DEPARTAMENTO DE ELECTRÓNICA Y AUTOMÁTICA
FACULTAD DE INGENIERÍA – UNIVERSIDAD NACIONAL DE SAN JUAN

Manual de usuario para la utilización del datalogger

Autores:

Matías López 26231
Jesús López 26349

Junio, 2019

Contenido

1. Introducción 1
2. Objetivos 1
3. Herramientas Utilizadas para el Desarrollo 1
   3.1. Lenguaje de Programación C++ 1
   3.2. Qt Creator 1
   3.3. Arduino 1
   3.4. Sensores 2
      3.4.1. Sensor de distancia 2
      3.4.2. Sensor de temperatura 2
   3.5. Computadora 2
4. Hardware 2
   4.1. Esquema de conexiones 2
   4.2. Funcionamiento 2
5. Clases 3
   5.1. Clase Sensor 3
   5.2. Clase Temperature 4
   5.3. Clase Distance 4
   5.4. Clase Communication 5
   5.5. Clase Datalogger 7
6. Software 10
7. Referencias 11

1. Introducción

El datalogger es el instrumento perfecto cuando se necesita documentar las mediciones como prueba de calidad, destaca por su nivel de facilidad operativa, fiabilidad y seguridad. Gracias a la disponibilidad de memoria, por utilizar una PC para el almacenamiento, se puede almacenar una gran cantidad de datos y contara con una gran robustez ante fallas imprevistas como puede ser un corte de energía eléctrica.
Se desarrolla la explicación del datalogger propuesto el cual cuenta con un Arduino en que se le pueden conectar sensores externos para todo tipo de aplicaciones y además cuenta con una PC en donde se ha desarrollado una interfaz gráfica simple y de fácil utilización donde se podrá almacenar, supervisar y gestionar todos los datos registrados.
El datalogger implementado dispone de un sensor de temperatura ideal para la supervisión continua de la temperatura de almacenamiento de, por ejemplo, alimentos o productos farmacéuticos. También es adecuado para el control de la temperatura en cámaras frigoríficas o almacenes refrigerados. Además, incluye un sensor de distancia para diversas aplicaciones.

2. Objetivos

El objetivo general de este proyecto fue volcar todos los conocimientos aprendidos en clase para lograr hacer un programa eficiente y los más óptimo posible, para lo cual se tuvo en cuenta conceptos como herencia, composición, paso de parámetros por referencia, tipo de acceso, tipo de variable, modificadores adecuados, y respetando todos los paradigmas fundamentales.
Uno de los objetivos específicos es el de lograr comprender e implementar objetos informáticos a partir de objetos del mundo real. Con esto se buscó distinguir mediante un profundo análisis si convenía derivar o si con una clase se encapsulaban los demás objetos buscando que sea lo más natural posible o si era conveniente componer, en donde objetos simples conforman una clase con un sentido conceptual mayor.
Otro objetivo específico que se persiguió, ya que se estaba comenzando un proyecto desde cero, es el de codificar respetando estilos de programación de manera de seguir un orden para que se pueda leer más fácilmente.

3. Herramientas Utilizadas para el Desarrollo

3.1. Lenguaje de Programación C++

C++ es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. Una particularidad del C++ es la posibilidad de redefinir los operadores, y de poder crear nuevos tipos que se comporten como tipos fundamentales.

3.2. Qt Creator

Es un IDE multi plataforma programado en C++, *JavaScript* y QML creado por Trolltech el cual es parte de *SDK* para el desarrollo de aplicaciones con Interfaces Gráficas de Usuario (GUI por sus siglas en inglés) con las bibliotecas Qt, Los sistemas operativos que soporta en forma oficial son:

- GNU/Linux 2.6.x, para versiones de 32 y 64 bits con Qt 4.x instalado. Además, hay una versión para Linux con gcc 3.3.
- Mac OS X 10.4 o superior, requiriendo Qt 4.x
- Windows XP y superiores, requiriendo el compilador MinGW y Qt 4.4.3 para MinGW.

3.3. Arduino

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo (software), diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

3.4. Sensores

3.4.1. Sensor de distancia

El HC-SR04 es un sensor de distancias por ultrasonidos capaz de detectar objetos y calcular la distancia a la que se encuentra en un rango de 2 a 450 cm. El sensor funciona por ultrasonidos y contiene toda la electrónica encargada de hacer la medición. Su uso es tan sencillo como enviar el pulso de arranque y medir la anchura del pulso de retorno.

3.4.2. Sensor de temperatura

El LM35 es un sensor de temperatura con una precisión calibrada de 1 °C. Su rango de medición abarca desde -55 °C hasta 150 °C. La salida es lineal y cada grado Celsius equivale a 10 mV.

3.5. Computadora

Una computadora portátil es un dispositivo informático que se puede mover o transportar con relativa facilidad. Los ordenadores portátiles son capaces de realizar la mayor parte de las tareas que realizan los ordenadores de escritorio, también llamados «de torre», o simplemente PC, con similares capacidades y con la ventaja de su peso y tamaño reducidos. Además, también tienen la capacidad de operar por un período determinado por medio de baterías recargables, sin estar conectadas a una red eléctrica. Se encargará de realizar la supervisión, control y gestión de los datos registrados. La notebook utilizada posee un procesador AMD Quad-Core A10-9600P (2.4 GHz), 12 GB de RAM DDR4, gráficos de video AMD Radeon™ R5 Graphics. EL sistema operativo que posee dicha notebook y donde se realizó el desarrollo fue Debian 9.9.

4. Hardware

4.1. Esquema de conexiones

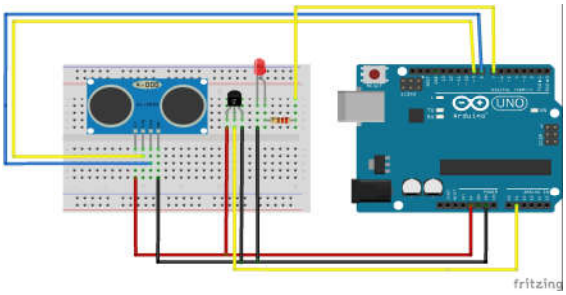

Figura 1: esquemático.

Se deberá realizar las conexiones indicadas en la figura 1 para realizar la función del datalogger.

4.2. Funcionamiento

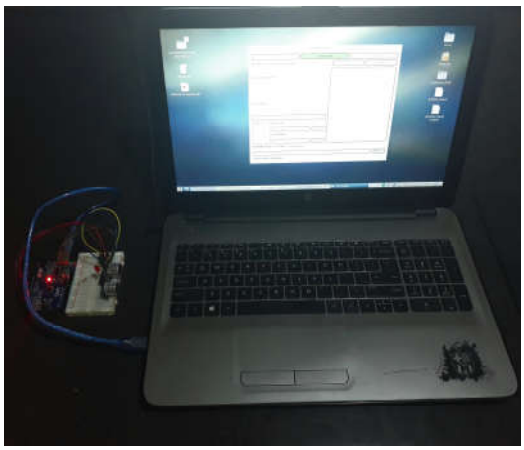

Figura 2: Datalogger.

Como puede observarse en la figura 2 se encuentran todos los elementos, mencionados en el apartado tres, que componen el datalogger. La función que realiza es la de establecer la comunicación con el Arduino y luego, como es una topología maestro-esclavo, la PC le indica al Arduino(que realmente trabaja como un conversor) que requiere que le mande datos cada un cierto tiempo(con la posibilidad de configurarlo) de los sensores que tiene conectados, cuenta además con un led que se encenderá cada vez que esté recibiendo datos, para que posteriormente la PC se encargue de realizar la decodificación y los cálculos necesarios para obtener los datos para almacenarlos y poder realizar distintas operaciones con estos en una interfaz gráfica, en el apartado siguiente se explica con mayor detalle la función de esta.

5. Clases

5.1. Clase Sensor

Se emplea una clase sensor abstracta que tendrá una interfaz común para las derivaciones que se hagan para un sensor en particular.

Header:	#include <datalogger>
Deriva:	-

Propiedades

• value: float	• date: Date

Métodos Públicos:

	Sensor()
float	getValue(void) const
Date	getDate(void) const
void	setDate(const QByteArray&, const int)
virtual void	setValue(const QByteArray&, const int) = 0
virtual	~Sensor()

Documentación de las propiedades

value: float

Esta propiedad se utiliza para almacenar el valor de un determinado sensor.

date: Date

Esta propiedad del tipo Date se utiliza para almacenar la fecha y hora proveniente del Arduino.

Documentación de los Métodos

Sensor::Sensor()

Constructor de *Sensor* donde los miembros de la estructura *Date* se establecen en cero.

float Sensor:: getValue(void) const

Método que se encarga de obtener el valor de *value*, retornando un tipo *float*.

Date Sensor::getDate(void) const

Método que se encarga de obtener el valor de *date,* retornado el tipo definido por el usuario *Date*.

void Sensor::setDate(const QByteArray&, const int)

Se encarga de establecer el valor de los miembros de la estructura *Date.* Recibe como argumentos un *QByteArray* constante por referencia que corresponderá al valor de la fecha desde el Arduino y un entero constante correspondiente al índice de la cabecera.

void Sensor::setValue(const QByteArray&, const int) [pure virtual]

Método virtual puro que se encarga de decirle al compilador que este método se definirá en la clase derivada.

Sensor:~Sensor() [virtual]

Destructor del objeto.

5.2. Clase Temperature

Esta clase se deriva públicamente de la clase sensor.

Header:	#include <datalogger>
Deriva:	Sensor

Métodos Públicos:

	Temperature()
virtual void	setValue(const QByteArray&, const int)
virtual	~Temperature()

Documentación de los Métodos

Temperature::Temperature()

Constructor de *Temperature* donde se establece el valor de *value* en cero.

void Temperature:: setValue(const QByteArray&, const int) [virtual]

Método que se encarga de establecer el valor de *value* donde recibe como argumento *QByteArray* que proviene del Arduino y un índice de la cabecera.

Temperature:~Temperature() [virtual]

Destructor del objeto.

5.3. Clase Distance

Esta clase se deriva públicamente de la clase sensor.

Header:	#include <datalogger>
Deriva:	Sensor

Métodos Públicos:

	Distance()
virtual void	setValue(const QByteArray&, const int)
virtual	~Distance()

Documentación de los Métodos

Distance::Distance()

Constructor de *Distance* donde se establece el valor de *value* en cero.

void Distance:: setValue(const QByteArray&, const int) [virtual]

Método que se encarga de establecer el valor de *value* donde recibe como argumento *QByteArray* que proviene del Arduino y un índice de la cabecera.

Distance:~Distance() [virtual]

Destructor del objeto.

5.4. Clase Communication

Esta clase contiene los métodos y propiedades necesarios para lograr la comunicación con el microcontrolador.

Header:	#include <datalogger>
Deriva:	-

Propiedades

- *arduino: QSerialPort
- arduino_uno_vendor_id: quint16
- arduino_uno_product_id: quint16
- arduino_port_name: QString
- arduino_is_available: bool
- headerL: qint8
- headerH: qint8
- stopt: qint8
- data: QByteArray

Métodos Públicos:

	Communication()
bool	startSerialPort()
void	closeSerialPort() const
bool	writeToStart()
bool	writeToStop()
void	readData()
QByteArray	getData() const
qint8	getHeader() const
qint8	getHeaderL() const
qint8	getStop() const
QSerialPort*	getArduino() const
	~Communication()

Documentación de las propiedades

*arduino: QSerialPort

La propiedad *arduino* permite crear un puerto serial virtual en QT.

arduino_uno_vendor_id: quint16

Permite, en conjunto con *arduino_uno_product_id*, identificar el Arduino conectado.

arduino_uno_product_id: quint16

Permite identificar, en conjunto con *arduino_uno_vendor_id*, identificar el Arduino conectado.

arduino_port_name: QString

Propiedad donde se guarda el nombre del puerto donde está conectado el Arduino.

arduino_is_available: bool

Devuelve un booleano que será true si se conectó correctamente, en caso contrario será false.

headerL: qint8

Parte baja de la cabecera que forma parte de la trama de datos.

headerH: qint8

Parte alta de la cabecera que forma parte de la trama de datos.

stopt: qint8

Byte que corresponde al final de la trama de datos.

data: QByteArray

Variable donde se almacenarán la trama de datos a decodificar.

Documentación de los Métodos

Communication::Communication()

Constructor de *Communication* donde se establecen el *header*, el *stop,* el *vendor, el product* y se pide memoria para el puntero *arduino* .

bool Communication::startSerialPort ()

Busca algún dispositivo conectado y configura el puerto serie. Retorna un booleano que será verdadero si el Arduino está disponible y logró abrir el puerto serie, en caso contrario sera falso.

void Communication::closeSerialPort()

Cierra el puerto serie.

bool Communication::writeToStart()

Envía un 1 al Arduino para que comience a enviar datos. Retorna un booleano que sera verdadero si se logró escribir en el arduino.

bool Communication::writeToStop()

Envía un 0 al Arduino para que cese él envió de datos. Retorna un booleano que será verdadero si se logró escribir en el arduino.

void Communication::readData()

Realiza la lectura de los datos, almacenandolo en la propiedad *data*.

QByteArray Communication::getData() const

Retorna un *QByteArray* con el dato recibido desde el Arduino (*data*).

qint8 Communication::getHeaderL() const

Retorna la parte baja de la cabecera.

qint8 Communication::getHeaderL() const

Retorna la parte alta de la cabecera.

qint8 Communication::getStop() const

Retorna el byte de final de trama.

QSerialPort* Communication::getArduino() const

Retorna el puntero *arduino*. Este se utilizará para realizar la conexión.

Communication::~Communication()

En el destructor se libera la memoria pedida por el puntero *arduino.*

5.5.Clase Datalogger

Esta es la clase principal del proyecto. Contiene métodos y propiedades para lograr construir el Datalogger.

Header:	#include <datalogger>
Deriva:	QMainWindow

Propiedades

- *ui: Ui::Datalogger
- db: QSqlDatabase db
- *m_timer: QTimer
- temp: Temperature
- dist: Distance
- Communication: communication

Métodos Públicos:

explicit	Datalogger(QWidget *parent = nullptr)
void	createTable()
void	insertData()
vood	showData(const QString)
void	sortData(const QString)
void	browerData(const QString, const QString)
void	deleteData(const QString, QString)
void	setLedRed()
void	setLedGreen()
	~Datalogger()

Slots Públicos

void	decoFrame()
void	onTimeOut()

Señales

void	timeout()
void	readyRead()

Documentación de las propiedades

*ui: Ui::Datalogger

Permite acceder a los objetos gráficos.

db: QSqlDatabase

Proporciona una interfaz para acceder a una base de datos a través de una conexión.

*m_timer: QTimer

Crea un puntero del tipo *QTimer*.
temp: Temperature

Crea el objeto del tipo *Temperature* que corresponde al sensor de temperatura.

dist: Distance

Crea el objeto del tipo *Distance q*ue corresponde al sensor de distancia.

communication: Communication

Crea el objeto del tipo *communication*.

Documentación de los Métodos

explicit Datalogger::Datalogger()

Se genera la interfaz, se crea la base de datos y abre para consulta, se pide memoria para el puntero *timer*, conecta una señal de lectura finalizada (*readyRead*) con el slot *decoFrame* y una señal de tiempo expirado del timer (*timeout*) con el slot *onTimeOut* y realiza algunas configuraciones iniciales de la interfaz gráfica.

void Datalogger::createTable()

Se crean dos tablas dentro de la base de datos donde se almacenarán los datos que se reciben desde el Arduino.

void Datalogger::insertData()

Con este método se almacenan los datos que se reciben desde el Arduino.

void Datalogger::showData(const QString)

Permite mostrar los datos almacenados en la base de datos. Recibe un *QString* con el nombre de la tabla a la que debe acceder para mostrar los datos registrados.

void Datalogger::sortData(const QString, const QString)

Permite ordenar los datos almacenados en la base de datos. Recibe dos *QStrings*, uno con el nombre de la tabla a la cual debe acceder y otro con la forma que debe ordenar ya sea en forma ascendente o descendente.

void Datalogger::browerData(const QString, const QString)

Permite buscar un valor particular almacenado en la base de datos. Recibe dos *QStrings*, uno con el nombre de la tabla que debe acceder y el otro que le indica cómo debe buscar ya sea por valor o por fecha.

void Datalogger::deleteData(const QString, const QString)

Permite eliminar algún valor particular almacenado en la base de datos. Recibe dos QStrings, uno con el nombre de la tabla que debe acceder y el otro le indica el índice que del dato a eliminar.

void Datalogger::setLedRed()

Establece un botón en rojo cuando el arduino se desconecta.

void Datalogger::setLedGreen()

Establece un botón en verde cuando se conecta correctamente.

Datalogger::~Datalogger()

En el destructor se cierra la base de datos y se libera la memoria pedida por los punteros *ui* y *m_timer*.

Documentación de los Slots

void Datalogger::decoFrame() [Slot]

Decodifica la trama de datos proveniente del Arduino.

Datalogger::onTimeOut() [Slot]

Permite tomar datos desde el Arduino cada intervalo de tiempo establecido.

6. Software

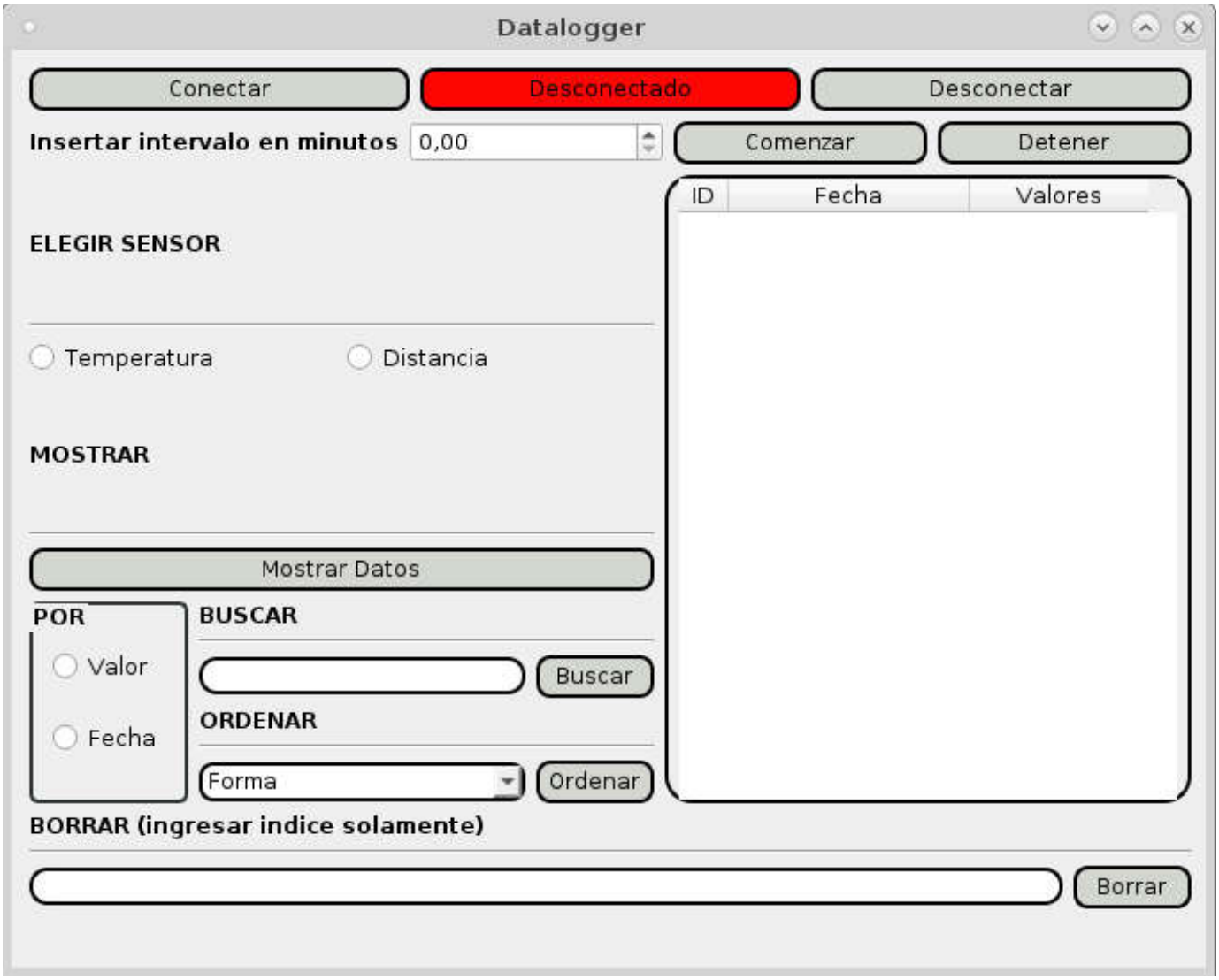

Figura 3: Interface gráfica.

Lo primero que se debe realizar es conectar el Arduino mediante un cable USB. Una vez hecho esto lo siguiente es darle clic al botón *conectar*, que si todo sale bien el indicador rojo pasara a color verde indicando, con un mensaje también, que este está conectado. Para comenzar con la toma de datos debemos indicarle el intervalo de tiempo, que dependerá de la exigencia de control a realizar, y darle al botón de *comenzar*, en caso de que se necesite parar la toma de datos solo hay que presionar el botón de *detener*. Si no se indica un intervalo de tiempo, es decir dejar este espacio cero, realizara una sola vez la toma de datos. Los datos registrados se irán almacenando en una base de datos que se podrán consultar posteriormente.
Luego se podrá gestionar los datos almacenados eligiendo, primeramente, con cual sensor se trabajará tildando la opción de *temperatura* o de *distancia*. Hecho esto basta con pulsar el botón de *mostrar datos* para visualizarlos en la tabla de la derecha, además cuenta con la opción de buscar algún valor para el cual se tiene la opción de realizarla por valor o por fecha y visualizándose también en la tabla de la derecha. Para ordenar los datos también cuenta con la posibilidad de ordenarlos por fecha o valor y además dando clic en el botón de *forma* se desplegará unas opciones para poder realizar el orden en forma ascendente o descendente.
Por Ultimo, en caso de que exista incoherencia en algún dato registrado ingresando el índice del dato que se quiera borrar se podrá eliminar de la base de datos asegurándose así de contar solamente con datos confiables.

7. Referencias

https://doc.qt.io/
http://www.learncpp.com/
http://www.cplusplus.com/reference/
http://www.yolinux.com/TUTORIALS/Cpp-DynamicMemory.html.
https://es.wikipedia.org/wiki/Qt_Creator
https://es.wikipedia.org/wiki/C%2B%2B
https://aprendiendoarduino.wordpress.com/2016/12/11/que-es-arduino-2/
https://electronilab.co/tienda/sensor-de-distancia-de-ultrasonido-hc-sr04/
https://www.luisllamas.es/medir-temperatura-con-arduino-y-sensor-lm35/
http://www.ti.com/lit/ds/symlink/lm35.pdf