Puente de diodos 2W10 [2A].
- ➢ 1 Capacitor de 10 uF [10V].
- ➢ 1 resistor de 100 Ω 1/4 W
- ➢ 1 Clavija.
- ➢ Alambre, cables banana-caimán y material para soldar.
- ➢ Varilla de hierro con un diámetro menor que el carrete y una longitud de 15 cm.
- ➢ Borneras y desarmadores.
- ➢ Multímetro.
- ➢ Osciloscopio.
- ➢ Tarjeta Arduino UNO
- ➢ Equipo de cómputo con el IDE de Arduino y el software LabVIEW

2) Conexión

Conectar el transformador reductor a la red eléctrica, este sirve para bajar la tensión de la red eléctrica de 127 V AC a 12 V AC. Utilizar la salida del transformador para excitar el embobinado primario. Conectar los embobinados secundarios en modo sustractivo para que se resten sus tensiones. Integrar el puente rectificador para obtener una señal DC y a la salida conectar en paralelo el capacitor y el resistor, con lo cual ya se puede usar el Arduino para tomar datos.

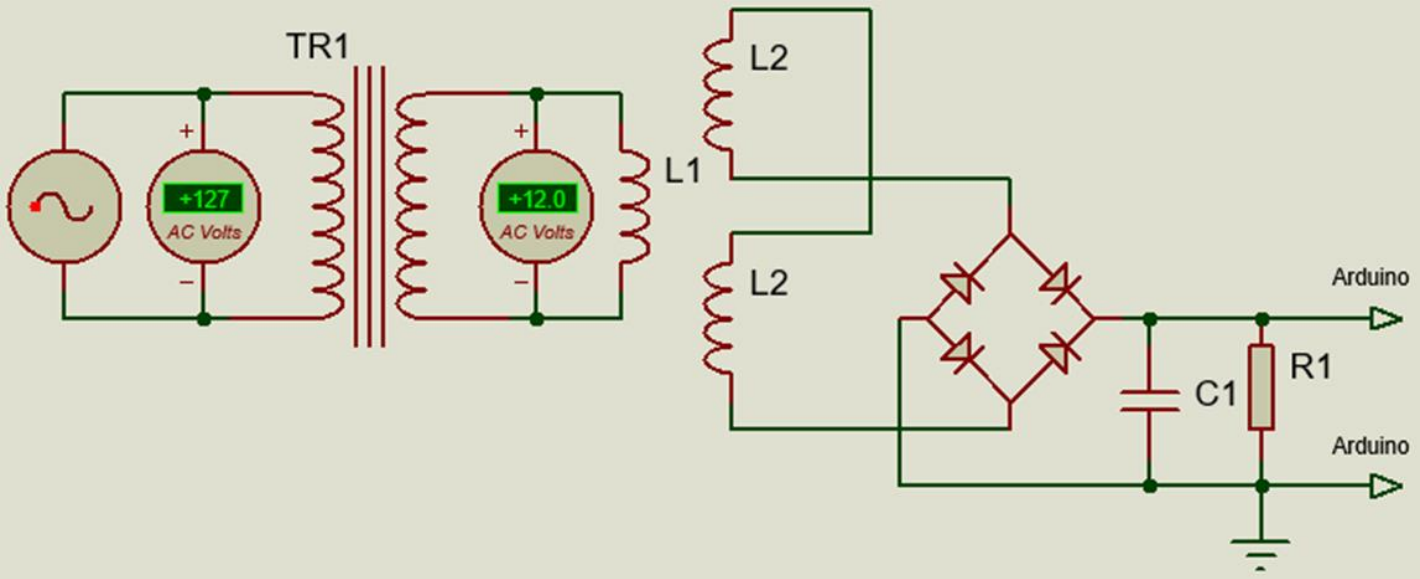


Figura 9. LVDT con excitación de transformador.

Se recomienda usar un multímetro para asegurarse que el LVDT genera un cambio al desplazarse el núcleo ferromagnético dentro de él y que la salida máxima no sea mayor a 5V, en caso que lo sea se deberá agregar un divisor de voltaje o una etapa de potencia antes del Arduino (se deja al criterio del alumno). Una vez que esté comprobado que el LVDT funciona, realizar las siguientes conexiones con el Arduino:

- Puentear las tierras
- salida del LVDT → A0

V. LVDT CON EXCITACIÓN DE FUENTE DC

1) Materiales

- 2 amplificadores operacionales tl081 o tl082.
- 2 diodos Zener 1N4728A.
- 2 capacitores cerámicos de 0.1 uF.
- 1 capacitor cerámico de 220 nF.
- 1 capacitor electrolítico de 2200 uF [10 V].
- 1 puente de diodos.
- 4 resistencias de 10 kΩ 1/4 W.
- 1 resistencia de 680 Ω 1/4 W.
- 1 resistencia de 100 Ω 1/4 W.
- 1 Diodo 1N4148.
- Alambre para unir las bobinas a los demás componentes (se deja al criterio del alumno) y material para soldar componentes electrónicos.
- Transistor TIP 31.
- Transistor TIP 32.
- Borneras y desarmadores.
- Varilla de hierro con un diámetro menor que el carrete con una longitud de 15 cm.
- Fuente DC bipolar de ±12 V.
- Multímetro.
- Osciloscopio.
- Tarjeta Arduino UNO.
- Equipo de cómputo con el IDE de Arduino y el software LabVIEW.

2) Conexión

Para la construcción de este modelo, se tiene que seguir la siguiente metodología:

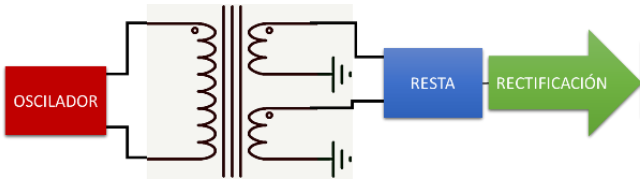


Figura 10. Metodología para armar el LVDT con excitación de fuente DC.

Primero construir el oscilador de Wien de acuerdo con la figura 7, donde R1=20 kΩ (2 resistores de 10 kΩ en serie), R2=10 kΩ y R3=680 Ω. Se usan 2 capacitores electrolíticos de 100 nF y 2 resistencias de 2.7 kΩ para obtener una frecuencia de 589.46 Hz. Se realizan las conexiones mostradas en la figura 11. Se conectan los embobinados secundarios en modo sustractivo para restar sus señales y finalmente se agrega una etapa de rectificación.

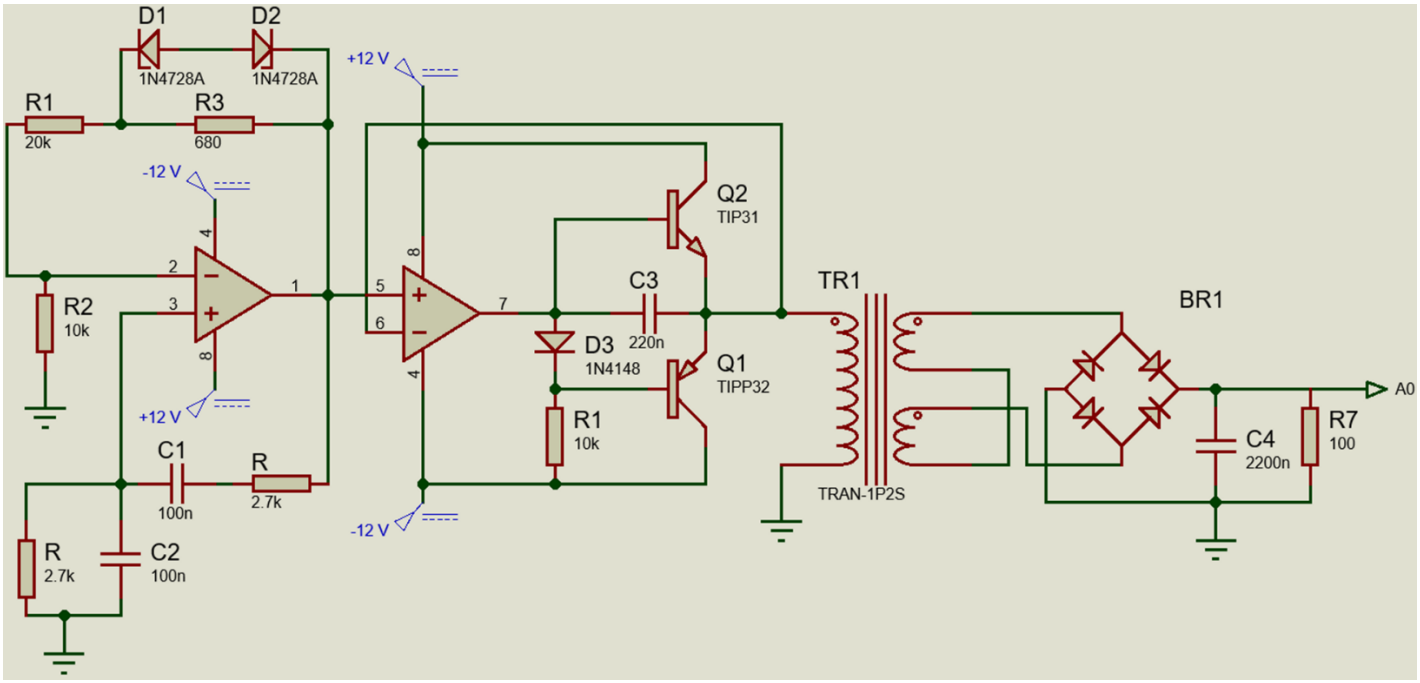


Figura 11. Circuito completo del LVDT con excitación de puente DC.

Conectar las terminales de alimentación de los amplificadores operacionales a una fuente de 12 V DC. Hay que tomar en cuenta que la corriente máxima soportada por los amplificadores operacionales TL081 son 15 mA. En caso de utilizar una fuente que exceda la corriente soportada por los amplificadores se recomienda utilizar un divisor de corriente.

Finamente conectar la salida del LVDT a la terminal A0 del Arduino y puentear las tierras.

VI. DISEÑO DE LA INTERFAZ GRÁFICA

A. Código de Arduino

Generar el programa de Arduino:

```
void setup() {
Serial.begin(9600);
}
void loop() {
Serial.println(analogRead(A0));
delay(100);
}
```

Este programa manda la tensión de salida del LVDT a través del puerto serial.

B. Adquisición de datos

El programa de LabVIEW recibe los datos del Arduino a través de la comunicación serial y posteriormente elimina el parpadeo (ver "Introducción al entorno de desarrollo de LabVIEW"). Dentro del programa de LabVIEW se han incluido los bloques para transformar el valor de la tensión a una unidad de longitud utilizando el valor de la tensión máxima y la longitud del LVDT. Para esta conversión se utiliza lo que coloquialmente se conoce como "regla de tres".

Tensión máximo =distancia máxima
Tensión de entrada = distancia actual
Por lo tanto:
Distancia actual = (tensión de entrada * distancia máxima)/ (tensión máxima)
La distancia máxima es la mitad del largo del LVDT. Para conocer la tensión máxima es necesario observar la entrada sin parpadeo y el valor máximo mostrado en el indicador corresponde a la tensión máxima.

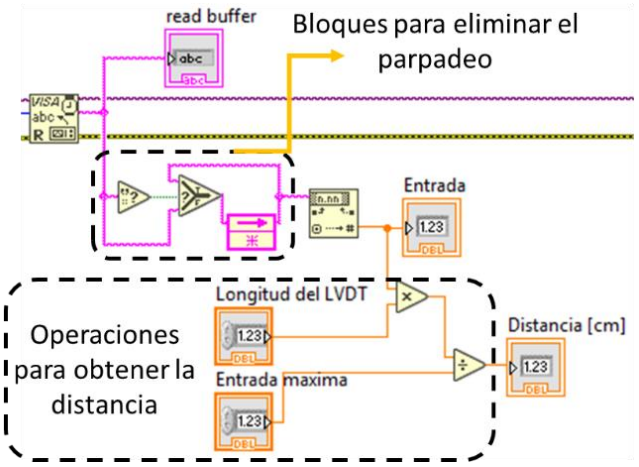


Figura 9. Eliminación del parpadeo y operaciones para traducir el voltaje a distancia.

Finalmente se agregan los bloques para guardar los datos dentro de un documento de texto (ver "Introducción al entorno de desarrollo de LabVIEW").

C. Diseño final

A continuación se muestra el diagrama de bloques completo.

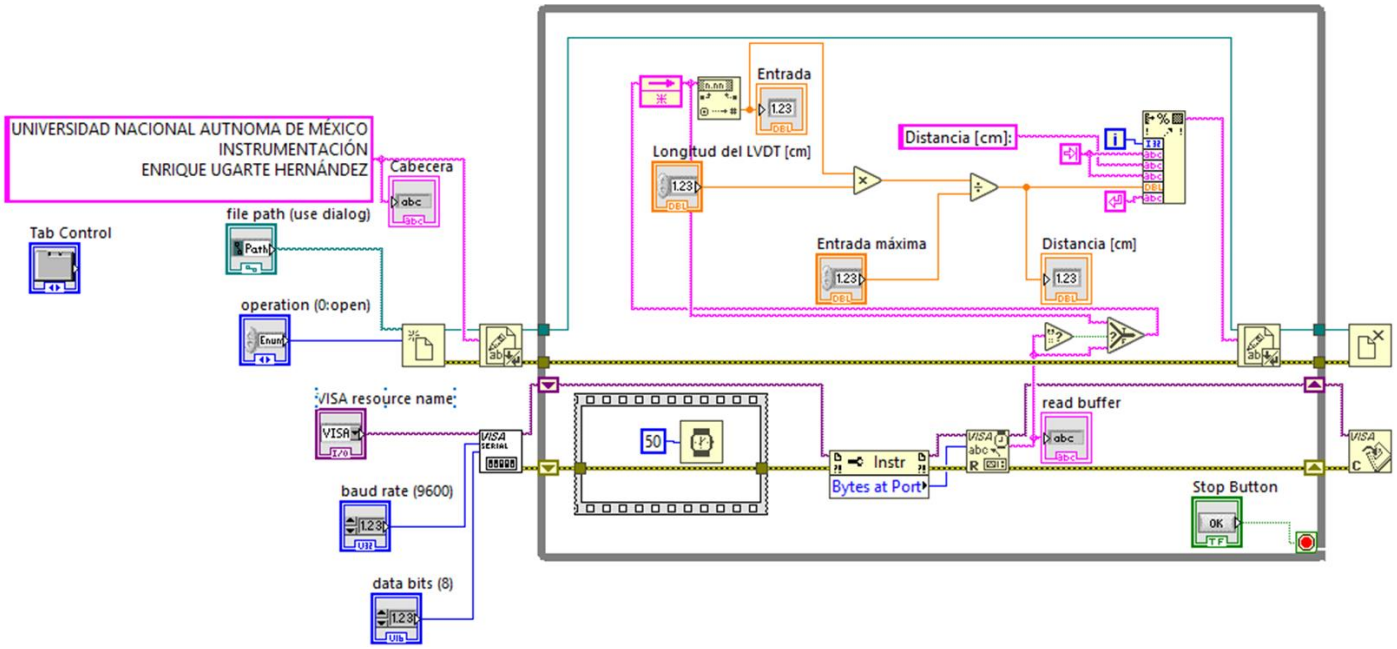


Figura 11. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña. Los controles e indicadores numéricos para conocer el desplazamiento del núcleo del LVDT se encuentran en la segunda pestaña y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

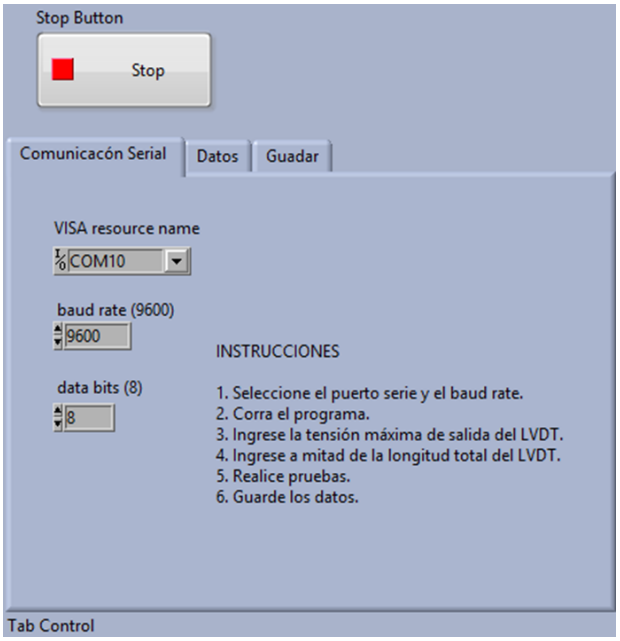


Figura 12. Primera pestaña de la interfaz.

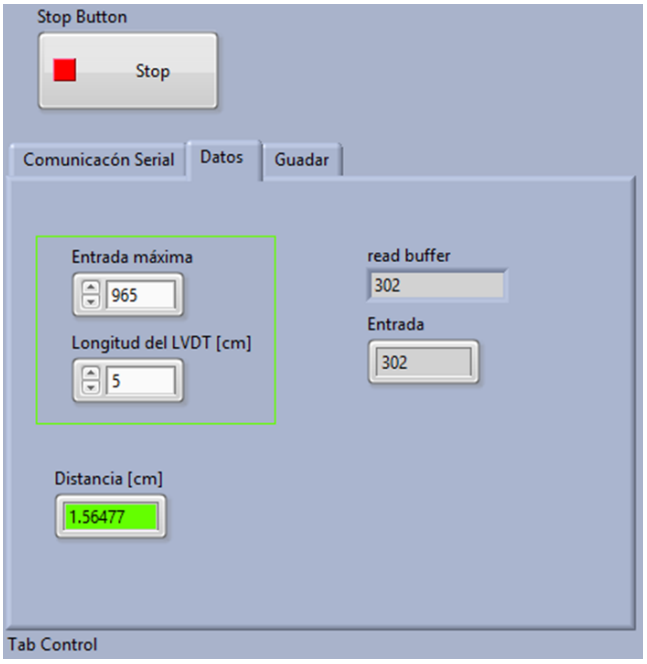


Figura 13. Segunda pestaña de la interfaz.

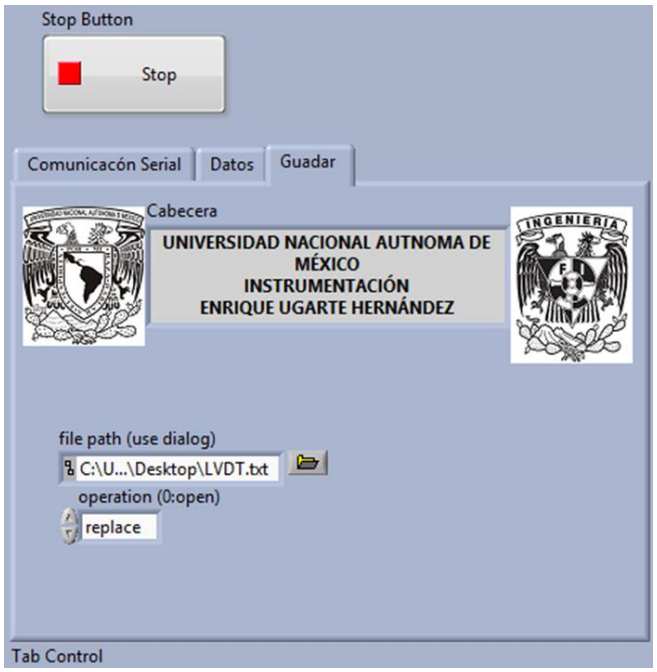


Figura 14. Tercera pestaña de la interfaz.

Figura 15. Ejemplo de LVDT con excitación de transformador.

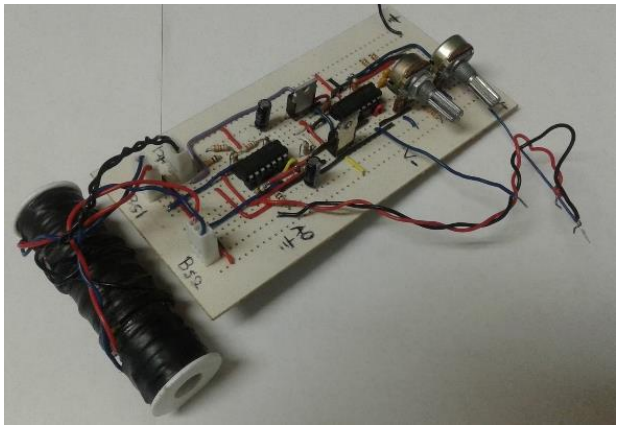
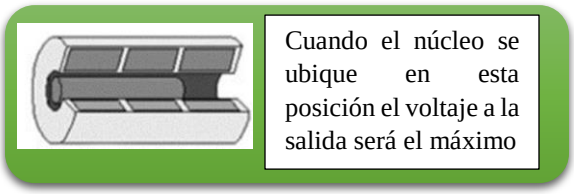


Figura 16. Ejemplo de LVDT con excitación de fuente DC.

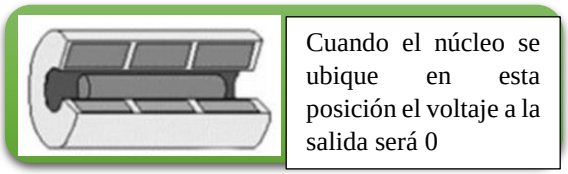
VII. PRUEBAS

Al deslizar un núcleo hay 3 posibilidades [6]:

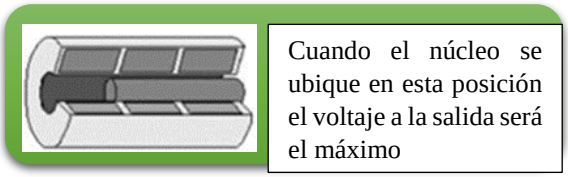
- Núcleo deslizado a la izquierda



- Núcleo en posición central



- Núcleo desplazado a la derecha



Con esta configuración, al comenzar a ingresar la varilla en la interfaz, la distancia irá incrementando hasta llegar a la distancia máxima en la mitad del LVDT.

VIII. FUENTES DE CONSULTA

[1] Granda, M., Mercedes y Mediavilla B., Elena. (2010). Instrumentación electrónica: Transductores y acondicionadores de señal. Cantabria, España. Editorial Universidad de Cantabria.

[2] KEYENCE CORPORATION (2018). ¿Qué es un sensor de proximidad inductivos? .Recuperado de https://www.keyence.com.mx/ss/products/sensor/sensorbasics/proximity/info/ .

[3] (2015). File:Lvdt how.gif. *Wikipedia*. Recuperado de https://commons.wikimedia.org/wiki/File:Lvdt_how.gif .

[4] Floyd, Thomas L. (2008). Dispositivos electrónicos. Editorial PERSON Prentice Hall, México, Octava Edición.

[5] Joken (2011). Cómo construir un LVDT y su driver analógico. *j0k3n.com*. Recuperado de http://j0k3n.com/hardware/electronica/lvdt-driver/ .

[6] Ingeniería mecafenix (2018). Sensor LVDT (Transformador diferencial de variación lineal). *Ingmecafenix.com*. Recuperado de http://www.ingmecafenix.com/automatizacion/lvdt/ .

Práctica 5. Sensor ultrasónico

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En este trabajo se explica cómo utilizar el sensor ultrasónico de manera práctica, el cual sirve para medir la distancia entre algún objeto y el sensor; utilizando la tarjeta Arduino UNO para la adquisición de datos y el entorno de desarrollo de LabVIEW para la generación de una interfaz gráfica para visualizar los datos de manera clara para el usuario, valiéndose de la comunicación serial para la comunicación entre el Arduino y LabVIEW.**

I. INTRODUCCIÓN

A. Sensores ultrasónicos

Son muy utilizados para medir distancia en línea recta, tienen la ventaja de no ser invasivos, es decir, que no requiere contacto alguno para realizar la medida [1]. Para generar la señal ultrasónica, se excita el material piezoeléctrico repetidas veces generando una onda cíclica de alta frecuencia y corta duración, al chocar con alguna superficie, esta regresa en forma de eco al receptor, que al ser de un material piezoeléctrico convierte la onda en una señal eléctrica. La onda debe ser de corta duración, ya que de no serlo el sensor recibiría eco continuamente, lo que provocaría que se pierda precisión.

Cabe mencionar que la velocidad de la onda de propagación depende de las condiciones del entorno en que se desenvuelva [1].

Velocidad del viento en gases:

$$v = \sqrt{\frac{\gamma \cdot P}{\rho}}$$

Donde:

$v = Velocidad\ del\ viento\ \left[\frac{m}{s}\right]$

$\gamma = \ Coeficiente\ de\ dilatación\ adiabática$

$P = Presión\ [Pa]$

$\rho = Densidad\ \left[\frac{kg}{m^3}\right]$

Dentro de la atmósfera terrestre, con una temperatura de 20 °C, 50% de humedad y a nivel del mar, la velocidad del sonido es 343.2 m/s. Al no contar con estas condiciones específicas siempre va a existir un ligero error, el cual es despreciable, a menos que se esté realizando un trabajo que exija una alta precisión.

Velocidad del viento en sólidos:

$$v = \sqrt{\frac{E}{\rho}}$$

Donde:

$E = Módulo\ de\ Young\ [Pa]$

Velocidad del viento en líquidos:

$$v = \sqrt{\frac{K}{\rho}}$$

Donde:

$K = Módulo\ de\ compresibilidad\ [Pa]$

En agua salada, el sonido viaja a aproximadamente a 1500 m/s y en agua dulce a 1435 m/s. Estas velocidades varían según la presión, temperatura y salinidad [2].

B. Características y funcionamiento del sensor

El sensor ultrasónico HC-SR04 permite medir distancia a partir de 2 [cm] y hasta 4 [m] de longitud [3]. Este se puede ocupar para medir el nivel del agua, se alimenta con 5 [V], tiene 4 terminales: 2 para alimentación, uno llamado *Trigger* con el cual al enviar un pulso de al menos 10 [us] enviará 1 pulso ultrasónico de 40 [kHz] colocando la salida echo en alto, esta se mantendrá en este estado hasta que la señal ultrasónica regrese. El tiempo que dure el pin *echo* en alto es proporcional a la distancia [4].

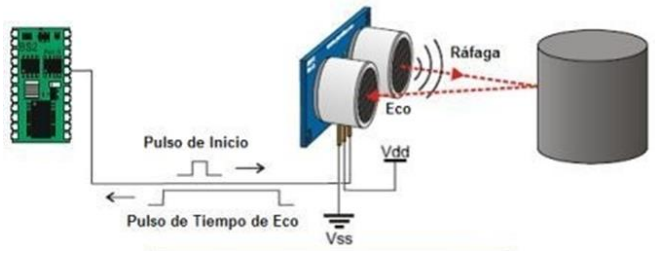


Figura 1. Funcionamiento del sensor ultrasónico [5].

II. OBJETIVOS

- ✓ Aprender a utilizar de manera práctica el sensor ultrasónico.
- ✓ Desarrollar una interfaz en LabVIEW que permita visualizar los datos recibidos por el sensor utilizando un contenedor con pestañas. La aplicación tiene que ser clara para el usuario y debe permitir guardar los

datos en un documento de texto, además se debe poder ingresar la magnitud de la velocidad del sonido desde la interfaz gráfica.

III. MATERIAL

- Sensor ultrasónico HC-SR04.
- Tarjeta Arduino UNO.
- 4 Jumpers M-H.
- Computadora con el software IDE de Arduino y LabVIEW.

IV. DESARROLLO

A. Conexión

Realizar las siguientes conexiones:

- VCC → 5V.
- Trig → terminal 9.
- Echo → terminal 8.
- GND → GND.

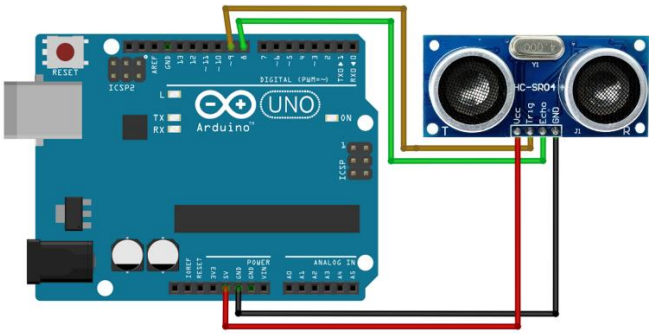


Figura 2. Conexión del sensor ultrasónico.

B. Código de Arduino

Desarrollamos el programa de Arduino que va a mandar el tiempo que dura el pin *echo* en alto a la interfaz mediante el puerto serial.

Programa de Arduino:

```
long tiempo;
int trig=9;
int echo=8;
void setup()
{
  Serial.begin(9600);
  pinMode(trig,OUTPUT);
  pinMode(echo,INPUT);
}
void loop()
{
  digitalWrite(trig,LOW);
  delayMicroseconds(5);
  digitalWrite(trig,HIGH);
  delayMicroseconds(10);
  tiempo=pulseIn(echo,HIGH);
  Serial.println(tiempo);
  delay(100);
}
```

Este programa envía el tiempo que dura la salida *echo* en alto al monitor serial para que posteriormente dentro del programa de LabVIEW se realicen los cálculos para obtener la distancia.

C. Adquisición de datos

El programa de LabVIEW obtiene el tiempo del puerto serial. El programa cuenta con unos bloques que eliminan el parpadeo de la entrada (ver "Introducción al entorno de desarrollo de LabVIEW"). Para obtener la distancia es necesario multiplicar el tiempo por la velocidad del sonido entre dos. Hay que recordar que el tiempo está en microsegundos, así que se debe tener cuidado con el manejo de las unidades. Para la realización de la práctica se tomó la velocidad del viento como $0.034\ x10^{4}$[m/s]. Se utiliza un indicador numérico y un *Tank* para visualizar la distancia.

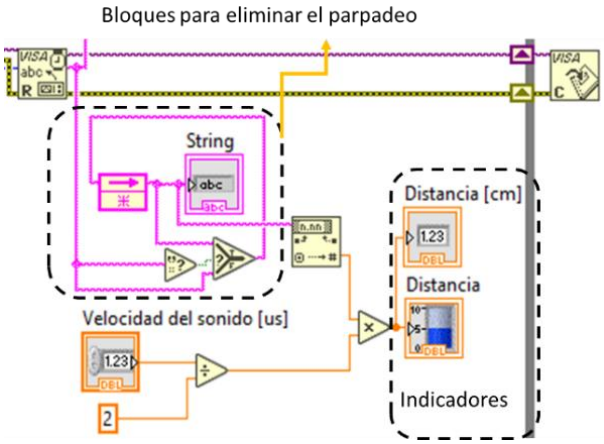


Figura 3. Eliminación del parpadeo e indicadores de distancia.

D. Almacenamiento de datos

Los bloques para guardar los datos son: *Open/Create/Replace File*, *Close File* y *Write Text File*, estos se encuentran en *Programming* → *File I/O*, se utiliza el bloque: *Format Into String* para acomodar la información de manera ordenada, este se encuentra *en Programming* → *String*.

E. Diseño final

A continuación se muestra el diagrama de bloques completo.

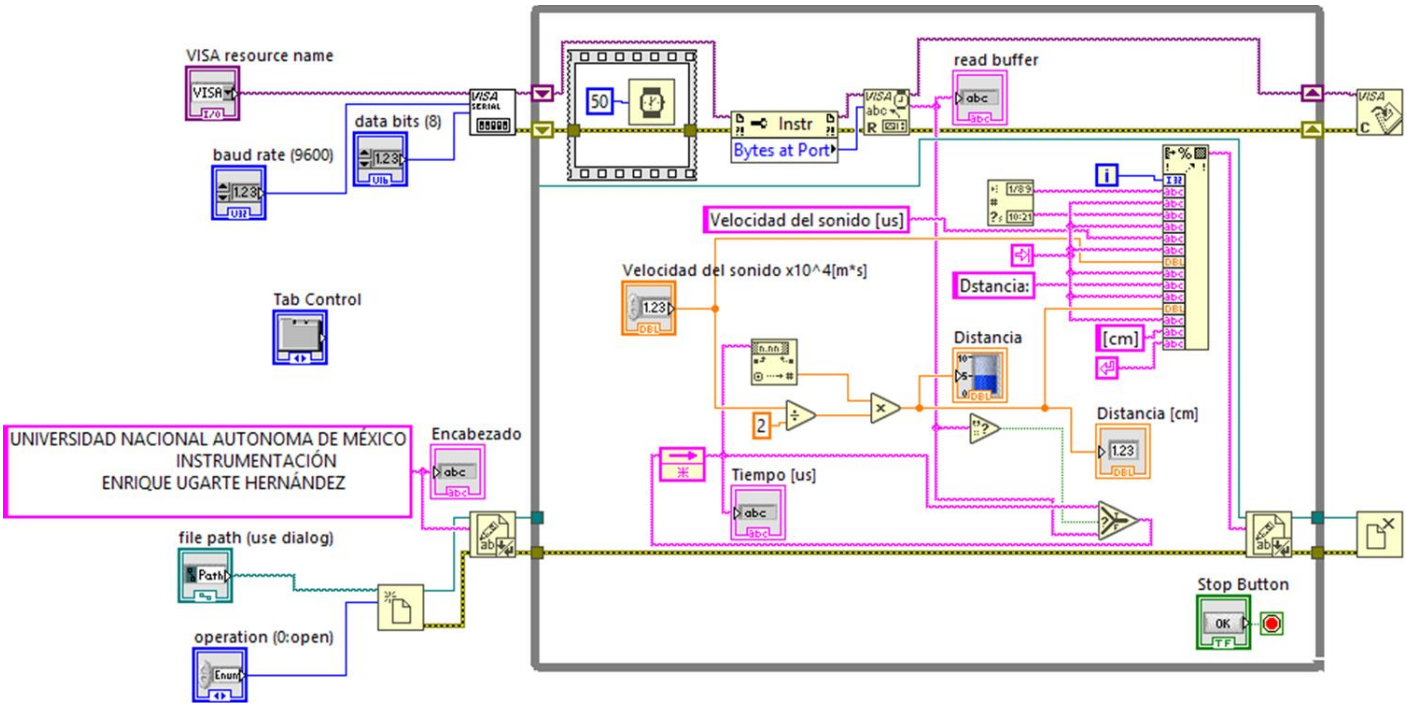


Figura 4. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña, la visualización de los datos y el control numérico para ingresar la velocidad del sonido en la segunda y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

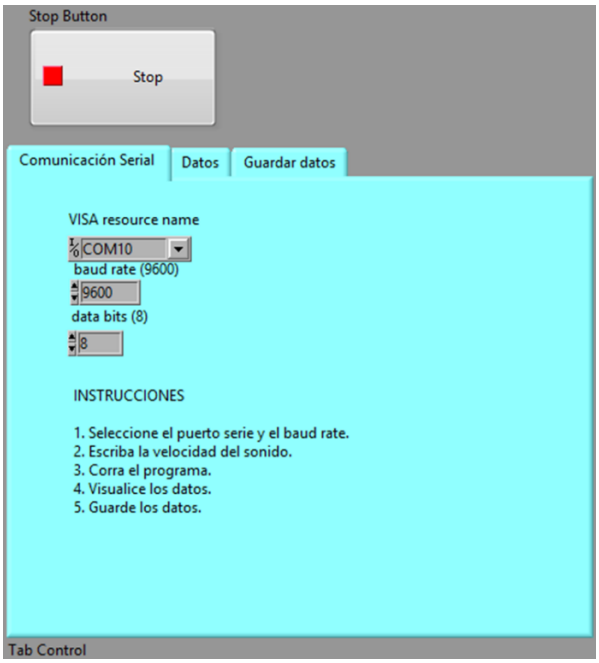


Figura 5. Primera página del contenedor con pestañas.

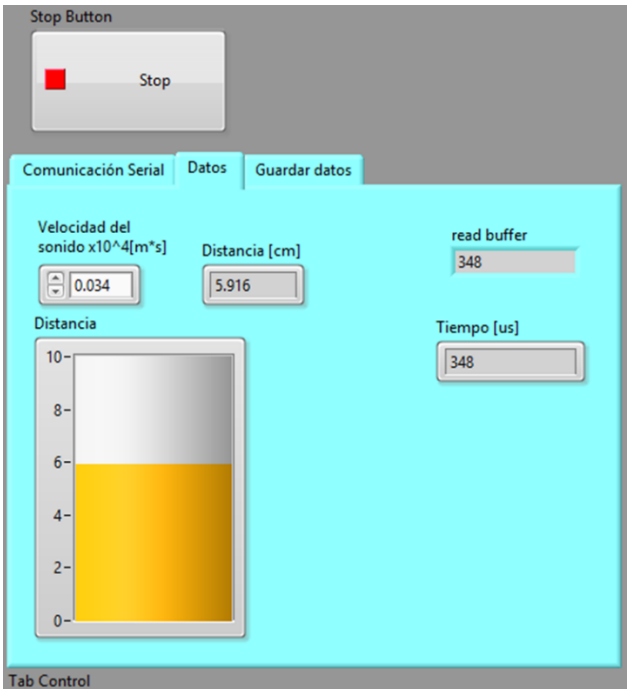


Figura 6. Segunda página del contenedor con pestañas.

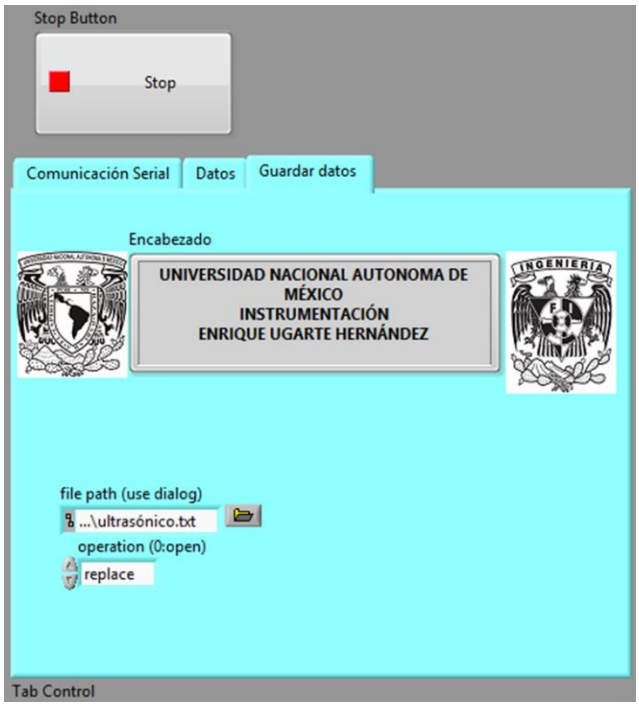


Figura 7. Tercera página del contenedor con pestañas.

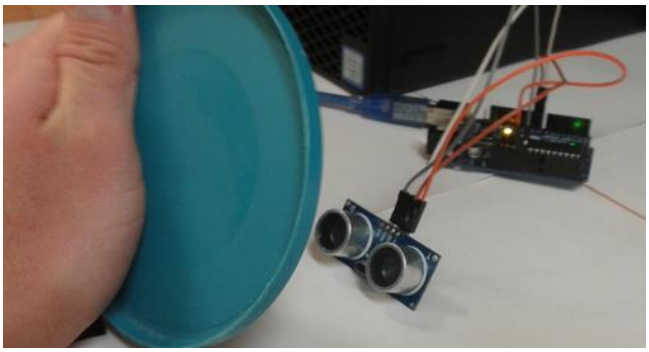


Figura 8. Pruebas con el sensor ultrasónico.

```
UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO
INSTRUMENTACIÓN
ENRIQUE UGARTE HERNÁNDEZ
0 08/10/2018     11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      0.000000       [cm]
1 08/10/2018     11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.242000       [cm]
2 08/10/2018     11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.259000       [cm]
3 08/10/2018     11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.276000       [cm]
4 08/10/2018     11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.293000       [cm]
5 08/10/2018     11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.310000       [cm]
6 08/10/2018     11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.327000       [cm]
7 08/10/2018     11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.344000       [cm]
8 08/10/2018     11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.361000       [cm]
9 08/10/2018     11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.378000       [cm]
10 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.395000       [cm]
11 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.412000       [cm]
12 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.429000       [cm]
13 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.446000       [cm]
14 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.463000       [cm]
15 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.480000       [cm]
16 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.497000       [cm]
17 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.514000       [cm]
18 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.531000       [cm]
19 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.548000       [cm]
20 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.565000       [cm]
21 08/10/2018    11:52 a. m.     Velocidad del sonido x10^4[m*s]        0.034000       Distancia:      7.582000       [cm]
```

Figura 9. Datos dentro del documento de texto.

V. FUENTES DE CONSULTA

[1] Corona R., L. G., Abarca J., G.S. y Mares C., J., (2015), *Sensores y Actuadores. Aplicaciones con Arduino*, Azcapotzalco, México D.F. (ahora Cuidad de México), Grupo Editorial PATRIA.

[2] Wikipedia (2018). Velocidad del sonido. *Wikipedia.org*. Recuperado de https://es.wikipedia.org/wiki/Velocidad_del_sonido .

[3] Cana, Julio. (2016). Sensor HC-SR04 para crear una alarma con Arduino. *Hetpro*. Recuperado de https://hetpro-store.com/TUTORIALES/sensor-hc-sr04/ .

[4] Soria, Kevin. (2013). Todo lo que tienes que saber sobre: HC-SR04 Sensor Ultrasónico. *♣BKAR♣ELECTRONICA♣ blog de Kevin*. Recuperado de http://bkargado.blogspot.mx/2013/09/todosobrehc-sr04.html .

[5] Latorre, Sebastian. (2015). [Arduino 08] Medir distancia con sensor ultrasónico HR-SR04. Seba Electronic LABs. Recuerado de http://sebalabs.blogspot.mx/2015/09/arduino-08-medir-distancia-con-sensor.html .

Práctica 6. Sensor de temperatura a prueba de agua

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En este trabajo se explica de forma práctica como utilizar el sensor de temperatura a prueba de agua utilizando la tarjeta Arduino UNO para la adquisición de datos y el entorno de desarrollo de LabVIEW para la generación de una interfaz gráfica que permita la visualización de la información de manera clara para el usuario.**

I. INTRODUCCIÓN

A. Sensores térmicos

La temperatura es probablemente la variable física que con mayor frecuencia se mide y controla en los procesos industriales. Un transductor de temperatura produce una señal eléctrica, ya sea una intensidad de corriente, diferencia de potencial o una impedancia en función de la temperatura en la que se encuentra. Los transductores de temperatura se pueden clasificar en cinco grupos principales:

1) Bimetales (sensores de temperatura)

Son piezas formadas por dos metales con diferente coeficiente de dilatación térmica unidos, la pieza se deforma en función de la temperatura.

2) Detectores de temperatura resistivos (RTD) (También llamados termistores)

Se basan en la variación de la resistencia en función de la temperatura.

3) Dispositivos termoeléctricos (termopares)

Son los más comunes en la medida de temperatura, estos generan una señal eléctrica en función de la temperatura sin necesidad de alguna alimentación eléctrica.

4) Transductores de temperatura integrados (basados en diodos y transistores bipolares)

Es la combinación de un transductor de temperatura y los circuitos de acondicionamiento de la señal [1].

B. Funcionamiento y modo de uso del sensor DS18B20

El sensor de temperatura a prueba de agua utiliza el protocolo de comunicación *OneWire*, el cual permite mandar y recibir datos utilizando solamente un cable. Normalmente este sensor viene blindado, lo que significa que se encuentra cubierto para protegerse del agua. Se recomienda utilizar una resistencia de 5 kΩ; sin embargo, con un resistor de 4.7 kΩ funciona perfectamente. La manera más fácil de utilizarlo es con sus dos bibliotecas.

II. OBJETIVOS

- ✓ Aprender a utilizar de manera práctica el sensor de tempera a prueba de agua.
- ✓ Desarrollar una aplicación en LabVIEW que permita visualizar los datos obtenidos utilizando un contenedor con pestañas separando los componentes dentro de las pestañas de forma ordenada. La aplicación debe poder guardar los datos dentro de un documento de texto.

III. MATERIAL

- ➢ Tarjeta Arduino UNO.
- ➢ Sensor de temperatura DS18B20.
- ➢ 1 Resistor de 4.7 kΩ 1/4 W.
- ➢ Material para soldar circuitos eléctricos (cautín, soldadura, etcétera).
- ➢ Alambre para conectar el sensor a los demás componentes (se deja al criterio del alumno).
- ➢ Thermofit o cinta de aislar.
- ➢ Computadora con el software IDE de Arduino y LabVIEW.

IV. DESARROLLO

A. Montaje del sensor

Montar el sensor dentro de algún recipiente con agua, cuya temperatura se pueda variar. Se deja a criterio del alumno los materiales a utilizar.

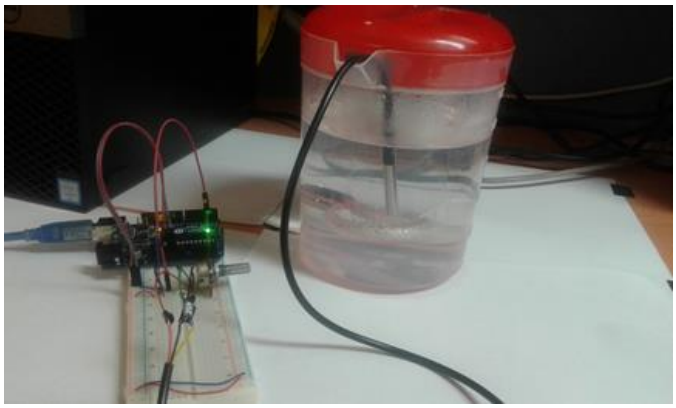


Figura 1. Montaje del sensor dentro de un recipiente. El recipiente mostrado tiene integrado una resistencia, la cual al conectarse a la red eléctrica incrementa la temperatura del líquido generado pequeñas burbujas en el interior.

B. Conexión

Figura 2. Terminales del sensor DS18B20

Armar el circuito; a continuación se muestra la conexión clásica:

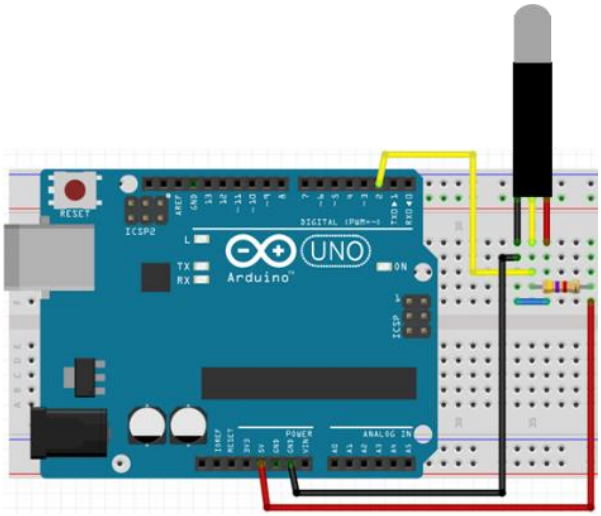


Figura 2. Conexión clásica del sensor de temperatura a prueba de agua.

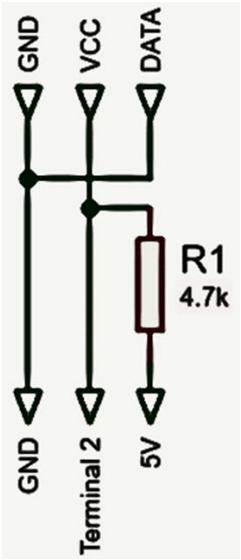


Figura 3. Esquema de conexión.

Como se puede observar, los pines VCC y GND deben ir conectados entre sí. La alimentación se introduce en el pin DATA por medio de una resistencia pull-up [2].

C. Código de Arduino

Entrar a los siguientes enlaces para descargar las dos bibliotecas [2]:

- DallasTemperature
- OneWire

Las agregamos al IDE de arduino, para esto, entramos en *Programa → Incluir Libreria → Añadir libreria ZIP* y seleccionamos la ubicación de las bibliotecas.

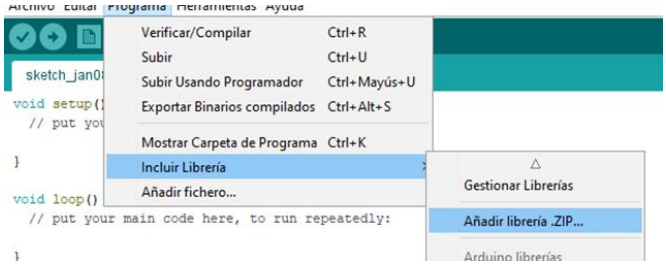


Figura 3. Procedimiento para añadir bibliotecas en el IDE de Arduino.

Ingresar el siguiente código [2]:

```
#include <OneWire.h> //Se importan las bibliotecas
#include <DallasTemperature.h>
#define Pin 2
OneWire ourWire(Pin); //Se establece el pin declarado como bus para la comunicación OneWire
DallasTemperature sensors(&ourWire); //Se instancia la librería DallasTemperature
void setup() {
Serial.begin(9600);
sensors.begin(); //Se inician los sensores
}
void loop() {
sensors.requestTemperatures(); //Prepara el sensor para la lectura
Serial.print(sensors.getTempCByIndex(0)); //Se lee e imprime la temperatura en grados Celsius
Serial.print(",");
Serial.print(sensors.getTempFByIndex(0)); //Se lee e imprime la temperatura en grados Fahrenheit
Serial.println();
delay(1000); //Se provoca un lapso de 1 segundo antes de la próxima lectura
}
```

Este programa manda la temperatura en grados Celsius y Fahrenheit a través del puerto serie, separados por una coma.

D. Adquisición de datos

El programa de LabVIEW obtiene las temperaturas en °C y °F del Arduino separadas por una coma. El programa cuenta con unos bloques que eliminan el parpadeo de la entrada (ver

"Introducción al entorno de desarrollo de LabVIEW"). Después del eliminar el parpadeo se separa la información, se convierten los datos tipo *String* a numéricos fraccionarios y finalmente se muestran en indicadores numéricos y del tipo *Thermometer*.

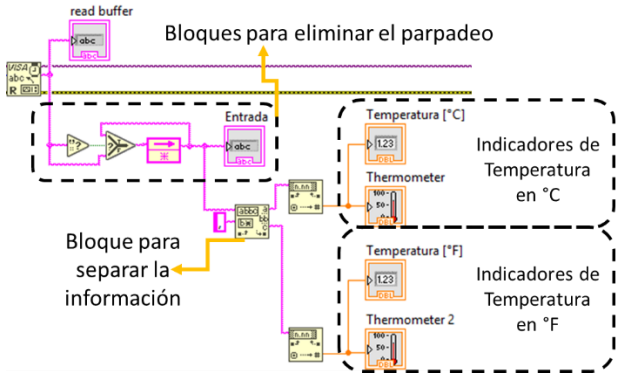


Figura 5. Eliminación del parpadeo e indicadores de Temperatura.

Finalmente se agregan los bloques para guardar los datos en un documento de texto (ver "Introducción al entorno de desarrollo de LabVIEW").

E. Diseño final

A continuación se muestra el diagrama de bloques completo.

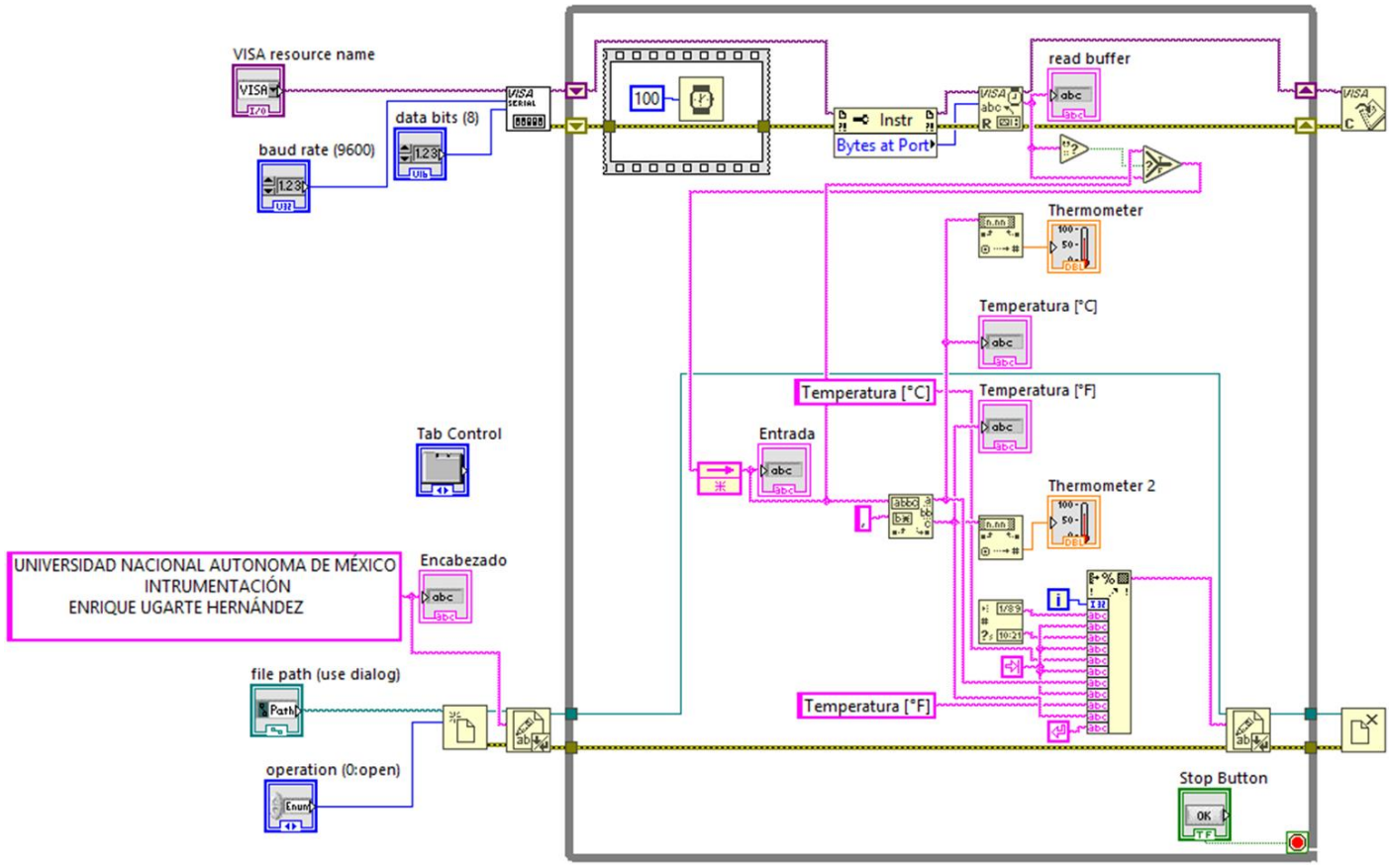


Figura 7. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña, en la segunda pestaña se muestran los datos de la temperatura en °C y °F y finalmente en la tercera pestana se encuentran los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

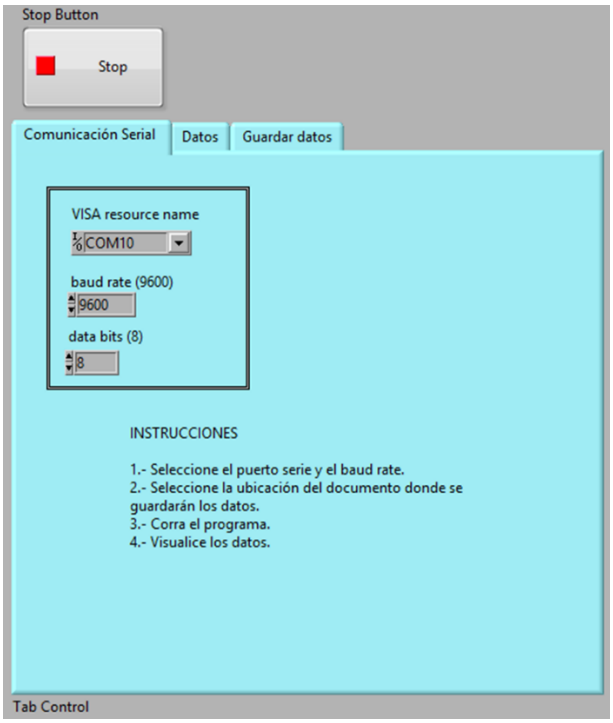


Figura 8. Primera página del contenedor con pestañas.

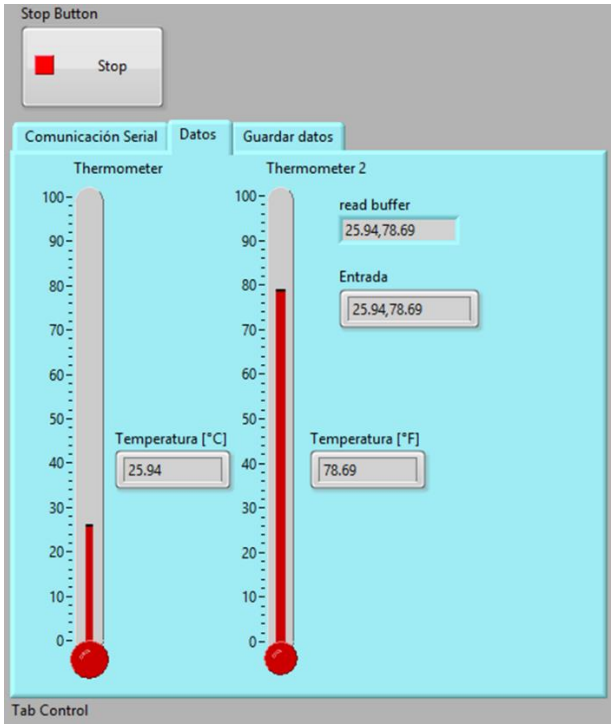


Figura 9. Segunda página del contenedor con pestañas.

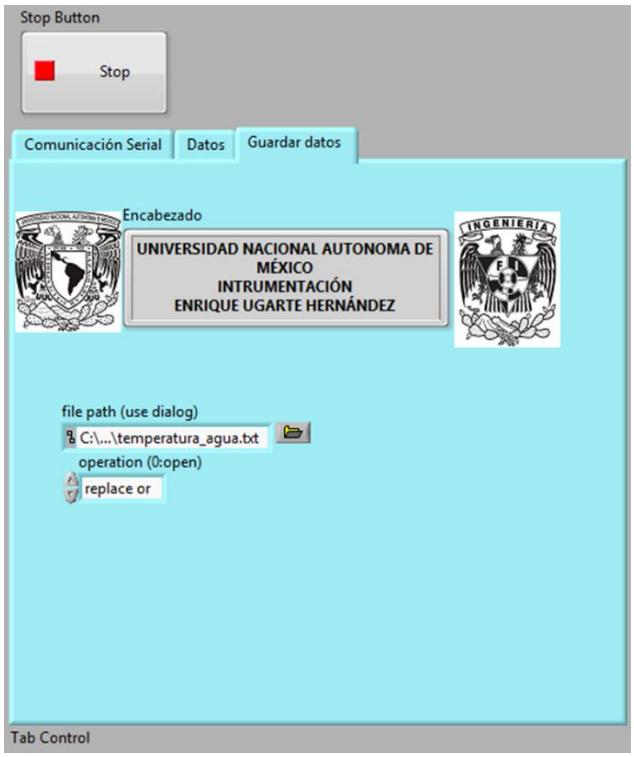


Figura 10. Tercera página del contenedor con pestañas.

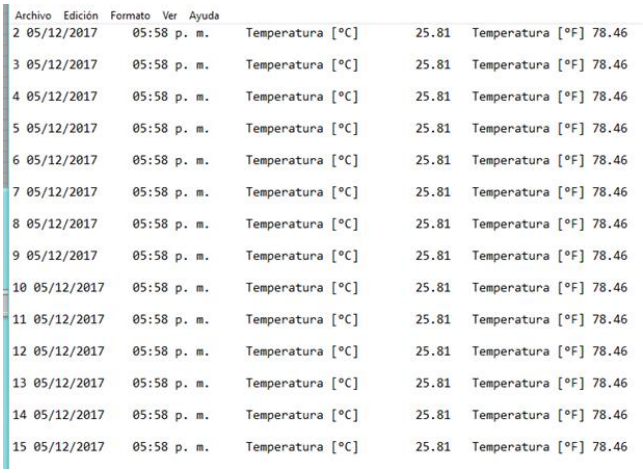


Figura 11. Datos dentro del documento de texto.

V. FUENTES DE CONSULTA

[1] Granda, M., Mercedes y Mediavilla B., Elena (2010). Instrumentación electrónica: Transductores y acondicionadores de señal. Cantabria, España. Editorial Universidad de Cantabria

[2] García G., Antony (2014). Aprendiendo a utilizar el sensor de temperatura DS18B20. Panamá. *PANAMAHITEK*. Recuperado de http://panamahitek.com/aprendiendo-utilizar-el-sensor-de-temperatura-ds18b20/ .

[3] Ramírez, Rolando (2015). Sensor de temperatura DS18B20 con Arduino. *HETPRO*. Recuperado de https://hetpro-store.com/TUTORIALES/sensor-de-temperatura-ds18b20/ .

Práctica 7. Celda de carga

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En este trabajo se explica de forma práctica como construir una báscula sencilla usando una celda de carga para generar una señal proporcional al peso de algún objeto valiéndose del efecto piezorresistivo de la celda de carga, posteriormente utilizando la tarjeta Arduino UNO para la adquisición de datos y el entorno de desarrollo de LabVIEW para generar una interfaz gráfica que permita calibrar la balanza desde la misma interfaz.**

I. INTRODUCCIÓN

A. Sensores piezorresistivos

El efecto piezorresistivo se refiere al fenómeno que ocurre en ciertos materiales que hace que cambie su resistencia cuando se ejerce alguna presión o esfuerzo normal sobre la superficie de dicho material, por lo general se ocupan galgas de manganina también llamadas galgas extensiométricas (aleación aproximadamente 84% Cu, 12% Mn y 4% Ni) para medir presiones muy altas (del orden de 1400 MPa), aunque también se usan elastómeros conductores y fibras de carbono [1, 2].

B. Galgas extensiométricas

Son un sensor utilizado para medir fuerza a través de su principio piezorresistivo. Para su uso es necesario fijar un extremo de la galga. Para acondicionar la señal de salida por lo general se ocupa el puente de Wheatstone. Existen comercialmente varias capacidades para la celda, desde 1 kg hasta 50 kg. Por lo general tienen una indicación que muestra la dirección en que hay que aplicar la fuerza para su correcto funcionamiento [3].

Figura 1. Galga extensiométrica con capacidad de 50 kg [4].

C. Driver HX711

Este dispositivo contiene un puente de Wheatstone, el cual al conectarse con la celda de carga y el microcontrolador, permite medir peso [5,6]. El puente de Wheatstone es un circuito eléctrico que se utiliza para medir resistencias eléctricas de valor desconocido [7].

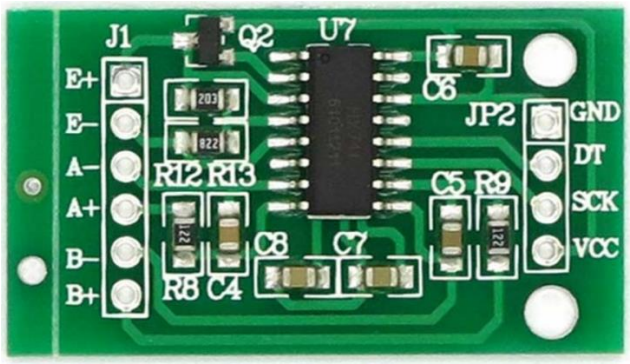


Figura 2. Driver HX711 [8].

II. OBJETIVOS

- ✓ Aprender a utilizar de forma práctica una celda de carga.
- ✓ Crear una báscula utilizando la celda de carga y el driver HX711.
- ✓ Diseñar una interfaz en LabVIEW para manejar la báscula. Esta debe utilizar el puerto serie para mandar y recibir datos. La interfaz debe ser generada dentro de un contenedor con varias pestañas, una para la comunicación serial, las necesarias para mandar, recibir datos y procesarlos y una para guardar la información. La interfaz tiene que ser clara para el usuario y debe ser capaz de calibrarse desde el mismo ambiente de LabVIEW.

III. MATERIALES

- ➢ Celda de carga de 1 kg de capacidad.
- ➢ Tarjeta Arduino UNO.
- ➢ Driver HX711.
- ➢ Material para fijar la celda de carga.
- ➢ Soldadura.
- ➢ Cautín.
- ➢ Pasta para soldar.
- ➢ Thermofit (opcional).
- ➢ Jumpers H-H y M-H.
- ➢ Headers hembra (opcional).
- ➢ Materiales para la construcción de la báscula.
- ➢ Computadora con el software IDE de Arduino y LabVIEW.

IV. DESARROLLO

A. Construcción de la balanza

Primero se arma la báscula. Se deja al criterio del alumno los materiales a utilizar. Es importante revisar la dirección en que se aplica la carga; la dirección está indicada en algún costado de la celda de carga.

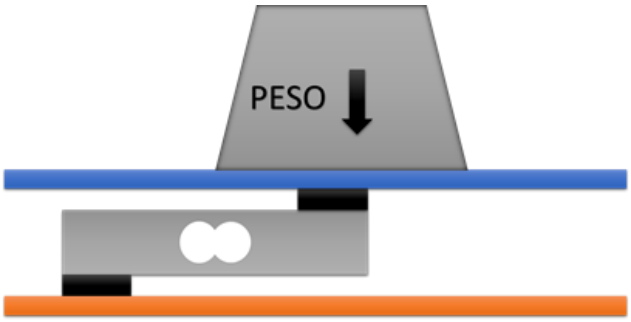


Figura 1. Esquema para la construcción de la báscula.

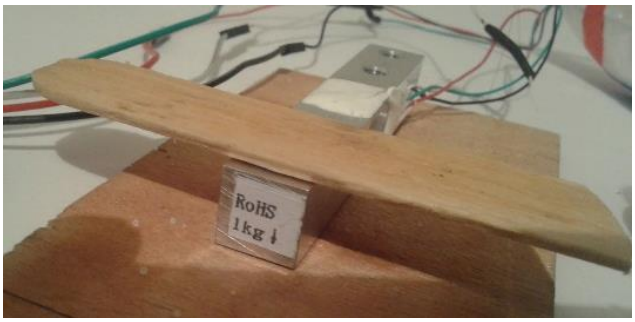


Figura 2. Ejemplo de báscula construida.

B. Conexión

Soldar los headers al driver HX711. Si se usan los headers macho que vienen incluido con el driver tendremos que usar jumpers H-H para realizar la conexión de la celda al driver. Se deja al criterio del alumno que headers y jumpers utilizar. Es recomendable soldar cable con un calibre más grueso a las terminales de la celda de carga para facilitar la conexión con el Arduino. Se puede utilizar thermofit para que los cables queden aislados de una manera más estética.

Realizar las conexiones correspondientes:

Celda de carga a driver:

- Cable rojo → E+.
- Cable negro → E-.
- Cable verde → A-.
- Cable blanco → A+.

Driver a Arduino:

- GND → GND.
- DT→ A1.
- SCK → A0.
- VCC → 5V.

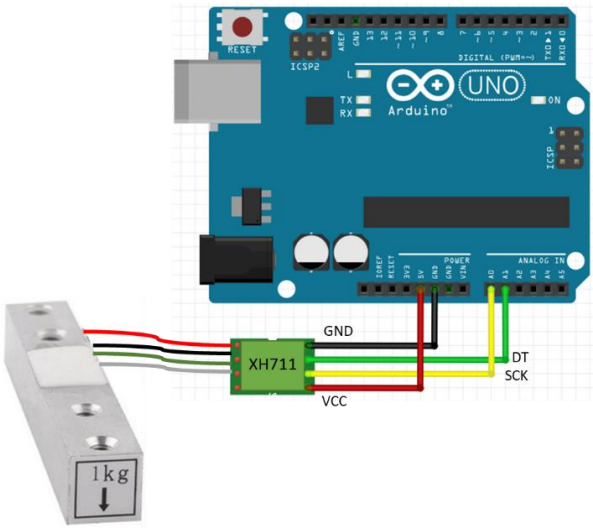


Figura 3. Ejemplo de conexión [3].

C. Generación del código de Arduino.

Descargar e instalar la siguiente biblioteca [9]:
https://github.com/bogde/HX711
Conectar el Arduino, abrimos el IDE e ingresar el código [3]:

```
#include "HX711.h"
#define DOUT  A1
#define CLK  A0
int a;
HX711 balanza(DOUT, CLK);
void setup() {
  Serial.begin(9600);
  Serial.println(balanza.read());
  balanza.set_scale(); //La escala por defecto es 1
  balanza.tare(20);  //El peso actual es considerado Tara.
}
void loop() {
a=Serial.read();
  Serial.println(balanza.get_value(a),0);    //el    0    indica    los
números después del punto decimal
  delay(100);
}
```

El código primero lee el valor que marca el Arduino cuando no hay peso sobre la galga, ese valor lo guarda y se lo resta a las siguientes mediciones y finalmente utiliza el comando balanza *get_value(n)* para tomar *n* número de muestras y obtener el promedio. El número de muestras se ingresa desde la interfaz.

D. Generación de la Interfaz gráfica

1) Manejo de datos

El programa manda y manda datos a través del puerto serie (ver "Introducción al entorno de desarrollo de LabVIEW"). Antes del bloque *VISA Write* hay un control numérico, este permite manipular el número de datos adquiridos por el Arduino, los cuales posteriormente se promedian dentro del código de Arduino para obtener un solo valor, el cual se muestra en *read buffer*.

Se requiere de un control numérico en el panel frontal que permita calibrar la balanza, este dato se multiplica por la entrada para obtener el peso real. Se recomienda que la unidad de medida sea el kilogramo. Finalmente se agregan blques para guardar los datos en un documento de texto (ver “Introducción al entorno de desarrollo de LabVIEW”).

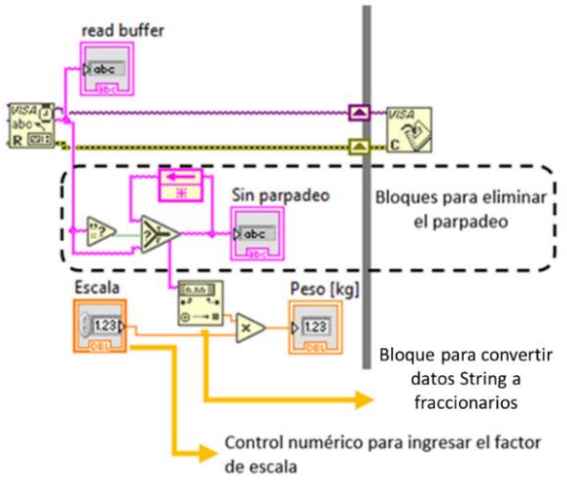


Figura 5. Eliminación del parpadeo y procesamiento de datos.

2) Diseño final

a) Diagrama de bloques final

A continuación se muestra el diagrama de bloques completo.

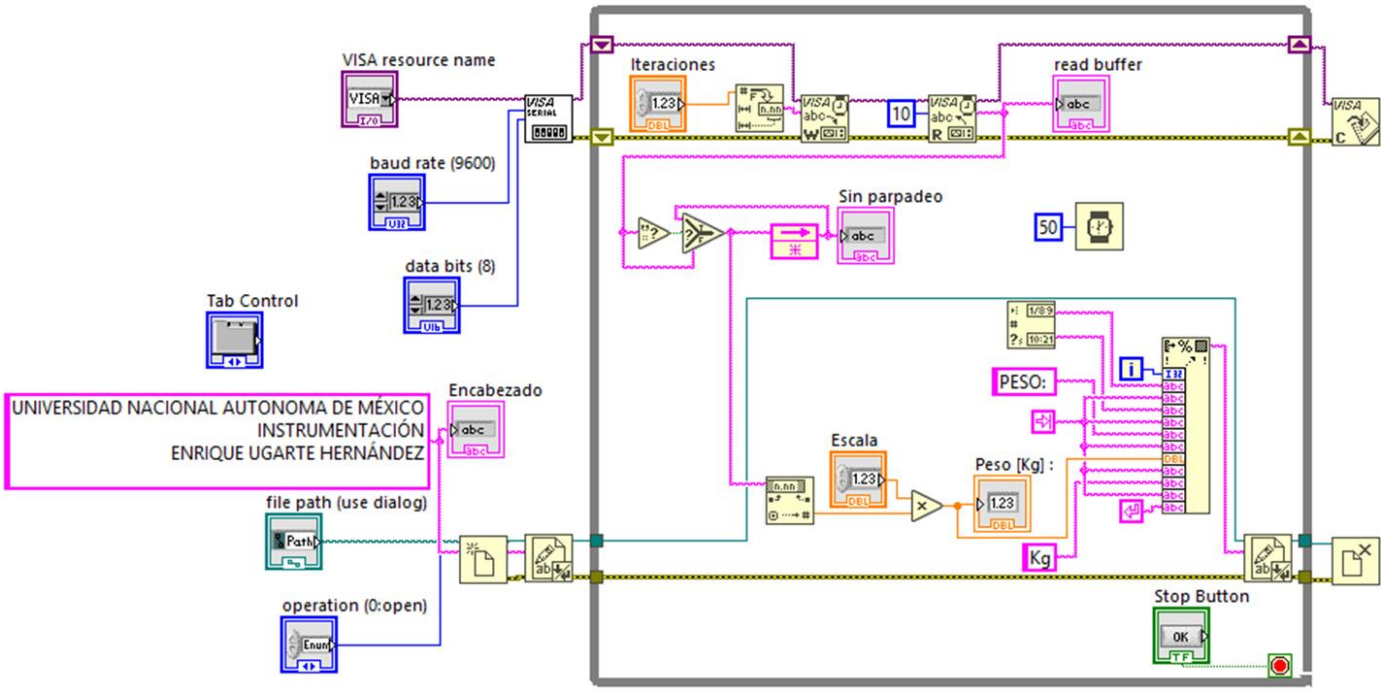


Figura 7. Diagrama de bloques completo.

b) Panel frontal

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña, la visualización de los datos en la segunda y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

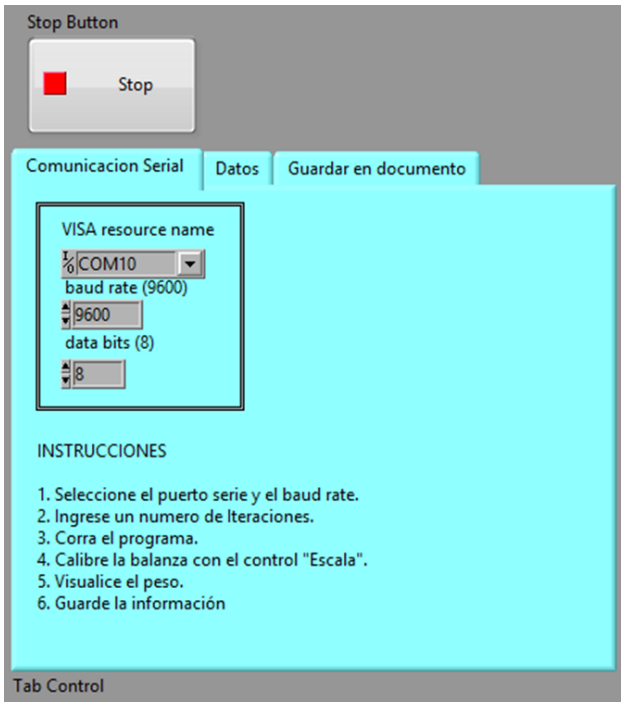


Figura 8. Primera página del contenedor con pestañas.

La segunda pestaña contiene dos controles numéricos, uno donde se ingresa el número de mediciones, las cuales se promedian para posteriormente mostrarse *en read buffer* y otra para ingresar el factor de escala.

Figura 9. Segunda página del contenedor con pestañas.

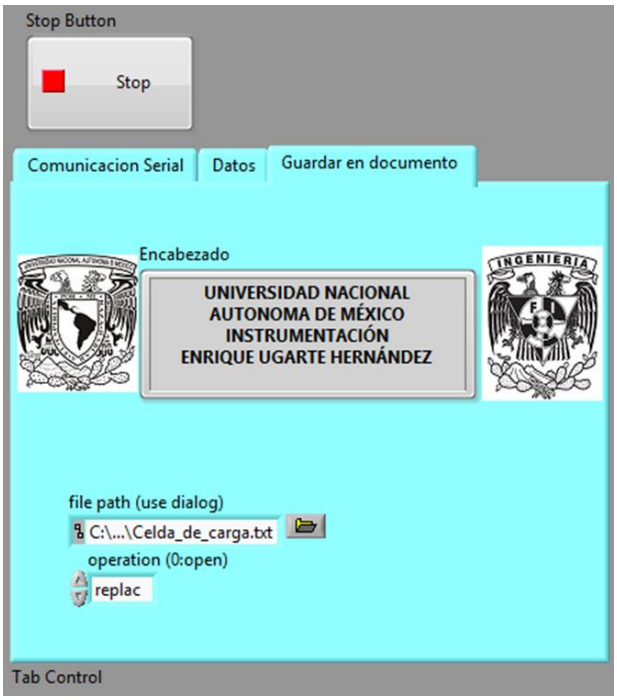


Figura 10. Tercera página del contenedor con pestañas.

c) Calibración de la báscula

Para calibrar la escala se tiene que montar sobre la balanza algún objeto, cuyo peso sea conocido, por ejemplo: si montamos sobre la báscula una bolsa de arroz de 1/2 kg, aparece algún valor en el *read buffer*, el cual diferirá del valor real, entonces se debe ajustar el valor de *Escala* para que en *Peso [kg]* aparezca el valor real.

Escala= Peso real conocido/ valor del *read buffer*

d) Datos en el documento de texto

Al guardar los datos dentro del documento de texto, se debe generar una imagen similar a la siguiente.

```
UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO
INSTRUMENTACIÓN
ARTURO RONQUILLO
0 05/12/2017     10:44 a. m.     PESO:          0.000000        Kg
1 05/12/2017     10:44 a. m.     PESO:          0.006000        Kg
2 05/12/2017     10:44 a. m.     PESO:          0.702000        Kg
3 05/12/2017     10:44 a. m.     PESO:          0.702000        Kg
4 05/12/2017     10:44 a. m.     PESO:          0.702000        Kg
5 05/12/2017     10:44 a. m.     PESO:          0.700000        Kg
6 05/12/2017     10:44 a. m.     PESO:          0.702000        Kg
7 05/12/2017     10:44 a. m.     PESO:          0.706000        Kg
8 05/12/2017     10:44 a. m.     PESO:          0.708000        Kg
9 05/12/2017     10:44 a. m.     PESO:          0.708000        Kg
10 05/12/2017    10:44 a. m.     PESO:          0.704000        Kg
11 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
12 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
13 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
14 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
15 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
16 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
17 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
18 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
19 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
20 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
21 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
22 05/12/2017    10:44 a. m.     PESO:          0.700000        Kg
```

Figura 11. Datos dentro del documento de texto.

V. FUENTES DE CONSULTA

[1] Granda, M., Mercedes y Mediavilla B., Elena. (2010). *Instrumentación electrónica: Transductores y acondicionadores de señal*. Cantabria, España. Editorial Universidad de Cantabria.

[2] Corona Rodríguez, L. G. (2014*). Sensores y Actuadores: Aplicaciones con Arduino*. Grupo Editorial Patria.

[3] Tutoriales. Tutorial trasmisor de celda de carga HX711, Balanza Digital. Trujillo, Perú. *Naylamp Mechatronics*. Recuperado de http://www.naylampmechatronics.com/blog/25_tutorial-trasmisor-de-celda-de-carga-hx711-ba.html .

[4] AVD. (2016). Celda De Carga 50 Kg Sensor Peso Bascula Galga Extensiometrica. *www.lionchipmexico.com* . Recuperado de https://www.lionchipmexico.com/product-page/celda-de-carga-50-kg-sensor-peso-bascula-galga-extensiometrica.

[5] Alzate E. J., Montes J. W. y Silva C. A. (2007). MEDIDORES DE DEFORMACION POR RESISTENCIA: GALGAS EXTENSIOMÉTRICAS. *Scientia et Technica Año XIII* (34), 9.

[6] Elegant Themes. HX711 Módulo Amplificador Celda de Carga. *Geek Factory*. Recuperado de https://www.geekfactory.mx/tienda/modulos-para-desarrollo/hx711-modulo-amplificador-celda-de-carga/ .

[7] Morales R. (2017). Puente de Wheatstone. México. CURSO EN LÍNEA FÍSICA III. Recuperado de http://www.academico.cecyt7.ipn.mx/FisicaIII/temas/wheatstone.htm .

[8] PotentialLabs (2013 - 2018). HX711 Weighing Sensor Dual-Channel 24 Bit Precision A/D weight Pressure Sensor. Kondapur & Mehdipatnam (Asifnagar), *Hyderabad.potentiallabs.com*. Recuperado de https://potentiallabs.com/cart/hx711-weighing-sensor-module-india

[9] Bodge. HX711. *Github*. Recuperado de https://github.com/bogde/HX711 .

Práctica 8. Sensor piroeléctrico y modulación por ancho de pulso (PWM)

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En este trabajo se explica cómo utilizar el sensor PIR de manera práctica y se muestra como mandar y recibir datos en LabVIEW a través del puerto serie, utilizando una señal PWM manipulada por un slider para controlar la led intensidad luminosa de un led. Para esta práctica se utiliza la tarjeta Arduino UNO para la adquisición de datos y el entorno de desarrollo de LabVIEW para la generación de una interfaz gráfica clara para el usuario.**

I. INTRODUCCIÓN

A. Sensores piroeléctricos

El efecto piroeléctrico es análogo al piezoeléctrico, solo que en vez de tratarse de una carga que produce una diferencia de potencial, se trata de un material, el cual al ser afectado por un cambio en la temperatura produce una tensión [1]. Esto se produce ya que este material realiza un cambio en su polarización con los cambios de temperatura. Todos los materiales desprenden energía, esta energía se transmite en forma de calor y este calor se transmite mediante radiación infrarroja (RI), por este motivo normalmente son utilizados para detectar movimiento o CO2 [1, 2].

A. PWM (Pulse Width Modulation)

La modulación por ancho de pulsos (también conocida como PWM, siglas en inglés de pulse-width modulation) de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

El ciclo de trabajo de una señal periódica es el ancho relativo de su parte positiva en relación con el período. Expresado matemáticamente [3]:

$$D = \frac{\tau}{T}$$

Donde:

$D = Ciclo\ de\ trabajo\ [s]$

$\tau = Es\ el\ tiempo\ en\ que\ la\ función\ es\ positiva$

$(ancho\ del\ pulso)[s]$

$T = Periodo\ de\ la\ función\ [s]$

El control digital se usa para crear una onda periódica, una señal conmutada entre encendido y apagado. Este patrón de encendido-apagado puede simular tensiones entre el encendido total (5 volts) y el apagado (0 volts) al cambiar la porción del tiempo que la señal pasa frente al tiempo que la señal se apaga. La duración de "encendido" se llama ancho de pulso. Para obtener valores análogos variables, cambia o modula ese ancho de pulso. Si repite este patrón de encendido-apagado lo suficientemente rápido con un led, por ejemplo, el resultado es como si la señal fuera una tensión constante entre 0 y 5 [V] que controla la intensidad con que prende del led [4].

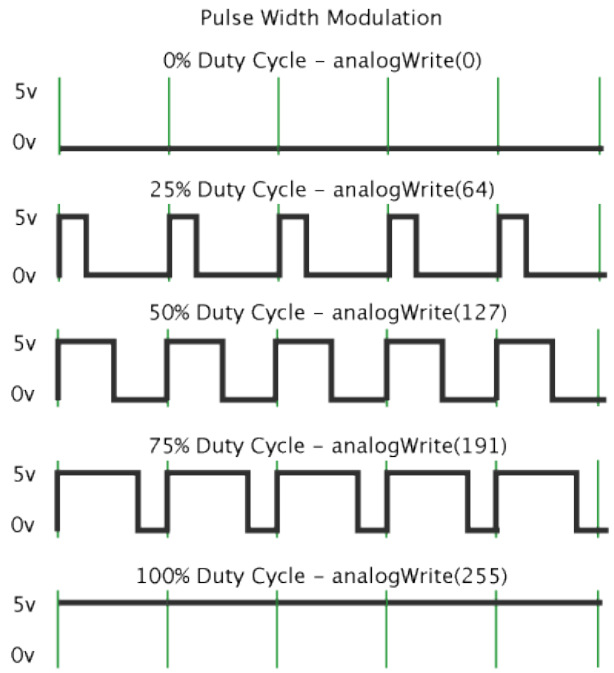


Figura 1. Señal PWM generada por la tarjeta Arduino [4].

B. Sensor PIR HC-SR501

Los sensores PIR se basan en la medición de la radiación infrarroja. Todos los cuerpos emiten una cierta cantidad de energía infrarroja, cuanto mayor es su temperatura, mayor es la energía que desprenden.

En realidad cada sensor está dividido en dos campos. Disponen de un circuito eléctrico que compensa ambas mediciones. Si ambos campos reciben la misma cantidad de infrarrojos la señal eléctrica resultante es nula. Por el contrario, si los dos campos realizan una medición diferente, se genera una señal eléctrica.

De esta forma, si un objeto atraviesa uno de los campos se genera una señal eléctrica diferencial, que es captada por el sensor y se emite una señal digital [5].

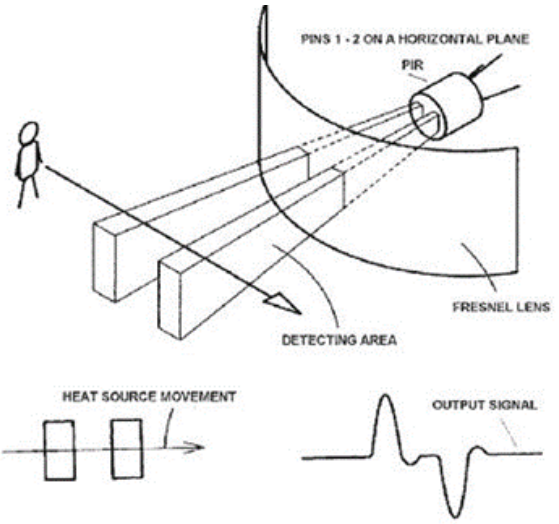


Figura 1. Funcionamiento del sensor PIR [1].

Sobre el sensor PIR hay un encapsulado semiesférico llamado Lente de Fresnel, el cual centra las señales infrarrojas sobre el elemento. Tiene solamente 3 pines para su conexión: 5 V, salida y GND. Cuenta con 2 potenciómetros, los cuales permiten regular el tiempo entre cada medición y la distancia máxima que puede detectar. Su rango de medición llega hasta los 7 metros, con un ángulo de apertura entre 90° y 110° [5].

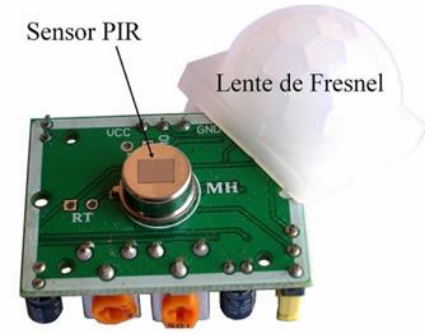


Figura 2. Sensor PIR y lente de Fresnel [4].

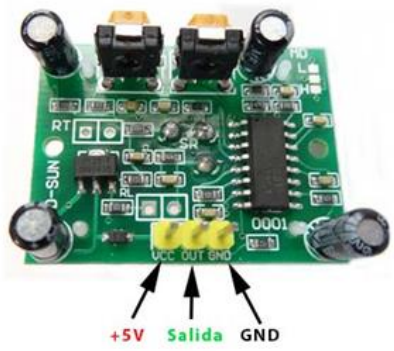


Figura 3. Terminales del sensor PIR [4].

Figura 4. Trimpots del sensor PIR [4].

Figura 5. Rango de medición del sensor PIR [4].

C. Arduino UNO

Cuenta con 6 salidas PWM, las cuales están señaladas con una virgulilla, los pines PWM son el 3, 5, 6, 9, 10 y 11 y el rango de valores están entre 0 y 255. Para declarar una salida PWM se utiliza la instrucción *analogWrite(pin, valor)*. Para convertir datos de tipo *String* a entero, se utiliza el comando *parseInt()* .

II. OBJETIVOS

- ✓ Aprender a utilizar el sensor PIR de forma práctica para detectar presencia.
- ✓ Realizar una interfaz en LabVIEW que permita visualizar la salida de un sensor piroeléctrico y que pueda manipular la intensidad de un led con una señal PWM. El programa debe contar con un contenedor con pestañas, debe usarse una pestaña para los componentes relacionados con la comunicación serie, las necesarias para mostrar a información y una con las opciones para guardar los datos en un archivo de texto.

III. MATERIAL

- ➢ Tarjeta Arduino UNO.
- ➢ Sensor PIR HC-SR501.
- ➢ 1 Resistencia de 330 Ω 1/4 W.
- ➢ 1 Led de 5 mm (que no sea emisor infrarrojo).
- ➢ 3 Jumpers [M-H].
- ➢ Alambre para alambrar sobre protoboard (se deja al criterio del alumno).

- Protoboard.
- Computadora con el software IDE de Arduino y LabVIEW.

IV. DESARROLLO

A. Conexión

Realizar las siguientes conexiones:

- La resistencia con el led → terminal 5.
- Salida del sensor PIR → terminal 2.
- Se conectan los 5V y GND.

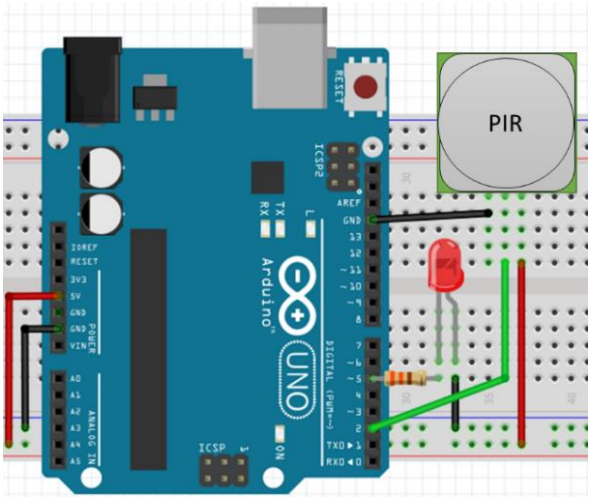


Figura 6. Conexión del sensor PIR y el LED.

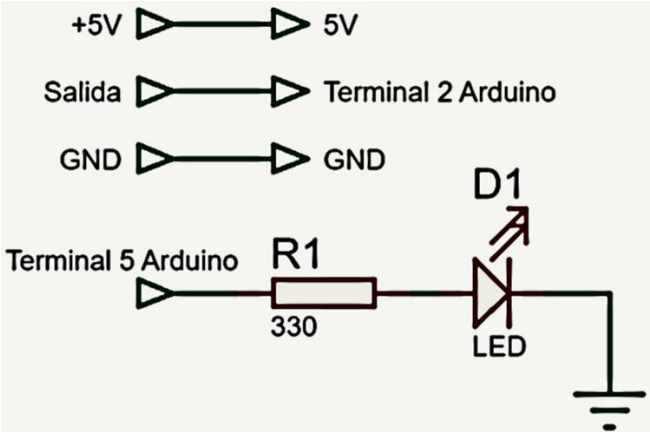


Figura 7. Esquema de conexiones.

B. Código de Arduino

Se abre el IDE de Arduino e ingresamos el siguiente código:

```
int pir=0;
int pwm;
void setup() {
Serial.begin(115200);
pinMode(2,INPUT);
pinMode(5,OUTPUT);
}
void loop() {
pwm=Serial.parseInt();
analogWrite(5,pwm);
pir =digitalRead(2);
if (pir ==1)
Serial.println("Ha detectado algo");
else
Serial.println("No se ha detectado nada");
delay(10);
}
```

Este programa lee a través de la comunicación serial el valor PWM que envía al led y lee la respuesta del sensor. Si se abre el monitor serial es posible visualizar cuando se detecta algún cambio de temperatura, ya que aparecerá: “Ha detectado algo”, en caso contrario se mostrará un letrero que dice: “No se ha detectado nada”.

C. Adquisición de datos

Este programa manda y recibe información a través del puerto serie (ver “Introducción al entorno de desarrollo de LabVIEW”), no utiliza el bloque *VISA Bytes at Serial Port* porque este fue reemplazado por una constante numérica para reducir el tiempo de muestreo. El valor de la constante depende del número de bits a leer. Es importante que la constante sea igual o mayor al número de bits de entrada para que pueda leer toda la información. En caso de exceder el número de bits a leer simplemente incrementará el tiempo de muestreo.

El programa usa un slider para manipular la salida PWM y finalmente cuenta con los bloques para guardar los datos dentro de un documento de texto (ver “Introducción al entorno de desarrollo de LabVIEW”).

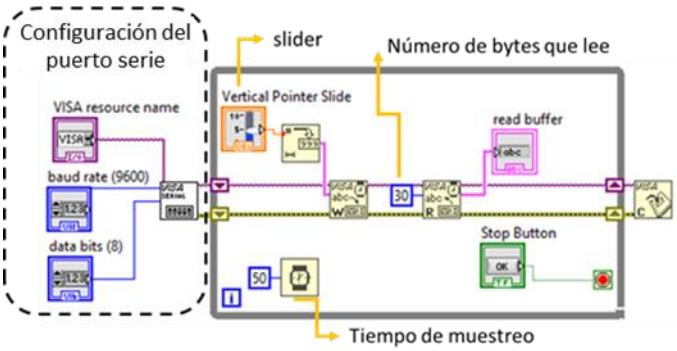


Figura 7. Diagrama de bloques para la comunicación serie.

D. Diseño final

A continuación se muestra el diagrama de bloques completo.

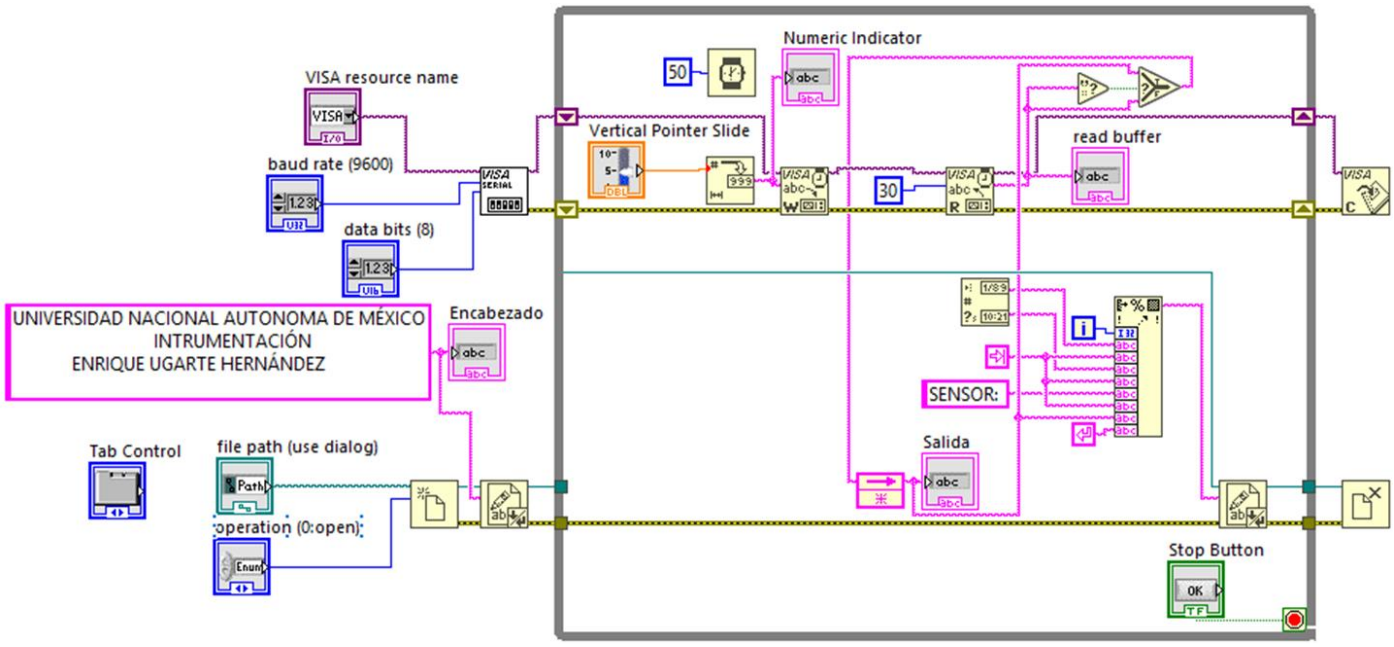


Figura 10. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña, la visualización de los datos y el *slider* en la segunda y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

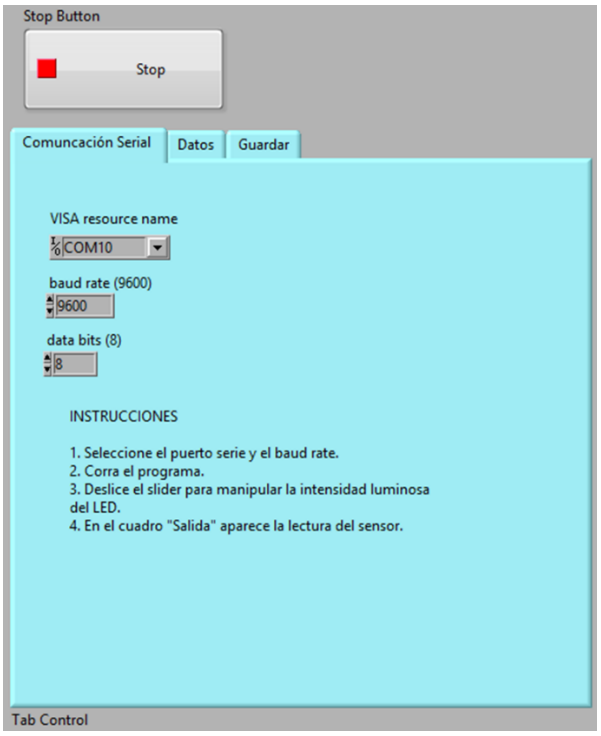


Figura 11. Primera página del contenedor con pestañas.

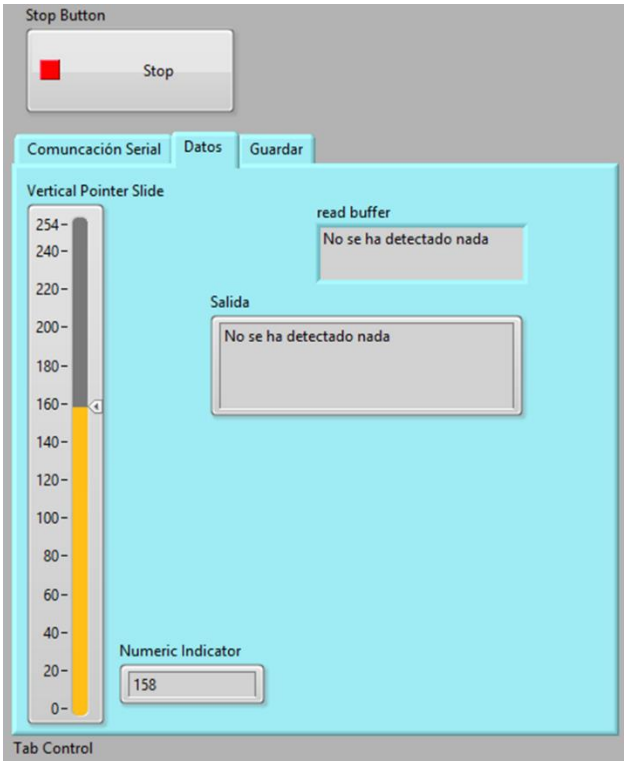


Figura 12. Segunda página del contenedor con pestañas.

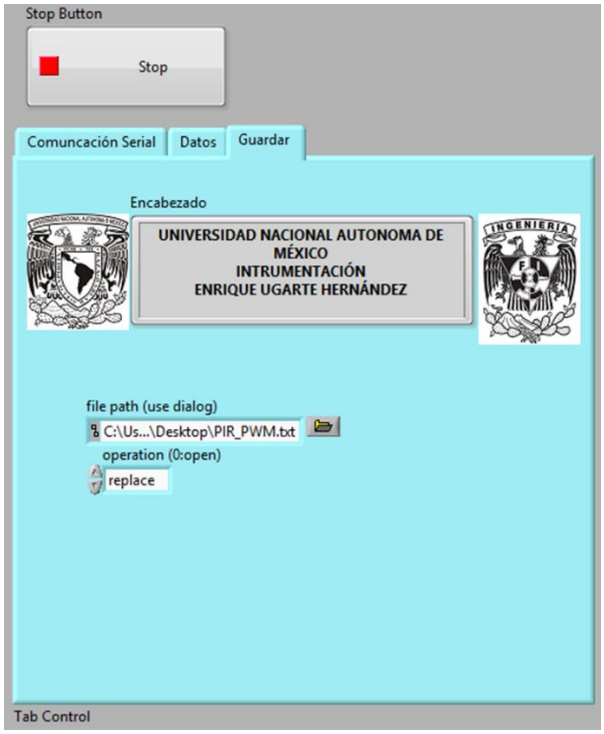


Figura 13. Tercera página del contenedor con pestañas.

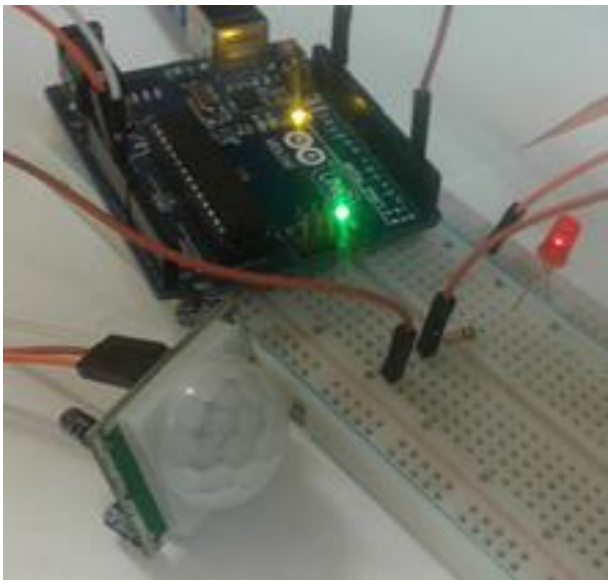


Figura 14. Pruebas con el sensor PIR y el led cuando la señal PWM está al máximo.

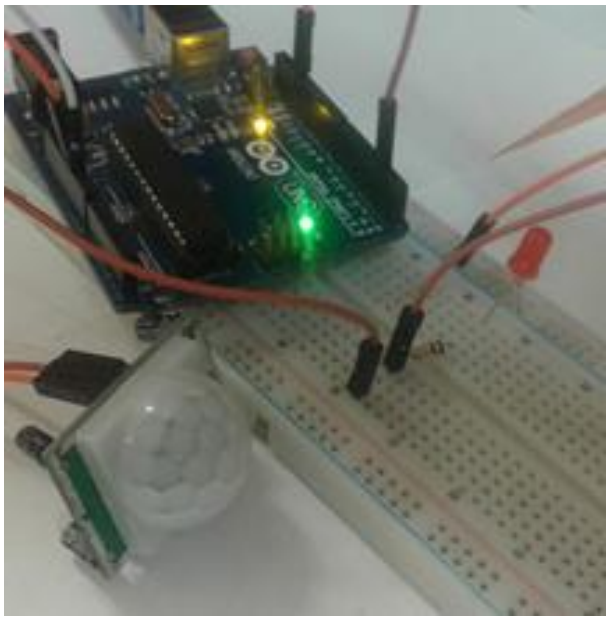


Figura 20. Pruebas con el sensor PIR y el led cuando la señal PWM está al mínimo.

Archivo Edición Formato Ver Ayuda

UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO
INTRUMENTACIÓN
ARTURO RONQUILLO

0	05/12/2017	10:29 a. m.	SENSOR:	No se ha detectado nada
1	05/12/2017	10:29 a. m.	SENSOR:	No se ha detectado nada
2	05/12/2017	10:29 a. m.	SENSOR:	No se ha detectado nada
3	05/12/2017	10:29 a. m.	SENSOR:	No se ha detectado nada
4	05/12/2017	10:29 a. m.	SENSOR:	Ha detectado algo
5	05/12/2017	10:29 a. m.	SENSOR:	No se ha detectado nada
6	05/12/2017	10:29 a. m.	SENSOR:	Ha detectado algo
7	05/12/2017	10:29 a. m.	SENSOR:	Ha detectado algo
8	05/12/2017	10:29 a. m.	SENSOR:	Ha detectado algo
9	05/12/2017	10:29 a. m.	SENSOR:	Ha detectado algo
10	05/12/2017	10:29 a. m.	SENSOR:	Ha detectado algo
11	05/12/2017	10:29 a. m.	SENSOR:	No se ha detectado nada
12	05/12/2017	10:29 a. m.	SENSOR:	Ha detectado algo
13	05/12/2017	10:29 a. m.	SENSOR:	Ha detectado algo

Figura 15. Datos dentro del documento de texto.

V. FUENTES DE CONSULTA

[1] Castillo, Omar & Villegas, Baryeric. (2007). Sensor piroelectrico. Universidad nacional experimental politécnica "Antonio José de Sucre". *Mediciones Industriales*. Recuperado de http://piroelectrico.blogspot.mx/ .

[2] Macho M., Juan C. (2016) Sensores de temperatura DHT11. Sensor de temperatura y humedad. Bilbao, España. *Prometec*. Recuperado de https://www.prometec.net/sensor-pir/comment-page-1/#comments .

[3] Wikipedia. (2018). Modulación por ancho de pulsos. *https://es.wikipedia.org* . Recuperado de https://es.wikipedia.org/wiki/Modulaci%C3%B3n_por_ancho_de_pulsos

[4] Arduino. (2018). PWM. *Arduin*o. Recuperado de https://www.arduino.cc/en/Tutorial/PWM .

[4] Punto Flotante S.A. (2017). Sensor infrarrojo de movimiento PIR HC-SR501. Recuperado de https://puntoflotante.net/MANUAL-DEL-USUARIO-SENSOR-DE-MOVIMIENTO-PIR-HC-SR501.pdf .

Práctica 9. Par emisor-receptor

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En este trabajo se explica cómo utilizar los diodos emisores de luz infrarrojos para la detección de algún objeto, usando la tarjeta Arduino UNO para la adquisición de datos y el entorno de desarrollo de LabVIEW para la elaboración de una interfaz gráfica que indique cuando algún objeto opaco se cruce entre el diodo emisor infrarrojo y el fototransistor, almacenando la información dentro de un contador, el cual se podrá reiniciar desde la misma interfaz; valiéndose de la comunicación serie para mandar datos desde el Arduino a LabVIEW.**

I. INTRODUCCIÓN

A. Sensores Fotoeléctricos

También llamados sensores ópticos, son dispositivos de usan un haz de luz para detectar la presencia de algún objeto [1]. Algunos constan de un emisor y receptor, los que cuentan con estas dos partes se le clasifica en tres tipos:

1) Modelo Reflectivo

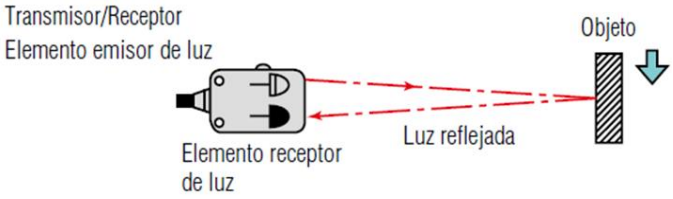


Figura 1. Modelo reflectivo.

El circuito se encuentra normalmente abierto, el emisor manda una señal luminosa y solo cuando es reflejada por un objeto el circuito se cierra y manda una señal eléctrica.

2) Modelo de barrera

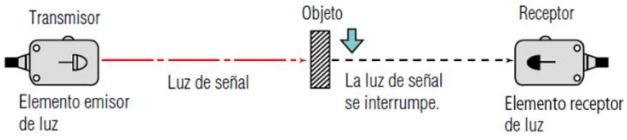


Figura 2. Modelo de barrera.

El circuito es del tipo normalmente cerrado, de esta manera el circuito siempre está mandando una señal eléctrica, excepto cuando algún objeto se interponga entre el emisor y el receptor, por lo tanto cuando no manda señal significa que ha detectado algo.

3) Modelo retroreflectivo

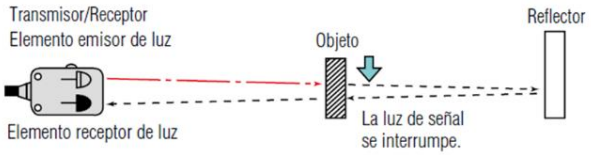


Figura 3. Modelo retroreflectivo.

El circuito es normalmente cerrado, ya que el emisor manda una señal que se refleja gracias a la superficie reflectora y se dirige hacia el receptor. Al interponerse un objeto entre el sensor y la superficie reflectora se abre el circuito. De esta manera el circuito se abre al detectar algún objeto [2].

Nota: Las conexiones de cada configuración es la misma en todos los casos, simplemente cambia la forma en que los elementos se encuentran acomodados. Las conexiones se muestran en la figura 4.

B. Características y funcionamiento de los leds infrarrojos

Los leds son componentes eléctricos hechos de un material semiconductor, su nombre viene de las siglas: *light emitting diode*, que significa: diodo emisor de luz. Así como su nombre lo dice, emiten luz; dependiendo de los elementos químicos de los materiales de sus compuestos pueden generar diversos colores [3]:

- Arseniuro de galio → infrarrojo.
- Carburo de silicio → azul.
- Fosfuro de galio → verde.
- Arseniuro fosfuro de galio → rojo, naranja y amarillo.

Los leds infrarrojos emiten radiación electromagnética, la cual no puede ser vista por los humanos, solo puede ser visualizada a través de algunas cámaras de video; se utilizan junto a un fototransistor para la detección de algún objeto.

Los fototransistores son transistores con la particularidad que la señal que regula la salida del emisor del fototransistor proviene de una fuente luminosa, específicamente la luz infrarroja. Aunque su apariencia es la de un led es importante tomar en cuenta que su conexión es al revés, la terminal corta va al voltaje positivo, y la terminal larga a tierra [4].

II. OBJETIVOS

- ✓ Aprender a utilizar los leds infrarrojos (emisores) y fototransistores (receptores).

- ✓ Programar un contador en LABVIEW utilizando un par emisor-receptor como codificador (coloquialmente llamado encoder).
- ✓ Desarrollar una interfaz gráfica en LabVIEW que distribuya los controladores e indicadores del programa dentro de un contenedor con pestañas, donde los elementos relacionados con la comunicación se encuentren en la primera pestaña, los indicadores en la segunda y los elementos para guardar datos en un documento de texto en la tercera pestaña.
- ✓ Aprender nuevas funciones en LabVIEW.

III. MATERIAL

- ➢ 2 Resistores de 330 Ω 1/4 W.
- ➢ 1 Resistor de 10 kΩ 1/4 W.
- ➢ 1 Led infrarrojo (emisor).
- ➢ 1 Fototransistor (receptor).
- ➢ 1 Led de 5mm (que no sea infrarrojo).
- ➢ Alambre para protoborad (se deja al criterio del alumno) y jumpers.
- ➢ Tarjeta Arduino UNO.
- ➢ Protoboard.
- ➢ Computadora con el software IDE de Arduino y LabVIEW.

IV. DESARROLLO

A. Conexión

Conectar el led infrarrojo (emisor) y el fototransistor con sus resistores correspondientes, existen dos maneras de conexión:

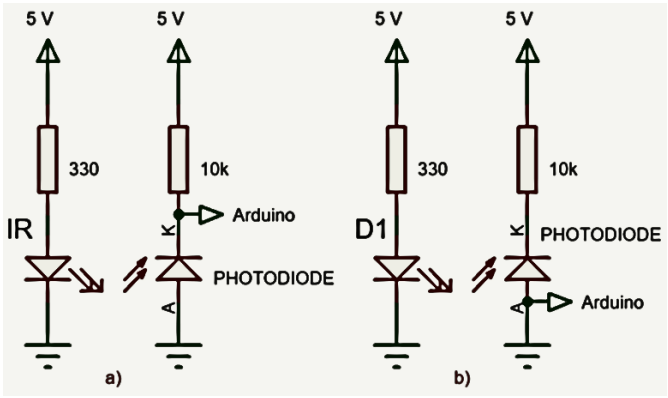


Figura 4. Conexión del emisor y receptor.

Es importante reconocer cual es el emisor y receptor. El led infrarrojo es el transparente y el negro es el fototransistor. Al conectar el emisor es posible es posible ver la luz de este led a través del celular (excepto los iPhone, ya que tienen un filtro óptico). Para que se note bien la luz es recomendable que se encuentre en un lugar oscuro, ya que en un ambiente muy iluminado podría notarse muy poco.

Conectar un led a la salida para probar la salida del circuito. El led indicador debe estar apagado cuando no haya nada entre el emisor y receptor, y al cruzarse algún objeto opaco debe prender.

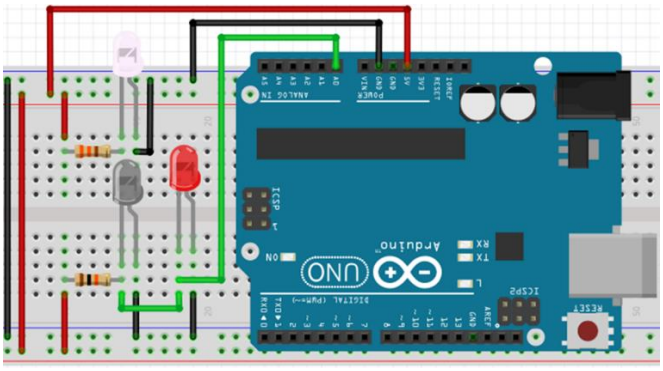


Figura 5. Conexión de los componentes a).

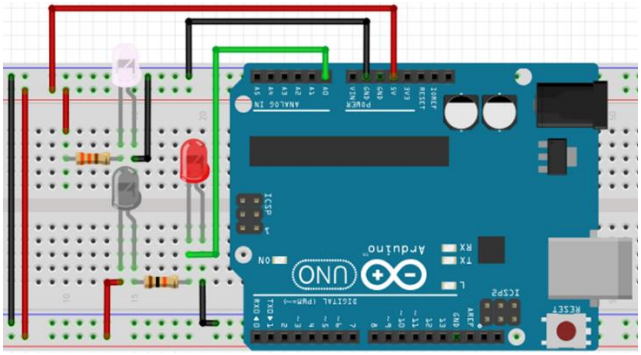


Figura 6. Conexión de los componentes b).

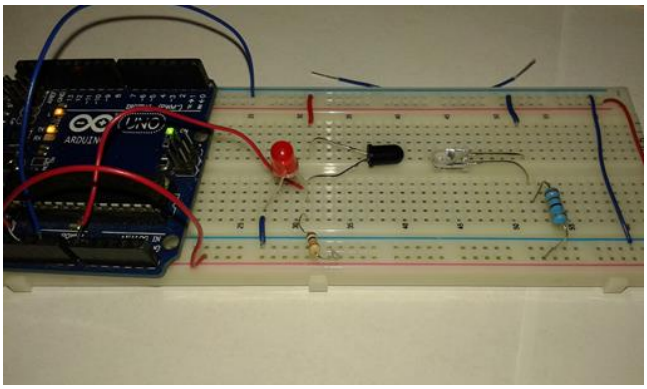


Figura 7. Conexión real de los diodos emisores sobre una protoboard.

B. Código de Arduino

Realizamos un primer código en el IDE de Arduino.

```
void setup() {
Serial.begin(9600);
}
void loop() {
Serial.println(analogRead(A0));
delay(500);
}
```

Este programa lee el valor analógico del emisor del fototransistor, de esta manera se puede conocer el valor en alto y bajo para obtener solamente dos valores diferentes. Con base en estos resultados se crea un segundo programa de Arduino.

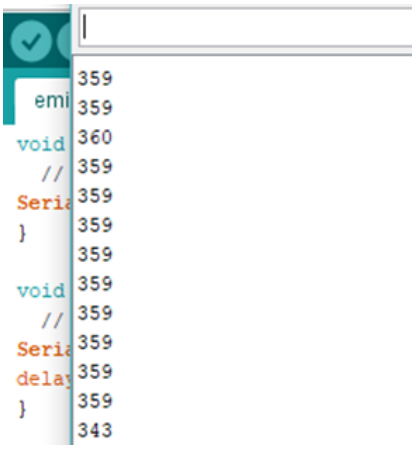


Figura 8. Lectura analógica del fototransistor.

```
void setup() {
Serial.begin(115200);
}
void loop() {
if(analogRead(A0)>350)
{
  Serial.println("1");
}
else
{
  Serial.println("0");
}
delay(100);
}
```

Con esto ya se tiene una señal lógica. Abrir LabVIEW para diseñar la interfaz gráfica.

C. Adquisición de datos

El programa de LabVIEW obtiene el dato lógico del Arduino mediante comunicación serie (ver "Introducción al entorno de desarrollo de LabVIEW"), como la entrada llega siendo de tipo String, es necesario volverla lógica. Se usa el bloque *Boolean Crossing* para detectar cambios en la entrada, ya sea de alto a bajo, bajo a alto o cualquier cambio y finalmente se agregan los bloques y conexiones necesarias para generar el contador y el reinicio. El bloque *Boollean Crossing* se ubica en *Signal Processing → Point by Point → Other Functions PtByPt*.

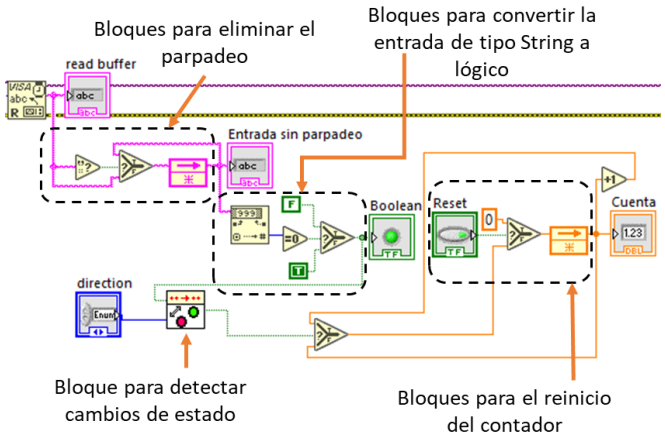


Figura 9. Eliminación del parpadeo, conversión del tipo de dato de la entrada, contador y reinicio.

D. Diseño final

A continuación se muestra el Diagrama de Bloques completo.

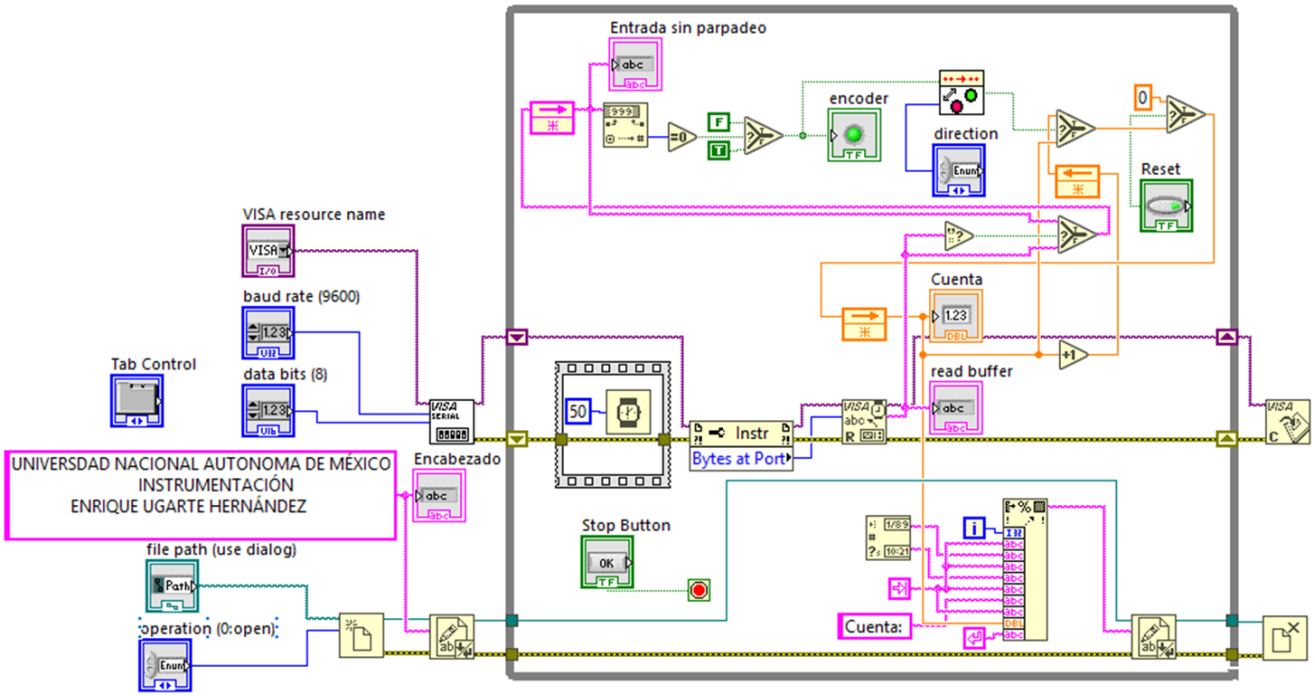


Figura 10. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña; los indicadores, el reinicio del contador y el control del *Boolan Crossing* en la segunda pestaña y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

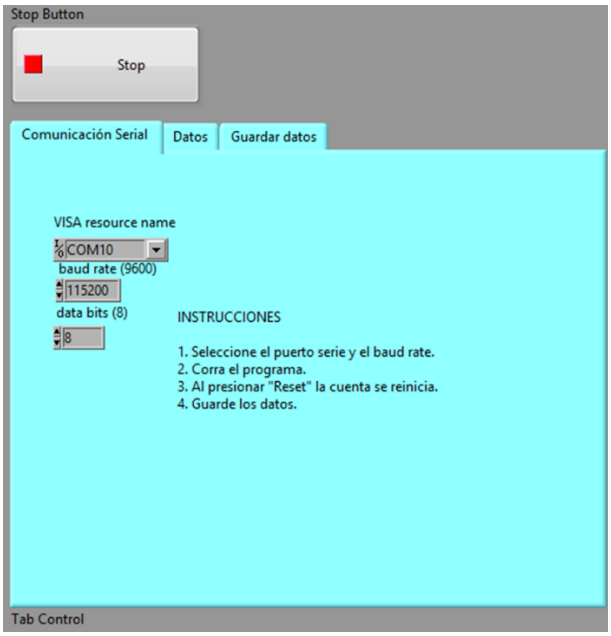


Figura 11. Primera página del contenedor con pestañas.

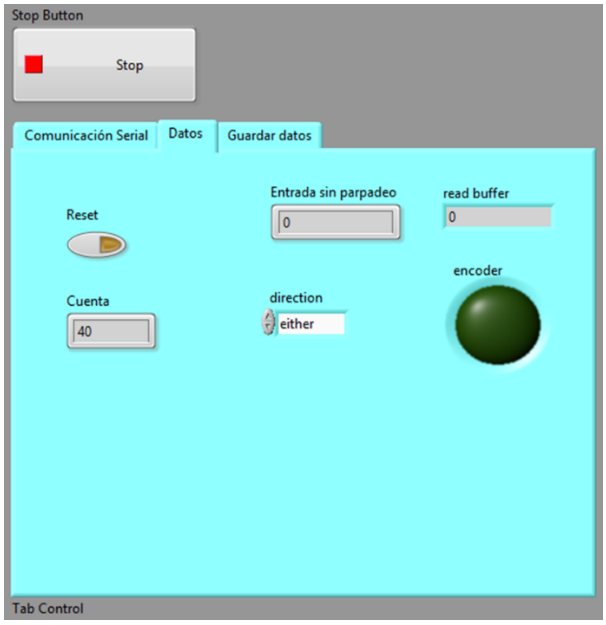


Figura 12. Segunda página del contenedor con pestañas.

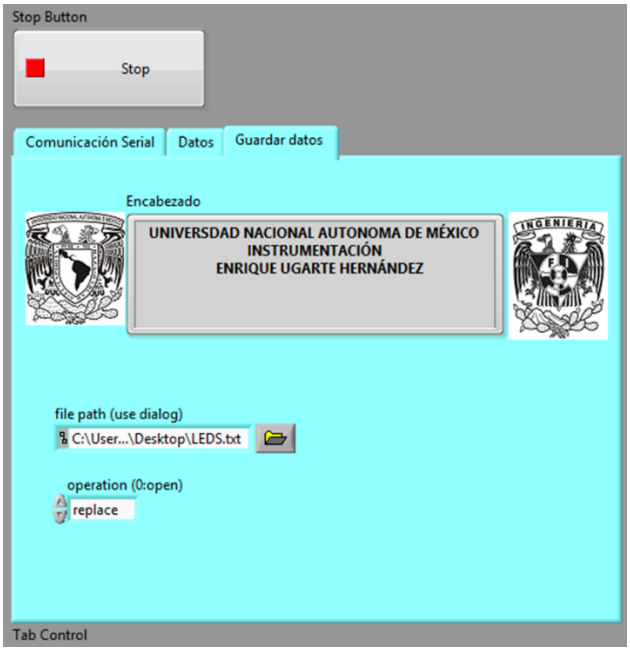


Figura 13. Tercera página del contenedor con pestañas.

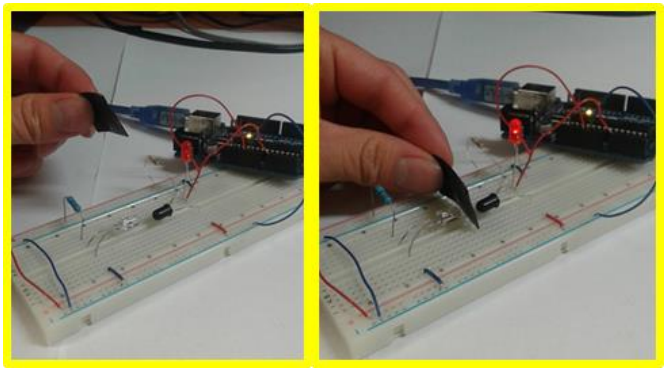


Figura 14. Pruebas con los leds. Cuando un objeto opaco se interpone entre el led infrarrojo (emisor) y el fototransistor (receptor) se prende el led rojo, mientras que cuando se retira se apaga el led.

Archivo Edición Formato Ver Ayuda

UNIVERSDAD NACIONAL AUTONOMA DE MÉXICO
INSTRUMENTACIÓN
ARTURO RONQUILLO

0	05/12/2017	01:33 p. m.	Cuenta:	0.000000
1	05/12/2017	01:33 p. m.	Cuenta:	0.000000
2	05/12/2017	01:33 p. m.	Cuenta:	1.000000
3	05/12/2017	01:33 p. m.	Cuenta:	1.000000
4	05/12/2017	01:33 p. m.	Cuenta:	1.000000
5	05/12/2017	01:33 p. m.	Cuenta:	1.000000
6	05/12/2017	01:33 p. m.	Cuenta:	1.000000
7	05/12/2017	01:33 p. m.	Cuenta:	1.000000
8	05/12/2017	01:33 p. m.	Cuenta:	1.000000
9	05/12/2017	01:33 p. m.	Cuenta:	1.000000
10	05/12/2017	01:33 p. m.	Cuenta:	1.000000
11	05/12/2017	01:33 p. m.	Cuenta:	1.000000
12	05/12/2017	01:33 p. m.	Cuenta:	1.000000
13	05/12/2017	01:33 p. m.	Cuenta:	1.000000
14	05/12/2017	01:33 p. m.	Cuenta:	1.000000
15	05/12/2017	01:33 p. m.	Cuenta:	2.000000
16	05/12/2017	01:33 p. m.	Cuenta:	2.000000
17	05/12/2017	01:33 p. m.	Cuenta:	3.000000
18	05/12/2017	01:33 p. m.	Cuenta:	3.000000
19	05/12/2017	01:33 p. m.	Cuenta:	3.000000
20	05/12/2017	01:33 p. m.	Cuenta:	3.000000
21	05/12/2017	01:33 p. m.	Cuenta:	3.000000
22	05/12/2017	01:33 p. m.	Cuenta:	3.000000
23	05/12/2017	01:33 p. m.	Cuenta:	3.000000
24	05/12/2017	01:33 p. m.	Cuenta:	3.000000

Figura 15. Datos dentro del documento de texto.

V. FUENTES DE CONSULTA

[1] Rockwell Automation, Inc. (2018). Sensores fotoeléctricos. *Allen-Bradley*. Recuperado de http://ab.rockwellautomation.com/es/Sensors-Switches/Photoelectric-Sensors .

[2] KEYENCE CORPORATION (2018). ¿Qué es un sensor fotoeléctrico? . Recuperado de https://www.keyence.com.mx/ss/products/sensor/sensorbasics/photoelectric/info/ .

[3] Que son los LED's y cómo funcionan. Santiago, Chile. *Gescomchile.com*. Recuperado de http://www.gescomchile.com/que_son_los_leds_y_como_funcionan.html .

[4] Sensor de infrarrojos (emisor y receptor) (2010). circuitoselectronicos.org. Recuperado de http://www.circuitoselectronicos.org/2010/05/sensor-de-infrarrojos-emisor-y-receptor.html .

Práctica 10. Sensor de efecto Hall

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En este trabajo se explica cómo utilizar de manera práctica el sensor de efecto Hall, utilizándolo como codificador para medir la velocidad de un motor DC. Se utiliza la tarjeta Arduino UNO como microcontrolador y el entorno de desarrollo LabVIEW para la generación de una interfaz gráfica que permite manipular la rapidez angular del motor DC y al mismo tiempo medir la rapidez del mismo. Para manipular la rapidez se usa una señal PWM. Se vale de la comunicación serie para el intercambio de información entre el Arduino y LabVIEW.**

I. INTRODUCCIÓN

A. Sensores magnéticos

Los sensores magnéticos son dispositivos capaces de detectar los campos magnéticos producidos por imanes o corrientes eléctricas. Dos claros ejemplos son los reed switch y el sensor de efecto Hall [1].

B. Funcionamiento del sensor de efecto Hall

El sensor de efecto Hall funciona bajo el principio del efecto Hall, el cual establece que si hay algún material conductor recibiendo corriente eléctrica y este es expuesto a un campo magnético que fluya en dirección vertical al sensor, este moverá carga hacia el conductor (este efecto solo aplica en materiales conductores, semiconductores y soluciones electrolíticas). Sirven para detectar campos magnéticos, aunque son muy usados en distintos aparatos (por ejemplo, ventiladores, unidades de DVD, impresoras láser, sensores de posición del cigüeñal en los carros, entre otros) para medir velocidad de motores funcionando como encoder con ayuda de un imán [2]. Se pueden encontrar en presentaciones con 4 terminales, los cuales sirven para medir intensidades de campos magnéticos y pueden identificar la polaridad del campo siendo alimentados con una fuente positiva. También los hay con 3 terminales, los cuales producen una señal binaria, sirven como codificadores (coloquialmente llamados encoders) y según la polaridad del campo magnético que se le aplique al sensor, la tensión de salida será positiva o negativa; la salida también depende de la fuente de alimentación: si la fuente de alimentación el positiva se obtendrá una tensión de salida positiva y si se obtiene una fuente de alimentación negativa solo se obtendrá una salida negativa.

II. OBJETIVOS

- ✓ Construir una base que permita medir la rapidez angular de un motor DC con ayuda de un sensor de efecto Hall.
- ✓ Escribir un código de Arduino que permita mandar a través del puerto serie el número de revoluciones por segundo.
- ✓ Desarrollar una interfaz gráfica en LabVIEW que distribuya los controladores e indicadores del programa dentro de un contenedor con pestañas, donde los elementos relacionados con la comunicación se encuentren en la primera pestaña, el control de la rapidez y los indicadores en la segunda y tercera pestaña y los elementos para guardar datos en un documento de texto en la cuarta pestaña.

III. MATERIALES

- ➢ Tarjeta Arduino UNO.
- ➢ Sensor de efecto Hall A3144 (3 terminales).
- ➢ 1 Imán (se deja al criterio del alumno).
- ➢ Resistencia de 10 kΩ 1/4 W.
- ➢ 1 motor DC (se recomienda con una baja potencia, que requiera 5 V para ser alimentado).
- ➢ Alambre para Protoboard (optativo, se deja al criterio del alumno).
- ➢ Material para la construcción del modelo.
- ➢ Computadora con el software IDE de Arduino y LabVIEW.
- ➢ Puente H (opcional).

Nota: En caso de usar un motor que demande mucha corriente se recomienda agregar una etapa de potencia, utilizando una fuente de alimentación externa.

IV. DESARROLLO

A. Construcción del modelo

Para poder medir la rapidez del motor se tiene que unir un cople al eje del motor y sobre éste montar un imán. Al girar el motor cambia la posición del imán; de esta manera cuando el imán se encuentra sobre el sensor de efecto Hall, éste manda una señal alta y cuando se aleja una señal baja. Se deja al criterio del alumno los materiales a utilizar. La base debe ser lo suficientemente fuerte para sujetar el motor.

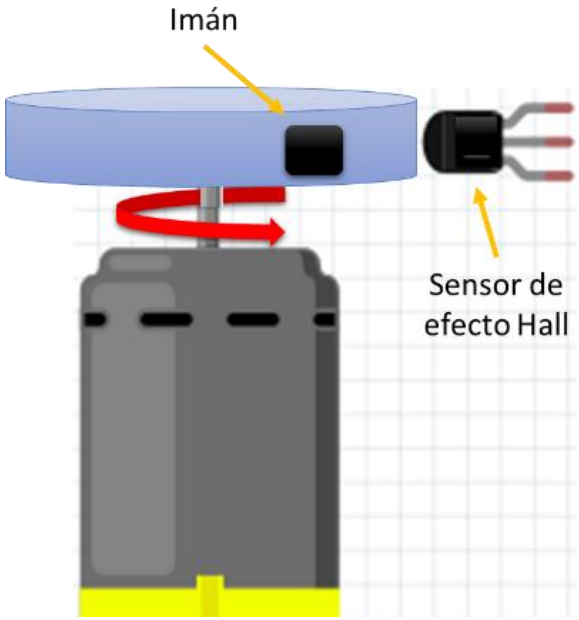


Figura 1. Esquema de modelo físico.

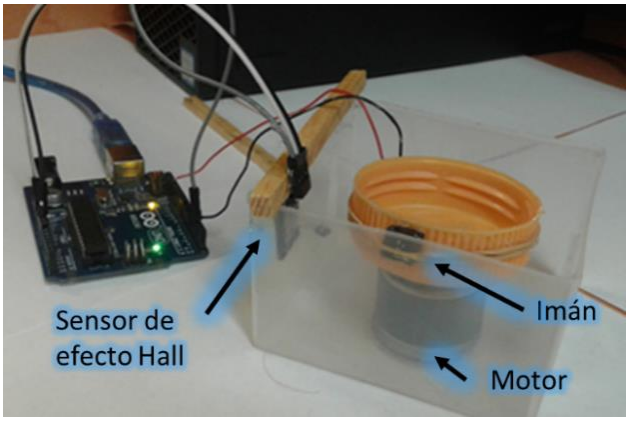


Figura 2. Ejemplo de armado del modelo físico.

B. Conexión

Conectar la salida del sensor de efecto Hall a la terminal 2 del Arduino, la terminal 3 del Arduino a alguna terminal del motor y puentear tierras.

Figura 3. Terminales del sensor de efecto Hall A3144.

A continuación se muestran las conexiones:

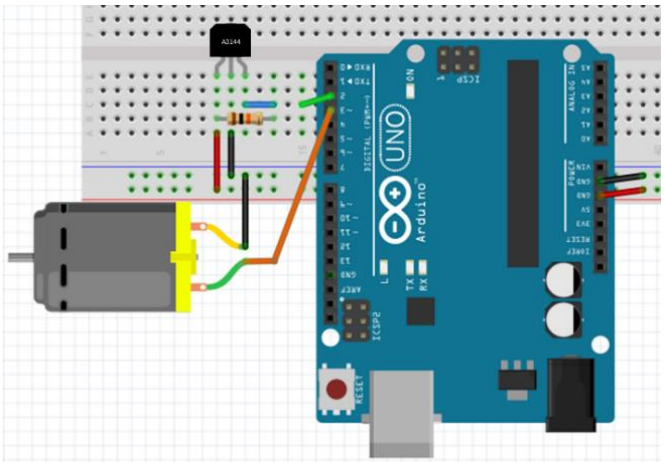


Figura 4. Conexión de los componentes.

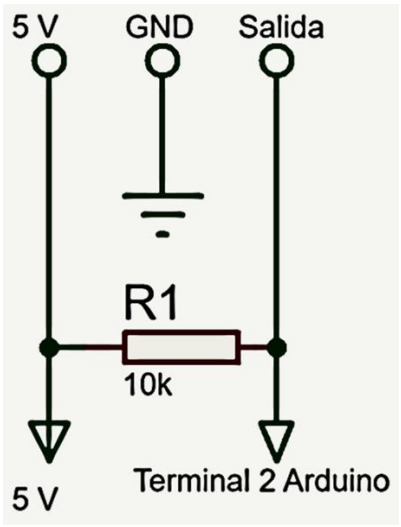


Figura 5. Diagrama de conexiones del sensor de efecto Hall A3144.

C. Código de Arduino

Abrir el IDE de Arduino y escribir el siguiente código:

```
int entrada_actual, entrada_anterior;
int pwm;
int contador;
unsigned long tiempo_final;
int milisegundos;
void setup() {
  Serial.begin(115200);
  pinMode(2, INPUT);
  contador = 0;
  milisegundos = 1000;
}
void loop() {
  entrada_actual = digitalRead(2);
  if (entrada_actual == 0 && entrada_actual !=
entrada_anterior)
  {
    contador = contador + 1;
  }
```

```
 //fin de contador
 //inicio timer
 if (millis() - tiempo_final >= milisegundos)
 {
   pwm = Serial.parseInt();
   analogWrite(3, pwm);
   Serial.print(contador);
   Serial.println();
   tiempo_final = millis();
   contador = 0;
 }
 entrada_anterior = entrada_actual;
}
```

Este programa recibe el valor del PWM mediante el comando *Serial.parseInt()*, lo manda como salida PWM por el puerto digital 3 y con ayuda del comando *millis()* cuenta el número de veces que el sensor detecta el imán en un segundo para posteriormente mandar este valor por el puerto serial hacia la interfaz. Este programa no tiene un *delay()* al final debido a que el mismo programa manda el valor de la rapidez cada segundo.

D. Adquisición de datos

El programa de LabVIEW recibe la rapidez angular en revoluciones por segundo, proveniente del Arduino a través de la comunicación serie (ver "Introducción al entorno de desarrollo de LabVIEW"), estos datos se muestran en un indicador numérico. No utiliza el bloque *VISA Bytes at Serial Port* porque este fue reemplazado por una constante numérica para reducir el tiempo de muestreo. El valor de la constante depende del número de bits a leer. Es importante que la constante sea igual o mayor al número de bits de entrada para que pueda leer toda la información. En caso de exceder el número de bits a leer simplemente incrementará el tiempo de muestreo.

La interfaz posee un slider, el cual sirve para mandar el valor del PWM al Arduino para posteriormente modificar la rapidez del motor. Este programa no tiene ningún bloque para crear algún retardo entre cada iteración, ya que estamos trabajando con la rapidez máxima de LabVIEW.

La interfaz muestra la rapidez del motor DC en diferentes unidades: en revoluciones por segundo, radianes por segundo y revoluciones por minuto. La rapidez se muestra con distintos tipos de indicadores: tipo *Gauge* (en varios diseños) e indicadores numéricos. Hay un *Waveform Char*, el cual permite graficar la rapidez en revoluciones por segundo. Además el slider cuenta con un indicador numérico.

Finalmente se agregan los bloques para guardar los datos en un documento de texto (ver "Introducción al entorno de desarrollo de LabVIEW").

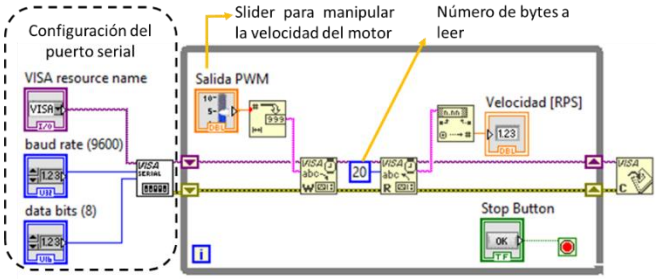


Figura 6. Diagrama de bloques para la comunicación serial.

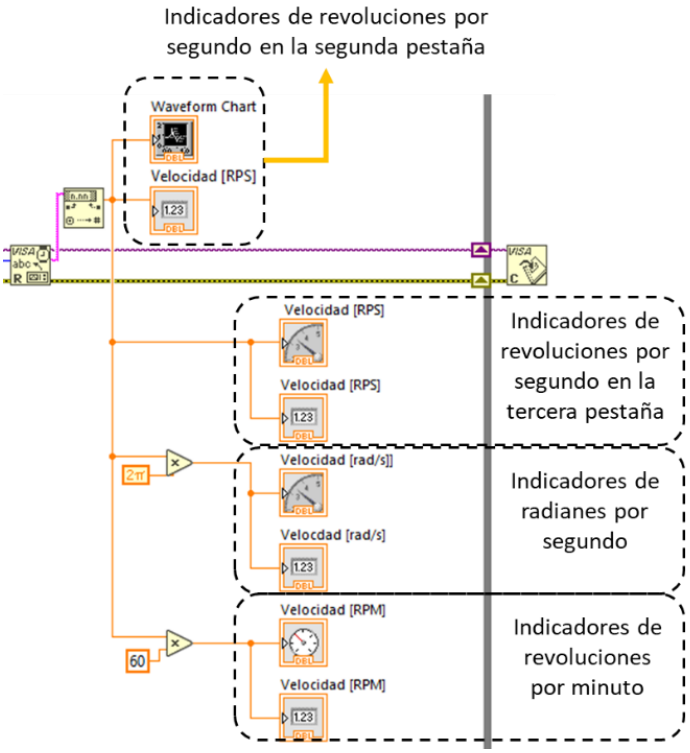


Figura 7. Indicadores de rapidez.

E. Diseño final

A continuación se muestra el diagrama de bloques completo.

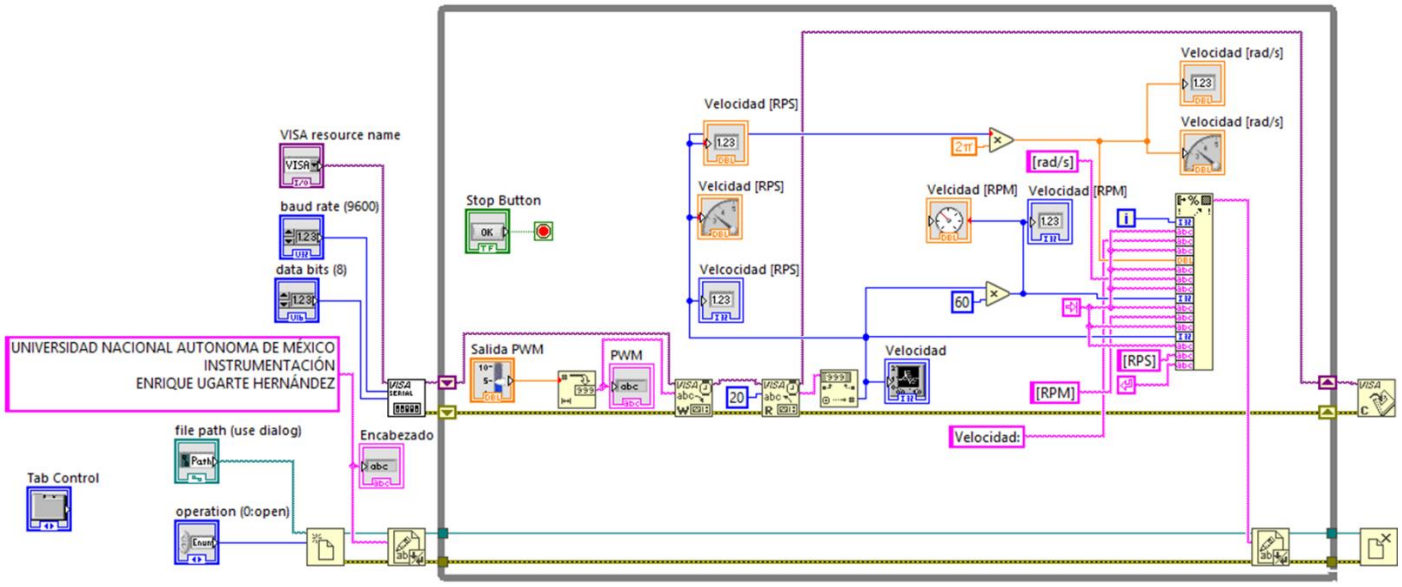


Figura 8. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes se encuentran distribuidos de la siguiente manera:

- Pestana 1: Los componentes relacionados con la comunicación serie.
- Pestana 2: El slider para manipular la rapidez angular del motor, el indicador y *Wavechart* de la rapidez en revoluciones por segundo.
- Pestana 3: Rapidez del motor DC en diferentes unidades: en revoluciones por segundo, radianes por segundo y revoluciones por minuto. La rapidez se muestra con varios indicadores tipo Gauge e indicadores numéricos.
- Pestana 4: Los elementos para guardar los datos en un documento de texto.

Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (cuarta pestaña). También se tiene que esperar unos segundos después de correr el programa para que la tarjeta Arduino y LabVIEW se sincronicen para comenzar a realizar las pruebas.

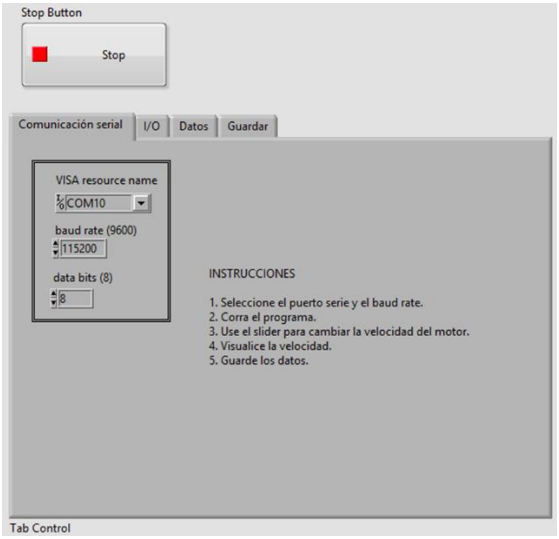


Figura 9. Primera página del contenedor con pestañas.

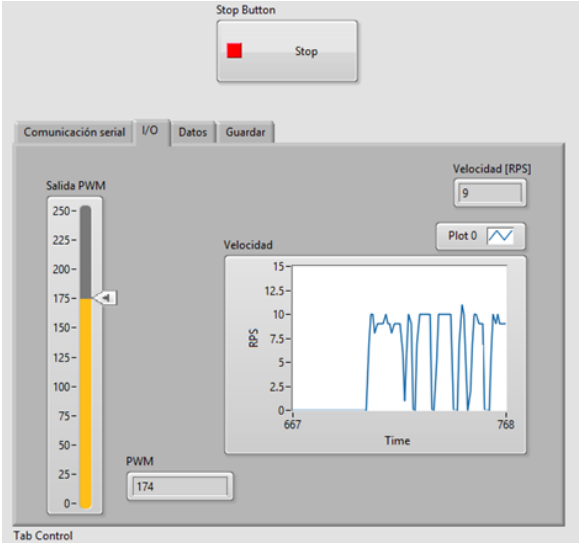


Figura 10. Segunda página del contenedor con pestañas.

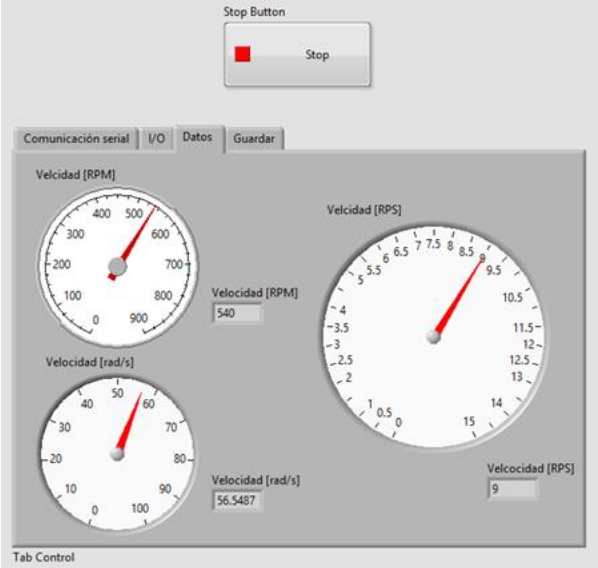


Figura 11. Tercera página del contenedor con pestañas.

Figura 12. Cuarta página del contenedor con pestañas.

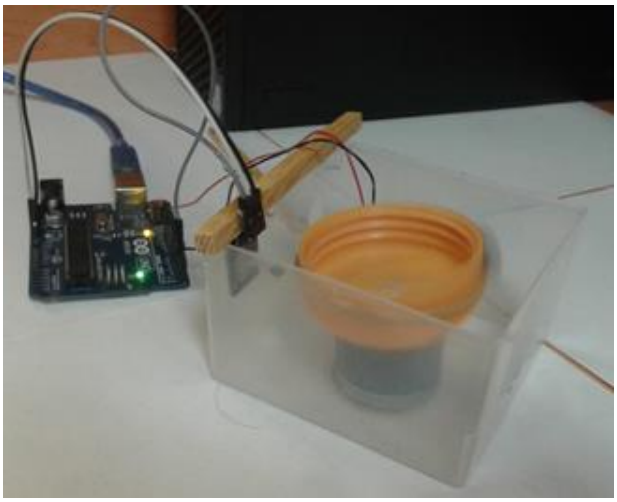


Figura 13. Pruebas con el sensor de efecto Hall.

Archivo Edición Formato Ver Ayuda

```
UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO
INSTRUMENTACIÓN
ARTURO RONQUILLO ARVIZU
0       Velocidad:      62.831853       [rad/s]         600     [RPM]   10      [RPS]
1       Velocidad:      0.000000        [rad/s]         0       [RPM]   0       [RPS]
2       Velocidad:      0.000000        [rad/s]         0       [RPM]   0       [RPS]
3       Velocidad:      0.000000        [rad/s]         0       [RPM]   0       [RPS]
4       Velocidad:      43.982297       [rad/s]         420     [RPM]   7       [RPS]
5       Velocidad:      69.115038       [rad/s]         660     [RPM]   11      [RPS]
6       Velocidad:      62.831853       [rad/s]         600     [RPM]   10      [RPS]
7       Velocidad:      31.415927       [rad/s]         300     [RPM]   5       [RPS]
8       Velocidad:      0.000000        [rad/s]         0       [RPM]   0       [RPS]
9       Velocidad:      12.566371       [rad/s]         120     [RPM]   2       [RPS]
10      Velocidad:      43.982297       [rad/s]         420     [RPM]   7       [RPS]
11      Velocidad:      62.831853       [rad/s]         600     [RPM]   10      [RPS]
12      Velocidad:      62.831853       [rad/s]         600     [RPM]   10      [RPS]
13      Velocidad:      56.548668       [rad/s]         540     [RPM]   9       [RPS]
14      Velocidad:      56.548668       [rad/s]         540     [RPM]   9       [RPS]
```

Figura 14. Datos dentro del documento de texto

V. FUENTES DE CONSULTA

[1] Carbonell G., Angélica et al. (2011). Sensores Magnéticos. España. *Ministerio de Educación, Cultura y Deporte*. Recuperado de http://recursostic.educacion.es/secundaria/edad/4esotecnologia/quincena11/4quincena11_contenidos_3f.htm .

[2] Zuffli. (2017). Módulo Sensor Magnético Efecto HALL 49E KEYES. Puebla, México. *Teslabem*. Recuperado de http://teslabem.com/productos/modulos-breakouts/modulo-sensor-magnetico-efecto-hall-ky.html .

Práctica 11. Sensor de temperatura y humedad

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- Es este trabajo se explica cómo utilizar de forma práctica el sensor DHT11, el cual sirve para medir temperatura y humedad en el ambiente, utilizando la tarjeta Arduino UNO para la adquisición de datos y el entorno de desarrollo de LabVIEW para la realización de una interfaz gráfica que permite visualizar la información de manera clara para el usuario.**

I. INTRODUCCIÓN

A. Sensores capacitivos

Los sensores capacitivos, a diferencia de los inductivos, que sólo detectan metales, son capaces de detectar cualquier material cuya constante dieléctrica sea mayor que la del aire [1]. Su funcionamiento se basa en un circuito oscilante RC y las líneas del campo eléctrico que fluyen a través del aire. De esta manera la aproximación de un material con una constante dieléctrica mayor que el aire provocará el desequilibrio del circuito y el inicio de las oscilaciones [1].

Son sensibles a la mayoría de los líquidos y materiales sólidos, permiten la detección de objetos a través de paredes no conductoras, por ejemplo: detectar agua dentro de un tubo de plástico [1].

B. Funcionamiento y características

Una de las ventajas que nos ofrece el DHT11, además de medir la temperatura y la humedad con un solo dispositivo, es que es digital. A diferencia de sensores como el LM35, este sensor utiliza una terminal digital para enviar la información y por lo tanto, se encuentra mejor protegido del ruido.

Aunque se conecte a un pin digital, se trata de un dispositivo analógico. Dentro del propio dispositivo se hace la conversión entre analógico y digital [2].

Por lo tanto, parte de una señal analógica que luego es convertida en formato digital y se envía al microcontrolador. El DHT11 manda una serie de pulsos repartidos en 5 conjuntos de 8 bits, distribuidos de la siguiente manera:

"8 bits humedad entero + 8 bits humedad decimal + 8 bits temperatura entero + 8 bits fracción de temperatura + 8 bits suma de todos los bits". La manera más sencilla de utilizarlo es con su biblioteca. Para su conexión requiere una resistencia pull-up [3].

Se recomienda no utilizar un cable de alimentación mayor a 20 m, ya que podría generar errores en la medición debido al incremento de la resistencia del cable [3].

II. OBJETIVOS

- ✓ Aprender a utilizar de forma práctica el sensor de temperatura y humedad.
- ✓ Desarrollar una aplicación en LabVIEW que permita visualizar la temperatura y humedad medida por el sensor utilizando un contenedor con pestañas, distribuyendo los componentes de cada tipo dentro de las pestañas.

III. MATERIAL

- Sensor DHT11.
- 2 Resistores de 10 kΩ.
- 4 Jumpers M-H y alambre.
- Tarjeta Arduino UNO.
- Computadora con el software IDE de Arduino y LabVIEW.

Nota: En caso de conseguir el sensor montado en una placa no necesitará los resistores de 10 kΩ debido a que la placa ya viene con los resistores.

IV. DESARROLLO

A. Conexión

a. Sensor montado en placa

- Vcc → 5 [V].
- Signal → pin 2.
- Ground → GND.

Figura 1. Sensor DHT11 montado en placa.

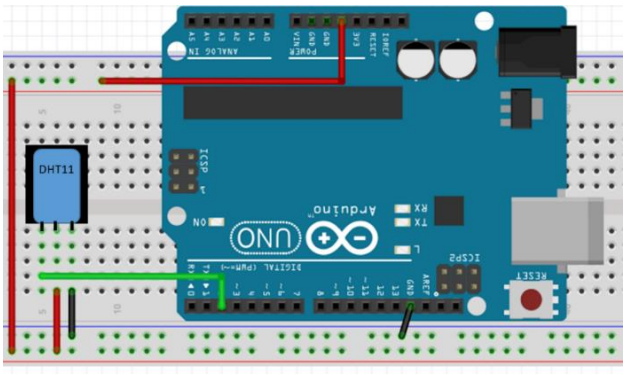


Figura 2. Conexión del sensor DHT11 montado en placa.

b. Sensor solo

- Vcc → 5 V.
- Signal → 2 Resistencias de 10kΩ en paralelo → 5V.
- Signal → Terminal 2 del Arduino.
- Ground → GND.

Figura 3. Sensor DHT11 solo.

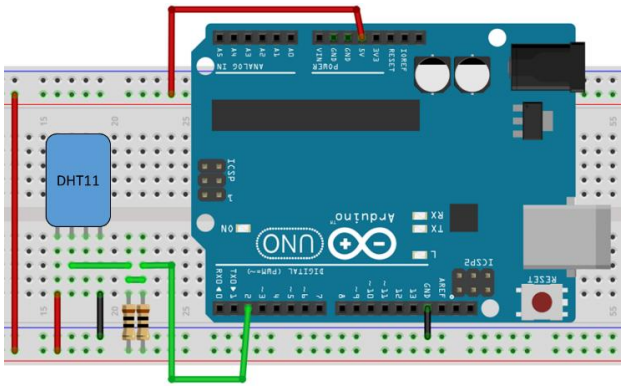


Figura 4. Conexión del sensor DHT11 solo.

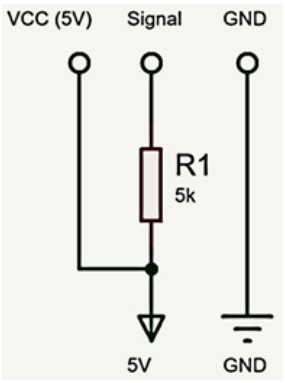


Figura 5. Esquema de conexión del sensor DHT11.

B. Código de Arduino

La manera más fácil de utilizar este sensor es con su biblioteca, existen muchos sitios donde se puede conseguir, aquí se presenta uno:

Descargar la biblioteca [3]:https://www.prometec.net/wp-content/uploads/2014/10/DHT11.zip

Agregar la paquetería al IDE de Arduino.

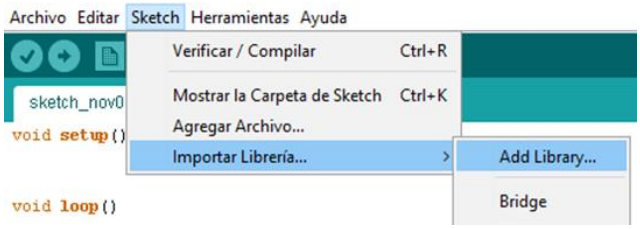


Figura 6. Importación de la biblioteca.

Con esto ya es posible agregar la biblioteca a nuestro código.

Abrimos el IDE de Arduino e ingresamos el siguiente código [4]:

```
#include <DHT11.h>
int pin=2;
DHT11 dht11(pin);
void setup()
  {
    Serial.begin(9600);
  }
void loop()
  {
    int error;
    float temperatura, humedad;
    if((error = dht11.read(humedad, temperatura)) == 0)    // Si devuelve 0 es que ha leído bien
      {
        Serial.print(int(temperatura));
        Serial.print(',');
        Serial.print(int(humedad));
        Serial.println();
      }
    delay(1000);
    }
```

Este código manda la temperatura y humedad a través del puerto serie, separados por una coma. La temperatura esta en °C.

C. Adquisición de datos

El programa de LabVIEW recibe los datos del Arduino a través de la comunicación serie (ver "Introducción al entorno de desarrollo de LabVIEW"), después se separa la información con el bloque *Match Pattern* y finalmente se muestra en indicadores numéricos.

El programa contiene los bloques para guardar los datos dentro de un documento de texto (ver "Introducción al entorno de desarrollo de LabVIEW).

A. Diseño final

A continuación se muestra el diagrama de bloques completo.

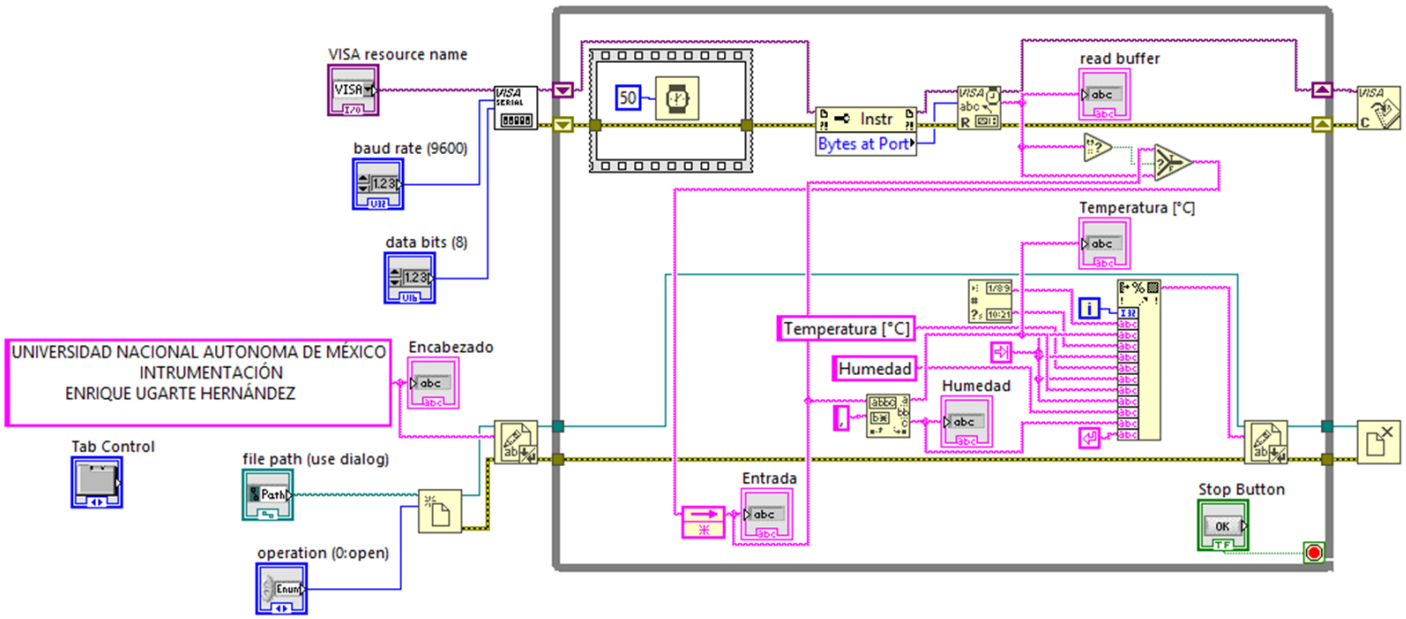


Figura 7. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña, la visualización de los datos en la segunda y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

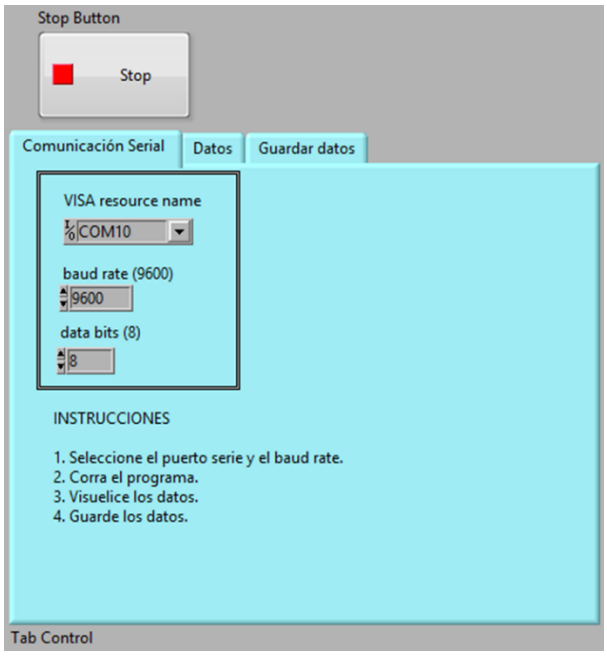


Figura 8. Primera página del contenedor con pestañas.

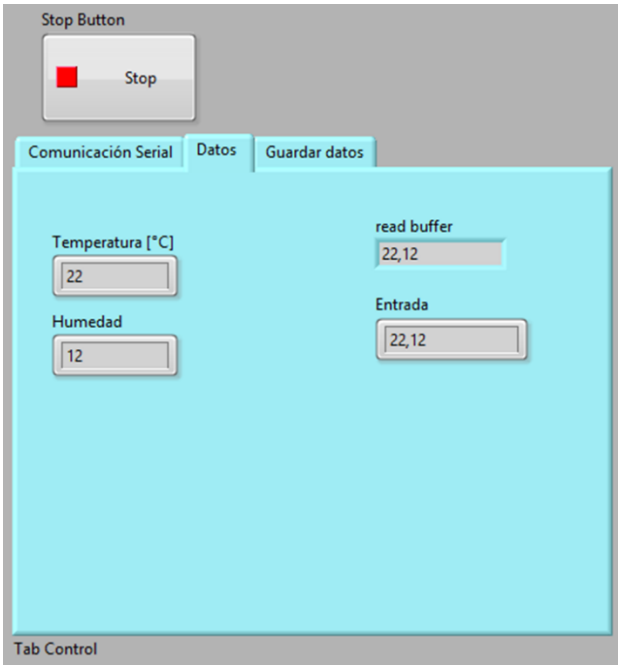


Figura 9. Segunda página del contenedor con pestañas.

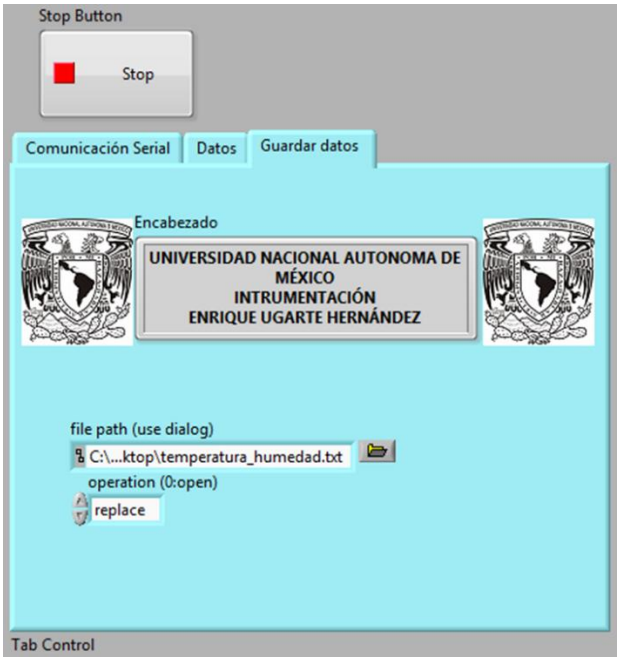


Figura 10. Tercera página del contenedor con pestañas.

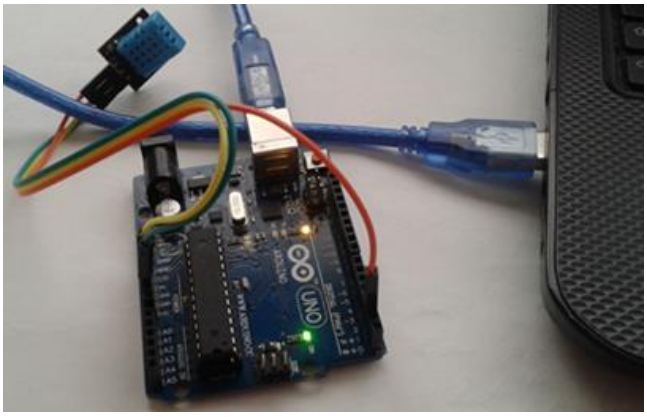


Figura 11. Pruebas con el sensor de temperatura y humedad.

Archivo Edición Formato Ver Ayuda

UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO
INTRUMENTACIÓN
ARTURO RONQUILLO

0 05/12/2017	10:24 a. m.	Temperatura [°C]	22	Humedad 36
1 05/12/2017	10:24 a. m.	Temperatura [°C]	21	Humedad
2 05/12/2017	10:24 a. m.	Temperatura [°C]	21	Humedad 34
3 05/12/2017	10:24 a. m.	Temperatura [°C]	21	Humedad 34
4 05/12/2017	10:24 a. m.	Temperatura [°C]	21	Humedad 34
5 05/12/2017	10:24 a. m.	Temperatura [°C]	21	Humedad 34
6 05/12/2017	10:24 a. m.	Temperatura [°C]	21	Humedad 34
7 05/12/2017	10:24 a. m.	Temperatura [°C]	21	Humedad 34
8 05/12/2017	10:24 a. m.	Temperatura [°C]	21	Humedad 35
9 05/12/2017	10:24 a. m.	Temperatura [°C]	21	Humedad 35
10 05/12/2017	10:24 a. m.	Temperatura [°C]	22	Humedad 35
11 05/12/2017	10:24 a. m.	Temperatura [°C]	22	Humedad 35
12 05/12/2017	10:24 a. m.	Temperatura [°C]	22	Humedad 35

Figura 12. Datos dentro del documento de texto.

V. FUENTES DE CONSULTA

[1] Hosting Joomla Web Empresa. (2018) Barcelona, España. *Sensorstecnic & Semiconductors*. Recuperado de http://www.sensorstecnics.net/es/productos/category/96/sensores-y-transmisores/sensores-capacitivos .

[2] Del Valle Hernández, Luis (2018). Cómo utilizar el sensor DHT11 para medir la temperatura y humedad con Arduino *programarfacil.com*. Recuperado de https://programarfacil.com/blog/arduino-blog/sensor-dht11-temperatura-humedad-arduino/ .

[3] Aosong (Guangzhou) Electronics Co.,Ltd. *DHT11 Product Manual.Temperature and humidity module*. Recuperado de https://akizukidenshi.com/download/ds/aosong/DHT11.pdf.

Práctica 12. Secuencia de un semáforo con relevadores

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En esta práctica se fabrica un semáforo simple y se realiza la secuencia utilizando un módulo de relevadores, una tarjeta Arduino UNO y el entorno de desarrollo de LabVIEW para generar una interfaz gráfica que monitoriza la activación de las luces y permita ajustar su tiempo de encendido.**

I. INTRODUCCIÓN

En algunas aplicaciones es indispensable contar con una interfaz de usuario y en otras no, depende de las necesidades del proyecto. Los semáforos que controlan el flujo vehicular en las esquinas de las calles de casi todas las ciudades no cuentan con una interfaz de usuario, por no ser indispensable y solo basta con un pequeño cerebro donde se encuentra almacenado el programa que coordina los tiempos de encendido y apagado de las luces.

En la presente práctica se desarrolla una interfaz de usuario en LabVIEW para tener opción de controlar el tiempo de encendido y apagado de las luces de un semáforo simple a través de un módulo de relevadores. La idea de controlar un semáforo simple con un módulo de relevadores a través de una interfaz en LabVIEW surgió del hecho de que los estudiantes pueden transportar fácilmente tres focos (semáforo simple) al salón de clase; sin embargo, es posible sustituir los focos por algún actuador, por ejemplo: motores de corriente alterna de 127V o 220V, válvulas, pistones, etcétera. Cabe mencionar que estos ejemplos no son tan fáciles de transportar al salón de clase y requieren relevadores que puedan soportan más tensión y corriente que los relevadores que normalmente se usan para prototipos escolares (5 V, 10 V y 15 V). Al desarrollar esta práctica se pretende que el alumno pueda implementar sus propias interfaces de usuario para controlar actuadores de distintos tipos a través de relevadores, según las necesidades a las que se enfrente.

A. Relevador

El relé (en francés, *relais*, "relevo" o en inglés, *relay*) o relevador es un dispositivo electromagnético o de estado sólido. Funciona como un interruptor controlado por un circuito eléctrico en el que, por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes.

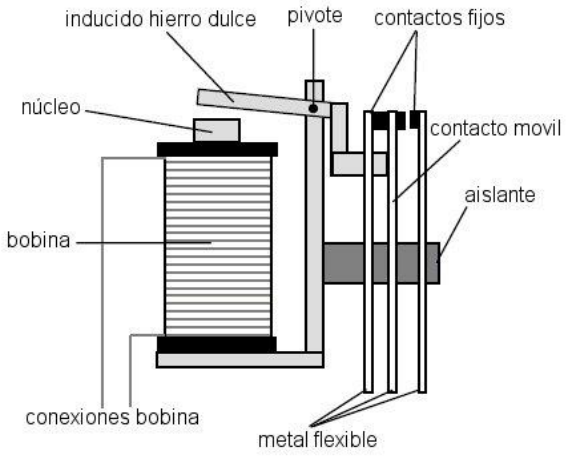


Figura 1. Partes de un relevador electromecánico [1].

El electroimán hace girar la armadura verticalmente al ser alimentada, cerrando los contactos dependiendo de si es NA o NC (normalmente abierto o normalmente cerrado). Si se le aplica un voltaje a la bobina se genera un campo magnético, que provoca que los contactos hagan una conexión. Estos contactos pueden ser considerados como el interruptor, que permite que la corriente fluya entre los dos puntos que cerraron el circuito. Ver Figura 1.

B. Relevador de estado sólido

Se llama relé de estado sólido a un circuito híbrido, normalmente compuesto por un optoacoplador que aísla la entrada, un circuito de disparo, que detecta el paso por cero de la corriente de línea y un triac o dispositivo similar que actúa de interruptor de potencia. Su nombre se debe a la similitud que presenta con un relé electromecánico; este dispositivo es usado generalmente para aplicaciones donde se presenta un uso continuo de los contactos del relé que en comparación con un relé convencional generaría un serio desgaste mecánico, además de poder conmutar altos amperajes que en el caso del relé electromecánico destruirían en poco tiempo los contactos. Estos relés permiten una velocidad de conmutación muy superior a la de los relés electromecánicos [2]; sin embargo, también tienen algunas desventajas, por ejemplo: Con circuito cerrado, mayor resistencia (pérdidas en forma de calor); en abierto, menor resistencia, con una pequeña corriente inversa de pérdida (del orden de µA); posibilidad de falsas conmutaciones debido a cargas transitorias, debido a la capacidad de conmutación mucho más rápida; entre otras.

Figura 2. Relevador de estado sólido [3].

II. OBJETIVOS

- ✓ Conocer el funcionamiento de los relevadores y su aplicación en la activación de actuadores.
- ✓ Fabricar un semáforo sencillo.
- ✓ Realizar un programa para controlar tres relevadores desde una interfaz de LabVIEW, aplicándolo a un semáforo simple.

III. MATERIAL

- ➢ 3 Focos de 100 [W] 127 [V] de corriente alterna.
- ➢ 3 sockets.
- ➢ 1 Módulo de cuatro relevadores o 3 relevadores.
- ➢ 10 Jumpers macho-hembra.
- ➢ 1 Tarjeta Arduino UNO.
- ➢ 1 Computadora con software LabVIEW 2012 o superior e IDE de Arduino.
- ➢ Cable calibre 12 AWG (2 metros).
- ➢ 1 Clavija.
- ➢ Material para construir el semáforo.
- ➢ Fuente de alimentación (opcional).
- ➢ Materiales para una construir etapa de potencia (opcional).

IV. DESARROLLO

A. Conexión

Para la conexión entre el Arduino y los focos al módulo de relevadores se sigue el siguiente esquema:

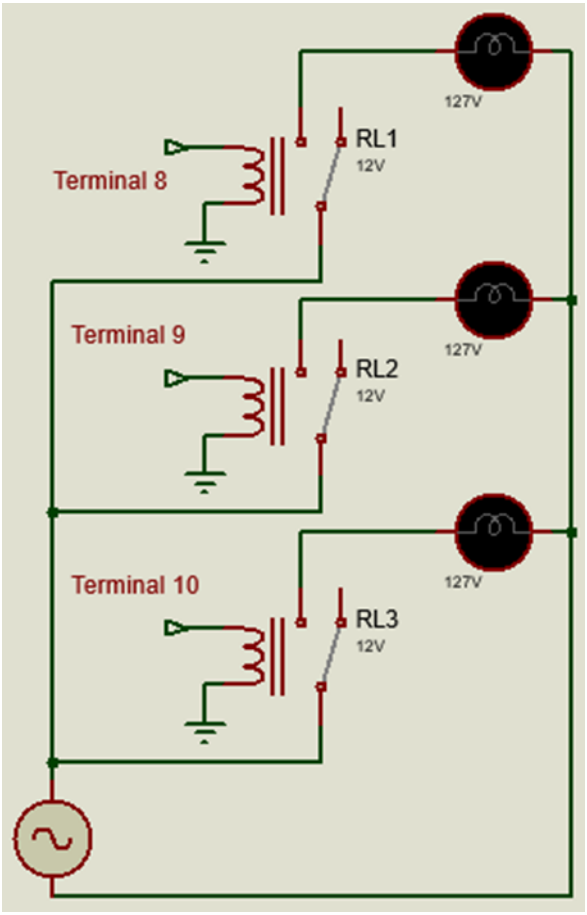


Figura 3. Es quema de conexiones del módulo de relevadores.

Los focos son conectados como se muestra en el diagrama de la figura 3, utilizando los contactos de los relevadores normalmente abiertos.

Conectar el módulo de relevadores (IN1, IN2 e IN3) a la tarjeta Arduino. Se utilizan las salidas digitales (8, 9 y 10) del Arduino para cada una de las tres lámparas del semáforo.

Terminal digital del Arduino	Módulo de relevadores	Salida
8	IN1	Rojo
9	IN2	Amarillo
10	IN3	Verde

Tabla 1. Conexión del Arduino al módulo de relevadores.

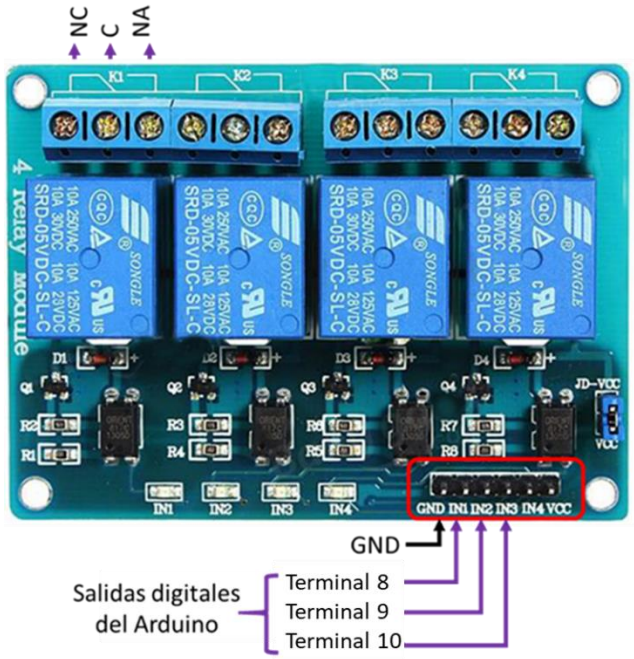


Figura 4. Terminales del módulo de relevadores.

Donde:

NC → Normalmente cerrado
C → Común.
NA → Normalmente abierto.

El relevador es muy sencillo de utilizar, antes de energizar el relevador en las terminales NC (normalmente cerradas) hay continuidad y en las terminales NA (normalmente abiertas) no hay continuidad, cuando el relevador se energiza en las terminales NC deja de haber continuidad y en las terminales normalmente abiertas hay continuidad. Para energizar el módulo de relevadores, se puede usar la fuente de alimentación de la tarjeta de Arduino (5V, DC y GND) o una fuente externa de corriente directa.

Nota: Si se usan relevadores, cuyo electroimán requiera un voltaje superior a 5 [V], es necesario diseñar una etapa de potencia que se ajuste a cada caso.

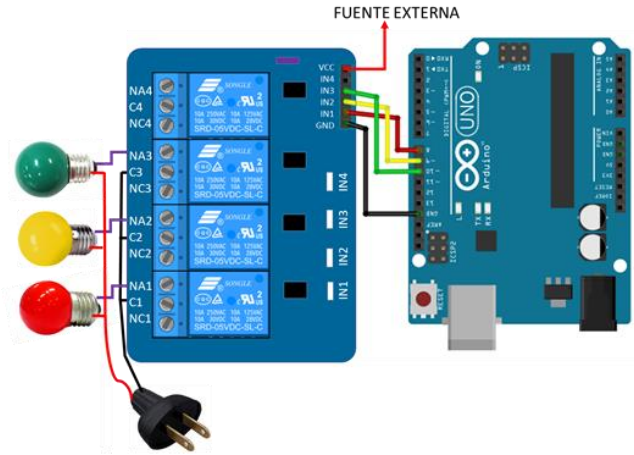


Figura 5. Conexión completa del módulo de relevadores al Arduino y al circuito eléctrico de los focos.

B. Construcción del modelo

Fabricar un semáforo sencillo con los focos y la clavija similar al mostrado en las figuras 6 y 7. Se deja al criterio del alumno los materiales a utilizar.

Figura 6. Conexiones del circuito a los relevadores.

Figura 7. Prototipo terminado.

V. PROGRAMACIÓN

Al igual que en las otras prácticas presentes en este manual, siempre que se utiliza el toolkit de Arduino en LabVIEW, se tiene que cargar el LIFA_Base (figura 9) al Arduino, esto permite realizar la comunicación serie entre la tarjeta Arduino y LabVIEW. El programa se ubica en ***C:\Program Files\National Instruments\LabVIEW 2016\vi.lib\LabVIEW Interface for Arduino\Firmware\LIFA_Base***.

Antes de cargar el LIFA_Base al Arduino se tiene que asegurar que el puerto de comunicación seleccionado en el IDE de Arduino es realmente el que la computadora le asignó al Arduino cuando fue conectado (figura 6). Dentro del IDE de

Arduino ir a la barra de menús y dar clic en *Herramientas* → *Puerto.*

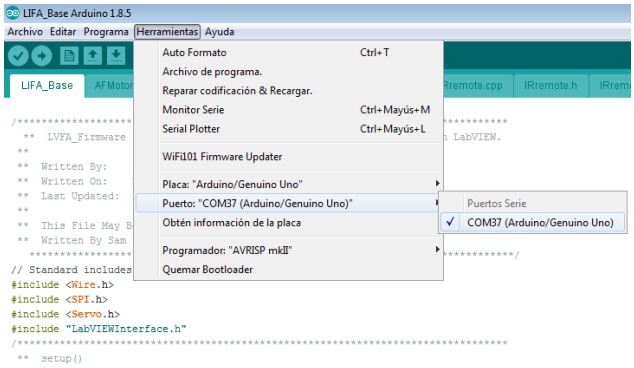


Figura 8. Selección del Puerto en el IDE de Arduino.

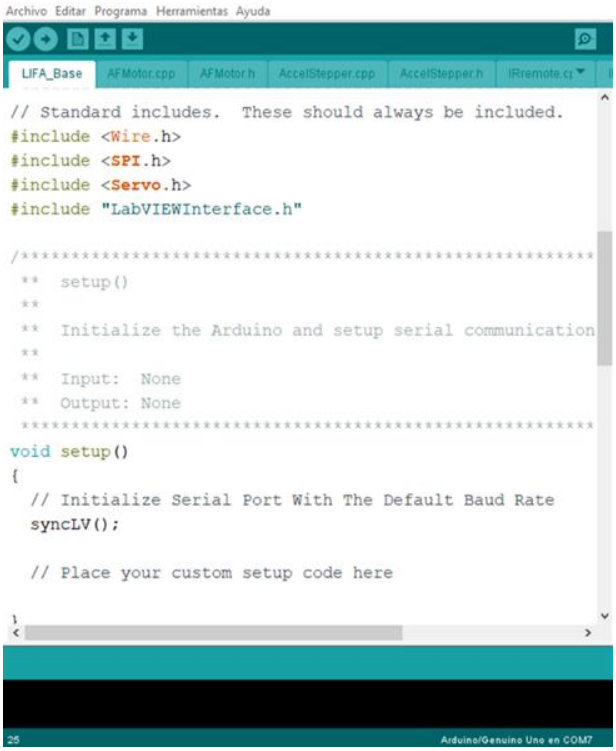


Figura 9. Programa LIFA Base.

C. Bloques para la comunicación

Para abrir y cerrar la comunicación con el Arduino se necesitan los bloques *Init* y *Close*, ubicados en *Funtions* → *Arduino*. Para escribir la secuencia se utilizan 3 bloques *Digital Write Pin*, este se ubica en *Funtions* → *Arduino* → *Low Level*. Se realizan las conexiones dentro de un ciclo *While.*

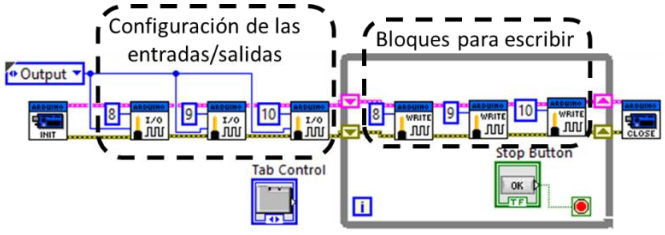


Figura 10. Configuración de las terminales como salidas.

D. Secuencia del semáforo

Este programa usa los registros de corrimiento para cambiar el caso de un *Case* con facilidad. Se usan los registros de corrimiento cuando quiere pasar valores de iteraciones previas a través del ciclo a la siguiente iteración. Un registro de corrimiento aparece como un par de terminales directamente opuestas entre sí en los lados verticales del borde del bucle [4].

La terminal en el lado derecho del bucle contiene una flecha hacia arriba y almacena datos sobre la finalización de una iteración. LabVIEW transfiere los datos conectados al lado derecho del registro a la siguiente iteración. Después de que se ejecuta el bucle, el terminal en el lado derecho del bucle regresa el último valor almacenado en el registro de desplazamiento.

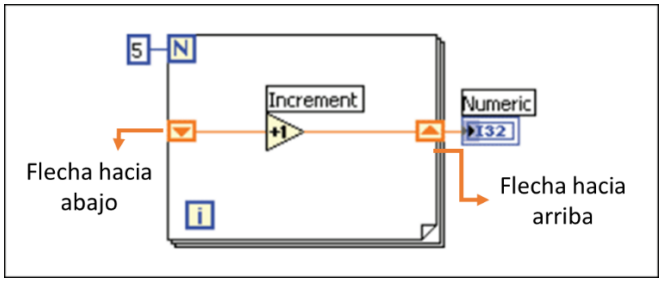


Figura 11. Ejemplo de registro de corrimiento.

Se crea un registro de desplazamiento haciendo clic con el botón derecho en el borde izquierdo o derecho de un bucle y seleccionando Agregar registro de desplazamiento en el menú de acceso directo.

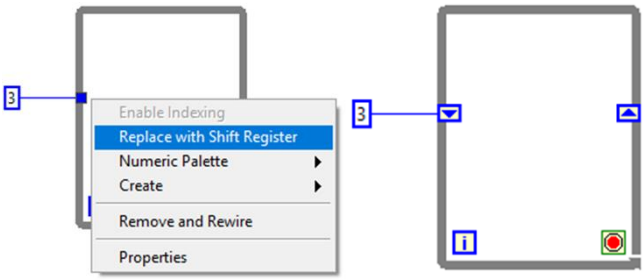


Figura 12. Creación de un registro de corrimiento.

Un registro de corrimiento transfiere cualquier tipo de datos y cambia automáticamente al tipo de datos del primer objeto conectado al registro de desplazamiento. Los datos que conecte a los terminales de cada registro de desplazamiento deben ser del mismo tipo [4].

Se programa un *Case* para definir la secuencia en función de tres estados: Cuando prende el rojo, el verde y el amarillo. Se crean tres controladores numéricos dentro del panel frontal para ajustar el tiempo de encendido de cada foco antes de cambiar.

Para cambiar de caso del *Case* con un registro de corrimiento se utiliza el bloque *Enum constan*, ubicado *en Functions* → *Programming* → *Numeric*. Una vez teniendo el *Enum constant* en la ventana del diagrama de bloques, hacer clic derecho sobre él y seleccionar la opción *Edit ítems (*figura 14). Hacer clic en *Insert* e ingresar los casos del *Case*. Realizar las conexiones; de

esta manera, cada vez que termina el tiempo de duración de un caso (prendido de un foco), manda el siguiente.

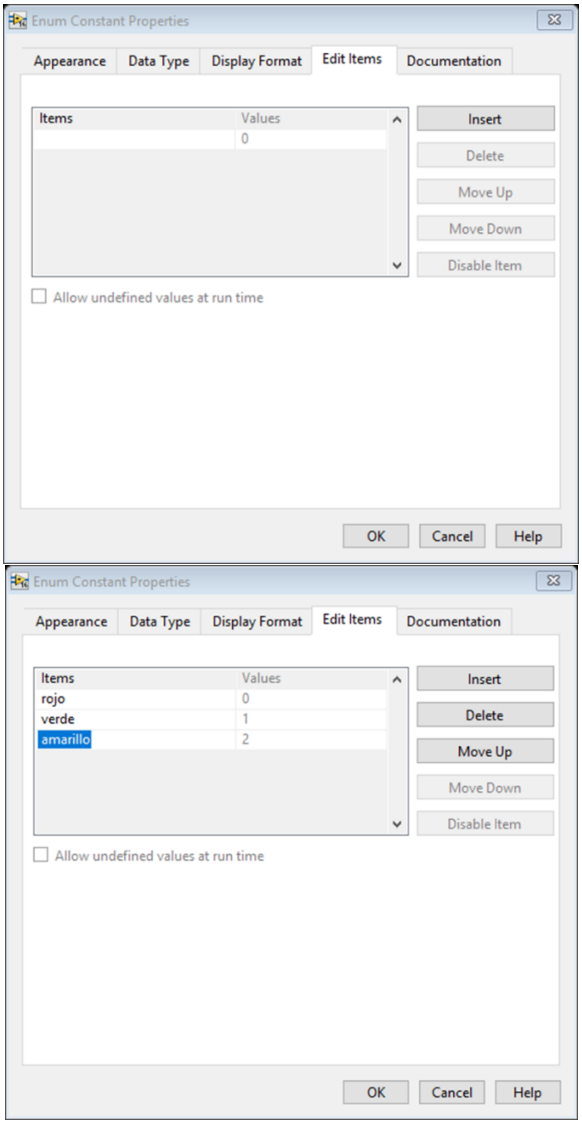


Figura 13. Casos del boque *Enum constant*.

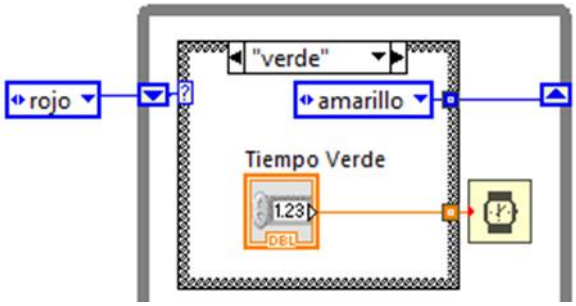


Figura 14. Bloque *Enum constan* dentro del *Case*.

Para crear un caso Default, la entrada del Case no debe ser lógica. Se hace clic derecho sobre el nombre del caso y se selecciona la opción *Make This The Default Case*.s

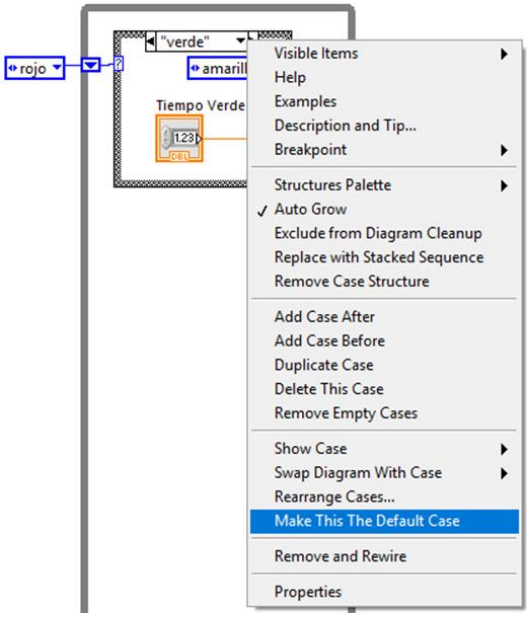


Figura 15. Opciones del *Case*.

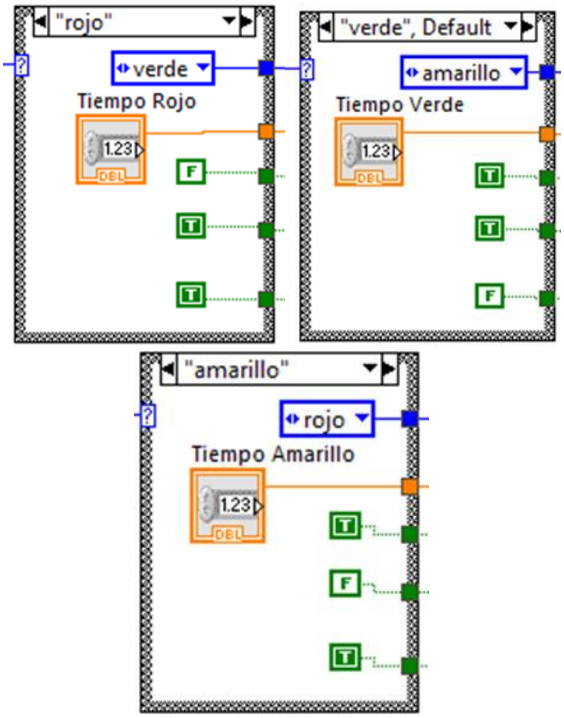


Figura 16. Tres casos de la estructura *Case*, uno para cada luz del semáforo.

Se cambian los datos lógicos de la salida del Case a numéricos para que sea consistente con la entrada de los bloques del escritura de las terminales del Arduino.

E. Diseño final

La interfaz en LabVIEW se diseña como se muestra en la figura 18. En La primera pestaña del contenedor en el panel frontal se dan las instrucciones de uso del programa.

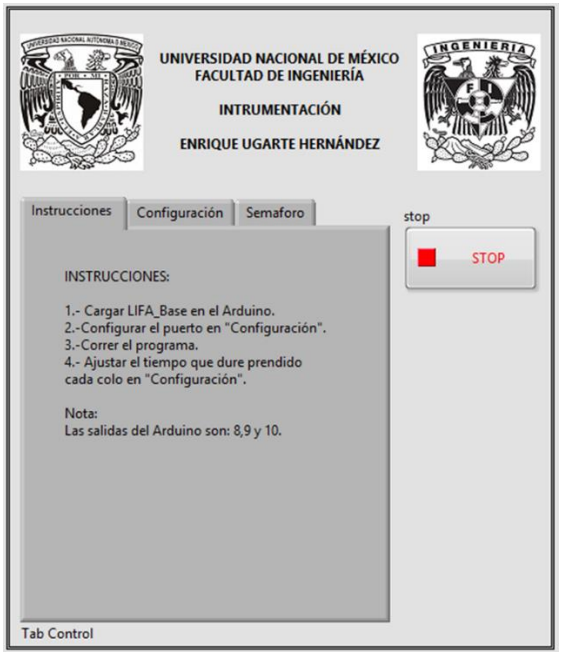


Figura 17. Instrucciones de uso del Programa.

En la segunda pestaña del contenedor se reserva para la configuración de la comunicación serie, el *baud rate* se dejó como una constante en el diagrama de bloques. También tenemos en esta pestaña los controles numéricos para introducir los tiempos de activación de las luces del semáforo. Ver Figura 19.

Figura 18. Configuración de la comunicación serie y tiempos de activación y desactivación de las luces del semáforo.

A continuación se muestra el diagrama de bloques completo.

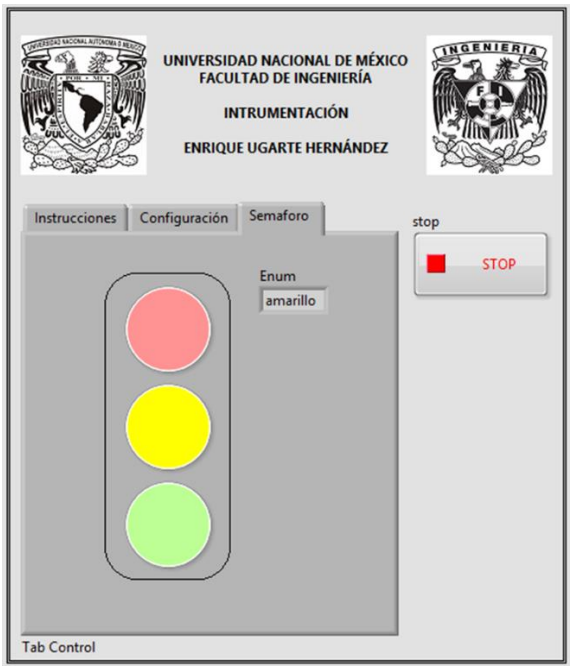


Figura 19. Secuencia de encendido de las luces.

En la última pestaña del contenedor se puede observar la secuencia de encendido de luces del semáforo (figura 20).

El diseño del panel frontal puede variar dependiendo del ingenio de los estudiantes; sin embargo es recomendable que el panel frontal contenga las instrucciones de uso del programa, la configuración de la comunicación serie, los controles e indicadores de nuestro programa, etcétera. El uso de un contenedor del tipo *Tab Control* permite organizar y ahorrar espacio en el panel frontal de nuestro programa.

Nota: Para este caso las luces prenden al mandar un 0 logico, razón por cual, hay compuertas NOT antes de los indicadores lógicos.

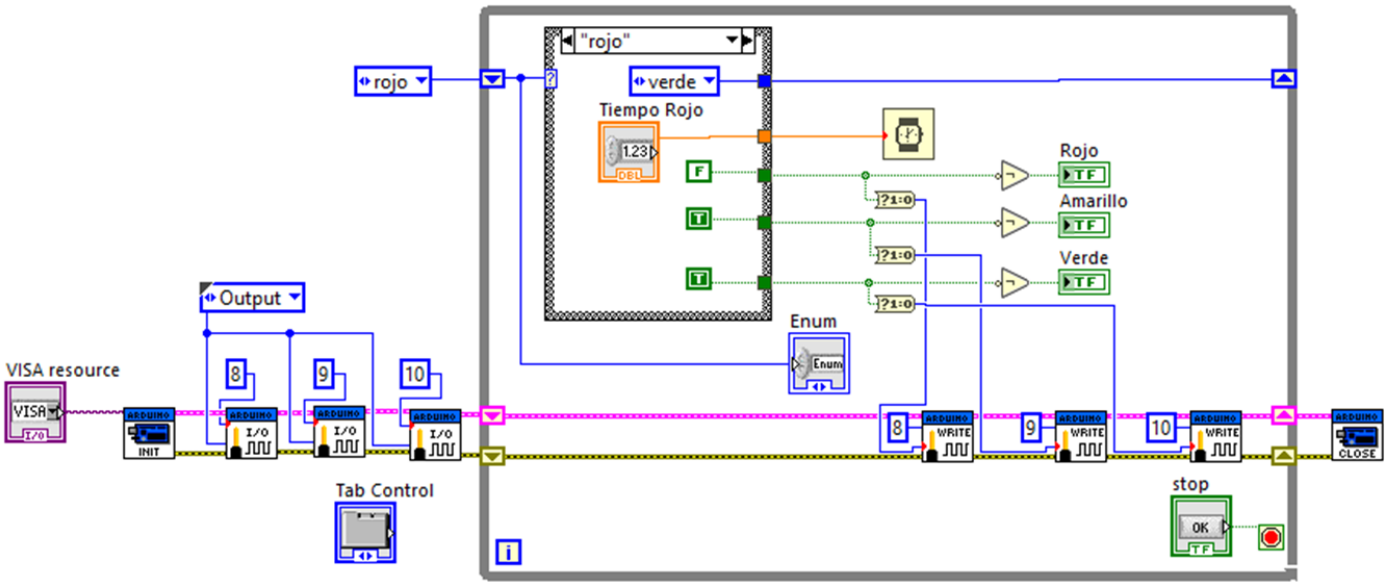


Figura 20. Diagrama de bloques del programa en LabVIEW.

VI. FUENTES DE CONSULTA

[1]Automatización y electrónica. (2014), Relevadores. [Online]. Recuperado de: https://www.dirind.com/dae/monografia.php?cla_id=18 .

[2] Wikipedia. (2018). Relé. *Wikipedia.org*. Recuperado de https://es.wikipedia.org/wiki/Rel%C3%A9#Rel%C3%A9_de_estado_s%C3%B3lido

[3] NUWAVE AUTOMATION. (2012), Solid State Relay – Random Firing (Instant On). *Nuwaveautomation.com*. Recuerado de https://nuwaveautomation.com/product/solid-state-relay-random-firinginstant-on/

[4] National Instruments. (2018), Passing Data Between Loop Iterations in LabVIEW. *Ni.com*. Recuperado de http://www.ni.com/getting-started/labview-basics/shift-registers.

Práctica 13. Sensor de gas

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*– En este trabajo se explica cómo usar de manera práctica el sensor MQ-2 para medir concentraciones de humo en el ambiente, utilizando la tarjeta Arduino UNO para la adquisición de datos y el entorno de desarrollo de LabVIEW para generar una interfaz gráfica que permita visualizar la información de manera clara para el usuario. Para realizar el experimento quema una hoja de papel; es importante tomar en cuenta las precauciones mencionadas en este texto para evitar cualquier accidente.**

I. INTRODUCCIÓN

Sensores químico-eléctricos

Un sensor electroquímico es un dispositivo químico que responde a cambios específicos en la diferencia de potencial o en la intensidad de corriente eléctrica como consecuencia de la presencia de una especie química (o sustancia pura) que interactúa con él [1].

Descripción de sensor

El MQ-2 es un sensor de gas capaz de detectar GLP, propano, metano, alcohol, hidrógeno, humo. Siendo más sensible al GLP (gas licuado de petróleo) y propano [2]. Puede detectar concentraciones de humo en un rango de 30 a 10,000 ppm (partes por millón) [3], normalmente se consigue montado sobre una placa listo para usarse, cuenta con cuatro terminales: VCC, GND, AO y DO y se alimenta con 5 [V]. La terminal AO saca una señal analógica y la terminal DO una salida digital, la cual puede calibrarse con el trimpot que contiene el sensor.

Figura 1. Sensor MQ-2 solo y montado sobre una placa [4,5].

II. OBJETIVOS

- ✓ Aprender a usar de forma práctica el sensor de CO MQ-2.
- ✓ Fabricar un contenedor capaz de soportar un pequeño fuego controlado para medir las concentraciones de humo.
- ✓ Aprender a realizar programas con el toolkit de Arduino y LINX.
- ✓ Desarrollar una aplicación en LabVIEW que le permita mostrar los datos recibidos por el sensor. La aplicación debe contar con un contenedor con pestañas. Debe usarse una pestaña para los componentes relacionados con la comunicación serie, las necesarias para mostrar a información y una con las opciones para guardar los datos en un archivo de texto.

III. MATERIAL

- Sensor de gas MQ-2.
- Tarjeta Arduino UNO.
- 4 Jumpers M-H.
- Contenedor para el fuego controlado.
- Materiales para sujetar el sensor al contenedor.
- Encendedor o cerillos.
- Una hoja de papel
- Computadora con el software LabVIEW y el IDE de Arduino.

IV. DESARROLLO

A. Conexión

Realizar las siguientes conexiones:

- VCC → 5V.
- GND → GND.
- DO → terminal 2.
- AO → A0.

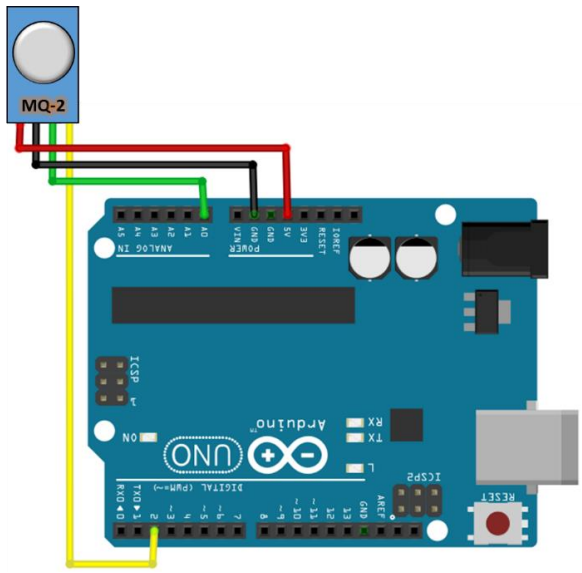


Figura 2. Conexión del sensor

B. Montaje en el contenedor.

Montar el sensor sobre algún material que no se queme con facilidad, ya que para este propósito se va a quemar una hoja de papel. Se deja al criterio del alumno los materiales a usar. Se recomienda utilizar un contenedor cerámico, ya que son resistentes a altas temperaturas.

Figura 3. Montaje del sensor sobre un contenedor resistente.

C. Generación del código de Arduino

Abrir el IDE de Arduino y escribir el siguiente código:

```
void setup() {
Serial.begin(9600);
pinMode(2,INPUT);
}
void loop() {
Serial.print(analogRead(A0));
Serial.print(",");
Serial.println(digitalRead(2));
delay(300);
}
```

Este programa manda los datos analógicos y digitales a través del puerto serie separados por una coma. Al abrir el puerto serie del IDE de Arduino debe aparecer una pantalla similar a la que se muestra continuación:

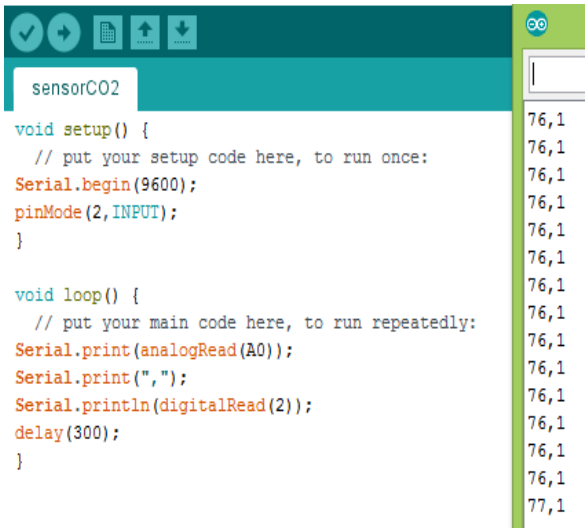


Figura 4. Datos a través del puerto serial.

D. Generación de la Interfaz gráfica.

1) Adquisición de datos

El programa de LabVIEW lee los datos del Arduino a través de la comunicación serie (ver "Introducción al entorno de desarrollo de LabVIEW"), después separa la información con el bloque *Match Pattern*. La lectura analógica entra dentro del bloque *Formula Node*, el cual sirve para realizar los cálculos para obtener la concentración en función de las constantes de cada gas y el valor digital se muestra con un *Round LED*. El bloque *Match Pattern* se encuentra *en Functions → Programming → String* y el bloque *Formula Node* se encuentra en *Functions →Mathematics →Scripts & Formulas*.

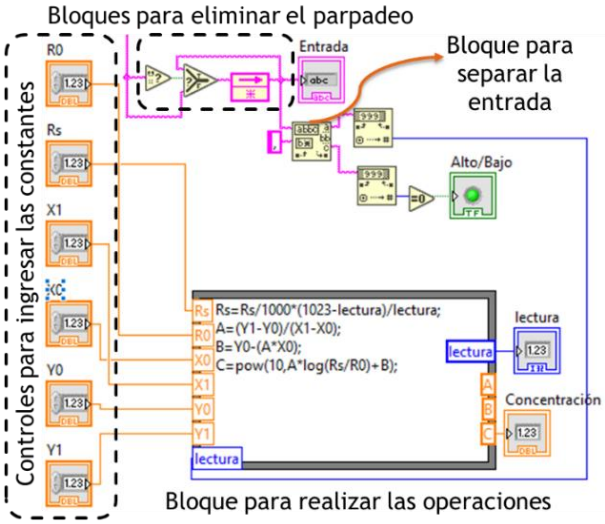


Figura 5. Eliminación del parpadeo, separación de los datos y operaciones dentro *de Formula Node*.

Texto dentro de *Formula Node*:

```
Rs=Rs/1000*(1023-lectura)/lectura;
A=(Y1-Y0)/(X1-X0);
B=Y0-(A*X0);
```

C=pow(10,A*log(Rs/R0)+B);

Finalmente se agregan los bloques para guardar los datos dentro de un documento de texto (ver “Introducción al entorno de desarrollo de LabVIEW”).

2) Diseño final

A continuación se muestra el diagrama de bloques completo.

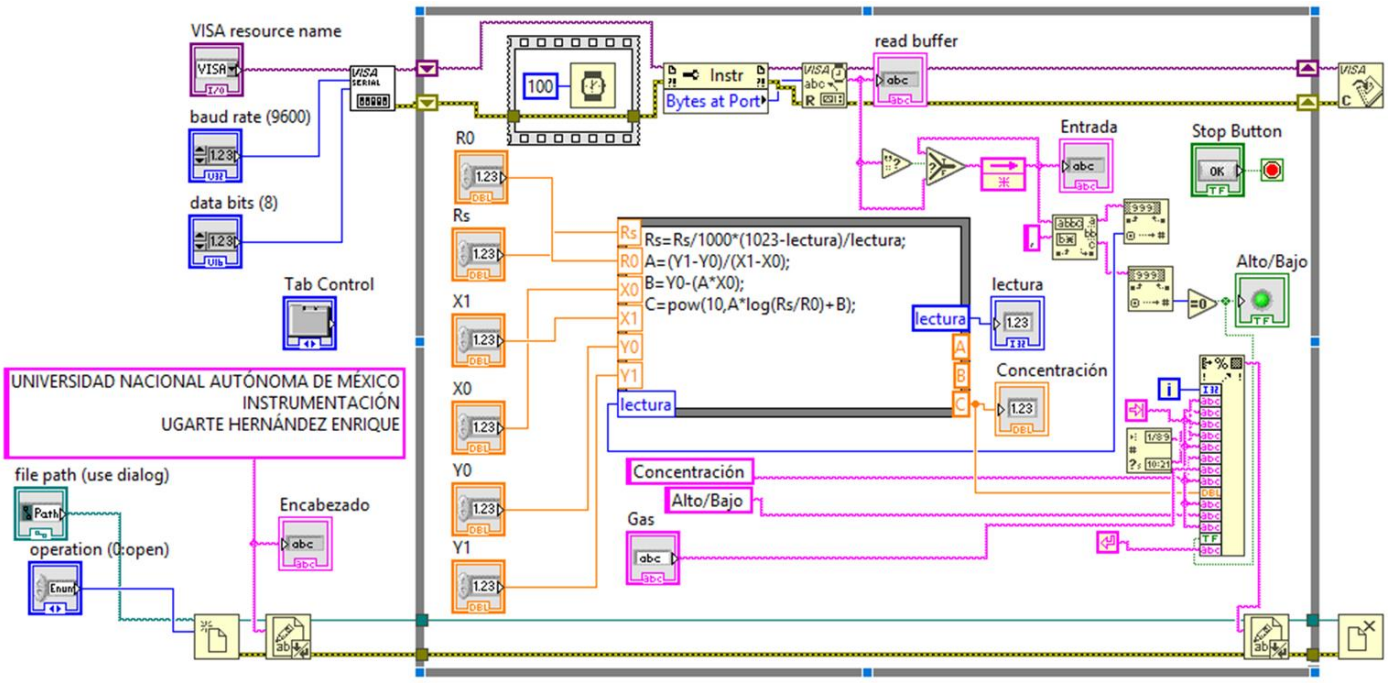


Figura 6. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña, la visualización de los datos en la segunda y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

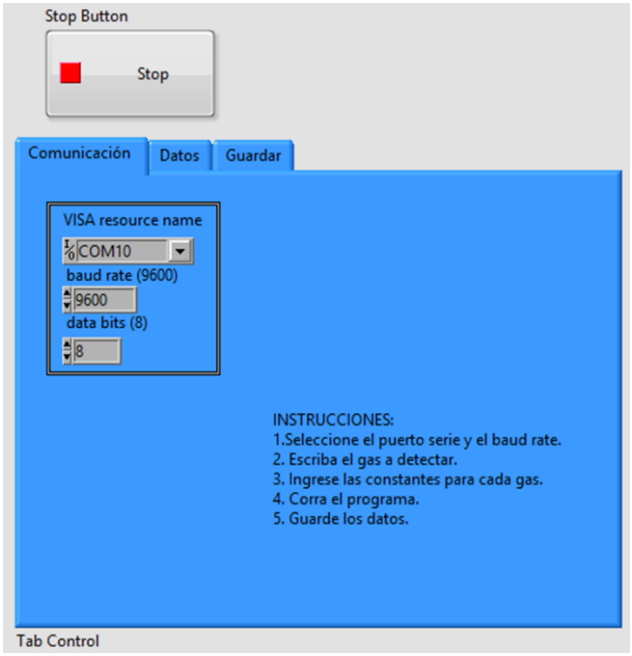


Figura 7. Primera página del contenedor con pestañas.

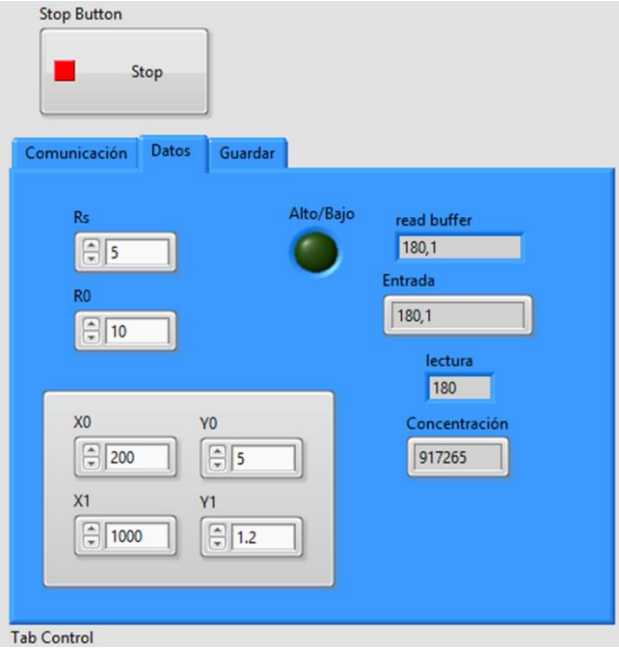


Figura 8. Segunda página del contenedor con pestañas.

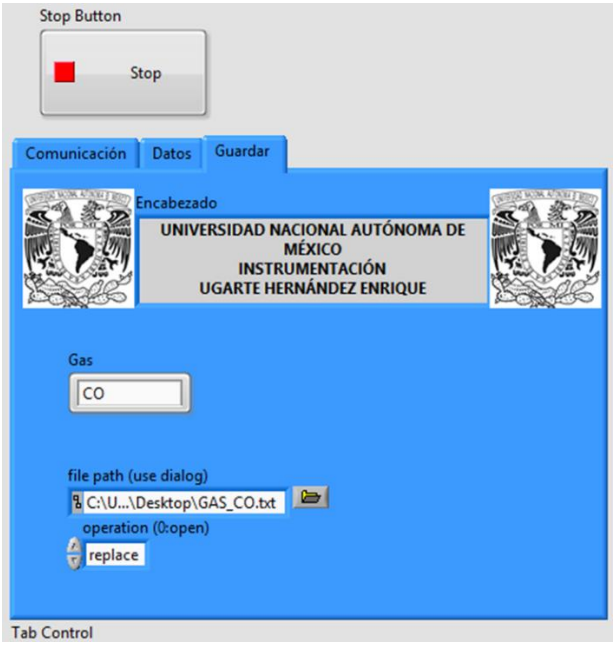


Figura 9. Tercera página del contenedor con pestañas.

3) Obtención de las constantes a partir de la curva logarítmica

A continuación se muestra la gráfica de los gases que es capaz de detectar el sensor MQ-2, esta fue obtenida de su hoja de datos (*datasheet*). Esta gráfica tiene la particularidad que está en escala logarítmica. Para poder conocer la concentración de un gas, primero se tiene que localizar la curva del gas en cuestión. Una vez ubicada la curva se seleccionan 2 puntos.

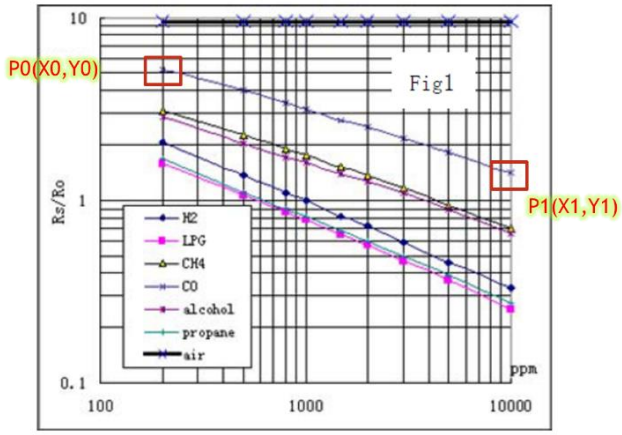


Figura 10. Curvas logarítmicas de los gases.

Para este caso particular se busca conocer la concentración de CO, por lo que se seleccionaron 2 puntos: P0(X0,Y0) y P1(X1,Y1), los cuales corresponden a los siguientes valores:

X0=200
Y0=5
X1=1000
Y1=1.2

Los valores de entrada Rs y R0 corresponden a la resistencia RL del módulo en Kilo ohms y a la resistencia R0 del sensor en kilo ohms. Para este caso particular se toman los valores [5]:

Rs=5;
R0= 10;

Los cálculos realizados para conocer la concentración se realizaran en el Formula Node, este bloque permite realizar operaciones matemáticas en lenguaje C y es muy útil en este tipo de casos, ya que ocupan menos bloques que si se ocupara un bloque para cada operación. Las operaciones realizadas son:

Rs=Rs/1000*(LecturaMáxima-lecturaAnalógica)/lecturaAnalógica;
A=(Y1-Y0)/(X1-X0);
B=Y0-(A*X0);
Concentración=pow(10,A*log(Rs/R0)+B);

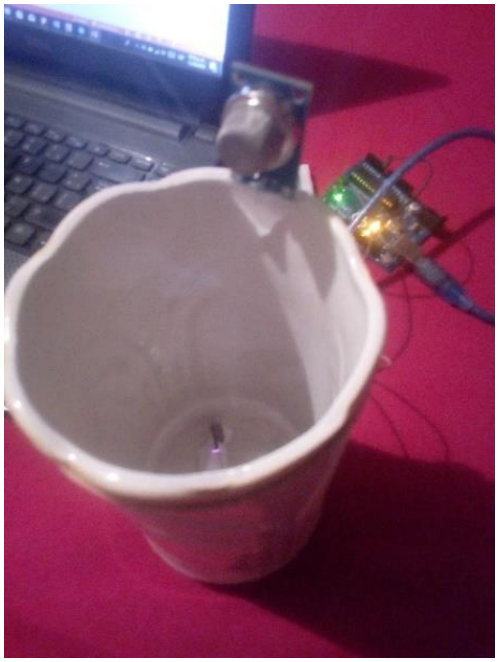


Figura 11. Realización de la práctica.

hola: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
INSTRUMENTACIÓN
UGARTE HERNÁNDEZ ENRIQUE
0       18/03/2018      10:44 p. m.     Concentración  CO      0.000000        Alto/Bajo       TRUE
1       18/03/2018      10:44 p. m.     Concentración  CO      897050.541479   Alto/Bajo       FALSE
2       18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
3       18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
4       18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
5       18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
6       18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
7       18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
8       18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
9       18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
10      18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
11      18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
12      18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
13      18/03/2018      10:44 p. m.     Concentración  CO      0.000000        Alto/Bajo       TRUE
14      18/03/2018      10:44 p. m.     Concentración  CO      916250.231495   Alto/Bajo       FALSE
15      18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
16      18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
17      18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
18      18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
19      18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
20      18/03/2018      10:44 p. m.     Concentración  CO      916284.516689   Alto/Bajo       FALSE
```

Figura 13. Datos dentro del documento de texto.

V. PROGRAMA DE LABVIEW USANDO LINX

A. Configuración de la conexión

Para realizar un programa con LINX, primero ir a *Tools* → *MakerHub* → *LINX* → *LINX Frimware Wizard*. Una vez dentro aparece una ventana donde se tiene que ingresar los datos de la tarjeta de adquisición de datos a utilizar.

Figura 12. Ajustes para la comunicación con el Arduino.

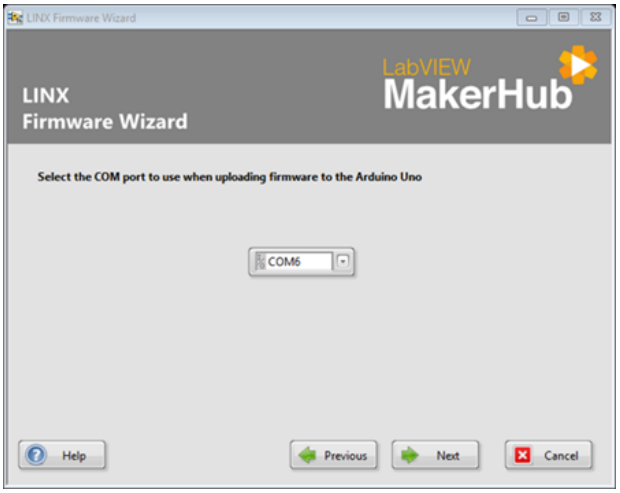


Figura 13. Selección del puerto serie.

Una manera sencilla de conocer el puerto en el que se encuentra conectador el Arduino es revisando el IDE.

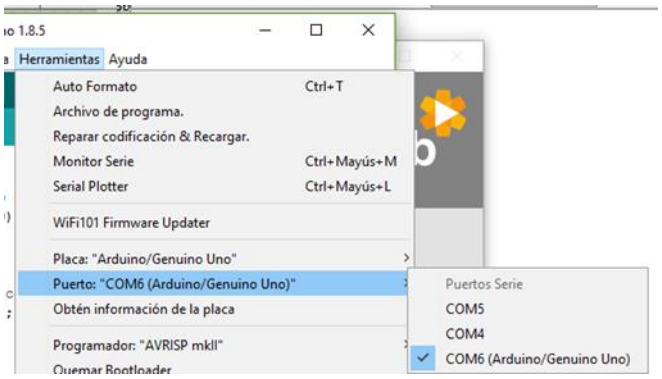


Figura 14. Puerto donde se encuentra conectado el Arduino.

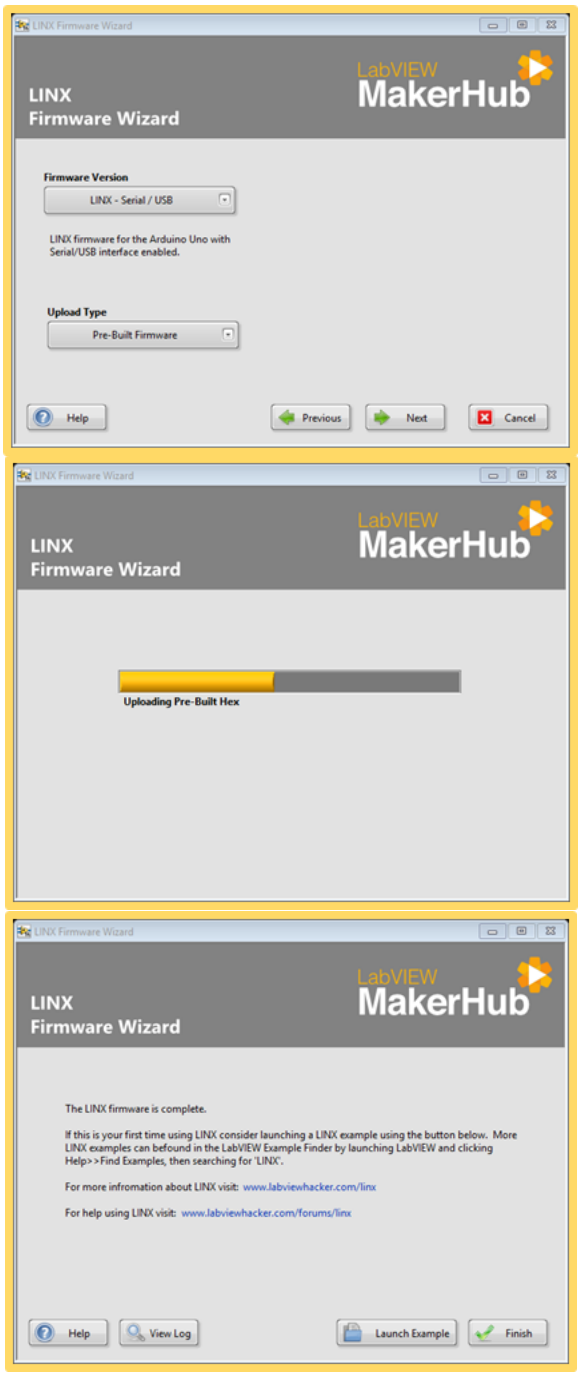


Figura 15. Programa cargando.

B. Comunicación serie

Para abrir y cerrar la comunicación se necesitan los bloques *Open.vi y Close.vi* ubicados *en Functions → MakerHub → LINX*. Para leer la información usar los bloques *Analog Read* y *Digital Read*, ubicados en *Functions → MakerHub → LINX → Peripherals* y realizar las conexiones dentro de un ciclo *While*. Nótese que la lectura analógica se encuentra en volts.

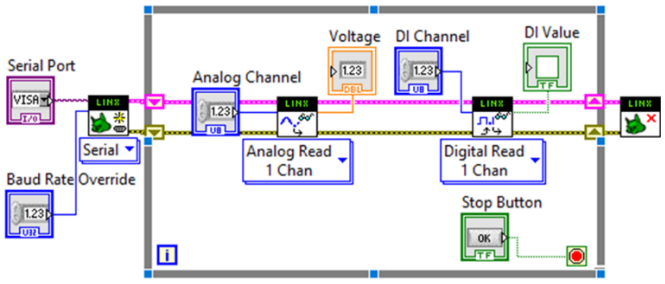


Figura 16. Comunicación serie.

C. Adquisición de datos

La lectura analógica entra dentro del bloque *Formula Node*, el cual nos sirve para realizar los cálculos para obtener la concentración en función de las constantes de cada gas y el valor digital se muestra con un *Round LED*. El bloque *Match Pattern* se encuentra *en Functions → Programming → String* y el bloque *Formula Node* se encuentra en *Functions →Mathematics →Scripts & Formulas*.

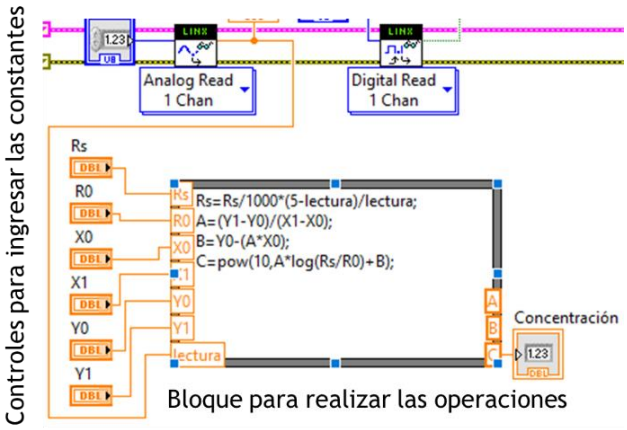


Figura 17. Operaciones dentro de Formula Node.

Texto dentro de *Formula Node*:

```
Rs=Rs/1000*(5-lectura)/lectura;
A=(Y1-Y0)/(X1-X0);
B=Y0-(A*X0);
C=pow(10,A*log(Rs/R0)+B);
```

D. Almacenamiento de datos

Se repiten los mismos bloques mostrados en la figura 5 para guardar los datos en un documento de texto.

E. Diseño final

A continuación se muestra el Diagrama de bloques completo.

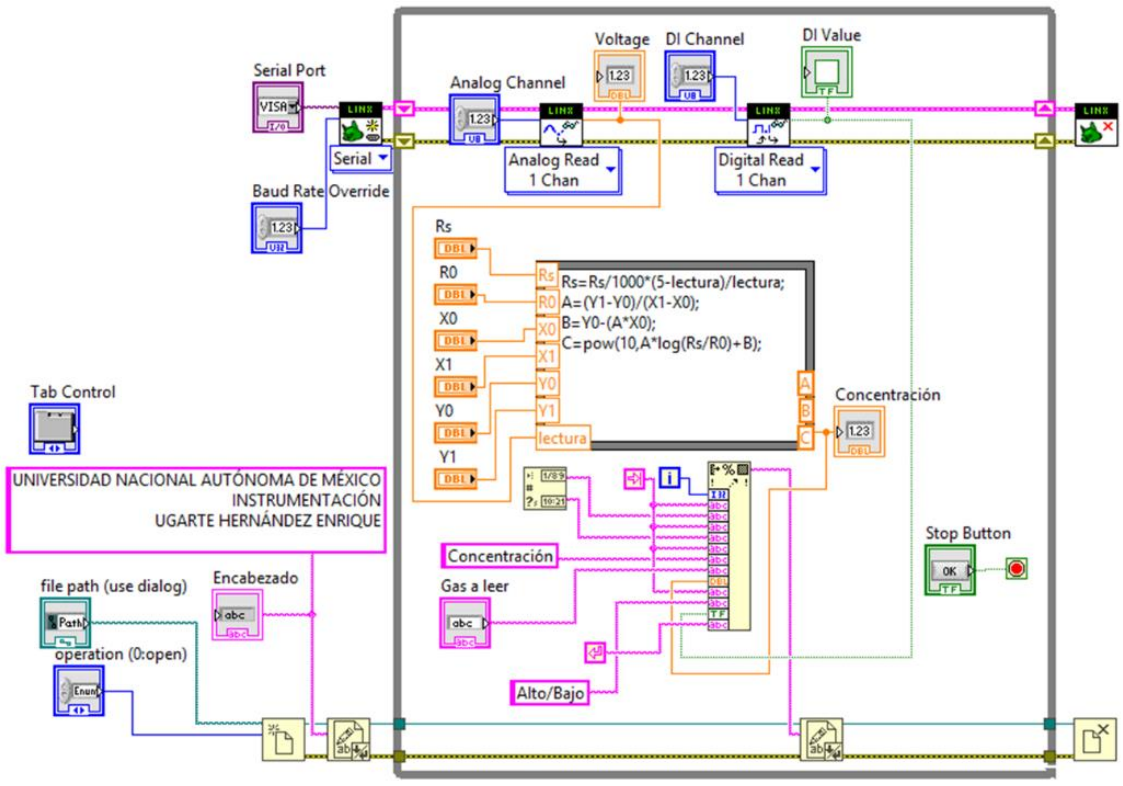


Figura 18. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña, la visualización de los datos en la segunda y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

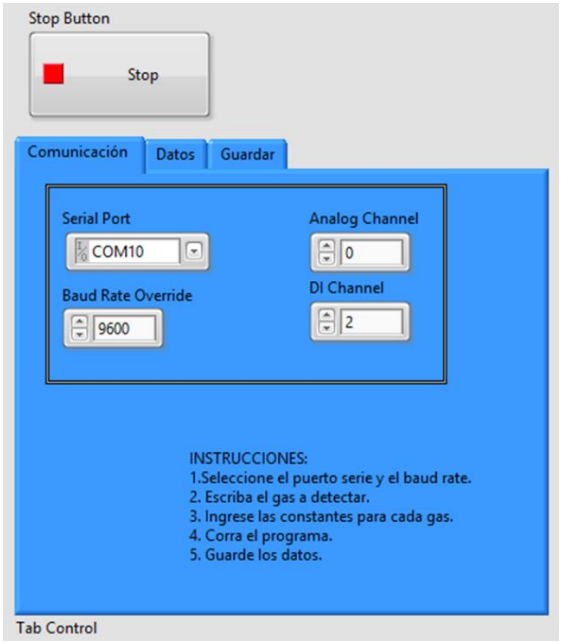


Figura 19. Primera pestaña de la interfaz.

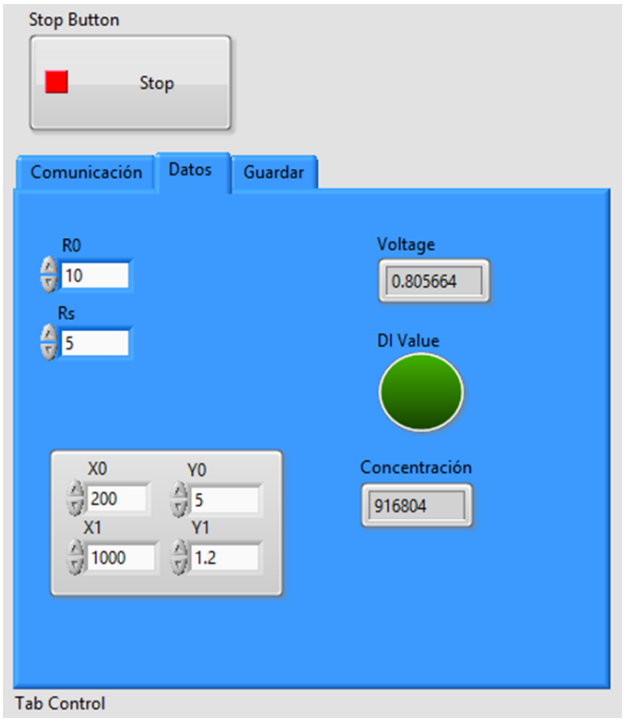


Figura 20. Segunda pestaña de la interfaz.

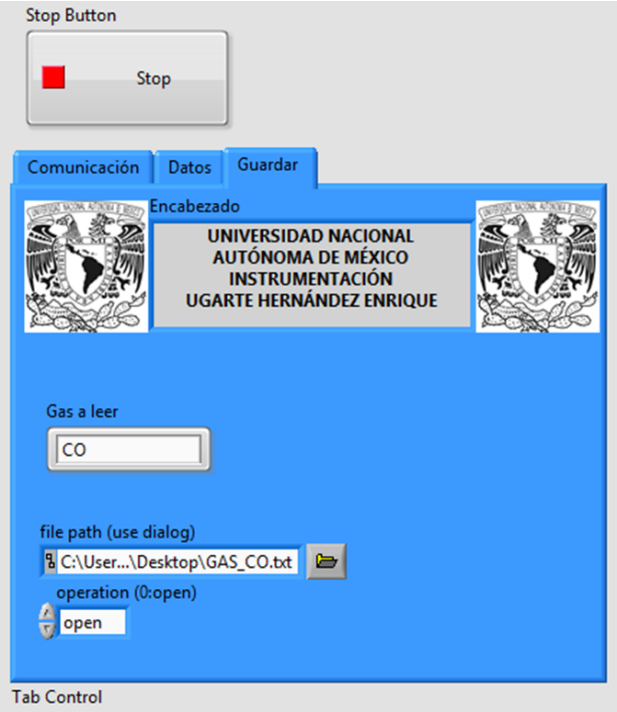


Figura 21. Tercera pestaña de la interfaz.

VI. PROGRAMA USANDO EL TOOLKIT DE ARDUINO

A. Configuración de la conexión

Cargar el código ubicado en ***C:\Program Files\National Instruments\LabVIEW 2016\vi.lib\LabVIEW Interface for Arduino\Firmware\LIFA_Base*** dentro de la tarjeta Arduino.

B. Comunicación serie

Para abrir y cerrar la comunicación con el Arduino se necesitan los bloques *Init* y *Close*, ubicados en *Funtions* → *Arduino*. Para leer la información utilizar el bloque *Analog Read Pin* y *Digital Read Pin*, ubicados en *Funtions* → *Arduino* → *Low Level*. Usar el *bloque Set Digital Pin Mode* para establecer la entrada digital, este se encuentra en *Funtions* → *Arduino* → *Low Level*. Realizar las conexiones dentro de un ciclo *While*. Hacer clic derecho sobre el bloque *Analog Read Pin* y *Digital Read Pin* para crear indicadores. Nótese que la lectura analógica se encuentra en volts.

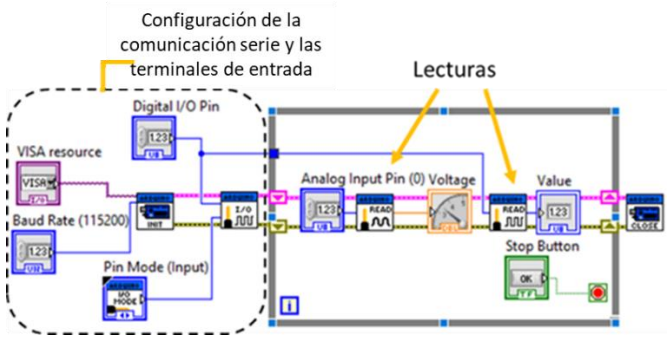


Figura 22. Diagrama de bloques para la comunicación serie y las lecturas analógica y digital.

C. Adquisición de datos

La lectura analógica entra dentro del bloque *Formula Node*, el cual sirve para realizar los cálculos para obtener la concentración en función de las constantes de cada gas y el valor digital se muestra con un *Round LED*. El bloque *Match Pattern* se encuentra *en Functions* → *Programming* → *String* y el bloque *Formula Node* se encuentra en *Functions* →*Mathematics* →*Scripts & Formulas*.

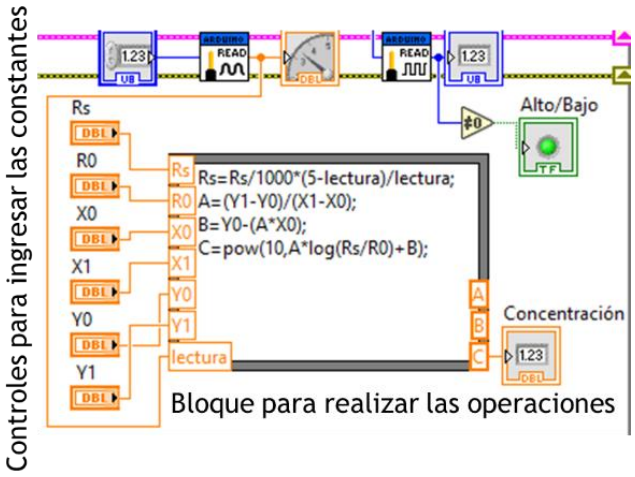


Figura 23. *Formula Node* y binarización de la entrada digital.

Texto dentro de *Formula Node*:

```
Rs=Rs/1000*(5-lectura)/lectura;
A=(Y1-Y0)/(X1-X0);
B=Y0-(A*X0);
C=pow(10,A*log(Rs/R0)+B);
```

D. Diseño final

A continuación se muestra el diagrama de bloques completo.

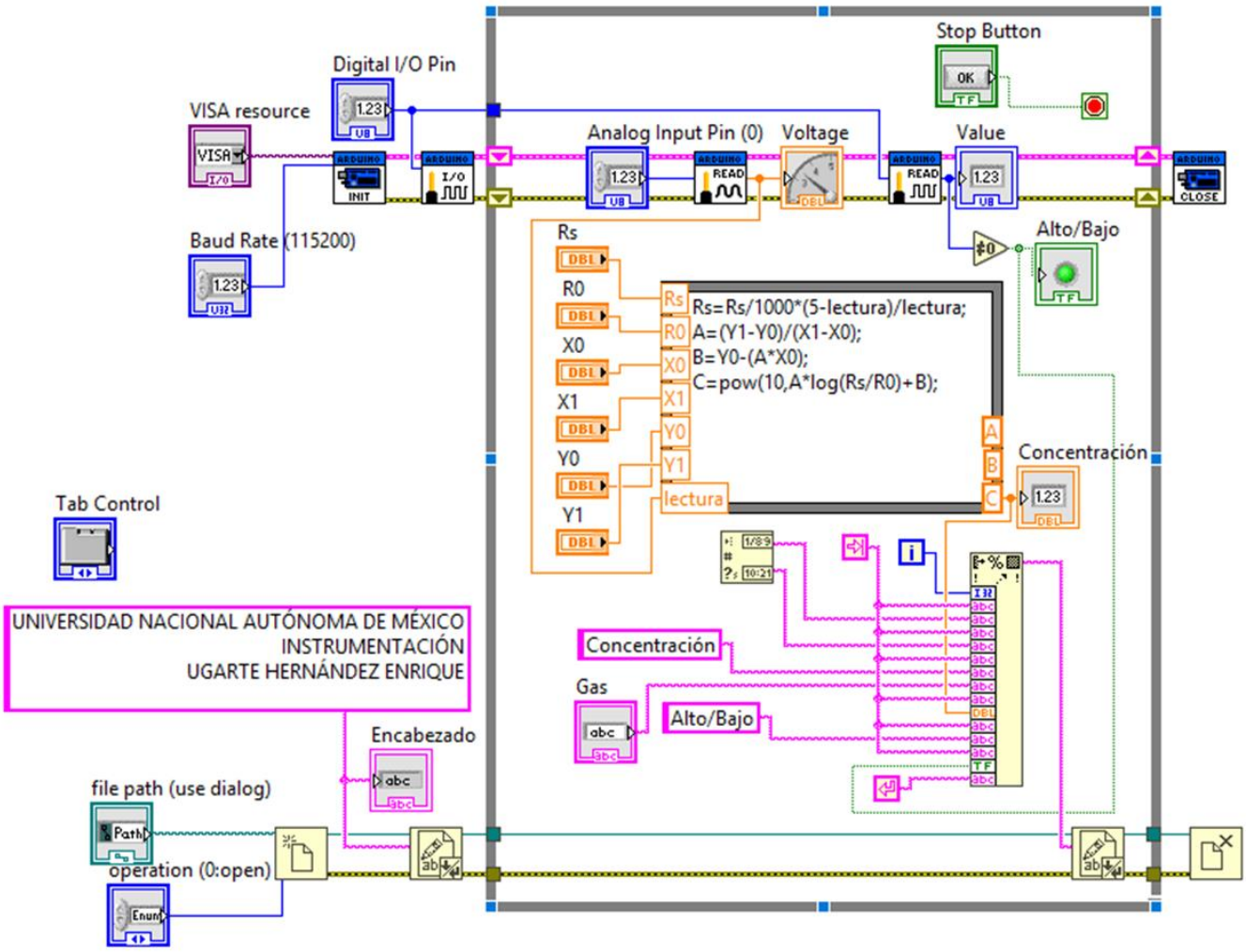


Figura 24. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación se encuentran en la primera pestaña, la visualización de los datos en la segunda y los elementos para guardar los datos en un documento de texto en la tercera. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

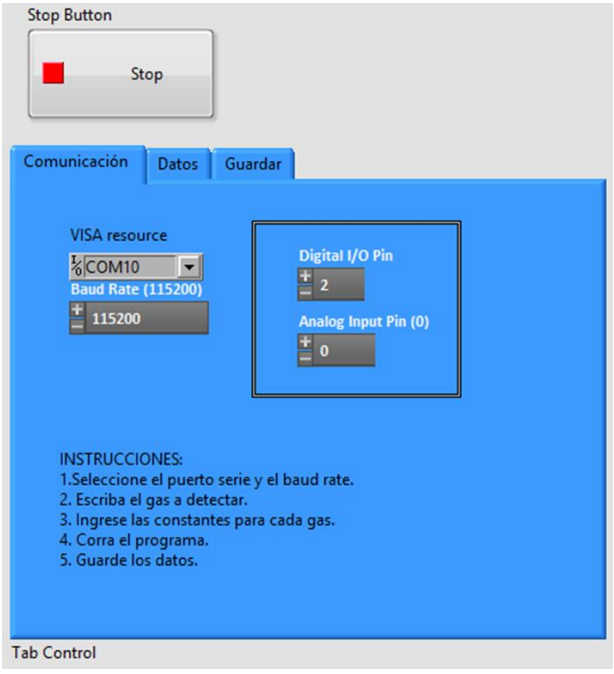


Figura 25. Primera pestaña de la interfaz.

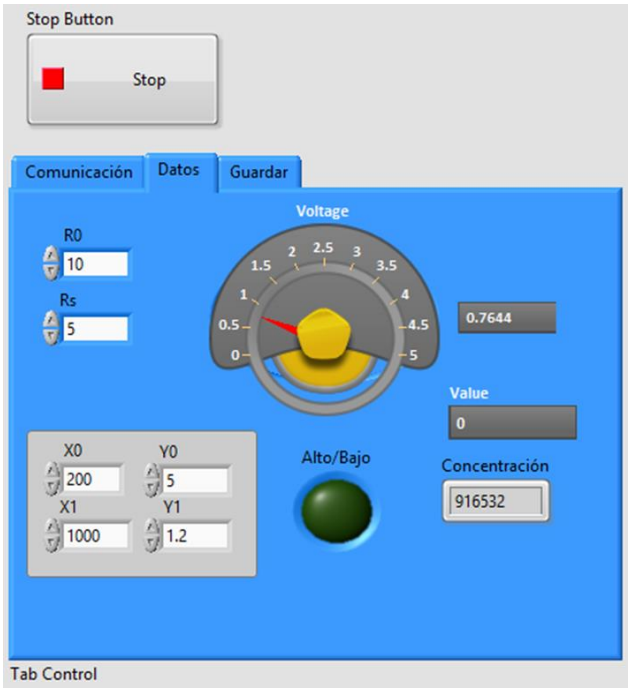


Figura 26. Segunda pestaña de la interfaz.

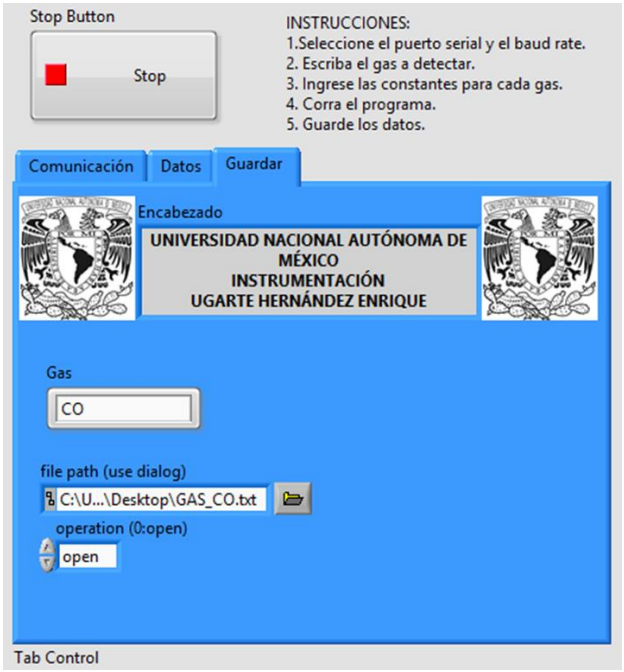


Figura 27. Tercera pestaña de la interfaz.

VII. PRECAUCIONES

- Se debe evitar quemar materiales tóxicos como el unicel.
- Es recomendable no permanecer expuesto al humo por mucho tiempo.
- El fin de esta práctica es educativo y no desea fomentar el quemar hojas de papel.

VIII. FUENTES DE CONSULTA

[1] Baeza, Alejandro. Sensores y Biosensores Electroquímicos. Facultad de Química, UNAM. Departamento de Química Analítica, 2-6.

[2] Tutoriales. Tutorial sensores de gas MQ2, MQ3, MQ7 y MQ135. Trujillo, Perú: *Naylamp Mechatronics*. Recuperado de http://www.naylampmechatronics.com/blog/42_Tutorial-sensores-de-gas-MQ2-MQ3-MQ7-y-MQ13.html

[3] Torres, Héctor (2014). Sensor de Gas MQ2 con Arduino UNO. *Hetpro Tutoriales*. Recuperado de https://hetpro-store.com/TUTORIALES/sensor-de-gas-mq2/

[4] hub360. (s.f.). MQ2 Smoke,Methane,Butane Gas Sensor. Oshodi, Lagos, Nigeria. *http://hub360.com.ng* . Recuperado de http://hub360.com.ng/product/mq2-smokemethanebutane-gas-sensor/

[5] Amazon.com, Inc. or its affiliates. (1996-2018). Wavesahre MQ-2 Gas Sensor Module LP,Propane,Hydrogen Detection Sensor Gas Detector Sensor Module for Arduino Raspberry pi. *https://www.amazon.co.uk* . Recuperado de https://www.amazon.co.uk/Wavesahre-MQ-2-Gas-Sensor-Detection/dp/B00NJOIB50

Práctica 14. Tubo de Pitot

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- Esta práctica consiste en el estudio y comprensión del funcionamiento de un tubo de Pitot para medir la presión dentro de un túnel de viento y posteriormente obtener la velocidad del viento. Los datos obtenidos son visualizados en la plataforma LabVIEW® a través del microcontrolador Arduino UNO mediante una comunicación serie.**

Palabras clave: Tubo de Pitot, piezoeléctrico, MPXV7002.

I. INTRODUCCIÓN

A. Funcionamiento del tubo de Pitot

Los sensores de presión piezorresistivos se rigen bajo un principio de transducción que consiste en generar un voltaje a través de una fuerza aplicada en un área y como resultado se obtiene una señal eléctrica determinada. La cantidad de presión en un fluido se define como la fuerza por cada unidad de área y su unidad de medida es el pascal (Pa). Existen dos tipos de sensores, los de presión absoluta (Pabs) y los de presión diferencial (Pdif); los primeros consisten en una cámara sellada y con una presión de referencia, todo esto sucede con la finalidad de poseer un área de control y evitar compensaciones por la variación de presión que exista en la cámara. [1].

Los sensores de presión diferenciales funcionan debido a una diferencia de presiones denominadas (P1) y (P2); si se considera la presión barométrica entonces se dice que es un sensor de presión relativo. El tubo de Pitot es un instrumento que sirve para medir la presión total o de estancamiento; no mide directamente la velocidad, pero permite tener una estimación con bastante precisión.

La presión total se compone de la suma de la presión estática (Pe) y la presión dinámica (Pd); considerando a la ecuación de Bernoulli, deducimos la siguiente ecuación [2]:

$$\boldsymbol{P_e + \rho\left(\frac{v^2}{2}\right) = P_t} \ldots(1)$$

Donde:

$\boldsymbol{P_e}$ = Presión estática [Pa]

$\boldsymbol{\rho}$ = Densidad del aire [kg/m^3]

$\boldsymbol{P_t}$ = Presión total [Pa]

$\boldsymbol{v}$ = Velocidad del viento [m/s]

Despejando la velocidad (v) de la ecuación 1:

$$\boldsymbol{v = \sqrt{\frac{2(P_t - P_e)}{p}}} \ldots (2)$$

El tubo de Pitot cuenta con una entrada principal por donde entra el flujo de aire y a los costados unos agujeros por donde entra la presión atmosférica; cuenta con dos salidas, una para la presión atmosférica y otra para la presión del fluido, las cuales entran al transductor piezoeléctrico.

Figura 1. Tubo de Pitot para Ardupilot.

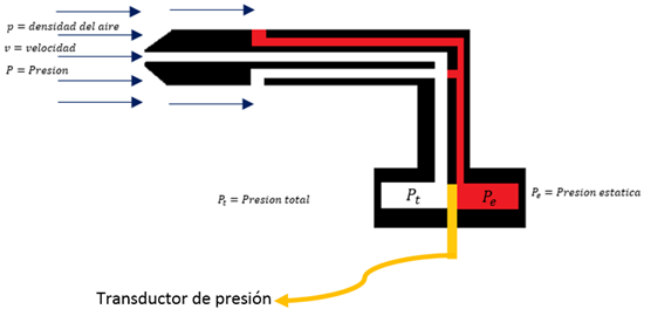


Figura 2. Funcionamiento del tubo de Pitot [3].

Algunas de las áreas de aplicación que tiene el tubo de Pitot son:

- Industria aeronáutica.
- Estaciones meteorológicas.
- Automovilismo.

B. Sensor de presión MPXV7002

Es un sensor de presión de silicio monolítico diseñado para ser utilizado mediante un microcontrolador o un microprocesador con entradas de tipo Analógico / Digital; este realiza la función de un transductor del tipo piezorresistivo. Es fabricado bajo la tecnología MEMS; según la hoja de datos del fabricante utiliza una película delgada y procesamiento bipolar para proporcionar una señal de salida analógica precisa y de alto nivel que es proporcional a la presión aplicada [4].

Este transductor fue diseñado para medir la presión positiva y negativa con una compensación específica de 2.5 [V] en lugar de la convencional 0 [V]. Permite medir presión hasta 7 kPa a través de cada puerto para detección de presión, pero también para detección de vacío.

Figura 3. Sensor de presión MPXV7002.

II. OBJETIVOS

- ✓ Conocer el funcionamiento del tubo de Pitot.
- ✓ Medir la presión dentro de un túnel de viento y posteriormente obtener la magnitud de la velocidad del viento.
- ✓ Diseñará una interfaz gráfica utilizando el software LabVIEW que permita visualizar los datos de forma clara y guardarlos en un documento de texto.

III. MATERIAL

- ➢ Tubo de Pitot para Ardupilot.
- ➢ Sensor de presión MPXV7002.
- ➢ Tarjeta Arduino UNO.
- ➢ Mangueras con un diámetro interno de 2 mm.
- ➢ 3 Jumpers H-M
- ➢ Equipo de cómputo con el software LabVIEW e IDE de Arduino.
- ➢ Material para fabricar un soporte para el tubo de Pitot.
- ➢ Túnel de viento.
- ➢ Anemómetro de tubo de Pitot con su respectivo tubo de Pitot.

IV. DESARROLLO

A. Fabricación del soporte

Para la realización de la práctica es necesario un túnel de viento, razón por la cual se solicitó permiso al Laboratorio de Termofluidos de la Facultad de Ingeniería. Uno de los requisitos para la utilización del túnel de viento es fabricar un soporte para este dispositivo. Antes de fabricarlo es necesario tomar las medidas del tubo de Pitot del laboratorio.

Figura 4. Ejemplo de soporte.

B. Conexiones

Realizamos las siguientes conexiones:

- ➢ +5V → 5V.
- ➢ GND → GND.
- ➢ ANALOG → A0.

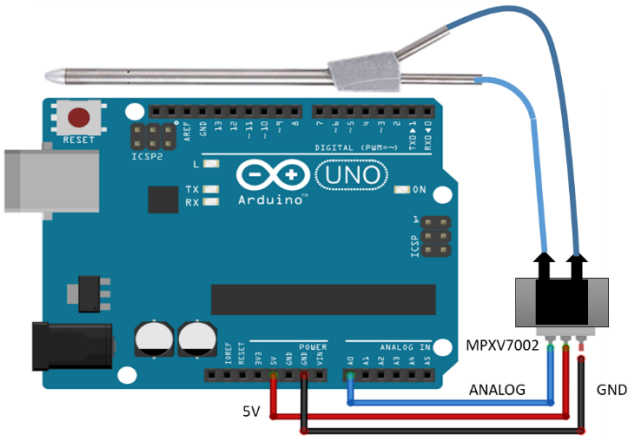


Figura 5. Conexión de los componentes.

El tubo de Pitot normalmente se consigue conectado con el MPXV7002.

C. Código de Arduino

Una vez realizadas las conexiones se genera un código sencillo de Arduino para leer la señal analógica.

```
int sensor;
void setup() {
Serial.begin(9600);
}
void loop() {
  sensor=analogRead(A0);
  Serial.println(sensor);
  delay(400);
}
```

Se utiliza un anemómetro con tubo de Pitot para calibrar el tubo de Pitot, este lo proporciona el laboratorio; para este caso particular se obtuvieron los siguientes datos:

Frecuencia del motor [Hz]	Lectura del Arduino	Lectura del anemómetro [Pa]
0	540	1
40	561	136

De esta manera queda un sistema de 2 ecuaciones con 2 incógnitas:

B(540) + A =1
B(561) + A =136

Al resolverlo da como resultado:

A=-3470.428571

B=6.42857
Por lo que el código final es:

```
float sensor;
float B=6.42857;
float A=-3470.428571
void setup() {
Serial.begin(9600);
}
void loop() {
  sensor=analogRead(A0);
  Serial.println(B*sensor+A);
  delay(400);
}
```

Este código manda a través del puerto serie la presión en pascales.

Figura 6. Anemómetro usado para calibrar el tubo de Pitot.

D. Interfaz en LabVIEW

1) Adquisición de datos

El programa de LabVIEW obtiene la presión del puerto serie (ver "Introducción al entorno de desarrollo de LabVIEW"). Para obtener la velocidad del viento aplicar la ecuación 2. Finamente mostrar los datos con indicadores numéricos y gráficas. El programa cuenta con los bloques para guardar los datos dentro de un documento de texto (ver "Introducción al entorno de desarrollo de LabVIEW").

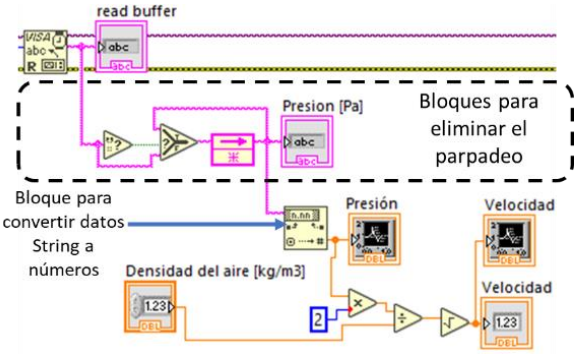


Figura 7. Eliminación del parpadeo, obtención de la velocidad del viento y gráficas.

2) Almacenamiento de datos

Los bloques para guardar los componentes son: *Open/Create/Replace File*, *Close File* y *Write Text File*, estos se encuentran en *Programming → File I/O*, se utiliza el bloque: *Format Into String* para acomodar la información de manera ordenada, este se encuentra en *Programming → String*.

3) Diseño final

A continuación se muestra el diagrama de bloques completo.

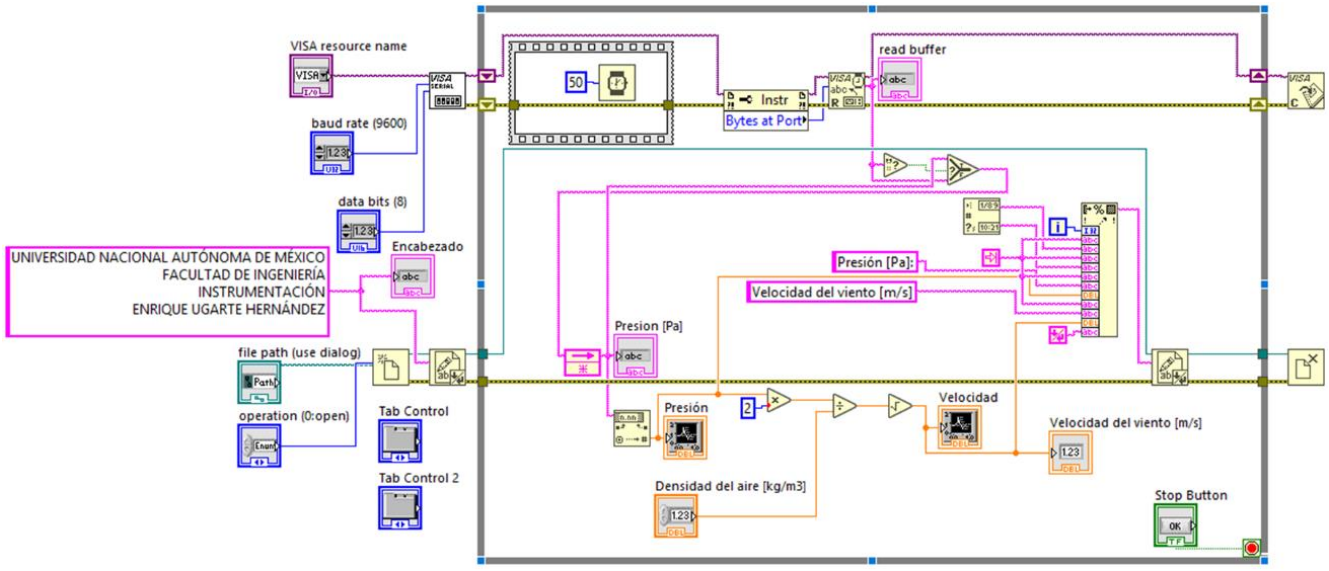


Figura 8. Diagrama de bloques completo

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña, la visualización de los datos

y el control numérico para ingresar la densidad del aire en la segunda, las gráficas en la tercera y los elementos para guardar los datos en un documento de texto en la cuarta. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña). Se usó 1.225 kg/m^3 como densidad del aire.

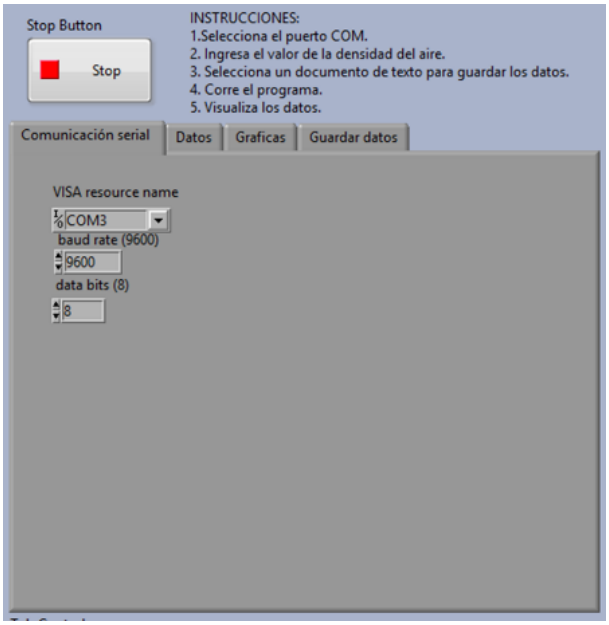


Figura 9. Primera página del contenedor con pestañas.

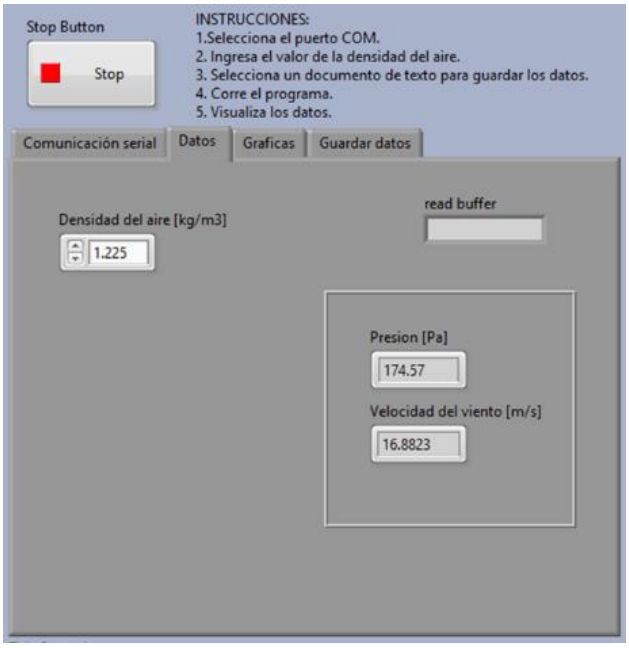


Figura 10. Segunda página del contenedor con pestañas.

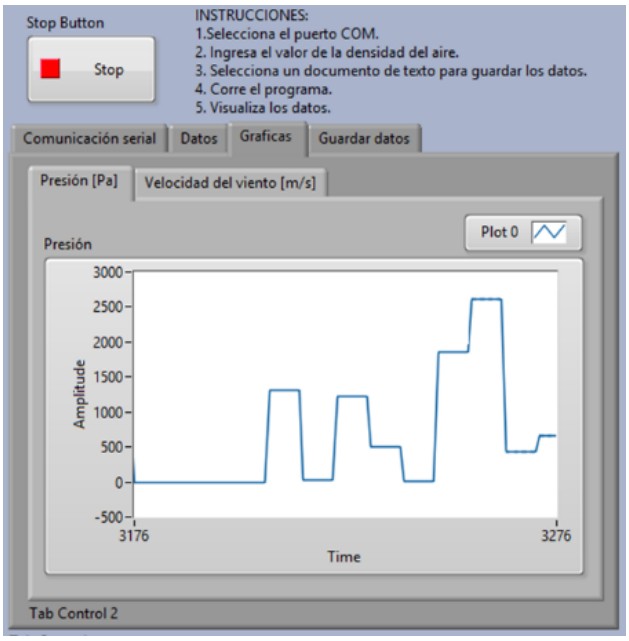


Figura 11. Gráfica de presión en la tercera pestaña.

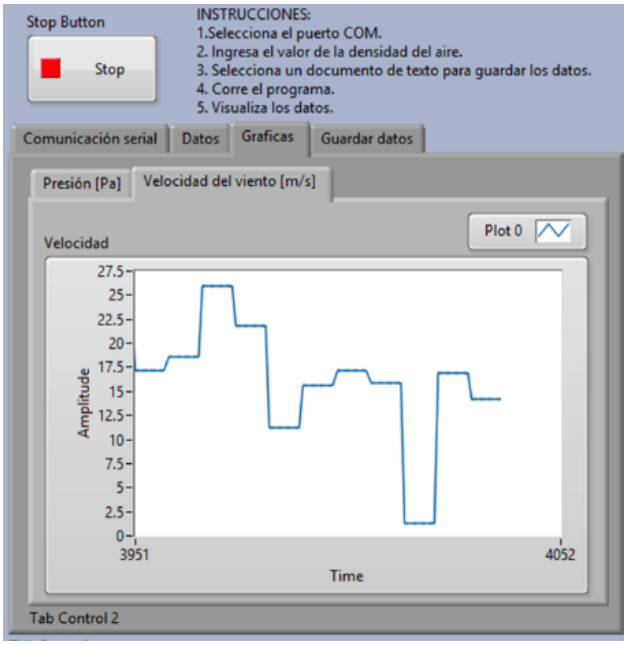


Figura 12. Gráfica de rapidez en la tercera pestaña.

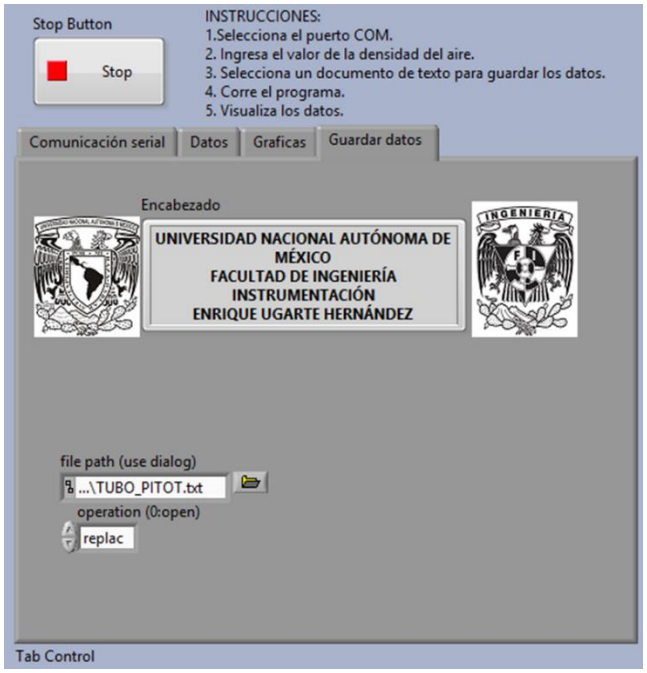


Figura 15. Cuarta página del contenedor con pestañas.

prueba: Bloc de notas

Archivo Edición Formato Ver Ayuda

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA
INSTRUMENTACIÓN
ENRIQUE UGARTE HERNÁNDEZ

0	14/05/2018	04:13 p. m.	Presión [Pa]: 0.000000	Velocidad del viento [m/s] 0.000000
1	14/05/2018	04:13 p. m.	Presión [Pa]: 1.000000	Velocidad del viento [m/s] 1.277753
2	14/05/2018	04:13 p. m.	Presión [Pa]: 7.430000	Velocidad del viento [m/s] 3.482903
3	14/05/2018	04:13 p. m.	Presión [Pa]: 7.430000	Velocidad del viento [m/s] 3.482903
4	14/05/2018	04:13 p. m.	Presión [Pa]: 1.000000	Velocidad del viento [m/s] 1.277753
5	14/05/2018	04:13 p. m.	Presión [Pa]: 7.430000	Velocidad del viento [m/s] 3.482903
6	14/05/2018	04:13 p. m.	Presión [Pa]: 1.000000	Velocidad del viento [m/s] 1.277753
7	14/05/2018	04:13 p. m.	Presión [Pa]: 1.000000	Velocidad del viento [m/s] 1.277753
8	14/05/2018	04:13 p. m.	Presión [Pa]: 1.000000	Velocidad del viento [m/s] 1.277753
9	14/05/2018	04:13 p. m.	Presión [Pa]: 7.430000	Velocidad del viento [m/s] 3.482903
10	14/05/2018	04:13 p. m.	Presión [Pa]: 1.000000	Velocidad del viento [m/s] 1.277753
11	14/05/2018	04:13 p. m.	Presión [Pa]: 206.710000	Velocidad del viento [m/s] 18.370784
12	14/05/2018	04:13 p. m.	Presión [Pa]: 20.280000	Velocidad del viento [m/s] 5.754147

Figura 16. Datos en el documento de texto.

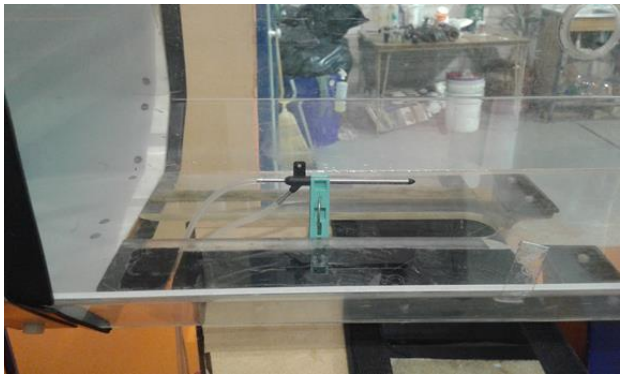


Figura 17. Tubo de Pitot sujeto al soporte.

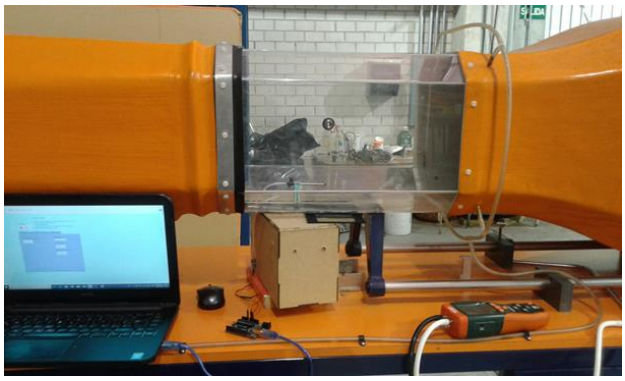


Figura 18. Pruebas con el túnel de viento.

V. FUENTES DE CONSULTA

[1] Corona Rodríguez, L. G. (2014). Sensores y Actuadores: Aplicaciones con Arduino. Grupo Editorial Patria.

[2]. Castro F, Roberto A. (1985). Apuntes de Mecánica de Fluidos, Facultad de Ingeniería UNAM, Pág. 168 - 178.

[3] Turchetto, Matteo. (2011). Espirometría intraoperatoria: principios del funcionamiento del tubo de Pitot. *Anestesia en línea*. Recuperado de http://www.anestesiaenlinea.com/2011/04/espirometria-intraoperatoria-principios-del-funcionamiento-del-tubo-de-pitot/ .

[4] [Datasheet]. Recuperado de

https://www.nxp.com/docs/en/data-sheet/MPXV7002.pdf.

Práctica 15. Unidad de medición inercial (IMU)

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- Esta práctica consiste en el estudio, comprensión y funcionamiento de la unidad de medición inercial IMU GY-80. Los datos obtenidos serán visualizados en la plataforma LabVIEW® a través del microcontrolador Arduino UNO mediante comunicación serie.**

I. INTRODUCCIÓN

A. IMU GY-80

El término "unidad de medición inercial" (del inglés *inertial measurement unit*) se refiere a un dispositivo electrónico que tiene como finalidad medir la posición y orientación de un objeto en el espacio.

Las unidades de medición inercial constan de diferentes sensores, como lo son: acelerómetros, giróscopos y magnetómetros para poder realizar estimados de posición, orientación, rapidez lineal y angular combinando los datos de medición de una manera eficaz y certera. Si nuestra unidad de medición inercial dispone de más sensores, las estimaciones serán más precisas y confiables.

Las unidades de medición inercial generalmente recolectan los datos de los sensores, por ejemplo: acelerómetro, magnetómetro y giróscopo. Posterior a esto se continúa con el procesamiento de la señal; algunos sensores lo hacen simultáneamente y arrojan el dato ya calibrado vía I^2C/SPI a la memoria de almacenamiento de la IMU, dependiendo de los modelos del tipo de sensor de nuestra IMU. Para algunos modelos se necesita un convertidor analógico digital [1].

La IMU GY-80 consta de 4 sensores:

- ADXL345 (acelerómetro).
- BMP085 (barómetro).
- HMC5883L (magnetómetro).
- L3G4200D (giróscopo).

Este tipo de dispositivo electrónico es ampliamente utilizado en cualquier sistema que necesite navegación y orientación, por ejemplo: aeronaves, satélites, barcos, vehículos no tripulados, entre otros.

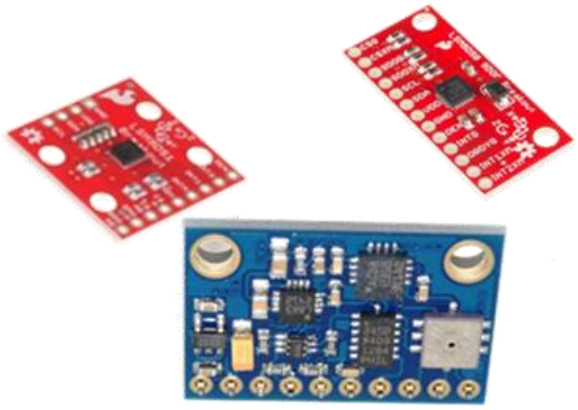


Figura 1. Varios modelos de IMU. La GY80 se encentra en el centro de la figura.

B. Acelerómetro ADXL345

Los acelerómetros pueden detectar las fuerzas de aceleración, ya sea estáticas o dinámicas. Las fuerzas estáticas incluyen la inclinación y vibración respecto a la fuerza de gravedad, mientras que las fuerzas dinámicas se usan para determinar la aceleración traslacional. Los acelerómetros pueden medir la aceleración en uno, dos o tres ejes.

Los acelerómetros están basados en el principio de transducción piezorresistivo, piezoeléctrico o capacitivo; sin importar que principio que utilicemos la respuesta del acelerómetro será una salida lineal, es decir, se observará una respuesta proporcional a la entrada que se reciba.

Los acelerómetros se rigen bajo la segunda ley de Newton y la ley de Hook; utilizando este principio podemos decir que los acelerómetros contienen una masa móvil que al percibir una aceleración del exterior genera un desplazamiento proporcional al cociente entre la fuerza aplicada y su rigidez asociada [2,3].

Los acelerómetros piezorresistivos y piezoeléctricos son muy utilizados para la industria, principalmente para detectar vibraciones de la maquinaria. También existen los acelerómetros capacitivos comercialmente fabricados con la tecnología MEMS (Microelectromechanical Systems).

Los acelerómetros capacitivos pueden comunicarse vía analógica o digital. Los acelerómetros digitales generalmente utilizan los protocolos I^2C o SPI. El ADXL345 es un acelerómetro de tres ejes con salida de datos digitales de 16 bits y se puede comunicar vía protocolos de comunicación ya sea I^2C o SPI [4,5].

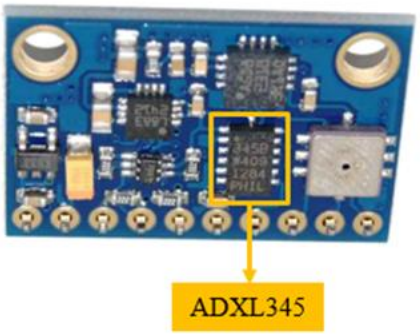


Figura 2. Acelerómetro ADXL345 integrado en una IMU.

C. Giroscopio L3G4200D

El giróscopo o giroscopio L3G4200D es un sensor de movimiento angular de tres ejes. Este sensor fue desarrollado por STMicroelectronics® y utiliza el protocolo de comunicación I^2C/SPI para establecer una conexión entre el microcontrolador y el sensor.

El sensor de rapidez angular L3G4200D posee una escala completa de ± 250 / ± 500 / ± 2000 dps (del inglés *degrees per second*) y es capaz de medir las tasas con un ancho de banda seleccionable por el usuario a través de la programación de sus registros [3,4].

Table 18. Register address map

Name	Type	Register address		Default
		Hex	Binary	
Reserved	-	00-0E	-	-
WHO_AM_I	r	0F	000 1111	11010011
Reserved	-	10-1F	-	-
CTRL_REG1	rw	20	010 0000	00000111
CTRL_REG2	rw	21	010 0001	00000000
CTRL_REG3	rw	22	010 0010	00000000
CTRL_REG4	rw	23	010 0011	00000000
CTRL_REG5	rw	24	010 0100	00000000
REFERENCE	rw	25	010 0101	00000000
OUT_TEMP	r	26	010 0110	output
STATUS_REG	r	27	010 0111	output
OUT_X_L	r	28	010 1000	output
OUT_X_H	r	29	010 1001	output
OUT_Y_L	r	2A	010 1010	output
OUT_Y_H	r	2B	010 1011	output
OUT_Z_L	r	2C	010 1100	output
OUT_Z_H	r	2D	010 1101	output
FIFO_CTRL_REG	rw	2E	010 1110	00000000
FIFO_SRC_REG	r	2F	010 1111	output
INT1_CFG	rw	30	011 0000	00000000
INT1_SRC	r	31	011 0001	output
INT1_TSH_XH	rw	32	011 0010	00000000
INT1_TSH_XL	rw	33	011 0011	00000000
INT1_TSH_YH	rw	34	011 0100	00000000
INT1_TSH_YL	rw	35	011 0101	00000000
INT1_TSH_ZH	rw	36	011 0110	00000000
INT1_TSH_ZL	rw	37	011 0111	00000000
INT1_DURATION	rw	38	011 1000	00000000

Figura 3. Tabla de registros [3].

Con el mapa de registros podemos activar uno de los ejes del giróscopo o los tres al mismo tiempo, entre otras funciones, el giróscopo esta calibrado desde fábrica y cada vez que se reinicia se restauran de manera automática a los valores de fabricación.

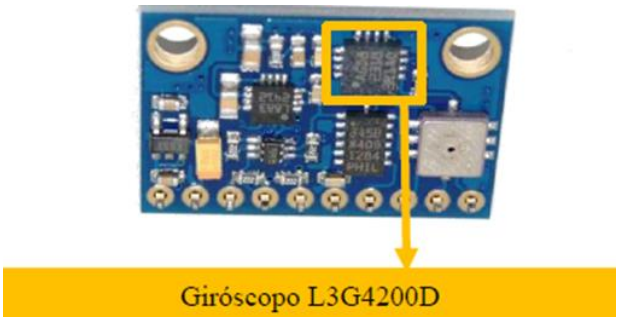


Figura 4. Giróscopo L3G4200D integrado en una IMU.

D. Barómetro BMP085

El barómetro BMP085 es un dispositivo piezoeléctrico desarrollado por Robert Bosch®; contiene un convertidor analógico a digital, una unidad de memoria EPROM y una interfaz I^2C para comunicaciones de tipo serie. Su funcionalidad se basa en la recolección de datos no compensados de presión y temperatura que con ayuda de la memoria EPROM almacenará 176 bits de datos que serán calibrados posteriormente [6].

El barómetro BMP085 se diseñó para establecer una conexión vía I^2C con un microcontrolador. Los datos recolectados de presión y temperatura en la EPROM son calibrados posteriormente [7].

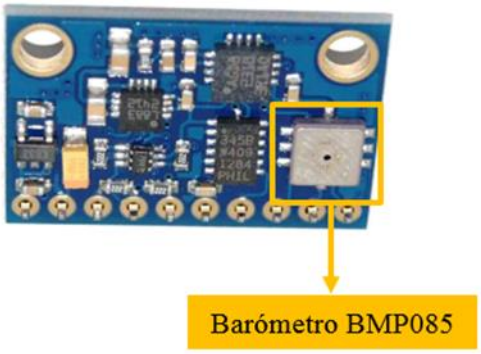


Figura 5. Barómetro BMP085 integrado en una IMU.

Medición de presión y temperatura

El microcontrolador inicia una secuencia para el inicio de la medición de presión o temperatura, dimensiona el tiempo y muestra el valor de temperatura o presión recolectada; posterior a este evento se deben utilizar los datos de calibración según la tabla de la figura 6.

	BMP085 reg adr	
Parameter	MSB	LSB
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

Figura 6. En esta imagen se muestran 11 palabras de 16 bits cada una. Las once palabras representan los coeficientes de calibración del sensor antes del cálculo de presión y temperatura [8].

La hoja de especificaciones del fabricante presenta un algoritmo para calcular la presión y temperatura, el cual consiste en la lectura de los datos en la memoria EPROM con los valores de calibración de la tabla de la figura 9, seguido de una compensación de presión y temperatura, para finalmente realizar los cálculos correctos y obtener las variables ya calibradas [8].

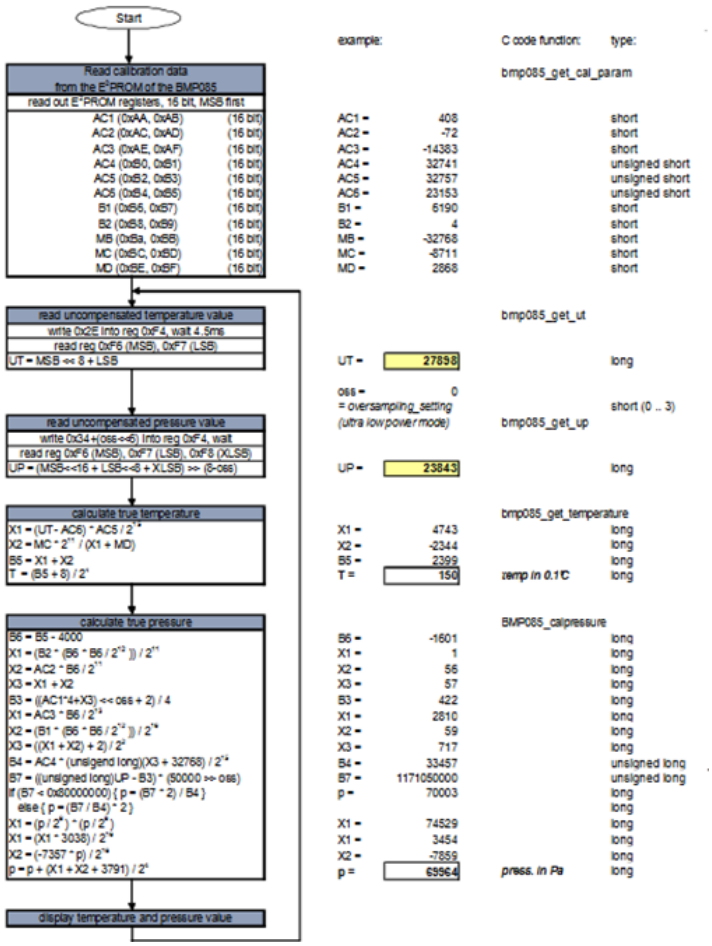


Figura 7. Algoritmo para la obtención de la medición presión y temperatura [8].

Medición de altitud

El barómetro BMP085 calcula la altitud con base en el modelo matemático utilizado por la Atmosfera Estándar Internacional (ISA) que se obtiene a partir de la siguiente ecuación [9]:

$$P_{ISA} = p_0\left(\frac{(T_0+\lambda)*z}{T_0}\right)^{-\frac{g}{R*\lambda}} \ldots(1)$$

Donde:

$p_0 = Presion\ a\ nivel\ del\ mar\ (101325\ Pa)$
$T_0 = Temperatura\ a\ nivel\ del\ mar\ (288.15\ K)$
$\lambda = Gradiente\ termico\ (-6.5x10^{-3}\ K/m)$
$g = Gravedad\ (9.81\ m/s^2)$
$R = 287\ m^2/s^2K$
$z = altitud\ (metros)$

Despejando a z (altitud) de 1:

$$z = \frac{T_0\left(\left(\frac{P_{ISA}}{p_0}\right)^{-\frac{R*\lambda}{g}} - 1\right)}{\lambda} \ldots(2)$$

Sustituyendo variables finalmente obtenemos:

$$z = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{0.1903}\right)$$

Mientras que para el cálculo de la presión a nivel del mar, solo despejamos a p_0.

E. Magnetómetro

El Magnetómetro HMC5883L es un sensor magnetorresistivo desarrollado por Honeywell®; este sensor detecta el campo magnético terrestre. Utiliza el protocolo de comunicación I^2C para establecer una conexión entre el microcontrolador y el sensor.

Este sensor funciona de la siguiente manera: el sensor detecta un campo magnético incidente en la dirección de un eje que es más sensible a una salida de voltaje [10,11].

La brújula de navegación se puede obtener utilizando solo las componentes del campo magnético Hx & Hy. Estas componentes son las direcciones planas de la superficie terrestre. Se utiliza la siguiente ecuación, la cual posteriormente se implementa en el código de programación de la interfaz Arduino IDE para obtener la brújula de navegación:

$$heading = \tan^{-1}\frac{Hx}{Hy}$$

$Donde$:

Hx
$= valor\ dobtenido\ del\ magnetometro\ del\ campo\ magnetico\ terrestre\ en\ x$ [T].
Hy
$= valor\ dobtenido\ del\ magnetometro\ del\ campo\ magnetico\ terrestre\ en\ y$ [T].

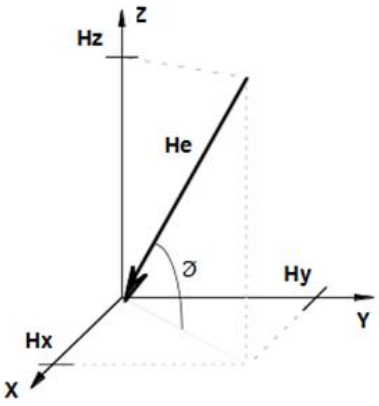


Figura 8. Campo magnético terrestre (He) en los tres ejes [10].

Figura 9. Magnetómetro HMC5883L integrado en una IMU.

II. OBJETIVOS

- Usar una IMU para el muestreo de valores del giróscopo (velocidad angular), del barómetro (presión, temperatura y altitud), acelerómetro (aceleración en los tres ejes) y del magnetómetro (campo magnético terrestre en los tres ejes y brújula magnética).
- Diseñar una interfaz gráfica que muestre los datos de la IMU de manera ordenada dentro de un contenedor con pestañas, que permita guardar los datos en un documento de texto.

III. MATERIALES

- IMU GY-80.
- Tarjeta Arduino UNO.
- 4 Cables Dupont macho-hembra.
- Equipo de cómputo con el software LabVIEW e IDE de Arduino.

IV. DESARROLLO

A. Conexión

Realizamos las siguientes conexiones:

IMU	Tarjeta Arduino
VCC_IN	5V
GND	GND
SCI	A5
SDA	A4

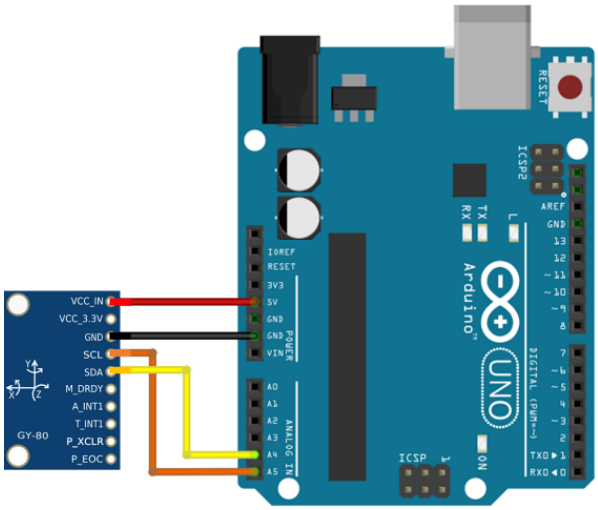


Figura 10. Diagrama de conexiones.

B. Obtención de los datos del acelerómetro

En caso que solo se desee obtener los datos del acelerómetro; la manera más fácil es descargando las siguientes bibliotecas:

- https://github.com/adafruit/Adafruit_ADXL345
- https://github.com/adafruit/Adafruit_Sensor

Estas dos bibliotecas son las únicas para leer los datos del acelerómetro de la IMU; sin embargo, existen otras bibliotecas para utilizar los demás sensores:

https://github.com/adafruit/Adafruit_HMC5883_Unified

https://github.com/adafruit/Adafruit_BMP085_Unified

Para instalar la biblioteca ir a *Programa* → *Incluir Librería* → *Añadir librería ZIP* y seleccionar la ubicación de las carpetas comprimidas.

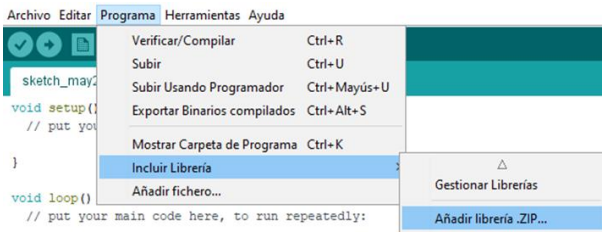


Figura 11. Inclusión de una biblioteca en el IDE de Arduino.

Para obtener la aceleración buscamos el ejemplo ubicado en *Archivo* → *Ejemplos* → *Adafrut ADXL345* → *sensortest* y correr el programa.

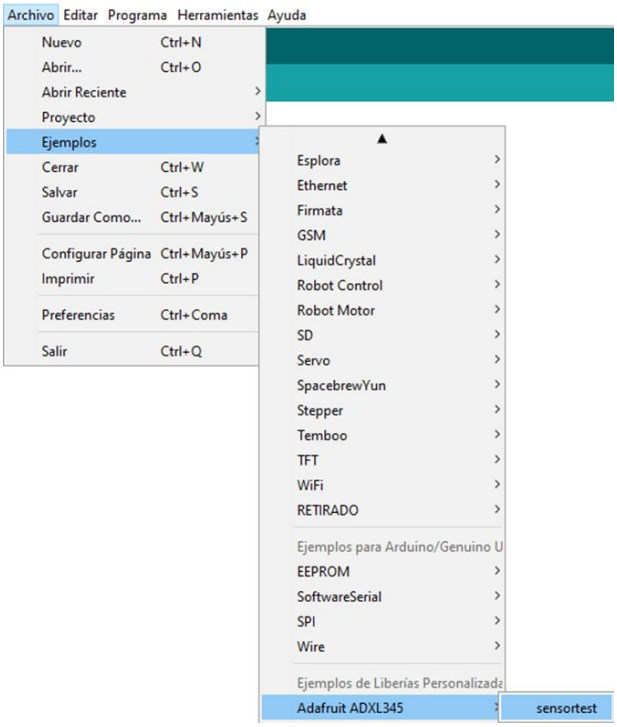


Figura 12. Pasos para abrir el programa para obtener solamente la aceleración en los tres ejes.

Al abrir el monitor serie aparecen los datos de la aceleración de manera ordenada.

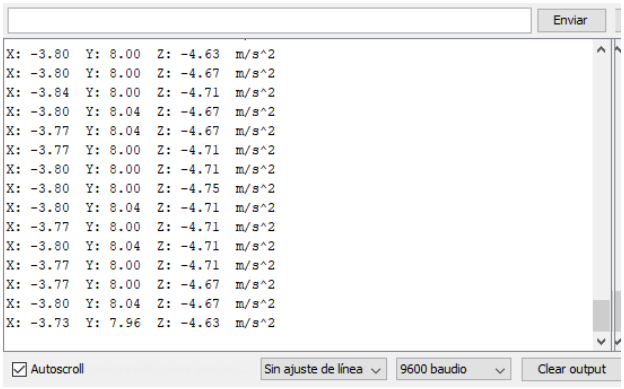


Figura 13. Datos de la aceleración a través del monitor serie.

C. Obtención de todos los datos de la IMU y generación de la interfaz gráfica

1. Código de Arduino

Ejecutar el siguiente código en el IDE de Arduino para leer todos los datos de la IMU:

```
#include <Wire.h>

#define BMP085_ADDRESS 0x77 //Direccion I2C para Barometro
#define ADXAddress 0x53     //Direccion I2C para Acelerometro
#define Compas 0x1E         //Direccion I2C para Magnetometro
//Declaracion de variables para gyro
#define XG 0
#define YG 1
#define ZG 2
//-------Registros para acelerometro------
#define Register_ID 0
#define Register_2D 0x2D
#define Register_X0 0x32
#define Register_X1 0x33
#define Register_Y0 0x34
#define Register_Y1 0x35
#define Register_Z0 0x36
#define Register_Z1 0x37

bool calibrated = false;
bool gyroInitState = false;

double gyroRaw[3];
double gyroRate[3];
double gyroCal[3];

const unsigned char OSS = 0;

// Calibracion de valores datasheet pag 12
int ac1;
int ac2;
int ac3;
unsigned int ac4;
unsigned int ac5;
unsigned int ac6;
int b1;
int b2;
int mb;
int mc;
int md;
long b5;

//----- Acelerometro -----
int reading = 0;
int val=0;
int X0,X1,X_out;
int Y0,Y1,Y_out;
int Z1,Z0,Z_out;
double Xg,Yg,Zg;
//------Magnetometro ------
int XM,YM,ZM;
void setup()
{
  Serial.begin(9600);//Recomendable a 9600 baudios
  Wire.begin();
  bmp085Calibration();
  gyroInit();
  Wire.beginTransmission(ADXAddress);
  Wire.write(Register_2D);
  Wire.write(8);
  //Inicia la biblioteca Wire (I2C)
  Wire.beginTransmission(Compas);
  // Modo registro es inicializado
  Wire.write(0x02);
  // Configuracion de registro pag 11 Datasheet
  Wire.write(0x00);
```

```
  Wire.endTransmission(); //Para la transmision del
acelerometro
}
void loop()
{
  float temperature =
bmp085GetTemperature(bmp085ReadUT());
  float pressure = bmp085GetPressure(bmp085ReadUP());
  float atm = pressure / 101325;
  float altitude = calcAltitude(pressure);
  //--------------X
  Wire.beginTransmission(ADXAddress); // Transmitir al
dispositivo
  Wire.write(Register_X0);
  Wire.write(Register_X1);
  Wire.endTransmission();
  Wire.requestFrom(ADXAddress,2);
  if(Wire.available()<=2)
  {
    X0 = Wire.read();
    X1 = Wire.read();
    X1=X1<<8;
    X_out=X0+X1;
  }

  //------------------Y
  Wire.beginTransmission(ADXAddress); // Transmitir al
dispositivo
  Wire.write(Register_Y0);
  Wire.write(Register_Y1);
  Wire.endTransmission();
  Wire.requestFrom(ADXAddress,2);
  if(Wire.available()<=2)
  {
    Y0 = Wire.read();
    Y1 = Wire.read();
    Y1=Y1<<8;
    Y_out=Y0+Y1;
  }
  //------------------Z
  Wire.beginTransmission(ADXAddress); // Transmitir al
dispositivo
  Wire.write(Register_Z0);
  Wire.write(Register_Z1);
  Wire.endTransmission();
  Wire.requestFrom(ADXAddress,2);
  if(Wire.available()<=2)
  {
    Z0 = Wire.read();
    Z1 = Wire.read();
    Z1=Z1<<8;
    Z_out=Z0+Z1;
  }
  //
  Xg=X_out/256.0;
  Yg=Y_out/256.0;
  Zg=Z_out/256.0;
    // Inicia lectrura
  Wire.beginTransmission(Compas);
  Wire.write(0x03); //Selecionamos registro 3 (X MSB
register)
  Wire.endTransmission();
  Wire.requestFrom(Compas, 6);//6 bits
  if(6<=Wire.available())
  {
    XM = Wire.read()<<8; //X MSB
    XM |= Wire.read();   //X LSB
    ZM = Wire.read()<<8; //Z MSB
    ZM |= Wire.read();   //Z LSB
    YM = Wire.read()<<8; //Y MSB
    YM |= Wire.read();   //Y LSB
  }
  float heading=atan2(XM, YM)/0.0174532925;
  if(heading < 0) heading+=360;
  heading=360-heading;// N=0/360, E=90, S=180, W=270
  //-----------Acelerometro----
  Serial.print("I");
  Serial.print(",");
  Serial.print(Xg*10);
  Serial.print(",");
  Serial.print(Yg*10);
  Serial.print(",");
  Serial.print(Zg*10);
  //----------Barometro--------
  Serial.print(",");
  Serial.print(temperature, 2);
  Serial.print(",");
  Serial.print(pressure, 0);
  Serial.print(",");
  Serial.print(atm, 4);
  Serial.print(",");
  Serial.print(altitude, 2);
  //----------Gyro------------
  gyroReadRaw();
  gyroCaculate();
  Serial.print(",");
  Serial.print(gyroRate[XG]);
  Serial.print(",");
  Serial.print(gyroRate[YG]);
  Serial.print(",");
  Serial.print(gyroRate[ZG]);
  //-----------Magnetometro----
  Serial.print(",");
  Serial.print(XM);
  Serial.print(",");
  Serial.print(YM);
  Serial.print(",");
  Serial.print(ZM);
  Serial.print(",");
  //Serial.print("heading= ");
  Serial.println(heading);
  delay(1000);
}
//--------------------*** DECLARACION DE FUNCIONES
***-----------------------------
//--------------------***        BAROMETRO        ***-------------
----------------
// almacena todos los valores de calibración en BMP085
```

```
// Las variables globales. valores de calibración son necesarios
para
// Calcular la temperatura y la presión
// Esta función debe ser llamada / ha desencadenado el inicio
del programa
void bmp085Calibration()
{
  ac1 = bmp085ReadInt(0xAA);
  ac2 = bmp085ReadInt(0xAC);
  ac3 = bmp085ReadInt(0xAE);
  ac4 = bmp085ReadInt(0xB0);
  ac5 = bmp085ReadInt(0xB2);
  ac6 = bmp085ReadInt(0xB4);
  b1 = bmp085ReadInt(0xB6);
  b2 = bmp085ReadInt(0xB8);
  mb = bmp085ReadInt(0xBA);
  mc = bmp085ReadInt(0xBC);
  md = bmp085ReadInt(0xBE);
}

// Calcular la presión. Los valores de calibración deben ser
conocidos
// B5 eh También es necesario, por lo que la función
bmp085GetTemperature (...) debe
// Se ejecutará primero.
// La función devuelve la presión en unidades de Pa
float bmp085GetTemperature(unsigned int ut)
{
  long x1, x2;

  x1 = (((long)ut - (long)ac6)*(long)ac5) >> 15;
  x2 = ((long)mc << 11)/(x1 + md);
  b5 = x1 + x2;
  float temp = ((b5 + 8)>>4);
  temp = temp /10;

  return temp;
}


long bmp085GetPressure(unsigned long up){
  long x1, x2, x3, b3, b6, p;
  unsigned long b4, b7;

  b6 = b5 - 4000;
  // Calcula B3
  x1 = (b2 * (b6 * b6)>>12)>>11;
  x2 = (ac2 * b6)>>11;
  x3 = x1 + x2;
  b3 = (((((long)ac1)*4 + x3)<<OSS) + 2)>>2;

  // Calcula B4
  x1 = (ac3 * b6)>>13;
  x2 = (b1 * ((b6 * b6)>>12))>>16;
  x3 = ((x1 + x2) + 2)>>2;
  b4 = (ac4 * (unsigned long)(x3 + 32768))>>15;

  b7 = ((unsigned long)(up - b3) * (50000>>OSS));
  if (b7 < 0x80000000)
    p = (b7<<1)/b4;
  else
    p = (b7/b4)<<1;

  x1 = (p>>8) * (p>>8);
  x1 = (x1 * 3038)>>16;
  x2 = (-7357 * p)>>16;
  p += (x1 + x2 + 3791)>>4;

  long temp = p;
  return temp;
}


char bmp085Read(unsigned char address)
{
  unsigned char data;

  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(BMP085_ADDRESS, 1);
  while(!Wire.available())
    ;

  return Wire.read();
}


int bmp085ReadInt(unsigned char address)
{
  unsigned char msb, lsb;

  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(BMP085_ADDRESS, 2);
  while(Wire.available()<2)
    ;
  msb = Wire.read();
  lsb = Wire.read();

  return (int) msb<<8 | lsb;
}


unsigned int bmp085ReadUT(){
  unsigned int ut;

  // Write 0x2E into Register 0xF4

  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF4);
  Wire.write(0x2E);
  Wire.endTransmission();

  // Wait at least 4.5ms
  delay(5);
```

```
  // Read two bytes from registers 0xF6 and 0xF7
  ut = bmp085ReadInt(0xF6);
  return ut;
}

// Read the uncompensated pressure value
unsigned long bmp085ReadUP(){

  unsigned char msb, lsb, xlsb;
  unsigned long up = 0;

  // Write 0x34+(OSS<<6) into register 0xF4
  // Request a pressure reading w/ oversampling setting
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF4);
  Wire.write(0x34 + (OSS<<6));
  Wire.endTransmission();

  // Wait for conversion, delay time dependent on OSS
  delay(2 + (3<<OSS));

  // Read register 0xF6 (MSB), 0xF7 (LSB), and 0xF8 (XLSB)
  msb = bmp085Read(0xF6);
  lsb = bmp085Read(0xF7);
  xlsb = bmp085Read(0xF8);

  up = (((unsigned long) msb << 16) | ((unsigned long) lsb <<
8) | (unsigned long) xlsb) >> (8-OSS);

  return up;
}

void writeRegister(int deviceAddress, byte address, byte val) {
  Wire.beginTransmission(deviceAddress); // start
transmission to device
  Wire.write(address);       // send register address
  Wire.write(val);         // send value to write
  Wire.endTransmission();     // end transmission
}

int readRegister(int deviceAddress, byte address){

  int v;
  Wire.beginTransmission(deviceAddress);
  Wire.write(address); // register to read
  Wire.endTransmission();

  Wire.requestFrom(deviceAddress, 1); // read a byte

  while(!Wire.available()) {
    // waiting
  }

  v = Wire.read();
  return v;
}

float calcAltitude(float pressure)
{
  float A = pressure/101325;
  float B = 1/5.25588;
  float C = pow(A,B);
  C = 1 - C;
  C = C /0.0000225577;

  return C;
}
//------------------------ ***   FIN DE BAROMETRO  ***-------
----------------------
//------------------------ ***      INICIO GYRO    ***----------
----------
void gyroInit()
{
  Wire.begin();
  Wire.beginTransmission(0x69);
  Wire.write(0x20);
  Wire.write(0x0F);
  Wire.endTransmission();
  Wire.beginTransmission(0x69);
  Wire.write(0x23);
  Wire.write(0x90);
  Wire.endTransmission();
  gyroInitState = true;
}
//Inicializacion de comunicacion y direccion I2C
//Registros tomados del datasheet (L3G4200D)
void gyroReadRaw()
{
  byte highByte, lowByte;
  Wire.beginTransmission(0x69);
  Wire.write(168);
  Wire.endTransmission();

  Wire.requestFrom(0x69, 6);
  while(Wire.available() < 6);

  lowByte = Wire.read();
  highByte = Wire.read();
  gyroRaw[XG] = ((highByte<<8)|lowByte);

  lowByte = Wire.read();
  highByte = Wire.read();
  gyroRaw[YG] = ((highByte<<8)|lowByte);
  gyroRaw[YG] = - gyroRaw[YG];

  lowByte = Wire.read();
  highByte = Wire.read();
  gyroRaw[ZG] = ((highByte<<8)|lowByte);
  gyroRaw[ZG] = -gyroRaw[ZG];

  if(calibrated) {
    gyroRaw[XG] -= gyroCal[XG];
    gyroRaw[YG] -= gyroCal[YG];
    gyroRaw[ZG] -= gyroCal[ZG];
  }
}
//Conversion a grados/segundo
void gyroCaculate()
{
```

```
  gyroRate[XG] = (gyroRate[XG] * 0.8) + ((gyroRaw[XG] /
57.14286) * 0.2);
  gyroRate[YG] = (gyroRate[YG] * 0.8) + ((gyroRaw[YG] /
57.14286) * 0.2);
  gyroRate[ZG] = (gyroRate[ZG] * 0.8) + ((gyroRaw[ZG] /
57.14286) * 0.2);
}
//-------------------------***    FIN GYRO     ***------------------
```

Este programa manda todos los datos a través del puerto serial.

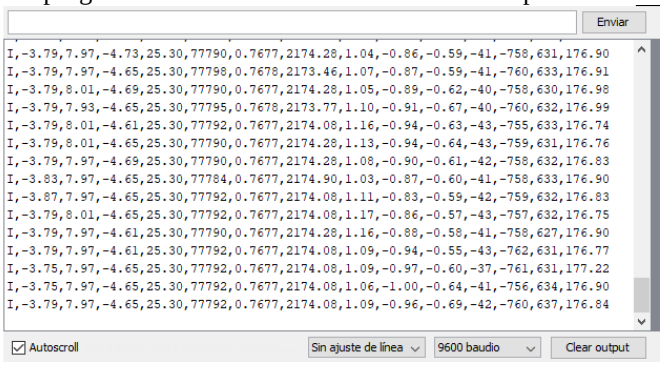


Figura 14. Todos los datos de la IMU a través del monitor serie.

2. Adquisición de datos

El programa de LabVIEW lee los datos del Arduino a través de la comunicación serie (ver "Introducción al entorno de desarrollo de LabVIEW"). Utilizamos el mismo tiempo de muestreo de la tarjeta Arduino para evitar que aparezcan símbolos extraños durante la lectura.

Los datos leídos, mostrados en *read buffer*, se deben separar. Debido a que son muchos datos; utilizar varios bloques *Match Pattern* para separar la información, *Fract/Exp String to Number* para convertir datos tipo String a Numéricos, *Bundle* para juntar varios elementos en un Cluster y así graficar varios datos en una sola gráfica y distintos indicadores para mostrar la información.

El bloque *Match Pattern* se ubica *en Functions → Programming → String*, el bloque *Fract/Exp String to Number* en *Functions → Programming → String → Number/String Conversion* y el bloque *Bundle* en *Functions → Programming → Cluster, Class & Variant*.

Finalmente el programa con los bloques para guardar los datos dentro de un documento de texto (ver "Introducción al entorno de desarrollo de LabVIEW").

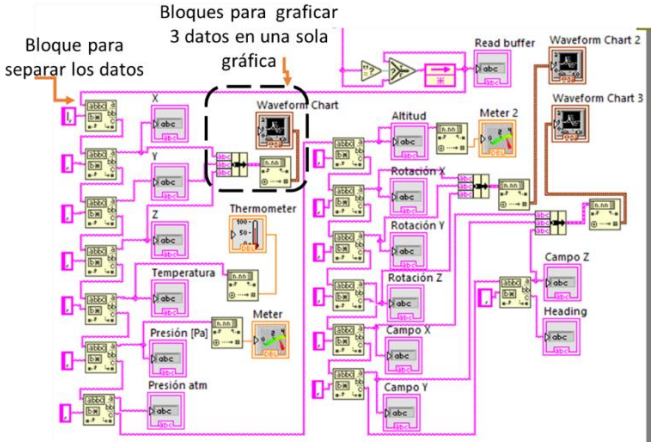


Figura 16. Separación de los datos y graficación.

3. Diseño final

A continuación se muestra el diagrama de bloques completo.

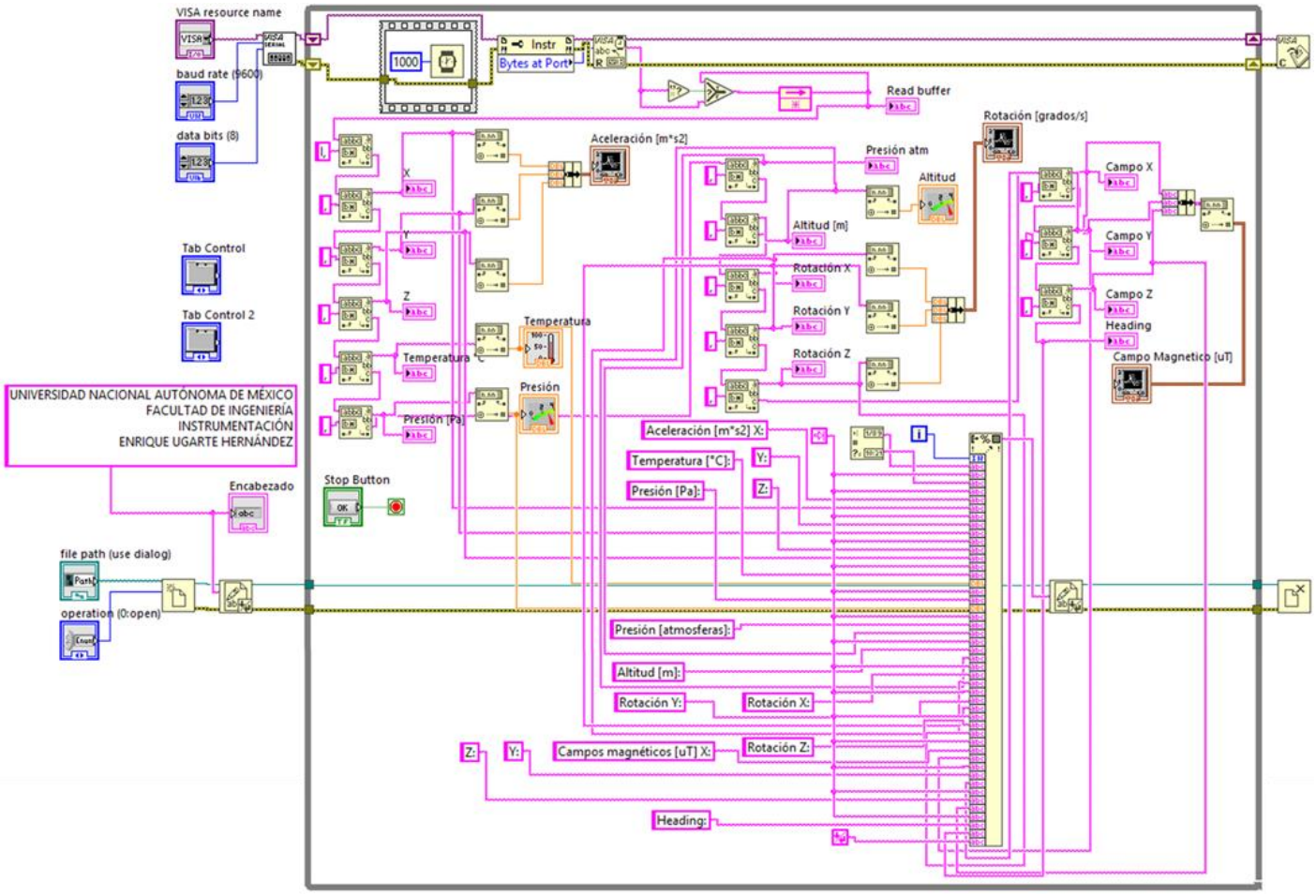


Figura 18. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña, la visualización de los datos de cada sensor de la IMU se muestran dentro de un segundo contenedor con pestañas y los elementos para guardar los datos en un documento de texto en la cuarta. . Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

Es posible que al correr el programa el acelerómetro no funcione adecuadamente, en este caso primero corremos el código de Arduino del ejemplo del acelerómetro y después nuestro código final.

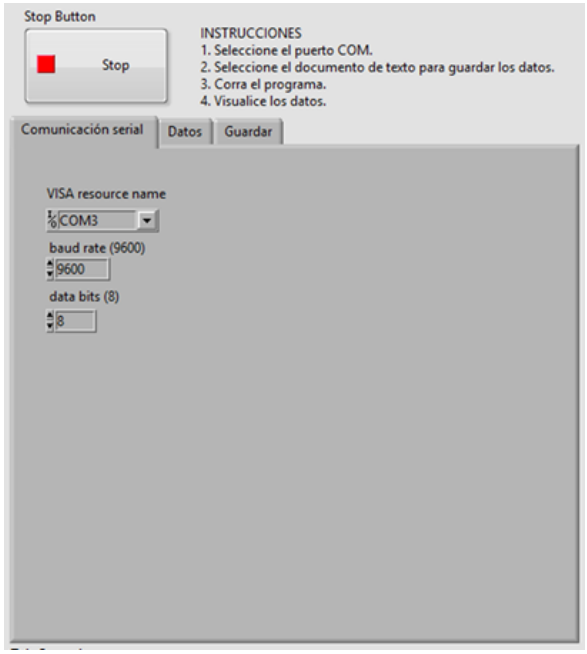


Figura 19. Primera página del contenedor con pestañas.

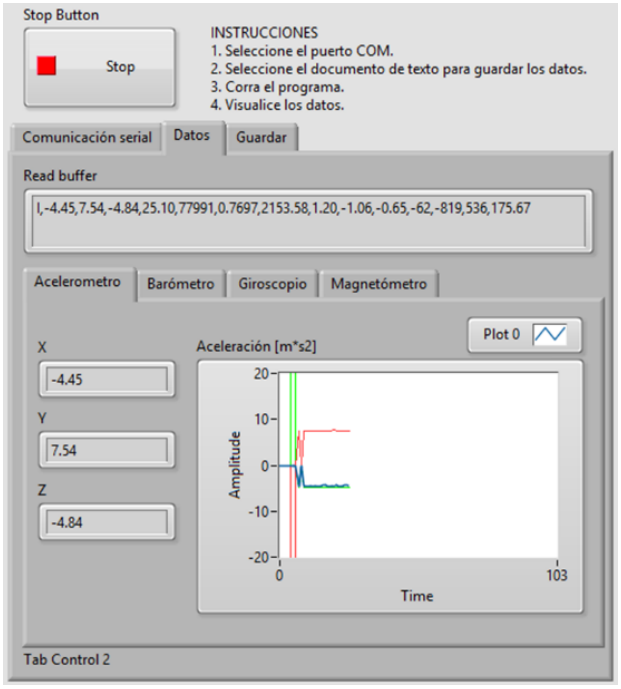


Figura 20. Segunda página del contenedor con pestañas. Datos del acelerómetro.

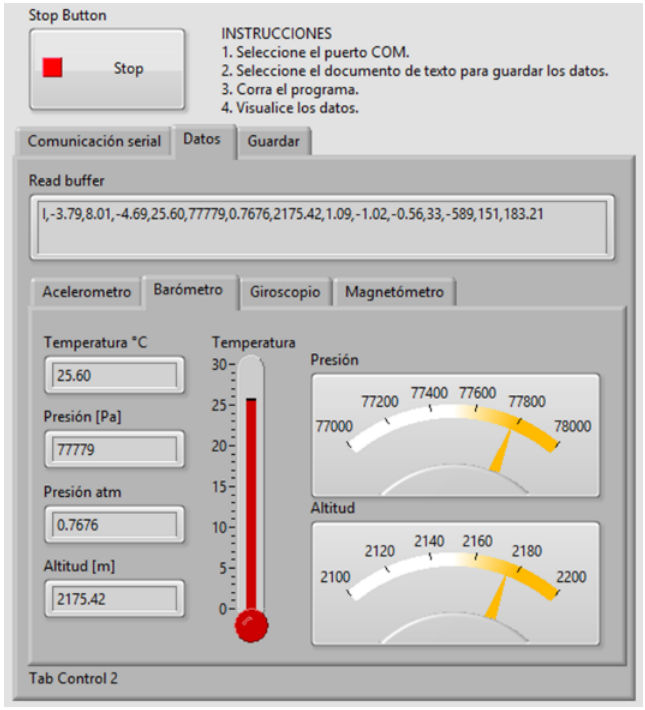


Figura 21. Segunda página del contenedor con pestañas. Datos del barómetro.

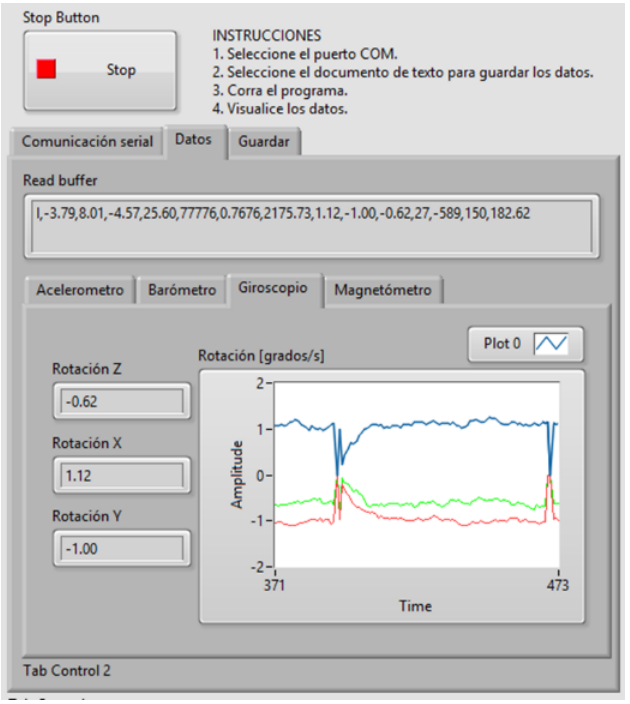


Figura 22. Segunda página del contenedor con pestañas. Datos del giroscopio.

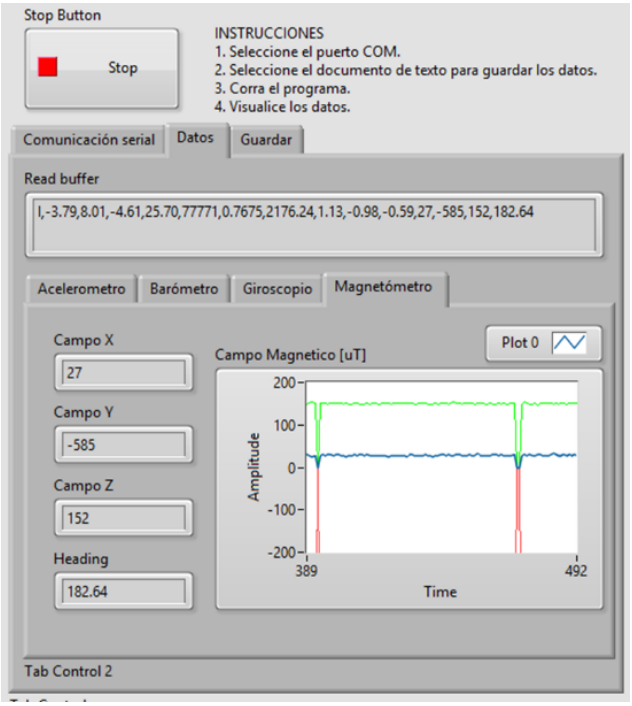


Figura 23. Segunda página del contenedor con pestañas. Datos del magnetómetro.

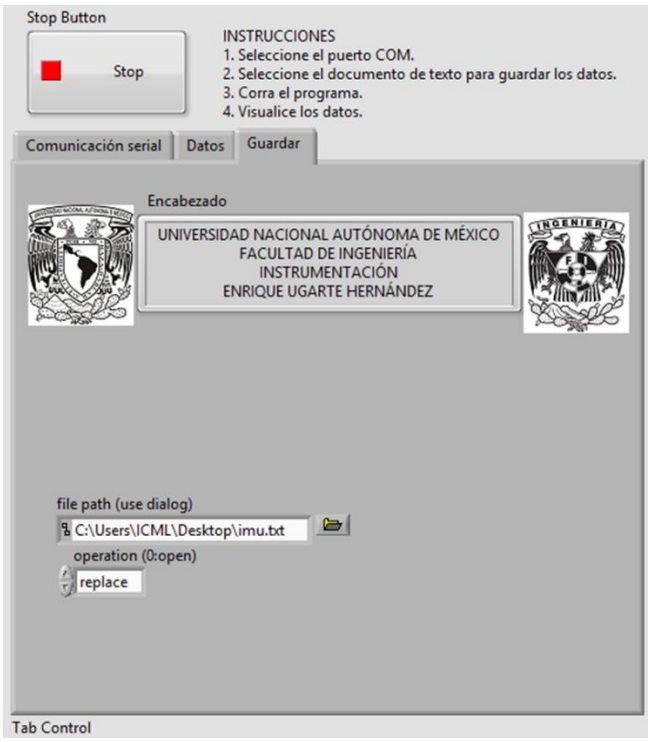


Figura 24. Tercera página del contenedor con pestañas.

V. FUENTES DE CONSULTA

[1] T.J.Deyle. (2015). Low Cost Inertial Measurement Unit , *Sandia National Laboratories*, Albuquerque , USA. Recuperado de http://prod.sandia.gov/techlib/access-control.cgi/2005/051548.pdf

[2] [Datasheet] Analog Devices, Inc. (2015). ADXL345. Analog Devices. Recuperado de http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf

[3] [Datasheet] Honeywell. (2010). 3-Axis Digital Compass IC HMC5883L. Recuperado de https://cdn-shop.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf

[4] Luis Martín Prieto. (2013). Diseño de un sistema de seguimiento automático del movimiento de la cabeza mediante el uso de microcontroladores. Tesis de Licenciatura, Universidad de Salamanca Escuela politécnica superior de Zamora, España.

[5] Edwinn Gamborino Castañeda (2013), Compensación Mecánica para la Estabilización de un Sistema de Procesamiento Visual, Tesis de Licenciatura, Facultad de Ingeniería UNAM, México D.F.

[6] [Datasheet] STMicroelectronics. (2011). L3G4200D: three axis digital output gyroscope. AN3393 Application note. Recuperado de https://www.elecrow.com/download/L3G4200_AN3393.pdf.

[7] Corona Rodríguez, L. G. (2014). Sensores y Actuadores: Aplicaciones con Arduino. Pág. 98,99., Grupo Editorial Patria.

[8] [Datasheet] BOSCH. (2009). BMP085 Digital pressure sensor. Bosch sensortech. Recuperado de https://www.sparkfun.com/datasheets/Components/General/BST-BMP085-DS000-05.pdf

[9] Olvera L., Marco A. (2014). Diseño, pruebas e implementación de un sistema de adquisición de datos y comunicación inalámbrica para la enseñanza de tecnología espacial. Tesis de maestría, instituto de astronomía, México DF.

[10] Vigouroux C., Demian P. (2010), Implementación de Unidad de Mediciones Inerciales (IMU) para robótica utilizando filtro de kalman. Tesis de Licenciatura, Universidad Simón Bolívar, Caracas Venezuela.

[11] [Datasheet] Honeywell. COMPASS HEADING USING MAGNETOMETERS. Recuperado de https://aerocontent.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/Magnetic__Literature_Application_notes-documents/AN203_Compass_Heading_Using_Magnetometers.pdf.

Práctica 16. Datos de estaciones mareográficas

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En este trabajo se explica cómo visualizar datos de internet usando el entorno de desarrollo de LabVIEW. Ingresando a un portal para obtener datos del nivel del mar y mostrándolos en una interfaz gráfica clara para el usuario.**

I. INTRODUCCIÓN

Hoy en día existen estaciones de monitoreo de diferentes tipos en muchas partes del mundo, cualquier persona puede acceder a dichos datos presentes en internet en tiempo real e incluso datos históricos.

Algunos ejemplos de estaciones de monitoreo son: estaciones meteorológicas, sísmicas, mareográficas, de monitoreo de calidad del aire, boyas oceanográficas, etc. En la actualidad se pueden obtener datos de diferentes variables monitoreadas desde satélites y con mucha exactitud, ya que dichas mediciones son calibradas y comparadas con mediciones in situ.

Algunos ejemplos de variables medidas vía satelital de la superficie del mar son: temperatura, salinidad, fluorescencia, clorofila, etc. En esta práctica se presenta un ejemplo de aplicación utilizando datos presentes en internet de estaciones mareográficas.

Es importante aclarar que en esta práctica no se aplica ningún algoritmo a los datos descargados de dichas estaciones; sin embargo, se recomienda que el alumno aplique algún algoritmo en el programa en LabVIEW a los datos descargados con la finalidad de analizar los datos e interpretarlos. Los algoritmos pueden ser muy sencillos, como obtener el valor máximo o mínimo, el promedio, la desviación estándar, análisis de varianza, etc. o un poco más avanzados como un análisis de componentes principales (PCA), una regresión lineal múltiple, etcétera.

En esta práctica no se propone enviar los datos a un archivo de texto porque no se considera necesario, pero si fuera el caso, enviar los datos a un archivo de texto es un ejercicio que se lleva a cabo en muchas de las prácticas presentes en este manual [1].

A. Datos de estaciones mareográficas

El Sistema Mundial de Observación del Nivel del Mar (The Global Sea Level Observing System GLOSS) fue establecido por la Comisión Oceanográfica Intergubernamental (COI) de la UNESCO en 1985 para establecer una red de observación del nivel del mar in situ bien diseñada y de alta calidad para respaldar una amplia base de usuarios operacionales y de investigación.

GLOSS contribuye al Sistema Mundial de Observación de Océanos (GOOS), particularmente sus módulos de servicio climático, costero y operacional, mediante el desarrollo progresivo de la red de medición del nivel del mar, intercambio de datos y sistemas de recolección y la preparación de productos a nivel del mar para varios grupos de usuarios.

La Comisión Oceanográfica Intergubernamental (COI) de la UNESCO promueve la cooperación internacional y coordina programas de investigación marina, servicios, sistemas de observación, mitigación de riesgos y desarrollo de capacidades para comprender y gestionar eficazmente los recursos de las zonas oceánicas y costeras. Al aplicar este conocimiento, la Comisión pretende mejorar los procesos de gobernanza, gestión, capacidad institucional y toma de decisiones de sus estados miembros con respecto a los recursos marinos y la variabilidad climática y fomentar el desarrollo sostenible del medio ambiente marino, en particular en los países en desarrollo.

La COI coordina la observación y el seguimiento oceánicos a través del Sistema Mundial de Observación de los Océanos, cuyo objetivo es desarrollar una red unificada que proporcione información e intercambio de datos sobre los aspectos físicos, químicos y biológicos del océano. Los gobiernos, la industria, los científicos y el público utilizan esta información para actuar en cuestiones marinas.

La importancia de tener acceso a los datos del nivel de marea radica en conocer mejor este tipo de fenómeno natural y sobre todo comprender su comportamiento durante un tsunami, ya que hoy en día existen sistemas de medición del nivel del mar en boyas oceanográficas ancladas en alta mar para la detección de tsunamis [2].

II. OBJETIVOS

- ✓ Utilizar el lenguaje de programación grafica LabVIEW para descargar datos de una dirección de internet utilizando el URL (*uniform resource locator*) de una estación mareográfica.
- ✓ Realizar una interfaz gráfica en LabVIEW para visualizar los datos del nivel del mar de una estación mareográfica.

III. MATERIALES Y SOFTWARE

- ➢ Equipo de cómputo con el software LabVIEW 2012 o superior.
- ➢ Portal de internet SEA LEVEL STATION MONITORING FACILITY.

IV. DESARROLLO

A. Uso del portal de navegación

Para obtener los datos del nivel del mar ir al siguiente sitio web:

http://www.ioc-sealevelmonitoring.org/

Allí vez dentro, seleccionar la pestaña “Map” para visualizar en un mapa las diferentes estaciones mareográficas disponibles.

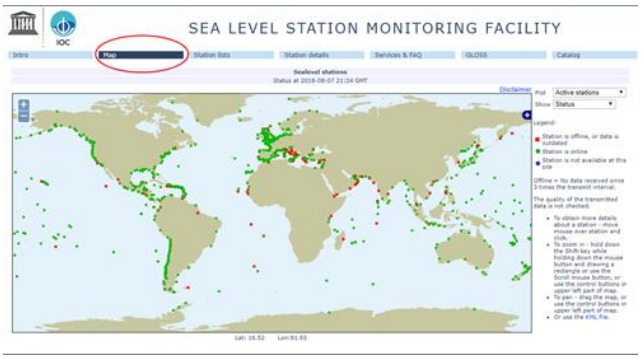


Figura 1. Mapa con las estaciones mareográficas.

Los puntos verdes corresponden a las estaciones mareográficas disponibles, mientras que los rojos indican las estaciones que se encuentran fuera de línea o sus datos no están actualizados. Se puede realizar un acercamiento para seleccionar la estación cómodamente.

Figura 2. Acercamiento y selección de una estación disponible.

Una vez seleccionada la estación deseada aparece una gráfica del nivel del mar y una tabla con datos sobre los sensores. Para el ejemplo mostrado en la figura 3, la estación cuenta con solo un sensor de tipo radar; sin embargo, puede haber más sensores de otro tipo para estimar el nivel de la marea y/o algún sensor para medir la batería, como se muestra en la figura 4.

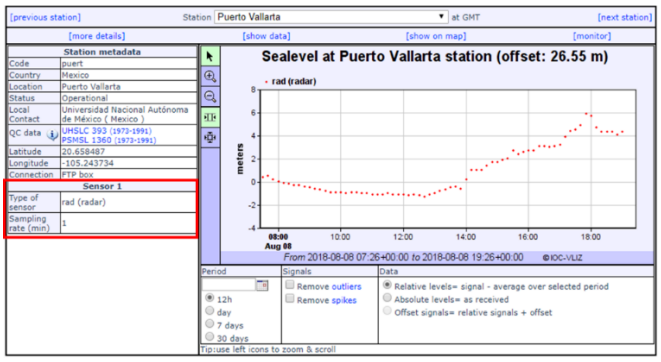


Figura 3. Número y tipo de sensores de la estación.

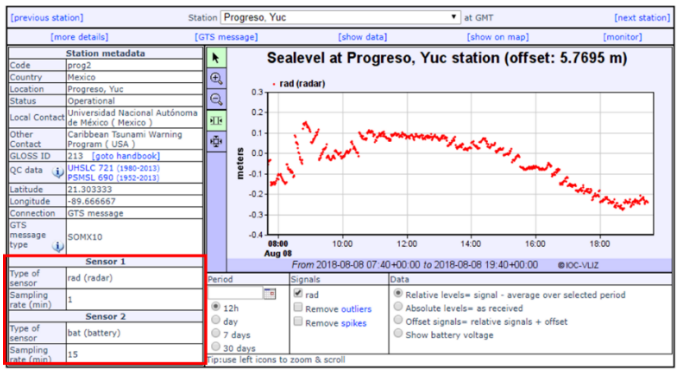


Figura 4. Estación mareográfica con dos sensores: uno de tipo radar para estimar el nivel de la marea y otro para la batería.

Entrar a *[show data]* para visualizar los datos de la estación.

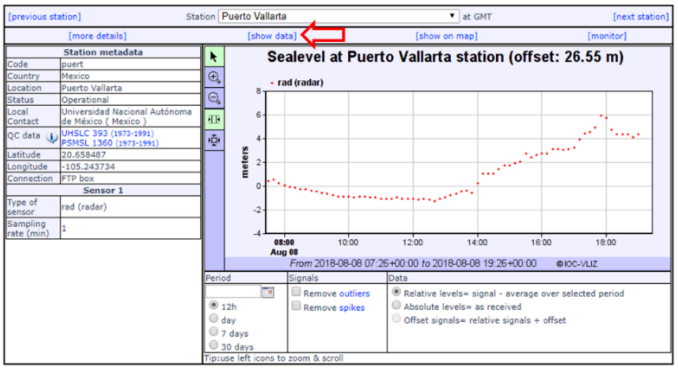


Figura 5. Selección de la opción *show data*.

Seleccionar el URL para descargar los datos en LabVIEW.

Tide gauge at Puerto Vallarta	
Time (UTC)	rad(m)
2018-08-08 07:40:00	27.1
2018-08-08 07:50:00	26.8
2018-08-08 08:00:00	26.6
2018-08-08 08:10:00	26.5
2018-08-08 08:20:00	26.4
2018-08-08 08:30:00	26.3
2018-08-08 08:40:00	26.3

Figura 6. URL necesario para descargar los datos.

B. Obtención de los datos crudos

LabVIEW permite leer el código HTML de una página de internet con la función *GET* que se encuentra en la siguiente paleta: *Programming* → *Data Communication* → *Protocols* → *HTTP Client*. Visualizar los datos crudos (*RAW Data*), para conocer el formato de los datos y las cadenas de caracteres que se pueden utilizar como referencia (*regular expression*) en la función *Match Pattern*.

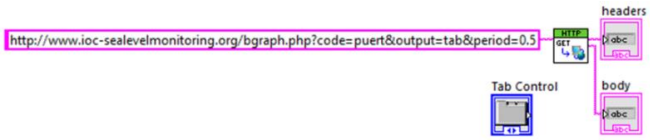


Figura 7. Diagrama de bloques para obtener os datos crudos.

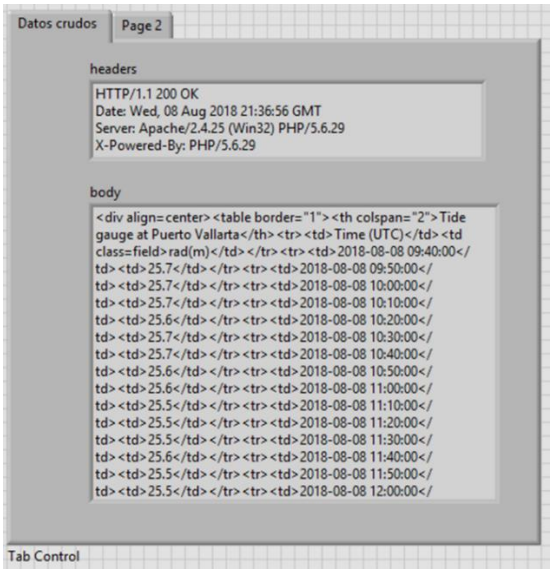


Figura 8. Datos crudos de una estación con un sensor.

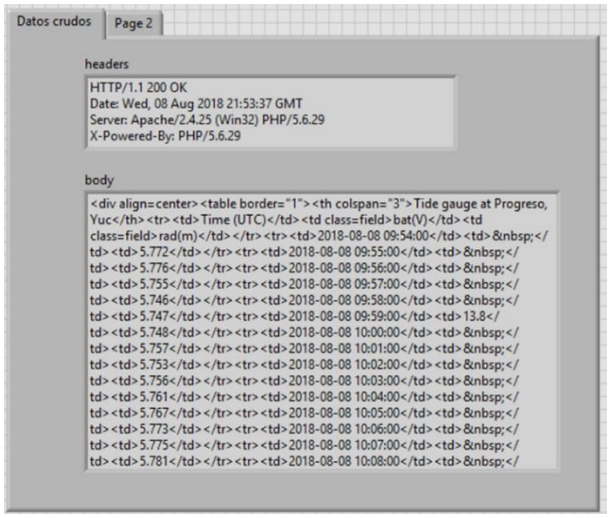


Figura 9. Datos crudos de una estación con dos sensores.

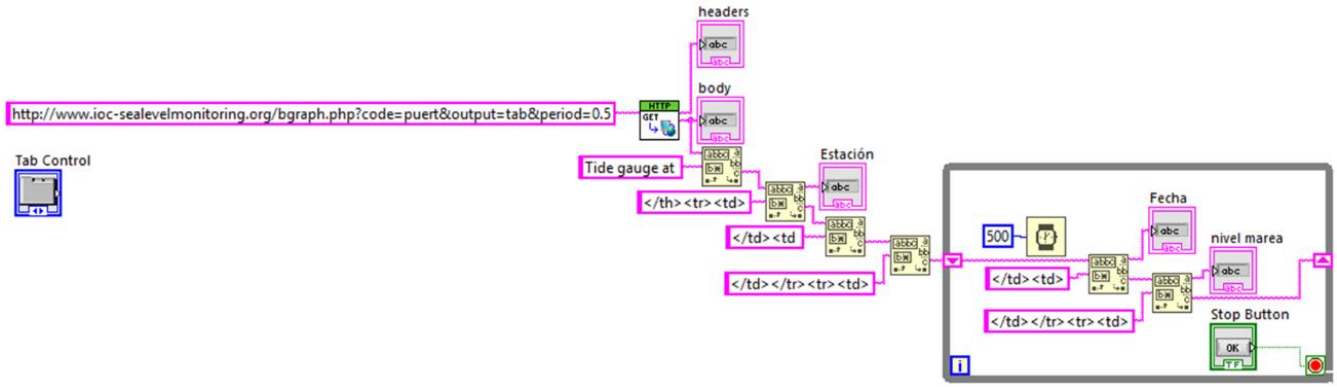


Figura 10. Extracción de los datos de una estación con solo un sensor (nivel de la marea).

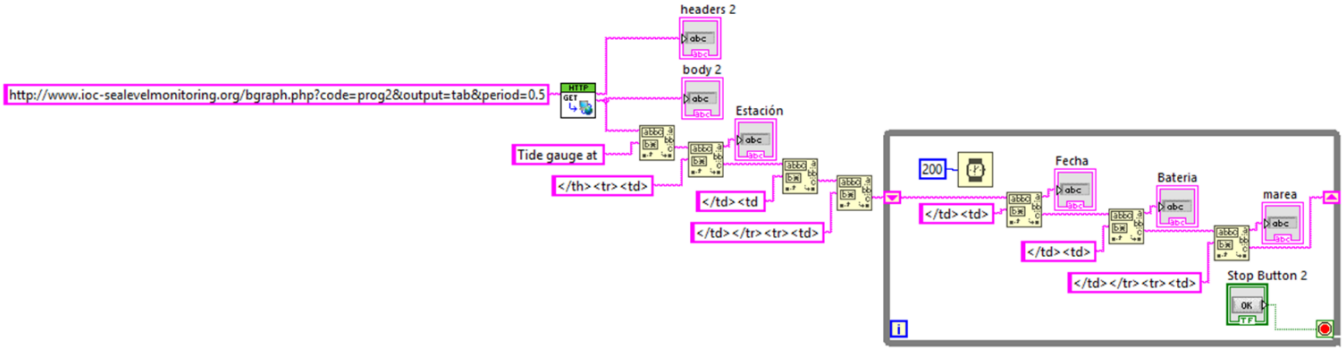


Figura 11. Extracción de los datos de una estación con dos sensores (nivel de la marea y batería).

C. Obtención de la gráfica de la marea y la batería

Antes de graficar los datos se debe dar formato a la fecha, para esto se ocupa el bloque *Scan From String*, ubicado en *Programming → String*. Posteriormente se usa el bloque *To Double Presicion Float* y *Bundle* para generar un cluster; estos se ubican en *Programming → Numeric → Conversion y Programming → Cluster, Class, & Variant VIs and Functions* respectivamente. Finalmente utilizar los bloques *Insert Into a Array* y *Array Subset* ubicados en *Programming → Array* para generar las gráficas [3].

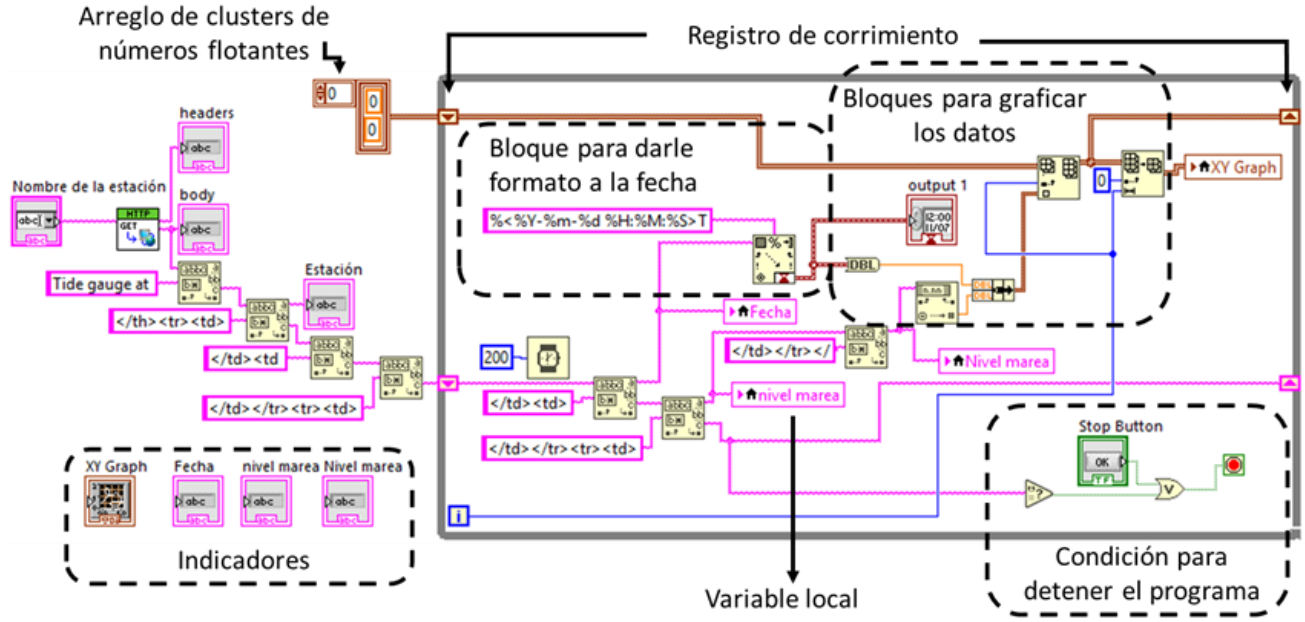


Figura 12. Diagrama de bloques para extraer la información de una estación con un solo sensor.

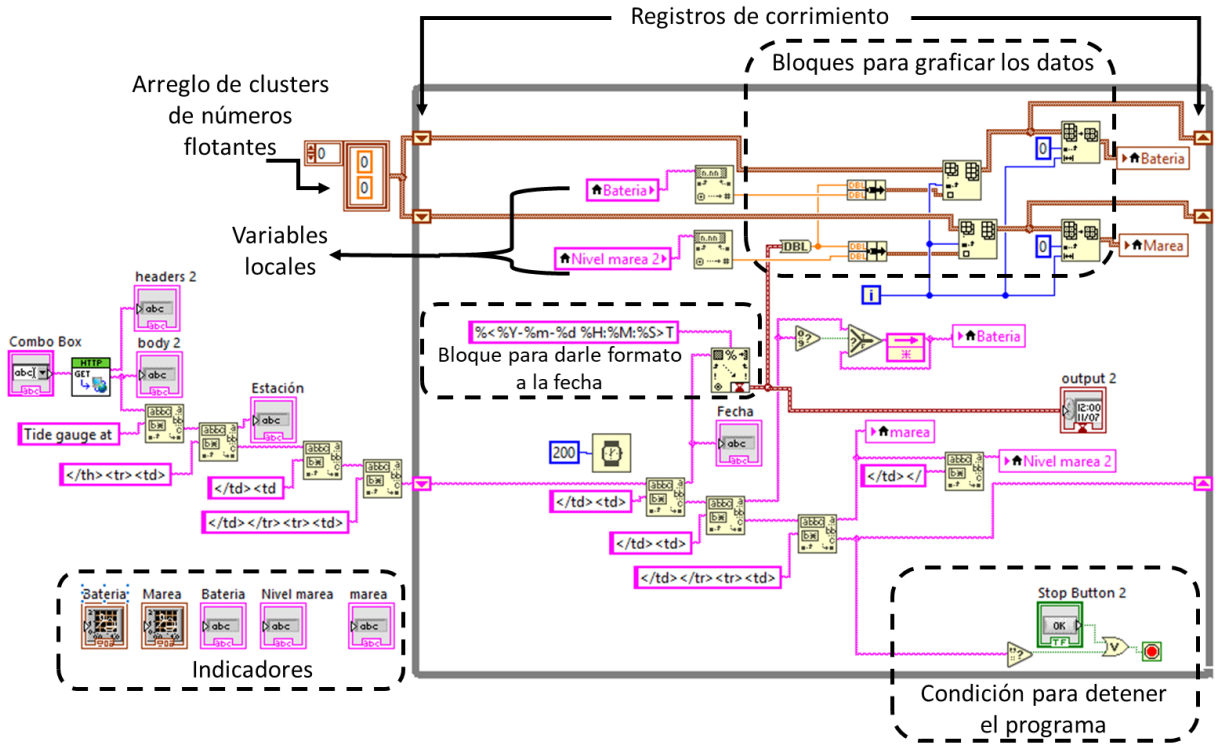


Figura 13. Diagrama de bloques para extraer la información de dos sensores (nivel del mar y bateria).

Para darle formato a la fecha se debe escribir en la terminal *format string* del *Scan From String* los comandos mostrados en la siguiente tabla:

input string	format string	default(s)	output(s)	remaining string
abc, xyz 12.3+56i 7200	%3s, %s%f%2d	—	abc	00
		—	xyz	
		0.00 +0.00 i	12.30 +56.00 i	
		—	72	
Q+1.27E-3 tail	Q%f t	—	1.27E-3	ail
0123456789	%3d%3d	—	12.00	6789
		—	345	
X:9.860 Z:3.450	X:%fY:%f	100 (I32)	10	Z: 3450
		100.00 (DBL)	100.00	
set49.4.2	set%d	—	49	.4.2
color: red	color: %s	blue (enum {red, green, blue})	red	—
abcd012xyz3	%[a-z]%d%[a-z]%d	—	abcd	—
			12	
			xyz	
			3	
welcome to LabVIEW, John Smith	%[^,],%s	—	welcome to LabVIEW	Smith
			John	
Time: 23:15:04.25 5/31/2004	Time: %<%H:%M:%S%2u%m/%d/%Y>T	1/1/1904	11:15:04.250 PM 5/31/2004	—

Figura 14. Tabla de ejemplos de *Formatting Strings*.

El *string constant* conectado al *format string* del *Scan From String* contiene el texto: **%<%Y-%m-%d %H:%M:%S>T**

Para ingresar la información en el *Combo Box*, hacer clic derecho y seleccionar la opción *Edit Items*. Posteriormente aparecerá un ventana con una tabla en la que se van a ingresar los casos del *Combo Box* y sus respectivos valores. Escribir en la columna *Items* el nombre de las estaciones mareográficas y en *Values* la dirección web. El cuadro *Values match ítems* no debe estar seleccionado para poder editar los valores.

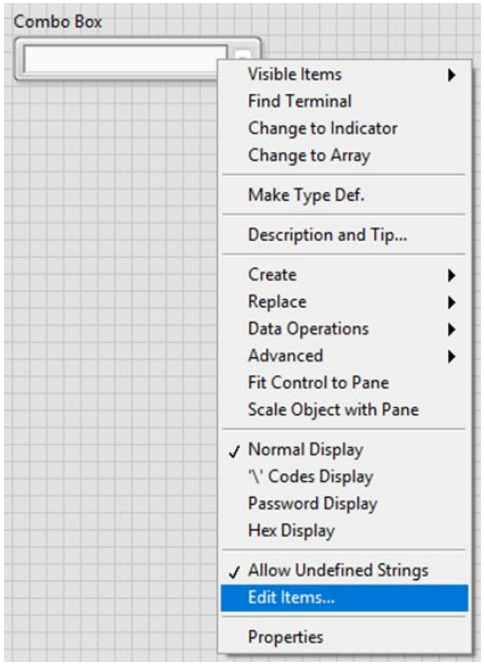


Figura 15. Opciones del *Combo Box*.

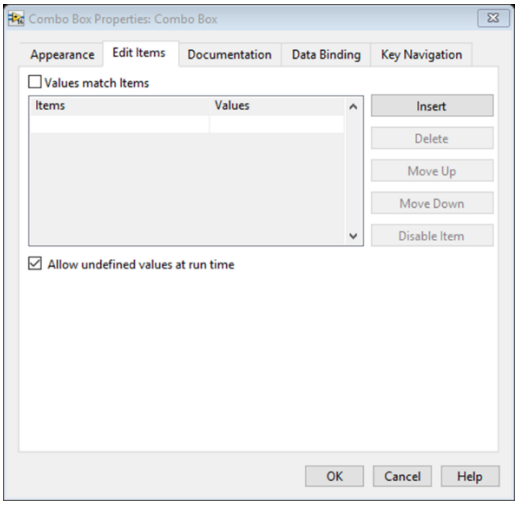


Figura 16. Combo Box vacío.

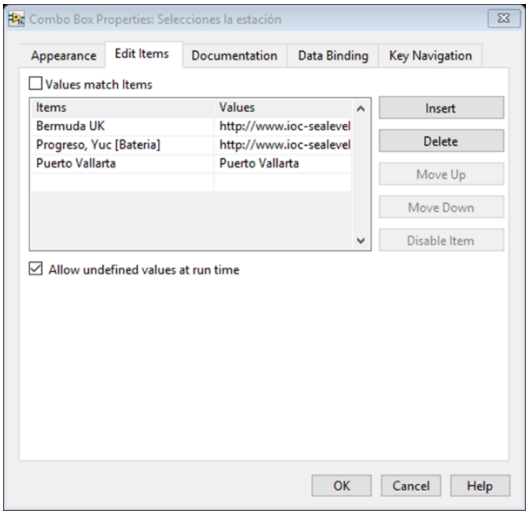


Figura 17. Combo Box con tres opciones.

Para crear el arreglo de clusters para inicializar el registro de corrimiento se crea un cluster con dos constantes de tipo número flotante y se colocan dentro de un arreglo. Para ajustar el cluster dentro del arreglo se hace clic derecho y se selecciona la opción *AutoSizing* [4].

Figura 18. Elementos para construir el arreglo de clusters.

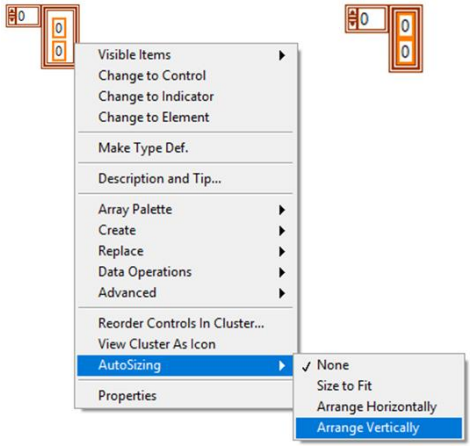


Figura 19. Ajustamiento del arreglo de clusters.

Para que la gráfica muestre de forma adecuada la fecha y hora hacer clic derecho y entrar ya sea en *X Scale* o *Y Scale* → *Properties*. Una vez dentro ir a *Display Format* y seleccionar *Absolue Time*.

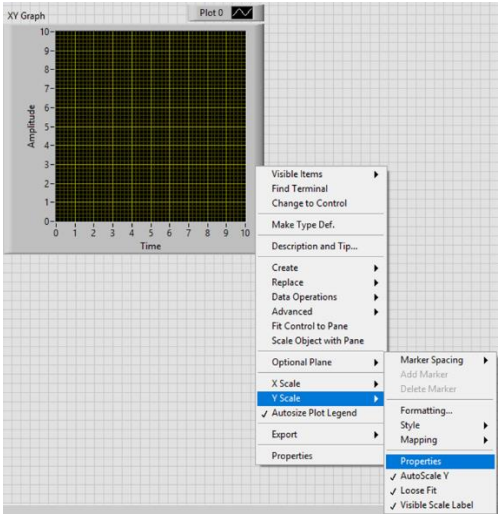


Figura 20. Opciones de la gráfica XY.

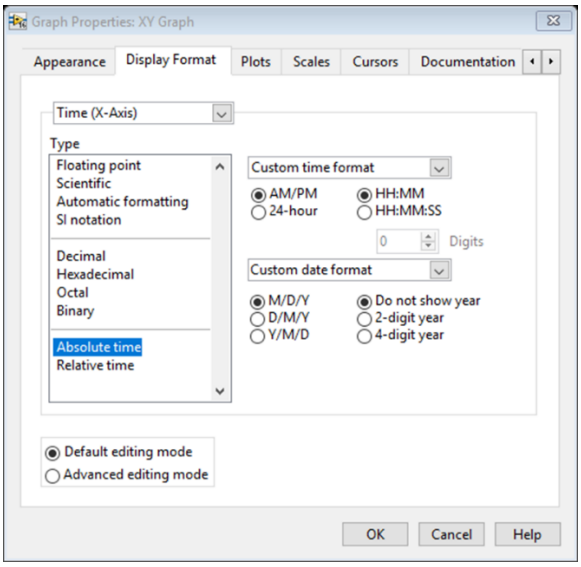


Figura 21. Opciones dentro de Display Format.

D. Diseño final

Unir el diagrama de bloques de la estación con un sensor y el de dos sensores dentro de un solo diagrama usando una estructura *Case* y un botón. Nótese que el botón de paro fue eliminado. Es recomendable agregar más estaciones dentro del Combo Box, en este caso particular se agregó solo una más. Al añadir más estaciones es importante considerar la cantidad de sensores de cada estación ya que este programa solo trabaja con estaciones con solo un sensor de nivel de marea y un sensor de carga de batería. En caso de querer trabajar con una estación con más sensores se debe modificar el programa.

Ejemplos de estaciones con solo un sensor:

- http://www.ioc-sealevelmonitoring.org/station.php?code=bmda
- http://www.ioc-sealevelmonitoring.org/station.php?code=alva
- http://www.ioc-sealevelmonitoring.org/station.php?code=ptis

A continuación se muestra el diagrama de bloques completo.

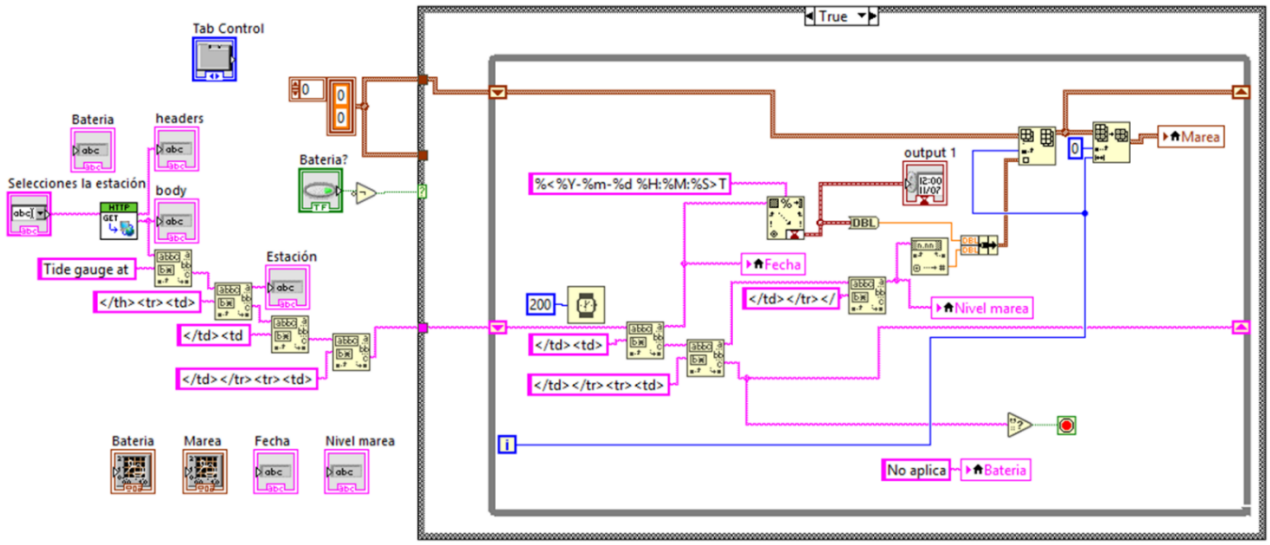


Figura 22. Diagrama de bloques completo (True).

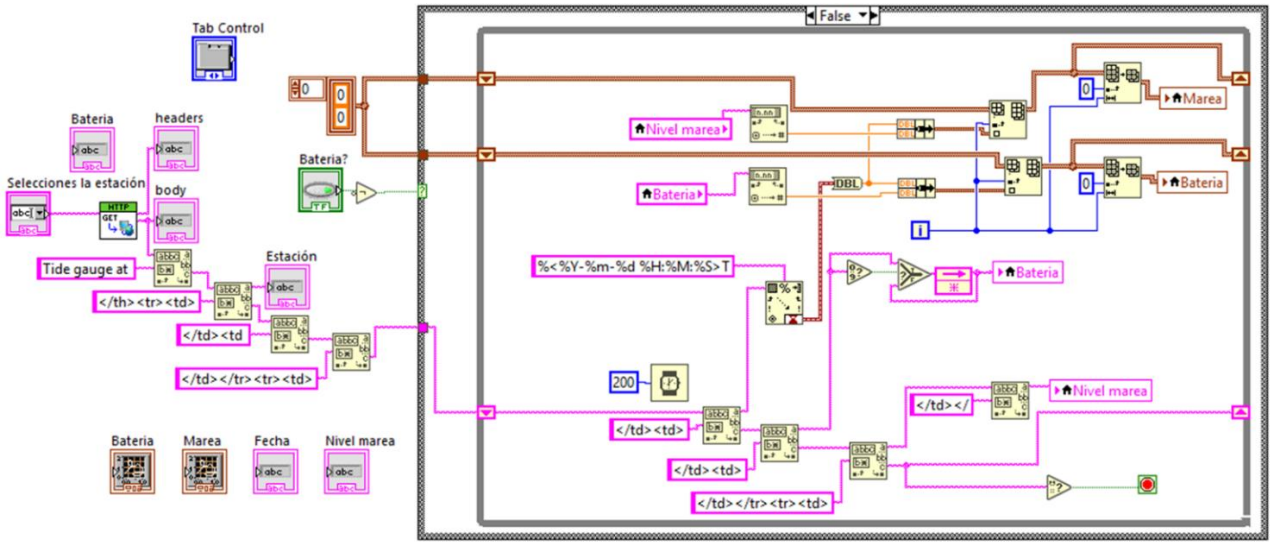


Figura 23. Diagrama de bloques completo (False).

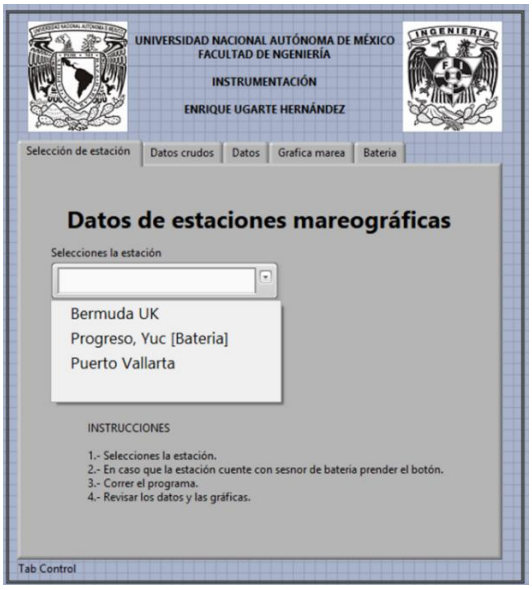


Figura 24. Primera página del contenedor con pestañas. Selección de la estación antes de correr el programa.

Antes de correr el programa se selecciona la estación mareográfica en la primera pestaña del contenedor con pestañas. En caso que la estación cuente con un sensor de batería se activa el botón. En la segunda pestaña se aprecian los datos crudos, en la tercera los datos ordenados, en la cuarta la gráfica del nivel de la marea y en la última se grafica la batería.

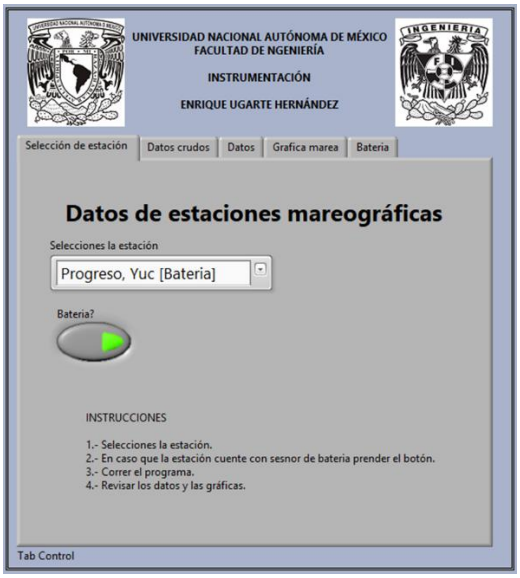


Figura 25. Primera página del contenedor con pestañas.

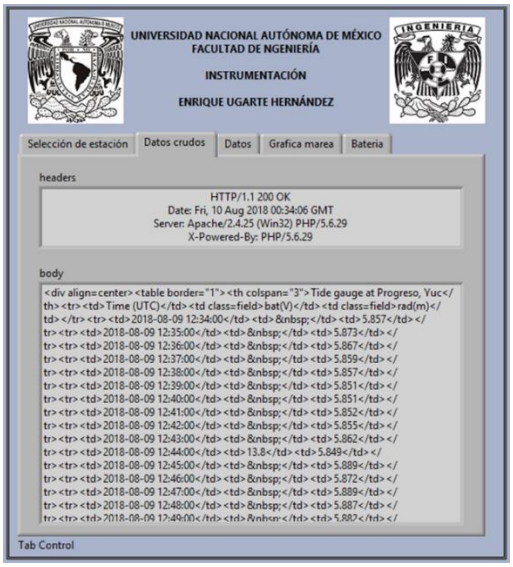


Figura 26. Segunda página del contenedor con pestañas.

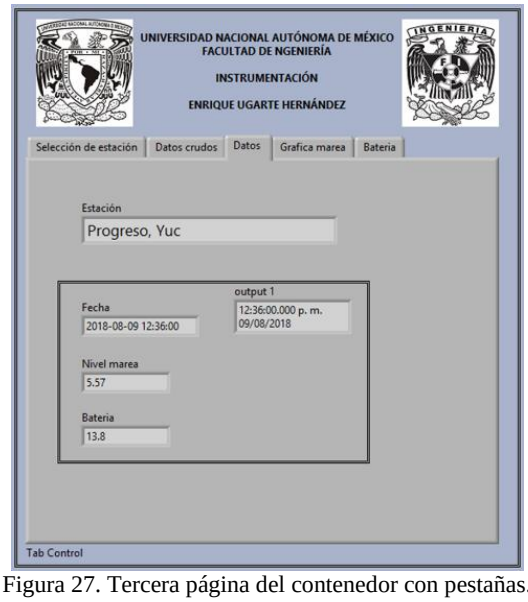


Figura 27. Tercera página del contenedor con pestañas.

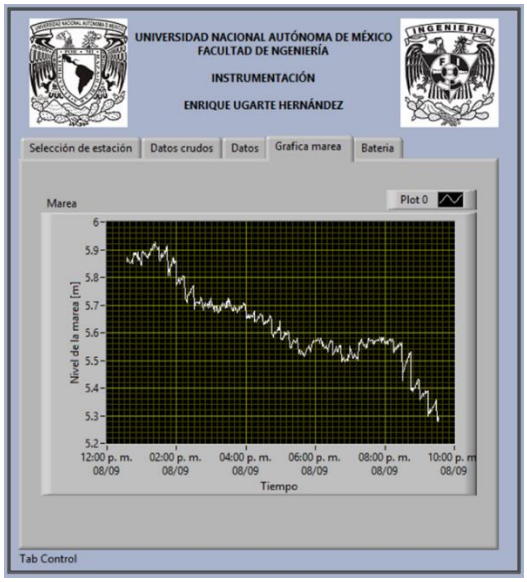


Figura 28. Cuarta página del contenedor con pestañas.

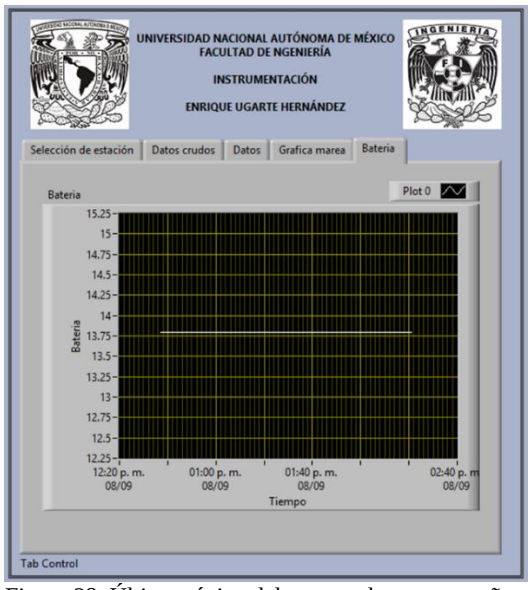


Figura 29. Última página del contenedor con pestañas.

V. FUENTES DE CONSULTA

[1] UNESCO. (2017). *About the Intergovernmental Oceanographic Commission (IOC)*. http://www.unesco.org. Recuperado de http://www.unesco.org/new/en/natural-sciences/ioc-oceans/about-us/

[2] Tabla de mareas. (2018). TIPOS DE MAREAS: PLEAMAR Y BAJAMAR; MAREAS VIVAS Y MAREAS MUERTAS. tablademareas.com. Recuperado de http://www.tablademareas.com/mareas/tipos-mareas

[3] National Instruments Corporation. (2014). ¿Cómo Establezco la Hora y Fecha en el Eje-X en una Gráfica en LabVIEW?. http://digital.ni.com. Recuperado de http://digital.ni.com/public.nsf/allkb/9BC8A4C39144F76D86257410006EC8F6

[4] Lajara, José y Pelegrí, José,(2007), LabVIEW Entorno gráfico de programación, Barcelona, España, Alfaomega.

Práctica 17. Datos de boyas oceanográficas

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En esta práctica se explica cómo descargar datos en tiempo real de boyas oceanográficas del programa National Data Bouy Center (NDBC) que pertenece al National Oceanic and Atmospheric Administration (NOAA) para posteriormente desplegar los datos en un panel frontal con gráficas.**

I. INTRODUCCIÓN

La Administración Nacional Oceánica y Atmosférica (*National Oceanic and Atmospheric Administration*, NOAA) es una agencia científica del Departamento de Comercio de los Estados Unidos cuyas actividades se centran en las condiciones de los océanos y la atmósfera. NOAA avisa del tiempo meteorológico, prepara cartas de mares y de cielos, guía sobre el uso y la protección de los recursos oceánicos y costeros y conduce estudios para mejorar el entendimiento y la administración del ambiente.

El *National Data Buoy Center* (NDBC) es parte del Servicio Meteorológico Nacional (NWS) de la Administración Nacional Oceánica y Atmosférica (NOAA); este diseña, desarrolla, opera y mantiene una red de recolección de datos de boyas y estaciones costeras. El NBDC está ubicado en el sur de Mississippi como inquilino en el Centro Espacial John C. Stennis, una instalación de la Administración Nacional de Aeronáutica y del Espacio (NASA) [1].

Para las tomar mediciones se usan boyas. Las boyas son señales flotantes situadas en el mar que comúnmente se encuentran ancladas al fondo. La principal función es la de orientar a los barcos mediante la señalización de canales, rutas navegables, punto de viraje, bifurcación, indicar puntos de peligro, áreas especiales y señalar la existencia de cables o cañerías submarinas [2].

Pueden ser ciegas o luminosas, con características diurna y/o nocturna. Existen diversas formas de boyas, cilíndricas, cónicas, esféricas, etcétera; de ellas, las dos primeras son las más usadas actualmente. Las boyas meteorológicas y oceanográficas incorporan sistemas de adquisición de datos para obtener datos meteorológicos y oceanográficos en alta mar.

El NDBC cuenta con el programa de evaluación en el océano profundo y notificación de tsunamis (DART® *Deep-ocean Assessment and Reporting of Tsunamis*). Los sistemas DART® consisten en un registrador de presión en el fondo del lecho marino (BPR) y una boya de superficie amarrada para comunicaciones en tiempo real. Un enlace acústico transmite datos desde el BPR en el lecho marino a la boya de superficie.

Figura 1. Boya de la bahía de Campeche [3].

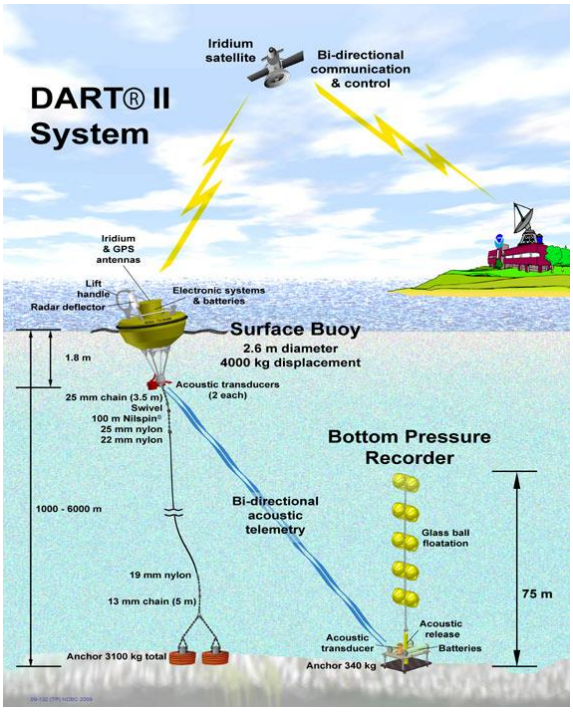


Figura 2. Sistema DART [4].

II. OBJETIVOS

- ✓ Describir las variables que monitoriza una boya oceanográfica del NDBC, así como las unidades de cada variable.
- ✓ Descargar los datos de una boya oceanográfica y los presentarlos en una interfaz gráfica en LabVIEW.

III. MATERIALES Y SOFTWARE

- Equipo de cómputo con el software LabVIEW 2012 o superior.
- Portal de Internet *National Data Buoy Center* (NDBC).

IV. DESARROLLO

A. Uso del portal de navegación

Entrar a la siguiente dirección de internet para obtener datos que se monitorizan en una boya del NDBC:

http://www.ndbc.noaa.gov/

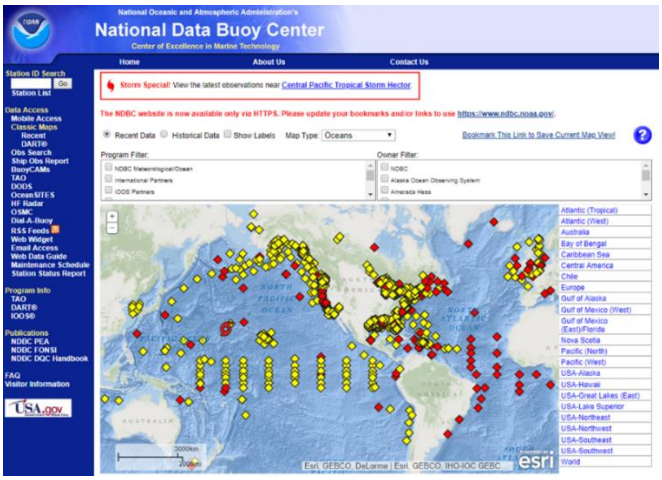


Figura 3. Página principal del NDBC. En la página se encuentra un mapa donde se muestran las ubicaciones de cada boya. Los puntos amarillos corresponden a las boyas disponibles para ver sus datos y los rojos a las boyas no se encuentran disponibles.

Seleccionar una boya oceanográfica dando clic en alguno de los puntos amarillos, cada uno de los cuales representa una boya con datos recientes. Es importante mencionar que cada boya contiene distinta información, razón por la cual se maneja la boya: Station 42055 (LLNR 1122) - BAY OF CAMPECHE - 214 NM NE of Veracruz, M.

https://www.ndbc.noaa.gov/station_page.php?station=42055

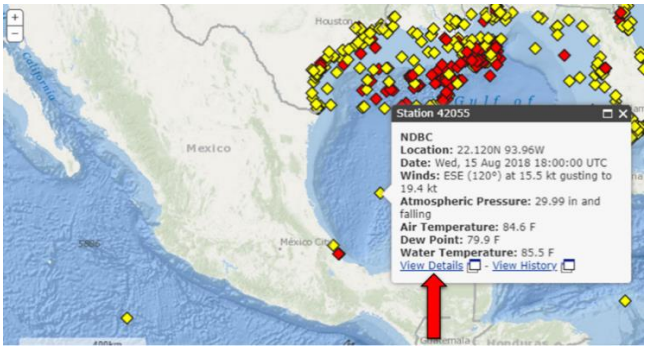


Figura 4. Selección de la opción *View Details*.

Al dar clic en la opción de *View Details* se abre una página donde se muestran las especificaciones de la boya. Al desplazarse hasta la parte inferior de esta nueva página hay varias opciones, de las cuales para esta práctica solo las dos primeras son de importancia: *Measurement Descriptions and Units*, donde se encuentra la descripción de todas y cada una de las mediciones que se realizan en dicha boya, así como las unidades correspondientes y *Real Time Data*, donde están los diferentes enlaces de datos de la boya.

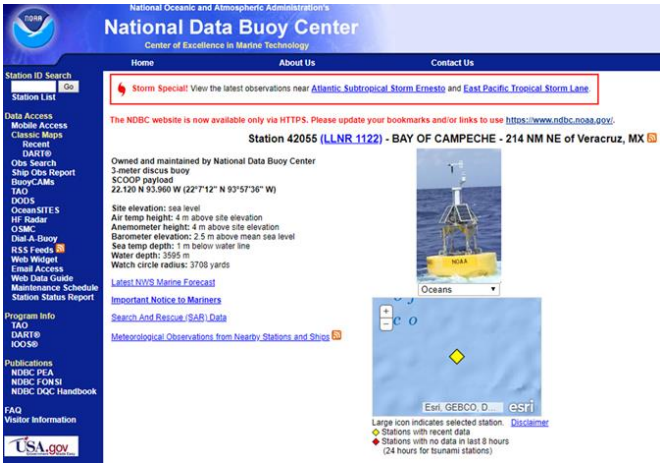


Figura 5. Página principal de la boya de la bahía de Campeche.

Entrar a *Measurement Descriptions and Units* del siguiente enlace para conocer a detalle las variables que se monitorean en dicha boya.

https://www.ndbc.noaa.gov/measdes.shtml

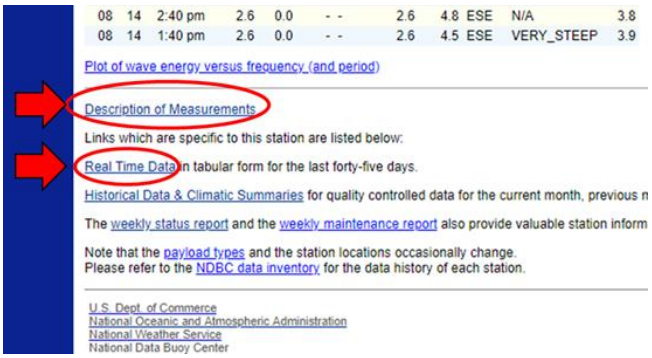


Figura 6. Final de la página principal de boya de la bahía de Campeche.

Los archivos en tiempo real (*Real Time Data*) generalmente contienen los últimos 45 días de datos "en tiempo real", datos que pasaron por controles automáticos de calidad y se distribuyeron tan pronto como se recibieron. Los archivos históricos han pasado por el análisis posterior al procesamiento y representan los datos enviados a los centros de archivo. Los formatos para ambos son generalmente los mismos, con la diferencia principal es el tratamiento de datos faltantes. Los datos perdidos en los archivos en tiempo real se indican con "MM", mientras que un número variable de 9 se utiliza para indicar datos faltantes en los archivos históricos, según el tipo de datos (por ejemplo: 999.0 99.0).

Datos meteorológicos estándar

WDIR Dirección del viento (la dirección en la que viene el viento en grados en el sentido de las agujas del reloj de la verdadera N) durante el mismo período utilizado para WSPD.

WSPD Velocidad del viento (m / s) promediada durante un período de ocho minutos para las boyas y un período de dos minutos para las estaciones terrestres. Informado por hora.

GST Velocidad máxima de ráfaga de 5 u 8 segundos (m / s) medida durante el período de ocho minutos o dos minutos.

WVHT La altura de ola significativa (metros) se calcula como el promedio del tercio más alto de todas las alturas de ola durante el período de muestreo de 20 minutos.

DPD El período de onda dominante (segundos) es el período con la energía de onda máxima.

APD Período de onda promedio (segundos) de todas las ondas durante el período de 20 minutos.

MWD La dirección desde la cual se aproximan las ondas en el período dominante (DPD). Las unidades son grados del norte verdadero, aumentando en el sentido de las agujas del reloj, con el norte como 0 (cero) grados y el este como 90 grados.

PRES Presión a nivel del mar (hPa).

ATMP Temperatura del aire (Celsius).

WTMP Temperatura de la superficie del mar (Celsius).

DEWP Temperatura del punto de rocío tomada a la misma altura que la medición de la temperatura del aire.

VIS Visibilidad de la estación (millas náuticas).

PTDY Tendencia de presión es la dirección (más o menos) y la cantidad de cambio de presión (hPa) durante un período de tres horas que termina en el momento de la observación.

TIDE El nivel del agua en pies arriba o abajo.

B. Obtención de los datos

Una vez que se haya seleccionado una boya, entrar a *Real Time Data*. Estando allí seleccionar *Real time standard meteorological data*.

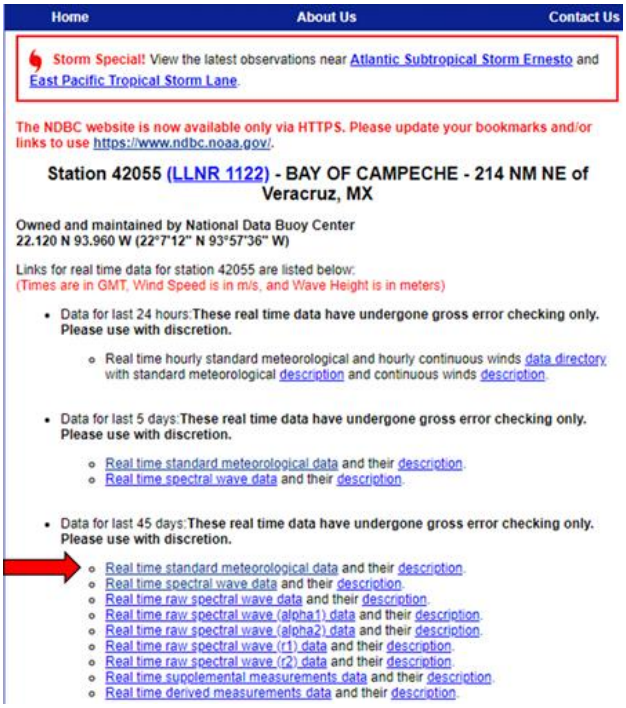


Figura 7. Enlaces dentro de *Real Time Data*.

Copiar el URL (*Uniform Resource Locator*) de la página.

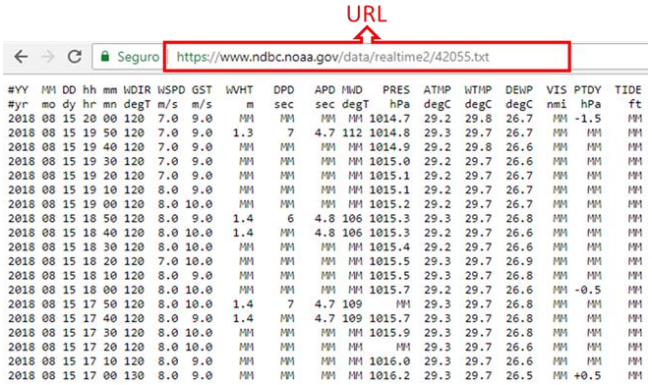


Figura 8. URL de la página de los datos a obtener.

LabVIEW permite leer el código HTML de una página de internet con la función *GET* que se encuentra en la siguiente paleta: *Programming → Data Communication → Protocols → HTTP Client*. Visualizar los datos crudos (*RAW Data*), para conocer el formato de los datos y las cadenas de caracteres que se pueden utilizar como referencia (*regular expression*) en la función *Match Pattern*. Se usa el bloque *String Length*, ucicado en *Functions → String* para obtener la longitud de la cadena de caracteres obtenidos de la página web; el valor entregado será muy útil para graficar los datos comenzando por el más antiguo y terminando con el más reciente.

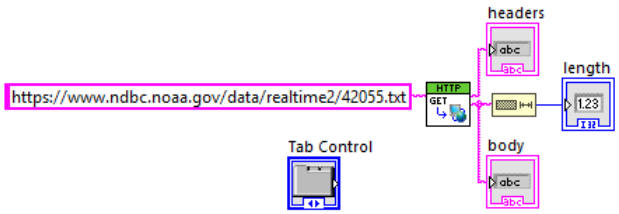


Figura 9. Diagrama de bloques para obtener los datos crudos.

Al correr el programa de la figura 10 se muestra en el panel frontal los datos crudos.

Utilizar el bloque *String Subset*, ubicado en *Functions* → *String* para ir separando la información. Para obtener los valores de las constantes numéricas se tienen que realizar muchas pruebas. Debido a que la longitud de la cadena obtenida de internet es bastante grande se debe crear un control para el bloque *GET* para incrementar el tiempo de espera para recibir toda la información. Se crean dos registros de corrimiento: uno para leer la cadena de datos de entrada y otro para ajustar la posición de la cadena comenzando con el valor del bloque *String Lengh* y disminuyendo progresivamente.

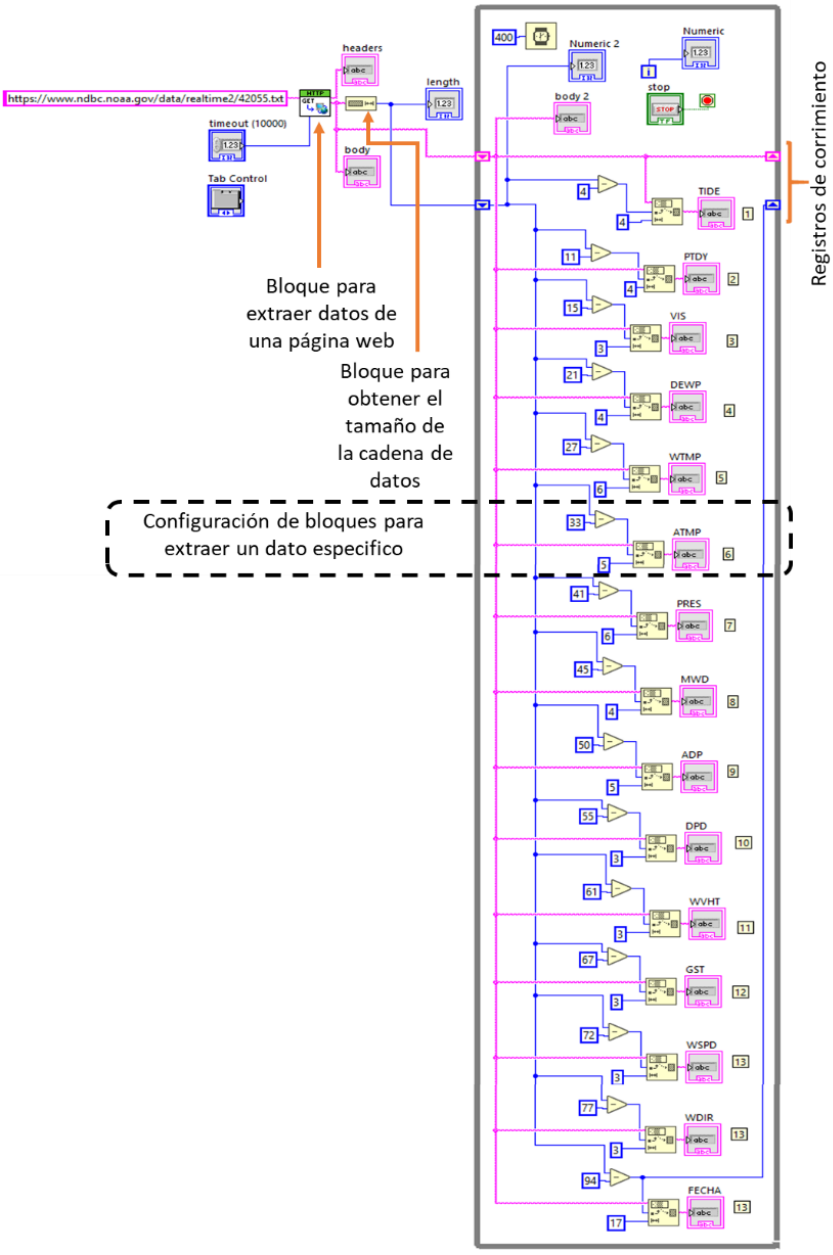


Figura 10. Diagrama de bloques para obtener todos los datos de la página de internet y mostrarlos con un indicador.

C. Obtención de las gráficas

Antes de graficar los datos se debe dar formato a la fecha, para esto se ocupa el bloque *Scan From String*, ubicado en *Programming* → *String*. Posteriormente se usa el bloque *To Double Presicion Float* y *Bundle* para generar un cluster; estos se ubican en *Programming* → *Numeric* → *Conversion y Programming* → *Cluster, Class, & Variant VIs and Functions*

respectivamente. Finalmente utilizar los bloques *Insert Into a Array* y *Array Subset* ubicados en *Programming* → *Array* para generar las gráficas.

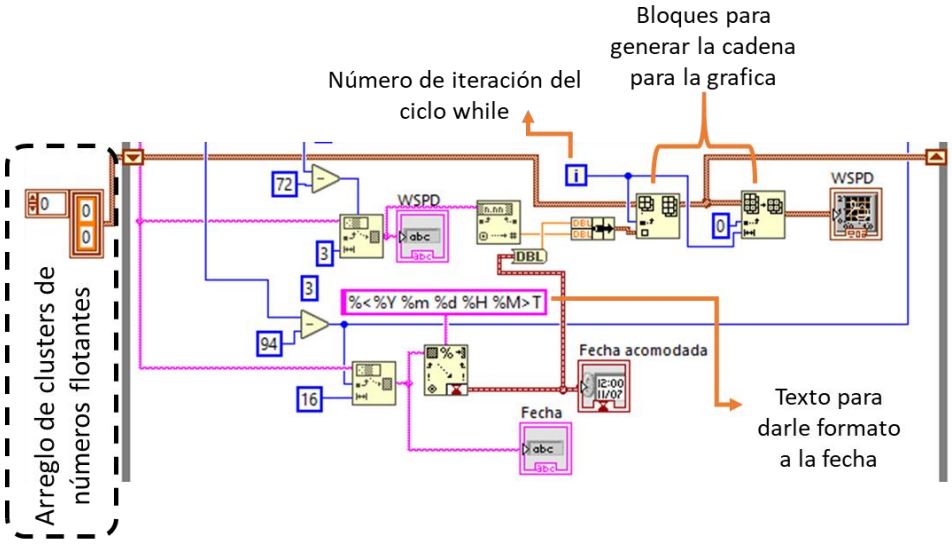


Figura 11. Conexión de bloques para generar una gráfica.

El *string constant* conectado al *format string* del *Scan From String* contiene el texto: **%<%Y%m%d %H%M>T**

Para crear el arreglo de clusters para inicializar el registro de corrimiento se crea un cluster con dos constantes de tipo número flotante y se colocan dentro de un arreglo. Para ajustar el cluster dentro del arreglo se hace clic derecho y se selecciona la opción *AutoSizing* [4].

Figura 12. Elementos para construir el arreglo de clusters.

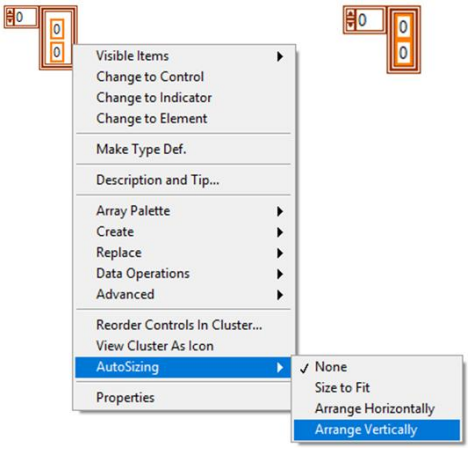


Figura 13. Ajustamiento del arreglo de clusters.

Finamente reemplazar la el URL de la página por un Combo Box y agregar otra boya con el mismo formato de datos.

Para ingresar la información en el *Combo Box*, hacer clic derecho y seleccionar la opción *Edit Items*. Posteriormente aparecerá un ventana con una tabla en la que se van a ingresar los casos del *Combo Box* y sus respectivos valores. Escribir en la columna *Items* el nombre de las estaciones mareográficas y en *Values* la dirección web. El cuadro *Values match ítems* no debe estar seleccionado para poder editar los valores.

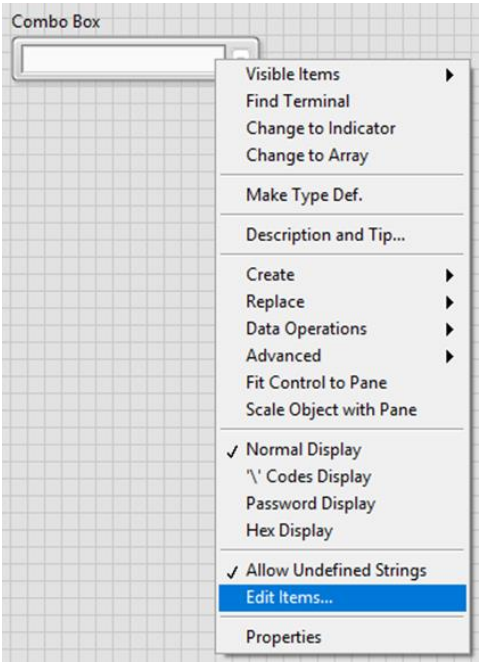


Figura 14. Opciones del *Combo Box*.

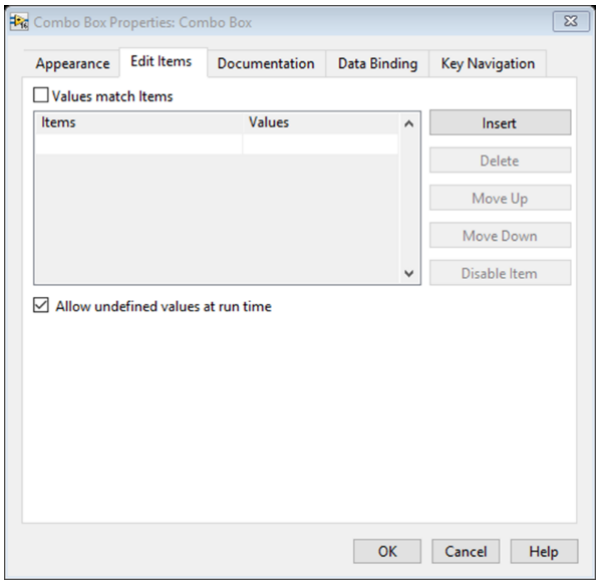


Figura 15. Combo Box vacío.

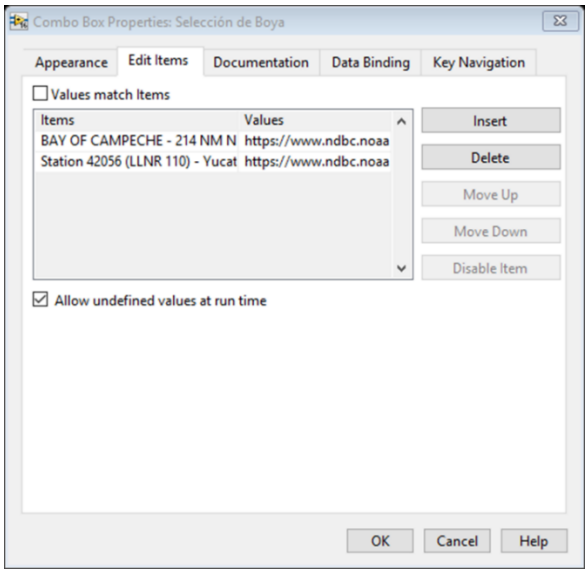


Figura 16. Combo Box con tres opciones.

Para que la gráfica muestre de forma adecuada la fecha y hora hacer clic derecho y entrar ya sea en *X Scale* o *Y Scale* → *Properties*. Una vez dentro ir a *Display Format* y seleccionar *Absolue Time*.

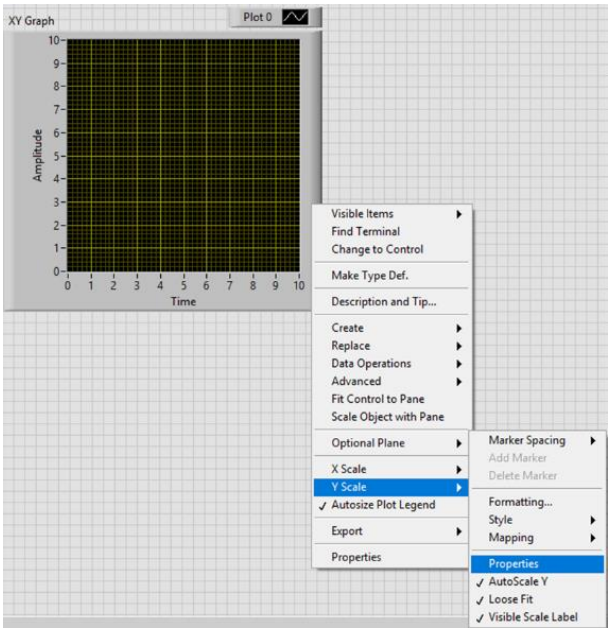


Figura 17. Opciones de la gráfica XY.

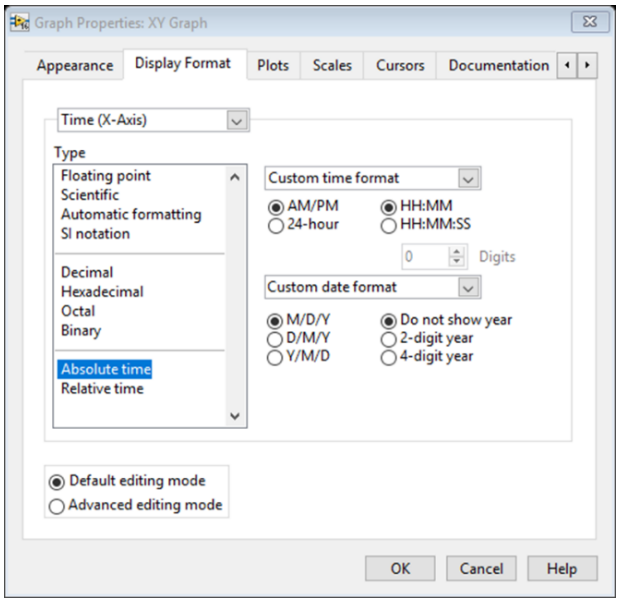


Figura 18. Opciones dentro de Display Format.

D. Diseño final

A continuación se muestra el diagrama de bloques completo.

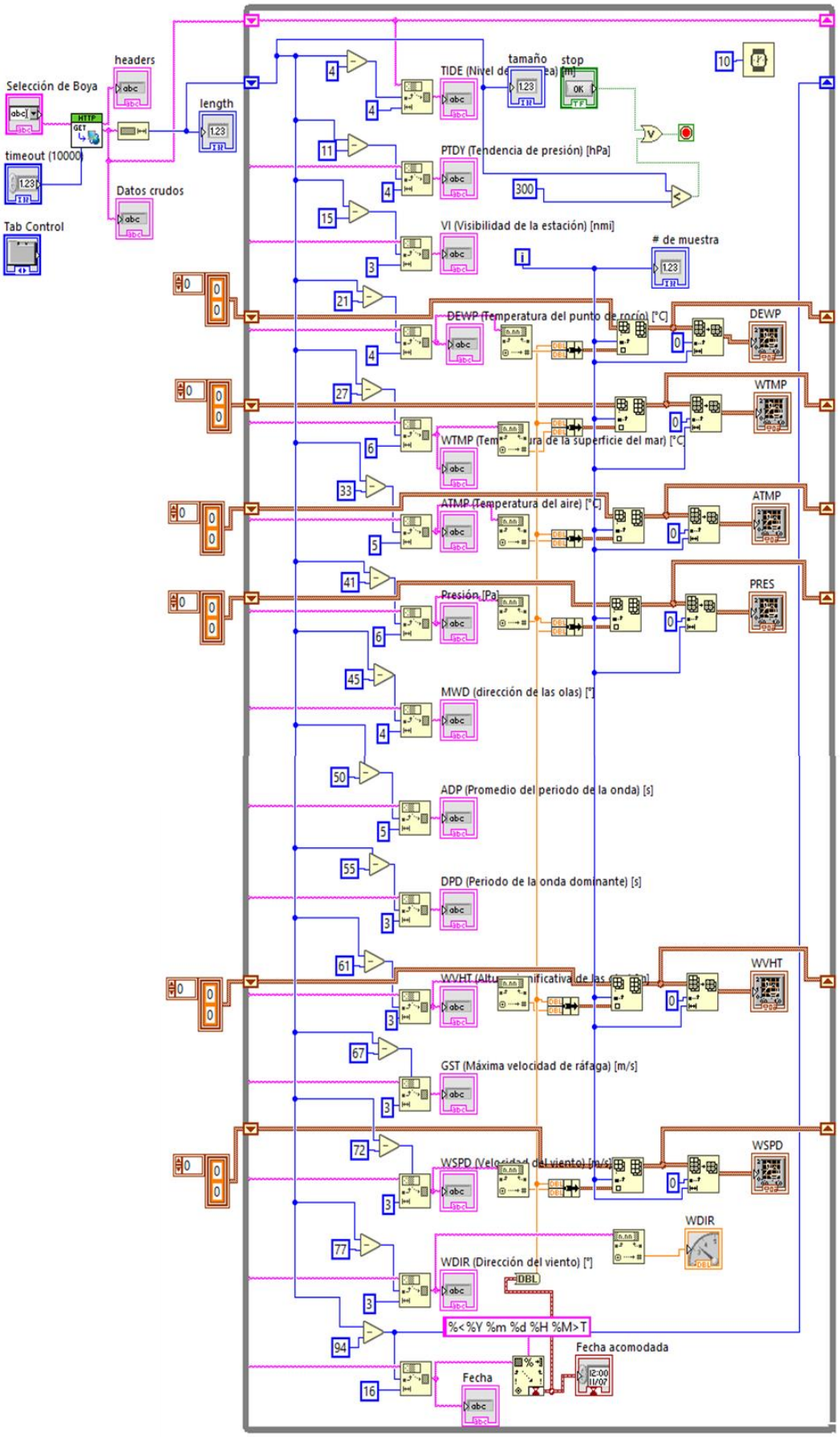


Figura 19. Diagrama de bloques completo.

Antes de correr el programa seleccionar la boya mareográfica en la primera pestaña del contenedor con pestañas y ajustar el tiempo de espera del bloque *GET*. En las siguientes pestañas se muestran los datos obtenidos de la página web. Las variables que son mostradas en gráficas XY son: WSPD, WVHT, PRES, ATMP, WTMP y DEWP; mientras que WDIR se muestra en un *Gauge* y las demás variables en indicadores sencillos.

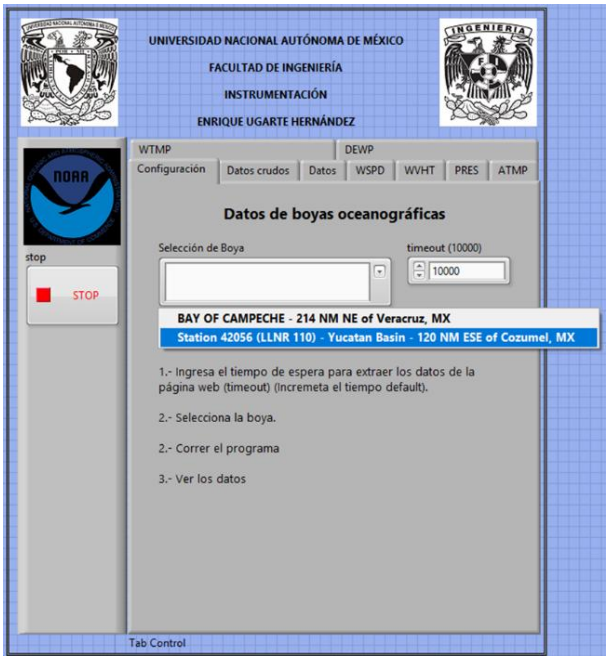


Figura 20. Selección de la boya antes de correr el programa.

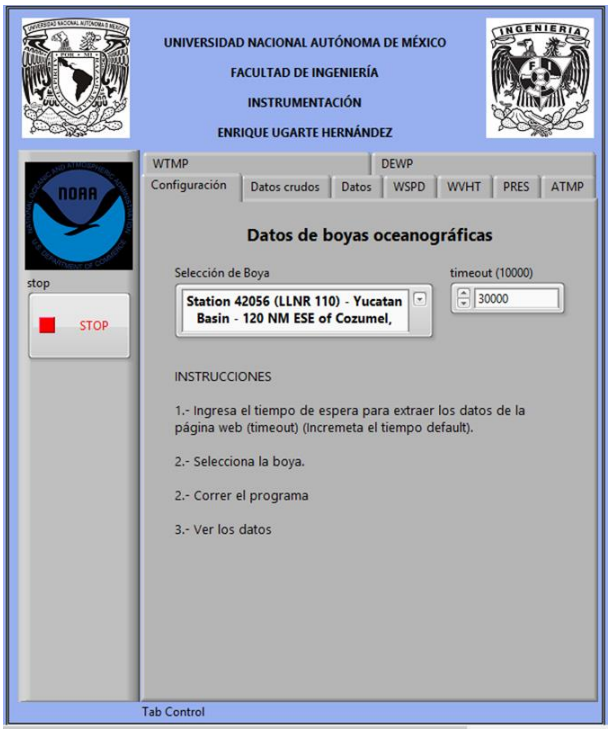


Figura 21. Pestaña de la selección de la boya. Nótese que el control para el tiempo de espera del bloque GET marca 30 segundos.

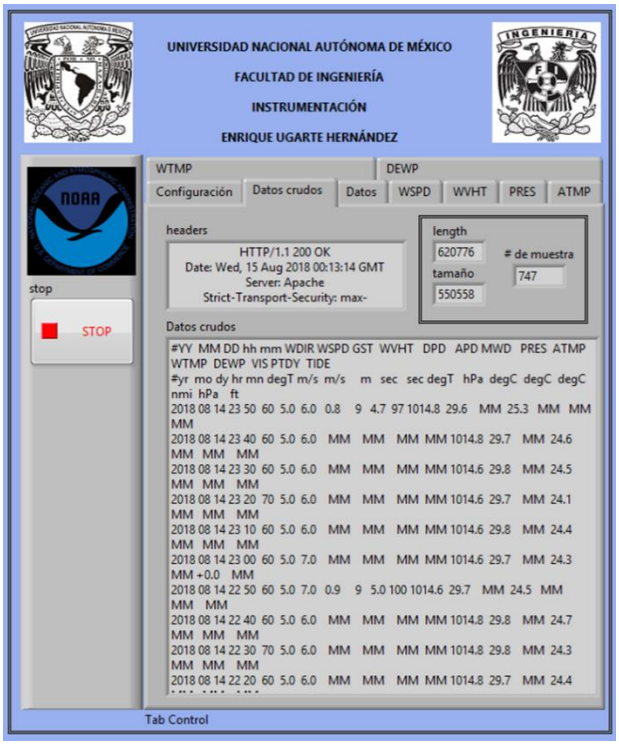


Figura 22. Pestaña con los datos crudos de la página web.

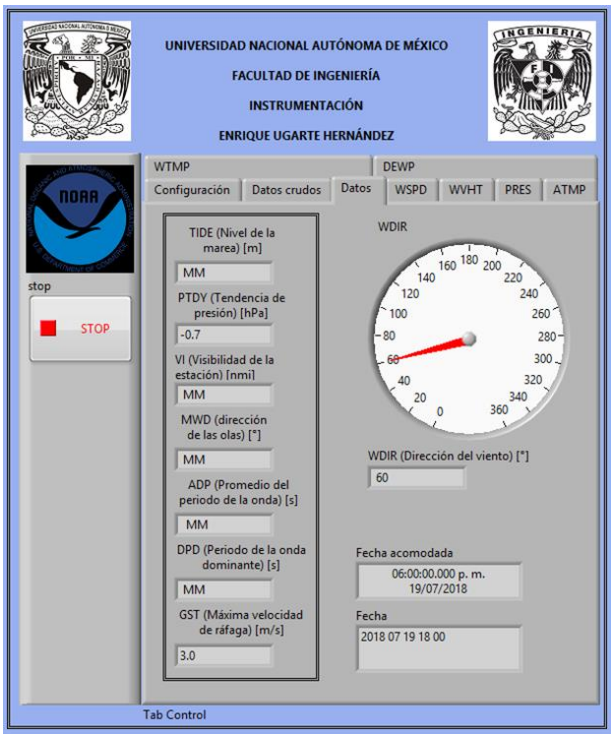


Figura 23. Indicadores de los datos que no se presentan en una gráfica XY. Nótese que se usó un *Gauge* para mostrar la dirección del viento.

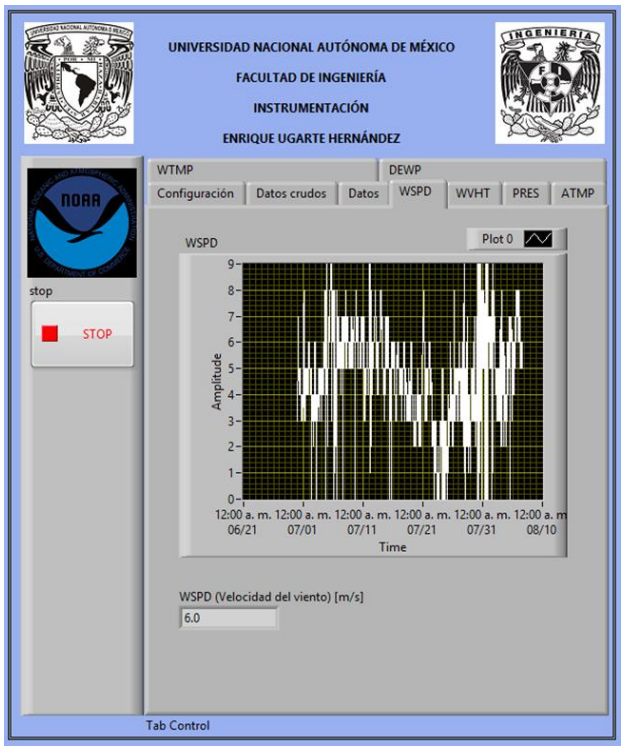


Figura 24. Gráfica WSPD (velocidad del viento).

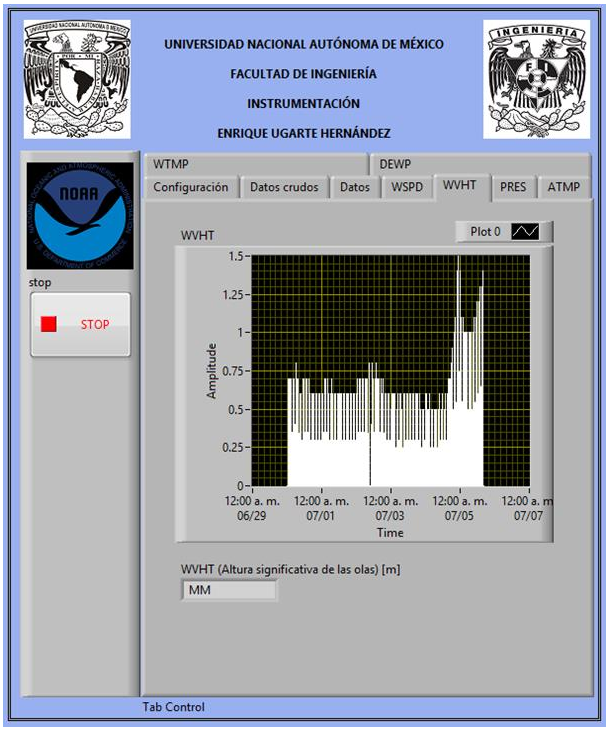


Figura 25. Gráfica WVHT (altura significativa de las olas).

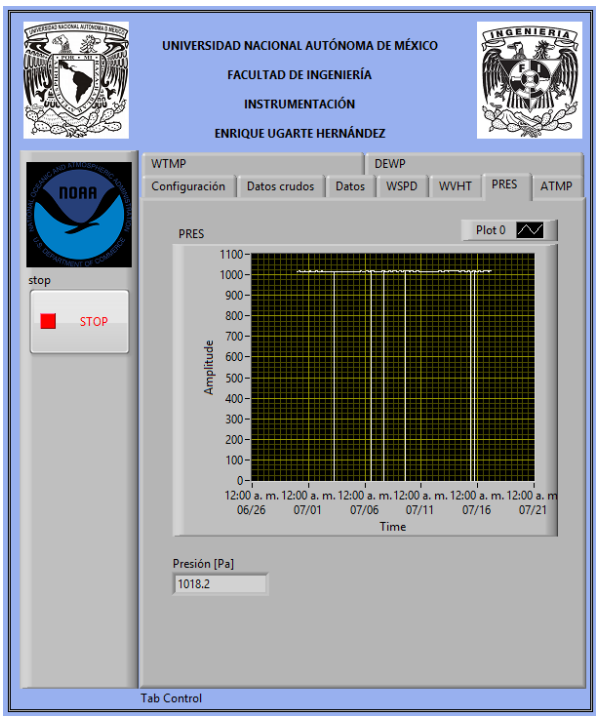


Figura 26. Gráfica PRES (presión).

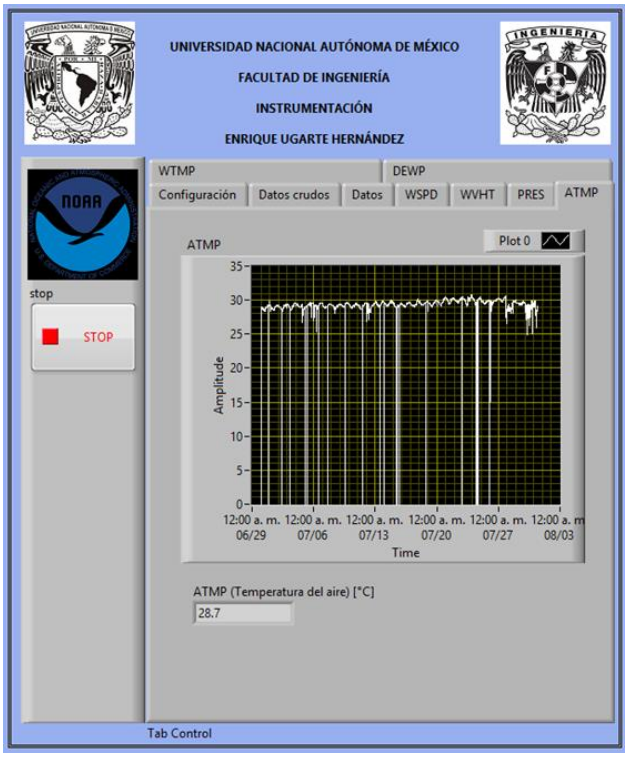


Figura 27. Gráfica ATMP (temperatura del aire).

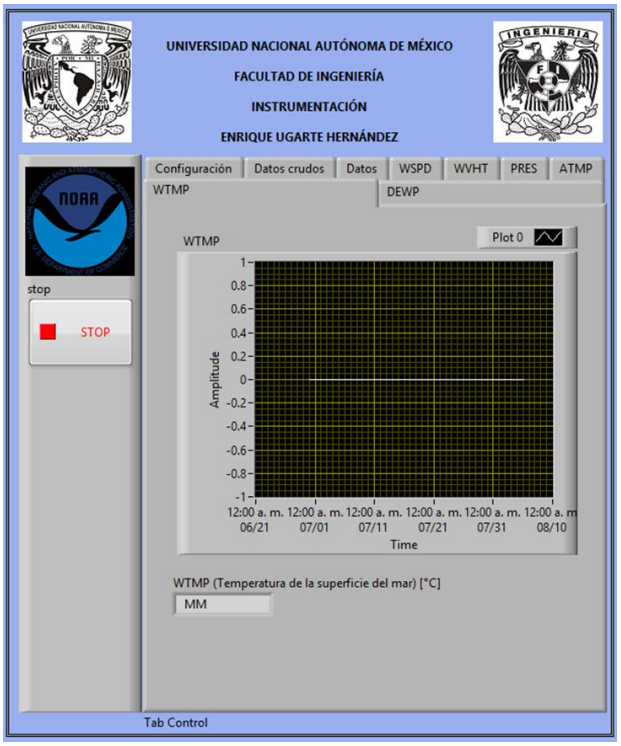


Figura 28. Gráfica WTMP (temperatura de la superficie del mar).

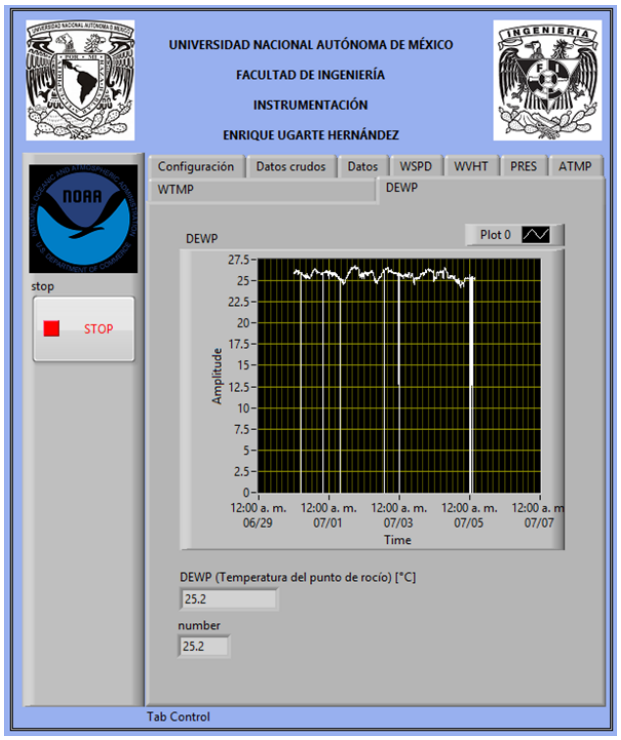


Figura 29. Gráfica DEWP (temperatura del punto de rocío).

V. FUENTES DE CONSULTA

[1] National Oceanic and Atmospheric Administration. (2018). National Data Buoy Center. *www.ndbc.noaa.gov*. Recuperado de https://www.ndbc.noaa.gov/ .

[2] EcuRed. (2018). Boya. *ecured@idict.cu*. Recuperado de https://www.ecured.cu/Boya .

[3] National Oceanic and Atmospheric Administration. (2018). Station 42055 (LLNR 1122) - BAY OF CAMPECHE - 214 NM NE of Veracruz, MX. *www.ndbc.noaa.gov*. Recuperado de https://www.ndbc.noaa.gov/station_page.php?station=42055 .

[4] NOAA's National Geophysical Data Center (NGDC). (2018). Deep-ocean Assessment and Reporting of Tsunamis (DART®) Description. *www.ndbc.noaa.gov*. Recuperado de https://www.ndbc.noaa.gov/dart/dart.shtml .

Práctica 18. Sistema de posicionamiento global (GPS)

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En esta práctica se presenta como utilizar la shield GPS con una tarjeta Arduino para adquirir los datos, interpretar el código NMEA y mostrar la posición geográfica en una interfaz gráfica en LabVIEW.**

I. INTRODUCCIÓN

1. Sistema de posicionamiento global

GPS significa sistema de posicionamiento global. Es un receptor de radio especial que mide la distancia desde su ubicación a los satélites que orbitan la tierra transmitiendo señales de radio. El GPS puede ubicar su posición en cualquier parte del mundo [1].

El GPS tiene tres componentes: el espacial (satélites), el de control (estaciones de tierra) y el de usuario.

a) Satélites

En el ámbito de los GPS se considera que un satélite es el segmento espacial. Una constelación de 24 satélites GPS (21 operacionales y 3 recambios) orbita alrededor de 12,000 millas sobre la Tierra. Los satélites se mueven aproximadamente a 7,000 millas por hora. Un satélite tarda aproximadamente 12 horas en orbitar completamente la Tierra, pasando sobre el mismo punto aproximadamente cada 24 horas. Los satélites están posicionados donde un receptor de GPS pueda recibir señales de al menos seis de los satélites en cualquier momento y cualquier lugar de la Tierra (si nada obstruye las señales). Un satélite tiene tres piezas clave de hardware [1, 2]:

Computadora: esta computadora a bordo controla su vuelo y otras funciones.

Reloj atómico: Mantiene el tiempo preciso dentro de tres nanosegundos (alrededor de tres mil millonésimas de segundo).

Transmisor de radio: Envía señales a la Tierra.

b) Estaciones de tierra

Las estaciones de tierra son el segmento de control del GPS. Cinco estaciones terrestres no tripuladas alrededor de la Tierra monitorean los satélites. La información de las estaciones se envía a una estación maestra de control: el Centro de Operaciones Espaciales Consolidado (CSOC) en la Base Aérea Schriever en Colorado, EEUU, donde los datos se procesan para determinar las efemérides y los errores de temporización de cada satélite. Una efeméride es una lista de las posiciones pronosticadas de cuerpos astronómicos como los planetas o la Luna. Las efemérides (el plural de efemérides) existen desde hace miles de años debido a su importancia en la navegación celestial. Las efemérides se compilan para rastrear las posiciones de los numerosos satélites que orbitan alrededor de la Tierra. Los datos procesados se envían a los satélites una vez al día con antenas de tierra ubicadas en todo el mundo. Esto es como sincronizar un asistente digital personal (PDA) con su computadora personal para asegurar que todos los datos estén sincronizados entre los dos dispositivos. Debido a que los satélites tienen pequeños propulsores incorporados, el CSOC puede controlarlos para garantizar que se mantengan en una órbita correcta [1].

c) Usuario

El componente del usuario incluye todos aquellos que usan un receptor GPS para recibir y convertir la señal GPS en posición, velocidad y tiempo. Incluye además todos los elementos necesarios en este proceso, como las antenas y el software de procesamiento [2].

2. GPS Adafruit

Es una shield GPS que funciona con Arduino UNO o Leonardo y está diseñado para registrar datos en una tarjeta micro SD. Tiene soporte de antena externa. Simplemente se conecta una antena de GPS activa externa a través de un cable uFL / SMA al blindaje y el módulo cambiará automáticamente para usar la antena. Sus características son [3]:

- -165 dBm de sensibilidad, 10 Hz de actualización, 66 canales.
- Módulo de baja potencia: solo consumo de corriente de 20 mA, la mitad de la mayoría de los GPS.
- Shield para Arduino Uno / Duemilanove / Diecimila / Leonardo.
- Ranura para tarjeta microSD para el registro de datos en una tarjeta extraíble.
- Incluye batería RTC (*The Real Time Clock*), hasta 7 años de respaldo. Una batería RTC es una batería de litio usada en muchos dispositivos.
- Registrador de datos incorporado.
- Salida de PPS (*pulse per second*) en el arreglo. PPS significa que es una señal con una duración menor a un segundo.
- Antena de parche interno + conector u.FL para antena activa externa.
- Potencia, pin # 13 y LED de estado de reparación.

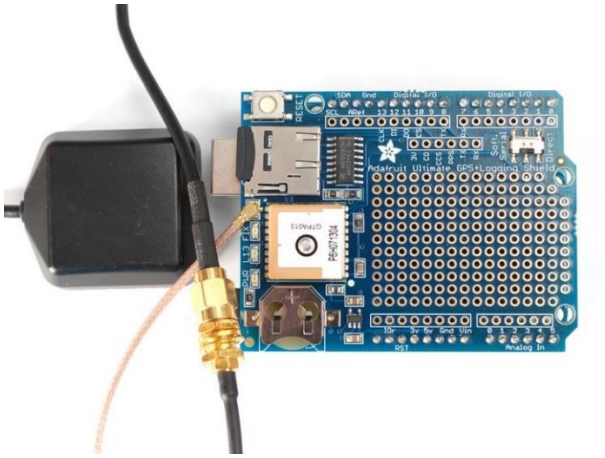


Figura 1. Adafruit Ultimate GPS Logger Shield con todos sus componentes.

3. Código NMEA

NMEA es la abreviatura de *National Marine Electronics Association*. Es una asociación fundada en 1957 por un grupo de fabricantes de electrónica para obtener un sistema común de comunicación entre las diferentes marcas de electrónica naval. Poco a poco se fueron sumando todos los fabricantes a este estándar, además de organizaciones oficiales y gubernamentales.

NMEA se creó para el intercambio de información digital entre productos electrónicos marinos. El primer protocolo estándar se llamó NMEA 0183, y es el que todavía se utiliza y aceptan la mayoría de los equipos electrónicos, es un protocolo que define los requerimientos de datos y tiempo de transmisión en el formato serie a una velocidad de 4800 baudios (bits por segundo). Define también la norma que cada equipo sea emisor de NMEA y pueda ser escuchado por muchos receptores.

Hay muchas cadenas de datos en el estándar NMEA para todo tipo de dispositivos que pueden usarse en un entorno marino. Algunos de los que tienen aplicabilidad a los receptores GPS se enumeran a continuación: (todos los mensajes comienzan con $GP) [4].

$GPBOD Teniendo, origen a destino
$GPBWC Rumbo y distancia al punto de ruta, gran círculo
$GPGGA Datos fijos del sistema de posicionamento global
$GPGLL Posición geográfica, latitude/longitud
$GPGSA GPS DOP y satélites activos
$GPGSV GPS Satélites a la vista
$GPHDT Encabezado, verdadero
$GPR00 Lista de puntos en la actual ruta activa
$GPRMA Datos de Loran-C específicos mínimos recomendados
$GPRMB Infotmación de navegación mínima recomendada
$GPRMC Datos mínimos específicos de GPS / tránsito recomendados
$GPRTE Rutas
$GPTRF Datos fijos de transito
$GPSTN ID de datos multiples
$GPVBW Doble tierra/ velocidad del agua
$GPVTG Buen camino y velocidad de tierra
$GPWPL Ubicación del punto de camino
$GPXTE Error de vía cruzada, medido
$GPZDA Fecha y hora

Las líneas NMEA más importantes incluyen el GGA que proporciona los datos de Posición actuales, el RMC que proporciona la información de oraciones GPS mínimas y el GSA que proporciona los datos de estado de Satélite.

4. Interpretación de la cadena &GPGGA

```
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,
545.4,M,46.9,M,,*47

Where:
     GGA      Global Positioning System Fix
Data
     123519       Fix taken at 12:35:19 UTC
     4807.038,N  Latitude 48 deg 07.038' N
     01131.000,E Longitude 11 deg 31.000' E
     1 Fix quality: 0 = invalid
                    1 = GPS fix (SPS)
                    2 = DGPS fix
                    3 = PPS fix
                    4 = Real Time Kinematic
                    5 = Float RTK
    6 = estimated (dead reckoning) (2.3
feature)
    7 = Manual input mode
    8 = Simulation mode

08         Number of satellites being tracked
0.9        Horizontal dilution of position
545.4,M    Altitude, Meters, above mean sea
level
46.9,M     Height of geoid (mean sea level)
above
.          WGS84     ellipsoid
(empty field) time in seconds since last DGPS
update
(empty field) DGPS station ID number
*47    The checksum data, always begins with *
```

Como se observa en la interpretación anterior de la cadena $GPGGA los primeros caracteres hacen referencia a:

123519: 12 horas, 35 minutos y 19 segundos.
Latitud: 48° 07.038´ Norte.
Longitud: 11° 31.000´ Este .
Indicador de posición (Fix): 1.
Número de satélites usados: 8.
HDOP (Precisión de disolución horizontal): 0.9.
Altitud con respecto al nivel del mar (MSL): 545.4 metros.
Separación geoidal: 46.9 metros
Check sum: *47.

II. OBJETIVOS

- ✓ Estudiar el código NMEA para interpretar los datos que entrega un GPS.
- ✓ Realizar una interfaz gráfica de usuario para mostrar en un panel frontal algunos de los datos que entrega un GPS, principalmente fecha, hora, latitud, longitud, etcétera.

III. MATERIALES

- Tarjeta Arduino UNO.
- 1 Adafruit Ultimate GPS Logger Shield.
- Antena para Adafruit Ultimate GPS Logger Shield (opcional).
- Equipo de cómputo con el software IDE de Arduino y LabVIEW 2012 o superior.

IV. DESARROLLO

A. Conexión

Montar la shield GPS sobre el Arduino. En caso de contar con la antena, conectarla al dispositivo.

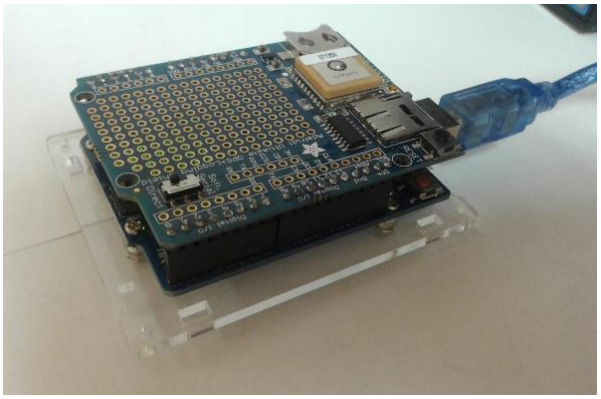


Figura 2. GPS montado sobre Arduino.

En caso de encontrarse en un lugar cerrado es probable que el GPS no logre recibir la información completa de los satélites; en estos casos es muy útil la antena, pues mejora la recepción de datos.

B. Datos a través del IDE de Arduino

Antes de abrir el IDE de Arduino posicionar el Selector en *Direct*.

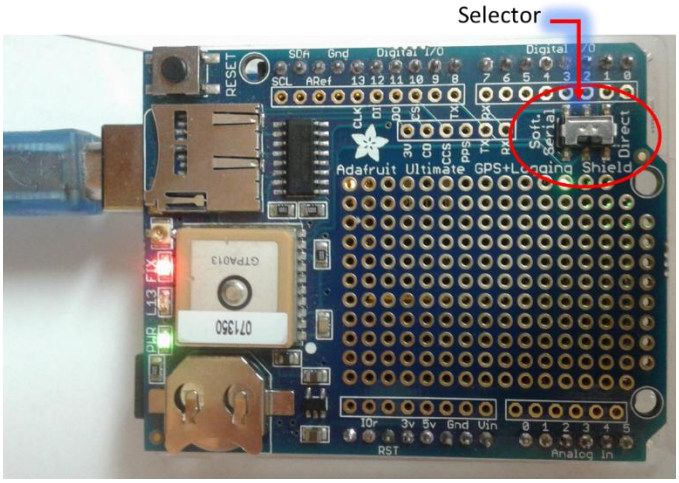


Figura 3. Ubicación del selector.

Cargar un programa vacío de Arduino para obtener los datos directos del GPS y abrir el monitor serie con el baud rate en 9600.

```
void setup() {
}
void loop() {
}
```

Nota: Este GPS, a diferencia de otros dispositivos que usan el formato NMEA, manda datos con una tasa de 9600 baudios.

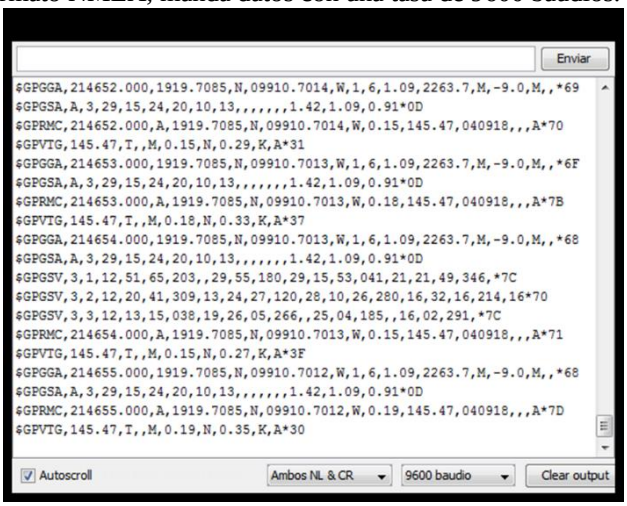


Figura 4. Monitor serie del IDE de Arduino.

C. Adquisición de datos

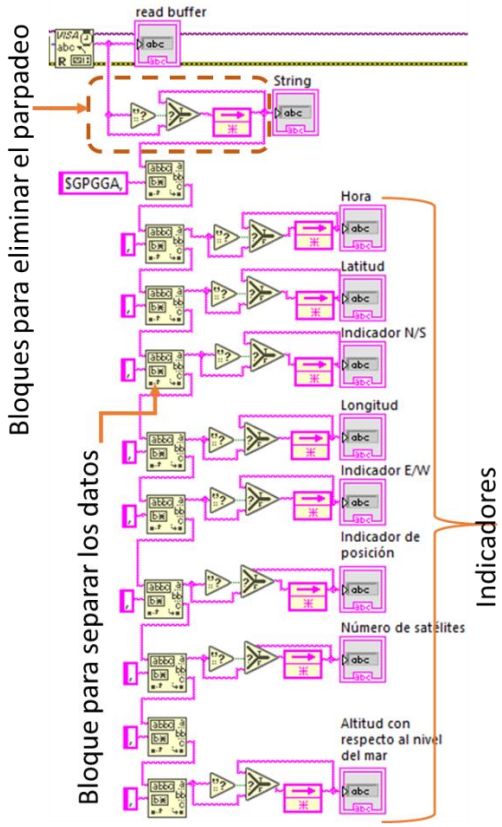


Figura 6. Separación de los datos que entrega el GPS y bloques para eliminar el parpadeo.

El programa de LabVIEW recibe los datos del Arduino a través de comunicación serie (ver "Introducción al entorno de desarrollo de LabVIEW). Los GPS entregan varios datos separados por una coma, por esta razón se ocupa el bloque Match Pattern, para ordenar la

información. Cabe mencionar que la hora que se obtiene del GPS es la hora UTC (*Coordinated Universal Time*), la cual corresponde a la hora del meridiano de Greenwich.

Finalmente se agregan los bloques para guardar los datos dentro de un documento de texto (ver "Introducción al entorno de desarrollo de LabVIEW").

D. Diseño final

A continuación se muestra el diagrama de bloques completo.

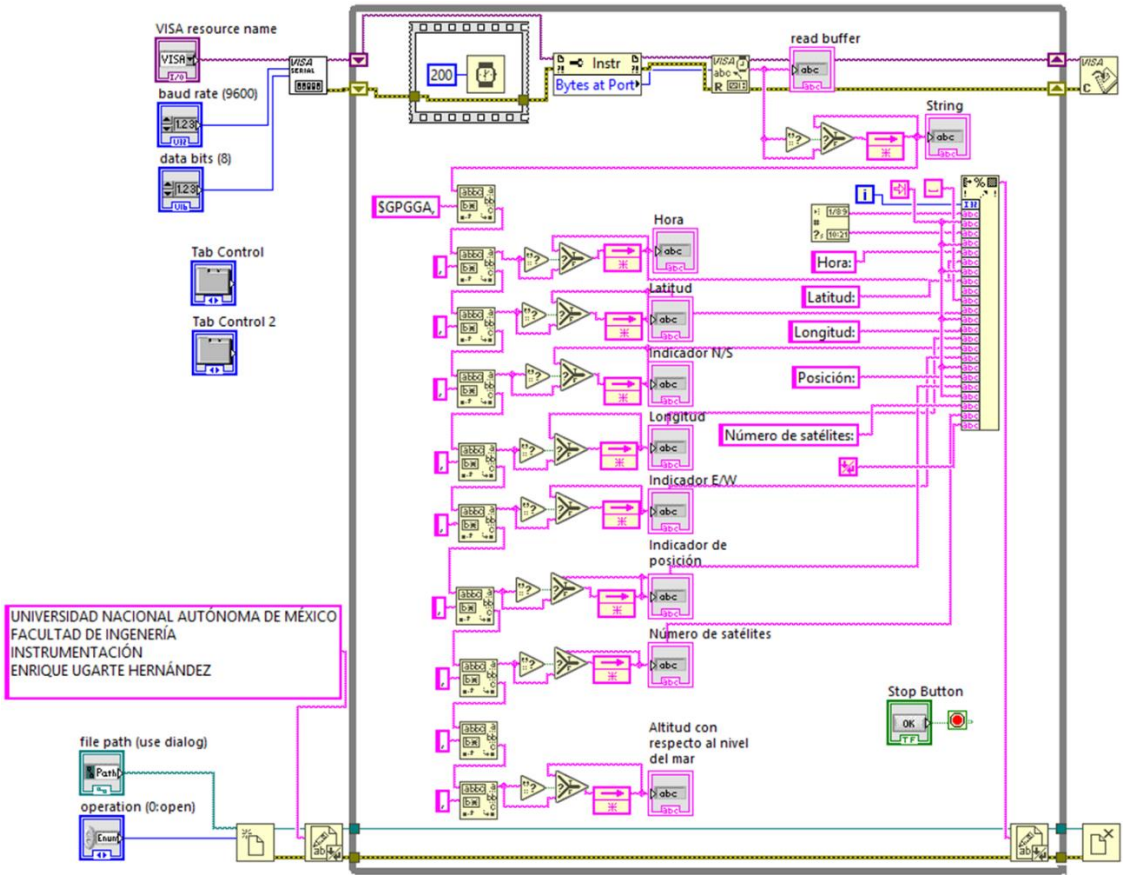


Figura 6. Diagrama de bloques completo.

En el panel frontal todos los controles e indicadores se distribuyen dentro de un contenedor con pestañas, donde los componentes relacionados con la comunicación serie se encuentran en la primera pestaña; en la segunda pestaña hay un segundo contenedor, el cual muestra los datos crudos y ordenados dentro de indicadores y finalmente los elementos para guardar los datos en un documento de texto en la tercera pestaña. Antes de correr el programa es necesario crear un documento de texto (bloc de notas) y guardarlo con algún nombre en alguna ruta, misma que será ingresada en el *file path* (tercera pestaña).

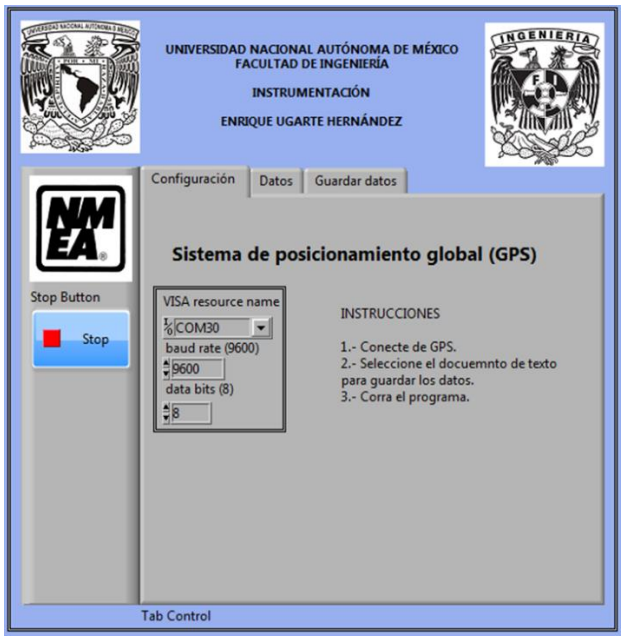


Figura 7. Primera página del contenedor con pestañas principal.

Figura 8. Datos crudos del GPS. Segunda página del contenedor con pestañas principal.

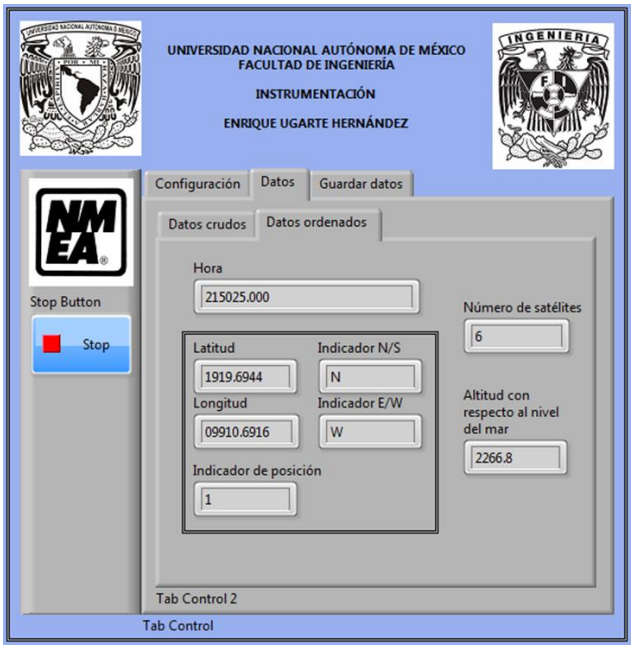


Figura 9. Datos ordenados. Segunda página del contenedor con pestañas principal.

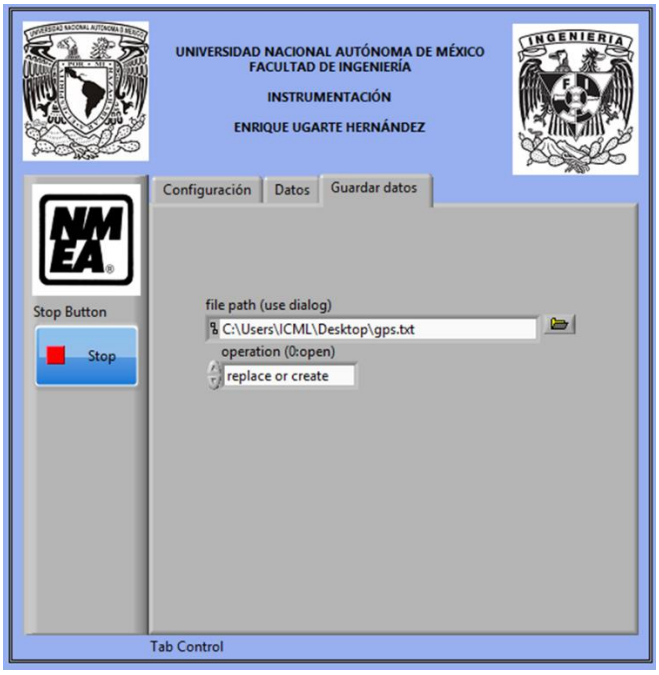


Figura 10. Tercera página del contenedor con pestañas principal.

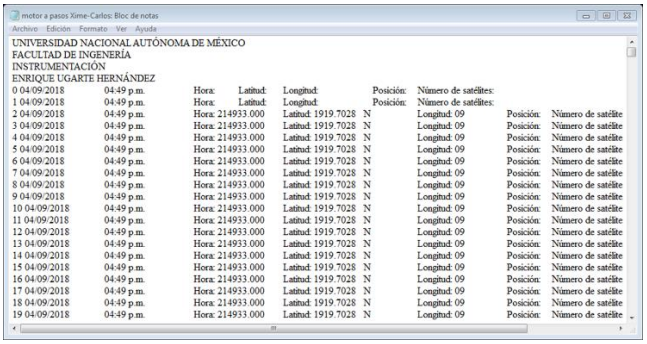


Figura 11. Datos en un documento de texto.

V. FUENTES DE CONSULTA

[1] McNamara, Joel, (2004), *GPS FOR DUMMIES*, Indianapolis, Indiana, Wiley Publishing.

[2] El Equipo de Atlantis IT. (2016). Características del posicionamiento global gps. *www.atlantis-technology.com*. Recuperado de https://www.atlantis-technology.com/caracteristicas-del-posicionamiento-global-gps/

[3] Adafruit ®. (2016). Adafruit Ultimate GPS Logger Shield. *learn.adafruit.com*. Recuperado de https://learn.adafruit.com/adafruit-ultimate-gps-logger-shield?view=all

[4] Mehaffey, Joe y Yeaze, Jack. (2013). NMEA data. *www.gpsinformation.org*. Recuperado de https://www.gpsinformation.org/dale/nmea.htm .

Práctica 19. Cámara OV528 y lector micro SD

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En este trabajo se explica cómo utilizar la cámara OV528 para tomar una foto con un botón y posteriormente guardar la imagen en una tarjeta micro SD con ayuda de un lector de tarjetas micro SD y una tarjeta Arduino UNO.**

I. INTRODUCCIÓN

A. Cámara OV528

El sistema de cámara de bus serie del modelo OV528 funciona como una cámara de vídeo o una cámara fotográfica compacta JPEG comprimida y se puede conectar a un servidor de conectividad inalámbrica o PDA.

Cuando se realiza como una cámara de video, el panel TFT-LCD del servidor (dispositivo donde se pueda mostrar la imagen y guardarla) funciona como un visor. Los usuarios pueden enviar un comando de instantánea desde el servidor para capturar una imagen fija de una sola imagen de resolución completa. La imagen es entonces comprimida por el motor JPEG y transferida al servidor [1].

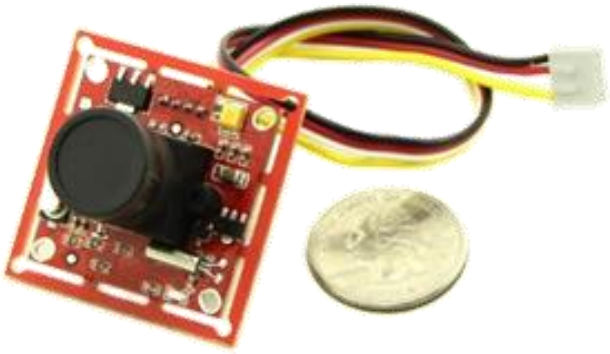


Figura 1. Cámara OV528.

B. Sensores de la cámara

La cámara OV528 admite OmniVision OV76x0 / y OV66x0 Camera Chips, son dos sensores son los que se necesitan para poder operar este modelo de cámara. Requiere de una memoria de programa para que el microcontrolador responda correctamente a los comandos del servidor, así como para almacenar todos los parámetros necesarios para ajustar las cualidades de imagen / compresión. También se necesita de una memoria de programa de tipo serie para OV528 y su contenido de la memoria del programa se actualiza sobre la marcha.

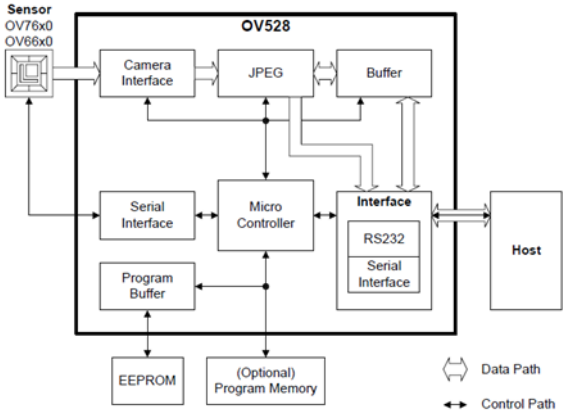


Figura 2. Diagrama de bloques de la cámara OV528 [2].

C. El puerto serie OV528

La cámara OV528 Serial Bridge es un chip controlador que puede transferir datos de imagen de Camera Chips a servidores inalámbricos / PDA. La cámara OV528 toma datos de vídeo progresivos YCbCr 422 de 8 bits de un OV76x0 / OV66x0 Camera Chip.

La interfaz de la cámara se sincroniza con los datos de video de entrada y realiza funciones de muestreo, fijación y ventanas con la resolución deseada, así como la conversión de color que solicita el usuario a través de comandos del servidor de bus serie. El CODEC JPEG con ajustes de calidad variable puede lograr una mayor relación de compresión y una mejor calidad de imagen para diversas resoluciones de imagen. El bus serie de control de cámara se utiliza para lograr una mayor flexibilidad en la interfaz de la cámara.

D. Funcionamiento

Una subcategoría de los sensores ópticos son los sensores de visión. Los sensores de visión también llamados sensores inteligentes, término que se les da porque necesitan de una gran variedad de sensores que se encargan de procesar la información (imágenes o video), usualmente utilizan protocolos de comunicación como I2C, SPI, Serial o USB, dependiendo finamente del modelo de sensor. Los sensores de visión se encargan de obtener una imagen como resultado final, estas imágenes están conformadas por celdas unitarias denominadas pixeles.

Existen dos tipos de tecnologías que utilizan los sensores de visión: la tecnología CMOS (del inglés *complementary metal – oxide- semiconductor*) y la CDD (*charge coupled device*). La

más utilizada hoy en día se basa en la tecnología CMOS debido a bajo coste de fabricación, estandarización con otros sensores, manejo sencillo de señales analógicas-digitales y menor consumo de potencia. Los sensores que utilizan CMOS, poseen elementos fotosensibles que puede configurarse, de maneras distintas, utilizando fototransistores-fotodiodos o fotocompuestas. [1].

La tecnología CMOS funciona mediante las siguientes tres fases de operación:

1) Conversión de una imagen – objeto en un conjunto de cargas situadas en elementos fotosensibles (pixeles).
2) Transferencia de cargas.
3) Conversión de la carga en una tensión por celdas unitarias denominadas pixeles.

1. Fase 1: Conversión de una imagen - objeto en un conjunto de cargas situadas en elementos fotosensibles (pixeles)

En esta fase consiste en la conversión de la imagen de un objeto. Los rayos de luz pasan a través de una lente y la imagen se detecta en una matriz de elementos fotosensibles; posteriores a esto, la imagen es convertida en lo que será la imagen resultante.

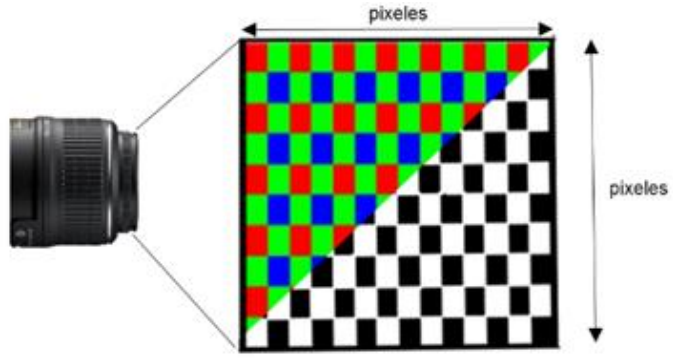


Figura 3. Imagen mitad color mitad escala de grises. Cada recuadro representa un pixel.

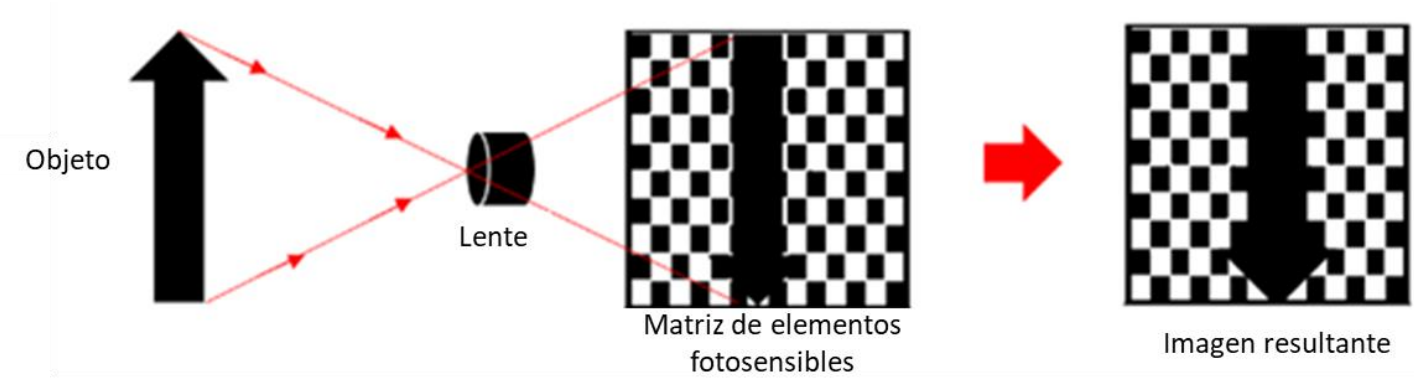


Figura 4. Proceso de conversión de una imagen – objeto.

2. Fase 2: Transferencia de cargas

Esta fase consiste en la conversión de la luz en una señal generada a través de la transferencia de cargas, dependiendo del fotosensor capacitivo serán los números de impulsos efectuados.

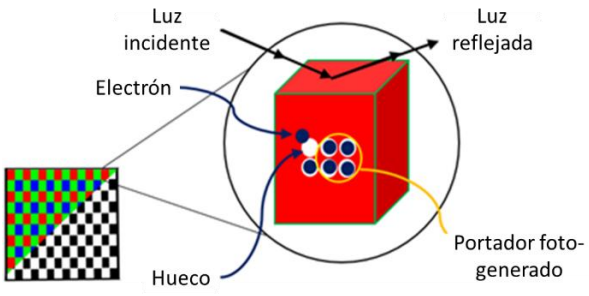


Figura 5. Transferencia de cargas.

3. Fase 3: Conversión de la carga en una tensión

El objetivo principal consiste en convertir la luz en una señal eléctrica. Cada pixel está conformado por un elemento fotosensible, por un fototransistor y amplificadores operacionales; todos bajo la tecnología CMOS, encargados de acondicionar la lectura de señal encargada de leer el pixel tanto vertical como horizontalmente [3].

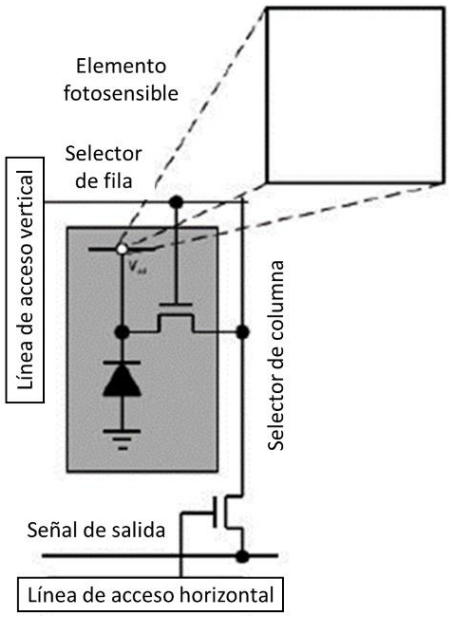


Figura 6. Conversión de una carga a tensión [3].

II. OBJETIVOS

- Almacenar una imagen en una memoria micro SD por medio de un sensor de imagen (Grove - Serial Camera OV528) utilizando un código de demostración.

III. MATERIALES

- Grove - Serial Camera OV528.
- Arduino UNO.
- Cables Dupont macho-hembra y alambre para protoboard.
- Lector de tarjetas micro SD.
- 1 Push button.
- Computadora con el software IDE de Arduino.

IV. DESARROLLO

1.- Conectar la Serial Camera OV528, el lector de tarjetas SD y el push button al Arduino UNO, como lo indica la figura 7.

Arduino UNO	Camera OV528
5 V	VDD
GND	GND
RX	TX
TX	RX

Arduino UNO	Lector de memoria micro SD
5 V	VDD
GND	GND
D12	MISO
D11	MOSI
D13	SCK
D9	CS

Arduino UNO	Push Button
GND	Terminal 1
D4	Terminal 2

Figura 7. Tabla de conexiones.

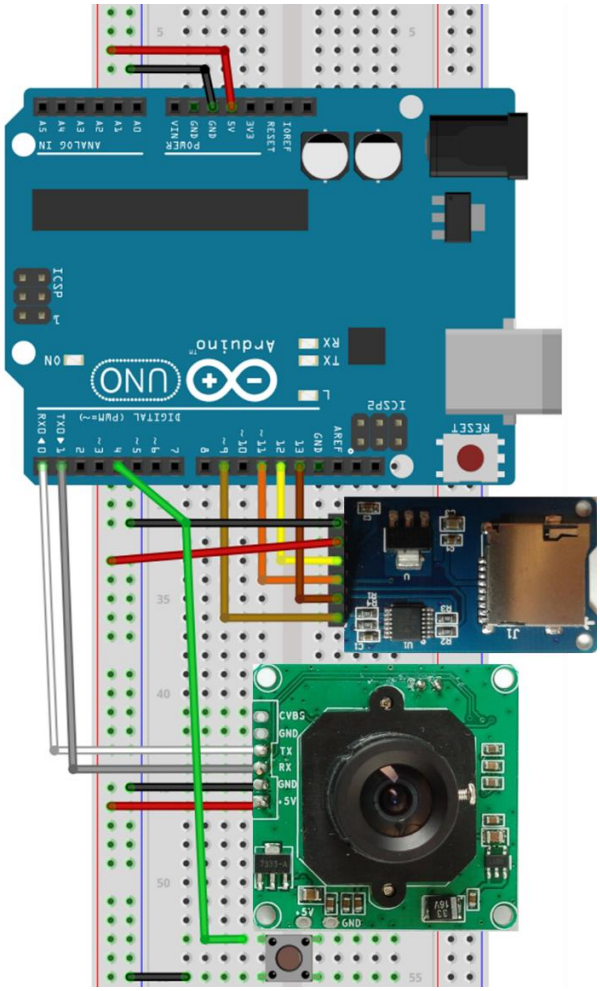


Figura 8. Conexiones.

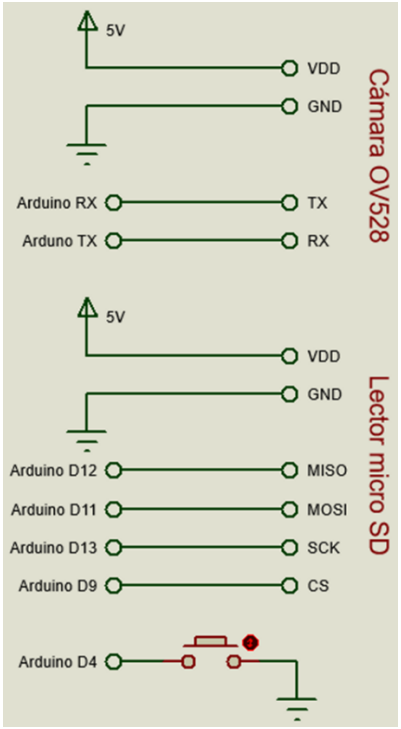


Figura 9. Esquema de conexiones.

2.- Analizar e implementar el siguiente código ejemplo que proporciona el fabricante del Grove - Serial Camera OV528 en la interfaz del Arduino (IDE).

```
//  File SerialCamera_DemoCode_CJ-OV528.ino
//  8/8/2013 Jack Shao
//  Demo code for using seeeduino or Arduino board to cature
jpg format
//  picture from seeed serial camera and save it into sd card.
Push the
//  button to take the a picture .
//  For more details about the product please check
http://www.seeedstudio.com/depot/

#include <SPI.h>
#include <arduino.h>
#include <SD.h>

#define PIC_PKT_LEN    128
//data length of each read, dont set this too big because ram is
limited
#define PIC_FMT_VGA    7
#define PIC_FMT_CIF    5
#define PIC_FMT_OCIF   3
#define CAM_ADDR       0
#define CAM_SERIAL     Serial

#define PIC_FMT        PIC_FMT_VGA

File myFile;

const byte cameraAddr = (CAM_ADDR << 5);  // addr
const int buttonPin = 4;                  // the number of the
pushbutton pin
unsigned long picTotalLen = 0;            // picture length
int picNameNum = 0;

void setup()
{
   Serial.begin(115200);
   pinMode(buttonPin, INPUT);    // initialize the pushbutton
pin as an input
   Serial.println("Initializing SD card....");
   pinMode(9,OUTPUT);         // CS pin of SD Card Shield

   if (!SD.begin(9))
   {
      Serial.print("sd init failed");
      return;
   }
   Serial.println("sd init done.");
   initialize();
}
void loop()
{
   int n = 0;
   while(1){
      Serial.println("\r\nPress the button to take a picture");
      while (digitalRead(buttonPin) == LOW);      //wait for
buttonPin status to HIGH
      if(digitalRead(buttonPin) == HIGH){
         delay(20);                          //Debounce
         if (digitalRead(buttonPin) == HIGH)
         {
            Serial.println("\r\nbegin to take picture");
            delay(200);
            if (n == 0) preCapture();
            Capture();
            GetData();
         }
         Serial.print("\r\nTaking pictures success ,number : ");
         Serial.println(n);
         n++ ;
      }
   }
}
void clearRxBuf()
{
   while (Serial.available())
   {
      Serial.read();
   }
}
void sendCmd(char cmd[], int cmd_len)
{
   for (char i = 0; i < cmd_len; i++) Serial.print(cmd[i]);
}
void initialize()
{
   char cmd[] =
{0xaa,0x0d|cameraAddr,0x00,0x00,0x00,0x00} ;
   unsigned char resp[6];

   Serial.setTimeout(500);
   while (1)
   {
      //clearRxBuf();
      sendCmd(cmd,6);
      if (Serial.readBytes((char *)resp, 6) != 6)
      {
         continue;
      }
      if (resp[0] == 0xaa && resp[1] == (0x0e | cameraAddr)
&& resp[2] == 0x0d && resp[4] == 0 && resp[5] == 0)
      {
         if (Serial.readBytes((char *)resp, 6) != 6) continue;
         if (resp[0] == 0xaa && resp[1] == (0x0d |
cameraAddr) && resp[2] == 0 && resp[3] == 0 && resp[4]
== 0 && resp[5] == 0) break;
      }
   }
   cmd[1] = 0x0e | cameraAddr;
   cmd[2] = 0x0d;
   sendCmd(cmd, 6);
   Serial.println("\nCamera initialization done.");
}
```

```
void preCapture()
{
  char cmd[] = { 0xaa, 0x01 | cameraAddr, 0x00, 0x07, 0x00,
PIC_FMT };
  unsigned char resp[6];

  Serial.setTimeout(100);
  while (1)
  {
    clearRxBuf();
    sendCmd(cmd, 6);
    if (Serial.readBytes((char *)resp, 6) != 6) continue;
    if (resp[0] == 0xaa && resp[1] == (0x0e | cameraAddr)
&& resp[2] == 0x01 && resp[4] == 0 && resp[5] == 0)
break;
  }
}
void Capture()
{
  char cmd[] = { 0xaa, 0x06 | cameraAddr, 0x08,
PIC_PKT_LEN & 0xff, (PIC_PKT_LEN>>8) & 0xff ,0};
  unsigned char resp[6];

  Serial.setTimeout(100);
  while (1)
  {
    clearRxBuf();
    sendCmd(cmd, 6);
    if (Serial.readBytes((char *)resp, 6) != 6) continue;
    if (resp[0] == 0xaa && resp[1] == (0x0e | cameraAddr)
&& resp[2] == 0x06 && resp[4] == 0 && resp[5] == 0)
break;
  }
  cmd[1] = 0x05 | cameraAddr;
  cmd[2] = 0;
  cmd[3] = 0;
  cmd[4] = 0;
  cmd[5] = 0;
  while (1)
  {
    clearRxBuf();
    sendCmd(cmd, 6);
    if (Serial.readBytes((char *)resp, 6) != 6) continue;
    if (resp[0] == 0xaa && resp[1] == (0x0e | cameraAddr)
&& resp[2] == 0x05 && resp[4] == 0 && resp[5] == 0)
break;
  }
  cmd[1] = 0x04 | cameraAddr;
  cmd[2] = 0x1;
  while (1)
  {
    clearRxBuf();
    sendCmd(cmd, 6);
    if (Serial.readBytes((char *)resp, 6) != 6) continue;
    if (resp[0] == 0xaa && resp[1] == (0x0e | cameraAddr)
&& resp[2] == 0x04 && resp[4] == 0 && resp[5] == 0)
    {
      Serial.setTimeout(1000);
      if (Serial.readBytes((char *)resp, 6) != 6)
      {
        continue;
      }
      if (resp[0] == 0xaa && resp[1] == (0x0a |
cameraAddr) && resp[2] == 0x01)
      {
        picTotalLen = (resp[3]) | (resp[4] << 8) | (resp[5] <<
16);
        Serial.print("picTotalLen:");
        Serial.println(picTotalLen);
        break;
      }
    }
  }

}
void GetData()
{
  unsigned int pktCnt = (picTotalLen) / (PIC_PKT_LEN - 6);
  if ((picTotalLen % (PIC_PKT_LEN-6)) != 0) pktCnt += 1;

  char cmd[] = { 0xaa, 0x0e | cameraAddr, 0x00, 0x00, 0x00,
0x00 };
  unsigned char pkt[PIC_PKT_LEN];

  char picName[] = "pic00.jpg";
  picName[3] = picNameNum/10 + '0';
  picName[4] = picNameNum%10 + '0';

  if (SD.exists(picName))
  {
    SD.remove(picName);
  }

  myFile = SD.open(picName, FILE_WRITE);
  if(!myFile){
    Serial.println("myFile open fail...");
  }
  else{
    Serial.setTimeout(1000);
    for (unsigned int i = 0; i < pktCnt; i++)
    {
      cmd[4] = i & 0xff;
      cmd[5] = (i >> 8) & 0xff;

      int retry_cnt = 0;
      retry:
      delay(10);
      clearRxBuf();
      sendCmd(cmd, 6);
      uint16_t cnt = Serial.readBytes((char *)pkt,
PIC_PKT_LEN);

      unsigned char sum = 0;
      for (int y = 0; y < cnt - 2; y++)
      {
        sum += pkt[y];
```

```
            }
            if (sum != pkt[cnt-2])
            {
               if (++retry_cnt < 100) goto retry;
               else break;
            }

            myFile.write((const uint8_t *)&pkt[4], cnt-6);
            //if (cnt != PIC_PKT_LEN) break;
         }
         cmd[4] = 0xf0;
         cmd[5] = 0xf0;
         sendCmd(cmd, 6);
      }
      myFile.close();
      picNameNum ++;
}
```

3.- Probar el funcionamiento del programa, así como también, el funcionamiento del sensor.
Al presionar el botón, se capturan las imágenes y se almacenan en el lector de memoria micro SD.

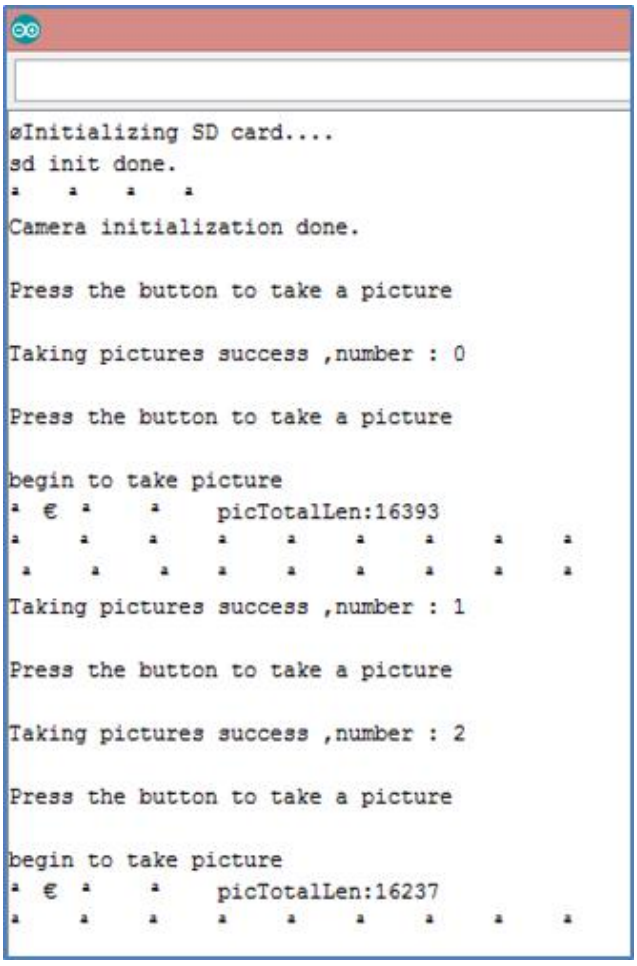


Figura 10. Captura del monitor serie.

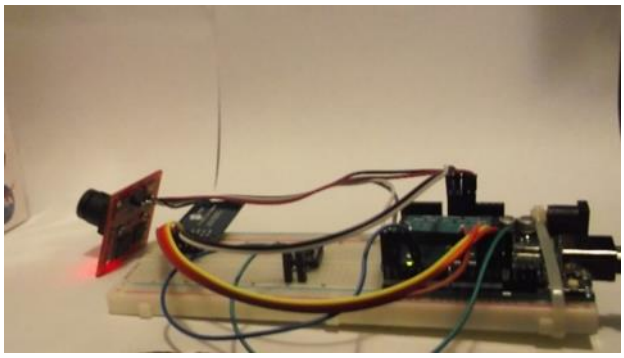


Figura 11. Montaje final.

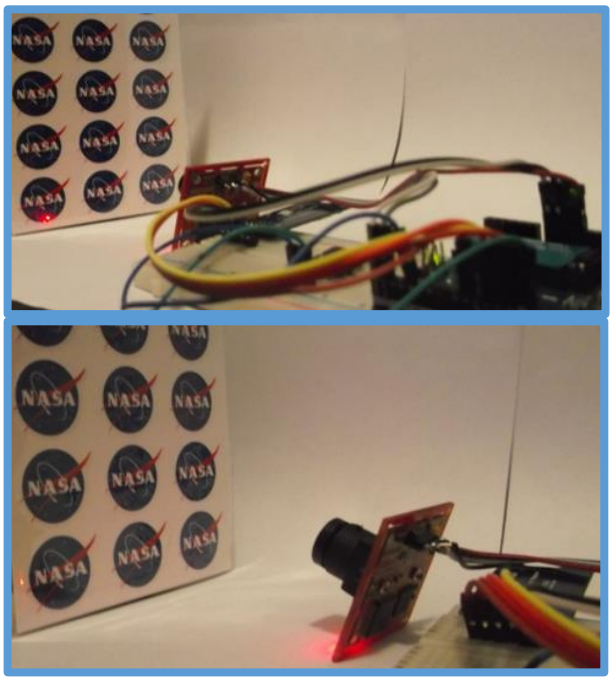


Figura 12. Cámara OV528 capturando una imagen del logo de la NASA.

Figura 13. Imagen tomada con la cámara.

V. FUENTES DE CONSULTA

[1] http://libroweb.alfaomega.com.mx/book/487/free/ovas_statics/sensores/temas/SA_TEMA_05-OPTOELECTRONICA_1_.pdf

[2] Leonel G., 2015, Sensores y Actuadores Aplicaciones con Arduino, Pág. 133 - 135., Grupo Editorial Patria.

[3] http://www.adrirobot.it/datasheet/processori/pdf/OV528_DS_1.3%20Camera%20chips.pdf

[Datasheet]

Práctica 20. XBee (Módulos de radiofrecuencia)

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

Resumen- En este trabajo se explica cómo conectar y configurar los módulos XBee S2 para comunicar dos tarjetas Arduino UNO de forma remota y finalmente usando una como maestro y la otra como esclavo elaborar dos tarea sencilla: prender y apagar un led y leer una señal analógica.

I. INTRODUCCIÓN

A. Tipos de redes

Los Xbee son pequeños dispositivos que pueden comunicarse entre sí de manera inalámbrica. Son fabricados por *Digi International*, los cuales ofrecen una gran variedad de combinaciones de hardware, protocolos, antenas y potencias de transmisión [1].

Los módulos XBee's proveen soluciones inalámbricas para la conexión entre dispositivos. Estos módulos utilizan el protocolo de redes IEEE 802.15.4 para rápida conexión de redes punto-a-multipuntos o punto-a- punto [2].

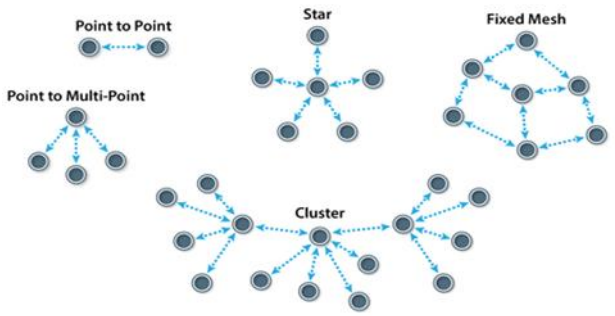


Figura 1. Tipos de redes.

Estos módulos han sido diseñados para un alto rendimiento en aplicaciones que requieran una baja latencia y un tiempo de comunicación predecible.

Existen tres tipos de dispositivo en una red [3]:

- Coordinador: Es el administrador de red y solo puede existir uno.
- Router: Se asocia con el coordinador y el dispositivo final.
- Dispositivo final: Es el dispositivo destino de la conexión.

B. XBee

Existen varios tipos de XBee. Las serie 1 son las más sencillas de trabajar, debido a que no deben ser configuradas. Además no son compatibles con otros tipos de Xbee. A la serie 2 se le puede configurar el firmware y otras características por lo cual es altamente configurable [1].

Esto significa que pueden trabajar de una forma más transparente. De igual manera, se pueden utilizar comandos AT y API para trabajar. Las XBee serie 2 se pueden combinar con las XBee serie 2 pro.

Figura 2. XBee S2.

La única forma para diferenciar una XBee serie 1 de una de serie 2, es porque en la placa no dirá nada si se trata de una serie 1, en cambio dirá Series 2 o S2 si se trata una de serie 2.

La principal diferencia entre una Xbee regular y una pro, es el precio y el alcance, por ejemplo, si el rango en una Xbee regular es 300 ft en una Xbee pro el rango será de 1 milla.

Figura 3. XBee S1 PRO.

C. XBee shield

La Xbee shield permite configurar fácilmente la XBee, por lo cual no se necesita de algún elemento extra o alimentación externa. De la misma manera permite pasar del modo programación al modo de uso con tan solo un cambio en los Terminales-headers.

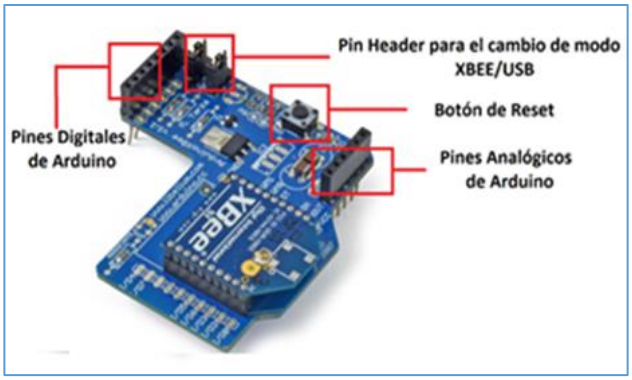


Figura 4. Partes de la shield XBee.

II. OBJETIVOS

- ✓ Conocer el funcionamiento de un módulo XBee y su funcionamiento de conexión inalámbrica entre varios dispositivos.
- ✓ Entender el funcionamiento básico del protocolo de redes IEEE 802.15.4 para rápida conexión de redes punto-a-mulipuntos y punto-a-punto.
- ✓ Aprender a realizar comunicación inalámbrica de una XBee conectada a una placa arduino.
- ✓ Conocer algunas aplicaciones de los módulos XBee.

III. MATERIALES

- ➢ 2 Tarjetas Arduino UNO
- ➢ 2 shields XBee para Arduino
- ➢ 2 Xbee configurables (Serie 2), ambos del mismo modelo.

IV. DESARROLLO

A. Descarga e instalación del XCTU

Primero se descarga la versión 5.2.8.6 de XCTU de la página:

https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu#productsupport-utilities

Figura 5. Descarga del XCTU.

Antes de montar las shield sobre los Arduino es necesario colocarlos en modo "reset", conectando un cable del pin "reset" a GND, con lo cual se habilita el uso directo de las XBee, en vez de la programación del Arduino.

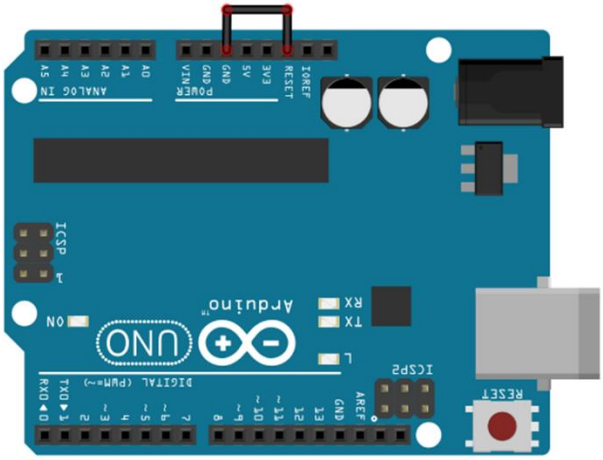


Figura 6. Conexión del RESET a GND.

Una vez hecho esto, montar las Shield sobre las tarjetas Arduino y conectarlas a la computadora. Hay que tomar en cuenta que las terminales-headers de la shield se encuentren en modo USB, de lo contrario no funcionará.

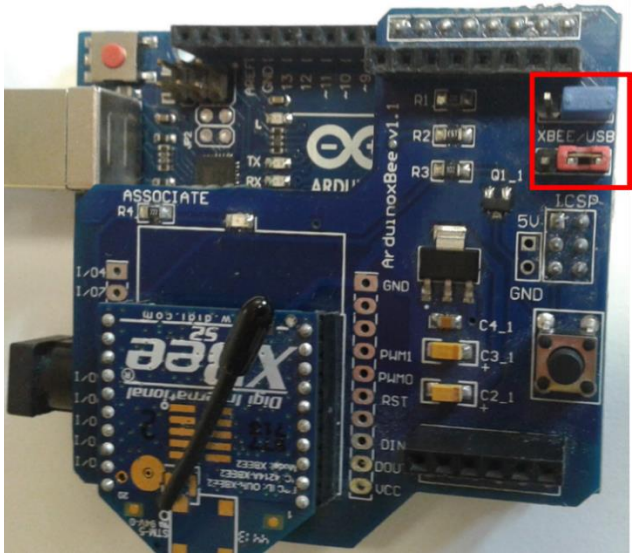


Figura 7. Terminales-headers en modo USB.

A continuación se inicia el software X-CTU, este permite la configuración de los XBee. Ya sea la versión 5.2.8.6 o la más reciente, las modificaciones a realizar son las mismas. Se deben abrir dos ventanas.

Conectar ambas tarjetas Arduino a la computadora, en la primera pestaña "PC Settings" aparecen los puertos en los que se conectaron las tarjetas Arduino. En caso de que no aparezcan, se deberá a que los drivers del Arduino se están instalando. Esperar a que se instalen y abrir de nuevo el XCTU [4].

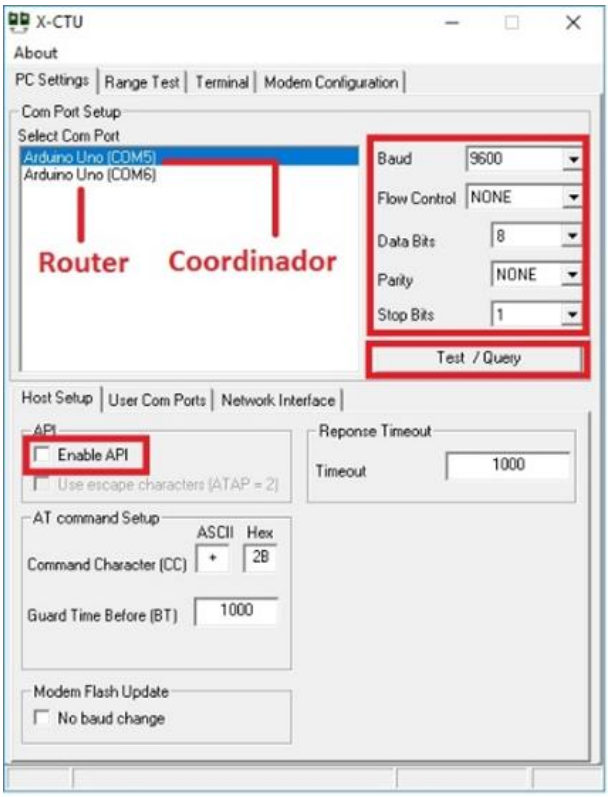


Figura 8. Ventana de XCTU.

Las opciones de velocidad, control de flujo, bits de datos, paridad y bits de parada deben mantenerse como muestran en la imagen

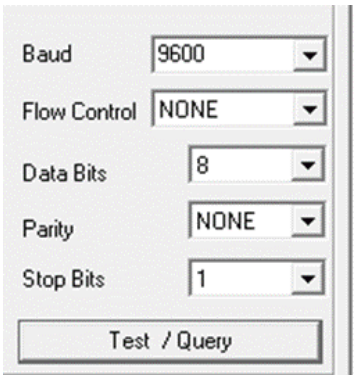


Figura 9. Configuración para la conexión.

Al hacer clic en Tes/Query aparece una imagen como la siguiente:

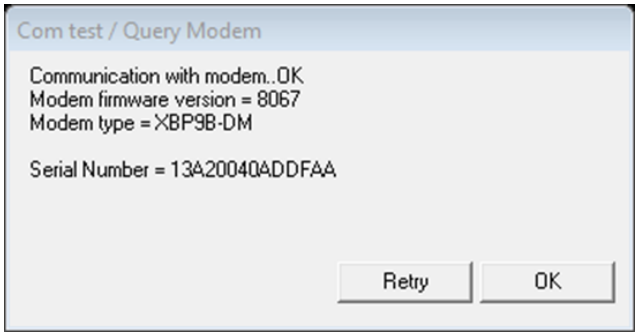


Figura 10. Conexión exitosa.

Lo que comprueba que hay conexión entre los módulos XBee y el software.

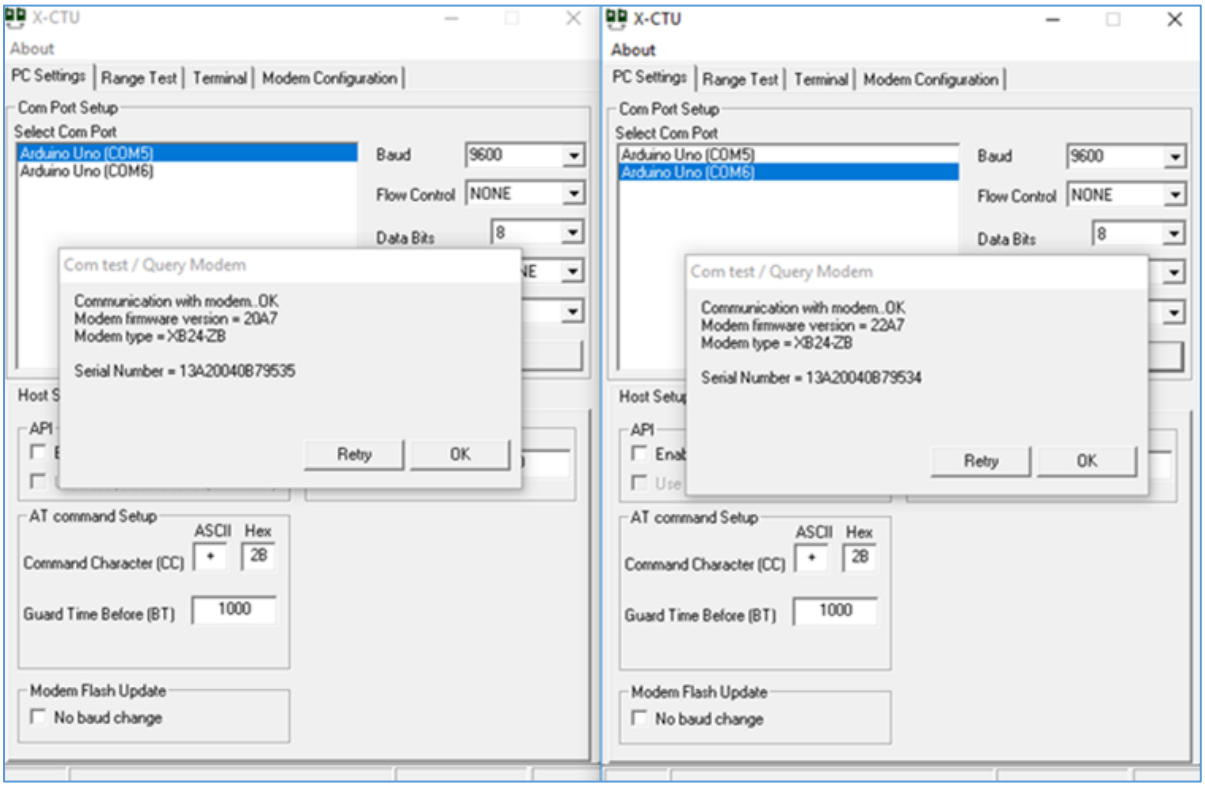


Figura 11. Conexión exitosa en ambos módulos XBee.

B. XBee como Coordinador

Seleccionado un primer puerto, en la pestaña "Modem Configuration", se activa la casilla "Always update firmware" y se selecciona el botón "Read". Con ello se obtiene la información actual del módulo XBee y estará listo para ser modificado, la sección "modem" también se actualizará automáticamente. Si es la primera vez que se está trabajando con XCTU este proceso de actualización podrá tardar entre 5 y 10 minutos. Del segundo menú desplegable se selecciona ZIGBEE COORDINATOR AT [4].

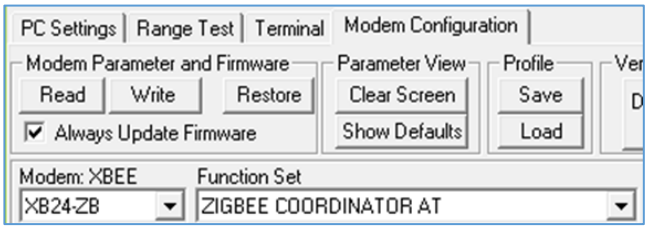


Figura 12. ZIGBEE COORDINATOR AT.

En la sección "Networking" se modifica el campo ID-PAN ID, escribiendo un número cualquiera para identificar la red.

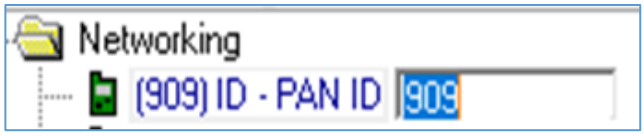


Figura 13. ID-PAN.

En la sección "Adressing" se modificará el campo *NI-Node Identifier*, con la palabra "Coordinator".

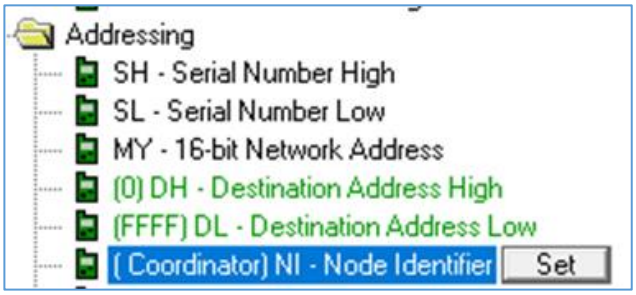


Figura 14. NI-Node Coordinador.

A continuación es necesario vincular los módulos mediante sus números de serie. Estos se pueden encontrar en la etiqueta detrás del módulo o ser capturados desde la sección "Adressing". En esa misma sección, los campos "Destination Adress" deben contener la información de los campos "Serial Number" del otro módulo, por lo general en "High" comparten el mismo identificador. En caso de que no aparezcan en sus respectivos campos serie, una llamada a lectura será suficiente.

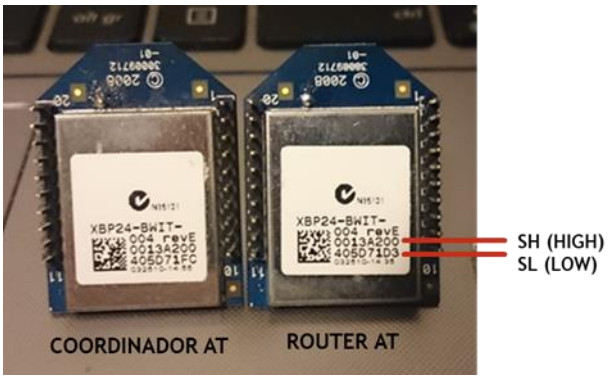


Figura 15. Números de serie.

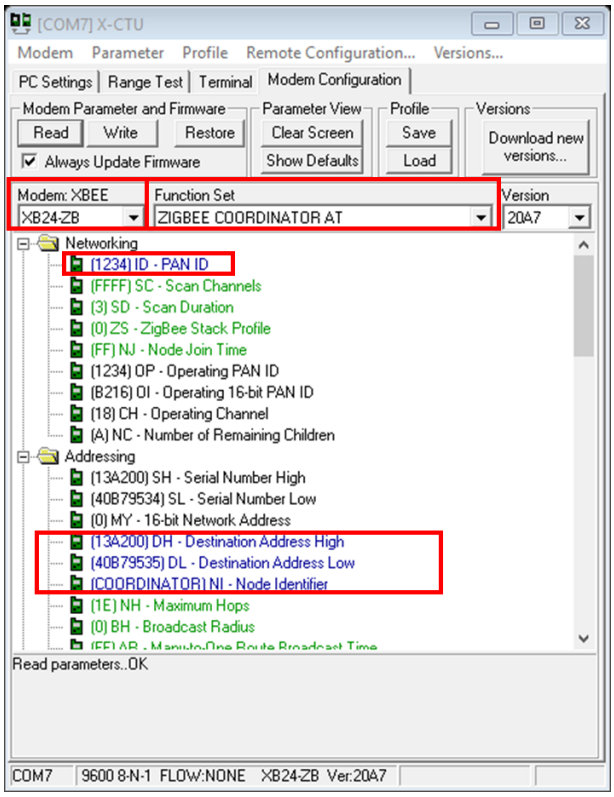


Figura 16. Xbee como Coordinador.

C. XBee como Router

Se realiza el mismo procedimiento en la segunda ventana, con el otro puerto seleccionado, sólo que esta vez en el campo "Function set" se elige ZIGBEE ROUTER AT. El campo *ID PAN-ID* debe contener el mismo número que el caso anterior, el campo *NI-Node Identifier* debe contener "Router" y se vinculan los números serie.

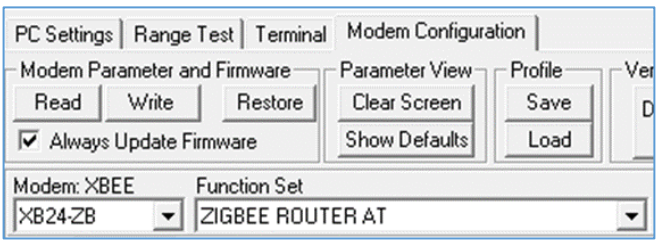


Figura 17. ZIGBEE ROUTER AT.

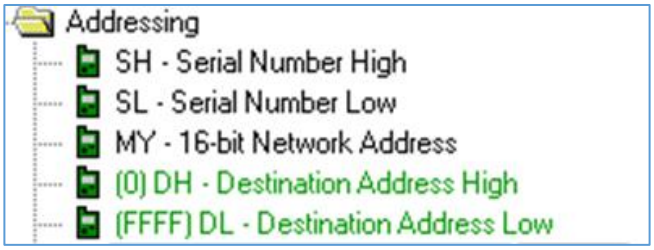


Figura 18. NI-Node Router.

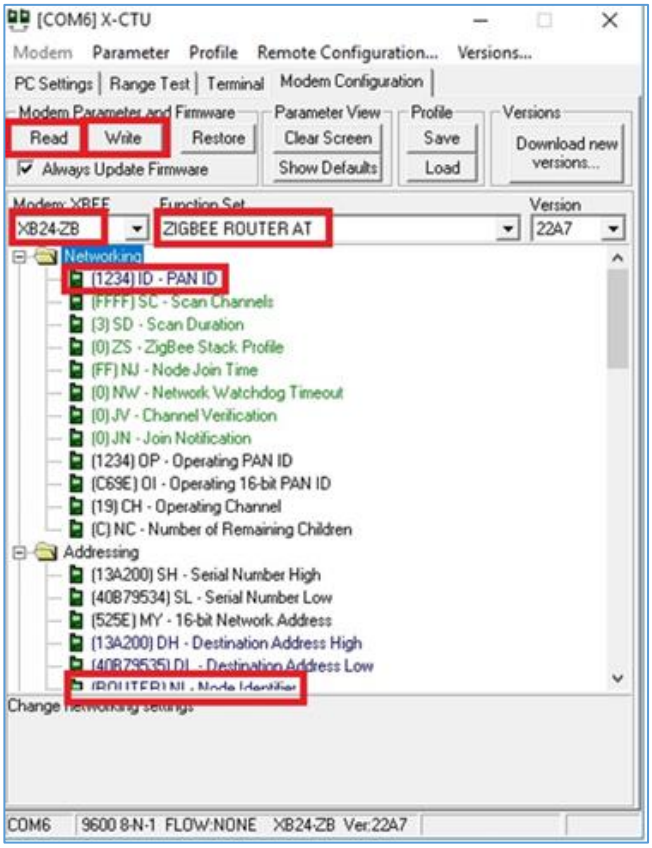


Figura 19. Xbee como Router.

A continuación se hace clic en el botón “Write”, para cargar las nuevas configuraciones. Es recomendable esperar alrededor de 30 segundos antes de realizar alguna acción. El XCTU debe enviar una notificación de éxito en la barra de estado.

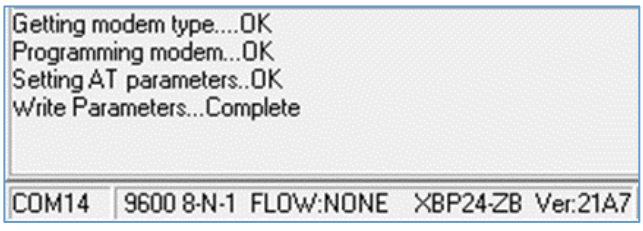


Figura 20. Carga completa.

En ambas ventanas ir a la pestaña “Terminal”. Lo que se escriba en una debe aparecer en la otra, aquello escrito en la terminal “local” aparece en azul, mientras que todo aquello que se recibe aparece en rojo.

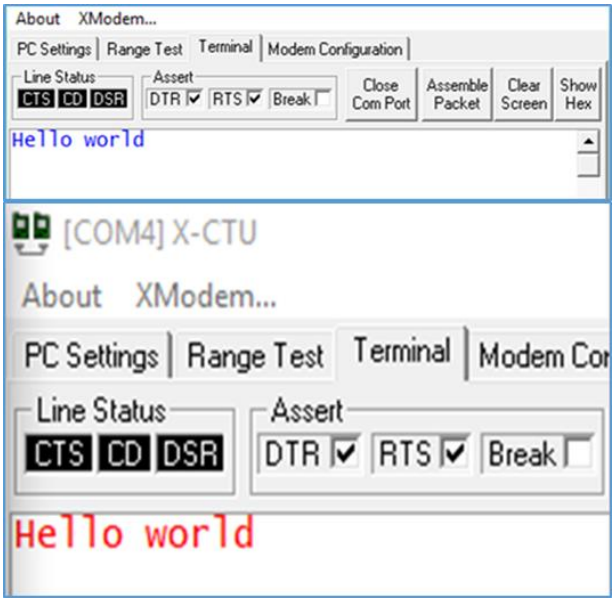


Figura 21. Texto enviado y recibido.

Los módulos XBee admiten comandos AT.A. Una primera prueba implica escribir “+++AT”, donde AT se refiere a “attention”, se debería recibir “OK” como respuesta. “+++” se interpreta como el inicio de una línea de comandos, por lo que debe escribirse antes de cualquier orden AT, por lo general se recibe “OK”, lo cual implica que el módulo está listo para recibir un comando. Algunos comandos básicos son los siguientes:

+++ – Permite entrar al modo de Comandos.
ATDL – Permite obtener la dirección baja de direccionamiento (DL).
ATSL – Permite obtener el número serial bajo (SL).
ATID – Permite obtener el PAN ID o número de identificación.

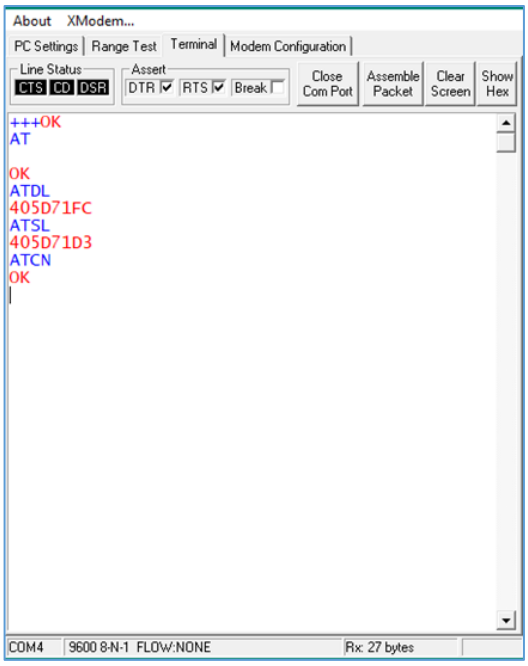


Figura 22. Prueba de comandos.

Otros comandos AT permiten la reconfiguración total de los módulos desde la línea de comandos y es posible almacenar una secuencia de estos en un archivo de texto; sin embargo, la configuración establecida en esta práctica es suficiente para una transmisión simple de datos por comunicación serie inalámbrica.

Al actualizar la configuración de los XBee, es importante observar que los leds propios de las shield parpadeen; el coordinador lo hará más lentamente que el router. El que un led se mantenga encendido puede implicar el daño del módulo o una configuración incorrecta, en tal caso es recomendable verificar ésta última o dar clic en el botón *Restore* en la pestaña *Modem configuration*, con lo cual se carga la configuración inicial del módulo y se puede volver a programar.

D. XBee con Arduino

En esta actividad se va a utilizar la comunicación serie para comunicar los dos Arduinos para cumplir una tarea sencilla: prender y apagar un led cada segundo.

Nota: Se recomienda usar 2 tarjetas Arduino UNO, aunque si se ocupa otra, por ejemplo, la tarjeta Arduino DUE, esta no permite la configuración de XBee, por lo que ambos módulos se deben ajustar uno a la vez desde la placa Arduino UNO.

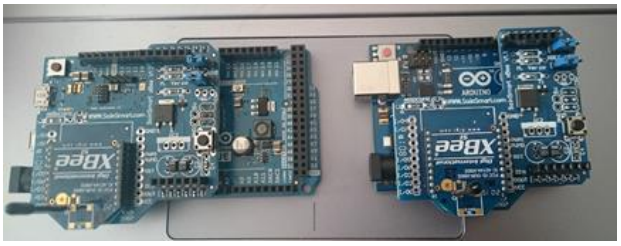


Figura 23. Tarjetas Arduino UNO y DUE.

Desconectar el pin “RESET” del Arduino de GND

Figura 24. Pin RESET desconectado.

Las shield se conectan directamente a las terminales serie del Arduino. En el caso del modelo DUE se utilizan el conjunto “serial0” en sus puertos 0 y 1.

Figura 25. Puerto serie 0 en la tarjeta Arduino DUE.

Una vez que cada shield esté montadas sobre sobre cada Arduino se van a cargar los programas que se presentan a continuación. En este caso es indiferente cual es el *Coordinador* y cual es *Router* ya que ambos están conectados entre sí y lo que se manda en uno llega al otro. Para que los módulos puedan operar como transmisores y receptores, las terminales-headers deben estar en modo XBee.

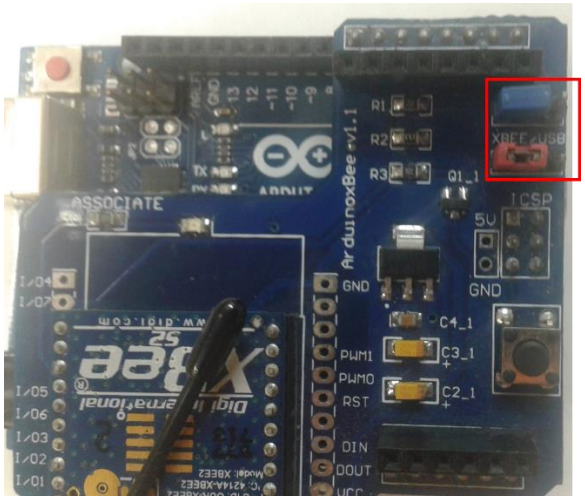


Figura 26. Terminales-headers en modo XBee.

Para una placa Arduino “Maestro” se escribe el siguiente código:

```
void setup() {
  Serial.begin(9600); //Velocidad de transmisión
}
void loop() {
  Serial.print('H'); //Impresión en puerto serial
  delay(1000); //Tiempo de espera
  Serial.print('L');
  delay(1000);
}
```

Se envía “H” como comando para encender el LED y “L” para apagarlo, en un ciclo de 2 segundos en total.

Para la segunda placa “Esclavo” se escribe el siguiente código:

```
const int ledPin=13; //Inicia el pin 13, el cual incluye un LED
int incomingByte; //Una variable para almacenar la información serial
void setup() {
  Serial.begin(9600); //La velocidad de transmisión debe ser la misma
  pinMode(ledPin, OUTPUT); //El pin 13 se establece como salida
}
void loop() {
  if (Serial.available()>0){ //Evalúa la existencia de información entrante
    incomingByte=Serial.read();//Se lee el buffer serie
    if (incomingByte=='H'){
      digitalWrite(ledPin, HIGH); //Al recibir "H" se enciende el LED
    }
    if (incomingByte=='L'){
      digitalWrite(ledPin, LOW); //Al recibir "L" apaga el LED
    }
  }
}
```

Los resultados son observables mediante el led TX de la placa “Maestro”: con cada encendido de éste, ocurre un cambio en el led 13 de la placa “Esclavo”.

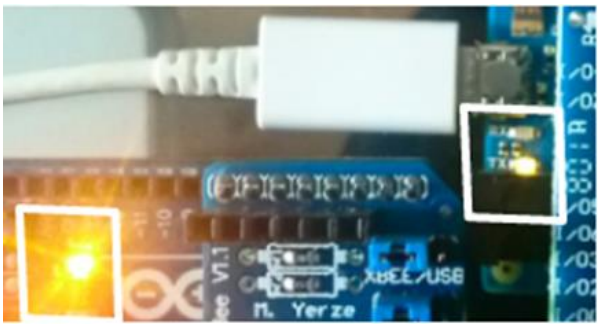


Figura 33. Leds de comunicación serie.

V. FUENTES DE CONSULTA

[1] Coronado, Enrique. (2013). Tutorial Xbee parte 1: ¿Qué es un Xbee y qué es necesario?. *mecatronicauaslp.wordpress.com*. Recuperado de https://mecatronicauaslp.wordpress.com/2013/07/04/xbee-parte-1-que-es-un-xbee-y-que-es-necesario/

[2] MCI electronics. (s.f.) ¿QUÉ ES XBEE?. *http://xbee.cl/*. Recuperado de http://xbee.cl/que-es-xbee/ .

[3] Plataformas Zigbee (2012). PRÁCTICA 1 CONFIGURACIÓN Y CONCEPTOS BÁSICOS XBEE PRO S2B. http://plataformaszigbee.blogspot.com . Recuperado de http://plataformaszigbee.blogspot.com/2012/05/practica-1-configuracion-y-conceptos.html

[4] Plataformas Zigbee. (2013). XBee configuration. http://robotic-controls.com . Recuperado de http://robotic-controls.com/book/export/html/14.

Práctica 21. Módulo WiFi

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Ciudad de México, México

***Resumen*- En esta práctica se explica cómo utilizar la shield con el módulo WiFi ESP8266 para manipular las entradas y salidas de una tarjeta Arduino realizando cuatro actividades con un Módulo WIFI ESP8266. La primera actividad consiste en una impresión de la dirección MAC y el escaneo de las redes de WiFi disponibles; después la conexión del el Arduino a una red WiFi encriptada por medio del módulo; en la siguiente se manipula el apagado y encendido de un led, por medio de una página web y la señal WiFi y en la última se imprime el valor de una entrada analógica.**

I. INTRODUCCIÓN

A. Módulo WiFi ESP8266

El módulo WiFi ESP8266 es un SOC (*System on a Chip*) de bajo consumo, el cual tiene integrado el protocolo TCP/IP [1], con el cual se puede conectar cualquier microcontrolador a una red WiFi. El ESP8266 es capaz de alojar una aplicación (*host*) o descargar (*offloading*) todas las funciones de una red WiFi desde otra aplicación.

Cada módulo viene preprogramado con un firmware de comandos AT, lo cual nos permite controlarlo a través de cualquier UART (*Universal Asynchronous Receiver/Transmitter*).

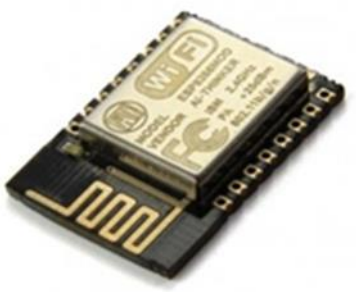


Figura 1. Módulo WiFi ESP8266.

B. Shield inalámbrico WiFi utilizando el ESP8266 con convertidor TTL

El módulo ESP8266 viene montado sobre una placa (*shield*) para facilitar las conexiones; gracias a esto, basta con montarlo sobre el Arduino y realizar unas conexiones básicas para empezar a programar y trabajar. El uso de esta shield está estrechamente relacionado con proyectos de IoT (*Internet of Things*). En esta placa tenemos el *Debug Port*, el cual nos permite realizar la comunicación entre el Arduino y la shield de manera directa. De igual manera tenemos un DIP Switch, el cual nos permite realizar la configuración manual de la shield .Esto significa que se puede realizar la actualización del Firmware [2]. También hay terminales de alimentación, terminales analógicas y terminales digitales del Arduino a disposición del usuario para realizar la conexión que se desee [3].

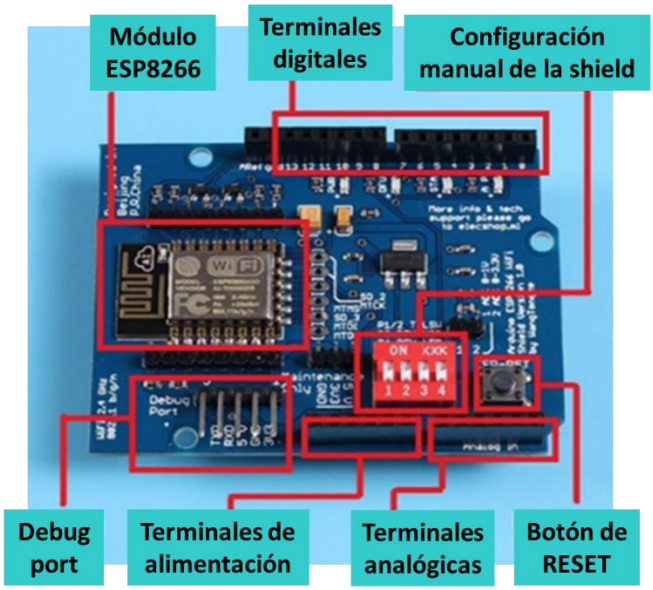


Figura 2. Descripción de la WiFi wireless shield con ESP8266.

II. OBJETIVOS

- ✓ Descargar y hacer uso de bibliotecas externas de WiFi.
- ✓ Aprender a usar la shield WiFi con la tarjeta Arduino UNO.
- ✓ Manipular y monitorear la tarjeta Arduino desde una página web.

III. MATERIALES

- ➢ Shield WiFi con el módulo ESP8266.
- ➢ Tarjeta Arduino UNO.
- ➢ 4 Jumpers macho-hembra.
- ➢ 3 Jumpers macho-macho.
- ➢ Protoboard.
- ➢ Alambre para protoboard.
- ➢ Computadora con el software IDE de Arduino.
- ➢ 1 Led.
- ➢ Red WiFi disponible.
- ➢ Antena WiFi o adaptador WiFi USB (opcional).

IV. PREPARACIÓN DE LA SHIELD Y LA TARJETA ARDUINO

Por default la shield WiFi se encuentra con un *baud rate* de 115200 baudios. El Arduino UNO no tiene un hardware serie aparte del utilizado por la conexión USB, por lo tanto se tiene que utilizar una comunicación serie a través de software. Esta comunicación serie no puede manejar esta velocidad de comunicación; por lo tanto, se debe de configurar la ESP8266 a 9600 baudios.

Antes de realizar cualquier conexión, abrir la IDE de Arduino y cargar un programa vacío para evitar cualquier problema, esto con la shield completamente desconectada.

Una vez realizado esto, desconectar el Arduino y asegurarse que todos los switches del Dip switch de la shield se encuentren hacia abajo. Después conectar las terminales del *Debug Port* de la shield a la tarjeta Arduino de la siguiente manera.

Forma 1 de conectar la Shield y la tarjeta Arduino:

Debug Port TX → Arduino UNO terminal 1 (TX).

Debug Port RX → Arduino UNO terminal 0 (RX).

Debug Port 5V → Arduino UNO 5V.

Debug Port GND → Arduino UNO GND.

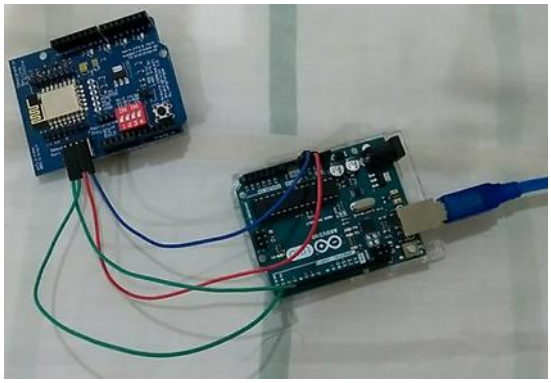


Figura 3. Conexión entre la tarjeta Arduino y la shield WiFi.

Conectar el Arduino a la computadora y abrir el monitor serie con una taza de 115200 baudios y con la opción de “Ambos NL & CR” (*Newline and Carriage Return*). Como se muestra en la imagen.

Figura 4. Monitor serie a 115200 baudios.

Una vez hecho esto se podrán usar los comandos AT [1], los cuales son muy utilizados en las comunicaciones. En el monitor serie copiar la siguiente instrucción AT:

AT+UART_DEF=9600,8,1,0,0

Después cambiar la taza a 9600 baudios y presionar el botón de reinicio de la shield WiFi por dos segundos.

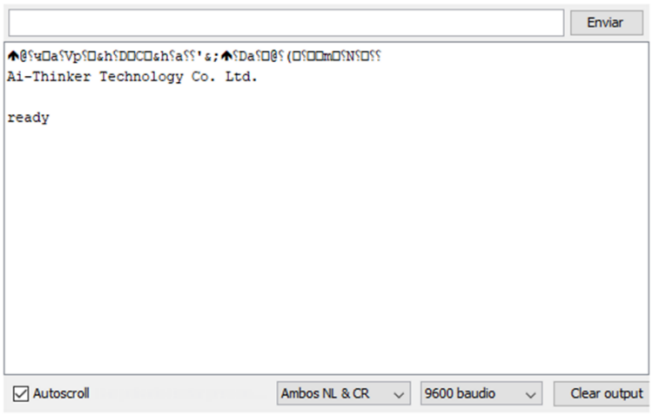


Figura 5. Monitor serie a 9600 baudios.

Finalmente mandar la instrucción:

AT

La instrucción AT permite saber si el módulo está respondiendo correctamente. Se debe recibir un OK a modo de respuesta.

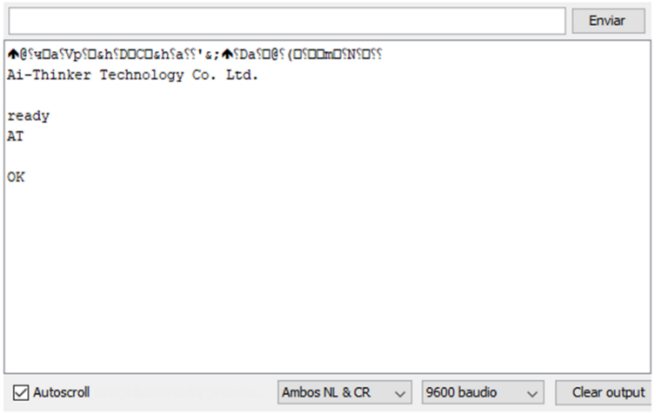


Figura 6. Módulo funcionando exitosamente.

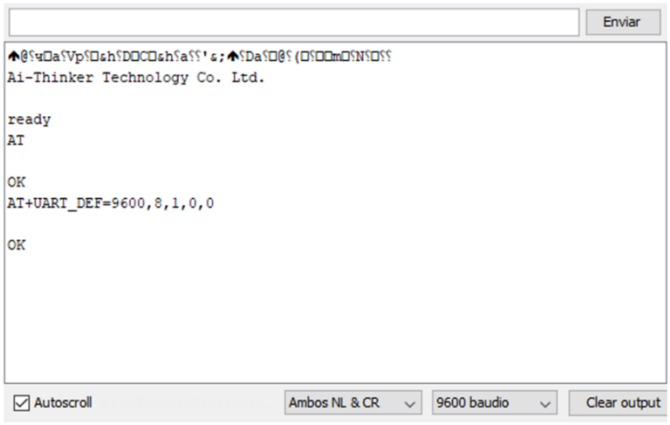


Figura 7. Comandos AT en el monitor serie.

A continuación se mencionan algunas instrucciones [1]:

AT – Revisa si el módulo (en este caso la shield) está conectada correctamente y está respondiendo correctamente.

AT+RST – Esta instrucción resetea el módulo Wi-Fi.

AT+GMR – Menciona la versión del firmware que se encuentra instalado en el ESP8266.

AT+CWLAP – Detecta los puntos de acceso y la intensidad de la señal de los módems que se encuentren cerca.

AT+CWJAP=”SSID”,”PASSWORD” – Conectará la ESP8266 a la SSID especificada.

AT+CIFSR– Obtendremos la dirección IP del ESP8266.

AT+CWJAP=””,””– Con este comando se desconecta del punto de acceso al cual se ha conectado previamente.

AT+CWMODE=1 – Configura el módulo WiFi al modo 1, esto significa que el módulo estará configurado como un nodo, como lo son nuestros teléfonos los cuales se conectan a los puntos de acceso (Módems)

Los comandos AT son instrucciones codificadas que conforman un lenguaje de comunicación entre el hombre y un terminal modem. Aunque la finalidad principal de los comandos AT es la comunicación con módems, la telefonía móvil GSM también ha adoptado como estándar este lenguaje para poder comunicarse con sus terminales.

Ahora el usuario esta listo para empezar a programar, pero primero se debe descargar la biblioteca WiFiEsp, con la cual se puede programar y comunicar el Arduino con la shield WiFi.

Enlace de Descarga [4]: https://github.com/bportaluri/WiFiEsp

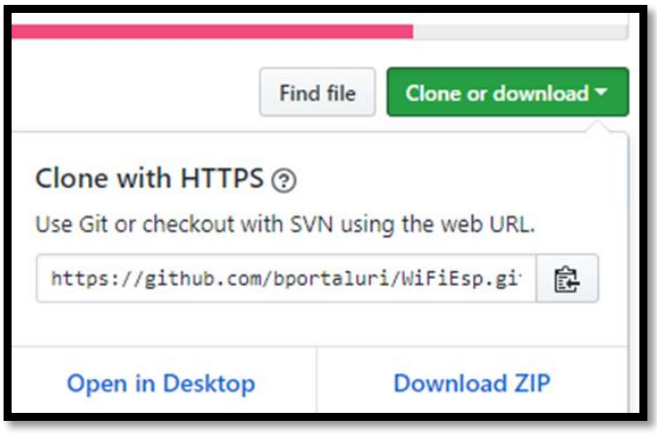


Figura 8. Página web donde se descarga de la biblioteca WiFiEsp.

Descargar la carpeta y descomprimirla. Dentro de esta carpeta se encuentran ejemplos de cómo utilizar la tarjeta y la biblioteca.

Ir a la IDE de Arduino: *Programa* → *Incluir Librería* → *Añadir librería* .ZIP y seleccionar la carpeta que se acaba de descargar. Una vez hecho esto, la biblioteca WifiEsp ya estará incluida.

V. CONEXIONES

Una vez preparada la shield, desconectar la tarjeta Arduino de la computadora y los jumpers. Después montar la Shield sobre la tarjeta Arduino y conectar de nuevo los cables pero ahora de la siguiente manera:

Forma 2 de conexión entre la shield y la tarjeta Arduino:

Debug Port TX → Shield terminal 6 (RX)

Debug Port RX → Shield terminal 7 (TX)

Debug Port 5V → Shield 5V

Debug Port GND → Shield GND

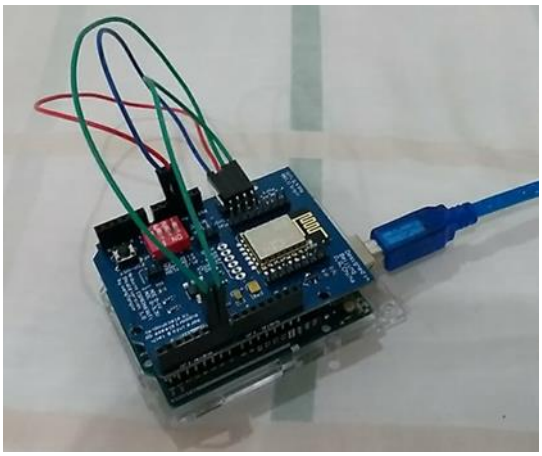


Figura 9. Conexión de la shield una vez montada sobre el Arduino.

Para los dos últimos programas es recomendable conectar un led y un potenciómetro a la shield WiFi. El led al pin 13 y el potenciómetro a A0.

Nota: Tanto el módulo WiFi como el navegador donde se abren las páginas web de los últimos dos programas deben estar conectados a la misma red WiFi.

VI. PROGRAMAS Y RESULTADOS

A. Conexiones Disponibles

Este programa imprime la dirección MAC de la shield WiFi y busca las redes WiFi disponibles cada 10 segundos.

```
#include "WiFiEsp.h"
//Emulando la comunicación serie a través de los pines 6 y 7
#ifndef HAVE_HWSERIAL1
#include "SoftwareSerial.h"
SoftwareSerial Serial1(6, 7); // RX, TX
#endif
void setup()
{
 Serial.begin(115200); //inicializando el serial para el
debugging
 Serial1.begin(9600); // iniciando el serial para el módulo ESP
 WiFi.init(&Serial1);  //Inicializando el módulo ESP
 // Checando la precencia de la shield de WiFi
 if (WiFi.status() == WL_NO_SHIELD)
{
            Serial.println("WiFi shield not present");
            while (true);
 }
 printMacAddress(); // Imprimiendo la dirección MAC WiFi
}
void loop()
{
 // Escaneando las posibles conexiones
 Serial.println();
 Serial.println("Scanning available networks...");
 listNetworks();
 delay(10000);
}
void printMacAddress()
{
 // Obteniendo la dirección MAC
 byte mac[6];
 WiFi.macAddress(mac);

 // Imprimiendo la dirección MAC
 char buf[20];
 sprintf(buf, "%02X:%02X:%02X:%02X:%02X:%02X",
mac[5], mac[4], mac[3], mac[2], mac[1], mac[0]);
 Serial.print("MAC address: ");
 Serial.println(buf);
}
void listNetworks()
{
 // Escaneando las conexiones cercanas
 int numSsid = WiFi.scanNetworks();
 if (numSsid == -1)
{
            Serial.println("Couldn't get a wifi connection");
            while (true);
 }
 //Imprimiendo la lista de conexiones disponibles
 Serial.print("Number of available networks:");
 Serial.println(numSsid);
 //Imprimiendo el número de red y el nombre para cada red
 for (int thisNet = 0; thisNet < numSsid; thisNet++)
{
            Serial.print(thisNet);
            Serial.print(") ");
            Serial.print(WiFi.SSID(thisNet));
            Serial.print("\tSignal: ");
            Serial.print(WiFi.RSSI(thisNet));
            Serial.print(" dBm");
            Serial.print("\tEncryption: ");
            printEncryptionType(WiFi.encryptionType(thisNet));
 }
}

void printEncryptionType(int thisType)
{
 // Lee el tipo de encriptación e imprime el nombre
 switch (thisType) {
            case ENC_TYPE_WEP:
            Serial.print("WEP");
            break;
            case ENC_TYPE_WPA_PSK:
            Serial.print("WPA_PSK");
            break;
            case ENC_TYPE_WPA2_PSK:
            Serial.print("WPA2_PSK");
            break;
            case ENC_TYPE_WPA_WPA2_PSK:
            Serial.print("WPA_WPA2_PSK");
     break;
            case ENC_TYPE_NONE:
            Serial.print("None");
            break;
```

```
  }
  Serial.println();
}
```

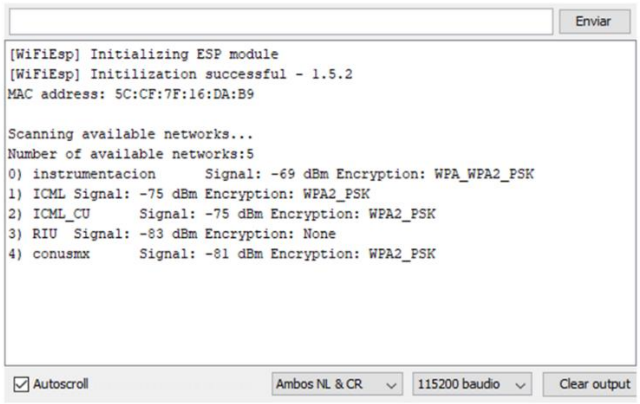


Figura 10. Redes WiFi detectadas por el módulo WiFi.

B. Conexión a un punto de acceso

Figura 11. Parámetros SSID y contraseña como referencia.

Este programa conecta el Arduino a una red WiFi encriptada utilizando el módulo ESP8266. Se imprime la dirección MAC de la WiFi shield, la dirección IP obtenida y otros detalles de la red.

```
#include "WiFiEsp.h"
//Emulando la comunicación serie a través de los pines 6 y 7
#ifndef HAVE_HWSERIAL1
#include "SoftwareSerial.h"
SoftwareSerial Serial1(6, 7); // RX, TX
#endif
char ssid[] = "INFINITUM5156_2.4"; //La SSID de tu red (nombre de tu punto de acceso)
char pass[] = "XXXXXXXXXXXX";          // La contraseña de tu red
int status = WL_IDLE_STATUS;   // El status del radio de Wifi
void setup()
{
  Serial.begin(115200); //inicializando el serial para el debugging
  Serial1.begin(9600); // iniciando el serial para el módulo ESP
  WiFi.init(&Serial1); //Inicializando el módulo ESP
  // Checando la presencia de la shield de WiFi
  if (WiFi.status() == WL_NO_SHIELD)
{
            Serial.println("WiFi shield not present");
            // don't continue
            while (true);
  }
  // Intento de conectarse a la red WiFi
  while ( status != WL_CONNECTED)
{
            Serial.print("Attempting to connect to WPA SSID: ");
            Serial.println(ssid);
            status = WiFi.begin(ssid, pass); // Contectandose a la red WPA/WPA2
  }
  Serial.println("You're connected to the network");
}
void loop()
{
  //Imprime la información de la conexión de la red cada 10 segundos
  Serial.println();
  printCurrentNet();
  printWifiData();
  delay(10000);
}
void printWifiData()
{
  // Imprime la dirección IP de la shield de WiFi
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  // Imprime la dirección MAC
  byte mac[6];
  WiFi.macAddress(mac);
  char buf[20];
  sprintf(buf, "%02X:%02X:%02X:%02X:%02X:%02X", mac[5], mac[4], mac[3], mac[2], mac[1], mac[0]);
  Serial.print("MAC address: ");
  Serial.println(buf);
}
void printCurrentNet()
{
  // Imprime la SSID de la red en la cual te encuentras conectado
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());
  //Imprimé la dirección MAC del router al cual te encuentras conectado
  byte bssid[6];
  WiFi.BSSID(bssid);
  char buf[20];
  sprintf(buf, "%02X:%02X:%02X:%02X:%02X:%02X", bssid[5], bssid[4], bssid[3], bssid[2], bssid[1], bssid[0]);
  Serial.print("BSSID: ");
  Serial.println(buf);
  // Imprimé la intensidad de la señal
```

```
 long rssi = WiFi.RSSI();
 Serial.print("Signal strength (RSSI): ");
 Serial.println(rssi);
}
```

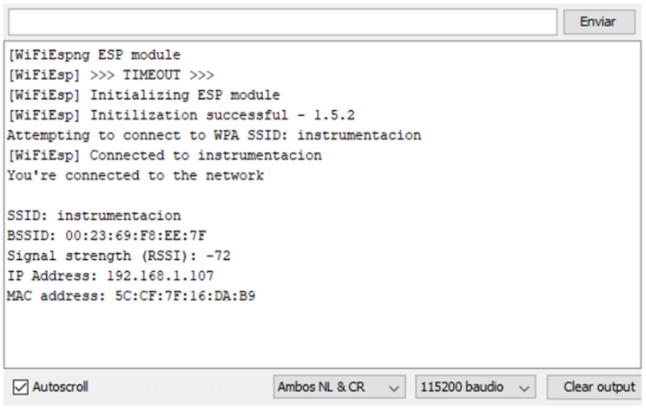


Figura 12. Conexión exitosa a una red WiFi.

C. Prendiendo y apagando un led a través de una página Utilizando un servidor web simple se puede prender y apagar un led a través de una página web. El programa imprime la dirección IP del módulo ESP8266 en el monitor serie; desde esa dirección se puede visualizar en el navegador web las opciones para prender y apagar el led de la terminal 13.

```
#include "WiFiEsp.h"
//Emulando la comunicación serie a través de las terminales 6
y 7
#ifndef HAVE_HWSERIAL1
#include "SoftwareSerial.h"
SoftwareSerial Serial1(6, 7); // RX, TX
#endif
char ssid[] = "INFINITUM5156_2.4"; //La SSID de tu red
(nombre de tu punto de acceso)
char pass[] = "XXXXXXXXXX";              // La contraseña
de tu red
int status = WL_IDLE_STATUS;    // El status del radio de
Wifi
int ledStatus = LOW;
WiFiEspServer server(80);
RingBuffer buf(8); // Se utilizará un anillo del buffer para
incrementar la
                //velocidad y reducir la memoria
void setup()
{
 pinMode(LED_BUILTIN, OUTPUT);       // Inicializamos el
pin 13 como salida
 Serial.begin(115200);  //inicializando el serial para el
debugging
 Serial1.begin(9600);       // iniciando el serial para el modulo
ESP
 WiFi.init(&Serial1);       //Inicializando el modulo ESP
 // Checando la presencia de la shield de WiFi
 if (WiFi.status() == WL_NO_SHIELD) {
          Serial.println("WiFi shield not present");
          // don't continue
          while (true);
 }
// Intento de conectarse a la red WiFi
 while (status != WL_CONNECTED) {
          Serial.print("Attempting to connect to WPA SSID: ");
          Serial.println(ssid);
          status = WiFi.begin(ssid, pass); // Conectándose a la
red WPA/WPA2
 }
 Serial.println("You're connected to the network");
 printWifiStatus();
 server.begin(); //Empieza el servidor web en el puerto 80
}
void loop()
{
 WiFiEspClient client = server.available();  // Lista a los
clientes entrantes
 if (client) {                             // Si consigues un cliente
  Serial.println("New client");                  //Imprime un
mensaje en el puerto serial
  buf.init();                               // Inicializa el buffer
circular
  while (client.connected()) {                  // Que siga el ciclo
mientras el cliente esté conectado
    if (client.available()) {             // Si hay bytes que leer, se
leen desde el cliente
      char c = client.read();             // Se lee el byte
      buf.push(c);                        // Se empuja el byte en al
buffer de anillo
          // La impresión del flujo del monitor serial bajará la
velocidad
      // de la información recibida del ESP llenando así el
buffer serial
      //Serial.write(c);

      // Si tienes 2 nuevas líneas de caracteres en una fila
      // entonces ese será el fin de la solicitud del HTTP, por lo
cual enviará la respuesta
          if (buf.endsWith("\r\n\r\n")) {
          sendHttpResponse(client);
          break;
          }
          //Checa si la solicitud del cliente fue "GET /H" or
"GET /L"
          //High - Encendido o  L-Apagado
           if (buf.endsWith("GET /H")) {
          Serial.println("Turn led ON");
          ledStatus = HIGH;
          digitalWrite(LED_BUILTIN, HIGH);   // Enciende el
LED
          }
          else if (buf.endsWith("GET /L")) {
          Serial.println("Turn led OFF");
          ledStatus = LOW;
          digitalWrite(LED_BUILTIN, LOW);          // Apaga
el LED
          }
```

```
            }
            }

   // Se cierra la conexión
            client.stop();
            Serial.println("Client disconnected");
  }
}
void sendHttpResponse(WiFiEspClient client)
{
  // Los encabezados HTTP siempre empiezan con una
respuesta con código  (e.g. HTTP/1.1 200 OK)
  // y el tipo de contenido por lo que el cliente sabe que va a
venir, entonces la línea en
  //blanco queda
  client.println("HTTP/1.1 200 OK");
 client.println("Content-type:text/html");
  client.println();
  //El contenido de la respuesta HTTP sigue con un
encabezado
  client.print("The LED is ");
  client.print(ledStatus);
  client.println("<br>");
  client.println("<br>");
  client.println("Click <a href=\"/H\">here</a> turn the LED
on<br>");
  client.println("Click <a href=\"/L\">here</a> turn the LED
off<br>");
  // La respuesta HTTP termina con una linea en blanco
  client.println();
}
void printWifiStatus()
{
  // Imprime la SSID de la red en la cual te encuentras
conectado
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());
  // Imprime la dirección IP de la shield de WiFi
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  // Imprime a donde in en el navegador
  Serial.println();
  Serial.print("To see this page in action, open a browser to
http://");
  Serial.println(ip);
  Serial.println();
}
```

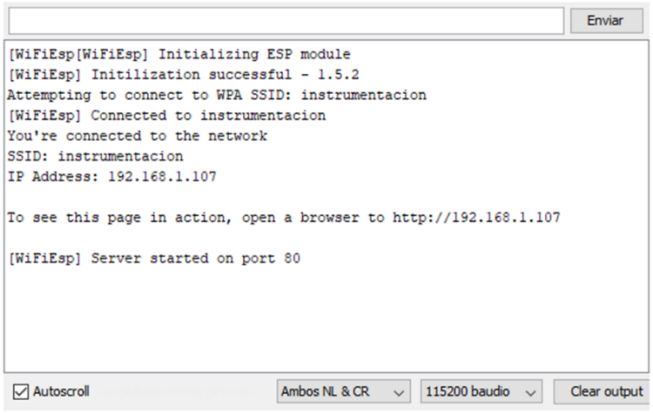


Figura 13. Conexión exitosa a una red WiFi y dirección de la página web para encender y apagar el led.

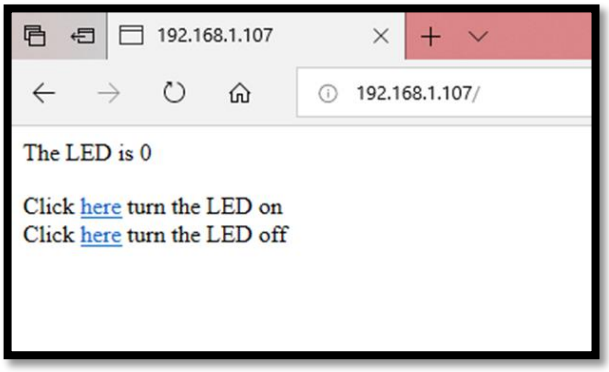


Figura 14. Página web abierta desde una computadora.

Figura 16. Página web abierta desde un smartphone.

D. Servidor web leyendo una entrada analógica

Se muestra el valor de una entrada analógica a través de una página web utilizando el módulo ESP8266. Este ejemplo imprime la dirección IP del módulo ESP8266 (una vez conectado) al monitor serie y la dirección de una página web, la cual al ser visualizada desde un navegador web muestra el valor de la entrada analógica A0, la cual se actualiza cada 20 segundos.

```
#include "WiFiEsp.h"
//Emulando la comunicación serie a través de los pines 6 y 7
#ifndef HAVE_HWSERIAL1
#include "SoftwareSerial.h"
SoftwareSerial Serial1(6, 7); // RX, TX
#endif
char ssid[] = "INFINITUM5156_2.4"; //La SSID de tu red (nombre de tu punto de acceso)
char pass[] = "XXXXXXXXXX";              // La contraseña de tu red
int status = WL_IDLE_STATUS;  // El status del radio de Wifi
int reqCount = 0;             // Número de solicitudes recibidas
WiFiEspServer server(80);
void setup()
{
 Serial.begin(115200);//inicializando el serial para el debugging
 Serial1.begin(9600);// iniciando el serial para el módulo ESP
 WiFi.init(&Serial1);//Inicializando el módulo ESP
 // Checando la presencia de la shield de WiFi
 if (WiFi.status() == WL_NO_SHIELD) {
          Serial.println("WiFi shield not present");
          // don't continue
          while (true);
 }
// Intento de conectarse a la red WiFi
 while ( status != WL_CONNECTED) {
          Serial.print("Attempting to connect to WPA SSID: ");
          Serial.println(ssid);
          status = WiFi.begin(ssid, pass);// Conectándose a la red WPA/WPA2
 }
 Serial.println("You're connected to the network");
 printWifiStatus();
 //Empieza el servidor web en el puerto 80
 server.begin();
}
void loop()
{
 WiFiEspClient client = server.available(); // Lista a los clientes entrantes
 if (client) {
          Serial.println("New client");
  //una solicitud http que termina con una línea en blanco
          boolean currentLineIsBlank = true;
          while (client.connected()) {
          if (client.available()) {
          char c = client.read();
          Serial.write(c);
          // Si has recibido al final de la línea(recibido un carácter de nueva línea)
          // Y la línea está en blanco, entonces la solicitud http ha terminado, por lo tanto
          // puedes responder
          if (c == '\n' && currentLineIsBlank) {
          Serial.println("Sending response");
          // manda una respuesta de encabezado http estándar
          // usa \r\n en lugar de varias instrucciones println para acelerar el envío de //información
        client.print(
          "HTTP/1.1 200 OK\r\n"
          "Content-Type: text/html\r\n"
          "Connection: close\r\n"  //  La conexión se cerrará después de completar la respuesta
          "Refresh: 20\r\n"        // Se refrescará la página automáticamente cada 20 segundos
          "\r\n");
          client.print("<!DOCTYPE HTML>\r\n");
          client.print("<html>\r\n");
          client.print("<h1>Hello World!</h1>\r\n");
          client.print("Requests received: ");
          client.print(++reqCount);
          client.print("<br>\r\n");
           client.print("Analog input A0: ");
          client.print(analogRead(0));
          client.print("<br>\r\n");
          client.print("</html>\r\n");
          break;
          }
          if (c == '\n') {
          //Se empezará una nueva línea
          currentLineIsBlank = true;
          }
          else if (c != '\r') {
          //Se ha recibido un caracter en la linea actual
          currentLineIsBlank = false;
          }
          }
          }
          // Se le da a la página web un tiempo para recibir la información
   delay(10);
          // Se cierra la conexión
          client.stop();
          Serial.println("Client disconnected");
 }
}
void printWifiStatus()
{
 // Se imprime la SSID de la red a la cual estas conectado
 Serial.print("SSID: ");
 Serial.println(WiFi.SSID());
// Se imprimé la dirección IP de la WiFi shield
 IPAddress ip = WiFi.localIP();
 Serial.print("IP Address: ");
 Serial.println(ip);
```

```
 //Se imprime la dirección a la cual tienes que ir en tu navegador
  Serial.println();
  Serial.print("To see this page in action, open a browser to http://");
  Serial.println(ip);
  Serial.println();
}
```

Figura 17. Página web abierta desde una computadora.

VII. FUENTES DE CONSULTA

[1] Bluehack: the Spanish Bluetoot Security Group (2005). Comandos AT. http://bluehack.elhacker.net . Recuperado de http://bluehack.elhacker.net/proyectos/comandosat/comandosat.html.

[2] VEYTON (2009). AI-Thinker ESP8266 ESP-12F WiFi/WLAN Module. watterott.com. Recuperado de http://www.watterott.com/en/ESP8266-ESP-12F-WiFi/WLAN-Module.

[3] Claus (2017). Using ESP8266 Shield ESP-12E (elecshop.ml) by WangTongze with an Arduino Uno. Claus.bloggt.es. Recuperado de https://claus.bloggt.es/2017/01/14/using-esp8266-shield-esp-12e-elecshop-ml-by-wangtongze-with-an-arduino-uno/.

https://www.youtube.com/watch?v=TnLurmCUdUE