Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

ADQUISICIÓN, PROCESAMIENTO Y MONITORIZACIÓN DE SEÑALES VÍA XBEE EN EL ENTORNO LABVIEW

DOCUMENTO 3- *CÁLCULOS SOFTWARE*

Alumno/Alumna: Fuente, Fernández, Ivon

Director/Directora: Oleagordía, Aguirre, Iñigo Javier

Curso: 2017-2018

Fecha: 23/07/2018

Índice

Índice de figuras

Índice de Tablas

1 INTRODUCCIÓN

El proyecto consta de dos entornos de programación diferenciados. Estos se emplean para la comunicación entre la placa del microcontrolador Arduino Uno, empleada como DAQ y el entorno de LabView, encargado de procesar las señales una vez adquiridas.

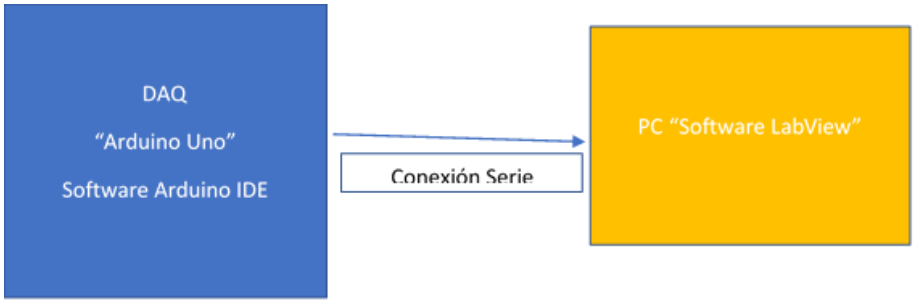

Figura 1.1 Diagrama de Bloques

Para establecer la comunicación, como se explicará posteriormente, se hace necesaria la carga de una determinada librería en el microcontrolador "Arduino Uno". Esta será la encargada de marcar las pautas para el establecimiento de la comunicación. Por otra parte, la selección de entradas, tipo de adquisición de los datos, periodos de muestreo… quedarán determinados en el entorno LabView. El mismo diagrama aplicado al hardware del proyecto aparece en la figura 1.2.

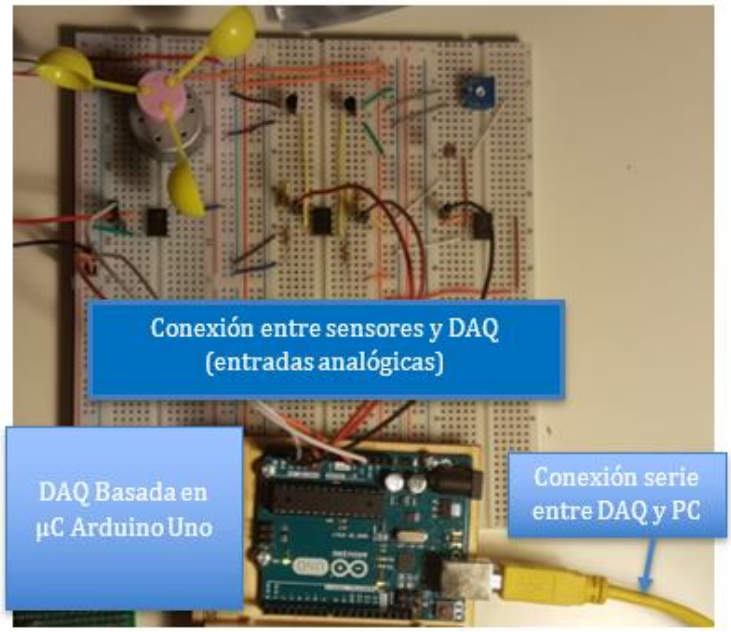

Figura 1.2. Conexiones en Hardware del proyecto

2 ENTORNOS DE PROGRAMACIÓN

En el presente apartado, se va a hacer un análisis inicial de los dos entornos software descritos en la introducción.

2.1 ENTORNO ARDUINO UNO IDE “Integrated Development Environment”

Arduino es una plataforma de hardware libre, su diseño como ya es sabido se basa en una placa con un microcontrolador *Atmel AVR.* En su arquitectura, como ya se comentó en el apartado anterior, dispone de varios puertos de entrada/salida, así como un entorno de desarrollo de fácil empleo para el desarrollo de proyectos.

El entorno de programación empleado por la plataforma Arduino para todos sus dispositivos es libre y de uso abierto. Es posible la descarga gratuita desde internet de las versiones del compilador y a fecha de redacción del presente documento, existe la posibilidad de descargarse desde la web https://www.arduino.cc/en/Main/Software la versión 1.8.5.

Figura 2.1. URL descarga Software Libre Arduino.

Como su nombre indica, el entorno software empleado por la plataforma es de tipo IDE (del inglés “Integrated Development Environment “) que viene a ser un entorno de desarrollo integrado.

Las plataformas que emplean este tipo de software, son beneficiosas para el desarrollo de aplicaciones y constan principalmente de las siguientes partes:

- Editor de código
- Un compilador
- Un depurador
- Constructor de interfaz gráfica

Los entornos de este tipo, son beneficiosos en el desarrollo paralelo global de las aplicaciones, permitiendo de este modo la generación de determinadas librerías software para uso público.

Como se comentará posteriormente, la generación de estas librerías nos permite realizar la conexión serie entre Arduino Uno y el PC con el entorno software LabView por puerto serie.

2.1.1 EDITOR DE CÓDIGO Y COMPILADOR

Según se aprecia en la Fig.2.2, el IDE de la plataforma Arduino está constituido por los siguientes apartados:

- Editor de texto para escribir el código. El lenguaje propio es muy similar al C++ con lo que la adaptación se hace de manera muy natural.
- Área de mensajes. Al igual que en el resto de compiladores más comunes, aparece una barra de mensajería en la que se reflejan los diferentes errores posibles en la programación / diseño del programa.
- Visualizador de ejecución. En la parte lateral aparece un icono que permite acceder a una pantalla de visualización de las variables. Esta acción se lleva a cabo una vez que el programa ya está corriendo en el microcontrolador.
- Barra de herramientas con botones para las funciones comunes, selección del puerto de comunicación, selección del dispositivo hardware que se va a emplear etc.

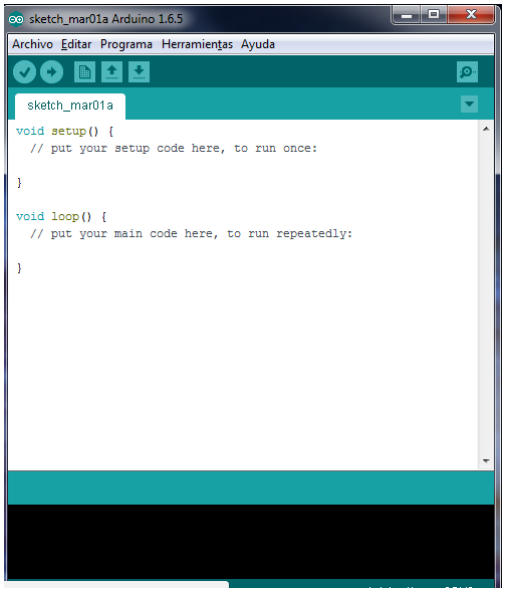

Figura 2.2. Entorno Programación Arduino

La denominación empleada para definir los programas compilados en Arduino IDE "***sketch***". Un *sketch* puede ser compilado y cargado a la placa rápidamente pulsando el botón de la flecha que se encuentra a la derecha del botón de confirmación.

En el área de mensajes se muestra información mientras se cargan los programas o los posibles errores que surjan.

La barra de herramientas permite verificar el proceso de carga, creación, apertura y guardado de programas, y la monitorización serie.

Como se ha comentado anteriormente, el lenguaje de programación de Arduino está basado en C++. Para la ejecución de un programa cíclico bastará con construir un programa definiendo las dos funciones que se detallan a continuación:

- *"Setup ()"*: Acción que se ejecuta una única vez en el programa y configura la placa Arduino empleada.
- *"Loop ()"*: Se trata del bucle de programación que se repetirá cíclicamente y determina la ejecución.

Lenguaje de programación

Como ya se ha comentado el lenguaje de programación es muy aproximado al C++ con lo que a continuación se detallan las declaraciones más comunes empleadas en dicha programación.

Declaración de Variables

Variables		
Concepto	**Nomenclatura**	**Descripción**
Booleana	Bool	Generalmente 1 Byte de memoria
Carácter	Char	Generalmente 1 byte 256
Entero sin signo	Unsigned short int	2 Bytes (valores desde 0 a 65535)
Entero con signo	short int	2 Bytes (valores desde -32768 a 32767)
Entero sin signo largo	unsigned long int	4 Bytes (valores desde 0 a 4 294 967 295)
Entero con signo largo	long int	4 Bytes (valores desde -2 147 483 648 a 2 147 483 648)
Doble	Double	64 Bbits (valores desde 5e-324 a 1,7e-308
Coma flotante	Float	32 bits (valores desde -3,4e38 hasta 3,4e38)

Tabla 2.1. Tipos de variables

En la tabla superior aparece la definición de las variables típicas empleadas en la programación.

Sintaxis Básica

Sintaxis Básica	
Concepto	**Nomenclatura**
Cabeceras	#define #include
Comentarios	// /**/
Delimitadores	; {}
Operaciones Aritméticas	- * / % +
Asignación	=
Comparativos	< > <= >= ==
Booleanas	&& || !
Operadores de bits	& | ^ << >>
Compuestos	!= ++ -- += -= *=

Tabla 2.2. Sintaxis Básica

El empleo de las cabeceras, permite incluir determinadas librerías para aderir diferentes programas escritos en lenguajes diferentes. Del mismo modo, como hemos comentado al tratarse de software libre, permite adquirir traducciones ya trabajadas con anterioridad sin necesidad de generar nuevas.

Estructuras de Control

Estructuras de Control	
Concepto	**Nomenclatura**
Condicionales	if if…else switch case
Saltos	for while do…while
Bucles (Cíclicos)	goto return break

Tabla 2.3. Estructuras de Control

Funciones Básicas

Funciones Básicas		
Concepto	**Nomenclatura**	**Descripción**
E/S Analógica	analogReference()	Configuración tensión referencia de entrada
	analogRead()	Lectura de señal analógica
	analogWrite()	Escritura en Pin analógico
E/S Digital	pin Mode ()	Definición del comportamiento del pin como E/S
	digitalWrite()	Escritura en Pin Digital
	digitalRead()	Lectura de señal digital
Tiempo	delay()	Definición de retardo del bucle
	delayMicroseconds()	Definición de retardo del bucle microsegundos
Matemáticas	min(x, y)	Calcula el mínimo de 2 números
	max(x, Y)	Calcula el máximo de 2 números
	abs(x)	Devuelve el valor absoluto
	sqrt(x)	Devuelve la raiz cuadrada de un número
	constrain(x, a, b)	Restricción a un rango
Trigonométricas	sin(rad)	Calcula el Seno
	cos(rad)	Calcula el Coseno
	tan(rad)	Calcula la tangente
Aleatoriedad	randomSeed()	Genera números aleatorios con semilla misma secuencia
Administracion Bits	lowByte()	Extrae el Byte inferior (derecha)
	highByte()	Extrae el Byte superior (izquierda)
	bitRead()	Lectura de un cierto número de bits
	bitWrite()	Escritura de un cierto número de bits
	bitSet()	Establece un valor 1 a un bit concreto
	bitClear()	Establece un valor 0 a un bit concreto
	bit()	Calcula el valor de un bit específico
Interrupciones	attachInterrupt()	Activación Interrupción
	detachInterrupt()	Desactivación Interrupción

Tabla 2.4. Funciones Básicas

2.2 ENTORNO LabView

LabView (Laboratory Virtual Instrumentation Engineering Workbench) es una plataforma de software cerrado que emplea un entorno de programación de tipo gráfico. Es una plataforma ideal para la adquisición, control, análisis, tratamiento y presentación de datos.

La programación gráfica (tipo G) de la plataforma, aporta determinados beneficios que no son tan evidentes en otro tipo de programaciones. Principalmente son las siguientes:

- Tiempo inferior para la formación en la programación. Está demostrado que la interfaz gráfica es más ágil a la hora de aprender el lenguaje y consecuentemente en desarrollar aplicaciones.
- Ordenación de los algoritmos desarrollados. El grafismo otorga una visión más realista del flujograma natural de las diferentes partes en la programación.
- Mejora en los tiempos de desarrollo. Vinculado a los puntos anteriores, el tiempo de desarrollo de aplicaciones es menor.
- Gestión de los datos. El entorno de LabView es tremendamente útil en el análisis y conversión de los datos. Al estar diferenciados también de manera visual (diferentes formas, cromática, bloques…) son fácilmente reconocibles.
- El tratamiento de los datos permite obtener ficheros legibles por otras plataformas software como hojas de cálculo…

En ocasiones y dependiendo del perfil del programador, que la herramienta sea tan evidente en su programación, puede generar desconocimiento en el desarrollo de aplicaciones complejas. El programador puede abstraerse de tal modo que desconozca lo que internamente hacen los bloques.

2.2.1 PROGRAMACIÓN EN LABVIEW

Como ya se ha comentado anteriormente, LabView es un entorno de programación gráfico. Los programas no se escriben, sino que se dibujan como diagramas de bloques en sistemas secuenciales, agilizando de esta manera su comprensión.

Existen una gran cantidad de bloques pre-diseñados que permiten la creación del proyecto de manera fluida, con lo que el usuario invierte mucho menos tiempo en programar.

Este hecho, permite al diseñador invertir más tiempo en el diseño y desarrollo de la interfaz gráfica con la que va a interactuar el usuario final.

Los programas generados en LabVIEW, se denominan "*Instrumentos Virtuales" (**VI**)*, porque su apariencia y funcionamiento imitan los de un instrumento real. Sin embargo, son análogos a las funciones creadas con los lenguajes de programación convencionales.

En la figura 2.3. aparece representado un VI completo con sus dos pantallas correspondientes.

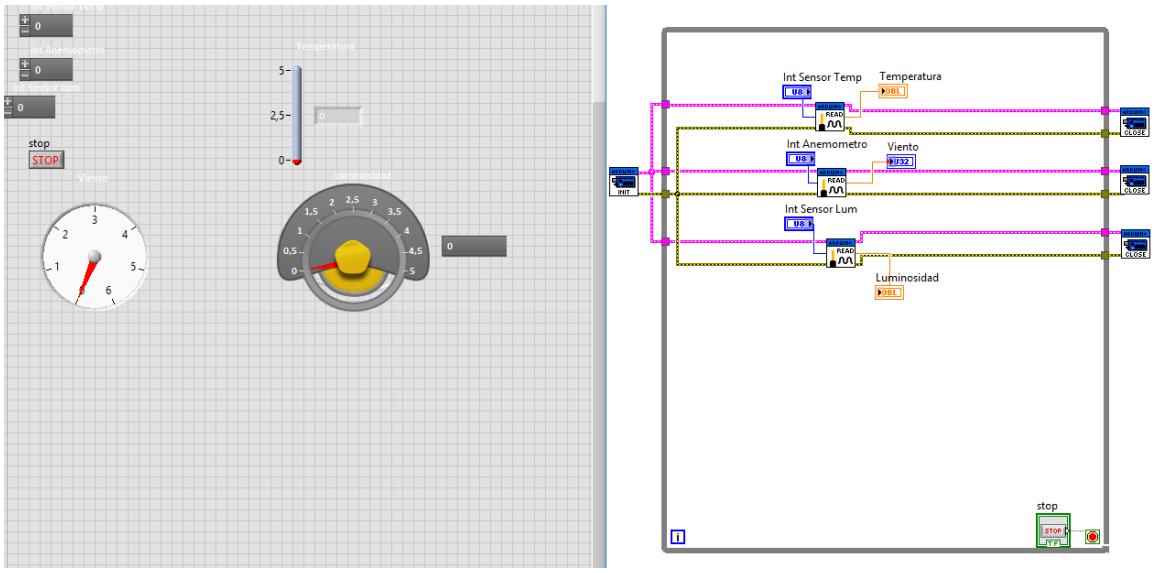

Figura 2.3. Estructura VI LabView

Como se puede observar, la interfaz consta de dos pantallas diferenciadas. La de la parte izquierda es la denominada *"Panel Frontal"* y la de la derecha *"Diagrama de Bloques"*. Las características de cada una de ellas son las siguientes:

- Panel Frontal → Se trata de una interfaz gráfica muy útil para el usuario. En ella se observan las variables implementadas en el diagrama de bloques y su comportamiento durante la simulación. Como se puede observar, existen diferentes gráficos que permiten simular indicadores analógicos y digitales. Empleando el denominado LabView Real Time, sería posible implementar simulaciones en tiempo real útiles en determinadas aplicaciones.

- Diagrama de Bloques → Se trata del código fuente del Instrumento Virtual. En esta ventana gráfica es donde se realiza la implementación del programa del *VI* para controlar o realizar cualquier procesado de las entradas y salidas. Los controles e indicadores que se visualizan en el panel frontal, se materializan en el Diagrama de Bloques mediante el empleo de determinados terminales. Realizando la ordenación de los bloques se obtiene el programa final mostrado. Como se puede observar, existen diferentes bucles para la ejecución, esto permitirá realizar diferentes estructuras.

Dentro del diagrama de bloques, existen diferentes paletas para realizar la programación. Cada una de ellas tiene un ámbito y de manera general estas serían las más representativas.

Paleta de Controles

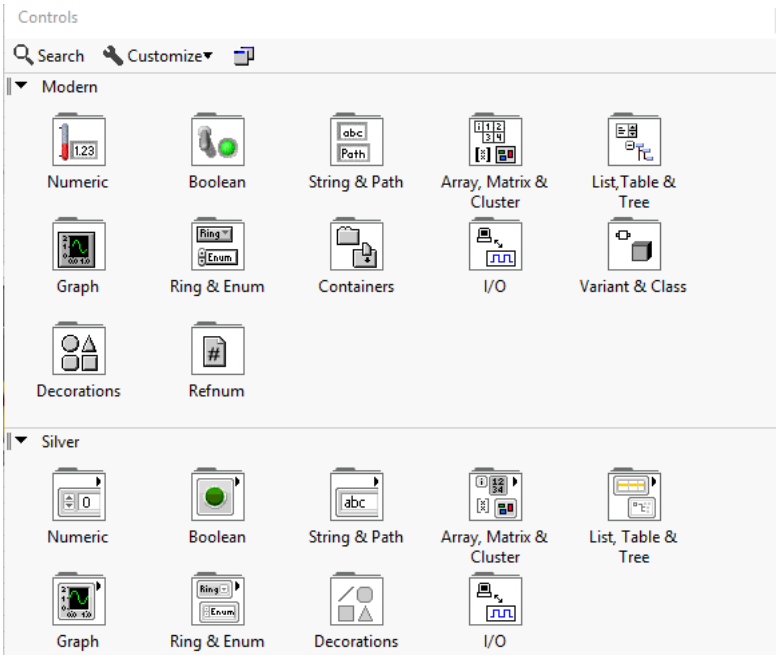

Figura 2.5. Paleta de controles

Es una paleta muy completa a través de la cual se definen indicadores para el panel frontal. Como se puede observar en la imagen, se pueden representar variables de tipo booleano, numéricas, digitales, analógicas etc.

Paleta de Funciones

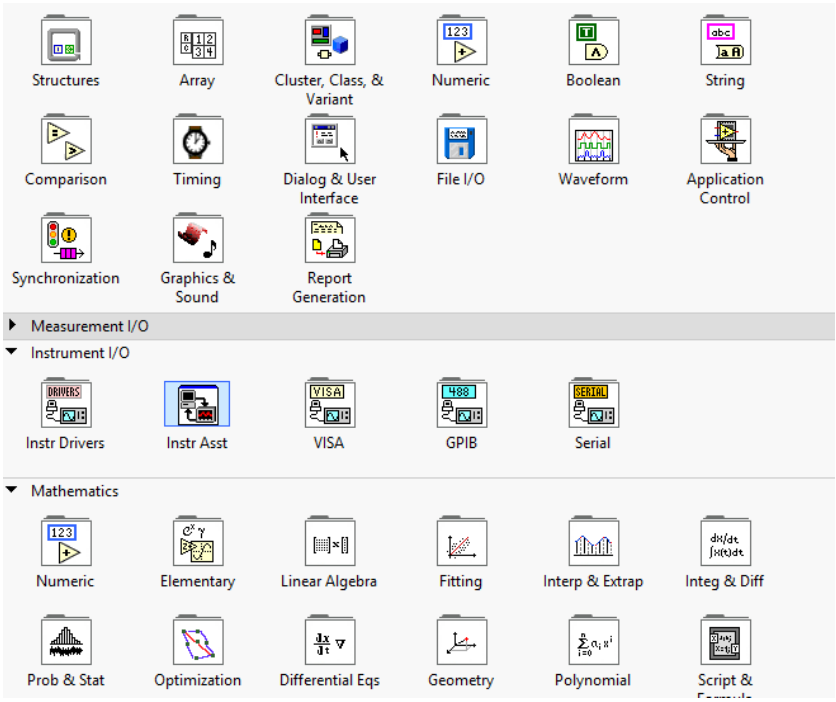

Figura 2.6. Paleta de funciones

La paleta de funciones es de utilidad en la pantalla del diagrama de bloques. Con ella se generan los bloques de programación necesarios para visualizarlos posteriormente en la pantalla del instrumento virtual. Existe cierta vinculación entra ambas pantallas a través de la generación de los indicadores. Las principales funciones que se pueden implementar son de tipo aritméticas, gestión de entrada/salida de señales, adquisición de datos, temporización de ejecución del programa, etc.

En esta paleta aparecerán los bloques para la adquisición a través de Arduino. En el apartado programación detallaremos como se obtienen estos bloques y sus principales aplicaciones.

Paleta de Herramientas

Figura 2.7. Paleta de herramientas

La paleta de herramientas es de uso general en ambas pantallas, tanto en el diagrama de bloque como en el instrumento virtual. Las principales utilidades que otorgan los diferentes puntos de esta paleta son:

- *"Operating tool":* cambia el valor de los controles.
- *"Positioning tool"*: desplaza, cambia de tamaño y selecciona objetos.
- *"Labeling tool"*: edita texto y crea etiquetas.
- *"Wiring tool"*: une los objetos en el diagrama de bloques.
- *"Object Pop-up Menu tool"*: abre el menú desplegable de un objeto.
- *"Scroll tool"*: desplaza la pantalla.
- *"Breakpoint tool"*: fija puntos de interrupción en la ejecución del programa en *VIs*, funciones y estructuras.
- *"Probe tool"*: crea puntos de prueba en los cables, donde se visualiza el valor en cada instante.
- *"Color Copy tool"*: copia el color seleccionado.
- *"Color tool"*: establece el color de fondo y el de los objetos.

3 CARGA DE LIBRERÍAS NECESARIAS PARA LA COMUNICACIÓN

3.1 DESCARGA SOFTWARE NECESARIO

Los pasos a seguir para el establecimiento de la comunicación aparecen detallados a continuación. No se excluye ninguno de los pasos y se parte de la descarga incluso del propio compilador de Arduino de la web oficial. Por lo tanto:

1. **Descargar e Instalar el entorno de programación de Arduino 1.8.5 de la página web oficial.**

   En una primera aproximación que realizamos en la asignatura de instrumentación virtual, tuvimos problemas de comunicación con las versiones del compilador posteriores a la 1.0.5. Actualmente se han subsanado con lo que se podrá emplear la última versión disponible.

El compilador necesario para programarlo está disponible de forma gratuita en www.arduino.cc y está disponible para Mac OS X, Windows y Linux.

Figura 3.1. URL descarga Software Libre Arduino.

La dirección URL para la descarga es la siguiente:

https://www.arduino.cc/en/Main/Software

Instalar el software de Arduino junto con los drivers en *C\Archivos de Programa*. Después conectar el Arduino al PC ignorando el asistente para agregar nuevo hardware. Ir al administrador de dispositivos y buscar el Arduino, que aparece como dispositivo desconocido.

Pinchar con el botón derecho y actualizar el controlador. Buscar el controlador manualmente en el directorio donde se ha instalado el software Arduino, seleccionar la carpeta "drivers" y aceptar.

Si los drivers de Arduino están instalados correctamente, en el administrador de dispositivos aparecerá el Arduino como un puerto COM seguido de un número entero (Normalmente el número más alto).

2. **Instalar VI Package Manager de la página web de JKI:**

Figura 3.2. VI Package Manager

La URL para llevar a cabo la descarga del software aplicado es la siguiente:

http://jki.net/vipm/download

Descargar y ejecutar el archivo de instalación. Se procede a la instalación del VI Package Manager.

3. **Abrir el VI Package Manager.**

   Se cargará una lista larga de dispositivos cuyos drivers se pueden instalar. Se selecciona ***"LabVIEW Interface forArduino"*** y se instala.

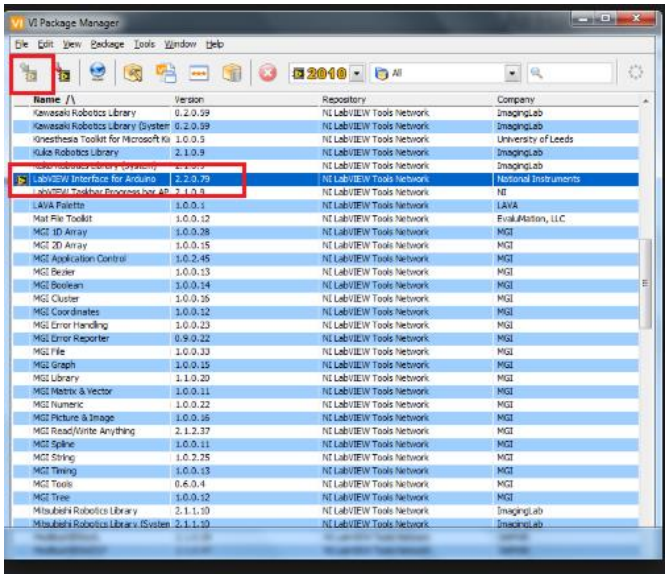

Figura 3.3. Pantalla selección Software VIPM

Una vez finalizada la instalación, se habrán cargado las funcionalidades nuevas en LabView para la programación del dispositivo. En la pantalla del diagrama de bloques del VI se habrá generado, en la paleta de funciones, un módulo de Arduino con las diferentes funcionalidades. En la figura 4.4 se puede apreciar el detalle.

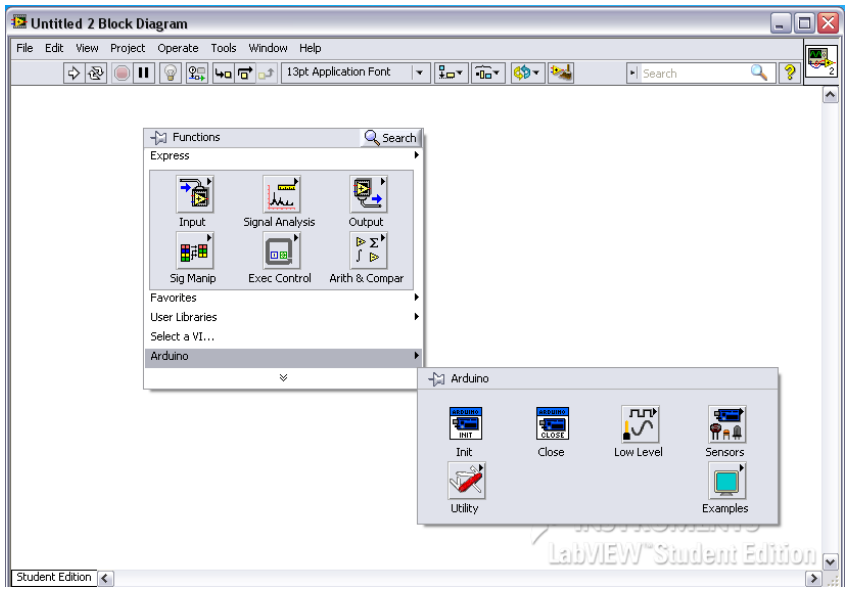

Figura 3.4. Módulo Arduino en LabView

4. Instalar NI-VISA 18.0 (Virtual Instruments Software Arquitecture).

A fecha de redacción del presente documento, la versión más actual de la librería VISA se corresponde con la versión 18. Para que la compatibilidad sea correcta, se deberá tener instalado la versión más actual de LabView. La URL de descarga es la siguiente:

http://www.ni.com/es-es/support.html

5. Cargar en Arduino las librerías para la comunicación con Labview

Abrir el compilador de Arduino y abrir el archivo "LIFA_Base", que se ha creado al instalar las librerías VISA en el siguiente directorio:

C\Archivos de programa\National Instruments\Labview 2018\vi.lib\Labview Interface forArduino\Firmware\LIFA_Base.

Esta librería será la encargada de hacer que el microcontrolador realice la comunicación serie con el entorno de LabView. (Se detallará el programa a continuación). En la figura 4.5 aparece el detalle del programa cargado.

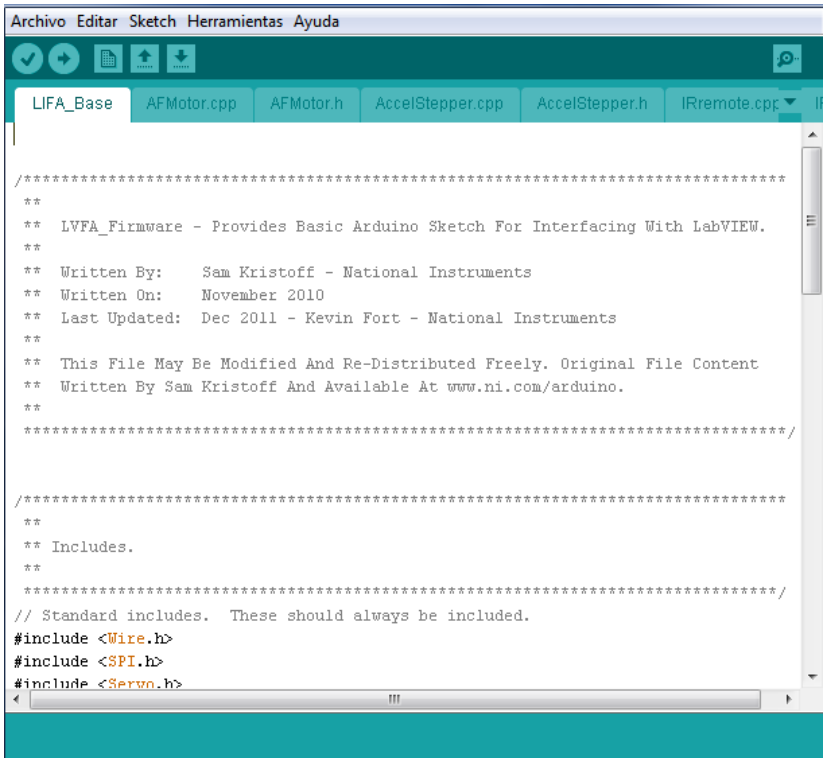

Figura 3.5. Librería LIFA Arduino

4 PROGRAMACIÓN MICROCONTROLADOR

Como se ha explicado con anterioridad, se carga la librería Lifa Base.Ino en el Arduino para establecer la comunicación correspondiente con LabView. Vamos a ver en profundidad la ejecución del programa que acciones contempla en el microcontrolador una vez cargado.

4.1 PROGRAMA MAIN

El código del programa principal es el siguiente:

```
#include <Wire.h> "Librería que permite comunicarse con dispositivos I2C / TWC. En el caso del Arduino Uno los pines correspondientes serían → A4(SDA), A5(SCL)"
#include <SPI.h> "Librería que permite la comunicación con dispositivos vía SPI puerto serie. En el caso que nos aborda será la metodología que emplearemos en la comunicación"
#include <Servo.h> "Librería que permite a Arduino el control de los servomotores de tipo RC"
#include "LabVIEWInterface.h" "Librería que se explica luego para la comunicación"
void setup ()
{   "Inicialización del Puerto serie con la frecuencia predeterminada"
 syncLV ();
}

 "El bucle principal, se ejecuta constantemente en Arduino. Recibe y procesa los comandos procedentes de LabView constantemente"
void loop ()
{  "Analiza  los comandos recibidos de LabView y los procesa"
 checkForCommand ();
 if(acqMode==1)
 {
  sampleContinously ();
 }
}
```

4.2 LIBRERÍA LIFA BASE COMUNICACIÓN

```
"LVFA_Firmware - Proporciona funciones para interactuar con el Arduino Uno"
"Definir constantes"
"Definir directivas que proporcionan nombres significativos para valores constantes"
# define FIRMWARE_MAJOR  02
# define FIRMWARE_MINOR  00
"si está definido (__ AVR_ATmega1280__) || definido (__ AVR_ATmega2560__)"
# define DEFAULTBAUDRATE  9600   "Define la velocidad en baudios predeterminada (Esto debe coincidir con la velocidad especificada en baudios en LabVIEW)"
# else
# define DEFAULTBAUDRATE  115200
# endif
# define MODE_DEFAULT  0 "Define modos Arduino (actualmente no utilizados)"
# define COMMANDLENGTH  15 "Define el número de bytes en un solo comando LabVIEW (Esto debe coincidir con el tamaño del paquete especificado en LabVIEW)"
# define STEPPER_SUPPORT  1 "Define si la biblioteca Stepper está incluida"
"Declarar variables"
 unsigned char currentCommand [COMMANDLENGTH]; "El comando actual para que Arduino procese"
"Globales para adquisición continua"
unsigned char acqMode;
unsigned char contAcqPin;
float contAcqSpeed;
float acquisitionPeriod;
iteraciones flotantesFlt;
int iteraciones;
float delayTime;
"Sincroniza con LabVIEW y envía información sobre la placa y el firmware"
void syncLV ();
"Establece el modo del Arduino (reservado para uso futuro)"
void setMode (int mode);
"Comprueba si hay nuevos comandos de LabVIEW y los procesa si existen"
int checkForCommand (void);
"Procesa un comando dado"
void processCommand (unsigned char command []);
"Escriba valores en los pines DIO 0 - 13. Los pines primero deben configurarse como salidas"
void writeDigitalPort (unsigned char command []);
"Lee los 6 puertos de entrada analógica, construye paquetes de 8 bytes y los envía a través de RS232"
void analogReadPort ();
"Configuración de los pines E / S digitales para usar en la visualización de siete segmentos. Los pines se almacenan en la matriz sevenSegmentPins."
void sevenSegment_Config (unsigned char command []);
"Escribe valores en la visualización de siete segmentos. Primero debe usar sevenSegment_Configure"
void sevenSegment_Write (unsigned char command []);
"Establecer el divisor de reloj SP 2,4,8,16,32,64,128l"
void spi_setClockDivider (unsigned char divider);
 "Este comando envía un byte serie a LV para cada byte de datos"
```

```
void spi_sendReceive (unsigned char command []);
"Valor de suma de comprobación de Char"
unsigned char checksum_Compute (unsigned char command []);
"Calcular suma de comprobación de paquetes y prueba contra suma de comprobación incluida"
int checksum_Test (unsigned char command []);
"Escribe la velocidad, dirección y número de pasos para viajar"
void AccelStepper_Write (unsigned char command []);
"Devuelve varios puntos de entrada analógica a la vez."
void sampleContinously (void );
"Devuelve el número de muestras especificadas a la velocidad especificada"
void finiteAcquisition (int analogPin, float acquisitionSpeed, int numberOfSamples);
"Imprime datos en la pantalla LCD con la base dada"
void lcd_print (unsigned char command []);
```

La comunicación se establece en pasos. El Arduino en un primer paso de ejecución se conecta a través del puerto serie con LabView. Dentro de LabView corre el programa que definiremos a continuación y que establece de qué manera se van a hacer las lecturas y a través de que puertos.

LabView otorga la respuesta de la configuración de los puertos que va a emplear y como. Del mismo modo se establece también la velocidad de comunicaciones. (Sino se modifica el parámetro correspondiente a la velocidad de comunicaciones, esta empleará el valor definido por defecto) Ej. El puerto 0 (analógica) se va a emplear como entrada de adquisición del dato del sensor de temperatura 1.

Arduino recibe la información de la configuración del puerto y tras configurarlo responde confirmando la configuración descrita si esta está bien definida. Tras la comprobación, LabView solicita el envío de la primera medición correspondiente a esa entrada. Arduino recibe la petición y adquiere y procesa la señal de la entrada correspondiente. Una vez hecho, envía el primer dato para el procesamiento en LabView. El programa se ejecutará hasta darle fin de manera cíclica como se ha descrito en la programación. En el diagrama de flujo adjunto como figura 4.1 se observa el comportamiento del programa para el ejemplo descrito de una manera muy sencilla.

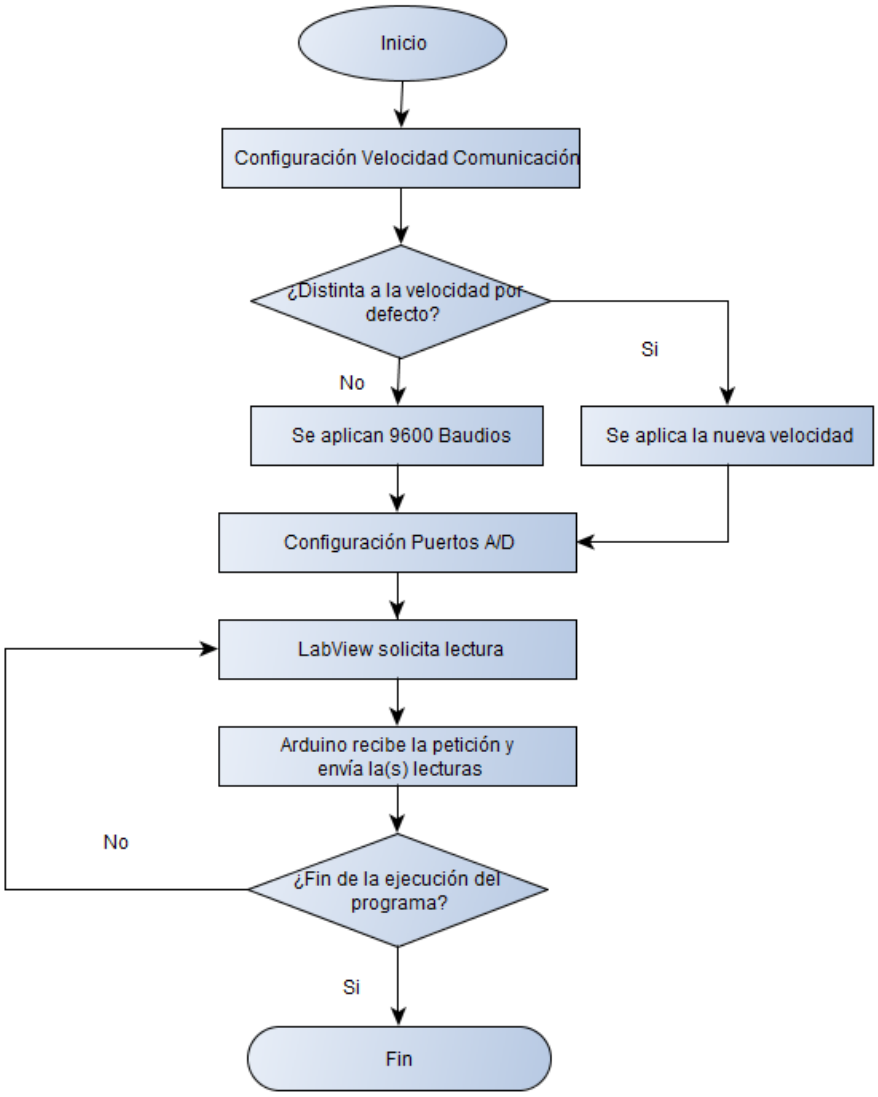

Figura 4.1. Diagrama de flujo conexión serie Arduino LabView

5 IMPLEMENTACIÓN DEL PROGRAMA LABVIEW

5.1 DIAGRAMA DE FLUJO ADQUISICIÓN EN LABVIEW

El programa final elaborado en LabView para la adquisición de las señales de los sensores obedece al diagrama de flujo que aparece en la figura 5.1.

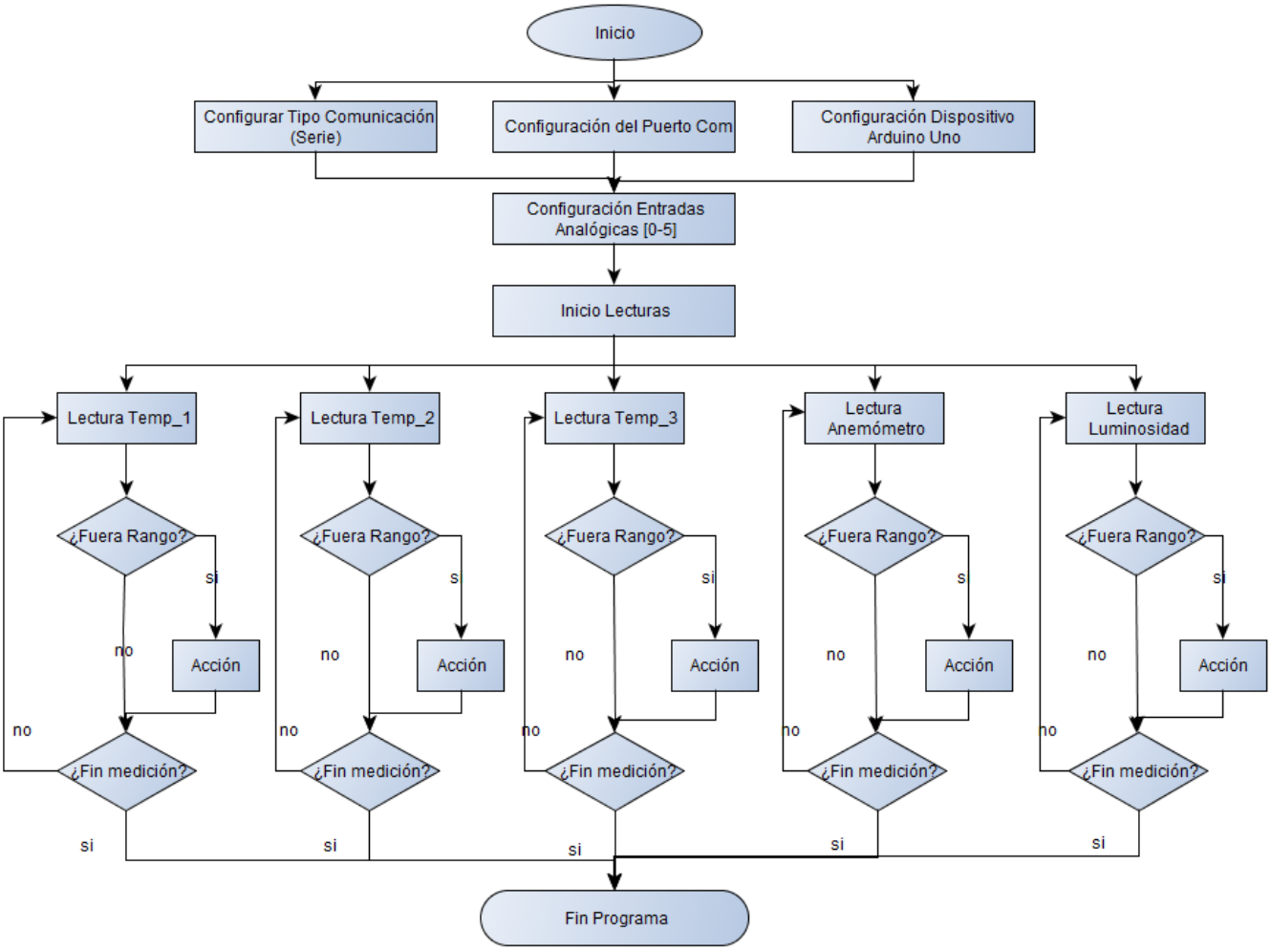

Figura 5.1. Diagrama Flujo LabView

Como se puede observar en el flujograma, una vez que se inicia el programa, lo primero que se hace es configurar el puerto de comunicación a través del cual se conecta el microcontrolador (puerto COMX). A la vez también se realiza la configuración del tipo de conexión y el dispositivo. En el apartado 5.2 se observará como se realizan las configuraciones con la programación modular de LabView.

El siguiente punto es la configuración de los puertos empleados para la adquisición de datos. En este caso se emplearán las entradas analógicas de [0-5] a través de las cuales se adquieren las señales de cada uno de los sensores. Como se vio en los cálculos Hardware, el rango de entrada de las señales nunca excede de 5V para no dañar el dispositivo.

Una vez configuradas las entradas (acción que solamente se ejecutará una vez), se inicia el proceso de lectura de las señales. En este caso y para simplificar el diagrama de flujo completo, no se han pintado todas las acciones que lleva a cabo el programa por lo que a continuación se adjunta un flujograma un poco más pormenorizado de la parte de la lectura. Se diferencian 3 apartados (lectura temperatura, lectura anemómetro y lectura sensor de luminosidad).

5.2 DIAGRAMA DE FLUJO LECTURA SENSORES

5.2.1 LECTURA TEMPERATURA

Existen 3 sensores de temperatura redundantes (2 cableados y 1 vía XBee) pero a efectos de programación se han acotado de la misma manera. El diagrama de flujo de la programación correspondiente a las lecturas de temperatura aparece en la figura 5.2.

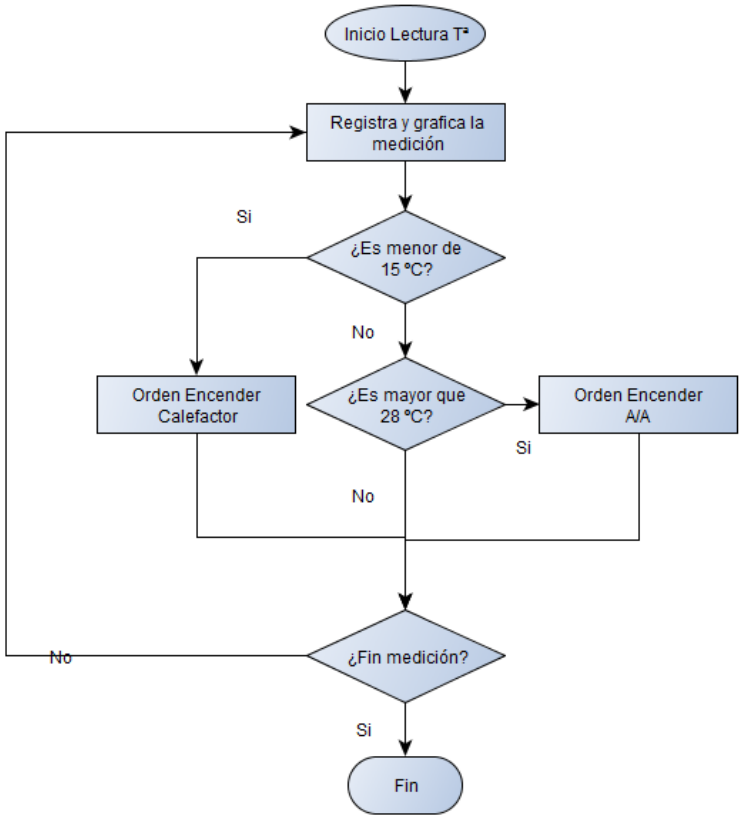

Figura 5.2. Flujograma lectura termómetros

Como se observa en el flujograma, el programa compara dos consignas diferentes sobre las cuales tiene dos posibles actuaciones. Por un lado, sería encender el climatizador para temperaturas superiores a los 28 grados centígrados y por el otro la calefacción para lecturas inferiores a 15 grados centígrados. En la realidad, estas consignas que ejecutan ese cambio de estado podrían alimentarse con una salida del micro que accionara el relé de conexión de la calefacción/refrigeración a modo de termostato. Mientras se ejecuta el programa, los registros de temperatura se grafican como se detallará posteriormente en la explicación de la programación modular.

5.2.2 LECTURA ANEMÓMETRO

En el caso de la lectura del anemómetro, solo existe una consigna que limita la acción. Si el viento en el entorno sobrepasa cierta velocidad (50 km/h), se ejecutará la acción de retraer el toldo para el sol que se encuentra en la fachada del edificio. Al igual que en el caso anterior, a futuro se podría implementar una salida que alimentara el motor del toldo.

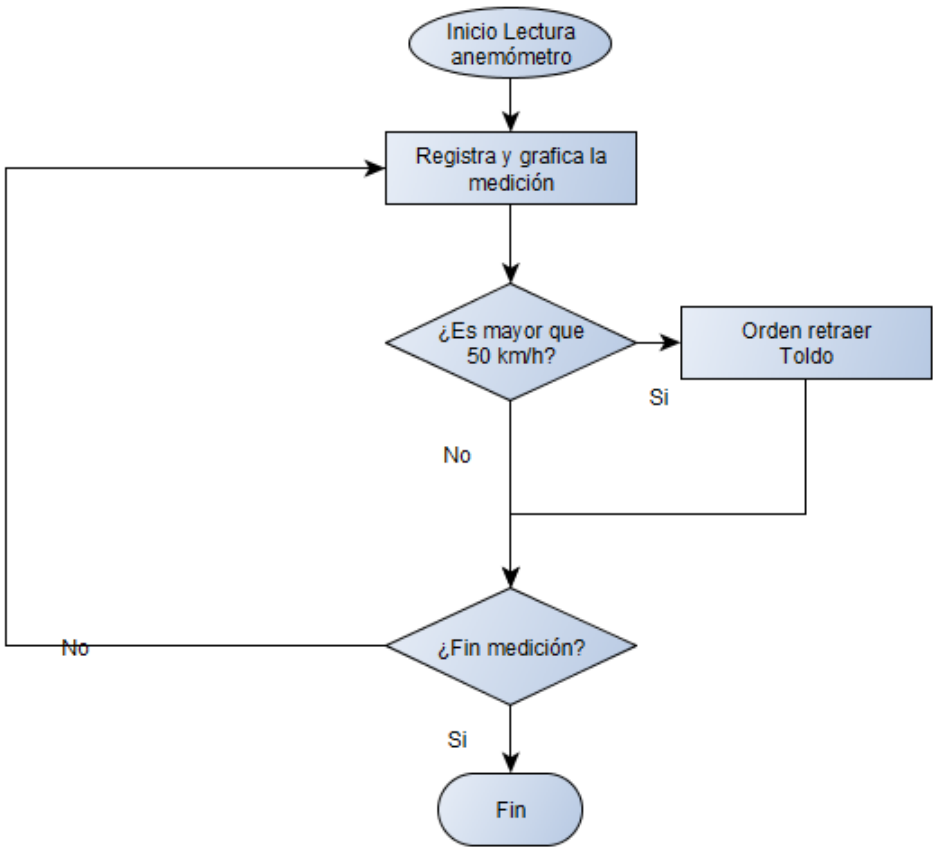

Figura 5.2. Flujograma lectura anemómetro

5.2.3 LECTURA SENSOR LUMINOSIDAD

En el caso de la lectura del sensor de luminosidad al igual que en el anemómetro, solo existe una consigna que limita la acción. Si la luminosidad exterior baja de un grado porcentual calculado empíricamente (no se basa en una variable física real ya que el grado de luminosidad se ha calculado desde 0 a 100% del rango completo que abarcaba el sensor implementado.

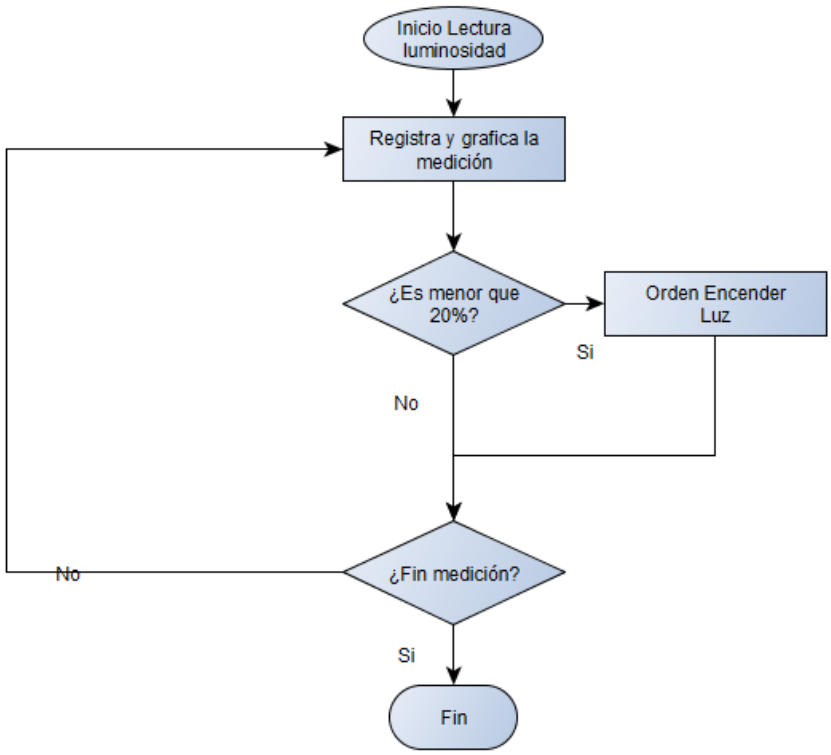

Figura 5.3. Flujograma lectura sensor luminosidad

Al igual que en los anteriores, a futuro se podría generar un actuador que habilite el encendido de las luces cuando sobrepase el rango establecido.

5.3 PROGRAMACIÓN MODULAR EN EL ENTORNO LABVIEW

5.3.1 PROGRAMA COMPLETO

El programa completo diseñado en la pantalla referida al diagrama de bloques aparece en la figura 5.4. En él se pueden apreciar todas las partes que se han designado en los flujogramas del apartado anterior.

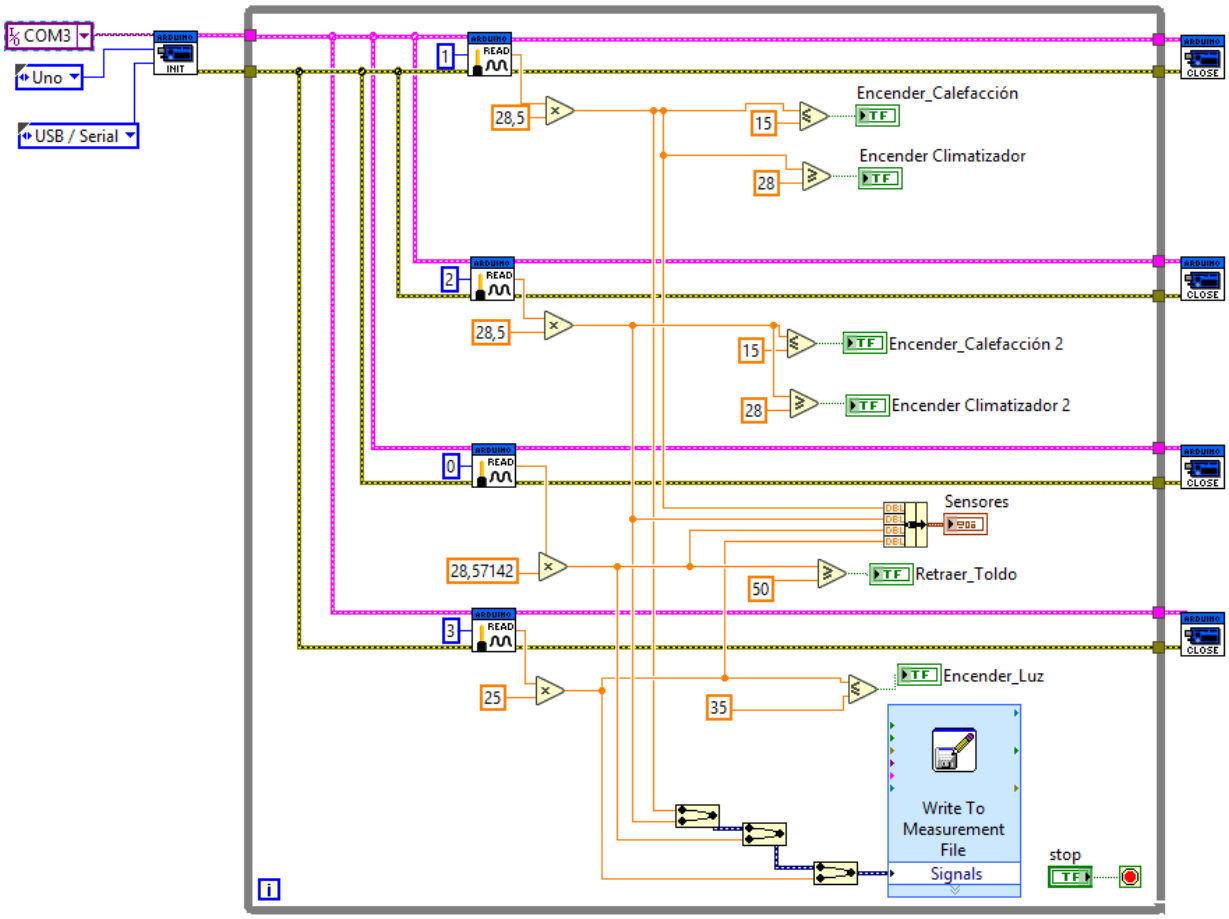

Figura 5.4. Diagrama de Bloques Completo

Al igual que se ha hecho en los flujogramas, a la hora de realizar la explicación de la programación, se va a ir paso por paso y bloque a bloque.

5.3.2 BLOQUE INIT.VI

El bloque Init.VI es el encargado de iniciar la comunicación entre LabVIEW y Arduino. Como aparece reflejado en la figura, se deberá configurar el puerto de comunicación serie utilizado (en este caso el COM3), el tipo de tarjeta empleada en la adquisición de datos (Arduino Uno) y el tipo de conexión.

A más se podría configurar la frecuencia de comunicación, pero por defecto ya aparece la más rápida con lo que no hay que modificarla.

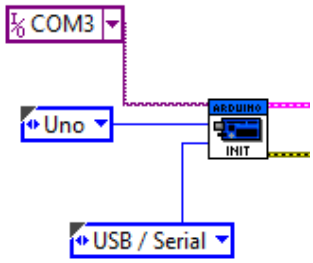

Figura 5.5. Bloque Init.VI

Como se puede observar en la figura, existen dos cables de conexión de color rosa y amarillo. El amarillo es el control de errores en la comunicación (se ha explicado en la librería de Arduino) y el rosa es el establecimiento de la comunicación.

5.3.3 BLOQUE CLOSE.VI

El bloque Close.VI cierra la conexión con Arduino. El establecimiento del cierre de la conexión permite cerrar el ciclo de comunicación y así reiniciar el bucle while general. Si no se hiciera el programa seguiría pensando que está conectado y no se podría volver a ejecutar.

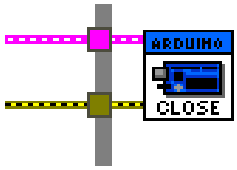

Figura 5.6. Bloque Close.VI

A diferencia del bloque Init, el Close se hace por cada línea de adquisición de datos. Como ya se explicó en el algoritmo, LabView envía masivamente los datos de todas sus entradas analógicas / digitales a la vez. Por lo tanto, para cerrar el ciclo se ha de cerrar cada una de las líneas de comunicación. No tiene ningún ajuste relevante y como se observa en la figura posee conexión a las líneas de detección de error y de comunicación.

5.3.4 BLOQUE ANALOG READ

El bloque analog read, es el encargado de tomar las muestras que adquiere el Arduino a través de sus entradas. En cada ciclo completo de programa recibe una muestra por cada una de las entradas. Estas se ejecutan en paralelo por cada ciclo y cada una de ellas procesa la información correspondiente en los bloques posteriores que veremos a continuación.

Figura 5.7. Bloque Analog Read

Como se puede ver en la figura 5.7, en la configuración de los módulos de lectura analógica, se define una constante que refleja el pin de la entrada correspondiente. En los ejemplos que aparecen, van de [0-3] y cada una recibe la información correspondiente al sensor cableado en cada entrada.

Por otra parte, también tienen conectadas virtualmente las líneas de error y comunicación correspondientes a cada una de ellas.

Por último, aparece un conector naranja en la parte derecha de cada uno de ellos. En LabView la cromática de los cables indica el tipo de variable que transportan y en el caso del color naranja se trata de un valor de tipo float. Se trata del voltaje de entrada que se obtiene a través de las entradas analógicas. Esta variable como se verá a continuación, va conectada a unos operadores para cada una de las entradas que permitirán visualizar las gráficas en el Front Panel.

5.3.5 BLOQUE MULTIPLICADOR

El bloque multiplicador realiza la multiplicación escalar entre dos variables de tipo float independientes. En el caso del proyecto desarrollado, se emplea para acomodar la señal de entrada (que es una tensión variable en rangos diferente) al dato que se quiere mostrar gráficamente.

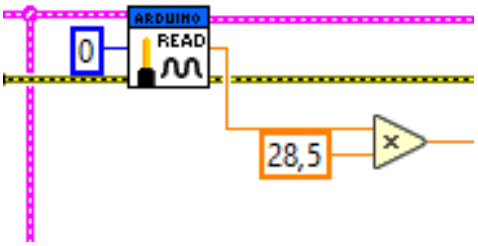

Figura 5.8. Bloque Multiplicador

Las conversiones por cada sensor son diferentes por lo que se han de implementar diferentes multiplicaciones escalares para cada uno. Como se observa en la figura 5.8, una de las entradas viene del bloque Analog Read (salida del voltaje Float) y la otra es un escalar con el factor de multiplicación necesario. Los factores de multiplicación por cada una de las entradas se exponen en la tabla adjunta.

Entrada	Sensor	Rango_Entrada [V]	Rango_Salida_Deseado	Factor_Multiplicador
0	Temperatura 1	[0 - 1,75]	[0 - 50] °C	28,5 °C/V
1	Temperatura 2	[0 - 1,75]	[0 - 50] °C	28,5 °C/V
2	Anemómetro	[0 - 3,5]	[0 - 100] km/h	28,57 (km/h) /V
3	Luminosidad	[0 - 4]	[0 - 100] %	25
4	Temperatura 3	[0 - 1,482]	[0 - 60] °C	40,48 °C/V

Tabla 5.1. Factores de multiplicación por entrada

5.3.6 BLOQUE COMPARADOR

El bloque comparativo permite, valga la redundancia, comparar dos valores de tipo float y otorgar una salida Booleana otorgando el resultado. En el caso de ser afirmativo, la salida otorgada por el bloque será un *"Verdadero"*, en caso contrario devolverá un *"Falso"*. En la figura adjunta, 5.9, se representa la comparativa que realiza para la adquisición de los datos sensor de temperatura 1.

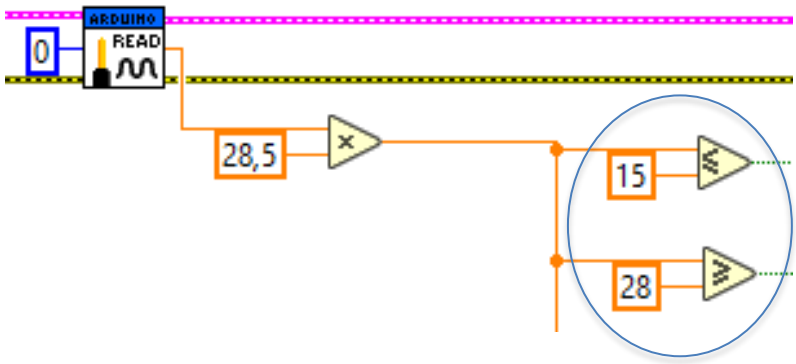

Figura 5.9. Bloque Comparador

En el caso de la primera comparativa, toma la lectura de la entrada analógica 0 multiplicada por el factor mostrado anteriormente. A este valor de tipo float, le hace dos comparativas diferentes que se acotan posteriormente. El primero otorga un verdadero cuando la lectura es igual o inferior a 15 ℃ y la segunda cuando es igual o superior a 28 ℃.

Estas comparativas permiten realizar acciones posteriores dependiendo del dato comparado como se verá en el siguiente apartado. Si la temperatura es inferior a 15 ℃ se dará la acción de encender la calefacción. Si por el contrario es superior a 28 ℃ se dará la orden de activación del climatizador.

En la tabla 5.2, aparecen las comparativas para cada una de las entradas. Dependiendo del tipo de sensor se establecen unos limitantes para ejecutar las acciones.

Entrada	Sensor	Rango_Mínimo	Rango_Máximo	Acción
0	Temperatura 1	15 °C	28 °C	Encender Calefacción/Climatización
1	Temperatura 2	15 °C	28 °C	Encender Calefacción/Climatización
2	Anemómetro	-	50 km/h	Recoger Toldo
3	Luminosidad	35%	-	Encender Luces
4	Temperatura 3	15 °C	28 °C	Encender Calefacción/Climatización

Tabla 5.2. Valores Comparativos por Sensor

5.3.7 BLOQUE REPRESENTACIÓN BOOLEANO

Una vez explicado el bloque comparativo, el siguiente paso es la representación en este caso del resultado de la comparativa. Como se ha comentado anteriormente, el bloque comparador otorga un *"Verdadero"* cuando se cumple la condición en la entrada del bloque. Pues bien, una manera de representar el resultado es mediante el bloque Boolean. En el diagrama de bloques, la representación es la que aparece a continuación.

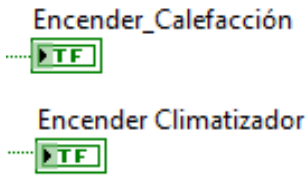

Figura 5.10. Bloque Boolean

Como se observará en el análisis de los resultados, en la pantalla de explotación se genera un led de confirmación, que en el caso de ser verdadera la comparativa, se encenderá visualmente.

5.3.8 BLOQUE BUNDLE

A la hora de mostrar los datos en la pantalla de explotación, resulta interesante realizar gráficas por cada una de las entradas. Cuando simplemente se quiere representar una gráfica, se añade el bloque de representación gráfica que se explica a continuación y listo. Pero en el caso del proyecto presente, se dispone de 5 entradas simultaneas cuyo comportamiento se desea ver al mismo tiempo.

Para evitar tener que emplear un bloque gráfico por cada entrada y con el fin de aunar los gráficos en una representación combinada más acorde, se emplea el bloque Bundle.

Figura 5.11. Bloque Bundle

Como se puede observar en la figura 5.11, a la entrada del bloque hay 4 variables float que entran por cada uno de los 4 canales diferenciados y se extrae un único hilo. El hilo marrón y de representación gruesa como aparece obedece al tipo forma de onda, ya que se unirá a posteriori a un bloque de representación gráfica.

5.3.9 BLOQUE REPRESENTACIÓN GRÁFICA

El bloque de representación gráfica, constituye la opción de obtener una representación gráfica de las señales en la pantalla de explotación. Como se observa en la figura 5.12, la salida del bloque Bundle se conecta a la entrada del bloque presente.

Figura 5.12. Bloque Representación Gráfica

5.3.10 BLOQUE DE ESCRITURA A FICHERO DE MEDIDA

El bloque de escritura a un fichero de medida, es el encargado de generar el fichero para la lectura en formato hoja de cálculo.

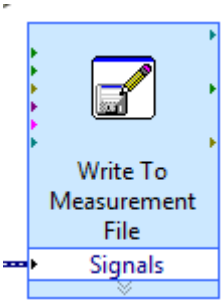

Figura 5.13. Bloque Escritura a Fichero de medida

Como se puede observar existe una entrada a través de la cual se introducen las señales provenientes de los sensores. Como solo se dispone de una entrada, se conectarán empleando el bloque merge signals que se detalla posteriormente. La configuración del bloque internamente se realiza de acuerdo a la figura 5.13.

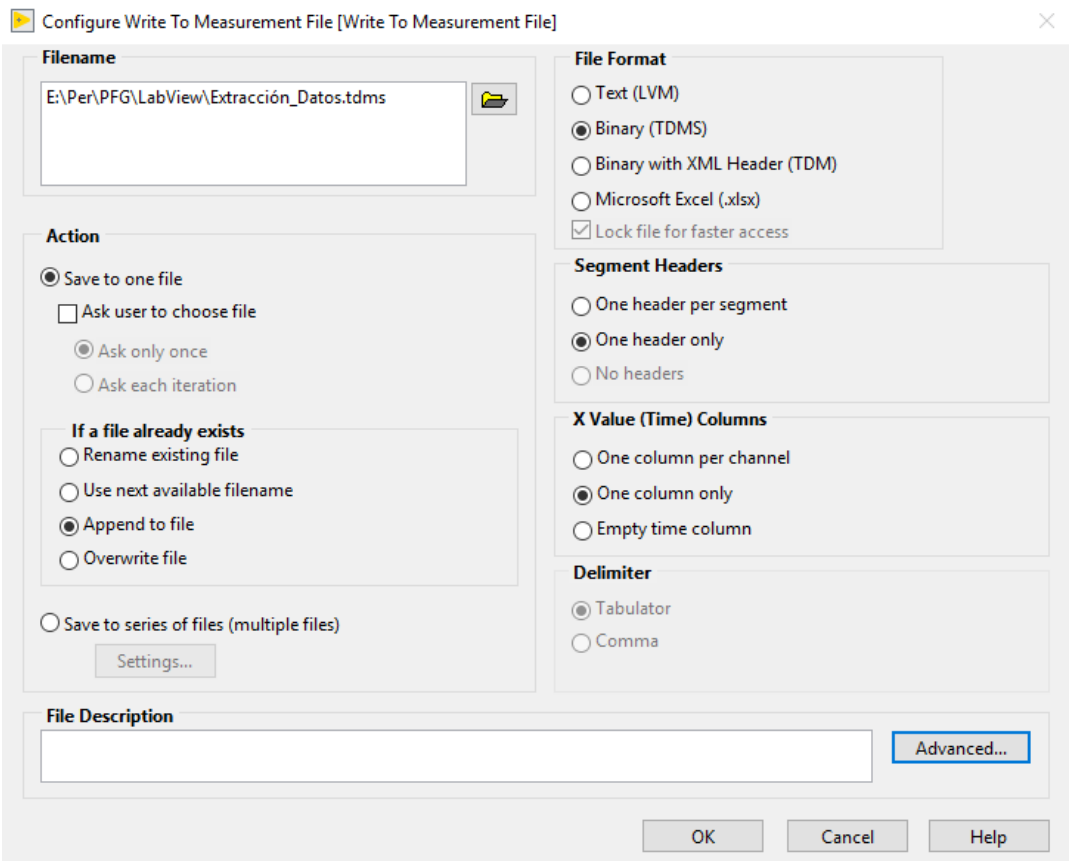

Figura 5.14. Configuración Bloque Escritura a Fichero de medida

5.3.11 BLOQUE MERGE SIGNALS

El bloque merge signals permite unir en una única salida los siguientes tipos de señales, así como reescalar la función para añadir más entradas:

- Arrays de datos numéricos de 1 o 2 dimensiones
- Arrays de datos Booleanos de 1 o 2 dimensiones
- Formas de onda
- Arrays de formas de onda

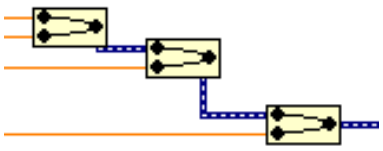

Figura 5.15. Bloque Merge

5.3.12 ESTRUCTURA WHILE

Como se ha observado en la figura 5.4, que mostraba el programa completo, alrededor del programa entre los bloques *Init.VI* y *Close.VI*, existe una estructura de tipo while. Esta estructura, al igual que en el resto de lenguajes de programación existentes, se utiliza para realizar operaciones repetitivas.

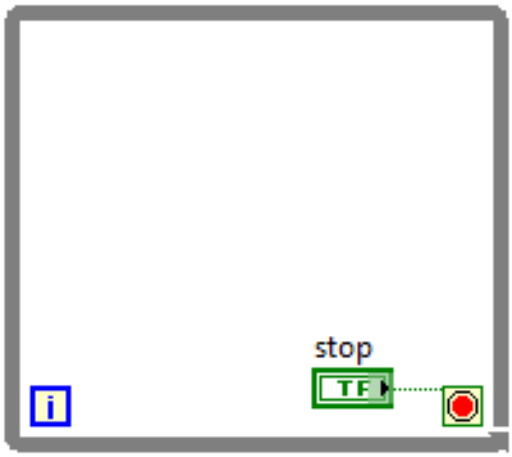

Figura 5.16. Estructura While

Por defecto, ejecutará el subprograma escrito en su interior hasta que la variable Booleana que aparece al lado del punto rojo pase a convertirse en un *"Falso"*. El código equivalente en programación sería:

Do

Ejecutar subprograma (Se cumple la condición)

While "condición" pasa a ser "Falso"

Como también se puede observar en la figura 5.16, el bucle dispone de un contador de iteraciones *"I"*, al cual una vez ejecutado el subprograma completo si la condición sigue siendo falsa, lo incrementará en una unidad. El contador de iteraciones se reseteará una vez ejecutado el subprograma con la condición en verdadero.

En este caso, la variable que genera la parada del bucle While, es un botón que aparece en la pantalla de explotación como *"Stop"*. Eso permitirá en un momento dado parar la ejecución.

6 PANTALLA DE EXPLOTACIÓN

Con la programación descrita en el panel del diagrama de bloques se ha generado la presente pantalla de explotación asociada. En ella se grafican las señales recibidas a través de las entradas de la DAQ. Las diferentes partes de las que consta se detallan a continuación siguiendo la figura 6.1.

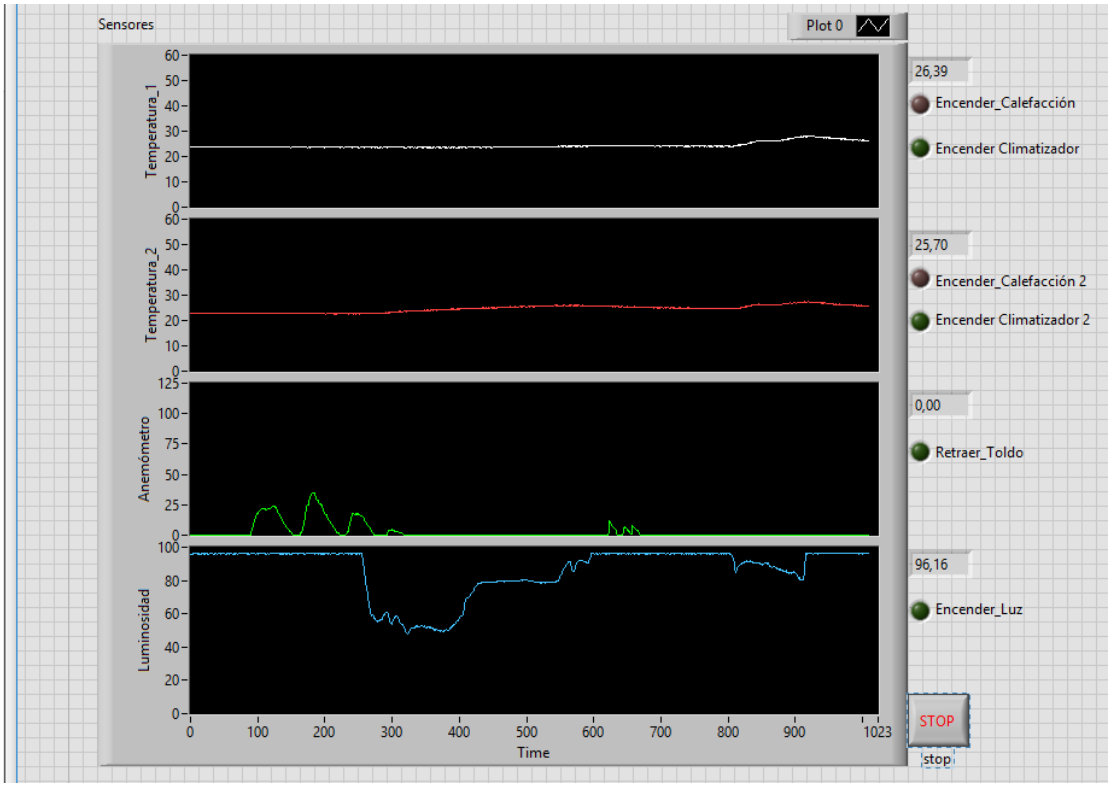

Figura 6.1. Pantalla de Explotación

6.1 GRÁFICOS

La representación gráfica de las señales de cada sensor se realiza, como se puede observar, de manera independiente para cada una de ellas.

Cada gráfico aparece diferenciado con su nomenclatura correspondiente y su fondo de escala. La generación de estos gráficos, viene asociada al bloque *"WaveForm Graph"* descrito en el apartado 5.3.9. Gracias por otra parte al bloque *"Bundle",* comentado en el apartado 5.3.8, se consigue aunar todos los gráficos en una sola estructura ordenada.

Dentro de cada uno de los gráficos, es posible acotar el fondo de escala de las coordenadas X e Y, ya que dependiendo del valor de conversión de las variables tendrán diferentes rangos figura 6.2.

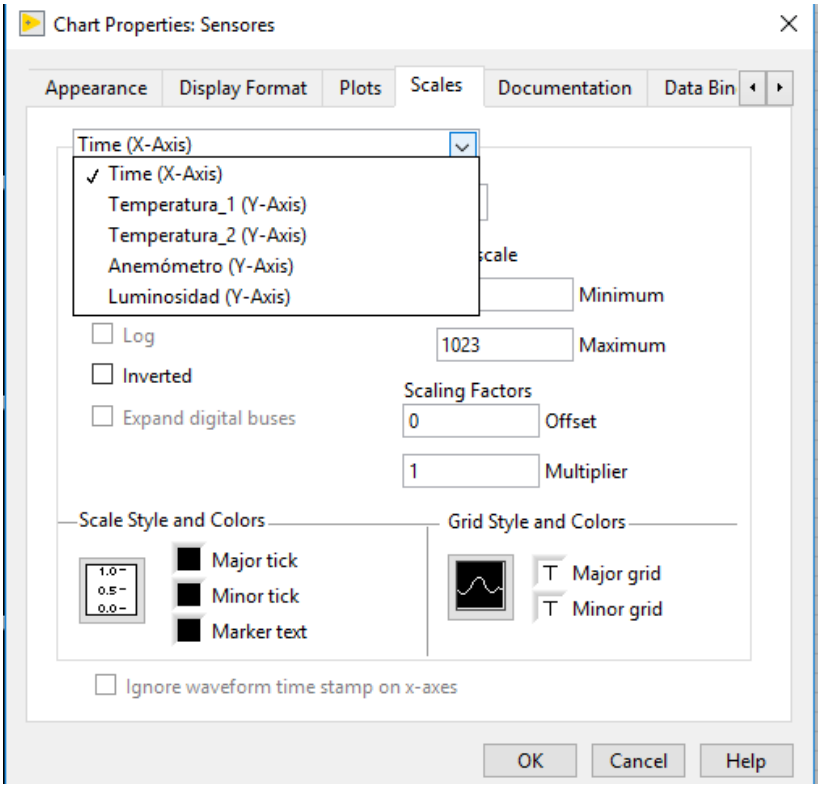

Figura 6.2. Configuración Gráficos PE

En este caso el eje temporal X, es único para todas las representaciones, ya que se trata de monitorizarlas todas de manera simultánea. Si se necesitase realizar muestreos a distintos fondos de escala, habría que graficar las señales de manera independiente.

6.2 REPRESENTACIÓN DIGITAL DE LOS DATOS Y VARIABLES BOOLEANAS

En la parte derecha de los gráficos, aparece la representación digital de la muestra de cada sensor. Se ha colocado al lado de su correspondiente gráfico para obtener la lectura instantánea de la entrada correspondiente. A su vez, se muestran empleando leds de control, las acotaciones definidas en el bloque comparador 5.3.6.

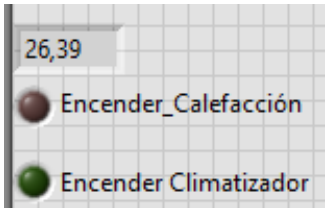

Figura 6.3. Indicadores Digital / Booleano

En la figura 6.3, se representan los indicadores para uno de los sensores de temperatura. Como se puede observar, la lectura digital instantánea puede ajustar su resolución configurando el número de dígitos decimales que se quieren mostrar. Los leds Booleanos de la comparativa, se encienden al cumplir la condición configurada en el boque 5.3.7 y 5.3.6.

6.3 BOTÓN STOP

Tal y como se ha comentado en la explicación del bucle *"While"* del apartado 5.3.12, la condición que hace que se detenga la ejecución puede ser configurada como un botón de *"Stop"* que aparezca en la pantalla de explotación. Esto es exactamente lo que se ha hecho y su representación gráfica aparece en la figura 6.4.

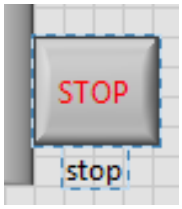

Figura 6.4. Botón Stop Ejecución

El usuario una vez que está ejecutando el programa, podrá detenerlo pulsando sobre el botón con el ratón.

7 RESULTADOS OBTENIDOS

7.1 PANTALLA DE EXPLOTACIÓN

Tras terminar la programación completa tanto en la plataforma Arduino como en LabView se procede a comprobar el comportamiento de la adquisición y monitorización de los datos. En la figura 7.1, se observan las mediciones llevadas a cabo en una prueba. Se han forzado las mediciones de los dispositivos para ver que el comportamiento es estable en todo el rango de medidas.

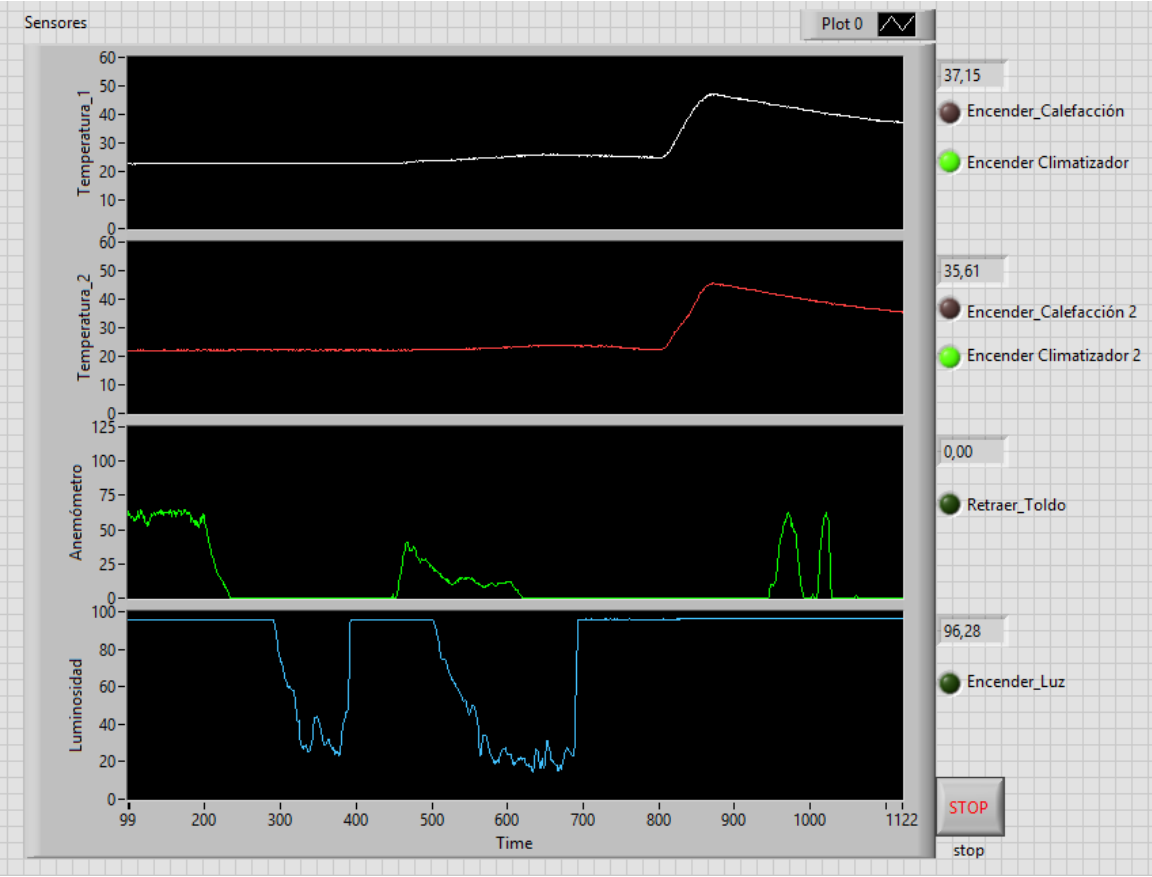

Figura 7.1. Pantalla de explotación pruebas

Como se observa durante la ejecución del programa, se van registrando los datos en sendos gráficos. El comportamiento de los sensores es parejo ya que se encuentran físicamente cerca en la placa. La fuente de calor empleada para elevar la temperatura es un secador e incidía directamente en los dos por eso el comportamiento parejo. Al llegar a la consigna superior, como se observa en la figura, se da la acción de encender el climatizador.

En el caso del anemómetro, al igual que para forzar los LM35 se ha empleado un secador. Como se observa la medición a máxima potencia es de unos 60 $\frac{km}{h}$ que es acorde a lo que se podría suponer en la salida de la boca del secador a máxima potencia. El comportamiento no es estable por que se forzaban de manera inconsistente para aproximarlo a la realidad.

Para el sensor de luminosidad, se pasaba la mano por delante forzando la sombra para ver como a medida que disminuye la luminosidad, baja la medición en el gráfico.

7.2 COMPARATIVA PANTALLA EXPLOTACIÓN-HOJA DE CÁLCULO

Como ya se ha explicado en la programación, se ha configurado una extracción de datos a formato hoja de cálculo. La extracción permite registrar datos con un periodo de:

$$T = 50\ ms$$

Como se observa no es un registro muy rápido, pero para la aplicación que nos ocupa es suficiente. Si se quisiera realizar un análisis más rápido, sería necesario realizar una comunicación vía puerto paralelo.

Si tras extraer los datos en hoja de cálculo se grafican se obtiene el mismo resultado que en la pantalla de explotación con lo que la extracción es óptima.

7.2.1 TEMPERATURA_1

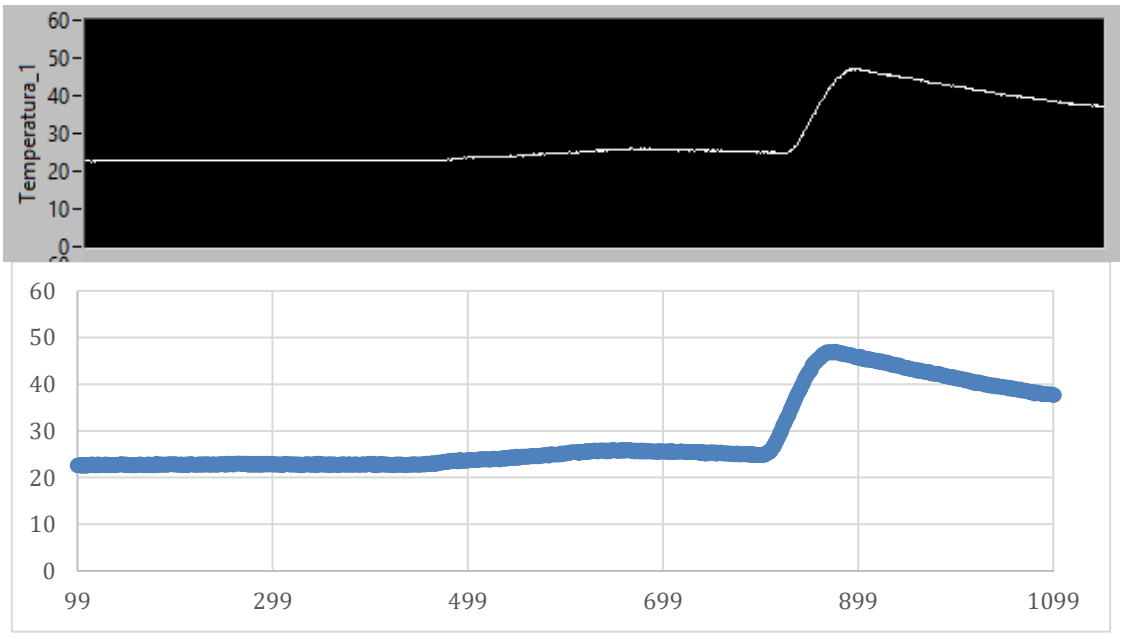

Figura 7.2. Comparativa Temperatura_1

En la figura superior aparece representada la gráfica extraída de la pantalla de explotación. En la inferior la conformada a raíz de los datos extraídos en Excel.

7.2.2 TEMPERATURA_2

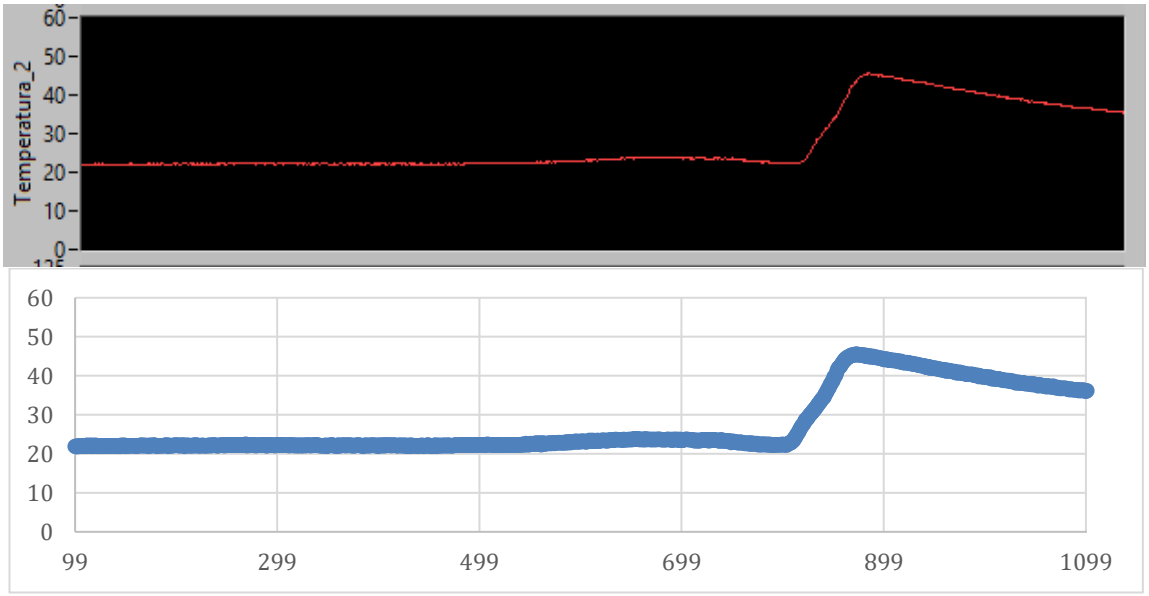

Figura 7.3. Comparativa Temperatura_2

7.2.3 ANEMÓMETRO

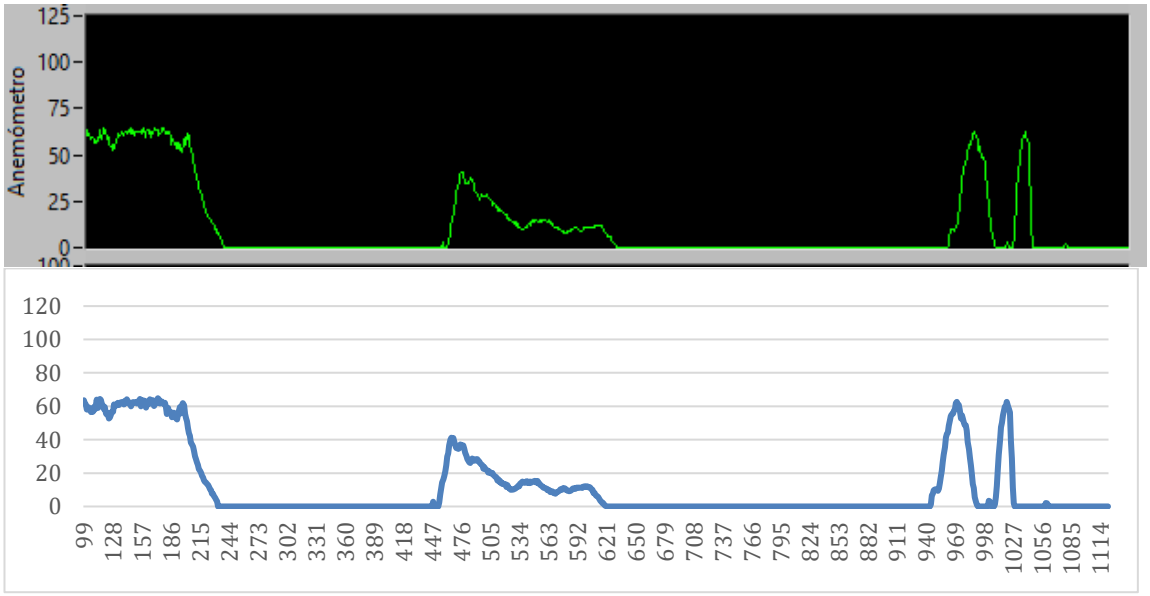

Figura 7.4. Comparativa Anemómetro

7.2.4 SENSOR DE LUMINOSIDAD

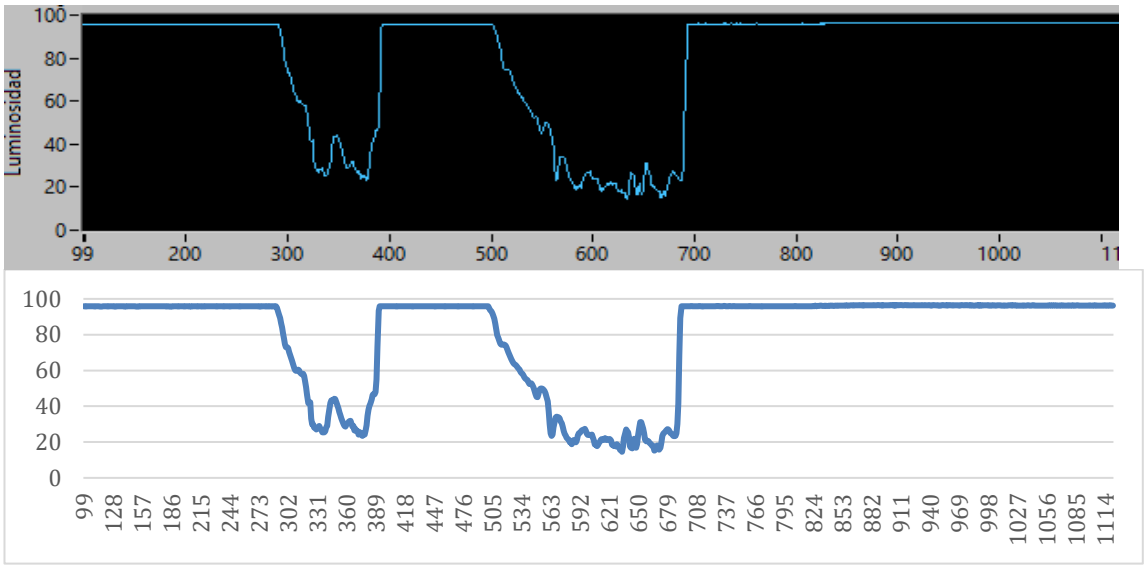

Figura 7.4. Comparativa Sensor Luminosidad

8 CONCLUSIONES

Tal y como se ha mostrado en las simulaciones, el algoritmo de control funciona de manera correcta, otorgando un muestreo fiable. La ejecución se hace en el orden correcto otorgando los resultados deseados y comportándose de manera eficiente forzando los sensores. Se ha demostrado que tiene velocidad de respuesta suficiente. Como ya se ha comentado el periodo de muestreo es de unos 50 ms.

No es necesaria la implementación de un algoritmo de control más rápido para el control de las variables del proyecto. Si a futuro se incluyeran variables cuyo comportamiento en respuesta fuera más rápido, se debería plantear otra arquitectura de conexión y programación (empleando por ejemplo puertos paralelos).

9 BIBLIOGRAFÍA

Las referencias bibliográficas empleadas en el apartado de cálculos software son las siguientes:

- *Essic. J. (2013). LabVIEW for Scientists and Engineers. New York. Oxford University Press.*
- *Swartch. M (2014). Programming Arduino with LabView. Packt Publising.*
- *Lozano Equisoain. D. (2017) Arduino Práctico. Anaya.*

Del mismo modo se han empleado las siguientes webs de interés:

[1] https://www.arduino.cc/en/Main/Software

[2] http://jki.net/vipm/download

[3] http://www.ni.com/es-es/support.html

[4] https://www.arduino.cc/

[5] http://www.ni.com/es-es/support.html

[6] https://forum.arduino.cc/index.php?board=32.0

[7] http://www.avr-tutorials.com/digital/

[8] http://geekytheory.com/arduino-y-labview

[9] http://neutrongeek.wordpress.com/tag/control-de-le-con-arduino-y-labview/